A Flexible, Line-Based JPEG2000 Decoder for Digital Cinema

***Abstract*— The image compression standard JPEG2000 proposes a large set of features, useful for today's multimedia applications. Unfortunately, its complexity is greater than older standards. Real-time applications such as Digital Cinema requires a specific hardware implementation. In this paper, a decoding scheme is proposed with two main characteristics. First, the complete scheme takes place in an FPGA without accessing any external memory, allowing integration in a secured system. Secondly, a customizable level of parallelization allows to satisfy a broad range of constraints, depending on the signal resolution.**

***Index Terms*— JPEG2000, line-based, arithmetic coding, bit-plane coding, wavelet transform, Digital Cinema.**

I. INTRODUCTION

DEVELOPMENT and diversification of computer networks as well as emergence of new imaging applications have highlighted various shortcomings in actual image compression standards, such as JPEG. The lack of resolution or quality scalability is clearly one of the most significant drawbacks. The new image compression standard JPEG2000 [1] enables such scalability : according to the available bandwidth, computing power and memory resources, different resolution and quality levels can be extracted from a single bit-stream. In addition to this, the JPEG2000 baseline (part I of the standard) also proposes other important features: good compression efficiency, even at very low bit rates, lossless and lossy compression using the same coder, random access to the compressed bit-stream, error resilience, region-of-interest coding. A comprehensive comparison of the norm with other standards, performed in [2], shows that from a functionality point of view JPEG2000 is a true improvement.

The techniques enabling all these features are a wavelet transform (DWT) followed by an arithmetic coding of each subband. The drawback of these techniques is that they are computationally intensive, much more for example than a cosine transform (DCT) followed by an Huffman coding, which are those used in JPEG [2]. This complexity can be a problem for real-time applications.

Digital Cinema is one of these real-time applications. As explained in [3], edition, storage or distribution of video data can largely take advantage of the JPEG2000 feature set. Moreover, a video format named Motion JP2000 has been designed, which encapsulates JPEG2000 frames and enables synchronization with audio data [4]. Nevertheless, a high output rate is required at the decoding process and in order to meet this real-time constraint, a dedicated implementation of the most complex parts of the algorithm is needed.

In this paper, a complete JPEG2000 decoder architecture intended for video decoding is proposed. It has been implemented in VHDL and synthesized in an FPGA (Xilinx XC2V6000 [5]). It takes about 90% of the chip and the estimated frequency of operation is 90 Mhz. The proposed architecture decodes images line by line without accessing any external memory. It is highly parallelized and depending on available hardware resources, it can easily be adapted to satisfy various formats, from Digital Cinema to Video-on-Demand, and specific constraints like secure decoding, lossless capabilities, and higher precision (over 8 bits per pixel).

The rest of the paper is organized as follows. Section II briefly describes the JPEG2000 algorithm. In Section III, we present our decoder architecture as well as our implementation choices. The main blocks of the architecture are described more in details in Sections IV to VI. The performance of the system is discussed in Section VII and the paper is concluded in Section VIII.

II. JPEG2000 OVERVIEW

In this Section, concepts and vocabulary useful for the understanding of the rest of the paper are presented. For more details, please refer to [1] or to [6]. Although a *de*coder architecture was achieved, *en*coding steps are explained here because their succession is easier to understand. Decoding process is achieved by performing these steps in the reverse order. Figure 1 presents the coding blocks which are explained below.

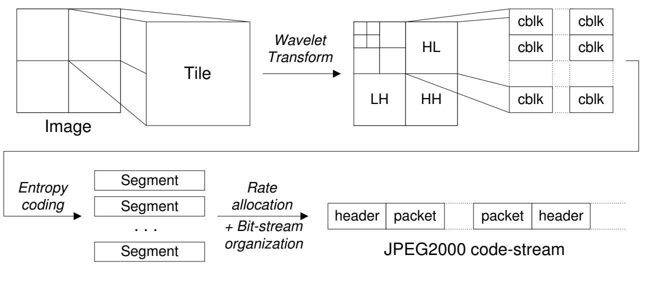

Fig. 1. Coding steps of the JPEG2000 algorithm.

