International Scholarly Research Network
ISRN Signal Processing
Volume 2012, Article ID 714176, 9 pages
doi:10.5402/2012/714176

Research Article

Area Efficient, High Speed EBCOT Architecture for Digital Cinema

Kishor Sarawadekar[1] and Swapna Banerjee[2]

[1] *ADVS Department, Xilinx India Technology Services Pvt. Ltd., 500 081 Hyderbad, India*
[2] *Department of Electronics and Electrical Communication Engineering, Indian Institute of Technology Kharagpur, Kharagpur 721302, India*

Correspondence should be addressed to Kishor Sarawadekar, to.skishor@gmail.com

Received 6 March 2012; Accepted 11 April 2012

Academic Editors: S. K. Bhatia and W.-L. Hwang

Embedded block coding with optimised truncation (EBCOT) is a key algorithm in digital cinema (DC) distribution system. Though several high speed EBCOT architectures exist, all are not capable of meeting the DC specifications. To meet this challenge, the relationship between contents of a code block (CB) and context generation is studied. Our study reveals that it is difficult to predict number of contexts generated in a bit plane. Even the nature of number of contexts produced varies from CB to CB. In such a situation, it is difficult to ensure the frame rate requirement of DC. To avoid this uncertainty, a pass parallel, concurrent sample coding EBCOT architecture is proposed in this paper. It is capable of encoding one bit plane in 288 clock cycles under any circumstances. This design is prototyped on XC4VLX80-12 FPGA with multiple clock domains. After synthesizing, the bit plane coder (BPC) and MQ coder operate at 450 MHz and 123 MHz, respectively. In order to maintain synchronism among different clock domains, the BPC and MQ coder units are operated at 432 MHz and 108 MHz, respectively. This entails that the proposed design is capable of processing 2048×1080 size 57 DC frames in a second.

1. Introduction

With the advancement of digital technology, many new imaging applications have emerged in the consumer electronics domain. These applications include digital camera, high resolution scanners and printers, and home theatre. To cater the demands of all these applications, a still image compression standard JPEG 2000 has been developed [1]. Some of the important features provided in this standard are capability of lossy as well as lossless compression, error resilience, region of interest coding, high compression ratio, manipulation of images in compressed domain, acceptable performance even at low bit rates, and secured image transmission. The key algorithms, which enable all these features, are Discrete Wavelet Transform (DWT) and Embedded Block Coding with Optimised Truncation (EBCOT) [2]. Both algorithms being computation intensive are not suited for a real-time application using JPEG 2000 without having dedicated hardware platform.

In March 2002, digital cinema initiative (DCI) was established [3] to develop new standard, suitable for digital cinema (DC) applications. In 2005, DCI officially selected JPEG 2000 part-1 as compression engine for DC applications [4]. The DC system consists of four parts: mastering, transport, storage and playback, and projection, as illustrated in Figure 1 [5]. The movie is compressed, encrypted, and packaged for delivery in the mastering stage. Next, these data are transported to the exhibition site where they are decrypted, decompressed, stored, and played back.

There are two different image resolutions, 2048×1080 (2k) and 4096×2160 (4k) used in DC application. Further, frame rate to be supported by the 2k image is either 24 or 48 frames per second (fps), whereas it is 24 fps in the case of 4k image. This connotes that the size of raw data to be processed is more than 9 terabytes for a three-hour film. Therefore, a high speed JPEG 2000 encoder is indispensable to compress this information in real time.

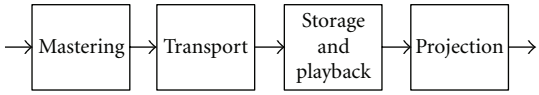

Figure 1: Block diagram of a digital cinema system.

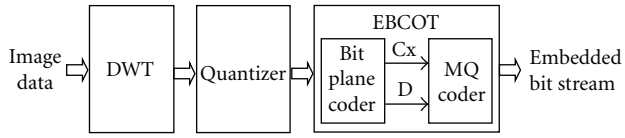

Figure 2: Block diagram of JPEG 2000 encoding system.

To satisfy the requirement of DC compression system, we have already developed a DWT core [6], and, in this paper, we are presenting a high speed VLSI architecture for EBCOT. The rest of this paper is organized as follows. Section 2 provides an overview of EBCOT algorithm and the related research. Section 3 presents the relationship between contents of a code block (CB) and number of contexts generated. The proposed VLSI architecture of EBCOT encoder is presented in Section 4. In Section 5, experimental results are given and are compared with the state of the art of JPEG 2000 architectures. Finally, brief conclusions are drawn in Section 5.

2. Overview of JPEG 2000 Encoding System

The block diagram of the JPEG 2000 coder is shown in Figure 2. Here, input image frame is partitioned into rectangular, nonoverlapping tiles. All unsigned image samples are DC level shifted and are centred around zero. Next, color space conversion, RGB to YCbCr, is performed and wavelet transform is applied independently to each of the image component. The DWT coefficients are supplied to EBCOT coder which comprises two processing stages: bit plane coder (BPC) and MQ coder. The BPC generates context (CxD) pairs which are supplied to MQ coder. The MQ coder performs entropy coding and produces embedded bit stream.

2.1. Bit Plane Coding. The DCI standard has confined to use 9/7 DWT filter. The coefficients generated after wavelet transform are in 2's complement format. These coefficients are converted to sign-magnitude format and stored in code block (CB) memory. In DC applications, CB size is restricted to 32×32 [5]. The CB memory comprises a sign plane and several magnitude planes, as shown in Figure 3(a). During encoding, the magnitude bit planes are scanned starting from the most significant bit (MSB) plane to the least significant bit (LSB) plane. Each bit plane is further divided into stripes of four rows (refer to Figure 3(b)). Within a stripe samples are scanned column-wise starting from left to right and within a column, from top to bottom as shown in Figure 3(c).

