HIGH EFFICIENCY VIDEO CODING PROCESSOR WITH RESIDUE NUMBER SYSTEM

by

Niras Cheeckottu Vayalil

Dissertation submitted in fulfilment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

Department of Engineering
Faculty of Science and Engineering
Macquarie University
Sydney, Australia

June 2017

ABSTRACT

The recent demand for high density video, such as ultra high definition (UHD) as well as its distribution over wired and wireless networks, led to the proposal of the latest video encoding standard, high efficiency video coding (HEVC/H.265), by the joint collaborative team on video coding (JCT-VC). HEVC/H.265 achieves a significantly better compression than its predecessor, advanced video coding (AVC/H.264), by roughly 50% for an equivalent visual reproduction quality. However, the improved compression efficiency comes with a drawback, the computational complexity. Since HEVC/H.265 encoding involves enormous computations, a hardware implementation of the encoder is necessary for real-time encoding, in particular for UHD video.

The most computationally intensive task in video encoding is motion estimation, which comprises up to 80% of the total time for video encoding. There have been several suggestions for motion-estimation algorithms for reducing the complexity, but many proposed for AVC/H.264 are no longer suitable for HEVC/H.265 due to the underlying coding changes and other complications. Hence, this research offers different algorithms and architectures for motion estimation, providing a trade-off between implementation cost and performance.

Hardware design is proposed for a full-search motion-estimation algorithm which always comes up with the best results. The memory requirement is reduced to a large extent together with the data bandwidth demand. Another important aspect of real-time video compression, including motion estimation, is the delay of

the arithmetic computations. Residue number systems have been used for decades for improving arithmetical operations performance. However, the non-positional nature of an RNS makes it difficult to do some mathematical operations such as sign detection, but it is a vital component for designing motion estimation and other elements of a video processor. The dissertation presents a fast algorithm and its architecture for sign detection, which decreases the area-delay product by 24% compared to designs in the literature.

Since the full-search algorithm searches every possible location in a search area, the algorithm involves much computation, therefore fast-search methods are preferred for low-cost solutions. The test zone (TZ) search is a fast-search algorithm and is widely used for HEVC/H.265 as it provides near optimal performance. In this dissertation, a TZ-search hardware architecture is presented, which shows 51% less gate count than existing proposals in the literature and considerably fewer memory requirements than most. Further improvement is achieved by developing a fast-search algorithm appropriate for hardware designs, providing an area-efficient, real-time UHD video-encoding-capable design without degradation in quality from the TZ search in HEVC reference software. An angle-restricted test zone (ARTZ) search motion estimation is also proposed for software applications exploiting directional probabilities of the search, saving about 17% to 55% of time for motion estimation compared to the TZ search.

The discrete cosine transform (DCT) is a standard method in several previous codecs and it is also a key factor for compression techniques in HEVC/H.265. A variable-length two-dimensional design is proposed for HEVC/H.265, where the architecture is optimised for the most likely block sizes in UHD video, thus eliminating unnecessary complexities found in many designs, and accomplishing more than 60% savings in hardware.

STATEMENT OF CANDIDATE

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of the requirements for a degree to any other university or institution other than Macquarie University.

I also certify that the thesis is an original piece of research and it has been written by me.

In addition, I certify that all information sources and literature used are indicated in the thesis.

. . . .

Niras Cheeckottu Vayalil

ACKNOWLEDGMENTS

This dissertation would not have been possible without the support and contribution of many people. First and foremost, I would like to express my sincere gratitude to my supervisor Dr Yinan Kong for his advice and support throughout the project. He provided many valuable comments and suggestions and constant encouragement during this work. I would also extend my thanks to A/Prof. Manoranjan Paul, Charles Sturt University, my adjunct supervisor, Dr Gengfa Fang and Prof. Michael Heimlich, my co-supervisors, for their excellent support.

I am also thankful to A/Prof. Joachim Rodrigues, EIT, Lund University for providing me with an opportunity to work with digital design tools in EIT, and the necessary guidance. I am also grateful to Dr Oskar Andersson, EIT, for his excellent advice and support during the visit to Lund University.

I am grateful to Dr Keith Imrie for proofreading, and his valuable suggestions on my research papers and this dissertation. I also thank to Md Selim Hossain, Dr. Shahzad Asif, all my friends, all my teachers, and my colleagues who contributed directly or indirectly to my research. I am also thankful to the admirable staff in the engineering department for their wonderful support.

I wish acknowledge to Macquarie University for awarding me an international Macquarie University research excellence scholarship (iMQRES), and the travel grants for collaborative work at Lund University.

I am blessed to have an amazing family and am grateful for their endless support, particularly during the research.

To My Family

Contents

List of publications

Following are the list of peer-reviewed research papers published, accepted or in review which resulted from the part of research.

Discussed in thesis

1. N. C. Vayalil, and Y. Kong, “VLSI Architecture of Full-Search Variable-Block-Size Motion Estimation for HEVC Video Encoding”, *IET Circuits Devices & Systems*, 2017, *in press.*
2. N. C. Vayalil, and Y. Kong, “Fast sign-detection algorithm for residue number system moduli set $\{2^n - 1, 2^n, 2^{n+1} - 1\}$”, *IET Computers Digital Techniques*, vol. 11, no. 1, pp. 33–42, 2016.
3. N. C. Vayalil, A. Safari, and Y. Kong, “ASIC design in residue number system for calculating minimum sum of absolute differences”, *Tenth International Conference on Computer Engineering & Systems (ICCES)*, Dec. 2015, pp. 129-132.
4. N. C. Vayalil, and Y. Kong, “ASIC design of test zone search motion-estimation hardware for high efficiency video coding (HEVC) with residue number system”, *IET Circuits Devices & Systems*, 2017, *in review.*
5. N. C. Vayalil, M. Paul, and Y. Kong, “A residue number system hardware design of

fast-search variable-motion-estimation accelerator for HEVC/H.265", *IEEE Transactions on Circuits and Systems for Video Technology*, 2017, *In review.*

6. N. C. Vayalil, M. Paul, and Y. Kong, "A novel angle-restricted test zone search algorithm for performance improvement of HEVC", *IEEE International Conference on Image Processing (ICIP)*, 2017, *accepted.*

7. N. C. Vayalil, J. Haddrill, and Y. Kong, "An efficient ASIC design of variable length discrete cosine transform for HEVC", *2016 European Modelling Symposium (EMS)*, Nov. 2016, pp. 229-233.

Other Publications

These publications are related but are not discussed in the thesis.

1. A. Safari, N. C. Vayalil, and Y. Kong, "Power-performance enhancement of two-dimensional RNS-based DWT image processor using static voltage scaling", *Integration, the VLSI Journal*, vol. 53, no. 2, pp. 145156, 2016.

2. M. Hossain, Y. Kong, E. Saeedi, and N. C. Vayalil, "High-performance elliptic curve cryptography processor over NIST prime fields", *IET Computers & Digital Techniques*, vol. 11, no. 1, pp. 33–42, 2017.

3. N. C. Vayalil, A. Safari, and Y. Kong, "Overlapped block-processing VLSI architecture for separable 2D filters", *Electronics, Communications and Networks IV,* , June 2015, pp. 1355–1358.

4. Y. Kong, A. Safari, and N. C. Vayalil, "A low-cost architecture for DWT filter banks in RNS applications," *2014 International Symposium on Integrated Circuits (ISIC)*, Dec. 2014, pp. 448–451.

List of contributors

Supervisor Dr. Yinan Kong, Department of Engineering, Macquarie University, NSW, Australia

Adjunct Supervisor A/ Prof. Manoranjan Paul, School of Computing and Mathematics, Charles Sturt University, NSW, Australia.

In all the publications discussed in the thesis, I have conducted the major investigations, designs measurements, data processing, and drafting. Dr. Yinan Kong, who is the principle supervisor provided suggestions, advices and invaluable guidance at every stage of the research. A/ Prof. Manoranjan Paul, adjunct supervisor provided invaluable guidance and reviewed, proof-read and corrected all the manuscripts that he has contributions. A graduate student and internal collaborator Joshua Haddrill, helped in designing of DCT architecture for HEVC/H.265 and writing a research paper co-authored with him. Table 1 gives detailed list of contributors.

List of contributors

Division of labour in co-authored articles

NC - Niras Cheeckottu Vayalil; YK Yinan Kong; MP Manoranjan Paul; JH Joshua Haddrill

	1	2	3	4	5	6	7
Conception& Design	NC	NC	NC	NC	NC	NC	NC, JH
Planning & implementation	NC	NC	NC, YK	NC	NC	NC, MP	NC, JH
Data collection	NC	NC	NC	NC	NC	NC	NC, JH
Analysis & interpretation	NC	NC	NC	NC	NC	NC	NC, JH
Writing the article	NC, YK	NC, YK	NC, YK	NC, YK	NC, MP, YK	NC, MP, YK	NC, JH, YK
Overall responsibility	YK	YK	YK	YK	MP, YK	MP, YK	YK

Table 1: *Author's contributions*

List of Figures

List of Tables

Chapter 1

Introduction

The influence of video and related applications in everyday life has increased exponentially since the advent of digital video. It has a great impact on almost all types of industries including communication, entertainment, and arts. With the advancement of digital technologies, there has been an ever-increasing demand for high-quality video in real-time communication as well as broadcasting or distribution, which necessitate efficient video data compression and standards. Since the early 1990s two main standardisation organizations, the International Telecommunication Union (ITU) and the moving picture expert group (MPEG) formed by International Standardisation Organizations/The International Electrotechnical Commission (ISO/IEC) proposed several standards. Although started for different tracks of applications, they have engaged in close cooperations and proposed advanced video coding (AVC/H.264) and high efficiency video coding (HEVC/H.265) standards, which became milestones in video coding.

On the other hand, the use of mobile devices also increased many-fold along with the advancement of digital technology. As a result, a significant amount of video processing is now taking place on these mobile devices or embedded processors, for which speed and power consumption are very critical. Every new proposal for video coding is targeted for efficient compression and improved quality, which in turn increases computational complexities in the encoding process; this applies to the latest video coding standard HEVC/H.265 as well, whose computation requirements increased substantially compared to its predecessor.

HEVC/H.265 video coding provides approximately 50% better compression efficiency compared to AVC/H.264. The improved compression capability is imperative for very-high-density videos such as in 4K ultra-high-definition (UHD) resolution. These high-resolution videos provide an immense display experience and are being widely accepted for multi-media display panels. The introduction of HEVC/H.265 saves data bandwidth and storage requirements, but the increased complexity demands hardware implementations specifically targeted for the application.

1.1 Challenges and objectives

Real-time video encoding is very challenging as it involves much computation. Moreover, the problem is aggravated by the increased coding complexities of the latest proposal and the increasing demand for high-quality or UHD-resolution videos. Application specific integrated circuit (ASIC) designs can exploit their parallel-processing features to mitigate several challenges in real-time processing. However, the new features of HEVC/H.265 have not been addressed before, in particular for motion estimation which comprises 60% to 80% of the overall computations for the encoding. HEVC/H.265 introduces several new choices of partitioning types, a hierarchical quad-tree structure, and increases the size of the basic coding block for motion estimation and the residual coding. Therefore, many proposals suggested for its predecessor cannot be adopted for HEVC/H.265.

Due to the difficulties in motion estimation, various algorithms are proposed for alleviating this bottleneck. However, the majority of suggestions are made for software platforms, but the time and power consumption for software implementation are very high, especially for processes with much computation. Furthermore, the solutions implied for software are not optimal for hardware architectures as the latter varies significantly in their operation. As an illustration, it is easy to distribute data in parallel into different sections and process it concurrently, whereas the representation of data is preferred in fixed-point notation as the floating-point notations significantly increase hardware area or cost. On the other hand, parallelism in software is limited by the number of processors available, but the floating-point data representations and algorithms with irregular structures or steps are relatively easy.

The speed of arithmetic units are crucial for hardware architectures, but carry propagation delays generally limit their performance. Residue number systems (RNSs) has been used for improving mathematical operations in hardware platforms for many decades. However, the research on RNS for video processing is not strong. The RNS has a potential to outperform traditional binary systems, as the former could have better arithmetic hardware, and this research investigates the possible areas of using RNS for video processing and its benefits.

In this dissertation, one of the primary objectives is to implement a real-time hardware for motion estimation for HEVC/H.265 as it is responsible for the majority of the computations. Most of the problems in current proposals in the literature are addressed in this dissertation. For example, the memory and data bandwidth requirements are minimised by broadcasting data and using specific reading patterns for full-search motion estimation. Different proposals for motion estimation, including new algorithms for both hardware and software platforms, are provided. These choices allow a trade-off between hardware implementation cost and coding performance, as both increase together.

1.2 Main contributions

In this dissertation, several architectures and algorithms have been presented to meet the challenges in real-time video encoder design with the HEVC/H.265 specifications and for UHD resolution, which is becoming very popular. A vital problem in hardware processing is the speed of arithmetic operations, which has been addressed using RNS whenever appropriate. However, some mathematical operations such as magnitude comparison and sign detection are not trivial in an RNS, hence a novel algorithm and architecture have been proposed for addressing this. Motion-compensated prediction is the key for efficient video compression but it is the most tedious and time-consuming process, and it is increased many times in HEVC/H.265 and also by the increased pixel resolution for UHD video. Therefore most of the dissertation is devoted to addressing this problem and providing fastest and most cost-effective solutions. Residual picture coding is another challenge in video encoding, and an ASIC design has been presented as a solution. The following are the some of the key scientific contributions:

- Motion estimation has been broadly classified into full-search and fast-search methods, where a full-search provides better results than fast-search methods. The memory and bandwidth requirements of full-search motion estimation in hardware implementations have been reduced using a different approach for the data reading pattern and by broadcasting data simultaneously to multiple processing elements.

- In this dissertation, the RNS has been used extensively for addressing challenges in video processor design. Nevertheless, one fundamental difficulty in RNS, the sign detection problem, has to be solved since it is the key to many hardware designs. A novel algorithm and architecture have been provided, which significantly reduces the area-delay product compared to others in the literature for a specific moduli set.

- Test-zone (TZ) search is a widely used fast-search motion estimation in HEVC/H.265

as it provides similar results to full-search methods with significant improvement in computational time. A TZ-search hardware implementation has been presented with RNS, providing a very cost-effective solution for motion estimation.

- A new algorithm has been proposed for hardware designs which provides equivalent or better results than a TZ search without increasing hardware cost significantly.
- Angle-restricted test zone (ARTZ) has been proposed for software implementations, exploiting directional probabilities of a motion-vector search, decreasing motion estimation time by 20% to 55% with little quality degradation compared to TZ search in HEVC/H.265 reference software.
- A variable-length discrete cosine transform (DCT) architecture has been presented for HEVC/H.265 specifications. The design is optimised for the most likely block sizes in UHD video, hence unnecessary complexities found in many architectures proposed have been eliminated, which brings about 60% hardware saving and plenty of processing power for the transform coding of UHD video.

1.3 Dissertation outline

This dissertation follows the non-traditional "Thesis-by-publication" format which has been approved by the *Macquarie University Higher Degree Office*. It consists of a general introduction, background, and a list of the PhD candidate's major scientific publications. The thesis materials are original texts and graphics of the author's publications, published or under review, that have been reformatted to improve readability. As mentioned before, motion estimation is the most time-consuming operation in video encoding and the dissertation provides a few alternative solutions with a trade-off between many parameters such as hardware cost, quality, and compression efficiency.

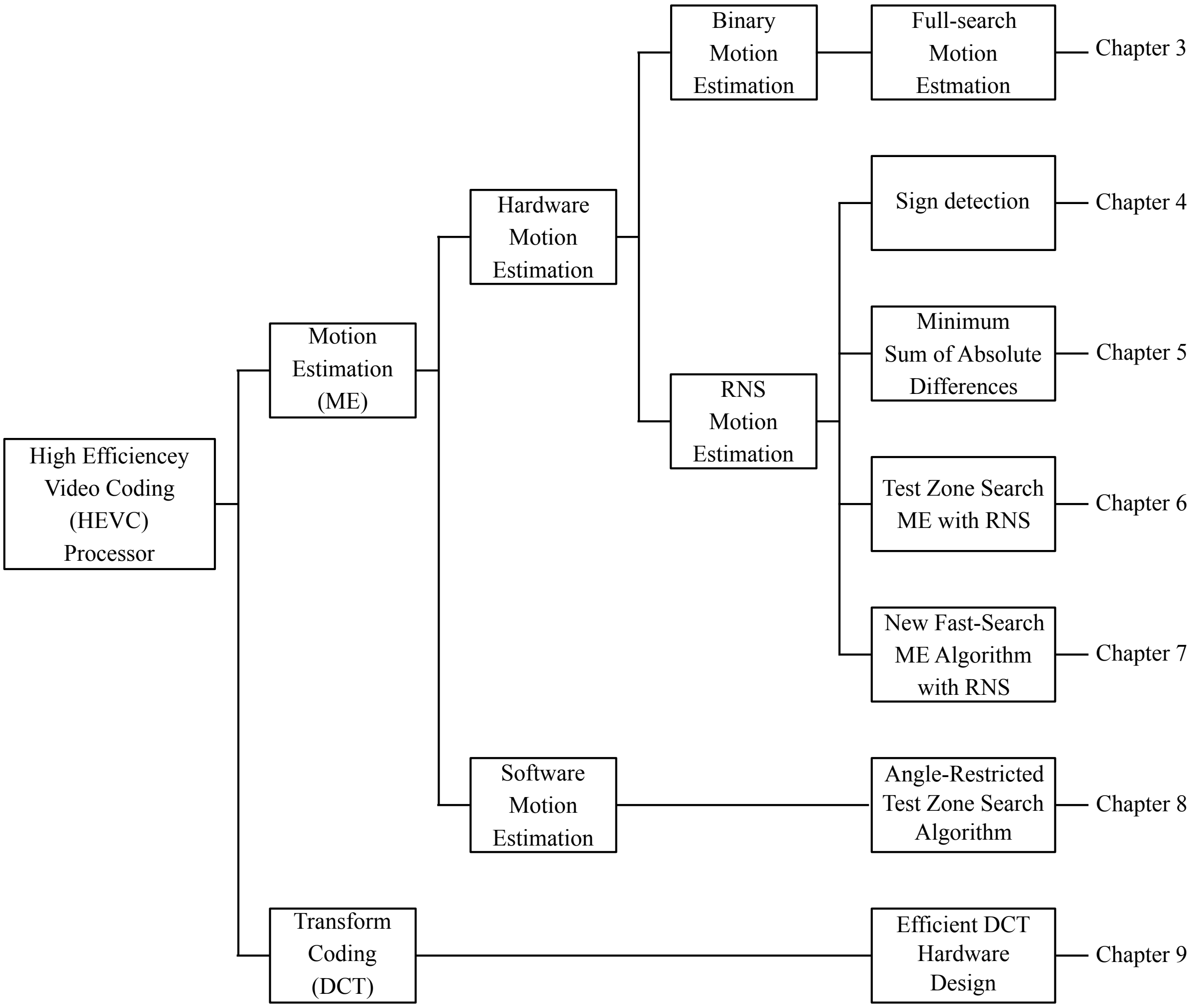

High Efficiencey Video Coding (HEVC) Processor
Motion Estimation (ME)
Transform Coding (DCT)
Hardware Motion Estimation
Software Motion Estimation
Binary Motion Estimation
RNS Motion Estimation
Full-search Motion Estmation
Sign detection
Minimum Sum of Absolute Differences
Test Zone Search ME with RNS
New Fast-Search ME Algorithm with RNS
Angle-Restricted Test Zone Search Algorithm
Efficient DCT Hardware Design
Chapter 3
Chapter 4
Chapter 5
Chapter 6
Chapter 7
Chapter 8
Chapter 9

Fig. 1.1: Thesis outline

Chapter 2 gives a brief overview of an overall structure of video compression including various basic concepts, encoder architectures and a short introduction to the latest HEVC/H.265 coding standard. A comprehensive review of different algorithms existing in the literature for motion estimation is included in this chapter. It also introduces RNS fundamentals and gives a brief explanation for the choice of a particular moduli set for the application.

Chapter 3 discusses architecture for motion estimation for HEVC/H.265 with the full-search method, which always comes up with better results than fast-search methods. The memory requirement for hardware has been reduced by following a Morton order for data reading and a sum of absolute differences (SAD) reuse strategy. The data bandwidth is also minimised by broadcasting data to multiple processing elements (PEs).

Chapter 4 presents a novel algorithm and architecture for sign detection, which is not trivial in RNS, for the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$. A brief overview of algorithms and architecture for sign detection is also provided, followed by results and comparisons. The area-delay product has been reduced by 24% compared to the proposed methods in the literature. The sign-detection architecture is used in RNS architecture described in other chapters for sign detection as well as magnitude comparison of data.

Chapter 5 investigates the benefits of using an RNS in computations of the minimum SAD. The SAD is widely used for motion estimation algorithms as it is relatively simple compared to other distortion measurement matrices. The architecture uses a relatively prime moduli set, and the results show enhanced computational speed in contrast with both non-relatively prime moduli sets and binary systems.

Chapter 6 comes up with an architecture for TZ search, which is the most-preferred fast-search motion estimation algorithm for the HEVC/H.265 standard. RNS has been incorporated to speed up arithmetic operations, and the proposed method brings down the gate count by about 51% compared to existing proposals in the literature.

Chapter 7 introduces a novel algorithm and its architecture design with RNS for motion estimation according to the HEVC/H.265 specifications. The hardware is capable of encoding UHD video in real time with an equivalent or in most cases a better performance than TZ motion estimation, and without a remarkable increase in hardware cost.

Chapter 8 explains a novel fast-search algorithm, called ARTZ, for software applications. The algorithm brings down the motion-estimation time by about 20% to 55% by exploiting directional probabilities in motion search, with little effect on the quality and compression efficiency compared to a TZ search in the HEVC reference software.

In Chapter 9, an architecture for transform coding (DCT) required by the HEVC/H.265 standard has been introduced. The design is targeted for encoding UHD videos as it requires more extensive computations than comparatively low-resolution video such as high-definition (HD). Hence the architecture is optimised for UHD video, but can also encode HD as it has fewer computations. This approach helps to remove unwanted complexities existing in many other architectures proposed in the literature, and achieves more than 60% hardware savings with enough computational power for real-time processing of UHD video.

Conclusion remarks are drawn in Chapter 10, and also included directions for the future research work.

Chapter 2

Background and Related Work

2.1 Video compression

The raw video generally consists of a massive amount of data, and the demand for high-quality video, with quality in terms of higher video resolution, higher frame rates, or higher fidelity, further increases the data size significantly. Moreover, video data transferring through wired as well as wireless networks has also grown considerably in recent years as there is a rapid growth in applications such as video phone, remote home surveillance, etc. These make raw video data compression a necessity, and efficient compression techniques are inevitable to save or transmit the data effectively.

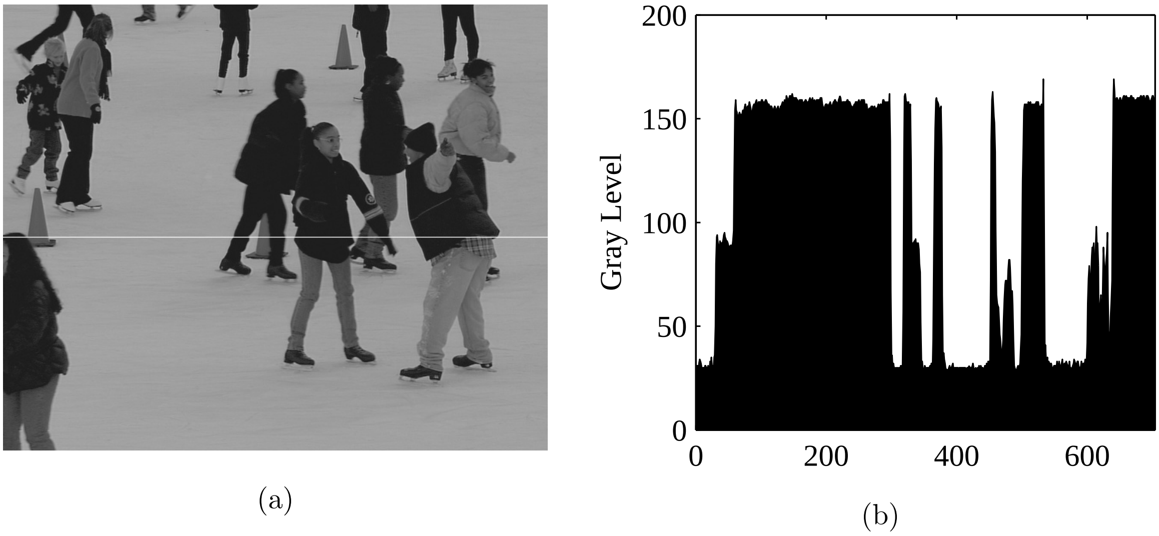
Compression can be either lossless, where no information is lost during compression, or lossy, where less important or unnecessary information is discarded without much lowering the subjective reproduction quality. Although lossless compression can manage a bit-rate reduction of 3.2% to 13.2% on average [1], lossy compression is very popular for video coding because of its better compression: a typical HEVC/H.265 lossy compression showed a bit-rate saving of 50% to 60% for the same perceived video quality [2]. Lossy compression is subject to a trade-off between various factors such as the degree of compression, quality degradation, or the complexity involved in compression. For instance, it may require expensive hardware resources to encode/decode a highly compressed video in real time even though it save storage space or transmission bandwidth.

Modern video compression methods exploit the redundancy that exists in the natural video such as statistical redundancy and psycho-visual redundancy. Compression is achieved by eliminating these redundancies from the raw video data.

2.1.1 Statistical redundancy

Generally, the pixels in a video are not statistically independent; some are similar to their neighbouring pixels, or to the pixels across successive images or frames. Another kind of statistical redundancy is coding redundancy which is associated with the coding technique.

Spatial redundancy



(a)

(b)

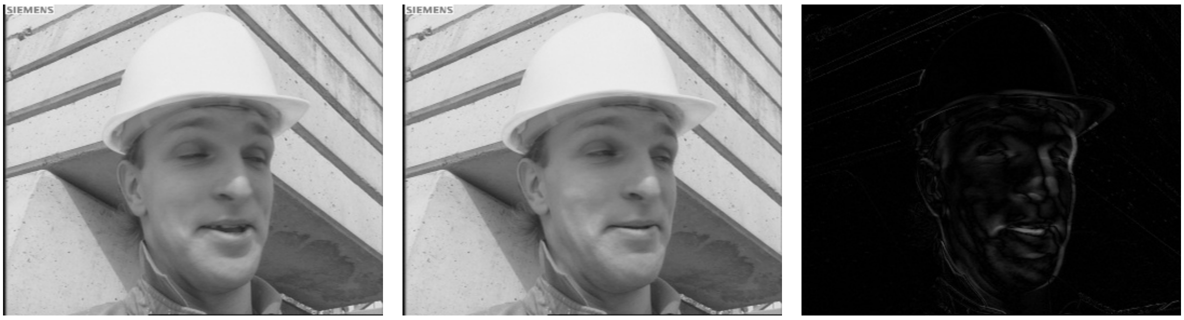
Fig. 2.1: (a) First frame of ice_4cif video sequence and (b) grey levels of pixels along the 300th row of this frame.

The pixel values in a frame change smoothly from one pixel to its neighbouring pixels, except at the boundaries of an object. This results in a very high statistical correlation between pixels within an image frame. Spatial redundancy implies that a pixel can be predicted from its neighbouring pixel values, hence it is not necessary to represent every pixel of a video frame independently. The spatial redundancy can be used for both lossless and lossy coding approaches. The first frame from a test video sequence is shown in Fig. 2.1 (a), and the pixel value change in grey level (luminance) over a horizontal line is plotted in Fig. 2.1 (b). Smooth changes in pixel grey level can be observed except at the edges or borders of objects.

Spatial redundancy is used in both lossless and lossy compression. Most lossless compression techniques include generating a statistical model for input data, then using this model to map input data into bit sequence that produces shorter output data, for instance, using a shorter code for frequently occurring symbols and a longer code for less frequent symbols. Examples of lossless formats are Huffman coding [3], Lempel-Ziv-Welch (LZW) algorithm [4, 5], etc. On the other hand lossy compression exploits the psycho-visual limitation of a human eye to discard redundant data. Slow and gradual changes of luminance are more perceivable to the human eye rather than finer details and rapid changes of intensity. Such psycho-visual characteristics are used in modern image and video compression formats.

Temporal redundancy

A video consists of successive frames; usually, they are statistically correlated, contributing to the temporal redundancy. For example, two frames from a video sequence Foreman are shown in Fig. 2.2 (a) and Fig. 2.2 (b); as seen from these frames several pixels of the images remain the same, or the similarity is very high as seen in the difference in Fig. 2.2 (c). This is the case for a majority of the frames in a video sequence except for



(a) Frame 1 (b) Frame 2 (c) Difference

Fig. 2.2: Frames from video sequence Foreman, and the differences.

frames having significant changes, such as scene changes. As a result, it is possible to predict adjacent frames along the temporal axis. In several cases, the changes from one frame to another frame are due to movement of some objects, as in Fig. 2.2; apart from the movement of the lip and face portions, the two frames are static in nature. Motion-compensated predictive coding considers this motion information, and it is thus possible to remove many temporal redundancies in the video, resulting in a very high compression efficiency.

2.1.2 Psycho-visual redundancy

Psycho-visual redundancy is associated with the perception of the outside world by the human visual system, where some information may be more valuable than others. For instance, the human visual system is more sensitive to luminance than colour. Hence less-relevant information can be represented using less data, and it will not affect perception qualitatively. Since these videos are made for human perception, psycho-visual redundancies can be utilised for video compression. However, many findings in psycho-visual effects have not been explored in the context of video compression applications.

2.1.3 Coding redundancies

A video is encoded after removing the redundancies mentioned above, although the redundancy exists itself in the coded data, which can be explored for further compression. Coding redundancy has nothing to do with redundancies in the information but with redundancy in its representation. Lossless compression methods, such as context-based adaptive binary arithmetic coding (CABAC) [6], are used in many modern video coding standards for eliminating these redundancies.

2.1.4 Video quality measurement

Quality measurement is necessary for evaluating two different compression standards or measuring the effects of an algorithm improvement. When two methods result in the same quality the advantages of algorithm or compression standards can be easily estimated regarding speed, bit rate etc. However the measurement is not straightforward. Two different approaches are used for this: subjective assessment (by human observation) and objective assessment (by analysing data). In the first method, observers evaluate picture quality by rating pictures or alternatively giving information about impairments in the pictures. In image and video data compression, peak signal-to-noise ratio (PSNR) is used as an objective quality assessment, defined as a ratio of the maximum possible power of the signal to the power of the noise. Define the error function $e(x, y)$ as the difference between the input image $f(x, y)$ and the output image $g(x, y)$, i.e.

$$e(x, y) = f(x, y) - g(x, y) \tag{2.1}$$

and the mean square error, E_{ms} is defined as

$$E_{ms} = \frac{1}{MN} \sum_{x}^{M-1} \sum_{y}^{N-1} e(x, y)^2 \tag{2.2}$$

where M and N are image dimensions. The PSNR value is then computed as

$$\mathrm{PSNR} = 10\log_{10}\left(\frac{(2^B-1)^2}{E_{ms}}\right) \tag{2.3}$$

where B is the bit depth of the image samples. Larger PSNR interprets as good image quality, that is the output image $g(x, y)$ is closer to the input image $f(x, y)$. PSNR is the most commonly used objective metric for video quality assessment. However, objective quality measurements do not always give reliable information about perceivable image quality [7]. Nevertheless, implementations of objective assessments are much faster, repeatable, and convenient; owing to these merits, objective measurements are widely used.

2.1.5 Video encoder

A video encoder converts uncompressed raw digital video data into a compressed format. Modern video encoders use both predictive and transform domain techniques. A simplified block diagram of a video encoder is given in Fig. 2.3. Since the coding scheme consists of both predictive and transform coding techniques, the scheme is called hybrid.

The first frame of a video (and also some random accessing points in the video sequence) is encoded using only intra-picture prediction, i.e. using only spatial correlations and no dependence on other pictures. All other frames can use intra-picture prediction as well as inter-picture prediction techniques, where inter-prediction exploits temporal redundancies across frames. The input frame is compared with its predicted frame; usually the input frame is subtracted from the predicted frame, and the result is called a residual image. The residual image is then subjected to a linear spatial transform, generally a DCT because of its high energy-compaction property, and most of the signal information tends to be concentrated in a few low-frequency components. After DCT operation quantisation is performed over the transformed image. This is essentially removing high-frequency components thus minimising psycho-visual redundancy in the residues. Note

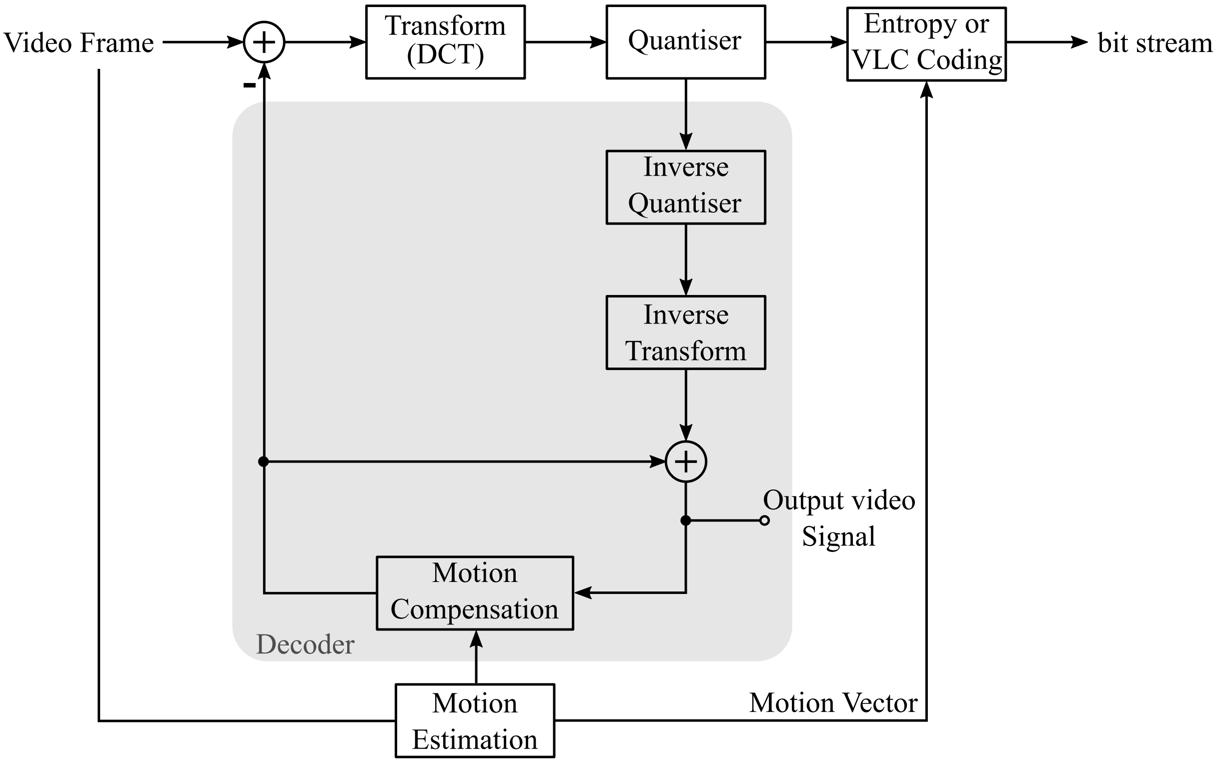


Fig. 2.3: Simplified block diagram of a video encoder.

that human eyes are more sensitive to low-frequency (smooth changes) components than high-frequency ones. The quantisation is controlled by a quantization parameter (QP); the information lost in this step is not recoverable but may not affect the perceivable image quality. After quantisation the data along with the prediction information are encoded using lossless compression methods such as entropy coding or variable-length coding (VLC).

A hybrid encoder also contains a decoder processing loop which is shown as the grey-shaded box in Fig. 2.3. Therefore the final picture representation, which is equal to the output of the decoder, is constructed in the decoder loop by subjecting residual picture to de-quantisation, inverse transform, and adding to the predicted picture from the motion estimator; this is stored in a picture buffer in order to use prediction for subsequent pictures. Generally, the order of encoding or decoding of pictures frames is different from the order in which they arrive from the source, or output order (display order).

2.2 High efficiency video coding (HEVC/H.265)

High Efficiency Video Coding (HEVC) or H.265 is the latest video compression standard proposed by an joint effort of the ITU-T video coding experts group (VCEG) and the ISO/IEC moving picture experts group (MPEG) standardisation organisations. Previously, the two organisations jointly produced the MPEG-2 Video/H.262 [8] and AVC/H.264 [9], and they had a very strong impact on the history of video standards and were applied in wide variety of products. All these proposals aimed to maximise the compression capability and improve many other factors such as robustness to data loss while considering the computational resources available in products for the estimated deployment time of each standard. As with the progress of computational power available in computer chips, every new proposal brings coding advantages and more processing power requirements.

Like most video coding standards, HEVC/H.265 is also a block-based video coding, where each video frame or image is split into square blocks of pixels, and then each block is predicted by either intra-picture prediction or inter-picture prediction. The latter prediction method generally compensates motion of real-world objects; it is also called motion-compensated prediction. In either case, the resulting prediction error of each block is subjected to DCT, quantisation and entropy coding. A typical HEVC/H.265 encoder block diagram is shown in Fig. 2.4, and is similar to a generic hybrid video encoder.

2.2.1 Coding block and coding tree units

In AVC/H.264 each video frame is divided into blocks of size 16×16 pixels, called macro blocks. Instead of macro blocks, HEVC/H.265 uses a coding tree unit (CTU) of size

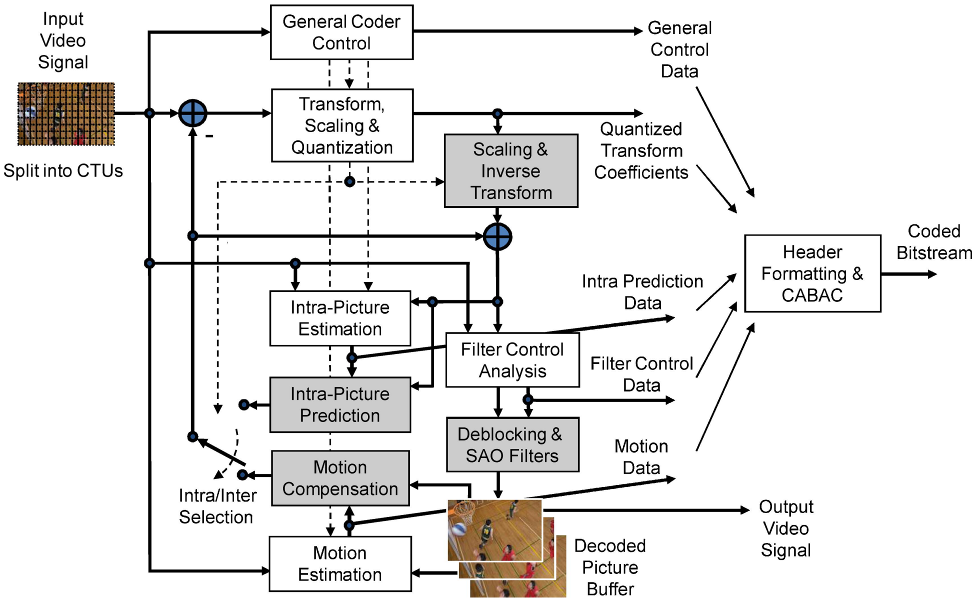


Fig. 2.4: HEVC Encoder (with decoder modelling elements shaded in light grey) [10].

$L \times L$, where L can take values of 16, 32, or 64, selected by the encoder. Larger blocks typically provide better compression, as it may possible to encode larger areas with fewer codes. For colour representation HEVC/H.265 generally uses a YCbCr colour space with 4:2:0 sampling, although later versions included 4:2:2 and 4:4:4 schemes, as the human eye is more sensitive to luma (Y component) than chroma (Cb, and Cr). The two chroma components Cb, Cr represent the blue-difference and red-difference from luma respectively. A luma coding tree block (CTB) and two corresponding chroma CTBs form a CTU.

