Embedded JPEG2000 Compression/Decompression Engines For Real-Time Processing Of Large-Scale Imagery

Case #: UD-539

US Patent Numbers: US 8,170,333 B2 , US 8,170,334 B2, US 8,170,335 B2,

Inventors: Eric Balster, William Turri, Frank Scarpino, David Walker, Thaddeus Marrara, Benjamin Fortener, Ken Simone, Nick Vicen, David Lucking, David Mundy, Mike Flaherty, Luke Hogrebe

Contact:
Mathew Willenbrink
Director, Technology & Entrepreneurial Partnerships
937-229-3472
Mathew.Willenbrink@udri.udayton.edu
udayton.edu/research/resourcesforresearchers/tpo/index.php

Embedded JPEG2000 compression/decompression engines for real-time processing of large-scale imagery

ABSTRACT:

Description: The invention described is an FPGA-based JPEG2000 compression / decompression engine for real-time processing of large-scale imagery. The invention has a highly scalable architecture that can be implemented with a number of different FPGA processing units, scaling across multiple chipsets, and across multiple accelerator processing cards. The design is completely embedded into FPGA fabric, and can compress very large-scale imagery on the order of hundreds of megapixels. The FPGA-based design allows for a JPEG2000 solution that is low-power and light-weight.

Application: The application that this invention is designed for is large-scale surveillance platforms. However, the commercial applications are numerous, and can range from the digital camera industry to the motion picture industry. Any industry that produces a high volume of imaging data which requires real-time processing, communication, or storage is a potential customer.

Advantages: The advantages of the described invention are numerous, and include:

- Processing of very large image sets, on the order of hundreds of megapixels
- Completely integer-based processing for fast computation
- Multiple chipset, multiple processing card design for highly parallel processing

Detailed Description:

Intent: The intent of the described invention is the fast computation of JPEG2000 compression and decompression algorithms. JPEG2000 compression is approximately 30 times more algorithmically complex than its predecessor, JPEG, and JPEG2000 decompression is approximately 10 times more computationally complex than JPEG. Because of the additional complexity, it is difficult to compress very large images with JPEG2000 in a reasonable amount of time; a problem for real-time imaging systems. The description invention uses FPGAs for parallel processing of the JPEG2000 compression / decompression algorithms for real-time performance.

Additionally, this invention differs from other embedded implementations of JPEG2000 due to its scalability. The proposed design can be implemented in many various architectures to facilitate the processing of imagery in a vast number of frame rates and image sizes.

Applicability: The described invention is applicable in any imaging system that requires real-time JPEG2000 compression and/or decompression. However, the focus of the described invention is the real-time compression of large scaling imagery; i.e. imagery is on the order of hundred of megapixels in size.

Function: Pertinent features of the invention:

1) The described invention is a fully functional JPEG2000 compression engine coded entirely in VHDL (VHSIC Hardware Description Language, IEEE standard 1076), which can easily be implemented on a number of FPGA processing cards which are commercially available. Because it can be implemented on various types of processing platforms, it can be utilized in many different processing environments.

2) The described invention is scalable in many ways. One, the number of processing cores is variable, so many parallel processing paths can exist on one FPGA. Therefore, merely an upgrade to a larger FPGA part will provide additional speed of the compression engine. Two, this design is scalable across multiple FPGAs, and across multiple FPGA acceleration cards. This vast scalability gives the described invention a virtually limitless upper bound in size of imagery that can be compressed, and what timeframe that imagery can be compressed in.

3) The described invention is the first, according to the open literature, JPEG2000 compression engine that utilizes integer only computation. The integer only restriction allows for a simpler design and faster computation.

4) The compression / decompression system has both software and an FPGA processing option. That is, engineers and developers can use software to accomplish compression in a non real-time fashion without utilizing the FPGA card. The software function call to the software or hardware accelerated compression is the same, so developers may work on a design without having to purchasing an FPGA accelerator card.

Inventors' Roles:

William Turri --- Designed memory interface of encoder FPGA. Designed state table processing element.

Frank Scarpino --- Developed original feasibility software.

David Walker --- Developed Tier I Encoder FPGA processing element, Developed Tier I Decoder FPGA processing element, developed pipe-lining of compressor. Developed memory interface.

Thaddeus Marrara --- developed software architecture, DMA architecture. Developed software baseline.

Benjamin Fortener --- Developed software architecture, integer-based processing of transform/quantization. Developed FPGA-based wavelet transform.

Ken Simone --- Developed FPGA processing architecture for Tier I coder. Developed FPGA processing architecture for MQ-encoder and decoder.

Nick Vicen Developed Tier I encoder FPGA processing element, Developed Tier I decoder FPGA processing element.

David Lucking --- Developed FPGA processing architecture for Tier I decoder. Developed software baseline. Developed software architecture.

Eric Balster --- Developed quantization technique. Developed integer-based processing of transform and quantization. Developed software baseline

David Mundy --- Developed software baseline. Developed FPGA-based MQ-coder.

Mike Flaherty --- Developed FPGA-based wavelet transform.

Technology Components:

1) Original Software Baseline
2) Tier I Encoder FPGA processing element
3) MQ Encoder FPGA processing element
4) FPGA Wavelet Transform
5) Integer Processing of Transform / Quantization
6) Pipelining of Encoder Processing Elements
7) Memory interface controller
8) Tier I Decoder FPGA processing element
9) MQ Decoder FPGA processing element
10) Overall Software Architecture:

Original Software Baseline: Original software which has been exclusively designed by UD/UDRI engineers, so there are no licensing needed to run the code. A complete software implementation of the JPEG2000 codec.

Inventors: David Mundy, Frank Scarpino, Benjamin Fortener, David Lucking, Eric Balster, Thad Marrara

Description:

The original UDRI software baseline includes many of the possible options of JPEG2000 provided in the standard. They include: both lossy and lossless operations, quantization and optimal truncation (OT) for data reduction in lossy mode, an integer-based OT algorithm [1], and a fast hybrid quantization / quantization option for computational speedup [2]. Additionally, the original software baseline allows for both floating point, and integer based irreversible compression [3]. The original software baseline also supports compression of color imagery, with an option for separate quantization schemes for the luminance and chrominance portions. The original software baseline also supports the different modes of JPEG2000 compression, including: BYPASS, CAUSAL, RESET, and RESTART modes [4].

The original software baseline is used as a testbed to investigate which options and modes are candidates for hardware acceleration, as well as a research tool, allowing for new and innovative processing techniques around the JPEG2000 standard. The original software baseline is written in ANSI C and does not depend on any special software modules or libraries for compilation. Additionally, the original software baseline can be compiled and run on both windows and unix/linux platforms.

References:

[1] E. J. Balster, and W. F. Turri. "Integer PCRD-Opt Computation in JPEG2000 Compression", *SPIE Journal of Optical Engineering*, vol. 49, no. 7, pp. 77005-77011, July 2010.

[2] E. J. Balster, and W. F. Turri. "Efficient Processing of Optimally Truncated JPEG2000 Imagery", *SPIE Journal of Optical Engineering*, vol. 49, no. 2, pp. 27001-27012, Feb. 2010.

[3] E. J. Balster, B. T. Fortener, and W. F. Turri. "Integer Computation of JPEG2000 Wavelet Transform and Quantization for Lossy Compression", in Proc. *IEEE Int. Symp. Communication Systems, Networks, and Digital Signal Processing*. Newcastle, UK. July 21-23, 2010.

