

## *Second Prize*

# **Nios II Processor-Based Hardware/Software Co-Design of the JPEG2000 Standard**

**Institution:** University of New South Wales

**Participants:** Mike Dyer, Amit Kumar Gupta, and Natalie Galin

## **Design Introduction**

JPEG2000 is a recently standardized image compression algorithm that provides significant enhancements over the existing JPEG standard. JPEG2000 differs from widely used compression standards in that it relies on discrete wavelet transform (DWT) and uses embedded bit plane coding of the wavelet coefficients [1]. Due to the bit-oriented processing techniques used in the standard, full implementation via software is inefficient, making embedded processing slow on standard microprocessors. Possible applications, such as scanners and printers, require a reasonable processing speed, which may be difficult to achieve using existing embedded processors. On the other hand, a full hardware implementation may not utilize the flexibility available in the standard. To improve the speed of the JPEG2000 algorithm while maintaining flexibility, we investigate the use of a co-design approach, using hardware acceleration for the bit-oriented and digital signal processing (DSP) tasks while leaving packet formation, code-stream formatting, and manipulation to software.

The Nios<sup>®</sup> II processor provides an ideal platform for implementing a co-design solution. The customizable arithmetic logic unit (ALU) allows for the addition of DSP-style instructions, which will improve the wavelet transform speed and code size. By adding custom peripherals to the system, the bit-oriented functions can be moved outside of the software into dedicated hardware. The provided real-time operating system (RTOS) ( $\mu$ C/OS-II) allows for parallel processing using multiple custom peripherals.

A software implementation of JPEG2000, called Kakadu [2], is used as the implementation framework and baseline for our design. Our proposed design adds the following features to Kakadu: multithreading with RTOS, custom instructions, and custom peripherals.

## Function Description

Our system is a JPEG2000 encoder based on a Kakadu software framework. Fully compliant with Part 1 of the JPEG2000 standard, the main features of the system are:

- Lossless and lossy compression
- Region of interest (ROI) coding
- Compression of color and gray-scale images

**Figure 1. JPEG2000 Encoding Flow**

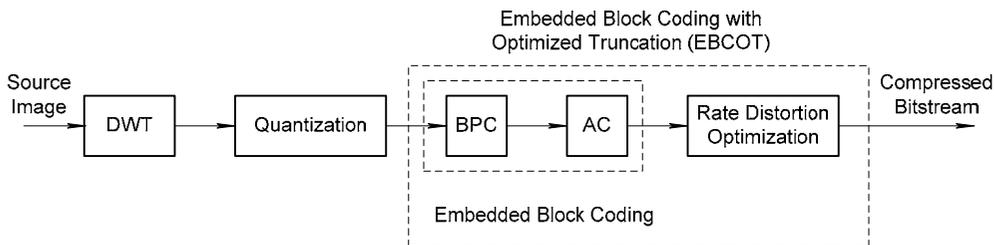


Figure 1 illustrates the compression system of the JPEG2000 algorithm. The image is compressed in the following steps:

1. Image samples are separated into color components (if any).
2. Image color components are optionally decomposed into rectangular tiles, with each tile to be compressed independently.
3. DWT is used to decompose each tile into four frequency subbands. JPEG2000-Part 1 specifies two wavelet kernels for lossy and lossless compression, 9/7 and 5/3 wavelet kernels respectively.
4. The output from the wavelet transform is quantized and separated into rectangular 'code-blocks', to be processed by EBCOT unit.
5. Each code-block is processed independently by the block coder (BC). The BC may be subdivided into bit-plane coder (BPC) and arithmetic coder (AC) modules. The BPC encodes a code-block in bit-plane by bit-plane order generating context-data (C x D) pairs. C x D pairs are then encoded by the AC module to generate the compressed bitstream.
6. Rate-distortion optimization selects optimal contribution of a code-block to the compressed bitstream for a given target bit rate such that the reconstructed image has minimum distortion. Kakadu uses the post compression rate distortion (PCRD) optimization algorithm [3].
7. Markers are added to the output bitstream to increase error resilience and packed into the JPEG2000 compressed bitstream.

The DWT and BC are two most resource intensive components of JPEG2000. A detailed study and analysis is performed to determine the strategy to best partition JPEG2000 into software and hardware components to optimize the compression stream using the rich feature set of the Nios II processor. We made the following changes to Kakadu:

- Custom instructions used to implement DWT
- Implementation of EBCOT in hardware, with BPC, AC, and the distortion estimation module implemented as hardware peripherals
- RTOS ( $\mu$ C/OS-II) used to instantiate multiple block-coders to increase throughput

## Performance Parameters

Table 1 presents the test environment used for comparison between the baseline and the proposed Kakadu implementation. It is to be noted that modules (DWT custom instructions, BPC, and AC) implemented in hardware are bit-exact with respect to the baseline, and thus do not alter the output bitstream. However, the hardware version of the module Distortion Estimation is not, and experimental results show that this change results in an average 0.02-dB PSNR difference between the baseline implementation and our proposed design when compressed for a given target rate.

