Scalable Architecture for Unified Transform Coding in Embedded H.264/AVC Video Coding Systems

João Vitor Gomes dos Santos

Thesis to obtain the Master of Science Degree in Electrical and Computer Engineering

Examination Committee
Chairperson: Prof. Nuno Cavaco Gomes Horta
Supervisor: Prof. Nuno Filipe Valentim Roma
Co-Supervisor: Prof. Leonel Augusto Pires Seabra de Sousa
Members of the Committee Prof. Horácio Cláudio de Campos Neto

April 2013
Acknowledgments

Moving to Lisbon and study at IST was definitely a very important phase of my life and for that I am very grateful to my parents, which provided me all that I needed to complete with success this challenge.

Life at IST was difficult but at the same time it provided to me a lot of opportunities and experiences to grow my technical and social skills. It is impossible to name everyone that marked my student life in the last years, but I have to highlight everyone that I have worked with in Junitec team for all of the opportunities that together we achieved.

I am also thankful to all my family and friends for all the support and patience. A special thanks to André Faria and Inês Ferreira for all the life support in the good and bad moments.

To my professors and supervisors, Nuno Roma, Tiago Dias and Leonel Sousa thank you for your trust, support and patience during the last months. A special thanks to Tiago for his dedication and help whenever it was needed.

To all that did not been named but contributed to my success and achievements directly or indirectly, thank you all!
Abstract

A DCT IP Core able to autonomously process all the encoding modes defined in the H.264/AVC standard for the 4:2:0 chrominance subsampling format is proposed in this document. The IP Core was developed as a scalable architecture based on a systolic array processor, which not only allows it to be easily configured to compute distinct transforms, but also permits its rescaling to support transforms of multiple sizes. Moreover, its scalability property allows it to be easily adapted to specific and resource limited applications.

The systolic processor architecture was designed based on an original architecture, which only supported transform kernels with dimensions up to 4 x 4 coefficients. This original architecture was now re-designed to support 8 x 8 transforms. All the units of the original architecture were properly adapted to compute the new transform coefficients and dimensions.

Besides the systolic array, the proposed IP Core is also composed by additional control logic to help in the computing process. The interface with the underlying general purpose processor is provided by a 16-bit control register and two data ports, for the input and output data values. These ports should be mapped in the GPP memory addressing space.

The experimental results considering the implementation of the conceived processing structure in a Xilinx Virtex-4 FPGA device demonstrated that it can compute all the H.264/AVC transforms for video sequences up to the HD1080p resolution in real-time (1920 x 1080 @ 30 fps).

Keywords

Discrete cosine transform, H.264/AVC video standard, Scalable systolic processor, Reconfigurable Devices
Resumo

Este documento visa apresentar um IP Core capaz de processar de forma autônoma todos os modos de codificação definidos na norma H.264/AVC para sub-amostragens de crominâncias no formato 4:2:0. O IP Core proposto foi desenvolvido como uma arquitetura escalável baseado num processador sistólico capaz não só de ser facilmente configurado para processar diferentes transformadas, mas também de ser facilmente escalado para suportar transformadas de várias dimensões. Para além disso, a escalabilidade permite também que a arquitetura seja facilmente adaptada a aplicações específicas e com recursos limitados.

A arquitetura do processador sistólico foi desenhada com base numa arquitetura original que suportava transformadas com dimensões de até 4 x 4 coeficientes. A arquitetura original foi re-desenhada para suportar transformas com as dimensões de 8 x 8. Todas as unidades da arquitetura original foram adaptadas para processar os coeficientes e as dimensões das novas transformadas suportadas.

Para além da matriz escalável, o IP Core proposto é também constituído por lógica de controlo adicional que ajudam todo o processo de cálculo. A interface com o processador de uso genérico adjacente é feita através de uma porta de 16 bits e duas portas de dados para os dados de entrada e os dados de saída. Estas portas devem ser mapeados no espaço de endereçamento da memória do processador de uso genérico.

Os resultados experimentais demonstraram que esta é capaz de calcular todas as transformadas da norma H.264/AVC para sequências de vídeo até a resolução HD1080p em tempo real (1920 x 1080 @ 30 fps).

Palavras Chave

Transformada discreta de cosseno, Norma H.264/AVC, Processador sistólico escalável, Dispositivos Reconfiguráveis
Contents

1 Introduction ................................................. 1
  1.1 Motivation ........................................... 2
  1.2 Objectives ........................................... 2
  1.3 H.264/AVC video encoding .......................... 3
  1.4 Integer Discrete Cosine Transform ................. 4
    1.4.1 8 x 8 integer DCT ............................. 4
    1.4.2 DCT Computation Algebra ...................... 5
  1.5 Main contributions ................................... 7
  1.6 Dissertation outline .................................. 8

2 State of the art .............................................. 9
  2.1 Introduction .......................................... 10
  2.2 General transform computational units ............. 10
    2.2.1 Single transform architectures ................ 10
    2.2.2 Multi transform architectures ................. 11
  2.3 Integer transform computational units ............. 13
    2.3.1 Non-Unified transform architectures .......... 13
    2.3.2 Unified transform architectures ............... 14
  2.4 Base Architecture ................................... 15
    2.4.1 Array of PEs .................................. 15
    2.4.2 Input Buffer ................................... 17
    2.4.3 Transposition Switch ........................... 17
  2.5 Summary .............................................. 17

3 The proposed architecture ................................ 21
  3.1 PE Array ............................................. 22
    3.1.1 Architecture of the new Processor Element .... 22
    3.1.2 The 8 x 8 Array ................................ 30
    3.1.3 Scalability of the array ....................... 31
  3.2 Input Buffer ......................................... 33
### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.1 Input buffer element</td>
<td>33</td>
</tr>
<tr>
<td>3.2.2 Central control unit</td>
<td>33</td>
</tr>
<tr>
<td>3.3 Transposition Switch</td>
<td>34</td>
</tr>
<tr>
<td>3.3.1 Delay buffer</td>
<td>35</td>
</tr>
<tr>
<td>3.3.2 Control of the transposition switch</td>
<td>35</td>
</tr>
<tr>
<td>3.4 Main Control Unit</td>
<td>37</td>
</tr>
<tr>
<td>3.5 Operation of the whole architecture</td>
<td>37</td>
</tr>
<tr>
<td>3.6 Summary</td>
<td>40</td>
</tr>
<tr>
<td>4 DCT IP Core</td>
<td>43</td>
</tr>
<tr>
<td>4.1 IP Core Structure</td>
<td>44</td>
</tr>
<tr>
<td>4.2 Memory Devices</td>
<td>45</td>
</tr>
<tr>
<td>4.2.1 Default and 16 x 16 Intra Modes</td>
<td>45</td>
</tr>
<tr>
<td>4.2.2 Intra 8 x 8 Mode</td>
<td>46</td>
</tr>
<tr>
<td>4.2.3 Chroma Mode</td>
<td>46</td>
</tr>
<tr>
<td>4.3 Control Unit</td>
<td>46</td>
</tr>
<tr>
<td>4.3.1 Operation modes</td>
<td>47</td>
</tr>
<tr>
<td>4.3.2 State Machine</td>
<td>49</td>
</tr>
<tr>
<td>4.3.3 Address Generation Units</td>
<td>51</td>
</tr>
<tr>
<td>4.3.4 Control Signals</td>
<td>54</td>
</tr>
<tr>
<td>4.4 DC Buffer</td>
<td>56</td>
</tr>
<tr>
<td>4.5 Input and Output interface</td>
<td>57</td>
</tr>
<tr>
<td>4.6 Operation modes</td>
<td>58</td>
</tr>
<tr>
<td>4.7 Example configuration</td>
<td>61</td>
</tr>
<tr>
<td>4.8 Summary</td>
<td>62</td>
</tr>
<tr>
<td>5 Experimental Results</td>
<td>63</td>
</tr>
<tr>
<td>5.1 Introduction</td>
<td>64</td>
</tr>
<tr>
<td>5.2 Evaluation Platform and Functional Validation</td>
<td>64</td>
</tr>
<tr>
<td>5.3 Implementation Results of the Processor Unit</td>
<td>64</td>
</tr>
<tr>
<td>5.3.1 Original architecture of the PE versus the new one</td>
<td>65</td>
</tr>
<tr>
<td>5.3.2 Scalability of the proposed architecture</td>
<td>65</td>
</tr>
<tr>
<td>5.3.3 Discussion</td>
<td>67</td>
</tr>
<tr>
<td>5.4 Implementation Results of the IP Core</td>
<td>67</td>
</tr>
<tr>
<td>5.5 Performance Analyzes</td>
<td>68</td>
</tr>
<tr>
<td>5.5.1 IP Core with an 8 x 8 PEs array</td>
<td>68</td>
</tr>
<tr>
<td>5.5.2 IP Core with an 4 x 8 PEs Array</td>
<td>69</td>
</tr>
<tr>
<td>5.5.3 IP Core with 2 x 8 PE Array</td>
<td>69</td>
</tr>
</tbody>
</table>
5.5.4 Discussion ........................................... 70
5.6 Summary ............................................ 70

6 Conclusions ........................................... 73
6.1 Future work ......................................... 75

A Appendix A ............................................ 81
A.1 IP Core Datasheet ....................................... 82
    A.1.1 Features ........................................ 82
    A.1.2 Description ...................................... 82
    A.1.3 Interface signals ................................... 82
    A.1.4 Modes of operation .................................... 84
    A.1.5 Memory Maps ....................................... 84
    A.1.6 Block Diagram ...................................... 86
    A.1.7 Step by step instructions................................. 86
    A.1.8 Time needed to computations and data latency ............ 86

B Appendix B ............................................ 93
B.1 Two different architectures of the PE Control Unit ............. 94
List of Figures

1.1 The H.264/AVC encoder diagram. ................................. 3
1.2 The H.264/AVC decoder diagram. ................................. 4
1.3 The architecture for 1-D 8 x 8 forward integer transform using the direct method extracted from [1]. ................................. 7
2.1 1-D DCT architecture extracted from [2]. ............................. 11
2.2 2-D DCT architecture extracted from [3]. ............................. 12
2.3 Example of the architecture extracted from [4]. ............................. 13
2.4 Fast papilionaceous algorithm flowchart (extracted from [5]). ............................. 14
2.5 The proposed FIT (a) and IIT (b) modules (extracted from [6]). ............................. 15
2.6 Base architecture of the processor element. ............................. 16
2.7 Scalable PE Array. ................................................. 16
2.8 The Input Buffer. ................................................. 17
2.9 The Base Architecture. ................................................. 18
3.1 PE Datapath. .................................................................. 24
3.2 Scalable Array of PEs. ................................................. 31
3.3 Array of PEs with 4 x 8 configuration. ................................. 32
3.4 Array of PEs with 2 x 8 configuration. ................................. 32
3.5 Modified Input Buffer. ................................................. 34
3.6 Modified Transposition Switch structure. ................................. 36
3.7 State Machine. ................................................. 38
3.8 Proposed architecture. ................................................. 39
3.9 Dataflow of the computation of a 2D 8 x 8 transform. ................................. 41
4.1 Block Diagram. ................................................. 44
4.2 Supported block organizations. ................................................. 47
4.3 Supported macroblock structures. ................................................. 47
4.4 Memory Maps with the relative addresses. ................................................. 48
4.5 Detailed Input Memory Maps showing where each sample is placed. ................................. 48
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6</td>
<td>Detailed Output Memory Maps showing where each sample is placed</td>
<td>49</td>
</tr>
<tr>
<td>4.7</td>
<td>Main state machine of the developed IP Core</td>
<td>50</td>
</tr>
<tr>
<td>4.8</td>
<td>Address Generator for the input memory</td>
<td>52</td>
</tr>
<tr>
<td>4.9</td>
<td>Address Generator for the output memory</td>
<td>53</td>
</tr>
<tr>
<td>4.10</td>
<td>Data input from the DC Buffer</td>
<td>55</td>
</tr>
<tr>
<td>4.11</td>
<td>Data input from the Input Memory</td>
<td>55</td>
</tr>
<tr>
<td>4.12</td>
<td>Selection of the first 4 lines of the input data</td>
<td>56</td>
</tr>
<tr>
<td>4.13</td>
<td>DC Buffer block diagram</td>
<td>57</td>
</tr>
<tr>
<td>4.14</td>
<td>Example of the DCT IP Core usage</td>
<td>59</td>
</tr>
<tr>
<td>A.1</td>
<td>Example of the DCT IP Core usage</td>
<td>83</td>
</tr>
<tr>
<td>A.2</td>
<td>Supported block organizations</td>
<td>86</td>
</tr>
<tr>
<td>A.3</td>
<td>Supported macroblock structures</td>
<td>86</td>
</tr>
<tr>
<td>A.4</td>
<td>Memory Maps with the relative addresses</td>
<td>87</td>
</tr>
<tr>
<td>A.5</td>
<td>Detailed Input Memory Maps showing where each sample is placed</td>
<td>87</td>
</tr>
<tr>
<td>A.6</td>
<td>Detailed Output Memory Maps showing where each sample is placed</td>
<td>88</td>
</tr>
<tr>
<td>A.7</td>
<td>Block Diagram</td>
<td>88</td>
</tr>
</tbody>
</table>
List of Tables

1.1 8 x 8 Integer DCT transform kernel ................................................. 5
2.1 Unified transform architectures ......................................................... 19
3.1 Multiplexer 1 outputs ................................................................. 24
3.2 Multiplexer 2 outputs ................................................................. 24
3.3 8 x 8 Integer DCT transform mapped into a Karnaugh map .................. 26
3.4 Types of transforms ................................................................. 26
3.5 States of the state machine ......................................................... 38
3.6 Input signals of the state machine ............................................... 38
4.1 Supported Modes of Operation ..................................................... 50
4.2 Control bits to manage the Input and Output interface ....................... 58
4.3 Process characterization for an 8 x 8 PE Array configuration ............... 60
4.4 Process characterization for a 4 x 8 PE Array configuration ............... 60
4.5 Process characterization for a 2 x 8 PE Array configuration ............... 60
5.1 Comparison of the implementation results (time and area) of the previous architecture and the new one ......................................................... 65
5.2 Implementation results for the HUTU processor with 8 x 8 PE Array .... 66
5.3 Implementation results for the HUTU processor with 4 x 8 PE Array .... 66
5.4 Implementation results for the HUTU processor with a 2 x 8 PE Array ... 66
5.5 Implementation results for the proposed IPCore with a 8 x 8 PE Array ... 68
5.6 Implementation results for the proposed IPCore with a 4 x 8 PE Array ... 68
5.7 Implementation results for the proposed IPCore with a 2 x 8 PE Array ... 68
5.8 Maximum processing rates offered by the proposed IP Core ............... 70
5.9 Minimum PE Array configuration to support some of the most commons video resolutions ................................................................. 70
5.10 Original architecture versus proposed one ..................................... 71
6.1 Comparison of some analysed architectures ..................................... 75
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>Inputs and Outputs</td>
<td>83</td>
</tr>
<tr>
<td>A.2</td>
<td>Modes of Operation</td>
<td>84</td>
</tr>
<tr>
<td>A.3</td>
<td>Process characterization for an 8 x 8 PE Array configuration</td>
<td>92</td>
</tr>
<tr>
<td>A.4</td>
<td>Process characterization for a 4 x 8 PE Array configuration</td>
<td>92</td>
</tr>
<tr>
<td>A.5</td>
<td>Process characterization for a 2 x 8 PE Array configuration</td>
<td>92</td>
</tr>
<tr>
<td>B.1</td>
<td>Two different architectures of the PE Control Unit</td>
<td>94</td>
</tr>
</tbody>
</table>
List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGU</td>
<td>Address Generation Unit</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application-Specific Integrated Circuit</td>
</tr>
<tr>
<td>AVS</td>
<td>Audio Video Standard</td>
</tr>
<tr>
<td>BRAM</td>
<td>Block RAM</td>
</tr>
<tr>
<td>CORDIC</td>
<td>Coordinate Rotation Digital Computer</td>
</tr>
<tr>
<td>DA</td>
<td>Distributed Arithmetic</td>
</tr>
<tr>
<td>DCT</td>
<td>Discrete Cosine Transform</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field-Programmable Gate Array</td>
</tr>
<tr>
<td>fps</td>
<td>frames per second</td>
</tr>
<tr>
<td>FRExt</td>
<td>Fidelity Range Extensions</td>
</tr>
<tr>
<td>GPP</td>
<td>General Purpose Processor</td>
</tr>
<tr>
<td>HDTV</td>
<td>High-Definition Television</td>
</tr>
<tr>
<td>HUTU</td>
<td>H.264/AVC Unified forward/inverse Transform Unit</td>
</tr>
<tr>
<td>IB</td>
<td>Input Buffer</td>
</tr>
<tr>
<td>IDCT</td>
<td>Inverse Discrete Cosine Transform</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>ITU</td>
<td>International Telecommunication Union</td>
</tr>
<tr>
<td>NEDA</td>
<td>New Distributed Arithmetic</td>
</tr>
<tr>
<td>PE</td>
<td>Processor Element</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>SMIC</td>
<td>Semiconductor Manufacturing International Corporation</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>TS</td>
<td>Transposition Switch</td>
</tr>
<tr>
<td>VHDL</td>
<td>VHSIC hardware description language</td>
</tr>
<tr>
<td>VLSI</td>
<td>Very-Large-Scale Integration</td>
</tr>
</tbody>
</table>
1. Introduction

1.1 Motivation

Unprecedented fast growing rates have been observed in the market of multimedia devices along the last decade and the users expectations are still growing with it. To support the execution of high definition video applications in portable devices, the development of high efficient architectures of video encoders and decoders has been an important research topic.

As one of the most computationally complex parts, in what concerns to the resulting efficiency, the computation of the transform matrix kernels is one of the topics which has deserved more attention by the research community.

The presented work consists in the development of an unified architecture to compute the transform kernels of one of the most popular video standards: the H.264/AVC, proposed by the International Telecommunication Union (ITU) and International Organization for Standardization (ISO). This standard describes and defines an efficient method of coding video that provides a better performance than any of the preceding standards[7].

Similarly to what happens in other standards, the transform coding algorithms of H.264/AVC codecs are those that contribute to most to the imposed computational cost, right after the intra- and inter-prediction, thus significantly influencing the performance of video encoding and decoding systems. Such constraints mostly result from the multi-transform nature and the finer granularity of the transform coding tool used in H.264/AVC, which supports block sizes of 8x8, 4x4 and 2x2 residue values and implements six different transforms: 8x8 and 4x4 forward and inverse integer Discrete Cosine Transform (DCT), 4x4 and 2x2 Hadamard transforms. All these transforms consist of simplified integer versions of the type-II DCT kernel used by the previous video standards, where the scaling and normalization factors were transferred to the quantization stage of the encoding algorithm. Hence, their implementation is characterized by very few different coefficient values, and can be implemented by using only integer additions, subtractions and shifts.

1.2 Objectives

The main objective of this work is to adapt an efficient, scalable and unified transform coding architecture, that can only compute the transform matrix kernels corresponding to the lower profiles of the H.264/AVC standard [8], and make it capable of implementing all the transform operations required in H.264 video coding in real time, with a minimum processing rate of 1080p HD@30 frames per second (fps).

While keeping the same performance goals, the base structure of the processor modules will be adapted to support the 8 x 8 transform matrices. With these adaptations, it will be also possible (in a future work) to make the architecture compliant with the requisites of other video standards.

