OVERVIEW OF AN ACCELERATED H.264/AVC ENCODER AND DECODER ON HETEROGENEOUS PLATFORMS

HAMY Guillaume
Video streaming and video recording have become the most popular features of mobile devices. However, these features suffer many limitations, such as quality and network limitation. These limitations have become one of the main concerns of mobile industry. Many improvements have been made in this field and several solutions have been found in order to solve these limitations. The solutions can be classified into two categories: Hardware and Software solutions. One of the hardware solutions is to use heterogeneous platforms to increase the flexibility of complexity management. Usually the heterogeneous platform contains one or several powerful processors and at least one dedicated processor. Among the different dedicated processors, there is the Digital Signal Processor (DSP), which is dedicated to signal processing. As a software solution, the H.264/AVC standard is now the most commonly used standard. It provides a high video quality for a low bit rate. However, as a main drawback, the H.264/AVC is much more complex than other standards.

Implementing a H.264/AVC encoder or decoder on a heterogeneous platform might lead to a good compromise between video quality and energy consumption. The logical way to implement an encoder or a decoder on a heterogeneous platform is to implement it in parallel. However, such implementation leads to several challenges related to the software design and hardware architecture. The implementation must take into account the asymmetric architecture of the heterogeneous platform. It must also take advantage of the DSP assets and consider its limitations as a dedicated processor. This thesis will present different ways to conduct this type of implementation, along with several improvements.

**Keywords:** Advanced Video Coding, Data Level Parallelization, DSP, H.264, Heterogeneous Platform, OMAP, Task Level Parallelization
CONTENTS

Abstract i

Contents ii

List of Figures iv

Glossary vi

1 Introduction 1
   1.1 Thesis Purpose ........................................... 1
   1.2 Thesis structure ....................................... 2

2 H.264/AVC Encoding and Decoding Techniques 3
   2.1 Overview of the H.264/AVC Standard ................. 3
       2.1.1 Introduction .................................... 3
       2.1.2 Color Space Conversion .......................... 4
       2.1.3 Frame Partitioning .............................. 8
       2.1.4 Network Abstraction Layer (NAL) ............... 9
   2.2 H264/AVC encoding technique ........................ 10
       2.2.1 Transform ...................................... 10
       2.2.2 Quantization .................................. 11
       2.2.3 Intra Prediction ............................... 12
       2.2.4 Inter Prediction (Motion Compensation, Motion Estimation) 16
       2.2.5 Entropy Coding .................................. 19
   2.3 H264/AVC decoding technique ........................ 25
       2.3.1 Entropy Decoding ............................... 26
       2.3.2 Inverse Quantization and Inverse Transform .... 26
       2.3.3 In-loop Deblocking Filter ....................... 26
   2.4 Conclusion ............................................... 28

3 Heterogeneous Platforms 30
   3.1 Heterogeneous Platform ................................. 30
   3.2 DSP .................................................. 31
   3.3 H.264/AVC Complexity Analysis ....................... 33
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.1</td>
<td>H.264/AVC Mapping Solutions</td>
<td>33</td>
</tr>
<tr>
<td>3.3.2</td>
<td>H.264/AVC Improvements</td>
<td>46</td>
</tr>
<tr>
<td>4</td>
<td>OMAP 4460 and Syslink/RPMsg : A Proposed Approach</td>
<td>52</td>
</tr>
<tr>
<td>4.1</td>
<td>Proposed Heterogeneous Platform : OMAP 4460</td>
<td>52</td>
</tr>
<tr>
<td>4.1.1</td>
<td>OMAP 4460</td>
<td>52</td>
</tr>
<tr>
<td>4.1.2</td>
<td>TMS320C64x</td>
<td>56</td>
</tr>
<tr>
<td>4.2</td>
<td>Syslink and RPMsg</td>
<td>56</td>
</tr>
<tr>
<td>4.2.1</td>
<td>From DSP Gateway to DSP Bridge Project</td>
<td>56</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Syslink</td>
<td>59</td>
</tr>
<tr>
<td>4.2.3</td>
<td>RPMsg</td>
<td>63</td>
</tr>
<tr>
<td>5</td>
<td>Conclusion and Future Work</td>
<td>64</td>
</tr>
<tr>
<td>5.1</td>
<td>Conclusions</td>
<td>64</td>
</tr>
<tr>
<td>5.2</td>
<td>Future work</td>
<td>64</td>
</tr>
</tbody>
</table>

Bibliography 66
## LIST OF FIGURES

2.1 The different H.264/AVC profiles and their features .................... 5
2.2 RGB representation of an image (upper picture) and its Luma compon-
ent in from YUV representation ........................................ 7
2.3 The video sampling patterns [1] ........................................... 8
2.4 H.264/AVC encoder ..................................................... 10
2.5 ZigZag scan pattern with an 8 × 8 matrix ................................ 12
2.6 Template of a 4 × 4 macroblock for Intra Prediction .................... 15
2.7 The nine intra prediction modes [2] ...................................... 16
2.8 Macroblock partitionning for Inter Prediction ............................. 19
2.9 Multi frame prediction [3] ................................................. 20
2.11 Quarter-pixel interpolation [3] .......................................... 21
2.13 H.264/AVC decoder ..................................................... 25
2.14 On the left, image without deblocking filter, on the right image with a
deblocking filter [5] ....................................................... 27
2.15 Horizontal filtering of two adjacent macroblocks ......................... 28
2.16 Boundary strength (BS) computation flowchart [5] ....................... 29

3.1 Dual-Core Architecture ................................................... 32
3.2 MSC8156 DSP Architecture [6] ......................................... 34
3.3 ARM and DSP decoding performance [7] ................................ 35
3.4 Exemple of 2D Wave for a 5×5 MBs image. The arrows indicate de-
pendencies [8] .............................................................. 38
3.5 Exemple of 3D Wave Strategy [8] ........................................ 38
3.6 Complexity diagram of H.264 encoder from [9] ........................... 40
3.7 Complexity diagram of H.264 encoder from [10] .......................... 41
3.8 Implementation of a H.264 encoder on a DSP (a), on an ASIP (b) and
on a VSIP (c) .............................................................. 45
3.9 Alternative Intra Prediction Decision Mode ................................ 48
3.10 H.264/AVC Motion Estimation pattern ................................... 49
3.11 Motion Estimation alternative patterns ................................... 50

4.1 OMAP4460 Block Diagram [11] ......................................... 54
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3</td>
<td>TMS320C64x CPU [12]</td>
<td>57</td>
</tr>
<tr>
<td>4.4</td>
<td>DSP Gateway structure [13]</td>
<td>58</td>
</tr>
<tr>
<td>4.5</td>
<td>DSP Link software architecture [14]</td>
<td>59</td>
</tr>
<tr>
<td>4.6</td>
<td>Syslink modules architecture [15]</td>
<td>60</td>
</tr>
<tr>
<td>4.7</td>
<td>DSP Link and Syslink features [15]</td>
<td>61</td>
</tr>
</tbody>
</table>
GLOSSARY

**CABAC**
Context-based Adaptive Binary Arithmetic Coding is a method of entropy coding.

**CAVLC**
Context-based Adaptive Variable Length Coding is another method of entropy coding.

**CCS** Code Composer Studio is a coding software which includes a compiler.

**DCT**
Discrete Cosine Transform is a transform similar to Discrete Fourier Transform which is widely used for image and sound compression.

**DSP** Digital Signal Processor is a processor dedicated to signal processing.

**GPP** General Purpose Processor is a powerful processor which can handle complex algorithms.

**H.264/AVC**
H.264/Advanced Video Coding is a widely used video coding standard nowadays.

**HLOS**
High-Level Operating System is an operating system which provides a lot of facilities (sometimes similar to a computer’s OS) but has a high computational cost.

**IDCT**
Inverse Discrete Cosine Transform is the reverse function corresponding to DCT.

**ILP** Instruction Level Parallelism is a method which consist of executing several instruction in parallel.
**MPEG**  
Moving Picture Experts Group is a group of expert in charge of setting the standards for audiovisual coding.

**RD**  
Rate-Distortion is number representing the average of losses due to video encoding.

**RGB**  
literally Red Green Blue is a color space based on the average of those three colors.

**RTOS**  
Real-Time Operating System is a basic operating system including only essential features as opposed to HLOS.

**SADT**  
Sum of Absolute Transform Differences is an operation which compares the picture before and the picture after the Transform.

**SMID**  
Single Instruction Multiple Data is a technique which consist of applying the same instruction to several data in parallel. It is commonly used to define a specific kind of dedicated processor.

**VLIW**  
Very Long Instruction Word is a single instruction which contains several simple instructions. It is also used to define a specific kind of dedicated processor.

**YUV**  
It is a color space based on a luminance (Y) and two of the three chrominances (U and V). It is also called YCbCr refering to the luminance (Y) and the two chrominances Blue and Red.
1 Introduction

With the expansion of mobile technology and the huge growth of videos on the Internet, new challenges arose: providing good video quality with a low bit rate in order to ease the video transmission at a low computational cost. Indeed, one of the major improvements in mobile technology was the increase of the screen resolution and screen size. Therefore maintaining a good video quality is important. Because the mobile network is limited, the bit rate should remain low in order to have a fast video streaming. Mobile devices face a real challenge: the energy cost. As they are "mobile" they cannot be continuously plugged to a power source, as a result they have their own embedded power supply which is limited. Consequently the video coding should remain as simple as possible in order to reduce the energy cost.

The mainly used H.264/AVC provides brand new features which improve the video quality while reducing the bit rate by up to 50%. In return it increases drastically the complexity and, as a result, the computational cost. Many studies have been made in order to find a compromise between the quality features of H.264/AVC standards and the power consumption. One of them is to use heterogeneous platforms. Heterogeneous platforms are platforms containing several different cores. By sharing the workload among the different cores, better performances can be achieved. The energy cost can also be reduced by using dedicated processors which suits the software requirements. One of these dedicated processors is the Digital Signal Processor (DSP) which is dedicated to signal processing such as applying filters or operating basic arithmetic on arrays.

1.1 Thesis Purpose

Implementing a video encoder or a video decoder on a heterogeneous platform and optimizing the implementation may be really difficult. The purpose of this thesis is to have an overview of the different solutions for such implementation and investigate
several improvements. However there is no unique way to implement a H.264/AVC encoder or decoder on a heterogeneous platform because it depends on the H.264/AVC profile and on the platform itself. This thesis is just providing some guidelines on how it can be done.