First of all, the image is split into rectangular blocks called tiles. They will be compressed independently from each other. An intra-components decorrelation is then performed on the tile: on each component a *discrete wavelet transform* is carried out . Successive dyadic decompositions are applied. Each of these uses a bi-orthogonal filter bank and splits high and low frequencies in the horizontal and vertical directions into four subbands. The subband corresponding to the low frequencies in the two directions (containing most of the image information) is used as a starting point for the next decomposition, as shown in Fig. 1. Two filter banks may be used : either the *Le Gall* (5,3) filter bank prescribed for lossless encoding or either the *Daubechies* (9,7) filter bank for lossy encoding.

Every subband is then split into rectangular entities called code-blocks. Each code-block will be compressed independently from the others using a *context-based adaptative entropy coder*. It reduces the amount of data without losing information by removing redundancy present in the binary sequence. "Entropy" means it achieves this redundancy reduction using the probability estimates of the symbols. Adaptivity is provided by dynamically updating these probability estimates during the coding process. And "context-based" means the probability estimate of a symbol depends on its neighborhood (its "context"). Practically, entropy coding consists of

- *Context Modeling* : the code-block data is arranged in order to first encode the bits which contribute to the largest distortion reduction for the smallest increase in file size. In JPEG2000, the Embedded Block Coding with Optimized Truncation (EBCOT) algorithm [7] has been adopted to implement this operation. The coefficients in the code-block are bit-plane encoded, starting with the most significant bit-plane. Instead of encoding the entire bit-plane in one coding pass, each bit-plane is encoded in three passes with the provision of truncating the bit-stream at the end of each coding pass. During a pass, the modeler successively sends each bit that needs to be encoded in this pass to the Arithmetic Coding Unit described below, together with its context.
- *Arithmetic Coding* : the modeling step outputs are entropy coded using a MQ-coder, which is a derivative of the Q-coder. According to the provided context, the coder chooses a probability for the bit to encode, among predetermined probability values supplied by the JPEG2000 Standard and stored in a look-up table. Using this probability, it encodes the bit and progressively generates codewords, called segments.

During the *rate allocation* and *bit-stream organization* steps, segments from each code-block are scanned in order to find optimal truncation points to achieve various targeted bit-rates. Quality layers are then created using the incremental contributions from each code-block. Compressed data corresponding to the same component, resolution, spatial region and quality layer is then inserted in a packet. Packets, along with additional headers, form the final JPEG2000 code-stream.

III. Proposed architecture

In this section, we first present the constraints we used for our JPEG2000 decoder architecture. Implementation choices made in order to meet these constraints are then explained. Finally, the complete architecture is presented.

A. Constraints

As our decoder is designed for real-time video processing, three main constraints have been identified :

- *High output bit-rate* : all implementation choices have been made in order to increase this bit-rate. With the Xilinx XC2V6000 used, we wanted our architecture to satisfy at least the 1080/24p HDTV format. This means an output rate of about 1200 megabits per second (Mbps) for 8-bit 4:4:4 images.
- *Security* : no data flow may transit outside of the FPGA if it is not crypted or watermarked. This constraint enables a completely secured decoding scheme, as the decompression block might be inserted between a decryption block and a watermark block, all these three blocks being in the same FPGA (Fig. 2).
- *Flexibility* : computationally intensive parts of the decoding process must be independent blocks which can easily be duplicated and parallelized. This allows the proposed architecture to satisfy a broad range of output bit-rate constraints and therefore to be easily adapted to upcoming Digital Cinema standards.

