A High-Performance JPEG2000 Architecture

Kishore Andra, Chaitali Chakrabarti, and Tinku Acharya, *Senior Member, IEEE*

***Abstract*—JPEG2000 is an upcoming compression standard for still images that has a feature set well tuned for diverse data dissemination. These features are possible due to adaptation of the discrete wavelet transform, intra-subband bit-plane coding, and binary arithmetic coding in the standard. In this paper, we propose a system-level architecture capable of encoding and decoding the JPEG2000 core algorithm that has been defined in Part I of the standard. The key components include dedicated architectures for wavelet, bit plane, and arithmetic coders and memory interfacing between the coders. The system architecture has been implemented in VHDL and its performance evaluated for a set of images. The estimated area of the architecture, in 0.18-μ technology, is 3-mm square and the estimated frequency of operation is 200 MHz.**

***Index Terms*—Binary arithmetic coding, bit-plane coding, JPEG2000, system architecture, wavelet transform.**

I. Introduction

THE DIFFERENCES in the computing power, bandwidth and memory of wireless and wired devices, as well as emergence of diverse imaging application requirements, have made resolution scalability and quality scalability essential in today's still image compression standards. Although these properties can be attained with present JPEG, they cannot be achieved in a single bit stream [1]. To overcome these drawbacks, the upcoming still-image compression standard JPEG2000 has been designed [2]. Error resilience, manipulation of images in compressed domain, acceptable performance even at very low bit rates ($\sim$0.1 bpp), region-of-interest coding, lossy and lossless performance using same coder, noniterative rate control, etc., are some of the other important features of the JPEG2000 standard. All these features are possible due to adaptation of the discrete wavelet transform (DWT) and intra-subband entropy coding along the bit planes using a combination of a bit plane coder (BPC) and binary arithmetic coder (BAC) in the core algorithm.

All three core blocks namely, the DWT, BPC, and BAC blocks are computationally, as well as memory, intensive. The DWT algorithm is a typical "DSP algorithm" with a small set of arithmetic operations performed continuously with symmetrical data access (read) and generation (write) pattern. These properties makes it amenable for implementation using DSP and media processors or even dedicated hardware. In contrast to DWT, both BPC and BAC are control intensive (i.e., contain substantial branching conditions) with few arithmetic operations and are performed bit-plane wise. As a result, they cannot be efficiently implemented on DSP or media processors. Even though inherent parallelization is present in the entropy-coding algorithm, the parallel paths are complex, data dependent, and are defined only at run time. Further, since the JPEG2000 kernel will be a part of digital cameras, scanners, printers, wireless devices with multimedia capabilities, etc., it is important that the kernel be area, time, and power efficient. Thus, we conclude that while DWT can be implemented by DSP or media processors, specialized implementations are needed for the BPC and BAC coders. Recently, Analog Devices has introduced a JPEG2000 co-processor [7], further supporting the hardware implementation paradigm.

The core algorithm in JPEG2000 has been defined in Part I of the standard and any JPEG 2000 system has to minimally comply with the Part I specification. In this paper, we propose an integrated architecture to implement the encoding and decoding for the JPEG2000 part I coder. The architecture primarily consists of three modules: 1) the DWT module; 2) the BPC module; and 3) the BAC module. The modules interface with each other via memory and buffers. The DWT module is capable of performing (5,3) filter in the lossless mode and (9,7) filter in the lossy mode on an 8-bit input data. Three pairs of BPC and BAC modules are used to reduce the time required for entropy coding. The architecture has been implemented in VHDL and its performance has been evaluated. The estimated area of the architecture, in 0.18-μ technology, is 3-mm square and the estimated frequency of operation is 200 MHz.

The rest of the paper is organized as follows. In Section II, we describe the JPEG2000 Part I coder in brief. The proposed system-level architecture is discussed in Section III. The DWT, BPC, and BAC algorithms and proposed architectures are presented in Sections IV–VI, respectively. The performance of the architecture is discussed in Section VII and the paper is concluded in Section VIII.

II. JPEG2000 Basics

The encoder proposed for the JPEG2000 Part I standard is explained using the block diagram in Fig. 1. During encoding, an image is split into rectangular structures called tiles. The tiles are coded separately as if they are different images. The encoding steps are summarized below. For more details, please refer to [2]. Note that decoding is symmetric to encoding and can be achieved by performing the encoding steps in the reverse order.

Wavelet Transform: In the first step, the DWT is applied on the tile to decompose it into a number of wavelet subbands. Recently, a new methodology called lifting [8], [9] has been proposed to perform the DWT. Lifting enables the DWT to be computed using a series of banded matrix multiplications. In Part I

Manuscript received August 7, 2001; revised September 30, 2002. This paper was recommended by Associate Editor A. Tabatabai.

The authors are with the Department of Electrical Engineering, Telecommunications Research Center, Arizona State University, Tempe, AZ 85287-5706 USA (e-mail: Kishore_andra@yahoo.com; chaitali@asu.edu; tinku_acharya@ieee.org).

Digital Object Identifier 10.1109/TCSVT.2003.809834

1051-8215/03$17.00 © 2003 IEEE

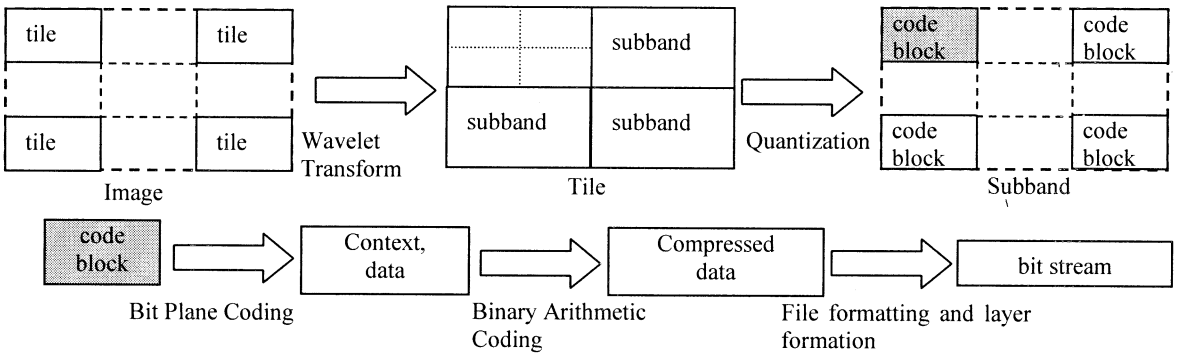

Fig. 1. Block diagram of the JPEG2000 encoder.