1.2 Thesis structure

This thesis is divided into three parts. Chapter 2 describes the H.264/Advanced Video Coding standard and the features of the encoder and decoder. The next chapter, Chapter 3 introduces the heterogeneous platforms, the Digital Signal Processors (DSP) and the different solutions to implement a H.264/AVC encoder or decoder on a heterogeneous platform containing a DSP. The last chapter proposes an approach of platform (OMAP 4460) and tools (RPMsg) to implement the previously introduced solutions.
2 H.264/AVC Encoding and Decoding Techniques

This chapter introduces the H.264 Advanced Video Coding Standard. The first part of this chapter is dedicated to MPEG 4 standard and general facts about H.264/AVC standard. Then the behavior model of the H.264/AVC encoder is explained followed by the analysis of the H.264/AVC decoder model.

2.1 Overview of the H.264/AVC Standard

2.1.1 Introduction

In 1998, the Moving Picture Experts Group (MPEG) introduced a new video standard: the MPEG 4 (ISO/IEC 14 496). The main specificity of this new standard is to introduce the notion of "object" in video coding. This means that the media is divided in several "objects" which can be manipulated (access, cut, copy, past, move, etc...). Unlike the previous MPEG standard, the MPEG 4 standard is not only television oriented. It was made to meet the new requirements of the evolution of audiovisual techniques (High Definition quality, 3D...). But it was also made cover the need of new techniques in the scope of video streaming and broadcasting on mobile devices, highly motivated by the development of mobile phone technology.

In order to match this new standard and the different industry requirements, the International Telecommunication Union Telecommunication Standardization Sector (ITU-T) and the MPEG decided to work together and created the Joint Video Team (JVT) in 2001. The idea was to develop a new video coding standard more efficient. Meaning a flexible standard which can provide good video quality for a low bit rate. One of the main challenge is that providing a good video quality and reducing the bit rate imply increasing the complexity of encoding and decoding requiring more computa-
tional power and therefore more powerful processors and more energy. The challenge was to limit the complexity while providing a flexible, low rate, good video quality coding standard. We will see further that complexity might still be a problem for an implementation on a dedicated processor such as a DSP. With these aims in mind, they created the H.264/AVC: Advanced Video Coding (MPEG 4 part 10, ISO/IEC 14 496-10). In order to make it flexible, the H.264/AVC standard proposes 21 profiles which can be gathered in 4 profile "families": The Main profile, the Baseline profile, Extended profile and High profile. Depending on the profile, different features are included in the encoder/decoder. The list of the features included in the different profiles can be seen in Figure 2.1

About the video quality, the H.264/AVC can provide HD quality. H.264/AVC is used for HDTV broadcasting, Digital Cinema applications, Blu-ray and HD DVD (which was made by Toshiba to compete with Sony’s Blu-ray). It is also now used for 3D video coding. Few new techniques such as variable block size, In-loop Deblocking filter, or Network Abstraction Layer (NAL) system are use to improve the error robustness, and reduce the transmission losses or the distortions resulting from the blocking artifacts and therefore provide a better image quality. All these features will be described in the next subsection. In order to lower the bit rate, H.264 introduces a set of new features which can save up to 50% of the bit rate which will be detailed later.

### 2.1.2 Color Space Conversion

There are different ways to represent a picture, these way are called color space. The most known is the RGB space which codes each pixel with a grayscale value of the three colors composing the pixel: Red Green and Blue. As our visual perception is based on the intensity of the light which penetrates our eyes, we are more sensitive to the brightness of the picture than to the color itself. Therefore there is another way of coding a picture which is actually more accurate: the YUV space. The luminance (Y) is the intensity of the light on the picture (its average of grey). Basically the luminance...
Figure 2.1: The different H.264/AVC profiles and their features
is a black and white representation of the picture as you can see on the Figure 2.2.

To complete the representation, there are 3 chrominance values: Cb, Cr and Cg which are the average of the different colors blue, red and green. The relation between the RGB space and the YUV space is the following:

\[ Y = K_r R + K_g G + K_b B \]

With \( K_r = 0.299 \), \( K_g = 0.587 \) and \( K_b = 0.144 \) So we have \( K_r + K_g + K_b = 1 \)

Then we can calculate the chrominance values: \( C_b = B - Y \) \( C_r = R - y \) \( C_g = G - Y \)

As one of the chrominance values can be calculated from the two others and the luminance, only 2 chrominance values need to be transmitted. Therefore only 2 chrominance values are transmitted in video coding: Cb and Cr. To pass from RGB space to YUV space, we can use the following equation:

\[
\begin{bmatrix}
Y \\
U \\
V
\end{bmatrix} =
\begin{bmatrix}
0.299 & 0.587 & 0.114 \\
-0.14713 & -0.28886 & 0.436 \\
0.615 & -0.51498 & -0.10001
\end{bmatrix}
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix}
\]

For the reverse operation we can use the following formula:

\[
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 1.13983 \\
1 & -0.39465 & -0.58060 \\
1 & 2.03211 & 0
\end{bmatrix}
\begin{bmatrix}
Y \\
U \\
V
\end{bmatrix}
\]

There are several patterns for video coding, the format is coded according to the following method: \( X:Y:Z \) where \( Y \) is the number of chrominance samples (both Cr and Cb) for X luminance samples in the first line and \( Z \) is the number of chrominance samples for X luminance samples in the second line. By convention, \( X = 4 \). Figure 2.3 shows the common video coding patterns.

The main H.264 sampling pattern is 4:2:0, meaning for every four luminance samples we have two chrominance samples in the first line and zero chrominance sample in the second row.
Figure 2.2: RGB representation of an image (upper picture) and its Luma component in YUV representation
2.1.3 Frame Partitioning

The H.264/AVC unit is the macroblock. Each video sequence is divided into Group of Picture (GoP). And each Group of Picture is divided into frames. A frame is then divided in a new level of partitioning: the slices. A frame can be split into one or several slices depending on the encoding dependencies. The slices are then split into fixed size sections called macroblocks. Each macroblock contains: 16*16 samples of luma component (Y) and 8*8 samples of both blue-difference chroma (Cb) and red-difference chroma (Cr). A group of macroblocks is called a slice. If the encoder/decoder does not use the Flexible Macroblock Ordering (feature included in the baseline profile and the extended profile), the slice are divided following a raster scan process (meaning line by line from left to right and from top to bottom). Each slice are self-contained meaning that they are coded and decoded with the active picture parameter set but do not require any data from the other slices to be coded or decoded. There are five different kinds of slice:

I slice: an I slice is a slice in which every macroblock might be coded with Intra Prediction only (therefore called I macroblocks). That means only spatial redundancies from the current slice can be used. I standing for Intra frame.

P slice: this slice can include I macroblocks (macroblock coded with intra prediction) and P macroblock. The P macroblock is a macroblock which use Inter Prediction. The
Inter Prediction, described in section 2.2.4, is a technique which allows the encoder to encode the current frame using parts from other frames. The frames available for Inter Prediction are referenced either in list 0 if they are previous frames or in list 1 if it is a future frame. In the case of a P macroblock it uses one reference from a previous frame in list 0. P referring to Predictive slice.

**B slice**: the B slice can include I macroblocks, P macroblocks but also B macroblocks which are macroblocks coded with inter prediction such as P macroblocks but which can use 2 references, one from a previous picture (list 0) and one from a future picture (list 1). B standing for Bidirectional frame.

**SP slice**: the SP slice is a transition slice which is coded using temporal references such as P frame but using different references. They can be used for a more efficient switching random access and error recovery.

**SI slice**: an SI slice is used with a specific SP slice and it uses spacial prediction, such as I frame, to reconstruct identically the dedicated SP slice.

For more information about SP slice and SI slice, please refer to [16]

### 2.1.4 Network Abstraction Layer (NAL)

In the video coding/decoding chain, the H.264 has a step independent from the VCL (Video Coding Layer) to provide network transmission facilities: the NAL or Network Abstraction Layer. The NAL provide facilities for friendly transmission through a large scope of systems. The video data are organized in NAL units, there are two types of NAL units: the VCL NAL units containing the data of the coded video itself and the non-VCL NAL units containing information related to the NAL process. The first byte of each NAL unit indicates which kind of data is included in the NAL unit. A parameter set contains data that are supposed to rarely change such as parameter sets which can be applied to the whole video sequence (called Sequence Parameter Set) or parameter sets which remain the same for the whole picture (called Picture Parameter Sets). Those parameter sets are a small amount of data which are applied to a large amount of data. Therefore it saves bit rate avoiding a repetition of parameter sets.
2.2 H264/AVC encoding technique

Figure 2.4 shows the H.264/AVC encoder diagram. Every steps of the encoding process will be detailed in this section.

The Color Space Conversion block is the conversion from RGB space to YUV space. The color space conversion process has been covered previously in the MPEG4 and H.264/AVC chapter 2.1.2

2.2.1 Transform

The H.264/AVC uses $4 \times 4$ block size while the prior standard based their transform on $8 \times 8$ block size which prevents from ringing artifacts and blocking artifacts which therefore provides a better video quality. This choice was motivated by the importance of prediction in H.264/AVC standard: smaller block size allow better prediction as they can provide more matches. As a result, an encoder based on $4 \times 4$ blocks
will provide better data compression than an $8 \times 8$ DCT based encoder. However, the $8 \times 8$ block can be use for the Transform step. There are several transforms available depending on the profile (see Figure 2.1) : the $4 \times 4$ transform, the $8 \times 8$ transform and the Hadamard transform (for $4 \times 4$ and $2 \times 2$). The H.264/AVC provides a new $4 \times 4$ or $8 \times 8$ spacial based transform, unlike the Discrete Cosine Transform, the new transform does not switch from spacial space to frequency space. It is a purely integer transform which, as a result, provide an exact inverse transform. Consequently, there is no mismatch due to rounding coefficients as in DCT/IDCT transform. This means when transform and an inverse transform steps are processed, the matrix before and after remain the same. In addition, the new H.264/AVC transform suits more the low complexity platforms (such as DSP). Indeed, the former standards were based on the Discrete Cosine Transform which requires 32bits multiplication and 32 bits memory access. The new transform is based on simple integer arithmetic which is an approximation of the DCT transform and require only 16 bits for the arithmetic computation and 16 bits memory. As a conclusion, the new H.264/AVC Transform enhances the video quality avoiding mismatch between the Transform and the Inverse Transform steps, and reduces the complexity of both Transform and Inverse Transform steps. For further information about the new arithmetic transform design, please refer to [17]

2.2.2 Quantization

Quantization generally refers to the quantization step and the ZigZag Scan. The quantization part scales the matrices of coefficients from the Transform bloc in order to reduce the difference between the coefficients. This reduces the amount of bits needed to code the matrix. Each coefficient is quantized using a Scalar quantization, and the parameter of the quantization can take 52 values (the H.263 only provided 31 values). As a result, we observe a better data compression which provides a 12% bit rate reduction. The ZigZag scan reads the quantized matrix and reorders the coefficient according to a zigzag pattern. The aim of rearranging the coefficients is to gather the zero values. As most of the non-zero coefficients are in the top-left part of the matrix, the ZigZag pattern starts with these coefficients. The purpose of reordering is to facilitate the next encoding step : the entropy coding. If we consider the matrix of Figure 2.5, after the ZigZag scan, we have the following output :

\{7, -1, -1, 0, 1, 3, 1, 0, 0, 0, 1, 0, 2, 2, 4, 3, -1, 1, 1, 2, 1, -1, 0, 0, -1, 1, 1, 0, 3, 2,
Figure 2.5: ZigZag scan pattern with an 8 × 8 matrix

More information about the quantization and the ZigZag scan in H.264/AVC standards can be found in [17].