[4] E. J. Balster and D. L. Lucking. "BYPASS and PARALLEL Modes for JPEG2000 Compression of Natural Imagery", in Proc. *IEEE National Aerospace and Electronics Conference*. Dayton, OH, July 14-16, 2010.

Tier I Encoder FPGA processing element: The Tier I encoder is considered the heart of JPEG2000 this element is able to be easily replicated so that multiple instances of the coder for faster throughput

Inventors: David Walker, Luke Hogrebe, Nick Vicen, Ken Simone

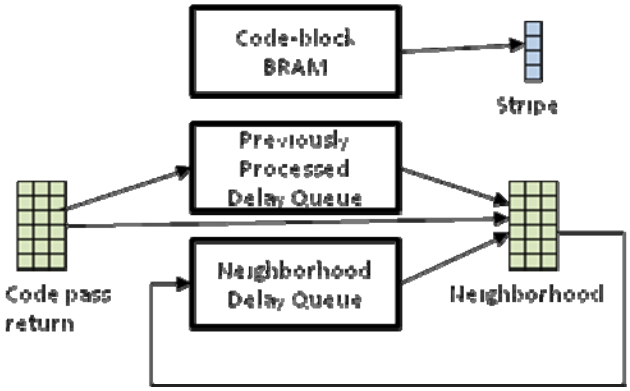
Description:

The Tier one system refers to the following processing elements:

1. Cleanup Code Pass
2. Significance Propagation Code Pass
3. Magnitude Refinement Code Pass
4. State Table memory controller
5. Memory – BRAM-devices or logic-based

The memory controller is critical to the efficiency of this system. Each of the code passes operate on a "neighborhood." A neighborhood is an 3 by 6 array of spatially related bits. The neighborhood is a subset of the state-table, which is spatially related to a bit-plane within a code-block. There are four state tables that the code passes use to control their operations.

State Table Memory Control Design



When a code pass operates on a neighborhood it will produce a series of context values to be coded by an arithmetic coder. It will also return updates for the state tables. The bit-plane is 32 by 32 bits. The state table is 34 by 34 bits. The data required from the bit-plane is 4 bits at a time. The data required from the state table is 6 bits at a time. Only 2 of those 6 bits are unique to the neighborhood produced for a particular stripe. The other 4 bits occupy a location that is in common with the neighborhood of another stripe. We store the state table data in two memory devices, referred to as delay queues. A delay queue is a FIFO-BRAM-device that is intended to accept inputs, hold them in order, and then release them after a specific number of read cycles. We use a short queue and a long queue to align the neighborhood data with the previously processed neighborhood data. The advantage of this scheme is

that we can read both delay queues and the code-block memory in a single clock. This design makes efficient use of memory devices and clock cycles.

MQ Encoder FPGA processing element: The MQ Encoder is the most primitive processing element in the Joint Photographic Experts Group (JPEG), JPEG2000 compression engine. The MQ Encoder produces Compressed Data (CD) output by processing together Decision (D) and Context (CX) pair input objects referred to as Symbols. The MQ Encoder is a sub-component of the Tier1 of the JPEG2000 Compression Scheme.

Inventors: David Mundy , Ken Simone

Description:

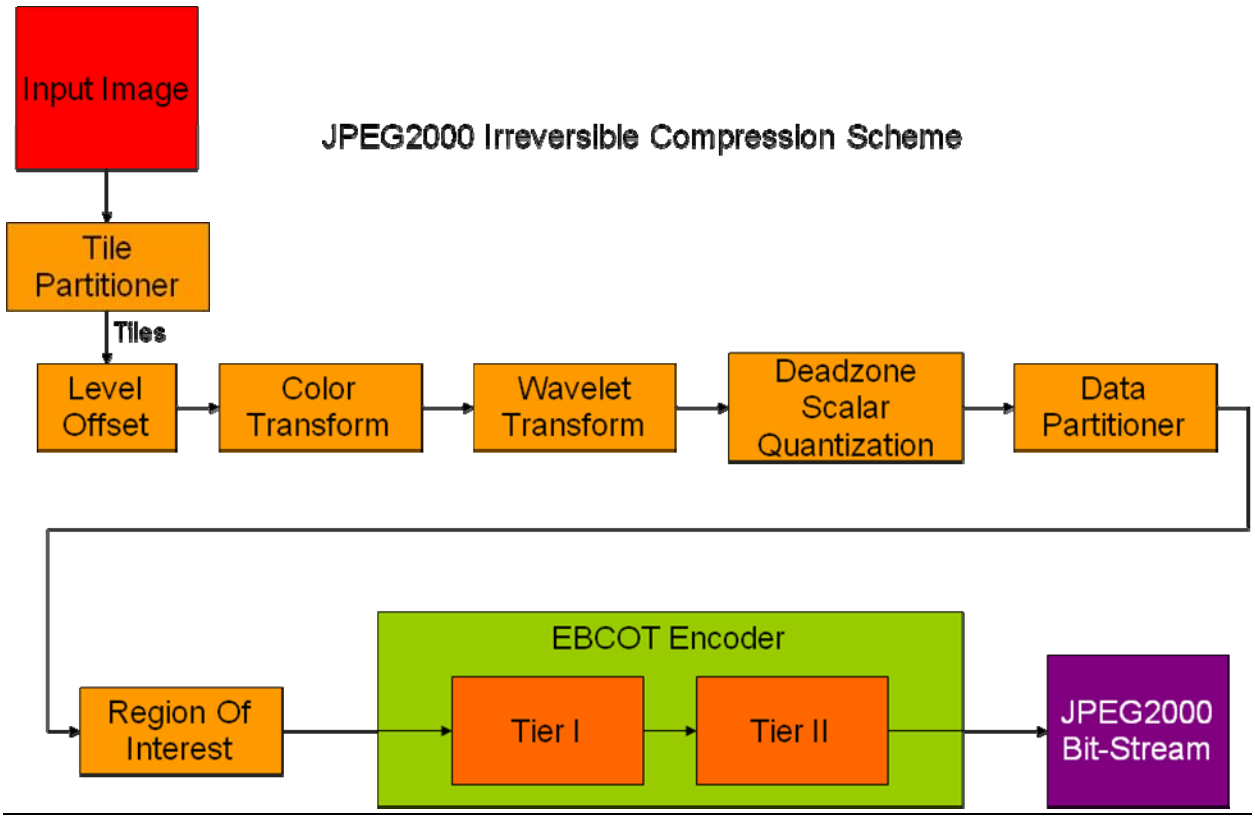


Figure 1 JPEG2000 Compression Scheme

Figure 1 illustrates the location of Tier1 within the Embedded Block Coding with Optimal Truncation (EBCOT) encoder of the JPEG2000 Compression Scheme. The MQ Encoder is within the Tier1 as illustrated in Figure 2. The MQ Encoder is a Binary Adaptive Arithmetic Encoder based on probability interval subdivision of Elias coding that is used by the JPEG2000 International Standard ISO/IEC 15444-

1Second Edition 2004-09-15 (reference: ISO/IEC 15444-1:2004(E)).An informative description of the operation of the Encoder can be found in Annex C of the standard.