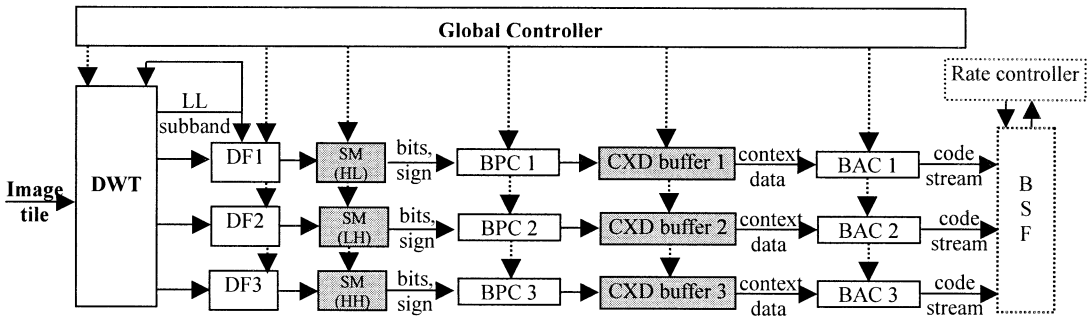

Fig. 2. Proposed architecture for JPEG2000 encoder.

of the JPEG 2000 standard, lifting-based implementation of the (5,3) filter is prescribed for lossless encoding and that of the (9,7) filter for lossy encoding.

Quantization: The wavelet coefficients in each subband are scalar quantized if lossy compression is required. In JPEG 2000, uniform scalar quantization with deadzone at the origin is applied to the subband samples for lossy compression. The quantization step size is determined by the dynamic range of the samples in a subband. It can vary from one subband to another based on the visual models, similar to the specification of Q-table in baseline JPEG.

BPC: The quantized subbands are divided into code blocks. The code blocks are entropy coded along the bit planes using a combination of embedded BPC and BAC.

In JPEG2000, the embedded block coding with optimized truncation (EBCOT) algorithm [10] has been adopted to implement the BPC. This algorithm exploits the symmetries and redundancies within and across the bit planes. It generates the input to the BAC block based on statistics (state information bits that are maintained across the bit planes) of the data coded previously.

BAC: The BPC outputs are entropy coded using BAC to generate the code stream. The MQ coder, which is a derivative of the Q coder [11], [12], has been proposed to implement the BAC. The algorithm is multiplication free. Predetermined probability values are supplied by the standard and are stored in a look up table. The adaptation state machine is also supplied by the standard.

File formatting and layer formation: For each of the code blocks, distortion for a fixed number of bit rates and code size is calculated by a suitable rate control mechanism. The final bit stream is formed based on the available bit rate by means of "layers" which contain incremental contribution from each code block. So even though neither the required resolution nor the required rate is known while encoding, the best possible image of required resolution is generated for a given bit rate.

III. Proposed Systems Architecture for JPEG2000

Here, we propose a systems architecture capable of performing the coding process described in the previous section. The input to the architecture is an image tile and the outputs are three code streams (one for each subband). The division of the image into tiles and formation of the layers at the end of coding process are handled by software. The block diagram of the proposed architecture is shown in Fig. 2.

The architecture primarily consists of a DWT module, three pairs of BPC and BAC modules and three data formatters (DF). It also consists of: 1) three subband memory (SM) blocks between the DWT coder and three BPC coders to store the code blocks formed from the subband data and 2) three CXD buffers between the BPC and BAC modules to store the context and symbol pairs generated by the BPC module. A global controller is present to control the interactions between all these blocks.

The data flow of the architecture is as follows. DWT is applied on the image tile to generate the three high-frequency subbands (HL, LH, HH) and one low-frequency subband (LL) at each level. The LL subband data is used by the DWT module to compute the next level of decomposition while the other three subbands are entropy coded. The subband data is quantized (if required) and broken up into rectangular structures called code blocks. The code blocks are then entropy coded, independently.

Code blocks are written into the SM blocks. Each BPC reads the data from the corresponding SM and writes the context-data pairs into the corresponding CXD buffer. BAC reads from the CXD buffer and generates the code stream for each code block. At the last level, the LL subband is entropy coded using the HL entropy coder pair. The code stream generated is supplied to the bit-stream formation (BSF) tool to form the final bit stream based on the resolution and quality needed. This process is controlled by a rate controller. The proposed architecture does not handle the rate controlling or the BSF tool; a host processor or an ASIC has to perform this function.

The entropy coding of JPEG2000 takes an inordinately long time. For instance, to entropy code a $N \times N$ code block, with one bit position being coded in each cycle, $N \times N \times 16 \times 3$ cycles are required. This is because the internal precision is 16 bits for lossless performance [4] and BPC performs the coding in three passes [10]. On top of this, the BAC requires at least two table lookups and two additions per bit [2]. The entropy coder still requires a few million cycles even if the bypass mode, proposed in [2] to speed up the entropy coding, is used. In contrast, the DWT requires only about 300 000 cycles to code a 128×128 block to five levels.

Fortunately, the time required for encoding can be reduced if multiple hardware modules are provided since the code blocks are entropy coded independently. For instance, for the case where the DWT coder and the entropy coder work in sync (i.e., while the DWT coder operates on level i, the entropy coder operates on coefficients of level $i-1$), at the most $4 \times 3 + 1$ hardware modules are needed at the first level.

This is because in the code block structure that we have considered, each subband is split into four code blocks at the first level and the whole subband forms a code block in the rest of the levels. In such a scheme, during entropy coding of levels 2, 3, and 4, three out of 13 modules are needed. Also, each hardware module costs ~6000 gates + memory interface. So the choice of the number of hardware modules is clearly a balance between the time constraint and the area constraint. In our design, we chose three hardware modules—one for the HH subband, one for the LH subband, and one for the HL and LL subbands. This makes the memory interface between the DWT coder and the entropy coder easier to handle and at the same time reduces the computation time by a factor of three.

The decoder architecture is similar to the encoder architecture with data flow in the opposite direction. The BSF tool is replaced by a code stream formation tool. The CXD buffer is replaced by a single register to hold the context. This is because the bit plane decoding cannot proceed before the data is obtained from the binary arithmetic decoder.

Next, we briefly describe the different architectural components.