Finally, after the processing structure adaptation, a suitable IP Core capable of autonomously performs the whole H.264/AVC encoding and decoding process will be designed.
1.3 H.264/AVC video encoding

As it was previously said, the H.264/AVC standard is composed by several profiles. The lower profiles are less complex and require more simple encoders and decoders. On other hand, the higher profiles are used for higher crominance sub-samplings. While the Baseline, Main and Extended profiles only use the 4 x 4 Inverse Discrete Cosine Transform (IDCT) and the 4 x 4 and the 2 x 2 Hadamard transforms to compute the DC coefficients the other profiles also use the 8 x 8 IDCT.

The H.264/AVC encoding process can be described in four main steps: prediction, transform, quantization and entropy encoding. Prediction takes advantage of spatial (Intra-predition) and temporal (Inter-predition) data redundancies in video frames to reduce the amount of data to be encoded. Due to the inherent lossy nature of the quantization step, the encoders have an embedded decoder to ensure that the prediction is computed with the same conditions of the decoder.

As it can be seen from Figure 1.1, the encoding process begins with the division of the video signal into macroblocks of 16x16 pixels and with the association of each macroblock to a reference frame. Hence, each frame may be coded either as an I-, P- or B-frame. While for I-frames the transform function is applied to the prediction difference that is obtained by applying an intra-prediction scheme (these frames are independent of other frames), for P- and B-frames the transform function is applied to the prediction error obtained from the motion compensation procedure, which exploits temporal correlation between frames (using the inter-prediction), i.e., for each P-frame the motion estimator must determine the coordinates of the macroblock that best matches its characteristics in a given reference frame. For B-frames the same estimation is done.
1. Introduction

using frames both from the past and from the future.

After the predictions, a transform function is applied to the resulting prediction residues and the transform coefficients are quantized. In this, each macroblock is represented as a matrix of frequency coefficients and many of which will be zero.

Finally, in the entropy encoding step, basic entropy encoding algorithms are applied to the computed coefficients to produce the final compressed bitstream.

Figure 1.2 represents one simple diagram of a H.264/AVC decoder. The decoding process is more simple. In fact, it is a part of the encoding process like it was previously said.

Figure 1.2: The H.264/AVC decoder diagram.

Firstly, an entropy decoding step are applied to the bitstream. Then, and after the determination of the frame type, the inverse of the quantization and transform steps are applied to the coefficients. From the addition of the decoded coefficient and the prediction estimation, computed from the previous frames and stored in a buffer memory, the resulting pixels are decoded.

Then, the decoded frames are filtered, to minimize the block effect, in adjacent blocks.

1.4 Integer Discrete Cosine Transform

A short explanation of the integer DCT properties and common methods used for its computation are presented in this section. As the main transform kernel which impulsed this work, the 8 x 8 integer DCT kernel matrix is analysed and finally, two of the most common computational methods for computing the integer DCT are also presented.

1.4.1 8 x 8 integer DCT

The 8 x 8 Integer DCT used in the high profiles of the H.264/AVC video standard, is characterized by some symmetries that can be useful to design it efficient hardware structures. The most visible symmetry is the vertical symmetry of the absolute values. As can be seen in Table 1.1 the magnitude (absolute value) of the coefficients in the 8 x 8 kernel matrix has a central vertical symmetry.

On other hand, the matrix kernel has also a vertical symmetry in the sign of the coefficients: the odd lines have direct symmetry, while the even lines have inverse symmetry. Another more
1.4 Integer Discrete Cosine Transform

Table 1.1: 8 x 8 Integer DCT transform kernel

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>6</td>
<td>3</td>
<td>-3</td>
<td>-6</td>
<td>-10</td>
<td>-12</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>-4</td>
<td>-8</td>
<td>-8</td>
<td>-4</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>-3</td>
<td>-12</td>
<td>-6</td>
<td>6</td>
<td>12</td>
<td>3</td>
<td>-10</td>
</tr>
<tr>
<td>8</td>
<td>-8</td>
<td>-8</td>
<td>8</td>
<td>8</td>
<td>-8</td>
<td>-8</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>-12</td>
<td>3</td>
<td>10</td>
<td>-10</td>
<td>-3</td>
<td>12</td>
<td>-6</td>
</tr>
<tr>
<td>4</td>
<td>-8</td>
<td>8</td>
<td>-4</td>
<td>-4</td>
<td>8</td>
<td>-8</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>-6</td>
<td>10</td>
<td>-12</td>
<td>12</td>
<td>-10</td>
<td>6</td>
<td>-3</td>
</tr>
</tbody>
</table>

complex symmetry property is also visible, corresponding to the cross symmetry in each vertical half of the matrix. However, the cost of exploiting this symmetry will probably exceed the benefits it may offer.

Finally, one other characteristic that arises from the orthogonal property of the 8 x 8 Integer DCT used by the H.264/AVC standard is that the inverse transform is the transposition of forward kernel matrix. This fact can be useful to make some simplifications, as it will be described along the document.

1.4.2 DCT Computation Algebra

Matrix computations often impose a complex challenge for efficient hardware designs. To minimize the impact of the computation of the DCT kernel matrices in video encoders and decoders, some algorithms have been devised to compute the DCT kernel matrices as efficiently as possible.

Two distinct approaches have been adopted to perform the DCT computations: the Row-Column decomposition method and the Direct Method. In the remaining of this section, these two main methods will be presented, as well as their main advantages and disadvantages.

**Row-Column decomposition**

The Row-Column decomposition method consists in dividing the 2D transform in two 1D transforms. Firstly, the coefficients (or data samples) are multiplied by the matrix kernel. Then, they are transposed and multiplied again by the same kernel.

Hence, the 8 x 8 Integer DCT is defined as:

\[ Y = C_8 X C_8^T = C_8 \left( C_8 X^T \right)^T \quad \text{(1.1)} \]

where \( C_8 \) is given by
1. Introduction

\[
C_{ef} = \begin{bmatrix}
  8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
  12 & 10 & 6 & 3 & -3 & -6 & -10 & -12 \\
  8 & 4 & -4 & -8 & -8 & -4 & 4 & 8 \\
  10 & -3 & -12 & -6 & 6 & 12 & 3 & -10 \\
  8 & -8 & -8 & 8 & 8 & -8 & -8 & 8 \\
  6 & -12 & 3 & 10 & -10 & -3 & 12 & -6 \\
  4 & -8 & 8 & -4 & -4 & 8 & -8 & 4 \\
  3 & -6 & 10 & -12 & 12 & -10 & 6 & -3 \\
\end{bmatrix}
\]

(1.2)

\[X\] represents the matrix with the input data samples, while \(Y\) is the resulting output coefficients.

From equation (1.1), equation (1.3) can be obtained:

\[
Y_{ij} = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} C_{ik} C_{jl} X_{kl} = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} C_{ik} C_{jl} X_{kl}^T, \quad i, j = 0, \ldots, N - 1
\]

(1.3)

Equations (1.4) and (1.5) illustrate the decomposition of the 2D transform in two 1D transforms.

\[
Y_{ij} = \sum_{k=0}^{N-1} C_{ik} M_{kj} = \sum_{k=0}^{N-1} C_{ik} M_{jk}^T, \quad j, k = 0, \ldots, N - 1
\]

(1.4)

with:

\[
M_{jk}^T = \sum_{l=0}^{N-1} C_{jl} X_{lk}^T, \quad i, j = 0, \ldots, N - 1
\]

(1.5)

Finally, and by using the equations (1.4) and (1.5), the matrix-product form of the 2D H.264/AVC transform can be obtained as it can be seen in the equation (1.6), where the two 1D transforms can be written again in the same form as equation (1.1).

\[
[Y]_{ij} = Y = CM^T = C (CX^T)^T = CXC^T
\]

(1.6)

Hence, and according to equation (1.7), the 2D H.264/AVC transform can be divided in two similar steps.

\[
Y = CM^T = C (CX^T)^T
\]

(1.7)

According to this formulation, the kernel matrix (C) is firstly multiplied by the block of transposed samples. Then, the matrix kernel is multiplied again by the transposition of the result of the first multiplication. In this way, it is possible to design architectures with only one matrix of multipliers and one additional module for transpositions.
1.5 Main contributions

Direct Method

Another method often used to compute the DCT transform is the direct method. The direct method computes the DCT by adopting the polynomial transform. It requires fewer computations, but incurs in a greater amount of irregularity.

This method consists in decomposing all the coefficient computations in some simple operations involving the input data samples. The set of the operations is designed and fixed for each transform kernel. In Figure 1.3 can be observed one example of the implementation of the 1-D 8 x 8 forward integer transform using the direct method devised in [1].

![Figure 1.3: The architecture for 1-D 8 x 8 forward integer transform using the direct method extracted from [1].](image)

Due to the characteristics of this method, it turns to be only attractive for low power chip designs, since its irregularity makes it too complex to Very-Large-Scale Integration (VLSI) based implementations.

1.5 Main contributions

The main contribution of the presented work is the conception of a efficient and scalable architecture to compute the forward and inverse integer DCT to be incorporated within the encoder and decoder structures of the H.264/AVC video standard.

The adaptations conducted from the original architecture make it able to compute the whole set of transform kernels defined in the H.264/AVC video standard. Moreover, the devised IP Core allied to the scalability property of the devised architecture make it adaptable to several different applications like being used in mobile devices or as co-processor of an General Purpose Processor (GPP).
1. Introduction

1.6 Dissertation outline

This dissertation is organized as follows. After the brief introduction and motivation that was presented in this chapter, chapter 2 will describe some other alternative and state of the art proposals of hardware architectures to encode and decode image and video. The architecture that served as base to this work will be also presented in this chapter. Chapter 3 contains the description of all the adaptations that were required in the base architecture to compute the 8 x 8 transforms. Chapter 4 describes the definition and implementation of the proposed IP Core.

Finally, in chapter 5 it will be possible to discuss all the obtained experimental results, as well as some functional examples of the proposed IP Core. Chapter 6 discusses the final conclusions and addresses possible future work directions.
## State of the art

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Introduction</td>
<td>10</td>
</tr>
<tr>
<td>2.2 General transform computational units</td>
<td>10</td>
</tr>
<tr>
<td>2.3 Integer transform computational units</td>
<td>13</td>
</tr>
<tr>
<td>2.4 Base Architecture</td>
<td>15</td>
</tr>
<tr>
<td>2.5 Summary</td>
<td>17</td>
</tr>
</tbody>
</table>
2. State of the art

2.1 Introduction

In this chapter it will be presented the state of the art of dedicated processing structures for the computation of the DCT as well as the analyses of different types of architectures. While the section 2.2 presents the architectures which are able to compute some different transforms, the section 2.3 presents the architectures which are able to compute some (Non-Unified) or the whole set (Unified) of the transforms defined in the H.264/AVC standard.

At the end of this chapter it will be presented one high performance and configurable architecture for the most common profiles of H.264/AVC, based on a systolic processor. This structure served as a base architecture for the conducted work.

As it was referred in chapter 1, the main objective of the presented research will be the modification of this base architecture which is able to compute the 4 x 4 and the 2 x 2 transform kernels, to an improved version that is also able to compute the 8 x 8 transform kernels.

2.2 General transform computational units

2.2.1 Single transform architectures

Being the most popular and effective transform applied to image and video coding, the DCT has been the object of several studies and many architectures have been proposed to efficiently perform this transform.

The architecture proposed in [9] uses the direct method, which consists in using the polynomial transform and mapping each real number input into a complex number. With this mapping it is possible to reduce the computation complexity by applying rotation techniques in the complex space. Despite using more hardware resources, this architecture brought an increment of data throughput to the proposed architectures until that date.

Similarly, in [10] it is presented an architecture that uses the direct method to compute the 2D DCT. This architecture was designed to perform the direct 2D DCT with a low power consumption, by using low power adders and memories. The presented simulations show that this architecture can operate at 100MHz with 138mW of power consumption, when implemented in a 0.6 µm single-poly double-metal technology. Meeting the requirements of the real-time High-Definition Television (HDTV) signal processing for the 4:2:0 and 4:2:2 chrominance subsampling format, this architecture can be used in low power equipments like portable computers or personal digital assistants, which are the most popular portable devices at the time of the proposed architecture.

On other hand, the architecture proposed in [11] uses the row-column decomposition technique to perform the 2D DCT and 2D IDCT for one MPEG-2 Video Encoder. This architecture has not been recognized as an efficient method for implementing MPEG-2 video encoder among hardware designers, but it was a good propose in terms of the hardware resources utilization.
The architecture proposed in [2] is an enhanced version of the New Distributed Arithmetic (NEDA) [12] structure. This architecture is able to perform the 1D DCT with a high performance. The high-throughput is achieved by pipelining the architecture (see Fig. 2.1). In every clock cycle, it receives eight pixels as inputs and produces eight DCT coefficients. Implemented in 0.35 \( \mu m \) technology it runs at 1.5 GHz and processes 108 Gbps of image/video sequence data. The 1D DCT can be used together with the addition of transposition hardware, to efficiently perform the 2D DCT.

![1-D DCT architecture extracted from [2].](image)

Other architecture to perform the 2D DCT using the direct method is proposed in [13]. One more time, the main advantage of the architectures using the direct method, when compared to the architectures using the row-column decomposition method is the operating speed. This architecture can run with a clock rate of 200MHz when implemented in a 0.6 \( \mu m \) double-metal technology.

The architecture proposed in [14] uses a modified multiplierless Coordinate Rotation Digital Computer (CORDIC) (Coordinate Rotation Digital Computer) [15] arithmetic to perform the 2D DCT with low-power consumption.

Based on the row-column decomposition method the architecture proposed in [3] uses Distributed Arithmetic (DA) to perform the 2D DCT. To efficiently compute the 2D DCT it uses two modules of DA to perform the 1D DCT as well as a transposer Random Access Memory (RAM).

Another architecture based on the row-column decomposition method is proposed in [16]. This architecture is able to perform the 2D DCT for the JPEG encoding. Just like in [3], it uses two modules to compute the 1D DCT with a transposition RAM like it can be observed in Fig. 2.2.

### 2.2.2 Multi transform architectures

Some video CODECs need to use more than one transform kernel. One example is the H.264/AVC, which needs to use Hadamard transforms to encode the DC coefficients in some of its encoding modes. In this section, it will be presented some proposed architectures that can
2. State of the art

![2-D DCT architecture extracted from [3].](image)

operate in multiple standards or can perform several different transform kernels.

The architecture proposed in [17] is a multi-standard architecture that can compute various transform kernels of the H.264/AVC. This architecture uses the row-column decomposition method and can compute the 4 x 4 and the 8 x 8 inverse integer transforms of the H.264/AVC standard, and the 8 x 8 inverse integer transform of the Audio Video Standard (AVS).

Similarly, the architecture proposed in [18] uses the row-column decomposition to perform the 8 x 8 inverse integer transform for the H.264/AVC standard and the 8 x 8 inverse integer transform for the VC-1 standard.

The architecture proposed in [19] is also based on the row-column decomposition method and is able to compute the 4 x 4 forward integer transform and the 4 x 4 and the 2 x 2 Hadamard transforms used in H.264/SVC standard.

One more decoding architecture for multiple video standards is proposed in [20]. This architecture, also based on the row-column decomposition method, is able to decode the AVS, VC-1, MPEG-4 and H.264 video standards.

Also based on row-column decomposition, but using multipliers divided in a common part and in a separated part (see fig. 2.3) to efficiently compute the transforms of multiple video standards, the architecture proposed in [4] is able to decode HD video with a 768 Megapixels/sec throughput. This architecture is also easily expandable, by simply adding new optimized arithmetic to compute the coefficients of new codecs.

In [21] it is proposed one reconfigurable architecture to perform 1D transforms. This architecture can compute the forward and the inverse transforms of the MPEG-2/4, the VC-1, the H.264/AVC and the AVS video codec standards. It is also easily reconfigured to compute other transforms. Despite being designed to perform 1D transforms, this architecture can easily compute 2D transforms, by applying the row-column decomposition method with the addition of one transpose memory.

The architecture proposed in [22] is another multi-standard decoder. It uses the relationship between the coefficients of all the transforms to minimize the hardware usage. With the lowest computational parameters, the IDCT of the AVS was used as a base. Then, different delta coefficient matrices were developed to compute other IDCT matrices from the coefficients of the AVS.
2.3 Integer transform computational units

2.3.1 Non-Unified transform architectures

H.264/AVC, one of the last video coding standard, support several profiles depending on the video resolution and the available bit rate. For the lowest profiles, only 4 x 4 transform kernel matrices are used, this permits to design some specialized and particularly efficient architectures to compute 4 x 4 transforms. In this sub-section, it will be reviewed some architectures that can compute a subset of the H.264/AVC transform kernels. In the next sub-section it will be presented some architectures that are able to compute all the transform kernels adopted by the H.264/AVC video standard.

The architecture proposed in [23] is able to perform the 2D forward and inverse integer DCTs of the lowest H.264/AVC profiles. This architecture exploits the symmetry of the 4 x 4 matrix to simplify the hardware. With this simplification, the processing structure is divided in two operation units that compute one 2 x 2 matrix each. Furthermore, this structure also has one transpose register array to compute the 2D transforms using the row-column decomposition technique. When implemented with 0.35 µm technology process, this architecture uses 3524 transistors and operates at more than 120 MHz.

By adopting a faster algorithm the architecture proposed in [24] can compute the forward 4 x 4 integer DCT of the H.264/AVC. By applying a Kronecker product and a direct addition operation, the architecture of the fast 2D 4 x 4 forward integer transform can be derived from the fast 1D 4 x 4 forward integer transform through matrix decompositions. Contrary to the architectures based on the row-column method, the architectures supported on this fast algorithm do not need any transpose memory.

Also using this fast algorithm, the architecture proposed by the same author in [25] can com-
2. State of the art

compute the 4 x 4 and the 8 x 8 H.264/AVC transform kernels. This proposal consists in two schemes based on a shared hardware architecture, one for the forward transforms and another for the inverse transforms. In 0.18 \( \mu m \) CMOS VLSI technology process implementation, the scheme for the forward transforms uses 6458 gates and the scheme for the inverse transforms uses 6573 gates. Both schemes can run at 125 MHz.

The architecture proposed in [26] uses the same fast algorithm as in [24], but this architecture is designed to perform the 1D inverse integer transforms for the H.264/AVC and VC1 video standards.

Based on the fast papilionaceous algorithm, the architecture proposed in [5] can compute the 2D 8 x 8 integer DCT of the H.264/AVC Fidelity Range Extensions (FRExt). Implemented with the Semiconductor Manufacturing International Corporation (SMIC) 0.18 \( \mu m \) technology process, this architecture can run at 300 MHz, processing 735 fps of 4:2:0 HDTV (1280x720) video images.

Figure 2.4: Fast papilionaceous algorithm flowchart (extracted from [5]).

Finally, the architecture proposed in [27] is a forward integer transform for a H.264/AVC Intraframe encoder, optimized to battery-supplied devices.

2.3.2 Unified transform architectures

Some architectures were also developed to compute the whole set of transforms of the H.264/AVC standard. These architectures are most useful to allow the processing devices to adapt their operations to distinct modes of the H.264/AVC standard.

The architecture proposed in [28] uses the traditional butterfly structures to decompose the 4 x 4 and the 8 x 8 transforms. This architecture can compute the 4 x 4 forward and inverse DCTs, the 4 x 4 Hadamard transform and the 8 x 8 forward and inverse DCTs. Despite being able to compute most of the transforms of the H.264/AVC standard, this architecture cannot compute the 2 x 2 Hadamard transform.