To examine each sample in a CB three coding passes—clean-up pass (CUP), significant propagation pass (SPP) and

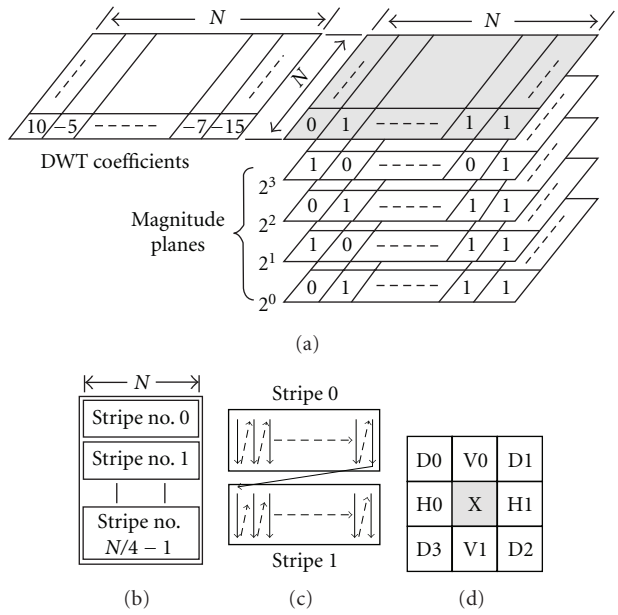

Figure 3: Samples scanning order in BPC. (a) Composition of a code block memory. (b) Stripes forming a bit plane. (c) Scanning order within a stripe. (d) Context window surrounding "M", that is, the sample to be coded.

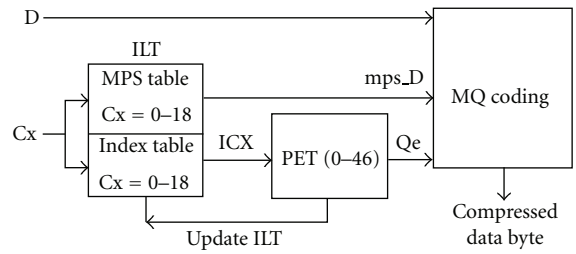

Figure 4: Functional block diagram of an MQ coder.

magnitude refinement pass (MRP)—are used. Type of coding pass to be applied on a sample is determined by forming a context window surrounding it, as shown in Figure 3(d), along with the three state variables σ, σ', and η. While scanning DWT coefficients, whenever first nonzero magnitude bit occurs, it is considered as a significant coefficient and σ state is updated to one. When MRP is run for the first time on a coefficient, σ' variable is set to one and if zero coding is applied on a coefficient in SPP, η is set to one. Initially, all state variables are assumed to be zero. The encoding starts from the first nonzero magnitude bit plane assuming that all state variables are zero. Only CUP is run on this bit plane, whereas rest other bit planes are coded sequentially using SPP, MRP, and CUP. All samples are encoded using four coding primitives: zero coding (ZC), sign coding (SC), run length coding (RLC), and magnitude refinement coding (MRC) and CxD pairs are generated.

2.2. MQ Coder. The MQ coder is a context-based adaptive binary arithmetic unit. CxD pairs generated by BPC are fed to the MQ coder as shown in Figure 4. The decision bit

(D) represents either more probable symbol (MPS) or less probable symbol (LPS). Each context (Cx) has an associated state which is stored in an index look-up table (ILT). These states provide the MPS value and index (ICX) to a probability estimation table (PET). The PET contains the information of Qe (the estimate of the LPS's probability), NMPS (next state if MPS is coded), NLPS (next state if LPS is coded), and SWITCH indicating whether the MPS value stored in the state has to be switched upon an LPS transition. The standard has predefined all 47 PET entries. Each entry has 28 bits which may be defined in a ROM. MQ coder module contains a set of registers: A (16 bit), B (8 bit), C (28 bit), CT (4 bit), and BP (16 bit). Register A is the interval register. It holds value of current interval as required in the MQ coder. Register A is initialised to 0x8000 which is equivalent to decimal 0.75. The code register C holds partial codeword at any stage of encoding. It is initialised with 0x000 0000 which means that no codeword has been generated yet. The interval in A is kept in the range $0.75 \leq A \leq 1.5$. With each binary symbol, entire range is subdivided into two subintervals and one of them is selected as a new interval. Whenever the range value in A falls below 0.75, data in registers A and C are doubled. This procedure is called renormalisation and it continues till data in register $A \not\geq 0.75$. During renormalisation, when A and C registers are doubled a down counter, CT is decremented by one. As soon as $CT = 0$, byte stored in register B is emitted as the compressed output byte (CD) and the buffer pointer (BP) are incremented.

2.3. Existing Architectures. Some of the special features supported by JPEG 2000 are discussed in [7]. Concurrent execution of several operations at different stages is proposed in [8]. Here, number shift operations performed by an MQ coder are analyzed to optimize the hardware requirement. Quad code block-based architecture for JPEG 2000 is proposed in [9]. Here, three EBCOT engines are used to improve the throughput which may increase hardware cost excessively.

An EBCOT coder to code all passes sequentially is proposed in [10]. Here, pixel skipping technique is implemented to improve the coding efficiency. As it is a sequential coder, huge number of clock cycles are required to process an image. Moreover, memory requirement is very high.