A. DWT Module

The architecture performs lifting-based DWT/IDWT for the (5,3) and (9,7) filters. The transform is computed in column-row fashion one level at a time (i.e., with no interleaving between the levels). Symmetric extension is used at the boundaries. The key components are: 1) a data path with two adders, one shifter, one multiplier; 2) a memory block of size equal to the tile size (with a read port and a write port); and 3) a controller (counter, signal generator, address generator). The architecture generates an output from a lifting step every cycle. Details of the architecture are given in Section IV.

B. Data Formatter

Data Formatter (DF) carries out the conversion between two's complement data that is generated by the DWT module and the sign magnitude data that is required by the BPC module. Further, DF also determines the most significant bit plane (i.e., the first bit plane which contains a "1") of each code block. The BPC starts coding from the significant bit plane. Quantization, if needed, can be performed by DF. As mentioned earlier, scalar quantization is prescribed in the standard and this can be handled with a multiplier if the quantization step sizes are known for each subband.

In the decoder, DF performs the conversion from sign-magnitude form to two's complement form. The significant bit-plane value for each code block is supplied by the encoder. The bit planes from the 15th bit plane to the significant plane are filled with zeros by the DF. Inverse quantization, if needed, is performed by DF.

C. SM

The data formatters write sign magnitude data to the SM blocks. The bit-plane coders read the data bits and sign bit from the SM blocks along the strips. A novel memory structure that can handle word-in–bit-out format combined with the strip structure required for the BPC has been designed.

The SM structure is shown in Fig. 3. Each row of the SM contains four words, where the four words are obtained from four consecutive rows along a column. Each word is 16-bits wide and so each row is $4 \times 16 = 64$ bits wide. The corresponding bits of each word are grouped together as shown in Fig. 3. If the maximum number of rows and columns of a code block are R and C, respectively, then memory structure would have $(R/4) \times C$ rows with 64 bits per row. It should be noted that all the elements in a four-row strip are stored in consecutive rows.

D. BPC Module

The architecture to carry out the BPC is based on the EBCOT algorithm. The encoder architecture consists of: 1) combinational logic blocks that transform the state information into input to the BAC module; 2) three memory blocks to hold the state information bits; 3) five registers (of various sizes and functionality) to hold the state and magnitude bits; and 4) a 24-state controller to control all the blocks. The decoder is very similar to the encoder architecture. Details of the architecture are given in Section V.

E. CXD Buffer

The CXD buffer is a FIFO with a read and a write port as shown in Fig. 4. Each entry contains a context data bit pair (6 bits). The length of the buffer needs to be as large as possible to account for speed difference between the BPC and BAC coders. A buffer with 128 entries has been used; the number of

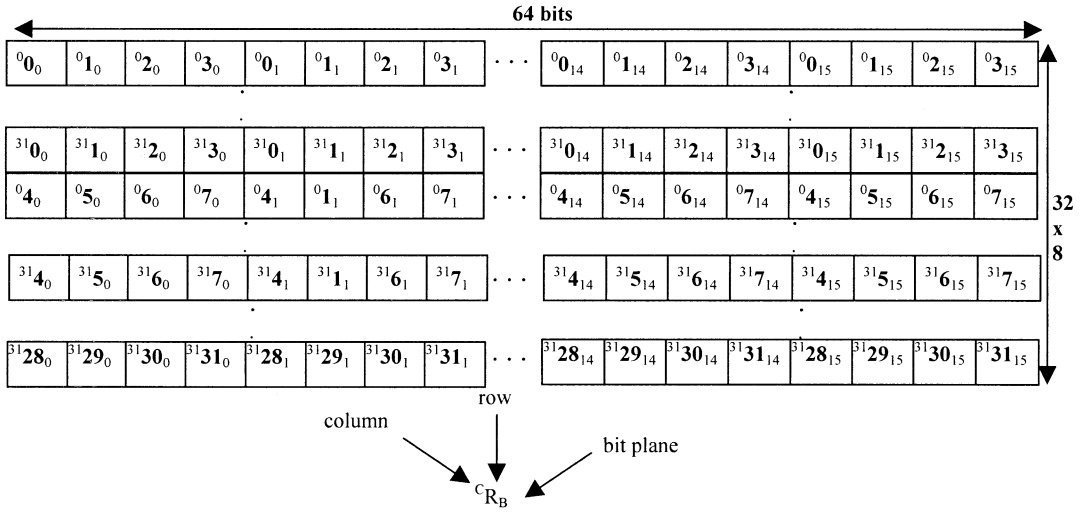

Fig. 3. SM structure to hold 32 rows and 32 columns.

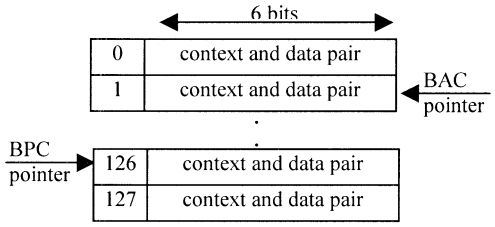

Fig. 4. CXD buffer structure.

entries was determined with experimentation. The global controller uses two pointers namely, a BPC pointer and a BAC pointer, to keep track of the FIFO stack. The pointers are reset whenever the BAC module is initialized or reinitialized. If the buffer is not large enough, the BPC module ends up following the BAC module with the buffer behaving like a register.

F. BAC Module

The architecture to implement the BAC module is based on the MQ coder. The architecture consists of a: 1) 16-bit adder; 2) registers (various sizes and functionality); 3) a logic block that helps in the adaptation process; 4) two memory blocks to perform the table look-up operations; and 5) a controller. The architectural components of the encoder and decoder are the same though the corresponding controllers are completely different. Architectural details are described in Section VI.

IV. Lifting-Based DWT

In JPEG2000, the DWT is implemented using a lifting-based scheme. The lifting-based scheme breaks up the high pass and low pass filters into a sequence of upper and lower triangular matrices, and converts the filter implementation into banded matrix multiplications. Such a scheme has several advantages, including "in-place" computation of the DWT, integer-to-integer wavelet transforms (which are useful for lossless coding), symmetric forward and inverse transform etc.

To be JPEG 2000 (Part I) compliant, the DWT module should be able to support (5,3) filter in lossless mode and the (9,7) filter in lossy mode. We propose an architecture which is capable of performing the above filters using the lifting scheme. The architecture supports both forward and inverse DWT. The architecture is very simple and consists of a processor (two adders, one shifter and a four-level pipelined multiplier), a memory block, and a controller.

