POLITECNICO DI TORINO

Master degree in Electronic Engineering

Master Course of Micro & Nano Systems

Master's Degree Thesis

Study of Approximate Architectures for the Motion Estimation

Supervisor: Prof. Martina Maurizio

Candidate: Paltrinieri Alberto

September 2018

Summary

The motion estimation is the most efficient technique to model the motion inside a video sequence and, at the same time, it is highly power consuming and heavily time demanding. The generation of motion vectors and the frame prediction inside the modern multimedia systems represent the two higher costs in terms of performance that a standard must support. Power/energy efficient designs try to reduce the power requirements of the video compression. In this work of thesis, focused on the video encoding process inside the standard HEVC (the most recent video standard), a particular approach is used to overcome the motion estimation requirements: the approximate computing. The best match search of the motion estimation is based on a mathematical function to match couples of frames, very common is the use of the Sum of Absolute Difference (SAD), which can be approximated through the implementation of approximate adders.

Every encoder system is composed of an accelerator and a control unit; the accelerator contains the criteria of the best match and, hence, it can be modified for this purpose. The introduction of inaccurate adders inside the architecture provides faster but lower precise systems, which translates into lower resolution but lower power demanding result. The inaccurate adders were applied to the SAD in order to improve the performance, reducing the power/energy consumption; such implementations are suitable for error-tolerant applications.

The SAD accelerator, on which the approximate adders' substitution is based, is a pre-existent fast, flexible and dynamically reconfigurable hardware accelerator, which follows all the specifications of the HEVC standards, and is able to deal with every standard Prediction Block size up to 64x64, implementing a local on-chip memory. The implementations and test of the architectures has been possible through a precise selection of approximate adders and substitutions' positions inside the system. The selected inaccurate adders are: Almost Correct Adder (ACA), Accuracy Configurable Approximate Adder (ACAA), Error-Tolerant Adder I (ETAI), Lower-OR Part Adder (LOA) and Speculative Carry Select Adder (SCSA).

Some appreciable parameter resulting from the simulations on the proposed work are: in the worst case (64x64 SADs), the efficiency of the approximate configurations, calculated as the ratio between power saving and error, results 25% improved on average with respect to the accurate ones, the power reduction reaches a maximum of 8.61 mW and the frequency reachable exceeds the 2 GHz, with the only penalties of an increase on the number of required gates and a reduction of the encoded video quality.

At the beginning, the thesis work fills the basic theory about the video compression, the standard HEVC, the Motion Estimation (ME) and Compensation (MC). The middle part describes how the accelerator works, its operations and the introduction of the approximate adders inside its architecture. After, a double validation step is performed, to calculate the error introduced by the approximation and to ensure the correct working condition of the structure. In the end, the synthesis results are exposed, discussed and critically studied. The final analysis reveals a satisfying reduction of the power

consumption of the system, leading to a lighter motion estimation and best match procedure, which leaves the window open for future implementation and further verification.

Acknowledgements

First, I would like to extend my thanks to my supervisor Maurizio Martina, professor of Dipartimento di Elettronica e Telecomunicazioni (DET) at Politecnico di Torino. He always supported my work and helped me to clear up any doubt with comprehensible and exhaustive explanations. Prof. Martina was always available and made me feel comfortable for the duration of my work, driving me in the right way or proposing me new possibilities to proceed.

Second, I have to thanks the VLSI lab, where I found a workstation and exploit some advice from doctoral candidates or other candidates.

Third, I would like to thank the people who accompanied me on this trip and a special commendation is dedicated to my girlfriend, always patient and affectionate, close to me during difficult moments and model to emulate.

Fourth and last, I have to express my profound gratitude to my family. My family supported me these two years spent in the Politecnico and in my previous Bachelor degree, providing me with continuous encouragement throughout my years of study and economical support for conclude the school career. I would never have achieved these goals without them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, a large scale of applications exploits video recording and playback, in fields such as video streaming, TV, surveillance, video monitoring, mobile phones video reproduction and shooting, but also computer training, video conferences/calls and education. All these heterogeneous areas require different resolutions, and so different bandwidth. For instance, the reproduction of YouTube video on a mobile phone requires a lower resolution than a 4K TV monitor, rather than a streaming video on Internet requires an higher resolution than a system of video security. The higher is the demanded resolution of the specific application, the larger is the memory capacitance required to store or the larger is the memory bandwidth required to transmit. All the application fields concerning videos requires a specific video compression process and a corresponding video decompression; to introduce the next topics related to video compression, in figure 1.1, a brief and simple scheme is represented in order to give a general idea about the main steps of a video compression and reconstruction.

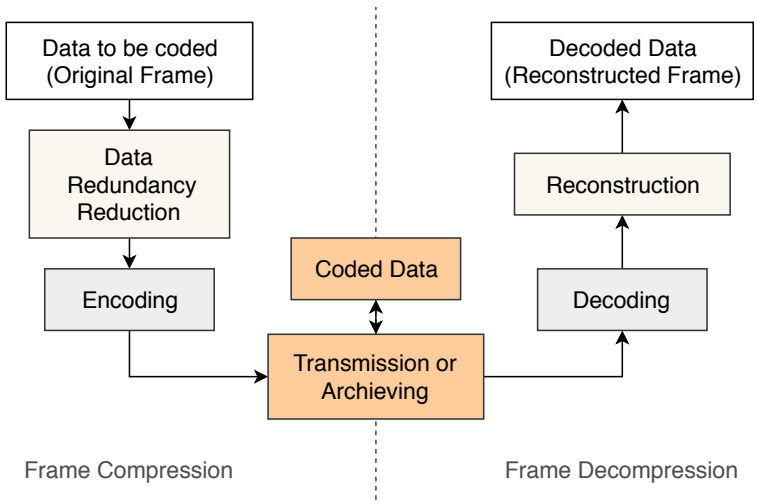


Figure 1.1: Summary scheme of a compression process.

Videoclips are composed of a series of frames (still images) and the video compression techniques take advantage of the redundancies and similarities between consecutive frames to perform data compacting, reducing its dimension and facilitating the storage or transmission. Historically, the first compression algorithm was an image compression method called "Still-Image Compression", whose working principle was to ignore similarities between neighboring frames and compress each frame

independently. An example of still image compression is the JPEG. Modern video compression algorithms exploit the correlation between consecutive frames through motion compensation and motion estimation, advance techniques of compression between matching frames, whose match is calculated through mathematical functions, like the Sum of Absolute Difference (SAD). Motion compensation and motion estimation are optimized processes which include some advance redundancy elimination method and intra/inter-frame coding. For instance, two example of modern compression algorithms are H.264/AVC and HEVC, latest video coding standards. Intra- and infra-frames definitions are briefly hereafter exposed:

- The Inter-frame compression is a powerful compression mechanism which recombines data from previous frames (or subsequent, it depend if its bidirectional or not), called reference frames, to describe the actual or current frame.
- The Intra-frame is a lossy coding technique which exploit data within the current frame and is more used in still image compression. [1]

Video compression is carried out with algorithms called "codecs". They allow to manipulate videos, with the aim of reduce the bandwidth or storage, maintaining constant or maximizing the video quality; codecs exploit both spatial and temporal redundancy. In general, data compression algorithms have the aim to shrink number of bits required to represent a video, as well as an image or a music piece, inside a precise standard. Data compression is the art of representing information in a compact form. [1]

Dealing with compression video is not an easy task and, before entering in details, is better to distinguish between compression techniques (or similarly compression algorithms), because, rigorously, the compression techniques implies a compression process and a reconstruction precess:

1. **Compression Technique:** The compression consists in taking an input χ and generating a representation χ_c; this process requires a small number bits.
2. **Reconstruction Technique:** The reconstruction operates on the compressed χ_c representation in order to generate a reconstructed representation φ.

In the general convention, compression and reconstruction algorithms refer together to the compression algorithm, without further distinction and for sake of simplicity. Video, or more in general, data compression can be divided into two classes: lossless compression algorithms and lossy compression algorithms. In the first, φ is identical to χ; in the latter, φ is different from χ but this method provides higher compression rate than lossless. Lossless compression is suitable for exact reconstruction of the initial sequence, while lossy algorithm is suitable for error-tolerant applications. [1]

The data compression development of an algorithm is based on modeling and coding. The modeling phase is used to extract from the data the informations about any redundancy and to model it. The second phase is a description of the model, coding in binary information how data and the model differ. To clarify these concepts, an example can be considered: given a sequence of numbers, the task is to compress it to simplify the possible transmission or storage. The sequence is:

$$\{x_1, x_2, x_3, ..., x_n\} = \{9, 11, 11, 11, 14, 13, 15, 17, 16, 17, 20, 21\}$$

The binary representation of these numbers requires 5 bit each, but it's possible to reduce this number observing the linear relation between position of the numbers and the value of the numbers. Plotting this relation, a line almost straight is obtained and, hence, an equation can be written to modeling the set of numbers. The existence of a law represents the presence of redundancy in the system. Hence, if

in the structure exists redundancy, there is the possibility to predict the value of each element inside the sequence of number and, then, encode the residual [1]. This is example is a very linear and simple case, made with numbers, but the same reasoning in possible for a video sequence.

All these ideas, exposed in this introduction, will be analyzed during the thesis work and deepened. The main aspects, such as motion estimation and compensation, intra- and inter-frame definitions, the encoder structure and all the theory behind the video compression will be deeply described in the initial part of the thesis. In the second experimental part, the core of this work is carried out: the implementation of approximate adders inside the architecture of the encoder. This work of thesis is based on a pre-designed hardware accelerator, which is the fulcrum of the encoder, extracted from the work of P. Selvo in ref.[2]. An encoder is composed of a SAD accelerator unit (the SAD is the Sum of Absolute Difference, a mathematical) and a Controller unit. The accelerator is made of Datapath and Memory unit, while the controller manages the signals and the interaction with the accelerator.

The SAD accelerator's architecture has been employed and, inside the adders of the Sum of Absolute Difference (SAD), a selection of approximate adders has been implemented. The introduction of an approximation in the computation of the motion estimation, and in particular in the SAD computation, allows to save in terms of power, energy, delay and area, with a concomitantly decrease of the quality, resolution and accuracy of the results. The SAD accelerator is an application specific integrated circuit (ASIC) ables to perform in hardware the sum of absolute difference function for all the possible block sizes, using the same hardware blocks and restraining the area and leakage power [2]. Its task is to speed-up the encoder execution, being flexible and dynamically reconfigurable. The higher is the speed, the larger will be the power consumption; to avoid an uncontrolled increase of the dissipations, an approximation in the decoding phase could be a valid solution. The aim of this work is, in fact, to verify how the system reacts to the replacement of accurate adders with approximate ones. In other words, the presented work deals with an energy-efficient variable block-size motion estimation hardware design and the original part of the work is the introduction inside the architecture of approximate adders to obtain an improvement of the performance, in particular in terms of power saving, without affecting the encoding efficiency. The starting point is the best result of the previous work of ref.[2]: an optimized PDE-based SAD accelerator, standardized to the HEVC. PDE, which stands for partial distortion elimination, is a technique discussed at the end of chapter 4.

A summary of the topics contained in this work is the following:

1. An initial theoretical discussion is carried out, immediately after this introduction, regarding the video signal representation, the simplest encoder structure and a brief history of the video compression technologies.

2. The second chapter discusses the HEVC standard, its characteristics and its improved encoder structure.

3. The third chapter is related to the motion estimation and compensation description. To better understand two examples are exposed, and the second addresses to the usage of the filter for the motion estimation.

4. The forth contains the definition of the whole architecture of the SAD accelerator, part by part, explaining in detail the control circuit and the memory unit.

5. In the fifth chapter the concept of approximate computing is investigated, followed by the definitions of the selection of the approximate adders and the selection of the substitutions.

6. The sixth is entirely dedicated to the error verification of the approximation inside the SAD accelerator, which consists in the error evaluation calculated on the mean error mathematically

introduced by the approximation and the mismatch inside the motion vectors between accurate and inaccurate results.

7. The seventh chapter explains the Modelsim implementation and simulations done to validate the architectures, accurate and approximated.

8. The eight chapter consists in the synthesis of the system, with a critical frequency, area, gate number and power evaluation analysis of the different combinations, both for the PDE-included system and the particular cases.

9. The penultimate chapter of the thesis is a final critical analysis of the obtained results, based on efficiency computation, both for the PDE-included system and the particular cases.

10. Finally, the last chapter is nothing but the conclusion, where some future works and implementations are proposed.

1.1 Video Signal Representation

The video signal representation has undergone numerous changes throughout history. A brief summary of the three main historical steps regarding video signal elaborations are illustrated.

1. **B&W:** Black and white (B&W) televisions required one unique video signal. The structure of the cathode tube derives directly from the cold cathode diode, in turn derived from the Crookes tube, to which is added a screen coated with fluorescent material, also called Braun tube.

   The cathode is a small metal element heated up (till the incandescence) that emits electrons by thermionic effect. Inside the cathode tube, where the vacuum is created, these electrons are directed in a bundle (cathode rays) by means of a high difference in electrical potential between the cathode and the anode, with the help of other electric fields or magnets appropriately arranged to accurately focus the beam. The beam (also known as an electronic brush) is deflected by the action of magnetic fields (Lorentz force, magnetic deflection) or electric fields (electrostatic deflection) so as to get to hit any point on the inner surface of the screen, the anode. The overall structure is represented in figure 1.2.

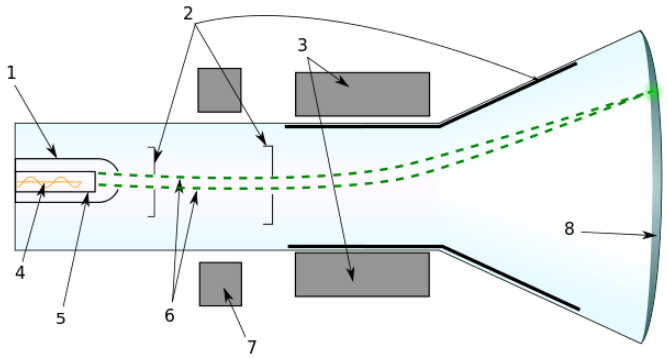
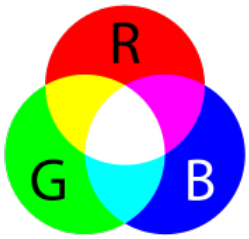


Figure 1.2: Schematic section of a monochromatic cathode ray tube: 1) Grid control 2) Anode 3) Deflector coil 4) Heater 5) Cathode 6) Electron beam 7) Focusing coil 8) Fluorescent screen

   This surface is coated with fluorescent material (typically phosphors) which, when energized by the electrons, emits light. The cathode tube has a characteristic response curve of the triode, which leads to a non-linear relationship between the electronic current and the intensity of the

light emitted, called the gamma function. In the early televisions this was positive because it had the effect of compressing the contrast (reducing the risk of saturation of the lighter or darker parts), but in some computer applications where the color rendering must be linear, as in desktop publishing, it must be a gamma correction applied. [15]

2. **RGB:** RGB is a model of colors whose specifications were described in 1936 by the CIE (Commission Internationale de l'clairage).



It is an additive type color model: the colors are defined as the sum of the three colors Red (Red), Green (Green) and Blue (Blue), from this the acronym RGB. Other 5 remarkable colors of this type of model are Yellow (Red + Green), Magenta (Red + Blue) and Cyan (Green + Blue), besides the sum of the three colors constitutes White, and their total absence is Black, as represented by the image on the left.

Due to its characteristics, it is a particularly suitable model in the representation and visualization of images in electronic devices. In fact, the most of the devices normally uses combinations of Red, Green and Blue to display the pixels of an image, but this also means that the same is particularly dependent on the device itself: the same image may be displayed in different way, if displayed on two different devices, as the materials used to make the screens vary depending on the manufacturer. Furthermore, differences can be noticed over time even in the same device, due to the natural deterioration of the same device.

Almost all types of screens use the RGB model to display the images, mixing pixels of the 3 colors mentioned above and adjusting the respective brightness to show the desired color. For example cathode ray tube(CRT), liquid crystal display (LCD), plasma display, or organic light emitting diode (OLED) monitors use the RGB model to generate images.

Pixels on the screen are constituted by three small and very close but RGB signal sources. The light sources are close but still separated, in an imperceptible way for human eye.

3. **YUV:** In a colored television (like RGB TVs), three kinds of phosphors are coated on the screen, respectively phosphor for the red color, green color and blue color, excited by three different electron beams. To control the three beams, three separated signals are needed; the bandwidth increase of 3 times with respect to the single signal but leads to backward compatibility problems with B&W television. The alternative has been to create a composite signal, based on the RGB video signal: the "YUV". YUV is made of 3 components, the luminance signal and two chrominance signals. The former (Y) is exactly the black-and-white signal (backward compatible with B&W TVs), while the two chrominance (C_b and C_r) can be computed directly from the luminance component.

   One could say that YUV was invented to fill the need of color TV in a B&W infrastructure, or, rather, the need of a video signal compatible with B&W signal but with color annexed. Moreover, the luma (or luminance) signal was already present, because it corresponds to the the B&W component; the two chrominance signals were added afterwards.

   The YUV signal can be calculated from the RGB signal, through the following expressions [2]:

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{1.1}$$

$$C_b = B - Y \tag{1.2}$$

$$C_r = R - Y \tag{1.3}$$

Colored TVs inverts the aforementioned expressions 1.1, 1.2 and 1.3 in order to obtain the three R, G and B signals. Nowadays, video representation results from the transition of the analog color television to the digital domain, so, because of this transition to the digital world, the luminance and the two chrominance need a sampling process to be saved in digital formats. [2]

Another advantage of YUV is the compression, and the consequent considerable saving of bandwidth, which is obtained by discarding some color information to which the human eye is not very sensitive. In fact, much of the perceived detail is given by the information on the light levels present in the luminance signal. As a consequence, the signals U and V can be compressed substantially. Generally, the chrominance occupies a band considerably lower than the luminance band. This feature has been maintained in all analog systems and also in most digital ones, where the chrominance sub-sampling is used, reducing the resolution of color information. Two examples of sampling scheme are 4:2:2 or 4:2:0. The imposition of the 4:x:x format was realized in the '70s. In the 1970s, CCIR (International Radio Consultative Committee) recommended a worldwide sampling standard, the so-called "CCIR recommendation 601-2". This standard states that the sampling rates of the video signals must be all multiple of a frequency equal to 3.725 MHz. After taking one sample per pixel, a rectangular array of samples is created and a picture is formed. With the CCIR recommendation, each Y, U and V signal can be sampled with a frequency multiple of 3.725 MHz, up to four times (14.9 MHz upper limit). This rule is consistent with the compression characteristic of the YUV format.

All the video sequences, that will be tested in the elaboration of this thesis work, follows the 4:2:0 format. The 4:2:0 sampling rate implies that the two chrominance C_b and C_r components are sampled at a frequency halved with respect to the luma component. Once the samples are converted into digital domain, the three components are called Y, U and V. First, the analog Y samples are normalized to the interval [0,1], Y_s, while the chrominance signals are normalized to the interval $[-\frac{1}{2}, \frac{1}{2}]$, C_{rs} and C_{bs}. Then, the normalized values are converted to 8 bits numbers, according to the following expressions [1]:

$$Y = 219 \cdot Y_s + 16 \tag{1.4}$$

$$U = 224 \cdot C_{bs} + 128 \tag{1.5}$$

$$V = 224 \cdot C_{rs} + 128 \tag{1.6}$$

Hence, the Y component assumes value between 16 and 235 while the U and V ones between 16 and 240.

1.2 Predictive (P), Intra Predictive (I), Bidirectionally Predictive (B) Coded Frames

The compression is possible because, instead of store or transmit every pixel of each frame, the current frame is sent with a motion vector, which defines the movement vector between two frames. In this way, the receiver is able to reconstruct the compressed images starting from the current frame. To be precise, the motion vector is not a univocal computation frame-to-frame but blocks or windows of samples (group of pixels) are searched in previous or next frames of the same video sequence and the movement in between them is registered into a motion vector.

The sequence of frames, sent or transmitted, cannot be formed by just one current frame referred to all the previous ones. The decode process cannot always start from the first frame, its needed the

periodical insertion of frames which are coded with no references to others [2]. These type of frame are named intra-predictive pictures, or briefly I frames. This type of frame allows the viewer to watch a video not necessarily from the first frame but from that point forward, skipping all the previous pictures. As can be easily guessed, the first frame must be an I frame.

In addition to the I frame, there are two other types of frames: the P (predictive coded) and B (bidirectionally predictive coded) frames. The P frames are encoded employing motion-compensation prediction from previous coded pictures of type I or P, while the B frames are coded in the same manner but reference both previous and subsequent frames [2]. The bidirectional predictive frames increase the complexity of the system but ensure an higher compression efficiency with respect to P ones. Video sequences are organized into groups of pictures (GOP). The structure of a GOP is illustrated in figure 1.3.

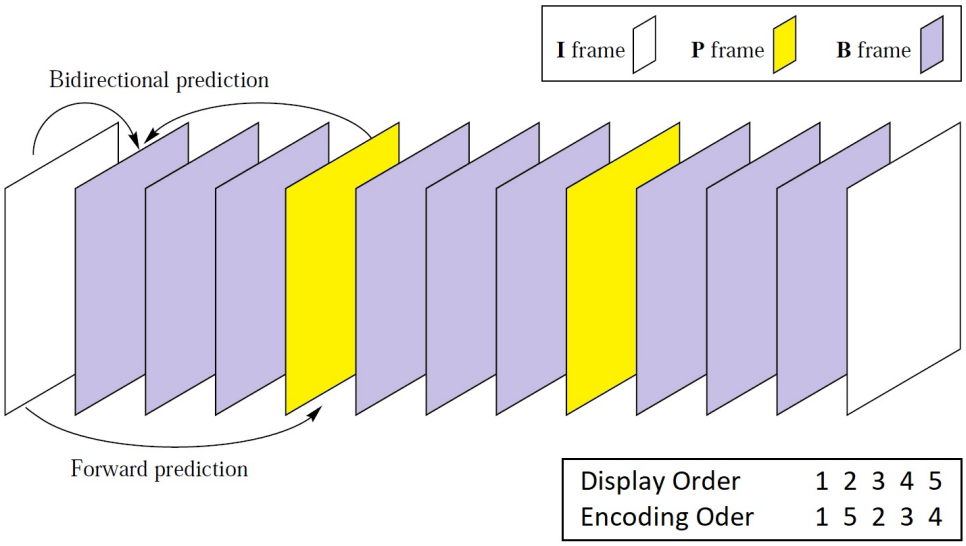


Figure 1.3: Example of a GOP structure. Highlighted immediately under the figure there is the difference in display and encoding (bit-stream) order of the frames. [2]

GOP must have at least one I frame (the first) and may contain several P and B frames. The GOP structure and the interval between two I frames can be defined to the encoder since it is better to adapt them to the kind of video application. The first frame to be coded is the I frame and it has no reference to other frames; the second frame in encoding order is the fifth one which uses prediction from the first picture. Then, the encoder processes the frame two by employing motion compensated prediction both from frame one and frame five since it is B type. Frame three and four are predicted in the same way using as references frame one and five. The encoder architecture is deeply analyzed in section 1.6. The decoder uses the same encoding order because the reference frames are needed before the frame that points them [2]. For instance, a quite high number of intra-prediction frames are required in TV broadcasting because the tuning of a televiewer cannot be always at the beginning of a TV program but suddenly happens. On the opposite side, for example, a lower number of I frames is required in a video conference to avoid line delays and not burden the communication, because it would result in a loss of efficiency.

1.3 Lossless and Lossy Data Compression

There are two types of data compressions: lossless and lossy data compression. In the first case, as suggested by its name, the lossless technique involves no loss of information after compression. In the latter, the lossy technique involves an unrecoverable loss of data/information after the compression, gaining in terms of efficiency. The two following subsections give more informations about these two compression processes.

1.3.1 Losseless Compression

Lossless compression techniques does not cause the loss of any data. The lossless compression process allows to recover exactly the original data from the compressed informations. An example of application is the text compression. The text transmitted and, after, reconstructed needs to be mandatorily identical to the original text, because even a different letter can result in a different statement. The typical example, shown in the articles, is the sentences "do not send money", which can be incorrectly been reconstructed as "do now send money." [1] The tolerance in such applications, which employ the lossless compression technique, is null, meaning that there must be no difference between original and reconstructed data.

The lack of error between reconstructed and compressed data doesn't imply that the compression doesn't exploit the redundancy elimination. The efficiency of the process is surely limited with respect to the lossy compression but one of the advantage is that the process reversible. Redundancy elimination is possible because real-world data exhibits statistical redundancy.

1.3.2 Lossy Compression

Lossy compression process causes a loss of data. The result coming from the introduction of an error in the compression technique produces higher compression ratios. Of course, the lossy techniques must be adopted by error-tolerant applications, where a difference between compressed data transmitted or stored and reconstructed data is allowed. An example of application field is in the transmission of a telephone conversation, because a human ear has a low sensitivity and tolerates loss of quality in the signal sent and reconstructed. In a similar way, also the human eye can tolerate a loss of quality in a video: the fact that the reconstructed frames are slightly different from the original sequence is not perceived as long as the differences do not result in annoying defects, allowing the compression [1]. An example of artifact is the change in color, which is difficult to perceive, while an artifact, to which the human eye is sensible, is the change in brightness.

The improved efficiency is the main target of the video compression. Hence generally, video compression employs lossy compression techniques. The lossy compression tries to achieve the best trade-off between quality and cost of compression and decompression. The drop of nonessential details is the key point to reduce the size of a video sequence because the attempt is to deceive the eye. As already cited, the example of the color and brightness is perfect in this instance: the compression cannot emphasize on the human eye's sensitivity to the variations in luminance, but can act the colors' variation. This is one the mechanism exploited by the JPEG standard.

1.4 Motion Estimation (ME)

In most cases, a video sequence consists of a succession of images (frames) very similar to each other, the differences are usually due to translations in their content. These temporal redundancies are exploited by the movements estimation techniques (Motion Estimation), which calculate the movements (Motion Vectors) verified in the contents of a scene, between a frame and its next/previous in order to

minimize their difference (Motion Compensation). The motion vectors can represent the block's position change in the overall frame (global motion estimation) but, more frequently, it's specialized for specific parts, such as rectangular blocks, called search windows. The second choice is more convenient in terms of efficiency.

The motion estimation (ME) is one of the most expensive aspect of the compression video, in particular in terms of computations. In practice, the current frame is decomposed into blocks of pixels and the problem consists in finding a block identical or very similar to the one given in the previous frame, called reference frame. In practice, a sort of prediction takes place exploiting the information of the reference frame. In other words, the ME process eliminates the temporal redundancy between frames by exploiting temporal correlation: a block of pixels inside the current frame, which occupies a specific position inside the block, is searched in the reference frame and it will occupy a second, slightly different, position at a certain distance from the initial one.

To evaluate the best match between two blocks, objective criteria of a numerical type are employed. One of the most widely diffused function is the Sum of Absolute Difference (SAD), that can be defined as [3]:

$$SAD = \sum_{i=1}^{N} \sum_{j=1}^{M} |C(i,j) - R(i,j)| \tag{1.7}$$

where C and B represent current and reference blocks. One macroblock inside the current frame and one macroblock inside the reference frame are subtracted pixel by pixel. After, all the differences are subjected to the absolute value and, finally, are all added together. This operation is performed several times inside the search window: the macroblock inside the reference picture that brought to the minimum value of the sum of absolute differences is the best match. The value of the best match needs to be higher than a defined threshold, in order to compute the motion vector. In this case, the motion vector is transmitted (after the coding) together with the residual (the difference between the two blocks); otherwise, the current block arrives to the spatial transform phase without prediction. [2]

It must be highlighted that the computation of the motion vector for chroma macroblocks is obtained by halving the motion vector's components of the luma blocks. In other words, the block matching is just carried out for luminance case because the motion vector of the chrominance macroblocks is half of the components of the luminance.

There are several methods to computes motion vectors. These techniques can be classified into direct (pixel based) and indirect methods (feature based). The direct methods are those of interest to this work, in particular the most linear from a conceptual point of view but also the most performant: the block matching algorithm. In block matching, the current frame is usually decomposed into blocks of nxn pixels (typically 16x16), called macroblocks. For each of these the algorithm search for an identical or similar block in the reference frame, within a limited area called search window. Once a possible candidate has been identified, three cases may be presented:

1. The contents of the two macroblocks are identical (close match): the macroblock of the frame current is replaced by the single displacement vector with respect to the reference frame.

2. The content of the two macroblocks is very similar (best match): the motion vector and the difference or residual (prediction error) between the two macroblocks replaces the macroblock relative to the current frame.

3. The contents of the two macroblocks are very different: the macroblock is coded normally. No motion estimation is performed.

There are variants of this procedure with the aim of decreasing the amount of calculations necessary and at the same time obtain good results. Examples of these algorithms are the block hierarchical matching (Hierarchical Block Matching) and bi-dimensional logarithmic research (2-D Logarithmic Search). In addition to block matching techniques, others are used, an example is Gradient Matching and Phase correlation (phase correlation).

With the aforementioned methods, the displacement predictions occur at the pixel level. To obtain better results in estimating the motion vectors, although at the cost of greater complexity, interpolation techniques are used to determine these vectors with an accuracy of fractions of pixels (half a pixel, a quarter of a pixel and even an eighth of pixels).

1.5 Motion Compensation (MC)

The motion compensation (MC) is the technique that allow the frame prediction. The mechanism of the MC works in terms of transformation of the reference frame into the current one; it is used to predict a frame in a video inside the previous frames (and future frames if bidirectional). MC takes advantage of the small difference between frames, usually resulting from a movement of the camera or the movement of an object inside the frame itself. From a conceptual point of view, the motion compensation is based on an implicit hypothesis that the image represented in the current frame consists of a spatially transformed version of the previous image. This kind of correspondence can be mathematically modeled as a geometric transformation of the image plane.

In general, a motion compensated difference between two frames inside a video sequence shows less details than a not-compensated difference. The memory space or bandwidth required to encode compensated frames is highly smaller with respect to the not-compensated ones, at the cost of a loss of quality; a good trade-off needs, as always, to be found. A more detailed analysis regarding MC is carried out in chapter 3, where a graphical example is illustrate, to better understand how the compensation happens, and a motion compensated temporal filtering approach is proposed.

1.6 Encoder and Decoder

Video compression process is alternated between encoding and decoding steps. On the one hand, the encoder applies the compression to an input video signal and constructs a digital bitstream. It is used to save space inside the memory or to reduce the transmission bandwidth. On the other hand, the decoder reads the bitstream and reconstructs the original video signal. The encoder operates frame after frame. It uses parts of prior encoded pictures to perform a prediction on the current frame through motion compensation. After, the current predicted frame is compared with the original one and the difference, defined as the residual, is translated into a bitstream [2]. The digital bitstream is the information source taken by the decoder to reconstruct the video stream. One of this information is the display order. The decoder unpacks the bitstream and applies the gained residual to the prior decoded frames so that to obtain the current picture.

The architecture of the encoder is illustrated in figure 1.4.

Figure 1.4 shows a video encoder block diagram of the first digital systems, H.261 standard. Several standards came later to the H.261, the last is the HEVC, which is discussed in chapter 2. The task of the new standards was to improve the compression efficiency but all them are based on the shown block scheme of figure 1.4. Every cycle starts with the comparison between the input frame and the predicted one. A picture is taken from an input video sequence and the difference (residual) with its predicted is performed. The residual enters the loop, where, as can be seen from figure 1.4, it is subjected to a transformation, to remove the spatial redundancy, and a quantization. Then, the

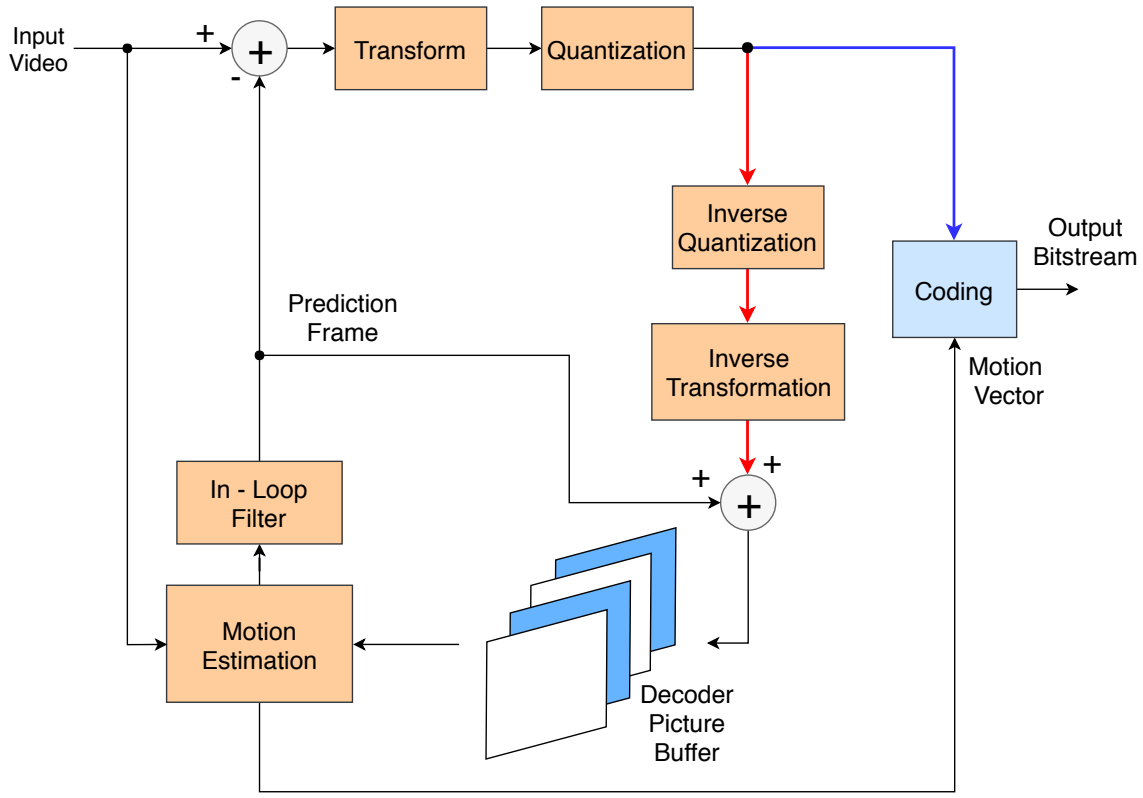


Figure 1.4: Example of a typical video encoder block diagram.

signal generated takes two branches. The blue line reaches the coder, which generates the output bitstream. The red line reaches the decoder: inside each encoder there is an internal decoder which performs the inverse quantization, the inverse transformation and adds the predicted frame to the result, reconstructing the original frame, which is used for the following picture encoding. In this way it gets the decoded picture that is saved inside the Decoded Picture Buffer (DPB), where it will be recalled during next motion estimations [2]. The motion estimation process receives, as input, the current frame (from the main input on the top left part of figure 1.4) and the pictures stored in the DPB and, generates, as output, the predicted frame and the motion vectors which will arrive to the bitstream coder. To deeply understand the role of each part, the subsections 1.6.1 to 1.6.5 describe the behavior and the characteristics of every part; the motion estimation is just mentioned because already discussed in section 1.4.

1.6.1 Transformation Block

At the input of the decoder, frames are divided in 8x8 blocks of pixels (H.261 standard) and are spatially transformed. The spatial transformation is carried out on the input blocks if the motion compensation didn't find a match on blocks of 8x8 pixel differences. The inverse transform, in the decoder stage (red line in figure 1.4), is carried out on the input blocks if the motion compensation didn't find a match on blocks of 8x8 pixel residuals. The typical transform function, which represents the transformation in video encoding, is the Discrete Cosine Transform (DCT). Generally, the discrete transform of the cosine, is the most widespread function that provides for spatial compression, able to detect the variations in information between an area and the contiguous one of a digital image neglecting repetitions; the function that supports temporal compression is instead entrusted to a special "motion vector", which identifies the dynamic components while leaving out the static ones.

The transformation shows two tasks:

- **Decorrelation:** decorrelation at this juncture is intended as the removal of spatial redundancy between neighboring pixels; the decorrelation allows to increase the coding efficiency.
- **Low frequency coefficients:** the input differences (or residual in the opposite process) are converted into few low-frequency coefficients; these coefficients are a threshold for successive quantizations, because the next quantizer can neglect those with smaller amplitude. This selection speed-up the future procedures without decreasing the quality of the rebuilt frame inside the video sequence. Therefore, since the information is concentrated in few components, there is an energy compaction effect.

1.6.2 Quantization Block

To transmit and codify a quantity with a finite number of bits, it is necessary to assume only a finite number of discrete co-domain values; this happens through a subsequent process of quantization of the value. To achieve this, the possible values are first and foremost limited between a maximum and a minimum around discrete values already defined, thus defining the relative decision regions and the dynamics of the quantizer itself: in this way the analog value of the original quantity, will be brought back to the nearest of the discrete values already defined through the decision process. In summary, the quantization block quantizes the coefficients calculated during the transformation (which acts as the aforementioned discrete values already defined), discarding the smaller amplitude coefficients. In this step, the efficiency increases, inducing, on the other hand, a loss in video quality.

1.6.3 Coding

The most important task of the coding block is to generate the output bitstream from the quantized coefficients and the motion vectors by employing variable length codes. Another task of the coder is to handle the rate control to maintain a fixed encoder output rate. The codewords generated by it are stored inside a temporary transmission buffer. The coder controls the fullness of this buffer and sends commands to the other encoder units to adjust their throughput [2]:

1) if the buffer is almost full, the coder orders to reduce the number of input coefficients;

2) if the buffer is almost empty, it commands to increase the throughput, increasing the number of generated coefficients.

If this solution is not enough to reduce the information produced by other units and the buffer is about to go in overflow, the coder can drop frames from transmission.

1.6.4 Motion Estimation

The motion estimation searches similarities between successive frames, computes and output their displacements (motion vector) to the transform block. The motion vector describes the movement value to be applied to the block inside the reference frame to obtain that within the current picture. In H.261 every frame is subdivided into square macroblocks of 16x16 pixels blocks, and the best match search is performed in the reference pictures for each block extracted from the current frame. The standard fixes the window search dimension to ± 15 pixels in the vertical and horizontal directions.

The best match between ref. and cur. frames can be calculated through mathematical function. The chosen is the SAD, whose expression is shown in section 1.4.

For sake of completeness, it's interesting to briefly describe how the decoder execute the opposite steps of the motion estimation to reconstruct the original video. Knowing the relative coordinates of each block, the decoder is able to reconstruct a predicted version of the current frame, through the

juxtaposition of appropriate blocks of the previous frame. This prediction mechanism, by defining a correspondence between information in the current frame and information from the previous one, is nothing but the motion compensation. The encoder must therefore only code the difference between the image predicted by means of motion compensation and the current framework.