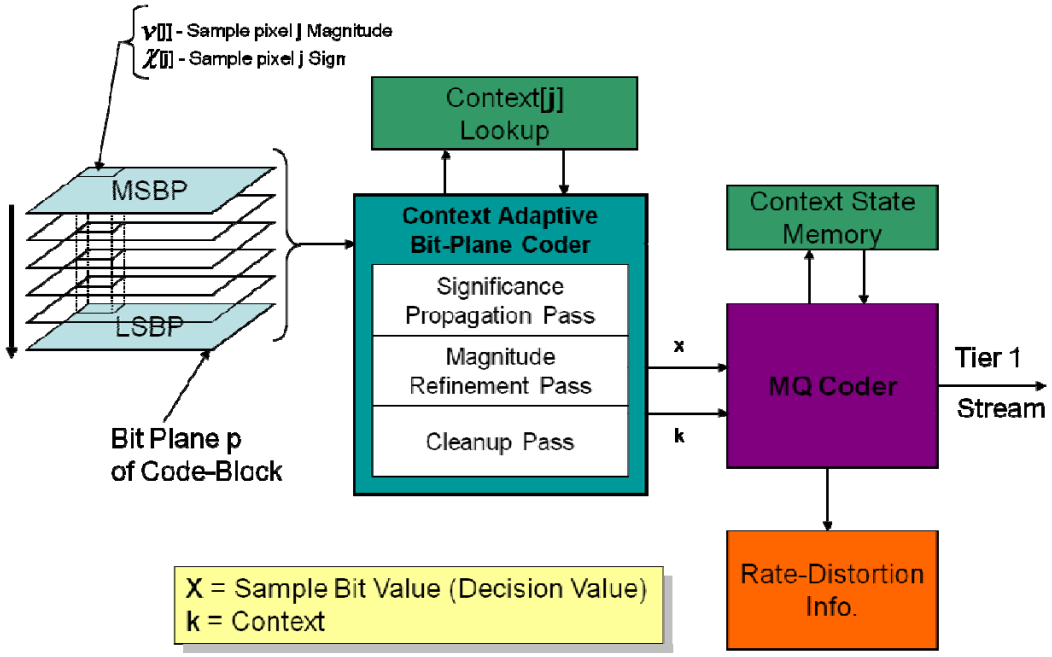


Figure 2 Tier1 Components

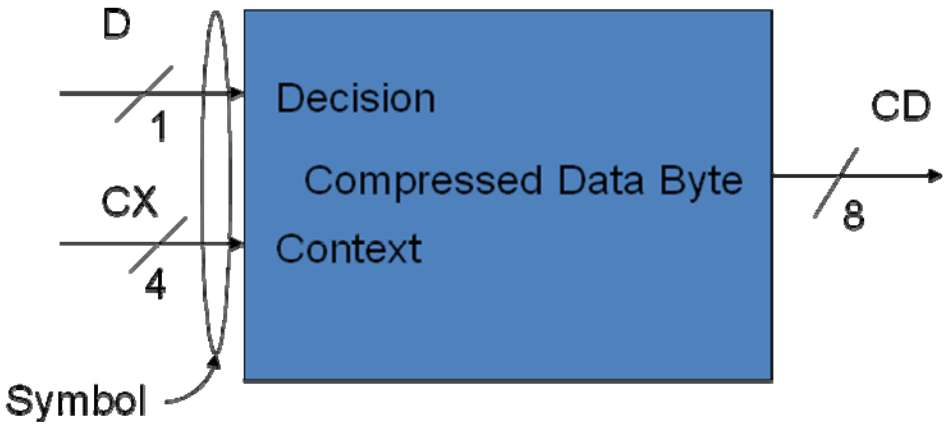


Figure 3 MQ Encoder Top Level Block

The MQ Encoder as integrated within Tier1 accepts the Symbols from the coding passes of the Bit-Plane Coder to produce the Compressed Data Bytes out. The Bytes out are provided to the Tier2 processing element to produce the JPEG2000 Data Stream.

The MQ Encoder consists of a context table, a probability (Qe) table, probability interval subdivision processing, a code register, and the Byte-out Buffer as illustrated in figure 4. Additional status and

control mechanisms exist and are used by the MQ State Machine to maintain the temporal integrity of the data objects during processing and Byte-out formation. The probability interval subdivision processing operations use fixed precision integer arithmetic. The hex value of 0x8000 represents a decimal fraction of .75. "A" is the interval and it is kept within the range of .75 ≤ "A" < 1.5 by doubling "A" whenever the integer value falls below 0x8000. This allows for an approximation to be used for the interval subdivision. For each D the current probability interval is divided into (2) intervals. The code is modified to point to the lower bound of the probability assigned to the symbol. This method requires ordering of the sub-intervals of the More Probable Symbols (MPS) and Less Probable Symbols (LPS). This method also requires sensing and analysis of the intervals for recursive processing. The sub-interval for the MPS is approximated to be "A" – Qe. The sub-interval for the LPS is approximated to be Qe. Whenever an MPS is sensed and coded, the value of Qe is added to the code register and the interval is reduced to "A" – Qe. Whenever an LPS is sensed and coded, the code register does not change and the interval is reduced to Qe. The process of reading and coding Context (CX) and Decision (D) symbols continues until all symbols are read and Bytes are output when they cannot be modified further.

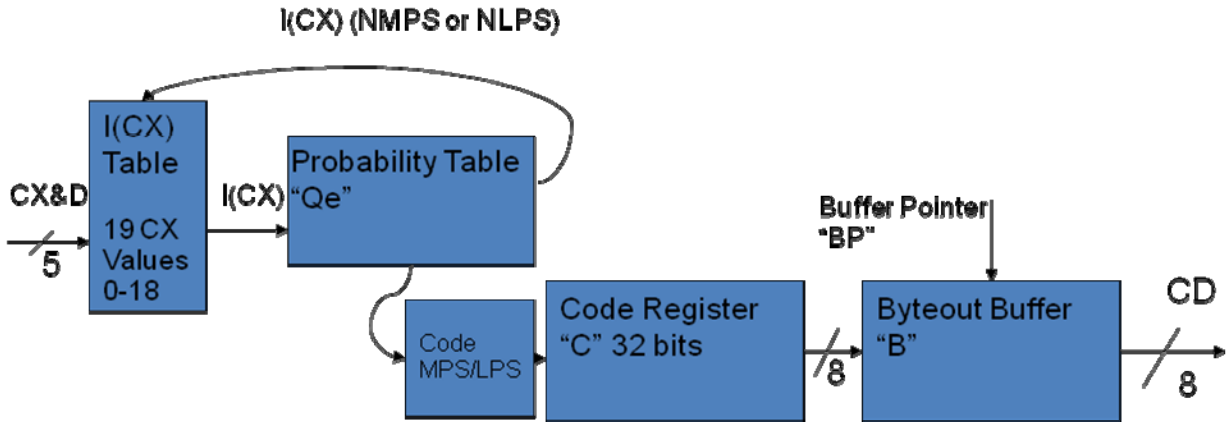


Figure 4 MQ Encoder Internal Elements

In our implementation of the MQ Encoder, we have partitioned all of the math functions and other equivalence checking functions into leaf level hardware processing objects to enable complete concurrent processing of all functions. This minimizes the combinatorial logic levels of all of the functions to the lowest number possible and enables us to run the frequency of the clock of the MQ to run at the max allowed by the longest combinatorial path amongst the functions. We have achieved greater than 3X clock frequency improvements versus other published papers and commercially available products. We have implemented an intelligent multiphase clocked multiplexing scheme to "hide" function processing time. We have also created a multiple pipe-lined scheme to enable overlap of processing cycles and reduce the total cycles to that of the largest number of cycles amongst the pipelined processing stages. Our design is very efficient in resource utilization, thus allowing for high numbers of parallelized MQ Encoders on a single device.