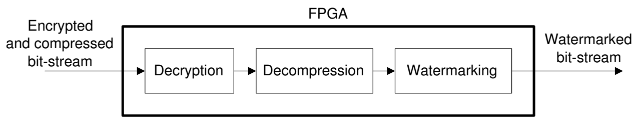

Fig. 2. A secured decoding scheme.

B. Implementation choices

To meet these constraints, the following implementation choices have been made.

No external memory has been used which meets the security constraint and also increases the output bit-rate, as the bandwidth outside the FPGA is significantly slower than inside. As internal memory resources are limited, large image portions cannot be stored and the decoding process must be achieved in a line-based mode.

In order to increase the output bit-rate, three *parallelization* levels have been used. The first one is a duplication of the entire architecture which allows various tiles to be decoded simultaneously. The second parallelization level tries to compensate the compute load difference between the entropy decoding unit (EDU) and the inverse wavelet transform (IDWT). The EDU is indeed much more complex than the IDWT and must therefore be parallelized. This is possible as each code-block is decoded independently from the others. Finally, a third level of parallelization, known in the JPEG2000 standard as the parallel mode, is obtained inside each EDU. By default, each bit-plane is decoded in three successive passes but specifying some options ([8], p.508) during the encoding process makes it possible to decode the three passes simultaneously. This implies that each EDU contains one Context Modeling Unit (CMU) and three Arithmetic Decoding Units (ADU).

Another option specified during the encoding process that increases the output bit rate of the decoder is the *bypass mode* ([8],p.504). The more correlated the probability estimates of the bits to encode are, the more efficient the ADU is. This is especially the case in the most significant bit-planes while the last bit-planes are most of the time totally uncorrelated. With the bypass mode enabled, these last bit-planes are therefore raw-coded[1].

[1]This means they are inserted "as is" in the bit-stream.

Some choices about *image partitioning* have also been made. A 512x512 tile partition avoids an external memory use and enables one of the parallelization level mentioned above. Inside each tile, even if the code-block maximum size specified in the norm is 4096 pixels, code-blocks in our implementation do not exceed 2048 pixels. As we shall see, this implies no significant efficiency loss but allows a 50% memory resources saving. Furthermore, the code-block dimensions have been chosen so that each of them systematically covers the width of the subband to which it belongs (Fig. 3). As the IDWT processes the subband data line by line, such code-block dimensions enables a line-based approach of the overall process, reducing the size of the image portions to store between EDU and IDWT.

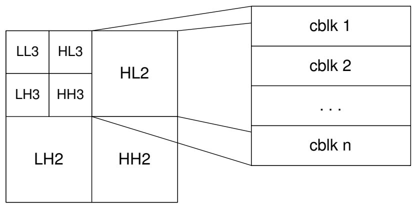

Fig. 3. Customized code-block dimensions.

These last implementation choices (parallel mode, bypass mode and image partitioning) imply an efficiency loss during the encoding process. Table I shows the corresponding *psnr* losses for various compression ratio. In comparison to the improvements provided by these choices, quality losses are quite reduced, especially for small ratios which are the ones used in the targeted applications.

TABLE I
AVERAGE PSNR FOR A SET OF IMAGES 1920X1080, 8 BPP

Compression ratio	**PSNR [dB]**	
	Default options	**Options used**
1:50	37,36	35,83 (-1,53)
1:25	40,10	38,74 (-1,36)
1:10	43,25	42,40 (-0,85)
1:5	45,85	45,31 (-0,54)

Another choice has to be made in order to enable a line-based processing of the image. To reconstruct one line of the original image, the IDWT needs the corresponding line at each resolution. In order to minimize the image portions size to store, data inside the bit-stream is organized so that the whole compressed data corresponding to a specific spatial region of the image is contiguous in the bit-stream. Various progression orders are allowed during the JPEG2000 encoding process and one of them enables such kind of feature.

A last implementation choice aims at achieving some lightweight operations in software. These operations are indeed essentially data handling and are easily implemented using pointers in C. To keep the decoding process secure, headers and markers (needed by these operations) are not crypted and only the packet bodies are.

