A Novel Architecture for Digital Image Compression based on EBCOT Parallel Processing

K.P.S SRAVANTHI¹, SOLOMON J V GOTHAM²

¹PG Scholar, Dept of ECE, Kaushik College of Engineering, Visakhapatnam, Andhra Pradesh–INDIA,
E-mail: sravanti.kps@gmail.com.
²Assoc Prof, Dept of ECE, Kaushik College of Engineering, Visakhapatnam, Andhra Pradesh –INDIA.
E-mail: s_gotham@yahoo.com

Abstract: Digital image processing and compression technologies have significant market potential, especially the JPEG2000 standard which offers outstanding code-stream flexibility and high compression ratio. Strong demand for high performance digital image processing and compression system solutions is forcing designers to seek proper architectures that offer competitive advantages in terms of all performance metrics, such as speed and power. When targeting the JPEG2000 standard, the core tasks such as 2-D Discrete Wavelet Transform (DWT) and Embedded Block Coding with Optimal Truncation (EBCOT) are implemented and optimized on the proposed architecture. A novel 2-D DWT architecture based on vector operations associated with RICA paradigm is developed, and the complete DWT application is highly optimized for both throughput and area. For the EBCOT implementation, a novel Partial Parallel Architecture (PPA) for the most computationally intensive module in EBCOT, termed Context Modeling (CM), is devised. A Ping-Pong memory switching mode with efficiently designed. The project is designed using Verilog HDL Language. Initial input pixel values are obtained from MAT Lab Tool. The functionality of the design is verified using Modelsim Tool and synthesized using Xilinx Tool.

Keywords: EBCOT, Context Modeling, DWT, VERILOG.

I. INTRODUCTION

With the rapid development of computer technologies, digital image processing stands as a pivotal element in people’s life. It has been widely utilized in not only academic and research aspects such as medical image processing and radar image analyzing, but also people’s daily life such as mobile phones and digital cameras. Meanwhile, together with the growth of the Internet technology and portable storage devices, digital image compression techniques are drawing more and more attention with the objective to reduce irrelevance and redundancy of image data in order to store or transmit data in an efficient form [1]. Take a digital camera as an example: usually it employs image processing technologies including demosaicing, Gamma correction, white balancing, smooth filtering, etc. After processing, the obtained digital image is compressed and stored in an SD card or transmitted through internet or other media. The compressed image may be represented in different forms such as TIFF [2], JPEG [3] and GIF [4]. Recently, a newer version of JPEG, termed JPEG2000 [5], has been presented. Based on the wavelet-based method, JPEG2000 compression standard offers significant flexibility and outstanding performance compared with other existing standards. Given these exciting technologies, a question is likely to arise: What is desired for digital image processing solutions in applications such as mobile phones and digital cameras? Obviously, a solution which is able to provide high throughput is highly desirable. Meanwhile, the power-efficient feature is also very important especially targeting those portable applications powered by batteries. Moreover, in advanced digital cameras, a good image processing solution is normally required to have significant flexibility in order to support different algorithms. Generally, an ideal digital image processing solution is expected to have high throughput, strict low power and outstanding flexibility/reconfigurability for various tasks in advanced digital cameras. Project targeting efficient solutions for digital image compression based on EBCOT parallel processing.

II. DIGITAL IMAGE PROCESSING TECHNOLOGIES AND ARCHITECTURES

Digital image processing is the use of computer algorithms to perform processing on digital images. It is a subcategory of digital signal processing which provides many advantages over analog image processing such as wider range of algorithms to select and avoiding buildup of noise and distortion during processing [8]. Digital
image processing has been extremely widely used in fields of digital cameras, remote sensing, multimedia, satellite and so on. A typical digital image processing system can be viewed to be composed of the following modules: a source of input digital image data, a processing module and a destination for the processed image, as illustrated in Figure 1. Usually, the digital image source is provided by a digitization procedure, which means the process of converting an analog image into an ordered array of discrete pixels. This procedure is normally executed by a digital camera, scanner, etc. The processing module in digital image processing is usually a digital processor, which can be a computer or a dedicated chip. The image destination can be realized by different kinds of digital storages and output terminals for transmission. Generally, digital image processing is to apply different processing algorithms onto a matrix consisting of digitized pixels, which is the fact of a digital image. According to different pixel formats, the image can be presented in grayscale or colour.