The blocks used for luma and chroma CTBs can be partitioned into one or multiple CBs in a quad tree structure as shown in Fig. 2.5. Mostly partitioning applies to both luma and chroma CTBs simultaneously, with an exception when a certain minimum size is approached for chroma. The splitting process can be iterated until it reaches a block size 8×8 pixels or larger, selected by the encoder, for luma samples. One luma CB and associated chroma CBs together with syntax elements form a coding unit (CU). Every CU has prediction units (PUs) and transform units (TUs) partitions.

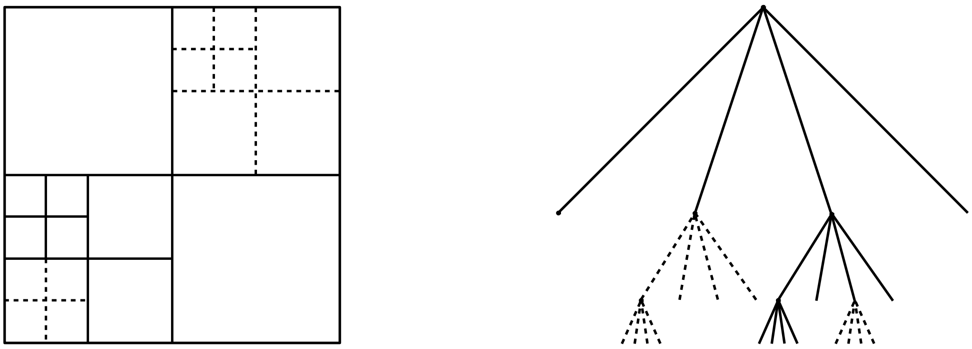


Fig. 2.5: Subdivision of a CTB with its partitioning and corresponding quad-tree structure. CBs represented in solid lines and TBs in dotted lines.

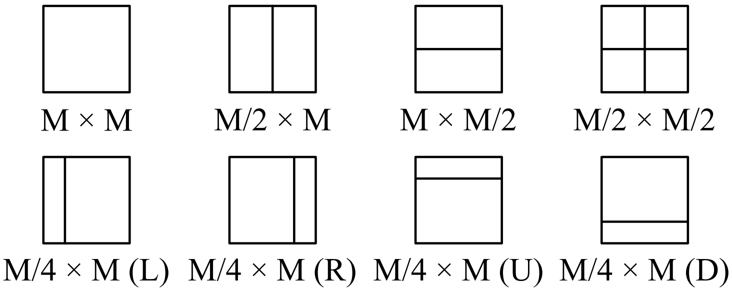


Fig. 2.6: Splitting coding blocks (CBs) into prediction blocks (PBs); the lower four are called asymmetric motion partitioning (AMP).

The prediction mode for a CU can be either inter or intra prediction, and is decided at CU level. The luma and chroma CBs can be split further and predicted by PBs. The CB can be split into one, two or four PBs as illustrated in Fig. 2.6. For the intra-picture prediction mode only square blocks ($M \times M$ and $M/2 \times M/2$) are supported. The lower four partitioning types of PBs are known as asymmetric motion partitioning (AMP) and are only allowed for blocks having a size of 16×16 or larger. Some other restrictions are also applied to PB partitioning for saving worst-case memory bandwidth; for example, 4×4 partitions are not allowed in inter-prediction and 4×8 and 8×4 blocks are restricted to uni-predictive coding. A luma and two chroma PBs with associated coding syntax form a PU.

Table 2.1: Comparison of supported prediction block sizes in different generations of video coding standards.

Video coding standard	Supported prediction block sizes
MPEG-2 Video/H.262	16×16
H.263	16×16, 8×8
MPEG-4 Visual	16×16, 8×8
MPEG-4 AVC/H.264	16×16, 16×8, 8×16, 8×8, 8×4, 4×8, 4×4
HEVC/H.265	64×64, 64×48, 64×32, 64×16, 48×64, 32×64, 16×64, 32×32, 32×24, 32×16, 32×8, 24×32, 16×32, 8×32, 16×16, 16×12, 16×8, 16×4, 12×16, 8×16, 4×16, 8×8, 8×4, 4×8, 4×4

In the development of coding standards from one generation to the next generation, a key aspect of improving coding efficiency is to increase the number of supported prediction block sizes. As an illustration, a comparison of different video standards is given in Table 2.1. The early generations of video coding standards such as MPEG/H.262 support only one block size, of 16×16 pixels without any subdivisions, and over the generations more partitioning types and larger block sizes are allowed by the standards.

Transform blocks (TBs) are used for residual transform coding. Luma or chroma TBs can take the size of a CB or be further split into luma and chroma TBs. Integer DCTs of sizes 4×4, 8×8, 16×16, and 32×32 are defined for transform coding, also an integer transform derived from the discrete sine transform (DST) is specified for 4×4 intra-predictive residuals as an alternative transform. The splitting is implicit if the CB size is greater than the maximum TB size. The HEVC/H.265 specification allows TBs to span multiple PBs for inter-picture predicted CUs, for maximising coding efficiency. The variable block sizes in transform block sizes enable adaptation of different space frequency characteristics of CBs. Larger transform block sizes provide better frequency resolution, whereas smaller block sizes provide better spatial resolution. The trade-off between these two is chosen by the encoder control.

2.2.2 High-level parallelisation features of HEVC/H.265

Multi-threading and parallel processing are generic features of today's microprocessor, even in low-power platforms. Parallelisation features enable real-time processing of high-quality video such as UHD in mobile devices, not possible otherwise. Efficient multi-threading is only feasible when the video coding standards support this feature. In order to overcome difficulties of parallelisation approaches in previous standards, HEVC/H.265 introduces two new features, tiles and wavefront parallel processing (WPP); both allow partitioning video into groups of CTBs and processing them independently [11, 12].

If tiles are enabled the picture is divided into rectangle groups of CTBs with horizontal and vertical boundaries. Dependencies between tiles across the boundaries are not allowed, including motion-vector prediction, with an exception for in-loop filtering. Since tiles are not dependent on each other, the encoder can process tiles in parallel. Tiles in the 'active region', where the picture has more motion, need more processing time. As an advantage of tile segmentation, the computation time over different processing elements could be balanced using an adaptive tile-size selection. In previous standards slices were used to achieve parallelism, but this has some disadvantages like improper pixel segmentation which quite often reduces pixel correlations; also the slices contain header information, and the overhead introduced by these slice headers is not negligible in coding standards with high efficiency, such as HEVC/H.265. Experimental studies reveal that tile-based parallelism has an average luma bit-rate saving of from 2.2% to 5.5% over the slice-based approach [11].

In WPP, each row of a CTB is considered as a separate partition, and a number of parallel processing threads up to the number of CTB rows can be used in WPP. The entropy coding dependencies are propagated to the next row with two CTB delays, so forming a diagonal ‘wavefront’-like processing. One of the advantages of WPP over tiles is that it does not break coding dependencies over boundaries, thus reduces block artifacts at the partition boundaries. Since WPP has little effect in the analysing methods and compression, it has a subtle effect on coding efficiency. However when processing of a row is finished and no more rows are available in the frame, the processor becomes idle. This problem can be mitigated by overlapping the processing of consecutive pictures, as with overlapped wavefront (OWF) [12]. It seems that coding losses or coding inefficiency due to tiles and WPP decrease with increasing frame size or resolution. Empirical results show that both WPP and OWF have 0.58% coding losses on average for 4K UHD (2160p) videos whereas tiles suffer 2.17% average coding losses [12].

2.2.3 Intra-picture prediction

Intra-picture prediction in HEVC/H.265 techniques can be classified into two categories, angular prediction methods and a second category consisting of planar prediction and DC prediction. HEVC/H.265 supports 35 directional modes in these two categories, compared to the 8 directional modes in AVC/H.264. This in turn increases the coding efficiency as well as the complexity of the intra-picture prediction. All prediction modes use adjacent reference block samples, two 1-D arrays of upper and left neighbouring pixels which are twice as long as the intra block size, for the prediction. The DC prediction mode simply uses a single value, and planar estimate as a smooth gradient of the neighbouring samples.

2.2.4 Inter-picture prediction

Inter-picture prediction exploits temporal redundancies across neighbouring frames. As mentioned earlier the picture is divided into CTUs and then PBs. Assume that an object

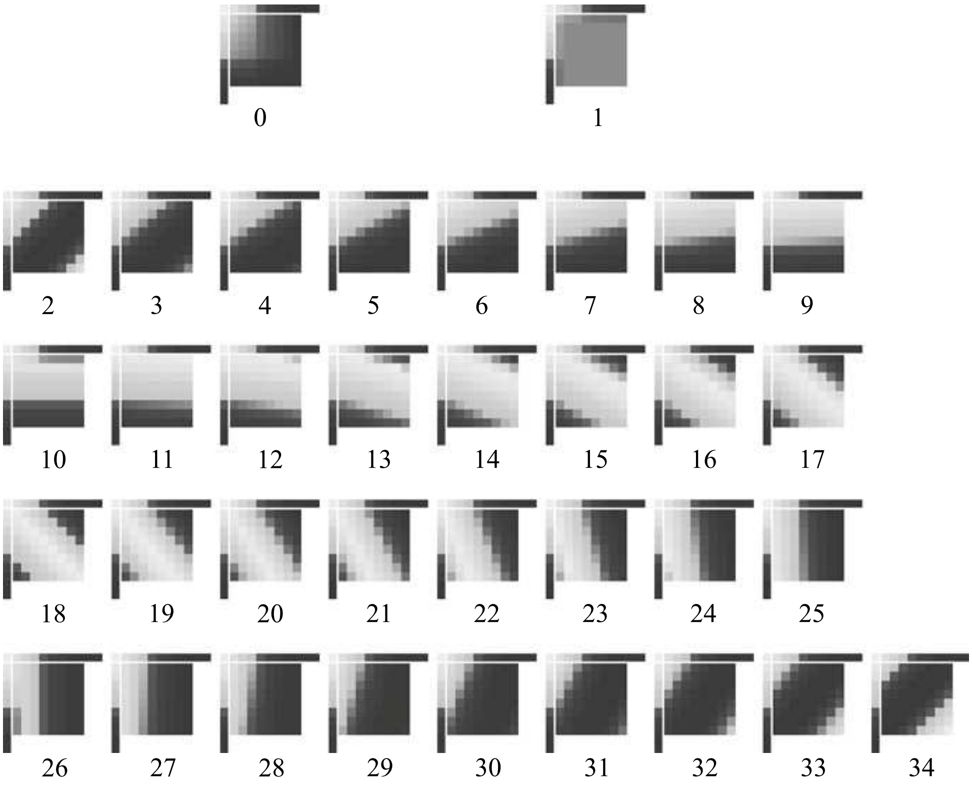


Fig. 2.7: Examples for HEVC intra-prediction for an 8 × 8 luma block, where 0 is the planar mode, 1 is the DC mode and the rest are angular predictions mode [13].

is moved in a video sequence, and the motion estimation finds the best matched block from a previously encoded frame, which acts as a predictor. The positions of the block in previous frames are indicated by a motion vector (Δx, Δy), where Δx and Δy represents horizontal and vertical movements respectively.

Motion estimation is an important operation in video coding and also contributes most of the computation complexities in video encoding. HEVC/H.265 introduces a quad-tree structure and many additional partitioning modes for PB. The target is to reduce the bits requirement for representing motion vectors and prediction errors or residues. Motion estimation can be classified into two types, forward and backward motion estimation, according to the position of the reference frame – whether it is a future or past frame on the temporal axis. In forward motion estimation a current frame, which is currently encoding, references a frame from the future, implying that search is forward, and vice versa for backward motion estimation.

The SAD is widely used as a criterion for estimating the cost function, because of its simplicity. For an $M \times N$ block of pixels SAD is defined as

$$\mathrm{SAD} = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |C(i,j) - R(i,j)| \tag{2.4}$$

Since the encoder has to choose an optimal combination of coding options from parameters like partitioning types, and number of bits required for both motion-vector coding and residuals; a Lagrangian (J_{mv}) cost function is used to select best motion vector

$$J_{mv} = \mathrm{SAD}(MV) + \lambda \cdot b(MV - PMV) \tag{2.5}$$

where λ is the Lagrangian multiplier, MV is the motion vector obtained, PMV is the predicted motion vector of the block, and $b(MV - PMV)$ is denotes the number of bits required to represent the motion vector difference $(MV - PMV)$. The vector of the block having the minimum cost function is selected as the motion vector. Note that the predicted frame many not have the same pixel values as the corresponding regions in the current frame. Therefore the difference between the predicted frame and the current frame, the residues, are sent to the decoder after transform coding and further processing.

Generally there are two kinds of inter-prediction techniques used in HEVC/H.265, namely unidirectional and bidirectional predictions. In unidirectional predictions ('P' frames) only backward motion estimation is used for prediction, whereas both forward and backward motion estimations are used for bidirectional predictions ('B' frames). Examples of coding using both these techniques with a group of pictures (GOP) of 4 are shown in Fig 2.8, where the picture order count (POC) is the display order or the order in which they arrive from the source, and it is different from the decoding or encoding order.

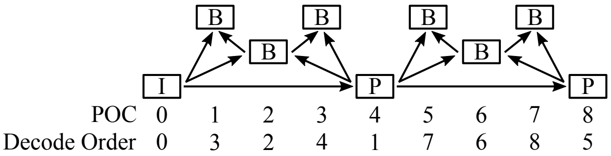


Fig. 2.8: A hierarchical GOP coding structure with size 4.

Merge and skip modes

A merge mode similar to the one in AVC/H.264 is also included in HEVC/H.265. The CB, if encoding as merge mode, takes its motion estimation parameters from its spatially or temporally neighbouring blocks. HEVC/H.265 also incorporates a skip mode, which is a special case of merge mode, and the block is considered to be encoded in skip mode if the motion vectors difference between the current encoding $M \times M$ PU and neighbouring blocks is zero, and if all the residuals are quantised to zero. Skip mode requires a minimum amount of bits to encode, and homogeneous and motionless regions are the best candidates for skip mode.

Fig. 2.9 shows CTB structures and motion vectors of a video frame encoded in HEVC/H.265, showing the 3^{rd} picture frame of the video sequence 'bus' encoded as a 'B' frame. The frame is used almost every kind of mode and partitioning types available in HEVC/H.265, and also it is quite clear that most stationary regions are encoded with skip mode.

2.2.5 Transform coding

In block-based hybrid video compression the residual image, i.e. the resulting image after subtracting the inter/intra predicted picture frame from the current encoding frame, is subjected to transform. Then quantisation is applied to the resulting transform coefficients, which is essentially a division by a parameter called quantisation step (Q_{step}), followed by rounding. Even though the rounding removes some of the information, it does

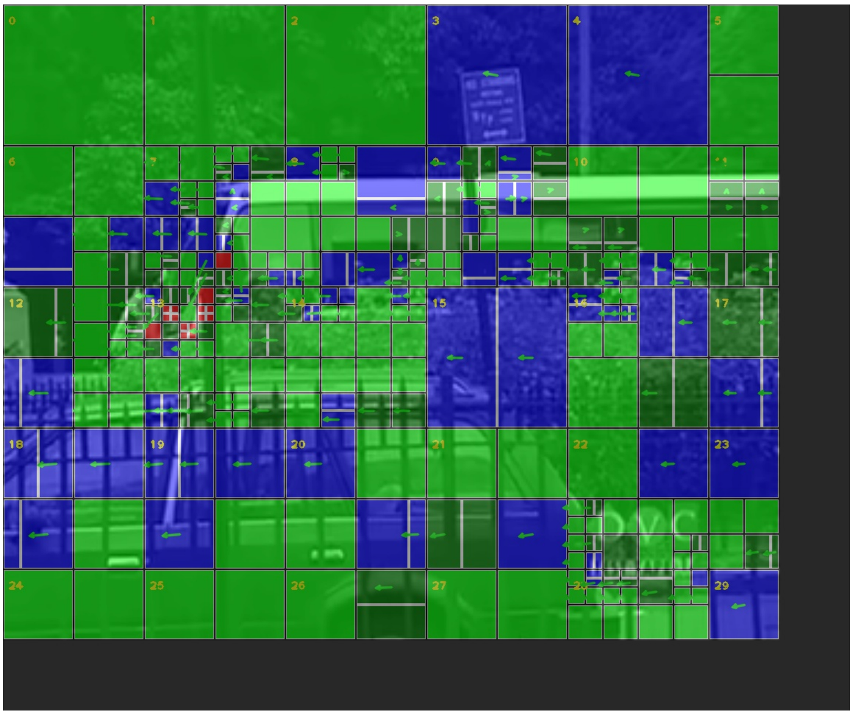


Fig. 2.9: CTB structure and motion vectors of the 'bus' sequence frame 3, where intra blocks are in red, inter blocks are in blue, merge-mode blocks are in dark green and skip-mode blocks are in light green.

not affect the perceived quality as mentioned before. At the decoder side, the residual picture frame can be created by applying de-quantisation and the inverse transform.

HEVC/H.265 specifies DCT transforms of sizes 4 × 4, 8 × 8, 16 × 16 and 32 × 32, and for efficient computation, HEVC/H.265 uses integer approximations of DCT II. A 4 × 4 DST transform is also specified for encoding of 4 × 4 intra-predicted blocks. The DCT transforms are called core transforms in HEVC/H.265 to differentiate it with DST [10]. HEVC/H.265 tries to preserve symmetric properties; the even rows have symmetry whereas odd rows maintain an anti-symmetric property; this property is useful to reduce the number of computations. As an example, 4 × 4 matrix coefficients are in the form:

$$\mathbf{T}_4 = \begin{bmatrix} \mathbf{t}_0 \\ \mathbf{t}_1 \\ \mathbf{t}_2 \\ \mathbf{t}_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_0 & a_0 & a_0 \\ b_0 & b_1 & -b_1 & b_0 \\ a_0 & -a_0 & -a_0 & a_0 \\ b_1 & -b_0 & b_0 & -b_1 \end{bmatrix} \tag{2.6}$$

Although HEVC/H.265 relaxed the orthogonality constraint for approximating to integer values, the deviation is small and the effect is negligible for the overall transform and quantisation process. The coefficients of inverse transform matrices can be obtained by transposing the forward DCT transform matrices.

2.3 Block matching motion estimation algorithms

The larger CTU size and the quad-tree encoding structures bring most of the coding efficiency present in HEVC/H.265. It is possible to encode stationary or homogeneous regions and some object movements, in larger blocks which results in smaller overhead, in particular for high-resolution videos as portions of objects are represented with more pixels than in low-resolution videos. However, the computations increase drastically due to these changes. Consider the CTU of 64 × 64 pixels, which has 25 partitioning types (see Table 2.1), and for the worst-case scenario it is necessary to find motion vectors for all these available partitioning types. Each of these blocks needs much computation to find a motion vector from the predefined search area. As an example, an 8 × 8 block needs 63 additions/subtractions to find the SAD of one search position. Thus for searching an area of 128 × 128 (±64 in horizontal and vertical) requires 1032192 additions, and larger PBs like 64 × 64 necessitate 67092480 additions; note that a UHD of resolution 3840 × 2160 pixels has 2040 such CTUs. In other words HEVC/H.265 is a good performer regarding video quality and compression efficiency; nonetheless, it suffers very large computational complexity as it has many partitioning types and the quad-tree structure.

A motion vector (MV) expresses the displacement of a pixel block or pixels with respect to its position in another picture frame, and the block matching technique, which finds the motion vector of a block of pixels, is the most popular. Since the neighbouring frames of a video are very close along the temporal axis, the motions of objects are usually confined to a small area, and the motion search is conducted in this particular region, called the search area or search window. The height and width of the search area depend on motions in video, and typically the search area is kept wider since a panning motion is more common in videos. The block-matching algorithms can be broadly classified into full-search algorithms and fast-search algorithms.

2.3.1 Full-search block-matching algorithm

The full-search block-matching algorithm always comes up with the best results, as it searches all possible locations in the search window, and thus provides the least possible residues. However, the required computations are too high since it has $(2 \times W_x + 1) \times (2 \times W_y + 1)$ search points, where $\pm W_x$ and $\pm W_y$ are the search range in the x and y directions respectively. Even the simplest distortion metric such as SAD involves extensive computations for a search position, and it requires to find cost function at each search candidates; the full search is extremely time consuming. Algorithms that reduce search points and hence decrease the computational complexity are called fast-seach block-matching algorithms.

2.3.2 Fast-search block-matching algorithms

Even though fast-search algorithms omit several search points, many algorithms are as good as full-search methods in terms of PSNR and bit rate, such as the successive elimination algorithm (SEA) [14], and partial distortion elimination (PDE) [15]. On the other hand, there are fast-search algorithms that have very much improvement in computation speed but with slight decreases in PSNR and/or bit rate.

Successive elimination algorithm

The SEA is a two-step algorithm; first the algorithm computes the absolute difference between the sums (ADS) of current and reference block pixels, as given below

$$\mathrm{ADS} = \left| \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} C(i,j) - \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} R(i,j) \right| \tag{2.7}$$

where M and N are the width and height of the pixel block respectively, C represents current block pixels, and R represents reference block pixels. In the second step, the algorithm eliminates computation of SAD for blocks based on the inequality given in (2.8) since, for any two blocks of equal size, the ADS is always less than or equal to their SAD.

$$\mathrm{ADS} \leq \mathrm{SAD} = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |C(i,j) - R(i,j)| \tag{2.8}$$

Hence in searching for a better matching block, it is only required to consider candidate blocks having ADS less than the current minimum ADS. Although there is an additional overhead of computing the sum of intensities of $M \times N$ pixels at candidate blocks, the omitted SAD computations account for more, and the computations in SEA are reduced by approximately 85% compared to the full-search algorithm [14].

Multilevel successive elimination algorithm

The multilevel successive elimination algorithm (MSEA) [16] extends the concepts of the above SEA algorithm into a multilevel case. The efficiency of the SEA depends on the gap between ADS and SAD in each search position. The MSEA algorithm partitions each blocks into several sub-blocks in hierarchical levels, for instance, the block is partitioned into four sub-blocks, then each sub-block partitioned into four sub-blocks until the size of sub-blocks becomes 2×2. An inequality condition similar to (2.8) is tested in each level. The SAD is calculated only if it satisfies the inequality conditions at all levels.

Partial distortion elimination algorithms

In PDEs [15, 17, 18] the distortion calculation is stopped whenever the computed distortion is greater than the minimum obtained before. The computational savings is from the probability of early rejection of non-possible candidate motion vectors. Like SEA algorithms, the PDE algorithms also reduce the number of computations involved in motion estimation. Note that in these approaches calculation is considered as a sequential process, thus may not be a good candidate for highly parallel environments such as hardware implementations.

Hierarchical fast-search algorithms

In hierarchical algorithms the search window is down-sampled to find the motion vector in the down-sampled area, typically uses an hierarchical technique as in [19]. After the motion search in the topmost search window, the results are passed to the lower levels for refinement. The hierarchical motion estimation is often combined with other techniques. For instance, a Kalman filter is used in [19] to get low-resolution hierarchical levels, and in [20], an adaptive sampling positioning technique is used for down-sampling.

Three step search algorithm

The three step search (TSS) algorithm [21] is one of the oldest fast-search algorithms proposed for motion estimation. The algorithm can be explained as follows; it starts the search taking the collocated block as the search centre. Initially with the step size ‘S’ taken as 4, the algorithm searches eight locations $\pm S$ around the location (0, 0), the collocated block. In the next step, it picks a location which has the least cost function among the nine locations including the centre and sets this as the new search centre for the next step. The algorithm continues the above procedure, taking the step size $S = S/2$, until the step size becomes one, i.e. the third step.

2D logarithmic search

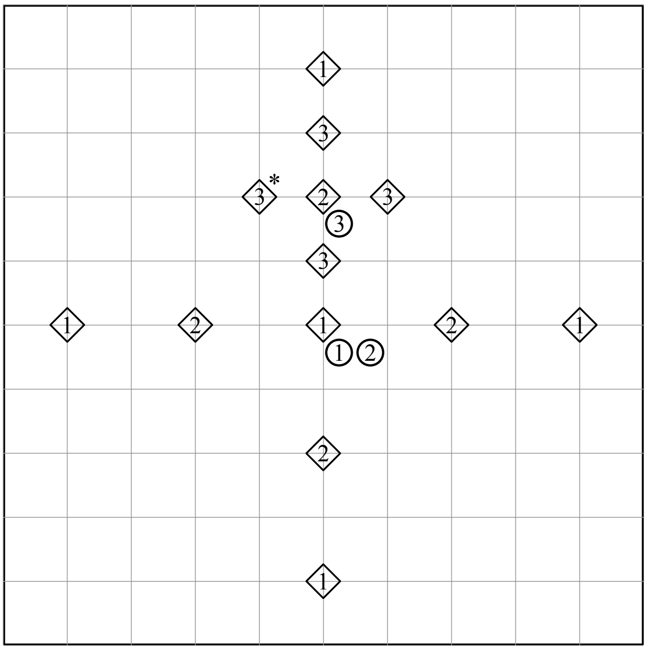
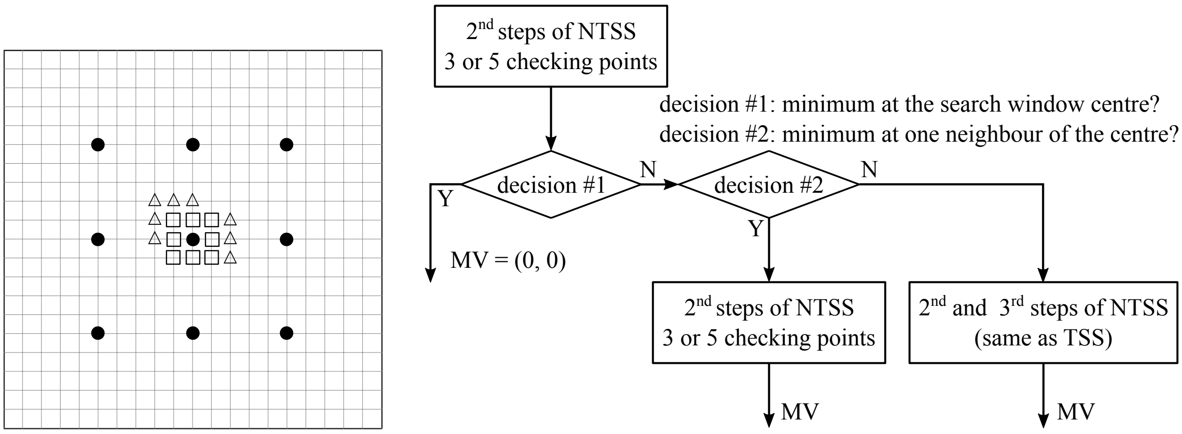


Fig. 2.10: A 2D logarithmic search procedure for finding minimum distortion, $\Diamond$ with the step number inside indicates the searched positions, the number in a circle shows the location of the minimum in each step, and '$*$' shows the final minimum.

The 2D logarithmic search [22] is similar to TSS but is more accurate to find motion vectors in large search areas. The algorithm illustration is in Fig. 2.10, which starts with a search location at the centre and takes a step a size, for example $S = 4$ initially, and searches 4 locations at the distance S on the x and y axes. The location of the minimum cost function is set as the centre for the next search with $S = S/2$. The procedure continues until S becomes one.

New three-step search algorithm



(a) (b)

Fig. 2.11: NTSS algorithm. (a) Showing search points, where filled circles are the checking points in TSS and squares are the additional 8 points in the first step of NTSS, and the triangle shows the second step if the minimum is at one of the eight neighbours. (b) NTSS flow chart [23].

The new three-step search algorithm (NTSS) is an improved version of TSS; it assumes that the error surface is monotonic in a small area near the global minimum [23]. The algorithm also assumes that the block motion field is gentle, smooth and varies slowly, in particular for low-bit-rate video applications. Hence there is proposed a central-biased checking in its first step, which adds eight points additional to the TSS algorithm, and a halfway stop for stationary or quasi-stationary blocks. NTSS was one of the first widely accepted fast-search algorithms and was used very often in implementations of earlier standards such as MPEG 1 [24]. Fig. 2.11 explains NTSS with a flow chart.

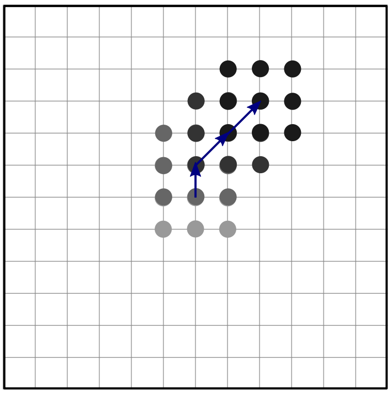


Fig. 2.12: Illustration of block-based gradient-descent algorithm; the arrows demonstrate movements of the search centre.

Block-Based Gradient-Descent Search Algorithm

The block-based gradient descent algorithm (BBGDS) [25] also assumes monotonic error surfaces near the global minimum. The concept is similar to that of the gradient-descent algorithm, widely used in optimisation theory. The first step is similar to TSS, search around the centre point. If the optimum is found at this centre, the search stops, otherwise continue the search around the point where the optimum is found. It will continue until the minimum is at the centre of the checking points, or when the block of checking points reaches the search window boundary. The search procedure of the BBGDS algorithm with a 3×3 checking block, is illustrated in Fig. 2.12. The BBGDS always moves in the direction of optimal gradient descent, but there is always a chance of trapping in local minima.

Diamond search algorithm

Searching with a small pattern, such as the 3×3 checking block in BBGDS [25], causes the search to get trapped in local minima and deteriorates performance, in particular for picture frames having large motion content. On the other hand, a larger search pattern like in TSS [21] 9 × 9 sparse checking points, is quite likely to lead the search in a wrong direction. The diamond search (DS) [26] algorithm uses two search patterns, the first one comprises 9 checking points forming a large diamond shape, and the second one consists of 5 checking points forming a small diamond search pattern. The search patterns used for DS are shown in Fig. 2.13. The search proceeds similarly to BBGDS with the large diamond search pattern until the minimum lies in the centre of the search pattern, then it switches to small diamond pattern, with the minimum point of this selected as the motion vector. Since the search is not constrained by the number of steps, the probability of finding the same motion vectors as obtained in the full search is comparatively high [26].

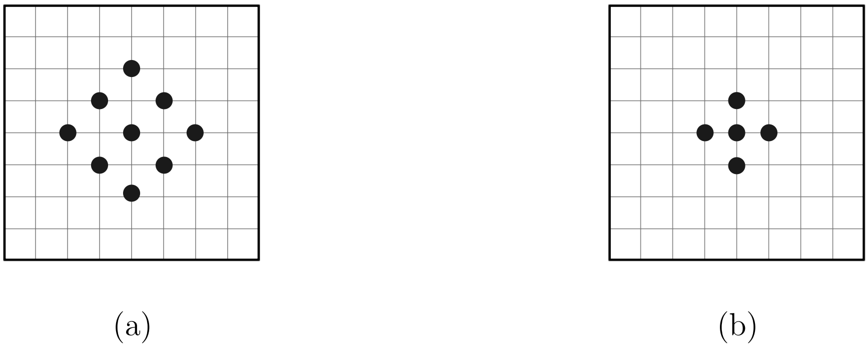

(a) (b)

Fig. 2.13: Search patterns used for the diamond search algorithm. (a) Large diamond search pattern. (b) Small diamond search pattern.

Predictive motion-estimation search algorithms

A good technique to reduce the computation complexity in motion estimation is to start the motion search near the possible final location of the motion vector. The prediction-based algorithm predict this point from already-encoded motion vectors, for example, the spatially nearest blocks. The algorithm may incorporate more than one method and select the one which has the least cost function. One of the earliest prediction methods is the median predictor, which is the median of the left, right and top (right or left) block's motion vectors, as these blocks may be already encoded and can be used for prediction. For example, the motion-vector field-adaptive fast motion estimation (MVFAST) [27] and predictive motion-vector field-adaptive search technique (PMVFAST) [28] algorithms, both accepted by the MPEG-4 optimisation model [29] as a recommendation for motion estimation, use median predictor and temporal candidates.

The advanced predictive diamond zonal search (APDZS) algorithm [30] considers a set of predictors, such as the left, top, top-right block's motion vectors, their median and the motion vector of the previous frame collocated block. Some other likely locations, such as linear merging of temporally and spatially adjacent blocks, are also considered. From these predictors, the one having the least distortion is selected as the centre for a diamond pattern search. APDZS also uses an adaptive thresholding for early termination of the motion search.

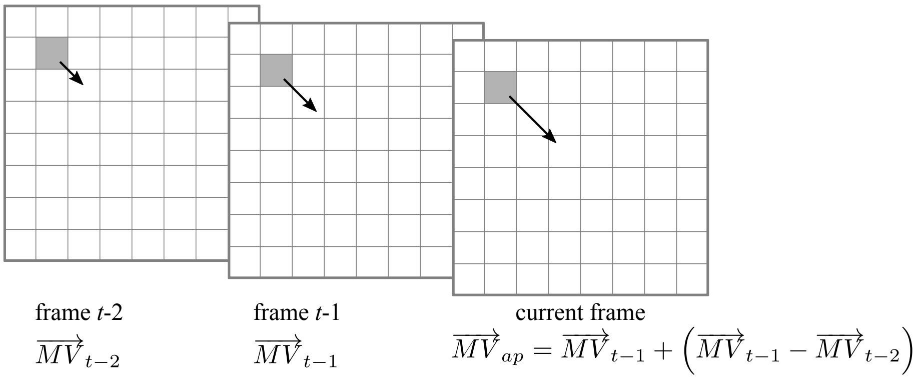

frame *t*-2
$\overrightarrow{MV}_{t-2}$
frame *t*-1
$\overrightarrow{MV}_{t-1}$
current frame
$\overrightarrow{MV}_{ap} = \overrightarrow{MV}_{t-1} + \left(\overrightarrow{MV}_{t-1} - \overrightarrow{MV}_{t-2}\right)$

Fig. 2.14: Use of acceleration information for motion vector prediction in EPZS algorithm [31].

Enhanced predictive zonal search (EPZS) [31] is an improvement of the PMVFAST and APDZS algorithms, using an additional set of predictors and efficient selection of the criteria for early stopping. The prediction selection is the key feature of this algorithm; the speed of convergence and the accuracy results depend on the prediction. The

EPZS algorithm uses an accelerated motion vector predictor $\overrightarrow{MV}_{ap}$ (Fig. 2.14), which is the increased or decreased motion vector depending the motion vector of the collocated frame and the frame before that. The threshold value for early termination in the EPZS algorithm is adaptive and derived from the neighbouring block's cost values.

Test zone search algorithm

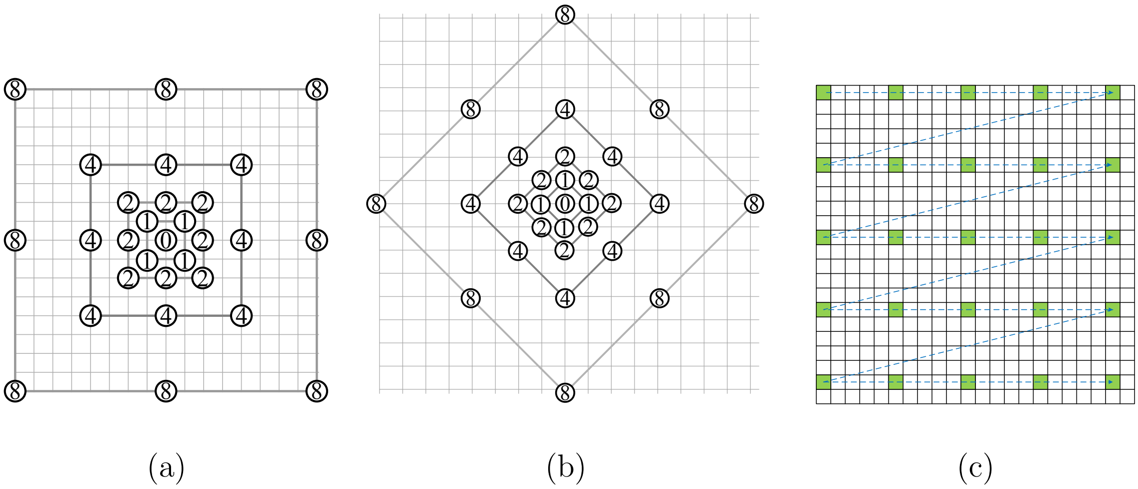

(a) (b) (c)

Fig. 2.15: Search patterns used in test zone search algorithm. (a) Square pattern. (b) Diamond pattern. (c) Raster scan pattern

The TZ search algorithm [32] is widely used for motion estimation in HEVC/H.265 applications, and the algorithm is also used in HEVC test model (HM) software [33]. The algorithm consists of an initial grid search and raster search which is a full search like searching but with a down-sampled search window. The TZ search algorithm uses median predictor, left predictor, and upper-right predictor, where the starting point for the initial grid search is the minimum distortion point of these predictors. For the initial grid search the algorithm uses either a square pattern or a diamond pattern with stride length changing from 1 to the 'search length' in powers of two. An example of search patterns for the initial grid search of stride length up to 8 are shown Fig. 2.15 (a) and

Fig. 2.15 (b). If the minimum distortion point obtained in this step is far from the starting point, i.e. greater than a predefined value 'iRaster', the algorithm will go to the next step, the raster scan; otherwise it will be skipped.

In the raster search, the algorithm will search in the whole window with a stride length equal to the previously defined value 'iRaster'. An illustration of raster scan with stride length equal to 5 pixels is in Fig. 2.15 (c). This step is the most time-consuming in a TZ search; the total search points S for the window of size $W_x \times W_y$ pixels is given by

$$S = \left\lceil \frac{W_x}{iR} \right\rceil \cdot \left\lceil \frac{W_y}{iR} \right\rceil \tag{2.9}$$

where iR represent the 'iRaster' value.

The final step in the TZ search algorithm is a refinement search; either raster refinement or star refinement is enabled. Both of these refinements use an eight-point diamond or square pattern, and they only differ in their search operation. The raster refinement brings down the stride length of the square/diamond pattern by a factor of two while changing the search centre to the best location in every step. The star refinement is similar to the initial grid search except for the change of search centre after every round. The two refinements stop when the distance from the search centre to the minimum distortion point becomes zero. Hence the total number of search points is not predictable as it depends heavily on the video sequence.

2.4 Residue number systems

Video processing involves a large number of computations, and in their hardware implementations in traditional binary systems their performance is usually restricted by the carry propagation, and it is worsened as the word length of arithmetic operations increases. If large numbers could be split into smaller numbers and processed independently, then the performance can be enhanced. RNSs represent large integers in a smaller

Table 2.2: Residues of various moduli sets

X	Relatively prime moduli			Relatively non-prime moduli		
	$m_1 = 2$	$m_2 = 3$	$m_3 = 5$	$m_1 = 2$	$m_2 = 3$	$m_3 = 4$
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	0	2	2	0	2	2
3	1	0	3	1	0	3
4	0	1	4	0	1	0
5	1	2	0	1	2	1
6	0	0	1	0	0	2
7	1	1	2	1	1	3
8	0	2	3	0	2	0
9	1	0	4	1	0	1
10	0	1	0	0	1	2
11	1	2	1	1	2	3
12	0	0	2	0	0	0
13	1	1	3	1	1	1
14	0	2	4	0	2	2
15	1	0	0	1	0	3

set of integers, and mathematical operations can be performed on each of these smaller integers independently, thus generally outperforming its binary equivalent.

The RNS is a non-weighted number system, where a number is represented by the remainders of a moduli set, called residues. An ordered residues set $(x_1, x_2, \ldots x_N)$ represents the binary number X corresponding to a moduli set $(m_1, m_2, \ldots m_N)$, where $N \geq 2$, and each residue x_i is the remainder from the integer division of X by m_i, mathematically $x_i = X \mod m_i$, or more conveniently $x_i = |X|_{m_i}$. If the moduli are mutually prime then the product of these moduli M is the dynamic range of the system, since any number X less than M can be represented uniquely using the residues, whereas if they are relatively non-prime then the dynamic range M will be the least common multiple (LCM) of all the moduli.

Table 2.2 shows an example for both relatively or mutually prime moduli and non-prime moduli residues of binary number X. The moduli 2, 3, and 5 are relatively prime; the residues of this moduli set given in the left half of the table, and the residues of a mutually non-prime moduli set, 2, 3, and 4 are provided in the right half of the table. For the first case, the dynamic range $M = \prod m_i$ which is equal to 30, hence any number less than this has a unique representation by residues. On the other hand, the moduli set of the right-half side of the table is not mutually prime, hence $M = \mathrm{lcm}\, m_i$ and the dynamic range is 12. Observe that there are repetitions on the right-half side of the table after $X \geq 12$.