As it can be seen, some options, known by any universal JPEG2000 encoder, must be specified during the encoding process. Our architecture is unable to decode a JPEG2000 code-stream that has not been encoded using these options. As this architecture is dedicated to decode video data at the highest output bit-rate, we did not consider it efficient to realize a universal decoder.

C. Architecture

Figure 4 presents the hardware part of our architecture. Each EDU contains three ADU's which reflects the parallel mode. The bypass mode is also illustrated by the bypass line under each ADU. The Dispatch IN and OUT blocks are used to dissociate the entropy decoding step from the rest of the architecture and enable the flexibility mentioned above. When Dispatch IN receives a new JPEG2000 code-stream from the PCI, it chooses one of the free EDU's and connects the data stream to it. Dispatch OUT retrieves decompressed data from each EDU and connects it to the correct subband FIFO. In this way, a maximum of EDU's is always used simultaneously. ADU, CMU and IDWT blocks are explained below more in details.

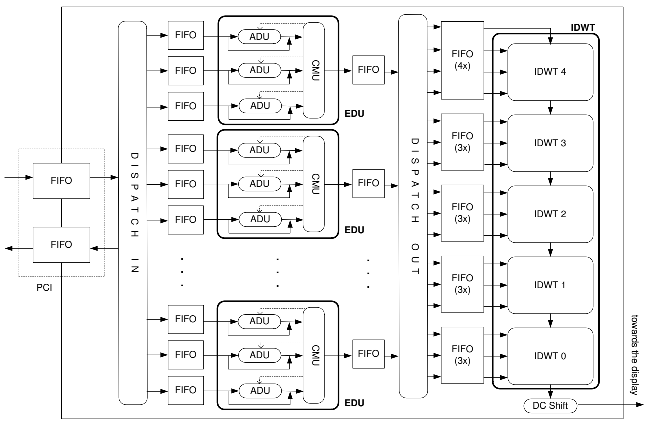

Fig. 4. Proposed architecture.

IV. CONTEXT MODELING UNIT

A. EBCOT algorithm

In the EBCOT algorithm, each code-block is encoded along its bit-planes, beginning with the most significant one. Each bit-plane is encoded in three passes. Each bit of the bit-plane is encoded only once, by one of the three passes. When the EBCOT decides to encode a bit, it sends this bit along with its context to the MQ-coder. The MQ-coder will then generate the compressed bit-stream. The decoding process is very similar: the main difference is that the EBCOT sends only the context to the MQ-decoder and waits for the decoded bit. This implies a little efficiency loss given that the EBCOT is inactive while waiting for the MQ-decoder's answer.

To achieve a specific bit-rate while guaranteeing the minimum distortion for this bit-rate, the EBCOT defines a set of optimal truncation points during the code-block coding process. If each bit-plane was encoded in one pass, the only optimal truncation points would be the bit-planes borders. Distributing the bits of a bit-plane between three subgroups according to the distortion reduction they bring defines two optimal truncation points more inside each bit-plane and allows a more precise bit-stream truncation.

Concretely, each coefficient in a code-block has three state variables and each bit in a bit-plane has two. These state variables indicate for example if the coefficient has already become significant, or if the bit in the bit-plane has already been processed by one of the three passes. They are held up to date by the EBCOT and are used to decide in which pass the bit has to be encoded. They influence also the neighbors context.