Simultaneous coding of two stripe columns is presented in [11]. Two processing elements are used to execute passes concurrently. But within a column, processing is sequential. Besides this, processing of the next stripe does not begin until all samples are coded by the second processing element.

Architecture for concurrent coding of all bit planes is presented in [12]. With this approach, need for state variables memory is completely eliminated. Further, one arithmetic coder is shared between two bit planes. However, knowledge of leading zero bit planes is not embedded in this design. Consequently, CxD pairs are generated by these planes too, which are extra burdens on the arithmetic coder.

The architecture in [13] processes all symbols, within a column, concurrently. Multiple-columns skipping technique is also incorporated in this architecture. The MQ coder accepts two CxD pairs in every clock cycle. However, logic required to generate the pass membership flag is much complicated. Additionally, the memory requirement of the MQ coder is approximately three times higher than that of the reference coder.

Two different pass parallel coding schemes—group 2 and group 4—are presented in [14]. In group 4 method, all four samples within a column are coded at a time, whereas only two samples are coded simultaneously in group 2 method. However, the statistical analysis reported for group 4 method is incorrect because while coding four samples concurrently, two or three CxD pairs are not generated by any chance.

A software-hardware codesign for JPEG 2000 encoder is proposed in [15]. The computation intensive blocks (DWT, BPC, and MQ coder) are implemented on FPGA and rest other blocks of the encoder are implemented on a power PC processor. This architecture is capable of coding five bit planes at a time and it requires 2–5 cycles to encode a column. However, the FPGA implementation details are not presented.

A serial hardware implementation of the EBCOT is proposed in [16]. The BPC module can produce up to four CxD pairs, whereas the MQ coder consumes only one CxD pair in one clock. Therefore, in this design four arithmetic coders are used. But, the maximum operating frequencies of BPC and MQ coder are 18.5 MHz and 9.25 MHz, respectively, which are very low.

Instead of iterative method, batch processing of renormalisation and byte-out procedures in MQ coder are presented in [17]. However, only software implementation is discussed without any hardware solution. High performance MQ coder architectures are presented in [18, 19]. However, the performance improvement is traded off with silicon area. A high performance MQ encoder, which can process two CxD pairs in one clock cycle consistently, is presented in [20]. The requirement of PET memory bits is also reduced in this design. However, a lot of resources in the data path and controller design are underutilised.

All these architectures require either large encoding time or large space on silicon. Both area and execution time are critical parameters while developing an intellectual property core for real-time applications. The performance of the proposed EBCOT encoder is improved by compromising both these parameters.

3. Analysis of EBCOT Algorithm

Analysis of context generation in the BPC is performed to develop a technique by which BPC's throughput can be improved. To implement high speed EBCOT architecture concurrent context coding technique is adopted in this design. If the draft of JPEG 2000 standard is followed to implement BPC, all passes are executed in a sequential manner. This implies that to encode a CB of size $N \times N$ with bp number of bit planes, $3 \times bp \times N \times N$ clocks are required. It is a cumbersome task. The BPC encoding performance may be improved by executing all passes in a single go. To investigate this fact, five grayscale ISO images—Lena, Boboon, Boat, Peppers, and Barbara—of size 512×512 are

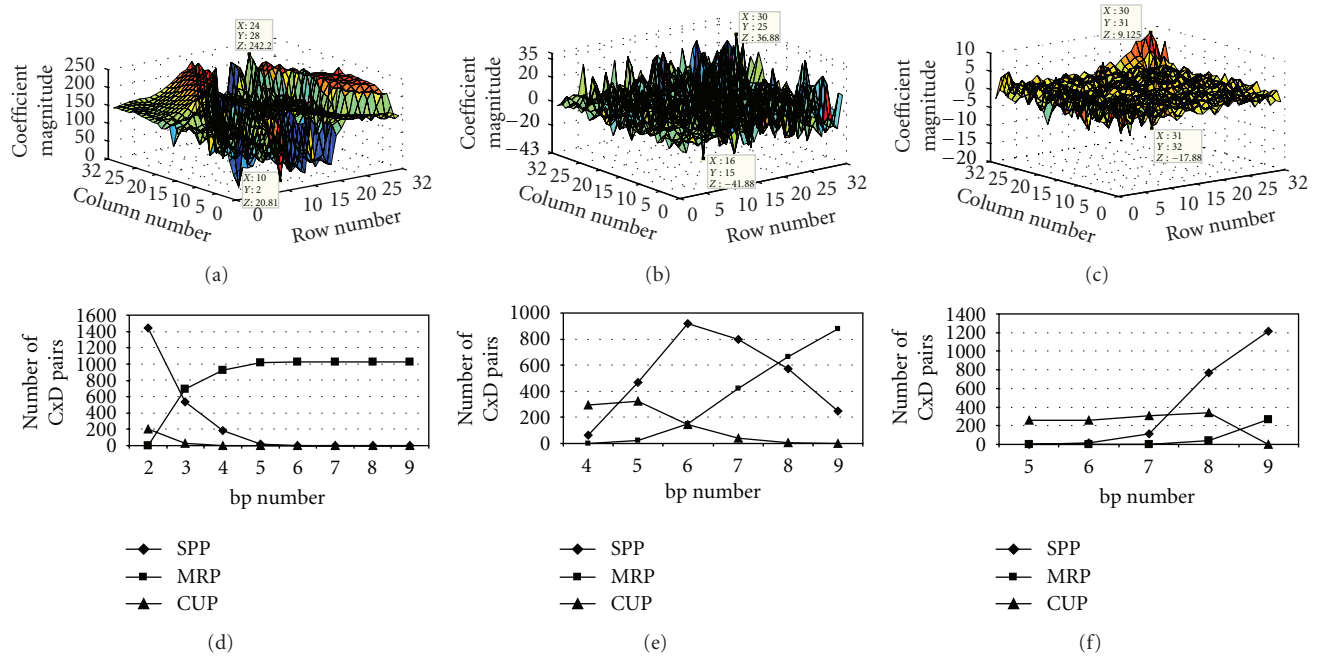

Figure 5: Relationship between contents of a CB and number of contexts generated. (a)–(c) Magnitude plot for all coefficients in a CB with maximum number of positive coefficients, negative coefficients, and zero coefficients, respectively. (d)–(f) Number of CxD pairs generated for the corresponding CBs in each pass and for each bit plane.