A. JPEG2000 Compression Standard

This chapter presents the system-level integration and optimization of the JPEG2000 encoder and targeting an efficient data transfer scheme between 2-D DWT and EBCOT, the scanning pattern of the 2-D DWT and AE modules in EBCOT are integrated by a memory relocation module with a carefully designed communication scheme[15]. A Ping-Pong memory switching mode is developed in order to further reduce the execution time. JPEG2000 image compression standard was created by Joint Photographic Experts Group (JPEG) committee in 2000 and introduces by ISO/IEC. Instead of using DCT, it employs DWT technique and supports both lossless and lossy compression schemes. Numerous features are provided in JPEG2000 standard in addition to the basic compression functionality, including 1–5 [14], providing JPEG2000 a very large potential application base.

Figure 2. Illustrates a block diagram of the JPEG2000 encoding algorithm. The original image is decomposed into rectangular blocks termed tiles and codeblocks for processing in order to avoid massive memory usage. The main modules in JPEG2000 encoder are: Component Transform, Tiling, 2-D Demonsion DWT, Quantisation, EBCOT Tier-1 encoder including Context

B. 2-D DWT and EBCOT Integration

It is observed that 2-D DWT and EBCOT have different data processing patterns, as 2-D DWT is line (column) based while EBCOT processes data samples at bit-level within codeblocks. In this case, EBCOT has to wait for 2-D DWT to finish transforming a number of complete lines and columns in order to obtain a full codeblock, as illustrated in Error! Reference source not found. In order to improve the coding efficiency, the 2-D DWT scanning pattern is modified in this work as illustrated in Error! Reference source not found.. Instead of scanning a complete line or column of the image, modified 2-D DWT takes an area of 4 codeblocks as its processing unit at a time. After 2-D DWT, this area is directly transformed to four codeblocks belonging to different subbands. Codeblocks for LH, HL and HH subbands are then coded.
by EBCOT separately, while codeblock for LL subband is reserved and stored for a deeper level transform. When the current DWT finishes, the next four codeblocks become the processing unit. In this way, codeblocks for EBCOT are generated directly without delay of finishing a complete line/column, and the required intermediate storage for 2-D DWT is reduced since only four codeblocks are transformed at a time instead of the entire image.

- The storage space for all the CX/D pairs belonging to a single bitplane of a complete codeblock. Usually the maximum size of a codeblock is 64x64. Considering the extreme condition (all the samples are sign coded), the required storage depth is 2x64x64 = 8192.
- Some reserved space for storing communication variables (less than 32).

Based on the above analysis, a 32K x 32-bit DPRAM is select in this work. This DPRAM can be accessed by both RICA based architecture and ARM via its two ports. Several communication variables are utilised to control the communication between RICA based architecture and ARM in order to avoid memory accessing conflict, and these variables will be introduced.

2. Memory Relocation Module

As discussed in the previous chapter, coded CX/D pairs from CM are generated in parallel within codewords. When referring to the JPEG2000 standard, the following AE module needs to receive CX/D pairs separately

C. CM and AE Integration

1. System Architecture

Since CM and AE modules are implemented separately on two different architectures, a shared DPRAM, which acts as the communication channel between RICA based architecture and ARM core, is utilized to integrate CM and AE. Error! Reference source not found. Illustrates the proposed architecture with DPRAM. As both RICA paradigm and AE are based on 32-bit operand, a 32-bit DPRAM is selected in this work. The depth of the DPRAM can be flexible, but its minimum capacity should be able to satisfy the following requirements:

- bitplane of a complete codeblock generated by CM. Totally the required depth is 3x64x64+64x64 = 16384.
according to different coding passes together with the numbers of these pairs. In this case, CX/D pairs generated by CM need to be derived from codewords and relocated. Due to the memory access and conditional branch restrictions of RICA paradigm, deriving and relocating operations cannot be performed simultaneously with CM. In this work, a module termed Memory Relocation (MR) is added between CM and AE, in order to ensure AE module receives CX/D pairs with the correct order.