The residue number system using mutually prime moduli can be considered as a 'standard residue number system', and is widely used. Nevertheless, 'non-standard residue number systems' are also useful in many applications, for instance, in error-tolerant systems using redundant residue number systems. The digit positions with errors may be excluded in such systems while still retaining sufficient dynamic range for the application. Moreover, error detection and correction are also possible with redundant moduli.

2.4.1 Arithmetic operations in RNS

The basic arithmetic operations such as addition, subtraction and multiplication can be easily implemented in an RNS. The addition or subtraction operations are carried out independently in each of these residues and do not need to propagate carries between these residues. For a given moduli set $\{m_1, m_2, \ldots, m_N\}$, $X = \{x_1, x_2, \ldots, x_N\}$ and $Y = \{y_1, y_2, \ldots, y_N\}$ then the addition of $Z = X + Y$ is given as

$$Z = |X + Y|_M = \{|x_1 + y_1|_{m_1}, |x_2 + y_2|_{m_2}, \ldots, |x_N + y_N|_{m_N}\} \tag{2.10}$$

As an example, consider the moduli set {2, 3, 5}; the binary numbers seven and eight are represented by {1, 1, 2}, and {0, 2, 3}, and adding these two according to the above equation gives {1, 0, 0}, the representation of fifteen.

Subtraction and multiplication are also carried out in a similar way, simply subtracting or multiplying residue pairs, relative to their position to modulus. For example, multiplication can be defined by

$$Z = |X \times Y|_M = \{|x_1 \times y_1|_{m_1}, |x_2 \times y_2|_{m_2}, \ldots, |x_N \times y_N|_{m_N}\} \tag{2.11}$$

As an illustration, with the moduli set {2, 3, 5}, multiplication of seven and two can be achieved by multiplying their residue representations {1, 1, 2} and {0, 2, 2} respectively, which gives {0, 2, 4}, the equivalent of the number fourteen.

However, division in an RNS is not trivial. The basic division operation consists of a sequence of subtractions and magnitude comparisons. Since an RNS is non-weighted, or non-positional, the comparison is difficult. For an illustration, in the moduli set {2, 3, 5}, the residue set {1, 0, 0} represents a value five times that of the residue set {1, 0, 3}, but this is far from obvious. One way of doing division could be converting them to a weighted number system, performing division and converting back. However, the conversion is also a time-consuming process. Despite the recent progress in magnitude comparison, sign detection or conversion [34–37], research on topics such as division is in its early stages, and it is very costly to implement. Therefore an RNS is mainly suggested in applications where addition, subtraction, or multiplication predominates.

2.4.2 Negative numbers

Many real-world applications require numbers represented as both positive and negative. As in binary systems, signed-number representations such as sign-magnitude, radix complement, are possible in RNS as well. However, since it is a non-weighted number representation, the magnitude comparison is much more difficult in RNS, and the sign determination is essentially a magnitude comparison. This is the case even with the signed-magnitude representation, where the sign of a resulting mathematical operation like addition or subtraction is unknown even though the signs of the operands are known.

In signed magnitude, an additional bit is used to represent sign in the RNS, similar to binary. For the complement notation the dynamic range of the system is partitioned into two approximately equal parts, and generally the lower half belongs to positive numbers, and the rest are used for negative. Thus, if the dynamic range M is even then the numbers $[0, M/2-1]$, else if M is odd then the numbers $[0, (M-1)/2]$, are considered as positive numbers, and $-X$ is represented by $\{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_N\}$, where $\bar{x}_i$ is the m_i's complement. The complement or negation of each residue x_i can be found by subtracting it from the moduli $|m_i - x_i|_{m_i}$, similar to finding the 2's complement of an n bit binary number x by subtracting it from the value 2^n.

2.4.3 Moduli selection

The performance of RNSs depends on the effective selection of moduli set. One concern is the dynamic range required for the application; also it should be efficient in the implementation of the required arithmetic operations. Different moduli sets are available [38–41] targeting distinctive applications. Some of the commonly used RNS moduli sets are given below

- $\{2^n-1, 2^n, 2^n+1\}$
- $\{2^{n-1}-1, 2^n-1, 2^n\}$
- $\{2^n-1, 2^n, 2^{n+1}-1\}$
- $\{2^n-1, 2^n, 2^n+1, 2^{n-1}-1, 2^{n+1}-1\}$
- $\{2^{n-3}, 2^n-1, 2^n+1, 2^n+3\}$
- $\{r^a-1, r^b, r^c+1\}$

- $\{r^n - 2, r^n, r^n + 1\}$

Finding the best moduli is basically done heuristically. Nevertheless, for a guaranteed best selection some theoretical analysis and guidelines are available. The most important considerations for a successful moduli set $\{m_1, m_2, \ldots, m_N\}$ given in [42] are

1. The moduli should be mutually prime.
2. Smaller moduli (m_i) are better as they require minimum computation time.
3. The moduli set should allow simple realisation of arithmetic as well as conversion to and from the weighted system. Sets having moduli in the form of $2^{k_1} + 1$, $2^{k_2} - 1$, and one 2^{k_3} are found to satisfy the above requirements.
4. The dynamic range of the moduli set M should be large enough to implement the desired dynamic range to avoid overflows. Although scaling is possible, it is not as easy as in binary systems.

The form of moduli set $\{2^n - 1, 2^n, 2^{n+1} - 1\}$, where n is a positive integer, has simple conversion and arithmetic and is suitable for many applications. However, if the required dynamic range is very large, the size of each modulo m_i increases and the performance of such a moduli set tends to deteriorate. All Mersenne or Fermat numbers [43] moduli with one 2^k result in an unbalanced distribution of the dynamic range, which is not desirable since some channels become too fast compared to others due to the unbalanced bit widths of residues. On the other hand, sets having arbitrary small prime numbers generally result in complicated arithmetic and conversion systems. For example, the modulo 23 requires a complex hardware for modulus operation after the arithmetic's of residues, whereas it is much simpler for modulo 31 or 33 based systems. Probably the arbitrary moduli systems need look-up-table designs using read-only memorys (ROMs), however the delay and area of ROM-based systems could be prohibitive.

The moduli set $\{2^n-1, 2^n, 2^{n+1}-1\}$ consisting only of the 2^n and 2^k-1 forms of moduli, has attained research interest in recent years. The modulo operations, as well as the reverse conversion [44], are efficient. The forward conversion, i.e. finding the remainder, is also simple for 2^k-1 type moduli [45], and the remainder of a 2^n modulus is just the least n bits of the binary number. Furthermore, [46] compares various moduli sets; sets having 3 moduli show the least delay for reverse conversion, and the delay of modular adders and multipliers are also very close to the least among the sets. For the same number of bits, the dynamic range of the $\{2^n-1, 2^n, 2^{n+1}-1\}$ moduli is almost double that of the $\{2^n-1, 2^n, 2^n+1\}$ moduli set.

For video processing applications, a dynamic range of 25 bits is generally sufficient for many modules. Thus in this thesis $\{2^n-1, 2^n, 2^{n+1}-1\}$ with n set to 8 has been used for developing timing-critical modules such as motion-estimation hardware for the video encoder.

2.4.4 Applications of RNS

Due to the absence of carry propagation between residual digits, an RNS has been used mainly for systems where arithmetic operations such as additions and multiplications are dominant. Another important property is the ability to isolate individual digits, which is useful to design fault-tolerant applications. This is more important in extremely dense digital IC designs where the chips cannot be tested completely. One of the major applications of RNS is in digital signal processing (DSP). In DSP, real-world analogue signals are represented by numbers, after sampling and quantization, for processing the data using digital computers. The advantage of DSP includes flexibility in modification and coding of algorithms, stability, repeatability, and relatively easy implementation. However, DSP is computationally intensive, where the majority of operations involves multiplications and accumulations. The multiplication is an expensive operation involving larger chip area or resource utilisation and speed. Since most DSP applications are in real-time systems, the speed of multiplication is very crucial, as it is required to complete a certain set of operations before the new data arrives. Arithmetic operations on large numbers are inevitably time consuming, so transforming data into an RNS can effectively reduce delays in such operations.

In DSP, digital filters are quite important as they are essential in applications such as noise reduction, interpolation, decimation, echo cancellation, and equalisation. Digital filters primarily rely on multiplication and accumulation operations, and can be easily adapted to changes in environment by simply changing system parameters. Furthermore, as these systems are re-programmable, their characteristics are interchangeable without any making any physical changes. DSP filters are commonly classified into finite impulse response (FIR) and infinite impulse response (IIR), and an RNS has been used for both finite and infinite impulse response filters [47, 48]. RNS is also used for several other applications including Fourier transform, discrete wavelet transform, image processing and cryptography [49–52].

2.4.5 HEVC procesor with RNS

An RNS has found an application in video processing also. An RNS-based transform architecture for AVC/H.264 is described in [53], and a video filter with RNS is proposed in [54]. SAD is an essential component in video encoding, and an RNS-based specific processor for computing SAD is proposed in [55]. However, application of RNS to the latest video coding standard, HEVC/H.265, is in its early stage. Furthermore, many improvements to the overall performance of RNS are still possible, in particular for some mathematical operations, such as sign detection, that are difficult to do in an RNS. This dissertation aims to address the research gap and proposes various RNS hardware implementations for video processing in HEVC/H.265 standard and a fast arithmetic unit for sign detection.

The hybrid video coding structure shown in Fig. 2.3 has been used for all video coding standards and recommendations of ITU-T and ISO/IEC MPEG since H.261 [56], and is also followed by this dissertation. Even though the high-level architecture was not changed, the underlying algorithm for each block has been modified and becomes more flexible with each new proposal. The coding scheme is called hybrid as it uses both prediction and transform techniques. A more detailed coding scheme for HEVC/H.265 is shown in Fig. 2.4. This dissertation presents various efficient algorithms and architectures for HEVC/H.265 encoder blocks, in particular for motion estimation, as it is the most computationally intensive task in video encoding, providing a trade-off between cost and performance. This dissertation also incorporates an RNS to enhance the performance of video encoding.

Chapter 3

VLSI Architecture of Full-Search Variable-Block-Size Motion Estimation for HEVC Video Encoding [1]

3.1 Abstract

Motion estimation (ME) is the most computationally intensive task in video encoding. This study proposes a full-search variable-block-size ME for the high-efficiency video coding or H.265 specification. The proposed method reduces memory requirements to a large extent by following a Morton order for data reading and a sum of absolute differences reuse strategy. The data bandwidth demand is also diminished by broadcasting data into

---

[1] N. C. Vayalil, and Y. Kong, "VLSI Architecture of Full-Search Variable-Block-Size Motion Estimation for HEVC Video Encoding", *IET Circuits Devices & Systems*, 2017 *in press.* Reproduced by permission of the Institution of Engineering & Technology.

multiple processing elements. This ME accelerator supports variable-block-size prediction blocks ranging from 8×4 *to* 64×64*, and is reconfigurable in various search ranges for a trade-off between performance and area. The proposed method for very-large-scale integration (VLSI) architecture is synthesized with* 32 nm *technology, and is capable of real-time encoding of ultra-high-definition (4K UHD, at* 30 Hz*) video with a search range of 64 pixels in both horizontal and vertical directions, operating at a frequency of* 282 MHz*.*

3.2 Introduction

The latest video compression standard developed by the joint collaborative team on video coding (JCT-VC) is known as HEVC/H.265 [10]. HEVC/H.265 achieves significantly better coding efficiency than its predecessor, AVC/H.264 [57], by incorporating a larger block size and a variety of partitioning modes. In video encoding, motion estimation (ME) is the most computationally intensive task with approximately 80% of the total computational load [58]. The computational complexity increases manyfold for ME in HEVC/H.265 as compared to the previous standard since the basic encoding block, the CTU, size is increased from 16×16 to 64×64 (16 times as many pixels as the former standard) and by introducing a hierarchical quad-tree structure. On the other hand, the demand for UHD with a resolution of 3840×2160 (4K UHD) introduces another 4 times increase in computations for ME.

Several algorithms exist for AVC/H.264 to reduce the number of search points for ME such as DS [59], new TSS [23] and EPZS [31]. Many of them assume a unimodal error surface with a global minimum, but that model is not true in general for videos, whereas many other fast-search algorithms are not suitable for hardware implementation due to the algorithm complexity, a sequential processing requirement, or the algorithm dependency on thresholds. A hardware-oriented modified diamond search is proposed in [60] to mitigate the above problems. Another search-point reduction algorithm for hardware implementation is proposed in [61]. A low-complexity variable-block-size motion estimation is proposed in [62], utilizing a fast-search algorithm based on a 9-point block concept. As a result of the quad-tree partitioning structure and other complexities, various ME algorithms proposed for AVC/H.264 as in [62–64] are no longer suitable for its successor HEVC/H.265 [65].

Due to the variety of partitioning modes and the quad-tree structure present in HEVC/H.265, the mode decision and CU size selection involve enormous computing time if it tries all available CUs and selects the best. CU decision or mode decision algorithms are proposed in [66–70] to decide the best CU, and some algorithms reduce the computational complexity by up to 50% in experimental cases. An adaptive-search-window algorithm (ASWA) presented in [71] tries to reduce the search window size based on the motion vector and the cost value of previous blocks. ME is suitable for parallelization, which leads to a parallel framework for the CPU + GPU (graphics processing unit) platforms [72–75]. Several algorithm-level optimizations are recommended in [76–78] including suggestions of early termination (ET). However most of these proposals are for software implementations and many of them are not attractive for hardware realization. Moreover software implementation in video generally consume more power and are not suitable for real-time encoding.

In order to meet the high computational requirements for HEVC/H.265 this paper proposes a hardware architecture for full-search ME. Although the fast-search algorithm requires less hardware resources [79], full-search ME always comes up with the best results as it covers all the search points of the fast-search method. The proposed architecture utilizes the Morton order for data reading as well as the SAD [80] reuse strategy for finding motion vectors in the hierarchical quad-tree coding structure of HEVC/H.265. SAD reuse is a crucial technique to achieve real-time processing in hardware, where computed SAD values are stored and hierarchically used for larger blocks.

A straightforward implementation of variable-block-size ME is to use a $P \times Q$ size SAD computation unit similar to [81] or [82], and a search range of $L \times L$ pixels which generates $L \times L$ SADs of $P \times Q$ blocks. For the ME in a video with a frame rate f Hz and a frame size of $W \times H$ pixels, the aforementioned hardware requires a bandwidth of $(W/P \times H/Q) \times (L \times L) \times (P \times Q) \times f$ bytes per second. The architecture introduced in this paper saves bandwidth by broadcasting reference frame pixels to multiple PEs or SAD computation units, for both horizontal and vertical directions. The architecture has an $L \times A$ PE array, which has an input of $L + A - 1$ pixels in a row, and computes a row of L different SADs of an $A \times A$ block in each clock cycle, thus requiring L clock cycles to complete a search window of $L \times L$ pixels. Hence the bandwidth requirement is $(W/A \times H/A) \times (L + A - 1) \times (L + A - 1) \times f$ bytes per second, providing an approximate bandwidth saving of A^2 times over the conventional methods, as $L \gg A$. L and A are taken as 64 and 8 respectively, reducing the bandwidth by a factor of 52.

Since a set of SADs is generated from each rows of PE array, the output pattern results in groups of varying x coordinates, which is not most suited for hierarchical SAD computation [83]. This issue is addressed in [83] by saving all SAD data into a random access memory (RAM), reading the data and reorganizing them. However this demands at least 512 kB RAM to save all 8×8 block's SADs inside a 64×64 CTU and for streaming data continuously. The proposed architecture uses Morton order for data reading, thus it does not need to save all SAD data, instead it is sufficient to save L rows of L SADs in each hierarchical level. To facilitate computation of $M \times M/2$ and $M/2 \times M$ type prediction block (PB) SADs, some additional memory is required in each stage, thus a total of 36 kB RAM is used by the proposed architecture.

3.3 Full-Search Variable-Block-Size Motion Estimation in HEVC

The HEVC/H.265 standard introduces a very flexible and hierarchical coding structure. A picture is divided into CTUs of $L \times L$ where L takes the values of 16, 32 or 64, depending on the encoders, for different memory and computational requirements [10]. All CTUs have the same size, and the CTU replaces a fixed 16×16 macroblock in the previous standard. The support for larger block sizes provides higher coding efficiency, approximately 50% bit-rate savings in HEVC/H.265 compared to its predecessor for an equivalent reproduction quality [57], and is also beneficial for encoding higher-resolution videos such as UHD. Each CTU consists of $L \times L$ samples of a luma CTB and the corresponding chroma CTBs of $L/2 \times L/2$ samples. The luma and chroma CTBs can be split recursively into square coding block (CB) in 32×32, 16×16 down to 8×8 pixels (luma samples) in a quad tree structure.

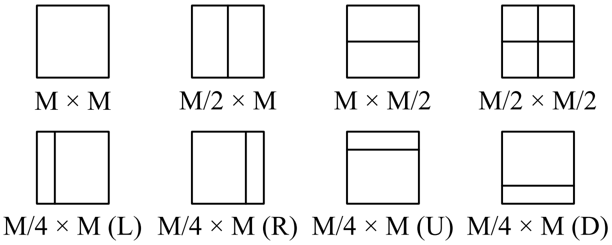


Fig. 3.1: Partitioning of CB into PB in HEVC.

One luma CB and the associated chroma CB form a CU. A CB can be split differently to prediction blocks and transform blocks. The CB is split non-recursively into the PB as shown in Fig. 3.1, where the lower 4 partition types are known as AMP. The prediction mode for a CU may be either intra or inter depending on whether it uses intra-picture prediction or inter-picture prediction respectively. In intra-picture prediction mode, only $M \times M$ and $M/2 \times M/2$ partition types are allowed, whereas in inter-picture prediction mode the AMP cannot be used if M is less than 16 for luma or AMP is disabled. In the inter-picture prediction mode the size 4×4 is not allowed to reduce memory bandwidth. The luma PB with associated chroma PBs forms a PU.

The objective of the ME is to find the best matching block for all kinds of PU types of a current frame from a previous or future frames (reference frames) that gives minimal residues. In an area of picture frame that has many details the smaller block size gives minimal residues. On the other hand for the smooth areas larger block size gives better compression. The ME is usually done in a predefined search window because it is expected that motion is confined to that area for near or consecutive frames, and generally the search window size increases with increasing resolution and decreases with increasing frame rates. The ME should be done for all the PU types (variable block size ME) to get the best coding efficiency.

The SAD is one error measure criterion and is widely used because of its simplicity. The SAD of an $M \times N$ block can be calculated using (3.1)

$$\mathrm{SAD} = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |C(i,j) - R(i,j)| \tag{3.1}$$

where C and R represent current block and reference block pixels, respectively.

3.4 Hardware Architecture of HEVC/H.265 Variable-Block-Size Motion Estimation

The proposed architecture for HEVC/H.265 Variable-Block-Size Motion Estimation uses a full-search method. The schematic of the proposed architecture is given in Fig. 3.2 and consists of three main blocks, namely SAD computation, SAD summation, and SAD comparator. The SAD computation block finds SAD values with the help of PE, the SAD summation block calculates various SAD values of different kinds of blocks in the

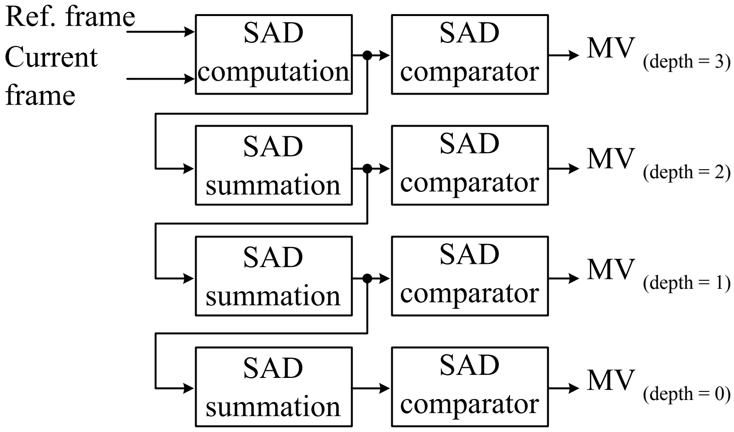


Fig. 3.2: Motion Estimation (ME) hardware architecture. RAMs are used for storing partial and SAD results in each summation stage.

HEVC/H.265 quad-tree structure, and the comparator blocks find the minimum SAD values and thus MVs of these blocks.

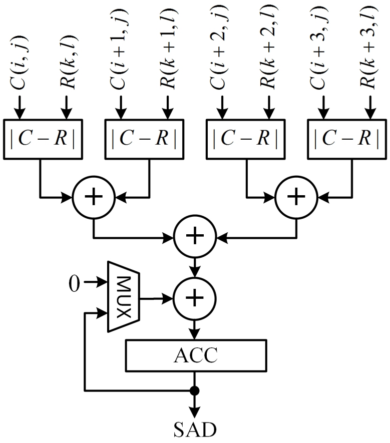


Fig. 3.3: Processing Element (PE) for calculating SAD of 4 pixels.

3.4.1 SAD Computation

The heart of the SAD computation is a PE as given in Fig. 3.3, which calculates the SAD values of 4 pixels of the current frame (CF) and the corresponding reference frame

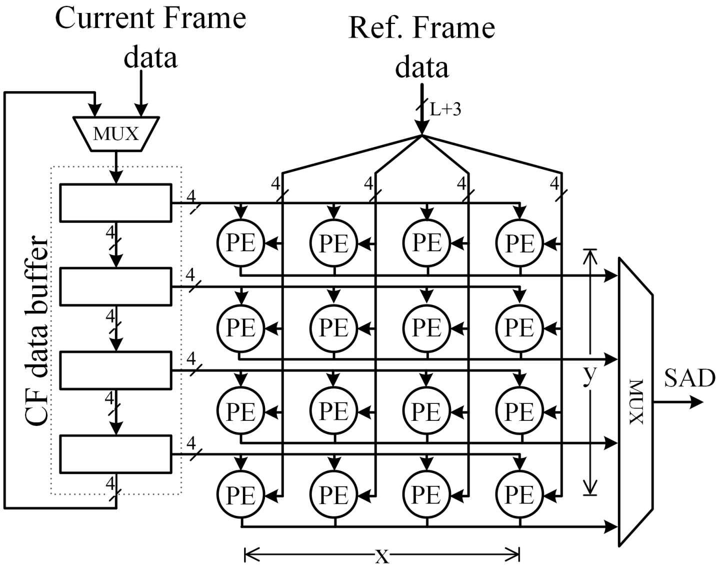


Fig. 3.4: Processing Element (PE) array for Motion Estimation SAD computation. The $L \times 4$ PE array can be reconfigurable in the x direction; in the above $L = 4$.

(RF). Each PE accumulates a 4×4 pixel SAD. The minimum block size according to the HEVC/H.265 specification is 4×4 pixels, so that the PEs are arranged as an array of $L \times 4$ as shown in Fig. 3.4, where L is shown as 4 for simplicity, which is configurable in the x direction. In each cycle a row of reference frame data ($L+3$ pixels) are broadcast into PE elements, whereas the current frame data is stored in local memory along with the PE array (CF data buffer). One important difference of this arrangement from many other structures, including [83,84], is that the proposed structure does not use any multiplexers in the reference data lines, hence halves the number of reference data lines. Although the mux with double RF data lines increases throughput, most of the time one half of the RF-lines are idle, which reduces hardware efficiency.

The detailed data flow of this arrangement is given in Table 3.1, where C and R represent current frame and reference frame pixels respectively. The PEs are arranged as a 2D array of size $L \times 4$, and each PE computes the SAD of a row of 4 current frame

Table 3.1: The detailed data flow in the PE array with $L = 4$.

Cycle		PE($x = 0$)	PE($x = 1$)	PE($x = 2$)	PE($x = 3$)	SAD
0	PE($y = 0$)	$C(0:3,0), R(0:3,0)$	$C(0:3,0), R(1:4,0)$	$C(0:3,0), R(2:5,0)$	$C(0:3,0), R(3:6,0)$	
	PE($y = 1$)	–	–	–	–	
	PE($y = 2$)	–	–	–	–	
	PE($y = 3$)	–	–	–	–	
1	PE($y = 0$)	$C(0:3,1), R(0:3,1)$	$C(0:3,1), R(1:4,1)$	$C(0:3,1), R(2:5,1)$	$C(0:3,1), R(3:6,1)$	
	PE($y = 1$)	$C(0:3,0), R(0:3,1)$	$C(0:3,0), R(1:4,1)$	$C(0:3,0), R(2:5,1)$	$C(0:3,0), R(3:6,1)$	
	PE($y = 2$)	–	–	–	–	
	PE($y = 3$)	–	–	–	–	
2	PE($y = 0$)	$C(0:3,2), R(0:3,2)$	$C(0:3,2), R(1:4,2)$	$C(0:3,2), R(2:5,2)$	$C(0:3,2), R(3:6,2)$	
	PE($y = 1$)	$C(0:3,1), R(0:3,2)$	$C(0:3,1), R(1:4,2)$	$C(0:3,1), R(2:5,2)$	$C(0:3,1), R(3:6,2)$	
	PE($y = 2$)	$C(0:3,0), R(0:3,2)$	$C(0:3,0), R(1:4,2)$	$C(0:3,0), R(2:5,2)$	$C(0:3,0), R(3:6,2)$	
	PE($y = 3$)	–	–	–	–	
3	PE($y = 0$)	$C(0:3,3), R(0:3,3)$	$C(0:3,3), R(1:4,3)$	$C(0:3,3), R(2:5,3)$	$C(0:3,3), R(3:6,3)$	rdy
	PE($y = 1$)	$C(0:3,2), R(0:3,3)$	$C(0:3,2), R(1:4,3)$	$C(0:3,2), R(2:5,3)$	$C(0:3,2), R(3:6,3)$	
	PE($y = 2$)	$C(0:3,1), R(0:3,3)$	$C(0:3,1), R(1:4,3)$	$C(0:3,1), R(2:5,3)$	$C(0:3,1), R(3:6,3)$	
	PE($y = 3$)	$C(0:3,0), R(0:3,3)$	$C(0:3,0), R(1:4,3)$	$C(0:3,0), R(2:5,3)$	$C(0:3,0), R(3:6,3)$	
4	PE($y = 0$)	$C(0:3,0), R(0:3,4)$	$C(0:3,0), R(1:4,4)$	$C(0:3,0), R(2:5,4)$	$C(0:3,0), R(3:6,4)$	
	PE($y = 1$)	$C(0:3,3), R(0:3,4)$	$C(0:3,3), R(1:4,4)$	$C(0:3,3), R(2:5,4)$	$C(0:3,3), R(3:6,4)$	rdy
	PE($y = 2$)	$C(0:3,2), R(0:3,4)$	$C(0:3,2), R(1:4,4)$	$C(0:3,2), R(2:5,4)$	$C(0:3,2), R(3:6,4)$	
	PE($y = 3$)	$C(0:3,1), R(0:3,4)$	$C(0:3,1), R(1:4,4)$	$C(0:3,1), R(2:5,4)$	$C(0:3,1), R(3:6,4)$	
⋮	⋮	⋮	⋮	⋮	⋮	⋮

pixels and reference frame pixels, and accumulate them in the ACC register as shown in Fig. 3.3. In every 4 clock cycles PE starts to accumulate new absolute differences by with the help of MUX selection. Each PE accumulates the SAD of 4 × 4 blocks in different x, y search positions. As an example consider PE at $x = 1$ and $y = 0$. As shown in Table 3.1, in cycle 0 the PE finds the SAD of 4 current pixels of columns 0 to 3 (or $x = 0$ to 3) of row 0, denoted as $C(0 : 3, 0)$, and 4 reference pixels of columns 1 to 4 of row 0, denoted as $R(1 : 4, 0)$. In the next cycle PE adds the SADs of $C(0 : 3, 1)$ and $R(1 : 4, 1)$ to the accumulator. Hence, in cycle 3 this PE has SAD of a 4 × 4 block $C(0 : 3, 0 : 3)$ and $R(1 : 4, 0 : 3)$. Note that the reference data are broadcasted to PEs, and the PEs are finding SADs of different positions; thus in cycle 4 the PEs in row 0 start new accumulation and row 1 has the SADs of x from 0 to $L-1$ and $y=1$.

After an initial latency of 4 clocks, the PE array generates L SAD values of $4 \times L$ pixels in each clock cycle. In other words the PE array requires N clocks to cover a search area of $N \times (L+3)$ pixels. As an example, if the search area is $128 \times L$ pixels, the RF line moves one row in the y direction in each clock, and 132 clock cycles (because of initial latency) are required to search an area of $128 \times (L+3)$ pixels. After completion of scanning in the y direction the RF lines start to broadcast the next set (advancing $L+3$ pixels in the x direction) of reference data, thus RF lines loses its continuity, and the SAD computation has again some latency but still negligible i.e. 4 in 128 clocks (4 out of the search range of the y direction). The CF data circulates in the CF buffer until scanning in the y direction completes, and then loads a new set of data. Since only one row of SAD values are ready in each clock cycle, 4 to 1 muxes are used to select a row and route it to the SAD summation block. The description is given with $4 \times L$ PEs as an example, but the actual hardware uses four sets of $4 \times L$ PEs, where $L=64$, and it can compute 8×4, 4×8 and 8×8 SAD values in parallel.

3.4.2 SAD Summation

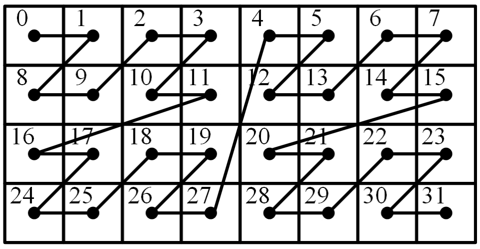


Fig. 3.5: Morton order or Z-order method for loading data into CF data buffer. 32×64 pixels are shown, where each square represents a block of 8×8 pixels.

The architecture is designed to stream data through SAD computation, SAD comparator and the SAD comparator blocks to generate MVs. In order to achieve this, CF data are loaded in a Z order or Morton order [85], i.e. after finding all SADs of an 8 × 8 CF block, the next 8 × 8 load into the CF buffer memory from a 64 × 64 block following Morton order as shown in Fig. 3.5. This allows the design to free the memories associated with lower-level HEVC quad-tree blocks of the SAD computation as soon as those block computations are complete. This reduces the large memory requirements and the associated bandwidths, and only a few kB of RAM is required in each summation stage, and that can be easily integrated inside the chip.

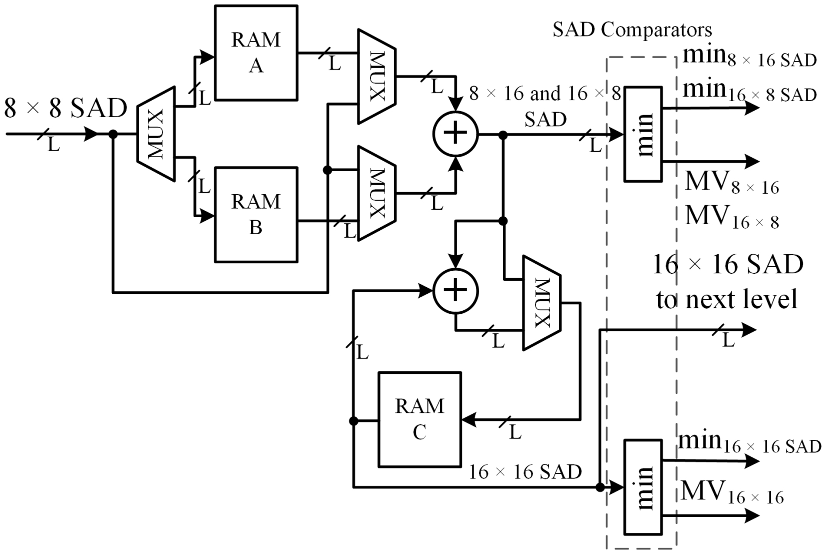

SAD Comparators
8 × 8 SAD
L
MUX
RAM A
RAM B
RAM C
8 × 16 and 16 × 8 SAD
16 × 16 SAD
min
$\text{min}_{8 \times 16\ \text{SAD}}$
$\text{min}_{16 \times 8\ \text{SAD}}$
$\text{MV}_{8 \times 16}$
$\text{MV}_{16 \times 8}$
16 × 16 SAD to next level
$\text{min}_{16 \times 16\ \text{SAD}}$
$\text{MV}_{16 \times 16}$

Fig. 3.6: SAD Summation block with the SAD comparators. This calculates SAD values for higher levels using SAD values from the lower level in HEVC quad-tree SADs, and finds the minimum of these SAD values. The above shows calculation of 8 × 16, 16 × 8 and 16 × 16 block SAD values and MVs from 8 × 8 block SAD values.

The SAD summation block with the SAD comparators are detailed in Fig. 3.6. Three such units are required to compute SAD values for block hierarchies of up to 64 × 64 in size. The figure depicts computation of 8 × 16, 16 × 8 and 16 × 16 block-size SADs from the 8 × 8 SADs. The working of this module is as follows: when the SAD computation block finds SAD values for block ‘0’ in Fig. 3.5, it saves them in RAM ‘A’. Next the SAD computation finds SADs of block ‘1’ and these are saved in RAM ‘B’, and simultaneously finds the first 16×8 SAD values by summing with the previously saved SAD of block ‘0’. Similarly while receiving block ‘8’ (next block in Morton order) SADs, 8×16 block SAD values are calculated and overwrite RAM ‘A’ with block ‘8’ SAD values (block ‘0’ values are no longer required). In this way the SAD summation finds all SAD values of the 8×16 and 16×8 blocks. The 16×16 SAD can be found by summing up either from 8×16 or from 16×8 blocks. RAM ‘C’ and one adder in the SAD summation block are used for this purpose. The first 16×8 or 8×16 block SADs are saved in RAM ‘C’ and then added with the next 8×16 or 16×8 to get SADs of 16×16 blocks. The subsequent blocks can be clocked at a half rate by saving 16×16 SAD results in RAM ‘C’ again and reading at half clock rate. This improves power efficiency as well as critical path timing, because during additions the bit width of the data path increases, thus higher-level modules need more time for computation than lower-level modules.

3.4.3 SAD Comparator

The SAD Comparator consists of a comparator tree of $\log_2(L)$ stages to find the minimum of L SAD values, where each comparator finds the lowest value of two binary inputs and gives it as a result, and also indicates the position of the lowest. The proposed architecture’s critical path lies in the comparator tree used for finding minimum of the L 8×8 SADs obtained from the SAD computation step, where L has been set as 64 and thus the critical path consists of a total of six 14-bit comparators. Note that the 8×8 SAD has 14-bit wide, as the bit-width increases in each addition unless truncating. The comparator trees are also used after each SAD summation block. In each cycle from a row of SAD data, the comparator finds the minimum of these and then compares it with the previous row minimum. Hence when the hardware completes the search in the y direction the comparator has the SAD minimum of that area and the MVs.

3.5 Results and discussion

Table 3.2: Comparison of space time complexities of different integer motion estimation hardwares.

	EL '13 [65]	ICIP '14 H64 P8 [83]	This work
Area			
Adders	8708 [a]	58304 [a]	$LA^2/4 \times 7 = 7552$
Comparators	920 [a]	256 [a]	$8L = 512$
Memory (kB)	20	128 [a]	$3L^2 = 32$
Bandwidth required [b]	B [a]	$B/232$	$B\left(\frac{L+A-1}{L\cdot A}\right)^2 = B/52$
Critical path	Tree adders for 16 numbers [a]	6 comparators or Memory access [a]	$\log_2(L) = 6$ comparators

[a] estimated from the given information

[b] $B = W \cdot H \cdot L^2 \cdot f$

A first order comparison of designs are given in Table. 3.2; the most common proposals for full-search motion estimation are usually a variation of the structure given in [65]. Less number of comparators in the design [83] is obvious as it has fewer kinds of supported PB type. The bandwidth decreases, and the area increases in proportion to A^2, approximately. Most importantly the critical path of the proposed design is six comparators whereas it could be the memory accessing in [83]. The table gives the minimum memory requirement for a highly parallelized version H64 P8 [83], but such structure is big enough for many FPGAs, for instance, it can not be implemented in XC5VLX330T [83], and the less parallelized versions require memory many times of this.

The proposed architecture is coded in VHDL and simulated and verified thoroughly

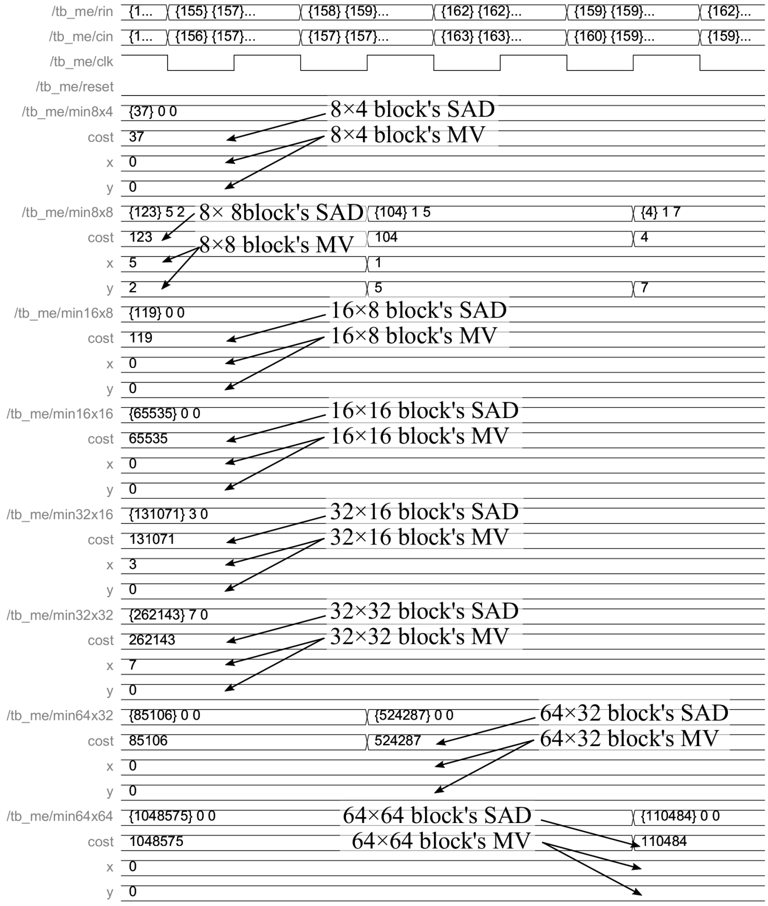


Fig. 3.7: Simulation window of ModelSim simulator, simulated with ICE (4CIF) test video sequence. The snapshot shows motion vectors and SAD values of the first CTU of the video when hardware completes motion search with search range of ±8 pixels.

using ModelSim simulator. A snapshot of the simulation window is given in Fig. 3.7, showing the motion vectors and SAD values of different blocks when hardware completes ME for the first CTU of the test video sequence ICE (4CIF). The proposed architecture is configured for a search height of 64 pixels and a search width of 64 pixels. Although many other configurations are possible, this provides a good trade-off between device area (or resource utilization) and speed. This requires 72 clock cycles for a 8×8 block, and $72 \times 64 = 4608$ clocks are required to complete a search for one 64×64 size CTU, for a search area of 64×64 pixels. Since the data bandwidth is comparatively low and

Table 3.3: Comparison of proposed design with other HEVC/H.265 full-search integer motion estimation implementation in Xilinx Virtex-5 devices.

	ICIP '14 H64 P8 [83]	JRTIP '16 [87]	ICCE '16 [81]	This work
Block size	8×8 to 64×64 (4 kinds)	8×4 to 64×64 (27 kinds)	4×4 to 64×64	8×4 to 64×64 (15 kinds)
Search range	64×64	64×64	64×64	64×64
Max. resolution	4K-UHD (2160p @ 13.4 Hz)	4K-UHD (2160p @ 19 Hz [a])	HD (720p @ 6 Hz [a])	4K-UHD (2160p @ 9 Hz)
Op. frequency	125 MHz	159 MHz	190.785 MHz	84.96 MHz
LUTs	209,434	184,288	17,992 [b]	153,314
Filp-flops	199,066	178,620	8,841 [a]	36,368
BRAM	9.57 MB	36 kB	Nil	Nil

[a] Estimated using operating frequency of the design.
[b] SAD architecture section only.

data fetch is in a predetermined order, it is assumed that data can be fed directly from the external RAM where the picture frames are stored. The proposed hardware does not support AMP, and experimental studies demonstrate that AMP only improves the coding efficiency by 0.8% with the computational increase of 14% [86].