used. Using DWT core, developed in house [6], images are decomposed up to three levels. By storing ten bits of each DWT coefficient, CB data is prepared. This analysis uses 32×32 size CB. Finally, with the help of C simulation, all CBs in an image are encoded and context pairs are generated.

The DWT coefficients are converted to sign magnitude format and stored in CB memory. Next, contents of all CBs are analysed using coefficient's sign and magnitude. CBs with maximum number of positive coefficients, negative coefficients and zero coefficients are selected for further analysis. These CBs are encoded using BPC core and CxD pairs are generated. The relationships between coefficients of a CB and number of contexts generated are presented in Figure 5. Here, bit plane numbers 1 to 9 represent the order of magnitude planes, starting from MSB to LSB planes.

From Figure 5(a), it is observed that all coefficients in this CB are positive. Additionally, the maximum and the minimum values for coefficients are 242 and 21, respectively. During encoding MSB plane, bp1, is skipped because all samples in this plane are insignificant. It entails that while encoding bp2, MRP will not generate any CxD pair. Figure 5(d) demonstrates this fact. Large numbers of samples become significant in bp2. Due to this, huge numbers of CxD pairs are produced by the SPP and a few are generated by the CUP. The least magnitude coefficient becomes significant in bp6. Therefore, after this bit plane, no CxD pairs are generated by SPP and CUP.

In case of a CB with maximum number of negative coefficients, the values range from -42 to 37, please refer to Figure 5(b). However, large number of coefficients need four bits to represent their magnitudes. Consequently, contributions of SPP CxD pairs are much higher in the bp6, as shown in Figure 5(e). Similarly, in case of a CB with maximum number of zero coefficients, the magnitudes are in the range of -18 to 9, refer to Figure 5(c). However, large number of coefficients need only one bit to represent their magnitude. As a result, contributions of CUP CxD pairs are much higher in all bit planes except in the LSB plane, as shown in Figure 5(f).

The plots shown in Figure 5 correspond to the Lena image. Each CB has 1024 number of coefficients and there are 256 CBs. However, CBs numbers 18, 90, and 216 are used in this analysis because all coefficients in CB 18 have positive sign, CB 90 has maximum (525) number of coefficients with negative sign and CB 216 has maximum (303) number of coefficients with zero magnitude. After analyzing other images in this way, similar relationships are observed between the contents of a CB and number of CxD pairs generated.

From this analysis it can be concluded that:

(i) as encoding progresses, the contribution of CxD pairs generated by MRP increases gradually, whereas it decreases for the SPP and CUP;

(ii) the contribution of CxD pairs generated in the first non-zero magnitude bit plane depends on the contents of a CB. Here, either SPP or CUP is dominant;

(iii) in the first non-zero magnitude bit plane, MRP does not generate any CxD pair;

(iv) when maximum number of samples become significant in a bit plane, from the next bit plane onwards large number of CxD pairs are generated by MRP;

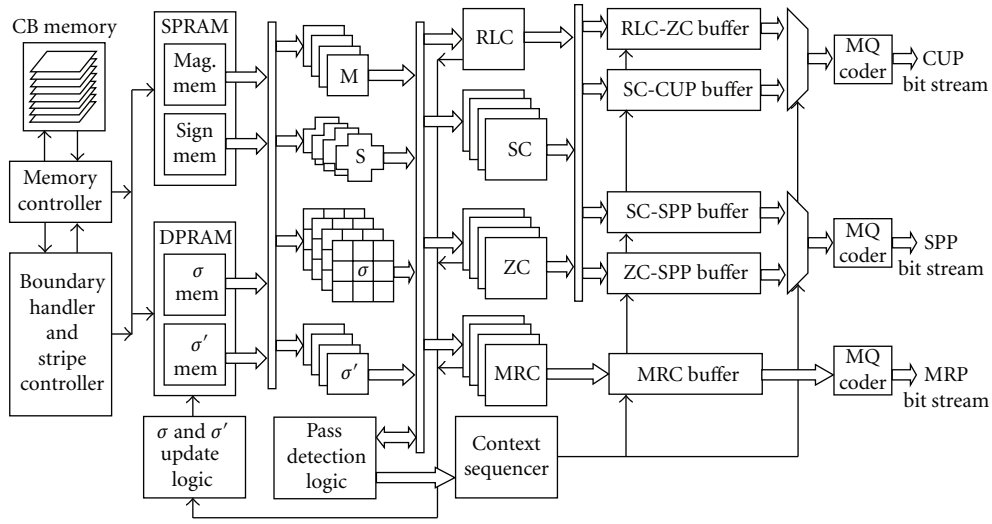

Figure 6: Proposed EBCOT architecture.

(v) the rate of CxD pair generation is totally dependent on the contents of a CB. Therefore, it is difficult to predict number of contexts produced in a bit plane.