The MR module is implemented on RICA based architecture as illustrated in Error! Reference source not found. Given a codeword, MR first splits it and obtains all information provided, and then the magnitude CX/D pair is relocated to the corresponding coding pass storage by MR. After that, MR checks whether the current codeword contains any sign CX/D pair. If yes, the sign CX/D is relocated together with the magnitude CX/D pair. Meanwhile, the CX/D pair count increments whenever a CX/D pair is relocated. The valid state is also utilised by MR in order to derive and relocate RLC coded CX/D pairs correctly.

3. Communication Scheme between CM and MR

In order to reduce the implementation complexity, communication between RICA based architecture and ARM core is directly realized through variables located on specified memory addresses which are listed in Error! Reference source not found. These variables can be accessed by both RICA based architecture and ARM through simple memory operations. Error! Reference source not found. gives a pseudo code illustration of how the integrated EBCOT architecture works [5]. At the beginning of encoding, AE_START signal is initialised to “0” to make sure AE does not work until CX/D pairs of the current bitplane are ready; at the same time MEMORY_READY is set to “1”. On the ARM side, after initialisation, AE_READY is set to “1” indicating that ARM is ready for AE coding.

When coding starts, the index of current bitplane is indicated by BITPLANE_INDEX and is ready to be passed to ARM. RICA checks the shared DPRAM. If it is ready, CM coding starts. After completing CM coding of the current bitplane, RICA checks AEREADY to see whether ARM is


CA
// Initialise
set AE_START = 0; // AE only starts when relocated CX/D pairs are ready
set MEMORY_READY = 1; // shared DPRAM is initialised ready for use
for (k=0; k <bitdepth; k++) // loop for bitplanes
{  
  BITPLANE_INDEX = k; // index the bitplane
  do{} while (MEMORY_READY == 0); // waiting for DPRAM ready
  // RICA based architecture processing starts
  RICA based architecture processing; // CM
  // processing finishes
  do{} while (AE READY == 0); // waiting for ARM finishing
  // Memory relocation starts
  set MEMORY READY = 0;
  MR starts:
  set MEMORY READY = 1; // indicating relocated CX/D pairs are ready
  set AE START = 1; // starting ARM
}

ARM:
// Initialise
set AE READY = 1;
while (BITPLANE_INDEX <bitdepth)
{
  // waiting for relocated CX/D pairs ready and the
  // starting signal from RICA
  do{} while ((MEMORY READY == 0) or (AE_START==0));
  // ARM processing starts
  set AE READY = 0;
  ARM processing; // AE
  set AE READY = 1;
  // wait until MR of the next bitplane starts
  do{} while (MEMORY READY == 1);
}

Ready for AE. If yes, MR is executed with MEMORY READY being set to “0” in order to avoid any unexpected access to the shared DPRAM. When MR finishes, both MEMORY READY and AE_START are set to “1” to start AE on ARM.

On the ARM side, ARM checks the BITPLANE_INDEX regularly until finishing the entire codeblock. AE does not start until both MEMORY READY and AE_START signals are “1” to ensure it is safe to access the shared DPRAM. The AE READY variable is set to “0” at the time AE starts

<table>
<thead>
<tr>
<th>CM Bitplane 0</th>
<th>MR Bitplane 0</th>
<th>AE Bitplane 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage 1</td>
<td>Stage 2</td>
<td>Stage 3 &amp; stage 1</td>
</tr>
<tr>
<td>Stage 3</td>
<td>CM Bitplane 1</td>
<td>MR Bitplane 1</td>
</tr>
<tr>
<td>Stage 2</td>
<td>AE Bitplane 1</td>
<td>MR Bitplane 2</td>
</tr>
</tbody>
</table>

Figure 6. Pipeline Structure of the JPEG2000 Encoder

in order to indicate that ARM is busy. When AE coding finishes, AE READY is set to “1” again and ARM is put to wait state and jumps back to the beginning of bitplane loop only when the next MR starts.
One of the benefits of this working mode is that it is possible to pipeline between RICA based architecture and ARM core during the coding process. Excluding 2-D DWT which is executed at beginning of the coding procedure, the JPEG2000 encoder on the proposed architecture is considered to be composed of three stages: CM, MR and AE. Since the MR module acts as the intermediate between CM and AE, a 3-stage pipeline is established with which CM and AE can be executed in the same time slot. Error! Reference source not found. Illustrates this pipeline architecture. As described in Error! Reference source not found., four communication variables take charge of controlling the pipeline structure. By strictly controlling accesses to the shared DPRAM and the starting time of different coding engines, the pipeline architecture offers significant improvement in system performance[15].