3.5.1 FPGA Synthesis Results

The proposed design is compiled using Xilinx ISE for a XC5VLX330T FPGA. Although the proposed design minimizes the RAM requirement, it needs a few and that is resolved with FPGA slices. The synthesis report shows that the design can operate with a maximum clock frequency of 84.96 MHz. Table 3.3 shows a comparison with [83] implemented in the same FPGA. There are three proposals in [83], namely H64 P2 H64 P4 and H64

P8, and the H64 P8 design has the best performance among them even though it does not fit in this Xilinx Virtex-5 FPGA. Thus our design is compared with the H64 P8 design of [83]. A basic difference between our proposed architecture and [83] is that the latter saves all first SAD computation results in memory then reorganizes using a transposer, and feed to SAD comparators and 4-to-1 adders for calculating other higher-level SADs in the quad-tree structure. Whereas the proposed method utilizes the Morton order for reading data as well as a unique RAM arrangement, and its control structure facilitates data flowing in a streamline fashion through the hardware, resulting in high throughput. From the table it is clear that the proposed design uses fewer resources, with similar frame rates and search area.

Hardware architectures for SAD computations are proposed in [82] and [88], which compute whole 64×64 CTU SADs at single search position within 372.2 ns and 167.7 ns, thus to finish ME in an entire 64×64 search range these demands 1.525 ms 686.9 µs respectively. Hence these can not be used for real-time encoding of 4K-UHD (3840×2160) video as it has 2040 CTUs of size 64×64 pixels. The architecture proposed in this paper broadcasts or shares the same reference data lines in both x and y directions as shown in Fig. 3.4, where $L + 3$ reference data lines are used by $L \times 4$ PEs, efficiently utilizing data loaded to the hardware and hence speeding up overall computations.

Another full-search integer motion estimation architecture is proposed in [87], and implemented in Xilinx Virtex-5 and virtex-7 FPGAs. The comparison is given in Table 3.3, where the frame rate with 4K-UHD is an estimated value from Virtex-5 operating frequency as it requires 4096 cycles neglecting initial delays, for the search range of 64×64 pixels. The design is essentially a 2D array of 64×64 absolute difference units followed by an SAD adder-tree block. The design can compute different block size SAD values in an entire CTU block in a single clock cycle for a search position. A 2D shift register array is used for storing current and reference block data, and the shift registers for

Table 3.4: Comparison of proposed design with previous HEVC/H.265 full-search integer motion estimation in ASIC.

Design	EL '13 [65]	JSSC '14 [89]	This work
Process	65 nm	40 nm	32 nm
Block size	8×4 to 64×64 (27 kinds, support AMP)	4×4 to 16×16 (All AVC)	8×4 to 64×64 (15 kinds, no AMP)
Search range	64×64	±211H ±106V	64×64
Max. definition	2160p at 30 Hz	4320p at 48 Hz	2160p at 30 Hz
Operating frequency	250 MHz	210 MHz	282.009 MHz
Gate Count (k)	3560	2458	2878
Memory (kB)	20	552	Nil [a]
Area	-	15.52 mm^2	7.3 mm^2
Throughput	250 M pixels/s	1.59 G pixels/s	250 M pixels/s
Clocks required / CTU	4105	-	4608
Video standard	H.265/HEVC	H.264/AVC	H.265/HEVC

[a] Included in the gate count.

reference block data need to shift the data in up, down and right directions, resulting in a complex arrangement. Although the design performs better in terms of throughput, it consumes 29% more LUTs and 5 times as many flip-flops than the architecture proposed in this paper. As discussed earlier these hardwares needs approximately 64 times more bandwidth than the proposed architecture.

3.5.2 ASIC Synthesis

The proposed architecture is compiled using the Synopsys Design Compiler (version K-2015.06) in topographical mode, which gives the best correlation with physical design results, using the SAED 32 nm digital standard-cell library. This proposed motion-estimation accelerator requires a clock frequency of 282.009 MHz for encoding 4 K-UHD (3840 × 2160p at 30 Hz) videos in real time. Thus the design is constrained to 330 MHz while compiling in Synopsys. The proposed design met this constraint with operating conditions 1.16 V and worst-case temperature 125 °C. The critical path, lies in comparator tree after the 8 × 8 SAD computation block, has a delay of 2.98 µs. A comparison of various hardware proposals is shown in Table 3.4, where the gate count of the proposed architecture is estimated from the synthesized area compared to the standard 2-input NAND gate.

The result shows that the proposed design, operating with slightly higher frequency, is able to search the same area with the same video resolution as [65]. The design requires 36 kB RAM and is realized using standard logic cells, included with the gate count given for comparison. The proposed design has 17% less gate count and 20 kB less RAM than the previous architecture in the literature for full-search motion estimation for HEVC/H.265. The proposed hardware has significantly less gate count if the gate count for RAM is considered, even though a direct comparison with [89] is not fair because the AVC/H.264 is much simpler than its successor.

3.6 Conclusion

This paper proposes a new architecture for full-search variable-block-size motion estimation in the HEVC/H.265 specification. The proposed architecture minimizes the usage of RAM by incorporating a data streaming and SAD reuse strategy hierarchically. With a similar clock rate, this architecture has 17% less gate count and 20 kB less RAM than previously proposed full-search motion-estimation accelerator architectures for HEVC/H.265. The proposed architecture reduces the bandwidth requirement by a factor of approximately 64 compared to the other full-search ME for HEVC/H.265. The architecture is designed with 32 nm SAED EDK and can encode up to 4-K UHD videos in real time.

Chapter 4

Fast Sign-detection Algorithm for Residue Number System Moduli Set $\{2^n-1, 2^n, 2^{n+1}-1\}$ [1]

4.1 Abstract

Sign detection is an essential part of many computer hardware designs, and is not a trivial task in residue number systems because it is a function of all the residues. This paper proposes an algorithm for the Residue Number System with a three-moduli set $\{2^n-1, 2^n, 2^{n+1}-1\}$ *using New Chinese Remainder Theorem II. The unit is built with one* n*-bit carry-save adder and a* $2n$*-bit parallel prefix carry-generation unit. In the best case the Synoposys* 90 nm *synthesis result shows a 24% reduction in the area-delay product than existing algorithms for the same moduli set.*

---

[1] published as: N. C. Vayalil and Y. Kong, "Fast sign-detection algorithm for residue number system moduli set $\{2^n-1, 2^n, 2^{n+1}-1\}$," *in IET Computers & Digital Techniques*, vol. 10, no. 2, pp. 54-58, 3 2016. doi: 10.1049/iet-cdt.2015.0050, Reproduced by permission of the Institution of Engineering & Technology.

4.2 Introduction

The Residue Number System (RNS) has been intensively researched for decades for designing computer hardware. Traditional binary systems have carry propagation which is one of the major performance limiting factors in such systems. In RNS, large integers are represented as a smaller set of integers or residues of mutually prime moduli sets. Since there is no need to propagate carries between these residues, RNS outperforms the binary systems where addition, subtraction or multiplication dominate. However, the non-positional nature of RNS makes it difficult to do some mathematical operations such as division, comparison, and sign detection, but these are essential in certain applications.

Since sign in RNS is a function of all residue digits, sign detection in RNS is not easy, whereas the sign is usually indicated the by most-significant bit (MSB) in weighted number systems. Traditional approaches for comparison or sign determination involve conversion of RNS into a weighted number using Chinese Remainder Theorems (CRT) or Mixed Radix Conversion (MRC) [45, 90]. However a direct implementation of CRT requires modulo-M operations (where M is the dynamic range of the RNS), thus is not an efficient method. On the other hand MRC is a slow sequential method involving a long delay. [91] proposes a sign-detection algorithm based on New Chinese Remainder Theorem II [92], and presents a simplified case for a special four-moduli set. CRT II decreases the modulo multiplier size to less than $\sqrt{M}$. ROM-based architectures are proposed in [93, 94] but ROMs are not suitable for pipelining. A class of monotone functions are proposed in [95] that can be used for magnitude comparison in RNS. Fully combinational logic sign-detection architectures are proposed in [34, 35, 96, 97] for moduli sets $\{2^n-1, 2^n, 2^n+1\}$, $\{2^n-1, 2^n, 2^n+1, 2^{2n}+1\}$, $\{2^{n+1}-1, 2^n-1, 2^n\}$ and $\{2^n-1, 2^{n+k}, 2^n+1\}$.

In this paper, we propose another fully combinational fast algorithm for the moduli set $\{2^n-1, 2^n, 2^{n+1}-1\}$. The moduli set selection has an influence on the efficiency of the system. This moduli set shows the lowest delay in reverse conversion and is also more efficient in modulo additions and multiplication than other similar moduli sets [46]. Also this moduli set contains only $2^n - 1$ and 2^n type moduli, therefore forward conversion from a binary number to RNS requires a few modulo adders for modulo $2^n - 1$, and for modulo 2^n just needs to keep the least-significant n bits [45].

4.3 Residue Number Systems

RNS and its properties are detailed in many places [45, 98], so only a brief description is provided here. RNS represents large integers as a set of small integers or residues. A natural number X in RNS is represented as an ordered set $(x_1, x_2, \ldots, x_N)$ of a corresponding moduli set $\{m_1, m_2, \ldots, m_N\}$, where $N \geq 2$, and the residues $x_i = X \mod m_i$. In order to avoid redundancy, the moduli set must be selected as mutually prime, i.e., the greatest common divisor (GCD) of any pair must be 1. Then the dynamic range M of this system has been defined as

$$M = \prod_{i=1}^{N} m_i \tag{4.1}$$

The Chinese Remainder Theorem (CRT) [45] relates an integer X and its unique representation $(x_1, x_2, \ldots, x_N)$ in RNS as

$$X = \left| \sum_{i=1}^{N} w_i x_i \right|_M \tag{4.2}$$

where $w_i = m_i |M_i^{-1}|_{m_i}$, $M_i = M/m_i$ and $|M_i^{-1}|_{m_i}$ is the multiplicative inverse of M_i with respect to m_i. The most important property of RNS is that the resultant of a particular digit in some arithmetic operations such as addition, subtraction, and multiplication depends only on the corresponding operand digits:

$$X \odot Y = Z \Leftrightarrow x \odot y = z \mod m_i \tag{4.3}$$

where $\odot$ represents addition, subtraction, multiplication or modular division. Therefore, these operations can be implemented in parallel, without carry propagation between residues. Consequently the execution time is significantly reduced, which results in faster and low-power hardware.

The following addition and multiplication properties [45] of RNS will be used for deducing the proposed algorithm:

$$\left|X \pm Y\right|_m = \left||X|_m \pm |Y|_m\right|_m \tag{4.4}$$

$$\left|X \times Y\right|_m = \left||X|_m \times |Y|_m\right|_m \tag{4.5}$$

Modulo $(2^n - 1)$ addition can be described as

$$\left|X + Y\right|_{2^n-1} = \left|X + Y + w\right|_{2^n} \quad \begin{cases} w = 1 & \text{if } \ X + Y \geq 2^n - 1 \\ w = 0 & \text{otherwise} \end{cases} \tag{4.6}$$

Examples of addition of modulo $2^n - 1$, $m = 7$, $n = 3$

$$\begin{array}{lll} \text{A} & 0\,1\,0 & (2) \\ \text{B} & \underline{0\,1\,1} & (3) \\ & 1\,0\,1 & (5) = 2 + 3 \mod 7 \end{array}$$

$$\begin{array}{llll} \text{A} & 0\,1\,1 & (3) & \\ \text{B} & \underline{1\,0\,0} & (4) & \\ & 1\,1\,1 & (7) & \\ & \underline{+1} & & (w = 1) \\ & 0\,0\,0 & (0) = 3 + 4 \mod 7 & \end{array}$$

Furthermore the additive inverse of X in modulo $(2^n - 1)$ addition is

$$\left| - X\right|_{2^n-1} = 2^n - 1 - X = \overline{X} \tag{4.7}$$

where $\overline{X}$ denotes the complement of X.

4.4 Sign detection

Sign determination in RNS is a major challenge because in RNS the magnitude is not readily available from the residues. Attaching a sign bit, as in the case of sign-and-magnitude representation, does not help in this situation. For example, without knowing the magnitude of operands, it is impossible to assign sign to the resultant after an arithmetic operation such as addition or subtraction between negative and positive numbers. Another problem is to detect overflows in arithmetic operations. It might be expected that to derive one bit of the required information (sign) then all residue information is necessary. Szabo [98] has demonstrated that such schemes are impossible and all the residue information from a residue digit must be used in any sign-determination process, provided that the modulus of the digit is less than $\sqrt{M}$.

4.4.1 Sign determination for special moduli sets

Generally RNS is defined as positive integers in the range $[0, M-1]$. To accommodate negative numbers, this representable number range is usually partitioned into two approximately equal parts. For example, the numbers $[0, M/2-1]$ (or $[0,(M-1)/2]$, if M is odd) are considered as positive numbers and the $[M/2, M-1]$ (or $[(M-1)/2+1, M-1]$) are interpreted as negative numbers.

Consider a mutually prime moduli set $\{m_1, m_2, \dots, m_N\}$, and let any one of the modulo m_k be 2^n so that the dynamic range M is an even number. According to our definitions the numbers greater than or equal to $M/2$ are negative numbers because M is even. In other words the binary number X is negative if $M = m_k M_k > X \geq M/2 = m_k M_k/2$ or equivalently $m_k > X/M_k \geq m_k/2$. Since $m_k = 2^n$, X is negative if $2^n > X/M_k \geq 2^{n-1}$, and therefore the sign of $X = (x_1, x_2, \dots, x_N)$ is determined as

$$\text{Sign} = \text{MSB}\big(\lfloor \frac{X}{M_k} \rfloor\big) \tag{4.8}$$

where $\lfloor \bullet \rfloor$ denotes the floor function. Since $X/M_k < 2^n$, the result of $\lfloor X/M_k \rfloor$ can be representable in n bits, and the most-significant bit (MSB) will be set if $\lfloor X/M_k \rfloor \geq 2^{n-1}$.

Instead of division, this paper proposes an algorithm based on New Chinese Remainder Theorem II (CRT II) [92] to find $\lfloor X/M_k \rfloor$ for the moduli set $\{2^n-1, 2^n, 2^{n+1}-1\}$.

4.4.2 Proposed sign detection algorithm for moduli set $\{2^n - 1, 2^n, 2^{n+1} - 1\}$

A Sign detection architecture for the moduli set $\{2^{n+1} - 1, 2^n - 1, 2^n\}$ is proposed in [35] based on mixed radix CRT. The architecture has CSA, comparator, carry-generation and post processing units. In this paper proposes a simple algorithm based on CRT II which consist CSA and carry-generation units.

The New Chinese Remainder Theorem II pairwise translates smaller modulo residues into higher modulo residue until we get the binary representation of all residues. Consider a moduli set of two numbers (P_1 and P_2); using CRT II the corresponding decimal number $X = (x_1, x_2)$ can be found using the following:

$$X = x_2 + P_2\left|P_2^{-1}(x_1 - x_2)\right|_{P_1} \tag{4.9}$$

where the multiplicative inverse $\left|P_2^{-1}\right|_{P_1}$ is defined by $\left||P_2^{-1}|_{P_1}P_2\right| = 1$. For a general moduli set $\{P_1, P_2, \ldots, P_N\}$, the binary number can be obtained by recursively applying (4.9).

The binary number equivalent $X = (x_1, x_2, x_3)$ with the special moduli set $\{m_1, m_2, m_3\} = \{2^n - 1, 2^n, 2^{n+1} - 1\}$, can be found as follows. In the first iteration of (4.9), let $P_1 = m_1 = 2^n - 1$ and $P_2 = m_3 = 2^{n+1} - 1$; then

$$\left|P_2^{-1}\right|_{P_1} = |(2^{n+1} - 1)^{-1}|_{2^n-1} = 1 \tag{4.10}$$

$$\begin{aligned}\because \left|1 \times (2^{n+1} - 1)\right|_{2^n-1} &= \left||2^n|_{2^n-1} \times 2 - 1\right|_{2^n-1} \\ &= 1 \times 2 - 1 \\ &= 1\end{aligned}$$

Using the above in equation (4.9), an intermediate result Y is obtained as

$$\begin{aligned}Y &= x_3 + m_3|m_3^{-1}(x_1 - x_3)|_{m_1} \\ &= x_3 + m_3|x_1 - x_3|_{m_1}.\end{aligned} \tag{4.11}$$

In the second iteration of (4.9), take $P_1 = m_2$ and $P_2 = m_1m_3$, then

$$\left|P_2^{-1}\right|_{P_1} = |(m_1m_3)^{-1}|_{m_2} = 1 \tag{4.12}$$

$$\begin{aligned}\because \left|1 \times (2^n - 1)(2^{n+1} - 1)\right|_{2^n} &= \big|(|2^n|_{2^n} - 1)\\ &\quad \times (|2^n|_{2^n}2 - 1)\big|_{2^n}\\ &= (0 - 1) \times (0 \times 2 - 1)\\ &= 1\end{aligned}$$

Applying the above in (4.9), the binary equivalent X can be calculated

$$\begin{aligned}X &= Y + m_1m_3\left|(m_1m_3)^{-1}(x_2 - Y)\right|_{m_2}\\ &= Y + m_1m_3\left|x_2 - Y\right|_{m_2}.\end{aligned} \tag{4.13}$$

As a numerical example, consider conversion for the moduli set $\{3, 4, 7\}$, i.e. $n = 2$, with $X = 73 = (1, 1, 3)$

$$\begin{aligned}Y &= 3 + 7\left|1 - 3\right|_3\\ &= 3 + 7 \times 1\\ &= 10\\ X &= Y + 21\left|1 - Y\right|_4\\ &= 10 + 21 \times 3\\ &= 73\end{aligned}$$

Applying (4.13) in (4.8) gives

$$\lfloor\frac{X}{m_1m_3}\rfloor = \lfloor\frac{Y}{m_1m_3} + \left|x_2 - Y\right|_{m_2}\rfloor \tag{4.14}$$

Note that Y is representable by the mutually prime moduli set $\{m_1, m_3\}$, hence $Y \leq m_1m_3 - 1$, therefore $Y/M_2 = Y/(m_1m_3) < 1$. Since the first term in (4.14) is always a fraction with value less than 1 and the second term is an integer, we can safely discard the first term in (4.14), and substituting (4.11) gives

$$\begin{aligned}\lfloor\frac{X}{m_1m_3}\rfloor &= \left|x_2 - Y\right|_{m_2} \\ &= \left|x_2 - x_3 - m_3|x_1 - x_3|_{m_1}\right|_{m_2}\end{aligned} \tag{4.15}$$

Note that $|m_3|_{m_2} = |2^{n+1} - 1|_{2^n} = -1$, and so

$$\lfloor\frac{X}{m_1m_3}\rfloor = \left|x_2 - x_3 + |x_1 - x_3|_{m_1}\right|_{m_2} \tag{4.16}$$

Hence (4.8) can be written as

$$\text{Sign} = \text{MSB}\left(\left|x_2 - x_3 + |x_1 - x_3|_{m_1}\right|_{m_2}\right) \tag{4.17}$$

It may be possible to derive similar equations for other moduli sets, where at least one of the moduli is 2^n.

4.4.3 Optimization for hardware

The equation (4.17) can be further optimized for efficient hardware implementation. Modulo $(2^{n+1} - 1)$ requires $n + 1$ bits and the other two moduli require n bits. Let $x_{3,n}$ be the most-significant bit in x_3, and $x_{3,n-1:0}$ the remaining n bits of x_3. Since $|2^n|_{2^n-1} = 1$,

$$\begin{aligned}\left|x_3\right|_{m_1} &= \left|2^n \times x_{3,n} + x_{3,n-1:0}\right|_{2^n-1} \\ &= \left|x_{3,n} + x_{3,n-1:0}\right|_{2^n-1}\end{aligned} \tag{4.18}$$

$$\left|x_1 - x_3\right|_{m_1} = \left|x_1 - x_{3,n-1:0} - x_{3,n}\right|_{2^n-1} \tag{4.19}$$

Note that $m_2 = 2^n$, so $|x_3|_{m_2} = x_{3,n-1:0}$, and with (4.19) the terms in (4.17) can be re-written as

$$\begin{aligned}&\left|x_2 - x_3 + |x_1 - x_3|_{m_1}\right|_{m_2} \\ &= \left||x_2 - x_3|_{m_2} + |x_1 - x_3|_{m_1}\right|_{m_2} \\ &= \left||x_2 - x_{3,n-1:0}|_{m_2} + |x_1 - x_{3,n-1:0} - x_{3,n}|_{m_1}\right|_{m_2}\end{aligned} \tag{4.20}$$

Modulo (2^n) and modulo (2^n-1) subtractions can be performed as 2's and 1's-complement additions respectively.

$$\begin{aligned} &\left|x_2 - x_3 + |x_1 - x_3|_{m_1}\right|_{m_2} \\ &= \left|x_2 + \bar{x}_{3,n-1:0} + 1 + x_1 + \bar{x}_{3,n-1:0} - x_{3,n} + w\right|_{m_2} \\ &= \left|x_2 + 2\bar{x}_{3,n-1:0} + 1 + x_1 - x_{3,n} + w\right|_{m_2} \end{aligned} \tag{4.21}$$

where w is as given in (4.6) for the modulo ($2^n - 1$) addition of $|x_1 - x_{3,n-1:0} - x_{3,n}|_{m_1}$ and $\bar{x}$ is the complement of x. Note that $1 - x_{3,n} = \bar{x}_{3,n}$, and multiplication of $\bar{x}_{3,n-1:0}$ by 2 is the one-left shifting of $\bar{x}_{3,n-1:0}$. Since the least n bits are sufficient for modulo 2^n arithmetic, adding $2\bar{x}_{3,n-1:0}$ to $\bar{x}_{3,n}$ becomes a concatenation of $\bar{x}_{3,n-2:0}$ with $\bar{x}_{3,n}$ which is denoted as $\hat{x}_3$.

$$\left|x_2 - x_3 + |x_1 - x_3|_{m_1}\right|_{m_2} = \left|x_2 + \hat{x}_3 + x_1 + w\right|_{m_2} \tag{4.22}$$

The w is equal to 1 if $(x_1 - x_{3,n-1:0} - x_{3,n})$ is greater than or equal to $(2^n - 1)$ else 0. Add a 1 always with addition to generate a carry even if the sum is equal to $(2^n - 1)$. Hence

$$\begin{aligned} w &= \lfloor \frac{x_1 + \bar{x}_{3,n-1:0} + 1 - x_{3,n}}{2^n} \rfloor \\ &= \lfloor \frac{x_1 + \bar{x}_{3,n-1:0} + \bar{x}_{3,n}}{2^n} \rfloor \end{aligned} \tag{4.23}$$

which can be found using an n-bit parallel-prefix logic structure [99] for carry generation with carry-in input for $\bar{x}_{3,n}$. Substituting (4.22) in (4.17) gives

$$\text{Sign} = \text{MSB}\left(\left|x_2 + \hat{x}_3 + x_1 + w\right|_{m_2}\right) \tag{4.24}$$

where $\hat{x}_3$ is $\bar{x}_{3,n-2:0}$ concatenated with bit $\bar{x}_{3,n}$, and w is the carry out from $x_1 + \bar{x}_{3,n-1:0} + \bar{x}_{3,n}$.

4.4.4 Hardware

The circuit comprises two main blocks: a multi-operand adder and a carry-generation unit (CGU), as shown in Fig. 5.1. A single n-bit carry-save adder (CSA) is used as a

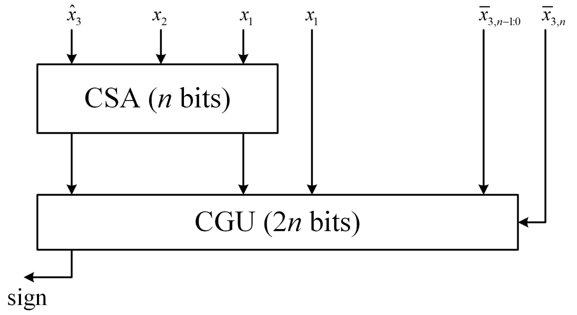


Fig. 4.1: Sign-detection schematic. $\hat{x}_3$ is $\bar{x}_{3,n-2:0}$ concatenated with bit $\bar{x}_{3,n}$, and CSA, CGU are n-bit carry-save adder and $2n$-bit carry-generation units respectively.

multi-operand adder [100] to add $x_2+\hat{x}_3+x_1$ in equation (4.24). The results are obtained as two n-bit vectors S and C (sum and carry). A simplified version of a $2n$-bit parallel-prefix adder, also known as a carry-tree adder, is used for finding w and adding it to the sum of properly weighted S and C vectors. The performance of this sign detector is greatly dependent on the parallel prefix structure used.

Let $A = a_{n-1}a_{n-2}\ldots a_0$ and $B = b_{n-1}b_{n-2}\ldots b_0$ be two numbers to be added and $S = s_{n-1}s_{n-2}\ldots s_0$ their sum. The signals generate bit g_i and propagate bit p_i are defined as

$$g_i = a_i \cdot b_i \quad \text{and} \quad p_i = a_i + b_i \tag{4.25}$$

These are combined to form carry-generate function $G_{i:j}$ and carry-propagate function $P_{i:j}$, from position i to j and defined as

$$\begin{aligned} G_{i:j} &= G_{i:k} + P_{i:k}G_{k-1:j} \\ P_{i:j} &= P_{i:k}P_{k-1:j} \end{aligned} \tag{4.26}$$

where $G_{i:i} = g_i$ and $P_{i:i} = p_i$ The sum can be calculated as $s_i = d_i \oplus c_i$, where the carry $c_i = G_{i:0}$ and $d_i = a_i \oplus b_i$. The above is a radix-2 method; higher radices are also possible, for example ternary tree equations are given below:

$$\begin{aligned} G_{i:j} &= G_{i:k} + P_{i:k}G_{k-1:k'} + P_{i:k}P_{k-1:k'}G_{k'-1:j} \\ P_{i:j} &= P_{i:k}P_{k-1:k'}P_{k'-1:j} \end{aligned} \tag{4.27}$$

Ling further simplifies the parallel prefix equations [101], which helps to speed up additions. Ling defined a pseudo-carry $H_{i:0} = g_i + G_{i-1}$; note that from (4.25) $p_i g_i = g_i$ and so

$$\begin{aligned} G_{i:0} &= g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \dots \\ &\quad + p_i p_{i-1} p_{i-2} \dots p_1 g_0 \\ G_{i:0} &= p_i(g_i + g_{i-1} + p_{i-1} g_{i-2} + \dots \\ &\quad + p_{i-1} p_{i-2} \dots p_1 g_0) \end{aligned} \tag{4.28}$$

The Ling carry $H_{i:0}$ can be computed faster than the corresponding $G_{i:0}$ because it is based on simpler boolean functions. As in the case of parallel prefix, the Ling carry $H_{i:0}$ can also be computed recursively:

$$H_{i:j} = H_{i:k} + P_{i-1:k-1}H_{k-1:j} \tag{4.29}$$

In order to generate the actual carry out ($G_{i:0}$), p_i and $H_{i:0}$ need to be combined. This extra delay can be eliminated because the critical path for the n-bit adder is in producing the s_{n-1} bit in sum S.

$$s_{n-1} = a_{n-1} \oplus b_{n-1} \oplus P_{n-2}H_{n-2:0} \tag{4.30}$$

$$\begin{aligned} s_{n-1} &= \overline{H_{n-2:0}}(a_{n-1} \oplus b_{n-1}) \\ &\quad + H_{n-2:0}((a_{n-1} \oplus b_{n-1}) \oplus p_{n-2}) \end{aligned} \tag{4.31}$$

Since all the terms except $H_{n-2:0}$ in equation (4.30) can be computed faster than $H_{n-2:0}$, a mux can be used to find s_{n-1}, where the mux delay is approximately equal to that of an XOR gate. As in the case of the parallel prefix method, more than two groups can be

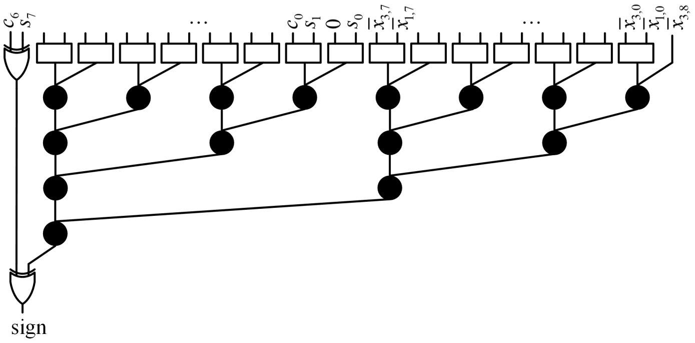


Fig. 4.2: $2n$-bit carry-generation Unit (CGU) based on Kogge-Stone architecture; $n = 8$.

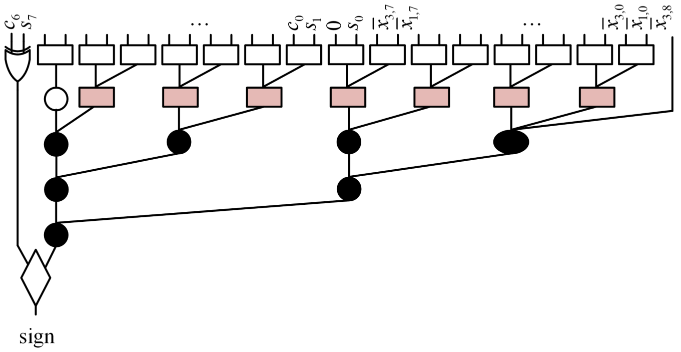


Fig. 4.3: $2n$-bit carry-generation Unit (CGU) based on Ling architecture; $n = 8$.

combined to find Ling carries; for example a ternary tree can be described as

$$\begin{aligned} H_{i:j} = H_{i:k} &+ P_{i-1:k-1}H_{k-1:k'} \\ &+ P_{i-1:k-1}P_{k-2:k'-1}H_{k'-1:j} \end{aligned} \tag{4.32}$$

Parallel-prefix Kogge-Stone [102] and Ling CGUs are shown in Fig. 4.2 and Fig. 4.3 respectively. These are built from blocks given in Fig. 4.4. One black node used in the parallel prefix structure is slightly modified to add a carry-in for inputting $\bar{x}_3$, as in Fig. 4.4.

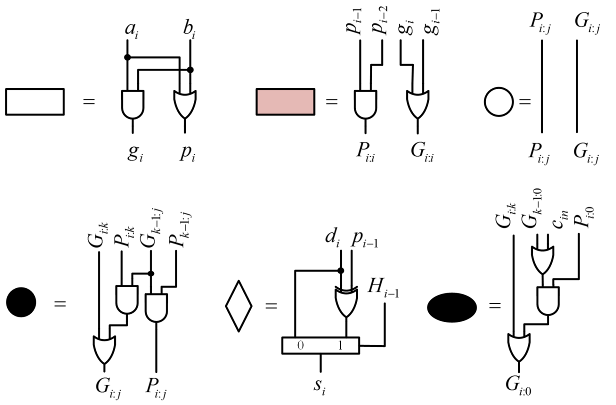


Fig. 4.4: Blocks of the carry-generation unit.

4.5 Performance analysis

The circuit delay and size are first evaluated using a unit-gate model [103]. In this approach, each two-input monotonic gate (e.g., AND, OR, NAND, NOR) counts as one gate (for area and delay) and an XOR gate as two gates (area and delay). The area and delay of inverters are negligible. Hence a full adder has an area of 7, and a delay of 4. An n-bit CSA consists of n such full adders, so the area $= 7n$ (delay $= 4$).

The carry-generation unit based on the the Kogge-Stone structure has $2(n-1)$ black nodes, $2n-1$ square nodes, and two XOR gates. Hence the delay of the carry-generation unit is $2\log_2(2n)+3 = 2\log_2(n)+5$. Therefore the total area and delay of the proposed sign-detection unit in the Kogge-Stone structure are given by

$$\begin{aligned} \text{Area} &= 17n-1 \\ \text{Delay} &= 2\log_2(n)+9 \end{aligned} \tag{4.33}$$

In the Ling structure the carry-generation unit comprises a total of $2n-1$ square nodes, $n-1$ white square nodes, $n-1$ black nodes, one diamond node, one XOR gate. The multiplexer delay is almost equivalent to that of an XOR gate, therefore the total area

Table 4.1: Area and delay comparison for the unit-gate model

n	Area			Delay		
	K-S	Ling	[35]	K-S	Ling	[35]
4	67	63	72	13	10	13
8	135	127	148	15	12	15
16	271	255	300	17	14	17
32	543	511	604	19	16	19

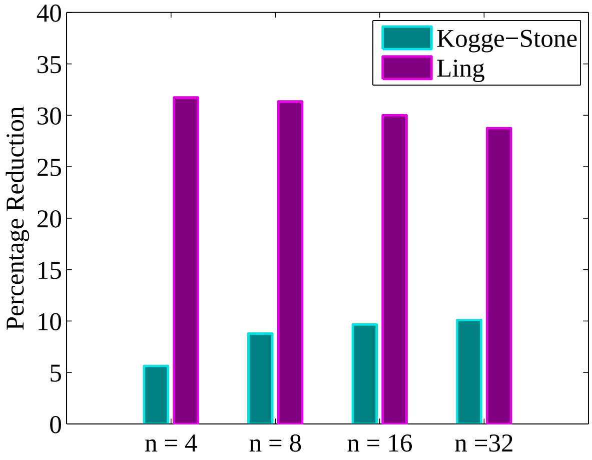


Fig. 4.5: Percentage reduction of area-delay product based on the unit model.

and delay of the sign-detection unit with the Ling CGU are given by

$$\begin{aligned} \text{Area} &= 16n - 1 \\ \text{Delay} &= 2\log_2(n) + 6 \end{aligned} \tag{4.34}$$

Using (4.33) and (4.34) the area and delay were calculated and compared with similar results for the architecture proposed in [35], and given in Table 4.1. The percentage reduction in area-delay product against [35] is plotted in Fig. 4.5. Note that the architecture in [35] uses the Kogge-Stone structure for the carry-generation unit and for the com-

Table 4.2: Area, delay and power comparison of radix-2 architecture experimental results

n	Area (µm^2)			Delay (ns)			Power (µW)		
	K-S	Ling	[35]	K-S	Ling	[35]	K-S	Ling	[35]
4	1154	1426	909	1.0522	1.0871	1.03	92	106	69
8	2384	2354	2176	1.1969	1.2014	1.2120	183	184	160
16	4279	3434	4138	1.4270	1.3735	1.4988	352	294	324
32	7506	6564	6871	1.5687	1.6022	1.6148	594	536	557

parator, hence the comparison of the Kogge-Stone structure provides the improvement of the algorithm itself over [35]. Fig. 4.5 shows an improvement of 32% for the best case using Ling architecture. Since the proposed architecture uses a $2n$-bit carry-generation unit (CGU) instead of an n-bit CGU and an n-bit comparator, incorporating higher-radix cells such as radix-3 or radix-4 may give better results.

The circuits were described in VHDL, then simulated and tested thoroughly for 4 bits, 8 bits, 16 bits and 32 bits. The hardware was then synthesized using Synopsys Design Compiler Ultra with the SAED standard logic cell library (90 nm, 0.7 V, 125 °C) for worst-case analysis. In our experiment, the Synopsys design compiler gives better results if the OR gates are exchanged for XOR gates in (4.25) for the Kogge-Stone circuit. The results are given in Table 4.2. The proposed architecture shows better timing performance in 8-bit, 16-bit and 32-bit designs for the both Kogge-Stone and Ling architecture. For the best case the area-delay product improves 24% than [35] for 16-bit design using Ling architecture. The power-delay product slightly higher for 4-bit and 8-bit designs but less in 16-bit and 32-bit designs .

For further improvement, the proposed designs were implemented with radix-4 architectures for both Kogge-Stone and Ling-based structures. The result shows a slight improvement in speed (1%, 0.5%, and 0.4% for 8 bits, 16 bits and 32 bits respectively)

Table 4.3: Area, delay and power comparison of radix-4 architecture experimental results

n	Area (µm^2)		Delay (ns)		power (µW)	
	K-S	Ling	K-S	Ling	K-S	Ling
4	1481	1425	1.0483	1.087	112	106
8	1991	1989	1.1845	1.3285	159	162
16	3923	3991	1.3670	1.4458	325	338
32	8313	8182	1.6462	1.5953	624	605

by radix-4 implementation, as shown in Table 4.3. The area is less in the case of 8 bits, but it increases slightly for 16 bits and 32 bits. It seems that the Design Compiler was unable to produce better results with an abstract VHDL coding description of the radix-4 structures. It may be possible to get better results using parallel prefix cells specifically designed for higher-radix adders [104]. All the above architectures were synthesized for timing optimization.

4.6 Conclusion

This paper presents a fully combinational fast sign-detection algorithm for one of the popular moduli sets $\{2^n - 1, 2^n, 2^{n+1} - 1\}$ in RNS using New CRT II [92]. This algorithm is then implemented in hardware and the experimental results show improved speed as well as area over existing architectures in the literature. The proposed algorithm improves the speed and area of general RNS hardware implementations because sign detection has a key role in many computing applications, where it is not a trivial task in RNS in contrast with binary systems.

Chapter 5

ASIC Design in Residue Number System for Calculating Minimum Sum of Absolute Differences [1]

5.1 Abstract

Sum of Absolute Difference (SAD) is widely used in motion estimation algorithms which is the most computationally intensive task in video compression, and in determining similarities between two data sets. This paper proposes a SAD hardware implementation using Residue Number System (RNS) which results in superior performance than conventional binary systems. Residue Number Systems (RNS) have been used for decades in designing low-power and high speed computer hardware, because of their inherent parallel structure. In RNS, large integers are represented as sets of smaller integers or residues, where the number base or moduli are mutually prime. Since these residues are independent from

---

[1] published as: N. C. Vayalil, A. Safari, and Y. Kong, "ASIC design in residue number system for calculating minimum sum of absolute differences", *Tenth International Conference on Computer Engineering & Systems (ICCES)*, Dec. 2015, pp. 129-132.

each other, mathematical operations such as addition, subtraction and multiplication can be carried out without any carry propagation between residues, which is in most cases a limiting factor in binary systems. However, some arithmetical operations such as comparison and division are more difficult in RNS than in conventional binary systems, such as determining sign and magnitude comparison of two numbers. The proposed SAD architecture is based on a very recent advancement in fast sign detection algorithm for RNS and the experimental results shows the proposed architecture has higher speed and less area than previous SAD implementation.

5.2 Introduction

Interest in residue number systems (RNS) for making computer hardware is not new. The conventional binary number systems have carry propagation which limits the performance of such systems. Instead, RNS represents large integers as a set of small integers, defined as the residues of mutually prime moduli. Since these residues are independent from each other and there is no carry propagation between these residues in arithmetic operations, the RNS systems outperform traditional binary systems in many computer hardware implementations. However, the non-weighted nature of RNS makes it difficult to do some arithmetic operations such as comparison and division. Recently, some fast architectures were proposed for comparison and sign detection which are essential for computing the minimum sum of absolute differences [34, 35, 105].

Sign detection or magnitude comparison has a key role in finding absolute differences in RNS. Generally residue number systems are defined exclusively for positive numbers in the range of $[0, M-1]$, where $M = \prod m_i$, and m_i are moduli. To accommodate negative numbers, the range of representable numbers are usually partitioned into two approximately equal parts, i.e. the integers X in the range of $[0, M/2-1]$ (or $[0,(M-1)/2]$

if M is odd) are considered as positive numbers and the remaining half are interpreted as negative numbers [45].

Sign detection in RNS has been investigated by many researchers. Some of them used Mixed Radix Conversion [106], and some others Chinese Remainder Theorem [93]. The authors in [107] and [91] proposed new implementations of sign detection based on Chinese Remainder Theorem II. A magnitude comparison with two pairs of conjugate moduli $\{2^n - 1, 2^n + 1, 2^{n+1} - 1, 2^{n+1} + 1\}$ is proposed in [105], and the sum of absolute difference module implemented in FPGA. In [55] a sum of absolute difference module for ASIC in RNS is implemented, and both FPGA and ASIC implementations show superior performance over the binary counter part. Sign detection for the specific moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ and $\{2^{n+1} - 1, 2^n - 1, 2^n\}$ are proposed in [34] and [35] respectively, and showed better performance over previous implementations for sign detection.