From this analysis, it may be concluded that number of contexts generated in a bit plane depends on the contents of the CB. To achieve specifications imposed by DCI not only pass parallel but also concurrent sample coding architecture for EBCOT must be adopted. Such architecture is presented in this paper.

4. Proposed EBCOT Architecture

The proposed EBCOT architecture is illustrated in Figure 6. The memory controller selects a magnitude and sign bit plane to be encoded. Additionally, it reads σ and σ' state variable data for the selected CB. Since magnitude and sign data are never updated, these planes are implemented using single port RAM (SPRAM). The state memories are implemented using dual port RAM (DPRAM). Once a bit plane is selected, the stripe controller selects a stripe to be processed. While processing the first column of every stripe, its left side neighboring column is assumed to be zero. Similarly, while processing the last column of every stripe, its right side neighboring column is assumed to be zero. These boundary conditions are handled by the boundary handler. In order to process all magnitude bits concurrently, entire magnitude column and the corresponding sign bits column are read in one clock. In addition, four sign neighbors, eight σ neighbors,n and four σ' values of the column to be processed are read which form reference windows. Based on the values in these windows, pass detector determines the coding pass to be run and enables necessary primitive coding modules. With the help of one RLC, four ZC, four SC, and four MRC primitives, all samples in a column are coded concurrently. As a result, 1 to 10 CxD pairs get generated simultaneously. The context sequencer segregates these CxD pairs and stores them in the respective pass buffers. The MQ coders consume buffered CxD pairs and produce separate compressed bit stream corresponding to each coding pass.

4.1. Pass Detection Logic. Figure 7(a) shows a magnitude column to be processed and its neighboring σ states. The σ values from neighboring stripe are assumed to be zero because stripes are scanned in vertical causal mode. Here, M0 to M3 indicate current sample column which is to be encoded. Similarly, σ values A to D, E to H, and I to L represent previous, current, and next σ columns, respectively. When all (A to L) σ values are zero, CUP pass is run, vide Figure 7(b). All significant samples are coded using MRP. Therefore, if any σ value in the current σ column (i.e., E to H) is set, the corresponding sample is coded in MRP as shown in Figure 7(c). An insignificant sample with significant neighbor/s is coded in SPP. To detect this condition, significance of the neighboring samples is tested and for each context window, one bit result is generated. Bits P, Q, R, and S in Figure 7(c) represent the significance test results for samples M1, M2, M3, and M4, respectively. The logic diagram to code a sample in SPP is depicted in Figure 7(c). For further details on concurrent pass detection logic, reader is referred to [21].

4.2. Primitive Coding

4.2.1. Zero Coding. To encode DWT coefficients context tables are provided in the JPEG 2000 standard [1]. The ZC primitive encodes DWT coefficients in LL and LH subband using single table, which is provided in the standard [1]. First, summation of horizontal, vertical, and diagonal σ values is computed and next they are compared against the standard table to determine context of a sample. Therefore, first σ values are summed and next contexts are produced. The VLSI realisation of ZC primitive for LL and LH subband is presented in Figure 8. In ZC, current sample's magnitude bit (i.e., Mx) is treated as a decision bit. To encode all samples, in a column, concurrently four such modules are implemented in this architecture.

4.2.2. Sign Coding. To determine SC context value, a table is provided in the JPEG 2000 standard [1]. Context values 9 to 13 are produced by this primitive. The values of H and V

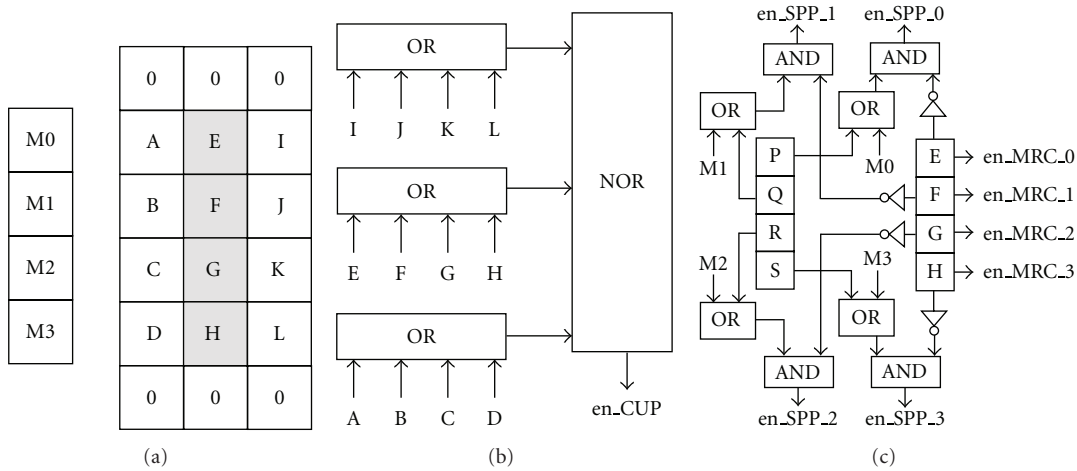

Figure 7: Pass detection logic. (a) Magnitude samples to be processed and their neighboring σ states. (b) CUP detection logic. (c) SPP and MRP detection logics.