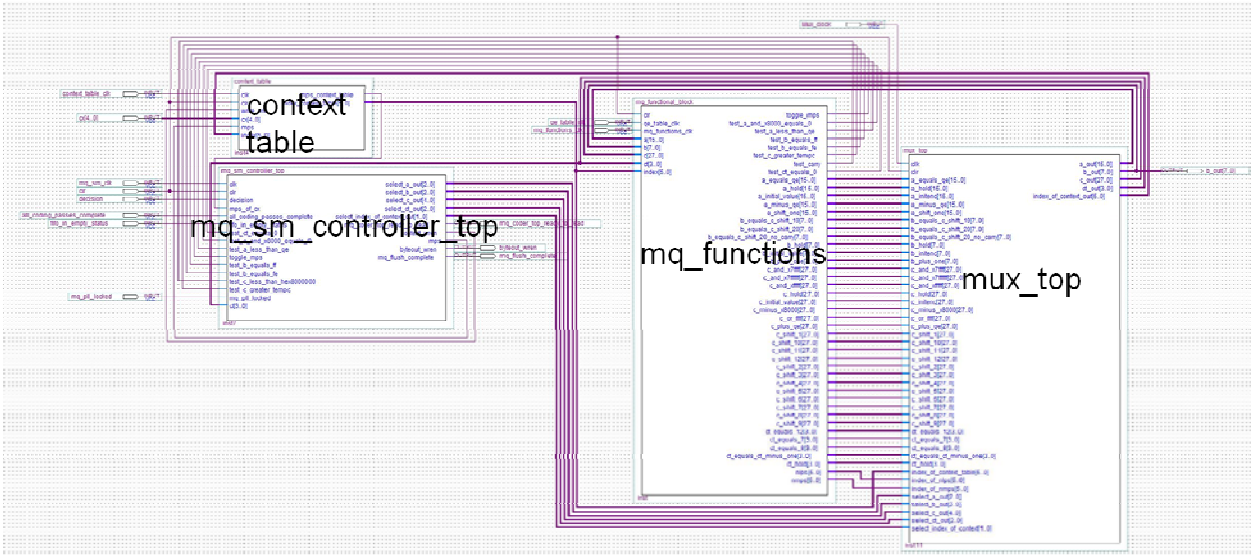


Figure 5 MQ Encoder Hardware Implementation Block Diagram

FPGA Wavelet Transform: This particular design allows for processing of rows and columns concurrently for faster throughput.

Inventors: Mike Flaherty, Ben Fortener

Description:

FPGA Implementation of a generic wavelet transform

Background

The wavelet transform as implemented in the JPEG2000 standard can be reversible (lossless) or irreversible (lossy). In the reversible case, the transform is performed using the LeGall 5/3 wavelet, an integer wavelet that allows for a lossless forward and inverse transform to preserve all of the data. In the irreversible case, the Daubechies 9/7 wavelet is used, a floating-point wavelet that—when implemented using finite precision floating-point numbers—is lossy through the forward and inverse transform. It is desirable to have both lossy and lossless compression options in a JPEG2000 compression engine, as many fields have uses for both.

In an FPGA, arithmetic operations on floating-point data are significantly more costly for time and resources than operations on integer data. Thus, an implementation of a wavelet transform that can perform both the integer, 5/3 wavelet and the floating-point, 9/7 wavelet is most efficiently designed using fixed-point operations.

Typically, the wavelet transform of an image is performed by filtering the columns of an image by both a low-pass and high-pass filter (in the case of the LeGall 5/3, the 5 stands for a 5-tap, FIR, low-pass filter and the 3 for a 3-tap, FIR, high-pass filter) to obtain the low- and high-pass coefficients. The resulting coefficients are then downsampled by 2 and arranged so that all the low-pass columns make up the top half of the image and the high-pass columns make up the bottom half. These coefficients are then filtered and downsampled again, this time on the rows. The output of the filters are organized into four quadrants called *subbands*, with the top left containing coefficients that have been low-pass filtered twice (LL-subband), the top right containing coefficients resulting from a low-pass filter in the vertical direction and a high-pass filter in the horizontal direction (HL-subband, or horizontally high-passed), the bottom left, coefficients from a vertical high-pass and horizontal low-pass (LH-subband), and the bottom right those coefficients resulting from two high-pass filter passes (HH-subband).

The wavelet transform can be performed multiple times by performing the transform again on the LL-subband only. This serves to "push" or compact the energy in the image into a smaller space, since low-frequency components make up the majority of information in most imagery. Performing the wavelet transform on multiple levels of the image allows for a multi-resolution analysis on each level of the wavelet coefficients.

Because of the large sizes of the target imagery, the multi-resolution wavelet transform is very processing intensive. Because the traditional method of applying the FIR filter prior to downsampling is so inefficient, W. Sweldons [1] proposed a method of downsampling a signal prior to implementing the wavelet transform. This method is called *lifting*, and the block diagram data flow is shown in Figure 1.

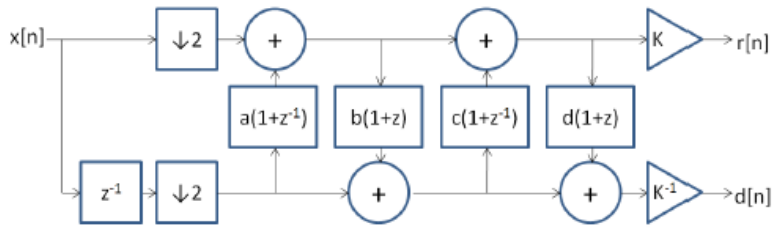


Figure 1 --- Lifting implementation of the Daubechies 9/7 Wavelet Transform

Summary of the invention:

Lifting and pipelining rows with columns

In order to achieve the desired processing throughput, lifting was used, allowing the transform to be performed on half the amount of data. In addition to lifting, the row transform was pipelined with the column transform. As the low- and high-pass column coefficients are calculated, they are input to the row transform. As soon as there are enough coefficients for the row FIR filter to begin working, the row transform begins processing as the column transform is continuing. The data streams through the column and row filters in this fashion until all the subbands are computed.

Multiresolution Feedback

For multiresolution analysis, the LL subband output can be directed back through the column and row transforms again and again until all transform levels are complete. The hardware is dynamically configurable to run the wavelet transform on any height and width and a controller manages the data flow moving it to either memory storage or back through the wavelet transform.

Resolution Pipelining

For chips with enough memory and logic, the individual levels of the wavelet transform may be pipelined so that the first level of the wavelet transform is being processed while other levels are processing. Because the second and subsequent levels are performed on only the LL subband, any number of levels after the first will require less arithmetic operations than that of the first. For this reason, the optimal pipeline stages for N levels are Stage 1 – Level 1, and Stage 2 – Levels 2 thru N.

[1] W. Sweldens. "The lifiting Scheme: A construction of second generation wavelets". SIAM Journal on Mathematical Analysis, 29(1):511–546, March 1998.