This paper proposes a hardware architecture for computing minimum SAD, utilizing one of the fast sign detection algorithm [35] with modifications for further improvement in its hardware implementation and with modified modular adders. Since RNS adders are faster than binary adders the SAD computation which mainly consists of additions becomes simpler and faster in RNS. A SAD architecture using RNS is proposed in [55] but with a moduli set which are not relatively prime. A mutually prime or relatively prime moduli set gives dynamic range $M = m_1 m_2 \dots m_n$ whereas M is determined by least common multiple (LCM) of the moduli if they are not relatively prime, i.e. $M = \mathrm{lcm}(m_1, m_2, \dots, m_n)$. In other words the former moduli set gives better hardware efficiency than the later one, and our experimental results validates it.

5.3 Proposed architecture for calculating minimum of SAD

The Sum of absolute differences (SAD) takes the absolute difference between pixels of the current block and the corresponding pixel in the reference block, followed by aggregation of the absolute differences. Due to its simplicity SAD is widely used in hardware implementations. The SAD between a current and reference block of size $M \times N$ can be calculated using (5.1)

$$\begin{aligned} \mathrm{SAD} &= \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |C(i,j) - R(i,j)| \\ \mathrm{SAD}_{\mathrm{MIN}} &= \min(\mathrm{SAD}(i,j)) \end{aligned} \tag{5.1}$$

where C and R represent current block and reference block pixels, respectively.

Motion estimation (ME) is the most computationally intensive task in video processing. SAD algorithms are widely used in block matching algorithms for motion estimation. In motion estimation, search methods for finding similar blocks can vary from full search to sub-optimal fast search such as the 2D-logarithmic search [22], however full search matching algorithms produce the best results in finding motion vectors. Nevertheless, it is not necessary to search all pixels in the reference image; depending on the characteristics of the movement, the resolution of the image, and the frame rate, it is sufficient to consider a rectangular area centered at the matching block. For a typical application, the block size $M \times N$ varies from 16 to 4096 [10, 108]. Generally, the intensity of the pixels can be represented by 8 bits, so that $\mathrm{SAD} < 2^{24}$. The binary implementation of the SAD requires 24 bits, where the performance is certainly affected by carry propagation. Hence in this paper we propose a RNS-based hardware architecture for computing minimum SAD.

The $\mathrm{SAD}_{\mathrm{MIN}}$ computation can be considered as a three stage process: first find absolute difference of two pixels, then sum all the absolute differences, and finally find the SAD_{MIN}. A comparison or sign-detection module is an essential for the first and third stages, but as mentioned earlier comparison is generally a difficult task in RNS, because of its non-positional nature. This paper utilises *Fast Sign Detection Algorithm for the RNS Moduli Set* $\{2^{n+1}-1, 2^n-1, 2^n\}$ [35] and takes the mutually prime moduli set $\{m_1, m_2, m_3\}$ as $\{511, 255, 256\}$ which can represent binary equivalent values in the range of 0 to $m_1m_2m_3$ (0 to $\sim 2^{25}$), or as signed numbers where the numbers $\geq m_1m_2m_3/2$ is considered as negative numbers.

5.3.1 Sign detection

The sign for the moduli set $\{2^{n+1}-1, 2^n-1, 2^n\}$ can be found using the following equation [35]:

$$\operatorname{sgn}(x_1, x_2, x_3) = \operatorname{MSB}\left(\left|\left\lfloor -2x_1 + x_2 + x_3 + \frac{x_2 - x_1}{2^n - 1}\right\rfloor\right|_{2^n}\right) \tag{5.2}$$

where $\operatorname{sgn}(x_1, x_2, x_3)$ represents the sign of $X = (x_1, x_2, x_3)$. Equation (5.2) can be re-written for hardware implementation as [35]:

$$\operatorname{sgn}(x_1, x_2, x_3) = |\bar{x}_1^{''} + x_2 + x_3 + W|_{2^n} \tag{5.3}$$

where

$$\begin{aligned} W &= 1 + \left\lfloor \frac{x_2 - x_1^{'} - x_{1,n}}{2^n - 1} \right\rfloor \\ &= \begin{cases} 0 \text{ if } x_{1,n} \text{ and } x_2 < x_1^{'} \text{ or } x_{1,n} = 1 \text{ and } x_2 \leq x_1^{'} \\ 1 \text{ if } x_{1,n} \text{ and } x_2 \geq x_1^{'} \text{ or } x_{1,n} = 1 \text{ and } x_2 > x_1^{'} \end{cases} \end{aligned} \tag{5.4}$$

where $x_{1,n}$ is the $n+1^{\text{th}}$ bit of x_1, $x_1^{'}$ is the least significant n bits of x_1, and $x^{''}$ is an n-bit digit that equals $2x_{1,n-2:0} + x_{1,n}$, that is concatenated by least $n-1$ bits of x_1 and $x_{1,n}$.

Fig. 5.1 shows the schematic for a hardware implementation of sign-detection module using the moduli set $\{2^{n+1}-1, 2^n-1, 2^n\}$ [35]. The circuit consists of a carry generation

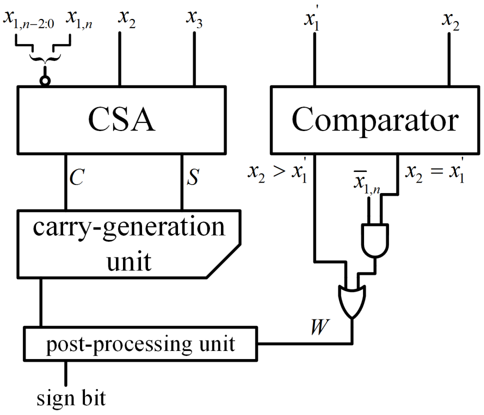


Fig. 5.1: Sign detection module for the moduli set $\{2^{n+1}-1, 2^n-1, 2^n\}$.

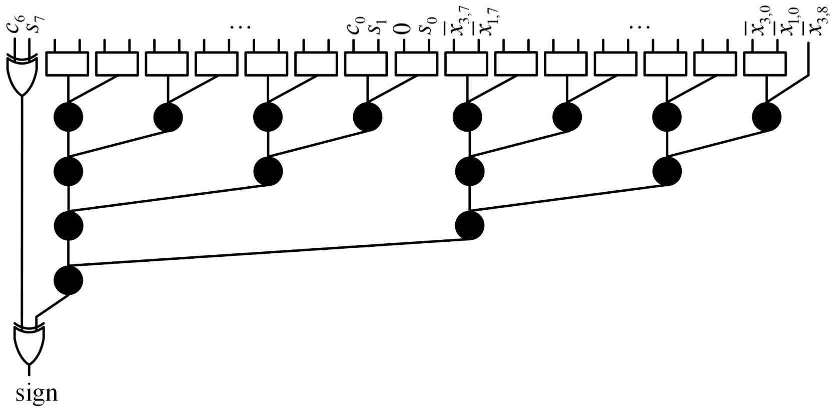


Fig. 5.2: Carry-generation unit and post-processing unit, modified with the blocks in Fig. 5.3.

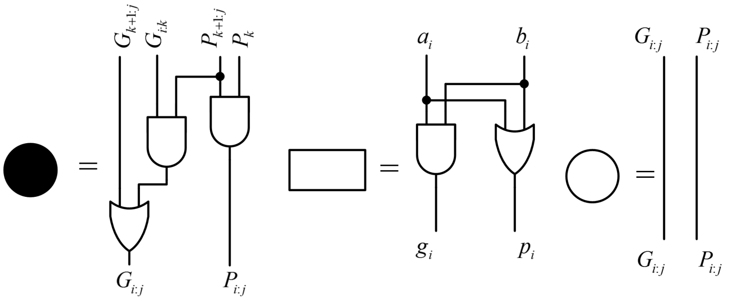


Fig. 5.3: Modified blocks for carry-generation unit.

unit, a comparator unit, and a post-processing unit with a few gates. The carry generation unit and comparator units [34, 35] are built used prefix tree structure [99]. It is possible to replace the XOR gates of carry generation unit in [35] with a more simple circuit of OR gates, shown in Fig. 5.2 and Fig. 5.3. This reduces delay because a OR gate is much simpler than XOR gate, but in order to generate $P_{n-1:n-1}$ an XOR gate is required, but this is not in the critical path.

5.3.2 Modulo-$(2^n - 1)$ adder and subtractor

In the RNS, addition and subtraction can be performed in parallel on the small encoded integers. A modulo-2^n addition/subtraction becomes a common integer addition/subtraction with discarding carry, whereas modulo-$(2^n - 1)$ additions and subtractions are computed as 1's-complement additions and subtractions with end-around carry. Subtraction can be performed by adding a negated subtrahend, and in the case of modulo-$(2^n - 1)$ negating is obtained by simply inverting each bit. In order to speed up the addition the latency of the carry should be minimised. The standard implementation of modulo-$(2^n - 1)$ addition uses a conventional binary adder, where end-around carry is achieved by connecting the carry output to the carry input. This creates a potential race problem between two stable states and may exhibit long delays. In the proposed design, we follow a parallel prefix modulo-$(2^n - 1)$ adder structure [109] modified for single zero representation at the output. The Ling Adder [110] prefix structure is used instead of the Kogge-Stone [102] prefix structure as proposed in [111].

The schematic of a prefix modulo-$(2^n - 1)$ adder using the Ling prefix structure is shown in details in Fig. 5.4, and the prefix blocks are in Fig. 5.5. The parallel prefix modulo-$(2^n - 1)$ adder structures in [111] has two representation for zero, either all '0' or all '1'. The two zero representations are sometimes not desirable. The 'all one' condition

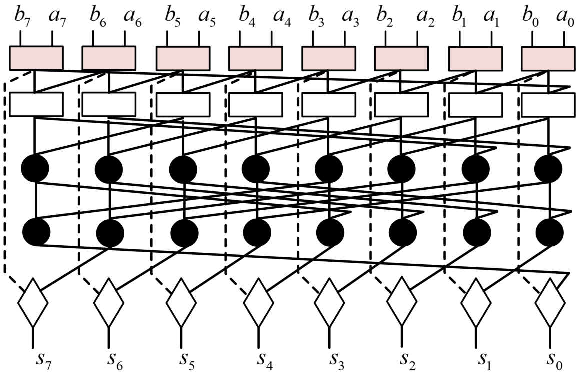


Fig. 5.4: Modulo-255 adder based on Ling structure.

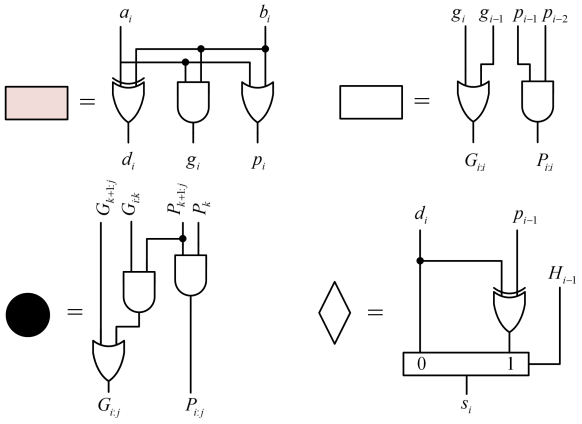


Fig. 5.5: Prefix logic operators for modulo-$(2^n - 1)$ adder.

Table 5.1: Performance comparison of modulo adders, designed with GLOBALFOUNDRIES® 0.18 µm

	Area (μm^2)	Delay (ns)
Modulo-255 with single zero format	675	0.89
Modulo-255 with double zero format	415	0.88
Modulo-511 with single zero format	748	1.01
Modulo-511 with double zero format	675	0.94

at the output can be easily detected by observing $H_{n-1:0}$ and $P_{n-1:0}$ (by '$P_{n-1:0}$' 'AND'ing with an inverted '$H_{n-1:0}$', where H is the Ling carry) of the final layer in the prefix structure. This signal is used to select the output from the prefix structure or from all zeros using a mux.

The comparison of modulo adders with a single-zero representation and with the double-zero representation are given in Table 5.2. It is clear that single-zero representations are slightly slower due to additional logic, hence are used only when required. Carry-save adders (CSA) with end-around carry are used for adder trees in the absolute difference (AD) unit.

5.3.3 Absolute difference unit

Before using RNS logic, the binary numbers should be converted into RNS. Considering the maximum input value ($X = 255$) for the moduli set $\{511, 255, 256\}$ the conversion is not required, because the input is always less than the modulus except for modulo-255. The modulo-255 subtracter considers the input $X = 255$ (all ones) as zero, as desired ($255 \bmod 255 = 0$). Hence, the subtractions can be directly performed on binary data, in all the three channels in parallel. The output is fed to a mux, after negation and without

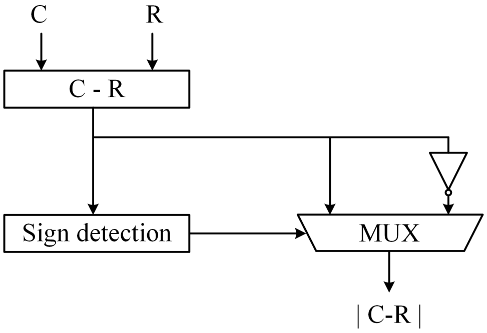


Fig. 5.6: Absolute difference unit.

negation, where the selection is decided after finding the sign of the output using a sign-detection module as shown in Fig. 5.6. The negation is performed as 2's-complement for modulo-2^n, and 1's-complement (inversion of each bit) for modulo-$(2^n - 1)$. The modulo-$(2^n - 1)$) adder with double zero representations is used here, because it does not create problems for the downstream modules. There are sixteen such AD modules in the proposed design.

5.3.4 Accumulation and comparison

The accumulator unit is used for computing the sum of 'absolute difference' values obtained from AD units. This can be implemented using an adder tree of carry-save adders (CSA). We can discard the carry of modulo-2^n additions, whereas end-around carry is achieved by connecting the 'carry output' to the next-stage 'carry in' for modulo-$(2^n - 1)$ adders.

The accumulated value is then compared with the minimum SAD value using a subtracter and a sign-detection unit as shown in Fig. 5.7. The minimum of these values is stored in the SAD_{MIN} registers.

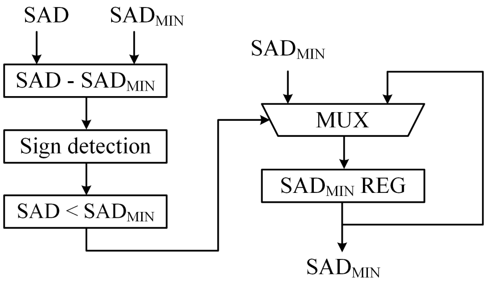


Fig. 5.7: Finding minimum SAD.

Table 5.2: Performance comparison of proposed SAD unit

	Area (mm^2)	Delay (ns)
Binary design with 12 stage pipeline [55]	0.15	4.55 (219 MHz)
RNS design with 12 stage pipeline [55]	0.73	3.00 (333 MHz)
Proposed design with 5 stage pipeline	0.02	2.9 (345 MHz)

5.4 Implementation and results

The proposed design is implemented using VHDL with a five-stage pipeline in the hardware. The functionality is tested thoroughly. The proposed design is compiled with Synopsys tools with the GLOBALFOUNDRIES® csm18os120 standard-cell library (180 nm, 1.62 V, 125 °C) for the worst-case analysis. The results are compared with different implementations in RNS and binary [9] and are given in Table 2. The proposed design area is only 3% of that of [55] and achieved greater speed using 5 pipeline stages instead of 12 pipeline stages. Hence the proposed design has superior performance over the RNS-based design and the binary design of [55].

5.5 Conclusion

This paper proposes a specific structure for computing the minimum sum of absolute difference based on a residue number system using the hardware efficient moduli set $\{2^{n+1}-1, 2^n-1, 2^n\}$. This paper also proposes some modifications in hardware for the *Fast Sign Detection Algorithm for the RNS Moduli Set* $\{2^{n+1}-1, 2^n-1, 2^n\}$ [35] and for the *High-speed Parallel-prefix Modulo*-(2^n-1). The proposed hardware for finding minimum SAD is compiled for Application Specific Integrated Circuit (ASIC) using Synopsys tools. The obtained results show superior performance over previously implemented hardware.

Chapter 6

ASIC design of test zone search motion-estimation hardware for high efficiency video coding (HEVC) with residue number system [1]

6.1 Abstract

Residue Number Systems have been used for designing computer hardware, as carry propagation in conventional binary systems limits their performance. Motion-estimation is the most computationally intensive task in video processing, increasing considerably with every new proposed video coding standard. High efficiency video coding (HEVC), also known as H.265, is the latest video compression standard proposed by the joint collaborative team on

---

[1] N. C. Vayalil, and Y. Kong, "ASIC design of test zone search motion-estimation hardware for high efficiency video coding (HEVC) with residue number system", *IET Circuits Devices & Systems,*, 2017, *in review*

video coding (JCT-VC), and is a successor to the popular advanced video coding (AVC) or H.264. Along with the introduction of new standards, the quest for high quality or high resolution, such as ultra high definition (UHD), and the increased bit depth for consumer video applications, greatly increase computing complexity for motion-estimation. This paper proposes a hardware approach based on a residue number system for implementing a test zone (TZ) search algorithm for motion-estimation in the HEVC/H.265 standard. The implemented results show that the proposed method has 51% less gate count than existing proposals in the literature and considerably less memory requirements than most.

6.2 Introduction

Recent demand for UHD video content in consumer devices is an important driving factor for improving video coding efficiency, which led to the proposal of HEVC/H.265 [10], by the JCT-VC. It is estimated that approximately 80% of the total consumer internet traffic will be video in 2018, a 64% rise from 2014 [112]. The HEVC/H.265 project achieves a significantly better bit-rate than its predecessor AVC/H.264 [9] by roughly 50% [57]. However the improvement comes with a drawback, the computational complexity. The coding efficiency and complexity together increase for every new proposal, as in the case of AVC/H.264 and MPEG-2 [108]. The efficiency of HEVC/H.265 mainly comes from increasing the basic CU block size from 16 × 16 to 64 × 64, introducing a variety of partitioning types, and a recursive quad-tree coding structure. Larger block sizes are important for HD video, especially UHD, because portions of objects are represented by more pixels in HD/UHD video and this could be effectively compressed or encoded by larger block sizes.

The most computationally intensive task in video coding is ME which comprises almost 80% of the total computations for video encoding [84]. In order to cope with this complexity, there have been several suggestions for motion-estimation algorithms, but many algorithms proposed for AVC/H.264 such as [62–64] are no longer suitable for HEVC/H.265 due to its quad-tree coding structure and other complexities [65]. The quad-tree structure allows the CU to be split recursively into four equally sized blocks, which is completely different from the AVC/H.264 motion-estimation procedure. The determination of the best CU size (or depth level in the tree structure), involves much computations, as it may require testing all the possible prediction modes to find the one with the least rate distortion (RD) cost, and algorithms such as [113] are suggested to reduce this CU mode decision complexity.

Motion-estimation algorithms can be broadly classified into full-search and fast-search algorithms. Generally, full-search algorithms are preferred for hardware architectures since they have a regular data-flow structure, preferable for pipelining. Although full-search algorithms produce the best results for finding motion vectors, they usually require enormous hardware resources, as seen in [65], which has 3.56M gates. This paper proposes an aASIC architecture of a TZ search algorithm with the aim to minimize the PSNR degradation. The TZ search is a preferred fast-search algorithm for HEVC/H.265, and is used in the HEVC/H.265 reference software HM [33]. Since the TZ search algorithm has a definite search pattern, pipelining is also feasible. The proposed method is cost-efficient for motion-estimation of 4K UHD videos by employing 4 units with the parallel processing methods supported by the HEVC/H.265 specification, such as tiles or WPP. Encoding with tiles or WPP further helps the decoder to use multithreading, therefore speeding up the decoding process. Furthermore this proposal uses a RNS, which enhances the computational speed.

6.3 Residue Number Systems

RNSs use a set of N residues $\{x_1, x_2, \ldots, x_N\}$ corresponding to a moduli set $\{m_1, m_2, \ldots, m_N\}$ to represent an integer X. A residue r is the remainder of an integer division of X by m, and by definition $X \equiv x \mod m$, denoted by $x = |X|_m$. Let M be the least common multiple, or simply the product of all moduli m_i if they are relatively prime. M is the dynamic range of the system, and any number X which is smaller than M can be uniquely represented by a residue set $\{x_1, x_2, \ldots, x_N\}$ of the moduli $\{m_1, m_2, \ldots, m_N\}$, where $x_i = |X|_{m_i}$.

The Chinese remainder theorem (CRT) [114] relates the integer X and its RNS representation $\{x_1, x_2, \ldots, x_N\}$ by

$$X = \Big|\sum_{i=1}^{N} w_i x_i\Big|_M \tag{6.1}$$

where $w_i = |M_i^{-1}|_{m_i}$, $M_i = M/m_i$, and $|M_i^{-1}|_{m_i}$ is the multiplicative inverse of M_i with respect to m_i. An RNS does not require to propagation of carries between these residues; in other words the mathematical operations such as addition, subtraction and multiplication can accomplished by performing the same operation on this smaller residue set independently, i.e.

$$X \odot Y = Z \Leftrightarrow x_i \odot y_i = z_i \mod m_i \tag{6.2}$$

where $\odot$ represents addition, subtraction, multiplication or modular division.

The following properties of an RNS [45] are also used in designing hardware:

$$\big|X \pm Y\big|_m = \big||X|_m \pm |Y|_m\big|_m \tag{6.3}$$

$$\big|X \times Y\big|_m = \big||X|_m \times |Y|_m\big|_m \tag{6.4}$$

6.4 Hardware design of test zone search motion estimation in RNS for HEVC

HEVC/H.265 introduces a very flexible hierarchical quad-tree coding structure. A picture frame is divided into CTUs, each containing a luma CTB, two chroma CTBs and associated syntax elements [10]. A luma CTB consists of $L \times L$ blocks of luma components, where L takes values of 16, 32 or 64, depending on the memory requirements and computational capabilities of the encoders. The luma and chroma CTBs can be directly used as the CB, or further split into multiple square CBs in a quad-tree structure. The splitting process stops when the luma CB reaches the minimum allowed CB size selected by the encoder, and is always greater than or equal to 8×8 luma samples.

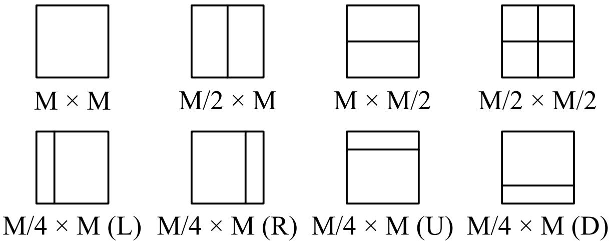


Fig. 6.1: Partitioning of CB into PBs in HEVC.

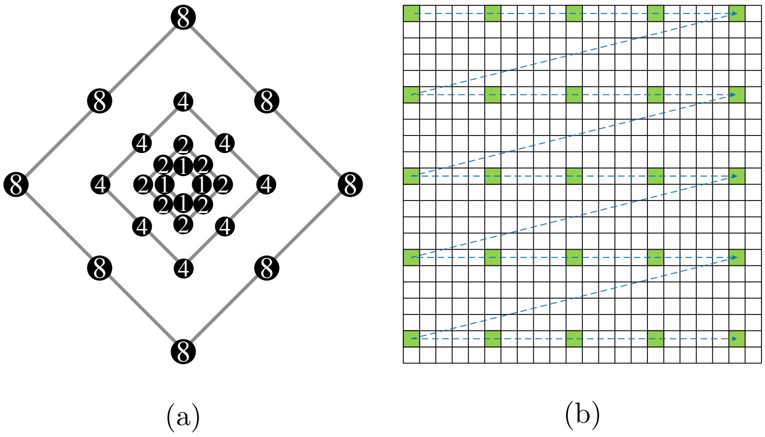
One luma CB and the associated chroma CB form a CU. In the interpicture prediction mode a CB is split into one, two or four prediction blocks as shown in Fig. 6.1. The lower four partitions are referred to as AMP, which can not be used if M is less than 16 for luma. The PB size 4×4 is not allowed in interpicture prediction to reduce memory bandwidth. In intra-picture prediction mode PB may split into four quadrants. The luma PB with associated chroma PBs and syntax elements forms a PU.

The purpose of the motion-estimation is to select the best matching PB for each block in the current encoding frame from previous or future frames. Note that the encoding or decoding order of picture frames is different from the order they arrive from the source or the displaying order. The SAD is commonly used as the matching criterion, because of its simplicity. The SAD between current and reference blocks of size $M \times N$ is defined as

$$\text{SAD} = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |C(i,j) - R(i,j)| \tag{6.5}$$

where C and R represent current and reference block pixels. Due to the said quad-tree structure and the introduction of varieties of PB partitioning types, the motion-estimation becomes more complicated in HEVC/H.265.

6.4.1 Hardware architecture for test zone (TZ) search motion-estimation



(a) (b)

Fig. 6.2: Search pattern for initial grid search and raster search in test zone (TZ) search algorithm: (a) diamond pattern for initial grid search, (b) raster-scan pattern.

The TZ search combines an initial grid search, using either a diamond or square search pattern, and a raster search. The first step, the initial grid search, searches with variable stride lengths of 1 to the 'search range' (64 for example) through multiples of two. If the best distance, i.e. the distance from the initial start point to the global minimum, is greater than 'iRaster' (a predefined value) the algorithm does a raster search, otherwise

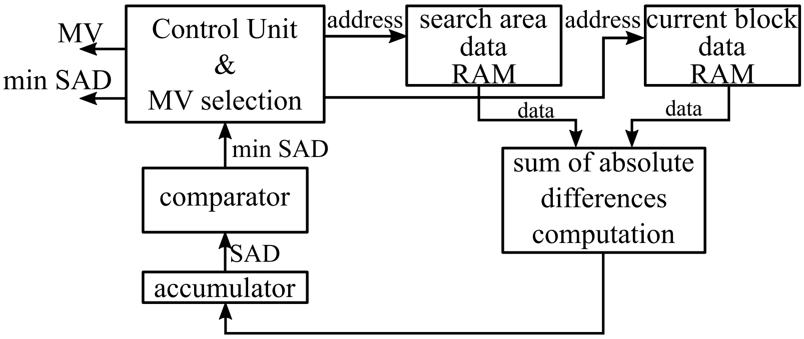


Fig. 6.3: Architecture of test zone (TZ) search motion-estimation.

skips this step. This hardware proposal for a TZ search uses an 8-point diamond grid pattern for the initial grid search, as shown in Fig. 6.2 (a), with stride lengths ranging from 1 to 5. In HM Reference Software, the TZ search uses a stride length of from 1 to the ‘search range’, but it is best to stop the search if the best distance is greater than iRaster (the default value in the HEVC/H.265 reference software for iRaster is 5, used in this hardware configuration), because then it always needs to do the raster search and that covers all these points. This modification of TZ does not affect the PSNR and bit rate appreciably, reported as $-0.044\,\text{dB}$ and 0.554% respectively on average [32]. A raster search is a kind of full-search motion-estimation but advances search points by iRaster pixels, as shown in Fig. 6.2 (b). The algorithm also has an optional refinement step using either a raster or a star pattern for refinement [32].

Fig. 6.3 shows the architecture of the proposed TZ search motion-estimation hardware implementation. The search area and current block data of the reference picture frame and the current picture frame respectively are stored in RAM for accessing quickly. The search starts by finding the minimum SAD of the first PB partition of the largest CU size, i.e. the 64×64 block size. Note that it requires multiple computations by the SAD computation module to find the SAD of a 64×64 block, because in one clock cycle it can only compute a smaller 4×4 or 8×8 block’s SAD, depending on the size of SAD module used. The accumulator accumulates these SADs to get higher block SADs such as

32×64, 64×64. After finding a 64×64 SAD the control logic offsets the RAM addresses to the next position according to the TZ search algorithm and starts accumulating to get the next 64×64 block SAD. These accumulated SADs are compared with the previous SAD and the minimum saved in a register. It also keeps track of the position where the minimum occurs, that is the motion vector of this PB partition.

In a similar way the hardware computes the minimum SADs and motion vectors of other partitions of the CB. The proposed hardware supports symmetric as well as square partitions but not AMP. If the total cost function of square partitions (sum of four $M/2 \times M/2$) is less than that of the other partitioning types, then the control logic decides to split the CB into quad partitions. Otherwise the hardware chooses a motion vector and partitioning type from one of the first three partition kinds of Fig. 6.1 where the minimum cost function belongs to, and exits computation. If the hardware decides to divide the block into square quad partitions, then the above procedure repeats with smaller block sizes till the CU size reaches 16×16. The hardware selects the best partitions for a CU as well as motion vectors based on the cost function; for simplicity the minimum SAD itself is taken as the cost function in this design. The hardware gives the best motion vectors and cost function as its outputs.

The detailed data path for the ME architecture is shown in Fig. 6.4. The absolute-difference (AD) module receives pixels values from the current picture frame block and reference picture frame block from respective RAM and finds the absolute differences. A series of adders accumulates these values, followed by modular adders for converting the accumulated value into the RNS. RNS systems show better performance in mathematical operations such as addition and multiplication, especially when the word length is high. The accumulator has a 24-bit word length for accumulating the SADs of larger blocks, and in such word lengths the carry propagation definitely affects the performance of a traditional binary addition. The binary number system is used in the initial adders in

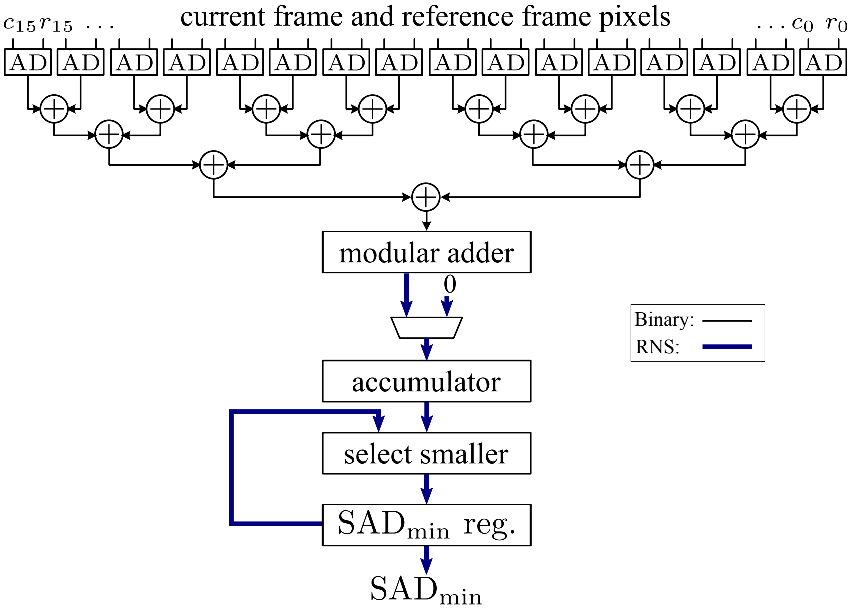

$c_{15} r_{15}$ . . .
current frame and reference frame pixels
. . . c_0 r_0
AD
modular adder
0
accumulator
select smaller
$\mathrm{SAD}_{\mathrm{min}}$ reg.
$\mathrm{SAD}_{\mathrm{min}}$
Binary:
RNS:

Fig. 6.4: Data path of motion-estimation architecture; modular adders are used to convert binary into RNS, thus carry-propagation delays in long-word additions in the accumulator are diminished.

the data path of the ME architecture as the word length is comparatively small, and the word length increases one bit in each adder. Instead if the RNS is used at these stages it may requires a 24-bit equivalent RNS system from the beginning, resulting in an under-utilized hardware system in the early stages of the data path. Although scaling is possible in RNS, it is not trivial as in binary, requiring additional hardware at each stage.

Two different designs are proposed: (a) Using a 4 × 4 SAD in the data path, (b) Using an 8 × 8 SAD in the data path. The architecture (a) accesses 16 pixels from both current picture and reference picture frames whereas architecture (b) accesses 64 pixels from both current and reference picture frames. 4 kB of RAM is required for storing the pixels in the current frame and another 16 kB of RAM for search-area data or pixels in the reference frame, for both cases (a) and (b). Since configuration (b) has 8 × 8 SAD computation hardware in its data path, composed of four 4 × 4 SAD units, it can compute

Table 6.1: Number of 4 × 4 SAD computations involved in various test videos for TZ search inter-picture motion-estimation.

	Resolution / fps / bits	No. of 4 × 4 SADs/sec (max)
Netflix_DrivingPOV[a]	4096 × 2160, 30 fps, 8bit	4,929,299,677
Netflix_Aerial[a]	4096 × 2160, 30 fps, 8bit	4,927,921,301
Netflix_RollerCoaster[a]	4096 × 2160, 30 fps, 8bit	4,861,365,984
Netflix_ToddlerFountain[a]	4096 × 2160, 30 fps, 8bit	4,942,194,456
Netflix_PierSeaside[a]	4096 × 2160, 30 fps, 8bit	4,845,479,084

[a] original video has 10 bit, 60 fps, converted using FFmpeg software [115].

both 4 × 8 and 8 × 4 PB's SADs from the partial 4 × 4 SAD results if M equals 8 in motion-estimation search.

For calculating the timing requirements, various UHD test videos have been encoded in the HM [33]. A few hundred frames of each video are encoded using HM, obtaining the peak value the of number of 4 × 4 SAD computations involved in the encoding process. These results are tabulated in Table 6.1. The maximum among different videos of the number of 4×4 SAD computations required for the motion search, with additional 5% as an allowance, is taken for estimating the number of clock cycles for the hardware. More than this many computations are not expected for determining motion vectors by the TZ search method for encoding 4K UHD videos.

6.4.2 Residue number systems for the motion-estimation hardware architecture

For this application, a three-mutually-prime moduli set $\{2^n - 1, 2^n, 2^{n+1} - 1\}$ is selected because it has the lowest delay in reverse conversion as well as being efficient in modulo additions and multiplications [46], where n is determined as 8 for accommodating the largest possible value. As discussed, the RNS is used in the accumulator, where the carry-propagation delay is dominant because the bit length is so high. Forward conversion, i.e. binary to RNS conversion, is a trivial task in the above moduli set, and can be accomplished employing a few modular adders. The residue with respect to modulus 2^n is easily obtained by dividing the number by 2^n and taking the remainder. Since the division is an n-bit right-shift operation, to get the remainder we just need to keep the least n bits. The residues modulus $2^n - 1$ can be found as follows [45]; note that

$$|2^n|_{2^n-1} = |2^n - 1 + 1|_{2^n-1} = 1 \tag{6.6}$$

and that can be easily extended to 2^{nq}

$$|2^{nq}|_{2^n-1} = \Big|\prod_{i=1}^{q} |2^n|_{2^n-1}\Big|_{2^n-1} = 1 \tag{6.7}$$

Although the dynamic range M has $3n+1$ bits for the RNS with moduli set $\{2^n - 1, 2^n, 2^{n+1} - 1\}$, the binary to RNS conversion happens at a lower bit width, so $3n$ bits are enough to represent the binary input X:

$$X = b_{3n-1}b_{3n-2}\dots b_{2n-1}b_{2n-2}\dots b_{n-1}b_{n-2}\dots b_1 b_0$$

Partition the input binary number X to three n-bit blocks

$$B_1 \triangleq \sum_{j=2n}^{3n-1} b_j 2^{j-2n}$$

$$B_2 \triangleq \sum_{j=n}^{2n-1} b_j 2^{j-n}$$

$$B_3 \triangleq \sum_{j=0}^{n-1} b_j 2^{j}$$

The residue with respect to modulo $2^n - 1$ is obtained as

$$x_1 = \big|X\big|_{2^n-1} = \big|B_1 2^{2n} + B_2 2^n + B_3\big|_{2^n-1} \tag{6.8}$$

Applying (6.3) and (6.7) in the above equation results in

$$x_1 = \left|B_1 + B_2 + B_3\right|_{2^n-1} \tag{6.9}$$

The residue modulo $2^{n+1}-1$ is also obtained in exactly the same way, except that the bit width is $n+1$ for additions. Parallel-prefix 2^n-1 and $2^{n+1}-1$ modulo adders [80,109] are used in the data path of the ME architecture for this forward conversion. A multiplexer is used to select a new start or continuing accumulation to find larger block's SADs. Some of the mathematical operations such as sign detection and comparison are not trivial in RNS, but recent proposals [35,37] helps to mitigate this problem. The fast sign-detection architecture for moduli set $\{2^n-1, 2^n, 2^{n+1}-1\}$ proposed in [37] is used for the comparator implementation, which compares a value with the previous value in the accumulator and saves in the SAD_{min} register.

The synthesized results shows that both hardwares can operate up to a frequency of 375 MHz. From Table 6.1 it is determined that 16 parallel processing units of hardware in configuration (a) or 4 parallel processing units in configuration (b) are capable of encoding 4K UHD videos, as these require 1 clock cycle for each 4×4 block SAD computation and 8×8 block SAD computation respectively. Parallel processing of four units of configuration (b) is selected as it seems more optimal than configuration (a) for encoding real-time UHD videos. The HEVC/H.265 specification has several promising proposals for parallel processing including WPP and tiles; one of them can be adopted to achieve real-time encoding of videos up to 4K UHD resolution. Experimental studies show that WPP and OWF have 0.58% coding losses on average for 4K UHD (2160p) videos whereas tiles suffer 2.17% average coding losses [12].

Table 6.2: Comparison of HEVC/H.265 motion-estimation architectures.

Design	This work	TCASVT'15 [63]	JSTSP'13 [116]	EL'13 [65]
Technology (nm)	32	90	65	65
Clock freq. (MHz)	333	270	200	250
Gate Count (k)	101.6	206 [a]	1067.5	3560
Memory (kB)	80	11.9 [a]	89	20
Video format	4096 × 2160p30	4096 × 2048p60	3840 × 2160p30	3840 × 2160p30
Search range	64	±64	64	64
Block sizes	Except AMP (12 kinds)	Exludes AMP, 32×64, 64×32, 16×16, 32×16 (8 kinds)	16×16, 32×32, 64×64 (3 kinds)	All HEVC (27 kinds)
MV accuracy	1-pixel	1-pixel [a]	1/4-pixel	1-pixel
Frame types	P/B	P/B	P/B	P
Reference frames	4 [b]	1 [b]	1 [b]	1

[a] only integer motion-estimation is considered
[b] in each direction

6.5 Analysis of Results

The proposed design is written in VHDL hardware description language, simulated and verified. The design is synthesized using the SAED 32 nm digital standard cell library for the operating conditions 1.16 V and 125 °C using the Synopsys design compiler version K-2015.06. The obtained results are tabulated in Table 6.2. The proposed hardware uses residue number systems to speed up computations in the SAD accumulator stages. A simplified version of the TZ search motion-estimation hardware architecture is proposed in [63]. The integer motion estimation (IME) section of [63] excluded the raster scan, which is the most time-consuming process in the algorithm, and also several PB types such as 64 × 32, 32 × 64, 32 × 16 and 16 × 32 are omitted from the search. Hence the PSNR is worsened by approximately 5% [116]. Nevertheless the design in [63] requires roughly 51% more gates than the proposed architecture in this paper, considering bi-directional prediction with integer motion-estimation.

On the other hand full-search motion-estimation architecture designs are very straightforward but require more resources as seen in [65, 117]. A simplified version of full search, like the cost and coding efficient (CCE) architecture, is proposed in [116] which supports fewer PB types, 3 kinds instead of 15 kinds (without AMP), which results in a 12% bit-rate increase [116]. Even though the hardware has quarter-pixel resolution for motion-estimation, it has almost 10 times the area (in terms of gate count) and the supported reference frame is only one fourth of the design in this paper. It is clear from the table that the proposed hardware has fewer gates and also in most cases less memory requirement than various proposals in the literature, even though it does not compromise much on the algorithm and hence the PSNR. Although the principles behind motion-estimation in AVC/H.264 and HEVC/H.265 are similar, the computations involved in HEVC/H.265 are much greater, hence a direct comparison on these architectures is meaningless and not included in the table.

6.6 Conclusion

This paper investigate the possibilities of the TZ search algorithm for hardware implementation with residue number systems to speed up the arithmetic operations involved in motion-estimation. The results show that less hardware is used without compromising the bit rate or PSNR compared to other ME search implementations in the literature.

The parallelized version of the proposed method requires a gate count of 101.6 k with 80 kB RAM in total for encoding 4096 × 2048p30 videos in real time. Parallel processing in encoding also helps the decoder to utilize multi-threading, which is favorable for decoding UHD video. The authors expect a good future for the RNS in video processing as the quality, or in other words the bit depth, of video increases; the propagation delays in binary arithmetic become a bottleneck in most of the computations involved in video processing, where RNS can outperform traditional binary systems.