1.6.5 In-Loop Filter

Sharp edges in the motion estimated blocks generate high peaks in the transform coefficients, since the sharp changes transfer in turn to the prediction error, which is nothing but the difference between current and predicted frames. High coefficients values clash with the hypothesis made in subsection 1.6.1: the transformation causes an energy compaction effect which in turn generates a reduction in the transmission rate. Not to create unwanted contrasts, a two-dimensional spatial filter can be inserted at the motion estimation output, with the task of smooth the undesired sharp edges. [2]

1.7 Background of the Video Standards

Historically, the most of the image, audio and video compression standards have been developed by the ITU (International Telecommunication Union), IEC (International Electrotechnical Commission) and the ISO (International Organization for Standardization) groups, in partnership with MPEG (Moving Pictures Expert Group) and VCEG (Video Coding Experts Group). These bodies proposed standards such as H.261, H.263, H.264/AVC (Advanced Video Coding), H.265-HEVC (High Efficiency Video Coder), MPEG-1, MPEG-2 (H.262), MPEG-4/AVC and other minor standards. Each standard was thought targeting a defined application or service, distinguishing one from the other by different characteristics and features.

In 1988, H.261 was introduced by the VCEG as the first digital video compression standard, used especially for videoconferencing and Integrated Services Digital Network (ISDN). This codec supported bit-rates between 64 kbps to 1.920 Mbps, and a fixed subsampling of 4:2:0. It worked with block size of 8x8 pixels and it supported QCIF (176x144) and CIF (352x288) formats. It was the first standard able to carry out the motion compensation, even if it didn't support random video access: all frames were of type P except for first (I frame). In that years, JPEG (Joint Photographic Experts Group) became a popular codec for images and MPEG started take care of the broadcast industry.

In 1993 MPEG adopted H.261 and JPEG to create a Suite called MPEG-1. The bit-rates reached the 1.5 Mbps, 8x8 pixels, sampling format 4:2:0 and only stereo audio. However, it introduces the P and B frames and the random access capability, missing in H.261. These optimizations were developed in order to use the MPEG-1 in the Video Compact Disc (VCD) or CD-ROM. A bitrate of 1.5 Mbps supported a maximum resolution up to 352x240 (30 frames per second) or 352x288 (25 frames per second).

In 1995, ISO, IEC and ITU introduced the upgrade of the MPEG-1 Suite, the MPEG-2. The majority of broadcast television implemented MPEG-2 and and it is not difficult to find it even nowadays for broadcast. It supports resolutions up to 1080p (30 frames per second). Moreover, MPEG-2 is the founding technology of DVDs. MPEG-2 is the first application-independent video compression standard. It introduces new motion compensation prediction modes, like the 16x8 motion compensation.

In 1996, the H.263 standard was designed. The structure of this standard is based on H.261, MPEG-1 and MPEG-2 with the introduction of the intra-prediction. It is optimized as low bit rate encoder (bit-rate $\leq$ 28.8kbps), suitable for videoconferencing, and supports resolutions up to 1408x1152. The H.263 allows half-pixel motion compensation and the computation of the motion vectors differs from previous standards: the motion vectors are calculated as the average the motion

vectors of neighboring blocks. H.263 also introduces a new filter inside the block diagram before the Decoded Picture Buffer.

In 2003, the H.264/AVC was developed by ITU-T Video Coding Experts Group (VCEG) and the ISO MPEG as the result of a joint venture effort, called Joint Video Team (JVT). The task of H.264/AVC was to provide broadcast video quality employing low bit rates on a vast field of networks, applications and systems. It supports resolutions up to 4096x2304, 4K Ultra-HD included. The macroblock follows the standard H.263, made up by 4 8x8 luma and 2 chroma blocks, adding the possibility to further divide each block in 8x4, 4x8 or 4x4 sub-blocks or using 16x8 or 8x16 block sizes. Both motion compensation and spatial transform can use 4x4 blocks. Also the sub-pixel accuracy increases: H.264 allows to use quarter-pixel accuracy.

In 2013, the most recent joint video project is the HEVC, introduced from the collaboration between ITU-T, VCEG and the ISO/IEC MPEG. The aim of this standard is to improve the compression performance, going to maintain the same video quality but halving the bit-rate with respect to H.264/AVC. A finer description of the HEVC is carry out in the following chapter, since it is the central topic of this work of thesis.

To summarize all the historical steps and characteristics of the standards, table 1.1 illustrated the developed standards from the first H.261 to the most recent HEVC. All the data concerning the standards, both of the description and the following table, are extracted from ref.[1] and ref.[4].

Table 1.1: History of the video compression standards, with year, developing organizations and specifics highlighted

Standard	Year	Organizations	Specifics
H.261	1988	ITU-T and VCEG	Resolution: 352x240 or 352x288, Bit-Rate: 64 - 1920 kbps, Format: 4:2:0.
MPEG-1	1993	ISO, MPEG, IEC	Resolution: QCIF (176x144) and CIF (352x288), Bit-Rate: up to 1.5 Mbps, Format: 4:2:0.
MPEG-2	1995	ISO, IEC, ITU-T	Resolution: up to 1080p, Bit-Rate: $\leq$ 28.8kbps, Format: 4:2:0.
H.263	1996	ITU-T	Resolution: up to 1408x1152, Bit-Rate: up to 1.5 Mbps, Intra-Prediction.
H.264/MPEG-4 AVC	2003	ISO, IEC, ITU-T, VCEG	Resolution: up to 4096x2304, Bit-Rate: up to 1.5 Mbps, Format: 4:2:0, 4:2:2 and 4:4:4.
H.265-HEVC	2013	ISO, IEC, ITU-T	Resolution: up to 8192x4320, Double Compression Rate, Improved variable-block-size, segmentation Bit-Rate: Half of H.264, Format: 4:2:0, 4:2:2 and 4:4:4.

Chapter 2

HEVC

HEVC (High Efficiency Video Coding) is the last video compression standard, known also with the code H.265, and was developed together by the Video Coding Experts Group (VCEG) from ITU-T and Moving Picture Experts Group (MPEG) from ISO/IEC. [2] [3] [5] [6] [7] [8]
Nowadays, the continuous demanding for larger bandwidth or larger memory storage led to the introduction of a new standard for high resolution videos. The development of the HEVC started while the dominant video coder was H.264/AVC. The reason behind the progress of a new standard was the implementation of the Full-HD resolution among mobile devices, used to watch or record high resolution videos, and, moreover, the advent of the new ultra-HD resolutions, 4k and 8k, both requiring a larger memory space/bandwidth to be stored/sent.

HEVC supports resolutions up to 8192x4320, doubling the video compression rate compared to H.264, while still offering the same level of video quality [3]. This means greater efficiency, since video can be recorded, stored and transmitted with the same quality, consuming less bandwidth. The H.265 codec is similar to H.264 because it compares different parts of a video frame in order to find redundant areas, both within each frame and in subsequent frames. The main advantages that H.265 offers, compared to H.264, include the expansion of the comparison model, i.e. the different encoding area from 16x16 to an area up to 64x64 pixels. The segmentation and prediction of blocks of variable size within the same image, as well as a greater prediction of the motion vector, has also improved. An effective use of these improvements leads to greater signal processing capacity for video compression, and at the same time less impact on the amount of computation required for decompression. Another strength of this standard is the flexibility: HEVC defines only the format and syntax of the encoder output bitstream without specifying how the encoder or decoder have to be made [2].

H.265 supports all the applications and resolutions that AVC could manage with a particular focus on very high-resolution videos and high frame rates. It can handle 4:2:0, 4:2:2 and 4:4:4 luma and chroma resolution ratio and 8, 10, 12 or 16 bits per sample. These two properties are classified into HEVC profiles: Main Profile employs 8 bit and 4:2:0 format.

2.1 Encoder

As introduced in chapter 1, the structure of the HEVC is an improved architecture derived from the basic structure of the H.261 encoder. It is illustrated in figure 2.1 and immediately after described.

The structure shows the same configuration of figure 1.4 with the addition of the switch, hybrid architecture: the decoder can perform both intra-picture and inter-picture prediction. Any input

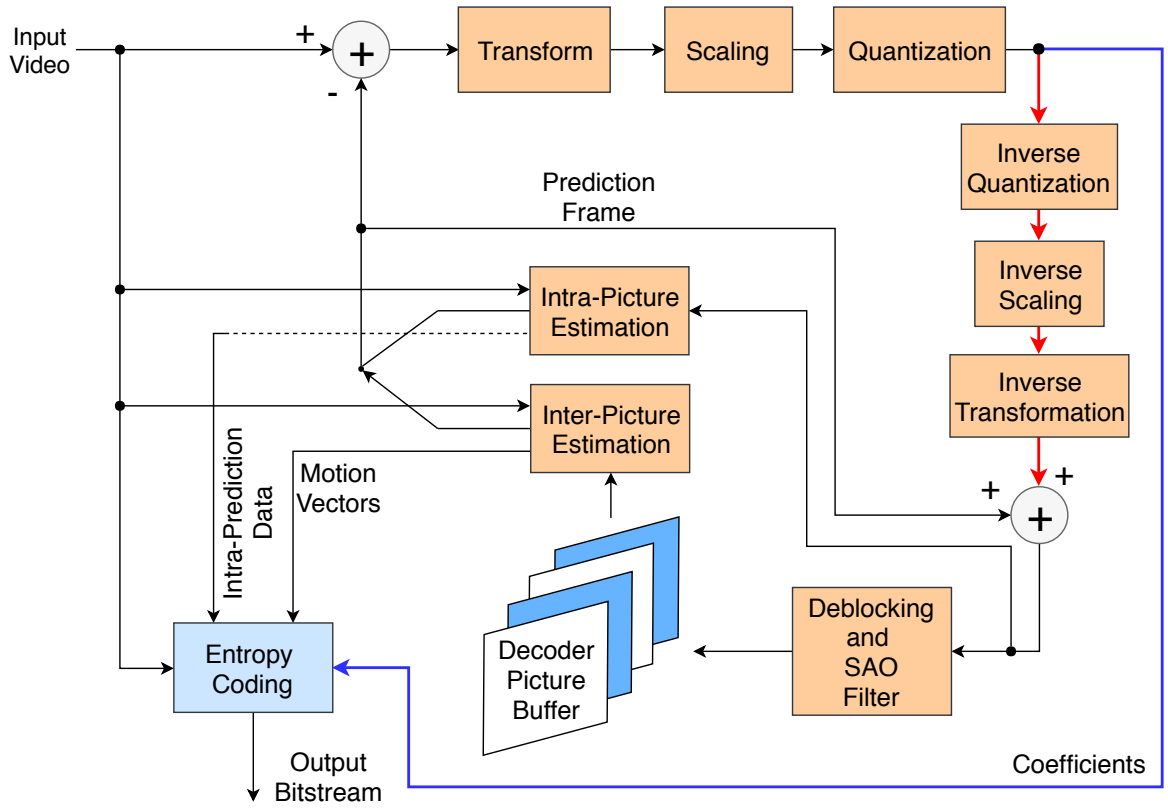


Figure 2.1: Example of a typical HEVC video encoder block diagram. Figure reworked from ref.[5]

frame is taken and, after the decoding steps (red line), is subdivided into blocks of pixels and each block is predicted using intra- or inter-prediction:

- The Intra-prediction technique compares blocks only inside the same picture in order to remove spatial redundancy from neighboring regions and it is used for I frames (Intra predictive frames).
- The Inter-prediction technique employs motion compensation between a reference frame and a current frame in order to remove temporal redundancy; the mechanism is the same described in section 1.4 and 1.5.

Another evident difference in this second structure with respect to the basic structure of the H.261 encoder is the different coding phase. In fig.1.4, motion vectors were calculated and sent to the coder, which, together with the quantized coefficients (blue line), generates the output bitstream. In fig.2.1, it can be seen that the motion vectors are not the only data generated by the motion estimation, but also intra-picture estimation generates a similar output, called intra-prediction data. Both intra and inter prediction generate the predicted frame that are compared with the original one. The comparison returns the prediction error, computed between the original frame and the predicted frame. The result is subtracted to the input frame and, after, transformed, scaled and quantized. The coefficients obtained from these steps are, with the input frame, the third and fourth input of the entropy coding block. Intra-prediction data, motion vectors, quantized coefficients and input frames are used to generate the output bitstream. [5]

The new parts, such as filters and intra-picture estimation are discussed in the following subsections.

2.1.1 Block Partitioning

The previous video standards, such as H.264/AVC, divided the video sequences' macroblocks into 16x16 lumimance samples and two 8x8 chrominance samples. In H.264 prediction, a macroblock can be predicted as one 16x16, two 16x8, two 8x16 or four 8x8 partitions. In the case of 8x8, each macroblocks can be further divided into one 8x8, two 8x4, two 4x8 or four 4x4 partitions [9]. This scheme is illustrated in figure 2.2.

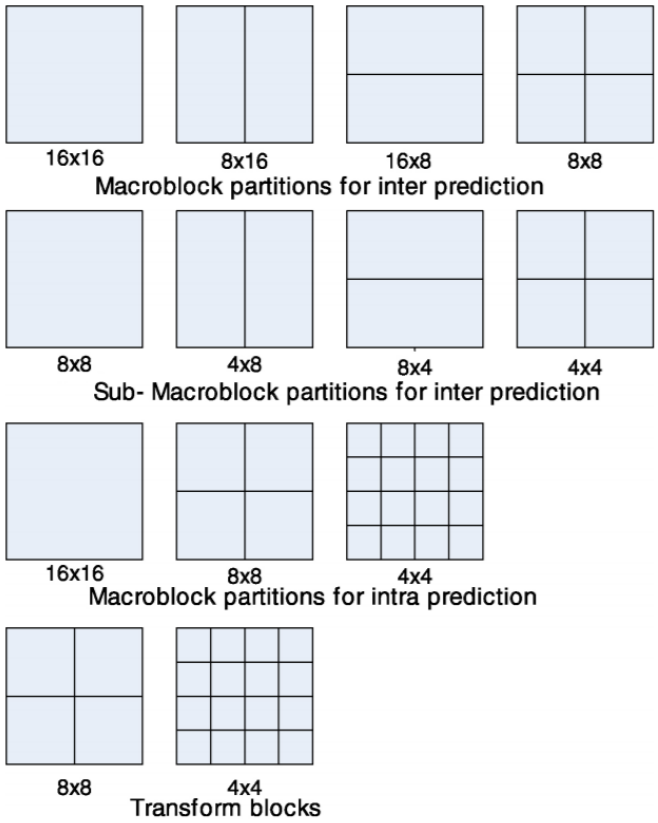


Figure 2.2: H.264/AVC: block partitioning structure for prediction and transformation. [9]

The standard partitioning can be approximated to a 3-levels quad-tree structure (nodes from 4x4 to 16x16) and, even if non-squared partition is possible, asymmetric division is not supported. [9]

The 16x16 division has two problematic consequences: it is a limiting factor for the compression of high resolution video and , at the same time, the compression efficiency of low resolution video requires small pixel blocks. HEVC tries to solve both limits with a smart architecture which incorporates a flexible mechanism to subdivide frames into variable size blocks.

In HEVC, each frame is subdivided into disjunct blocks of different size, each of which becomes the root for a quad-tree structure since they can be further divided in a recursive way until the smallest block dimension is reached. These root blocks are called Coding Tree Blocks (CTB). The aforementioned operation is performed on luminance and chrominance samples: with 4:2:0 format, a luma CTB covers an area of (2^N x 2^N) luma pixels; the two chroma CTBs cover an area of (2^{N-1} x 2^{N-1}), where N is equal to 4, 5 or 6. Hence, the luminance CTB can have an area of 16x16, 32x32 and 64x64 samples and the chrominance CTB an area equal to half the luma block, 8x8, 16x16 and 32x32 samples respectively. A luma CTB and the two corresponding chroma form a Coding Tree Unit (CTU) that is the HEVC basic compression parallel processing unit. On the one hand, larger

CTUs correspond to higher coding efficiency but, on the other hand, it may increase the encoder delay and computational complexity.

According to the quad-tree subdivision scheme, a CTU (the root) can be split into different Coding Units (CU) of variable size until the 8x8 block size is reached. The CU contains a luma CB and two chroma CBs. It chooses between intra- or inter-prediction which should be applied on its CBs. Moreover, it can be further divided into Prediction Units (PU) and Transform Units (TU). The intra-prediction mode is based on TU that is the basis element also for the spatial transform. If the luma CU has minimum size (8x8), it can be split into four 4x4 TU. The blocks for a TU must be square starting from 4x4 up to 32x32. In total, they can be 4x4, 8x8, 16x16 and 32x32. Inter-prediction is instead based on PUs, alternatively called PBs (Prediction Blocks). The subdivision into PUs is independent from that for the TUs. There are eight different mode to split a CU into PUs since for CUs bigger than 8x8 the asymmetric partitioning modes can be employed and the dimensions do not have to be square. The smallest PUs have size 8x4 or 4x8. All the different way to decompose a frame into blocks (TU or PU) are illustrated in figure 2.3.

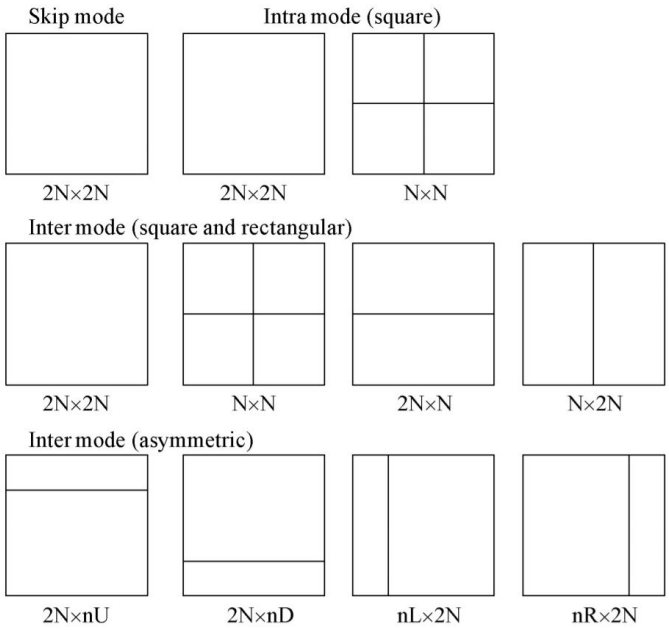


Figure 2.3: Prediction units (PUs) splitting modes in HEVC: CU partitioning modes used for deriving the PUs. The NxN mode and the inter asymmetric modes can be applied only if the CU size is larger than 8x8 luma samples. [9]

The possible motion compensation block sizes are 24 and are collected in the following list:

64x64	64x48	64x32	64x16	48x64	32x64
16x64	32x32	32x24	32x16	32x8	24x32
16x32	8x32	16x16	16x12	16x8	16x4
12x16	8x16	4x16	8x8	8x4	4x8

The choice of the best partitioning is attributed to the encoder, which calculates a trade-off between distortion and bit-rate, in order to have a good ratio between performance and video quality, and,

based on the trade-off, selects the best possible partitioning. Informations about the partitioning selected are written inside the bitstream.

2.1.2 Inter-Picture Estimation

The inter-picture prediction is employed for P and B frames and removes the temporal redundancy between different pictures by employing motion estimation and compensation. The I frames are predicted by the intra-prediction, which is based on the TBs, while the inter-picture prediction is based on the PBs. The explanation of the inter-picture estimation is exactly the same of motion estimation and compensation of the previous chapter: each block of samples of the current frames is searched in previously decoded reference frames. The best matching operation is execute through the SAD, because the best match is represented by the block showing minimum SAD with the current block. At this point, the motion vector is computed: it is indicated as (Δx, Δy), which indicates the displacements in the two directions between the positions of the two blocks. In the encoder the Inter-prediction estimation provides to the entropy coder the motion vectors and the reference frames indexes (Δt), index to understand to which reference picture the motion vector refers. [2]

All these steps are the same described for the H.261 standard, a part for the intra-prediction. HEVC adds not just the intra-frame prediction, but also the possibility to perform two types of inter-prediction, the uni-directional and bi-directional predictions (concisely uni-prediction and bi-prediction). The uni-prediction was already present; the bi-prediction uses two sets of motion data for each block and by averaging them it obtains a unique prediction.

Furthermore, the best match is searched in two steps: Integer Motion Estimation (IMO) and Fractional Motion Estimation (FMO).

- **IMO:** The first step is the motion estimation process (refer to chapter 1.4 and 1.6.4), set with a window search of ± 64 pixels.
- **FMO:** After the best match search, the samples are interpolated in order to estimate the values at fractional positions. Then, a second search is performed, inside a smaller window, to find a fractional motion vector.

Summarizing, the first step makes a coarse search inside the whole search window and the second performs a finer search around the IME result. HEVC supports $\frac{1}{4}$ accuracy for luminance and $\frac{1}{8}$ pixel accuracy for chroma components. [2]

2.1.3 Intra-Picture Estimation

Intra-picture prediction is employed for I frames and removes the spatial redundancy within the same picture. The intra prediction is performed at transform block level, defined as square blocks from 4x4 to 32x32 block sized [5]. It compares the samples of the selected TB with the ones on the boundaries of the neighboring decoded TB (reference samples) and, once obtained the best match, generates the intra prediction data. This kind of prediction supports 33 directional modes to model structures with directional edges, one planar prediction and one DC prediction modes. The last two are used for image areas with smooth contents. The decoder has to select one of the 35 modes. The choice is then signaled in the bitstream together with the intra prediction data. [2]

Smooth filters are usually used for reference samples, because the intra-prediction works with boundary TBs: the filters operates on the boundaries to reduce the in-between discontinuities that can, potentially, cause a wrong choice in the angular mode. If a filter has been applied, a post-processing smoothing step substitutes the boundary samples with their filtered ones.

2.1.4 Entropy Coding

The entropy coder reduces the number of bits used to represent a given symbol on the quantizer output in order to improve the coding efficiency. HEVC employs the Context-based Adaptive Binary Arithmetic Coding (CABAC) algorithm to perform entropy coding. It is a lossless compression method that assigns to a symbol a number of bits logarithmically proportional to the probability of that symbol [5]. Therefore, frequently used characters are represented by less bits while rare symbols are written on more bits.

The steps defining the encoding process are:
1) binarization,
2) context modeling,
3) binary arithmetic coding.

First, each syntax element is binarized inside the entropy coder; binarization means to map each element to a binary symbol (called "bin"). This operation is performed by the binarizer: different binarization processes can be employed in HEVC and the right one is chosen on the type of the syntax element. After the first phase, the coder can choose between two modes of operations: regular or bypass modes. In regular mode, a probability estimator and assigner chooses a probability model for the current bin according to the statistics of the last coded bins and according to other specific context. In the last step, the Arithmetic Encoder compresses the bins into bits according to the selected probability model and updates the latter for the next computations. In bypass mode, the bins are directly passed to the Arithmetic Encoder without choosing a probability model. This can speed-up the output coder and is chosen for bins that are uniformly distributed. [5]

2.1.5 Deblocking and SAO Filters

The decoder is always embedded in the encoder structure in order to reconstructs the frames from the coefficients. Then, the reconstructed frames needs to be stored inside the Decoded Picture Buffer (DPB). Before reaching it, these frames pass through two filters: the deblocking and the SAO.

Deblocking Filter: it attenuates discontinuities between the blocks, more in details between PB and TB boundaries. This filter works only on the boundaries between CUs samples and not on those on the edge of the picture or inside the CUs. [5]

SAO Filter: SAO means Sample Adaptive Offset; it is applied to the output of the deblocking one and has the aim to remove ringing artifacts. It works on entire CTUs and can add to the samples a negative or positive offset taken from a look-up table (LUT table) generated by the encoder in order to smooth out artifacts. [5]

2.1.6 HEVC Reference Software

The HEVC model is available as C++ open-source code, developed by the JCT-VC (Joint Collaborative Team on Video Coding) ITU-T [10], in particular the HM16.6 model. This HM software provides a reference implementation of both HEVC encoder and decoder. This model gives the possibility to carry out simulations and tests: one of the task of this code has been verify the contribution of the motion estimation process inside the encoder with respect to all the other processes. This weight and time consumption of the ME in the encoding process are discussed in chapter 3.

In the end, table 2.1 illustrates a summary of the characteristics of three standards, MEPG-2, H.264 and HEVC. For HEVC, the most evident consequences are an improvement of the bit-rate in

every field of application of the codec and, therefore, at the same bit-rate, the possibility to use the same broadcasting resources for larger quality images (4k), at the cost of an increase of the general complexity.

Table 2.1: Summary table to compare three standards, highlighting the improvements developed in the HEVC standard.

	MPEG-2	**H.264**	**HEVC**
Partitioning Size	16x16 (Macroblocks)	16x16 (Macroblocks)	8x8 to 64x64 (Coding Unit)
Partitioning	Inter 16x16, Intra 8x8	Sub-Block down to 4x4	Intra: Down to 4x4 (symmetric); Inter 4x8 or 8x4 uni-directional
Transform	Floating DCT	Integer DCT 8x8 4x4	Square IDTC from 32x32 to 4x4 + DST Luma Intra 4x4
Intra Prediction	DC Predictor	Up to 9 Predictors	35 Predictors
Motion Prediction	Vector from one neighbor	Spatial Median (3 blocks)	Advance Motion Vector Prediction Spatial + Temporal
Motion-Copy Mode	/	Direct Mode	Merge Mode
Motion Prediction	1/2 Pixel Bilinear	1/2 Pixel 6 tap 1/4 Pixel bi-linear	1/4 Pixel 7 or 8 tap Luma 1/8 4-tap Chroma
Filters	/	Deblocking Filter	Deblocking and SAO Filters
Scalability Tools	Through Extensions	Through Extensions	Temporal Scalability Included

Chapter 3

Motion Estimation and Motion Compensation

The HEVC video standard doubles the compression capability with respect the H.264/AVC standard, introduces new features (described in the previous chapter and in table 2.1) but increases the complexity of the encoding process. The complexity is translated in a raise of the computational cost because of a larger encoder decision space to be explored during the video compression. The doubled compression rate without quality loss is not the only improvement brought by HEVC standard; some example are the larger combinations of block sizes, improved partitioning, more intra- and inter-prediction modes, new in-loop and interpolation filters. Consequently, also the negative aspects are not just a computational cost increase: a raise in the memory accesses number, in the temperature and in the power consumption of the encoder occur. A convenient way to deal with these drawbacks is the motion estimation analysis and optimization. Motion Estimation takes a large amount of the encoding time, the majority of the time, due to all the cost functions (like the Sum of absolute difference, SAD). Cost functions occupies $\sim 40\%$ of the total time, while the interpolation filter accounts for $\sim 20\%$. [2] [3] [6]

Therefore, the SAD hardware accelerator design optimization is based on the enhancement of the motion estimation, trying to overcome to the speed, power consumption and complexity limits. Being the SAD the most time consuming operation inside the accelerator, in this work the introduction of approximate adders is a possible solution to reduce the three HEVC limits. Before starting the discussione of approximate computing, approximate adders, substitutions and impact of them on the architecture, the motion estimation, compensation and the system behind the hardware encoder needs to be described. The following section and the next chapter have exactly this goal.

3.1 Motion Estimation

The motion estimation consists in a searching process for the causal block (macroblock) with highest similarity with respect to the target block to be predicted (current frame), employing the block matching algorithm over the previously reference encoded frames [11]. If the prediction is bidirectional, the reference frames can occupy positions both before or after the current frame. The block matching process executes a subtraction (SAD) between the current block and each equally sized block that exists in a search window defined in the reference frame. The best match is obviously achieved by the reference frame which returns the lowest difference (minimum SAD). In this instance, also the motion vector is computed: it is given by the displacement between the best matched reference block's

position and the corresponding co-located position of the current block [11]. The best match difference between two blocks is executed through the Sum of Absolute Difference, defined as [3]:

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

where C and B represent current and reference blocks.

HEVC uses two motion vectors if the process is bi-predictive coding, one if uni-predictive coding. The difference between the two procedures is that the latter computes the motion compensation between blocks in the usual and already described way of section 1.5, while the first, the bi-predictive coding, performs an average of two motion compensated blocks, obtained from the two references due to bi-prediction. Moreover, HEVC reports every index for every reference frame, sent with the motion vectors.

The block matching algorithm in HEVC is an adaptive block size, from 4x4 to 64x64, process which includes square and rectangular blocks generated by symmetric and asymmetric block partitioning [11]. H.264 was the first standard to step over the fixed block size, adopting the variable block size, but employed a reduced range of pixels' size due to the absence of the asymmetric block partitioning. Thanks to the larger block sizes and prediction blocks, HEVC reduces the number of bits to compute a motion vector, because it's able to predict more pixels using a single translational motion vector. For example, considering a complex textured region, large blocks don't provide a good prediction and a more precise motion estimation could be needed. In this case, the advanced partitioning of the HEVC standard offers a large number of possible block sizes and allow to compact complex textured blocks into a single motion vector, leading to reduced, improved and faster prediction results. Obviously, the improved HEVC partitioning increases the computational complexity of the system and increases the bitrate requirements. [11]

Coming back to the motion estimation, it is a best match search between the window search of a current and a reference frame. The best match window returns the minimum value of the SAD. Till now, the search window has been defined as a set of rectangular block defined in the reference frame to be compared with the actual block inside the current frame. The concept of search window can be further analyzed, suggesting a specific definition and a graphic explanation. The search window is represented in figure 3.1 (light blue region). It is constructed starting from the black dot in figure 3.1 inside the current PB and generating a window based on the dimension of the search area defined a priori. In next sections the experimental value is ± 64, so it's convenient to illustrate an example with such value. Thence, if the search region is ± 64, a square of 128x128 pixels is constructed. After, before starting the best match algorithm, the Motion Estimation unit inside the encoder verifies if the search region broadens outside the picture boundary and, eventually, clip it. At this point, the seek of the current block is carried out: the same search window (same dimensions and position) is searched inside the reference frame. This defines the inter-prediction process. This process can be executed through different algorithms, to fasten the search. The slower but more accurate is the so-called brute-force approach: it starts from the left upper corner of the search widow, takes a reference PB which begins in that exact position and of the same size of the current PB and performs the SAD operation. After, the reference PB is moved one sample column towards right and a new SAD is computed between the the same current PB and the new reference PB. This procedure is repeated until the reference block reaches the right edge of the picture and, then, it is moved back to the left side and translated downwards by one sample row. The same routine is cyclically repeated for the entire row and the following rows of the reference PB. It stops at the right lower corner of the search window [2]. Considering the previosly cited search window of 128x128 pixels, the brute force algorithm consists in 16384 SAD for each current PB. This is a very high number of computations,

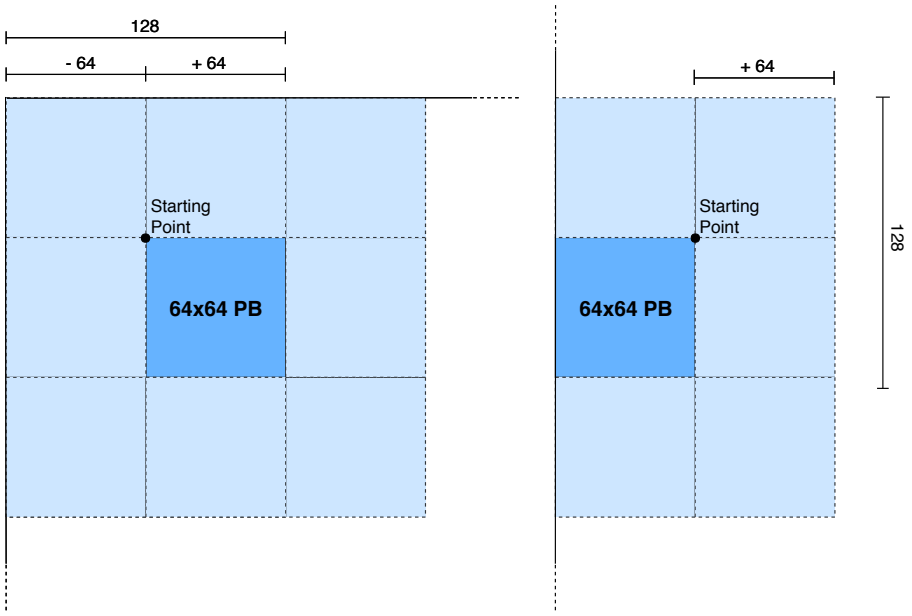


Figure 3.1: Two example of search windows: on the left, the PB has a complete search window around (no split); On the right, the search window is clipped, since it would broaden outside the frame edges.

complex and slow, but, at the same time, complete. Other examples of algorithms are the TZ search and the selective search. The TZ search starts to search not from the left upper corner like the brute force, but from the position of the current PB. Starting from there, it moves around this point in a rhomboidal pattern. It employs subsampling, meaning that there is not a comparison between the current block and all the reference blocks, but it discretizes. The selective search associates a specific subsampling in dependence on the dimension of the current frame's block size. Both ensures a quicker best matching process with an imperceptible loss in video quality. These two algorithms have not been used in this work of thesis but enhance the search process. These two approaches are interesting to be implemented in future works, maybe developing a Matlab program able to execute an advanced motion estimation not just to verify the effective speed-up giving by the new algorithms, but combine them with one the possible improvements that are discussed in the final chapter of future possible implementations.

The motion estimation and compensation are two lossy processes, meaning that a difference between compressed data and reconstructed data is allowed. Lossy processes are more efficient than lossless ones, ensuring higher transmission/storage speed and decreasing the bandwidth/memory space used to transmit/save a video. This is possible for error-tolerant applications, taking advantage from the limited human sensations, and it is exploited in video compression techniques during the motion estimation and compensation, removing temporal redundancy. A clear example is illustrated in the two figures 3.2 and 3.3 (video downloaded from ref.[12]): the pictures show, on the top left, the reference image (the subsequent frame), on the top right, the absolute value of the difference between the shown frame and its best match (the previous in this case), on the lower left part, the motion vector related to the difference (car in movement) and, on the bottom right, there is the motion compensated difference.

These are two separated (not consecutive) examples of the same video sequence. The motion compensated difference (bottom right) between two frames contains significantly less details than the direct difference (top right). Hence, the compression level with motion compensation increases, the process is faster, occupies less space, the informations required to encode the compensated frame

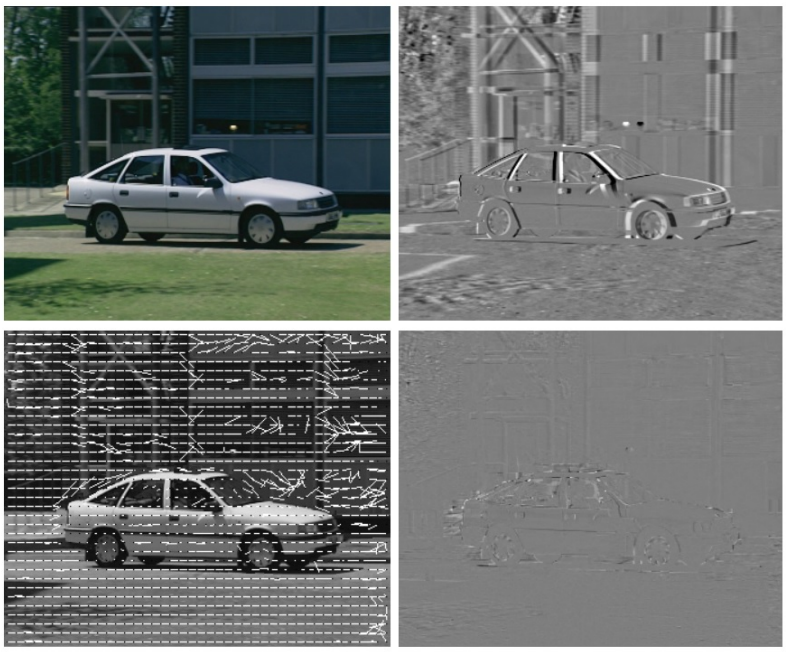


Figure 3.2: Motion estimation (motion vector) and compensation of two consecutive frames of the video sequence "Vectra21Frames.yuv" [12]: (top left) the reference frame; (top right) absolute value of the difference between consecutive frames; (bottom left) motion vector; (bottom right) motion compensated difference.

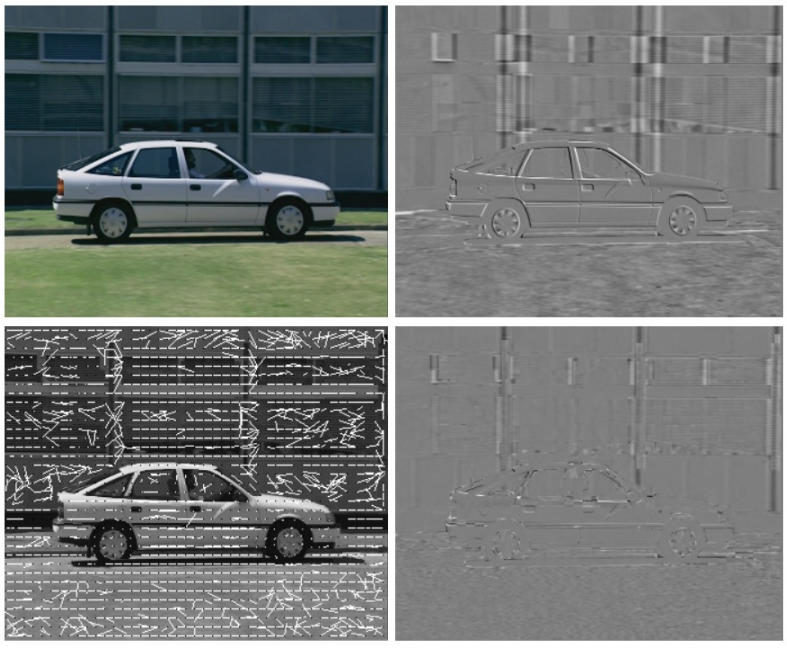


Figure 3.3: Motion estimation (motion vector) and compensation of two consecutive frames of the video sequence "Vectra21Frames.yuv". [12]

are smaller than with the difference frame but the general quality is affected by the transformation. To carry out this motion estimation and compensation, so to evaluate the motion vectors and the differences of the video, the freeware software VcDemo was employed [13]. A similar code, written in Matlab has been developed by the author of this thesis in the next section and for the error evaluation related to the introduction of approximate adders.

3.2 Motion Compensation - Matlab Script

The motion compensation principle is based on the repetitiveness of the scene content of a video. If the content is almost constant from frame to frame, the redundancy between frames can be estimated, through the motion estimation, then the motion can be compensated to obtain the compression. As introduced in chapter 1.4 and 1.5, motion compensation prediction for video compression is an algorithmic technique used to predict a frame in a video. Motion compensation describes a picture in terms of the transformation of a reference picture to a current picture; in other words, it's a technique based on the reconstruction of the best match frame into the current one, whose result is a predicted frame. A reconstructed copy of the reconstructed frame is kept at both the encoder and decoder.