B. CMU architecture

A simplified view of the CMU architecture adopted is presented in Fig. 5 and is based on the one developed by Andra *et al.* in [12]. Various optimizations have been made, most of them due to the use of the parallel mode. The architecture consists of :

- Three similar entities, each one performing one of the three decoding passes. A unique counter for the three passes manages the code-block's characteristics, which enables highly simplified control parts for the three blocks. It synchronizes the passes and controls the code-block's borders. This allows the three entities to work the same way either they decode bits inside a code-block or either on its border.
- An internal RAM which stores the state variables. The passes communicate these state variables through specific registers, offering a significant memory reduction (44% in comparison to an architecture without this registers system) and a higher output rate.
- An external RAM where the code-block is progressively decoded. As the EBCOT algorithm complexity makes the CMU the slowest component of the decoder, the memories designed to receive the decoded code-blocks are able to take care of two code-blocks at a time. In this way, the EDU can begin to decode a new code-block while the previous one is still being processed by the IDWT.

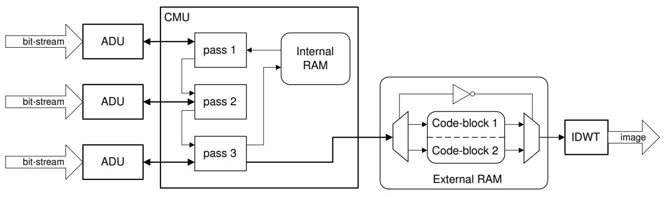

Fig. 5. Simplified view of the CMU architecture.

For the whole image, the CMU takes in average 2.1 clock cycles to produce one bit at its output[2]. The CMU decodes actually one bit in 2 to 4 clock cycles, depending on the position of the code-block in the image. In low frequencies, the code-blocks are much more complex than in high frequencies, but their size is much smaller (for an image of 5 levels of resolution, 0.04% of the coefficients belong to the lowest resolution, and 75% to the highest).

[2]This result does not take into account the ADU decoding time.

V. Arithmetic Decoding Unit

A. MQ algorithm

The basic idea of a binary arithmetic coder is to find a rational number between 0 and 1 which represents the binary sequence to encode. This is done using successive subdivisions of the $[0;1]$ interval based on the symbols probability. Fig. 6 shows the conventions used for the MQ-coder. C is the starting

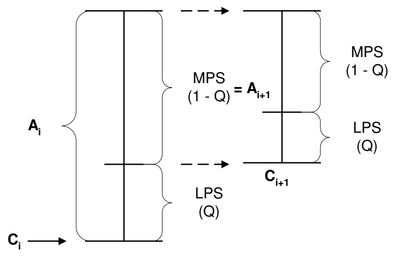

Fig. 6. Successive interval subdivisions in the MQ-coder (MPS-encoding case).

point of the current interval and represents also the current rational number used to encode the binary sequence. A is the size of the current interval. Q is the probability of the Least Probable Symbol (LPS) and is used to subdivide the current interval. According to the symbol we want to encode (MPS or LPS), we use respectively:

$$\begin{aligned} A_{i+1} &= A_i * (1-Q) \quad \text{and} \quad C_{i+1} = C_i + A_i * Q \\ A_{i+1} &= A_i * Q \quad \text{and} \quad C_{i+1} = C_i \end{aligned}$$

During the coding process, renormalization operations are performed in order to keep A close to unity. This leads to the following simplified equations, respectively for an MPS and an LPS:

$$\begin{aligned} A_{i+1} &= A_i - Q_e \quad \text{and} \quad C_{i+1} = C_i + Q_e \\ A_{i+1} &= Q_e \quad \text{and} \quad C_{i+1} = C_i \end{aligned}$$

At each step, the Q-value is retrieved from two serial look-up tables, using the context provided by the CMU.

In a reverse way, the decoding process consists in deciding to which interval (MPS or LPS) belongs the rational number provided and in progressively going up until the $[0;1]$ interval.