Chapter 7

A Residue Number System Hardware Design of Fast-Search Variable-Motion-Estimation Accelerator for HEVC/H.265 [1]

7.1 Abstract

A residue number system (RNS) has an inherent parallel structure that can be utilized for improving computer hardware systems. An RNS represents large integer numbers as a smaller integer set, or residues of a modulo set, without carry propagation between them. Hence mathematical operations such as addition or subtraction can be performed on residues independently. This paper proposes an RNS implementation of motion estima-

---

[1] N. C. Vayalil, M. Paul, and Y. Kong, "A residue number system hardware design of fast-search variable-motion-estimation accelerator for HEVC/H.265", *IEEE Transactions on Circuits and Systems for Video Technology*, 2017, *in review*

tion for the latest video coding standard known as high-efficiency video coding (HEVC) or H.265. Since motion estimation is the most computationally intensive task in video coding, several simplified algorithms are proposed for mitigating the problem, but the majority of them result in a worsening peak signal-to-noise ratio (PSNR) or bit-rate performance, or sometimes both. This paper also proposes a modified algorithm based on a test-zone (TZ) search algorithm, a widely used fast search algorithm with good rate-distortion (RD) performance, suitable for hardware implementation for encoding ultra-high-definition (UHD) videos in real time. The results show that worst-case PSNR degradation and bit-rate increases compared to the TZ search in HEVC reference software implementation are negligible, and the hardware gate count is less than for many other designs in the literature.

7.2 Introduction

The latest video coding standard introduced by the joint collaborative team on video coding (JCT-VC), known as HEVC/H.265 [10], allows more coding efficiency than its predecessors. The HEVC/H.265 project shows an approximately 50% bit-rate reduction to its predecessor, AVC/H.264 [9], in experimental results for an equivalent subjective reproduction quality, and is also shown to be effective for high-resolution video content [57]. The coding efficiency mainly comes from the introduction of a recursive quad-tree coding structure in its basic CTU as well as the escalation of the block size from 16×16 to 64×64 with additional partitioning modes. Motion estimation is used for eliminating temporal redundancy in the consecutive frames of a video sequence and relates the content of a picture frame, usually a rectangular block in current video coding standards such as AVC/H.264 HEVC/H.265, to the known content of an adjacent frame. Larger block sizes are effective for compressing smoother regions of a picture, and in high-resolution videos such as UHD videos are more likely to have large smoother regions [118]. The improved coding efficiency demands more computation as it needs to investigate larger areas with more complex and varied partitioning types. In a nutshell, more efficient hardware implementation and algorithms are required to cope with this computational complexity and for real-time encoding.

Several algorithms have been proposed for AVC/H.264 to confront the computational complexity [62–64], but they are no longer valid for HEVC/H.265 as it has a recursive quad-tree structure and other complexities [65]. As motion estimation dominates the video encoding process, requiring approximately 60% to 80% of the total computation [58], several algorithms are proposed for reducing the complexity of motion estimation. The CU size and mode decision in HEVC/H.265 take an enormous amount of computation if it tries all available CUs and selects the best. A latent SAD based CU decision approach proposed in [67] reduces the computational complexity to half in experimental cases. Several other CU sizes or mode decision approaches are found in [66,68,70,113]. Adaptive search methods are proposed in [119, 120] for reducing the number of search points. An early termination (ET) process is proposed in [78] to speed up the computation process for motion estimation. Most of the proposals in these papers are for software implementation, and are very hard to implement in hardware, or not efficient for hardware realization. Several of the above algorithms try to increase the computational speed by reducing the number of search points, which causes a performance reduction in the PSNR or bit rate, or sometimes both.

Motion-estimation algorithms are mainly classified as full-search algorithms or fast-search algorithms. As the temporal distance between consecutive frames is small, the motion is usually confined to a small region, and the search for a matching block can be restricted to this region, called a ‘search window.’ The full-search algorithms or exhaustive-search algorithms search for a match in all candidates in this window based on a cost function and come up with the best match. On the other hand, fast-search algorithms skip several candidates which are less probable to find the best match. Although full-search algorithms always come up with better results than fast-search methods, the full search algorithm is an extremely tedious time-consuming process. The TZ search algorithm is a fast-search algorithm implemented in [121] and also in HEVC/H.265 reference software HM [33]. The TZ search algorithm comprises a zonal search, centered around the predicted motion vector, and a raster search if the best location obtained in the zonal search is far from the predicted center. The TZ search shows good RD performance compared to the full-search algorithms and decreased coding time by 60% [32]. Commonly, SAD is used as the cost function for either full-search or fast-search methods, as it is simple and requires less computation than other matching criteria. The SAD takes the absolute difference between the current and reference block pixels and then finds the sum of all these absolute differences. The SAD between $M \times N$ current and reference block pixels is given by

$$\mathrm{SAD} = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |C(i,j) - R(i,j)| \tag{7.1}$$

where C and R represent current block and reference block pixels, respectively.

As HEVC/H.265 motion estimation is highly complex for hardware implementation for real-time encoding, several methods proposed such as [63, 116] simplify the hardware by omitting various search points, which affects their performance. In this paper, we propose a modified TZ algorithm which is suitable for hardware architectures, neither degrading PSNR nor increasing bit rate noticeably. The initial zonal search of the TZ search algorithm needs to get data from various locations of a reference picture frame to find the cost function. Most of these sites are far apart from each other and it necessary to get all $M \times N$ pixels of the reference pixels each time, which increases data bandwidth requirements. The proposed method uses a snake scan order which loads one row or column data and scans through more probable locations, found in our experiments. Another proposal in this paper is to use a RNS to speed up computation SAD, which usually lies in the critical path of the hardware. RNS has been used for hardware architectures for many decades, including video processing [54], but the research related to HEVC/H.265 applications is in its early stages. Carry propagation is the performance-limiting factor in many binary systems, but RNS does not need to propagate carries between the residues used for representing the number, thus shows better performance in several hardware systems than a binary counterpart.

7.3 Residue Number Systems

An RNS represents a large integer number as a smaller set of integers or residues of moduli set $\{m_1, m_2, \ldots, m_N\}$. If q and r are the quotient and remainder of an integer division of x by m, then the number r is said to be the residue of x with respect to m, and then by definition $a \equiv r \mod m$, denoted by $r = |a|_m$. Let M be the least common multiple, or simply the product of all moduli m_i if they are relatively prime. M is the dynamic range of the system, and any number X which is smaller than M can be uniquely represented by a residue set $\{x_1, x_2, \ldots, x_N\}$ of the moduli $\{m_1, m_2, \ldots, m_N\}$, where $x_i = |X|_{m_i}$.

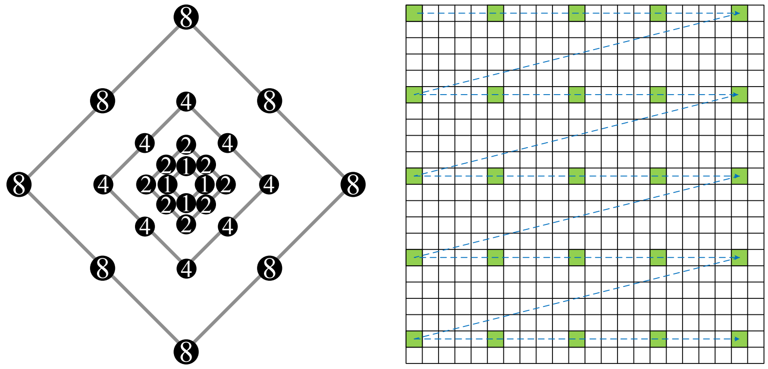
The Chinese remainder theorem (CRT) [122] relates the integer X and its RNS representation $\{x_1, x_2, \ldots, x_N\}$ as

$$X = \Big|\sum_{i=1}^{N} w_i x_i\Big|_M \tag{7.2}$$

where $w_i = M_i |M_i^{-1}|_{m_i}$, $M_i = M/m_i$, and $|M_i^{-1}|_{m_i}$ is the multiplicative inverse of M_i with respect to m_i. An RNS does not require carries to propagate between these residues; in other words mathematical operations such as addition, subtraction, and multiplication can be accomplished by performing the same operation on this smaller residue set, i.e.

$$X \odot Y = Z \Leftrightarrow x_i \odot y_i = z_i \mod m_i \tag{7.3}$$

where $\odot$ represents addition, subtraction, multiplication or modular division.



(a) (b)

Fig. 7.1: Search pattern for initial grid search and raster search in test zone (TZ) search algorithm: (a) diamond pattern for initial grid search, (b) raster scan pattern.

The following properties of an RNS [45] have been used in designing hardware:

$$\left|X \pm Y\right|_m = \left||X|_m \pm |Y|_m\right|_m \tag{7.4}$$

$$\left|X \times Y\right|_m = \left||X|_m \times |Y|_m\right|_m \tag{7.5}$$

7.4 Proposed algorithm for motion estimation

The proposed algorithm is derived from TZ search, which combines a zonal search and a raster search. The TZ search algorithm has the following four steps: motion vector prediction, initial grid search, raster search and raster/star refinement. The first step in the TZ search algorithm is a prediction of the motion vector based on the already obtained motion vectors of nearby blocks, and the predicted location is the starting point for the initial grid search. The initial grid search uses an eight-point pattern, either diamond or square, with a stride length ranging from ‘1’ to the ‘search length’ in multiples of two. The diamond search pattern for the initial grid search is shown in Fig. 7.1 (a). If the distance from the search center to the minimum distortion point is greater than a predefined value

Table 7.1: Percentage of raster searches in a TZ search motion estimation

Video Name	Frame No.	No. of Raster searches (%)
ToddlerFountain [a]	20	14.99
DrivingPOV [a]	29	0.87
TunnelFlag [a]	27	7.92
Crosswalk [a]	22	5.22

[a] 4K UHD video, encoded as P-frames using 4 reference frames

(called ‘iRaster’ in HM software, with a default value of 5 pixels) then the algorithm does the next step, i.e. raster search, otherwise skips the raster search. The raster search is similar to a simple full search or exhaustive search with the search window down-sampled by a factor equal to the ‘iRaster’ value. A raster search pattern with stride length 5 pixels is shown in Fig. 7.1 (b). Raster or star refinement is a fine adjustment of the motion vector obtained in the previous step. Usually one of the methods, either raster refinement or square/diamond pattern refinement (star refinement), is enabled in the algorithm to speed up computation.

The proposed algorithm is specifically designed for high-resolution videos such as 4K UHD or higher. Several 4K UHD videos are encoded in HEVC/H.265 format using HEVC reference software HM [33], with the TZ search motion estimation. Table 7.1 shows the percentage of raster search involved in motion estimation of 4K UHD test videos. All frames except an initial frame are encoded as a P-frame with 4 reference frames using the configuration file ‘encoderlowdelay_P_main.cfg’ provided with the HM. From these results, it is seen that the TZ search algorithm skips raster search in approximately 85% to 99% of the motion-vector searches. In other words, the best motion vectors are found near or within a 5-pixel radius of the predicted motion vector. The initial diamond search, as seen in the experiments, is dominant in the TZ search algorithm. Since most of its

Table 7.2: Percentage of first minimum error locations occurring after the ‘iRaster’ (5 pixels) distance, only considered search involving raster scans

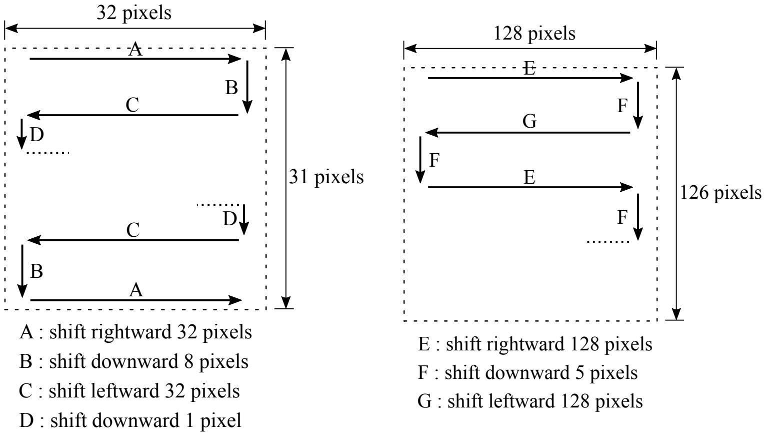
	% of first minima at the distance (pixels)					
Video Name	8	16	32	64	128	256
ToddlerFountain [a]	24.5	44.7	16.2	8.1	4.3	2.2
DrivingPOV [a]	53.5	34.8	7.7	2.7	0.9	0.4
TunnelFlag [a]	22.4	45.1	14.4	8.8	5.8	3.5
Crosswalk [a]	29.4	47.6	11.5	6.2	3.5	1.8

[a] 4K UHD @ 60 fps, frames 1 to 29 encoded as P-frames using 4 reference frames

search locations are far apart, a hardware implementation may need to get a new set of reference pixel data when moving to another search point. Grabbing an entirely new set of reference data requires very high data bandwidth, especially for larger block sizes.

The motion search generally has a non-uni-modal error surface with multiple local minima except for the small region surrounding the global minimum [60]. If the minimum obtained in the initial grid search is far from the predicted motion vector, a raster search is inevitable to avoid this local minima trap. Since the raster search covers all points in the initial grid search, as it is a full search like the search window down-sampled by ‘iRaster’, the initial grid search can be stopped if the minimum is found after the ‘iRaster’ distance. Table 7.2 shows the percentage of first minima occurring after the ‘iRaster’ distance while encoding various videos using HM software. The table only considers searches with the minimum after ‘iRaster’ distances; others are excluded. From the table, it is clear that approximately 70% of minima occur on or before a 16 pixels distance in the videos examined; also note that at maximum only 15% of searches involve raster scan or minima after a distance of 5 pixels.

From these observations, it has been concluded that most of the minima are confined


32 pixels
A
B
C
D
....
....
D
C
B
A
31 pixels
A : shift rightward 32 pixels
B : shift downward 8 pixels
C : shift leftward 32 pixels
D : shift downward 1 pixel
128 pixels
E
F
G
F
E
F
....
126 pixels
E : shift rightward 128 pixels
F : shift downward 5 pixels
G : shift leftward 128 pixels

(a)

(b)

Fig. 7.2: Search pattern for initial search and raster-like scan in the proposed search algorithm: (a) pattern for initial grid search, (b) raster-like search pattern.

in a search window of ±16 pixels. Hence, for the hardware implementation, the search pattern shown in Fig. 7.2 has been proposed. Like the TZ search algorithm, the proposed algorithm has an initial search and raster scan. The first step is an initial search which searches in a search window of 31 × 32 centered (15 rows above and below the center) on the predicted motion vector. The search starts from the left-top corner of the search window and moves like a snake-scan order. The search is equivalent to a full-search motion estimation except for after the first line and before the last line in the search window, where the search moves 8 pixels down at the end of the line, shown as B in Fig. 7.2 (a). Thus the initial search is composed of a full search for the 16 × 32 pixel window and a search in two lines 8 pixels above and below the window, thus including search points for a diamond or square pattern with the 16-pixels stride length of a TZ-search initial grid search.

This unique snake search pattern is used to exploit hardware design features, as the hardware is able to load a single column or row of 64 pixels in a cycle into its shift register. Thus to move a distance of, say, 16 pixels as an example, requires 16 cycles to load 16 new columns or rows of pixels. Thus to complete an initial search the hardware requires 556 clock cycles (total 17 lines, needs 17 × 32 cycles and 2 × 6 cycles in step 'B' in Fig. 7.2), and another 64 cycles to fill the reference block data into the shift registers initially.

As in the TZ search, the next step is a raster scan if the location of the minimum picked up is far from the predicted motion vector. Unlike in HM or in software, the hardware searches a 64×64 block and its sub-blocks in parallel, so making a decision for raster scan is complicated because it is required to consider all PBs. As a simplified approach, the number and size of PBs with minima found in an initial search farther than 'iRaster' pixels from the predicted motion vector in a 64×64 pixel block directs the decision whether to do a raster search or not; the larger PBs' have more weight than smaller PBs in making this decision. The raster scan also follows a snake-scan order where the horizontal movements are 128 pixels and the vertical movements are 5 pixels, and completes a search area of 126 × 128 pixels. The raster scan is the most time-consuming process in this algorithm: the search window has 26 lines and thus requires 26 × 128 clock cycles for all horizontal movements and 25×4 clock cycles for the vertical movements in between, thus 3428 clock cycles in total. Since the vertical movement by 5 rows downward each time ends in the 126th row, the search window height is adjusted to 126 pixels for the above scan pattern. As the initial search pattern covers all search points near the predicted motion vector, like a full search, the final step in the TZ search algorithm, a refinement search, is omitted from the proposed method.

7.5 Motion estimation hardware design

The HEVC/H.265 standard introduces a very flexible hierarchical quad-tree coding structure. A picture frame is divided into CTUs containing a luma CTB, two chroma CTBs and associated syntax elements [10]. A luma CTB consists of $L \times L$ blocks of luma com-

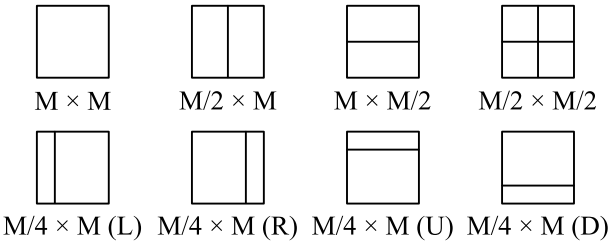


Fig. 7.3: Partitioning of CB into PB in HEVC.

ponents where L takes values of 16, 32 or 64, depending on the encoders, for different memory and computational requirements. The luma and chroma CTBs can be directly used as CBs or further split into multiple square CBs in a quad-tree structure. The splitting process stops when the luma CB reaches the minimum allowed CB size selected by the encoder and is always greater than or equal to 8×8 luma samples.

One luma CB and the associated chroma CB form a CU. In interpicture prediction mode a CB is split into one, two or four prediction blocks as shown in Fig. 7.3. The lower four partitions are referred to as AMP, which can not be used if M is less than 16 for luma. The PB size 4×4 is not allowed in interpicture prediction, to reduce the memory bandwidth. In intra-picture prediction mode PBs may be split into four quadrants. The luma PB with associated chroma PBs forms a PU.

7.5.1 Residue number systems for motion-estimation hardware architecture

An RNS is used with the motion-estimation architecture to enhance the computational speed. In this application a three-mutually-prime modulo set $\{m_1, m_2, m_3\} = \{2^n - 1, 2^n, 2^{n+1} - 1\}$ is selected as it is efficient in modulo additions and multiplications [46] of moduli sets having a dynamic range of $3n$ bits, and n is chosen as 8 to accommodate the largest possible values in the hardware. Forward conversion, i.e. binary to RNS conversion, is not a difficult task for the above modulo set, and it can be achieved with a few modulo adders.

The residues modulo $2^n - 1$ can be found as follows [45]; note that

$$|2^n|_{2^n-1} = |2^n - 1 + 1|_{2^n-1} = 1 \tag{7.6}$$

and that can be easily extended to 2^{nq}

$$|2^{nq}|_{2^n-1} = \Big|\prod_{i=1}^{q} |2^n|_{2^n-1}\Big|_{2^n-1} = 1 \tag{7.7}$$

Although the dynamic range M has $3n+1$ bits for the RNS with modulo set $\{2^n - 1, 2^n, 2^{n+1} - 1\}$, in the proposed design the binary to RNS conversion taken place after 8×8 SAD computation, which has only $(n+6)$-bit width, so $3n$ bits, where $3n > n+6$, are enough to represent the binary input X:

$$X = b_{3n-1}b_{3n-2}\ldots b_{2n-1}b_{2n-2}\ldots b_{n-1}b_{n-2}\ldots b_1b_0$$

Partition the input binary number X to three n-bit blocks:

$$B_{1,n} \triangleq \sum_{j=2n}^{3n-1} b_j 2^{j-2n}$$

$$B_{2,n} \triangleq \sum_{j=n}^{2n-1} b_j 2^{j-n}$$

$$B_{3,n} \triangleq \sum_{j=0}^{n-1} b_j 2^{j}$$

The residue with respect to modulo $2^n - 1$ is obtained as [45]:

$$\begin{aligned} x_1 &= \big|X\big|_{2^n-1} \\ &= \big|B_{1,n} \cdot 2^{2n} + B_{2,n} \cdot 2^n + B_{3,n}\big|_{2^n-1} \end{aligned} \tag{7.8}$$

Applying (7.4) and (7.7) in the above equation results in

$$\begin{aligned} x_1 &= \big|B_{1,n} \cdot 1 + B_{2,n} \cdot 1 + B_{3,n}\big|_{2^n-1} \\ &= \big|B_{1,n} + B_{2,n} + B_{3,n}\big|_{2^n-1} \end{aligned} \tag{7.9}$$

The residue modulo $2^{n+1}-1$ is obtained in exactly the same way, except that the bit width is $n+1$ for additions.

$$x_3 = \left|B_{1,n+1} + B_{2,n+1}, B_{3,n+1}\right|_{2^{n+1}-1} \tag{7.10}$$

The residue x_2, of modulo 2^n, is given as

$$x_2 = B_{3,n} \tag{7.11}$$

7.5.2 Motion-estimation architecture

The proposed design incorporated the following features. Firstly, the hardware processing element's size is selected to be the most likely block size found in UHD videos. Thus it could more efficiently utilize the data bandwidth than architectures which start the search with a lower block size and eventually come up with the higher block search results. Furthermore, the proposed hardware employs the SAD reuse technique and searches lower block sizes with the largest possible size allowed by the hardware. Secondly, our experimental results show that in most cases the search results are found near to the predicted motion vector, and the proposed hardware mostly searches in this area. Many proposals suggest a full-search method as it is more suitable for a hardware structure and easy to pipeline, but these result in wasting time by searching in the least-probable locations. However, to avoid local minima traps a raster-like scan is included in the proposed algorithm. Finally, an RNS is integrated into the hardware SAD data path to improve computational speed. The hardware size is selected to process whole 64 × 64 CBs in one cycle because, in 4K UHD or higher-resolution videos, it is expected that a significant number of motion searches will result in CBs of this size. This can be seen from the percentage of 64 × 64 CBs in 4K UHD videos in Table 7.3, where frames 2 to 30 of the videos are encoded as P-frames using the configuration file 'encoderlowdelay_P_main.cfg'. Even though the video 'TunnelFlag' has very high motion in its frames, approximately

Table 7.3: Percentage of 64 × 64 inter-prediction coding blocks (CBs) in 4K UHD videos

Video Name	Percentage of 64 × 64 size CBs
ToddlerFountain	25.8
DrivingPOV	79.8
TunnelFlag	47.6
Crosswalk	81.5

47% of CBs are in the 64×64 block size, and 'ToddlerFountain' has several small moving objects (water droplets for example) thus comparatively low 64 × 64-block count; others have approximately 80% of CBs in the largest block size.

The motion-estimation architecture is shown in Fig. 7.4; in essence, it is a 2-D array of PEs and a variable-block-size (VBS) SAD tree. The basic architecture is similar to a scaled version of the proposal for AVC/H.264 in [84], with necessary changes for incorporating a TZ-search-like motion estimation. The architecture has two arrays of data registers, which hold current-block and reference-block data, and are connected to a 16 × 16 PE array. Each PE computes an SAD of 4 × 4 pixels of current and reference blocks. A 2-D array of 64 adder blocks calculates 8 × 8 SADs from the 4 × 4 SADs, and the VBS adder tree find SADs of higher blocks, VBS also finds minimum SADs as well as motion vectors. A column of 64 pixels of data can be loaded into the shift registers of the current-block or the reference-block data registers at each clock. The reference data register array is capable of updating data in either left column or right column and shifting to the right or left direction respectively for left or right directional horizontal movement. The reference-block data needs to update the bottom row, shifting data in the registers in the up direction for a downward movement.

The VBS consists of a series of adders for reusing the SADs of smaller PBs to get the SADs of larger PBs. The word length required to represent the binary number inside the

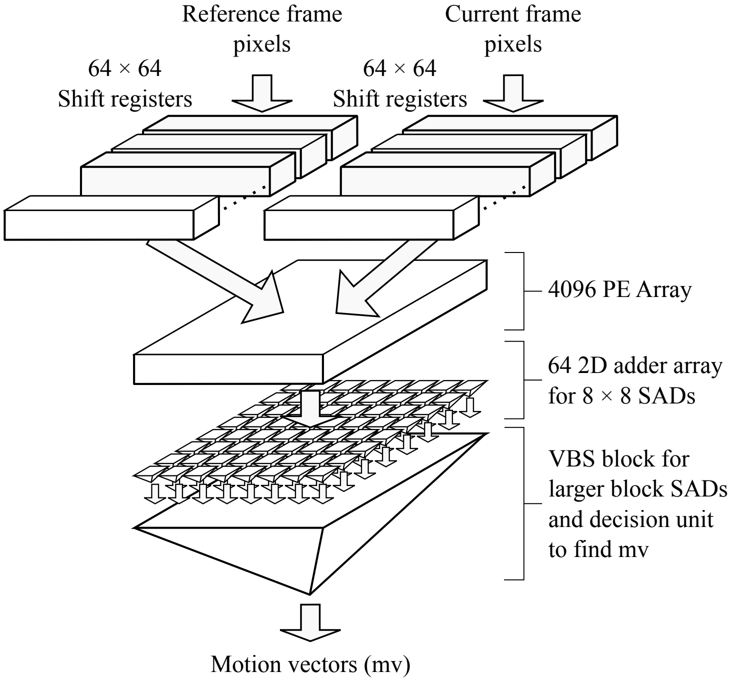


Fig. 7.4: Architecture of motion estimation hardware.

hardware increases after each addition, thus the carry propagation delay increases with each adder of the hierarchy. One of the important advantages of an RNS is that it does not need to propagate a carry between each modulo, and thus outperforms traditional binary systems where the propagation delay is a performance limiting factor of such mathematical operations. On the other hand scaling or increasing the dynamic range in an RNS is not trivial, so its bit length has to be selected to accommodate a possible maximum value of the calculation from the very beginning. Thus the design follows a hybrid approach, using binary for lower-bit-length computations and the RNS for higher-bit-length computations. The hardware uses a binary number system until the 8×8 SAD modules and an RNS after that, and a conversion from binary to RNS is required after the 8×8 SAD blocks, which is easily done with a few modular adders.

A detailed structure of a VBS module is shown in Fig. 7.5, where the RNS data path and the hardware modules are represented in different colors. A 4-to-1 adder is used to sum partial SADs to find larger PB's SADs in the quad-tree structure of HEVC/H.265. In RNS, a modulo-2^n addition uses a simple binary adder and discards the final carry generated, and a modulo $2^n - 1$ addition is given as:

$$(a+b) \mod (2^n - 1) = \begin{cases} (a+b) \mod 2^n, & \text{if } a+b < 2^n \\ (a+b) \mod 2^n + 1, & \text{if } a+b \geq 2^n \end{cases} \tag{7.12}$$

If two representations, either all '0's or all '1's, are allowed for zero in modulo $2^n - 1$ residues in hardware, equation (7.12) becomes an end-around-carry addition as

$$(a+b) \mod (2^n - 1) = (a+b) \mod 2^n + c_{n-1} \tag{7.13}$$

where c_{n-1} is the carry from the $(n-1)^{\text{th}}$ bit. Multi-operand carry-save-adders (CSAs) with end-around carry are used as modulo adders in both binary to RNS conversion and 4-to-1 adders. Parallel-prefix hardware realizations of equation (7.13) are provided in [80, 111], and are employed between CSA and minima comparator, to combine the carry and sum results of the CSA before feeding them to the comparator module. The comparator uses a subtracter, which adds negated subtrahend to minuend, followed by a sign detection [37] to find the minimum of SAD by comparing it with the previous minimum of SAD. Note that negation of modulo $2^n - 1$ residues are the one's complement (inversion of each bit), and 2^n residues are two's complement. MVs are the 'x' and 'y' (horizontal and vertical positions) locations of the minima that can be provided as binary numbers from the minima comparators. A detailed discussion of the advantages of an RNS over the traditional binary systems in SAD architectures is given in [55], based on a sign-detection algorithm provided in [105], with a different moduli set; the moduli set used in this proposal was found to be more attractive [80].

The performance of the proposed method is tested using a software model; the proposed method was integrated into the HM and encoded various test videos. The bit-rate increase and the PSNR degradation are compared with results obtained from the TZ

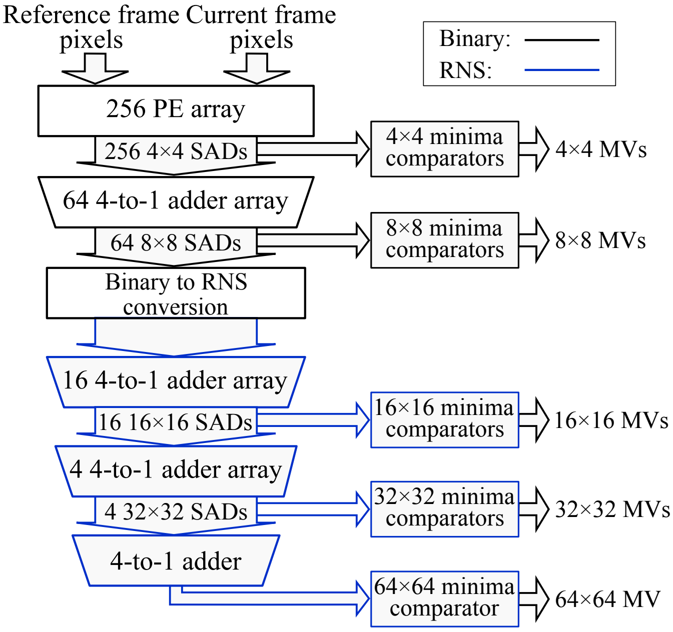


Fig. 7.5: Detailed structure of VBS block, where RNS data path and hardware sections are shown in different color. Motion vectors (MVs) are provided in binary from both binary and RNS minima comparators.

search method, using the same search range and QP (±64, and 32 respectively), as tabulated in Table 7.4. It is evident from the table that the worst-case PSNR degradation as well as bit-rate increase are negligible, 0.0026 dB and 0.9438% respectively in the case of the UHD videos tested, also showing similar results for FHD (full high-definition) and CIF (common intermediate format or common interchange format) videos. Moreover, it is evident from the table that in several cases the proposed algorithm for hardware improves performance in both the PSNR and bit rate, because the algorithm includes more search points, especially if the motion is near the predicted center.

The highest number of raster searches obtained in the experiments shown in Table 7.1 is approximately 15% of total searches, and this can be used for estimating the number of clock cycles required for the proposed hardware. Since the initial grid search and raster search require 556 and 3428 clock cycles respectively, on average 986.8 clock cycles are

Table 7.4: Bit-rate and PSNR changes of proposed algorithm compared to the TZ search.

Video Name	Bit-rate Increase (%)	PSNR reduction (dB)
ToddlerFountain	0.0487	0.0021
DrivingPOV	0.1386	0.0026
TunnelFlag	0.9438	−0.0376
Crosswalk	0.0618	0.0005
rush_hour	0.2968	−0.0038
Kimono1	0.1806	−0.0009
ParkScene	0.2741	−0.005
paris	−0.5271	−0.006
waterfall	−0.0652	−0.0144
tempete	−0.3017	0.0061

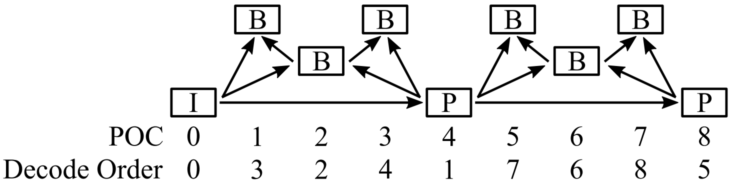


Fig. 7.6: A hierarchal GOP coding structure with size 4. This structure is used for determining the number of clock cycles for encoding videos as listed in Table 7.5.

Table 7.5: Bit rate and PSNR and total clock cycles to encode 64 UHD video frames using the GOP structure in Fig. 7.6.

Video Name	Bit rate		PSNR (dB)		Clock Cycles
	'P' Slices	'B' Slices	'P' Slices	'B' Slices	
ToddlerFountain	115398	46293	37.1945	33.8613	449444280
DrivingPOV	19578	398	38.5168	38.2155	191880232
TunnelFlag	41396	4368	39.6618	38.7374	402928657
Crosswalk	11549	1375	42.4185	41.8984	389560891

required for motion estimation of a 64 × 64 block, and an additional 64 clock cycles for loading the current frame data into the shift registers. Thus motion estimation with 2 reference frames for 4K UHD @ 60 fps (frames per second) requires $1050.8 \times 2176 \times 60 \times 2$, or approximately 275M clock cycles per second for real-time encoding.

For a more realistic estimation, the software model of the proposed hardware architecture can be used for determining the maximum number of clock cycles required for a motion search with a search range ±64 pixels in both horizontal and vertical directions. Videos are encoded in GOP size 4 as shown in Fig. 7.6, where all 'P' frames have 1 reference pictures and 'B' frames have 2. The POC is the display order of the frames within a GOP. A total of 65 frames of different UHD videos are encoded, and the clock cycles for encoding 64 frames (excepting the first 'I' frame) are listed in Table 7.5. The experiment shows that videos having a high motion content, such as 'ToddlerFountain' or 'TunnelFlag', require a clock frequency of 450 MHz to encode at the rate of 64 frames per second in real time.

The Bjøntegaard delta (BD) bit rate and RD curves [123] are shown in Table 7.6 and Fig. 7.7 respectively, for the hardware-search and TZ-search algorithms, with the UHD video sequences Crosswalk, DrivingPOV, TunnelFlag, and ToddlerFountain, where

Table 7.6: BD rate and BD PSNR comparison with TZ search.

Video Name	BD-PSNR (dB)	BD-rate (%)
ToddlerFountain	−0.0034	−0.08
DrivingPOV	−0.0009	−0.05
TunnelFlag	0.0201	0.82
Crosswalk	0.0022	0.09

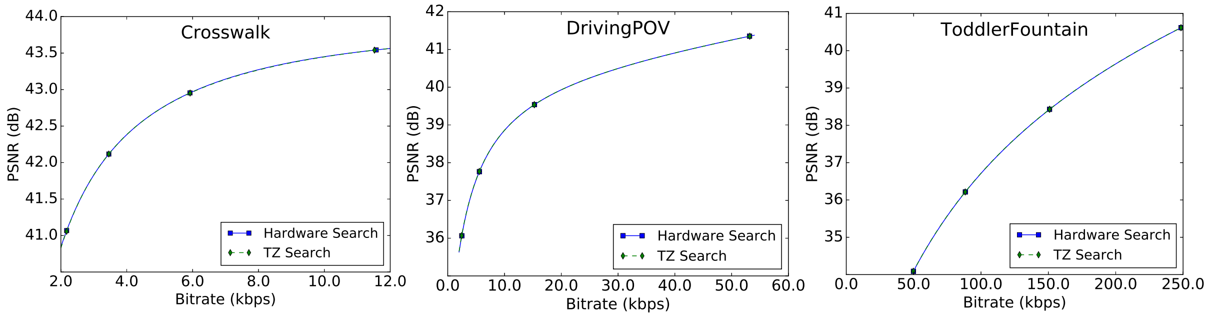


Fig. 7.7: RD curves for (a) Crosswalk (b) DrivingPOV (c) ToddlerFountain sequences with QPs 24, 28, 32, 36.

29 frames are encoded as 'P' frames with QPs 24, 28, 32 and 36. There is virtually no difference between these two and in some cases the performance improves, thus the proposed modifications have a negligible degradation of both bit rate and PSNR for integer motion estimation by the modification of the algorithm to make it suitable for hardware.

7.6 Analysis of results

The proposed hardware architecture for integer motion estimation can support 4K UHD videos with 2 reference frames ('B' frames) with a clock frequency of 450 MHz. The proposed design is written in VHDL hardware description language, and the design is compiled with Synopsys Design Compiler version K-2015.06. The design primarily consists of an SAD tree block, and control logic which sets the data address and performs other logic functions. The architecture is verified by simulating the design in ModelSim. Synopsys Armenia Educational Department (SAED) design kit standard 32 nm logic-cell libraries are used for synthesizing the design with operating conditions 1.16 V and 125 °C. The synthesized design has an area of 1.488 mm^2 when the design is constrained for a clock frequency of 450 MHz, and the compiler easily meets the constraint. Hence the design can be used for motion estimation of 4K UHD at 60 Hz videos in real time. The synthesized design has a 586k standard NAND-gate equivalent gate count.

Table 7.7 shows a comparison of different motion-estimation hardware architectures in the literature. The design proposed in [63] uses a modified TZ search where the search is restricted to an angle ±45°, but this can only be true for videos where the error surface increases monotonically as the search moves away from the global minimum (a unimodal error surface); that is not the general case for videos, where the error surface is usually non-unimodal with multiple local minima [60]. Furthermore, the proposal omits block sizes 64×32 and 32×64 and the raster scan from the search algorithm to reduce the

Table 7.7: Comparison of motion estimation architectures.

Design	This work	TCSVT'15 [63]	EL'13 [65]	JSTSP'13 [116]	JSSC'14 [117]	TCASVT'13 [120]
Technology (nm)	32	90	65	65	40	130
Clock freq. (MHz)	450	270	250	200	210	200
Gate Count (k)	586	206 [a]	3560	1068	2458 [b]	143 [b]
Memory (kB)	nil	11.9 [a]	20.23	89	552 [b]	256 [b]
Video format	4096 × 2160p60	4096 × 2160p60	3840 × 2160p30	3840 × 2160p30	7680 × 4320p48	1920 × 1080p30
Video standard	HEVC	HEVC	HEVC	HEVC	AVC	AVC
Search range	±64	±64	64	±64	±107H ±56V	±64
Supported blocks	All HEVC except AMP	All HEVC excluding AMP, 32 × 64, and 64 × 32	All HEVC	16 × 16, 32 × 32, and 64 × 64	All AVC	8 × 8 to 16 × 16
Frame types	P/B	P/B	P	P/B	P/B	P
BD-rate increase	nil	5.14%	-	12%	-	-

[a] Only integer motion estimation is considered

[b] Including fractional motion estimation engine

number of search points in order to speed up the motion search. These modifications in the algorithm considerably degrade its performance, as is seen from a BD-rate increase of 5.14% on average. The hardware gate count of [63] is a bit more than one-third that of the proposed architecture but requires approximately 11 kB memory for the integer motion sections of the proposal. On the other hand, the proposed architecture implements memory using registers as is essential for row/column-wise data movement, which is not easily achievable with RAM, and these registers are included in the gate count. Although the proposed design consumes more logic gates, a direct comparison is a little misleading because our design makes use of registers which increases our logic requirements.

The work in [65] proposes a full-search algorithm with a significantly higher gate count (3.56M). The full-search algorithm always seems better than fast-search algorithms like TZ searches, but require substantially higher computation time than fast-search algorithms. Hence, to achieve real-time encoding of UHD videos, the proposal in [65] reduces the search window to a 64×64 pixels area, which is one-fourth of the search area of the design in this paper; the reduced search area affects its performance or BD rate, especially for videos having higher motion content. The proposal in [65] is only capable of encoding at half the frame rate (30 Hz), but generally 4K UHD videos use a 60 Hz frame rate. Also, it uses only one reference frame and thus it may not be possible to encode ‘B’ frames of 4K UHD frames in real time; ‘B’ has a remarkably smaller bit rate than other frame types.

The cost-and coding-efficient (CCE) motion-estimation engine designed in [116] only uses block sizes 16×16, 32×32, and 64×64. The algorithm has an independent coarse search and localized 3-step search stages, and the better of these two parallel search results is taken as the motion vector. Even though this has a gate count approximately twice the architecture gate count proposed in this paper and an additional 89 kB of memory, the omission of several block sizes induces a bit-rate increase of 12% [116].

Since HEVC/H.265 is much more complex than its predecessor, a direct comparison of motion-estimation engines proposed for AVC/H.264 is not a fair approach. Nevertheless, some of the hardware designs for AVC/H.264 are given in Table 7.7. A hardware architecture is proposed for UHDTV applications in [117], which includes a fractional-motion-estimation engine as well. The hardware architecture achieves approximately 3 times the pixels processing of the design in this paper but has a 4 times gate count as well as a much a higher internal memory prerequisite. Another proposal in [120] is also for AVC/H.264 and includes a fractional-motion-estimation engine, has less gate count but is only capable of processing FHD videos with 30 fps (frames per second), and also ignores some block sizes inside the AVC/H.264 specification.