Integer Processing of Transform/Quantization: The integer processing of the transform and quantization allows for ease of implementation in an FPGA

Inventors: Benjamin Fortener, Eric Balster

Description:

The irreversible mode of JPEG2000 compression consists of a wavelet transform utilizing the Daubechies 9/7 wavelet. The 9/7 wavelet transform's basis functions are given by:

$$h[n] = \{0.02674, -0.01686, -0.07822, 0.2686, 0.60264, 0.2686, -0.07822, -0.01686, 0.02674\}$$
$$g[n] = \{0.04563, -0.02877, -0.29463, 0.55754, -0.29463, -0.02877, 0.04563\} \quad , (1)$$

where $h[n]$ and $g[n]$ are FIR filter coefficients with zero mean, $h[n]$ is a lowpass filter, and $g[n]$ is a highpass filter. The wavelet transform is performed when the original imagery is filtered by both $h[n]$ and $g[n]$ by column and then downsampled by 2. Afterwards, the 2 outputs of the process are filtered again by $h[n]$ and $g[n]$ by row, producing finally 4 partial outputs. The four outputs are referred to as subbands, specifically the LL (low-low), LH (low-high),HL (high-low), and HH (high, high). These names are given by the filters and order that the data in the particular subband has been processed. The LL subband is then further divided into subbands, allowing for multi-resolution analysis processing of the original imagery.

The wavelet transform processing can be sped up by the use of lifting [1]. Lifting is a process which allows the downsampling of the data to occur prior to the filtering. Figure 1 gives a block diagram of the lifted Daubechies 9/7 filter.

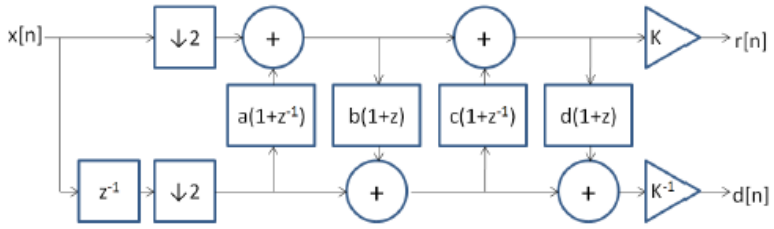


Figure 1 --- Lifting implementation of the Daubechies 9/7 Wavelet Transform

As seen in Figure 1, the downsampling of the input data occurs prior to the convolution step, allowing for a computational speedup. For the lifted 9/7 transform, $a = -1.58613$, $b = -0.05298$, $c = 0.88291$, $d = 0.44351$, and $K = 1.14960$. It is rather obvious that the lifting implementation of the Daubechies 9/7 transform requires the use of non-integer processing.

The Quantzation step in JPEG2000 reduces the dynamic range of the wavelet coefficients by utilizing an integer division step, reducing the wavelet coefficients to integers. The quantization step is given by:

$$q_i[n] = sign\left(x_i[n]\right)\left\lfloor \frac{x_i[n]}{\Delta_i} \right\rfloor. \qquad (2)$$

In Equation 2, $x_i[n]$ is the original wavelet coefficient at location n, $q_i[n]$ is the quantized wavelet coefficient, and Δ_i is the quantization step for subband i.

Typically, image data in its raw format is integer in nature, either 8, 12, or 16 bits per pixel. Therefore, the irreversible JPEG2000 compression engine takes in integers, transforms those integers into floating point coefficients, and then takes the floating point coefficients and quantizes them back to integers. This type of processing is not only computationally inefficient; it also makes hardware implementation difficult. Therefore, the proposed technology component is an integer-based irreversible transform and quantization method.

The integer-based irreversible wavelet transform is given in Figure 2.

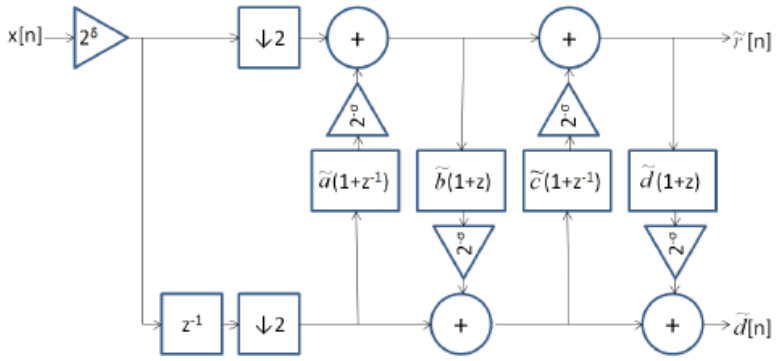


Figure 2 --- Integer based Daubechies 9/7 Transform

In Figure 2, each of the processing steps are integer based: $\delta = 5$, $\sigma = 13$, $\tilde{a} = \lfloor 2^{\sigma} a \rfloor$, $\tilde{b} = \lfloor 2^{\sigma} b \rfloor$, $\tilde{c} = \lfloor 2^{\sigma} c \rfloor$, and $\tilde{d} = \lfloor 2^{\sigma} d \rfloor$. The scaling of K is then moved to the quantization step.

The integer-based quantization step is given by:

$$\Delta_i = \frac{\Delta}{W_i},$$

where

$$W_i = 2^{-R} K^{2(R+s)} \sqrt{G_i}.$$

W_i, is a weight which includes the quantization step, as well as the K weight that was originally a part of the wavelet transform. Table 1 gives the values for W_i. s is the subband indicator, where $s = -1$ for the HH subband, $s = 0$ for the HL and LH subbands, and $s = 1$ for the LL subband. The gain of 2^{-R} is included in the subband quantization weight to obtain the proper biorthogonal gains of the wavelet coefficients to comply with the JPEG2000 standard. The values for W_i are given in Table 1.

R	$T = LL$	$T = LH, HL$	$T = HH$
9	34.4826270	26.4449975	20.2808763
8	26.0917269	20.0098620	15.3456525
7	19.7425499	15.1403824	11.6110218
6	14.9381023	11.4550870	8.7841827
5	11.3019848	8.6640553	6.6422742
4	8.5483659	6.5461329	5.0128710
3	6.4579878	4.9248048	3.7554210
2	4.8570731	3.6533364	2.7479239
1	3.6001023	2.6389679	1.9344316
0	2.5981240	2.0225730	1.5745212

Table 1 --- Values for W_i for various subbands

Integer quantization is then given by

$$q_i[n] = \mathtt{sign}\left(x_i[n]\right)\left\lfloor \frac{x_i[n]\lfloor 2^{\lambda} W_i \rfloor}{\lfloor 2^{\lambda}\Delta \rfloor} \right\rfloor$$

Where $\lambda = 7$. We can see that the quantization of $x_i[n]$ can be accomplished completely with integers.

Comparison of the traditional wavelet transform and quantization processes to the proposed integer based computation is accomplished by implementing the integer-based processing method into a JPEG2000 compression engine, and measuring the compression gain against the reference implementation of the standard, JasPer [2]. Figure 3 gives the test images used in the testing.