To better understand the complete video compression, a Matlab program has been written. It executes the motion estimation and compensation of an input video sequence; the script is divided in two parts: a first motion study frame to frame and a second part regarding the introduction of filter (next section). The current and reference frames are divided in blocks (8x8 pixels in this specific case). Each reference block is compared to the block in the current frame, and the sum of absolute difference (SAD) is computed, same procedure described in the previous chapter; the reconstructed block will consist of the blocks with minimum SAD. The motion compensation analysis and the frames prediction are the two main concepts developed by this Matlab program. The program is composed of a main and two functions: the main program works on two frames, the current and reference frames, captured from the input video sequence, and launches the "MotionCompensation" function, a function that execute the motion estimation and compensation of a YUV video sequence, which is read thanks to a "yuv_import" function used to import the video sequence and read its frames. This last function takes the YUV file, its original resolution and number of frames (since raw YUV file has no header), the number of temporal wavelet level decompositions, macro block size and search window size for block matching motion estimation as input parameters. The YUV video file chosen for the simulation is $foreman\_cif.yuv$, with a resolution of 352 x 288 and 300 frames, source found in ref.[12]. Two example frames represented in figure 3.4.

One group of pictures (GOP) at a time (2 consecutive frames) is extracted from the YUV file and only its Y component is used for motion compensation. Only the Y component of the YUV file was considered, thus catering only to gray-scale videos, but also U and V components can also be considered, thus performing the motion compensation for a colored video. The pair of frames are then subjected to motion estimation to calculate the forward motion vectors using block matching algorithm of macro block size 8 and window size 8 (these two quantities can be modified at will). The dimensions of the macroblocks and the search window are set following the standard H.261 for simplicity. This is the most interesting part of the program for the purpose of this work: it consists in the computation of the difference, absolute difference, motion compensated difference and motion compensated absolute difference between the reference and current frame, which can be appreciate in the first four pictures of figure 3.5, (a), (b), (c) and (d). These four figures show how the Sum of Absolute Difference (SAD) graphically operates. As expected, the motion compensated difference between two frames contains significantly less detail than the prior picture, and, thus, compresses much better than the rest. In this step, just two consecutive frames are represented, for sake of simplicity.

The last two pictures (e) and (f) sketch the predicted frame reconstruction: in the zoom-in part

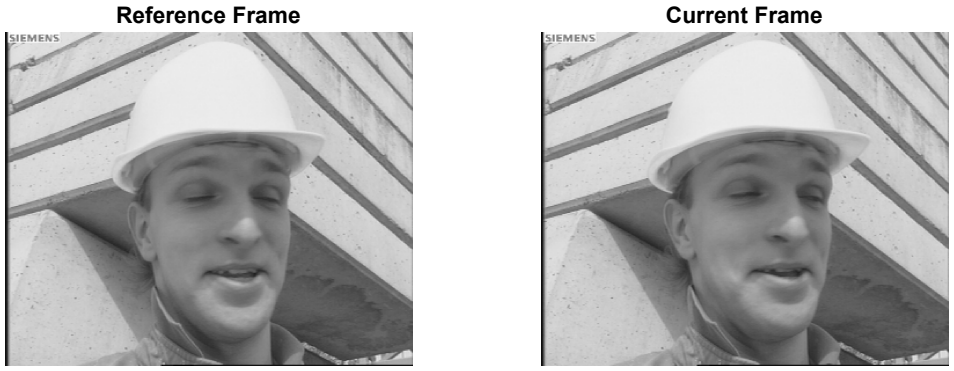


Figure 3.4: Reference and Current frames resulting from motion estimation of the Matlab program. In this case, they are consecutive frames and the best match motion estimation algorithm will just search in this two and not in previous or next ones of the video sequence for simplicity.

it is highlighted the pixel by pixel difference between reference and current; the best result difference substitutes the already present pixel.

3.3 Motion Compensated Temporal Filtering (MCTF)

A specific application of the motion compensation and estimation is the motion compensated temporal filtering (MCTF). More precisely the motion compensation adopting temporal filtering is an extension of the motion compensation prediction. MCTF has been introduce in all the H.26x versions and, nowadays, also in HEVC. MCTF is mainly optimized for low bit-rate video coding and for compressing high definition slow motion videos, because the motion between frame to frame is very minimal and videos have higher bit rate compared to the fast motion videos. The task of such extension is to obtain higher quality results employing a filtering step to the motion compensation.

To obtain higher compression's resolution, the idea of temporal scalability can be exploit: temporal scalability refers to the ability to reduce the frame rate of an encoded video bit stream by dropping packets and, thereby, reducing the bit rate of the stream. A given bit stream includes different sub-streams each with a different frame rate. The technique of motion compensation can be exploit with temporal scalability to remove the redundancy of data between the video frames. In particular, redundancy along the temporal axis is used to reduce the information in the succeeding frames; a clarifying figure is shown in figure 3.6.

In order to achieve this purpose, the Haar wavelet is employed. Haar wavelet is a sequence of rescaled "square-shaped" functions which together form a wavelet family or a wavelet basis. Wavelet analysis can be compared with the Fourier analysis because it allows to represent a target function in terms of an orthonormal basis. The Haar sequence is the first known wavelet basis, developed in 1909.

The GOP is taken and the Haar basis for wavelet decomposition is applied: this can be interpreted as decomposition of a frame pair (A,B) into one average (low-pass) and one difference (high-pass) frame, defined by the equations given below (eq. 3.2 and eq. 3.3):

$$L(m,n) = \frac{1}{2} \cdot [A(m,n) + B(m,n)] \tag{3.2}$$

$$H(m,n) = [A(m,n) - B(m,n)] \tag{3.3}$$

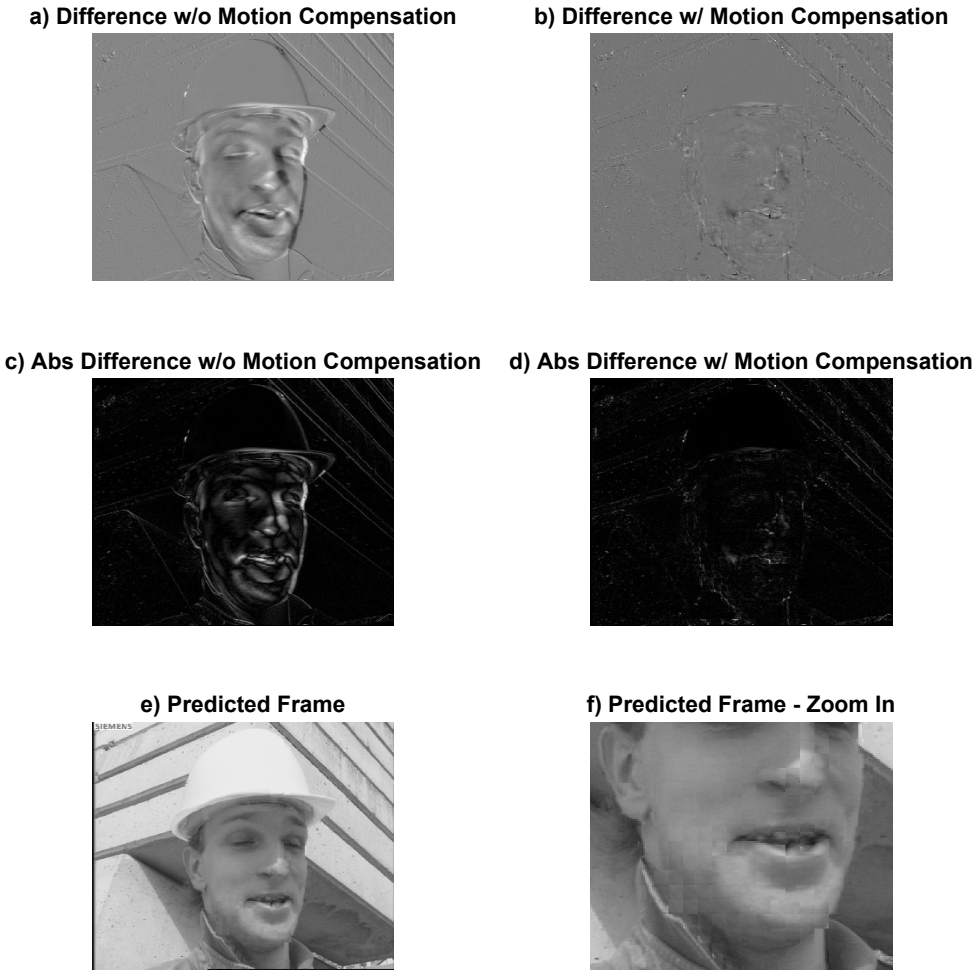


Figure 3.5: a) b) Difference without and with motion compensation of two best match consecutive frames of the video sequence $foreman\_cif.yuv$; c) d) absolute difference without and with motion compensation of two best match consecutive frames of the video sequence $foreman\_cif.yuv$; e) reconstruction of the reference into the current frame f) particular of the reconstruction highlighting the pixel by pixel difference and the following sum of the two frames. While the first four figures highlight the loss of precision/accuracy, resulting in a faster and dimensionally smaller compression, pictures e highlights the elimination of the temporal redundancy between frames.

where m and n represent the pixels of the two frames A and B. The low-pass frames are then again combined and subjected to the above equations resulting in subsequent levels of a wavelet tree decomposition [14]. The Haar wavelet decomposition is deeply studied by Jens-Rainer Ohm et al. [14], where details about the calculation made on the reference and current frames are analyzed and explained. If motion compensation is used in combination with a 2D spatial wavelet transform, the scheme is denoted as a 3D wavelet decomposition, the same observed in figure 3.6.

For sake of clarity, not all the calculation carried out in ref.[14] will be hereafter reported but only

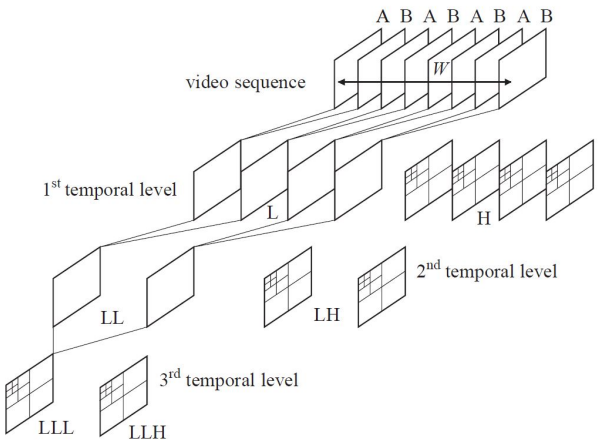


Figure 3.6: 3D Wavelet transform tree, where, in the simplest case, a Haar basis can be used for wavelet decomposition along the temporal. This allows, in combination with MC, to remove the redundant frames and reconstruct new frames where redundancy is partially eliminated. [14]

the final equations useful to execute the program, equations 3.4 and 3.5.

$$H(m,n) = A(m,n) - B(m+k, n+l) \tag{3.4}$$

$$L(m,n) = B(m,n) + \frac{1}{2} \cdot H(m+\tilde{k}, n+\tilde{l}) = \frac{1}{2} \cdot [B(m,n) + A(m+\tilde{k}, n+\tilde{l}) \tag{3.5}$$

where $[k,l]$ represents the forward motion vectors and $[\tilde{k},\tilde{l}]$ represents the backward motion vectors. The equivalence with 3.2 and 3.3 is evident.

These two last equations can be applied to the previous considered couple of frames to obtain a more detailed version of the predicted frame with respect to the simple motion compensation. As already sad, the all the calculations' step starting from to 3.2 and 3.3 to get equations 3.4 and 3.5 are not carried out, but the same procedure can be carried out graphically on the couple of frames, going to observe concretely how this image process works. The starting point is related to the concept of unconnected, uniquely connected and multiple connected pixels, which allow to transform the reference and current into predicted frame. The working principle of the motion estimation is based on the estimation of the the motion vector and the best match of the search window of the current inside the reference frame. In this case, blocks are fixed in the current frame while the most similar blocks are left floating in the reference frame. As a result, not all the pixels of the reference frame are used for prediction of the current frame, i.e. not all the pixels of the reference frame are uniquely mapped to the pixels in the current frame. This leads to the existence of unconnected pixels. Also, many fixed blocks of the current frame may overlap in the reference frame, leading to the existence of multiple connected pixels; unconnected, uniquely connected and multiple connected pixels are illustrated in Fig.3.7. Uniquely connected pixels works with the same mechanism found in the previous step, before introducing temporal scalability and representing two identical pixels of the two frames under compensation. Pixels which remain blank after MC are the unconnected pixels, while "multiple connection" comes from duplicate mappings after the MC.

Inside the Matlab program, the three uniquely, multiple and unconnected pixels are calculated through the motion vector in both reference and current. Uniquely connected and unconnected pixels don't show processing/selection problems, while a selection rule is defined for multiple connected to separate them into either uniquely connected or unconnected pixels. The selection rule was the

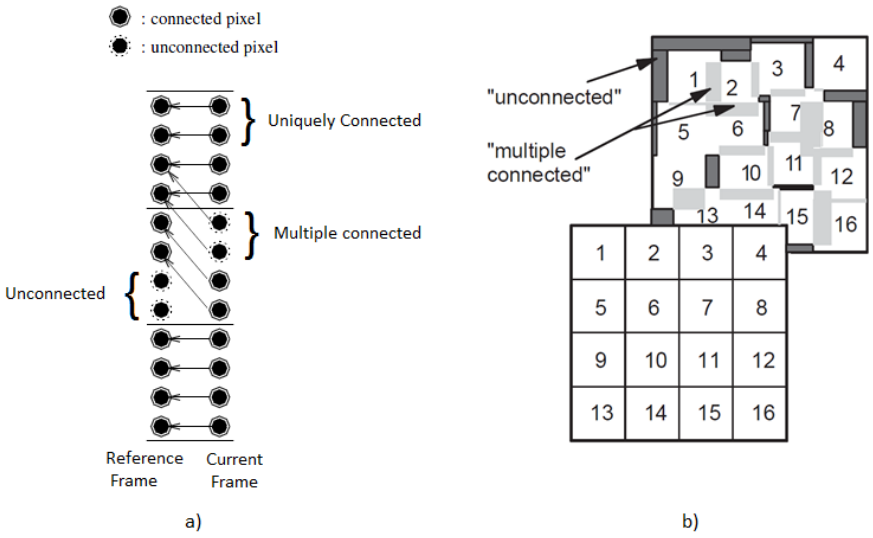


Figure 3.7: a) Uniquely connected, Unconnected and multiple-connected pixels ; b) Unconnected and multiple-connected areas in block matching. [14]

following: the smaller of the two vectors connecting to a pixel is made uniquely connected and the remaining one (the higher) remains unconnected.

Having now defined the three species of pixels, equations 3.4 and 3.5 can be applied on them to obtain the reconstruction. The result of this computation of the simulation is illustrated in figure 3.8. In this Matlab work, the cycle of equation is made just once because the program works with a single temporal level but can be applied several times to obtain more levels of temporal wavelet decomposition. The code has been written in order give as output also the reconstruction of several consecutive frames, until 16 consecutive frames, but it's not interesting for the goal of this thesis.

What can mainly be highlighted is the higher detailed result of figure 3.8(f) where pixels are not simply substituted like in the first step of the simulation, but, employing uniquely, multiple and unconnected pixels, the compression step is more accurate even if heavier. This intermediate result represent the trade-off between pure motion estimation/compensation (previously analyzed) and the full detailed compression. In one hand, temporal decomposition through Haas filter, obtained by the application of an high-pass and a low-pass filtering along the temporal axis, allows the motion compensated techniques to produce better compression performance by effectively removing temporal redundancy. In the other hand, speed performance could be penalized by the search for a more precise result.

It needs to be highlighted that the motion compensation, previously carried out, was performed using specifics of non-recent standards; the nowadays result should be far improved with respect to figure 3.5, where poor block size and search windows were used. Such weak choices has been made to the speed up of the program execution (poor hypotheses shortens the computational time) and, mainly, to better visualize the difference between filtered and not-filtered compression. In the following chapter, the study of the motion estimation is related just to the first two steps of the simulation, without exploit the concept of temporal scalability but employing the mechanism of best match of search windows between minimum SAD frames. MCTF has been introduced because it can be the beginning for future works regarding video compressing techniques.

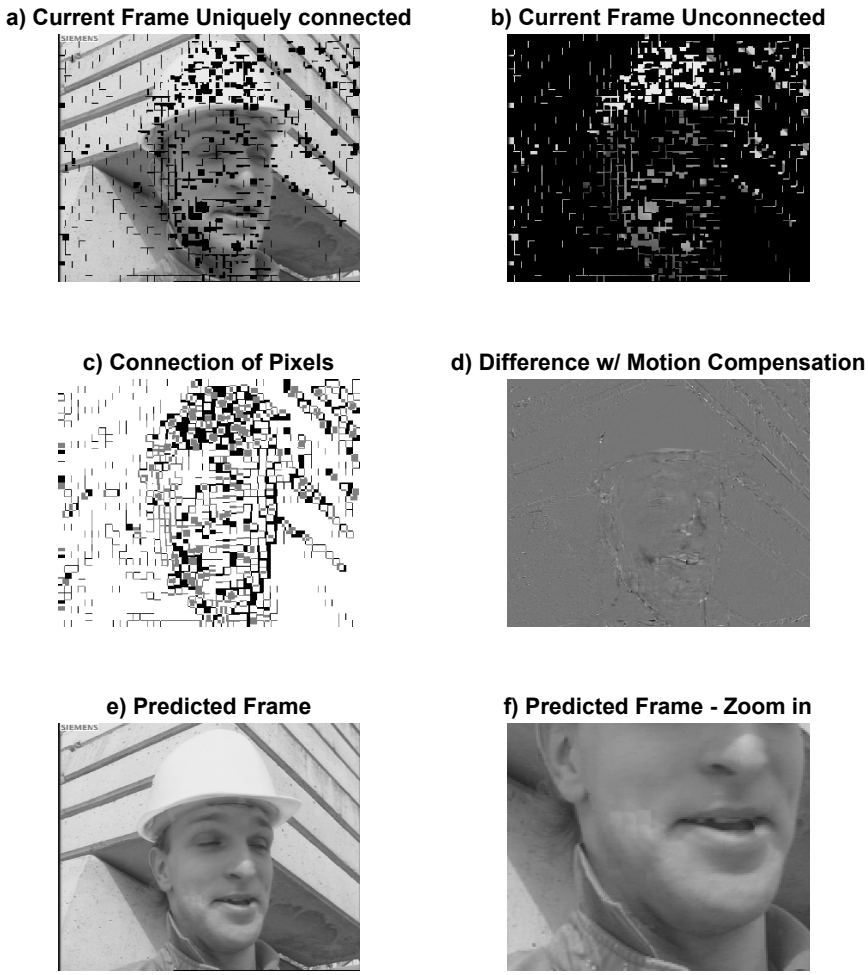


Figure 3.8: a) b) Current frame where uniquely connected and unconnected frames has been subtracted, to evidence which are the parts that will be elaborated during the compensation and, on the other hand, which the ones that are uniquely connected (equal) between the two consecutive frames; c) Connection of the three classes of pixels

Chapter 4

SAD Accelerator Architecture

HEVC encoder performs the SAD computation of the reference and current samples. In the worst case, the encoder withdraws from the memory of 4096 samples of the current Prediction Block and 4096 samples of the reference Prediction Block, carries out the difference between 4096 samples, 4096 absolute values and the 4095 additions. In software, as will be highlight in the C++ analysis of chapter 6, this set of operations is very slow, while executed sequentially. In hardware the architecture can be strongly parallelized to speed up the execution, working on more samples (pixels) at the same time. Since no data dependencies intervenes among samples, the encoder can be implemented through an adder tree: the 4096 differences and the 4096 absolute values are executed in parallel inside Processing Elements (PEs) and the outputs of the PEs are summed together. In detail, 256 pairs of samples will be elaborated in parallel, 128 for the current and 128 for the reference; in next chapters all these key features will be clarified with figures and explanations.

One of the problem that arises from a strongly parallelized architecture is the memory bandwidth limit. The memory bandwidth is responsible for throughput reduction in motion estimation, in particular delaying with high resolution/quality video, numerous frames and large search range [8]. In the worst case, the system loads from memory and works on $4096 \cdot 2 = 8192$ samples at the same time. Supposing to work at a frequency of 200 MHz, we would obtain a bit-rate equal to:

$$8192 \cdot 200 \cdot 10^6 Hz \cdot 8bits = 13.1 \cdot 10^{13} bit/s = 1.64 TB/s \tag{4.1}$$

1.64 TB/s is a very strict requirement and no memory system can support such value of memory bandwidth. Therefore, the algorithm must concern a trade-off between memory bandwidth and level of parallelization.

The designed architecture is a hardware structure which compute the SADs on all the input PBs for every input dimension, from 4x4 to 64x64. It's right to highlight that the computation of the 4x4 SAD is not allowed in the HEVC standard but the design accelerator can support the calculation. One of the strength of the architecture is to be dynamically reconfigurable because it is able to compute the SAD operation even if their dimensions change. Since the architecture is able to handle variable size matrices, it takes a variable amount of clock cycles to execute a SAD instruction that has a minimum in the 4x4 matrix and a maximum on the 64x64 matrix. The key point is that the hardware that controls the accelerator does not have to check the employed number of clock cycles but the circuit receives a "Start" command to begin a new operation and, when it finishes, a "Done" signal is raised. Hence, no problem related to the latency in terms of clock cycles, which can be different from matrix to matrix, needs to be considered. The "Done" signal will act as a sort of interrupt, so that, while the accelerator is performing a SAD operation, the Controller can do another work and, receiving the "Done" signal, it interrupts and answer to the request.

The architecture was designed to improve the performance of the HEVC encoder with particular care on its execution time and power consumption. Let's just remember that an encoder is composed of a SAD accelerator unit and a Controller unit. The accelerator is made of Datapath and memory, while the controller manages the signals and the interaction with the accelerator. In the following sections, the entire architecture is called SAD unit, formed by Datapath unit, Interface unit and a memory unit. In the next sections, all the encoder parts are going to be described. At the end of the chapter, also the Partial Distortion Elimination technique (PDE technique) is discussed.

4.1 Memory Bandwidth

The first step during the development of the accelerator was the choice of memory bit-rate. It was chosen a 50 GB/s bit-rate to maintain a common thread between this thesis work with thesis of Paolo Selvo [2], in order to have a direct comparison with the results obtained in the simulation phase and to try to derive an optimized architecture on various fronts. Moreover, maintaining the same system's specifics, this work tries to optimize the pre-existent architecture and gives the possibility in the future to improve further.

A bit-rate of 50 GB/s can be achieved employing a common SDRAM working at 1600 MHz. This frequency value is the common frequency of main memory of the majority of PCs today. Using double data rate (DDR) and a bus of 128 bit, the final throughput would be [2]:

$$1600 \cdot 2(DDR) \cdot 128 = 4.096 \cdot 10^{11} bit/s = 51.2 TB/s. \tag{4.2}$$

Equations 4.2 shows that the proposed memory bandwidth is capable of endure the calculated bit-rate value of equation 4.1. 1600 MHz is the maximum value to which the system can push, but setting the core frequency at 200 MHz ($\frac{1}{8}$ of 1600 MHz) 4096 bits can be taken from memory in one clock cycle: 2048 bits from the current PB and 2048 bits from the reference frame. In this way, 256 samples can be elaborated in parallel with a cost acceptable for consumer market. [2] Therefore, the SAD architecture core is set at 200 MHz (reading operations) because it's able to elaborate 256 samples in parallel, 128 for reference PB and as many for current PB.

4.2 Datapath

The structure of the Datapath is illustrated in figure 4.1. Trying to summarize without being repetitive, the datapath is composed of 16 PEs able to work in parallel, which receive 16 pairs of samples, 16 samples for the current frames and 16 samples for the reference, which corresponds to $128+128 = 256$ bits. The vertical orange lines, which divide the picture in 7 parts, are the pipiline stages:

1. **PIPE X**: "PIPE X" is the first step in the elaboration of the samples, it's the phase in which all the pixels are sampled, both references and currents.

2. **PIPE 0**: during "PIPE 0" the PEs perform the differences, the absolute values (sample by sample) and sums the results together in a three branches adder tree, as can be seen in figure 4.2;

3. **PIPE 1 to PIPE 4**: the "PIPE1", "PIPE2", "PIPE3" and "PIPE4" represent the adder-tree steps, the fulcrum of the encoding process. There are 4 stages to reduce as much as possible the critical path;

4. **PIPE Acc**: "PIPE Acc" is the pipeline stage related to the accumulator, designed as an adder plus a register, working at 20 bits. The accumulator is on mandatory for SAD with block

size higher than 32x16 because the samples require more than one cycle to be loaded from the memory.

The rate of the operations of the datapath is scanned by clock cycles. To each of them corresponds the computation of a macrocolumn:

- During the first cycle the PEs elaborate the first macrocolumn;
- During the second cycle the PEs elaborate the second macrocolumn and the "PIPE 1" is carried out, during which all the output of the PEs of the first cycle are summed together;
- During the third, the PEs elaborate the third macrocolumn, while "PIPE 1" and "PIPE2" stages are parallely executed;
- The process goes on until all the ref. and cur. matrices are read and elaborated.

To calculate, for example, the number of clock cycles to compute the 64x64 SAD, one has to take the 16 clock cycles to read the two reference and current matrices and adds 5 additive cycles taken by the adder-tree to sum all the results toghether. The total number of cycles the 64x64 SAD requires are 21. The multiplexer represents the last step of the SAD computation: it's used to select the adder-tree output enabled by a control signal generated by the control unit that is described in section 4.4. Therefore, at the adder tree inputs there are up to 256 values to be added. Each adder tree level adds one bit to its inputs in order to avoid overflow. At the output of the last adder there are 16 bits. If according to YUV, samples values go from 16 to 235, in the worst case of a 64x64 reference matrix which values are all 235 and a 64x64 current matrix which values are all 16 the final SAD result wants a 20 bits register.

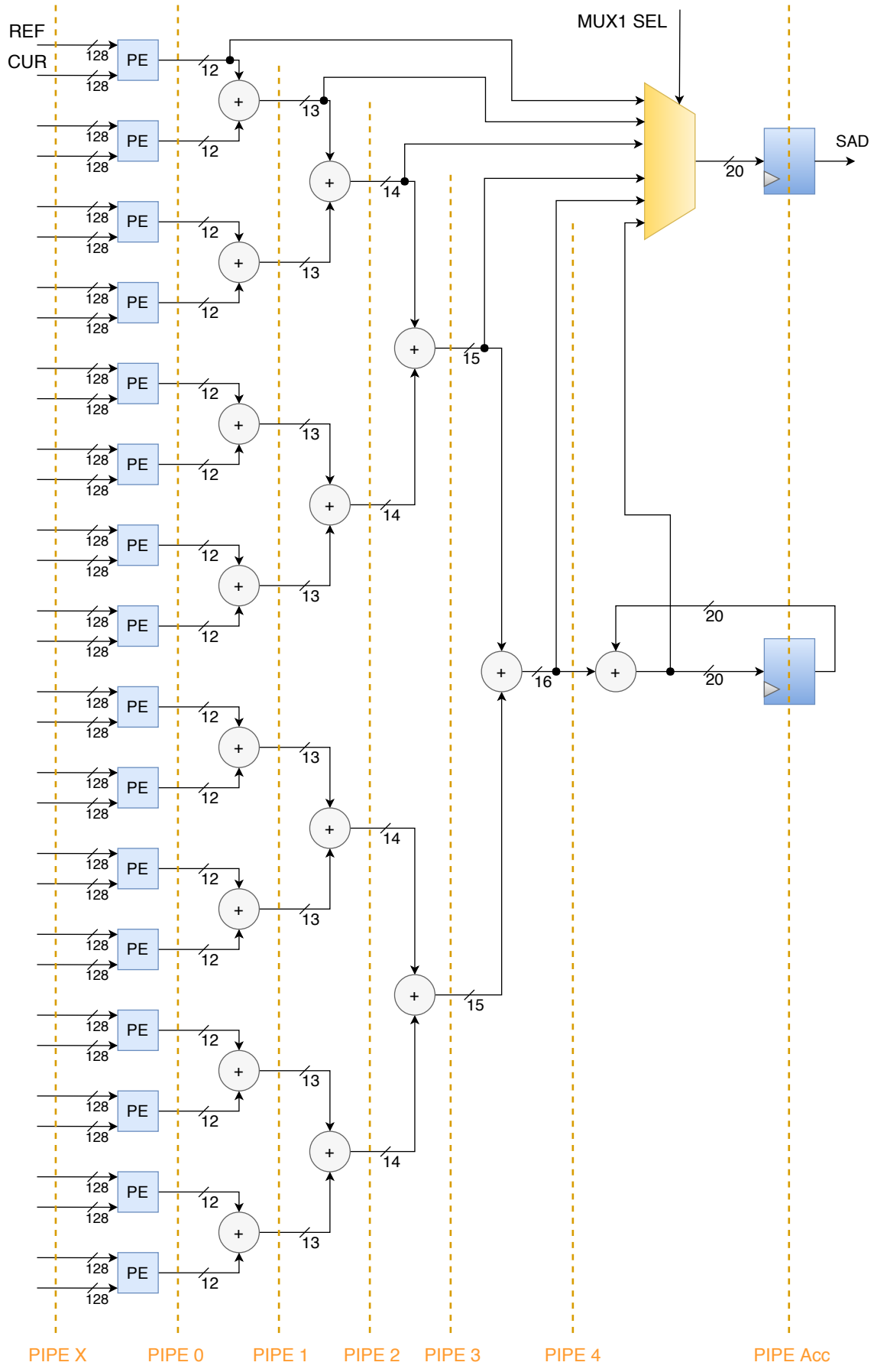


Figure 4.1: Datapath architecture. The dashed orange lines indicate the pipeline stages.

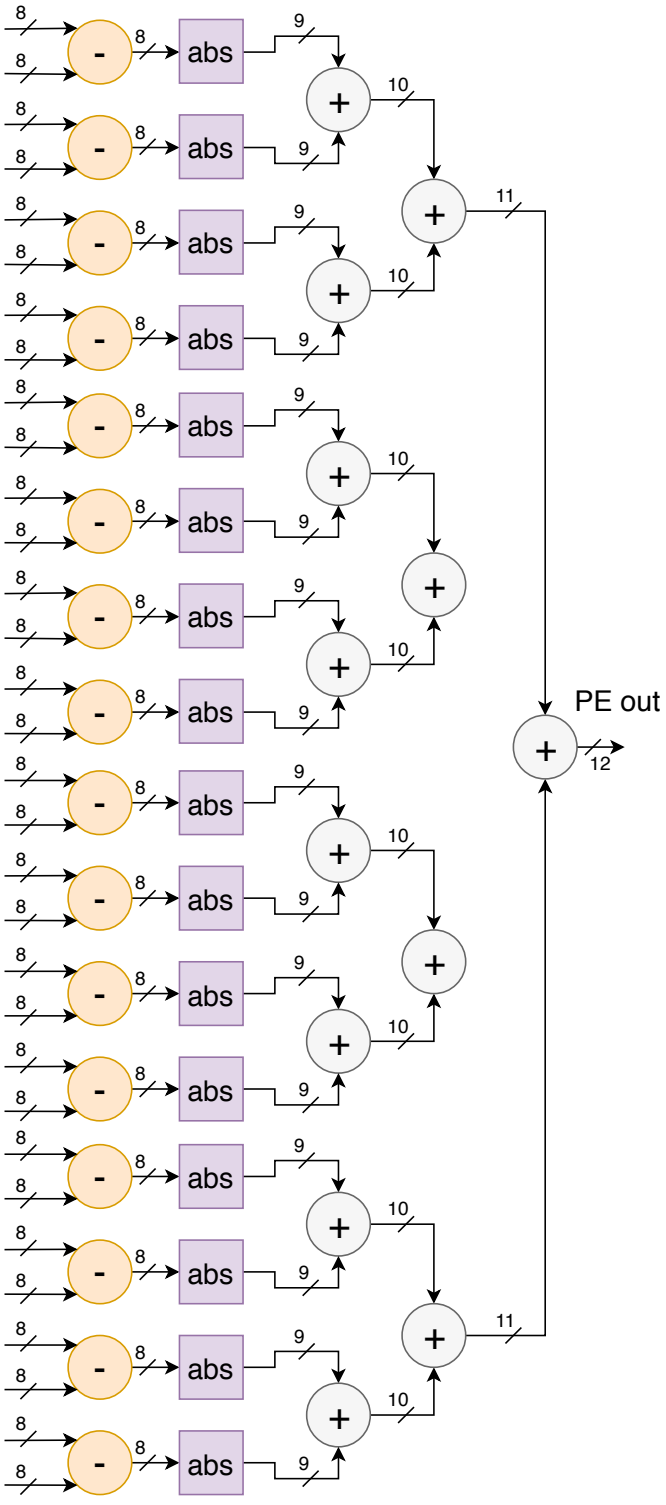


Figure 4.2: Architecture of a single processing element. The inputs to the subtractors are 1 current sample (128 bits) and 1 reference sample (128 bits). Samples are elaborated inside the PE in one clock cycle; the difference and the absolute value do not modify the parallelism.

4.3 Algorithm Description of the Architecture

The chosen memory of chapter 4.1, whose speed of operation is 200 MHz, imposes a constraint on the accelerator, which, in a maximum period of 5 ns, has to perform differences, absolute values and additions of 256 pairs of samples in parallel, inside the adder-tree.

The two inputs (inside the PEs) are the two fluxes of reference and current PBs, sequence of matrices of pixels. The two matrices have same dimensions, compliant with the standard dimensions cited in section 2.1.1. The architecture receives these matrices as a set of 4x4 elements; in other words, the rows and the columns of a matrix are combined into set of 4 consecutive elements. To better understand the drawing of the matrices, rows and columns will be grouped in elements of four 4x4 consecutive elements, called "macrorow" and "macrocolumn": one macrorow is a set of four consecutive rows and a macrocolumn is a set of four consecutive columns. Let's now consider an example to clarify this concept: a 64x64 matrix is seen by the architecture as a set of 256 4x4 smaller matrices; these smaller matrices are collected in 16 macrorows and 16 macrocolumns. A 16x4 PB will be made of 4 macrorows and one macrocolumns, a 16x16 PB as a 4 macrorows x 4 macrocolumns, and so on. In this way the 4x4 element acts as the minimum unit. If the entire architecture is taken in analysis, the elaboration of 256 pairs in parallel corresponds to working on 16 4x4 elements for the current and 16 4x4 elements for the reference frames in parallel and, every clock cycle, up to 16 4x4 elements for the first and the latter can be set as input ot the PEs. In other words, each PE is uniquely associated to one macrorow of these matrices, in the sense that receives data always from the same four rows [2]. The organization inside the memory cells follows the same logic of macrorows and macrocolumns but tries to optimize the parallelism of the accelerator. The matrices inside the memory system are written, read and sent to the datapath macrocolumn after macrocolumn, one each clock cycle: for example, a matrix with block size 16x16 is transformed in a 64x4, which corresponds to a transformation of a 4x4 macromatrix into a 16x1; during the writing phase, matrices are transformed and stored inside the memory system and, during the following transfer to the datapath, all the 16 PEs will work in parallel thanks to this new arrangement. This technique maximize the active PEs and, consequently, maximize the parallelism of the accelerator. [2]

Figure 4.3 shows how the matrices elaboration is executed.

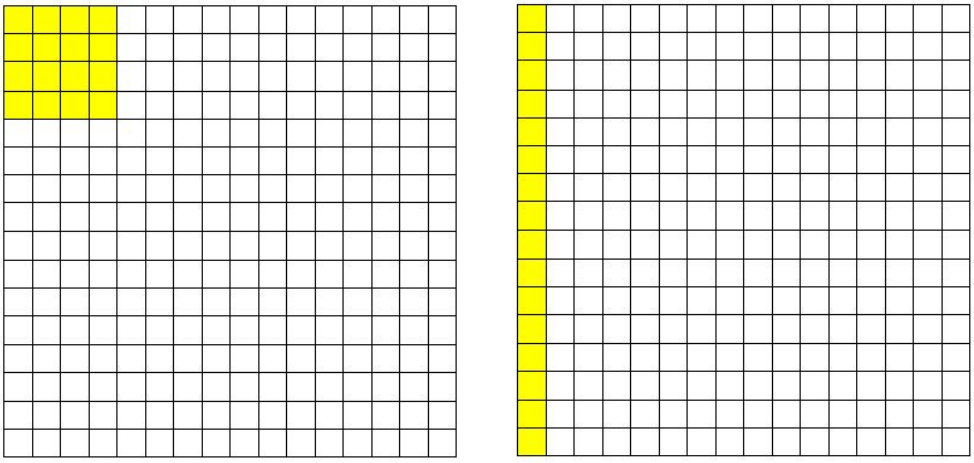


Figure 4.3: (left) 16x16 (equivalently 4x4 in macroblocks) PB without transformation inside the memory system, (right) 16x16 PB transformed into a 64x4 (equivalently 16x1 in macroblocks) to maximize the PEs usage.

One square of figure 4.3 is equivalent to a 4x4 minimum unit and, if yellow, means that the cell is occupied by one 4x4 of the PB (indifferent to either the reference or the current PB). In the example aforementioned, only 1 cycle is employed to process the 16x16 matrices in the core, because the 16x1 macromatrix (equivalent to a 64x4 matrix) occupies one single macrocolumn. If the transformation wasn't performed, the 16x16 PB would take 4 cycles to be loaded instead of one and only 4 PE would work, leaving the other 12 idle, instead of all 16 PEs as observed in the example. Focusing on figure 4.3, it can be observed how, for small block size like 16x4, some PE remains idle. The behavior of the accelerator in such cases is analyzed in the synthesis analysis in chapter 8. In reality also the 16x16 case represents a particular case, because for block sizes lower than 32x16 the accumulator remains idle. Also the 16x16 case will be analyzed in the synthesis chapter. Larger elements, 32x16 until 64x64, need more than 1 cycles to be loaded. The 64x64 PB requires 16 cycles to be loaded from the memory to the PEs because composed of 256 4x4 minimum units. Summing up, until the 16x16 PB, the loading from the memory can happen in one cycle, from the 32x16 PB upwards, the loading takes more than one cycle, neglecting for the moment the pipeline stages of the adder-tree. [2]

The adder-tree is designed to perform the sum of all the 16 parallel results coming from the difference and absolute value steps. In chapter 5, the analysis of the architecture was carried only in terms of bits, sum, difference and absolute value of bits, which can now be processed in operations between matrices and macromatrices. The structure of the accelerator is designed to process the input reference and current matrices, compute the SAD and sum, through the adder-tree, all the results. In terms of minimum units, the accelerator performs the SAD on 16 4x4 elements in parallel, giving 16 results, and, inside the adder tree, sums together the 16 results: the core is composed by 16 PEs that work in parallel on a 4x4 SAD in one clock cycle and the outputs of the PEs are summed through the 5 states of the adder-tree pipeline.

4.4 Enable signals

To manage the accelerator, a set of registers and associated enable signals are required. Looking at figure 4.1, every time an orange dashed line intersects a black datum line, there is a register and the corresponding enable signal is used to activate such register, in order to sample its input only when commanded by unit interface, described in the next section. The control unit sends 16 enable signals to the PEs, enabling just the required ones in dependence of the block size, 16 enables to the 12 bits register of the "PIPE 0", 8 enables to "PIPE 1", 4 enables to "PIPE 2", 2 enables to "PIPE 3", 1 enable to "PIPE 4" and 2 enables to "PIPE Acc" for accumulator and multiplex register. Figure 4.4 shows the position of the registers and enables inside the pipeline stages. The distribution of the register can be approximately considered as a conical distribution where the registers samples only when necessary, effectively reducing the dissipated power.