2.2.3 Intra Prediction

Based on spacial redundancy within the frame (as mentioned in section 2.1.3), the Intra Prediction is one of the key feature of H.264/AVC standard. While the previous standards included only one spacial prediction mode (the DC mode which will be explained briefly below), the H.264/AVC standard adds eight new modes for intra prediction and thus improve the video compression. The different modes are shown in Figure 2.7. Moreover, the previous MPEG 4 video standard performed the Intra Frame Prediction after the Transform process which means in the transform domains. That led to
error propagation due to the rounded numbers and transform processing losses. In order to avoid that, the H.264/AVC performs the Intra Prediction in the spacial domain (before the Transform block). Referring to Figure 2.6, the samples a to p are calculated as follows:
●mode 0 : Vertical
a, e, i and m are equal to A
b, f, j and n are equal to B
c, g, k and o are equal to C
d, h, i and p are equal to D

●mode 1 : Horizontal
a, b, c and d are equal to I
e, f, g and h are equal to J
i, j, k and l are equal to K
m, n, o and p are equal to L

●mode 2 : DC
Every samples have the same value
which is an average of the values of A, B, C, D, I, J, K and L

●mode 2 : Diagonal down left
a = \frac{A+B+C+2}{4}
b = c = \frac{B+2C+D+2}{4}
c = f = i = \frac{C+2D+E+2}{4}
d = g = j = m = \frac{D+2E+F+2}{4}
h = k = n = \frac{E+2F+G+2}{4}
l = o = \frac{F+2G+H+2}{4}
p = \frac{G+3H+2}{4}

●mode 4 : Diagonal down right
a = f = k = p = \frac{I+2M+2+2}{4}
b = g = l = \frac{M+2A+B+2}{4}
c = h = \frac{A+2B+C+2}{4}
d = \frac{B+2C+D+2}{4}
e = j = o = \frac{I+2J+M+2}{4}
i = n = \frac{K+2J+I+2}{4}
m = \frac{L+2K+J+2}{4}

●mode 5 : Vertical right
a = j = \frac{M+A+1}{2}
b = k = \frac{A+B+1}{2}
c = l = \frac{B+C+1}{2}
d = \frac{C+D+1}{2}
e = n = \frac{I+2M+A+2}{4}
f = o = \frac{M+2A+B+2}{4}
g = p = \frac{A+2B+C+2}{4}
h = \frac{B+2C+D+2}{4}
i = \frac{M+2I+J+2}{4}
m = \frac{I+2J+K+2}{4}

●mode 6 : Horizontal down
a = g = \frac{M+I+1}{2}
b = h = \frac{I+2M+A+2}{4}
c = \frac{M+2A+B+2}{4}
d = \frac{A+2B+C+2}{4}
e = k = \frac{I+J+1}{2}
f = l = \frac{M+2I+J+2}{4}
i = o = \frac{J+K+1}{2}
j = p = \frac{I+2J+K+2}{4}
m = \frac{K+L+1}{2}
n = \frac{J+2K+L+2}{4}

●mode 7 : Vertical left
a = \frac{A+B+1}{2}
b = i = \frac{B+C+1}{2}
c = j = \frac{C+D+1}{2}
d = k = \frac{D+E+1}{2}
e = A+2B+C+2
f = m = \frac{B+2C+D+2}{4}
g = n = \frac{C+2D+E+2}{4}
h = o = \frac{D+E+F+2}{4}
l = \frac{E+F+1}{2}
p = \frac{E+2F+G+2}{4}
Figure 2.6: Template of a $4 \times 4$ macroblock for Intra Prediction

- mode 8 : Horizontal up
  
  \[
  \begin{align*}
  a &= \frac{i+j+1}{2} \\
  b &= \frac{i+2j+k+2}{4} \\
  c &= e = \frac{j+k+1}{2} \\
  d &= f = \frac{j+2k+l+2}{4} \\
  g &= i = \frac{k+l+1}{2} \\
  h &= j = \frac{k+3l+2}{4} \\
  k &= l = m = n = o = p = L
  \end{align*}
  \]

As slices are encoded independently, the neighboring blocks might not be available. Therefore we can have 3 different cases:

- case 1 : the samples A-D are not available, therefore the modes which include these samples are disabled (which includes the modes 0 : Vertical, 2 : DC, 3 : Diagonal down-left, 4 : Diagonal down-right, 5 : Vertical right, 6 : Horizontal down and 7 : Vertical left).

- case 2 : the samples E-H are not known, in this case value of the sample D is used to replace them.

- case 3 : the samples I-L are not available, thus every mode using these samples cannot be used (meaning the modes 1 : Horizontal, 2 : DC, 4 : Diagonal down-right, 5 : Vertical right, 6 : Horizontal down and 8 : Horizontal up).
2.2.4 Inter Prediction (Motion Compensation, Motion Estimation)

While Intra Prediction is based on spacial redundancy, the Inter Prediction process is based on temporal redundancy. For Inter Prediction, the macroblocks can be split into smaller blocks from $16 \times 16$ block to $4 \times 4$ blocks while in the previous standards, the only supported format was the $16 \times 16$ block. The different possibilities are shown.
in Figure 2.8. Basically, for each and every Inter Predicted macroblock, a syntax element is sent indicating the shape of the macroblock which can be one block of $16 \times 16$ samples, two blocks of $16 \times 8$ samples, two blocks of $8 \times 16$ samples or four blocks of $8 \times 8$ samples or a combination of different formats. Furthermore, for every $8 \times 8$ blocks, an additional syntax element is sent, indeed, every $8 \times 8$ block can either remain a single $8 \times 8$ samples block or it can be further split into two $8 \times 4$ block, two $4 \times 8$ blocks or four $4 \times 4$ blocks or a combination of the different block formats. This increases the different layout possibilities for a macroblock coded with Inter Prediction and therefore enhance the Inter Prediction possibilities. Whether the $8 \times 8$ block is split or not the second syntax element is still transmitted for every $8 \times 8$ block. The Macroblock partitioning increase the possibilities of inter coding. Indeed, the smaller the areas are, the better the chances are to have a match. As a result Inter Prediction process increases the data compression and save around 15% of the bit rate. It also increases the video quality by providing more accurate matches and diminishes the blocking artifact. The blocking artifacts are further discussed in part 2.3.3.

Every block in an Inter Predicted macroblock is predicted from an area of the same size belonging to another picture. The offset between the current predicted block and its reference area is coded as a motion vector. As mentioned previously in section 2.1.3, there are two kinds of inter frame coded macroblocks, the P macroblocks and B macroblocks. The blocks within the P macroblocks are coded using only one reference area, from a previous frame, per block. So it is encoded using one motion vector per block and support every format of block (from $16 \times 16$ to $4 \times 4$ blocks). The blocks of a B macroblocks can use a weighted average of two reference areas from both previous and future frames. Therefore, every block within the B macroblock uses two motion vectors, and it can support block partitioning from $16 \times 16$ to $4 \times 4$. To sum up, every block of a P macroblock is coded using one area of the same size from a previous picture, while the blocks of a B macroblock are coded using two areas of the same size from previous or future picture for each block. Finally, a P macroblock can be coded using at most 16 different reference areas (in case of macroblock divided into sixteen $4 \times 4$ blocks) and a B macroblock can be predicted from at most 32 different reference areas.

The previous standards allowed inter coding among, at most, two previous pictures
only. In the H.264/AVC, the number of pictures that can be used for Inter Prediction is four. So in case of P macroblock, the references can be taken within four previously decoded pictures (which are stored in list 0). In case of B macroblock, the references can be taken among the next picture (stored in list 1) and three prior pictures (stored in list 0). Each used picture will have a reference index parameter $\Delta$ (which correspond to the number of frame before the one used for the Inter Prediction) and so when the current macroblock is coded with Inter Prediction, the index is sent with every motion vector. Like that, the same macroblock can be coded with several reference index parameter meaning with reference areas from several pictures as shown in Figure 2.9. Even if different motion vector can be used for each and every block of every block size from $16 \times 16$ to $4 \times 4$, there can be different reference index parameter for each block (one reference index parameter for each block of P macroblocks, two for each block of B macroblocks) only if the block size is $16 \times 16$, $16 \times 8$, $8 \times 16$ or $8 \times 8$. For every blocks with lower block size (block size between $8 \times 8$ and $4 \times 4$), the same reference index parameter is used for the whole subdivision of the $8 \times 8$ block (or the same reference index parameters are used for the whole subdivision in case of block within a B macroblock). There cannot be Inter Prediction across the slice boundary as mentioned in section 2.1.3. In other words, a picture from a previous slice cannot be used for Inter Prediction, every pictures used for Inter Prediction must belong to the same slice. As a result, the flexibility of inter coding is increased and therefore enhance the data compression providing about 5-10% bit rate reduction compared to standard using only one reference picture. On the other hand, it requires much more memory to store the frame on both the encoder and the decoder.