B. ADU architecture

The ADU architecture is presented in figure 7. An analysis of the MQ-algorithm shows that only four steps are needed to decode one symbol. The first one is the $LOAD$ step. It uses two small RAM's and a ROM, all realized with look-up tables (LUT's). Given a context, it retrieves the corresponding probability and the MPS-value. During the $COMPUTE$ step, three adders/subtracters perform all the operations needed to decide if a MPS or a LPS must be decoded, which is done during the $DECIDE$ step. For some symbols only, a fourth

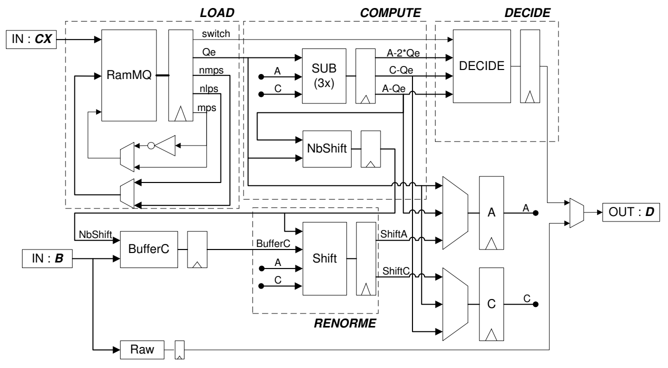

Fig. 7. ADU architecture.

step is performed during which a renormalization (several left-shift operations) of the A-interval takes place. This causes the bit-stream to be progressively decoded.

Some characteristics of this architecture are worth being noticed.

- The main control state machine consists of only 5 states. Furthermore, the CMU is waiting for the ADU answer during only three of them. Therefore a symbol may be decoded in 3 clock cycles. The bypass mode still improves this result (Raw-block in Fig. 7). This explains why three adders/subtracters were needed in place of one: all the arithmetic operations must be done in one clock cycle.
- The compressed data loading ($BufferC$-bloc in Fig. 7) is performed in a independent process and during non-critics moments of the whole decoding process, i.e. when the CMU is not waiting for an answer.
- To speed up the RAM initialization which takes place each time a new codeword is sent to the ADU, two RAM's are used alternatively.
- Finally, to allow the renormalization to be executed in one clock cycle, a speculative computation of the number of left-shifts to execute is performed ($NbShift$-block).

VI. Inverse DWT

A. DWT basics

In JPEG2000, the DWT is implemented using a lifting-based scheme [10]. Compared to a classic implementation, it reduces the computational and memory cost, allowing in-place computation. The basic idea of this lifting-based scheme is first to perform a $lazy$ wavelet transform which consists in splitting odd and even coefficients of the 1D-signal in two sequences. Then, successive *prediction* and *update* steps are applied on these sequences until wavelet coefficients are obtained. The 2D-transform is simply performed by successively applying the 1D-transform in each direction.

In the proposed architecture, the (5,3) transformation is implemented. It is an integer-to-integer wavelet transform and enables therefore lossless coding. Only one prediction step and one update step are needed to perform the whole 1D-transformation. Let $x(n)$ be the spatial coefficients sequence and $y(n)$ be the wavelet coefficients sequence, equations used to realize the transformation are :

$$\begin{aligned} y(2n+1) &= x(2n+1) - \left\lfloor \frac{x(2n)+x(2n+2)}{2} \right\rfloor \\ y(2n) &= x(2n) + \left\lfloor \frac{y(2n-1)+y(2n+1)+2}{4} \right\rfloor \end{aligned}$$

The inverse transformation described below is simply obtained using the reverse system :

$$\begin{aligned} x(2n) &= y(2n) - \left\lfloor \frac{y(2n-1)+y(2n+1)+2}{4} \right\rfloor \\ x(2n+1) &= y(2n+1) + \left\lfloor \frac{x(2n)+x(2n+2)}{2} \right\rfloor \end{aligned}$$

B. IDWT architecture

To reconstruct one resolution level, an horizontal transformation is applied first, followed by a vertical one. The horizontal transformation architecture, further detailed in [9], is shown in Fig. 8.