The full-search hardware designs have a higher hardware cost as is seen from the results of [65, 117], whereas simplified fast-search approaches for hardware such as [63, 116] have a lower hardware cost but worse search performances. In this proposal, we describe a modified TZ search algorithm which is suitable for hardware implementation in such a way that it retains its performance compared to the TZ search algorithm in the reference software (HM) application.

7.7 conclusion

This paper introduces a search algorithm which is suitable for hardware realization. From our observations the majority of motion vectors are found near the predicted motion vectors. Therefore, a full search around this location brings the best motion vectors in most cases, and searching other areas results in a wastage of time or energy. However, due to the non-uni-modal nature of the error surface, it must also test some locations outside of this adjacent region, and if a minimum is found outside the adjacent region a raster-like scan is necessary to avoid local minima traps. Therefore, the proposed methods use two search methods, an initial search and a raster-like search, which search patterns are appropriate for a hardware motion-estimation architecture. Since the initial search is a full search around the predicted motion vector, the proposed algorithm was found to be better than the TZ search in many cases.

The synthesized results with SAED 32 nm libraries show that the hardware can process 4K UHD @ 60 fps videos in real time. The major drawback found in many hardware-optimized designs for motion search in HEVC/H.265 is that the motion-search performance deteriorates due to simplification of the hardware to meet the new challenges in HEVC/H.265. This performance deterioration is practically imperceptible in the proposed architecture in this paper. The hardware also utilizes an RNS to increase the computational speed, as the RNS does not have the carry-propagation delay which reduces the speed of traditional binary systems. The proposed architecture along with the suggested algorithm achieves better PSNR and BD-rate results, maintaining the hardware cost approximately equal to other proposals in the literature.

Chapter 8

A Novel Angle-Restricted Test Zone Search Algorithm for Performance Improvement of HEVC [1]

8.1 Abstract

High Efficiency Video Coding (HEVC) is the latest video encoding standard and has approximately 50% bit-rate saving compared to its predecessor. However, the motion estimation (ME) is considerably complicated by the incorporation of varieties of partitioning modes and a quad-tree based coding structure, and also by increasing the basic coding unit size by a factor of 16. Motion estimation is the most complex task in the video encoding process, consuming 60-80% of overall encoding time. This paper proposes a new algorithm, angle-restricted test zone (ARTZ) for motion estimation which is based on a test zone (TZ) search, exploiting directional probabilities of motion vector search. In our

---

[1] N. C. Vayalil, M. Paul, and Y. Kong, "A novel angle-restricted test zone search algorithm for performance improvement of HEVC", *IEEE International Conference on Image Processing (ICIP)*, 2017, *accepted*

experiments, this proposal achieves a time saving in motion estimation of about 20% to 50% compared to a TZ search in the HEVC test model (HM) implementation for UHD videos without significant degradation of PSNR.

8.2 Introduction

HEVC/H.265 [10] is a video encoding standard proposed by the Joint Collaborative Team on Video Coding (JCT-VC), a collaboration between ISO/IEC MPEG and ITU-T VCEG. HEVC can achieve an equivalent subjective reproduction quality with approximately 50% less bit rate on average than its predecessor [57], AVC/H.264 [9]. A primary driving factor to improve video coding efficiency is the recent high demand for ultra high definition (UHD) video content in consumer devices. It is expected that video traffic will be 82% of all consumer Internet traffic by 2020, a 70% increase from 2015 [124]. The coding efficiency along with the computational complexity increases with every new proposal as in the case of MPEG2 and AVC [108]. In HEVC the size of the basic CU block is 64×64 whereas it is 16×16 for its predecessor, and also introduced quad-tree structuring as well as a variety of partitioning types. Larger block sizes are useful for compression as it could efficiently compress a larger area, in particular for high-resolution videos, where smoother or similar regions may be found in larger blocks.

Motion estimation is the most computationally intensive task in video compression, consuming 60% to 80% of the total encoding time [58]. Due to this high complexity in ME, numerous algorithms are proposed to tackle its complexity [77,125,126]. The TZ algorithm is one of the best fast-search algorithms, providing a good RD and a 60% improvement in encoding time compared to full-search algorithms. There are several proposals for TZ search improvement by reducing the number of search points, by changing the search pattern of its initial grid search to hexagon, pentagon or similar [127–129] instead of a diamond/square pattern. However, it seems that they suffer considerable PSNR (peak signal-to-noise ratio) degradation of from 0.026 dB to 0.137 dB. The proposal in [32] avoids searching after a certain distance in the initial grid search of the TZ algorithm, assuming a monotonic error surface, but generally this assumption is not valid.

This paper proposes a new algorithm; ARTZ search, based on the directional properties and probability of finding global minima in a search for motion vectors. The experimental observations reveal that ultimate motion vectors are in proximity of predicted motion vectors and the searching of motion vectors follows certain directions. The proposed method mainly targets UHD videos as they requires larger computations for motion estimation than high definition (HD) or lower resolution videos. The algorithm is based on the TZ search, a preferred algorithm in HEVC, also implemented in HEVC reference software, HM [33]. This proposed method considerably reduces search points by discarding the least-probable locations to find minima, thus achieving a speed-up of computation of ME of from 20% to 50% compared to standard TZ search depending on the nature of the video.

8.3 Overview of Test Zone Search Algorithm

A flow chart of the TZ search algorithm is given in Fig. 8.1. The TZ search starts with an initial grid search, either using a diamond or square search pattern with different stride lengths ranging from 1 to ‘search length’ in multiples of two. An 8-point diamond grid pattern used for an initial grid search is shown in Fig. 8.2 (a). The search starts with a predicted motion vector as its search center. Initial grid searches find a minimum distortion point, usually using a sum of absolute differences (SAD) [80] as a block matching criterion, because SAD is the simplest of the various matching criteria. If the distance from the search center to this minimum distortion point (best distance) is greater than a

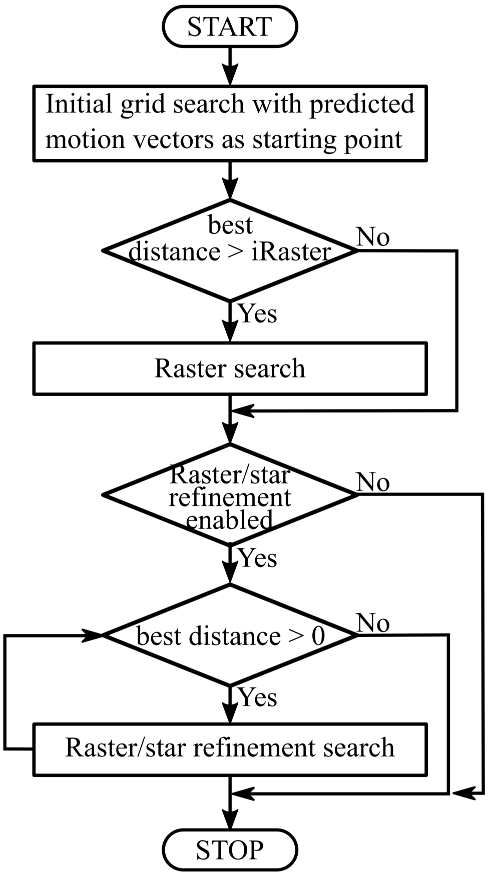

START
Initial grid search with predicted motion vectors as starting point
best distance > iRaster
No
Yes
Raster search
Raster/star refinement enabled
No
Yes
best distance > 0
No
Yes
Raster/star refinement search
STOP

Fig. 8.1: Flow chart for the Test Zone Search Algorithm.

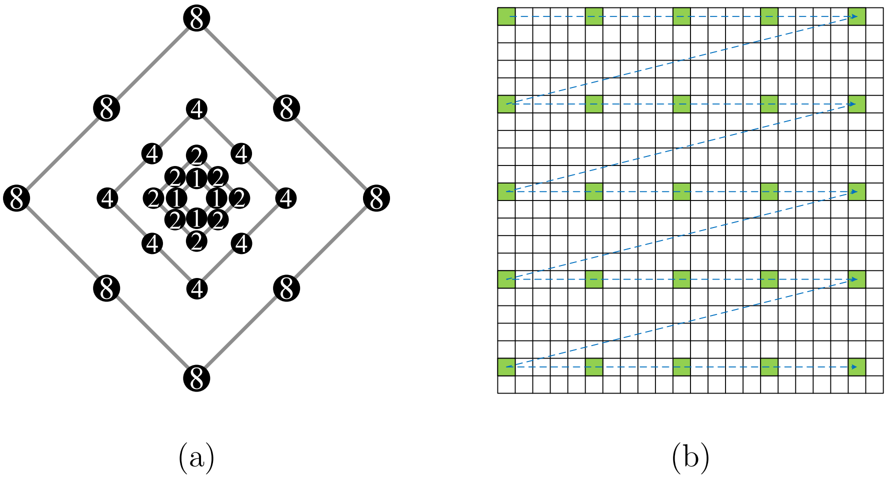


Fig. 8.2: Search pattern for initial grid search and raster search in test zone (TZ) search algorithm: (a) diamond pattern for initial grid search, (b) raster scan pattern.

predefined value, called 'iRaster' in HM, then the algorithm does a raster search; otherwise, skip this step. The raster search is a kind of full-search algorithm where the search window is sub-sampled by a factor equal to 'iRaster', which is set at compilation time. A raster-search pattern with an 'iRaster' equal to 5 is shown in Fig. 8.2 (b). The next step in this TZ search algorithm is a raster/star refinement if enabled. Generally, only one of them is enabled to speed-up computation. In this step, the algorithm makes a fine adjustment to the motion vector obtained from the previous steps. Both refinements use a diamond or square pattern, and they differ in their search operation. In the raster scan, the stride length of the search pattern is halved in every step and it also changes the search center, whereas the star refinement is similar to the initial grid search except changing its starting point to the minimum distortion point in every round. The refinement search stops when the best distance becomes zero.

8.4 Proposed Algorithm

Table 8.1 shows the number of raster searches involved in the TZ search algorithm for different 4K UHD resolution videos when encoded as P-frames using 4 reference frames. Although the error surface for the motion search is not monotonic, from observing various videos it seems that most of the motion vectors are not too far from the predicted location. This is evident from the table, as in some cases only 0.87% of motion searches involve a raster search, where the criterion for doing a raster search is that the distance to the minimum error location from the search center is larger than 5 pixels. In other words, the majority of motion vectors are found within a 5-pixel radius of the predicted location. Another important property from our observations is that the minimum error location in each step of the TZ motion search algorithm moves in a single direction or does not change its movement direction drastically. In this experiment, we traced

Table 8.1: Percentage of raster searches in a TZ search

Video Name	Frame No.	No. of Raster searches (%)
ToddlerFountain [a]	20	14.99
DrivingPOV [a]	29	0.87
TunnelFlag [a]	27	7.92
Crosswalk [a]	22	5.22

[a] 4K UHD video, encoded as P-frames using 4 reference frames

the movement of the minimum point in each step of the TZ search using HM reference software. The first 30 frames of the 4K UHD videos were encoded using configuration file ‘encoder_lowdelay_P_main.cfg’, which encodes all frames except the initial one (I frame) as P frames with 4 reference frames. The motion vectors which involve a raster scan have a particular interest because others (motion vectors found without raster scan) are in the close vicinity of the predicted motion vector, and thus we do not expect any directional properties for their minima movements. In the TZ search algorithm, if the minimum is found outside of the predefined ‘iRaster’, the next step involves a raster search, which is equivalent to a full search where the search window is sub-sampled by a factor equal to ‘iRaster’, i.e. searching blocks in increment of ’iRaster’ pixels in both horizontal and vertical directions. The initial searches, involving raster scan searches, are monitored for various video sequences and the point where the first minimum occurs after an ‘iRaster’ distance is noted, and the percentages of such incidents are tabulated in Table 8.2. The raster search modifies or finds a better minimum if the initial search lies outside of ‘iRaster’ because it covers all the blocks within the ‘iRaster’ distance. For this reason the proposal in [32] entirely avoids a search after the ‘iRaster’ distance in the initial grid search of the TZ search algorithm. This assumption is only valid for a monotonic error surface; that is not the case generally, thus there is a good probability of finding a minimum

Table 8.2: Percentage of first minimum error locations occurs after the ‘iRaster’ (5 pixels) distance

	% of first minima at the distances					
Video Name	8	16	32	64	128	256
ToddlerFountain, frame no. 25	25.1	44.5	16.0	7.9	4.3	2.2
DrivingPOV, frame no. 27	51.8	35.9	7.9	2.8	1.1	0.5
TunnelFlag, frame no. 19	20.2	45.4	14.9	9.4	6.3	3.8
Crosswalk, frame no. 25	27.5	48.6	11.8	6.6	3.6	1.9

after the ‘iRaster’ distance even though the error is increasing till the ‘iRaster’ distance (or even more than ‘iRaster distance’), which is evident from Table 8.2. In some video sequences, approximately more than 80% of minimum locations first arise after a distance of 8 pixels; if these points are skipped 80% of raster scans would not happen in the TZ search algorithm for some cases, which affects finding global minimum error locations and predictions of further blocks. In these observations, it is also found that in several cases the minimum location updates follow the direction of previous updates. Fig. 8.3 shows search directional changes in the initial search of the TZ algorithm for the 22$^{\mathrm{nd}}$ frame of the Netflix_Crosswalk video sequence. Only searches having minima outside of the ‘iRaster’ distance are used for the plot. From the figure, it is clear that large number of searches (28.41% in total) follow the previous direction (0°) especially at the 3$^{\mathrm{rd}}$ step of the initial diamond search. In the 4$^{\mathrm{th}}$ and 5$^{\mathrm{th}}$ steps, 21.1% and 4% respectively are at ±90°. After this step (i.e. distance larger than 32 pixels) only a few minima are found, approximately equal to 10% in total. In a similar way, the videos in Table 8.2 also analyzed; even though the results vary, they all follow a similar pattern for the probability of finding minima in the search steps. These videos are selected because of having high motion content in 4K UHD resolution. It is obvious that higher resolution and high motion demands more

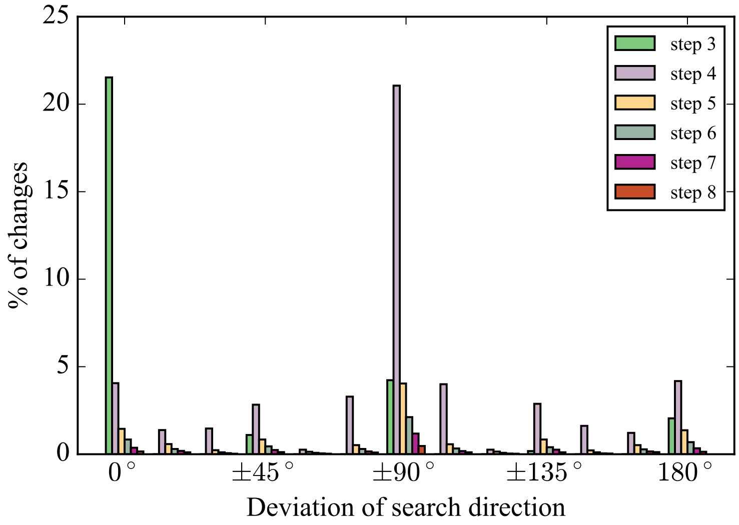


Fig. 8.3: Change of search direction in each step of initial search in the TZ Search Algorithm. This image shows the location of minima changes in the 22$^{\text{nd}}$ frame of the Netflix_Crosswalk video sequence.

computations, also the motion estimation characteristics of 4K UHD videos differ from the lower resolutions.

The algorithm was developed based on the above results; the search points are restricted to certain locations after analyzing the probability of each point in the diamond-grid search. Search all points at step 4 (16 pixel distance), and search restricted to 0°, ±90° and 180° points for step 3 (8 pixel distance), and 0° and ±90° points for step 5 (32 pixel distance). No other points in the initial diamond search seem to have a higher probability of finding a first minimum; thus they are omitted from the initial grid search, hence saving considerable computational effort for motion estimation. These changes reduce the computational effort for the initial grid search by approximately 56% (HM uses a 16-point diamond grid if the distance is more than 8 pixels) if the search distance is set at 64 pixels, but the TZ search has a raster search in certain cases, which is the most time-consuming part in this algorithm. Hence the time saving for ME depends on how

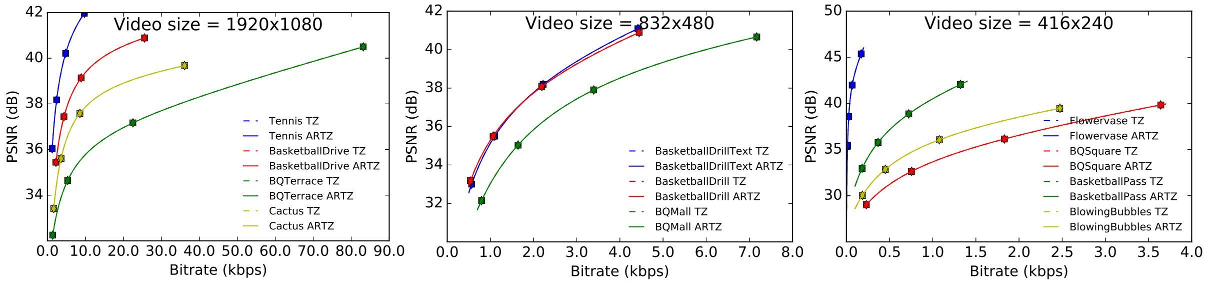


Fig. 8.4: RD curves for different video sequences with QPs 22, 27, 32, 37, with search length 64.

many times the algorithm does this raster search, which in turn depends on the motion characteristics of the video.

8.5 Simulation and results

Computation time for the motion-estimation function is estimated by reading CPU cycles from the CPU's hardware cycle counter according to the benchmarking instructions provided in [130]. The simulations are run in the CentOS 6.8 platform on an Intel® Xeon® CPU @ 2.67 GHz, having disabled hyper-threading and frequency scaling for getting accurate timing results. The first 30 frames of videos with 4K UHD are encoded with and without modification in the HM software for various test videos, and the obtained results are tabulated in Table 8.3. All video frames except the initial I-frame are encoded as P-frames using one reference frame, the QP set as 32, and with the three search ranges of 64, 128, and 256 pixels in both horizontal and vertical directions. From the results it is clear that PSNR degradation with the standard TZ search algorithm is negligible; for the worst case, it is 0.013 dB with a search range of 64 pixels in 4K UHD video. Bit-rate increases are also insignificant in the results, which have a worst-case value of 0.459%. It seems that in some cases the proposed ARTZ algorithm is better able to track global minima instead of being trapped by local minima due to its directional properties,

Table 8.3: ARTZ search algorithm results (negative values are improvements) with different search lengths for 4K UHD resolution

Video Sequence	Search range	Bitrate increase (%)	PSNR reduction (dB)	ME time saving (%)
TunnelFlag	64	0.291	0.013	39.9
TunnelFlag	128	0.2693	0.0035	44.2
TunnelFlag	256	0.459	0.0048	39.2
ToddlerFountain	64	−0.0016	−0.0001	25.5
ToddlerFountain	128	0.0057	−0.0008	24.0
ToddlerFountain	256	−0.0105	0.0005	20.7
DrivingPOV	64	−0.0485	−0.0009	42.9
DrivingPOV	128	−0.3278	−0.0011	50.3
DrivingPOV	256	−0.0675	0.003	49.5

thus improving PSNR and bit rate (negative values indicates improvements by ARTZ) in several cases. Although the algorithm is developed for 4K UHD videos, the experiments with standard test videos of different sizes reveal that the proposed algorithm performance closely follows that of an HM software implementation of a TZ search as seen in the RD curves [123] plotted in Fig. 8.4, and from Table 8.4.

8.6 Conclusion

The algorithm proposed for computation saving in motion estimation is based on the probability of obtaining minima in certain locations of an initial diamond grid-search. The proposal is mainly intended for UHD-resolution video encoding in the HEVC standard. The proposed ARTZ algorithm improves the computation time for motion estimation considerably, with a maximum of 50.3% time saving compared to the standard TZ search

Table 8.4: ARTZ results with search length 64 and different QPs

Video sequence	QP	Bitrate increase (%)	PSNR reduction (dB)	ME time saving (%)
Tennis	37	0.0342	0.0022	33.6
Tennis	32	0.0167	0.0022	31.6
Tennis	27	−0.027	0.0019	30.1
Tennis	22	0.0596	0.0004	28.5
BasketballDrive	37	−0.0284	0.0007	36.9
BasketballDrive	32	0.0384	0.0002	35.6
BasketballDrive	27	−0.0132	0.0006	34.2
BasketballDrive	22	−0.0242	0.0007	31.9
Cactus	37	0.026	−0.0021	39.4
Cactus	32	0.0425	0.0044	39.1
Cactus	27	0.0391	−0.0004	38.5
Cactus	22	−0.0818	0.0004	38.2
BasketballDrillText	37	0.3566	0.0095	39.5
BasketballDrillText	32	0.1318	0.0056	38.0
BasketballDrillText	27	−0.0446	0.0027	37.8
BasketballDrillText	22	−0.0139	−0.0062	37.8
BasketballDrill	37	0.5209	0.0181	39.6
BasketballDrill	32	−0.0224	−0.0056	38.8
BasketballDrill	27	0.2136	0.0001	37.8
BasketballDrill	22	−0.0052	0.0055	37.6
BQMall	37	−0.137	0.0119	39.6
BQMall	32	0.0402	−0.0047	39.6
BQMall	27	0.0247	0.0023	39.4
BQMall	22	−0.0269	−0.0031	39.8
BQSquare	37	0.0294	−0.0012	41.6
BQSquare	32	0.2745	0.0023	41.4
BQSquare	27	−0.0336	−0.0024	40.7
BQSquare	22	−0.0792	−0.0053	41.2
BasketballPass	37	0.0	0.0017	40.0
BasketballPass	32	0.0936	0.0054	39.0
BasketballPass	27	−0.0962	0.0087	38.9
BasketballPass	22	−0.0107	0.0078	40.4
BlowingBubbles	37	−0.0759	−0.0239	37.6
BlowingBubbles	32	−0.2183	−0.0015	37.6
BlowingBubbles	27	−0.0933	0.0032	37.1
BlowingBubbles	22	0.0911	−0.0059	37.5

algorithm. The worst-case bit-rate degradation and PSNR degradation are 0.459% and 0.013 dB respectively, which are negligible, and for the best-case bit rate and PSNR are improved by 0.52% and 0.0239 dB respectively. Hence the modified algorithm can be used for encoding in HEVC and may also be suitable for other encoding standards such as AVC.

Chapter 9

An Efficient ASIC Design of Variable-Length Discrete Cosine Transform for HEVC [1]

9.1 Abstract

The latest video coding standard introduced by the joint collaborative team on video coding (JCT-VC) is known as high-efficiency video coding (HEVC) or H.265. HEVC/H.265 is mainly targeted for high-definition videos, and offer more compression than its predecessor. The discrete cosine transform (DCT) is widely used for image and video compression including HEVC. This paper proposes a variable-length DCT architecture for encoding video according to the HEVC/H.265 specifications. The architecture is optimized for most likely block sizes in ultra-high definition (UHD) video, and eliminates unnecessary complexities found in many architectures proposed. The synthesized results with Synopsys design tools

---

[1] published as: N. C. Vayalil, J. Haddrill, and Y. Kong, "An efficient ASIC design of variable length discrete cosine transform for HEVC", *2016 European Modelling Symposium (EMS)*, Nov. 2016, pp. 229-233.

show that the proposed method can encode 8K UHD videos @ 60 fps in real-time and accomplishes more than 60% in hardware savings.

9.2 Introduction

As the demand for HD video content increases, so does the need for efficient compression techniques. The HEVC/H.265 standard [10] is a relatively new codec that is poised to replace AVC/H.264 [9] as the standard for high-definition video encoding. HEVC/H.265 offers more compression, approximately a 50% bit-rate reduction, than its predecessor AVC/H.264 for an equivalent subjective reproduction quality [57]. The use of the DCT is a common method in several previous codecs and could be a key factor in the development of compression techniques for HEVC due to its near-optimal efficiency for performing this task. To be compatible for proper use with HEVC/H.265 the DCT needs to be computed for a matrix of varying length.

To accommodate the varying size of the architecture it would be ideal to develop components that can be utilized by other lengths such that the architecture is more area efficient, but the common method of multiplying by a constant matrix would not be effective in this case, due to its architecture not being able to be reused for other lengths.

Mehr et al proposed a reusable integer DCT architecture [131] providing same throughput in all supported transform lengths, but resulting in a higher area or gate count. An approximated architecture of DCT through the Walsh-Hadamard Transform (WHT) followed by a set of Givens rotations in [132] reduces gate count. Another approximate DCT architecture is proposed in [133] and offers better PSNR. High-resolution video such as UHD video is more likely to have large smoother regions [118], thus transforms of larger size are mostly used. Hence the design is targeted mainly for the most likely block sizes instead of all possible sizes, and this assumption can reduce the hardware complexity significantly. This paper proposes a 2D-DCT architecture which has substantial throughput, with block sizes 16×16 and 32×32 for real-time encoding of 8K UHD. This architecture leads to a simple memory and DCT hardware structure and thus is smaller (lower gate count) and faster than many proposals in the literature.

9.3 Hardware Architecture for DCT computation

The DCT is a Fourier-related transform that only uses real numbers to represent a set number of discrete data points within a signal; unlike the discrete Fourier transform (DFT) the DCT only uses cosine functions to represent the data points [134]. There are multiple versions of the DCT that range from DCT-I to DCT-IV, the most common of which is DCT-II and referred to as 'the DCT' and defined as

$$X_k = \sum_{n=0}^{N-1} \cos\left[\frac{\pi}{N}(n+\frac{1}{2})k\right] x_n \quad 0 \leq k < N \tag{9.1}$$

For processing two-dimensional signals such as images, a two-dimensional version of the DCT (2D-DCT) is used; it is a trivial expansion of the standard DCT, given as

$$\begin{aligned} X_{k_1,k_2} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \cos\left[\frac{\pi}{N_1}(n_1+\frac{1}{2})k_1\right] \\ \cos\left[\frac{\pi}{N_2}(n_2+\frac{1}{2})k_2\right] x_{n_1,n_2} \end{aligned} \tag{9.2}$$

where $0 \leq k_1 < N_1, 0 \leq k_2 < N_2$.

One property of the 2D-DCT is separability [135], i.e. the 2-D DCT can be computed in two steps, a column-wise 1-D DCT followed by a row-wise 1-D DCT, or vice versa. This procedure of calculating a multidimensional separable transform is called row-column decomposition, which reduces the number of computations.

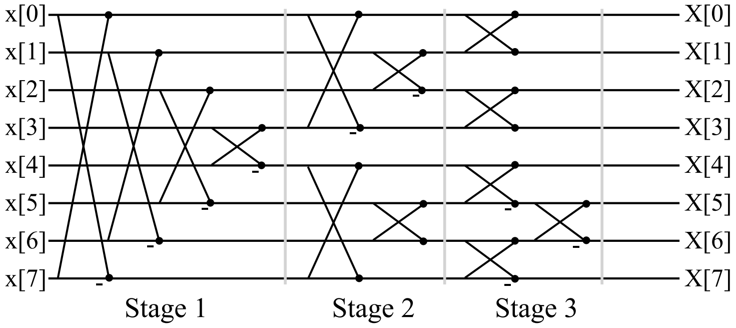


Fig. 9.1: Stick diagram of the butterfly technique applied to the DCT.

The DCT has become a staple in image compression, specifically in the JPEG format, due to the resulting lossy compression that occurs as a result of the transform and quantization, allowing larger image data to be compressed. This is done by applying the DCT to a quantization of an image's pixels to obtain an approximation that requires less data to be stored. DCT possesses a strong energy compaction property [136]; most of the signal information tends to be concentrated in few low-frequency components, making DCTs useful for image compression.

The related Fourier properties of the DCT make it possible to use the butterfly multiplication approach described by Budagavi et al, in such a way that the overall transformation completes in sections that effectively 'fold' into the next section [137]. This method is ideal as it is more efficient than the brute-force matrix multiplication method which is very costly in terms of computing time [137]. A stripped-down representation of this process can be seen in Fig. 9.1, where the horizontal lines represent the input and manipulated data after operations while a (−) beneath the dot represents subtraction or addition.

Table 9.1: Four-Point DCT Algorithm by Stage

Stage	Computation	Binary expression	Notes
Stage 1 (IAU)	$a(i) = x(i) + x(3-i)$ $b(i) = x(i) - x(3-i)$		for $i = 0$ to 3
Stage 2 (SAU)	$m_{i,9} = 9b(i)$ $m_{i,64} = 64a(i)$ $t_{i,83} = 83b(i)$ $t_{i,36} = 36b(i)$	$(b(i) << 3) + b(i)$ $a(i) << 6$ $(b(i) << 6) + (m_{1,9} << 1) + b(i)$ $m_{1,9} << 1$	for $i = 0$ to 3
Stage 3 (OAU)	$y(0) = t_{0,64} + t_{1,64}$ $y(1) = t_{0,83} + t_{1,36}$ $y(2) = t_{0,64} + t_{1,64}$ $y(3) = t_{0,36} + t_{1,83}$		

9.3.1 Four-point DCT architecture

The four-point DCT module is based on the algorithm by Meher et al [131]. The algorithm used to implement the DCT for a 4 × 4 matrix is outlined in stages in Table 9.1. For efficient implementation the algorithm is divided into an input adder unit (IAU), a shift adder unit (SAU) and an output adder unit (OAU), such that each stage can be examined and implemented individually to ensure that necessary values are available as each stage completes. This algorithm is used to replicate the kernel matrix represented by equation (9.3) to perform the transform without directly performing matrix multiplication. This is done to improve the computational speed and efficiency of the architecture.

$$C_4 = \begin{bmatrix} 64 & 64 & 64 & 64 \\ 83 & 36 & -36 & -83 \\ 64 & -64 & -64 & 64 \\ 36 & -83 & 83 & -36 \end{bmatrix} \tag{9.3}$$

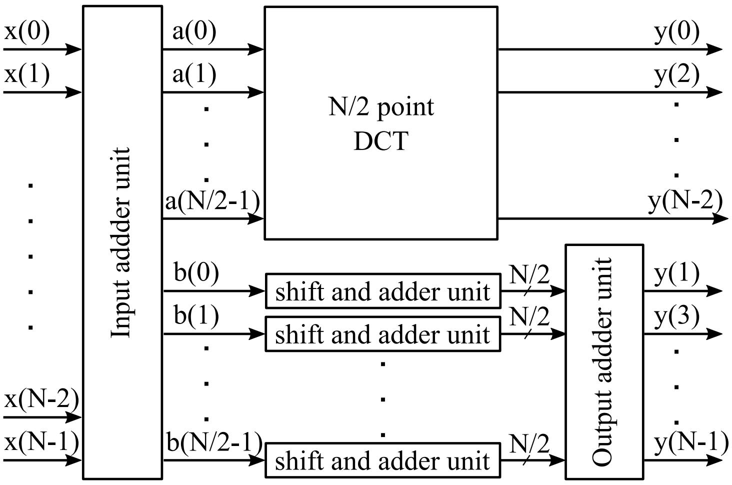


Fig. 9.2: A generalized structure of higher radix DCTs, where N = 8, 16, 32. [131]. The N point DCT is build upon $N/2$ DCT with adder units and shift units.

9.3.2 Architecture for higher length 1-D DCTs

Higher-length DCTs with N = 8, 16 and 32 are built upon 4 point DCTs recursively. A generalized structure of $N = 8, 16, 32$ point integer DCTs are shown in Fig. 9.2. At each stage of this process the input data is first manipulated by an IAU to create intermediate data that is then used as the input for further operations. The even-numbered rows/-columns, including zero, are processed as an $N/2$ point DCT to obtain the corresponding output values. The odd-numbered rows and columns are passed through a SAU, which is specific to the point length of the DCT being performed, to produce the corresponding values in the output matrix.

Once the lower level transforms and operations have been completed the resulting data is then further manipulated by an OAU to complete the transformation. The complete architecture implementation operates recursively by gradually calling $N/2$-point DCTs until it reaches the four-point DCT.

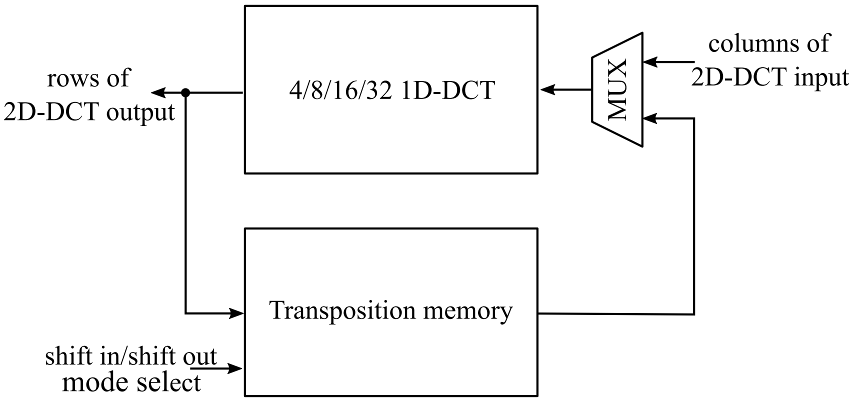


Fig. 9.3: The proposed 2D-DCT architecture, transposition memory implemented using a 2-D register array.

9.4 Proposed hardware architecture for variable-length two-dimensional DCT

The separability property is used to design the 2-D DCT because the row-column decomposition results in computational savings but this introduces another problem of data storage or memory for saving first-step results. It is clear that the second step (row/-column 1D-DCT) can only begin after completing the first step (column/row 1D-DCT), thus it is necessary to save all data and retrieve it in a transposed order. In this proposed architecture we use a 2 dimensional register array to save and transpose first 1-D DCT results, which results in an efficient 2D-DCT architecture.

The architecture of the proposed method is shown in Fig. 9.3, where the transpose module has a size of 32×32 words (1-D DCT input and output have 16 bit word length) which can hold all column transforms of a 32×32 blocks pixels. Initially, the 2D shift registers are set into 'shift-in mode' and input data is given into the 1D-DCT module column-wise. During the 'shift-in mode', the results of the first 1D-DCTs are stored into the leftmost column of the 2D register array, and each column of data in the register array is shifted right-ward at every clock cycle. A detailed digram of the 2D shift register

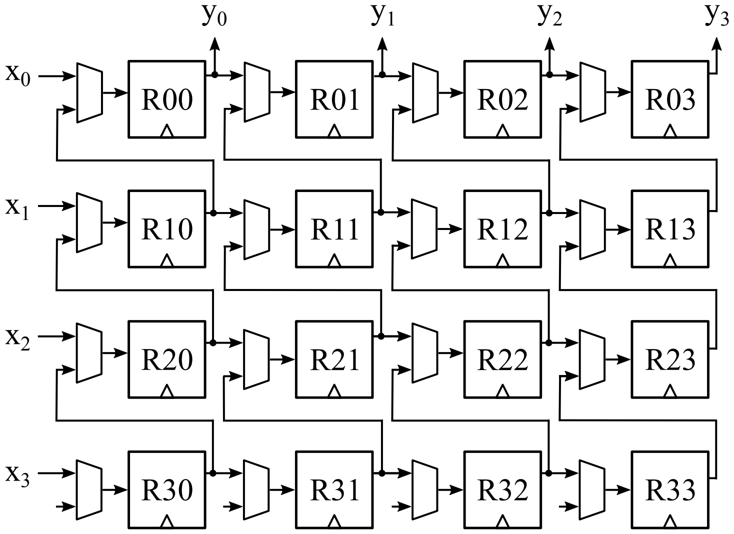


Fig. 9.4: The proposed 2D shift register architecture, showing 4 inputs and 4 outputs. Data is shifted in the horizontal direction from left to right, and shifted out in the up direction; all MUX selection changes accordingly.

arrangement is shown in Fig. 9.4.

After completion of all column transformations, the 2D shift register changes into 'shift-out' mode and the DCT module input connects to the 2D shift register outputs with the help of a multiplexer (MUX). In 'shift-out' mode the shift register's data shifts in the upward direction, and the output is taken from the top row. This write and read arrangement facilitates the transpose operation. The shift registers do not load data in this mode of operation. Since the DCT module completes a row transform in each cycle, the proposed architecture requires $2N$ clock cycles to complete an N point 2D-DCT transform. An an example, for a 32 point 2D-DCT, the first 32 clock cycles are required to complete all column transforms which are stored into the shift registers, and another 32 clock cycles are required to shift-out these data row-wise and complete all row transformations.

In HEVC/H.265, the transform is performed after intra and inter prediction, on the

Table 9.2: Comparison of 2D-DCT architectures

Design	Technology	Gates	Max. Freq.	Throughput	Supported format
TCSVT'14 [131] Arch. 1	90 nm	347 k	187 MHz	5.984 G	8K UHD @ 60 fps
TCSVT'14 [131] Arch. 2	90 nm	208 k	187 MHz	2.992 G	8K UHD @ 60 fps
TCSVT'16 [132] Arch. 1	90 nm	243 k	250 MHz	3.212 G	8K UHD @ 64 fps
TCSVT'16 [132] Arch. 2	90 nm	157 k	250 MHz	1.302 G	8K UHD @ 26 fps
Proposed	32 nm	96 k	450 MHz	3.600 G	8K UHD @ 60 fps

residues obtained by the differences between the original pixels and predicted pixels. For the residual coding, HEVC/H.265 employs recursive quad tree-structured partitioning of coding blocks [10]. The HEVC/H.265 specification supports four transform sizes: 4×4, 8×8, 16×16 and 32×32. The different block sizes in the specification are introduced for accommodating varying space-frequency characteristics of the residuals. The RD cost computation is to be done for all CU sizes to select the best among the various block sizes. However this 'trial and error' method has a very high computational cost. Several algorithms are proposed for early TU decision reducing this complexity. Chio et al. propose a method for early TU decision by determining the number of nonzero DCT coefficients as a threshold to stop further RD cost evaluation in the quad tree structure [138]. But this method still has enough complexity, especially for sequences with active motion or rich textures, thus further optimizations are proposed in [139]. Quad-tree TU encoding process termination based on the residual coefficients is proposed in [140–143]

One of the options in the HM [33] to reduce the computational complexity is to use the largest available transform size. The homogeneity of the transform block residuals has a strong relation to the homogeneity of input block; when the TU covers multiple PUs these transform residues may not be consistent and also there is a chance for introducing blocks artifacts which in turn increases the high-frequency energy in the residuals. To cope with computational complexity and the aforementioned problems, this architecture decided to use the maximum TU size that fits in the PU as the TU size. This decreases the computational complexity but the BD-rate [123] increases by 3.02% in the low-delay P configuration [144].

9.5 Results and comparison

The proposed architecture is written in the VHDL hardware description language, and is verified by simulating the design in ModelSim. The design is synthesized using Synopsys Design Compiler version K-2015.06 with Synopsys Armenia Educational Department (SAED) design kit 32 nm standard logic cell libraries, for operating conditions of 1.16 V and a worst-case temperature 125 °C. The highest throughput of the architecture is 16 pixels per clock cycle while processing a 32×32 block within 64 clock cycles, and varies to a worst-case 2 pixels per clock cycle when processing blocks are in 4×4 size. In high-resolution video, especially for 8K UHD video, lower block sizes are rarely expected, hence as an average, throughput of 16×16 blocks are taken for calculation purposes. Thus to encode 8K UHD @ 60 Hz in 4:2:0 YUV format requires $7680 \times 4320 \times 60 \times 1.5/8$ clock cycles per second or 374 MHz. The design can operate up to 450 MHz, a much higher clock frequency than is required, and the synthesized results are in Table 9.2.

The synthesized design has an area of 0.2443 mm^2 or a 96 k standard 2-input NAND equivalent gate count. There are two architectures proposed in each of [131] and [132] for the 2D-DCT, based on unfolded and folded 1D-DCT modules, and are referred to as Architecture-1 and Architecture-2 respectively. The unfolded or full-parallel structures have higher throughput at the expense of larger area or gate count. A comparison is given in the table with the proposed architecture, and is clear that the proposed method has approximately 61% gate count of [132] Architecture-2 and is twice as fast. For an equivalent throughput, the proposed method saves more than 60% gate count of the designs in the table.

9.6 Conclusion

In this paper we propose a 2-D DCT architecture for encoding UHD video in the HEVC/H.265 standard. The hardware has substantial throughput for block sizes that are more likely to be found in HD or UHD video. This assumption removes several unnecessary complexities established in many other architectures. The proposed method has an efficient and fast DCT structures as well as transposition memory. Thus the synthesized results show a lower gate count or a smaller area than the architectures in the literature.