The previous standards provided half pixel accuracy for the motion estimation of the luma blocks. The H.264/AVC goes further and provides one quarter pixel accuracy for luma component (and one eighth for chroma components). The value of the half-samples are calculated by applying a 6-tap FIR filter horizontally and vertically to the integer samples while the quarter-samples values are obtained by averaging the values at integer position and half-samples position. The chromas are calculated using bilinear interpolation. Figure 2.10 presents a half-pixel representation while Figure 2.11 is his corresponding quarter-pixel pattern.

Considering Figure 2.10, the half pixel values are calculated as follow:

$$b = \text{round} \left( \frac{E-5F+20G+20H+5I+J}{32} \right)$$
Figure 2.8: Macroblock partitioning for Inter Prediction

\[ h = \text{round}\left(\frac{A - 5C + 20G + 20M + 5T + T_32}{32}\right) \]

And according to Figure 2.11 the quarter pixel values are calculated this way:

\[ a = \text{round}\left(\frac{G + b}{2}\right) \]
\[ d = \text{round}\left(\frac{G + h}{2}\right) \]
\[ e = \text{round}\left(\frac{b + h}{2}\right) \]

Once the Motion Estimation has found the best match, the motion vector is calculated. The quarter pixel accuracy increases the Inter Prediction flexibility and can yield 20% in bit rate saving.

2.2.5 Entropy Coding

The Entropy Coding is a lossless data compression technique. It consists in coding the output with the fewest amount of bits possible. H264/AVC proposes two different
Figure 2.9: Multi frame prediction [3]

Figure 2.10: Half-pixel interpolation [3]
Figure 2.11: Quarter-pixel interpolation [3]
entropy coding methods: CAVLC (Context-based Adaptive Variable Length Coding) and CABAC (Context-based Adaptive Binary Arithmetic Coding).

The CAVLC method is a new coding method based on the Variable Length Coding method: instead of coding every coefficient of the ZigZag scan output, CAVLC will take advantage of the fact that the output contains many 0 value gathered at the end and most of the non-zero coefficients are either 1 or -1. The data are computed in the reverse scan order. In this configuration, the output will begin with the bottom-right zero coefficients. If we consider the ZigZag scan example, the data stream will be 0, 0, 0, 0, [...] 1, 0, -1, -1, 7. The CAVLC is based on a set of parameters which are:

- **Number of non-zero coefficients** ($N$) which is the number of coefficients in the output whose value is not zero.
- **Trailing ones** ($T1s$) which is the number of coefficients in the output whose value is either 1 or -1 ($T1s$ value cannot be higher than 3, if there is more than three coefficient whose value is 1 or -1, the three first will be coded as $T1s$ but the others will be coded as other non-zero coefficients) The Number of non-zero coefficients and the Trailing ones are coded as a single event.
- **Sign of Trailing ones** indicate the sign of the $T1s$, it is coded in the reverse scan order, one bit per sign: 0 if the coefficient is 1, 1 if the coefficient is -1.
- **Level of the other non-zero coefficients**: Every non-zero coefficient which are not included in the Trailing ones (this might include 1 and -1 coefficients if there are more than 3 of these coefficient in the output) are coded both sign and value in the reverse scan order.
- **Total Zeros** indicate the number of zero coefficients between the first non-zero coefficient and the last non-zero coefficient in the reverse scan order. By adding this value to the Number of non-zero coefficients, we can deduce the number of zero coefficients at the end of the ZigZag scan.
- **Run Before** is the number of successive zero coefficients before every non zero coefficient. It is performed in the reverse scan order. With the Total Zeros parameter, we know how many zeros are between the first and the last non-zero coefficient and with $N$ we know the number of non-zero coefficients. Therefore as soon as every 0 are counted (meaning there are no zero coefficient left until the last non-zero coefficient, so every next Run Before value are 0) or if the last non-zero parameter is reached (so the remaining coefficients are 0), the remaining Run Before parameter are not coded as they can be easily deduced.
When the parameters are defined, they are then coded using Lookup Tables (LUT) based on Exp-Golomb codewords. The same Lookup Table is used to code a parameter except for the parameter Level of the other non-zero coefficients which may use different tables based on the value of the different coefficients and for the parameter Run Before which can use different tables for each position.

The CABAC method is based on binary arithmetic coding technique, which means that recurrent sequences of bits are linked to a specific shorter code. By doing that, every time the sequence is repeated, the code is sent instead of the sequence which leads to a save of bits. The most frequently used sequences will be encoded with the shortest codes, the less redundant the sequences are, the longer the code will be so the compression will be optimum and lossless. This technique suits the H.264/AVC pattern which is to exploit redundancy to reduce the bit rate without decreasing too much the video quality. It is admitted that CABAC is extremely beneficial (and much more efficient than CA VLC) for a sequence with a probability higher than 0.5. The Adaptive term in CABAC comes from the fact that the probability model is updated while the data are processed (as explained below in the Context Modeling part). The CABAC process is shown on Figure 2.12. There are three steps in the CABAC : the Binarization, the Context Modeling and the Binarization Arithmetic Coding. The Binarization categorizes the zigzag sequences depending on the probabilities of the values of the bins. They can be sorted out in three categories : the regular bins which have variable probabilities, the bypass bins which have even probabilities and the terminate bins which have dominant probabilities. The Context Modeling update the probability models for the regular bins : as they have variable probabilities of value, the probability model will change. This part chose a template which is called context model, encode the regular bins with it and update the context model. Instead of calculating the probability of the model which would imply complex operations (basically multiplications), the assumption was made that the context models could be defined by a set of representative values (64 values precisely in the case of CABAC) so the whole CABAC method is multiplication free. The last step is the Binary Arithmetic Coding according to the probability model given in the previous step (or the context model for the regular bins).
Figure 2.12: Block diagram of the CABAC encoder of H.264/AVC [4]
Owing to its adaptivity (meaning that the context model for regular bits is always updated according to the previous and current encoded sequences), the binary arithmetic coding improves the compression efficiency by 2-3 times comparing with non-adaptive binary arithmetic coding. Although CABAC is multiplication free, it is based on lookup tables matching and complex decision processes which generally increase its complexity compared to CAVLC. On the other hand, the overall bit rate saved between CAVLC and CABAC is between 5 and 15 per cent. Entropy coding is further investigated in [4].

### 2.3 H264/AVC decoding technique

Figure 2.13 shows the H.264/AVC decoder diagram.

The **Color Space Conversion** block is the conversion from YUV space to RGB space. The conversion process was explain before in the MPEG 4 and H.264/AVC section 2.1.2.
2.3.1 Entropy Decoding

The **Entropy Decoding** is the reverse operation of CAVLC or CABAC, it is about reconstructing the "macroblock coefficients" from the NAL bit stream using the LookUp Tables.

2.3.2 Inverse Quantization and Inverse Transform

The **Inverse Quantization and Inverse Transform** are the inverse operation of the Transform and Quantization steps of the encoder. As mentioned before, the Inverse Transform is the exact inverse operation of the Transform so it avoids losses due to mismatches. As a result, the video quality is increased. The Inverse Quantization consists of restructuring the matrix from the entropy decoding step output and scaling it to get the matrix as it was before the Quantization step in the encoder.

2.3.3 In-loop Deblocking Filter

The **In-loop Deblocking Filter** is a new feature introduced in the H.264/AVC standard. Its purpose is to reduce the video quality losses due to macroblocks segmentation. Indeed, coding a video using macroblock partition can create what is called "blocking artifacts". As the video is encoded macroblock by macroblock, (and mainly because of the Quantization step) the edges of the macroblocks become less accurate as they tend more to the "general value" of the macroblock. As a result a strong difference between the edges of two neighboring macroblocks might be observed, and so the picture might look like "pixelized" as we can see on the Figure 2.14. The filter is called "In-loop" because it is performed right after the Inverse Transform and before the Frame Store. The filter is included in the process, it is not a post-processing step, which means that it filters the current macroblocks and not a reconstructed frame, so the filtering is taken into account for intra and inter prediction.

The filter processes separately on luminance and chrominance components. It filters first vertically (from the top to the bottom) and then horizontally (from the right to the left). Sometimes, there is a real difference between the edge of the macroblocks not due to the encoding/decoding process but to the image itself. That is why a decision
process is included in the In-loop Deblocking Filter. The process is the same for both vertical and horizontal filtering. To explain the decision process, we will consider the horizontal filtering of two macroblocks as shown in Figure 2.15.

The strength of the filtering is calculated according to the Boundary Strength (BS) which can take value between 0 and 4 (0 means no filtering is needed, 4 means a strong filtering will be applied). The value of the Boundary Strength is decided following the chart in Figure 2.16. The Boundary Strength is not the only condition required to apply a filter. If we consider \( \alpha \) and \( \beta \) which value depends on the Quantization Parameter used for their macroblocks, a filter is applied only if:

- \( BS \neq 0 \)
- \( \text{abs}(p1 - p0) < \beta \)
- \( \text{abs}(p0 - q0) < \alpha \)
- \( \text{abs}(q0 - q1) < \beta \)

If all these condition are fulfilled, then the filter is applied and its strength is based on the value of BS, p0, p1, p2, p3, q0, q1, q2 and q3. The Deblocking Filter is a quality
step which minimizes the block distortion and therefore enhances both the complexity of the encoder/decoder and the video quality. It also saves 5 to 10\% of the bit rate. The In-loop Deblocking Filter process is more detailed in [19].

\section*{2.4 Conclusion}
To conclude, this chapter provided a brief overview of the H.264 Advanced Video Coding standard focusing on the mechanisms which will be relevant for the next chapter. This standard introduces a lot of new features compared to prior encoding/decoding standard improving the video quality and the bit rate. However, the H.264/AVC standard is made of different profiles and these different profiles do not include the same features. As a result, depending on which profile the encoder/decoder is based, the complexity of the encoding/decoding process, and the performances are not the same, but this matter will be discussed in the next chapter. The encoder and decoder are structured with "functional blocks", each of these blocks performing its own dedicated task. For a deeper understanding of H.264/AVC standard, more information can be found in [3][20][21][22][23].
Figure 2.16: Boundary strength (BS) computation flowchart [5]
3 HETEROGENEOUS PLATFORMS

This chapter starts with an introduction to Heterogeneous Platforms. Then it presents a specialized processor: the Digital Signal Processor which is the co-processor family chosen for the study. The last section of this chapter is dedicated to the solutions for the implementation of a H.264/AVC encoder or decoder on a heterogeneous platform with a DSP. The first subsection proposes the different solutions of mapping while the second subsection is about different techniques and algorithms to reduce the complexity of the H.264/AVC encoder and decoder in order to offer other mapping solutions.