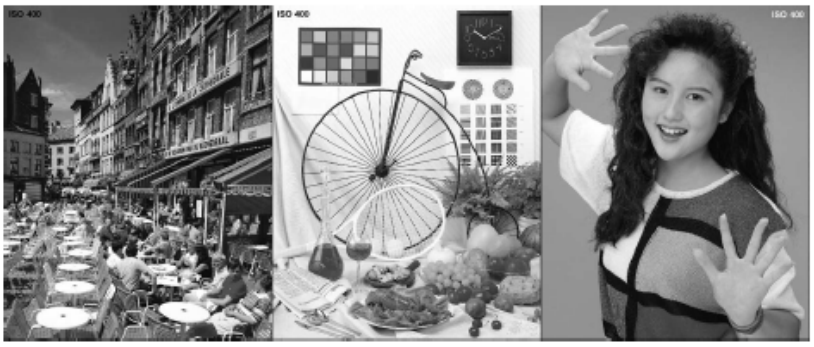


Figure 3 --- Test images, “café”, “bike”, and “woman”

The images given in Figure 3 are ISO/IEC test images, and are 2560x2048 pixels, grayscale. Table 2 gives the performance comparison of JasPer and the proposed integer-based processing method.

bpp	2	1	0.5	0.25	0.125	0.0625
			”Woman”			
JasPer	**43.984**	38.437	33.673	29.946	**27.395**	**25.625**
Integer	43.919	**38.466**	**33.695**	**30.055**	27.391	25.621
			”Bike”			
JasPer	**44.002**	38.098	33.549	29.551	26.421	23.827
Integer	43.855	**38.122**	**33.589**	**29.692**	**26.434**	**23.841**
			”Cafe”			
JasPer	39.095	32.037	26.845	23.090	**20.801**	**19.080**
Integer	**39.109**	**32.111**	**26.878**	**23.205**	20.794	19.073

Table 2 --- Performance comparison between JasPer and Proposed Method

As shown in Table 2, the performance of the proposed method is equivalent to that of JasPer. Additionally, there is a computation reduction utilizing the proposed method. Table 3 gives the performance comparison of the 2 methods.

Image Size	Jasper DWT	Integer DWT	% Improvement
1024x1024	0.062	0.032	**48.39**
2048x2048	0.391	0.266	**31.97**
4096x4096	1.844	1.438	**22.02**
Image Size	Jasper Quant	Integer Quant	% Improvement
1024x1024	0.031	0.015	**51.61**
2048x2048	0.078	0.031	**60.26**
4096x4096	0.329	0.141	**57.14**

Table 3 --- Performance Improvement of Proposed Method over JasPer

As shown in Table 3, the proposed method gives a substantial computation reduction when compared to JasPer. For more information regarding the integer computation of the transform and quantization step, please refer to [3].

References:

[1] W. Sweldens. ”The lifiting Scheme: A construction of second generation wavelets”. *SIAM Journal on Mathematical Analysis*, 29(1):511–546, March 1998.

[2] M. D. Adams. The jasper project home page. http://www.ece.uvic.ca/ mdadams/jasper/.

[3] E. J. Balster, B. T. Fortener, and W. F. Turri. "Integer Computation of JPEG2000 Wavelet Transform and Quantization for Lossy Compression", in Proc. *IEEE Int. Symp. Communication Systems, Networks, and Digital Signal Processing*. Newcastle, UK. July 21-23, 2010.

Pipelining of Encoder Processing Elements: The pipelining of the encoder elements allows for faster throughput.

Inventors: David Walker

Description:

The pipeline in the AFRL-JPEG2000 system refers to the data flow control mechanism. The processing elements controlled are:

1. Input from host to DDR
2. Tiling procedure
3. DWT
4. Copy from RAM to sequential memory
5. Tier One Encoder system
6. Output from DDR to host

A pipeline normally refers to a series of processes that operate parallel to each other, on a point of data that is passed through each of them sequentially. Each clock, a process must finish its operation and pass its product to the next phase of the pipeline. We take this concept and apply it to a set of processes acting on a set of data. Elements 1, 2, and 6 operate on the entire data set, while 3, 4, and 5 operate on a subset of data, called a tile. Element 3 can begin as soon as element 2 produces a complete tile. The pipeline control system maintains a flow of tiles though each successive tile processing element. For each tile, element 3 must be finished before element 4 can begin, and 4 must finish before 5 can begin. Element 6 will operate on the entire data set, once 5 has completed all the tiles. This system is implemented on an FPGA chip, which does not have enough BRAM for the entire data set. Some of the processing elements require access to larger RAM modules, necessitating the use of off chip DDR resources. The IP we use to access the off chip RAM allows for two types of access: 1. High throughput, sequential access. 2. variable throughput, or “bursty,” random access. Our processes require random access, but the bursty nature of the IP severely reduces our system performance. This type of memory control issue is common among DDR controllers. Our innovation is the use of an additional pipeline stage, element 4, to remove the penalty we incur for the random access required by element 5. This solution will be viable in any system that uses a similar DDR controller.

Memory interface controller: The memory interface controller allows for quick, easy access to on-board DRAM with hold the RAW imagery.

Inventors: William Turri, David Walker

Description:

Memory control in Tier one system

The Tier one system refers to the following processing elements:

1. Cleanup Code Pass
2. Significance Propagation Code Pass
3. Magnitude Refinement Code Pass
4. State Table memory controller
5. MQ Coder input and output memory
6. MQ Coder

The memory controller is critical to the efficiency of this system. The MQ coder operates on the context values produced by the code passes. Each of the code passes, elements 1, 2, and 3, operate on a "neighborhood." A neighborhood is an 3 by 6 array of spatially related bits. The neighborhood is a subset of the bit plane, which is a subset of a code block. The details of the relationships required by the jpeg2000 standard are not important to this discussion of the state table memory controller, except that, the neighborhood is associated with a spatial area of the code block. When it is given to a code pass, the neighborhood will be altered on return, and must be stored in such a way that the next code pass can retrieve the modified neighborhood when it operates on that spatial region of the code block. A bit plane is composed of 8 rows, with each row made up of 32 neighborhood columns. A neighborhood column is an array of 4 bits of code block data, a neighborhood contains the bits in the state table that spatially correspond to the 4 code block bits, and the surrounding 14 bits. It is important to note that the bottom two bits of a neighborhood will overlap the top two bits of the neighborhood below it. The best way to avoid problems from this is to make the state table memory an array of registers. This will act like a bit addressable RAM that can write to every address in one clock. This approach will require a relatively large amount of logic resources. The best alternative is to use two B-RAM devices as a set of delay queues. One row of code pass responses is stored in a short queue and 7rows of responses are stored in a long queue. If the outputs are properly synchronized it is possible to output a new neighborhood every clock.

Tier I Decoder FPGA processing element: The Tier I decoder is considered the heart of JPEG2000 this element is able to be easily replicated so that multiple instances of the coder for faster throughput.

Inventors: David Walker, Nick Vicen, David Lucking

Description:

The JPEG2000 decoder is an arithmetic decoder that de-compresses JPEG2000 encoded images. The input to the FPGA processing element is a collection of code-stream packets parsed from the encoded code-stream by the Tier-2 module. The output is a collection of wavelet coefficients decoded into smaller segments called code-blocks. The flow of the JPEG2000 decoder can be seen in Figure 1.

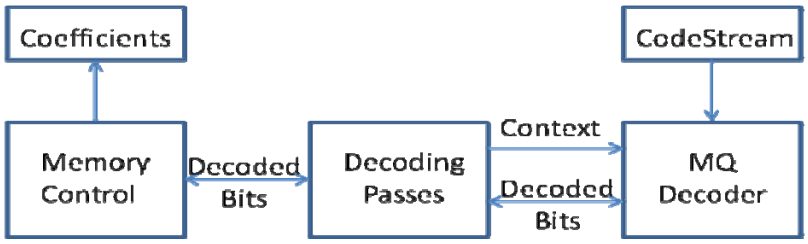


Figure 1: JPEG2000 Decoder FPGA Processing Element Overview

The JPEG2000 decoder consists of three decoding passes, a memory controller and a binary arithmetic decoder called the MQ decoder. The three coding passes are called the clean-up pass (CUP), significance propagation pass (SPP), and the magnitude refinement pass (MRP). The memory controller feeds the coding passes with context and encoded bit information. The coding passes use these inputs to create contexts to pass to the MQ decoder, which returns the decoded bits. These bits are then returned to the memory controller and organized into code-block segments to be returned as wavelet coefficients to the software.