Chapter 10

Conclusions and Future Work

10.1 Conclusions

This dissertation has presented research on a video processor for the HEVC/H.265 standard with an RNS. Several novel algorithms and architectures were proposed and investigated to achieve encoding of very-high-density video such as 4K UHD in real time. As motion estimation is the bottleneck in video compression, the research provided a number of alternative solutions in ASIC implementation and also presented an algorithm for software applications. Although RNS has been used for designing high-speed computation hardware, the research in video encoding is not carried out extensively; and this dissertation tried to address this gap. This dissertation showed that an RNS could advance the performance of arithmetic units, hence critical modules of video encoding such as motion estimation. Even though HEVC/H.265 is a complex proposal among video coding standards, efficient designs are still possible, and various optimised solutions were presented in this dissertation.

Motion estimation is the most computation demanding task in video processing and consumes 60% to 80% of the overall encoding time. Full-search motion estimation, which searches in every possible location in a search window, gives better PSNR and bit-rate reduction than any other methods. A motion-estimation architecture for a full-search algorithm was presented in Chapter 3, which brought down the data bandwidth by a factor of about 52 times from conventional architectures by broadcasting data into multiple processing elements, and an SAD reuse strategy. Morton order has been used for data reading which decreased the memory requirement, and its ASIC implementation results showed it is able to process 4K UHD videos in real time. The proposed design has 17% less gate count and 20 kB less memory than other full-search motion-estimation architectures in the literature.

Motion estimation requires enormous arithmetic operations, mainly addition and subtraction, where an RNS could improve the performance. Since the RNS is a non-weighted number system, the sign detection is substantially burdensome but is an essential component of motion estimation or other similar components. For instance, sign detection is required to implement an absolute difference, or as a magnitude comparison for finding minima in motion estimation. A novel algorithm based on CRT II and its fully combinational ASIC architecture were presented in Chapter 4, and a synthesised result with Synopsys Design Compiler showed improved speed, area and a 24% reduction in the area-delay product.

A distortion-measuring matrix is required for motion estimation for comparing different PBs, and SAD is commonly used, because of its simplicity compared to other criteria. Chapter 5 investigated the advantage of using an RNS system for SAD as it involves numerous addition and subtraction operations. A relatively prime moduli set has been used for the SAD design and the synthesised results had been compared with an equivalent non-relatively prime moduli set and a binary design, and showed higher speed than both of them with fewer pipelines (5 instead of 12 in the former design) using approximately 3% of the area.

Since the full-search designs involve a large number of computations and require much hardware resources, Chapter 6 described a TZ search-algorithm implementation in ASIC with an RNS to speed up the computations of its arithmetic units. The TZ search is a commonly used fast-search algorithm due to its performance near to that of full-search methods but having less computation complexity. Although there have been proposals for a TZ search for hardware, those are significantly simplify the algorithm and worsening the quality; for instance one implementation increases the bit rate by approximately 5%. Results showed that a parallelised version of the proposal could process 4K UHD video in real time with 51% less gate count than existing proposals and less memory requirements than most.

A major drawback in many proposals on motion estimation for hardware designs is the performance degradation, as they simplify the algorithms to fit the hardware. A fast-search motion-estimation algorithm suitable for hardware designs has been introduced in Chapter 7, utilising an RNS for improving the speed of arithmetic units, where performance degradation was practically absent, and was even improved in several cases. The design is capable of processing 4K UHD @ 60 fps videos in real time and the hardware resources used for ASIC design, i.e. the gate count, was even less than many other proposals.

Even though the error surface for the motion search is non-unimodal, in a good number of cases the motion search follows a directional tracking of minima. A novel algorithm based on TZ search, the angle-restricted test zone search (ARTZ) was proposed in Chapter 8, exploiting the directional probabilities of motion searches. This algorithm is targeted for software platforms, and reduced computations by from 20% to 55% without noticeable degradation of PSNR and bit rate compared to the TZ search in HEVC reference test model (HM) software.

After the motion estimation, a transform coding is applied to the residual image that results from subtracting the motion-compensated prediction frame from the currently encoding picture frame. DCT is widely used for transform coding because of its high energy-compaction property, including HEVC/H.265. It is a well-known fact that in several cases many simple designs provide better performance than complex, versatile one as in the principle behind reduced-instruction-set computer (RISC) CPU design. A variable-length DCT architecture required for the HEVC/H.265 standard was presented in Chapter 9, which optimised for the most-probable block sizes of 4K UHD video and eliminated the unwanted complexities found in many other proposals. The resulting design has plenty of computing power for encoding UHD video, which can also encode relatively low-density video such as HD resolution since it requires less computation. The hardware utilisation has been reduced to 60% that of proposals in the literature.

10.2 Future work

The research lays down a foundation for video encoding in HEVC/H.265 with an RNS. It seems that RNS has a good future in video processing, as the consumer demand for higher quality videos increases as time passes. The RNS can present better arithmetic units especially as the bit width increases, and an increasing trend of greater supported bit-width can be seen with every new proposal or standard. Hence further contributions may be possible with a smaller moduli set instead of the moduli set used in the dissertation, for videos where the pixels are represented with higher bit depths.

All the motion-estimation architectures described in this thesis used an approach of computing the cost function of PBs and selecting the best. However, it may be possible to make a pre-decision about block partitioning from its temporal neighbours, and this may avoid the overhead of computing the cost function of several partition types, and further enhance motion estimation computation speed. Further improvement to the PSNR and bit rate is possible with incorporating sub-pixel or fractional motion estimation to the proposed architectures for motion estimation. Fractional motion estimation refines the integer motion estimation by interpolating pixels around the integer motion vector followed by a fractional search process. HEVC/H.265 supports motion vectors with a precision of one quarter of the distance between luma pixels. There are several proposals for very-large-scale integration (VLSI) architectures of fractional motion estimation in the HEVC/H.265 standard [145–147], which can be combined with the proposed methods.

Appendix A

TCL scripts sample for fast search motion estimation design

```
# Top-level Module
set toplevel top

# All VHDL files, separated by spaces
set vhdl_files [list reg_array.vhd me_64.vhd control_64.vhd top_64.vhd]
#
set vhdl_libs [list lib_csa.vhd lib_adder.vhd lib_sign_det.vhd \
                    std_logic_array.vhd lib_rns.vhd pe_rns.vhd]


set vhdl_files [concat $vhdl_libs $vhdl_files]

set op_fname .saed

# Directories containing logical design and script files.
set additional_search_path    ”./rtl ./rtl/lib ./scripts”

```

```
#------------------------------------------------------------------------
#  Other settings (see .synopsys_dc.setup)
#-------------------------------------------------------------------------
# Common ALIB library location
set_app_var alib_library_analysis_path Synopsys/proj ;
define_design_lib WORK -path ./work   ; # Location of "analyze"d files

set search_path "$additional_search_path $search_path"

file delete -force work
analyze -f vhdl $vhdl_files

elaborate $toplevel

if {[link] == 0} {
    echo "Linking Error"
    exit
}
if {[check_design] == 0} {
    echo "Check Design Error"
    exit
}
# TCL command to create directories if not exists
file mkdir reports output

write -f ddc -hierarchy -output output/${toplevel}.unmapped.ddc

source top.con
uniquify
set_host_options -max_cores 4
compile_ultra
```

```
check_design > reports/check_design.rpt
compile_ultra -incremental
report_constraint -all_violators > reports/${toplevel}${op_fname}
    .const_violaters.rpt
change_names -rules verilog -hierarchy

write -f verilog -hierarchy -output output/${toplevel}${op_fname}.v
write_sdc output/${toplevel}${op_fname}.sdc
write -f ddc -hierarchy -output output/${toplevel}.mapped${op_fname}.ddc

report_area > reports/${toplevel}${op_fname}.area.rpt
report_cell > reports/cells.rpt
report_qor > reports/qor.rpt
report_resources > reports/resources.rpt
report_timing > reports/${toplevel}${op_fname}.timing.rpt
report_power > reports/${toplevel}${op_fname}.power.rpt

quit
```

Appendix B

List of acronyms

AMP	asymmetric motion partitioning.
APDZS	advanced predictive diamond zonal search.
ARTZ	angle-restricted test zone.
ASIC	application specific integrated circuit.
AVC/H.264	advanced video coding.
BBGDS	block-based gradient descent algorithm.
BD	Bjøntegaard delta.
CABAC	context-based adaptive binary arithmetic coding.
CB	coding block.
CIF	common intermediate format or common interchange format.
CRT	Chinese remainder theorem.
CTB	coding tree block.
CTU	coding tree unit.
CU	coding unit.

DCT	discrete cosine transform.
DS	diamond search.
DSP	digital signal processing.
DST	discrete sine transform.
EPZS	enhanced predictive zonal search.
ET	early termination.
FHD	full high-definition.
FIR	finite impulse response.
fps	frames per second.
GOP	group of pictures.
HD	high-definition.
HEVC/H.265	high efficiency video coding.
HM	HEVC test model.
IAU	input adder unit.
IIR	infinite impulse response.
IME	integer motion estimation.
JCT-VC	joint collaborative team on video coding.
LCM	least common multiple.

ME	motion estimation.
MSEA	multilevel successive elimination algorithm.
MV	motion vector.
MVFAST	motion-vector field-adaptive fast motion estimation.
NTSS	new three-step search algorithm.
OAU	output adder unit.
OWF	overlapped wavefront.
PB	prediction block.
PDE	partial distortion elimination.
PE	processing element.
PMVFAST	predictive motion-vector field-adaptive search technique.
POC	picture order count.
PSNR	peak signal-to-noise ratio.
PU	prediction unit.
QP	quantization parameter.
RAM	random access memory.
RD	rate distortion.
RNS	residue number system.
ROM	read-only memory.

SAD	sum of absolute differences.
SAU	shift adder unit.
SEA	successive elimination algorithm.
TB	transform block.
TSS	three step search.
TU	transform unit.
TZ	test-zone.
UHD	ultra-high-definition.
VBS	variable-block-size.
VLSI	very-large-scale integration.
WPP	wavefront parallel processing.

Bibliography

[1] M. Zhou, W. Gao, M. Jiang, and H. Yu, "HEVC lossless coding and improvements," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1839–1843, Dec 2012.

[2] R. Weerakkody, M. Mrak, V. Baroncini, J. R. Ohm, T. K. Tan, and G. J. Sullivan, "Verification testing of HEVC compression performance for UHD video," in *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Dec 2014, pp. 1083–1087.

[3] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, Sept 1952.

[4] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Transactions on Information Theory*, vol. 24, no. 5, pp. 530–536, Sep 1978.

[5] T. A. Welch, "A technique for high-performance data compression," *Computer*, vol. 17, no. 6, pp. 8–19, June 1984.

[6] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the h.264/avc video compression standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 620–636, July 2003.

[7] Y. Q. Shi and H. Sun, *Image and Video Compression for Multimedia Engineering: Fundamentals, Algorithms, and Standards.* CRC Press, Taylor & Francis Group, FL, 2008.

[8] ITU-T and ISO/IEC JTC 1, *Generic Coding of Moving Pictures and Associated Audio Information Part 2: Video*, ITU-T Rec. H.262 and ISO/IEC 13818-2 (MPEG 2 Video), Nov. 2003.

[9] ——, *Advanced video coding for generic audiovisual services*, ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC), 2003.

[10] G. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[11] K. Misra, A. Segall, M. Horowitz, S. Xu, A. Fuldseth, and M. Zhou, "An overview of tiles in hevc," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 6, pp. 969–977, Dec. 2013.

[12] C. C. Chi, M. Alvarez-Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, and T. Schierl, "Parallel scalability and efficiency of HEVC parallelization approaches," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1827–1838, Dec. 2012.

[13] V. Sze, M. Budagavi, and G. J. Sullivan, Eds., *High Efficiency Video Coding (HEVC): Algorithms and Architectures.* Springer International Publishing, Switzerland, 2014.

[14] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *IEEE Transactions on Image Processing*, vol. 4, no. 1, pp. 105–107, Jan. 1995.

[15] C.-D. Bei and R. Gray, "An improvement of the minimum distortion encoding algorithm for vector quantization," *IEEE Transactions on Communications*, vol. 33, no. 10, pp. 1132–1133, Oct 1985.

[16] X. Q. Gao, C. J. Duanmu, and C. R. Zou, "A multilevel successive elimination algorithm for block matching motion estimation," *IEEE Transactions on Image Processing*, vol. 9, no. 3, pp. 501–504, Mar. 2000.

[17] C.-K. Cheung and L.-M. Po, "Normalized partial distortion search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, no. 3, pp. 417–422, Apr 2000.

[18] Y.-L. Chan and W.-C. Siu, "Search strategy for partial distortion elimination in motion estimation," *Electronics Letters*, vol. 38, no. 23, pp. 1427–1428, Nov 2002.

[19] S. Tedmori and N. Al-Najdawi, "Hierarchical stochastic fast search motion estimation algorithm," *IET Computer Vision*, vol. 6, no. 1, pp. 21–28, Jan. 2012.

[20] H. Lim and T. G. Ahn, "An hierarchical motion estimation method using adaptive image down-sizing," in *2014 IEEE International Conference on Consumer Electronics (ICCE)*, Jan 2014, pp. 361–362.

[21] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion compensated interframe coding for video conferencing," in *Proc. of National Telecommunication Conference*, New Orleans, L. A., Nov./Dec. 1981, pp. C9.6.1–9.6.5.

[22] J. Jain and A. Jain, "Displacement measurement and its application in interframe image coding," *Communications, IEEE Transactions on*, vol. 29, no. 12, pp. 1799–1808, Dec. 1981.

[23] R. Li, B. Zeng, and M. L. Liou, “A new three-step search algorithm for block motion estimation,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 4, no. 4, pp. 438–442, Aug. 1994.

[24] D. Le Gall, “MPEG: A video compression standard for multimedia applications,” *Commun. ACM*, vol. 34, no. 4, pp. 46–58, apr 1991. [Online]. Available: http://doi.acm.org/10.1145/103085.103090

[25] L.-K. Liu and E. Feig, “A block-based gradient descent search algorithm for block motion estimation in video coding,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 4, pp. 419–422, Aug. 1996.

[26] S. Zhu and K.-K. Ma, “A new diamond search algorithm for fast block matching motion estimation,” in *Proceedings of ICICS, 1997 International Conference on Information, Communications and Signal Processing. Theme: Trends in Information Systems Engineering and Wireless Multimedia Communications (Cat.*, vol. 1, Sep 1997, pp. 292–296 vol.1.

[27] P. Hosur and K. Ma, “Motion vector field adaptive fast motion estimation,” in *Second International Conference on Information, Communications and Signal Processing (ICICS ’99)*, Singapore, Dec. 7–10, 1999, pp. 883–892.

[28] A. M. Tourapis, O. C. L. Au, and M. L. Liou, “Predictive motion vector field adaptive search technique (PMVFAST): enhancing block-based motion estimation,” in *Proc. SPIE*, vol. 4310, 2000, pp. 883–892. [Online]. Available: http://dx.doi.org/10.1117/12.411871

[29] *Optimization Model Version 3.0*, ISO/IEC JTC1/SC29/WG11 Coding of moving pictures and audio/N4344, 2001.

[30] A. M. Tourapis, O. C. Au, and M. L. Liou, "New results on zonal based motion estimation algorithms-advanced predictive diamond zonal search," in *ISCAS 2001. The 2001 IEEE International Symposium on Circuits and Systems (Cat. No.01CH37196)*, vol. 5, 2001, pp. 183–186 vol. 5.

[31] A. M. Tourapis, "Enhanced predictive zonal search for single and multiple frame motion estimation," *SPIE Visual Communications and Image Processing*, vol. 4671, pp. 1069–1079, Jan. 2002.

[32] N. Purnachand, L. N. Alves, and A. Navarro, "Improvements to TZ search motion estimation algorithm for multiview video coding," in *2012 19th International Conference on Systems, Signals and Image Processing (IWSSIP)*, April 2012, pp. 388–391.

[33] "HEVC reference software 16.3," [Online]. Available: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/.

[34] T. Tomczak, "Fast sign detection for RNS $(2^n - 1, 2^n, 2^n + 1)$," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 55, no. 6, pp. 1502–1511, July 2008.

[35] M. Xu, Z. Bian, and R. Yao, "Fast sign detection algorithm for the RNS moduli set $\{2^{n+1} - 1, 2^n - 1, 2^n\}$," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 2, pp. 379–383, Feb 2015.

[36] S. Kumar, C. H. Chang, and T. F. Tay, "New algorithm for signed integer comparison in $\{2^{n+k}, 2^n - 1, 2^n + 1, 2^{n\pm1} - 1\}$ and its efficient hardware implementation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. PP, no. 99, pp. 1–13, 2016.

[37] C. V. Niras and Y. Kong, "Fast sign-detection algorithm for residue number system moduli set $\{2^n - 1, 2^n, 2^{n+1} - 1\}$," *IET Computers Digital Techniques*, vol. 10, no. 2, pp. 54–58, 2016.

[38] R. Chaves and L. Sousa, "2n + 1, 2n+k, 2n - 1 : a new rns moduli set extension," in *Euromicro Symposium on Digital System Design, 2004. DSD 2004.*, Aug 2004, pp. 210–217.

[39] W. Wang, M. N. S. Swamy, and M. O. Ahmad, "Moduli selection in rns for efficient vlsi implementation," in *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, vol. 4, May 2003, pp. IV–512–IV–515 vol.4.

[40] E. Setiaarif and P. Siy, "A new moduli set selection technique to improve sign detection and number comparison in residue number system (rns)," in *NAFIPS 2005 - 2005 Annual Meeting of the North American Fuzzy Information Processing Society*, June 2005, pp. 766–768.

[41] N. I. Chervyakov, P. A. Lyakhov, D. I. Kalita, and K. S. Shulzhenko, "Effect of rns moduli set selection on digital filter performance for satellite communications," in *2015 International Siberian Conference on Control and Communications (SIBCON)*, May 2015, pp. 1–7.

[42] M. Abdallah and A. Skavantzos, "A systematic approach for selecting practical moduli sets for residue number systems," in *Proceedings of the Twenty-Seventh Southeastern Symposium on System Theory*, Mar 1995, pp. 445–449.

[43] V. S. Dimitrov, T. V. Cooklev, and B. D. Donevsky, "Generalized fermat-mersenne number theoretic transform," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 41, no. 2, pp. 133–139, Feb. 1994.

[44] P. V. A. Mohan, "Rns-to-binary converter for a new three-moduli set $\{2^{n+1} - 1, 2^n, 2^n - 1\}$," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 9, pp. 775–779, Sept 2007.

[45] A. Omondi and B. Premkumar, *Residue number systems: theory and implementation*. Imperial College Press, London, 2007.

[46] D. Younes and P. Steffan, "A comparative study on different moduli sets in residue number system," in *Computer Systems and Industrial Informatics (ICCSII), 2012 International Conference on*, Dec. 2012, pp. 1–6.

[47] R. Conway and J. Nelson, "Improved RNS FIR filter architectures," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 51, no. 1, pp. 26–28, Jan. 2004.

[48] J. C. Bajard, L. S. Didier, and T. Hilaire, "ρ-direct form transposed and residue number systems for filter implementations," in *2011 IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug 2011, pp. 1–4.

[49] N. S. Szabo and R. I. Tanka, *Residue Arithmetic and Its Applications to Computer Technology*. Mc Graw-Hill, New York, 1967.

[50] A. Safari, N. C. Vayalil, and Y. Kong, "VLSI architecture of multiplier-less DWT image processor," in *IEEE 2013 Tencon - Spring*, Apr. 2013, pp. 280–284.

[51] E. Vassalos, D. Bakalis, and H. T. Vergos, "RNS assisted image filtering and edge detection," in *2013 18th International Conference on Digital Signal Processing (DSP)*, July 2013, pp. 1–6.

[52] S. Asif, M. S. Hossain, and Y. Kong, “High-throughput multi-key elliptic curve cryptosystem based on residue number system,” *IET Computers & Digital Techniques*, 2017.

[53] R. B. Are and K. Rajan, “An rns based transform architecture for h.264/avc,” in *TENCON 2008 - 2008 IEEE Region 10 Conference*, Nov. 2008, pp. 1–6.

[54] T. Toivonen and J. Heikkila, “Video filtering with fermat number theoretic transforms using residue number system,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 1, pp. 92–101, Jan. 2006.

[55] P. Matutino and L. Sousa, “An RNS based specific processor for computing the minimum sum-of-absolute-differences,” in *Digital System Design Architectures, Methods and Tools, 2008. DSD ’08. 11th EUROMICRO Conference on*, Sept. 2008, pp. 768–775.

[56] M. Wein, *High Efficiency Video Coding (HEVC): Coding Tools and Specification.* Springer-Verlag Berlin Heidelberg, 2015.

[57] J. Ohm, G. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, “Comparison of the coding efficiency of video coding standards – including high efficiency video coding (HEVC),” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1669–1684, Dec. 2012.

[58] Z. Chen, J. Xu, Y. He, and J. Zheng, “Fast integer-pel and fractional-pel motion estimation for H.264/AVC,” *Journal of Visual Communication and Image Representation*, vol. 17, no. 2, pp. 264–290, 2006, introduction: Special Issue on emerging H.264/AVC video coding standard. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1047320305000787

[59] J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, “A novel unrestricted center-biased diamond search algorithm for block motion estimation,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 4, pp. 369–377, Aug. 1998.

[60] O. Ndili and T. Ogunfunmi, “Algorithm and architecture co-design of hardware-oriented, modified diamond search for fast motion estimation in H.264/AVC,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 9, pp. 1214–1227, Sept. 2011.

[61] A. C. Tsai, K. Bharanitharan, J. F. Wang, and K. I. Lee, “Effective search point reduction algorithm and its VLSI design for HDTV H.264/AVC variable block size motion estimation,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 7, pp. 981–988, July 2012.

[62] S. C. Hsia and P. Y. Hong, “Very large scale integration (VLSI) implementation of low-complexity variable block size motion estimation for H.264/AVC coding,” *IET Circuits, Devices Systems*, vol. 4, no. 5, pp. 414–424, Sept. 2010.

[63] S. Y. Jou, S. J. Chang, and T. S. Chang, “Fast motion estimation algorithm and design for real time QFHD high efficiency video coding,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 9, pp. 1533–1544, Sept. 2015.

[64] C. Y. Kao and Y. L. Lin, “A memory-efficient and highly parallel architecture for variable block size integer motion estimation in H.264/AVC,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 6, pp. 866–874, June 2010.

[65] J. Byun, Y. Jung, and J. Kim, "Design of integer motion estimator of HEVC for asymmetric motion-partitioning mode and 4K-UHD," *Electronics Letters*, vol. 49, no. 18, pp. 1142–1143, Aug. 2013.

[66] L. Shen, Z. Liu, X. Zhang, W. Zhao, and Z. Zhang, "An effective CU size decision method for HEVC encoders," *IEEE Transactions on Multimedia*, vol. 15, no. 2, pp. 465–470, Feb. 2013.

[67] J. Xiong, H. Li, F. Meng, Q. Wu, and K. N. Ngan, "Fast hevc inter cu decision based on latent sad estimation," *IEEE Transactions on Multimedia*, vol. 17, no. 12, pp. 2147–2159, Dec. 2015.

[68] P. K. Podder, M. Paul, and M. Murshed, "Efficient coding strategy for HEVC performance improvement by exploiting motion features," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, April 2015, pp. 1414–1418.

[69] L. Shen, Z. Zhang, and Z. Liu, "Adaptive inter-mode decision for HEVC jointly utilizing inter-level and spatiotemporal correlations," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 10, pp. 1709–1722, Oct. 2014.

[70] J. Zhang, B. Li, and H. Li, "An efficient fast mode decision method for inter prediction in HEVC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 8, pp. 1502–1515, Aug. 2016.

[71] A. Medhat, A. Shalaby, M. S. Sayed, M. Elsabrouty, and F. Mehdipour, "Adaptive low-complexity motion estimation algorithm for high efficiency video coding encoder," *IET Image Processing*, vol. 10, no. 6, pp. 438–447, 2016.

[72] S. Radicke, J. U. Hahn, Q. Wang, and C. Grecos, “Bi-predictive motion estimation for hevc on a graphics processing unit (gpu),” *IEEE Transactions on Consumer Electronics*, vol. 60, no. 4, pp. 728–736, Nov. 2014.

[73] F. Luo, S. Ma, J. Ma, H. Qi, L. Su, and W. Gao, “Multiple layer parallel motion estimation on GPU for high efficiency video coding (HEVC),” in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015, pp. 1122–1125.

[74] W. Xiao, B. Li, J. Xu, G. Shi, and F. Wu, “HEVC encoding optimization using multicore CPUs and GPUs,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 11, pp. 1830–1843, Nov. 2015.

[75] S. Radicke, J. U. Hahn, Q. Wang, and C. Grecos, “A parallel HEVC intra prediction algorithm for heterogeneous CPU + GPU platforms,” *IEEE Transactions on Broadcasting*, vol. 62, no. 1, pp. 103–119, Mar. 2016.

[76] N. Hu and E. H. Yang, “Fast motion estimation based on confidence interval,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 8, pp. 1310–1322, Aug. 2014.

[77] S. H. Yang, J. Z. Jiang, and H. J. Yang, “Fast motion estimation for hevc with directional search,” *Electronics Letters*, vol. 50, no. 9, pp. 673–675, Apr. 2014.

[78] I. Zupancic, S. G. Blasi, and E. Izquierdo, “Multiple early termination for fast HEVC coding of UHD content,” in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2015, pp. 1419–1423.

[79] G. Pastuszak and M. Trochimiuk, “Algorithm and architecture design of the motion estimation for the H.265/HEVC 4K-UHD encoder,” *Journal of Real-Time Image Processing*, vol. 12, no. 2, pp. 517–529, 2016. [Online]. Available: http://dx.doi.org/10.1007/s11554-015-0516-4

[80] N. C. Vayalil, A. Safari, and Y. Kong, "ASIC design in residue number system for calculating minimum sum of absolute differences," in *2015 Tenth International Conference on Computer Engineering Systems (ICCES)*, Dec 2015, pp. 129–132.

[81] V. N. Dinh, H. A. Phuong, D. V. Duc, P. T. K. Ha, P. V. Tien, and N. V. Thang, "High speed SAD architecture for variable block size motion estimation in HEVC encoder," in *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*, July 2016, pp. 195–198.

[82] P. Nalluri, L. N. Alves, and A. Navarro, "A novel SAD architecture for variable block size motion estimation in HEVC video coding," in *2013 International Symposium on System on Chip (SoC)*, Oct. 2013, pp. 1–4.

[83] T. P. K. C. D'huys, S. Momcilovic, F. Pratas, and L. Sousa, "Reconfigurable data flow engine for HEVC motion estimation," in *IEEE International Conference on Image Processing (ICIP)*, Aug. 2014.

[84] C. Chen, S. Chien, Y. Huang, T. Chen, T. Wang, and L. Chen, "Analysis and architecture design of variable block-size motion estimation for H.264/AVC," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 53, no. 3, pp. 578–593, Mar. 2006.

[85] H. Samet, *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, Massachusetts, 1990.

[86] I. K. Kim, S. Lee, M. S. Cheon, T. Lee, and J. Park, "Coding efficiency improvement of HEVC using asymmetric motion partitioning," in *Broadband Multimedia Systems and Broadcasting (BMSB), 2012 IEEE International Symposium on*, June 2012, pp. 1–4.

[87] E. Alcocer, R. Gutierrez, O. Lopez-Granado, and M. P. Malumbres, "Design and implementation of an efficient hardware integer motion estimator for an HEVC video encoder," *Journal of Real-Time Image Processing*, pp. 1–11, 2016. [Online]. Available: http://dx.doi.org/10.1007/s11554-016-0572-4

[88] L. F. Ding, W. Y. Chen, P. K. Tsung, T. D. Chuang, P. H. Hsiao, Y. H. Chen, H. K. Chiu, S. Y. Chien, and L. G. Chen, "A 212 M pixels 4096 × 2160p multiview video encoder chip for 3D/quad full HDTV applications," *Solid-State Circuits, IEEE Journal of*, vol. 45, no. 1, pp. 46–58, Jan. 2010.

[89] D. Zhou, J. Zhou, G. He, and S. Goto, "A 1.59 Gpixel/s motion estimation processor with -211 to +211 search range for UHDTV video encoder," *Solid-State Circuits, IEEE Journal of*, vol. 49, no. 4, pp. 827–837, Apr. 2014.

[90] H. Henkelmann and W. Anheier, "Implementation of sign detection in RNS using mixed radix representation," in *Electronics, Circuits and Systems, 1999. Proceedings of ICECS '99. The 6th IEEE International Conference on*, vol. 1, 1999, pp. 323–326.

[91] E. Al-Radadi and P. Siy, "RNS sign detector based on Chinese remainder theorem II (CRT II)," *Computers & Mathematics with Applications*, vol. 46, no. 1011, pp. 1559–1570, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S089812210390191X

[92] Y. Wang, "New Chinese remainder theorems," in *Signals, Systems & Computers, 1998. Conference Record of the Thirty-Second Asilomar Conference on*, vol. 1, Nov. 1998, pp. 165–171.

[93] T. V. Vu, "Efficient implementations of the Chinese remainder theorem for sign detection and residue decoding," *Computers, IEEE Transactions on*, vol. C-34, no. 7, pp. 646–651, July 1985.

[94] G. Alia and E. Martinelli, "Sign detection in residue arithmetic units," *Journal of Systems Architecture*, vol. 45, no. 3, pp. 251–258, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762197000854

[95] G. Pirlo and D. Impedovo, "A new class of monotone functions of the residue number system," *International journal of mathematical models and methods in applied sciences*, vol. 7, no. 1, pp. 802–805, Oct 2013.

[96] C. H. Chang and S. Kumar, "Area-efficient and fast sign detection for four-moduli set RNS $\{2^n-1, 2^n, 2^n+1, 2^{2n}+1\}$," in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, June 2014, pp. 1540–1543.

[97] L. Sousa and P. Martins, "Efficient sign identification engines for integers represented in RNS extended 3-moduli set $\{2^n-1, 2^{n+k}, 2^n+1\}$," *Electronics Letters*, vol. 50, no. 16, pp. 1138–1139, July 2014.

[98] N. Szabo, "Sign detection in nonredundant residue systems," *Electronic Computers, IRE Transactions on*, vol. EC-11, no. 4, pp. 494–500, Aug. 1962.

[99] R. Zimmermann, "Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication," in *Computer Arithmetic, 1999. Proceedings. 14th IEEE Symposium on*, 1999, pp. 158–167.

[100] S. Piestrak, "Design of residue generators and multioperand modular adders using carry-save adders," *Computers, IEEE Transactions on*, vol. 43, no. 1, pp. 68–77, Jan. 1994.

[101] R. Jackson and S. Talwar, "High speed binary addition," in *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Eighth Asilomar Conference on*, vol. 2, Nov. 2004, pp. 1350–1353.

[102] P. M. Kogge and H. S. Stone, “A parallel algorithm for the efficient solution of a general class of recurrence equations,” *Computers, IEEE Transactions on*, vol. C-22, no. 8, pp. 786–793, Aug. 1973.

[103] A. Tyagi, “A reduced-area scheme for carry-select adders,” *Computers, IEEE Transactions on*, vol. 42, no. 10, pp. 1163–1170, Oct 1993.

[104] F. Gurkayna, Y. Leblebicit, L. Chaouati, and P. McGuinness, “Higher radix Kogge-Stone parallel prefix adder architectures,” in *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, vol. 5, 2000, pp. 609–612.

[105] L. Sousa, “Efficient method for magnitude comparison in RNS based on two pairs of conjugate moduli,” in *Computer Arithmetic, 2007. ARITH ’07. 18th IEEE Symposium on*, June 2007, pp. 240–250.

[106] S. Bi and W. Gross, “The mixed-radix Chinese remainder theorem and its applications to residue comparison,” *Computers, IEEE Transactions on*, vol. 57, no. 12, pp. 1624–1632, Dec. 2008.

[107] Y. Wang, X. Song, and M. Aboulhamid, “A new algorithm for RNS magnitude comparison based on new Chinese remainder theorem II,” in *VLSI, 1999. Proceedings. Ninth Great Lakes Symposium on*, Mar. 1999, pp. 362–365.

[108] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi, “Video coding with H.264/AVC: tools, performance, and complexity,” *Circuits and Systems Magazine, IEEE*, vol. 4, no. 1, pp. 7–28, First Quarter 2004.

[109] L. Kalampoukas, D. Nikolos, C. Efstathiou, H. Vergos, and J. Kalamatianos, "High-speed parallel-prefix modulo $2^n - 1$ adders," *Computers, IEEE Transactions on*, vol. 49, no. 7, pp. 673–680, Jul. 2000.

[110] D. Wang, X. Cui, and X. Wang, "Optimized design of parallel prefix ling adder," in *Electronics, Communications and Control (ICECC), 2011 International Conference on*, Sept. 2011, pp. 941–944.

[111] G. Dimitrakopoulos, D. Nikolos, H. Vergos, D. Nikolos, and C. Efstathiou, "New architectures for modulo $2^n - 1$ adders," in *Electronics, Circuits and Systems, 2005. ICECS 2005. 12th IEEE International Conference on*, Dec. 2005, pp. 1–4.

[112] Cisco visual networking index: Forecast and methodology, 2014–2019. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html

[113] L. Shen, Z. Zhang, and Z. Liu, "Adaptive inter-mode decision for HEVC jointly utilizing inter-level and spatiotemporal correlations," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 10, pp. 1709–1722, Oct 2014.

[114] Y. Kong, S. Asif, and M. A. U. Khan, "Modular multiplication using the core function in the residue number system," *Applicable Algebra in Eng., Commun. and Computing*, vol. 27, no. 1, pp. 1–16, 2016.

[115] FFmpeg software version git-2014-11-14-b186b71. [Online]. Available: https://www.ffmpeg.org/

[116] M. Sinangil, V. Sze, M. Zhou, and A. Chandrakasan, "Cost and coding efficient motion estimation design considerations for high efficiency video coding (hevc) standard," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 7, no. 6, pp. 1017–1028, Dec. 2013.

[117] D. Zhou, J. Zhou, G. He, and S. Goto, "A 1.59 Gpixel/s motion estimation processor with - 211 to +211 search range for UHDTV video encoder," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 4, pp. 827–837, April 2014.

[118] M. T. Pourazad, C. Doutre, M. Azimi, and P. Nasiopoulos, "HEVC: The new gold standard for video compression: How does HEVC compare with H.264/AVC?" *IEEE Consumer Electronics Magazine*, vol. 1, no. 3, pp. 36–46, July 2012.

[119] Z. T. Liao and C. A. Shen, "A novel search window selection scheme for the motion estimation of hevc systems," in *2015 International SoC Design Conference (ISOCC)*, Nov. 2015, pp. 267–268.

[120] G. Pastuszak and M. Jakubowski, "Adaptive computationally scalable motion estimation for the hardware H.264/AVC encoder," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 23, no. 5, pp. 802–812, May 2013.

[121] JVT of ISO/IEC MPEG, ITU-T VCEG, *MVC software Reference Manual-JMVC 8.2*, Mar. 2010.

[122] H. L. Garner, "The residue number system," *IRE Transactions on Electronic Computers*, vol. EC-8, no. 2, pp. 140–147, June 1959.

[123] G. Bjøntegaard, *Calculation of average PSNR differences between RD-curves*, ITU-T SG16 Document VCEG-M33, Joint Collaborative Team on Video Coding (JCTVC), Apr. 2001.

[124] "Cisco visual networking index: Forecast and methodology, 2015–2020," [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html.

[125] N. Purnachand, L. N. Alves, and A. Navarro, “Fast motion estimation algorithm for HEVC,” in *2012 IEEE Second International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, Sept. 2012, pp. 34–37.

[126] X. Li, R. Wang, W. Wang, Z. Wang, and S. Dong, “Fast motion estimation methods for HEVC,” in *2014 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting*, June 2014, pp. 1–4.

[127] H. Kibeya, F. Belghith, H. Loukil, M. A. B. Ayed, and N. Masmoudi, “TZ search pattern search improvement for HEVC motion estimation modules,” in *Advanced Technologies for Signal and Image Processing (ATSIP), 2014 1st International Conference on*, March 2014, pp. 95–99.

[128] J. H. Jeong, N. Parmar, and M. H. Sunwoo, “Enhanced test zone search algorithm with rotating pentagon search,” in *2015 International SoC Design Conference (ISOCC)*, Nov 2015, pp. 275–276.

[129] T. Nguyen, P. Nguyen, P. Nguyen, and C. Dinh, “A novel search pattern for motion estimation in high efficiency video coding,” in *2016 International Conference on Computer Communication and Informatics (ICCCI)*, Jan 2016, pp. 1–6.

[130] G. Paoloni, “How to benchmark code execution times on Intel® IA-32 and IA-64 instruction set architectures,” [Online]. Available: http://www.intel.com/content/www/us/en/embedded/training/ia-32-ia-64-benchmark-code-execution-paper.html, Intel, Sept. 2010.

[131] P. K. Meher, S. Y. Park, B. K. Mohanty, K. S. Lim, and C. Yeo, “Efficient integer DCT architectures for HEVC,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 1, pp. 168–178, Jan. 2014.

[132] M. Masera, M. Martina, and G. Masera, "Adaptive approximated DCT architectures for HEVC," *IEEE Transactions on Circuits and Systems for Video Technology*, no. 99, pp. 1–1, 2016.

[133] M. Jridi and P. Meher, "A scalable approximate DCT architectures for efficient HEVC compliant video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, no. 99, pp. 1–1, 2016.

[134] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE Transactions on Computers*, vol. C-23, no. 1, pp. 90–93, Jan 1974.

[135] N. C. Vayalil, A. Safari, and Y. Kong, "Overlapped block-processing VLSI architecture for separable 2D filters," in *Electronics, Communications and Networks IV*, Jun 2015, pp. 1355–1358.

[136] K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications.* Academic Press, Boston, 1990.

[137] M. Budagavi, A. Fuldseth, G. Bjntegaard, V. Sze, and M. Sadafale, "Core transform design in the high efficiency video coding (HEVC) standard," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 6, pp. 1029–1041, Dec 2013.

[138] K. Choi and E. S. Jang, "Early TU decision method for fast video encoding in high efficiency video coding," *Electronics Letters*, vol. 48, no. 12, pp. 689–691, June 2012.

[139] C. C. Wang, Y. C. Liao, J. W. Wang, and C. W. Tung, "An effective TU size decision method for fast HEVC encoders," in *Computer, Consumer and Control (IS3C), 2014 International Symposium on*, June 2014, pp. 1195–1198.

[140] J. Su, K. Nitta, M. Ikeda, and A. Shimizu, "Residue role assignment based transform partition predetermination on HEVC," in *2013 IEEE International Conference on Image Processing*, Sept 2013, pp. 2019–2023.

[141] J. Kang, H. Choi, and J. G. Kim, "Fast transform unit decision for HEVC," in *Image and Signal Processing (CISP), 2013 6th International Congress on*, vol. 01, Dec 2013, pp. 26–30.

[142] Z. Pan, J. Lei, Y. Zhang, W. Yan, and S. Kwong, "Fast transform unit depth decision based on quantized coefficients for hevc," in *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on*, Oct 2015, pp. 1127–1132.

[143] J. T. Fang, Y. C. Tsai, J. X. Lee, and P. S. Yu, "Computation reduction in transform unit of high efficiency video coding based on zero-coefficients," in *2016 International Symposium on Computer, Consumer and Control (IS3C)*, July 2016, pp. 797–800.

[144] V. Sze, M. Budagavi, and G. J. Sullivan, Eds., *High Efficiency Video Coding (HEVC): Algorithms and Architectures.* Springer International Publishing, Switzerland, 2014, ch. 11.

[145] S. C. Hsia and L. S. Chen, "Parallel very large-scale integration chip implementation of optimal fractional motion estimation," *IET Circuits, Devices Systems*, vol. 8, no. 6, pp. 499–508, 2014.

[146] D. Kang, Y. Kang, and Y. Hong, "VLSI implementation of fractional motion estimation interpolation for high efficiency video coding," *Electronics Letters*, vol. 51, no. 15, pp. 1163–1165, 2015.

[147] G. He, D. Zhou, Y. Li, Z. Chen, T. Zhang, and S. Goto, "High-throughput power-efficient VLSI architecture of fractional motion estimation for ultra-HD HEVC video

encoding," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 12, pp. 3138–3142, Dec 2015.