3.1 Heterogeneous Platform

Today’s computing challenges are based on three problematics: power consumption, performance and portability. In the early 2000s, while computing industry is growing fast, engineers face a major issue: the physical limitations. Indeed the CPU frequency reached several GigaHertz but could not be improved anymore due to heat and data synchronization issues. Therefore the engineers worked on another solution: instead of increasing the power of a single processor to execute itself the whole workload, they shared the workload among several processors and that was the beginning of parallel programming. Going further on that idea, they decided to design processors containing two identical but independent processing unit which would communicate through a common bus interface as shown on Figure 3.1. In 2001, IBM released the first Dual-Core processor, the POWER4, after them, AMD and Intel released their own multi-core processors. Those multi-core chips are Homogeneous Multi-core Systems meaning that they contain several time the same "core". There is another category of multi-core chips called Heterogeneous Multi-core Systems. The word "heterogeneous" refers to the fact that the same chip contains several different cores. This leads to several issues whose most critical are the workload distribution and data synchronization. As the processors within the system are different they cannot handle the same tasks.
Heterogeneous Platforms are a satisfying solution for today’s computing challenges. Nowadays a Heterogeneous platform is made of Central Processing Units (CPUs) and one or several specialized processor (usually a Graphics Processing Unit, known as GPU). While a CPU is made to perform complicated operations, a specialized processor is usually a parallel structure made to execute simpler, redundant tasks but to be able to perform them faster than a CPU and for a lower energy cost. As the cores of a heterogeneous platform cannot perform the same tasks or do not perform them at the same speed (the same operation will not require the same amount of cycles to be performed on a GPU and on a CPU, also usually the frequency of the core is not the same). The main challenge of a good implantation on a heterogeneous platform is the workload sharing (due to processing optimization, data synchronization...)

Video decoding and especially H.264/AVC is really complex and requires a lot of computing. A Heterogeneous Platform with a CPU for complicated tasks and a co-processor dedicated to hardware acceleration in order to perform faster the simple and redundant tasks is a perfect solution to manage the computational cost of a H.264/AVC encoder or decoder. Due to the boom in demand of heterogeneous platforms, a non-for-profit organization, the Heterogeneous System Architecture Foundation was created to improve this technology and define a standard. This association is supported by the leaders in processor, SoC, semiconductor, operating systems and software markets such as ARM, AMD, Texas Instruments, Linaro and Ubuntu. Even if the Heterogeneous Platform is the chosen solution for this study, it is not the only solution to deal with audiovisual requirements and challenges. The Chapter four of [3] propose different kind of media processor (including HSA). The next section introduces the Digital Signal Processor, a co-processor family dedicated to Digital Signal Processing.

3.2 DSP

The Digital Signal Processor is a dedicated processor with an optimal designed architecture for Real Time Digital Signal Processing. In other words it is made to process basic arithmetic operations and filtering which are not highly complex operations. On the other hand, it can perform it really fast due to its optimized assembly instructions and its very efficient parallelization oriented architecture. Both make DSP able to perform several instructions per clock cycle. The other main advantage of DSPs is that they are really low energy consumers compare to General Purpose Processors. It is
really hard to set a general profile as there are different manufacturers, each of them offering different kind of DSPs with different inner architecture, different frequencies or different instructions. As an example, in Figure 3.2, the MSC8156 DSP is an advanced DSP with 6 Star Core 3850 DSPs. Each of them is a single DSP with a frequency up to 1GHz. As it is a specialized processor, the companies should provide suitable solutions for the different needs and this is the reason why there are so many different DSPs. However we can define some general facts about DSPs. A DSP is a specialized processor optimized for digital signal processing. It can perform several basic operations in parallel. However it cannot perform multitasking. Even if some DSP can perform complex operations or advanced data manipulation, a DSP is not designed for this purpose therefore, completing these tasks will require a lot of computational power to the DSP while it can be perfectly handled by a GPP.

In order to enhance the performance of a DSP, the compilation is usually really complex and requires a lot of time due to code optimization and the mapping part of the compiler. Like every hardware acceleration device, the architecture of a DSP is really complex and needs to be exploited correctly to provide the best performances. One of the approaches offered by some DSPs is the Very Long Instruction Word (VLIW). The idea is to propose complex instructions gathering several simpler operations which will be distributed on different parts of the processor and executed in parallel. The aim of VLIW is to provide advanced instructions without making concessions on the com-
plexity of the computation. However, the mapping and therefore the compilation are much more complex.

While most of the DSP can only use integer (or fixed-point numbers), there are some which can use floating-point numbers. Those DSP are generally more expensive than the ones using only integers or fixed-point numbers, but this price is usually balanced by the complexity and the energy cost that would represent floating-point calculation on a DSP dealing only with fixed-point numbers or integers.

The DSP takes advantage of Instruction Level Parallelism (ILP) meaning that the parallelization is done at a very low level: the instruction level (this explains why the compilation is complex and requires a lot of time). One technology related to ILP is commonly used with DSPs: Single Instruction Multiple Data (SIMD). The goal is to perform simultaneously the same instruction on several independent data. The main advantage is that some heavy calculation process due to redundant simple operations (such as applying a filter on a full picture, or incrementing a huge matrix) that would require a lot of time on a GPP can be perform much faster on a DSP using SIMD.

3.3 H.264/AVC Complexity Analysis

3.3.1 H.264/AVC Mapping Solutions

Running a H.264/AVC decoder or encoder on a heterogeneous platform can be done in different ways. This thesis only presents the solutions which implies running the encoder (or decoder) using the different cores (at least two) on the heterogeneous platform, one of these core being a DSP. Therefore we can define three main solutions: data level parallelization, task level parallelization and serial execution.

The serial execution is based on the same argument than the ones for task level parallelization, the main difference is that only one core works and the other are idle. That is why this specific case will not be further investigated.

As mentioned before, there are many different kinds of DSPs, some really powerful, other much more simple. As a logical conclusion, each and every case requires specific solution. The solutions that will be presented in this part are based on general facts about different range of DSPs.
Figure 3.2: MSC8156 DSP Architecture [6]
**Data Level Parallelization**: The Data Level Parallelization is about processing the full encoder (or decoder) on the different cores and encode (or decode) in parallel a certain amount of data. Here, the complexity of the different functions of the encoder or decoder does not matter, the whole encoding or decoding need to be processed on the DSP. As a logical conclusion, the Data Level Parallelization require a powerful DSP capable of handling the full encoding or decoding process and to deal with a large amount of data as well. This amount depends on the partitioning method as it will be explained below. However the overall complexity of the process does matter and so does the amount of data that need to be encoded or decoded.

Figure 3.3 shows a comparison of video decoding at different resolution (QCIF : 128kb, CIF : 1024kb and 4CIF : 5120kb) of the same video depending on the bit rate and the clock frequency. The ARM platform is a Cortex A8 ARM and the chosen DSP is a TMS320C64x. This shows that a DSP can achieve better performance than a GPP (ARM) on video decoding.

The explanation of these results is that the average of computing required for the overhead is approximately 10% of the total decoding time on the GPP while on the DSP, this average is 52.64% for a low resolution (QCIF) versus 28.11% for a higher resolution (CIF) and versus 8.48% for the highest resolution (4CIF). The whole evaluation of video decoding performances and energy consumption between a GPP and a DSP can be found in [7].

The first logical partitioning method is the Group of Picture partitioning. As every Group of Picture is coded independently, they can also be decoded independently. Therefore, the only tricky thing is the synchronization of the encoded data in case of an encoder or of the decoded video in case of a decoder. Indeed, as the cores are different and as the amount of operation to encode or decode a Group of Picture in not
the same (mainly due to the inter prediction and intra prediction), a synchronization process is required either before the NAL processing in case of an encoder, or before the video screening in case of a decoder. If the implementation is easy, there is a major drawback for this solution: the hardware requirement. A Group of Picture represents a lot of data. That is why either a huge inner memory is needed for every core or a shared memory with fast access for every core, which are both expensive solutions. This solution can be considered as non-suitable solution for a heterogeneous platform with a DSP because it does not really improve the performances of the encoding or decoding.

The next partitioning level is a Frame partitioning. Due to the frame dependencies for frames containing P slices and B slices (which imply Inter prediction), the implementation of such a solution is difficult. Therefore this solution will not be further explored.

The H.264/AVC introduces a new granularity level: the slices. Slices are subdivision of frames, they are divided according to coding facilities. Where previous standards defined the intra prediction and inter prediction at a frame level, the H.264/AVC does it at the slice level, therefore it is easier to identify the dependencies and it provides slices that can be decoded without any requirement from other slices. For instance, if we consider the implementation of a decoder on a heterogeneous platform, the I slices, P slices and B slices will be decoded on a GPP as they require a lot of memory and a higher complexity for intra prediction and inter prediction. The other slices can be decoded on the DSP. The same synchronization issue as mentioned in the Group of Picture partitioning must be taken into account in case of slice partitioning. While the asset of slice partitioning is obvious for the H.264/AVC decoder, it cannot be used for the encoder. As a matter of fact, the slice division is done late in the encoding process making a slice partitioning impossible for encoding.

The last partitioning level is both the lowest partitioning level and the most widely used: the macroblock partitioning. While the block (subdivision of macroblock for inter prediction and intra prediction) partitioning exists, it can only be used with both Data Level Parallelism and Task Level Parallelism. The combination of both will be mentioned in the end of this subsection but is not the topic of this thesis. Therefore,
the macroblock partitioning is indeed the lowest partitioning level. The major inconvenient of macroblock partitioning is the coding (or decoding) dependencies between the macroblocks. Here again, there is a major difference between the encoder and the decoder. As the dependencies between the macroblocks are created during the encoding process, it is tricky to foresee them without restricted the inter prediction or the intra-prediction. A solution would be to establish a pattern for inter-prediction and for intra-prediction based on the probabilities that we might have a recurrent scheme (a bit like in the CABAC technique). This would provide a good scalability for macroblock encoding but might increase significantly the bit rate. As for the decoder, there are several techniques to anticipate the macroblock dependencies. Two of the most well known techniques are the 2D wave and 3D wave decoding. As shown on Figure 3.4, the 2D Wave technique uses the fact that Intra prediction and In-Loop Deblocking filter’s dependencies can be predicted. Hence the macroblocks are decoded on a "wave" order to optimize the scalability. On the other hand, the 3D Wave technique is based on the fact that Inter Prediction is limited to close geographic areas as shown on Figure 3.5. As a consequence, there is no need to wait until the current frame is fully decoded to start decoding the next frame. It means that the macroblocks from different frames can be decoded in parallel. Both techniques (2D waves and 3D waves) require a dynamic scheduling to provide optimum performances. The 2D Wave and 3D Wave techniques are more detailed in [8]. Macroblock partitioning scalability for a H.264 decoder (including the different bottleneck, an evaluation of different scheduling and few improvements) and 2D Wave technique are further investigated in [24] and [25].