A. Precision Analysis

The first step in the design of the architecture is to determine the number of bits required for satisfactory lossy and lossless performance in the fixed point implementation. The study was conducted in [4] on three gray-scale images—baboon, barbara, and fish—each of size 512×512 for five levels of decomposition. The results were validated with 15 gray-scale images from the USC-SIPI database [13]—5.2.08–10, 7.1.01–04, 7.1.06–10, boat, elaine, ruler, and gray21 from the Miscellaneous directory.

From the study, we concluded that 10 bits are required to represent the coefficients and 14 (16) bits are required to represent the signals for lossy (lossless) performance. A rounding operation (all the number are rounded toward $+\infty$) is employed in the product terms and the internal precision is maintained with 14 (16) bits for lossy (lossless) performance. Based on this precision analysis, the size of the data path units is chosen to be 16-bits wide.

B. Proposed Architecture for Lifting-Based DWT

The proposed architecture performs the DWT in column-row fashion one level at a time. We chose this method over a recursive pyramid algorithm (RPA) [6] based method that generates coefficients of multiple levels in an interleaved fashion since the entropy coder works on coefficients level by level and use of RPA-based method would result in an unnecessary increase in the latency of the system.

The architecture performs one lifting step (i.e., calculating high pass terms from the low pass terms or vice versa) in each iteration. So, the (5,3) filter requires four iterations (two lifting steps along each dimension) while the (9,7) requires nine iterations (four lifting steps in each dimension and one modified scaling step).

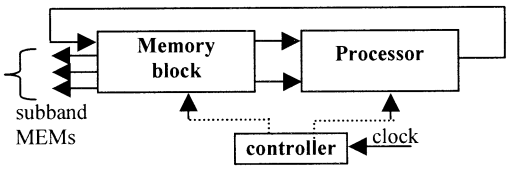

Fig. 5. Proposed architecture for the lifting-based DWT.

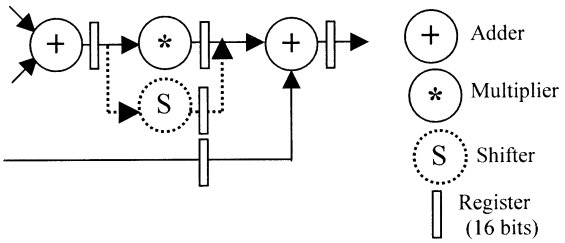

Fig. 6. Processor structure for DWT computation.

The architecture shown in Fig. 5 consists of a processor (two adders, one multiplier, one shifter), a memory and controller blocks. The processor reads in the data from the memory block and writes back into it after the transform computation. The controller generates the input/output signals for both the processor and the memory modules. The data flow remains the same for both DWT and IDWT.

In [3], we had presented an architecture with four processors which generated coefficients from two subbands in each cycle. The four-processor architecture was not used here because the entropy coder following the DWT module would not be able to handle such high data rates.

1) Processor: All the lifting steps for DWT and IDWT are of the form

$$x_{i,\,\mathrm{new}} = a(x_{i-1} + x_{i+1}) + x_{i,\,\mathrm{old}}.$$

The multiplication factors for the (5,3) filter are multiples of two, so multiplication can be replaced with a shift operation. To perform the above general structure, a processor with two adders, a shifter, and a multiplier is required (see Fig. 6). The registers between the units and at the input are not shown.

Based on the precision analysis, the adder and shifter are chosen to be 16-bits wide. The multiplier performs a signed 16 $\times$ 10 multiplication. A rounding operation is performed on the product so that multiplier output is 16-bits wide. The shifter is capable of shifting 1 or 2 bits, right or left in a single cycle. Further, we assume that the adder has a unit delay and that the multiplier is pipelined to four levels, with each stage of pipe having a delay equal to adder delay.

2) Memory Block: To support the proposed processor architecture with in-place style of memory accesses, a memory block of size $N \times N$ (where N is the size of the tile row/column) is required. The memory has to support two read accesses and a write access per cycle and so we chose a dual port memory with two access per cycle on read port.

While encoding, the three SM blocks have to be written from the DWT memory block. So, three read accesses are required per cycle. While decoding, the three SM blocks write into DWT memory block, so three write accesses are required per cycle. However, data is not generated or consumed by the BPC modules simultaneously. So, the global controller can perform the operations in a staggered manner thereby limiting the access requirement to two read accesses and one write access per cycle.

3) Controller: The controller consists of three blocks: a counter, a signal generator, and an address generator.

- *Counter*—keeps track of the number of elements in a row, number of rows, and number of levels processed. It also keeps track of the total number of elements and total number of rows at each level that needs to be processed.
- *Signal generator*—generates the control signals for the processor, the address generator, and the memory block using state machine with six states. The states are changed in a sequential order based on the counter input, the latency of the data path units, and the specific filter being used.
- *Address generator*—performed such that in-place computation (i.e., the old values are overwritten with the updated values instead of using new memory locations) is performed. The generator logic is simple as the size of the subbands decrease/increase while performing DWT/IDWT by a factor of two in each dimension. The address generation is achieved with two adders and a shifter.

C. Timing

If Ta, Ts, and Tm are the delays of the adder, shifter, and multiplier, respectively, then the latency for each iteration is—$(2*Ta+Ts+1)$ for the (5,3) filter and $(2*Ta+Tm+1)$ for the (9,7) filter. So, the total time to finish a iteration (assuming a $N \times N$ block) is *latency* $+ N \times N/2$. Recall that the (5,3) filter requires 4 iterations and the (9,7) filter requires eight iterations and a modified scaling step. So, the total time required to calculate one level of transform on a $N \times N$ block is $(2 * Ta + Ts + 1 + N \times N/2) * 4$ for the (5,3) filter and $(2 * Ta + Tm + 1 + N \times N/2) * 8 + (Tm + 1 + N \times N/2)$ for the (9,7) filter.

V. Bit-Plane Coding

In this section, we briefly describe the embedded block coding with optimized truncation (EBCOT) algorithm followed by the proposed architecture.

A. EBCOT Algorithm

The EBCOT algorithm is summarized here for the sake of completeness. Each bit plane is coded in three passes: *significance pass* (SP), *magnitude refinement pass* (MRP), and *clean up pass* (CP). In each pass, only a part of the bit plane is coded and each bit position is coded only once by one of the three passes. The BPC works on strips of four elements along the rows. The code block scan is carried from left to right. Two modes of coding, namely *"regular"* and *"vertical causal"* (VC), are possible [2]. The proposed architecture assumes VC mode, although the *regular* mode can easily be supported at the expense of extra memory.