5. Ping-Pong Memory Switching Scheme

Based on the above discussion, core tasks in JPEG2000 encoder are implemented and optimized on the proposed architecture. A performance evaluation is performed with the standard 256x256 grayscale Lena test image. Error! Reference source not found. illustrates the execution time ratio of different modules for encoding the entire image (codeblock size = 64x64, 5/3 DWT, 1-level). It is seen that the MR module takes 33% of the overall execution time and become the system bottleneck. In other words, if a more efficient pipeline scheme can be established which ensures that MR module can also be executed simultaneously with other modules instead of only pipelining CM

![Figure 7. Execution Time Ratio of Different Modules in JPEG2000 Encoder](image)

Figure 2. Ping-Pong Memory Switching Architecture and AE, the system performance will be significantly improved. In this work, another memory block with the same size of the CX/D pair storage space in the shared DPRAM is employed to construct a Ping-Pong memory switching scheme, as illustrated in Error! Reference source not found.

These two memory blocks in the shared DPRAM are accessed alternately by both RICA based architecture and ARM core. When CM and MR finish coding the 1st bitplane, CX/D pairs are stored in memory block A and then coded by AE. At the same time the 2nd bitplane is coded by CM and MR and stored in memory block B. After that, AE fetches CX/D pairs from memory block B to code the 2nd bitplane, meanwhile CM and MR switch to memory block A again for the next bitplane and so until the complete codeblock is coded. In this way, CM and MR are executed in the same time slot with AE, leading to further execution time reduction. When the Ping-Pong memory switching mode is applied, the DPRAM illustrated in Section 7.3.1 can be replaced by a similar DPRAM with different address space for the two Ping-Pong data blocks or even a 4-port RAM. Obviously, the capacity for storing CX/D codewords and relocated CX/D pairs need to be doubled.

III. DESIGN & IMPLEMENTATION OF 2-D DWT

This paper presents a reconfigurable lifting-based 2-Dimensional Discrete Wavelet Transform engine for JPEG2000 [13] to support both 5/3 and 9/7 transform modes for lossless and lossy compression schemes in JPEG2000 standard. VO with SIMD technique is utilised in the proposed 2-D DWT engine and the advantages and disadvantages brought by VO are well discussed. Simulation results [13] demonstrate that the proposed 2-D DWT engine provides high throughput that reaches up to 103.1 Frames per Second (FPS) for a 1024x1024 image.
A. JPEG2000 Encoding Standard

1. Tiling and DC Level Shifting

The first preprocessing step in JPEG2000 standard is tiling, which partitions the original image into a number of rectangular non-overlapping blocks, termed tiles. Each tile has the exact same colour component as the original image. Tile sizes can be arbitrary and up to the size of the entire original image. Generally, a large tile offers better visual quality to the reconstructed image and the best case is to treat the entire image as one single tile (no tiling). However, a large tile also requires more memory space for processing. Typically, tiles with the size of 256x256 or 512x512 are considered to be popular choices for various implementations based on the evaluation of cost, area and power consumption [17].

2. Component Transformation

Component transformation is effective on reducing correlations amongst multiple components in the image. Normally, the input image is considered to have three colour planes (R, G, B). JPEG2000 standard supports two different transformations: (1) Reversible Colour Transformation (RCT) and (2) Irreversible Colour Transformation (ICT). RCT can be applied to both lossless and lossy compression, while ICT can only be used in the lossy scheme [17]. In the lossless mode with RCT, pixels can be exactly reconstructed by inverse RCT. The forward and inverse transformations are given by:

Forward RCT:
\[
Y_r = \left[\frac{R + 2G + B}{4}\right]
\]
\[
U_r = B - G
\]
\[
V_r = R - G
\]