On the other hand, the architecture proposed in [1], based on the fast algorithm, is able to compute the whole set of transforms for the H.264/AVC standard. This architecture consists of 6 specialized architectures, one for each transform, mapped to an unified two stages pipelined architecture, which processes 8 pixels per clock cycle.

Another unified architecture was proposed in [6]. This architecture is composed by two main
modules, one for the forward transforms and other for the inverse transforms (see fig. 2.5), which can operate at 162.1 and 230.9 MHz, respectively.

Figure 2.5: The proposed FIT (a) and IIT (b) modules (extracted from [6]).

2.4 Base Architecture

The base architecture that served as a starting point for the conducted research work was a systolic processor that can implement the DCTs for the most common profiles of H.264/AVC. This architecture is composed by a control unit and three functional modules: an array of processor elements, a row-column transposition switch and an input buffer.

The next sub-sections detail each of these three functional modules.

2.4.1 Array of PEs

For the Baseline, Extended and Main profiles of the H.264/AVC standard (the profiles supported by this architecture) only the 4 x 4 forward (eq. 2.1) and inverse (eq. 2.2) integer DCT and the 4 x 4 (eq. 2.3) and the 2 x 2 (eq. 2.4) Hadamard transforms are required. These transforms can be computed by using only integer powers of 2 coefficients, which allows to compute each single coefficient by simply shifting the input signal value.

The processor element designed for the base architecture is composed by two main blocks: the arithmetic module and the control module. In the arithmetic module, the data values to be processed ($X_{in}$) are multiplied by a specific multiplier value (i.e., 1, -1, 2, -2, $\frac{1}{2}$ or $-\frac{1}{2}$) and accumulated with the value computed by an adjacent Processor Element (PE) in a previous clock cycle. This final value is also stored in another internal standing-data register before being broadcasted to the next PEs, in order to shorten the critical path of the circuit [8].

Besides the processing datapath, each processor element also contains one small control module which controls the operation to be done and transmits it to the neighbor PE by the simple signal interface that can be seen in Figure 2.6.

$$C_f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}$$ (2.1)
2. State of the art

The array is formed by 16 \((4 \times 4)\) PEs and it is scalable in \(2 \times 4\) and \(1 \times 4\) configurations (see fig. 2.7) that permit to reduce the hardware costs and the power consumption, with a consequent penalization of the latency and data throughput. This scalability is a great advantage and a differentiating factor of this architecture and it will be preserved in the conducted work.

![Figure 2.6: Base architecture of the processor element.](image)

\[
C_i = \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 0 & -1 & -1 \\
1 & -1 & 1 & 1 \\
1 & -1 & -1 & -1
\end{bmatrix} \quad (2.2)
\]

\[
H_{4\times4} = \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & -1 & -1 \\
1 & -1 & 1 & 1 \\
1 & -1 & -1 & -1
\end{bmatrix} \quad (2.3)
\]

\[
H_{2\times2} = \begin{bmatrix}
1 & 1 \\
1 & -1
\end{bmatrix} \quad (2.4)
\]

![Figure 2.7: Scalable PE Array.](image)
2.4.2 Input Buffer

The input buffer is used to feed the PE array. This unit is highly required to minimize the delays when accessing the external data memories where the input data is stored, and also to guarantee a regular data flow within the systolic array. When the PE array is implemented with less than four lines, the input buffer stores the input data to be loaded by the PE array more than once. The structure of this unit can be observed in figure 2.8.

In addition, this unit is also capable of feeding the transform processing array with the previously processed row-column transposed data [8].

![Figure 2.8: The Input Buffer.](image)

2.4.3 Transposition Switch

The Transposition Switch (TS) is used to implement a hardwired row-column transposition of the data processed by the 1-D transform kernel in the PE array. It consists in a set of multiplexers that allow the direct row-column transposition for the 4 x 4 and the 2 x 2 blocks of data. Furthermore, this transposition switch allows the inclusion of a variable and programmable delay, to support the row-column transposition operation when other configurations of the array with fewer PEs are considered.

This module is not scalable. According to the authors [8], its scalability would significantly increase the complexity of the control module and would not provide any relevant performance advantages.

In Figure 2.9 it is illustrated a possible configurations of the Transposition Switch, interconnected to the remaining modules of the architecture, considering four lines in the PE array.

2.5 Summary

Several architectures to implement the forward and inverse DCT in image and video standards have been analyzed in this chapter. The majority of these architectures are supported on two techniques: the row-column decomposition and the direct method.
2. State of the art

Figure 2.9: The Base Architecture.

Although most of the transform kernels are based on the DCT, the similarities between the transform kernels permit to design architectures to compute multi-transforms using shared hardware, like in [4] and [22].

Some of the non-unified architectures for the computation of the DCTs in the H.264/AVC standard that were analyzed in this chapter are good solutions for portable devices and dedicated applications. The majority of these architectures have a good relationship between hardware costs, data throughput and power consumptions. However, in applications where it is needed to perform all the transforms required by the H.264/AVC standard, these non-unified architectures are not useful.

The unified architectures that are able to compute all the DCTs required by the higher profiles of the H.264/AVC standard are the most relevant to analyze in this work. In Table 2.1 it is presented a comparison between some unified transform architectures that were considered in the conducted analysis.

The architecture that was used as a starting point of the conducted research is able to efficiently compute all the H.264/AVC profiles, except the High Profiles. The scalability of the PE’s array permits it to easily adapt to the limitations of some specific applications, such as portable devices with low power capacity.

Upgrading this base architecture to support the computation of 8 x 8 transforms is a significant improvement, not only because it will allow the implementation of the High Profiles of H.264/AVC, but also because most image and video CODECs use 8 x 8 transforms. With this improvement, it will be even possible to implement the recently proposed HEVC video standard.
## 2.5 Summary

### Table 2.1: Unified transform architectures

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[28]</td>
<td>FPGA StratixII</td>
<td>Enc</td>
<td>n.a.</td>
<td>100 MHz</td>
<td>5143 LUTS</td>
</tr>
<tr>
<td>[28]</td>
<td>FPGA StratixII</td>
<td>Dec</td>
<td>n.a.</td>
<td>100 MHz</td>
<td>5258 LUTS</td>
</tr>
<tr>
<td>[28]</td>
<td>AMS 0.35 ASIC</td>
<td>Enc</td>
<td>n.a.</td>
<td>82 MHz</td>
<td>35347 Gates</td>
</tr>
<tr>
<td>[28]</td>
<td>AMS 0.35 ASIC</td>
<td>Dec</td>
<td>n.a.</td>
<td>83 MHz</td>
<td>35232 Gates</td>
</tr>
<tr>
<td>[6]</td>
<td>FPGA</td>
<td>Enc</td>
<td>1336.8 Mpps</td>
<td>167.1 MHz</td>
<td>2.1K Slices</td>
</tr>
<tr>
<td>[6]</td>
<td>FPGA</td>
<td>Dec</td>
<td>1068.0 Mpps</td>
<td>133.5 MHz</td>
<td>2.1K Slices</td>
</tr>
<tr>
<td>[6]</td>
<td>0.18μm ASIC</td>
<td>Enc</td>
<td>1296.8 Mpps</td>
<td>162.1 MHz</td>
<td>62.2K Gates</td>
</tr>
<tr>
<td>[6]</td>
<td>0.18μm ASIC</td>
<td>Dec</td>
<td>1847.2 Mpps</td>
<td>230.9 MHz</td>
<td>33.3K Gates</td>
</tr>
</tbody>
</table>
2. State of the art
The proposed architecture

Contents

3.1 PE Array ................................................................. 22
3.2 Input Buffer ............................................................. 33
3.3 Transposition Switch .................................................. 34
3.4 Main Control Unit ...................................................... 37
3.5 Operation of the whole architecture ............................... 37
3.6 Summary ................................................................. 40
3. The proposed architecture

In this chapter it is presented the set of performed modifications in the base architecture of the previously described processor. In order to make it capable to compute 8 x 8 transforms, as well as all the previous supported transforms, it was necessary to adapt the whole architecture.

In section 3.1 it is presented the set of adaptations in the PEs and the PE array. Section 3.2 describes the modifications performed in the Input Buffer module. Section 3.3 presents the adaptations performed in the Transposition Switch. Section 3.4 describes the modifications in the Control module. Finally, in section 3.5 it is presented one example of the operation of the whole proposed architecture.

3.1 PE Array

As it was discussed in chapter 2, the PE array consists of a set of PEs that compute one 1D transform in a systolic way. Since this architecture is now required to allow the computation of 8 x 8 transforms, 8 columns of PEs must be instantiated in the PE array.

However, this not only required the adaptation of the array to efficiently support these new operation modes, but also the definition of a new PE structure, capable of computing all the operations for the forward and the inverse 8 x 8 and 4 x 4 DCT as well as the 4 x 4 and 2 x 2 Hadamard transforms.

3.1.1 Architecture of the new Processor Element

The High Profiles of H.264/AVC, used in the FRExt [29], allow the use of the 8 x 8 integer DCT (see eq. (3.1)). In order to adapt the existent architecture to compute this transform, it was necessary to add some circuitry in the arithmetic structures.

The added complexity came from the set of coefficients of the 8 x 8 integer DCT. While the coefficients computed by the base architecture were obtained by simple arithmetic shifts that were chosen with two 2-to-1 multiplexers, the new coefficients need larger multiplexers and one extra adder.

The performed modifications in the PE architecture, covering the Datapath and the Control Unit, can be seen, in detail, in the next sub-sections.

3.1.1.A Datapath

The first modifications implemented in the architecture of the PE involved the Datapath Unit, which is required to compute the new set of coefficients corresponding to the 8 x 8 transforms, as well as the previous ones, with maximum efficiency.

To simplify, all the coefficients of the forward H.264/AVC 8 x 8 DCT (see eq. (3.1)) were multiplied by 8. In this way, all the involved coefficients became integer numbers that can be decomposed as one or two powers of 2. This manipulation makes it possible to compute them only with
3.1 PE Array

one or two shifts and one addition. In addition it allows to perform all the involved computations without any fractional parcels.

In the end, the final result is divided by 8 (which is simply performed by a 3-bit arithmetic shift-right).

The kernel matrix of the inverse H.264/AVC 8 x 8 DCT is simply the transposed kernel of the forward transform. As a consequence, all the coefficients of the inverse transform can be computed similarly to the forward transform.

\[
C_{cf} = \begin{bmatrix}
8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
12 & 10 & 6 & 3 & -3 & -6 & -10 & -12 \\
8 & 4 & -4 & -8 & -8 & -4 & 4 & 8 \\
10 & -3 & -12 & -6 & 6 & 12 & 3 & -10 \\
8 & -8 & -8 & 8 & 8 & -8 & -8 & 8 \\
6 & -12 & 3 & 10 & -10 & -3 & 12 & -6 \\
4 & -8 & 8 & -4 & -4 & 8 & -8 & 4 \\
3 & -6 & 10 & -12 & 12 & -10 & 6 & -3 \\
\end{bmatrix} \frac{1}{8} \tag{3.1}
\]

The first step consists in the decomposition of the absolute value of all the coefficients of the 8 x 8 integer DCT (i.e., 12, 10, 8, 6, 4 and 3) in integer powers of 2. Such decompositions are represented in equations 3.2 to 3.7.

\[
12 = 2^3 + 2^2 \tag{3.2}
\]
\[
10 = 2^3 + 2^1 \tag{3.3}
\]
\[
8 = 2^3 \tag{3.4}
\]
\[
6 = 2^2 + 2^1 \tag{3.5}
\]
\[
4 = 2^2 \tag{3.6}
\]
\[
3 = 2^1 + 2^0 \tag{3.7}
\]

Equations 3.8 to 3.10 represent the same decompositions, but for the coefficients of the 4 x 4 DCT and the Hadamard transforms, which are already powers of 2.

\[
2 = 2^1 \tag{3.8}
\]
\[
1 = 2^0 \tag{3.9}
\]
\[
\frac{1}{2} = 2^{-1} \tag{3.10}
\]

As it can be seen, all the coefficients can be computed using the add-and-shift operation, and by considering one single adder. To minimize the hardware cost of this circuit, all the combinations of shift operations were carefully organized, so that only two multiplexers are required. In addition, as it can be seen in Figure 3.1, all the coefficients of the 4 x 4 and the 2 x 2 transforms are computed using only one multiplexer, in order to simplify the control unit.
3. **The proposed architecture**

To further optimize the circuit implementation, negative multiplier values all computed by converting this result to the two’s complement. This is implemented by performing a bit-by-bit XOR operation to the computed coefficient and by setting the Cin bit of the accumulator, thus avoiding the use of a subtractor.

In what concerns to the performance of the circuit, it should be noted that the powers of two are simply computed by hardwired arithmetic shifts, which does not introduces delays. Hence, all the significant delays result from the multiplexers and the adders. The selectors of each multiplexer, which can be seen in Table 3.1 and Table 3.2, are decoded by the control unit (detailed in the next sub-section). Although the decoding of these signals is still a bit complex, the total delay of this operation is much lower than when using a 23-bit multiplier.

![Figure 3.1: PE Datapath.](image)

### Table 3.1: Multiplexer 1 outputs.

<table>
<thead>
<tr>
<th>$S_1S_0$</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>$X_{in}$</td>
</tr>
<tr>
<td>01</td>
<td>$X_{in} \times 4$</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>$X_{in} \times 8$</td>
</tr>
</tbody>
</table>

### Table 3.2: Multiplexer 2 outputs.

<table>
<thead>
<tr>
<th>$S_1S_0$</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>$X_{in} \times 4$</td>
</tr>
<tr>
<td>01</td>
<td>$X_{in} \times 2$</td>
</tr>
<tr>
<td>10</td>
<td>$\frac{X_{in}}{2}$</td>
</tr>
<tr>
<td>11</td>
<td>$X_{in}$</td>
</tr>
</tbody>
</table>
3.1.1.B Control Unit

The control unit is responsible for propagating all the control signals between the PEs, in order to turn it possible to perform all the computations in a systolic way. Furthermore, this unit is also responsible to generate the control signals used by the PE datapath, to compute the transform coefficient in each PE.

To decode the control signals that control the selectors \((S_1, S_0)\) of the two multiplexers, the input of the XOR gates and the Cin input of the adder, the control unit uses the identification (type) of the desired transform and the coordinates of the coefficients within the kernel matrix, in order to select the right coefficient used in the computations in each PE.

The H.264/AVC 8 x 8 DCT kernel (see eq. (3.1)) has a vertical symmetry axis, that controls the coefficients absolute values (see eq. (3.11)). Hence, only half of the matrix needs to be taken in the account when defining the signals that controls the selectors of the multiplexers.

Furthermore, other simple changes of the base architecture were also made:

- The addition of the T2D signal, which will be set to ‘1’ when the PE is computing the second 1D transform. This signal is asserted at the end of the computations, to indicate that the data is ready to be sent to the output port.
- The CLR signal, which is used to clear the accumulator, was further simplified. With the conducted changes, this signal is no longer passed from the exterior. Instead, this signal is now implemented with some simple internal logic that permits or not the accumulation of the value computed by the previous PE.

Two alternative architectures were tested to understand which one has the best relation between the hardware cost and speed. Since the results were not significantly different, the choice of the adopted architecture was not very important. Even so, the small differences can be observed, as described in Appendix B.1.

$$C_{ef} = \begin{bmatrix}
8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
12 & 10 & 6 & 3 & -3 & -6 & -10 & -12 \\
8 & 4 & -4 & -8 & -8 & -4 & 4 & 8 \\
10 & -3 & -12 & -6 & 6 & 12 & 3 & -10 \\
8 & -8 & -8 & 8 & 8 & -8 & -8 & 8 \\
6 & -12 & 3 & 10 & -10 & -3 & 12 & -6 \\
4 & -8 & 8 & -4 & -4 & 8 & -8 & 4 \\
3 & -6 & 10 & -12 & 12 & -10 & 6 & -3
\end{bmatrix}$$

(3.11)

The coordinates of the kernel matrix are mapped in two 3-bit signals \(X_2X_1X_0\) and \(Y_2Y_1Y_0\). These signals are used to obtain the control signals with the help of some Karnaught maps. Firstly, to determine the best mapping organization, the 8 x 8 kernel matrix was directly mapped into a Karnaught map, as it can be seen in Table 3.3. From this table it is possible to better understand the relationship between the coordinates of the 8 x 8 kernel matrix and the corresponding coefficients.
3. The proposed architecture

Table 3.3: 8 x 8 Integer DCT transform mapped into a Karnaught map.

<table>
<thead>
<tr>
<th></th>
<th>000</th>
<th>001</th>
<th>011</th>
<th>010</th>
<th>110</th>
<th>111</th>
<th>101</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>001</td>
<td>12</td>
<td>10</td>
<td>3</td>
<td>6</td>
<td>-10</td>
<td>-12</td>
<td>-6</td>
<td>-3</td>
</tr>
<tr>
<td>011</td>
<td>10</td>
<td>-3</td>
<td>-6</td>
<td>-12</td>
<td>3</td>
<td>-10</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>010</td>
<td>8</td>
<td>4</td>
<td>-8</td>
<td>-4</td>
<td>4</td>
<td>8</td>
<td>-4</td>
<td>-8</td>
</tr>
<tr>
<td>110</td>
<td>4</td>
<td>-8</td>
<td>-4</td>
<td>8</td>
<td>-8</td>
<td>4</td>
<td>8</td>
<td>-4</td>
</tr>
<tr>
<td>111</td>
<td>3</td>
<td>-6</td>
<td>-12</td>
<td>10</td>
<td>6</td>
<td>-3</td>
<td>-10</td>
<td>12</td>
</tr>
<tr>
<td>101</td>
<td>6</td>
<td>-12</td>
<td>10</td>
<td>3</td>
<td>12</td>
<td>-6</td>
<td>-3</td>
<td>-10</td>
</tr>
<tr>
<td>100</td>
<td>8</td>
<td>-8</td>
<td>8</td>
<td>-8</td>
<td>8</td>
<td>-8</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

After some manual simplifications, and in conjugation with the work already done for the base architecture, several logical functions were obtained (eq. (3.13)), (eq. (3.15)), (eq. (3.17)) and (eq. (3.18)), which correspond to the signals that control the first two multiplexers and where $t_{2h}$, $t_{4h}$, $t_{4f}$, $t_{4i}$, $t_{8f}$ and $t_{8i}$ define the cases where the PE is processing the 2 x 2 and the 4 x 4 Hadamard transforms, the 4 x 4 (forward and inverse) and the 8 x 8 (forward and inverse) integer DCT, respectively.

The signals $t_{2h}$, $t_{4h}$, $t_{4f}$, $t_{4i}$, $t_{8f}$ and $t_{8i}$ are directly obtained from the type of transform that is being computed, given by the input signal $T_{TYPE}$. The definition of these signals can be seen in Table 3.4.

Table 3.4: Types of transforms.

<table>
<thead>
<tr>
<th>T_Type(3 to 0)</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0X00</td>
<td>$t_{2h}$</td>
</tr>
<tr>
<td>0X01</td>
<td>$t_{4h}$</td>
</tr>
<tr>
<td>0010</td>
<td>$t_{4f}$</td>
</tr>
<tr>
<td>0011</td>
<td>$t_{8f}$</td>
</tr>
<tr>
<td>0110</td>
<td>$t_{4i}$</td>
</tr>
<tr>
<td>0111</td>
<td>$t_{8i}$</td>
</tr>
</tbody>
</table>
Like it was previously said, only the multiplexer 2 is used to compute all the transforms previously supported by the original architecture. As it is shown in Table 3.1, $C_0$ must be ‘0’ to activate the input with value ‘0’ and, in this way, turn the final result only dependent of the multiplexer 2. Hence, with the simplifications that can be seen in (eq. (3.12)), equation (3.13) was obtained. The coordinate signal $X_2$ was not used in the simplifications due to the symmetry that can be seen in (eq. (3.11)).