The main portion of the JPEG2000 is the coding passes. Each pass operates on a stripe of four bits and calculates contexts for each bit in the stripe using a context window as illustrated in Figure 2. The context window is the 9 bits surrounding the current bit as seen as the shaded area of Figure 2.

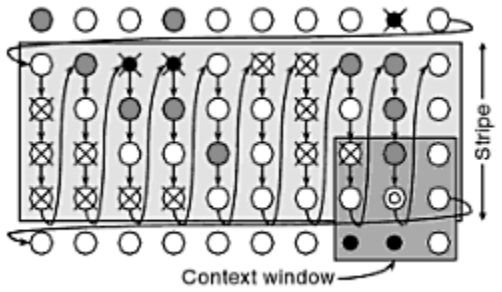


Figure 2: Context Window and Stripe Overview

The three coding passes operate sequentially starting with the SPP, then the MRP, finally the CUP. Figure 3 shows the top level overview of the bit-plane decoder showing the order of each coding pass.

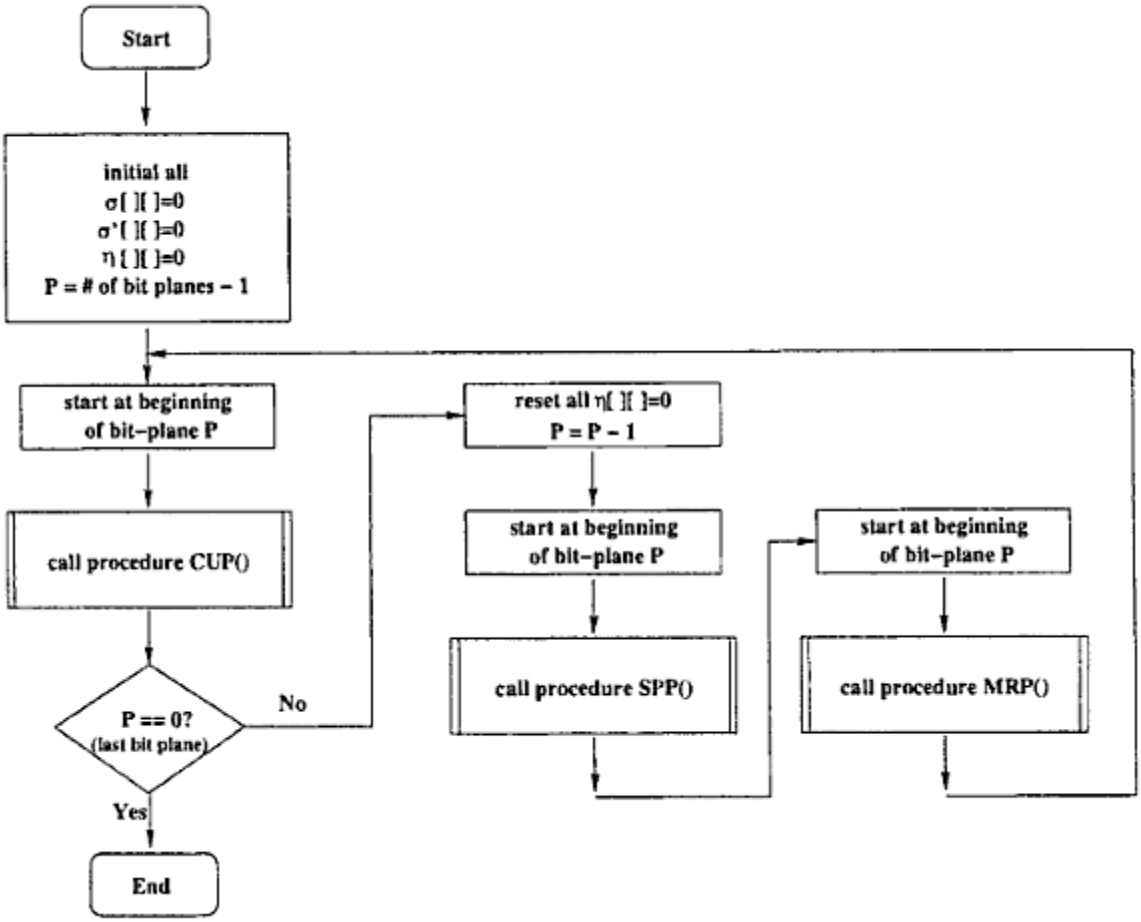


Figure 3: Top Level Flowchart for bit-plane Decoder

Each pass follows a procedure that determines what type of coding should be performed on the current bit of the stripe. There are four different coding modes performed within the passes: zero coding, sign coding, magnitude refinement coding, and run-length or uniform coding. Once the coding mode is determined, the coding pass creates a context based on the coding mode and the contexts surrounding the current bit. The context is passed to the MQ decoder which returns a decoded bit. A list of the contexts corresponding to the specific coding mode is seen in Table 1.

Table 1: Coding Mode Context Table

Operation	Context CX	Initial Index I(CX)
Zero Coding	0	**4**
	1	0
	2	0
	3	0
	4	0
	5	0
	6	0
	7	0
	8	0
Sign Coding	9	0
	10	0
	11	0
	12	0
	13	0
Magnitude Refinement Coding	14	0
	15	0
	16	0
Run-Length Coding	17	**3**
UNIFORM	18	**46**

The memory controller processes the bits returned from the coding passes and orders them into small code-block segments. These segments are then returned to the software as decoded wavelet coefficients to be processed by the inverse wavelet engine.

The novelty of the design is the removal of coding pass states from the standard. This was accomplished by performing parallel decisions while waiting on the MQ decoder to return its decision bit. Instead of wasting cycles in a wait state, two parallel paths were followed (if the bit was a 0 or a 1) and then the correct state was selected.

MQ Decoder FPGA processing element: The MQ coder is the most basic processing element in the JPEG2000 decompression engine.

Inventors: David Lucking , Ken Simone

Description:

The MQ decoder is a binary arithmetic decoder used by the JPEG2000 algorithm to accept the compressed codestream and contexts from the decoding passes and generate the wavelet coefficients with decoded bits, as shown in Figure 1. The compressed codestream is processed by the MQ decoder in segments called codeblocks.

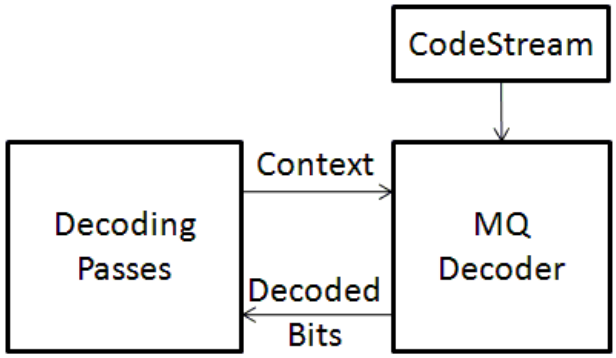


Figure 1: MQ Decoder Overview

The MQ decoder consists of two lookup tables (the context state table and the probability state table), a state machine module, an arithmetic module, a comparator module, and a controller module. The block diagram of the design, Figure 2, displays these components.