Inverse RCT:
\[
G = Y_r - \left[\frac{U_r + V_r}{4}\right]
\]
\[
R = V_r + G
\]
\[
B = U_r + G
\]

ICT is only applied for lossy compression because of the error introduced by using non-integer coefficients as weighting parameters in the transformation matrix [7]. Different from RCT, ICT uses \(Y, C_r, C_b\) instead of \(YUV\), in which \(Y\) is the luminance channel while \(Cr\) and \(Cb\) are two chrominance channels. The transformation formulas are given by:

Forward ICT:
\[
\begin{bmatrix} Y_r \\ C_r \\ C_b \end{bmatrix} = \begin{bmatrix} 0.299000 & 0.587000 & 0.114000 \\ 0.500000 & -0.418688 & -0.081312 \\ -0.168736 & 0.331264 & 0.500000 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}
\]

Inverse ICT:
\[
\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0 & 0.0 & 1.402000 \\ 1.0 & -0.344136 & -0.714136 \\ 1.0 & 1.772000 & 0.0 \end{bmatrix} \begin{bmatrix} Y_r \\ C_r \\ C_b \end{bmatrix}
\]

3. Quantization

Quantization of DWT coefficients is one of the main sources of information loss in JPEG2000 encoder. In lossy compression mode, all the DWT subbands are quantized in order to reduce the precision of DWT subbands to aid in achieving compression [27]. The quantization is performed by uniform scalar quantization with dead-zone around the origin. As illustrated in Error! Reference source not found., step size of the dead-zone scalar quantize is set to be \(\Delta_d\) and the width of the dead-zone is \(2\Delta_d\). The formula of uniform scalar quantization with a dead-zone can be given by

\[
\psi_{a,b}(t) = \frac{1}{\sqrt{a}} \psi\left(\frac{t-b}{a}\right)
\]

Where \(a\) is the scaling factor and \(b\) represents the shifting parameter.

Based on this definition of wavelets, the wavelet transform of a function \(f(t)\) can be mathematically represented by

\[
W(a,b) = \int_{-\infty}^{\infty} \psi_{a,b}(t)f(t)dt
\]

When targeting discrete signals, DWT can be considered to convolve the input discrete signal with two filter banks, one for low pass and the other is high pass. The two output streams are then down-sampled by a factor of 2. The transforms are given by [12]

\[
W_L(n) = \sum_{i=0}^{\tau_L-1} h(i)f(2n - i)
\]

\[
W_H(n) = \sum_{i=0}^{\tau_H-1} g(i)f(2n - i)
\]

Where \(\tau_L\) and \(\tau_H\) are taps of the low-pass \((h)\) and the high-pass \((g)\) filters. After the transform, the original input signal is decomposed into two subbands: lower band and higher band. Practically, the lower band can be further decomposed for different resolutions. The architecture is illustrated in .

\[
\begin{array}{c}
\text{X(z)} \\
\downarrow \\
\text{LPF} \\
\downarrow \\
L \\
\downarrow \\
\text{LPF} \\
\downarrow \\
\text{LL} \\
\end{array}
\]

\[
\begin{array}{c}
\text{X(z)} \\
\downarrow \\
\text{HPF} \\
\downarrow \\
H \\
\end{array}
\]

\[
\begin{array}{c}
\text{X(z)} \\
\downarrow \\
\text{LPF} \\
\downarrow \\
L \\
\downarrow \\
\text{HPF} \\
\downarrow \\
\text{LL} \\
\end{array}
\]

\[
\begin{array}{c}
\text{X(z)} \\
\downarrow \\
\text{LPF} \\
\downarrow \\
L \\
\downarrow \\
\text{LPF} \\
\downarrow \\
\text{LL} \\
\end{array}
\]

\[
\begin{array}{c}
\text{X(z)} \\
\downarrow \\
\text{HPF} \\
\downarrow \\
H \\
\end{array}
\]
A Novel Architecture for Digital Image Compression based on EBCOT Parallel Processing

The original image

![Discrete Wavelet Transform Diagram](image)

Figure 9. Discrete Wavelet Transform

For digital image processing, it is essential to have 2-dimensional DWT to perform the transformation of a 2-D image. The approach for 2-D DWT is to implement a 1-D DWT at the horizontal direction first and then another 1-D DWT along the vertical direction is performed. After a 2-D transform, four subbands are generated, which are LL, LH, HL and HH respectively. LL is a coarser version of the original input image, while LH, HL and HH are high-frequency subbands containing the detail information [27].