Concluding, the $C_0$ must be ‘0’ for all the $4 \times 4$ and $2 \times 2$ transforms and for when the $8 \times 8$ coefficient assumes the values less or equals to ‘4’ (see Table 3.3).

\[
C_0 = \left( (Y_2 + Y_1 + X_1 + X_0) \cdot (Y_2 + Y_1 + X_1 + X_0) \right) \cdot (Y_2 + Y_1 + X_1 + X_0) \cdot (Y_2 + Y_1 + X_1 + X_0).
\]

\[
(Y_2 + Y_1 + Y_0 + X_1 + X_0) \cdot (Y_2 + Y_1 + Y_0 + X_1 + X_0) \cdot (Y_2 + Y_1 + Y_0 + X_1 + X_0) \cdot (t8f + t8i).
\]

(3.13)
3. The proposed architecture

Similarly, the $C_1$ coefficient (eq. (3.15)) was obtained from eq. (3.14) and by considering that $C_1 = 1$ for the 4 x 4 and the 2 x 2 transforms.

$$C_1 = ((Y_2.Y_1.Y_0.X_0) + (Y_2.Y_1.X_0.X_0) + (Y_2.Y_1.X_1.X_0) + (Y_2.Y_1.X_0.X_1) + (Y_2.Y_1.Y_0.X_1.X_0) + (Y_2.Y_1.Y_0.X_1.X_0) + (Y_2.Y_1.Y_0.X_1.X_0) + (Y_2.Y_1.Y_0.X_1.X_0) + t8f + t8i)$$

$$C_2_{nsj} :$$

Similarly, the $C_2$ coefficient (eq. (3.16)) was obtained from eq. (3.15) and by considering that $C_2 = 1$ for the 4 x 4 and the 2 x 2 transforms.
3.1 PE Array

The \( C_2 \) signal controls multiplexer 2, and is also used to generate the coefficients of the transforms that were already computed by the base architecture. Consequently, to obtain its logical function, it must be taken into account that \( C_2 \) must be '0' when the \( t \) coefficient is selected for the \( t4i \) transform, and \( C_2 = 1 \) in the remaining cases. The karnaught maps do not help to achieve a simpler logic function, since there are only four coefficients that must be divided by 2 and there is no direct relationship between them. So, the minterms corresponding to these four specific cases were simply added to the original logical function (3.17).

\[
C_2 = ((Y_2 Y_1 Y_0 X_1) + (Y_2 Y_0 X_0) + (Y_2 Y_1 Y_0 \overline{X_1}) + (Y_2 Y_0 \overline{X_0}) +
(Y_2 Y_1 Y_0 \overline{X_1}) + (Y_2 Y_0 X_1) + (t4i + t8f + t8i) +
(t4i ((Y_1 Y_0 X_1 X_0) + (Y_1 Y_0 \overline{X_1} X_0) + (Y_1 Y_0 X_1 X_0) + (Y_1 Y_0 \overline{X_1} \overline{X_0}))))
\]

(3.17)

To generate the coefficients of the H.264/AVC 8 x 8 integer DCT, only the first two data signals of the multiplexer 2 are used. Consequently, only the coefficients of all the transforms previously supported by the original architecture are considered to activate \( C_3 \) to '1'. In all the considered transforms, only the 4 x 4 forward DCT has coefficients greater than 1, which requires the second input of the multiplexer. As a result, the logical function of \( C_3 \) (see eq. 3.18) can be obtained straightforwardly.

\[
C_3 = t4h + t2h + t4i +
(t4f, ((Y_1 Y_0 \overline{X_1} X_0) + (Y_1 Y_0 X_1 X_0) + (Y_1 Y_0 \overline{X_1} X_0) + (Y_1 Y_0 X_1 \overline{X_0}))))
\]

(3.18)

In all of these functions, whenever \( X_2 = 1, X_1 \) and \( X_0 \) are replaced by their symmetrics, due to the symmetry already presented. Furthermore, when \( t8i = 1 \) the X's signals are replaced by Y's (and Y's by the X's), since the inverse transform kernel is the transposition of the forward transform kernel.
3. The proposed architecture

The $C_4$ signal is the signal that controls whether the datapath unit multiplies the coefficient by -1 or not, to add or subtract the computed coefficient to the accumulated value. As observed before, the signal of the coefficients of the 8 x 8 integer DCT has a symmetry in the odd lines and an inverse symmetry in even lines.

Hence, with the simplifications obtained in (eq. (3.19)) and the logical functions implemented in the base architecture for the others transforms, the logic function for the $C_4$ signal was obtained (see eq. (3.20)).

$$C_4 = ((X_0, \overline{X}_1, Y_1) \cdot (i8f + t8i)) + ((X_0, Y_0, Y_1) \cdot (i8f + t8i)) +$$
$$((X_1, \overline{X}_0, (Y_0 \oplus Y_1)) \cdot (i8f + t8i)) + ((X_0, (Y_1, Y_0, (X_1 \oplus t2h)) \cdot (i8f + t8i)) +$$
$$(((Y_2, ((X_2, ((Y_1, Y_0) + (Y_0, X_1, X_0)) + (Y_1, Y_0, X_1))) + Y_1, ((Y_0, X_2, X_0)) +$$
$$Y_2, ((\overline{Y}_1, ((\overline{Y}_0, (X_1, X_0)) + (X_1, X_0, X_0)) + (Y_0, X_2, X_0)) + (Y_1, Y_0, (X_2, X_1)))) +$$
$$((\overline{X}_2, ((X_2, ((\overline{X}_1, X_0) + (Y_0, Y_1, X_0) + (X_1, \overline{X}_0, Y_1))) + (X_1, Y_2, Y_1))) +$$
$$X_1, ((X_0, (\overline{Y}_2, Y_0)) + (X_2, ((\overline{Y}_2, Y_0) + (X_0, Y_2, Y_0)))) +$$
$$((\overline{X}_2, ((X_2, ((\overline{X}_1, Y_0) + (Y_1, Y_0)) + (X_0, Y_2, Y_0))) + (X_1, X_0, Y_2, Y_0)).t8i))$$

3.1.2 The 8 x 8 Array

Although the enhanced version of the architecture retains the scalability properties of the original design, already described in Fig. 2.9, it was decided that only the 8 x 8, the 4 x 8 and the
2 x 8 configurations were to be supported in the new architecture. The 1 x 8 configuration was abandoned because it does not offer a good compromise between hardware costs and efficiency.

The 8 x 8 PEs configuration of the new PE Array is shown in the Figure 3.2. The lines represent not only the data signals, but also the control signals of the PEs.

![Figure 3.2: Scalable Array of PEs.](image)

### 3.1.3 Scalability of the array

Like it was previously said, this architecture retains the scalability properties of the original design. In figure 3.3 and in figure 3.4 is presented the 2 alternative configurations of the PE array.

When the array is implemented with less than 8 lines of PEs, it also includes one small unit responsible for changing the Y coordinates when the array starts to compute the second part of the 1D transform. Furthermore, in these cases, all the output signals are linked to the existent lines of PEs even if those lines are already connected to other output signal, i.e., if the array is configured to have only 4 lines of PEs they are connect not only to the first four output lines but also to the last four lines (see fig. 3.3 and fig. 3.4). In this way, the design of the other modules
3. The proposed architecture

Figure 3.3: Array of PEs with 4 x 8 configuration.

Figure 3.4: Array of PEs with 2 x 8 configuration.

does not need to be changed when the number of lines in the array is changed. This characteristic was already present in the base architecture.

The original architecture was able to compute one 4 x 4 transform or two 2 x 2 transform at a time. For computing two 2 x 2 transforms at time the original architecture of the array has two output signals connected directly to the outputs of the second column of PE. When the first two columns were computing the second 1-D transform of the first block, the third and fourth columns could be computing the first 1-D transform of the second block.

With the definition of the new set of possible configurations in this improved architecture, some changes in the PE Array had to be implemented. The most significant modification consisted in the addition of the output signals for the simultaneous computation of two 4 x 4 transforms and four 2 x 2 transforms, since now the array have eight columns instead of four. However, the final proposed architecture is only able to compute two 2 x 2 transforms at time, since the rate of the loads in the Input Buffer will not be sufficient for continuous computing of more than two 2 x 2 transforms.

Assuming that the whole architecture have the same clock rate it is not possible to load values to be computed in the last columns of the array while the first columns are processing 2 x 2 transforms, i.e., one 2-D 2 x 2 transform require 4 clock cycles of computations in each PE but it
3.2 Input Buffer

is needed 8 clock cycles to load values to all the columns of the array.

The computation in parallel of two 4 x 4 transforms is processed similarly to the process of computing two 2 x 2 transforms in the original architecture, i.e., the first 1-D transform of the second block is computed while the first columns are computing the second 1-D transform of the previous block.

3.2 Input Buffer

As a result of resizing the systolic array to accommodate 8 columns of PEs, the architecture of the input buffer was also extended in order to accommodate the required 8 shift-registers. Moreover, the central control unit was also modified: it was added a small control unit in each shift-register, to ease the definition of state of each shift-register, instead of having only one counter to define the state of the whole input buffer. This change proved to be very helpful when the architecture is used to compute transforms with kernels smaller than 8 x 8 matrices, in order to know when each element is ready to be loaded with new values.

3.2.1 Input buffer element

Similarly to what happens in the original architecture (see Fig. 2.9), the input buffer elements are responsible for storing one line of data of the matrix to be processed. To make the computation of the 8 x 8 transforms possible, the shift-registers capacity had to be extended from 4 to 8 data elements. In addition, in order to allow the computation of the 4 x 4 transforms for array configurations using only 2 lines of PEs, these shift-registers were split in two units of 4 data elements each. Hence, in a 2 x 8 array configuration, the Serial Input of each Input Buffer Element, which ensures the circular comportment of the buffer elements, is directly linked to the fourth position, instead of being linked to the first (see fig. 3.5).

As it was previously said, a small control unit was added to each Input Buffer (IB) Element. This control unit is simply one counter that controls the number of reads that will be needed to load the new data values. The number of reads is calculated by taking into account the type of transform to be computed and the configuration of the PE Array. The signal R2R indicates that there are available reads in the IB Element. Hence, with the addition of one simple counter in each element, it was gained more flexibility to control the input buffer even with different transforms sizes are being computed at the same time.

3.2.2 Central control unit

With the addition of some extra control logic in each element, it was possible to simplify the central control unit. The implemented counters in the input buffer elements permitted the remove of two counters from the central unit.
3. The proposed architecture

This unit defines the next element to be read from the buffers and verifies if such data is already available to be sent to the array of PEs. This unit also produces one single signal (R2W) that indicates if the next element is ready for to be written or to be read. It was assumed that if one element is ready to be read, the previous element is ready too. In fact, since the elements are always loaded by order, it is impossible that the second element has a valid data and the first does not for example.

In Figure 3.5 it is illustrated the proposed Input Buffer structure, composed of 8 elements.

![Figure 3.5: Modified Input Buffer.](image)

The included multiplexers are used to choose the transposed data from the transposition switch, during the computation of the second 1-D transform. The selection input of the multiplexers identified with the label ‘1’ controls the size of the circular buffers. It is generated with the signal corresponding to the type of transform being implemented. As it was previously said, the second input of these multiplexers is active when the processor is computing one 4 x 4 transform, with only two lines of PEs in the array. Finally, the selection input of the multiplexers identified with the label ‘2’ are generated with the signal ib_sel, defined by the control unit of the processor.

3.3 Transposition Switch

The introduced modifications in the architecture of the original Transposition Switch (see Fig. 2.9) aimed at supporting the transposition of the data for the computation of the 8 x 8, the 4 x 4 and the 2 x 2 transforms.

The improved structure has three data input ports, one for each transform size. The signals received at each of these ports came from a set of multiplexers, whose amount and number of data inputs equals the size of the corresponding transform. The 8 x 8 inputs are connected to 8-to-1 multiplexers, the 4 x 4 inputs are connected to 4-to-1 multiplexers and the 2 x 2 inputs are connected to 2-to-1 multiplexers.
3.3 Transposition Switch

3.3.1 Delay buffer

When the PE array is instantiated with less lines than the dimension of the kernel matrix being computed it is necessary to re-use the PE lines to compute all the lines of the kernel matrix. So the result of the first 1-D matrix have to be stored in a delay buffer while the PE lines are being re-used. So, the number of registers needed are obtained by $\text{Kernel Dimension} - \text{NumberOf PE Lines}$.

The delay buffer was conveniently adapted in order to allow saving the data from the 8 x 8 transforms. The implemented modifications consisted in instantiating four registers in the first memory level and two registers in the second level.

The first level uses the four registers to compute 8 x 8 transforms, when the PE array has only four lines in the PE Array. The two extra register of the second level are used for all the transforms (except the 2 x 2 Hadamard transform) when the PE Array only has two lines of PEs. In that case, whenever the transform to be computed is the 8 x 8 integer DCT, the two levels of registers are used together, in order to be possible to store six values before the computation of the second 1D transform.

Figure 3.6 illustrates the modified architecture of the proposed Transposition Switch, with 8 delay buffers.

3.3.2 Control of the transposition switch

For efficiency and complexity reasons, the control of this unit was decentralized through all the buffers. For such purpose, all the signals that pass through the PE Array are used. Such signals consist of the computed data values, the type of the transform to be implemented and the following three signals:

- The NEW_T signal (1 bit), which indicates that the corresponding data value is the first of a new block. This signal is used to appropriately reset all the counters at the Transposition Switch.

- The T2D signal, used during the row-column decomposition, indicates that the corresponding data value belongs to the second 1D transform and can be directly passed to the output, without the addition of any delay at the delay buffer.

- The CALC signal, which indicates that the corresponding data value is valid. This signal is used to increment the counters at the Transposition Switch.

All of these signals are replicated to the three different data inputs.

The selection inputs of the multiplexers are asserted by the counters that are controlled by the signals above described.
3. The proposed architecture

Figure 3.6: Modified Transposition Switch structure.
3.4 Main Control Unit

The main control unit is responsible for the control of all the interactions between all modules of the processor. This unit was not significantly changed from its original version. In fact, the most relevant modification consisted on the addition of one intermediate state between the computation of the first and the second 1D transform. This state introduces one stall cycle in the operation of the PE Array and it is required as a result of the introduction of a new register at the output of the transposition switch. (see section 3.5)

The state machine of this control module is represented in Fig. 3.7, where Sstdby is the initial state. The processing starts when the “STC” signal is activated and the state machine advances to the state SldIBs (load Input Buffers). It stays there until a whole block of data is loaded into the Input Buffer.

Due to the densed processing schema, the systolic array computes the first 1D transform with this operation, i.e., while all the input buffers are loaded the first 1-D transform is fully computed in the first PE. Consequently, as soon as the Input buffers are completely loaded and the data of the first transform is already computed, the machine goes into state Stall. This state is used to make the PE Array to wait for the values of the Transposition Switch, as a result of the modifications that were introduced in this enhanced version of the processor. Nonetheless, this operation mode can be slightly different, whenever the number of lines of PEs is smaller than the size of the transform. In such cases, the data in the array of PEs is not yet fully processed when the input buffers get completely loaded. As a result, state SrunS1 is used to complete the processing of such data.

The state SinitS2 is used to control the load of the data at the outputs of the Transposition Switch into the Array of PEs through the Input Buffer, as well as to realize the second 1D transform. Similarly to state SrunS1, state SrunS2 is used to complete such computations whenever the size of the array is smaller than the size of the considered transform. As soon as the transform is completed in SinitS2 or SrunS2 states, the machine will move to the SldIBs state if the “STC” is still active (i.e., the data of another block is already available to be processed) or to the Sdone state otherwise.

The Sdone state lasts only one clock cycle. The machine will move to the SldIBs state if the “STC” is active or Sstdby if it is not.

In Table 3.5 and Table 3.6 can be observed a summary of the states and signals involved in the state machine.

3.5 Operation of the whole architecture

Figure 3.8 illustrates the block diagram of the whole processor, with the units previously described. When compared with Figure 2.9, the complexity of all the units has slightly increased, in order to support the 8 x 8 transforms.
3. The proposed architecture

![State Machine Diagram](image)

**Figure 3.7: State Machine.**

**Table 3.5: States of the state machine.**

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sstdby</td>
<td>The stand by state where the architecture stays stopped waiting for the signal to start the computations</td>
</tr>
<tr>
<td>SldIBs</td>
<td>State where the Input Buffers are loaded with the data to be computed</td>
</tr>
<tr>
<td>SrunS1</td>
<td>State where the computations of the first 1-D transform are realized</td>
</tr>
<tr>
<td>Sstall</td>
<td>One clock cycle delay to wait for the values from the TS to the IB</td>
</tr>
<tr>
<td>SinitS2</td>
<td>The state used to start the computations of the second 1-D transform</td>
</tr>
<tr>
<td>SrunS2</td>
<td>The state used to finish the computations of the second 1-D transform</td>
</tr>
<tr>
<td>Sdone</td>
<td>The final state of the computations, when the two 1-D transforms are computed</td>
</tr>
</tbody>
</table>

**Table 3.6: Input signals of the state machine.**

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STC</td>
<td>Start Transform Computations - Used to start the whole process</td>
</tr>
<tr>
<td>BRL</td>
<td>Block of Residues fully Loaded - Used to indicate that the Input Buffer is fully loaded</td>
</tr>
<tr>
<td>EOT</td>
<td>End Of the 1-D Transform - Used to indicate the end of each 1-D transform</td>
</tr>
</tbody>
</table>
8-to-1 multiplexers were used to transpose the data of the 8 processing lines. However, the introduction of those multiplexers led to a consequent increase of the processing path. This extra delay slightly unbalanced the critical path of the Processor Elements. Since it did not exist any registers between the output of Transposition Switch and the Input of the PE Array, all this extra delay was accumulated to the delay of the first PE. To restore the balance of the processing path, it was introduced one additional register (identified by number 1 in fig. 3.8) between the TS and the PE Array (at the input of the Input Buffer unit).

To illustrate the computation procedure of one 2D transform, in Figure 3.9 it is represented the dataflow within the PEs Array for the computation of an 8 x 8 transform using this new proposed architecture.

Figure 3.9(a) represents the time instant when the first coefficient is computed in the first
3. The proposed architecture

Then, all the coefficients are computed and accumulated with the values obtained from the previous PEs. This is represented in Figure 3.9(b) to Figure 3.9(h). The introduction of the extra stall state is visible in Figure 3.9(i). At this instant, there are no coefficients to be loaded in the first PE, but the computations in the other PEs continue.

The stall operation is propagated through the PE Array in the subsequent clock cycles, while the computation of the second 1D transform is performed. The computation of the second 1D transform begins at the instant corresponding to Figure 3.9(j). Figure 3.9(r) corresponds to the beginning of the computation of the first 1D transform of the following block. At this time, the extra stall cycle is not needed, because the data comes again from the input.

3.6 Summary

The modifications that were introduced to the base architecture, in order to adapt it to be able to compute 8 x 8 transforms were presented in this chapter.

The most significant modifications focused the datapath of the processor element, to make it capable to compute the new coefficients. These changes increased the complexity of the datapath, but also the complexity of the control. Since there are more coefficients to be computed, the logic to choose them is more complex.