The BPC requires four-state information bits and 1 magnitude bit (v) for each bit position. The state information bits determine

in which pass each bit is coded and are used in the generation of context and data bits. The four-state information bits are as follows:

1) *Significance bit* (σ)—This bit is set whenever the magnitude bit of the corresponding subband coefficient is "1" for the first time.
2) *Visited once bit* (η)—This bit is set when the bit is coded in a pass.
3) *Magnitude refinement coded bit* (σ')—This bit is set the first time the magnitude refinement primitive (explained below) is used.
4) *Sign bit* (χ)—This bit is 0 for positive numbers and 1 for negative numbers and is obtained from the sign-magnitude representation of the subband values.

All the state bits except for η bits are maintained across all the bit planes. The η bits are reset at the end of each bit plane. It should be noted that σ and χ for the neighbors that are outside the strip are assumed to be zero. All the three passes make use of one or more of the following four primitives—zero coding (ZC), sign coding (SC), magnitude refinement coding (MRC), and run length coding (RLC). All the primitives use context which is a binary representation of the neighboring pixels. Context for the data in bit position X is formed from the eight neighboring values (D_0–D_3, H_0, H_1, V_0, V_1) in the σ matrix as shown in Table I.

1) Primitives

- *ZC*—uses nine (contexts 0–8) out of possible 19 contexts. The data is the magnitude of the bit position X.
- *SC*—uses five contexts (contexts 9–13) and is a two-step process. In the first step, the σ and χ of the horizontal and vertical *"contributions"* and a *"XOR"* bit [2]. In the second step, context is formed from the two contributions and data is formed by *exclusive OR* operation of the *sign bit* and the *XOR bit*.
- *MRC*—uses three contexts (contexts 14–16). The contexts are formed based on whether it is the first time the magnitude refinement is being used on a certain position and its eight immediate neighbors. The data is the magnitude bit.
- *RLC*—uses the remaining two contexts (contexts 17–18). It is invoked only at the beginning of a strip if the σ of all the eight neighbors is 0 for all the bits in a strip. If none of the bits in the strip become significant, context 17 with data $= 0$ is used. On the other hand, if any bit does become significant, context 17 with data $= 1$ is used. This is followed by MSB and LSB of zero index (ZI) (00–11) of the bit position which contains the "1" bit. Context 18 is used for ZI bits.

2) Coding passes: As mentioned earlier, each bit plane is coded in three passes. The first bit plane is coded just with the CP. In the SP, all the bits whose $\sigma = 0$ and have at least one of the immediate eight neighbors with $\sigma = 1$ are coded using ZC primitive. If the bit becomes significant, the SC primitive is used and σ of the bit being coded is set to 1. When ZC is applied, the corresponding η is set. In MRC, all the bits with corresponding $\eta = 0$ and $\sigma = 1$ are coded using MR primitive.

TABLE I
NEIGHBORHOOD USED FOR CODING THE INPUT BIT AT POSITION X

D_0	V_0	D_1
H_0	X	H_1
D_3	V_1	D_2

The corresponding σ' bit is set to 1. In CP, if $\eta = 0$ and $\sigma = 0$ for the first element in the strip, the RLC condition is checked. If the RLC condition (mentioned in the RLC primitive) is satisfied, the RLC primitive is used. If one of the bits in the strip become significant, then SC is used and σ is set for that bit. This is followed by application of ZC + SC for the rest of the bits in the strip. If the RLC condition is not satisfied, then ZC + SC is used for all the elements with $\eta = 0$ and $\sigma = 0$.

B. BPC Encoder Architecture

The block diagram of the proposed architecture for the EBCOT encoder is shown in Fig. 7. The architecture consists of the following key building blocks: 1) three combinational logic blocks to determine the contexts for ZC, MRC, and SC (the contexts for RLC are hard coded); 2) five shift registers of varying sizes and functionality to store the variables σ, η, σ', υ, χ; and 3) three memory blocks (for the three state bits σ, η and σ') each of size 32×4. The magnitude (υ) and sign (χ) bits are obtained from the SMs. In addition to these key building blocks, there is a multiplexer (MUX) to select the right context for the bit to be coded from the various contexts based on the coding pass, a counter to keep track of the number of strips processed and also the coding *pass* being used, and finally a controller. The functionality of these building blocks is described below.

1) *Combinational logic blocks*

The tables provided in [2] to form the context for each of the primitives can be expressed in terms of simple logic operations. These logic operations are mapped into gates and are placed in the combinational logic blocks. For more details, please refer to [14].

- *ZC context block:* The input to this block is σ of the eight neighbors of the bit being coded and magnitude of the bit position. The output is the ZC context and data pair.
- *SC context block:* Inputs to this block are the $\sigma(v_0, v_1, h_0, h_1)$ and $\chi(sv_0, sv_1, sh_0, sh_1)$ from σ and χ registers respectively. The output is the SC context and the sign data bit.
- *Magnitude refinement coding context block:* The two inputs to this block are σ' and the *nhood0* bit (which indicates if the eight neighbors are all zeros) from the σ register. The output is context and data pair for MRC.
- *RLC contexts:* The four possible contexts are—RLC condition satisfied and strip is all 0's, RLC condition satisfied and the strip contains at least one "1" bit. The latter case is followed by context 18 and two bits of the Zero Index (supplied by the υ register) of the bit position that is "1." These contexts and data pairs are hard coded.
- *Registers:* There are five registers of varying sizes to store the state variables (see [14]). All the registers are capable