The macroblock partitioning provides a good scalability which increases with the resolution: a higher resolution provides a higher number of potential independent macroblocks and consequently a better scalability. Even if it is the best partitioning option for parallel decoding on a heterogeneous platform, the Entropy decoding step cannot be parallelized using Data Level Parallelization based on Macroblock Partitioning. Hence the entropy decoding time is a major concern for the overall decoding performances.

**Task Level Parallelization** : While the Data Level Parallelization is based on splitting the data and run the encoding or the decoding process among the different cores, the Task Level Parallelization takes advantage of the H.264/AVC step processing struc-
Figure 3.4: Example of 2D Wave for a 5\times5 MBs image. The arrows indicate dependencies [8].

Figure 3.5: Example of 3D Wave Strategy [8].
ture. It allows different encoding or decoding steps to different cores and every data is processed through the different steps on the different cores which makes the inter-core communication a critical feature. As Data Level Parallelization is not suitable for the whole encoding process, a combination of both Data Level Parallelization and Task Level Parallelization is usually implemented for parallel encoding according to H.264/AVC standard. There are a lot of different DSPs which can handle different level of complexity. Therefore, this section will not provide the unique solution for implementing a H.264/AVC encoder or decoder on a heterogeneous platform containing a DSP as every heterogeneous platform configuration has its own advantages and drawbacks. But it will give an overview of the different H.264/AVC steps complexity and possible schedules.

Task Level Parallelization must take into account two aspects of H.264/AVC encoding and decoding: the tasks complexity and the data dependencies between the tasks. Even if the complexity is relative to the platform on which we implement the H.264/AVC encoder or decoder, we can sort out general facts about the different block. The H.264/AVC new features increase the complexity of the encoder by ten and the complexity of the decoder by two. Figure 3.6 and 3.7 show two complexity diagrams for the H.264/AVC encoder on different processors. As we can see, the inter prediction is the most complex step with more or less half of the decoding time. The entropy coding (here VLC entropy coding) is about twenty percents. The transform and quantization is about fifteen percents. However there are slight differences depending on the platform we use. Therefore what follows are general facts concerning complexity and dependencies for the different steps of the encoder and decoder.

The Color Space Conversion is a simple conversion from RGB space to luminance and chrominance space, this operation is not really complex and is the first step of video decoding. This step can perfectly be performed in parallel with any other step of the encoding process as it has no dependency with any other step. However, mapping this step only on a DSP core while the rest is executed on another core is not a real improvement of the overall process.

The Transform step is a low complexity step which only requires a spacial, purely integer transform which is much simpler than a frequency transform. However, even if the DCT transform would have been in use, almost every DSP provides a DCT
Figure 3.6: Complexity diagram of H.264 encoder from [9]
The DCT is instruction parallelized oriented which means it is optimized to perform a simple instruction on a large amount of data at the same time. The transform is independent for each and every macroblock and shares no bonds with any other task of the encoding process. This is why it is interesting to map the transform step on a DSP in case of Task Level Parallelization.

The Quantization is also a simple operation. It requires only a lookup table stored in the core memory. As the coefficient parameter can take only 52 values, it can easily be stored even in a very basic DSP. The ZigZag scan too requires a bit of memory because it needs to store in the core memory the whole quantized macroblock in order to perform the scan. The quantization is performed in the macroblock order and the ZigZag scan reorders the macroblock so the full macroblock needs to be stored between these two steps before the ZigZag scan is doing its data manipulation. Even if DSPs are usually not suitable for data manipulation (mainly due to the fact that they are short on internal memory), the ZigZag scan complexity and memory requirements are not too demanding. As a result, the Quantization step is perfectly suitable for a DSP implementation. Furthermore, this step does not depend on the other steps of the encoder, so it is perfectly suitable for Task Level Parallelization.
and the Quantization step are usually considered as only one step because they are a logical continuation, they process on the same data requirements, and they are similar operations with low complexity. As a result, these two steps are executed sequentially is the case of Task Level Parallelism even though they can be executed in parallel. Parallelizing these two steps does generally not enhance the encoding performances.

The Inter Prediction is the most complex step of H.264/AVC encoding and decoding. One of the reasons for their high complexity is that they manipulate data. Data manipulation requires a huge internal memory to store the data that needed to be manipulated. The data can be stored in a shared memory but there will be access conflict and the data synchronization will be harder then. In addition to memory requirement, data manipulation is usually based on complex and various operations which are not suitable for a DSP as the DSPs are optimized for simple and redundant operations. In the case of the inter prediction, the motion estimation looks for the best match using quarter pixel accuracy which enhance the complexity due to the calculation of the half and quarter samples values and to the matching comparison. The macroblock sub-partitioning increases the inter-prediction possible matches but, as a logical consequence, it also increases drastically the complexity of the Inter Prediction process. The calculation of the motion vectors and their storage for each sub-block is the less demanding part but it cannot be split from the Inter Prediction block. Inter Prediction and Intra Prediction cannot be performed on the same macroblock. Moreover, both Inter Prediction and Intra Prediction are performed on a non-transformed and non-quantized macroblock. Thence, Inter-prediction, Intra-prediction and Transform/Quantization can be executed in parallel. Considering the previously element, the Inter Prediction step must be assigned to a GPP and not to a DSP.

The Intra Prediction is using spacial redundancy. Therefore, it needs access to the whole frame in order to find the best possible match and then determine the Intra Prediction Mode. It is a decision process which has a high complexity and which has a high computational cost. In addition, the Intra Prediction is based on data manipulation. As stated previously, the DSPs are not suitable for data manipulation. Even if the Intra prediction remains much less complex than the Inter Prediction, it is still not a good match for DSP implementation, it is worthier to implement is on a GPP. Because the Intra Prediction shares no bonds with the Inter Prediction and the Transform/Quantization, it remains a good option for Task Level Parallelization with both Inter Prediction and Transform/Quantization.
The Entropy Coding offers two choices: CAVLC (Context-based Adaptive Variable Length Coding) and CABAC (Context-based Adaptive Binary Arithmetic Coding). The CAVLC basically consists of data manipulation but at a relatively simple level. As it was explained in Chapter 2.2.5, CAVLC, encodes the stream counting the number of zeros between two non-zero numbers and encode the sign and level of every non-zero value. The full macroblock need to be stored in the internal memory of the processor and so does the Exp-Golomb lookup Tables, which is not a critical requirement even for small DSPs. As shown on Figure 3.6 and Figure 3.7, the CAVLC complexity represents more or less twenty percent of the overall complexity. It is the last step of the encoding process. The complexity of the operations performed is low compared to Inter Prediction, but higher than Transform or Quantization which makes the CAVLC process a good candidate for DSP implementation on an advanced DSP. The CABAC process is made of different pattern probabilities which lead to the binary arithmetic coding of the sequence. While it is multiplication free, the CABAC step requires a significant internal memory to store and update the lookup tables and is made of a highly complex decision process. As a result, it is recommended to implement the CABAC on a GPP. The entropy coding is the last process of the H.264/AVC encoder. It requires the data from the Inter Prediction, Intra Prediction and Quantization. Logically, it cannot be executed in parallel with these steps.

The In-loop Deblocking filter is a filtering operation. Then it should logically fit a DSP implementation, but it requires having both the current macroblock and the edge of the neighboring macroblock as the goal of this filter is to avoid the blocking artifacts. For this reason, its internal memory requirement is important and it cannot be implemented on a basic DSP. But a mid-range DSP will take advantage of the filtering optimizations and its instruction parallelism to provide good performances for this step for both encoder and decoder. The In-loop filter has to be performed after the Inverse Quantization and the Inter Prediction and Intra Prediction because it requires edges from predicted block to apply the filter.

In a similar way than for the Transform and Quantization steps, the Inverse Quantization and the Inverse Transform are low-complexity steps that are usually considered as a single step. It can be performed in parallel with the Entropy Decoding, the Predictions (Intra Prediction or Inter Prediction) and the In-loop Deblocking Filter. Like the Entropy Coding step, the Entropy Decoding proposes CAVLC and CABAC. The arguments presented in the Entropy Coding paragraph are also valid for the En-
entropy Decoding which means that a CAVLC step can be run on a DSP while CABAC should be run on a GPP.

Even if the Inter Prediction and Intra Prediction processes are not the same in the case of the decoder, the conclusions are similar. Indeed, in the decoding process, the motion vector, provided in the bit stream, already points to the matches. Consequently, Motion Estimation is not included in the decoder as the decoder does not need to look for the matches to decode the current macroblock. Similarly, the prediction modes are already given for Intra Prediction. Therefore the current macroblock can be calculated according to the given mode, the decoder does not need to find the best matching mode. Whereas the process is simpler as the Intra Prediction mode is given, the Intra Prediction still needs to access the whole frame to locate the matching macroblock which implies a large internal core memory which is generally not the case in DSPs. As for the Inter Prediction, it still needs to access the reference frames which cannot obviously be stored in a DSP internal memory. In addition, the half pixels and quarters pixels might have to be calculated. Therefore, the Inter Prediction remains a highly complex step.