IV. DESIGN & IMPLEMENTATION OF EBCOT PARALLEL PROCESSING

A. Embedded Block Coding with Optimal Truncation

Physically the quantized wavelet coefficients is compressed by the entropy encoder in each code block in each subband [17]. The complete entropy encoding in JPEG2000 standard can be divided into two coding steps: the EBCOT [5] algorithm is adopted, which is composed of fractional bit-plane coding (Context Modeling) and binary arithmetic coding (Arithmetic Encoding). In CM coding, code blocks are encoded separately within bit-level. Given the precision of quantized DWT coefficients is \( p \), a code block will be decomposed into \( p \) bit-planes which are then coded sequentially from the Most Significant Bit-plane (MSB) to the Least Significant Bit-plane (LSB). Each coefficient is divided into one sign bit and several magnitude bits. Context modeling is applied on each bit-plane of a code block to generate intermediate data in the form of a pair of Context and binary Decision (CX/D); while arithmetic encoding codes these CX/D pairs and generates the final compressed bit-stream.

The detailed CM algorithm has been discussed in earlier. In JPEG2000 applications, EBCOT usually consumes most of the execution time (typically more than 50%) in software-based implementations[19] and CM is considered to be the most computationally intensive unit in EBCOT. Since CM adopts the fractional bit-plane coding idea and codes DWT coefficients in code blocks by three separate coding passes in bit-level, it is actually more suitable for specialised hardware implementation rather than general hardware. There have been several methods proposed to accelerate CM process, which are detailed as follows.

B. Primitive Coding Schemes in CM

In CM, the RLC coding scheme which may generate various numbers of CX/D pairs within a single stripe column. all CXs are generated depending on different combinations of the significant states/refinement states/magnitudes of the current bit and its eight neighbours, so the latter two challenges become critical. In order to overcome the above challenges, all the four primitive coding schemes in CM are carefully designed and implemented on desired architecture.

C. Context Modeling in EBCOT

The EBCOT CM algorithm has been built to exploit asymmetries and redundancies within and across bit-planes so as to minimize the statistics to be maintained and minimize the coded bit-stream that it would generated [17]. Before presenting detailed illustration of the CM algorithm, there are several concepts need to be clarified as follows:

1. Sign Array (\( \chi \)): \( \chi \) is a two-dimensional array representing signs of DWT coefficients in a codeblock. Each element \( \chi[m,n] \) in \( \chi \) represents the sign information of the corresponding sample \( y[m,n] \) in the codeblock as follows:

   \[
   \chi[m,n] = \begin{cases} 
   1 & \text{if } y[m,n] < 0 \\
   0 & \text{else} 
   \end{cases} 
   \]

   \[ (13) \]

2. Magnitude Array (\( \nu \)): \( \nu \) is a two-dimensional array consisting of unsigned integers. It has the same size with \( \chi \) so as the corresponding codeblock. Each sample \( y[m,n] \) in \( \nu \) represents the absolute value of the corresponding DWT coefficient, which is given by

   \[
   \nu[m,n] = |y[m,n]| 
   \]

   \[ (14) \]

   Where \( p \) represents the \( p^{th} \) bit-plane.

3. Scanning Pattern: EBCOT has a certain scanning pattern which is based on every four lines of coefficients, termed stripe. The scanning pattern within a stripe is from up to down in a column and from left to right for different columns, as illustrated in Error! Reference source not found. (a). Zero Coding. Zero Coding (ZC): In zero coding, CX is generated from three pre-defined Look-Up Tables (LUTs) for different DWT subbands (LH, HL and HH). Outputs of these LUTs depend on significant states associated to neighbours of the current sample, which are illustrated in Error! Reference source not found. [5]. The decision bit is equal to the current magnitude bit of the corresponding coefficient which is being coded. This coding scheme is used both in SPP and CUP.