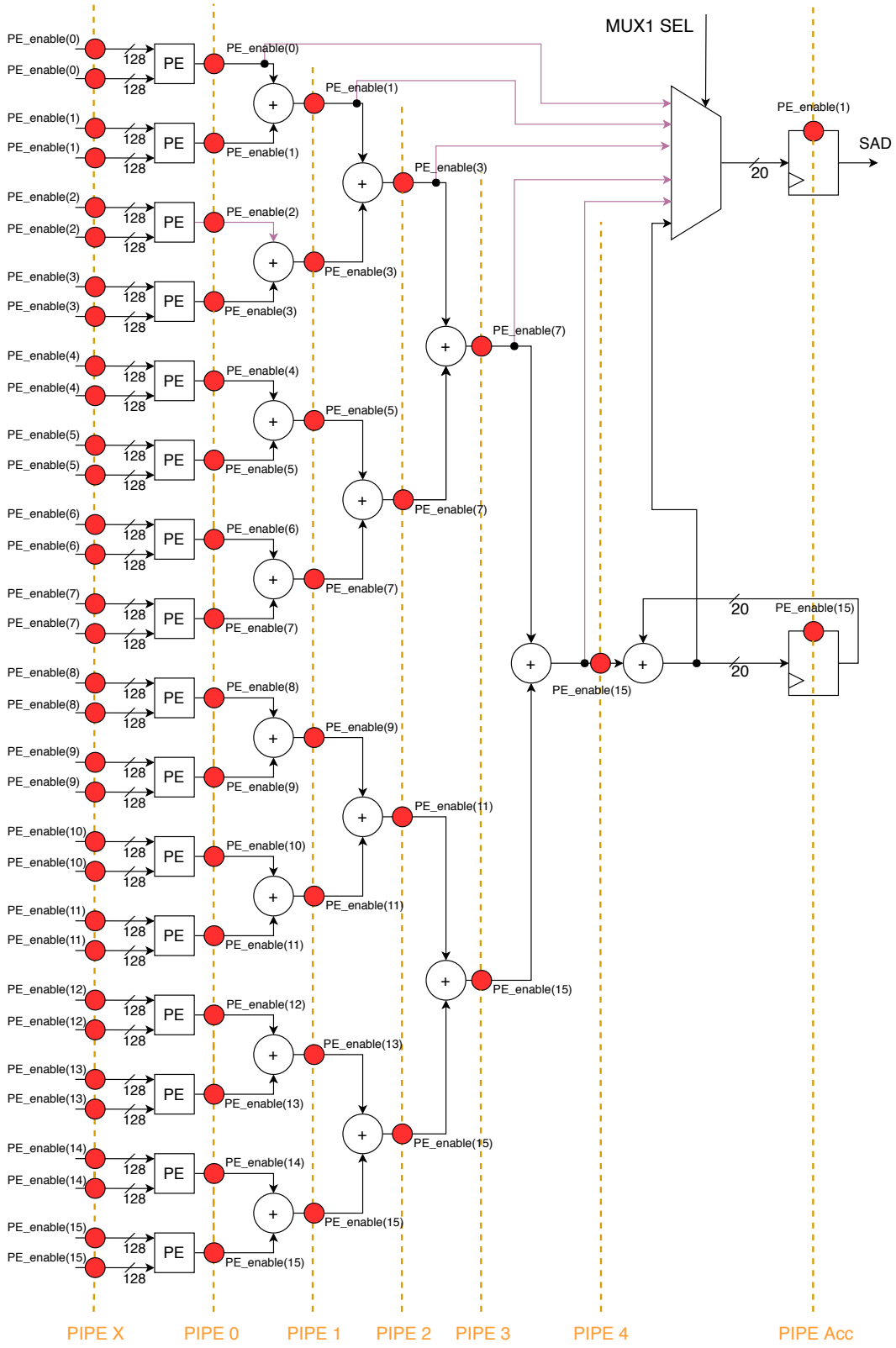


Figure 4.4: Datapath architecture with registers and PE_enable net highlighted. The vertical orange lines represents the pipeline stages and, at every stage, the red circles indicates the register, enabled through of the enable signals. It can be note that the register of the accumulator ("PIPE Acc") is controlled by "PE_enable(15)" because the use of the accumulator is needed only when all the 16 PEs are active.

4.5 Unit Interface

The unit Interface corresponds to the control logic handling the entire SAD architecture. The unit interface manages all the signal of the architecture, from the starting signal until the final signal. The logic parts, of which the circuit is made, are: a Look-Up Table (LUT), a counter, SRAM controller, a synchronous reset and a read cycles comparator. [2]

- The LUT is used to return to the architecture the number of clock cycles required depending on the number of rows and columns, to setup the multiplex selector (MUX1 SEL of figure 4.2) and to enable the PEs depending on the number of macrorows.
- The counter receives as input the "valid_in" signal, which works as a reset for the counter, and the cycles' number: it starts count and, when the cycles' number is reached, the "Done" signal is generated and remains high for one clock cycle.
- The SRAM controller activates with the "valid_in", deactivates with the "Done" or the "Read_Cycles" and it's used to select the memory cell. The "Read_Cycles" signal comes from the read cycles comparator which calculates and reports the number of clock cycles needed to read the PBs from the memory.
- In the end, the synchronous reset is the logic block that handles the reset for the datapath, depending on the "valid_in".

The control unit decision lasts exactly one clock cycle [2]. In figure 4.5 the unit interface is illustrated: it has 3 inputs, among which the valid_in behaves as the "Start" command, and 6 outputs.

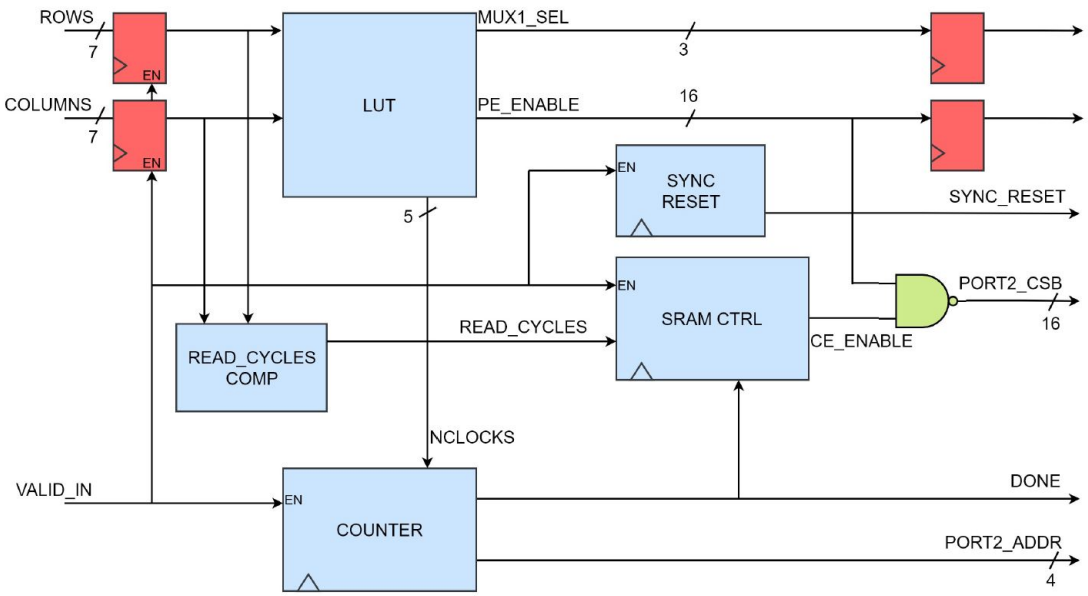


Figure 4.5: The control logic circuit, named unit interface. Inside this structure the red blocks indicate a register, while the blue blocks are the main logic components: Look-Up Table (LUT), counter, SRAM controller, synchronous reset and read cycles comparator. [2]

4.6 Memory Unit

The memory is organized starting from the association between the datapath and a macrorow because it receives data from the same 4 rows. The number of memory banks is equal to the parallelism, so to the number of PEs (16). Of course, the memory is divided in a unit for the reference and one for the current frames, both containing, in the worst case, 64x64 samples (128 bits each). Reading and writing are controlled by the control unit. The memory can enable in parallel up to 16 banks, if they share the same address, corresponding to the reading/writing of up to 4096 bits in a single clock cycle (2048 for the reference and 2048 for the current). This is equivalent to read an entire reference and/or current PB. The structure of the memory is illustrated in figure 4.6.

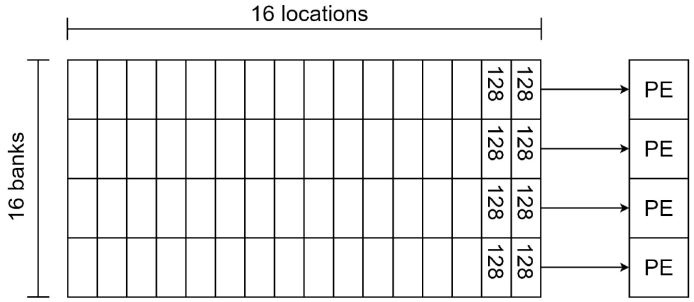


Figure 4.6: Memory banks organization associated with the number of PEs. For sake of simplicity, only 4 banks are here reported. Each location contains 128 bits. [2]

It can be highlight that the current PB remains the same for a set of SAD operations which belong to the same best match search. In the opposite way, the reference PB needs to be changed more frequently, not entirely but just partially in dependence of the window search. the tasks of this memory correspond to those of a cache memory with very high output parallelism, so it must be an internal chip memory. [2]

The system's architecture is arranged in 3 memory units, two for the reference PB, one for the current PB. Each memory unit is a dual port SRAM memory, one port for the writing and one for the reading operations [2]. The reading memory port is handled by the SAD architecture core, which has a frequency of 200 MHz as discussed in section 4.1 (maximum memory bandwidth of $4.096 \cdot 10^{11}$ bit/s, 1600 MHz as maximum frequency). The second port is used to write the reference PB for the next SAD operation and it can be pushed to a frequency of 1600 MHz, in order to provide the next data in the shortest possible time (the frequency is 8 times higher). This design decision allows to overcome the bottleneck given by the writing and reading of the reference PB during the motion estimation, because the reference PB is updated before every SAD operation. In this way the SAD accelerator is always active, every clock cycle, without dead times, and the control unit can enable the writing (slower) and reading (faster) operations at the same time. The only constraint is to fix the ratio between the two reading and writing clock regimes of 8.

Thanks to the transformations of section 4.3, the 24 rows-columns combinations of the input matrices are reduced to 14, operations managed by the control unit. The 14 cases, which will be stored inside this memory system, are the following: 8x4, 8x8, 16x4, 16x8, 16x12, 16x16, 32x8, 32x16, 32x24, 32x32, 64x16, 64x32, 64x48, 64x64. The employed SRAM is the same employed in the article of Paolo Selvo [2]. It is an IP of the library used for the synthesis (CMOS 65 nm standard cell technology): it has a VHDL behavioral description provided by the library provider. To further informations, address to ref.[2].

4.7 SAD Unit Structure and Clock Regimes

The SAD unit structure is illustrated in figure 4.7: it consists in the overall architecture, what has been called up to now hardware accelerator.

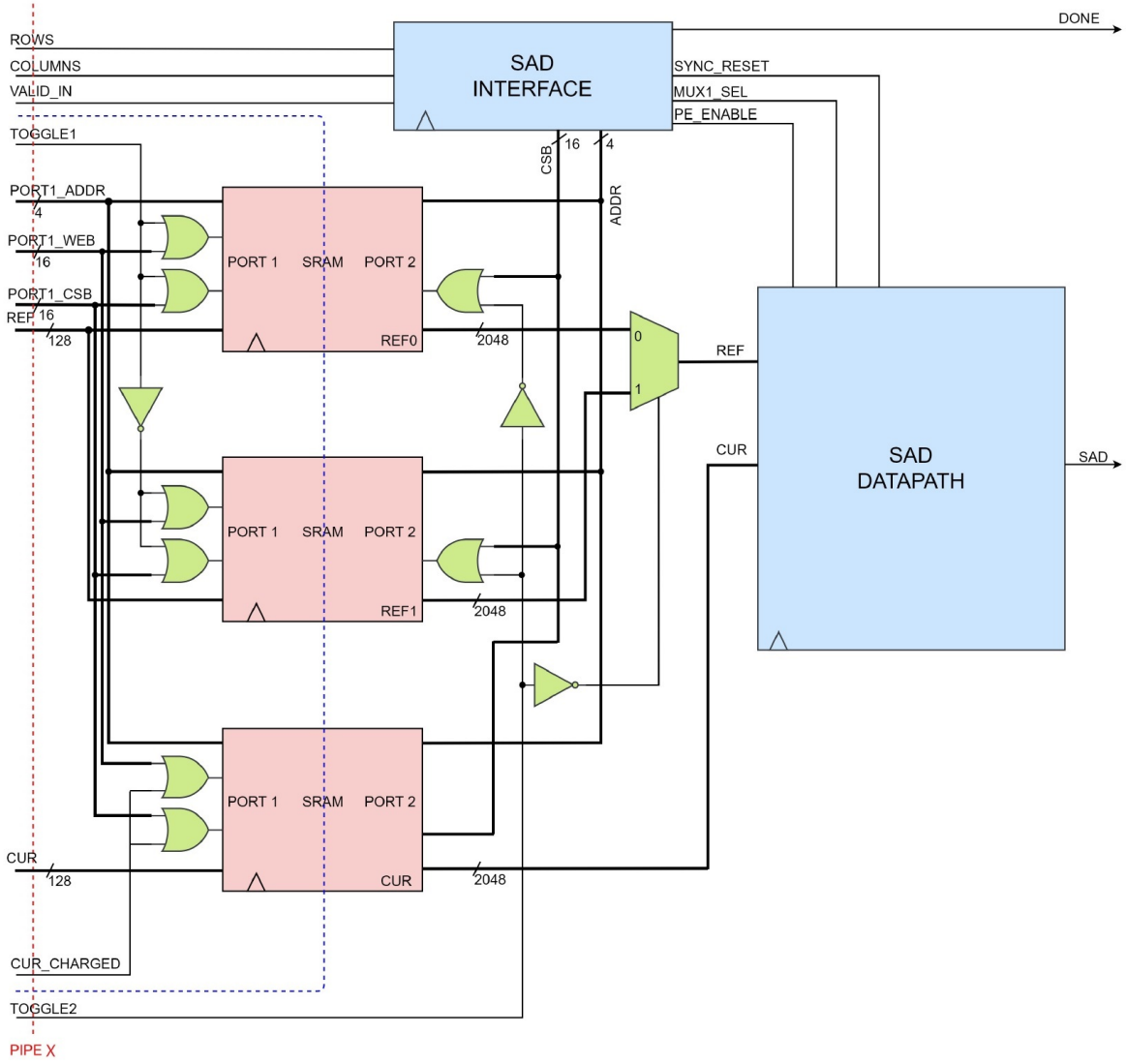


Figure 4.7: SAD accelerator clocks regimes: the vertical red line represents the first pipeline stage, while the blue line separates the signals into the 200 MHz regime (left) of port1 and 1600 MHz regime (right) of port 2. [2]

Figure 4.7 shows not just the general architecture of the accelerator, unit interface, datapath and memories, but highlights the begin of the pipeline stage and the two clocks regimes. The dashed red line is for the pipe, while the blue line is for the regimes. The pipeline registers all the inputs coming from the left into the accelerator. The blue line divides the system in two, the left part (ports 1) contains the writing ports and the clock regimes reaches the 1600 MHz; the left part (ports 2) contains the reading ports and works at 200 MHz ($\frac{1}{8}$ of the left part). All the informations regarding

input signals, units and outputs are exhaustively described in the previous section of this chapter. Figure 4.7 postulates the reset signal, which is used to initialize the accelerator and set the register to default value.

4.8 Introduction of PDE technique

From the beginning of chapter 4 until section 4.7 the SAD unit was presented. In this section, the basic SAD unit is enhanced with the optimization given introducing the Partial Distortion Elimination (PDE) algorithm, whose main task is to speed-up the SAD execution. This is one of the most critical step, because the primary structure, which will be analyzed at RTL level and through the synthesis, is the SAD encoder implementing the PDE technique and to which approximate adders will be replaced.

The PDE algorithm starts from the fact that, during the best match research of the Motion Estimation, if an adder returns a result greater than a SAD value previously computed, there is no reason to keep going in the adder-tree computing the final result, because will surely be greater than the final SAD result. In other words, it's useless to go on in the computation of a SAD if one of the adders inside the datapath gives an output value that is already greater than a SAD final value calculated before, because the greater SAD result won't be the best match [2]. If the computation goes on, the consequence is a useless expenditure of power and time. The PDE technique intervenes in these circumstances adding a set of comparators in the adder-tree which compares the actual SAD value with the minimum value calculated up to that moment. If a greater value is generated, the computation is stopped, the "Done" signal is raised and a new SAD operation starts. All these steps are managed by the control unit.

Figure 4.8 illustrates how the PDE has been implemented in the architecture of figure 4.1. Five comparator rectangles (COMP) are inserted in the adder-tree. These comparators indicate that a majority comparator is connected to the output of the adders. One important comparator among the five added is the one next to the accumulator which compares the minimum SAD with the partial SAD result.

The activation of the comparators depends on the block size of the reference and current PBs. For example, a 64x64 PB has 16 macrocolumns which take 16 cycles to be sampled, so, the accumulator comparator will sample the SAD result 16 times. In this situation, the other four comparators are not used because they would compare the temporal minimum with the result exiting the adders that are surely smaller with respect to the typical value of a great size SAD result. One one hand, the four comparators gives results that are approximately 1/128 of a great size SAD result [2]. On the other hand, in the accumulator comparator the value reached after one macrocolumn is typically 1/16, after two is 2/16, and it quickly increase: it's possible that after 9-10 macrocolumns the obtained SAD result is already grater and so discardable. This implies a save of 7-6 clock cycle in the SAD computation. The four comparators are specialized with smaller PB sizes, in particular smaller than 32x16.

Let's now list the changes, improvements and scripts' variations of the new design implementing the PDE:

- The two OR gates, the multiplexers and the priority encoder are added to warn the control logic that one of the result of an adder is greater than the temporal minimum and that the operation can be interrupted.
- Another aim of this circuitry is to put on the SAD output bus the value that activate a comparator: the priority encoder is used when a value larger than the actual minimum is received by more than one of the four comparator (in the third pipeline stage, 14 bits branches) and the multiplexer selects the adder output as commanded by the priority encoder.

- The last comparator, put beside the accumulator, is useful for big SADs, from 32x16 upwards, as briefly described for the example of the 64x64 matrices. For smaller SADs it's not used because they exit before.
- It was demonstrated than is needless to insert other majority comparators inside this datapath in the other pipeline stages.
- The only change in the Interface unit is related to the counter because an additive input is needed, the "MIN" signal, which stop the counting and reset it.
- In the Matlab script, a function that calculate the minimum value among the SAD results is introduced. The scripts creates a new file the value of the minimum SAD.
- From the RTL design point of view, the VHDL testbench has been modified. A new control is added which integrate the PDE. If the "PDE" signal is high, the technique is included in the simulations, if not the accelerator is tested without PDE. The modifications applied to the testbench, while PDE is included, are discussed in chapter 7.2.
- In ref.[2], a speed-up analysis is carried out, highlighting the improvement obtained by using the Partial distortion Elimination. The most interesting results are the improvement for 64x64 block size (the worst case), which shows a speed-up of the 26.5%, and for the 32x32, which shows a speed-up of 13.5%.

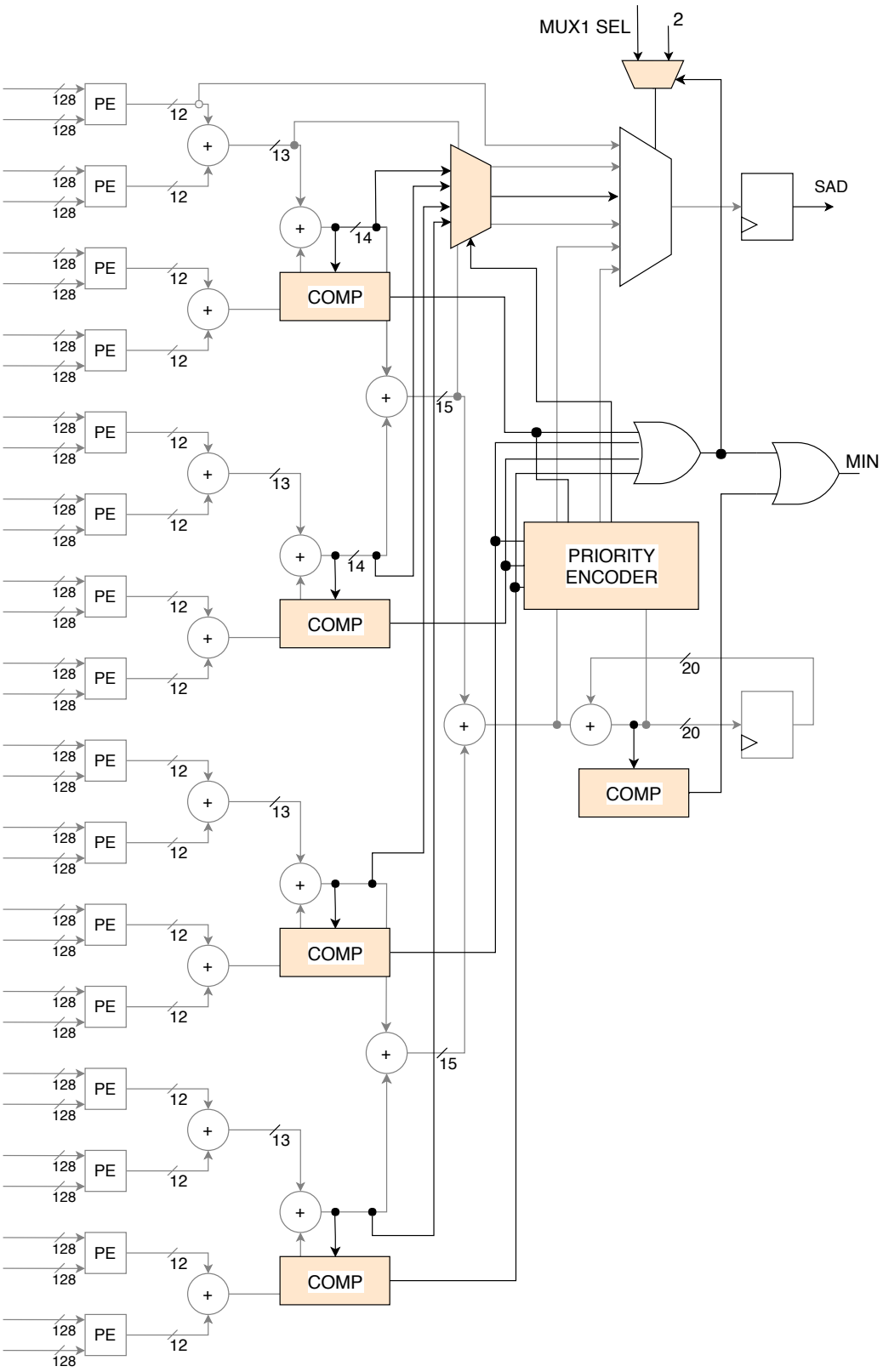


Figure 4.8: The new designed Datapath of the architecture including PDE.

Chapter 5

Approximate Computing

Among the video compression algorithms, approximate computing has become an emerging paradigm to reduce the power/energy consumption of the system and to reduce the computational load of a compressing procedure. It exploits the resilience of the applications to speed up the data processing process of the system and find out a trade-off between output quality with efficiency.

The main task of approximate computing is to relax the signal processing: considering the tolerance of computing imprecision due to human eye imperceptibility, highly power-efficient implementations can be realized [16]. The topic has attracted the interest of industrial groups (Intel, IBM and Microsoft [17]), many research groups and communities, because it's able to develop highly efficient encoding algorithms, reducing the general consumption and the complexity of the calculations, taking advantage of the limited sensitivity and susceptibility of human eye. If the amount of computing imprecisions of an image or a video is limited, the result is a negligible amount of perceptible visual change of the output sequence. Of course, the error introduced by the approximate arithmetic must not exceed a certain threshold, or the reduction in resolution can ruin the visual experience. The choice of the application field becomes critical in this phase and the best compromise between performance and tolerance suitable for the application needs to be considered. For instance, approximate computing could fit with several image and video processing applications, which today proliferate in entertainment, or even applications for automotive, security, and communication industries. [17]

The main task of this thesis work, and the main argument of this chapter, is the substitution of approximate adders instead of accurate adders, inside the best match algorithm of the motion estimation. In detail, the replacement takes places inside the sum of absolute difference (SAD) between current and reference frames to determine the macroblocks' best match. The expectations from the implementation of approximate adders inside the SADs computation would reveal a relaxation of the strict Boolean/Numeric equivalence of the hardware which compute the encoding process. The relaxation causes, on the one hand, an accuracy loss (resolution reduction, precision decrease and quality lowering) but, on the other hand, it improves the system in terms of area, power-energy and performance efficiency.

5.1 Approximate Arithmetic in the Motion Estimation

The motion estimation (ME) process is one of the most expensive in terms of power consuming, energy requirements and design resources in the video encoding. It has been calculated that the ME for different block sizes consumes more than the 50% of the overall energy consumption of a HEVC encoder, data confirmed in both ref.[2] and ref.[17]. In particular, the most demanding process of ME is the SAD computation. Nevertheless, ME shows a certain level of resilience or elasticity for small

computational errors [19]. Since ME consists in the search of the best match between the reference and current blocks, which can be defined as the search of the previous most similar block to the block under analysis, the use of an approximate and not perfectly accurate computation should not cause any inconsistence in the encoding [19].

As discussed in chapter 1.4, the motion estimation phase and the related algorithm to eliminate the temporal redundancy between frames by exploiting temporal correlation is a very active topic among the research groups and academical works; the implementation of ME in an encoding process results in the raise of the complexity of the encoder, with a corresponding improvement in the performance, and most of the works tries to reduce this global complexity. One of the possible approach, the approach discussed at the beginning of chapter 5, is the employ of approximate adder in the computation of the SAD. Been the most demanding step of the ME, a reduction in the power/energy consumption can highly influence the overall performance of the system, with minimum impact of the efficiency.

5.2 Approximate Adders

The introduction of the approximate computing techniques in the encoder algorithm is possible substituting approximate adders to the accurate adders inside the SAD accelerator design. Video compression allows error-tolerant applications, which can be translated in a loss of the video quality to realize an improved low-power energy-efficient design, able to reduce the power consumption, the area, the delays, the critical path and the SAD computational cost of the system.

Focusing on the SAD computation, the SAD accelerator needs to find the best match value for a given input sequence between current and reference frames or, in other words, find the reference Prediction Block showing the minimum value of SAD with respect to the current Prediction Block. Introducing an error, the SAD cost function has not to mandatorily produce precise results but it has a specific error tolerance. It accepts an error in two possible situations:

1. as long as the best match result remains constant and continues to be correct; in this way, the video is not affected and the outputs from the coding procedure don't change;
2. if the best match is wrong but the level of approximation is high enough to ensure a performance improvement and doesn't reduce too much the video quality, translated as acceptable mismatch between motion vectors or acceptable error distance (the two topics that will be the theme of the chapter 6).

To deeply study the accelerator behavior, a crossed study was carried out, analyzing both precise and approximate adders in order to verify not just the approximation level with consequence improvements introduced by the approximate adders but also to test the adaptability of any kind of adder to the design. Furthermore, the position, where the adders were substituted, can not be neglected: a related chapter is dedicated to show the three chosen substitutions, with related hypothesis of the impact on the design which the substitutions could have.

Precise adders: The study of the SAD accelerator required both perspectives: accurate computation and approximate computations. The accurate computation can be simply carried out using the "+" sign and letting the software compile it automatically, but it's interesting to analyze the difference between automatic implementation and "user written" code implementation. In particular, the behavior of the two codes especially varies in the synthesis phase, where the accelerator is, theoretically, synthesized automatically with ripple carry adder for weak timing constraints and carry look ahead for stronger limits. In fact, the task of consider precise adder is this: verify the correspondence between automatic implementation and not automatic one. Ripple carry adder (RCA) and carry look ahead adder (CLA) are the two selected adders for the precise computation, mainly because these are

the two possibilities provided inside the libraries used in the synthesis phase.

Approximate adders: The selection of the approximate adders to be inserted in the SAD accelerator for the motion estimation of the video compressing is one of the most critical and important step of this work. The choice of these adders has been carried out through the literature, investigating most of the possibilities. In brief, approximate adders can be classified in different classes [18]:

- Speculative adders,
- Segmented adders,
- Carry-Select adders,
- Approximate full adders.

The speculative adders exploit the concept of longest sequence the propagate signal (longest sequence of '1' inside the propagate signal p) for the parallel computing of long binary sequences. The segmented adders divide the bit sequence in several parts, sum the separated portion and, finally, add together the results, giving precedence to the most significant bits (MSBs). The carry select adders consist of several sub-adders which execute the sum for both carry-in equal to '0' and equal to '1' in parallel, and select the wanted result. In the end, the approximate full adders divides the n-bit inputs into 2 parts, the MSBs part is precisely computed, the least significant bits (LSBs) are approximated and generate no carry. To uniformly select the adders, one per speculative, segmented and carry-select has been chosen and two from approximate full adders.

The five approximate adders are:

- ACA: Almost Correct Adder - Speculative adders class;
- ACAA: Accuracy Configurable Approximate Adder - Segmented adders class;
- ETA-I (ETAI): Error-Tolerant Adder I - Approximate full adders class;
- LOA: Lower-OR Part Adder - Approximate full adders class;
- SCSA: Speculative Carry Select Adder - Carry-select adders class.

The final choice is fallen on these five possibilities of approximate adders, trying to have a complete perspective of the impact of approximate adders in the accelerator, completely characterizing the system in all the possible combination and considering that an increasing level of error is proportional to a rise in the performance and viceversa.

The following subsections will explain step by step the behaviour of the seven architectures, commenting improvement introduced in the critical path and timing, in parallel to the error of the approximation.

5.2.1 Ripple Carry Adder (RCA)

A series of n one-bit full adders can be cascaded to execute the sum of two n-bit numbers. The Ripple Carry Adder (RCA) is a chain of n one-bit full adders in which the carry-in of the i^{th} full adder is the carry-out of the previous full adder ($(i-1)^{th}$). An example of 4-bit RCA structure is described in figure 5.1.

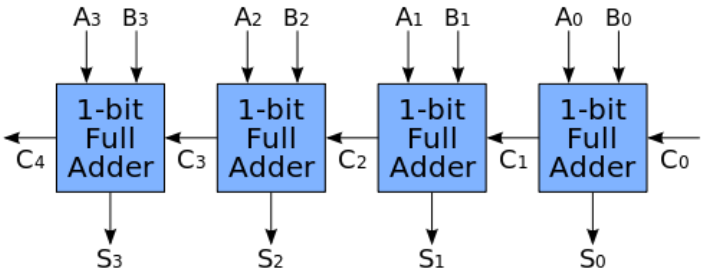


Figure 5.1: 4-bit Ripple Carry Adder

Considering two integers a and b, the general equations describing a full adder are the following:

$$s = a \oplus b \oplus c_{\text{in}} \tag{5.1}$$

$$c_{\text{out}} = (a \cdot b) + (c_{\text{in}} \cdot (a \oplus b)) \tag{5.2}$$

where s is the sum, c_{in} and c_{out} are the carry-in and carry-out. The RCA is called this way because it ripples every carry bit to the next full adder. Even if the linear and simple design allows fast design time, the RCA is quite slow because every full adder must wait for the carry of the previous step. The total delay can be calculated: every full adder requires 3 levels of logic, so the delay for a RCA is:

$$T_{\text{RCA}}(n) = (n-1) \cdot T_{carry} + T_{sum} = (n-1) \cdot 2 + 3 \tag{5.3}$$

For example, a 16-bit RCA has a delay of 33 gates delays of critical path. In an n-bit RCA, the carry propagates from one 1-bit full adder to the next, thus the delay of RCA grows in proportion to n (or $O(n)$). The RCA is one of the most diffused example of adder for small and medium architectures, with limited number of parallel computation.

5.2.2 Carry Look-Ahead Adder (CLA)

The Carry Look-Ahead Adder (CLA) represents an evolution of the RCA, in particular in terms of computational time. A CLA improves the system's speed, reducing the time related to the carry bits definition. An example of 4-bit CLA structure is shown in figure 5.2.

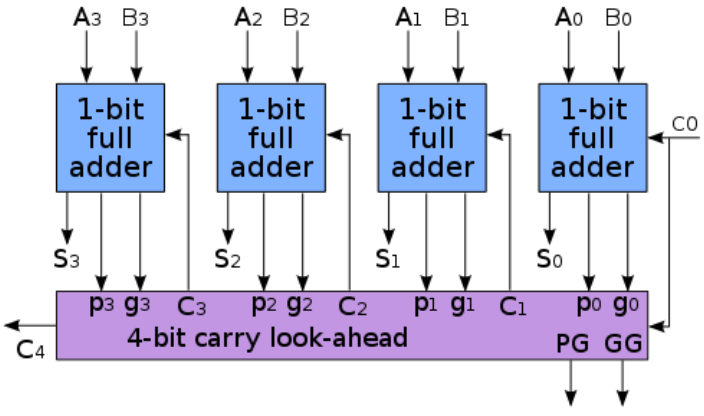


Figure 5.2: 4-bit Carry Look-Ahead Adder

To describe how does it works, three signals (p, g and k) are introduced. These three signals represent the propagate, generate and kill signals:

- Propagate signal: equal to '1' when at least one input is a '1' but not simultaneously; this is equivalent to 'XOR' gate: $p_i = a_i \oplus b_i$.
- Generate signal: equal to '1' if both inputs are '1' and it's equivalent to a "AND" gate: $g_i = a_i \cdot b_i$.
- Kill signal: equal to '1' if both inputs are '0': $k_i = \overline{a_i + b_i}$

The carry and the sum are calculated as expressed in equation 5.4 and 5.5:

$$s_i = p_i \cdot c_i \tag{5.4}$$

$$c_{i+1} = g_i + (p_i \cdot c_i) \tag{5.5}$$

Given two numbers to be summed, the CLA takes every pairs of bits and determine if the bit couple generates or propagates a carry, in a so-called "pre-process" which allows to save time. In fact, no delay is accumulated while waiting for the ripple-carry of the previous steps, like in the RCA.

Let's analyze, as an example, a 4-bit CLA, the same represented in figure 5.2. Starting from 0 to 3, the carry signal results to be:

$$C_1 = G_0 + P_0 \cdot C_0$$
$$C_2 = G_1 + P_1 \cdot C_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$$
$$C_3 = G_2 + P_2 \cdot C_2 = G_2 + G_1 \cdot P_2 + G_0 \cdot P_1 \cdot P_2 + C_0 \cdot P_0 \cdot P_1 \cdot P_2$$
$$C_4 = G_3 + P_3 \cdot C_3 = G_3 + G_2 \cdot P_3 + G_1 \cdot P_2 \cdot P_3 + G_0 \cdot P_1 \cdot P_2 \cdot P_3 + C_0 \cdot P_0 \cdot P_1 \cdot P_2 \cdot P_3$$

Substitution these results in the s_i bits, the final sum can be calculated. From previous Boolean expressions, it can be highlight that c_i doesn't depend or need to wait for c_{i-1} but, oppositely, c_i is propagated concomitantly with c_{i-1}, c_{i-2}, etc. Therefore, an n-bit parallel adder can be implemented with the CLA scheme to improve the performance and speed-up a binary addition. The delay of CLA is logarithmic in n (or $O(log(n))$), which is a value of delay far smaller than the delay shwon by RCA.

5.2.3 Almost Correct Adder (ACA)

A precise adder, such as CLA, generally shows a logarithmic delay. Sub-logarithmic delays requires approximations. Ajay K. Verma et al. [20] designed a fast unreliable adder able to provide correct results for the vast majority of input combinations. As introduce for the CLA, the addition of two n-bit integers a and b follows the equations 5.7 and 5.6:

$$s_i = a_i \oplus b_i \oplus c_{i-1}. \tag{5.6}$$

$$c_i = \begin{cases} 0 & \text{if } k_i = 1 \\ 1 & \text{if } g_i = 1 \\ c_{i-1} & \text{otherwise } (p_i = 1) \end{cases} \tag{5.7}$$

where s_i is the i^{th} bit sum, c_i is the i-th carry bit, p_i, g_i and k_i are the propagate, generate and kill signals, shown in equations 5.8 to 5.10:

$$g_i = a_i \cdot b_i \tag{5.8}$$

$$p_i = a_i \oplus b_i \tag{5.9}$$

$$k_i = \overline{a_i + b_i} \tag{5.10}$$

It can be highlighted that the c_i depends on c_{i-1} only if the p_i is true; in the other cases the carry bit will not depends on p_i but on kill or generate signals. This concept is also true for c_{i1} which depends on c_{i-2} only when p_{i-1} is true. When p_i and p_{i1} are true, c_i directly depends on c_{i-2}. In the case of three consecutive true p_i, p_{i-1} and p_{i-2}, c_i will depends on c_{i-3}, and so on. This dependency can be generalized asserting that c_i will depend on c_{i-k} only if every propagate p signal between bit position i and i-k+1 (inclusively) is true. [20]

If it was possible to know the longest sequence of propagation signals, the adder could jump a portion of bits during the computation of the carry, because the carry bit c_i will surely be independent of the previous bit position that simply propagates. Thence, also the sum s_i will only depends on the sum bit before the beginning of the propagate chain. For example, let's suppose to sum two 20-bit integers and let's suppose that the longest sequence of p is 4. c_i at position 'i' will be independent of c_{i-5}, while s_i can be calculated using just the input bits of six preceding bit positions starting from i^{th} bit position. [20]

This last is the starting point for the design of the ACA. It has been proved in [20] and [21], that the length of the longest propagate sequence can be calculated as $ln(n)$, where n is nothing but the bit-width. The ACA consists of $ln(n)$-bit adders, computing sum and carry in a precise position of the input numbers in parallel; the configuration is shown in figure 5.3. The advantage of such configuration is a strong reduction of the critical path and delay: taking the previous example, the delay of a 20-bit adder, using ACA, becomes the delay of a 6-bit adder.