neighbors' significance- and sign-bits must be known to code sign of a sample. Let us assume that σ_{v0} and σ_{v1} represent the σ state of the vertical neighbors, whereas σ_{h0} and σ_{h1} represent the σ state of the horizontal neighbors. Similarly, sc_{v0} and sc_{v1} represent the sign bit of the vertical neighbors, whereas sc_{h0} and sc_{h1} represent the sign bit of the horizontal neighbors. With the help of σ and sign bits, we can write

$$w0 = (\sigma_{v0} \wedge \sigma_{v1}) \mid ((\sigma_{v0} \,\&\, \sigma_{v1}) \&(\sim (sc_{v0} \wedge sc_{v1}))), \quad (1)$$

$$\begin{aligned} w1 = {} & (\sim (\sigma_{v1}) \,\&\, \sigma_{v0} \,\&\, sc_{v0}) \mid \\ & (\sim (\sigma_{v0}) \,\&\, \sigma_{v1} \,\&\, sc_{v1}) \mid \\ & (\sigma_{v1} \,\&\, \sigma_{v0} \,\&\, sc_{v0} \,\&\, sc_{v1}), \end{aligned} \quad (2)$$

$$w2 = (\sigma_{h0} \wedge \sigma_{h1}) \mid ((\sigma_{h0} \,\&\, \sigma_{h1}) \,\&\, (\sim (sc_{h0} \wedge sc_{h1}))), \quad (3)$$

$$\begin{aligned} w3 = {} & (\sim (\sigma_{h1}) \,\&\, \sigma_{h0} \,\&\, sc_{h0}) \mid \\ & (\sim (\sigma_{h0}) \,\&\, \sigma_{h1} \,\&\, sc_{h1}) \mid \\ & (\sigma_{h1} \,\&\, \sigma_{h0} \,\&\, sc_{h0} \,\&\, sc_{h1}). \end{aligned} \quad (4)$$

Here and elsewhere, $\sim$ represents bit-wise NOT, $\mid$ denotes bit-wise OR, & denotes bit-wise AND, and, $\wedge$ represents bit-wise XOR. By complementing (1)–(4), we can write four more expressions for $w4$, $w5$, $w6$, and $w7$, respectively:

$$Cx[0] = (w0 \,\&\, w2) \mid (w4 \,\&\, w5 \,\&\, w6 \,\&\, w7), \quad (5)$$

$$\begin{aligned} Cx[1] = {} & (w0 \,\&\, w6 \,\&\, w7) \mid \\ & (w0 \,\&\, w1 \,\&\, w7) \mid \\ & (w0 \,\&\, w2 \,\&\, w3 \,\&\, w5), \end{aligned} \quad (6)$$

$$\begin{aligned} Cx[2] = {} & (w2 \,\&\, w5 \,\&\, w7) \mid \\ & (w2 \,\&\, w4 \,\&\, w5) \mid \\ & (w0 \,\&\, w1 \,\&\, w2 \,\&\, w3). \end{aligned} \quad (7)$$

Boolean expressions (5)–(7) are mapped on FPGA, to generate three LSBs of the SC contexts value. Since two MSBs of the contexts remain "01" all through, hardware for these bits is not implemented and their values are assigned directly. The $\hat{x}$ value, given in the standard [1], and sign bit of the current sample (i.e., sc_x) are used to determine the decision value, in this primitive. To realise this, (8) is synthesised on FPGA:

$$\begin{aligned} \hat{x} = {} & (w0 \,\&\, w1 \,\&\, w6 \,\&\, w7) \mid \\ & (w2 \,\&\, w3 \,\&\, w4) \mid \\ & (w0 \,\&\, w2 \,\&\, w3), \\ D = {} & \hat{x} \wedge sc_x. \end{aligned} \quad (8)$$

4.3. Context Buffering. Analysis presented in Section 3 demonstrates that nature of the context generation varies from bit plane to bit plane and CB-to-CB. A maximum of ten CxD pairs may be generated while coding all samples in a column concurrently. These pairs must be buffered rapidly to improve throughput of the block coder. Separate buffers are designed for this purpose. Moreover, to speed up the context buffering process contexts generated by the SC primitive are stored separately. Context sequencer circuit is responsible for buffering these CxD pairs in proper order.

4.4. MQ Coder. Block level representation of the proposed MQ coder is depicted in Figure 9. This coder consumes one CxD pair in one clock cycle. It is partitioned into three stages: Interval Detector (ID), Renormaliser (REN), and Byte Out (BO). The CxD pairs generated by BPC are stored in the CxD buffer and are supplied to the ID stage. This stage reads an index value from the ILT and puts it on the address bus of the PET ROM. Depending on the index value received, PET ROM emits Qe, NMPS, NLPS, and SWITCH information. Further, based on the type of coding pass run, ID stage computes the new interval range. Index value in the ILT is

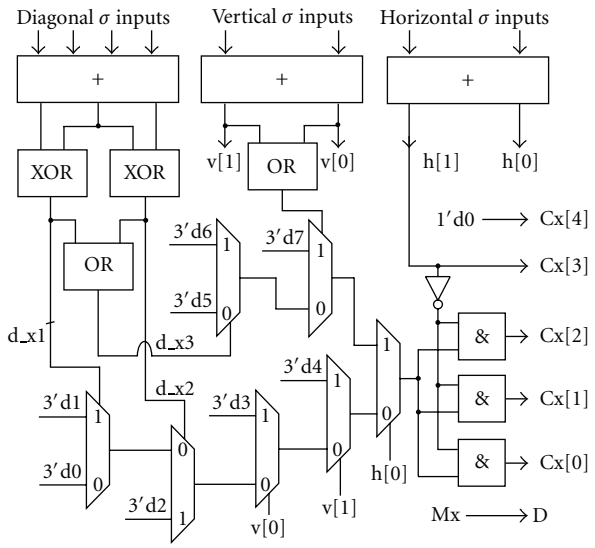

Figure 8: Architecture of ZC primitive to encode a sample in LL or LH subband.