To sum up, there are multiple possibilities for Task Level Parallelization. Such a parallelization is worthy as long as the time required for the "longest sequence of steps" is shorter than the time required for a sequential execution of every step. As an example, Figure 3.8 from [9] proposed to parallelize the Intra prediction, the Inter Prediction, the Entropy Coding and In-loop Deblocking Filter. The (a) represents the sequential execution on a DSP, the (b) is the parallel implementation on an Application Specific Instruction set Processor (ASIP) and (c) is the parallel implementation on a Video Specific Instruction set Processor (VSIP). For (b) and (c) the time required to encode one macroblock is determined by the "longest sequence step" meaning Intra Prediction, Transformation and Quantization and the In-loop Deblocking Filter, it is always shorter time than (a) which is the sequential execution of every step. Once again, there is no unique solution for Task Level Parallelization implementation of a H.264/AVC encoder or decoder on heterogeneous platform which contains a DSP core. Several parameters such as the video resolution, the internal memory of the DSP, the inter-core shared memory have to be taken into account. As a general fact, the most suitable steps for a DSP are the Transform and Quantization, Inverse Transform, Inverse Quantization, and the In-loop Deblocking Filter.
We have investigated the two main parallelization techniques to map a H.264/AVC encoder or decoder on a heterogeneous platform including a DSP. As mentioned before, the efficiency of these methods can be further improved by combining these methods. Although the techniques related to a combination of both Data Level Parallelization and Task Level Parallelization will not be reviewed in this thesis, elements related to task dependencies and data dependencies might be relevant for such implementation. Some examples of such a combination are presented in [26], [9], [24] and [27]. [28] offers an approach of a combination of Data Level Parallelism and Task Level Parallelism. It presents a solution to reduce the complexity due to the overheads by
-mh Allows speculative execution. But the appropriate amount of padding must be available in data memory to insure correct execution.

-o3 Represents the highest level of optimization available. Various loop optimizations are performed, unrolling such as software pipeline, and SMID.

-pm Combines source files to perform program-level optimization.

-op Indicates if functions in other modules can call a module’s external functions or modify a module’s external variables

-on When the -O3 option is used, -on option creates an optimization information file that you can read.

<table>
<thead>
<tr>
<th>Table 3.1: CSS Compiler Options [29]</th>
</tr>
</thead>
<tbody>
<tr>
<td>-mh</td>
</tr>
<tr>
<td>-o3</td>
</tr>
<tr>
<td>-pm</td>
</tr>
<tr>
<td>-op</td>
</tr>
<tr>
<td>-on</td>
</tr>
</tbody>
</table>

using a Preparing Step. The Preparing Step deals with the decision mode (dependencies and macroblocks properties) and passes the required information to the different processors depending on the scheduling. Three different schedules are reviewed in this article.

### 3.3.2 H.264/AVC Improvements

As previously mentioned, every implementation is unique and have its own requirements and possibilities. That is why, this section presents few optimization techniques in order to reduce the complexity of the different steps. Some DSP provides libraries with already made functions such as the DCT or quantization function which benefit from hardware mapping optimizations. Some other tools for code optimization can be provided. In the case of Texas instrument, the compiler for TMS320C64 family includes intrinsic functions which allow the programmer to write its own assembly operations and link them to C functions. Code Composer Studio (CCS) provides compilation options for code optimization as shown in Table 3.1. Code Composer Studio also provide tools for memory mapping and cache optimization which are detailed in [29] and [30].

One of the main reasons of the high complexity of some steps is their mode selection, especially for the Prediction steps. As an issue to this problem, [31] proposes a simpler decision process for Intra Prediction. Figure 3.9 illustrate the alternative mode decision for Intra Prediction.

Originally, the H.264/AVC encoder selects the best Intra Prediction mode for $16 \times 16$ block by comparing the Sum of Absolute Transform Differences (SADT) and chose
the mode with the smallest value. The different modes are presented in Figure 2.7. It compares the Rate-Distortion (RD) of the different modes for every $4 \times 4$ block. Then it obtains the MB residuals from the best mode of both $16 \times 16$ and $4 \times 4$ blocks. It calculate the Lagrangian Rate-Distortion cost for both best modes (the one from $16 \times 16$ and $4 \times 4$ blocks) and select the one with the lowest cost which will be the selected mode for Intra Prediction. In the proposed decision mode, the first step which define the best Intra Prediction mode for the $16 \times 16$ block remain the same, but the step for the $4 \times 4$ is divided into two steps in order to ease the Rate-Distortion measurement and comparison process. The first is the Coarse Mode Decision (CMD). It performs a Hadamard transform to get the SADT (which is easier and faster than a DCT transform). As the SADT has a strong correlation with the Rate-Distortion performance, it can be used to determine the implausible modes. Then it sorts out the modes with the smallest values taking into account the Quantization Parameter and the block activity (some residual blocks might have a lot of details while other might be flat). The remaining modes are called candidates. The Number of Candidate (NoC) is adaptive and defined with a lookup table which considers the block activity and the Quantization Parameter. The higher the Number of Candidate is, the more complex the whole decision process will be. The second level is the Fine Mode Decision (FMD). The Fine Mode Decision defines the best mode among the candidates using either the Rate-Distortion model or the Rate-Distortion Optimization (RDO) model which provides a better visual quality and coding rate. When the block residual is small, the RD model error is negligible. Therefore, if the residual block is small, the RD mode is chosen. Otherwise the RDO model is applied. Then the best Intra Prediction mode for $16 \times 16$ and the one for $4 \times 4$ are compared. The proposed Intra Prediction mode selection is five to seven times faster than the original H.264/AVC Intra Prediction mode selection.

The Motion Estimation is the most computational block of a H.264 encoder. As a consequence it is the most investigated block when it comes to complexity reduction. The purpose of Motion Estimation is to find the best match for Inter Prediction. Figure 3.10 illustrates the best match searching. It looks for the best value (the lowest block distortion) at the integer value (here point A), then it repeats this step at the half pixels around the previously selected value. If there is no better value than the previous best match (point A), then the previous best match is selected. However if there is a half-pixel better match, it selects this match and repeat the process for the quarter-pixels.
In our case, first point A is selected, then point B which finally leads to the final best match: point C. The Motion Estimation is computationally expensive because this process is done over several frames to find the best match for Inter Prediction. In order to reduce the computational cost, some encoders use different searching patterns. The widest used are shown in Figure 3.11. Despite the fact that these designs were based on the idea that temporal redundancy is generally expressed in a close area, and even considering that these patterns exploit axial symmetry which is the most logical choice, these patterns are not exhaustive, and as a result it reduces the data compression and increase the bit rate. In order to maximize the complexity reduction and minimize the bit rate increase, some algorithms combine different patterns as in [32] which uses a mixing of a square and a hexagon pattern. Another solution is to change the pattern itself by resizing it as it is the case in [33] which is measuring the performance of a resized diamond pattern.

H.264/AVC offers different customization possibilities through its different features. A way to reduce the complexity of an encoder is to choose carefully which features to use to fit our implementation. [27] analyzed the coding performances of
Figure 3.10: H.264/AVC Motion Estimation pattern
Figure 3.11: Motion Estimation alternative patterns
a H.264/AVC encoder on a GPP. It then measured the encoding performances of the same video samples but with different features. In the end it defines an optimized model for its implementation. Logically a particular attention is given to the Inter Prediction step. The chosen configuration includes the following options:

- The 3 levels with 5 B slices structure is chosen.

- The Hadamard Transform is disabled due to its low coding efficiency gain.

- The UMHexagonS (which is detailed in [32]) is chosen for Motion Estimation and the Quarter-Pixel Accuracy remains by cause of its important bit rate save. However, as the B-frame for Inter Prediction is not available, the number of reference frames is limited to 3 and the $8 \times 8$ block sub-partitioning option is disabled.

- The Rate-Distortion Lagrangian technique, which was mentioned earlier in this subsection, is maintained for Intra Prediction.

- For previously stated reasons, the Context-based Adaptive Variable Length Coding (CAVLC) option is chosen for Entropy Coding.

Other small improvements on Motion Estimation and Quantization and Transform are set out in [34].
4 OMAP 4460 AND SYSLINK/RPMsG : A PROPOSED APPROACH

This chapter proposes a Heterogeneous Platform to implement the different solutions presented in the previous chapter. The chosen platform is the OMAP4460. The first section is an overview of the OMAP4460 platform and its two different cores. The communication and message passing between the cores is the main bottleneck to such implementation. As a result, the software in charge of this task is a critical feature. The provided software which deals with message passing for the OMAP 4 family is RPMsG. RPMsG and its previous versions are detailed in the second section of this chapter.

4.1 Proposed Heterogeneous Platform : OMAP 4460

4.1.1 OMAP 4460

The OMAP 4460 is a heterogeneous platform designed by Texas Instrument and is part of OMAP 4 family. It was design for mobile applications such as mobile gaming, video streaming or video conferencing. Figure 4.1 shows the block diagram of the OMAP 4460. It includes the following components :

- A Dual Cortex A9 (MPU subsystem) which contains two ARM Cortex A9 cores.
- A TMS320DMC64x+ (DSP Subsystem) with the VLIW feature.
- A Dual Core Cortex M3 (Cortex M3 subsystem) MPU which itself includes two Cortex A3 microprocessors.
- An Image and Video Accelerator HD subsystem (IVAHD) with MPEG 1, MPEG 2 and H.264 accelerated functions, encoder and decoder.
- An Imaging Subsystem (ISS) which is the link between the imaging capture device (camera) and the video or image encoding. It is used for everything related to photography and video recording.

- An Audio Back End subsystem (ABE subsystem) which handles the audio processing.

- A 2D/3D Graphic Accelerator subsystem (SGX540 Subsystem) which is used for 2D/3D gaming applications.

- A System Direct Memory which deals with efficient data passing without the backup of the Dual Cortex A9 or the TMS320DMC64x+.

- A Face Detect module (FDID) which can perform face detection on a picture stored in the memory.

- A Display Subsystem which is in charge of screening the content of the memory frames to the LCD screen or the different screen outputs.

Even though we could perform the full encoding process or the decoding process on the Image and Video Accelerator HD subsystem, the purpose of this study is to take advantage of the strength of a heterogeneous platform to improve the encoding or decoding performances of a H.264/AVC encoder or decoder. Therefore, the implementation should be done only on the Dual Cortex A9 and the TMS320DMC64x+.

The parallelization options presented in 3.3.1 can be further improved by the fact that the Dual Cortex A9 can run two processes in parallel, one on each of its core which gives us a platform with two GPP and one DSP.