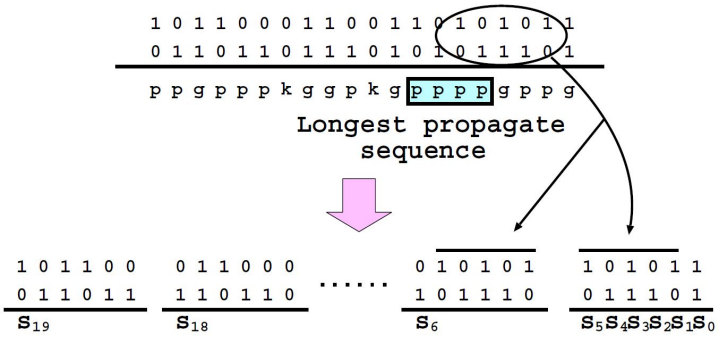


Figure 5.3: Configuration of the ACA for two 20-bit integers examples and the corresponding sum signal at each bit position. As briefly described, setting to 4 the longest propagate signal, the sum depends only on the inputs bits of 6 preceding bit positions. [20]

The error sources in this adder are related to very long propagate sequences. Always considering the previous example, an error will occur if a and b (integers) generate a corresponding p signal with seven or more consecutive ones. The error produced in this way, as can be seen from figure 5.3, will influence just the last bit of the nominee sum. This cannot be dangerous for the resolution of the compressed video/image and cannot result in a strong shift of the motion vector, but, at the same time, it's a non negligible error source. Probably, the value of the error, generated from the SAD accelerator implementing the ACA, will be small but the performance will not benefit too much from this approximate adder. In the implementations realized in the next chapters, considering that the maximum bit-width is n = 20, the supposed longest propagate signal will be $ln(20) \simeq 3$ bits, the

dimension of the adders in parallel will be 5 (a set of 5-bit adders), meaning that the delay and critical path of a 20-bit adder is reduced to a 5-bit adder. In an n-bit ACA, k LSBs are used to predict the carry for each sum bit ($n > k$), therefore, the critical path delay is reduced to $O(log(k))$. [18]

5.2.4 Accuracy Configurable Approximate Adder (ACAA)

The structure of the ACAA is represented in figure 5.4, in the case of the 16-bit.

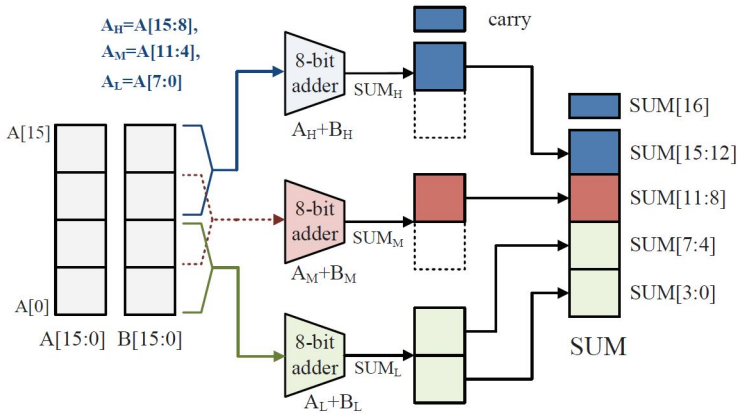


Figure 5.4: Proposed architecture for the ACAA approximate adder in the 16-bit case. [23]

The adder is design to be a trade-off between accuracy and performance. In a n-bit addition, the expensive part in terms of time and speed is the carry chain computation. The task of the ACAA is to cut, or rather, reduce the critical path delay to improve the performance, increase the clock frequency, lower the power dissipation and the operating voltage: the sum is divided into three sub-adders (MSBs, LSBs and middle sequence) which generate accurate results of partial sum. The middle sub-adder is used not to lose the accuracy during the calculus and to avoid critical error rate for long sequence of propagate signal. The 16-bit case can be, of course, generalized in the implementation of n-bit adder: with a parameter k, which represents the bit-width of each sub-adder result, the adder is divided into $[\frac{n}{k} - 1]$ 2k-bit sub-modules [23]. In the case of 16-bit addition, k is equal to 4 and there are $[\frac{16}{4} - 1] = 3$ 8-bit modules, as confirmed by figure 5.4.

In modern designs (such as CLA), the critical path is proportional to $log(n)$. The proposed ACAA has $(N/k - 1)$ sub-adders, each of which is a 2k-bit adder and, therefore, the delay related to the critical path grows in proportion to $log(2k)$, it's an $O(log(2k))$. It's higher than the ACA case but still lower than the accurate proposed architectures.

5.2.5 Error-Tolerant Adder I (ETAI)

Error tolerant adder is the first of the two approximate full adders applications. The operands are split in two parts: an accurate part, the MSBs, which includes the majority of the bits and an approximate part, the LSBs, which includes the remaining bits. The split can be positioned in the middle of the input sequences or, as explained in section 5.2.3, takes the first $ln(n)$ LSBs of the input numbers. The addition of the accurate part is performed from right to left and the RCA rules are applied (precise sum), while the LSBs require a specific approach. No carry signal is generated in order to eliminate the carry propagation path and reduce the delay. The mechanism for the LSBs elaboration consists of:

1. checking of the LSBs position from left to right;

2. if the input bits are both equal to '0' or only one of the two equal to '1', the ordinary sum is performed;

3. if both inputs are true (equal to '1'), the checking process is interrupted and all the subsequent bits of the sum are set to '1' (from the couple of '1' to the right).

The working scheme is also illustrated in figure 5.5. Supposing l the number of LSBs, the critical path for this type of application can be approximated to $O(log(n-l))$.

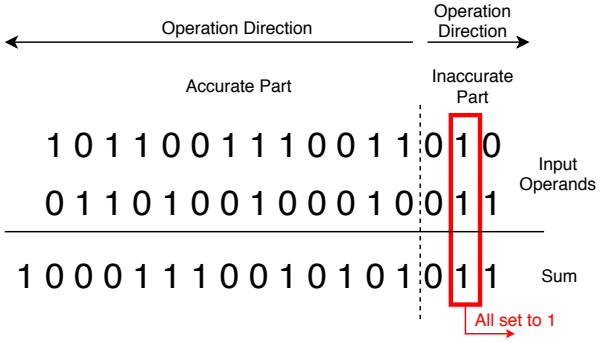


Figure 5.5: Error Tolerant Adder Type I (ETAI) - working scheme. [24]

5.2.6 Lower-OR Part Adder (LOA)

Lower-OR Part Adder divides a n-bit addition into two m-bit and d-bit smaller parts, where $m+d=n$ [25]. The m bits are calculated accurately, while the d bits ($d=ln(n)$) are treated differently, similarly to the ETAI configuration. Figure 5.6 highlights that the d remaining bits, the so-called lower part, are the inputs of a series of OR gates, one for each couple of the input bit sequences.

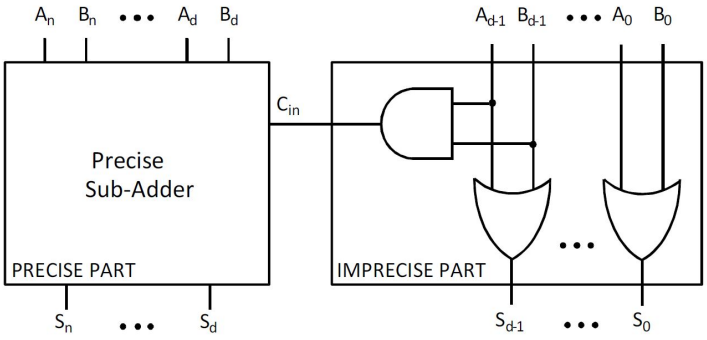


Figure 5.6: Lower-Part-OR Adder Structure. [19]

No carry is generated between the bitwise OR gates of the lower part. To generate the carry-in between the $d-1$ bits and the first m bits, an AND gate executes the product between the two $d-1$ bits of the lower part, represented in figure by the terms A_{d-1} and B_{d-1}. This AND gate is used to increase the accuracy of the approximate adder. The precise computation of the upper part can be executed through any desired accurate sub-adder (RCA or CLA for instance).

As previously cited in the ETAI subsection, the critical path for this example of adder grows as $O(log(n-d))$. This value is typical of all the applications inside the "approximate full adder" set

of approximate adder, cause they all act separating the sequence of bits into two or more parts and perform appropriate operations on the least significant bits.

5.2.7 Speculative Carry Select Adder (SCSA)

Speculative adders like Speculative Carry Select Adder (SCSA) are realized upon the fact that the critical path is rarely activated in ordinary adders. In traditional adders, the output sum will depend on the previous bits. In particular, the most significant output will depend on all the entire number of bits (the number's bit-width) [21]. Starting from this concept, the SCSA try to decrease the dependency on the whole bit-width sequence, reducing the dependency of the outputs. The key idea is to subdivide the chain of propagate signals into blocks of the same size, in a similar way to what concerned for the ACA, going to truncate the carry bits between blocks. Moreover, the addition between pairs of bit in each block exploit the concept of the sparse tree adder. The sparse tree adder is a parallel prefix carry select adder with a structure called "propagate tree", which relates couples of bits quartets of the two input numbers, XORing the propagated stages. [22]

An n-bit adder is segmented into $[\frac{n}{k}]$ k-bit adders. Each sub-adder is made of two other sub-adders, as illustrated in figure 5.7, adder$_0$ and adder$_1$. These two sub-sub-adders execute the precise sum for both c_{in} equal to '1' and '0' (where c_{in} is the carry-in). The value of k has been set to 2, to ensure a good trade-off between precision and performance and to facilitate the calculation of the sparse tree.

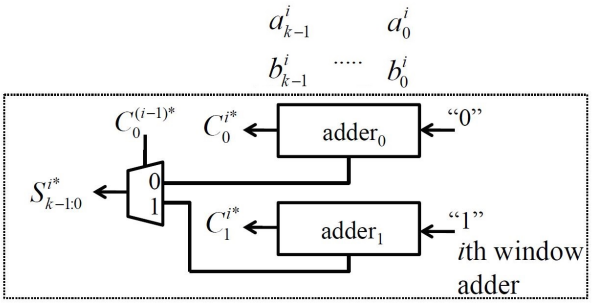


Figure 5.7: k-bit Sub-Adder implementation of the SCSA. [21]

The sparse tree equation states that the j^{th} sum bit of the i^{th} window (i^th sub-adder), s^i_j , for the cases of c_{in} equal to '0' and '1', is estimated as:

$$s^i_{j,1} = p^i_j \ \oplus \ (g^i_{j-1:0} + p^i_{j-1:0}) \tag{5.11}$$

$$s^i_{j,0} = p^i_j \ \oplus \ g^i_{j-1:0} \ , \ 0 \leq j < k \tag{5.12}$$

For sake of simplicity, only the most relevant equations regarding the sparse tree have been collected.

The computation of the SCSA critical path needs some considerations because the parallel sums work with precise additions, which increase the critical path but it is limited by the bit-width of such adders which is k: in general, the critical path delay of a speculative adder is $O(log(k))$; on the other side, the critical path delay of a traditional adder, n-bit adder, has complexity $O(log(n))$, which means that is this case would be $O(log(k))$. At each step of a sum, a maximum of k levels are present for intermediate group p or g signals (propagate and generate). Considering that the total number of sub-adders is $\frac{n}{k}$, the space complexity of an n-bit speculative adder is $O([\frac{n}{k}]klog(k)$, slightly faster than CLA for example. [21]

5.3 Substitutions

The approximate adders are inserted at different level in the SAD accelerator. The substitution of approximate adder into precise adder causes an error but improve the performance, decreasing critical path and timing requirements. Inside the architectures, three possibilities has been taken into account where approximate adders are positioned:

1. Adders are substituted in all the possible position, inside all the sums of the approximate adders, meaning inside the Processing Elements (PEs) and through all the sums of the adder tree, neglecting only the subtractors which remain accurate. The error introduced will be quite high, the higher among the substitutions, and the corresponding performance will increase, the smallest delays and higher power dissipation among the substitutions. From now on this substitution will be called "first substitution".

2. Adders are substituted only outside the PEs, from the twelfth bit on, till the end of the adder tree; this condition is similar to the previous but the error measurement reduces and the performance's improvement will be smaller than the first case. This second substitution is a trade off of the three possibility, because the errors is just generated from the 12^{th} bit on. From now on this substitution will be called "second substitution".

3. Subtractors are substituted from approximate adders, in all the difference inside the PEs. This substitution generates an initial error that propagates through the branches until the last computation, slightly decreasing branch by branch, but, been 4096 differences, produces a strong speed-up. This substitution is optimised for low-error approximate adders because, if the error propagation is reduced, the performance increases with respect to a partially negligible error. Differently from the previous substitutions, it is not only necessary to replace the summing operation with a function that refer to the approximate adder but the difference requires a step more: in software the difference can be seen as the sum of two numbers, one untouched and a second 2-complemented number. In particular, during the subtraction substitution, every couples of bit sequences of the reference and current frames are taken, the first is subjected to the 2-complement and summed to the second. The 2-complement is carried out through a NAND+1 operation. From now on this substitution will be called "third substitution".

Figure 5.8 is a summary of the three possibilities.
The first and the third substitution has been selected through analysis and synthesis of the literature. Walaa El-Harouni et al. [17] consider the implementation of approximate adders inside the accelerator architecture for all the sums present. This study is carried out considering 2, 4 and 6 LSBs approximations, showing a decreasing power consumption, areas and energy with the increase of the approximation. For 3rd substitution, Roger Porto et al. [19] introduce one of the proposed approximate adder (the LOA) inside the subtractors of the PEs inside the SAD accelerator. Moreover, the structure of this accelerator is very similar to the proposed one but support only a 16x16 maximum block size. The subtraction implemented as sum of two numbers, with one of the two 2-complemented, is a very simple trick used not to vary the adders architectures, maintaining same entities and code in all the substitutions. The yellow square of fig.5.8 shows the third substitution: looking into the architecture, PEs receive 16 samples (128 bits) from the current block and 16 samples from the reference block, and calculate the deference, the absolute values sample by sample and sum together the results. The operators in the next pipeline stages were not substituted, there is not a total substitution otherwise the error generated would be too large, accumulated in all the stages. As already outlined, the second substitution has been instead thought as the trade off of the two, without exceed in the approximation and without expecting the highest improvements but controlling the error imposed. Of course, the substitutions will generate results that can be suitable for several

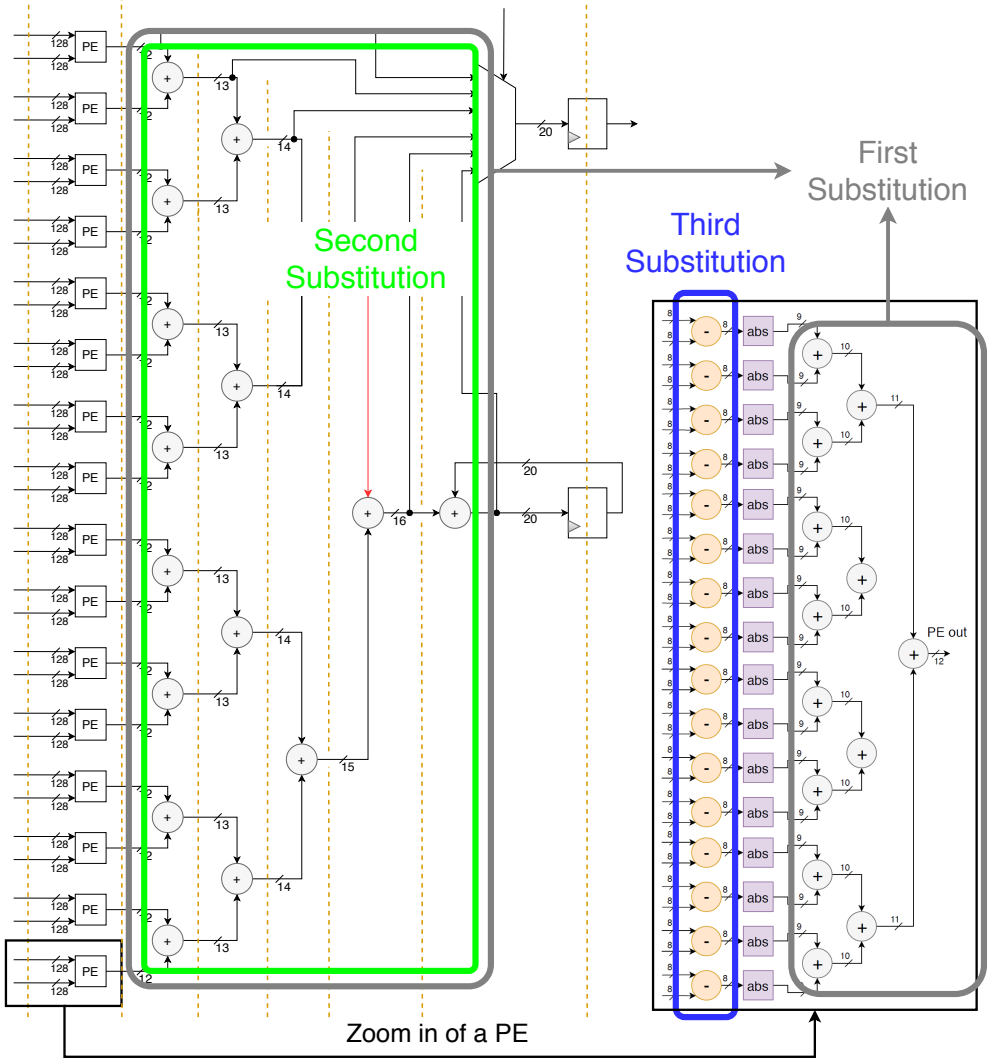


Figure 5.8: Summary of the three substituion inside the SAD accelerator: blue is the first substitution, concerning all the sums inside the system, both in PEs and adder-tree; red in the second one, concerning the adder tree; yellow is the third and last considering the subtractors inside the PEs.

applications in dependence of the ratio performance/error. So, every possible substitution can be considered, giving appreciable results or useless ones, it only depends on the application field.

Chapter 6

Verification

The verification of the three substitutions with approximate adders inside the SAD accelerator architecture is a mandatory procedure in the study of this encoding system. An error evaluation has the task to highlight the ratio between approximation and error connected to the level of approximation; moreover, the error computation can be useful for a final efficiency analysis, where the relation between power saving, area or delays and error rate can be computed.

In the following sections, the verification summarizes the behavior of the encoder employing approximate adders and the error generated by the substitutions through three main error measures: Mean Relative Error Distance (MRED), Total Percentage Error (TPE) and motion vector discrepancies. Section 6.1 deals with the first two mathematical quantities, section 6.2 is related to the graphical analysis of the motion vectors generated with precise and approximate algorithms. The error analysis is based on the articles of Cong Liu et al. [26] and Avishek Sinha Roy et al. [27].

6.1 Error Computation

In order to justify the substitution of approximate adders inside the SAD chain of operations, a C++ model has been developed, which describes the architecture of SAD accelerator. This C++ model emulates the behaviour of the SAD accelerator, step by step, supporting all the 14 SAD cases managed by the hardware and supporting the Partial Distortion Elimination. It operates as follows: it reads an input video sequence in ".yuv" format and performs a raster scan search of all the possible Prediction Blocks in the preceding frame, for all the frames of the video. In other words, the code divides every single frame in PBs, all with the same size, and for each PB, it constructs the search window around it and exploit the raster scan algorithm to search that block in the previous frame. The generation of the search window exploit the same mechanism explained in the motion estimation chapter.

The C++ code can be classified in three parts:

1. The initial phase regards the upload of the ".yuv" video sequence and the settings of all the useful parameter related to it.
2. The second part computes the SAD, frame after frame, and, in this instance, all the partial SADs are executed, meaning that the substitutions of the approximate adders inside the operations chain are inside this step.
3. The last phase is the comparison part of the C++ code, where the minimum SAD is updated and the error computation is carried out.

The model has been setting up with a default search window of ±64 and a subsampling of 4 but it is possible to modify these values at will. The aim of the C++ model is to evaluate of the accuracy of the SAD accelerator implementing different approximate adders. The error measurement of the system has been estimated through six error characteristics: the error distance (ED), the mean error distance (MED), the relative error distance (RED), the mean relative error distance (MRED), the percentage error resulting from the mean relative error distance ($E_{MRED}(\%)$) and the total percentage error (TPE(%)). The definitions of these six measures of error are collected in the following list:

- **Error Distance**: $ED = |S' - S|$, where S' is the sum of the approximate adder and S is the sum of the accurate adder;
- **Mean Error Distance**: $MED = \frac{\sum_i ED_i}{PB \cdot N_{frames}}$, where PB is the number of Prediction Blocks and N_{frames} is the total number of frames;
- **Relative Error Distance**: $RED = \frac{|S'-S|}{S}$;
- **Mean Relative Error Distance**: $MRED = \frac{\sum_i RED_i}{PB \cdot N_{frames}}$, where PB is the number of Prediction Blocks and N_{frames} is the total number of frames. It can be note that the product $PB \cdot N_{frames}$ represents the maximum number of iteration of the system;
- **Percentage Error**: $E_{MRED}(\%) = MRED \cdot 100$
- **Total Percentage Error**: TPE(%) is the most common error counting: it represents the number of errors calculated for every SAD case divided by the total number of event, times one hundred; in other words, it highlights how many times the output SAD differs in the case of precise or approximate adders, during all the C++ model processing, without considering any threshold.

In particular, several tests, employing the three aforementioned substitutions (complete adder-tree, part of the adder-tree and the subtractors), has been carried out to study and verify the different configurations; the model can perform the search of each PB of every SAD case and in every frame. Two ".yuv" video sequences has been tested, the first is "BUS_352x288_30_orig_01.yuv", with a low resolution of 288p, and the second is "ducks_take_off_420_720p50.y4m", with an high resolution of 720p (HD), available in ref.[28]. Generally, the code is optimized so that a 720p video sequence can be elaborated in 3 hours. Higher resolution videos require, as can be guessed, more time but was not necessary to investigate all the possible exiting cases. The error study has been carried out for just two examples of video sequences to point out the error variation from a low resolution to an high resolution video.

6.1.1 Results

The first substitution considered is the full adder-tree case, i.e. approximate adders inserted in every sum after the differences and absolute values (Processing Elements included). Tables 6.1 and 6.2 show the E_{MRED} and TPE calculation for the first sequence (150 frames) and a frequency of 30 MHz; tables 6.3 and 6.4 show the two error parameters computed for the second video sequence, with double frequency and the same number of frames.

The first consideration, regarding the obtained tables, is the expected higher error value in dependence of the higher approximation factor, coming from the segmented and full approximated adders.

The second compulsory consideration is related to the two error quantities: the two parameters show different results, being different evaluations, but in this specific case the most explanatory error form is surely the MRED (and the E_{MRED} resulting) because it represents the average error distance

Table 6.1: E_{MRED} Measurement in the 14 SAD cases with the YUV video sequence "BUS_352x288_30_orig_01.yuv", 150 frames, $f = 30MHz$, 1^{st} substitution.

$\mathbf{E}_{MRED}$(%)	LOA	ETAI	ACAA	ACA	SCSA
8x4	0	0	0	0	0
8x8	0.051	~ 0	~ 0	0.015	~ 0
16x4	0.1	~ 0	0.003	0.055	~ 0
16x8	0.288	~ 0	0.006	0.122	0.047
16x12	0.534	0.03	0.181	0.217	0.109
16x16	0.818	0.349	0.267	0.201	0.201
32x8	0.921	0.885	0.268	0.272	0.255
32x16	1.952	1.559	0.838	0.617	0.583
32x24	2.197	1.929	1.78	1.008	0.957
32x32	4.163	4.253	2.792	1.427	1.359
64x16	4.686	6.78	3.582	1.651	1.576
64x32	9.436	9.45	7.323	3.384	3.235
64x48	14.329	13.918	11.028	5.158	4.934
64x64	21.601	21.380	16.55	7.82	7.485

Table 6.2: TPE Measurement in the 14 SAD cases with the YUV video sequence "BUS_352x288_30_orig_01.yuv", 150 frames, $f = 30MHz$, 1^{st} substitution.

TPE(%)	LOA	ETAI	ACAA	ACA	SCSA
8x4	0	0	0	0	0
8x8	13.089	2.897	8.783	0.842	0.463
16x4	14.057	2.894	9.174	0.673	0.631
16x8	35.69	48.518	26.763	5.135	5.135
16x12	49.748	48.518	39.341	8.333	8.333
16x16	58.754	57.606	41.667	18.856	18.582
32x8	58.249	57.606	48.737	20.37	20.37
32x16	65.32	66.667	49.495	43.098	43.098
32x24	66.667	66.667	50.758	66.162	56.568
32x32	66.667	66.667	48.485	63.973	63.3
64x16	66.667	74.444	53.333	62.963	62.963
64x32	66.667	66.667	60	66.667	66.667
64x48	66.667	66.667	63.333	66.667	66.667
64x64	~ 70	66.92	66.667	66.667	66.667

among all the mistakes committed during the calculation of the SADs. On the one hand, MRED is a relative computation of the discrepancies between precise and approximate implementations. Moreover, it's the most diffused error parameter in the literature about the approximate adders. Ref.[26] and ref.[27] use the MRED, RED and ED to compare a set of approximate adders for error-tolerant applications: these error measures define how much the approximate result is detached from the precise one. On the other hand, TPE highlights the presence or not of error. It's implemented in the C++ code as an "if" clause where, if the difference between accurate and approximate SADs is different from zero, an error count in increased by one. In fact, TPE is a sort of counter for the number of errors, without considering that in video compressing a small compression error can be adopted to increase the process speed, losing in term of resolution.

Table 6.3: E_{MRED} Measurement in the 14 SAD cases with the YUV video sequence "ducks_take_off_420_720p50.y4m", 150 frames, $f = 60MHz$, 1^{st} substitution.

$\mathbf{E}_{MRED}$(%)	LOA	ETAI	ACAA	ACA	SCSA
8x4	~ 0	0	~ 0	0	~ 0
8x8	0.054	~ 0	0.044	0.008	~ 0
16x4	0.109	0.022	0.087	0.015	0.012
16x8	0.322	0.137	0.251	0.055	0.047
16x12	0.61	0.349	0.464	0.122	0.109
16x16	0.949	0.634	0.712	0.218	0.201
32x8	1.086	0.803	0.803	0.372	0.255
32x16	2.319	1.811	1.702	0.917	0.583
32x24	3.627	2.932	2.651	1.198	0.957
32x32	4.986	4.125	3.633	1.927	1.359
64x16	5.623	4.748	4.091	2.151	1.576
64x32	11.404	9.706	8.277	4.384	3.234
64x48	17.252	14.323	12.51	6.358	4.934
64x64	24.022	22.36	18.86	9.374	7.485

Table 6.4: TPE Measurement in the 14 SAD cases with the YUV video sequence "ducks_take_off_420_720p50.y4m", 150 frames, $f = 60MHz$, 1^{st} substitution.

TPE(%)	LOA	ETAI	ACAA	ACA	SCSA
8x4	0	0	0	0	0
8x8	18.651	13.333	45.581	0.842	0.463
16x4	18.671	13.333	48.927	0.673	0.631
16x8	51.852	48.518	52.399	5.135	5.135
16x12	55.303	48.518	54.545	8.333	8.333
16x16	56.566	55.303	57.071	18.855	18.852
32x8	62.259	60.101	58.838	20.37	20.37
32x16	66.667	60.101	59.091	43.098	42.761
32x24	66.667	67.424	68.939	56.566	56.566
32x32	66.667	66.667	67.677	63.973	63.3
64x16	68.889	68.889	73.333	62.963	62.963
64x32	75.556	75.556	73.333	66.667	66.667
64x48	76.667	73.333	70	66.667	66.667
64x64	80	75.556	70	66.667	66.667

A third consideration can be related to the strong increase of the E_{MRED} value going considering not HD and HD video sequences: the error gap in this phase is around the 2%, which means that, for higher resolution videos (full HD and 4K), the error will surely increase. Thus, the only acceptable values comes from the use of ACA and SCSA (ACAA is borderline) if the video under analysis is full HD or 4K.

A final fourth consideration regards the two different analysis, mathematical and graphical, carried out in sections 6.1 and 6.2: the error measurement mathematically calculated in this section, even if the error distance represents the discrepancy between precise and approximate results, don't give a clear idea of the tolerance and resilience of the video applications. An error such as $EMRED = 16\%$ of the ACAA for the first video sequence doesn't mean that the result will show the 16% of error but

that will distance from the precise implementation of a mean value equal to 16%, which graphically is very difficult to figure out with numbers, because the SAD is a matrix computation. The error evaluation is mainly used to classify which are the most erroneous approximate implementations and it can be use to compare performances and errors among the several combinations.

The second step in the error study consists in the calculation of E_{MRED} and TPE for an higher number of frames, to understand if an increase in the frame number can vary the error. For example, the error increase in the case of the LOA structure 64x64 of block size is around the 0.15% for E_{MRED} and constant for the TPE, as illustrated in Table 6.5.

Table 6.5: Error Measurement with the YUV video sequence "BUS_352x288_30_orig_01.yuv", 300 frames, $f = 30MHz$, 1^{st} substitution, employing the Lower-OR Part Adder

300 Frames	8x4	8x8	16x4	16x8	16x12	16x16	32x8
$\mathbf{E}_{MRED}$(%)	0	~ 0	0.01	0.287	0.532	0.815	0.921
TPE(%)	~ 0	1.052	2.946	9.68	10.732	14.409	21.729

300 Frames	32x16	32x24	32x32	64x16	64x32	64x48	64x64
$\mathbf{E}_{MRED}$(%)	1.955	3.045	4.174	4.707	9.529	14.409	21.729
TPE(%)	30.30	30.30	30.30	66.667	66.667	66.667	70

Tables regarding the other measurements, implementing the other four adders and the HD video, are not present because the most of the cases show no difference from the already commented tables 6.1, 6.2, 6.3 and 6.4. For sake of simplicity, just the LOA case is shown. Moreover, LOA is the adder showing the larger error and the higher increase raising the number of frames, so, the other adders would have shown a minor error. Differently from the previous step from not HD to HD videos, the increase in frames weighs just a little on the calculation of the error. This means that the increase of percentage is clean if the video under analysis is full HD or 4K and not if the number of frames is high; it is always useful to remember that the selection of the appropriate approximate adder depends on the application and that the certainly acceptable values comes only from ACA and SCSA.

The third step of the study make use of the second substitution. The second substitution consists of the insertion of the approximate adders in the adder-tree sums after the Processing Elements (PEs). This operation will surely show an error rate smaller than the previous case: the accuracy increases because the bit chains are elaborated by precise adders until the 12^{th} bit, and only after becomes affected by approximations; consequently there will be a reduction of the number of approximated terms.

One important observation coming from tables 6.1, 6.2, 6.3, 6.4 and 6.5 is the higher error value for LOA implementation. Among the five analyzed approximate adders, LOA is the is the most erroneous adder. Considering a block size of 64x64, for low and high resolution videos, LOA can be considered the "worst case" implementation, no other architectures reach higher value of error. Thus, for sake of clarity, the error study with the second substitution is represented just for this worst case. Table 6.6 illustrates the obtained error. As expected, the maximum assumed error quantity is largely smaller than in the 1^{st} substitution, both for E_{MRED} and TPE.

The results coming from table 6.6 are very positive: E_{MRED} and TPE reaches acceptable values with respect to the two previous steps and, as expected, second substitution looks satisfactory for its low error. The second substitution is definitely the most promising alternative, not just because the error results halved with respect to the previous cases but also because it's the best trade-off among the substitutions. Looking at table 6.6, it can be highlighted the reduced gap between HD and low resolution cases, which was a critical problem of the first substitution. Furthermore, this is the worst case analysis; the other results (calculated but not here illustrated) coming from the other

Table 6.6: Error Measurement with the two YUV video sequences, 150 frames, 2^{nd} substitution, employing the Lower-OR Part Adder. Results coming from $8x4$ to $16x12$ gave null results.

Resolution: 288p	16x16	32x8	32x16	32x24	32x32	64x16	64x32	64x48	64x64
$\mathbf{E}_{MRED}$(%)	0.241	0.345	0.799	1.308	1.856	2.148	4.403	6.713	10.178
TPE(%)	32.576	32.828	33.333	31.818	33.333	33.333	40	43.333	45

Resolution: 720p	16x16	32x8	32x16	32x24	32x32	64x16	64x32	64x48	64x64
$\mathbf{E}_{MRED}$(%)	0.241	0.345	0.799	1.308	1.856	2.148	4.4	6.715	10.181
TPE(%)	32.576	34.848	33.838	33.333	33.333	33.333	44.444	43.333	45

approximate adders implementations has similar trend, with even lower error values.

Forth and penultimate step considers the 3^{rd} substitution, which regards the subtractors substitution for the proposed approximate adders. Table 6.7 illustrates the obtained results of the C++ model's simulation.

Table 6.7: Error Measurement in 64x64 config. with the YUV video sequence "BUS_352x288_30_orig_01.yuv", 3^{rd} substitution

	LOA	ETAI	ACAA	ACA	SCSA
$\mathbf{E}_{MRED}$	8.91%	8.084%	7.696%	4.991%	4.380%
TPE	45.90%	46.05%	46,05%	40%	40%

In table 6.7, the error profile of the approximate adder slightly change with respect to the previous steps. Here, the error is illustrated just for the 64x64 block size for all the adders because, even if the classification that defines which are the most or least erroneous adders is constant, the results are surprising. One could expect from the third substitution that the error generated by the difference at the beginning of the SAD computation could increase step by step, like an avalanche effect. In reality, the error is probably compensated by the adder-tree of the SAD accelerator and mitigated through the branches, because this last substitution show the minimum mean relative error distances. The compensation of the MRED can be note also from the fact that the TPE in this last substitution is higher than in the 2^{nd} substitution, i.e. the number of iterations $PB \cdot n_{frames}$ reduces the MRED.

The final step maintains the same substitution (3^{rd} sub.) but the video sequence considered is the HD video "ducks_take_off_420_720p50.y4m". The only interesting approximate configuration is the LOA structure, thanks to its higher error rates. The values obtained for E_{MRED} and TPE are:

- E_{MRED} = 8.993%;
- TPE = 45.934%.

The results coming from this list highlights the same conclusion of table 6.6, both having a reduced error gap between 720p and 288p video sequences' simulations.

6.2 Motion Vectors Error Analysis

The results of the error validation from the previous section gives a satisfying justification regarding the implementation of approximate adders inside the accelerator, with the warning to use the approximate adders always considering which errors the system is going to meet. This characterization could be enough to compare the different architectures but, it can be useful to analyze the motion vector's

mismatch between precise and approximate systems because it is the real variation from implementation to implementation during the development of the motion estimation algorithm: the numerical error is an average result of the accuracy of the methods but what really change using approximate arithmetic are the motion vectors and the resulting motion estimated frames.

For this analysis, the employed video sequence is "BUS_352x288_30_orig_01.yuv" [28], whose 150 frames are divided in macroblocks of 64x64 samples with search window of ±64, simulating the HEVC algorithm. In reality, in HEVC the blocks can be divided in 16x16, 32x32 or 64x64, and further subdivide if necessary according to the quad-tree scheme. Moreover, the selected search window is very large, typically in HEVC it is set to ±27 or ±32. Block size of 64x64 and a search window of ±64 gives the best perspective of the error evaluation and returns the higher resolution result, which implies the higher motion vectors' mismatch, at the cost of a decrease of the algorithm's speed.

The first problem met in the study of the motion vectors is the selection of the software. The C++ code, used in the previous survey, has limited capability in dealing with matrix results. Nevertheless, the C++ code is essential in the development of the approximate adders, working of the signle bits of the input sequences. However, image processing is very efficient in Matlab, where is very easy to calculate the SAD and the motion vector between two frames, but Matlab is not suitable for bit elaboration. A good compromise can be obtained employing both softwares for the two separated tasks. The flow diagram related to the study methodology used for the motion vectors' error estimation is illustrated in figure 6.1.

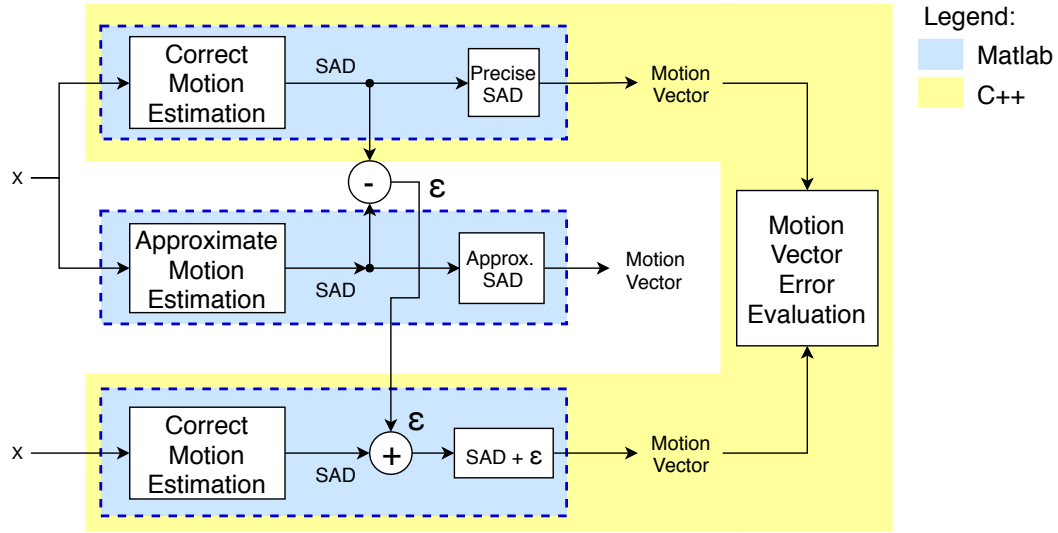


Figure 6.1: Flow diagram of the motion vector error evaluation

The light blue rectangles in figure 6.1 represent the steps suitable for the C++ process; Matlab (the yellow region) allows to compute all the steps of the diagram until the error evaluation, only for the accurate part. Matlab is a very powerful software, able to execute every step of the motion estimation, but it's particularly difficult working on the bits of a number, which are the essential part for the motion estimation of the approximate structures. The technological trick is to subtract each precise SAD from the approximate SAD, obtaining an error value ε which represent the mismatch between precise and inaccurate results. This error variable can be summed, in another chain of the diagram, to the precise SAD of an accurate motion estimation (which is equal to the initial ME), in order to simulate an approximate motion estimation, bypassing the numerical modification of the input elements; in other words, in Matlab it is possible to circumvent the bits alterations of the inputs numbers using a mathematical artifice to simulate the approximate motion estimation, starting from

a precise motion estimation. In this phase, the C++ code is employed for the ε calculation, Matlab for the motion vectors computation of the dummy approximate ME (shown in figure 6.1 in the lower part).

6.2.1 Results

The Matlab function has been designed to carry out the ME of a video sequence, during which the motion vectors' computation is realized. The error measure requires the worst case of motion vector; an average error value would be too similar to the calculation previously performed. For this purpose, the code has been modified in order to output the larger motion vector: large motion means the higher displacement between two frames, so the function returns also the couple of frames with the larger movement (worst case). The selected approximate adder is the LOA and the substitution is the 1^{st}, being the worst case adder and the worst case substitution of the previous analysis.

The Matlab instruction used to display graphically the motion vector is the "quiver" function. The quiver function displays the velocity vectors between two or more vectors, or two or more matrices, of the same size in the from of arrows. Figure 6.2 represents the motion vector between the two frames in the video "BUS_352x288_30_orig_01.yuv" with the higher values.