As discussed in Chapter 2, ZC generates a CX according to sums of significant states of the current bit and its eight neighbours by horizontal/vertical/diagonal directions as well as which DWT subband the current codeblock belongs to. In the proposed ZC implementation, all the H/V/D sums are judged by comparators and the output CX
is generated through a sequence of multiplexers, whose selecting inputs are the outputs of the comparators. Figure illustrates the detailed circuit structure of this CM coding engine. The comparator array compares the H/V/D values with the different parameters defined in Table2 (ZC LUT table) in Chapter II. Compared results are used as judgments for H/V/D contributions. The logic combination block combines these judgments with logic operations according to the ZC LUT table and generates decisions for the multiplexer sequence to decide the final CX value. In this ZC coding unit, CX has an initial value (normally zero), and the multiplexer sequence chooses the final CX value by utilizing decisions provided by the logic combination block. In this way, conditional branches can be totally eliminated and the ZC coding unit can be integrated within a kernel, with a fair trade-off of a number of extra multiplexers. For different DWT subbands, various parameters for the comparator array and new logic combinations in the logic block are utilized without modifying the coding unit architecture. Figure10. Detailed Architecture for ZC Unit

D. Sign Coding
Sign coding scheme utilizes sign bits and significant bits of the current bit and its horizontal/vertical neighbours to calculate the required H/V

Figure11. Detailed Architecture for SC Unit

Similar as the ZC unit, multiplexer sequences for both horizontal and vertical contributions generate the H and V contributions respectively without breaking the potential kernel. These two contributions are further used to generate decisions for CX and XOR bit. According to Error! Reference source not found. in Section II, conditions for generating XOR bit can be simplified as shown in Table2. With these simplified conditions, two logic combination blocks are employed to generate final decisions for CX and XOR bit respectively, which are utilized by another two multiplexer sequences in order to obtain the final CX and XOR bit. The Decision bit is generated by a simple XOR operation between the current sign bit and the XOR bit.

Table2. Simplified LUT for XOR Bit

<table>
<thead>
<tr>
<th>Combinations of H/V Contributions</th>
<th>XOR bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>H = 1, V = x (x means don’t care)</td>
<td>0</td>
</tr>
<tr>
<td>H = 0 and V &gt; 0</td>
<td>0</td>
</tr>
<tr>
<td>H = 0 and V = -1</td>
<td>1</td>
</tr>
<tr>
<td>H = -1</td>
<td>1</td>
</tr>
</tbody>
</table>

E. Magnitude Refinement Coding
MRC requires accumulation of significant states of the current bit’s eight adjacent neighbours and the information indicating whether the current bit has been coded by MRC. Compared with ZC and SC, MRC implementation is relatively simple as illustrated in Figure12. There are totally three comparators and three multiplexers utilised in the coding unit in order to eliminate conditional branches.
Figure 12. Detailed Architecture for MRC Unit

Magnitude Refinement Coding (MRC): This coding scheme is particularly used in MRP. In MRC, context is determined by whether the current bit is the first refinement bit, which says, the first time to be coded by MRP, in the corresponding coefficient. Also significant states of the current bit’s eight neighbours are taken into consideration. The referenced LUP is given in Error! Reference source not found. [5]. The Decision bit is simple equal to the magnitude bit. Run Length Coding

Run Length Coding (RLC): This coding scheme is only used in CUP. It generates one or more CX/D pairs by coding from one to four consecutive bits within a stripe [17]. Generally the number of CX/D pairs generated is determined by where the first “1” bit is located in the corresponding stripe column. There are two contexts adopted in RLC: 17 and 18. When all the four bits in the stripe column are zero, a CX/D pair (17,0) is generated. In the case there are one or more “1” bits existing, firstly a CX/D pair (17,1) is generated, indicating this is a non-zero stripe column. If there are two CX/D pairs (18, 0/1) and (18, 0/1) are produced, in which the two decision bits actually represent the location of the first “1” bit in this four-bit non-zero stripe column. When describing the complete CM coding progress, these three coding passes are applied to each bit-plane of a codeblock from the MSB to the LSB. As the first bit-plane (MSB) actually has no significance at the beginning, only CUP is applied on it.