The Cortex A9 are based on ARM technology which provides several features, some of them actually are DSP’s generic features such as Symmetric MultiProcessor Architecture (which is an architecture with similar processors and a shared memory) or Superscalar Architecture using Single Instruction Multiple Data (SMID). The diagram of Dual Cortex A9 subsystems is given in Figure 4.2. [11] provides more information about the OMAP 4460 and the Dual Cortex A9.
Figure 4.1: OMAP4460 Block Diagram [11]
Figure 4.2: Cortex-A9 MPU Subsystem Overview [11]
4.1.2 TMS320C64x

The TMS320C64x is a DSP designed by Texas Instrument and is a member of the TMS320C6x family which is a family of advanced Very Long Instruction Word DSPs. As shown in Figure 4.3, the TMS320C64x is made of eight functional units, two data paths and two register files. As a result, the TMS320C64x can execute up to eight 32 bit instructions per cycle. With a clock at 1.1GHz, the TMS320C64x can perform around 8800 Million Instructions Per Second (MIPS). The functional units are:

- M1 and M2, the multiplier units which can perform one $16 \times 16$ bit multiplication or four $8 \times 8$ bit multiplication per cycle.
- L1 and L2 which can execute bits shifting, logical operations, min/max operations or arithmetic operations. They can perform one 32 bit arithmetic operation, two 16 bit arithmetic operations or four 8 bit arithmetic operations during one clock cycle.
- S1 and S2 are almost the same as L1 and L2 but they can do saturated arithmetic operations and comparisons. On the other hand they cannot perform min/max operations.
- D1 and D2 can do linear and circular addressing and loading/storing with offset.

As for memory management, the TMS320C64x has two loads from memory (LD1 and LD2) and Store from memory (ST1 and ST2) data paths, two data address paths and two register file data cross paths. The TMS320C64x also have at its disposal a library containing optimized video/image processing functions. More details about this library can be found in [35] and [36]. Further information about TMS320C64x can be found in [37] and [12].

4.2 Syslink and RPMsg

4.2.1 From DSP Gateway to DSP Bridge Project

In June 2003, Nokia release DSP Gateway. This software, developed for the first OMAP generation, permitted to use the internal DSP of these chips. It is made of two
Figure 4.3: TMS320C64x CPU [12]
parts: A Linux driver code on the ARM part and a DSP library. As showed on Figure 4.4, the communication is made through the OMAP mailbox system.

DSP Gateway triggers the DSP only when a task needs to be performed. It also includes other management features [13]:

- Familiar APIs for Linux applications, using conventional Linux device file interfaces.
- Multi-task environment for DSP applications, by virtue of the TI's DSP/BIOS.
- Minimum copy count for maximum throughput in data exchange between the ARM and the DSP.
- Memory extension feature for DSP, i.e. external memory mapping for the DSP space.
- /proc file entries for debugging and run-time environment configurations.
- Power Management functions for DSP.
- Error detection on DSP.
The version 3.3.1 of DSP Gateway includes the second generation of OMAP. The mailbox system is slightly different than the one for the first generation (for more details please refer to the documentation paragraph 2.2 in [13]). One of the main assets of DSP Gateway is that it is Open Source. However, the project is not updated anymore.

DSP Bridge and DSP Link are two pieces of software developed by Texas Instrument for the second and third generation of OMAP chips. Because they are based on the same mailbox system through the shared memory than DSP Gateway, they can be seen as the evolution of this one. They keep the same structure with an ARM side code and a DSP side code as it is shown on figure 4.5.

DSP Bridge also includes most of the DSP Gateway’s features such as power management, DSP execution control or dynamic loading. It goes further than DSP Gateway providing more power management options and different Inter-Processor Communication standards as explained in [38] DSP Link is a lighter version of DSP Bridge, the main difference is that there are no power management tool included, and mainly focuses on the communication features [14]

4.2.2 Syslink

Syslink is the communication software for OMAP 4. The main difference between Syslink and the previous version is that the previous pieces of software were only GPP-DSP oriented. Syslink allows communication between different kinds of core whether
they run a HLOS or a Real Time OS. The only difference will be the architecture of the different modules which will have to cope with the different features and configurations of the operating systems. However, the "skeleton" of the architecture remains the same as you can see on figure 4.6.

As shown Figure 4.7, Syslink keeps almost every feature from DSP Link and adds few extra. The main interesting changes for our topic are the communication system, and two new IPCs : FrameQ and Multi-Processor Heap (please notice that the features in red were not implemented in the first version of Syslink)

About the communication system, DSP Link provided communication between a master core (GPP) running on a HLOS and one or several slave cores running on SYS/-BIOS. Syslink goes further into the communication system and now allows different combinations of master/slave. All the pairs are defined by their IPC and are identified by a system-wide unique name. Those pairs are created regardless of the fact that the core is running a HLOS or a RTOS.

The different modules of Syslink will now be detailed and the differences with their DSP Link versions will be explained. The name in the brackets is the name of the related module in DSP Link. This is a sum of the migration guide [39] and the user guide [15]. For more information about these modules and their functions, please refer to these documents.
Manager and Processor Manager (PROC) The IPC module (also called System Manager) is the module which has to match the OS and platform configuration and the Syslink products. The Processor Manager is the module which manages the slave processor. It boots and loads the slave processor, reads and/or writes to its memory and provides power management tools. In a similar way, PROC module in DSP Link deals with the management of the slave DSP and the synchronization between the DSP and GPP side codes.

IPC modules:
Notify (Notify) The Notify module is a module which sends 32 bit messages notification which can be prioritized. This module is highly related to the platform and the OS used. It has exactly the same function as in DSP Link.

MessageQ (MSGQ) The Message queues are identified by a single system-wide name and provide communication between one reader and one or many writers. The length of the message is not fixed. The main difference between MessageQ and MSGQ is that MessageQ use Heap functionalities (a new module which will be introduced later) while MSGQ use POOL. The other features remain the same.
**RingIO (RingIO)** As every IPC module, each RingIO instance is identified by a single system-wide name. It creates a ring buffer which can be accessed by one reader and one writer and provide different data management options. It is a good solution for audio and video processing. The module offers the same options and mechanism in DSP Link.

**ListMP (MPLIST)** ListMP provides a link list of elements which can be accessed by several readers and writers. It basically works as a First In First Out buffer but there is an API provided to navigate through the list. Again, every list is created with a unique system-wide name.

**GateMP (MPCS)** GateMP goes a step further in multi-processor. It creates instances with a unique system-wide name which is used by the client to access the instance. Every processor can create and/or delete any instance of GateMP, but the access to these memory regions is exclusive. In other words, two clients cannot access the same memory region at the same time.

**Multi-processor Heaps (POOL)** Mutli-processor Heaps is a new Syslink module which provides management and configuration features for the shared memory regions. It is deeply linked to the hardware architecture and the OS used on the different cores. It is based on the IHeap interface of XDC.

**FrameQ** This module provides instances working as queues which can carry video frames from a single writer to several writers. Each queue is identified by a single system-wide name. This is an asset for video decoding as it provides a convenient tool which allows work with video frames.

**Utility modules :**

**SharedRegion** The SharedRegion module deals with the configuration of the shared memory between the processor. It is a support for the IPC modules.

**MultiProc** Multiproc deals with the Processor ID attribution and management.

**List** This module provides local double linked circular list and all the features related
NameServer  The NameServer module uses MultiProc module to identify the processor, to manage and store the name/value pairs with all the management features.

Trace  This module can provide a record of every failures or unexpected behavior.

4.2.3  \textbf{RPMsg}

RPMsg is an evolution of Syslink 2.0 (it is sometimes called Syslink 3.0). It was specifically designed for the OMAP 4th and 5th generation. Like Syslink, RPMsg is multi-core oriented regardless of the type of core. It can be implemented on a multicore platform whose cores are running different OSs. The smallest core can perfectly run with the minimal RPMsg features:

- the device management (remotecore) which provides core management such as loading, bootup, power management, error recovery.

- the messaging framework (rpmsg) which provides the client driver and the messaging protocols.

- the device management is independent but the message framework requires the device management to run on the core to be operational.

[40] provides a summary of RPMsg modules and features which are similar to Syslink’s features. Additional information about RPMsg can be found in [41].
5 CONCLUSION AND FUTURE WORK

5.1 Conclusions

The OMAP 4460 with its Dual Cortex A9 and TMS320DMC64x+ is a good option to run a H.264/AVC High profile or Extended profile due to its computational capacity. Syslink and RPMsg are suitable software which can handle the communication between the two ARM9 cores and the DSP. However this is an expensive solution if we only want to implement a H.264/AVC Baseline profile or Main profile. As stated previously, there are many solutions depending on the parameters of the implementation like the H.264/AVC profile, the platform we want to use. When it comes to mobile technology, the video encoder or the video decoder is not the only program running on the platform which raises other issues like scheduling and priority issues. Each and every case of implementation is unique but this study has given the main guidelines for a parallel implementation.

The two main solutions are Data Level Parallelization which can be improved using 2D wave and 3D wave techniques and Task Level Parallelization which must take into account the computational capacity of the different cores especially the dedicated ones. Few improvements on the Decision Modes for the two prediction modes have also been reviewed in this thesis. Those improvements might help reducing the complexity of these two highly complex steps.

5.2 Future work

With the increase of the screen resolution (2K, 4K and 8K) new standards were needed. In January 2013, the H.265/HEVC (High Efficiency Video Coding) standard was finalized. It is expected to become the main standard of the new video coding generation. It provides more than 50 % bit rate reduction for the high resolutions. Furthermore it provides tool for parallelization which should ease the implementation on hetero-
geneous platforms. The recent popularization of 3D technology, will lead to further extensions, already in development, for the H.265/HEVC standard which will bring new possibilities.

The fifth OMAP Generation (OMAP 5) uses a combination of Dual Core A15 and PowerVR which could be a suitable platform for a H.265/HEVC implementation.
BIBLIOGRAPHY


[5] "blocky artifacts in the highly compressed image at left are reduced when a deblocking filter is applied, as seen in the image at right". URL: http://www.imakenews.com/eletra/mod_print_view.cfm?this_id=1550736&u=kin2&show_issue_date=F&issue_id=000392272&lid=b11&uid=0.


[31] C.-C. Jay Kuo Chang-Sung Kin, Qing Li. "Fast Intra-Prediction Model Selection for H.264 Codec".


[34] Chung-Yu Yang Daw-Tung Lin. "H.264/AVC Video Encoder Realization and Acceleration on TI DM642 DSP".