Some small modifications were also introduced in the PE Array and in the Input Buffer. These mostly consisted in the addition of more lines and more columns of processing elements.

Due to the increase of the propagation delay in the datapath, which resulted from the addition of 8-to-1 multiplexers in the Transposition Switch, it was necessary to add one register between the outputs of the transposition switch and the inputs of the PE Array. This increased the processor latency between the computation of the first and the second 1D transform by one clock cycle.
3.6 Summary

(a) $t=t_0$
(b) $t=t_0+1$
(c) $t=t_0+2$
(d) $t=t_0+3$
(e) $t=t_0+4$
(f) $t=t_0+5$
(g) $t=t_0+6$
(h) $t=t_0+7$
(i) $t=t_0+8$
(j) $t=t_0+9$
(k) $t=t_0+10$
(l) $t=t_0+11$
(m) $t=t_0+12$
(n) $t=t_0+13$
(o) $t=t_0+14$
(p) $t=t_0+15$
(q) $t=t_0+16$
(r) $t=t_0+17$
(s) $t=t_0+18$

Figure 3.9: Dataflow of the computation of a 2D 8 x 8 transform.
3. The proposed architecture
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>IP Core Structure</td>
<td>44</td>
</tr>
<tr>
<td>4.2</td>
<td>Memory Devices</td>
<td>45</td>
</tr>
<tr>
<td>4.3</td>
<td>Control Unit</td>
<td>46</td>
</tr>
<tr>
<td>4.4</td>
<td>DC Buffer</td>
<td>56</td>
</tr>
<tr>
<td>4.5</td>
<td>Input and Output Interface</td>
<td>57</td>
</tr>
<tr>
<td>4.6</td>
<td>Operation modes</td>
<td>58</td>
</tr>
<tr>
<td>4.7</td>
<td>Example configuration</td>
<td>61</td>
</tr>
<tr>
<td>4.8</td>
<td>Summary</td>
<td>62</td>
</tr>
</tbody>
</table>
4. DCT IP Core

This chapter presented the input/output interface of the processor described in the previous chapter. This interface provides the control signals to the processor and is also used to transfer the data under processing from the exterior devices.

With the proposed IP Core, composed by the interface and by the underlying systolic processor, it is possible to autonomously encode all the profiles of the H.264/AVC with the 4:2:0 chroma subsampling.

4.1 IP Core Structure

In order to support the desired functionality, the proposed IP Core is composed by three modules represented in Figure 4.1. The H.264/AVC Unified forward/inverse Transform Unit (HUTU) consists of the systolic processing array architecture, previously described in chapter 3.

The Input and Output Memory Banks allow the continuous operation without any addition of waiting or delay cycles. They act just like buffers to prevent any possible delays in concurrent accesses to the systems memories. By incorporating 2 pages in each memory bank, the IP Core can compute the data of one page while the main processor loads/stores the data of the other page.

The DC Buffer stores the DC coefficients to be computed in the 16 x 16 Intra Luma and Chroma mode.

Finally, the Control Unit is responsible for the control of the whole IP Core. As it can be seen in Figure 4.1 this unit is responsible to generate the signals that control the dataflow inside the IP
Core, as well as some control signals to drive the input data to the correct positions of the input buffer. This procedure involves multiplexer ‘2’ and will be further described in section 4.3.4.

4.2 Memory Devices

In order to achieve a continuous operation within the systolic array, the proposed IP Core includes two dual-port memory banks for the input and the output of the data. Each memory bank was designed to implement a double buffering scheme, so that data transfers within the IP Core can simultaneously occur with the processing of a block in the systolic processor. In this way, when one page of the input memory bank is in use by the IP Core, the other page can be used to load the data to be processed in the next processing cycle. Similarly, when one page of the output memory bank is in use by the IP Core to store the computed values, the values that were computed in the previous processing cycle can be loaded by the main processor in the other page.

As it was previously said, the proposed IP Core is able to perform a continuous operation, due to a convenient partition of the H.264/AVC algorithm. It was decided to optimize this continuous operation to the level of the macroblock. In this way, each page is able to store one macroblock of luma samples or four macroblocks of chroma samples. Internally, it is constituted by 8 different sub-banks, one for each matrix column.

According to the H.264/AVC standard the coefficients of the transform, as well as the data to be transformed, have to be represented with 16 bits. As a consequence, each bank is composed by 36 positions with 16 bits each (little endian): 32 for the lines (or columns) of all blocks and more 4 to store the 4 x 4 DC coefficients matrix in the 16 x 16 Luma Intra mode and the four 2 x 2 matrices of DC Coefficients in the chroma mode. This results in 576 bytes for each memory page.

A detailed description of the data organization for each mode is presented below.

4.2.1 Default and 16 x 16 Intra Modes

In Figure 4.3(a) it is illustrated the representation of the 16 4 x 4 blocks of pixels (see also Figure 4.2(a)) that compose each 16x16 luma macroblock. Since the memory pages were organized in 8 sub-banks (in order to simplify the storage of the 8 x 8 blocks), it means that two lines of a 4 x 4 block can be stored in the same memory positions of one line of a 8 x 8 block. Moreover, to respect the pixels order in the whole macroblock (raster mode), the second line of the first block is stored after the first line of the fourth block. Thanks to this alternation of the pixel data in the memories that compose the 8 different sub-banks and to the address generator module that will be presented in section 4.3.3, the data to be processed can be write in a sequential way and is conveniently stored to be processed by the systolic array.

In Figure 4.4(a) it is represented the memory map for the default organization of the luma
4. DCT IP Core

macroblocks. Figure 4.4(b) represents the memory map for the Intra 16x16 luma macroblocks. In this case, the last four positions are occupied with the 4 x 4 DC coefficients. Combining these Memory maps with Figure 4.3(a) it is more simple to understand the alternation above described and observe where each 4 x 4 block is placed.

To the systolic processor correctly process the data at the input memories, this data must be transposed, so in the first positions of the input memory are present the first column of the first block, as represented in Figure 4.5(a). In the output memory, the IP Core will save the data with the correct order, without the need of any transposition, as represented in Figure 4.6(a).

4.2.2 Intra 8 x 8 Mode

For macroblocks encoded using the 8 x 8 mode, represented in Figure 4.3(b), the data of each block (Figure 4.2(b)) is saved in alternate rows to respect the order of pixels in macroblock like in the 16 x 16 modes. In this way, all of the samples or quantized coefficients are placed in the memory like it can be seen in Figure 4.4(c). The last four positions are not used, since in this mode the Hadamard transform is no applied to the DC coefficients.

Again, at the input memory the data should be transposed, in the Figure 4.5(b) it can be seen the positions of the firsts two blocks in the input memory. The output memory will have the blocks disposed line by line like in the Figure 4.6(b).

4.2.3 Chroma Mode

A macroblock in the 4:2:0 format has four 4 x 4 Chroma blocks (see Figure 4.2(a)). Consequently, the memory banks densed for the proposed IP Core are able to store the data concerning four macroblocks. Moreover, for each 8 x 8 chroma macroblock there is one 2 x 2 DC coefficient block (Figure 4.2(c)) that is saved in the 4 last memory positions in the order represented in Figure 4.4(d). Since they are different macroblocks and each one has only 8 columns the columns or lines are inserted in memory in the original order.

At input memory all data should be transposed, including the DC coefficients when they are present, the example of one block stored in the input memory can be seen at Figure 4.5(c). The output memory will have the configuration represented in Figure 4.6(c).

As it can be seen in Figure 4.3(c)

4.3 Control Unit

The central control unit is responsible for generating the addresses for the input and for the output memory banks, as well as to control the whole dataflow between the modules of the IP Core.
4.3 Control Unit

The control unit receives the mode of operation as a configuration input and, with the help of an internal state machine, provides to the core architecture with the data and the control signals to perform the whole process of encoding or decoding a macroblock.

4.3.1 Operation modes

The proposed circuit supports 8 different operation modes, which correspond to all the possible transform processing modes defined in the H.264/AVC standard. Namely,

- Forward Luma Default mode, in which the IP Core performs the forward 4 x 4 integer DCT to each of the 16 blocks that compose a macroblock of luma samples;
- Forward Luma Intra 16x16 mode, in which the IP Core performs the forward 4 x 4 integer DCT to each of the 16 blocks of samples, followed by the 4 x 4 Hadamard transform to the resulting 16 DC coefficients;
- Forward Chroma transform, in which the IP Core performs the forward 4 x 4 integer DCT to each of the 4 chroma blocks, followed by the 2 x 2 Hadamard transform over each one of the four 2 x 2 block of DC coefficients;
- Forward Luma Intra 8x8 mode, in which the IP Core performs the forward 8 x 8 integer DCT over each one of the 4 luma blocks;
4. DCT IP Core

Figure 4.4: Memory Maps with the relative addresses.

(a) Default Luma Macroblock
(b) Intra 16x16 Luma Macroblock
(c) 8 x 8 Intra Luma Macroblock
(d) Chroma macroblocks

Figure 4.5: Detailed Input Memory Maps showing where each sample is placed.
4.3 Control Unit

- Inverse Luma Default mode, in which the IP Core performs the inverse $4 \times 4$ integer DCT over each one of the 16 blocks of coefficients;

- Inverse Luma Intra 16x16 mode, in which the IP Core starts by performing the $4 \times 4$ Hadamard transform to the block composed by the 16 DC coefficients, followed by the inverse $4 \times 4$ integer DCT to each one of the 16 blocks of coefficients, by using the decoded DC coefficients of luma blocks;

- Inverse Chroma mode, in which the IP Core starts by performing the $2 \times 2$ Hadamard transform over each one of the four $2 \times 2$ blocks of DC coefficients, followed the inverse $4 \times 4$ integer DCT over each one of the four blocks of coefficients of the 4 macroblocks by using the just decoded DC coefficients;

- Inverse Luma Intra 8 x 8 mode, in which the IP Core performs the inverse $8 \times 8$ integer DCT over each one of the four blocks of coefficients.

These eight modes of operation are encoded using 4 bits, as shown in Table 4.1. This allows to easily extend the IP Core functionality in the future, in order to support other operation modes.

4.3.2 State Machine

The operation of the IP Core is controlled by a state machine, whose state diagram can be seen in Figure 4.7. Sstdby is the initial state, and is used to maintain the IP Core stopped while there is no data to be processed.
### Table 4.1: Supported Modes of Operation.

<table>
<thead>
<tr>
<th>Code</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Forward Luma Default transform</td>
</tr>
<tr>
<td>0001</td>
<td>Forward Luma Intra 16x16 transform</td>
</tr>
<tr>
<td>0010</td>
<td>Forward Chroma transform</td>
</tr>
<tr>
<td>0011</td>
<td>Forward Luma Intra 8x8 transform</td>
</tr>
<tr>
<td>0100</td>
<td>Inverse Luma Default transform</td>
</tr>
<tr>
<td>0101</td>
<td>Inverse Luma Intra 16x16 transform</td>
</tr>
<tr>
<td>0110</td>
<td>Inverse Chroma transform</td>
</tr>
<tr>
<td>0111</td>
<td>Inverse Luma Intra 8x8 transform</td>
</tr>
</tbody>
</table>

**Figure 4.7:** Main state machine of the developed IP Core.
4.3 Control Unit

When new data is loaded on the input memory, the state machine goes to state $S_{\text{init}}$. In this state, the mode of operation is evaluated to define if the operation begins with the computation of the DC coefficients or with the AC coefficients. The next state will be $S_{\text{DC,Init}}$ or $S_{\text{AC,Init}}$ as a result of such evaluation. This decision is also used to initialize the address generator modules with the correct address. Both of these states only last one clock cycle.

In the $S_{\text{AC}}$ state, the machine processes the AC coefficients of all the blocks of the macroblock. If the operation mode specifies that the DC coefficients are also needed to be computed, then the state machine transits to the $S_{\text{DC,Wait}}$ state, until the data needed for the DC computation is ready. The same process occurs for the $S_{\text{DC}}$ state. After all the computations are completed in these two states, the state machine goes to the state $S_{\text{doneAC}}$, or $S_{\text{doneDC}}$, until the Input Buffer is ready and loaded with a new values. From these states, the state machine goes to state $S_{\text{init}}$, if the mode of operation is the same of the previous Macroblock, or goes to state $S_{\text{stdby}}$ otherwise.

4.3.3 Address Generation Units

The memory addresses for the read and write operations involving the input and the output memory banks are generated by using dedicated Address Generation Units (AGUs), which use for such purpose a set of multiplexers connected to constant signals, controlled by the mode of operation and the state machine.

Although similar AGUs are used to interface the input and the output memory banks, the use of two distinct and independent circuits allows parallel access to the 2 banks, as well as some flexibility to implement the PE Array with different line configurations. For example, page swaps at the end of each macroblock are achieved by only using one register, which updates its state to its complementary value at the end of the processing each macroblock.

From the point of view of the IP Core, each memory bank is independently addressed and word by word. Hence, the control unit only needs to generate addresses between 0 and 35.

Input addresses

To help the implementation of the address generation, it was adopted a counter that counts the number of columns that has already loaded into memory and another counter that counts the number of the block being processed. These counters are initialized with value ‘0’ and they start counting at the beginning of a macroblock computation.

The memories need to be addressed one clock cycle before the data is available. Because of this, it is necessary one extra state for initializations, before the start of the computations. In this state, the address of the first element of the first block is generated.

Figure 4.8 depicts the main components of the address generator for the input memory where the main inputs are constant signals. This constant signals represent the internal memory ad-
4. DCT IP Core

dresses. These addresses can be related to the addresses of the Figure 4.4 after they are divided by 16 (2 bytes of each 8 sub-banks, since internally all the sub-banks are addresses in parallel).

![Figure 4.8: Address Generator for the input memory.]

For multiplexers 1 and 2, the selection between the two inputs is made using the cbcr bit, which is directly decoded from the mode code and takes the value ‘1’ when the core is processing one block of crominances. The second input of multiplexer 3 is activated when the core is processing in the Intra 8 x 8 mode or when it is computing the DC coefficients of the crominances. All these selections are decided as a function of the number of the block to be loaded given by one internal counter of the control unit.

Multiplexer 4 is used to set the initial addresses for both the AC mode (first input) and the DC mode (second input). Consequently, the second input of multiplexer 5 is only active for the initializations of the AC mode or the DC mode.

The selection of the second input of multiplexer 6 is done with the help of the counter that counts the number of columns already loaded. At the beginning of each block, the address of the first position is given by the offset multiplexers.

The “Cin” bit of the adder is always activated, except for the crominance mode or for the DC mode.

Output addresses

The AGU of the output memory bank also uses two counters. However, in this specific case the counter of columns counts the number of valid lines at the output of the processor, i.e., the counter is only incremented when one valid value is at the output of the processor. This happens because the intermediate values of the DCT computation are also present at the output of the
4.3 Control Unit

processor and they are not valid to be stored in the output memory bank.

Since the processed output data could be still stored while the IP Core is processing the next macroblock, to correctly generate the output addresses based on the operation mode, the signals that indicate the mode of operation to the input AGU are stored in registers to this unit at the end of each macroblock.

In Figure 4.9 it depicted the main components of the address generator for the output memory.

![Address Generator for the output memory.](image)

Although this AGU is quite similar to the one that was used to interface with the input memory bank, its operation mode is slightly different, as it can be seen in Figure 4.9. The main difference is that the output memory is addressed in the same clock cycle as the data is available at its input. This fact allows to avoid all the initializations. Concerning this, in fact the first address can be given by the same multiplexers that gives all the first addresses of each block.

The selectors of multiplexers 1, 2 and 3 are similar to the input AGU.

However, since now the first multiplexers are also used for the initializations, the second input of multiplexers 2 and 3 are activated also in the DC mode of Intra 16 x 16 luma mode. The second input of multiplexer 4 is activated at the beginning of each block.

The "Cin" bit of the adder has the same behavior that in the input AGU.

Due to the diagonal wavefront behavior of the processor, the output address is linked to one array of registers to generate the addresses for the 8 sub-banks with the delay of one clock cycle between each one. This delay replicates the wavefront behavior to the 8 memory sub-banks.
4. DCT IP Core

4.3.4 Control Signals

Besides generating the addresses for the input and output memory banks, the control unit is also responsible for controlling the whole process of the IP Core (see fig. 4.1). In this section, all the signals involved in such operations are described.

- **DI_SEL** - This 4 bits signal is used to select the input from the memory or from the DC Buffer (for the first 4 lines of data inputs).
- **DI_SEL2x2** - This signal is connected to a multiplexer that selects between the pair of lines 0 and 1 or the pair of lines 2 and 3 to be loaded in the input buffer. It is used in the 2 x 2 transforms, to connect the 2 coefficients to the 2 first positions, whichever the memory bank or line of the DC Buffer they are stored.
- **DI_SEL4x4** - Similarly to the previous signal, this signal is used in the 4 x 4 transforms to connect the 4 coefficients to the 4 first positions of the input buffer.
- **MEM_I_SEL** - This signal is used to control the swap between the two pages of the input memory bank.
- **MEM_O_SEL** - This signal does the same as the previous one, but in the output memory bank.
- **DO_EN** - 8 bits that control the enable signals of the output memory banks.
- **DC_B_EN_WR** - Enable signals that control the write operations of each of the 4 DC Buffer columns.
- **DC_B_EN_RD** - Enable signals that control the read operations of each of the 4 DC Buffer lines.

Moreover, this unit also generates the signal to start the computations and the signal that indicates the type of the transform to be computed.

Signals DI_SEL, DI_SEL2x2 and DI_SEL4x4 are used to select of the lines of the input data. In what concerns the DC Buffer, the selection of the two first lines is done by signal DI_SEL2x2, as it can be seen in Figure 4.10.

At the same time, a similar procedure is done with the data from the input memory. However, in this case it is necessary to choose between the first 4 lines and the last 4 lines in the 4 x 4 transforms. As it can be seen in Figure 4.11, this selection is accomplished with the help of the DI_SEL4x4 signal.

The signal dcb_out_sel is obtained with the signal DC_B_EN_RD, as it is defined in eq. (4.1). When only one line of the DC buffer is activated for a read it will be linked to the first line of the
4.3 Control Unit

Figure 4.10: Data input from the DC Buffer.

Figure 4.11: Data input from the Input Memory.
4. DCT IP Core

input buffer. This happens for the inverse transforms. When more than one line of the DC buffer is activated for a read, all the lines are linked in the normal order.

\[
dcb_{out\_sel}(0) = DC_{B\_EN\_RD}(0).DC_{B\_EN\_RD}(1).DC_{B\_EN\_RD}(2).DC_{B\_EN\_RD}(3) + DC_{B\_EN\_RD}(0).DC_{B\_EN\_RD}(1).DC_{B\_EN\_RD}(2).DC_{B\_EN\_RD}(3)
\]

\[
dcb_{out\_sel}(1) = DC_{B\_EN\_RD}(0).DC_{B\_EN\_RD}(1).DC_{B\_EN\_RD}(2).DC_{B\_EN\_RD}(3) + DC_{B\_EN\_RD}(0).DC_{B\_EN\_RD}(1).DC_{B\_EN\_RD}(2).DC_{B\_EN\_RD}(3)
\]

Finally, the DI_SEL signal selects which of the lines will be loaded in the input buffer, as it can be seen in Figure 4.12. The last 4 lines always come from the input memory, and are only used in the 8 x 8 transforms that do not need these choices, since all the positions in one memory line belongs to the same block.