After finishing the first bit-plane, the next bit-plane turns up and these three coding passes scan and code it in order of SPP, MRP and CUP, with the scanning pattern illustrated in Error! Reference source not found. (a). RLC presents a more difficult problem as it may generate various numbers (1 or 3) of CX/D pairs according to the four bits in the stripe column. Due to the memory access and conditional branch restrictions in RICA paradigm, an efficient implementation has to ensure that this variation does not break the potential kernel. In this work, we managed to realise the RLC unit within a kernel by combining the generated CX/D pairs into a single codeword that can be read or write by a single memory operation, which is demonstrated in Figure. The codeword occupies 14 bits totally, which is constructed by two modified CX/D pairs in each of which the decision bit is expanded to two bits. For a zero stripe column coded by RLC, only one CX/D pair (17, 0) needs to be generated, and the two parts of the codeword are both filled with it. When coding a non-zero stripe column, firstly a CX/D pair (17, 1) is generated, which fills the highest 7 bits of the codeword. After that, another two CX/D pairs (18, 0 or 1), (18, 0 or 1) are generated and stored in the lowest 7 bits of the codeword by combining the two decision bits together. Accordingly to the different contents, the 14-bit codeword is actually represented in decimal as shown in Figure. Assume the four bits in a stripe column are b0, b1, b2, b3 (from top to bottom), a weight factor is employed to indicate the position of the first non-zero bit in the stripe column with which the codeword can be assigned with correct value, as illustrated in the figure 13.

F. Arithmetic Encoder in EBCOT

The top-level architecture and details of the key sub-modules in AE have been introduced in Section II. It is observed that the encoding algorithm is composed of frequent conditional branches and simple operations. These conditional branches will split potential kernels into separate steps, which will dramatically increase the configuration latency and extend the execution time. On the other hand, if we force the application to be executed in a single kernel, that is, to employ massive comparators and multiplexers to eliminate branches like the way CM is implemented, the kernel will have a long critical path even after being pipelined due to the serial nature of AE.

V. RESULTS

A. TOP_CGRA_MODULE
The implementation of EBCOT, which is the most challenging module in JPEG2000. When looking into the EBCOT CM algorithm and existing solutions, it was found that the current solutions were not suitable for RICA based architecture as they required either frequent conditional branches or massive computational resources. In this case, the novel PPA solution for CM in EBCOT was developed specially for RICA based applications. The required computational resource by the proposed PPA solution was almost only a half of the traditional PPCM method, while the processing speed of PPA was actually higher than PPCM when mapped onto RICA architecture. On the other hand, simulation results demonstrated that RICA based architecture is not a good solution for AE since the frequent branches strictly limited the performance, and this is the reason why an embedded ARM core was selected for AE implementation. RICA based architecture can provide good performance for computationally intensive applications with inherent parallelism.

B. 2D-DWT MODULE

The 2-D DWT engine was optimized by Hardwired Floating Coefficient Multipliers and SIMD based VO technique. Positives and negatives of VO technique was discussed in aspects of both throughput and area occupation in detail. The proposed 2-D DWT engine can reach up to 103.1 fps for a 1024x1024 image.

C. CONTEXT MODELING MODULE

CM is developed which makes good balance between throughput and area occupation for RICA based implementations.

VI. CONCLUSIONS

Based on the work in this thesis, it is concluded that RICA paradigm can provide good solutions for image processing applications. In this thesis, RICA paradigm’s potential and advantage for different imaging tasks was clearly investigated and evaluated. Various optimization approaches for RICA based applications including customization, kernel construction, VO technique utilization, parallel processing and hybrid architecture development were well performed and discussed. It was also demonstrated that it is possible to build up a multicore RICA based architecture for complex applications. Meanwhile, performance comparisons of different imaging tasks between RICA based architecture and other platforms were evaluated in detail. With this presented work, other developers can evaluate a given algorithm to future work in which the work in this paper can be further investigated. When processing different images, the possible bit-depth increment of 2-D DWT coefficients should be taken into account. With the current Lean image, the bit depth used for CM is fine. However there might be 1 or 2 bits increment for other images especially in the case more than 3 levels of 2-D DWT is applied. In this case, the number of bit-level iteration in CM also needs to be increased.

VII. REFERENCES


A Novel Architecture for Digital Image Compression based on EBCOT Parallel Processing