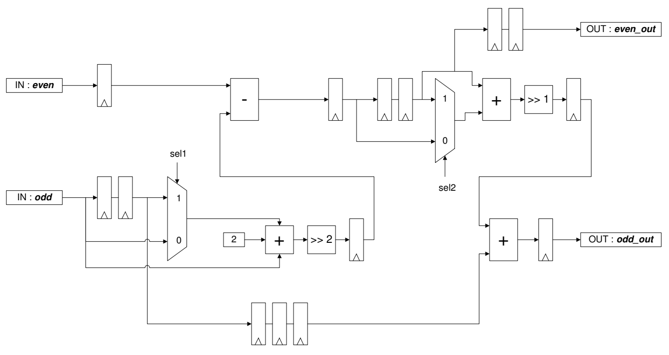

Fig. 8. Horizontal inverse DWT.

As it can be seen, only four adders/subtracters, two shifters, two multiplexers and some registers are needed to implement the equations above. In particular, no multiplier is used since the only divisions performed are implemented with shifters. This architecture is entirely pipeline : every new data couple "pushes" the already present ones a little more through the pipeline, toward the two outputs. The vertical transformation is very similar to the horizontal one. The major difference is that to reconstruct one coefficient, neighbors above and below are needed (in place of left and right neighbors). This implies to buffer two entire lines of the level being reconstructed. In Fig. 8, these buffers replace the two serial registers preceding each multiplexer.

The whole IDWT architecture has already been presented in Fig. 4. In comparison with Chrysafis who presented in [11] such kind of architecture, various optimizations have been made. First, as mentioned above, the lifting scheme has been adopted for each level. Second, interconnection of the blocks has been carefully studied and simplified. Each IDWTi-block ($i = 0..4$) reconstructs one resolution level and behaves at its output like a FIFO. Therefore, each block "sees" four FIFO's as its inputs. Finally, the pipeline characteristic, already present inside each level has been extended to the whole architecture.

Thanks to the progression order chosen, the sixteen FIFO's (one per subband) are filled as uniformly as possible. As soon as its input FIFO's contains data (including the one simulated by the preceding level), an IDWTi-block begins to reconstruct its level. When the pipeline is full, coefficients of the reconstructed image are provided line by line at each clock cycle.

The pipeline latency for a 512x512 tile has been computed and is about 2^{10}, which is 256 times smaller than the 2^{18} clock cycles needed to decode the entire image. Furthermore, this small latency enables a line-based image reconstruction, as only $\frac{1}{256}$ of the entire image needs to be buffered inside the IDWT architecture.

VII. Performances

The architecture presented has been implemented in VHDL and synthesized and routed in an FPGA (Xilinx XC2V6000) using 10 EDU in parallel. Table II presents the resources used with this configuration. As it can be seen, only 61.8% of the RAM resources are used.

TABLE II

Synthesis results of the decoding scheme in a Xilinx XC2V6000

Slices	30,323 over 33,792 (89.7%)
Look-Up Tables	51,416 over 67,584 (76.1%)
RAM blocks (16kbits)	89 over 144 (61.8%)
CLK1 (EDU's & Dispatch)	89.9 MHz
CLK2 (IDWT)	75,9 MHz

Table III presents the bit-rates achieved by our architecture with the configuration described above. 24bpp-images were encoded using options explained in section III.C. As we can see, this configuration yet enables real-time 8-bit 4:4:4 video decoding for the 1080/24p HDTV format and a compression ratio of 20. For a compression ratio of 11, the same format is supported with 8-bit 4:2:2 images.

TABLE III

Bit-rates achieved by the Proposed architecture

Compression ratio	**10-EDU [Mbps]**	**IDWT [Mbps]**	**Complete Scheme [#imgs(1920x1080)/sec]**
1:10	728.0	2 440	14.63
1:20	1 290	2 440	25.92
1:32	2 137	2 440	42.94

Several other JPEG2000 hardware implementations have been developed. The main differences between two recent ones and the proposed architecture are listed in Table IV.