![Figure 4.12: Selection of the first 4 lines of the input data.](image)

4.4 DC Buffer

In the 16 x 16 Intra Luma mode and in the Chroma mode it is necessary to compute the Hadamard transforms involving the previously computed DC coefficients of all the blocks of a macroblock. Although all these DC coefficients could be loaded from the output memory bank, which stores all the results of the previously computed DCTs, that would lead to a significant increase of the complexity of the output memory and its corresponding AGU. To ensure a fast and easy access to these DC coefficients, it was implemented a small memory unit, capable of storing at most 16 data values and of simultaneously unloading at most 4 of them at a time. To compute these DC coefficients, it is necessary to load one line of a 4 x 4 block at a time.

Hence, the DC Buffer unit only consists of sixteen 16-bit registers, disposed in a 4 x 4 array. There is one input per line, so that the data from the 4 columns can be simultaneously stored.
4.5 Input and Output interface

All the outputs and inputs have individual address generators, which are incremented whenever a read or a write operation is done, respectively.

Moreover, this unit produces a signal ("DA") that will be active whenever there is, at least, one valid value in the buffer. The signal ("FULL") will be active when all positions of the buffer have valid values. These two signals are used by the Control Unit to know when it is possible to start the computation of the DC coefficients.

As it can be seen in Figure 4.13 which illustrates the architecture of the DC buffer, the positions of the inputs and the outputs allow the transposition of the DC coefficients data. This transposition is necessary to load these coefficients into the Input Buffer, so that the processor can compute them correctly.

Figure 4.13: DC Buffer block diagram.

4.5 Input and Output interface

All the signals that are used to control the communication between this IP Core and the external are available in one 16-bits register. The specifications of this register can be seen in Table 4.2.

Like it can be observed in Figure 4.14 in addition to the 16-bit register it is necessary to connect the DCT IP Core to the memory shared with the GPP for the buses of input and output data. So, the interface of the proposed IP Core is composed by 2 data buses and 1 control bus.

**Input control signals:**

- **Mode** - Indicates the mode of operation (see details in section 4.3.1);
- **DFL0** - Data Fully Loaded on page 0 of the input memory - Indicates that data is fully loaded and ready to be computed on memory page 0;
Table 4.2: Control bits to manage the Input and Output interface.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>C₀ to 3</td>
<td>Mode</td>
</tr>
<tr>
<td>C₄</td>
<td>DFL0</td>
</tr>
<tr>
<td>C₅</td>
<td>DFL1</td>
</tr>
<tr>
<td>C₆</td>
<td>RDFC0</td>
</tr>
<tr>
<td>C₇</td>
<td>RDFC1</td>
</tr>
<tr>
<td>C₈</td>
<td>DFL0S</td>
</tr>
<tr>
<td>C₉</td>
<td>DFL1S</td>
</tr>
<tr>
<td>C₁₀</td>
<td>DFC0</td>
</tr>
<tr>
<td>C₁₁</td>
<td>DFC1</td>
</tr>
<tr>
<td>C₁₄</td>
<td>EN</td>
</tr>
<tr>
<td>C₁₅</td>
<td>RST</td>
</tr>
</tbody>
</table>

DFL1 - Data Fully Loaded on page 1 of the input memory - Indicates that data is fully loaded and ready to be computed on memory page 1;

RDFC0 - Reset Data Fully Computed on page 0 of the output memory - After reading all the data of output memory page 0 this bit is used to reset the status bit DFC0;

RDFC1 - Reset Data Fully Computed on page 1 of the output memory - After reading all the data of output memory page 1 this bit is used to reset the status bit DFC1;

EN - Global Enable operation of the IP Core;

RST - Synchronous Reset of the IP Core.

Output control signals:

DFL0S - Data Fully Loaded from page 0 of the input memory Status - Assumes the value '0' when the page 0 of the input memory can be written again;

DFL1S - Data Fully Loaded from page 1 of the input memory Status - Assumes the value '0' when the page 1 of the input memory can be written again;

DFC0 - Data Fully Computed already available on Output memory 0 - When data is fully computed and ready to be read on Output memory 0 the IP Core will set this bit to '1';

DFC1 - Data Fully Computed already available on Output memory 1 - When data is fully computed and ready to be read on Output memory 1 the IP Core will set this bit to '1'.

4.6 Operation modes

One of the main features of the proposed IP Core is the ability to run, in a fully autonomous way, all the computations that are needed to encode a video macroblock using the H.264/AVC standard. In this section, it is presented a detailed explanation to use the IP Core in all of its modes of operation.
The computations begin as soon as the RST signal is released to low and the DFL bit of the memory page in use is activated. The initialization procedure takes 3 clock cycles until the system is effectively computing the new values. After this initialization stage, the time needed to perform the encoding/decoding of one macroblock depends on the considered operation mode.

In tables 4.3, 4.4 and 4.5 it can be seen the number of clock cycles that are required to process one macroblock of video, where column A represents the initialization stage, column B represents the computation of the first transform (AC or DC, depending on if it is a encoding or decoding process, respectively), column C represents the transition from the first transform to the second transform, column D represents the second transforms computation (when it is supported), column E represents the final procedures to store the values remaining in the PE Array in the output memory and column F represent the number of needed clock cycles after the end of the second transform computation, to re-initialize with the computations of the next macroblock.

Finally, the duration of the whole process when operated in continuous mode is shown in column G. These values were obtained by adding columns B, C, D and F, since column A represents the cycles needed to initialize the processor and column E represents the cycles needed to store the coefficients present in the array at the end of all computations.

On other hand, the number of clock cycles that are needed to compute an isolated macroblock is obtained by summing columns A, B, C, D and E. This number represents the time that is spent from the moment that the RST signal is released to low and the DFL bit is activated, until the all the processed data is completely stored in the output memory.

For the chroma modes, the presented values represent the time that is needed to process four macroblocks. As it was explained in section 4.2 when the IP Core is operating in the chroma mode the memories can store four macroblocks at time, and the IP Core processes them, by always assuming that the four macroblocks are in the input memory. If the chroma data of only
### Table 4.3: Process characterization for an 8 x 8 PE Array configuration.

<table>
<thead>
<tr>
<th>Mode</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G = B+C+D+F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward and Inverse Default Luma modes</td>
<td>3</td>
<td>72</td>
<td>-</td>
<td>-</td>
<td>14</td>
<td>-</td>
<td>72</td>
</tr>
<tr>
<td>Forward Intra 16x16 Luma mode</td>
<td>3</td>
<td>72</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>92</td>
</tr>
<tr>
<td>Inverse Intra 16x16 Luma mode</td>
<td>3</td>
<td>9</td>
<td>2</td>
<td>72</td>
<td>14</td>
<td>3</td>
<td>86</td>
</tr>
<tr>
<td>4x4 Forward Chroma mode</td>
<td>3</td>
<td>72</td>
<td>9</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>93</td>
</tr>
<tr>
<td>4x4 Inverse Chroma mode</td>
<td>3</td>
<td>9</td>
<td>3</td>
<td>72</td>
<td>10</td>
<td>3</td>
<td>87</td>
</tr>
<tr>
<td>8x8 Forward and Inverse Luma modes</td>
<td>3</td>
<td>72</td>
<td>-</td>
<td>-</td>
<td>17</td>
<td>-</td>
<td>68</td>
</tr>
</tbody>
</table>

### Table 4.4: Process characterization for a 4 x 8 PE Array configuration.

<table>
<thead>
<tr>
<th>Mode</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G = B+C+D+F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward and Inverse Default Luma modes</td>
<td>3</td>
<td>72</td>
<td>-</td>
<td>-</td>
<td>14</td>
<td>-</td>
<td>72</td>
</tr>
<tr>
<td>Forward Intra 16x16 Luma mode</td>
<td>3</td>
<td>72</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>92</td>
</tr>
<tr>
<td>Inverse Intra 16x16 Luma mode</td>
<td>3</td>
<td>9</td>
<td>2</td>
<td>72</td>
<td>14</td>
<td>3</td>
<td>86</td>
</tr>
<tr>
<td>4x4 Forward Chroma mode</td>
<td>3</td>
<td>72</td>
<td>9</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>93</td>
</tr>
<tr>
<td>4x4 Inverse Chroma mode</td>
<td>3</td>
<td>9</td>
<td>3</td>
<td>72</td>
<td>10</td>
<td>3</td>
<td>87</td>
</tr>
<tr>
<td>8x8 Forward and Inverse Luma modes</td>
<td>3</td>
<td>132</td>
<td>-</td>
<td>-</td>
<td>13</td>
<td>-</td>
<td>132</td>
</tr>
</tbody>
</table>

### Table 4.5: Process characterization for a 2 x 8 PE Array configuration.

<table>
<thead>
<tr>
<th>Mode</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G = B+C+D+F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward and Inverse Default Luma modes</td>
<td>3</td>
<td>136</td>
<td>-</td>
<td>-</td>
<td>10</td>
<td>-</td>
<td>136</td>
</tr>
<tr>
<td>Forward Intra 16x16 Luma mode</td>
<td>3</td>
<td>136</td>
<td>3</td>
<td>18</td>
<td>6</td>
<td>3</td>
<td>160</td>
</tr>
<tr>
<td>Inverse Intra 16x16 Luma mode</td>
<td>3</td>
<td>17</td>
<td>2</td>
<td>136</td>
<td>8</td>
<td>3</td>
<td>158</td>
</tr>
<tr>
<td>4x4 Forward Chroma mode</td>
<td>3</td>
<td>136</td>
<td>4</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td>152</td>
</tr>
<tr>
<td>4x4 Inverse Chroma mode</td>
<td>3</td>
<td>9</td>
<td>2</td>
<td>136</td>
<td>11</td>
<td>3</td>
<td>150</td>
</tr>
<tr>
<td>8x8 Forward and Inverse Luma modes</td>
<td>3</td>
<td>256</td>
<td>-</td>
<td>-</td>
<td>10</td>
<td>-</td>
<td>256</td>
</tr>
</tbody>
</table>
one macroblock is placed in the input memory, the time used by the IP Core is the same.

Like it can be observed, the only difference from table 4.3 to table 4.4 is in the 8 x 8 mode. This behavior is expected since the majority of the modes uses only 4 x 4 transforms or 2 x 2 transforms they are not affected by the reduction of the number of lines in the PE Array from 8 to 4. As referred before, when the PE array is instantiated with a smaller number of lines (see fig. 3.3 and fig. 3.4) the array requires more clock cycles to process each 1-D transform.

4.7 Example configuration

As it was said in the previous section, the proposed IP Core is able to run, in a fully autonomous way, all the operations to compute the direct and inverse DCT in the H.264/AVC standard.

In this section it is presented a detailed description of what happens in the IP Core from the moment that the input memory bank is loaded with the macroblock data to the moment when the output memory bank is filled with the computed coefficients, corresponding to a macroblock encoded using the Forward 16 x 16 Luma Intra mode.

1. After all the luma samples of the macroblock are loaded in the input memory bank and the DFL bit of the control register has been set to ‘1’, the IP Core starts loading the data of the first block to the firsts four positions of the Input buffer (column by column). At this time, the state machine is at SAC,C state;

2. While the first block is being processed inside the PE Array, with the PEs configured to compute the 4 x 4 forward integer DCT a second block is loaded to the last four positions of the Input buffer;

3. Two 2D transforms are therefore performed, one for each block, and the obtained transformed coefficients are stored in the output memory bank, in the same positions as they had occupied in the input memory bank;

4. The DC Coefficients of each block are also saved in the DC Buffer;

5. These steps are repeated 8 times, so that all the 16 blocks of the luma macroblock are processed; at the end, 16 DC coefficients are available in the DC Buffer;

6. As soon as the last DC coefficient is saved in the DC Buffer, the state machine moves to the SDC,C state; in this state, the coefficients temporarily saved in the DC Buffer are loaded to the Input Buffer;

7. The 4 x 4 block of DC coefficients is then processed, for which the PEs of the array are reprogrammed to compute the 4 x 4 Hadamard transform;
8. After the realization of the two 1D transforms, the values of the processed coefficients are saved in the last four positions of the output memory bank, and the whole process is completed.

4.8 Summary

An IP Core for the computation of the several transform paths defined in the H.264/AVC standard was presented in this chapter. Such processing structure is based on the systolic array processor that was proposed in Chapter 3 with some additional circuits to allow an automatic processing.

The use of dual-port double-page memories allows to have a continuous operation for more than one macroblock without stalls in the processing. A specially developed DC Buffer improves the speed of the processing, by allowing to temporarily store the DC coefficients inside the IP Core.

All the operations are controlled by a control unit, which allows the implementation of all the involved operations of the IP Core in parallel, without affecting the critical path of the circuit.
Experimental Results

Contents

5.1 Introduction .................................................. 64
5.2 Evaluation Platform and Functional Validation .......... 64
5.3 Implementation Results of the Processor Unit .......... 64
5.4 Implementation Results of the IP Core ................. 67
5.5 Performance Analyzes ...................................... 68
5.6 Summary ...................................................... 70
5. Experimental Results

5.1 Introduction

In this chapter, the synthesis and implementation results concerning the prototyping of the proposed architectures in a general purpose Field-Programmable Gate Array (FPGA) device are presented and discussed. Despite the considered target implementation technology, i.e., a FPGA device, all the proposed hardware units have been described using the VHSIC hardware description language (VHDL) considering standard and fully generic configurations, in order to also allow its implementation in Application-Specific Integrated Circuit (ASIC) devices. To achieve such goal, only some simple adaptations are required in the definition of the circuit’s memory units.

To properly analyze the usability of the architecture in different and diverse applications domains, all the units were implemented by considering two different synthesis strategies: i) concerning the optimization of the circuits’ hardware resources; and ii) addressing the circuit performance.

5.2 Evaluation Platform and Functional Validation

The proposed processor unit and corresponding IP Core were fully described using VHDL and synthesized for a general purpose FPGA device, i.e., a Virtex4 XC4VLX100 FPGA, by using the Xilinx ISE 13.2i toolchain. The adopted FPGA device was chosen to implement a proof of concept realization of the proposed system, due to the simplicity of the involved implementations. Furthermore, the Virtex4 XC4VLX100 FPGA has 49,152 slices and 4,320 Kb of RAM arranged in 240 blocks of 18Kb Block RAM (BRAM). These memory resources are particularly useful to implement both the data input and the data output memory banks of the devised system (see fig. 4.1).

To validate the functionality of the proposed hardware circuits, a VHDL test bench was implemented. Such test bench loads random test vectors from txt files into the input memories of the devised circuit, triggers the processing of the data contained in the two input memory pages and completes its operation by storing the output results in txt files.

All the process was validated by using the ModelSim SE 10.0b simulation tool and by carefully analyzing the most relevant signal waveforms in the ModelSim interface. In addition, the final results were also validated using a simple MatLab script, that compares the obtained simulation results with the expected test vectors. All the simulations were made after the Place&Route procedure, in order to assess the timing results.

5.3 Implementation Results of the Processor Unit

The original architecture was improved to allow the computation of larger transforms, and thus to support larger arrays of PEs. In addition, the original PE architecture was also re-designed to
5.3 Implementation Results of the Processor Unit

support the new transforms. Since this unit is also the one that influences the most the critical path of the whole circuit, a special attention was given to this modification procedure. In the following, it is presented the implementation results of such improved PE design.

5.3.1 Original architecture of the PE versus the new one

Table 5.1 presents the implementation results for both the original and the proposed improved architecture of the PE. As it can be seen, the proposed architecture requires about the double of the hardware resources than the original PE architecture. Moreover, it imposes a reduction in the maximum clock frequency of about 25%. Such differences can be explained by the significant increase in the complexity of the PE control unit, together with the increase in the delay imposed by the new datapath (i.e., the addition of one adder and bigger multiplexers, to support the whole set of coefficients). While the original architecture only uses two 2-to-1 multiplexers for the three different coefficients, this improved architecture requires two 4-to-1 multiplexers and one adder to be able to compute nine different coefficients. Nevertheless, in spite of these losses in processing efficiency, the proposed architecture is able to compute all the transforms required by the H.264/AVC High Profiles, and is still able to process High Definition (HD) video up to the 1080p format (1920 x 1080 @ 30fps).

Table 5.1: Comparison of the implementation results (time and area) of the previous architecture and the new one.

<table>
<thead>
<tr>
<th>Synthesis strategic</th>
<th>Optimized for speed</th>
<th>Optimized for area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Original</td>
<td>Proposed</td>
</tr>
<tr>
<td>Nr. Slices</td>
<td>78</td>
<td>203</td>
</tr>
<tr>
<td>Max. Delay</td>
<td>6.373ns</td>
<td>7.994ns</td>
</tr>
<tr>
<td>Max. Freq.</td>
<td>156 MHz</td>
<td>125 MHz</td>
</tr>
</tbody>
</table>

5.3.2 Scalability of the proposed architecture

One of the main features of the proposed architecture concerns to scalability, which allows its adaptation for specific and hardware restricted applications. Tables 5.2, Table 5.3 and Table 5.4 depict the implementation results corresponding to all the units of the devised processor architecture, for array configurations with 8 x 8, 4 x 8 and 2 x 8 PE, respectively. These results concern synthesis strategies targeting optimizations for both performance and hardware cost.

As it can be seen, the use of different array dimensions does not significantly changes the maximum clock frequency of the architecture. However, by making use of the architecture's scalability property, it is possible to reduce the amount of required hardware resources to less than half of the 8 x 8 configuration (using the 2 x 8 array configuration). Such reduction, as it was previously discussed, does not diminishes the functionalities offered by the architecture. In fact, it only implies an increase in the amount of time to do all the required computations.
## 5. Experimental Results

### Table 5.2: Implementation results for the HUTU processor with 8 x 8 PE Array.

<table>
<thead>
<tr>
<th>Module</th>
<th>Optimized for Area</th>
<th>Optimized for Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Slices</td>
<td>Max. F.(MHz)</td>
</tr>
<tr>
<td>Processor Unit 8x8</td>
<td>8853</td>
<td>96</td>
</tr>
<tr>
<td>PE Array 8 x 8</td>
<td>6918</td>
<td>97</td>
</tr>
<tr>
<td>Processor Element</td>
<td>176</td>
<td>117</td>
</tr>
<tr>
<td>PE DataPath</td>
<td>95</td>
<td>155</td>
</tr>
<tr>
<td>PE Control Unit</td>
<td>33</td>
<td>263</td>
</tr>
<tr>
<td>Input Buffer</td>
<td>1248</td>
<td>123</td>
</tr>
<tr>
<td>Transposition Switch</td>
<td>1681</td>
<td>127</td>
</tr>
<tr>
<td>Control Unit</td>
<td>72</td>
<td>195</td>
</tr>
</tbody>
</table>

### Table 5.3: Implementation results for the HUTU processor with 4 x 8 PE Array.

<table>
<thead>
<tr>
<th>Module</th>
<th>Optimized for Area</th>
<th>Optimized for Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Slices</td>
<td>Max. F.(MHz)</td>
</tr>
<tr>
<td>Processor Unit 4x8</td>
<td>5609</td>
<td>99</td>
</tr>
<tr>
<td>PE Array 4 x 8</td>
<td>3555</td>
<td>104</td>
</tr>
<tr>
<td>Processor Element</td>
<td>176</td>
<td>117</td>
</tr>
<tr>
<td>PE DataPath</td>
<td>95</td>
<td>155</td>
</tr>
<tr>
<td>PE Control Unit</td>
<td>33</td>
<td>263</td>
</tr>
<tr>
<td>Input Buffer</td>
<td>1248</td>
<td>123</td>
</tr>
<tr>
<td>Transposition Switch</td>
<td>1681</td>
<td>127</td>
</tr>
<tr>
<td>Control Unit</td>
<td>72</td>
<td>195</td>
</tr>
</tbody>
</table>

### Table 5.4: Implementation results for the HUTU processor with a 2 x 8 PE Array.

<table>
<thead>
<tr>
<th>Module</th>
<th>Optimized for Area</th>
<th>Optimized for Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Slices</td>
<td>Max. F.(MHz)</td>
</tr>
<tr>
<td>Processor Unit 2x8</td>
<td>3933</td>
<td>102</td>
</tr>
<tr>
<td>PE Array 2 x 8</td>
<td>1856</td>
<td>104</td>
</tr>
<tr>
<td>Processor Element</td>
<td>176</td>
<td>117</td>
</tr>
<tr>
<td>PE DataPath</td>
<td>95</td>
<td>155</td>
</tr>
<tr>
<td>PE Control Unit</td>
<td>33</td>
<td>263</td>
</tr>
<tr>
<td>Input Buffer</td>
<td>1248</td>
<td>123</td>
</tr>
<tr>
<td>Transposition Switch</td>
<td>1681</td>
<td>127</td>
</tr>
<tr>
<td>Control Unit</td>
<td>72</td>
<td>195</td>
</tr>
</tbody>
</table>
The scalability characteristic offered by the devised PE array permits the adaptation of the proposed architecture to the most suitable configuration for possibly different applications, i.e., the smaller configurations are more appropriate for applications where the hardware and power consumption are limited and the rate of processing data can be slower, while the greater PE configurations can be used in more critical applications, where the maximum performance is needed and the hardware costs and the power consumption are not a limitation.