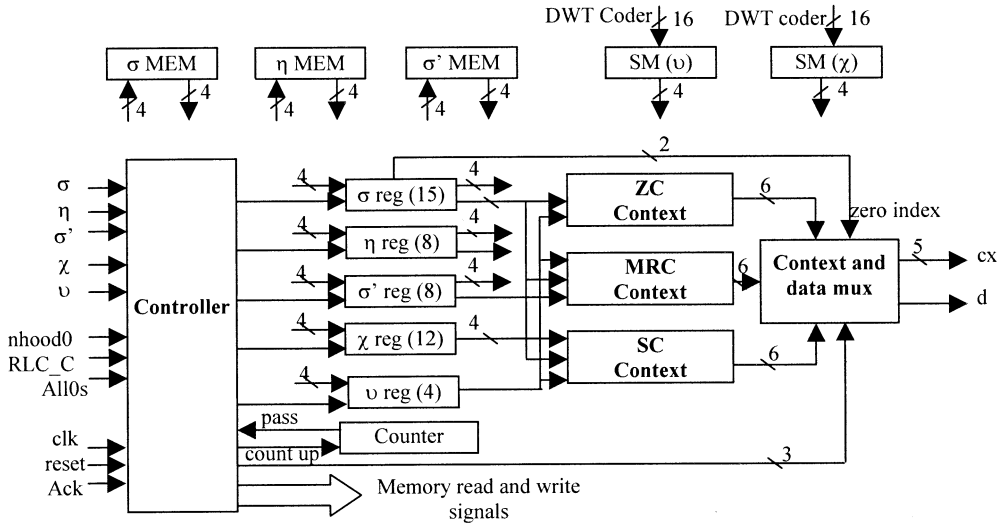

Fig. 7. Proposed architecture for EBCOT encoder.

of 1 bit left shift. For initialization and RLC, the σ register and χ register are capable of 5-bit and 4-bit left shift, respectively.

The σ, η and σ' registers have an "update" position, where a "1" is written to set the corresponding state variable, when required. Data from the (corresponding) memory is written into four least-significant-bit positions in the registers. But data can be read from different positions of the registers and written into memory. The registers are read and written at the end of coding of each strip.

- *Memory blocks:* Three memory blocks each of size 32×4 are used to store the state variables. The subband MEMs supply the υ and χ bits. The DWT module writes into the subband MEMs. The other three memories are written by the corresponding internal registers and they have a single read and write port, as shown in Fig. 7.
- *Context and data mux:* The multiplexer chooses the context from the outputs of ZC context block, SC context block, MR context block, or the hard coded RLC contexts (17–18). The data bit is chosen from the υ, sign data, χ, hard coded RLC data bits (0,1), or the ZI (MSB, LSB) bits. The mux is controlled with a 3-bit word. Based on the pass being performed, the controller generates the control word.
- *Counter:* It keeps tracks of the element in the strip being coded, the number of strips coded in each pass, the pass being processed, and the bit plane being processed. This information is required for the state machine.
- *Controller:* The state machine consists of 24 states. The state machine can be divided into five phases—initialization phase, ZC and SC phase, MRC phase, RLC phase, and termination phase. In initialization phase, the registers are reset and σ and χ registers are initialized as required. Based on the pass, one of the primitives is performed. The ZC and SC phase are performed during SP and during CP when some conditions are satisfied. The context generated by ZC context block is used in this phase. The MRC phase is performed during MRP. The context generated by MRC context block is utilized in this phase. The RLC phase is invoked during CP. If the RLC condition is satisfied and strip contains all zeros, then strip is not coded further. The contexts for RLC are hard coded. Finally, termination phase is invoked at the end of coding a strip. Using the counter information, the next coding step is determined. Please refer to [14] for a detailed discussion of the state diagrams.

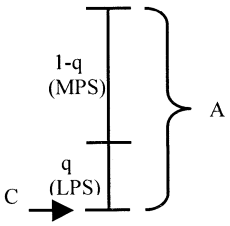

Fig. 8. Interpretation of the parameters in the MQ coder.

C. BPC Decoder Architecture

The architecture for the decoder remains almost the same as the encoder except for small changes to υ and χ memories and registers. For instance, data from υ and χ MEMs is written out. Also, while in the encoder, both the bits of zero index are known before RLC is started, in the decoder, the bits are obtained one at a time. The resulting state machine is slightly different, although the number of states required still remains the same.

VI. Binary Arithmetic Coding

A. MQ Coder Basics

The basic principle of an arithmetic coder is to recursively subdivide the 0–1 interval based on the conditional probability of the input symbols. The MQ coder uses the convention shown in Fig. 8: the current interval is A, the starting point of the interval is C (which also holds the code string), and the probability of LPS occurring next is "q." So to code a MPS, the code string has to be changed by adding the sub-interval of the LPS. Nothing needs to be done to code a LPS. From Fig. 8, it can be

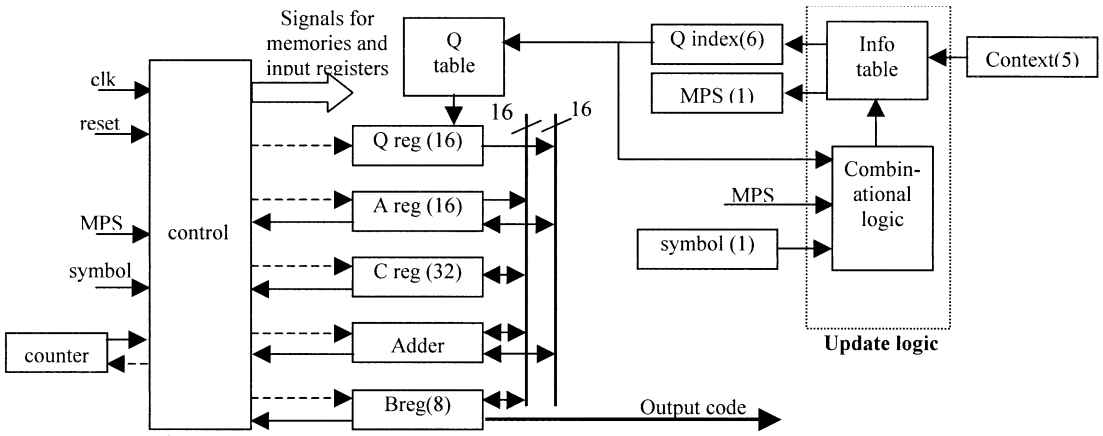

Fig. 9. Block diagram of the MQ coder.

observed that the interval and the starting point have to changed for the MPS and LPS cases as follows:

$$A = A * (1 - q), \qquad C = C + A * q \qquad \text{(for the MPS case)}$$
$$A = A * q, \qquad C = C \qquad \text{(for the LPS case).}$$

By making sure that A is close to unity, the A*q value is approximated to a value "Q." This simplifies the above equations to $A = A - Q$; $C = C + Q$ for the MPS case and $A = Q$ for the LPS case.

The statistics required to determine the Q value, given a context and symbol (generated by the BPC module), are maintained with the help of two look up tables. The index of the first lookup table is the context and each entry in the table is the MPS for that context and index to second table. The second table contains the pre-computed Q values, provided by the standard.