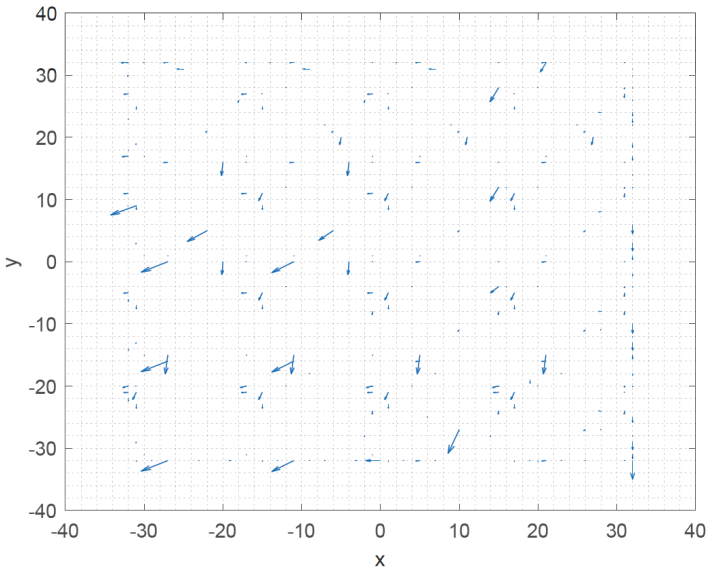


Figure 6.2: Motion vectors' difference between precise and approximate (LOA adder) motion estimation for the video sequence "BUS_352x288_30_orig_01.yuv". The variation is not very appreciable because of the 16x16 macroblock with a search window of ±64, typical configuration of the HEVC algorithm. The mismatch between precise and approximate architectures could be more evident for older standards or smaller blocks.

The general trend of this specific couple is a movement towards the left, slightly leaning towards the bottom. As can be seen, there is not a strong difference between the positions of the two frames, considering that this is the worst case. It has been proved that, if there is a best match mistake in the 4x4 case, the wrong estimated best match can be everywhere inside the frame and consequently the motion vectors of the right and wrong best match can be very different [2]. Instead, for higher block size and larger SADs, if there is a mismatch between correct and wrong best match, their difference is small. Hence, the motion vectors in the majority of cases are very similar and the mismatch between two frames assumes small values, such as in the case of the tested video sequence. For example, given

two components X_i current and Y_i reference frames, if the motion displacement is just of few pixels, the human eye is not sensible enough to perceive the difference between reference and current frames.

The visual graphical analysis can be mathematically described employing the numerical values that generated the motion vectors. From the previous data, two approaches has been selected to numerically express the error rate between approximate and precise results. One is the percentage mean relative error distance (E_{MRED}), although at the beginning of the section it was decided to avoid the calculation, and the second is an energy related to the inaccurate prediction.

1. The E_{MRED} is computed to have a connection point with the previous calculated error values and to give the possibility to make a comparison. The definition of E_{MRED} is the same introduced in chapter 6.1.

2. The energy is an internal value, defined as the sum of the square absolute values of differences between the current frames minus the motion compensation (MC) of the prediction units of the reference frames, based on the motion vector, for the entire video sequence. The motion compensation (MC) allows transformation of the reference frame into the current one; it is used to predict a frame inside the previous frames. The procedure of subtraction between the current and the motion compensated reference frame is the same of doing the subtraction between the current frame and its transformation, based on the motion vector. This corresponds to the square prediction error evaluation, extended to all frames of the video. This concept is illustrated in figure 6.3.

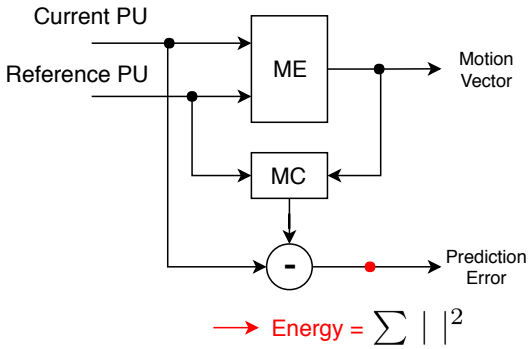


Figure 6.3: Block diagram for the energy (E_n) estimation

The expression, which defines the energy (E_n), is written in equation 6.1:

$$E_n = \sum_{i=1}^{N} |PU_{current} - PU_{MCreference}|^2 \tag{6.1}$$

Being an internal function, there are no other comparison error measures available; the only distinction is the comparison between the energy obtained from accurate ME and approximate ME.

Table 6.8 illustrates the results coming from the analysis.

The E_{MRED} coming from this last analysis is significantly lower than the E_{MRED} calculated from the C++ simulations. This could be a consequence of the already discussed small mismatch between blocks of large size, where the single displacement looks negligible among the 64x64 blocks. At the same time, the energy result highlights the same behavior: the energy calculated from precise and approximate prediction error is quite similar, showing an appreciable variation in the fourth significant digit, 3 order of magnitude smaller than the most significant.

Table 6.8: Results of the motion vector error estimation

	MRED	Energy = $\sum \mid \mid^2$
Accurate Results	0%	$1283544 \simeq 1.28 \cdot 10^6$
Approximate Results:	6.7%	$1282726 \simeq 1.28 \cdot 10^6$

6.3 Summary of the Error Evaluation

The feeling about the SADs error computation of section 6.1.1 is that an error assessment mathematically carried out highlights a mean error quite high because the average mismatch between the two SADs of the two cases (approx. and accurate) is not too representative of the actual video compression discrepancy. In other words, a percentage of 20% doesn't imply that the final approximate decompressed video will show the 20% more of visible artifacts to the human eye. The first analysis, which employs geometric models, was useful to understand the most and least erroneous combinations of adders and substitutions. The percentages do not reflect perfectly the reality of the approximation, they gives a general idea and the possibility to calculate an efficiency of the system, where the efficiency can be calculated as "parameter/error". The parameter can assume any variable's value, timing constraints, power dissipation or saving, energy consumed or other quantities and, dividing by the error, is possible to evaluate the impact of the approximate adders inside the architecture of the SAD accelerator.

The error evaluation related to the motion vectors is more representative: the motion vector is the motion estimation's element which dynamically changes frame by frame and in presence of an approximation. Its variation is the cause of the SAD variation and, consequently, a different minimum SAD of the best matches between accurate and imprecise cases. This explain why an analysis on the motion vector can give more appreciable results than the first study. Nevertheless, the error evaluation of the E_{MRED} (rather than TPE) in this second analysis is always an average parameter which doesn't give a clear perspective of the effective error generated by an approximation, it is just useful to rank the different approximation. The energy can probably provide a more detailed view but the energy ends in itself and, so, cannot be compared with the other error measures.

At the end of the analysis, the efficiency computation is probably the best conclusion to clearly compare the several proposed combinations, even if the errors got are just a classification of the approximate adders and not of the effective discrepancy. It is calculated in chapter 9, after the synthesis during which the useful parameters, to be divided for the error, are obtained. The motion vector analysis has been useful to graphically understand how an approximation acts on the motion estimation but analytically it didn't give usable results.

Chapter 7

Adders Architectures, Implementation and Simulation

7.1 Introduction of the Approximate Adders Inside the SAD Accelerator

This chapter implies all the hypotheses and conjectures introduced in chapter 5 and focus on the electronic part of the substitutions, how the adders has been designed and how the implementation was conducted. This section has the objective to describe the way the adders were inserted into the accelerator both in terms of VHDL code and circuit; it is the prelude to the next section, where the architectures has been simulated and tested through Modelsim.

The introduction of the approximate adders was not a tricky task: the SAD accelerator design by Paolo Selvo [2] has a dedicated function called "adder" which execute the sum of two input numbers. In the first substitution, it was enough to modify that function, going to specialize it for each of the approximate adders: instead of the usual sum, a port mapping to the approximate adder function is realized. In the second substitution, two adders are required, an accurate one (the previously cited "adder") and a specific adder function, specialized for each approximate adders. In the third substitution, the adder remained the previously implemented, while the difference of the absolute value of the SAD computation changed: instead of the automatically compiled subtraction, the 2-complement (NAND+1) is done and a port mapping to the specific approximate adder is inserted. The VHDL accelerator code is as much as possible generic, to let the possibility to modify it in the future. All constants are defined in a package. The system hierarchy is the same explained in chapter 4; the hierarchical order is the following: PE block, datapath, unit interface and then the top entity. In the end, the SRAM is an IP of the library used for the synthesis (CMOS 65 nm technology). [2]

The following list highlights the difficulties and artifices used to implement the approximate adders; the list can be translated into a series of tips that have been used in the design of the accelerator.

ACA: The almost correct adder is one of the most complicated approximate adders due to the parallel computation of the sums and carries, exploiting propagate, generate and kill signals, due to the variable size of the bit-width through the adder tree, starting from 8 to 20 bits (fig.4.1) and due to the calculus of the longest propagate sequence (see fig. 5.3), equal to $ln(n)$ for a n-bit sequence, which dynamically change from 8 to 20 bits. Thanks to the VHDL function "generic map", the bit-width can be transferred through the several functions, solving the problem related to the variable size oper-

ations through the adder tree and the difficulty of the variable propagate sequence value. The parallel computation is possible fixing a variable size function and port mapping it in parallel for all the bits, activating the port mapping just in the cases required. For example, the port mapping is executed for all the 20 bits but in the branch of the adder-tree where the algorithm works with 13 bits, only the first 12^{th} functions are called. The distribution of the bits to generate the sum, which, as can be seen from figure 5.3, takes, after the first $ln(n)$ bits, one bit at a time from the MSB of every parallel sum, can be easily set thanks to the versatility in working with bits given by the VHDL language.

ACAA: Conceptually, the ACAA has a simple structure, with the main limitation to divide the input vectors in three parts (begin, middle and end). The division by three implies that the bit-width must be splittable in 3 parts, and the only bitdepths of reference and current frames suitable with such limit were 8 (inside the PEs), 12 and 14 and 16. The other lengths (13 and 15) have been leaded back to these four bit-widths. As discussed for the ACA, the final sum can be easily obtained working of the single bits coming from the parallel sums.

ETAI: Error tolerant adder, and LOA, are two adders of the "approximate full adder" class, which split the reference and current inputs in two parts and execute an approximation of the smaller part. The approximated/inaccurate part for both the adders has been set to 3 ($ln(20) = 3$), which means a quite strong approximation inside the PEs, and a softer approximation inside the adder-tree. ETAI working mechanism can be realized through a combination of "for" loop and "if" clauses. The precise part is automatically implemented by the compiler, simply using the '+' operation.

LOA: The inaccurate part of the LOA has been implemented through an OR gate between the pairs of bits. In LOA, to improve the accuracy, a carry between approximate and accurate part is generated, exploiting an AND gate between the most significant bits of the imprecise part.

SCSA: The SCSA has logic and parallel computation similar to the ACA, but the complex implementation is due the sparse tree implementation. The parallel sums are implemented with the same design artifices of the ACA, while the sparse sum is carried out through a for loop with a "case-when" clause inside for both c_{in} equal to '1' and '0'. Although the sparse tree computation is a particularly cumbersome process, it has the advantage of being recursive, so the result of the first phase is used for the second, and so on.

In all the implementations, RCA and CLA included, c_{in} and c_{out} are inserted in the code, even if sometimes not useful. For example, in the ETAI no carry is generated in the imprecise part, or, in the ACA no carry-in among the parallel computation is required, but the presence of carry-in and carry-out can be worthwhile for future implementation or optimizations. Moreover, some carries that were not used in the addition, can be exploited in the subtractions of the third substitution, where one of the two input numbers is subject to a NAND+1.

Regarding the accurate adders, RCA and CLA didn't show particular problems or obstacles. Their implementation was linear, achievable with both "for" loops or parallel "port mapping" and both gave the same compilation time in Modelsim, highlighting that, for such small portions of codes, any approach can be used. It will be interesting to study the behavior of the RCA/CLA implementation automatically sensitized by the software with respect to the RCA/CLA written code.

7.2 ModelSim Simulations

The validation of the accelerators is carried out with Modelsim, a multi-language HDL simulation environment for simulation of hardware description languages, such as VHDL, developed by Mentor

Graphics. Two testbenches were written to perform a test of the architecture and to evaluate the inequality of the outputs while approximate adders were employed. To generate a reference and a current random input matrices, a dedicated Matlab script was written. Sections 7.2.1 and 7.2.2 discuss these two main topics: Matlab script and Modelsim testbenches.

7.2.1 Matlab Script

The Matlab script used for the validation generates four files in ".txt" format (easily readable by every other software): in two of them it writes the reference and current samples matrices, in the third it writes the row (R) and column (C) values of the test reference and current matrices and in the last it writes the SAD results. The latter can be exploit to verify if the approximate computing differs from the precise one. The working principle of the script is the following: Matlab generates N random input vectors, composed by a R value, a C value, a reference Prediction Block (PB) of dimension RxC, a current PB of dimension RxC and the SAD result. For each of these vectors the script generates randomly the values for R and C, considering the set of allowed block sizes, shown in the underlying list:

- 64 x 64
- 64 x 16
- 32 x 16
- 16 x 12
- 8 x 8
- 64 x 48
- 32 x 32
- 32 x 8
- 16 x 8
- 8 x 4
- 64 x 32
- 32 x 24
- 16 x 16
- 16 x 4
- 4 x 4

At this point, it generates RxC numbers for the reference PB and RxC numbers for the current PB. The values range can vary from 16 to 235 according to the YUV format specifications [1]. In the end, the last step consists in the SAD computation for the two ref. and cur. matrices. This kind of test is useful to mainly stress the accelerator by changing the SAD type almost every operation and to ensure the correct functioning of all SAD cases.

7.2.2 Modelsim Testbenches

The VHDL testbenches employed during this validation phase are two. The first testbench emulates the behavior of the system's controller, able to deal with the hardware accelerator. The function of the controller is to handle the input/output signals based on two clock regimes. The second testbench takes the results of the first and performs a direct comparison between the results from accurate measurements and results from approximate calculations, computing the error variation. The complete scheme of the testbenches is shown in figure 7.1.

First Testbench

The first testbench is a modified version of the testbench written by Paolo Selvo [2] and occupies the upper left part of the figure. This testbench is composed of five primary processes, represented by five blocks inside figure 7.1, which are discussed in the following list:

1. **Data Maker Process**
   The data maker is the process which works with rows, columns, block sizes, clocks and memory. It operates in the regime of the slower clock (SRAM IP clock of 200 MHz); it has five main output signals, "rows", "columns", "toggle2", "end_stimuli" and "valid_in". When a new SAD needs to be calculated, the data maker takes R and C from the specific ".txt" file, given as output from the Matlab script, and orders to the "Writer" process to start. At this point, reference and current PBs are written inside the memory and, when finished, the data maker charges R and C on their respective buses and put the "valid_in" to 1. Meanwhile, data maker orders the writer

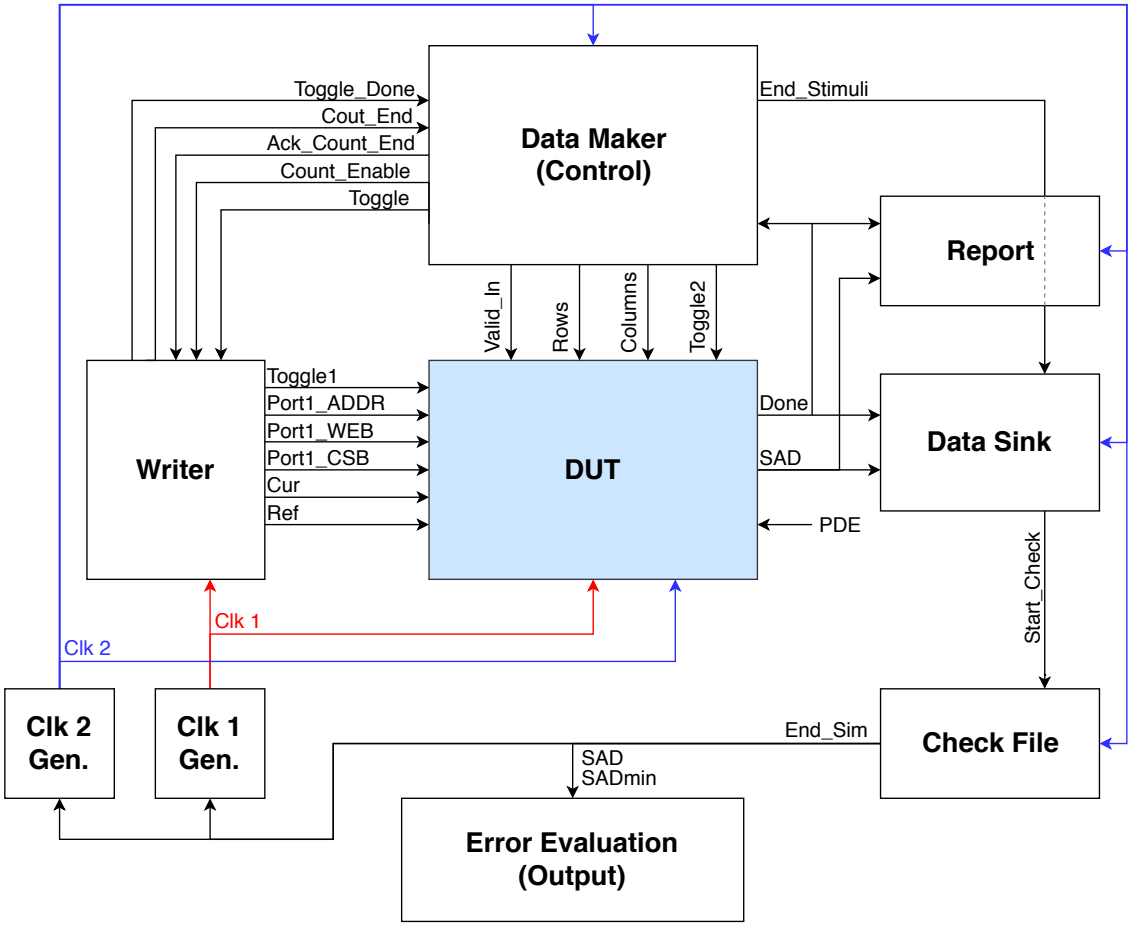


Figure 7.1: Summary block diagram of the Testbenches: in the middle of the figure the device under test (DUT) communicates with the logical blocks of the first testbench, data maker, writer, report, check file and data sink. In the lower part, the SAD comparison and error computation is synthesized in the 'Error Evaluation' block. The blue line represents the clock 2, generated by the Clk2 generator block; equivalently, the red line represents the clock 1, generated by the Clk1 generator block.

to charge on the empty memory unit (the one not used yet) two new current and reference PBs. When the second pair of PBs is written, a "done" signal is received by the data maker and until the writing operation is not finished the data maker waits.

If the writer is writing on the second memory unit, the data maker commutes toggle2 and starts a new SAD operation on the same memory unit on which the writer is charging samples but in a difference memory cell. This condition is called 'Overlap' and it occurs when the reader waits for the writer. When the writer finishes, the data maker receives a "done" signal; at this point, the reader and the writer switches the memory unit and the Overlap condition is avoided, or rather is solved. A critical situation occurs if the Overlap condition persists and the reader finishes the reading phase but the writing phase is in progress: in such conditions, the data maker refers the presence of a pending unresolved "done", solved when the writer finished the writing of the PBs [2]. Until then, if possible, the data maker changes the memory unit and start a new SAD operation. In spite of everything, the reader and the writer are still in Overlap condition. To avoid this critical overlapping of actions, the writer must start writing one macrocolumn before the reader. This approach avoids the overlap of reading and writing operation on the same

location. Meanwhile, the data maker waits one Clock 2 cycle and raises the "valid_in" signal.

The data maker stops working at the end of the RxC ".txt" file, when the whole file has been read, but its last operation is made at the end of the SAD computation: when the last "done" signal is received, the data maker rises the "end_stimuli" signal. A brief note must be considered during the ".txt" reading of the RxC file: if C is greater than R, the data maker simply exchange the two dimensions. Example: a block size of 32x64 is converted in 64x32.

2. **Writer Process**
   The writer process is the counterpart of the "Data Maker" process and works in the faster regime, 1600 MHz. The managed signals are "port1_WEB", "port1_CSB", "toggle1", "port1_datain_cur" (Cur signal in in Fig. 7.1), "port1_datain_ref" (Ref signal in in Fig. 7.1), "cur_charged" and "port1_addr". It deals with all the writing steps, ordered by the data maker. When "toggle" and "count_enable" are both high, it switches the memory unit and, according the dimension of the PB, it distinguishes two cases:

   (a) if $R \cdot C < 256$, the writer will write a PB with size between 8x4 to 16x8 (included). In this case, the samples requires only the first macrocolumn of the memory unit. This process holds 8 clock cycles (clock2): 256 bits every cycle (corresponding to 32 sample, a macrocolumn).

   (b) if $R \cdot C \geqslant 256$, the writer will write a 16x16 to 64x64 PB. Therefore, more than one macrocolumns are required and it could write up to 16 macrocolumns. The worst case process takes 16 times more than the previous analyzed: $8 \cdot 16$ cycles. The address is decided through a counter which increase of one every 8 cycles.

   The writer has always the priority to fill the 1^{st} macrocolumn of the memory, before charge the 2^{nd}. It starts from the first minimum unit (the basic 4x4 element) of the first macrocolumn and, when it reaches the last minimum unit, the value of the address is incremented by one; then, the writer proceeds with the second macrocolumn. In this way, the parallel PEs are pushed to the maximum capability [2].

   Another task of the writer is to accurately handles "port1_WEB" and "port1_CSB" signals. When both the signals of a memory bank are set to '0', the memory bank is active; when both are '1', the memory is in low power mode (inactive). The writer sets to '0' these two signals when is working on a specific bank and leave the others in low power mode. As introduced in section 4.6, the current PB remains constant during the motion estimation and only the reference needs to be updated. At the beginning of the best match search, the writer charges both reference and current PBs the first time into the SRAM memory. After, during the following operations, only the reference PB has to be loaded. When it has finished to charge a PB, it reports this event to the data maker process with the signal "count_end".

3. **Data Sink Process**
   The data sink process is connected to clock2 and has a single output signal, which is the "start_check" signal. When a high "done" signal is received, it writes the output SAD value on the output file. Finally, if the "end_stimuli" signal is raised, it starts the "Check File" process and stops. The check file process consists just in the verification of correctness or dissimilarity of the SAD results. No error computation is carried out in this phase, it's a task of the second testbench.

4. **Check file Process**
   This check file process compares the two ".txt" files where the SAD computations are carried out, one from Matlab and one written during the Modelsim simulation. If there is not any error,

the process prints on the console that the simulation was successful; otherwise, it prints that the simulation failed. When approximate adders are used, the test always failed and the second testbench was required. At the end, the "end_sim" signal is generated and the two clocks are stopped with all the previous processes.

5. **Communication between the two clock regimes**
   Clock1 and Clock2 scan the operations of the whole control unit. Clock1 controls the "Writer" Process; Clock2 times "Data Maker", "Data Sink" and "Check File" processes. The most complicated communication is the 'data maker - writer' communication: a 4 phases handshake protocol was employed, designed in order to avoid stalls and dead times. When the data maker orders the writer to toggle, the latter sends the signal "toggle_done" to inform the control process that the writing is started. When writing is finished, the data maker answers with an acknowledgement, "ack_count_end". [2]

In figure 7.1, a "PDE" signal inputs to the device under test (DUT), activating a specialized PDE routine if set to '1'. This routine acts on the "SADmin" value, which represents the minimum SAD of the input sequences. As observed in chapter 4.8, the PDE introduces a set of comparators to update the minimum SAD and reduce the critical path of the system in the best match search. The PDE technique can be arbitrarily activated or not and doesn't have input or output apart for the "PDE" signal aforementioned. It only overwrites the value of the minimum SAD in the case the actual SAD is smaller than the temporary one and stops the further steps, letting the system restarts its ordinary cycle.

The line followed in this work is to optimize the best result coming from the last year thesis: the optimized structure is the SAD accelerator shown in figure 4.8 with PDE technique activated. The first testbench was tested for both the conditions, PDE included and PDE excluded, to totally verify the correct working condition of all the architectures combination, but the second testbench will be tested only with PDE included.

Second Testbench

The second testbench is developed, in case of approximate computing, to evaluate the presence of error between the resulting SADs. The process is based on the three blocks of figure 7.1: "Check File" process, "Report" and "Error Evaluation (Output)". The report refers whenever the "done" signal is equal to '1'; then, it prints on the console of Modelsim the resulting SAD value. The check process works on the ".txt" output files where the SAD computation is carried out, without checking the correctness of the two results, as previously made, but critically estimating the error rate coming from the difference between values (error evaluation output block). Simulations are carried out with a subsampling of 4, block size of 64x64 (worst case), search window of $(\pm 64, \pm 64)$, and initial position (0, 0); the total number of operations is 1024, the total number of SADs are: $128 \cdot 128 = 16384$.

Starting from this, the second testbench carries out a verification about the SADs of all the configurations, looking for consistency or discrepancies among them all. The exact SADs (CLA, RCA or basic structure SADs) have been taken as reference in order to evaluate the mismatch between precise structures and approximate motion vector structures where approximate adders was employed. The adopted measure of error, to estimate the difference, is the mean relative error distance (MRED), already introduced in the "Validation" chapter 6. This error rating is introduced in every subsequent table: in this way not just the speed-up but also the related error relation can be observed and critically analyzed. The MRED for the SADs difference is described by the following expression:

$$MRED = \frac{\sum_{i=1}^{N} \frac{PreciseSAD_i - ApproximateSAD_i}{PreciseSAD_i}}{Iterations} \tag{7.1}$$

Where the number of iterations is 1024, imposing in the test bench a sub-sampling of 4 (4096/4=1024). Let's remember that the MRED results is a decimal number which, multiplied by 100, gives the E_{MRED}.

In section 6.1, the C++ analysis was carried out to precisely calculate the error for the approximate structures. The simulations revealed, as expected, that an approximate adder, introducing high approximation, results in an higher error percentage; the two most critical cases were LOA and ETAI, followed by ACAA, ACA and SCSA. In this chapter all the structures will be re-analyze, for all the three substitutions because, even if it's a little redundant, it's useful to perform a verification of the mean relative error distance for all the five approximate adders. In this way, a continuity can be confirmed in both measurements steps, expecting the same classification from the most to the least erroneous. Hence, a similar error measurement is realized for all the architectures with and without approximate adders, substituted in the three positions mentioned in chapter 5. Surely the results of this instance will be different from the previous C++ study because this second analysis is less accurate, because the number of iterations is lower due to considering only the 64x64 case and the presence of sub-sampling. Moreover the C++ code works on a real video sequence while Modelsim testbench was tested on vectors randomly created by Matlab, emulating a video. These second simulations, going to give different error results, offer the opportunity to figure out a classification based on the most erroneous results, putting, for a moment, the percentage of error in background: if both analysis give the same classification, the error ranking will be finally considered in the end study of the performance in dependence of approximate adders.

To be precise, one additive testbench was written to complete the simulations and to let the synthesis possible. This testbench is related to the switching activity report of the architecture and is used to register the switching power consumption of the structure. A section is dedicated to this topic in chapter 8.

7.3 Modelsim Results

First, the basic architecture was tested, verifying the correct working condition. In this instance, only the first testbench was useful, the second would have carry a null error. After this initial step, the approximate adders were introduced in the architecture: as said, to implement them, a new dedicated adder function was created, which address directly to the approximate adder; in every substitution the only change to do is to modify the name of the approximate adder from case to case. After having verified the correctness and working condition of all the combinations, the second testbench was exploit for the error evaluation. The basic structure simulated without approximate adders, with PDE activated, shows precise results, a compilation time around $102\mu s$ and a minimum SAD of 9941. The architecture with the ripple carry adder and the architecture with the carry look-ahead adder substituted show exact results as well, same compilation time and same minimum SAD of the basic structure. The other architectures implementing the approximate adders show, as expected, not accurate results, with different compilation times and different values of minimum SAD. The compilation time in approximate motion slightly decreases, while the value of the minimum SAD assumes an interval of numbers in a neighborhood of the expected minimum SAD.

Obviously, the value of minimum SAD, obtained from the Modelsim simulations, is not enough to make a characterization of the structures; to compare the different architectures, a comparison of all the MRED results is needed. For this purpose, Tables 7.1, 7.2 and 7.3 illustrates the MREDs obtained for the different combinations from the Modesim simulations. More precisely, tables 7.1, 7.2 and 7.3 contain all the informations previously touched on, listing the structures, the related approximate adders implemented, the MREDs calculated with that substitution, and highlight the classification among the structures. The classification stands for the order of decreasing error among

the approximate adders, the first (LOA) is the most erroneous, the last (Basic structure, RCA and CLA) the least.

Table 7.1: Basic structure (BS), RCA, CLA, LOA, ACA, ACAA, ETAI and SCSA Error Informations with 1^{st} substitution (288p video).

1st substitution	BS, RCA, CLA	LOA	ACA	ACAA	ETAI	SCSA
Modelsim MRED	0	0.265	0.198	0.203	0.226	0.181
Classification by Error	6	1	4	3	2	5

Table 7.2: Basic structure (BS), RCA, CLA, LOA, ACA, ACAA, ETAI and SCSA Error Informations with 2^{nd} substitution (288p video).

2nd substitution	BS, RCA, CLA	LOA	ACA	ACAA	ETAI	SCSA
Modelsim MRED	0	0.164	0.114	0.108	0.145	0.097
Classification by Error	6	1	4	3	2	5

Table 7.3: Basic structure (BS), RCA, CLA, LOA, ACA, ACAA, ETAI and SCSA Error Informations with 3^{rd} substitution (288p video).

3rd substitution	BS, RCA, CLA	LOA	ACA	ACAA	ETAI	SCSA
Modelsim MRED	0	0.139	0.087	0.084	0.137	0.050
Classification by Error	6	1	4	3	2	5

These simple table lead to a complete panorama of the architectures, similarly to the C++ study, and confirm the error classification. The fact that the classification of the error is constant between the two studies (C++ and Modelsim) is a very positive datum, giving the go-ahead to the synthesis phase. These structures will be synthesized, to be able to observe the improvement/worsening brought using an approximate adder to the detriment of a greater/smaller error and a loss/gain of precision. LOA confirms to be the most erroneous approximate adder, followed by the ETAI. ACAA represents the trade-off of the ranking, together with ACA which shows a very similar result with respect to the ACAA; SCSA is the most correct solution. From the synthesis, one can expect that SCSA with RCA and CLA shows worse result in terms of power consumption and frequency (or timing) than faster approximate adders like LOA or ETAI, which probably offset the higher error distance with better performance.

Chapter 8

Synthesis

The synthesis of an electronic circuit is a process through which an abstract (RTL level) is turned into a design implementation in terms of logic gates, using a synthesis tool. The synthesis process gives the possibility to collect all the informations about the logic design of the studied circuit starting from the VHDL code: timing, frequency, area, gates and power consumption are the main data that can be extracted in order to compare the different implementations of the SAD accelerator. The logical synthesis process can be divided into the following steps:

1. Setup of the synthesis;
2. Reading of the VHDL source files;
3. Application of the constraints;
4. Design compilation;
5. Saving of the results.

1. Setup: before starting the synthesis, "*.synopsys_dc.setup*" file needs to be setup: this file contains the names of the technology libraries used to perform the synthesis and the path to find them in the file-system. The employed libraries are four: the CMOS 65nm standard cell technology library, a proper IP library (contained in the 65nm library) for the SRAM, the Gtech and the Designware libraries. It was chosen to use a low power library. The SAD accelerator has been subjected to a synthesis operation, firstly without the introduction of approximate adders, and after substituting the approximate adders, in order to appreciate the improvements or drawbacks introduced in the design. The accelerator was synthesized with Synopsys Design Compiler (DC) on a CMOS 65nm standard cell technology.

2. Reading: the VHDL code of the desired architecture is read through an analysis step and, then, it is elaborated; in this phase, it's important to specify the library where Design Compiler will put the working files.

3. Constraints: timing, clock uncertainty and signal's delay are the three main constraints required. The timing need to be selected to set operations with a specific clock. The clock could be affected by jitter, therefore a clock uncertainty is fixed; moreover, each input signal could arrives with a certain delay with respect to the clock and, because of this, assuming that all input signals have the same maximum input delay, a delay is set. First, the timing is imposed as a constraint and two clock regimes were fixed:

- Clock1 = 1280 MHz;
- Clock2 = 160 MHz.

Chapter 4 gives important informations about timing limits: Clock2 can reach frequency a maximum of 200 MHz (SRAM library). Hence, one could fix Clock2 = 200 MHz and Clock1 = 1600 MHz but it was not possible to set a Clock1 of 1600 MHz for the SRAM input port 1 because it is not supported by that library IP (it shows a maximum of 1300 MHz). So, the hypothesis made in chapter 4 were no longer valid. To maintain a fixed clock's ratio of 8 between Clock1 and Clock2, Clock2 has been set to 160 MHz and the value of Clock1 consequently.

4. Compilation: the design compilation, which is the begin of the synthesis, can be carry out with instructions on the Design compiler's console or through prepared scripts. Inside the scripts one can add all the constraints, all the compilation steps and the final report instructions needed to deeply study the design. The next section is dedicated to this phase.

5. Saving: the reports collect the results of the simulations. The data that can be saved are: timing, area, design check, libraries, critical path, power consumption, error's informations or maybe other contents (it depends on the requirements). To ensure that the design is working without problems the time slack must be positive; the slack is defined as the difference between data required time and data arrival time, so, it represents the remaining interval between the time the system provides minus the effective working time. A positive slack implies that the arrival time can be further increased (theoretically by a quantity equal to the difference between data required time and data arrival time), without affecting the overall delay of the circuit. Conversely, negative slack implies that a path is too slow, and the path must be speed up (or the reference signal delayed) if the whole circuit has to work at the desired speed. Among all the reports, timing, area and power consumption are the parameters investigated in the following sections.

The standard cell components are taken from a low power library with typical working conditions: 1.2 V, 25°C, typical process (TT). The SRAM IP was chosen to be adaptable to the standard cells without the level shifters. It is a low power memory with the same working condition of the standard low power library and a 16x128 array. The use of a low power library comes from the need to avoid too high power consumption due to high parallelism of the architecture. Been strongly parallelized, the power usage of the SAD accelerator is likely to be too high with several components working in parallel at the same time. [2]

Furthermore, it was performed a research regarding the synthesized adders, the ones will be substituted by the approximate adders in a second moment inside the adder tree of the VHDL code. How does the synthesizer translate the simple sums can be discovered through a small series of instructions. The code is firstly read and the constraints are fixed, timing, delay and clock's uncertainty; then, the cells needs to be ungrouped to flatten the hierarchy through the command "ungroup -all -flatten"; whereupon, the compilation is launched. With the instruction "report_resources", the cells and modules are listed and theirs current implementations too. Design Compiler synthesized the adders as rpl (ripple carry adder). This is interesting because, theoretically, increasing the clock frequency the synthesizer changes the circuit to achieve the new constraint and it may use faster adder's architectures, like the carry-look-ahead adder.

The structures analyzed correspond to the best resulting structures obtained by Paolo Selvo in his work of thesis [2]; the task of the synthesis in the following chapters is the attempt to improve the performance, going to create the best possible and complete algorithm in terms of timing, resources, critical path, and power dissipation. Therefore, the starting accelerator considered is the basic structure implementing the PDE, without MSBs or LSBs optimization, which represent the best outcome

of ref.[2]. In a dedicated chapter, chapter 8.5, among the particular cases, the basic structure without optimization, PDE included, is studied for a reduced number of cases to give a general idea of the performance improvement in the not optimized case.

8.1 Compilation's scripts

The synthesis was carried out with three incremental compilations with ultra options:

1. The first compilation tries just to respect timing and compiles all the netlist without area and power constraints. Two instructions are required for this purpose:

   - set compile_ultra_ungroup_small_hierarchies FALSE;
   - compile_ultra -no_autoungroup.

   In this way it's possible to read the true name of the components inside the several reports and also to perform a hierarchical power estimation, because the compilation doesn't destroy the hierarchy. After, the tool performs the automatic re-timing on the pipeline stages and prints the first reports on area and timing.

2. The second compilation is incremental and optimizes the area thanks to the commands:

   - set_max_area 0;
   - compile_ultra -incremental.

   After, the reports about area, timing and constraints are generated.

   The next step concerns the power evaluation and it is necessary to get the netlist switching activity in order to estimate the dynamic power. The switching activity captures the toggling activity of all the circuit nodes of the design. Therefore, the netlist generated by Design Compiler is saved on a Verilog file while the delays are listed in a .sdf file. A proper Modelsim simulation is carried out for estimating the switching activity of all the accelerator nets and these values are written by the simulation tool into a $.saif$ file.

3. The third and last phase reads the $.saif$ file and prints a power report. Then, it sets as constraints the static and dynamic power and performs the last compilation:

   - set_dynamic_optimization true;
   - set power_opto_extra_high_dynamic_power_effort true;
   - set_max_leakage_power 0;
   - set_max_dynamic_power 0;
   - compile_ultra -incremental.

   then, it generates a second power report with a timing and area reports. The power report is hierarchical so that it is possible to understand which components consume more. [2]

These three compilations are collected inside three scripts, one for the timing/area reports that contains the first two compilations and two for the power, containing the last one but requiring the first script. In the end, a post-synthesis simulation is done to ensure the synthesis gave no error.

Ultra compilation commands has been used instead of ordinary compile instruction because it performs a higher effort compilation giving higher quality of results. This implies a better optimization in terms of area, timing and power with respect to the compile command. Ultra instructions incorporate

innovative topographical technology which allow a predictable flow resulting in faster time to results. Of course, the ultra command can be exploit in the same way as the compile command. The optimizations introduced by the ultra compilation are not compatible with the "design_resources" command; this instruction requires the ordinary compilation because the system must be free of optimization to have a clear description of the designed behavior.

8.1.1 Power Evaluation Step

One crucial step in the previous explained compilation is the generation of the *.saif* file for the power evaluation. Here, Modelsim and Design Compiler are alternated to calculated the switching activity of the system. A modified testbench (similar to the first one of chapter 7) is employed in order to register the switching activity of all nets. All the steps are listed below:

1. The testbench calls a Verilog component which contains the commands to start and stop the annotation of the activity. The commands are from PLI (Program Language Interface);
2. A process reads the two *.saif* files generated by Design Compiler, derived from the two .db library files (one is the standard cell components' library and the second is the SRAM IP library);
3. When the simulation is finished (signal "end_sim" = '1'), another process stops the annotation and writes all the data into the file *.saif*;

The testbench in step 1 is the very similar to the Modelsim testbench previously employed, so, it works on a test vectors of 1024 64x64 SADs. It is recommended to use a 64x64 SAD since it represents the worst case SAD and the one that stresses more, activating all PEs. A test vector of 1024 64x64 SADs is sufficient to solicit all the architecture nets and, therefore, to have an accurate estimation.

In the next sections the synthesis of the design are completed for the 64x64 case. They are used to compare all the architectures, implementing the three substitutions and the approximate adders, in terms of frequency, area and power.