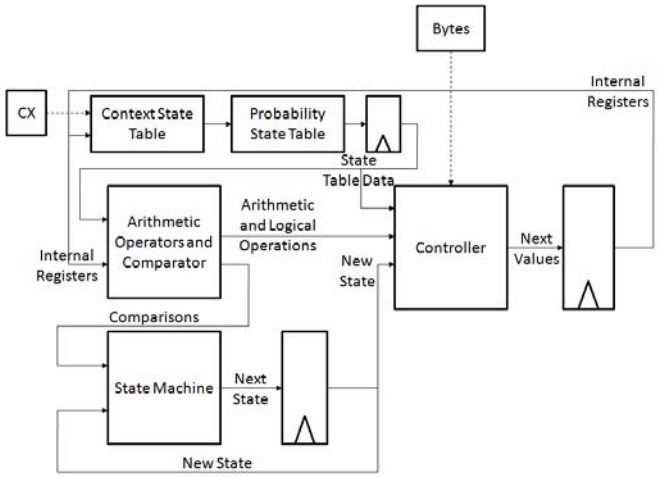


Figure 2: MQ Decoder Block Diagram

The look up tables (LUTs) output the probability estimations one clock cycle after a context is received. The arithmetic and logical operations are performed in parallel using the internal registers and the probability estimations during each rising edge of the clock. These operations are used in the controller to determine the output registers and the state machine to determine the next state.

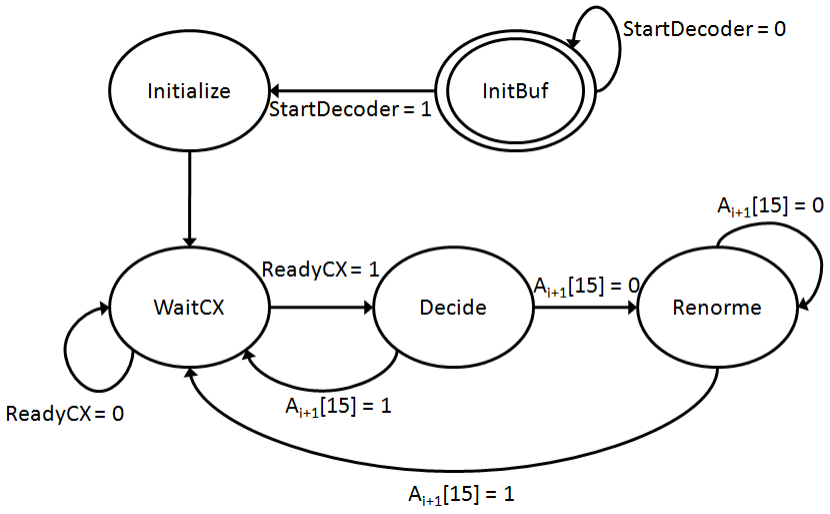


Figure 3: MQ Decoder State Diagram

The state machine component is detailed in Figure 3. The state machine consists of 5 states, 2 for initializing each codeblock, 2 for decoding a bit, and 1 for renormalizing the internal registers to their original values. The two initialization states load a total of two bytes from the codestream into the internal registers. The WaitCX state produces the probability estimations after the context is received. The decide state calculates the correct output bit and determines the values for the LUTs and the internal registers. If required, the Renorme state shifts the internal registers until the original value is reached and loads a byte if the register is empty.

The novelty of this design is shown in the block diagram and the state diagram. The arithmetic and logical operations performed in parallel combined with the state machine and controller only using one bit comparisons lowers the amount of logic required for this design. Another innovation provided by this design is the low number of states required to decode a bit. By manipulating the algorithm, the assignment of the LUTs, the internal registers and the output bit only depend on the probability estimation and the previous values of the internal registers.

The removal of a state decreased the number of clock cycles required to decode a bit increasing the throughput of the decoder. The block diagram innovation decreased the amount of logic required by

the decoder allowing more room for other items on the chip or to increase the throughput of the JPEG2000 decoder by instantiating multiple MQ decoders and processing codeblocks in parallel.

References

Lucking, David J. *FPGA Implementation of the JPEG2000 MQ Decoder*. Thesis. University of Dayton, 2010. *OhioLINK*. Web. 16 Sept. 2010.

Overall Software Architecture: The software architecture includes interfacing to the FPGA card, and controls the data transfer to and from the card. The Direct Memory Transfer based design allows for fast data transfer, and controls a serial-to-parallel processing interface.

Inventors: Thaddeus Marrara, Benjamin Fortener,

Description:

Encoder Software Architecture

The software architecture used in the hardware-accelerated compression system can run in two modes: software-only mode and hardware-accelerated mode. The software-only mode makes use of Intel Performance Primitives library function calls for operations performed on large sets of data, including:

- data copies for the JPEG2000 tiling process
- the color transform (RGB -> YC_rC_b)
- the wavelet transform (either 5/3 for reversible compression or 9/7 for irreversible)
- quantization
- entropy coding (JPEG2000 Tier 1)
- post-compression rate-distortion (optimal truncation)
- tag-tree coding and bitstream formation (JPEG2000 Tier 2)

Hardware-accelerated mode is broken into two portions: the client and the server. This client/server architecture allows for arbitration of the hardware resources by only allowing the server to communicate with hardware and the clients to communicate with the server. With this architecture, the system is very scalable, allowing as many clients as desired to work on their own images in parallel. They communicate to the server via pipes.

Client

The JPEG2000, hardware-accelerated compression engine is accessible via three custom library functions that are used by the clients.

- Initialization – the initialization function will check a semaphore used for inter-process communication to see if a server is running, and if the semaphore did not exist, the client will start the server, which creates the semaphore. The client also creates the communication channels (pipes) for sending images to and from the server process.
- Encoding – the encoding function sends an image to the server and receives the compressed image back from the server via the pipes.
- Cleanup – the cleanup function cleans up any memory and communication channels used by the client.

Server

Once a client has started the server, the server waits and monitors a registration pipe on which all clients will register themselves. Once a client has registered, the server receives a pointer to the client's image. The server takes the image pointer and uses it to copy the image data into a buffer with a specific structure that is sent to the hardware for compression. It monitors a hardware register to identify when hardware has finished the compression process, and then receives the compressed bitstream back from hardware. The server then sends a message back to the client with the compressed bitstream pointer, the size of the compressed bitstream, and the time it took hardware to complete the compression.

Data Buffer Structure

- The main header (first 32 bytes) contains how many tiles are being sent to the current FPGA.
- Following the main header is the tile header, which is present for each tile sent to hardware. It is 32 bytes and contains the scaling and quantization values (W_i from the *Integer Processing of Transform-Quantization* section) for each subband, 16-bits per value.
- Following the tile header is a single tile's data. Each subsequent tile is accompanied by its header and data until the end of the buffer.

Receive Buffer Structure

The hardware entropy coder codes the wavelet coefficient *codeblocks* in raster order within resolutions, within subbands. This is clarified in Figure 1. As the bitstream is stored in memory for software to read back, the hardware adds a 64-byte trailer to the end of each codeblock, specifying which codeblock it belongs to and how many bytes there are. This allows the software to skip to the end of the buffer and read from the end each trailer and then its corresponding codeblock to identify where to copy the data.