It can be observed in all the cases (Table 5.2, 5.3 and 5.4) that the conducted synthesis optimized for speed achieved a faster operating frequency for the whole architecture than for only the PE array. This situation can be explained with the heuristics algorithms of the synthesis and mapping tool, which occasionally can provide slightly better results with the addition of more hardware.

5.3.3 Discussion

As it was expected, the critical path of the developed processor is mainly located in the PE unit. This unit is composed by the PE Datapath and the PE Control units, which are connected in series. The PE Control unit has a greater influence in the overall critical path, since each PE is required to decode its own coordinates for a larger set of coefficients. This imposes a significant reduction in the maximum allowed clock frequency. However, such performance reduction does not significantly constraint its application scope. In fact, as it will be described in the next section, the proposed architecture is still able to process the High Definition video content using the H.264/AVC Standard.

5.4 Implementation Results of the IP Core

Tables 5.5 to 5.7 present the implementation results of the proposed IP Core, for which three different array configurations of the processor unit were considered.

As it can be seen, the addition of the required control logic and extra memories to accommodate the DC coefficients did not introduce a significant increase in the circuit’s complexity. In fact, for some particular cases (eg. 8 x 8 array configuration, optimized for spread) the synthesis tool was even able to achieve more efficient implementations (in what concerns of the hardware costs) for the whole IP Core than for a single processor unit. The maximum clock frequency for the IP Core implemented with the 4 x 8 and 2 x 8 PE array configurations is more affected by the addition of the extra circuitry. However, as it can be observed below, the utilization of the IP Core are not significantly affected. Nonetheless, it is necessary to take into account that the results presented in Tables 5.5 to 5.7 do not include the 32 BRAM modules of the FPGA device that were used to implement the 16 input and the 16 output memory banks.

From the presented results, it can be concluded that the extra control logic which is necessary
5. Experimental Results

Table 5.5: Implementation results for the proposed IPCore with a 8 x 8 PE Array.

<table>
<thead>
<tr>
<th>Module</th>
<th>Optimized for Area</th>
<th>Optimized for Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Slices</td>
<td>Max. F.(MHz)</td>
</tr>
<tr>
<td>IP Core (8 x 8 PE Array)</td>
<td>9538</td>
<td>95</td>
</tr>
<tr>
<td>Processor Unit 8x8</td>
<td>8853</td>
<td>96</td>
</tr>
<tr>
<td>DC Buffer</td>
<td>244</td>
<td>389</td>
</tr>
<tr>
<td>Control Unit</td>
<td>165</td>
<td>175</td>
</tr>
</tbody>
</table>

Table 5.6: Implementation results for the proposed IPCore with a 4 x 8 PE Array.

<table>
<thead>
<tr>
<th>Module</th>
<th>Optimized for Area</th>
<th>Optimized for Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Slices</td>
<td>Max. F.(MHz)</td>
</tr>
<tr>
<td>IP Core (4 x 8 PE Array)</td>
<td>6294</td>
<td>87</td>
</tr>
<tr>
<td>Processor Unit 4x8</td>
<td>5609</td>
<td>99</td>
</tr>
<tr>
<td>DC Buffer</td>
<td>244</td>
<td>389</td>
</tr>
<tr>
<td>Control Unit</td>
<td>165</td>
<td>175</td>
</tr>
</tbody>
</table>

Table 5.7: Implementation results for the proposed IPCore with a 2 x 8 PE Array.

<table>
<thead>
<tr>
<th>Module</th>
<th>Optimized for Area</th>
<th>Optimized for Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Slices</td>
<td>Max. F.(MHz)</td>
</tr>
<tr>
<td>IP Core (2 x 8 PE Array)</td>
<td>4643</td>
<td>101</td>
</tr>
<tr>
<td>Processor Unit 2x8</td>
<td>3933</td>
<td>102</td>
</tr>
<tr>
<td>DC Buffer</td>
<td>244</td>
<td>389</td>
</tr>
<tr>
<td>Control Unit</td>
<td>165</td>
<td>175</td>
</tr>
</tbody>
</table>

to implement the scalability properties of the conceived architecture resulted in a residual added hardware cost. However, such circuit can now be used in a larger variety of applications, such as mobile devices, media players or even as a co-processor of general purpose processors.

5.5 Performance Analyzes

To better understand the practical usability of the proposed architecture, the performance of the implemented circuits were analyzed in terms of the maximum frame size supported for video sequences with a frame rate of 30 [fps]

5.5.1 IP Core with an 8 x 8 PEs array

The encoding mode that needs more clock cycles to compute one entire macroblock in the 8 x 8 PE Array configuration is the Intra 16 x 16 Luma mode (worst case), in which the IP Core requires 97 clock cycles to process an entire luma macroblock. In addition, the crominance samples must also be processed, which in the worst case involves 95 clock cycles to process all the samples corresponding to four macroblocks. Since each macroblock has two crominance blocks, it will be assumed that for each macroblock the IP Core uses 97 clock cycles to process the luminance samples and 48 clock cycles to process the crominance samples. Consequently, by considering
5.5 Performance Analyzes

a clock frequency of 99 MHz, it is possible to process 682758 macroblocks per second.

Assuming a 30 \text{fps} video sequence, the proposed architecture is able to process frames with 22758 macroblocks on each second. This corresponds to the 2560 x 2048 frame size, i.e., the QSXGA video format. Hence, it can be concluded that the proposed IP Core can encode/decode video sequences with a resolution larger than 1080p HD @30fps, by using a 8 x 8 \text{PE} Array.

5.5.2 IP Core with an 4 x 8 PEs Array

Unlike for the 8 x 8 \text{PEs} configuration, the worst case scenario for the 4 x 8 \text{PEs} configuration consists in the processing of the luma macroblocks in the Intra 8 x 8 mode. This is owed to the fact that the Intra 8 x 8 luma mode is the only mode that uses more than 4 lines of the array, and it is therefore affected by a reduction in the amount of array lines. Hence, with this \text{PE} Array configuration, the Intra 8 x 8 mode needs 145 clock cycles to compute one macroblock of luma samples. With the addition of the 48 clock cycles to compute the chroma samples, a total of 193 clock cycles are therefore required to process a entire macroblock.

One other hand, by considering this smaller \text{PE} Array configuration, the IP Core can operate at 101 MHz. In these conditions, the IP Core is able to compute 512953 macroblocks per second, or 17098 macroblocks per frame, by considering a frame rate of 30 \text{fps}. This only allows to support the encoding/decoding of video sequences with a resolution of 2048 x 1080 pixels. Again, the 1080p resolution is guaranteed.

Finally, it should be noted that the maximum resolution obtained for this scenario is close to 2560 x 2048 pixels. Since it is highly improbable that every frame in a video sequence is encoded using the Intra 8 x 8 mode, it is reasonable to assume that even with the 4 x 8 \text{PE} Array configuration, the proposed IP Core can still process video at 2560 x 2048@30fps.

5.5.3 IP Core with 2 x 8 PE Array

When using the 2 x 8 \text{PE} Array configuration, all the encoding modes are affected by a significant increase of the clock cycles required to process one macroblock. As it was expected, the Intra 8 x 8 luma mode is the one that requires more clock cycles to process one macroblock. In this mode, 266 clock cycles are used to process the luma block and another 79 (158/2) clock cycles are used to process the chroma blocks. With a total of 345 clock cycles and a maximum clock frequency of 101 MHz, the proposed architecture is able to process 292753 macroblocks per second, or 9758 macroblocks per frame, by considering a frame rate of 30 \text{fps}. In such conditions, the devised system can support the 2048 x 1080@30fps video format. Consequently, even when using the smaller \text{PE} Array, the proposed IP Core is able to process 1080p HD video in real time.
5. Experimental Results

5.5.4 Discussion

Table 5.8 summarizes the conducted performance analyses. The worst cases have been obtained from tables 4.3, 4.4, and 4.5 by assuming that a change of the operation mode occurs for every macroblock and the advantage of operating in a continuous and steady mode is lost. Moreover, as it was already discussed, the amount of clock cycles required to process a macroblock of crominances was divided by 2, since the values presented in those tables correspond to the time required for the computation of 4 macroblocks (and for each luminance macroblock there are only 2 blocks of crominances).

Table 5.8: Maximum processing rates offered by the proposed IP Core.

<table>
<thead>
<tr>
<th>PE Array Configuration</th>
<th>Worst case for Luminance (CC)</th>
<th>Worst case for Crominance (CC)</th>
<th>Total (CC)</th>
<th>Max. Freq. (MHZ)</th>
<th>MB/frame @30fps</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 x 8</td>
<td>97</td>
<td>48</td>
<td>145</td>
<td>99</td>
<td>22758</td>
</tr>
<tr>
<td>4 x 8</td>
<td>145</td>
<td>48</td>
<td>193</td>
<td>101</td>
<td>17098</td>
</tr>
<tr>
<td>2 x 8</td>
<td>266</td>
<td>79</td>
<td>345</td>
<td>101</td>
<td>9758</td>
</tr>
</tbody>
</table>

The results presented in Table 5.8 were also used to obtain the most suitable array configuration for the processing of the most common video formats. Such analysis is presented in Table 5.9 and evidences that even using a 2 x 8 PE array it is possible to process video sequences in the 1080p format at a frame rate of 30 fps.

Moreover, it can also be concluded that for applications involving smaller frame rates, the proposed IP Core can support larger video resolutions.

Table 5.9: Minimum PE Array configuration to support some of the most commons video resolutions.

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Name</th>
<th>MB per frame (16 x 16 pixels)</th>
<th>Min PE Array Conf. @30fps</th>
</tr>
</thead>
<tbody>
<tr>
<td>640 x 480</td>
<td>VGA</td>
<td>1200</td>
<td>2 x 8</td>
</tr>
<tr>
<td>800 x 600</td>
<td>SVGA</td>
<td>1900</td>
<td>2 x 8</td>
</tr>
<tr>
<td>1024 x 768</td>
<td>XGA</td>
<td>3072</td>
<td>2 x 8</td>
</tr>
<tr>
<td>1280 x 720</td>
<td>720p</td>
<td>3600</td>
<td>2 x 8</td>
</tr>
<tr>
<td>1280 x 800</td>
<td>WXGA</td>
<td>4000</td>
<td>2 x 8</td>
</tr>
<tr>
<td>1280 x 1024</td>
<td>SXGA</td>
<td>5120</td>
<td>2 x 8</td>
</tr>
<tr>
<td>1600 x 1200</td>
<td>UXGA</td>
<td>7500</td>
<td>2 x 8</td>
</tr>
<tr>
<td>1920 x 1080</td>
<td>1080p</td>
<td>8160</td>
<td>2 x 8</td>
</tr>
<tr>
<td>2048 x 1080</td>
<td>2K</td>
<td>8704</td>
<td>2 x 8</td>
</tr>
<tr>
<td>2560 x 2048</td>
<td>QSXGA</td>
<td>20480</td>
<td>8 x 8</td>
</tr>
</tbody>
</table>

5.6 Summary

In this chapter it was presented the experimental results concerning the implementation of the proposed architecture in a general purpose FPGA device. Such results, which resulted from
Post-Place&Route simulations, proved that the proposed architecture is capable of processing high-definition video sequences, even when configured for smaller PE arrays. In fact, the obtained implementation results allow to conclude that the 2 x 8 PE Array configuration is capable of supporting video sequences up to the 2K @30fps format, while the 8 x 8 PE Array configuration is able to comply with more than 2 times larger resolutions (i.e., QSXGA format).

Despite their different application domains, the implemented architecture was compared with the original one (see Table 5.10), when they were both synthesized for a Virtex-4 FPGA device. The differences in terms of hardware costs can be easily explained by the increased complexity of the newer PE structure and the increased size of the new PE array, which contains 4 times more PEs.

Table 5.10: Original architecture versus proposed one.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Array Dimensions</th>
<th>Slices</th>
<th>Max. Freq. (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>4 x 4</td>
<td>700</td>
<td>265</td>
</tr>
<tr>
<td>Proposed</td>
<td>8 x 8</td>
<td>10517</td>
<td>102</td>
</tr>
</tbody>
</table>
5. Experimental Results
6 Conclusions

Contents

6.1 Future work .................................................. 75
6. Conclusions

The work that was conducted in the scope of this thesis consisted in improving one scalable and already developed architecture to compute some of the transforms of the H.264/AVC video standard. This improvements are mainly focused in turning the original architecture able to compute the whole set of transforms of the H.264/AVC standard, i.e., make it able to compute the 8 x 8 integer DCT.

Several modifications were made, mainly in the PE structure, which is now required to compute a greater amount of coefficients. This new set of coefficients has a significant impact in the complexity of the PE structure and, consequently, in the critical path of the whole architecture, since the PE is the main structural unit of the systolic array.

However, the introduced complexity in the PE structure did not compromise the ability to process High Definition video sequences up to the 1080p@30fps format. In fact, it was demonstrated that the final architecture is able to process larger formats.

On other hand, all the processing units of the original architecture had to be modified, in order to support 8 x 8 transforms. The dimensions of the PE array were extended to 8 columns and, consequently, new scalability options were introduced. As a result, the proposed architecture is able to compute, in parallel, two 4 x 4 transforms, which reduces the impact of the required increase in the complexity.

The Input Buffer unit was also extended. It is now able to store the 64 coefficients of an 8 x 8 block and some modifications were also made to correctly process all the possible transforms, independently of the PE array configuration.

Similarly, the Transposition Switch also had to be adapted. The introduced adaptations increased its complexity and penalized the critical path. As a consequence, it was decided to implement a set of new registers at the output of this unit. These registers implied an extra delay of one clock cycle, between the first and the second 1-D transform. However, this extra delay proved to have a small impact in the total processing time of each macroblock and it does not compromise the performance of the architecture.

Moreover, a new IP Core was designed. By incorporating the improved processing architecture, it was made possible an easy integration with the outer GPP thanks to an efficient interface between the GPP and the proposed dedicated processor. The design of the proposed IP Core did not introduce any relevant increase in the complexity nor significantly affected the maximum operating frequency of the processor, due to the simplicity of all the control logic.

In addition to the interface for the data and control communications with the GPP, the proposed IP Core also temporarily saves the computed DC coefficients of each block, to reprocess them. This new feature turns the proposed IP Core able to autonomously process whole video macroblocks. The use of dual-page memory banks also turns it able to continuously process several macroblocks, without additional delays, i.e., while the proposed IP Core is processing the macroblock data of one memory page, the other page can be loaded by the GPP with one different
macblock to be subsequently processed.

The VHDL description and the synthesis in a FPGA implementation demonstrated that all the presented work can be easily implemented and reach the expected performance. The analysis of the results proved the capability to process video sequences up to a 2560x2048@30fps resolution.

After all the conducted adaptations, the architecture still retains the same scalability of the original architecture and meets the requirements to process high definition video sequences. Moreover, the usability of the architecture was significantly improved with the ability to compute 8 x 8 matrix kernels. On the other hand, the implementation of the proposed IP Core proved to be a valuable help to simplify the usage of the improved architecture.

In table 6.1 can be seen the comparison between the proposed architecture and some of the architectures presented in Section 2, including the original architecture.

<table>
<thead>
<tr>
<th>Design</th>
<th>Transform Functions</th>
<th>Throughput</th>
<th>Max Freq.</th>
<th>Hardware util.</th>
</tr>
</thead>
<tbody>
<tr>
<td>[20]</td>
<td>$DCT_{4\times4}$, $H_{4\times4}$</td>
<td>4</td>
<td>110 MHz</td>
<td>2433 LUTS</td>
</tr>
<tr>
<td>[6]</td>
<td>$DCT_{4\times4}$, $H_{4\times4}$, $H_{2\times2}$</td>
<td>8</td>
<td>167 MHz</td>
<td>2100 LUTS</td>
</tr>
<tr>
<td>[28]</td>
<td>$DCT_{4\times4}$, $DCT_{8\times8}$</td>
<td>n.a.</td>
<td>100 MHz</td>
<td>5258 LUTS</td>
</tr>
<tr>
<td>[8]</td>
<td>$DCT_{4\times4}$, $H_{4\times4}$, $H_{2\times2}$</td>
<td>4</td>
<td>265 MHz</td>
<td>1800 LUTS</td>
</tr>
<tr>
<td>Prop.</td>
<td>$DCT_{4\times4}$, $DCT_{8\times8}$, $H_{4\times4}$, $H_{2\times2}$</td>
<td>8</td>
<td>102 MHz</td>
<td>10517 LUTS</td>
</tr>
</tbody>
</table>

### 6.1 Future work

Since the majority of the image and video standards are based on 8 x 8 matrix kernels, the adaptations of the original architecture to turn it able to compute 8 x 8 matrix kernels brought the opportunity to use the proposed architecture to compute different integer or even non-integer transforms. With some modifications in the PE structure to accommodate new coefficient values, the usage of the proposed architecture can be strongly improved.

Moreover, the described limitation of computing only two 2 x 2 transforms at a time can be resolved by adopting a more complex Input Buffer. Since not all the lines of the input data are used when computing transforms with less than 8 lines, it could be possible to load more than one input buffer element at a time. This modification can provide a good improvement in the performance of the whole architecture.

Finally, one interesting direction that deserves a further study is the possibility to design one different IP Core that incorporates more than one processor unit. Such parallelization of the processing unit will make it possible the processing of even greater video formats in real time, potentially approaching the processing rates required in cinema applications (60 fps).
6. Conclusions
Bibliography


A. Appendix A

A.1 IP Core Datasheet

A.1.1 Features

- Can compute the 2 x 2 Hadamard transform, the 4 x 4 Hadamard transform, the 4 x 4 integer Discrete Cosine Transform (DCT) forward and inverse transforms and the 8 x 8 integer DCT forward and inverse transforms.

- Can perform the Luma 16x16 (Intra and Default) transform, the Luma 8x8 transform and the Chroma 4:2:0 transform in an autonomous way.

- Have 2 memory pages for the input and the output to be able to work continuously without waiting times.

- Based on a 2D highly scalable and flexible systolic architecture.