**Table 1. Test Environment Parameters**

Property	Value
Image	café (ISO test image)
Image dimensions	2,560 × 2,048
Image format	pgm; 8-bit samples
DWT kernel	CDF 9/7
DWT levels	5
Block coder mode	Normal
Code-block size	64×64

## Profile Results

The profile results for a purely software implementation of Kakadu is presented in Figure 2. From the profile, we note that block coding accounts for 103.02 of the total 167.08 seconds (64.01%) used to compress the cafe test image. DWT, on the other hand, accounts for 11.36 seconds (6.89%) of computation time.

**Figure 2. Kakadu Profiling Results**

Flat profile:  
Each sample counts as 0.001 seconds.

time	% cumulative	self seconds	self seconds	calls	self s/call	total s/call	name
28.80	48.13	48.13	1286	1286	0.04	0.08	encode()
17.23	76.92	28.79	9420	9420	0.00	0.00	encode_cleanup_pass()
15.62	103.02	26.10	8134	8134	0.00	0.00	encode_mag_ref_pass()
3.89	109.52	6.51	4960	4960	0.00	0.00	perform_vertical_lifting_step()
3.54	115.44	5.92					__floatsidf()
3.48	121.25	5.82	2560	2560	0.00	0.00	transfer_bytes()
3.12	126.47	5.22					__pack_d()
2.90	131.32	4.85	4960	4960	0.00	0.02	horizontal_analysis()
2.89	136.16	4.83	122	122	0.04	0.92	encode_row_of_blocks()
2.69	140.64	4.49					__unpack_d()
2.47	144.77	4.13	4960	4960	0.00	0.00	push(kdu_line_buf&)
0.00	167.08	0.00	1	1	0.00	133.29	main()

index	% time	self	children	called	name
		48.13	56.70	1286/1286	encode_row_of_blocks()
[10]	62.7	48.13	56.70	1286	encode()
		28.79	0.00	9420/9420	encode_cleanup_pass()
		26.10	0.00	8134/8134	encode_mag_ref_pass()
		1.51	0.00	1286/1286	find_convex_hull()
		0.21	0.00	23116/25688	find_truncation_point()
		0.06	0.03	1286/1286	terminate(bool)
		0.00	0.00	1286/1286	start(unsigned char*, bool)
		0.00	0.00	2/2	set_max_bytes(int, bool)
		0.00	0.00	1/1	set_max_contexts(int)
		0.00	0.00	1/1	set_max_passes(int, bool)

## Block Encode Time

On average, the hardware implementation of the BC (BPC combined with an AC), will take:

$$T_{BPC} = \frac{CTX_{blk}}{CTX_{cyc}} \cdot \frac{1}{F_{clk}}$$

Where  $CTX_{blk}$  is the average number of C x D pairs produced per block,  $CTX_{cyc}$  is the average number of C x D pairs produced per cycle and  $F_{clk}$  is the system clock frequency. For a system running at 50 MHz and processing 64 sample code-blocks, the average code-block processing time is:

$$T_{BPC} = \frac{23 \times 10^6}{1.1} \cdot \frac{1}{50 \times 10^6} = 4.182 \times 10^{-4} \text{ sec}$$

For each code-block, the internal code-block RAM must be loaded via direct memory access (DMA). This time will accumulate in systems that use multiple BCs.

$$T_{DMA} = \frac{N_s W_s}{W_b F_{clk}}$$

$N_s$  is the number of samples in the code-block,  $W_s$  is the width of the sample in bits,  $W_b$  is the width of the bus in bits, and  $F_{clk}$  is the system clock frequency. This equation assumes that the DMA has exclusive access to the bus, as it will in our system. Using 64 sample code-blocks, where each sample is 16 bits, the system bus is 32 bits and the 50-MHz clock gives:

$$T_{DMA} = \frac{64 \times 64 \times 16}{32 \times 50 \times 10^6} = 4.096 \times 10^{-6} \text{ sec}$$

The average time taken to code a code-block in a system is approximately:

$$T_{\text{blk,avg}} = \frac{1}{N}(T_{\text{DMA}} + T_{\text{BPC}})$$

It should be noted that this equation is only accurate while the utilization of the data bus is low. When the time spent performing DMA transfers is greater than the time required to code a single code-block, the bus will become the limiting factor. This would indicate that the number of BCs should be

$$N \leq \left\lceil \frac{T_{\text{BPC}}}{T_{\text{DMA}}} \right\rceil = 10. \text{ Proper simulation is invaluable when determining the true value of } N.$$

Because block coding accounts for 103.02 of the total 167.08 seconds used to compress the cafe test image (Table 2), we expect that the implementation of block coding in hardware will give the highest performance improvement. We also note that the BC must process 1,286 blocks. Table 2 shows the average block coding time when using N parallel hardware BCs, and the time taken to code 1,286 blocks, while Table 3 shows the speed-up factor using multiple BCs.