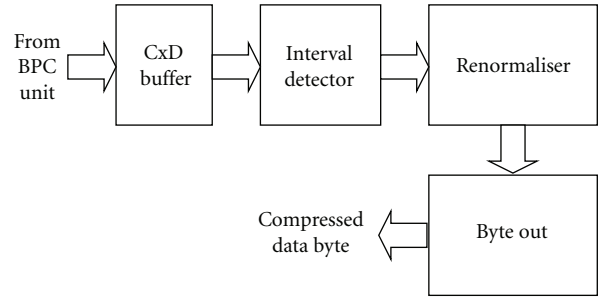

Figure 9: Block level architecture of the proposed MQ coder.

Table 1: Proposed EBCOT architectures.

Architecture	BPC	MQ coder
Max. frequency (MHz)	450.12	123.25
No. of 4 input LUTs	170	626
Total slices used	95	375
Total FF slices	39	307
Renormaliser speed	—	328.40
FPGA used	XC4VLX80-12	

updated, if renormalisation is called. Similarly, during LPS coding if SWITCH = 1, D bit value in the ILT is updated.

If contents of A register are less than 0x8000, renormalisation procedure is executed. Here, contents of both, A and C, registers are left shifted until the value in $A \geq$ 0x8000. To attain this, instead of barrel shifters, high speed shift registers are implemented in this design. The REN unit is capable of operating three times faster than that of the other stages. Due to this, large silicon space has been saved without sacrificing the coding efficiency. Whenever a compressed data byte is ready to output, REN unit passes it to the BO unit.

To improve the coding performance further, REN and BO stages are run concurrently. For this reason, upper nine bits of C register (i.e., C [27 : 19]) are always passed to the BO module. The judgment about bit stuffing requirement is made in this module. If bit stuffing is required, bits C [27 : 20] are extracted, otherwise bits C [26 : 19] are extracted. When REN module signals byte emission, bits in the C register are extracted and stored in a temporary data buffer. Simultaneously, previous data present in the byte out register (B) is emitted as a compressed data byte. Further details on the MQ coder design are available in [22].

5. Experimental Results and Discussion

To investigate relationship between contents of a CB and number of CxD pairs generated, five grayscale ISO images—Lena, Boboon, Boat, Peppers, and Barbara—of size 512×512 are used. In this experiment, Daubechies (9,7) filterbank is used. After three levels of image decomposition ten bits of each DWT coefficient are stored in 256 code blocks. Using C simulation all CBs in an image are encoded and context pairs are generated for analysis purpose. This study demonstrates that in an image the rate of CxD pair generation is totally dependant on the contents of a CB. Hence, it is difficult to predict the number of contexts produced in a bit plane.

The proposed pass parallel, concurrent sample coder is described using Verilog and prototyped on the Xilinx XC4VLX80-12 FPGA. The design implementation summary is given in Table 1 which demonstrates that the area requirement of the proposed design is less, whereas operating speed is very high. The BPC module requires only 288 clock cycles to encode one bit plane. To cope up with the BPC speed three MQ coders are used in this design. The speed of renormaliser is three times that of the MQ coder. This implies that if more than three rotations are required, one more clock is required to encode the symbol. But it is a rare situation [22]. Therefore, it can be assumed that one CxD pair is consumed in a clock cycle. To achieve the best case performance, multiple clock domains are used in this design. The MQ coder, renormaliser, and BPC modules are operated at 108 MHz, 324 MHz, and 432 MHz, respectively. Considering 12 bps in the DC application, 8 μS are sufficient to encode one CB. Thus, the proposed design is capable of processing 2048 × 1080 size 57 DC frames in a second.

The proposed architecture is implemented on XC2V1000 FPGA for the comparison purpose. Table 2 shows this comparison results. BPC architecture in [11] is sequential in nature. Moreover, it uses two processing elements to speed up the EBCOT performance. As a result, it has demanded large number of hardware resources. During MQ coding operation, a maximum 15 shifts may occur. Data width of the registers to be shifted (i.e., A and C) are much larger. Therefore, if barrel shifters are used in the renormaliser stage, hardware cost increases and at the same time operating frequency decreases. This is evident from the MQ coder comparison provided in Table 2. MQ coder design in [20] can consume two CxD pairs in one clock, whereas the

Table 2: Comparison with existing EBCOT architectures.

Module	BPC		MQ coder	
Architecture	[11]	Proposed	[11]	Proposed
Max. frequency (MHz)	50	337	50	97
No. of 4 input LUTs	7071	161	4430	603
Total slices used	4420	91	3024	367
Total FF slices	1560	93	1200	307
Renormaliser speed	—	—	—	200
FPGA used	XC2V1000-6			

Table 3: Comparison with existing MQ coder architectures.

Architecture	[20]	Proposed
Max. frequency (MHz)	48.30	123.25
Memory bits	1509	1449
$C \times D$ consumption rate	2	1
Throughput	96.60	123.25
Renormaliser speed	—	328.40
FPGA used	XC4VLX80	

proposed design encodes one CxD pair in one clock. But, still throughput of the proposed design is 1.28 times greater than that of the design in [20]. Table 3 furnishes the details of this comparison. From Tables 2 and 3, it reveals that the proposed EBCOT architecture is superior than that of the existing designs.