8.2 Frequency Study and Timing Constraints

To perform a full characterization of the proposed system, the synthesis has been carried out starting from a frequency analysis. Having a set of possible architectures working at different frequencies, having different areas and different consumptions, it's necessary to fix a common point, a common constraint. The standard way to act is to, at first, calculate the minimum period, to which corresponds the maximum frequency, for every structure and, after, fix a common timing constraint, like, for example, the larger period (to which corresponds the lower frequency) as common clock. In this way all the combinations will surely work and a common clock regime can be imposed. In this work, the SRAM library is a limiting factor, which holds the Clock2 to a f_{max} of 200 MHz. This constraint cannot be neglected, first because the memory bandwidth would not be respected with higher frequencies (bandwidth fixed at 50 GB/s) and, second, to maintain a common guideline between this work and the work of Paolo [2], the frequency of 200 MHz must be kept the same, or lower. Moreover, the SRAM ports depending on Clock1 (1600 MHz) don't need to increase their speed more than 8 times because, with a ratio of 8, both parts can write/read inside the memory in 1 clock cycle.

The solution has been to choose the two following clock constraints: Clock1 = 1280 MHz and Clock2 = 160 MHz. Clock1 is related to the speed of the SRAM, while Clock2 is related to the SAD interface and SAD Datapath part of the SAD accelerator (see figure 4.7 in section 4.7). A Clock2 of 160 MHz derives from the memory bandwidth limit, and from the fact that it is not necessary to get higher speed solutions; high spped solutions could uncontrollably increase the power, since the

architecture is strongly parallelized and many components are working at the same time. This choice is directly related to the low power library.

It's true that the system is theoretically able to work at higher frequencies, regardless of the consumption, because the architecture has several pipeline stages. Keeping a constant Clock2 of 160 MHz, f_{CLK1} can be increased and its maximum value can give a general trend of the systems' speed in the several implementations. The maximum frequencies of the architectures has been calculated during the first steps of the synthesis and the results are illustrated in table 8.1. The employed substitution is the first; none of the others has been used as this is only a rough estimation.

Table 8.1: Maximum achievable frequencies of the architecture employing the first substitution only.

	BS	RCA	CLA	LOA	ETAI	ACAA	ACA	SCSA
$\mathbf{f_{CLK1}[GHz]}$	2.13	2.13	1.8	2.44	2.42	2.33	2.17	2.22

These results are useless for the synthesis itself but draw up a small brief classification of the possible frequencies the architectures are able to reach. The consequence of pushing the accelerator to work at such speeds is an increase of the areas because, if the minimum period is decreased, the synthesizer has to find the best trade off between timing, area and power consumption: if the power remains unchanged, the area must increase. Equivalently, if the area remains constant, the power dissipation increases. Considering the table 8.1, the SAD accelerator synthesized at $f_{CLK1} = 2.13$ GHz has a total cell area of 1.29 mm^2, and with LOA ($f_{CLK1} = 2.44$ GHz) reaches a total cell area of 1.30 mm^2.

Imposing high frequencies, it's possible to test how the automatically compiled adders are synthesized. As said, with the instruction "report_resources" the adders are listed and theirs current implementations too. For lower frequency Design Compiler synthesized the adders as ripple carry adders; pushing the system to high frequencies, the synthesizer should change the implementation in order to support the new constraint and it should use faster adders, like the carry-look-ahead adder. In reality, even if the system is strongly parallelized, the adders works on bit sequences up to 20 bits, which is unsuitable for CLA, optimized for very highly parallelized architectures and long bit sequences additions. In fact, the report of the "report_resources" commands always gave for both low and high frequencies the same result: ripple carry adders.

In the next step, instead of synthesizing the accelerator at frequencies higher than 2 GHz or more and using it at 200 MHz maximum, it was chosen to directly select a frequency of 160 MHz so that to have less area (same approach of [2]). Fixing $f_{CLK2} = 160$ MHz and $f_{CLK1} = 1280$ MHz the cell area reduces to $\sim$120 mm^2. A more detailed area study is carried out in the following section.

8.3 Areas and Number of Gates

Table 8.2 illustrates the results regarding the system's area.

As it can be seen from the table, the highest amount of area is due to the Black Box that in this project consists in the SRAM memories: 1.07 mm^2 of the total cell area are occupied by the 3 SRAM units. They occupy the $\sim$ 90% of the total area, since they are 48 blocks of 16 locations and 128 bits of parallelism (in total 98304 bits). The remaining area is occupied by 0.09 mm^2 of combinational area, a very small area of buffers/inverters and 0.04 mm^2 of non combinational area. The Net Interconnect area was not estimated because the wire load has been assumed to have zero net area. These quantities are in line with the synthesis of the reference [2]: the Datapath represents the 8-9% of the total area, the Interface control logic only the 0.1%, the SRAM units together with the circuitry that controls the enables of their inputs occupy more than the 90% and the 2048-bit

Table 8.2: Area of the SAD Accelerator

Combinational Area	90412.20 μm^2 = 0.09 mm^2
Buf/Inv Area	4007.52 μm^2 = 0.004 mm^2
Non Combinational Area	40596.60 μm^2 = 0.04 mm^2
Macro/ Black Box Area	1071851.41 μm^2 = 1.07 mm^2
Net Interconnect Area	Undefined
Total Cell Area	1206980.13 μm^2 = 1.2 mm^2

multiplexer that brings the reference samples from the SRAM memory to the Datapath is a 0.6%; the remaining 0.4% is taken by the Minimum SAD Updater.

These results of the synthesis are valid for the SAD accelerator structure without approximation but can be generalized. The substitutions of all the approximate adders' cases in the different positions didn't return appreciable mismatch from one area to the other. The general trend of the synthesis with or without approximation is to occupy an area of approximately 1.20 mm^2. Some variation is present but it's negligible: values oscillate from 1.19 to 1.21 mm^2. Detailed informations are contained in table 8.3.

Table 8.3: Area of the SAD Accelerator in the several cases

Total Area [μm^2]	1^{st} substitution	2^{nd} substitution	3^{rd} substitution
RCA	1198678.89	1207457.13	1211045.25
CLA	1212051.81	1212051.81	1211575.53
LOA	1193862.81	1199014.77	1193283.57
ETAI	1194739.77	1198828.29	1192914.93
ACA	1209954.09	1208822.97	1220654.01
ACAA	1199484.93	1199260.29	1198904.97
SCSA	1201809.45	1207538.13	1219258.29

Being on average constant, the area cannot represent a comparison term but can be exploited to compute the number of gates. The gate number is generally calculated as the ratio of the system's area divided by the area of the smallest NAND2 gate (value extracted from the library). The system's area takes into account the total cell area, deprived of the memories' areas. For example, in the case of the basic structure each SRAM unit occupies 358950 μm^2, so the obtained value of gate number is:

$$N_{gate} = \frac{Area_{Total} - Area_{Memory} \cdot N_{memories}}{Area_{NAND2}} = \frac{1206980.13 - 358950 \cdot 3}{1.44} = 90.4K \simeq 90Kgates \tag{8.1}$$

The use of approximate adders does not significantly impact in terms of hardware resources, indeed, the chip area using adders like LOA or ETAI decreases with respect to the basic accelerator, although negligibly. What really changes from BS to the approximate adders' cases is the gate number. The number of gates for the eight architectures and the three substitutions are collected in table 8.4.

While the use of approximate adders didn't significantly impact in terms of hardware resources, this is not true for the gate number. When considering the gate count, the implementation of approximate adders causes a critical increase of the number of gates, while the area remains approximately constant. The explanation for this is related to the synthesis tool: Design Compiler uses complex

Table 8.4: Gates count for all the architectures in the three case studies

Number of Gates [KGates]	1^{st} substitution	2^{nd} substitution	3^{rd} substitution
BS	90.4		
RCA	90	90.5	90.4
CLA	94.7	94.7	94.4
LOA	102.1	95.7	101.7
ETAI	96.1	93.3	93.4
ACA	93.3	92.5	100.7
ACAA	96	95.8	95.6
SCSA	90.4	94.4	102.6

gates, specialized in ripple carry adder operations, available in the libraries; as a consequence, the RCA version of the adders synthesized by the synthesizer will use larger chip area but with fewer gates. [19]

8.4 Power Results of the Synthesis

The third presented result is the power evaluation of the basic structure without any approximate adder implemented, illustrated in table 8.5. The total dynamic power is the most relevant parameter because it takes every power contributions of the system into account; it's defined as the sum of the three main power factors: cell internal power, switching power and the leakage power (usually very small, 3 order of magnitude smaller than the other contributions).

Table 8.6 shows in depth the higher and smaller power group's consumptions of the four previously cited quantities.

Table 8.5: Basic structure (BS) power summary.

SAD Accelerator BS	
Cell Internal Power	114.60 mW (90%)
Net Switching Power	13.33 mW (10%)
Total Dynamic Power	127.94 mW (100%)
Cell Leakage Power	108.39 μW (<1%)

Table 8.6: Basic structure (BS) complete Power Report, including the Power Group's consumption. As can be easily guess, the "Total" line contains the values illustrated also in table 8.5, with the only difference that the total dynamic power includes also the leakages and, so, it's a little higher.

Power Group	Internal Power [mW]	Switching Power [mW]	Leakage Power [μW]	Total Power [mW]
Memory	92.06	0.25	97.77	92.41 (72.17%)
Register	15.00	5.45	3.32	20.44 (15.97%)
Combinational	7.54	7.64	7.32	15.19 (11.86%)
Total	114.60	13.33	108.39	128.04

Tables 8.5 and 8.6 highlight the following points:

- The cell internal power consumes the 90% of the total dynamic power, mainly because it takes into account the SRAM consumption.

- The switching activity registers a power dissipation of around 10%; the switching activity catches the power activity of all the circuit nodes, it's a function of pin capacitance, voltage, and toggle frequency.
- The 72.17% of the total power is related to the SRAM memory banks: this percentage accounts that the writing ports works at a frequency of 1280 MHz (8 times the reading ports). Hence, the memory accounts every single sample written in memory at a superior frequency, taking into consideration that the reference PB is loaded and overwritten several times during the motion prediction.
- The dynamic power ($\sim$ 128 mW) assumes quite high values, in line with the power estimation of ref.[2]. The explanation of such dissipation is connected to its subdivision into memory, register and combinational consumptions: it must highlighted that the designed part of the architecture, the effective SAD accelerator, has a power dissipation of 35.63 mW. This number is calculated only from the register and combinational contributions because the remaining contribution is attributed to the memory (SRAM IP) consumption.
- The cell leakage power, which, as can be guessed from its name, represents the leakage power, is almost negligible and doesn't influence the design's performance, showing a percentage under the 1%. This small value is probably related to the employment of a low power library.

The same tables may be drawn for the other implementations and substitutions. Table 8.7 illustrates all the results of the power evaluation for all the possible implementations. Just to remind, the first substitution consists in the insertion of the approximate adders in the complete adder-tree, the second is the substitution in a part of the adder-tree, out of the processing elements, from the 12^{th} bit onwards, and the third substitution is applied on the subtractors inside the processing elements.

Table 8.7: Summary table for the total power consumption of all the combination of approximate adders and substitution, measured in mW.

	1^{st} substitution	2^{nd} substitution	3^{rd} substitution
BS	127.94 mW		
ACA	123.56 mW	128.14 mW	128.93 mW
ACAA	121.13 mW	123.90 mW	125.45 mW
CLA	129.67 mW	129.67 mW	128.18 mW
RCA	128.07 mW	128.02 mW	128.14 mW
ETAI	120.77 mW	121.87 mW	121.88 mW
LOA	120.76 mW	121.79 mW	121.74 mW
SCSA	124.07 mW	127.99 mW	128.78 mW

As can be seen from table 8.7, the power consumption of the combinations depends on approximate adder as well as the substitution. As repeated several times, the larger the error rate, the best will be the performance: highly approximate architectures, such as ACAA, LOA and ETAI, give appreciable power saving results with respect to the other cases. The larger power gap (9 mW) is shown by the couple CLA-LOA and the most interesting gap is shown by the couples BS-LOA or BS-ETAI and RCA-LOA or RCA-ETAI ($\sim$ 7 mW in the three cases). A power gap of 7 mW between accurate and the best result of inaccurate implementation is equivalent to an average 6% of power variation. More precise results are discussed in the three following sections, where the three substitutions are deeply analyzed and more detailed power values are illustrated.

At the beginning of chapter 8, the design resources has been briefly commented about the implementation of the adder by the synthesizer. Ripple carry adder is the chosen implementation inside the basic structure and this is confirmed by table 8.7, where RCA assumes in the three substitutions always the same result; it's not perfectly matched in the three cases because the hand-written code is different from the RCA code of the "DesignWare" library but the results are in line with the expectations (only the decimals look different, hundreds of μW).

Unexpected is the power evaluation given by the Carry look-ahead adder. On the one hand, the CLA is an optimized version of the RCA in terms of computational time due to no delay accumulated because of the ripple-carry chain. On the other hand, the CLA is specialized for highly parallelized architecture working with 32 or 64 bits. The proposed accelerator has a strongly parallelized design but works with an interval of 8 to 16 bits, up to 20 in the accumulation phase. The inefficiency of this adder is surely related to the smaller input sequences, that doesn't allow a convenient expenditure of energy in the computation of the propagate, generate and kill signals and subsequent sum.

Unexpected results also come from the power evaluation of the ACA e SCSA in the subtractor substitution: going to analyze the netlist output file of the synthesis it's possible to verify that some node of the computational calculus of the synthesis requires more time than others. This means that the error generation, due to approximation, slows down the synthesis and increases the hypothetical power load. There are some calculations, in specific node of the synthesis, which carries an error caused by the approximation that make the calculus more complex than a precise calculations: in these instances, the synthesizer is so busy to increase the value of power consumption in an unexpected way.

The next three section are the sections dedicated to a deeper analysis of the results coming from the three substitutions. The analysis comments all the obtained results and tries to figure out the best configurations, for the moment without taking into account the error measurements. For example, at a first glance, the best configuration for power evaluation and maximum frequency is surely the SAD accelerator implementing the LOA. The main limit of such adder is the quite high error introduced during the SADs computation but, on the other hands, the power reduction needs to be underlined. In the chapter 9, a summary, considering all the quantities (frequency, area, power and error rate), is developed to give a clearer view of the possibility given employing approximate adders.

8.4.1 First Substitution

Tables from 8.8 to 8.14 illustrates the results obtained from the synthesis of the SAD accelerator implementing the first substitution in the 7 possible architectures (7 adders, 2 precise and 5 imprecise). A first positive aspect is related to the almost constant ratio between the power contributions which follows the analysis aforementioned regarding the power division:

- The cell internal power consumes the 90-91% of the total dynamic power because it accounts the SRAM consumption, which represents the 70-75% of the total power.
- The switching activity registers a power activity of the circuitry nodes of around 10%.
- The total dynamic power summaries all the consumptions, assuming a large variety of values, depending on the architecture.
- The cell leakage power is negligible in all the cases, maintaining a contribution smaller than 1% thanks to the low power library.

Before starting the power study, it must be highlighted that the BS in these three subsections is annexed to the RCA. From table 8.7, it can be seen that the values assumed by BS and RCA slightly

Table 8.8: Power analysis implementing ACA

Almost Correct Adder (ACA)	
Cell Internal Power	110.23 mW (90%)
Net Switching Power	13.33 mW (10%)
Total Dynamic Power	123.56 mW (100%)
Cell Leakage Power	107.24 μW (<1%)

Table 8.9: Power analysis implementing CLA

Carry Look-Ahead Adder (CLA)	
Cell Internal Power	115.73 mW (89%)
Net Switching Power	13.95 mW (11%)
Total Dynamic Power	129.67 mW (100%)
Cell Leakage Power	106.63 μW (<1%)

Table 8.10: Power analysis implementing RCA

Ripple Carry Adder (RCA)	
Cell Internal Power	115.01 mW (90%)
Net Switching Power	13.07 mW (10%)
Total Dynamic Power	128.07 mW (100%)
Cell Leakage Power	107.61 μW (<1%)

Table 8.11: Power analysis implementing LOA

Lower-OR Part Adder (LOA)	
Cell Internal Power	108.41 mW (90%)
Net Switching Power	12.35 mW (10%)
Total Dynamic Power	120.76 mW (100%)
Cell Leakage Power	106.12 μW (<1%)

Table 8.12: Power analysis implementing ACAA

Accur. Config. Approx. Adder (ACAA)	
Cell Internal Power	108.38 mW (89%)
Net Switching Power	12.75 mW (11%)
Total Dynamic Power	121.13 mW (100%)
Cell Leakage Power	106.52 μW (<1%)

Table 8.13: Power analysis implementing SCSA

Speculative Carry Select Adder (SCSA)	
Cell Internal Power	110.22 mW (90%)
Net Switching Power	13.85 mW (10%)
Total Dynamic Power	124.07 mW (100%)
Cell Leakage Power	106.63 μW (<1%)

Table 8.14: Power analysis implementing ETAI

Error-Tolerant Adder (ETAI)	
Cell Internal Power	108.38 mW (90%)
Net Switching Power	12.38 mW (10%)
Total Dynamic Power	120.77 mW (100%)
Cell Leakage Power	106.13 μW (<1%)

differs because the RCA is not the automatically version implemented but the code written by the author of this thesis work. This choice is necessary to place all the adders on the same level of analysis, not discriminating the approximate cases with respect to the automatically synthesized. Moreover, RCA doesn't differ too much from the BS power consumption, on the contrary, it assumes in the three substitutions always approximately the same result with a mismatch from BS smaller than 200 μW.

The larger power saving value, with respect to the RCA, is given by the LOA and ETAI configurations, whose total dynamic power differs for 0.01 mW = 10 μW. The power saving results to be 8.61 mW, equivalent to the 7% of the total power. Saving the 7% of the overall consumption is a remarkable result, in particular if we compute it not with respect to the entire dynamic power but with respect to the designed part of the accelerator. The memory unit consumes more than 70% of the total power; neglecting this percentage and considering only the consumption of the designed

architecture, 8.61 mW are equivalent to the 23.37% of the total consumption. This percentage is obtained from tables 8.15 and 8.16, from the following calculus:

$$Power\ Saving_{\ Designed\ Part} = 1 - \frac{Power\ LOA_{\ Designed\ Part}}{Power\ RCA_{\ Designed\ Part}} = 1 - \frac{21.32 + 6.69}{20.44 + 16.11}$$

where the "$Power\ LOA_{\ Designed\ Part}$" and "$Power\ RCA_{\ Designed\ Part}$" are the sum of the register+combinational power consumptions of tables 8.15 and 8.16, memory excluded.

Table 8.15: RCA complete Power Report, including the Power Group's consumption.

Power Group	Internal Power [mW]	Switching Power [mW]	Leakage Power [μW]	Total Power [mW]
Memory	91.02	0.22	97.77	92.26 (71.63%)
Register	14.99	5.14	3.32	20.44 (15.87%)
Combinational	8.10	7.51	6.53	16.11 (12.51%)
Total	115.65	13.71	107.60	129.88

Table 8.16: LOA complete Power Report, including the Power Group's consumption.

Power Group	Internal Power [mW]	Switching Power [mW]	Leakage Power [μW]	Total Power [mW]
Memory	91.23	0.52	97.77	91.85 (75.99%)
Register	3.55	7.78	2.96	21.32 (17.64%)
Combinational	3.51	3.17	5.37	6.69 (5.54%)
Total	108.41	12.35	106.12	120.87

The best compromise is surely given by the ACAA, which ensure middle value of power saving and good results also in terms of error, representing the trade-off among the architectures. The ACAA shows a power gap of 8.24 mW for this substitution. This power gap corresponds to the 6.4% of the total power and the 21.2% of the designed part total power (the entire tables with the memory, register and combinational contributions of the ACAA and other adders are not depicted for sake of simplicity nor here neither in the next sections).

ACA and SCSA shows the opposite point of view with respect to LOA, ETAI and ACAA, returning quite high power dissipation, consequently small power saved. Their power gap corresponds to less than the 4% of the total power. In fact, these two architectures ensures low error rate than good performance. This aspect is visible in the characteristic of figure 8.1, where the trend of the approximate adders with respect to the total dynamic power is illustrated. This bar chart gives a general panorama of the behavior of the architecture in this 1^{st} substitution depending on the approximate adder implemented.

It needs to be highlighted that this first substitution is the case showing the best power gaps and this is a result in line with ref.[17], one of the few articles nowadays present in the literature concerning this topic. In the next substitutions, the light approximations, such as ACA and SCSA, will exhibit smaller impact and percentages.

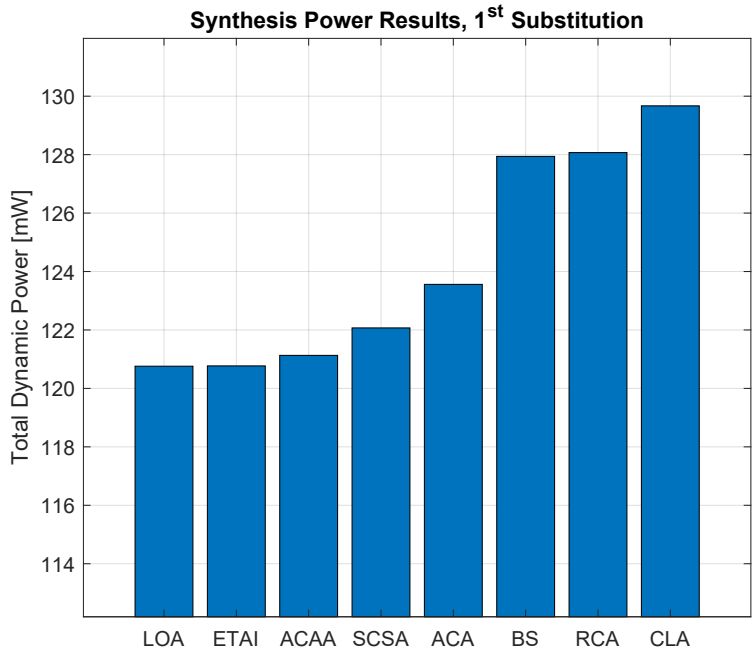


Figure 8.1: Summary of the total power consumption for all the structure with/without approximate adders, first substitution

8.4.2 Second Substitution

Tables from 8.17 to 8.23 illustrates the results obtained from the synthesis of the SAD accelerator implementing the second substitution in the 7 possible architectures (7 adders, 2 precise and 5 imprecise). The first impression related to the 2^{nd} approximation is a power reduction of at least 2mW for each cases with respect to the 1^{st} substitution, with a proportional reduction of the error value for each cases. Also in this phase, the ratio between the power contributions is almost constant and the analysis seems to be strict regarding the power division:

- The cell internal power is the ∼90% of the total dynamic power.
- The switching activity is around the 10%.
- The cell leakage power is negligible in all the cases, smaller than 1%.

Table 8.17: Power analysis implementing ACA

Almost Correct Adder (ACA)	
Cell Internal Power	114.73 mW (90%)
Net Switching Power	13.41 mW (10%)
Total Dynamic Power	128.14 mW (100%)
Cell Leakage Power	108.48 μW (<1%)

Table 8.18: Power analysis implementing CLA

Carry Look-Ahead Adder (CLA)	
Cell Internal Power	115.73 mW (91%)
Net Switching Power	13.95 mW (9%)
Total Dynamic Power	129.67 mW (100%)
Cell Leakage Power	108.63 μW (<1%)

Table 8.19: Power analysis implementing RCA

Ripple Carry Adder (RCA)	
Cell Internal Power	114.66 mW (90%)
Net Switching Power	13.36 mW (10%)
Total Dynamic Power	128.02 mW (100%)
Cell Leakage Power	108.41 μW (<1%)

Table 8.20: Power analysis implementing LOA

Lower-OR Part Adder (LOA)	
Cell Internal Power	109.08 mW (90%)
Net Switching Power	12.71 mW (10%)
Total Dynamic Power	121.79 mW (100%)
Cell Leakage Power	106.50 μW (<1%)

Table 8.21: Power analysis implementing ACAA

Accur. Config. Approx. Adder (ACAA)	
Cell Internal Power	110.07 mW (89%)
Net Switching Power	13.73 mW (11%)
Total Dynamic Power	123.90 mW (100%)
Cell Leakage Power	106.52 μW (<1%)

Table 8.22: Power analysis implementing SCSA

Speculative Carry Select Adder (SCSA)	
Cell Internal Power	114.63 mW (90%)
Net Switching Power	13.37 mW (10%)
Total Dynamic Power	127.99 mW (100%)
Cell Leakage Power	108.42 μW (<1%)

Table 8.23: Power analysis implementing ETAI

Error-Tolerant Adder (ETAI)	
Cell Internal Power	108.99 mW (90%)
Net Switching Power	12.87 mW (10%)
Total Dynamic Power	121.87 mW (100%)
Cell Leakage Power	106.48 μW (<1%)

In this case, the larger power saving, always with RCA as reference, is given by LOA, ETAI and ACAA architectures. The power saving results respectively equal to 6.23 mW, 6.30 mW and 4.12 mW, equivalent to respectively the 4.8%, 4.7% and 3.3% of the total power in the three cases. As introduced before, the real power saving is higher if the designed section of the architecture is considering, neglecting the memory. The power saving with respect to the designed part is on average equal to the 15% in the three cases. The power saving is reduced in favor of an error reduction because the substitution is more rigid.

Figure 8.2 shows the trend of the substitution.

Rather than observing the percentages of change, in this instance it is more interesting to observe the quasi-constant behavior of SCSA, ACA, Basic structure (BS), RCA and CLA. In this instance, there is negligible, or rather very small gain in terms of power for architectures implementing ACA or SCSA.

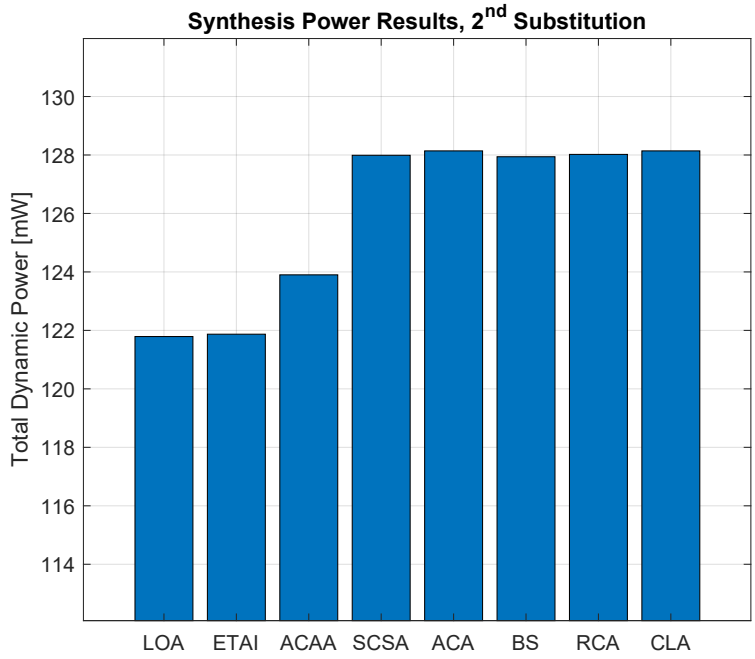


Figure 8.2: Summary of the total power consumption for all the structure with/without approximate adders, second substitution

8.4.3 Third Substitution

Tables from 8.24 to 8.30 illustrates the results obtained from the synthesis of the SAD accelerator implementing the third substitution in the 7 possible architectures. This substitution is the most unpredictable because the subtractors' substitution implies that some node of the computational calculus of the synthesis requires more time than others, as explained in section 8.4: the error generated by the approximation slows down the synthesis and increases the hypothetical power load. In fact, ACA and SCSA show very high power consumption. As the previous substitutions, the power is on average distributed in:

- The cell internal power is the ∼90% of the total dynamic power.
- The switching activity is around the 10%.
- The cell leakage power is negligible in all the cases, smaller than 1%.

Table 8.24: Power analysis implementing ACA

Almost Correct Adder (ACA)	
Cell Internal Power	114.97 mW (90%)
Net Switching Power	13.95 mW (10%)
Total Dynamic Power	128.93 mW (100%)
Cell Leakage Power	108.96 μW (<1%)

Table 8.25: Power analysis implementing CLA

Carry Look-Ahead Adder (CLA)	
Cell Internal Power	114.44 mW (89%)
Net Switching Power	13.74 mW (11%)
Total Dynamic Power	128.18 mW (100%)
Cell Leakage Power	108.47 μW (<1%)

Table 8.26: Power analysis implementing RCA

Ripple Carry Adder (RCA)	
Cell Internal Power	114.44 mW (90%)
Net Switching Power	13.70 mW (10%)
Total Dynamic Power	128.14 mW (100%)
Cell Leakage Power	108.46 μW (<1%)

Table 8.27: Power analysis implementing LOA

Lower-OR Part Adder (LOA)	
Cell Internal Power	109.31 mW (90%)
Net Switching Power	12.43 mW (10%)
Total Dynamic Power	121.74 mW (100%)
Cell Leakage Power	106.09 μW (<1%)

Table 8.28: Power analysis implementing ACAA

Accur. Config. Approx. Adder (ACAA)	
Cell Internal Power	112.84 mW (90%)
Net Switching Power	12.62 mW (10%)
Total Dynamic Power	125.45 mW (100%)
Cell Leakage Power	106.69 μW (<1%)

Table 8.29: Power analysis implementing SCSA

Speculative Carry Select Adder (SCSA)	
Cell Internal Power	114.53 mW (89%)
Net Switching Power	14.26 mW (11%)
Total Dynamic Power	128.78 mW (100%)
Cell Leakage Power	106.92 μW (<1%)

Table 8.30: Power analysis implementing ETAI

Error-Tolerant Adder (ETAI)	
Cell Internal Power	109.41 mW (90%)
Net Switching Power	12.47 mW (10%)
Total Dynamic Power	121.88 mW (100%)
Cell Leakage Power	106.03 μW (<1%)

Third substitution is the first revealing negative power gaps with respect to the reference RCA. Like in the first and second substitution LOA and ETAI give the best power saving, followed by the ACAA which imposes itself as trade-off. ACA and SCSA are not suitable for this substitution. RCA and BS reach, of course, the same value, accompanied by the CLA which is less affected by this third substitution than the previous two.

The power saving shown by LOA and ETAI results respectively equal to 6.4 and 6.26 mW, equivalent to approximately the 5% of the total power. The effective power saving is higher considering only the designed architecture, without the memory. In such conditions, the power saving is on average equal to the 18.5%. This is a very good result of power saving, also considering the reduced error rate calculated in chapter 6. At the same time, it is not surprising because this substitution is one of the few applications of approximate adders in decoders present in the literature. In ref.[19] the power saving percentage between the analyzed accelerator implementing the LOA and the RCA architecture shows a value around the 13%. In this article, no memory system is implemented, meaning that this 13% can be compared with the 18.5% obtained in this case. The two results are quite in line with each other.

Figure 8.3 shows the trend of the substitution.

In the figure, Basic structure (BS), RCA and CLA assume the same value of dynamic power dissipation. Both ACA and SCSA reach higher peaks, highlighting their unsuitability with this substitution.

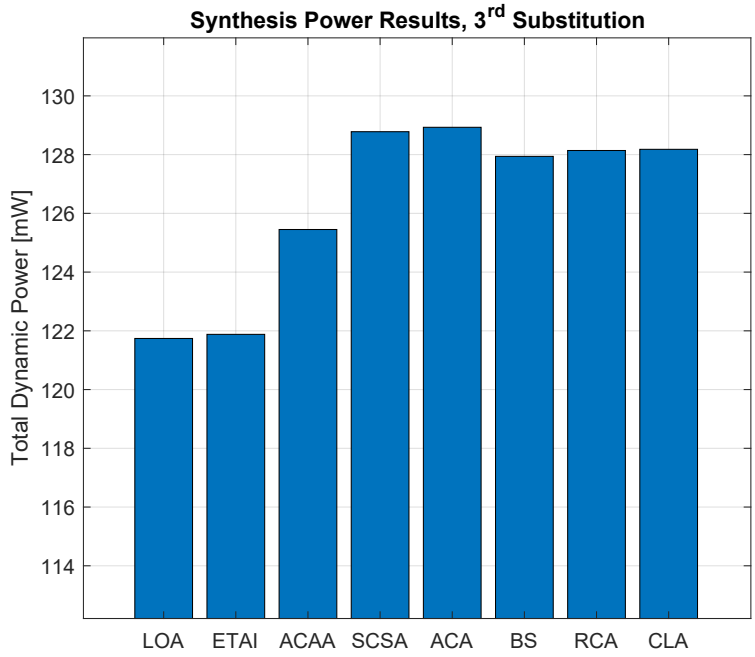
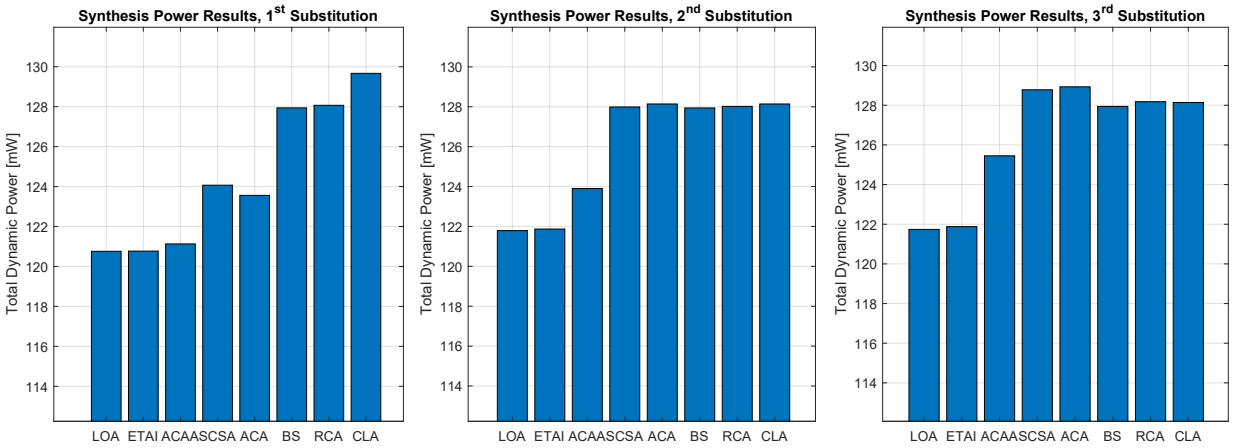


Figure 8.3: Summary of the total power consumption for all the structure with/without approximate adders, third substitution

8.4.4 Overview

The complete perspective of the three substitutions is illustrated in figure 8.4, which contains all the three total power graphs previously analyzed.

Figure 8.4: Comparison of the total power consumption in the three substitutions.



As can be seen from the graphs, LOA, ETAI and ACAA dissipate lesser amounts of energy. In direct relation with this positive trend, the error rate of the three implementations results to be far higher than the other 2 remaining approximate architectures, ACA and SCSA. In the first substitution ACA and SCSA show appreciable results, 4mW less than the BS; in the second and third substitution this condition is lost, their dissipation increase and the structure is no longer advantageous from a power

point of view. Lastly, considering the integration between BS and RCA, the CLA shows acceptable results just in the 2^{nd} and 3^{rd} substitutions, while in the first shows the limited performance due to the limited number of bits the architecture work with.

8.5 Analysis of Particular Cases

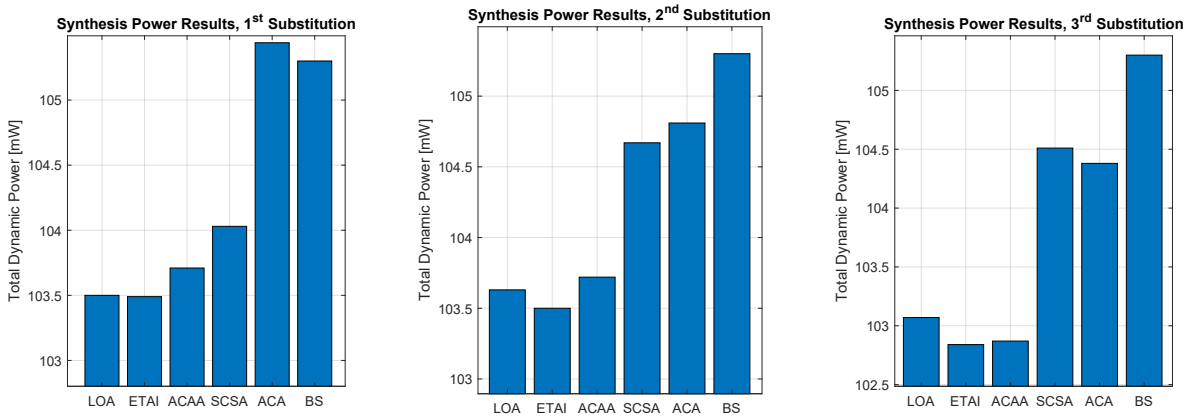
8.5.1 SAD architecture without PDE

The basic structure can be deprived of the partial distortion elimination technique. Without the PDE, the accelerator results to consume less at the cost of a decrease in speed: the comparators added at different level of the adder-tree are eliminated and the operations are fully executed, step by step, without the comparison of the actual and reference SADs. To evaluate the power shift from the precise structure to the approximate structures, a partial characterization has been developed for this case of accelerator. Table 8.31 illustrates the power consumption for the different combination with the 3 substitutions. Table 8.31 is graphically represented in picture 8.5.

Table 8.31: Power computation (in mW) for the three substitutions, employing the first version of the architecture without PDE.

Total Power [mW]	1^{st} Substitution	2^{nd} Substitution	3^{rd} Substitution
BS / RCA		105.30	
LOA	103.50	103.63	103.07
ETAI	103.49	103.50	102.84
ACAA	103.71	103.72	102.87
ACA	105.44	104.81	104.51
SCSA	104.03	104.67	104.38

Figure 8.5: Comparison of the total power consumption in the three substitutions, PDE excluded.



The first difference from table 8.7 is the reduced discrepancies between the approximate and precise cases. While in table 8.7 the power gap between RCA (or BS) and LOA (or ETAI) for the first substitution was around 8 mW, from table 8.31 the gap is decreased, reaching an average power value around 1.5 mW. This decrease can be related to the low power dependency of the accelerator

on the combinational part, which represent the heavier contribution when approximate adder are implemented. It has to be noted that the more promising architecture behavior comes from the ETAI substitution with respect to the LOA case, which was the most interesting coming from the previous analysis, clearly visible in figure 8.5. Anyway, the difference is very small and the error measures are still not taken into account. Chapter 9.1 will study in deep the efficiency of all the configurations to highlight the best effective combination.