It can be seen from the MQ coder algorithm [2] that to code a symbol, a minimum of two table lookups and two additions are required. This shows that the MQ coder is inherently slow. To speed up the bit-plane coding, by-pass mode is proposed in the JPEG2000 standard. In this mode, starting with the fifth bit plane, BAC coding is bypassed for symbols generated in significance and magnitude refinement passes. We have implemented the by-pass mode in the proposed architecture.

B. BAC Encoder Architecture

The proposed encoder architecture, shown in Fig. 9, consists of: 1) a 16 bit adder to perform the arithmetic operations and comparison; 2) a combinational logic block (part of the update logic "UL") to update the Q-index and MPS sense; 3) a counter which is used to keep track of the number of code bits generated; 4) two memories to store the "*Info table*" and the "*Q table*"; and 5) eight registers—A (to hold the interval), C (code string), B (the last byte generated), Q-index, Q, CX (context), MPS and symbol (data). All the units are controlled by a controller which also generates the read/write signals for the memories. The data transfer between the adder and the registers is carried out using two 16–bit data buses.

- *Adder:* The adder is used to calculate $A - Q$, $C + Q$, and also to perform the comparison operation $(A - Q) < Q$. Since the C register is 32-bits wide and the Q register is 16 bits wide, a 32-bit adder is required to compute $C + Q$. Since experimental results showed that only 30% of the time a carry is propagated to the 16 MSBs of the C register, the 32-bit addition is handled in two steps using a 16-bit adder.
- *Update Logic (UL):* The UL consists of a combinational logic block and the Info table. To generate the new Q-index and MPS, the present Q-index, MPS, and Symbol are supplied to the logic block. If a renormalization process is performed, the new information is written into the Info table. The new data is generated based on the state machine [2].
- *Counter:* The Counter is initialized to 12 (to account for the spacer bits) at the beginning of coding [2], [12]. Whenever the count becomes zero, the data present in B is written out and a new byte of data is written to B from C. Then, based on whether bit stuffing is required or not, the counter value is set to 7 or 8.
- *Registers:*

  A *register*—It is 16 bits wide and capable of a 1-bit left shift. The MSB ($A[15]$) is supplied to the controller. This bit helps in determining if renormalization has to be performed. This register can be written by the adder and the Q register.

  C *register*—is 32-bits wide and is capable of one bit left shift. The 28th bit is used by the controller to verify if a carry is available for the byte in the B register. It can be written by the adder. Various arrangements are required to reset parts of register during reading the code byte and during the "*flush*" procedure (used to terminate the coding) [2]. Also, it should be noted that only 16 bits at a time are accessed.

  B *register*—It is 8-bits wide and can be written by the adder or the C register. An *All1* detector is built into the register to indicate to the controller if bit stuffing is required. The code stream is written to the external memory as required.

  Other registers—All the other registers Q (16 bits), Q-index (6 bits), Context (5 bits), MPS (1 bit), and Symbol (1 bit) are just simple registers with no extra functionality. They do not generate any inputs to the controller.

  Controller: It generates control signals for all the registers and the memories. It is driven by a state machine which has 36 states. The by-pass mode is supported by the controller.

TABLE II
CYCLES (IN MILLIONS) REQUIRED TO ENCODE AND DECODE WITH AND WITHOUT BY PASS MODE

Encode	(9,7) filter		
	regular	bypass	%difference
baboon	7.887	6.687	15.22
barbara	6.944	5.930	14.61
fish	5.789	5.020	13.29
elaine	7.059	6.096	13.65

Decode	(9,7) filter		
	regular	bypass	%difference
baboon	13.001	9.727	25.18
barbara	11.864	8.880	25.15
fish	9.970	8.053	19.23
elaine	11.998	8.053	32.88

TABLE III
HARDWARE REQUIREMENT OF THE PROPOSED ARCHITECTURE

	datapath (2-input nand gate equivalent)	Memory (Kbits)
DWT	3500	128x128x16=256
SM	-	(32x8x64)x3=48
BPC	2000 (encoder/decoder)	(32x4x3)x3=1.125
CXD	-	(128x6)x3=2.25
BAC	3200 (encoder)/ 2500 (decoder)	(19x7+47x16)x3=2.6

TABLE IV
DIFFERENCES BETWEEN ADV JP2000 AND THE PROPOSED ARCHITECTURE

	ADV JP2000	Proposed architecture
Tile size	256x256	128x128
Code block size (max.)	64x64	32x32
Wavelet filters used	(5,3) lossy and lossless	(5,3)-lossless; (9,7)-lossy
Levels of DWT	3	5
Entropy coder pairs	1 (assumed from figure)	3

C. BAC Decoder Architecture

The proposed architecture is very similar to the encoder architecture with the following exceptions. The code byte is written into the B register instead of reading from it. The symbol register is not required; instead, MPS or a inverted version of it (LPS) is written out as required. The counter needs to count down from eight, unlike in the encoder counter which has to count down from 12 initially. All the arithmetic operations are performed on the 16 MSBs of the C register. To undo the bit stuffing, an incrementing function is required. The controller state machine consists of 28 states.

VII. PERFORMANCE OF THE PROPOSED SYSTEMS ARCHITECTURE

We have conducted the performance analysis of the proposed architecture with four images (baboon, barbara, fish and elaine) of size 512 $\times$ 512. The input to the architecture is a 128 $\times$ 128 image tile. DWT is carried out to five levels. The maximum size of the code block is fixed at 32 $\times$ 32. After the first level of encoding, each of the three subbands are of size 64 $\times$ 64. Each subband is split into four code blocks, each of size 32 $\times$ 32. For the rest of the levels, the whole subband is treated as a code block since the subband is of size 32 $\times$ 32 or smaller.

The number of cycles (in millions) required to encode and decode the images with and without bypass mode for the (9,7) filter is given in Table II. It can be seen that the speed up with by pass mode is around 15% for encoding and 25% for decoding. Note that the number of cycles required for decoding is significantly higher than that required by encoding. This is expected since during encoding, the BPC and BAC coders work independently most of the time due to the CXD buffer, while during decoding, the coders cannot work independently. Similar results have been obtained for the (5,3) filter.