6. Conclusion

In this paper, the relationship between contents of a CB and number of CxD pairs generated is studied. This study establishes that it is difficult to predict the number of contexts produced in a bit plane. Further, by incorporating parallelism at various stages coding speed of BPC can be easily improved. However, MQ coding procedure is sequential in nature and it is very difficult to speed up this module. MQ coder designs capable of encoding two symbols per clock, demand large number of hardware resources and operate at low speed. As a result, throughput of such MQ coders is not much high, as expected. The bottleneck in MQ coder design is the renormaliser stage. If this stage is designed using sequential shift registers, high speed MQ coder can be realised at low hardware cost. In addition, it is better to design a separate MQ coder dedicated to each coding pass. To improve the overall coding efficiency of the proposed EBCOT, multiple clock domains are used in this design. The MQ coder, renormaliser, and BPC modules are operated at 108 MHz, 324 MHz, and 432 MHz, respectively. With this speed the proposed EBCOT architecture is capable of encoding 2048 $\times$ 1080 size 57 frames of DC in a second. Thus, it satisfies all the constraints imposed by the DCI Committee. In fact, the routing delays are much higher compared to the logic delays, in this design which constraints the throughput of the entire system. It is expected that the ASIC implementation of this proposed design will have much better performance.

Acknowledgment

The authors would like to thank GE Healthcare India Pvt. Ltd. for supporting this research work.

References

[1] ISO/ IEC JTC1/SC29/WG1 N2678, "JPEG 2000 Part 1 020719," 2002.

[2] D. T. Lee, "JPEG 2000: retrospective and new developments," *Proceedings of the IEEE*, vol. 93, no. 1, pp. 32–41, 2005.

[3] R. Wintner, "Bits on the big screen," *IEEE Spectrum*, vol. 43, no. 12, pp. 42–48, 2006.

[4] Digital Cinema Initiatives, "LLC Member Representatives Committee: Digital Cinema System Specification V1.0," Final Approval, July 2005.

[5] Digital Cinema Initiatives, "LLC Member Representatives Committee: Digital Cinema System Specification V1.2," March 2008.

[6] A. Das, A. Hazra, and S. Banerjee, "An efficient architecture for 3-D discrete wavelet transform," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 2, pp. 286–296, 2010.

[7] T. Ebrahimi and D. D. Giusto, "Introduction to special section on JPEG 2000 digital imaging," *IEEE Transactions on Consumer Electronics*, vol. 49, no. 4, pp. 771–772, 2003.

[8] P. R. Schumacher, "An efficient JPEG2000 tier-1 coder hardware implementation for real-time video processing," *IEEE Transactions on Consumer Electronics*, vol. 49, no. 4, pp. 780–786, 2003.

[9] B. F. Wu and C. F. Lin, "An efficient architecture for JPEG2000 coprocessor," *IEEE Transactions on Consumer Electronics*, vol. 50, no. 4, pp. 1183–1189, 2004.

[10] Q. Huang, R. Zhou, and Z. Hong, "Low memory and low complexity VLSI implementation of JPEG2000 codec," *IEEE Transactions on Consumer Electronics*, vol. 50, no. 2, pp. 638–646, 2004.

[11] M. Gangadhar and D. Bhatia, "FPGA based EBCOT architecture for JPEG 2000," *Microprocessors and Microsystems*, vol. 29, no. 8-9, pp. 363–373, 2005.

[12] H. C. Fang, Y. W. Chang, T. C. Wang, C. J. Lian, and L. G. Chen, "Parallel embedded block coding architecture for JPEG 2000," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 9, pp. 1086–1097, 2005.

[13] A. K. Gupta, S. Nooshabadi, D. Taubman, and M. Dyer, "Realizing low-cost high-throughput general-purpose block encoder for JPEG2000," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 7, pp. 843–858, 2006.

[14] J. S. Chiang, C. H. Chang, C. Y. Hsieh, and C. H. Hsia, "High efficiency EBCOT with parallel coding architecture for JPEG2000," *EURASIP Journal on Applied Signal Processing*, vol. 2006, Article ID 42568, 14 pages, 2006.

[15] C. Zhang, Y. Long, and F. Kurdahi, "A scalable embedded JPEG 2000 architecture," *Journal of Systems Architecture*, vol. 53, no. 8, pp. 524–538, 2007.

[16] K. Varma, H. B. Damecharla, A. E. Bell, J. E. Carletta, and G. V. Back, "A fast JPEG2000 encoder that preserves coding efficiency: the split arithmetic encoder," *IEEE Transactions on Circuits and Systems I*, vol. 55, no. 11, pp. 3711–3722, 2008.

[17] C. Xiong, J. Hou, Z. Gao, and X. He, "Efficient fast algorithm for MQ arithmetic coder," in *Proceedings of the IEEE International Conference onMultimedia and Expo (ICME '07)*, pp. 759–762, July 2007.

[18] R. Wang, B. Li, H. Jiang, and H. Cao, "High-throughput and hardware-efficient architecture of MQ arithmetic coder," in *Proceedings of the 11th IEEE Singapore International Conference on Communication Systems (ICCS '08)*, pp. 783–787, November 2008.

[19] M. Rhu and I. C. Park, "A novel trace-pipelined binary arithmetic coder architecture for JPEG2000," in *Proceedings of the IEEE Workshop on Signal Processing Systems (SiPS '09)*, pp. 243–248, October 2009.

[20] K. Liu, Y. Zhou, Y. Song Li, and J. F. Ma, "A high performance MQ encoder architecture in JPEG2000," *Integration, the VLSI Journal*, vol. 43, no. 3, pp. 305–317, 2010.

[21] K. Sarawadekar and S. Banerjee, "An Efficient pass-parallel architecture for embedded block coder in JPEG 2000," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 6, pp. 825–836, 2011.

[22] K. Sarawadekar and S. Banerjee, "VLSI design of memory-efficient, high-speed baseline MQ coder for JPEG 2000," *Integration, the VLSI Journal*, vol. 45, no. 1, pp. 1–8, 2012.