For example, a significant power gap among the combinations of table 8.31 is the power discrepancy between ETAI and RCA in the first substitution, equal to 1.81mW, equivalently equal to the 2%. As previously carried out in the analysis with PDE, if the designed architecture without the memory unit is considered, the power saving clearly highlight the effective impact of the approximation on the encoder. The new percentage is based on values related to the synthesis reports (here not represented for sake of clarity) and defined as:

$$Power\ Saving_{\ Designed\ Part} = 1 - \frac{Power\ ETAI_{\ Designed\ Part}}{Power\ RCA_{\ Designed\ Part}}$$

The calculus results in a power saving $\leq$ 4%. This is a not satisfactory value but it's not the maximum achievable. The best combination for power saving is the couple ETAI (or ACAA) in the third substitution. In this case, the power gap results 2.46mW and the percentage, with respect to the total dynamic power, 2.5%. The effective saving with the designed part results in a little increase of the previous value, reaching the 5.5%. Both contributions looks very poor. This is the first trivial result of the thesis, because it highlights the limited influence of the approximate adders inside an architecture with no optimization such as the PDE technique. A power optimization around the 5% is not totally negligible but, at the same time, not remarkable. Without the usage of PDE mechanism, the accelerator reduces the consumption related to the addition/subtraction operations for the SAD calculation. This does not imply that the SAD reduces its weight in terms of resources spent, but the presence of the approximate adders doesn't reduce this cost. Indeed, a variation of 2-2.5 mW is perceptible but cannot be considered a satisfying improvement. In many works previously cited (ref.[19] and [17] for example), the power saving of the least efficient architectures with respect to the reference (accurate adders) is in the order of the 5%-8% on average. Such percentage means that, for example, the 4% of gap between ETAI and BS cases in the first substitution is in line with the literature results, even if it is a perceptible but not satisfactory result. The absence of comparators in the branches of the adder tree allows to save in terms of power, not system speed, and it is therefore more difficult to improve the system in terms of consumption.

Area and number of gates of this second groups of measurements revealed very similar results to the previously cited: the average gate number amounts to 86.3 KGates and the area remains approximately 1.20 mm^2. Number of gates decreased because the comparators and priority encoder are eliminated, without varying the area value particularly.

The CLA case was not reported in the table because it's the only case reporting a value of power consumption equal or higher than the basic structure. For sake of completeness, the medium value of power consumption in the CLA case is approximately 106 mW and the explanation of this power gap is the same previously described of the reduced parallelism architecture.

In conclusion, the positive outcome of this section is that the power classification among the approximate adders is approximately constant and in line with the outputs of the analysis coming from the study of the architecture with PDE technique. On the contrary, the analysis didn't give appreciable values of power saving; this implies that approximate adders couldn't be a strong optimization in the case of the accelerator without optimization techniques but can be a resource when the power consumption increases, following the addition of bit-optimization, distortion techniques, noise removal techniques, filter implementation or other types of application.

8.5.2 Clock Gating Technique

Clock gating is a technique concerning synchronous circuits whose task is to reduce the total dynamic power dissipation. Clock gating procedure is base on the clock pruning: pruning the clock turns off portions of the circuit to disable the flip-flops' activity, disabling the switch states which are source of consumption. Clock gating is a circuit level modification that can be imposed during the synthesis, without degrading the circuit performance.

The SAD architecture is parallelized, 16 Processing elements (PEs) can work in parallel, but all these PEs aren't always active at the same time; sometimes the reference and current matrices have a smaller block size, such that not all PEs will be required. Therefore, a possible improvement to decrease the total power could be the clock gating technique. Every time a register does not have to sample its input, its clock is disabled and, thence, the commutations are reduced. [2]

Clock gating has been adopted in the SAD accelerator synthesis to reduce power dissipation but the results of the simulations didn't give a significant result: it was not successful for block size higher than 16x8 because the 16 PEs are always on from 16x16 upward matrices. The optimization in terms of power for 16x8 and 16x4 is practically negligible; for 8x8 and 8x4 the power reduction reaches the 20%. The same results are shared by Paolo Selvo [2]. Moreover, the power saves results to be negligible for higher dimensions because the clock gating can optimize just registers. In fact, the most of the power is consumed by the SRAM memory (typically 75-80% of the total power) in the case of larger matrices.

8.5.3 16x16

The adder-tree inside the Datapath, that sums the results of the PEs, is divided into 4+1 stages to reduce the critical path, as cited in chapter 4.3. The "+1" is related to the accumulation phase (adder + register), accumulator placed at the end of the tree. This is used for that SAD that requires more than one clock cycle to load the samples from the memory: SAD from the 32x16 upwards requires the accumulator.

To verify the influence of the accumulator on the performance of the system when approximate adders are employed, an inverse power study is carried out for a SAD input matrix with block size 16x16. The 16x16 matrix is equivalent to a 64x4, because the matrices are organized in order to maximize the number of PEs that are used in parallel and transform the block size. Furthermore, the 16x16 matrix is transformed in a 64x4 one to be written or loaded inside/from the memory system in just 1 cycle.

Table 8.32 illustrates the power consumption for the accelerator implementing the approximate adders in the case of the second substitution. Only the second substitution has been taken into account because it represents the trade-off among the three possible substitution.

Table 8.32: 16x16 Block Size - Power evaluation

Structure	Total Power [mW]	Power Variation
BS	41.52	0 mW
RCA	41.51	$\sim$ 0 mW
CLA	41.76	+ 0.24 mW
LOA	41.46	- 0.6 mW
ETAI	41.37	- 0.15 mW
ACA	41.63	+ 0.11 mW
ACAA	41.47	- 0.5 mW
SCSA	41.50	- 0.2 mW

The general trend is the same as observed in the chapter of power evaluation for the second substitution: LOA, ETAI, ACAA, SCSA show a reduction while ACA and CLA a rise in the power consumption. In particular LOA and ETAI shows the best improvement. CLA is the worst choice probably for the same aforementioned reason: CLA is not optimized for 16 (or lower) bits parallelism. The power distance between the precise accelerator and the ETAI case is just 0.15 mW, meaning that the accumulator weigh down the system's performance. In fact, from the 64x64 to the 16x16 case, the power distance between the BS and ETAI implementations is decreased of 5.5 mW. This implies that approximate adders can be useful for heavy computation of high definition videos but, for lower resolution sequences, the improvement is not enough high.

8.5.4 16x4

The 16 PEs work in parallel and compute a 4x4 SAD in one clock cycle. They are all active if the input SAD matrix is a 16x16 or larger, as said in section 4.3. The transformation imposed by the Datapath modifies the input matrix to exploit the parallelism of the pipelined structure of the SAD accelerator. The worst case analyzed, 64x64 block size, employed, of course, all the PEs. To be more precise, in the case of a 64x64 SAD, 16 clock cycles are taken to read the two matrices related to reference and current frame and to compute the 4x4 SADs; after, an additive delay of 5 cycles is spent by the adder tree to calculate the final SAD results. Hence, the 64x64 SAD requires 21 clock cycles inside the Datapath. If a smaller SAD matrix is considered, a lower number of PEs will be employed, the adder-tree will reduce its length and a smaller number of clock cycles will be taken. Hereafter an analysis of a 16x4 block size matrix is carried out to compare the influence of the approximate adder substituted for a dimensionally smaller SAD matrix as input reference and current matrix. For a 16x4 matrix, the total clock cycles are 6 (4 PEs and 2 adder-tree's stages).

The synthesis for the several cases of SAD accelerators has been completed for 16x4 reference and current matrices. The power evaluation revealed not considerable change from one case to the other, maintaining the value of power around 18.59 mW, with variation from one case to the other of 5-10 μW. This negligible change highlights that approximate adder for lower block size results to be not necessary for performance improvement, as discovered in the previous section. As previously done, only the second substitution has been taken into account because it represents the trade-off among the three possible substitution.

8.5.5 32x32

For block size dimensions larger than 32x16 the accumulator start working. The previous 16x16 and 16x4 tests didn't need any accumulation steps and the results highlight a slight power saving coming from the implementation of approximate adder with respect to the basic architecture. This section focus on the 32x32 block size test to understand the impact of the accumulator in the power consumption, impact already clear comparing 16x16 case with the 64x64 complete analysis, but a further analysis could be useful.

Results coming from the 32x32 study are very similar to the 16x16 one, the only variation is that the power dissipation of every combination increases of 6-7 mW. The larger power gap between the best performance configuration and the basic structure gives a power saving of 1.2 mW, the smaller gap (negative excluded) is around 0.5 mW. These results look in line with the previously obtained values; it represent a sort of average quantity between the power saving obtained in the 64x64 test and the 16x16 one: it's doubled compared to the 16x16 and it's approximately $\frac{1}{4}$ of the 64x64 because other 4 accumulation cycles are required before completing the 64x64 SADs computation. This brief evaluation allows to understand the substantial weight of the accumulator on the SAD accelerator's performance.

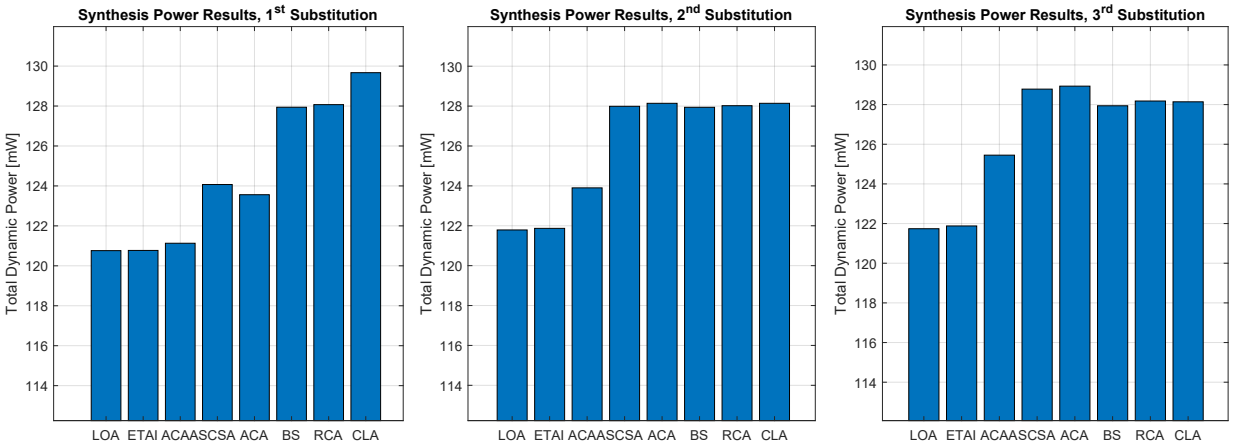
Chapter 9

Critical Analysis of the Architectures

The critical analysis of the architectures considers all the quantities resulting from the synthesis and compares the several informations shown during this work, at first in an introspective way and subsequently classifying the resulting performance with other works inside the literature. The parameters taken into account to achieve such purposes are: frequency, delays, area, gate number, power dissipation, block size and efficiency.

The most concise and explicative means concerning the power synthesis of the different architecture-substitution combinations is the comparison graphs of figure 9.1, already illustrated in figure 8.4.

Figure 9.1: Comparison of the total power consumption in the three substitutions. LOA: Lower-OR Part Adder, ETAI: Error-Tolerant Adder, ACAA: Accuracy Configurable Approximate Adder, SCSA: Speculative Carry Select Adder, ACA: Almost Correct Adder, BS: basic structure, RCA: Ripple Carry Adder, CLA: Carry Look-Ahead Adder.



It can be useful to remind some of the main concepts extrapolated from these pictures and from the accurate study carried out from the synthesis results:

- The introduction of an approximate adder in the structure of the encoder results in a large reduction of dissipated power and, in a complementary way, an appreciable power saving, at the cost of a quality loss (proportional to the error).

- Without considering the memory frequency constraint, the use of approximate adders gives the possibility to work at slightly higher frequencies than employing precise adders.
- The implementation of approximate adders causes an increase in the number of gates, with a small and, in some occasions, negligible decrease of the area.
- The comparison between accurate and inaccurate computations can be carried out using as reference both the automatically synthesized adders (Basic structure BS) or employing the author-written RCA; hand-written CLA didn't carry optimization.
- The approximate adders can increase the power saving mainly if the implementation exploits "strong" approximations in the motion estimation; "weak" approximations have proven not to be sufficiently efficient despite the substitutions applied. This was true of the power analysis without error computation. In this chapter a detailed survey will clarify every possibility.
- Among the three substitutions, the first resulted to be the most power saving, the second represents the trade-off between the three and the last and third is the most unpredictable. The first, being the most approximate, showed the larger mismatch between precise and imprecise results, suitable also for lightly approximated architectures such as SCSA and ACA implementations. The second and the third one showed a good tuning with the two "approximate full adders" (ETAI and LOA) and the ACAA.
- The system heavily depends on the memory unit consumption, representing the ∼75% of the total dynamic power in every combination. In fact, in the previous study the power saving analysis has been carried out both considering and, then, neglecting the memory.
- The approximation has achieved excellent results for large block sizes (64x64 and 32x32) but did not have the same effect on small block size like 16x16 and 16x4, discussed in section 8.5.

As it emerges from the ruminations extrapolated from the analysis of figure 9.1, it's clear that there is a strong dependency between power and error and, at the same time, this relation is not linear but depends on a series of variables: adders, approximation and substitutions. A graph highlighting the trend of the average error as a function of power reduction can be useful to evaluate the dependency and this is precisely what is shown in figure 9.2.

The graph illustrates an average result of the error's trend with respect to the power reduction in the three substitutions, giving the idea of the error impact on the power saving. The picture can be divided into two regions, separated from each other at 3.5mW in the power reduction axis. For null to 3.5mW the error is reduced and this is typical of medium and highly approximated adders replaced in the first substitution or lightly approximated adders replaced in the second or third ones. The most prominent region comes after the 3.5mW where the best results are obtained in terms of power saving with an inevitable increase in error. At a first glance, the region on the left contains the less erroneous and less performant configurations, while the right region the most promising ones but the power saving alone can not be a representative variable of the behavior of the whole system. For instance, the higher power saving (the extreme edge on the right) is obtained for an E_{MRED} above the 20% (equivalently MRED over the 0.2) which could result unsuitable for the most of the application fields. In a similar way, there are some interesting results before the 3.5mW, like the couple (2.49mW, 0.09) which represent a good trade-off between power saving and MRED.

Separated values of power and error don't give a clear idea of the efficiency of a system. A convenient way to compare the implemented versions of the accelerator is to analyze the relation between power savings and the error measures; this can be translated as an efficiency evaluation to decide which approximate adder is suitable for the specific application. The power saving, calculated

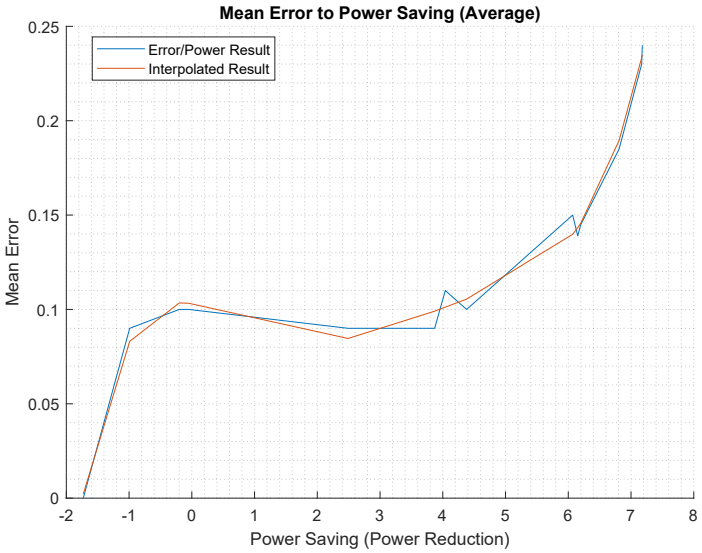


Figure 9.2: MRED vs. Power graph: the graph is obtained taking the MRED values increasingly in the three substitutions and represented with the respective power saving values (blue line). The red line is the interpolation of the error's and power's points to obtain a rounded average value.

as the difference between the power consumption of the specific application minus the reference one (BS or RCA), is divided by the MRED, considered as the reference error, in order to obtain the efficiency. The results are contained in table 9.1. This efficiency can be interpreted as how much coding efficiency is lost to allow the achieved power consumption reduction [19]. Of course, the higher is the calculated ratio, the best performance will give the structure. The numerical scale is 100, so the numbers are distributed inside the interval ±[1,100]; it acts like a percentage scale. If the power value had been divided by the E_{MRED} (%), the result would have been equivalently in the interval ±[0,1], as execute in ref.[19].

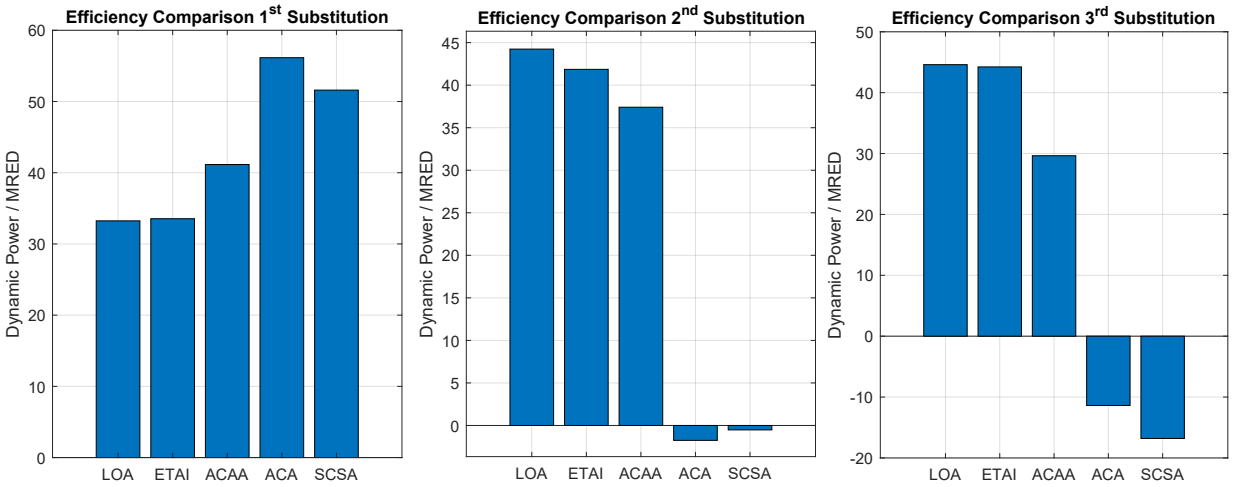
Table 9.1: System's efficiency, defined as the ratio between power saving and MRED, for all the possible combinations. RCA and BS are considered as reference, because of null power saving and MRED; CLA is not considered because of negative power saving.

Structures	$\frac{PowerSaving1^{st}Sub.}{MRED}$	$\frac{PowerSaving2^{nd}Sub.}{MRED}$	$\frac{PowerSaving3^{rd}Sub.}{MRED}$
LOA	33.24	44.24	44.60
ETAI	33.54	41.86	44.23
ACAA	41.15	37.41	29.64
ACA	56.15	-1.75	-11.38
SCSA	51.60	-0.52	-16.80

Table 9.1 illustrates the efficiency calculated with the error measures of the complete study in C++ code, for all the cases. One could think that the errors calculated using Modelsim could be employed but, as already discussed, the VHDL code gave far more accurate results than the C++ version. To better understand and visualize the table 9.1, figure 9.3 contains the three table's column

represented in a bar chart.

Figure 9.3: Comparison of the efficiency in the three substitutions.



For the first substitution, the high error of the most promising approximate adders, like LOA and ETAI, is a discriminant and the efficiency of these two cases is lower with respect to the ACA and SCSA implementations that, even if the power saving is smaller, have a lower MRED which increase the general efficiency. This behavior is valid just for the first substitution, where error rate of 20-25% were present for ETAI and LOA. Second and third substitutions show the opposite trend: LOA and ETAI give higher efficiency thanks to the acceptable MRED and quite high power saving with respect to the other choices. ACAA represent the trade-off of the architecture, maintaining a medium efficiency in all the 3 substitutions. The negative values of efficiency for ACA and SCSA in 2^{nd} and 3^{rd} substitutions are a consequence of the negative power saving (positive power loss with respect to the basic structure); ACA and SCSA are unsuitable for these two substitutions but, surprisingly, very efficient in the first one. From table 8.7 as well as figure 9.1, it can be observed that the power saving for these two structures is limited but is compensated by the low level of error.

In section 8.4, it was hypothesized that the best configurations, for all the three substitutions, was achieved replacing the ETAI and LOA to the accurate adders in dependence on the substitution, otherwise, to obtained a more equilibrated combination, the ACAA. These architectures showed the smaller power rate, and consequently the higher power saving with respect to the the reference accurate adder RCA. These hypotheses has been partially discredited by the efficiency computed in the first substitution. The E_{MRED} value for a low resolution video for LOA and ETAI assumed values around the 21%, which is equivalent to MRED = 0.21, with a power saving of 7 mW. Meanwhile, ACA and SCSA assumed an mean error around the 7%, equivalently MRED = 0.07, with a smaller power saving, about 4 mW. The power saving-error ratio emphasizes the lower MRED value. On the contrary, for example, the ratio in the second substitution between power saving and MRED emphasizes the LOA or ETAI substitution, showing a power saving of $\sim$ 6 mW and a limited MRED of 0.1.

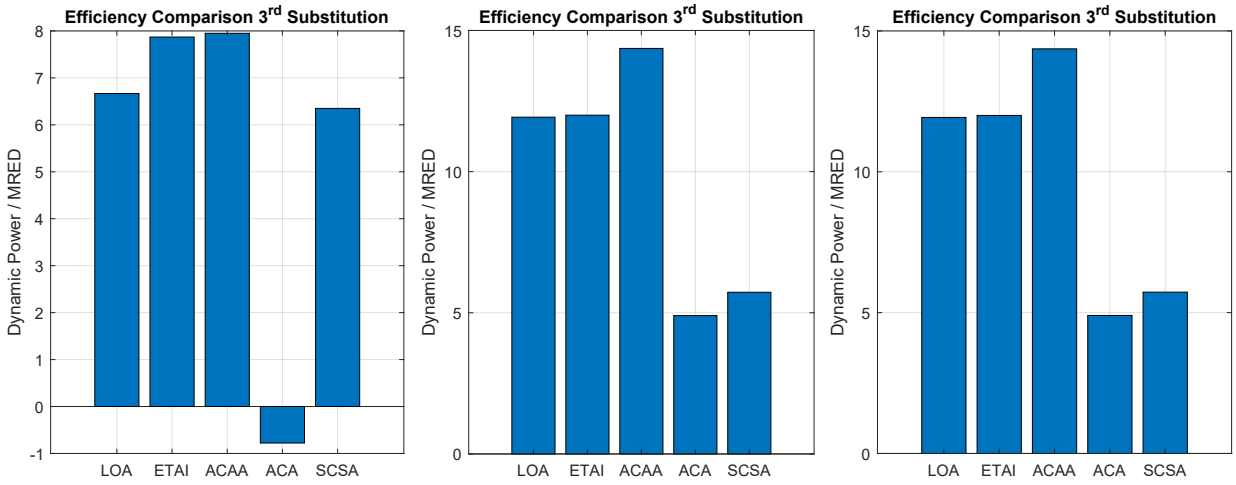
In this phase, the results are satisfying. There is not an overall solution, the selection of the adders directly depends on the application. If a general outcome is needed, a faster result can be achieved implementing highly approximated structures (like LOA or ETAI) for low resolution compressing processes, while a slower but more accurate result can be achieved with lightly approximated accelerator (implementing ACA or SCSA). As observed from the results in this and in the previous section, LOA and ETAI are suitable for fast and error-tolerant applications, or for higher resolution application if the

substitution is not severe (second or third substitutions); ACAA is a sort of average implementation, with a behavior fitting every substitution and every implementation, showing medium value of power saving and efficiency; SCSA and ACA are the most accurate possibilities, good implementations if used in large-scale substitution but absolutely not efficient if implemented in small and medium scales.

9.1 Efficiency Analysis without PDE

At the end of this work, a small comment is needed for the distinction between PDE excluded system and PDE included system. The efficiency is computed also for the case of the SAD accelerator without PDE. Without entering into details, because the process of calculation is the same previously described, the efficiency in the case of the encoder not including the partial distortion elimination technique is illustrated in figure 9.4.

Figure 9.4: Comparison of the efficiency in the three substitutions.



Results differ between PDE included system and PDE excluded system. Without the comparators' cost in power, the approximate adders don't have strong influence on performance. The best approach seems to be the substitution of the ACAA in all the three cases: this is probably related to the acceptable mismatch between speed and error that the ACAA reaches. ACA and SCSA have a power saving value not decisive in the efficiency computation, while ETAI and LOA, having a lower power saving with respect to the PDE included case, show a worse mismatch power saving-error. In section 8.5.1, it was hypothesized that the best approximate adder for the architecture without PDE was the ETAI, showing smaller power rate, but this hypothesis is now discredited from the lower power-error ratio reached (caused by a quite large error rate). In fact, LOA and ETAI play on equal terms and ACAA looks the best solution, unlike what was expected.

The results coming from this section, even if proposes a sufficient configuration, are not satisfactory, dealing with efficiency results 40 times lower than the previous calculated efficiency, PDE included. As highlighted in chapter 8.5.1, the outcome is trivial and no implementation is convenient in any configuration: without partial distortion elimination, the implementation of the approximate adders is unnecessary.

Even if the results differ from case to case, this is not problematic for the purpose of the thesis because the main objective of the work was to optimize the best result of the previous thesis work

[2]: the most relevant results come from the implementation of an approximate adder in the SAD accelerator with PDE technique included, both because showing clear results for power gap and appreciable efficiency comparing basic structure and approximate structures.

9.2 Comparison with Other Architectures

Nowadays the study of the motion estimation is a very popular topic and several application specialized algorithms are proposed. The comparison between architectures is not trivial because the synthesis carried out author by author differs in standards, softwares, implementations, compatibility, constraints, technology and application's field. It's not an easy task to find comparable design with similar requirements and specifications, simulated with the same technology and the same program. Even if some inconsistency between architecture and architecture is certain, a brief summary of the actual studies inside the literature from 2015 upward is realized, highlighting some key feature of the several configurations. In table 9.2 all the interesting data regarding gates number, delays, frequency and some other informations are collected; white sections inside the table means that no informations was given about the concerned parameter.

Table 9.2: Performance comparison of the proposed architecture with other works from the literature.

	[3]	[6]	[7]	[8]	[19]	[2]	Proposed
Technology	65 nm	Virtex-5	Virtex-5	65 nm	45 nm	65 nm	65 nm
Max Block Size	64x64	64x64	64x64	32x32	64x64	64x64	64x64
Gate Count	434K	26.8K	63.2K	617K	30.9K	90K	$\sim$ 90K
Frequency [MHz]	720	190.785	348	350	497.66	160 *	160 *
Delay for each 64x64 Block	22.24 ns	167.73 ns	45.96 ns			156.25 ns	156.25 ns
Search Window	(±27,±27)			(±32,±32)		(±64,±64)	(±64,±64)

It's very difficult to insert the proposed work inside the panorama of the SAD accelerators, mainly because the most of the literature reports design the adder tree unit without going in detail with the control part and signal processing. [3] and [8] also consider the memory interface, as carried out in this work, while [7], [6] and [19] do not mention or consider the implementation of a memory. This leads to the absence of constraints, as, for example, it was for the maximum bandwidth in this thesis work (set at 160 MHz). About this, the ' * ' symbol highlights that the maximum frequency achievable by that architecture is limited by the SRAM IP working frequency. The maximum frequency for the SRAM is 200 MHz. Moreover, the absence of a memory system is evident in the gate number of the citations: [3] and [8] have a gate number that is one order of magnitude higher because of the memory implementation. The proposed architecture and the architecture of ref.[2] show a gate number of 90K, because, during the gate count phase, the memories' areas has been eliminated from the useful area to calculate the gate number. If the computation is rerun, the total gate number becomes 833K.

All the proposed architectures work with a Full Search algorithm for the search window, and are compatible with the HEVC standard. [3], [6] and [7] are compatible with H.264 too. The delay for each 64x64 Block can be defined as the product of the maximum period of delay times the number of clock cycles for a 64x64 PB:

$$Delay = \frac{1}{f_{max}} \cdot \#CLK_{cycles} \tag{9.1}$$

In the case of the proposed work it results equal to: $Delay = \frac{1}{160MHz} \cdot 25 = 156.25ns$. If the limit frequency of the SRAM is considered (200 MHz) the delay decrease to $Delay = \frac{1}{200MHz} \cdot 25 = 125ns$ for computing a 64x64 SAD. This delay allow the system to perform 8 millions of 64x64 SADs in

one second. This is the case without the PDE, because, using the PDE, the accelerator can reach 11 millions of 64x64 SADs in one second.

In table 9.2, the power saving and area of the architectures are not considered as a comparison parameter, because the entire architecture of the encoder takes into account interface unit's, memory's and accelerator's power consumption. In literature, it's difficult to find a complete work and the synthesis is carried out, like in [3], [6], [8] and [19], only for the accelerator. It is quite a matter of being able to define which architecture is more or less efficient: in the attempt to prove the efficiency of the proposed architecture, it can be exalted the intelligent architecture and units interface, the flexibility, the dynamically reconfigurability and the general performance. The aim of this work of thesis was to develop a low power SAD hardware accelerator, employing approximate adders to improve performance and power reduction. The verification or comparison with other architectures does not nor confirm neither deny the effective efficiency of the system, because it's quite tricky to compare the work with the articles of the literature. Anyway, the result looks satisfactory and the behavior of the architecture seems adequate to the standard HEVC, low power and with good throughput.

A final critical mention can be made for the negative aspects of the overall project: the main limits of the architecture/work of thesis are related to the chosen on-chip memory, the selected synthesis technology and the quite large search window. The memory unit restricts the possible frequency range and timing delays and increases the power consumption. Any memory would have these limitations, so it's not an effective limit but a needed part of the system. Moreover, it doesn't sustain double or quad bit-rate; a more accurate selection for a new memory, suitable and specialized for low power operations, could be useful. The 65nm CMOS technology was exploit during the simulations to maintain a connection with the previous work of P. Selvo [2] and verify the correctness of the results. The majority of the most recent articles in the literature adopts the 45nm CMOS libraries. The use of the new library also enlarge the possibility of comparison with other architectures. In the end, the ($\pm$64,$\pm$64) window search can be useful to increase the output quality of the motion estimation, but several applications, also among the literature, employ smaller sizes, enough for the best match search and improving the performance.

Chapter 10

Conclusion and Future Works

This work of thesis modified a pre-existent HEVC encoder (developed by P. Selvo in ref.[2]), deeply analyzing the power behavior of the system and trying to optimize it. The reduction of the power dissipation in this SAD accelerator has been faced through the substitution of approximate adders inside the adder-tree of the architecture. In other words, the SAD accelerator structures was revised in favor of dynamically reconfigurable approximated architecture, ensuring better performance and lowering power consumption, at the cost of a quality and resolution loss of the compressed video.

The main parts of this work of thesis can be summaries in the following list:

1. Theoretical discussion about the most recent video compression technologies.
2. Study of the motion estimation and compensation with Matlab programs and introduction to the concept of filter.
3. Explaination of the pre-existent architecture.
4. Approximate computing concept, definitions of the approximate adders and selection of the substitutions.
5. Verification step, which consists in the error evaluation, for both SAD computation and motion vectors mismatch between accurate and inaccurate results.
6. Modelsim verification of the approximation implemented.
7. Synthesis of the system, power evaluation of the different combinations (approximate adders and substitutions alternated), both in the worst cases and in particular cases.
8. A final critical analysis of the obtained results, based on efficiency.

The introduction of approximate adders produces an error and a power reduction. This thesis carried out a study regarding the best configuration showing the best trade-off between these two quantities. The choice depends on the adder-substitution couple and there is not a general solution but the selection of the best suitable result depends on the application. Faster architectures are realized with highly approximated structures for low resolution compressing processes and are suitable for error-tolerant applications, or for higher resolution application if the substitution has a limited impact (1^{st} substitution had strong impact, 2^{nd} and 3^{rd} are less severe). Slower but more accurate architectures can be actualized with lightly approximated accelerator, large-scale substitution (high impact like the 1^{st} sub.) and application with smaller bit-rate.

Possible Future Implementations:

- Several modifications and improvements can be developed on this work of thesis. The first modifications and tests can concern the negative aspects highlighted at the end of chapter 9: memory unit, synthesis with 45nm technology and reduced window search. These are not effective improvements, are just adjustment that can be realized to further optimize the pre-existent architecture.

- At code/algorithmic level, Matlab offers a huge range of possibilities: for example, a Matlab analysis of the approximate motion estimation can be interesting; it's a non trivial task, because of the limited capability of Matlab to works on bit sequences, but, exactly as masks can be used in C++, in a similar way can be applied in Matlab. Another test executable on Matlab is the implementation of the several search algorithm for faster best match search. Some example are the TZ search, diamond search, three or four step search, and so on, in order to observe a speed-up results given by PDE.

- Surely, having the availability to work on an FPGA, the hardware accelerator can be converted into a project suitable for FPGA implementation: an idea is a microprocessor which runs the HEVC software helped by several hardware accelerators on an FPGA that performs the most critical tasks in terms of computational time and power consumption [2]. The most interesting part should be the realization of the accelerator-μprocessor interface.

- Further verifications can be executed to investigate the effective reduction of resolution and quality of the compressed video sequence. If the FPGA implementation is possible, one could verify the impact of the approximate computing of the encoding process and verify, effectively, which approximations can be accepted and which are too severe.

- At the hardware level, the PEs can be substituted with the Hadamard blocks in order to implement the sum of absolute Hadamard-transformed differences (SATD), available in H.264/AVC and achievable in HEVC. Experimentally the SATD should ensure higher rate-distortion performance compared with the ordinary SAD but increasing the computational load due to the Hadamard transformation, unless the fast Hadamard transform (FHT) is employed, which reduce by 50% the computational requirement. [29]

- As discussed in chapter 3, the use of filters gives higher detailed results and finer video compression, with an increase of the system complexity and cost. This implementation is optimized for low bit-rate coding and for the compression of slow motion videos, for example. This is a suitable application to the HEVC encoder: the addition of filter can be the right trade-off between highly approximated motion estimation, very fast and low power, maintaining a certain resolution and without exaggerated quality loss, and a detailed video compression.

- This last point is purely theoretical and dedicated to possile future implementations: the approximation to reduce the power dissipation can be extended also to the multiplier. Since multiplication is one of the most frequent operation and highly energy-consuming, approximated multiplier can be employed. The task of such operators is the same of the adders: reduce the power consumption over the accuracy.

Bibliography

[1] Khalid Sayood, "Introduction to Data Compression", THIRD EDITION, University of Nebraska

[2] Paolo Selvo, "An Optimized PDE-based SAD Accelerator for HEVC", Masters Degree Thesis, Politecnico di Torino, 2017

[3] Ahmed Medhat, Ahmed Shalaby, Mohammed S. Sayed, "High-throughput hardware implementation for motion estimation in HEVC encoder", Egypt

[4] Ioannis Makris, Harilaos Koumaras, Jurgen Mone, Vaios Koumaras, "Digital Video Coding Principles from H.261 to H.265/HEVC", Greece, 2015

[5] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard", IEEE, Transactions on Circuits and Systems for Video Technology, 2012.

[6] Vu Nam Dinh, Hoang Anh Phuong, Duong Viet Duc, Phung Thi Kieu Ha, Pham Van Tien, Nguyen Vu Thang, "High speed SAD architecture for variable block size motion estimation in HEVC encoder", 2016, Vietnam

[7] Ahmed Medhat, Ahmed Shalaby, Mohammed S. Sayed, Maha Elsabrouty, Farhad Mehdipour, "A highly parallel SAD architecture for motion estimation in HEVC encoder", 2015, Japan

[8] Seongmo Park, Byoung Gun Choi, In Gi Lim, Hyung-il Park, Sung Weon Kang, "An efficient motion estimation hardware architecture using Modified Reference Data Access(MRDAS) skip algorithm for high Efficiency Video Coding(HEVC) encoder", 2016, Berlin, Germany

[9] Il-Koo Kim, Junghye Min, Tammy Lee, Woo-Jin Han, and JeongHoon Park, "Block Partitioning Structure in the HEVC Standard", IEEE, Dec. 2012

[10] Joint Collaborative Team on Video Coding (JCTVC), HM 16.6 Reference Software, Website: "$https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-16.6/$".

[11] R- Lucas, Barros da Silva, Maciel de Faria, M. Rodrigues, L. Pagliari, "Efficient Predictive Algorithms for Image Compression", 2017, cap-2

[12] HEVC test video sequences available in the following website: "http://trace.kom.aau.dk/yuv/index.html"

[13] VcDemo: Website: "http://homepage.tudelft.nl/c7c8y/VcDemo.html", 2017 TU-Delft

[14] Jens-Rainer Ohma, Mihaela van der Schaarb, John W. Woods, "Interframe wavelet codingmotion picture representation for universal scalability", University of California Davis, USA

[15] R. W. G. Hunt, "The Reproduction of Colour", 6th ed., Chichester UK, Wiley, 2004

[16] Arnab Raha, Hrishikesh Jayakumar, Vijay Raghunathan, "Input-Based Dynamic Reconfiguration of Approximate Arithmetic Untis for video encoding", 2015

[17] Walaa El-Harouni, Semeen Rehman, Bharath Srinivas Prabakaran, Akash Kumar, Rehan Hafiz, Muhammad Shafique, "Embracing Approximate Computing for Energy-Efficient Motion Estimation in High Efficiency Video Coding", Vienna University of Technology (TU Wien), Austria, 2016

[18] Honglan Jiang, Jie Han, Fabrizio Lombardi, "A Comparative Review and Evaluation of Approximate Adders", 2015, USA

[19] Roger Porto, Luciano Agostini, Bruno Zatt, Marcelo Porto, Nuno Roma, Leonel Sousa, "Energy-Efficient Motion Estimation with Approximate Arithmetic", 01 December 2017, Luton, UK

[20] Ajay K. Verma, Philip Brisk, Paolo Ienne, "Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design"

[21] Kai Du, Peter Varman, Kartik Mohanram, " High Performance Reliable Variable Latency Carry Select Addition", Department of Electrical and Computer Engineering, Rice University, Houston, and Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh

[22] Yan Sun, Dongyu Zheng, Minxuan Zhang, and Shaoqing Li, "High Performance Low-Power Sparse-Tree Binary Adders", National University of Defense Technology, Changsha, China

[23] Andrew B. Kahng and Seokhyeong Kang, "Accuracy-Configurable Adder for Approximate Arithmetic Designs", ECE and CSE Departments, University of California at San Diego, 2012

[24] Ning Zhu, Wang Ling Goh, and Kiat Seng Yeo, "An Enhanced Low-Power High-Speed Adder For Error-Tolerant Application", School of Electrical and Electronic Engineering, Nanyang Technological University

[25] H.R. Mahdiani, A. Ahmadi, S.M. Fakhraie and C. Lucas, "Bio-Inspired imprecise computational blocks for efficient VLSI Implementation of soft-computing Applications", 2010, IEEE

[26] Cong Liu, Jie Han, and Fabrizio Lombardi, "An Analytical Framework for Evaluating the Error Characteristics of Approximate Adders", 2015, IEEE

[27] Avishek Sinha Roy and Anindya Sundar Dhar, "A Novel Approach for Fast and Accurate Mean Error Distance Computation in Approximate Adders", Department of Electronics and Electrical Communication Engineering, India, 2018

[28] YUV Test video sequences, website: "https://media.xiph.org/video/derf/"

[29] Mohammed Golam Sarwer, Q. M. Jonathan Wu, Xiao-Ping Zhang, "Enhanced SATD-based cost function for mode selection of H.264/AVC intra coding", London, SiViP, 2011