- Capable of operating with clock frequencies up to 94 MHz in a Xilinx Virtex-4 Field-Programmable Gate Array (FPGA).

A.1.2 Description

H.264/AVC Unified forward/inverse Transform Unit (HUTU) IP Core is one dedicated processor for accelerating the computation of the transform functions of digital video encoding and decoding. It is based on a systolic architecture that can be configured to compute all of H.264/AVC transforms.

To communicate with this IP Core it is used one 16 bit register for control signals, 2 memories for input data and 2 memories for output data.

A.1.3 Interface signals

All control signals are available in one 16-bit register, as it can be seen in Table A.1. Besides this 16-bit register that can be mapped in the General Purpose Processor (GPP) memory, it is needed to map the input and the output memory pages on the GPP memory. One using example of the interface signals can be observed in Figure A.1.

A.1.3.A Inputs

- Mode - Indicates the mode of operation (details at A.1.4).

- DFL0 - Data Fully Loaded on page 0 of the input memory - Indicates that data is fully loaded and ready to be computed on memory page 0

- DFL1 - Data Fully Loaded on page 1 of the input memory - Indicates that data is fully loaded and ready to be computed on memory page 1
Table A.1: Inputs and Outputs

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{0\to3}$</td>
<td>Mode</td>
</tr>
<tr>
<td>$C_4$</td>
<td>DFL0</td>
</tr>
<tr>
<td>$C_5$</td>
<td>DFL1</td>
</tr>
<tr>
<td>$C_6$</td>
<td>RDFC0</td>
</tr>
<tr>
<td>$C_7$</td>
<td>RDFC1</td>
</tr>
<tr>
<td>$C_8$</td>
<td>DFL0S</td>
</tr>
<tr>
<td>$C_9$</td>
<td>DFL1S</td>
</tr>
<tr>
<td>$C_{10}$</td>
<td>DFC0</td>
</tr>
<tr>
<td>$C_{11}$</td>
<td>DFC1</td>
</tr>
<tr>
<td>$C_{14}$</td>
<td>EN</td>
</tr>
<tr>
<td>$C_{15}$</td>
<td>RST</td>
</tr>
</tbody>
</table>

Figure A.1: Example of the DCT IP Core usage.
A. Appendix A

RDFC0 - Reset Data Fully Computed on page 0 of the output memory - After reading all data of output memory page 0 this bit is used to reset the status bit DFC0

RDFC1 - Reset Data Fully Computed on page 1 of the output memory - After reading all data of output memory page 1 this bit is used to reset the status bit DFC1

EN - Enable operation of the IP Core

RST - Synchronous Reset of the IP Core

A.1.3.B Outputs

DFL0S - Data Fully Loaded on page 0 of the input memory Status - Assumes the value ‘0’ when the page 0 of the input memory can be written

DFL1S - Data Fully Loaded on page 1 of the input memory Status - Assumes the value ‘0’ when the page 1 of the input memory can be written

DFC0 - Data Fully Computed on Output memory 0 - When data is fully computed and ready to be read on Output memory 0 IP Core will set this bit at ‘1’

DFC1 - Data Fully Computed on Output memory 1 - When data is fully computed and ready to be read on Output memory 1 IP Core will set this bit at ‘1’

A.1.4 Modes of operation

The HUTU Core supports 8 modes of operation, encoded by 4 bits where the first bit is reserved for future implementations. Table A.2 describe the possible modes of operation.

<table>
<thead>
<tr>
<th>Code</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Forward Luma Default transform</td>
</tr>
<tr>
<td>0001</td>
<td>Forward Luma Intra 16x16 transform</td>
</tr>
<tr>
<td>0010</td>
<td>Forward Chroma transform</td>
</tr>
<tr>
<td>0011</td>
<td>Forward Luma Intra 8x8 transform</td>
</tr>
<tr>
<td>0100</td>
<td>Inverse Luma Default transform</td>
</tr>
<tr>
<td>0101</td>
<td>Inverse Luma Intra 16x16 transform</td>
</tr>
<tr>
<td>0110</td>
<td>Inverse Chroma transform</td>
</tr>
<tr>
<td>0111</td>
<td>Inverse Luma Intra 8x8 transform</td>
</tr>
</tbody>
</table>

A.1.5 Memory Maps

The capacity of the input and the output memory pages is 576 bytes, organized in 8 banks (one for each column) of 36 16 bit words (little endian). The map of these memory pages for all of modes can be seen below.
A.1.5.A  Default and 16 x 16 Intra Luma Macroblocks

In Figure A.3(a) it is illustrated the representation of the 16 4 x 4 blocks of pixels (see also Figure A.2(a)) that compose each 16x16 luma macroblock. Since the memory pages were organized in 8 sub-banks (in order to simplify the storage of the 8 x 8 blocks), it means that two lines of a 4 x 4 block can be stored in the same memory positions of one line of a 8 x 8 block. Moreover, to respect the pixels order in the whole macroblock, the second line of the first block it is stored after the first line of the fourth block.

In Figure A.4(a) it is represented the memory map for the default organization of the luma macroblocks. Figure A.4(b) represents the memory map for the Intra 16x16 luma macroblocks. In this case, the last four positions are occupied with the 4 x 4 DC coefficients.

At the input memories the data must be transposed, so in the first positions of the input memory should be present the first column of the first block as represented in Figure A.5(a). In the output memory the IP Core will save the data with the correct order of samples without transposition as represented in Figure A.6(a).

A.1.5.B  Intra 8 x 8 Luma Macroblocks

For the 8 x 8 luma macroblocks, represented in Figure A.3(b), the data of each block (Figure A.2(b)) is saved alternately to respect the order of pixels in macroblock like in the 16 x 16 modes. In this way all of the samples or quantized coefficients are placed in memories like it can be seen in Figure A.4(c). The last four positions are reserved for DC Coefficients used in others modes.

Again, at the input memory the data should be transposed, in the Figure A.5(b) it can be seen the positions of the firsts two blocks in the input memory. The output memory will have the blocks disposed line by line like in the Figure A.6(b).

A.1.5.C  Chroma Macroblocks

A Chroma macroblock in 4:2:0 format have four 4 x 4 blocks (Figure A.2(a)) of samples. As it can be seen in Figure A.3(c) the memory banks of the IP Core can therefore store four macroblocks at time. Moreover, for each 8 x 8 macroblock there is one 2 x 2 DC coefficient block (Figure A.2(c)) that is saved in the 4 last memory positions in the order represented in Figure A.4(d).

Since they are different macroblocks and each one has only 8 columns the columns or lines are inserted in memory in the original order.

At input memory all data should be transposed, including the DC coefficients when they are present, the example of one block stored in the input memory can be seen at Figure A.5(c). The output memory will have the configuration represented in Figure A.6(c).
A. Appendix A

A.1.6 Block Diagram

In Figure A.7 it can be seen one simple Block Diagram of the IP Core.

A.1.7 Step by step instructions

To use the HUTU IP Core first is needed to follow some simple steps:

1. Set the operating mode
2. Put all the data into the Input memory bank
3. Set the DFL bit of that memory page = ‘1’

While the core is computing the values it is possible to put more data in the memory page that is not in use. The operation mode can be changed only 1 clock cycle after the set of the DFL bit.

When data is fully computed, the DFC bit is set to ‘1’. At this time, the computed data will be present at the corresponding Output memory page.

A.1.8 Time needed to computations and data latency

The processing of the data is initiated as soon as the RST signal is released to low and one of the DFL bits is activated. After that, it is needed 3 clock cycles to initializations purposes, the
Figure A.4: Memory Maps with the relative addresses.

(a) Default Luma Macroblock  (b) Intra 16x16 Luma Macroblock  (c) 8 x 8 Intra Luma Macroblock  (d) Chroma macroblocks

Figure A.5: Detailed Input Memory Maps showing where each sample is placed.
Figure A.6: Detailed Output Memory Maps showing where each sample is placed.

Figure A.7: Block Diagram.
computation process depends of the mode and number of lines in array. The detailed description of each process can be seen below.

A.1.8. A 8 x 8 PE Array

A – Forward and Inverse Default Luma modes These modes require 8 x 4 + 8 x 5 clock cycles to load and compute the 16 blocks of a macroblock. After this 72 clock cycles it is possible to start computing the coefficients of other macroblock. Nonetheless it is needed to wait more 14 cycles for the last computations and writes in output memories.

After 72 + 14 clock cycles the DFC bit will be active and data can be read from the output memory.

In continuous mode are only needed 72 cycles for each macroblock.

B – Forward Intra 16x16 Luma mode This mode requires 8 x 4 + 8 x 5 clock cycles to load and compute all the AC coefficients, 8 cycles to wait for the last DC coefficient and initialize the DC transform and more 17 cycles to compute and store all the DC coefficients. Like in the Default mode, the last 5 cycles (after the first column of the array is free) can be used to compute next macroblock.

In continuous mode are needed 72 + 8 + 12 (=92) clock cycles.

C – Inverse Intra 16x16 Luma mode This mode requires 11 clock cycles to load the DC coefficients, compute them and initialize the AC coefficients computing. Then it is required 72 clock cycles to load and compute all the AC coefficients, and another 14 cycles for the last computations and writes to the output memory.

In continuous mode, 3 of these 14 cycles are used for DC initializations, so in continuous mode 11 + 72 + 3 (=86) clock cycles are required.

D – 4x4 Forward Chroma mode This mode requires 72 clock cycles to load and compute all the AC coefficients, 9 clock cycles to wait for the last DC coefficient and initialize the DC transform, 9 clock cycles to compute all the DC coefficients and 8 more to compute and store the last DC coefficients.

In continuous mode, 3 of these 8 cycles are used for AC initializations, so in continuous mode 72 + 9 + 9 + 3 (=93) clock cycles are required.

E – 4x4 Inverse Chroma mode This mode requires 12 clock cycles to load the DC coefficients, compute them and initialize the AC coefficients computing. Then it is needed 72 clock cycles for load and compute all the AC coefficients, and more 13 for the last computations and writes to the output memory.
In continuous mode, 3 of these 13 clock cycles are used for DC initializations, so in continues mode $12 + 72 + 3 (=87)$ clock cycles are required.

**F – 8x8 Forward and Inverse Luma modes** These modes require $4 \times 17$ clock cycles to load and compute all the AC coefficients and 17 more clock cycles for the last computations and writes to the output memory.

In continuous mode are only required $4 \times 17$ ($=68$) clock cycles for each macroblock.

**A.1.8.B 4 x 8 PE Array**

All transforms, except the 8x8 Integer DCT transform, use only 4 lines of processors elements in the Processor Element (PE) Array. So, all the times presented in the previous section are the same as for the 8 x 8 configuration except in the 8x8 Forward and Inverse Luma modes.

**A – 8x8 Forward and Inverse Luma modes** These modes require $4 \times 33$ cycles to load and compute all the AC coefficients and 13 more clock cycles for the last computations and writes to the output memory.

In continuous mode are only required $4 \times 33$ ($=132$) cycles for each macroblock.

**A.1.8.C 2 x 8 PE Array**

**A – Forward and Inverse Default Luma modes** These modes require $8 \times 8 + 8 \times 9$ clock cycles to load and compute all the AC coefficients and 10 more clock cycles for the last computations and writes to the output memory.

In continuous mode are only required 136 clock cycles for each macroblock.

**B – Forward Intra 16x16 Luma mode** This mode requires $8 \times 8 + 8 \times 9$ clock cycles to load and compute all the AC coefficients, 3 clock cycles to wait for the last DC coefficient and initialize the DC transform and more 24 to compute and store all the DC coefficients. Like in the Default mode, last 3 clock cycles (after first column of array is free) can be used to compute next macroblock.

In continuous mode are required $136 + 3 + 21 (=160)$ clock cycles.

**C – Inverse Intra 16x16 Luma mode** This mode requires 19 clock cycles to load the DC coefficients, compute them and initialize the AC coefficients computing. Then it is needed 136 clock cycles to load and compute all the AC coefficients, and more 11 for the last computations and writes to the output memory.

In continuous mode 3 of these 11 clock cycles are used for DC initializations, so in continues mode are required $19 + 136 + 3 (=158)$ clock cycles.
**D – 4x4 Forward Chroma mode**  This mode requires $8 \times 8 + 8 \times 9$ clock cycles to load and compute all the AC coefficients, 4 cycles to wait for the last DC coefficient and initialize the DC transform, 9 cycles to compute all the DC coefficients and 7 more to compute and store the last DC coefficients.

In continuous mode 3 of these 9 cycles are used for AC initializations, so in continues mode are required $136 + 4 + 8 + 3 (=152)$ clock cycles.

**E – 4x4 Inverse Chroma mode**  This mode requires 11 clock cycles to load the DC coefficients, compute them and initialize the AC coefficients computing. Then it is needed 136 clock cycles to load and compute all the AC coefficients, and more 11 for the last computations and writes to the output memory.

In continuous mode 3 of these 11 cycles are used for DC initializations, so in continues mode are required $11 + 136 + 3 (=150)$ clock cycles.

**F – 8x8 Forward and Inverse Luma modes**  These modes require $4 \times 64$ clock cycles to load and compute all the AC coefficients and 10 more clock cycles for the last computations and writes to the output memory.

In continuous mode are only needed 4x64 (=256) clock cycles for each macroblock.

**Summary**

In tables A.3, A.4 and A.5 it can be observed a summary of the above explained, where column A represents the initialization stage, column B represents the computation of the first transform (AC or DC, depending on if it is a encoding or decoding process, respectively), column C represents the transition from the first transform to the second transform, column D represents the second transforms computation (when it is supported), column E represents the final procedures to store the values remaining in the PE Array in the output memory and column F represent the number of needed clock cycles after the end of the second transform computation, to re-initialize with the computations of the next macroblock.

Finally, the duration of the whole process when operated in continuous mode is shown in column G. These values were obtained by adding columns B, C, D and F, since column A represents the cycles needed to initialize the processor and column E represents the cycles needed to store the coefficients present in the array at the end of all computations.

On other hand, the number of clock cycles that are needed to compute an isolated macroblock is obtained by summing columns A, B, C, D and E. This number represents the time that is spent from the moment that the RST signal is released to low and the DFL bit is activated, until the all the processed data is completely stored in the output memory.
### Table A.3: Process characterization for an 8 x 8 PE Array configuration.

<table>
<thead>
<tr>
<th>Mode</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G = B+C+D+F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward and Inverse Default Luma modes</td>
<td>3</td>
<td>72</td>
<td>-</td>
<td>-</td>
<td>14</td>
<td>-</td>
<td>72</td>
</tr>
<tr>
<td>Forward Intra 16x16 Luma mode</td>
<td>3</td>
<td>72</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>92</td>
</tr>
<tr>
<td>Inverse Intra 16x16 Luma mode</td>
<td>3</td>
<td>9</td>
<td>2</td>
<td>72</td>
<td>14</td>
<td>3</td>
<td>86</td>
</tr>
<tr>
<td>4x4 Forward Chroma mode</td>
<td>3</td>
<td>72</td>
<td>9</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>93</td>
</tr>
<tr>
<td>4x4 Inverse Chroma mode</td>
<td>3</td>
<td>9</td>
<td>3</td>
<td>72</td>
<td>10</td>
<td>3</td>
<td>87</td>
</tr>
<tr>
<td>8x8 Forward and Inverse Luma modes</td>
<td>3</td>
<td>72</td>
<td>-</td>
<td>-</td>
<td>17</td>
<td>-</td>
<td>68</td>
</tr>
</tbody>
</table>

### Table A.4: Process characterization for a 4 x 8 PE Array configuration.

<table>
<thead>
<tr>
<th>Mode</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G = B+C+D+F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward and Inverse Default Luma modes</td>
<td>3</td>
<td>72</td>
<td>-</td>
<td>-</td>
<td>14</td>
<td>-</td>
<td>72</td>
</tr>
<tr>
<td>Forward Intra 16x16 Luma mode</td>
<td>3</td>
<td>72</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>92</td>
</tr>
<tr>
<td>Inverse Intra 16x16 Luma mode</td>
<td>3</td>
<td>9</td>
<td>2</td>
<td>72</td>
<td>14</td>
<td>3</td>
<td>86</td>
</tr>
<tr>
<td>4x4 Forward Chroma mode</td>
<td>3</td>
<td>72</td>
<td>9</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>93</td>
</tr>
<tr>
<td>4x4 Inverse Chroma mode</td>
<td>3</td>
<td>9</td>
<td>3</td>
<td>72</td>
<td>10</td>
<td>3</td>
<td>87</td>
</tr>
<tr>
<td>8x8 Forward and Inverse Luma modes</td>
<td>3</td>
<td>132</td>
<td>-</td>
<td>-</td>
<td>13</td>
<td>-</td>
<td>132</td>
</tr>
</tbody>
</table>

### Table A.5: Process characterization for a 2 x 8 PE Array configuration.

<table>
<thead>
<tr>
<th>Mode</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G = B+C+D+F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward and Inverse Default Luma modes</td>
<td>3</td>
<td>136</td>
<td>-</td>
<td>-</td>
<td>10</td>
<td>-</td>
<td>136</td>
</tr>
<tr>
<td>Forward Intra 16x16 Luma mode</td>
<td>3</td>
<td>136</td>
<td>3</td>
<td>18</td>
<td>6</td>
<td>3</td>
<td>160</td>
</tr>
<tr>
<td>Inverse Intra 16x16 Luma mode</td>
<td>3</td>
<td>17</td>
<td>2</td>
<td>136</td>
<td>8</td>
<td>3</td>
<td>158</td>
</tr>
<tr>
<td>4x4 Forward Chroma mode</td>
<td>3</td>
<td>136</td>
<td>4</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td>152</td>
</tr>
<tr>
<td>4x4 Inverse Chroma mode</td>
<td>3</td>
<td>9</td>
<td>2</td>
<td>136</td>
<td>11</td>
<td>3</td>
<td>150</td>
</tr>
<tr>
<td>8x8 Forward and Inverse Luma modes</td>
<td>3</td>
<td>256</td>
<td>-</td>
<td>-</td>
<td>10</td>
<td>-</td>
<td>256</td>
</tr>
</tbody>
</table>
B. Appendix B

B.1 Two different architectures of the PE Control Unit

One of the most important adaptations to the original architecture was the modification of the Processor Element (PE) structure. Due to the significant increase in the complexity of this unit, it was divided into two distinct units, control unit and datapath unit, connected in series.

Searching for the best solution for a better performance, two alternative architectures for the PE control unit were designed.

The first one, from now on referred to as CU1, uses \( X_2 \) to choose between the other coordinates or their inverses, making use of the matrix symmetry and of the fact that \( t_{8i} \) is simply the \( t_{8f} \) transposed. The other one, from now on referred to as CU2, always uses all the coordinates to obtain the control signals. This means that the matrix symmetry helped only in the simplifications, but it is not used in the logical functions. In Table B.1 it is presented the results of the PE unit implemented with the two architectures, using different optimization strategies for the synthesis.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Number of Slices</th>
<th>Max. Delay</th>
<th>Max. Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CU1</td>
<td>209</td>
<td>8.454 ns</td>
<td>118 MHz</td>
</tr>
<tr>
<td>CU2</td>
<td>203</td>
<td>7.994 ns</td>
<td>125 MHz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Synthesis and Mapping optimized for AREA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CU1</td>
</tr>
<tr>
<td>CU2</td>
</tr>
</tbody>
</table>

Although, the results were not significantly different, so the choice of the adopted architecture was not very important. The architecture CU1 was the final choice to obtain all the other results for the conducted work.