TABLE IV

Differences between two recent implementations and the proposed architecture

	Barco Silex[13]	**Arizona Univ.**[14]	**Proposed architecture**
Technology	FPGA XC2V3000	ASIC 0.18μm	FPGA XC2V6000
Tile size	128x128	128x128	512x512
Cblk size (max.)	32x32	32x32	64x32
Wavelet filters used	(5,3)-lossless (9,7)-lossy	(5,3)-lossless (9,7)-lossy	(5,3) lossy and lossless
Entropy coders	8	3	10

VIII. Conclusion

In this paper, we have proposed a hardware JPEG2000 decoder for real-time applications such as Digital Cinema. It has been implemented in VHDL, and synthesized and routed in an FPGA.

Various previous contributions have been joined together and optimized to provide a complete, flexible, secure, high performance decoding scheme.

The system proposed is secure because no external memory is used and the data flow is protected during the whole decoding process.

Thanks to three different levels of parallelization and a line-based data processing, high output rates are achieved. With a compression ratio of 20, the configuration synthesized in the FPGA supports the 1080/24p HDTV format for 8-bit 4:4:4 images.

Finally, the system proposed is highly flexible. In order to satisfy a broad range of constraints, including upcoming standards, two of the three parallelization levels are very easily customizable. They allow the proposed architecture to fit in any FPGA without further development.

References

[1] *ISO/IEC 15444-1: Information Technology-JPEG 2000 image coding system-Part 1: Core coding system,* 2000.

[2] D. Santa-Cruz, R. Grosbois, and T. Ebrahimi, "JPEG2000 performance evaluation and assessment", *Signal Processing: Image Communication,* vol. 17, no. 1, pp. 113-130, January 2002.

[3] S. Foessel, "Motion JPEG2000 and Digital Cinema", ISO/IEC JTC 1/SC 29/WG1 N2999, July 2003.

[4] *ISO/IEC 15444-3: Information Technology-JPEG 2000 image coding system-Part 3: Motion JPEG 2000,* 2002.

[5] Virtex^{TM}-II platform FPGAs: Complete Data Sheet. Xilinx. [Online]. Available: http://www.xilinx.com.

[6] M. Rabbani and R. Joshi, "An overview of the JPEG2000 still image compression standard", *Signal Processing: Image Communication,* vol. 17, no. 1, pp. 3-48, January 2002.

[7] D. Taubman, "High performance scalable image compression with EBCOT", *IEEE Trans. on Image Processing*, vol. 9, no. 7, pp. 1158-1170, July 2000.

[8] D. Taubman and M. W. Marcellin, *JPEG2000: Image Compression Fundamentals, Standards and Practice*, Kluwer Academic, Boston, MA, USA, 2002.

[9] G. Dillen and B. Georis, "JPEG 2000 : étude et conception du décodeur arithmétique et de la transformée en ondelettes". Microelectronics Laboratory (DICE), UCL, Belgium, June 2001.

[10] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps", *J. Fourier Anal. Applic.,* vol. 4, pp. 247-269, 1998.

[11] C. Chrysafys and A. Ortega, "Line based, reduced memory, wavelet image compression", *IEEE Trans. on Image Processing,* vol. 9, no. 3, pp. 378-389, March 2000.

[12] K. Andra, T. Acharya, and C. Chakrabarti, "Efficient VLSI implementation of bit plane coder of JPEG2000", in *Proc. SPIE Int. Conf. Applications of Digital Image Processing XXIV*, vol. 4472, pp. 246-257, December 2001.

[13] JPEG2000 Decoder: BA111JPEG2000D Factsheet. Barco-Silex, October 2003. [Online]. Available: http://www.barco.com.

[14] K. Andra, T. Acharya, and C. Chakrabarti, "A High-Performance JPEG2000 Architecture", *IEEE Trans. on Circuits and Systems for Video Technology,* vol. 13, no. 3, pp. 209-218, March 2003.