The system architecture has been implemented in VHDL. We have synthesized the data path units in the DWT coder, BPC encoder and decoder, and BAC encoder and decoder. The preliminary gate counts (in two input NAND gate equivalents) of the modules and the memory required by each module is given in Table III. The estimated area of the architecture, assuming the control is 20% of data path area in case of DWT, in 0.18-μ technology is 3-mm square and the estimated operation frequency is 200 MHz.

VIII. CONCLUSION

In this paper, we have proposed a systems architecture to perform the new JPEG2000 part I standard for compression and decompression of images. The architecture consists of modules to implement the DWT, BPC, and BAC algorithms and interfacing memory structures. The BPC and BAC modules are implemented by three sets of computation engines. Such a structure was necessary to compensate for the high computational requirements of these two modules. The system level architecture has been implemented in VHDL.

To the best of our knowledge, the only other JPEG2000 architecture is the JPEG co-processor, ADV-JP2000, by Analog Devices [7]. There are several differences between the two architectures some of which have been listed in Table IV. It is stated that the bypass mode has no effect on the coding speed for ADV. This is quite surprising since in our implementation bypass mode speeds up encoding by around 15% and decoding by 25%.

REFERENCES

[1] J. L. Mitchell and W. B. Pennebaker, *JPEG Still Image Data Compression Standard*. New York: Van Nostrand, 1993.
[2] JPEG2000 Final Committee Draft (FCD). JPEG2000 Committee Drafts. [Online]. Available: http://www.jpeg.org/CDs15444.htm.
[3] K. Andra, C. Chakrabarti, and T. Acharya, "A VLSI architecture for lifting based forward and inverse wavelet transform," *IEEE Trans. Signal Processing*, vol. 50, pp. 966–977, Apr. 2002.

[4] ——, "An efficient implementation of a set of lifting based wavelet filters," in *Proc. ICASSP 2001*, pp. 1101–1104.
[5] W. Jiang and A. Ortega, "Lifting factorization-based discrete wavelet transform architecture design," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, pp. 651–657, May 2001.
[6] C. Chakrabarti and M. Vishwanath, "Efficient realizations of the discrete and continuous wavelet transforms: From single chip implementations to mappings on SIMD array computers," *IEEE Trans. Signal Processing*, vol. 43, pp. 759–771, Mar. 1995.
[7] Analog products—ADV-JP2000 [Online]. Available: http://products.analog.com/products/info.asp?product=ADV%2DJP2000.
[8] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting schemes," *J. Fourier Anal. Applic.*, vol. 4, pp. 247–269, 1998.
[9] W. Sweldens, "The lifting scheme: A new philosophy in biorthogonal wavelet constructions," in *Proc. SPIE*, vol. 2569, 1995, pp. 68–79.
[10] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Trans. Image Processing*, vol. 9, pp. 1158–1170, July 2000.
[11] G. L. Langdon, Jr. and J. Rissanen, "Compression of black-white images with arithmetic coding," *IEEE Trans. Commun.*, vol. COM-29, pp. 858–867, June 1981.
[12] J. L. Mitchell and W. B. Pennebaker, "Software implementations of the *Q*-coder," *IBM J. Res. Develop.*, vol. 32, no. 6, pp. 753–774, Nov. 1988.
[13] USC-SIPI image database [Online]. Available: http://sipi.usc.edu/services/database/Database.html.
[14] 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. [Online]. Available: http://enws155.eas.asu.edu:8001/papers.html.

Kishore Andra received the B.Tech. degree in electrical and electronics engineering from J. N. T. University, Anantapur, India, in 1994, the M.S. degree from the Indian Institute of Technology, Madras, India, and the Ph.D. degree from Arizona State University, Tempe, both in electrical engineering, in 1997 and 2001, respectively.

Currently, he is with Maxim Integrated Products, Sunnyvale, CA, working on the design of low-power high-performance mixed-signal ICs.

Chaitali Chakrabarti received the B.Tech. degree in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, India, in 1984, and the M.S. and Ph.D. degrees in electrical engineering from the University of Maryland at College Park in 1986 and 1990, respectively.

Since August 1990, she has been with the Department of Electrical Engineering, Arizona State University, Tempe, where she is currently an Associate Professor. Her research interests are in the areas of low-power systems design, including memory optimization, high-level synthesis and compilation, and VLSI architectures and algorithms for signal processing, image processing, and communications.

Dr. Chakrabarti is currently an Associate Editor of the IEEE TRANSACTIONS ON SIGNAL PROCESSING and the *Journal of VLSI Signal Processing Systems*. She has served on the program committees of ICASSP, ISCAS, SIPS, ISLPED, and DAC. She is a member of the Center of Low Power Electronics (jointly funded by the National Science Foundation, the state of Arizona, and the member companies) and the Telecommunications Research Center. She received the Research Initiation Award from the National Science Foundation in 1993, a Best Teacher Award from the College of Engineering and Applied Sciences, ASU, in 1994, and the Outstanding Educator Award from the IEEE Phoenix section in 2001.

Tinku Acharya (M'96–SM'01) received the B.Sc. (Hons.) degree in physics in 1983 and the B.Tech and M.Tech degrees in computer science from the University of Calcutta, Calcutta, India, in 1983, 1986, and 1989, respectively, and the Ph.D. degree in computer science from the University of Central Florida, Orlando, in 1994.

Since 1997, he has been an Adjunct Professor in the Department of Electrical Engineering, Arizona State University, Tempe. He has been with Elution Technologies, Phoenix, AZ, since June 2002, a start-up company. Previously, he was a Principal Engineering in the Intel Architecture Group with Intel Corporation. Before joining Intel Corporation in 1996, he was a consulting Engineer at AT&T Bell Laboratories (1995–1996), a faculty member at the Institute of Systems Research, University of Maryland at College Park (1994–1995), and held visiting faculty positions at the Indian Institute of Technology (IIT), Kharagpur during 1998–2001. He contributed to over 60 technical papers published in international journals, conferences, and book chapters. He holds 37 U.S. patents and has more than 80 patents pending. His current areas of interest include VLSI Architectures and Algorithms, Electronic and Digital Image Processing, Data/Image/Video Compression and Media processing algorithms in general.

Dr. Acharya was awarded the "Most Prolific Inventor" by Intel Worldwide in 1999 and "Most Prolific Inventor" by Intel Arizona for the past five consecutive years for his significant contribution in intellectual property generation in different areas of development in Intel Corporation. He also served in the U.S. National Body of the JPEG2000 committee of the International Standard Organization (ISO) as the primary member of Intel Corporation. He is a Senior Member of the SPIE Optical Society.