**Table 2. Average Block Coding Times for Parallel BCs**

N	1	2	3	4
T_{blk,avg}	4.5916×10 <sup>-4</sup>	2.2958×10 <sup>-4</sup>	1.5305×10 <sup>-4</sup>	1.1479×10 <sup>-4</sup>
T_{blk,total}	0.59	0.30	0.20	0.15

**Table 3. Speed-Up Achievable With Multiple BCs**

N	1	2	3	4
Speedup	2.58	2.59	2.60	2.60

## DWT Flow

Amdahl's Law provides us with an estimate of the speed-up achieved from an improvement to a computation that affects a proportion P of that computation where the improvement has a speedup of S. (For example, if an improvement can speed up 30% of the computation, P will be 0.3; if the improvement makes the portion affected twice as fast, S will be 2.) Amdahl's law states that the overall speed-up of applying the improvement will be:

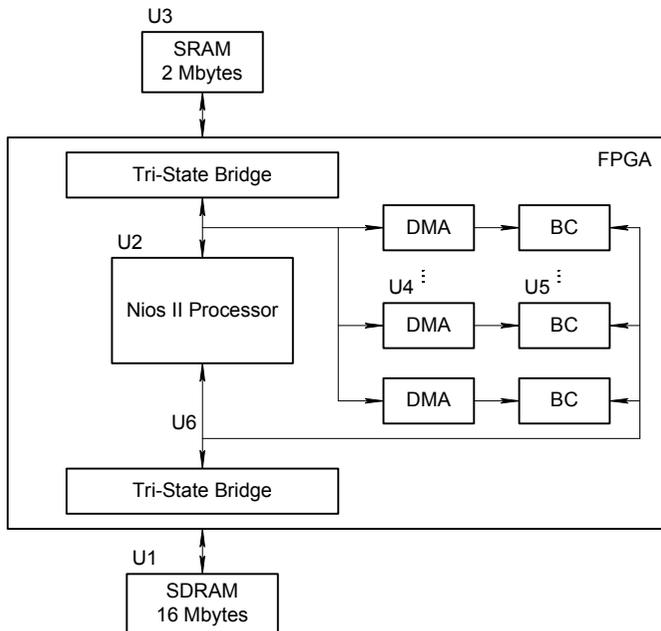
$$speed - up = \frac{1}{(1 - P) + \frac{P}{S}}$$

From the profiling analysis performed, we estimate that the DWT processing consumes approximate 6.78% (P=0.068) of the total image compression time. If we achieved a 5/2 improvement in the lifting step, by applying Amdahl's Law we estimate an approximately 1.0425 (4.3%) improvement in the processing speed overall.

## Design Architecture

The system structure is illustrated by Figure 3.

Figure 3. JPEG2000 Co-Design Configuration on Altera® EP1S40 FPGA



U1: 16 Mbytes of SDRAM containing the modified Kakadu program and image to be compressed

U2: Nios II processor

U3: 2 Mbytes of SRAM, which is loaded with code block data as it is created by Kakadu

U4: DMA controller configured to feed code block data at the rate of 16-bits per clock cycle to the BPC (U5)

U5: Block encoder hardware peripheral as described in detail later.

The number of DMAs and BPCs determined by the available bandwidth on the Avalon® bus. We chose to load the image directly onto the SDRAM so that the speed of our system is not limited by data transfer rates of external data I/Os. In this way, a fair comparison is made between the baseline and our proposed system. In the future, it will be possible to integrate the system with a fast bus fabric that will not saturate the speed advantages it provides.

Figure 4. BC Detailed System Configuration

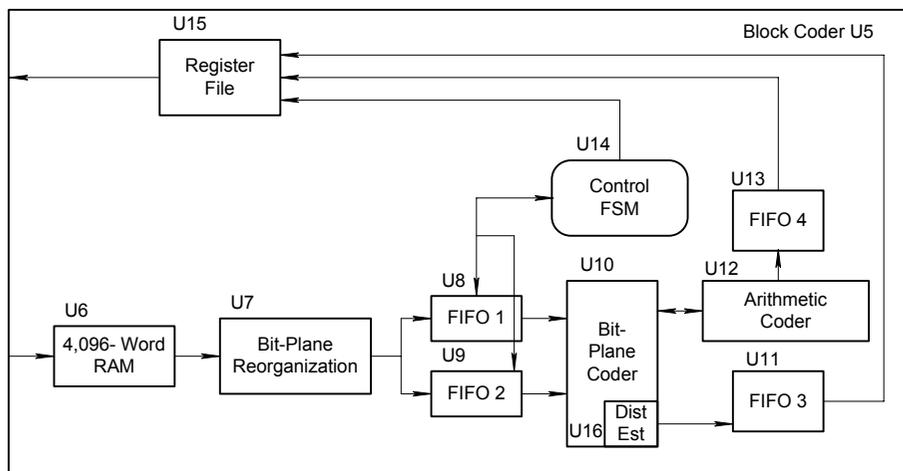


Figure 4 shows a detailed block diagram for the BC hardware peripheral, where:

U6: 4,096-word RAM to buffer the code-block data (entire code-block, if necessary)

U7: Reorganizes sub-band samples in a bit-plane by bit-plane fashion to feed the BPC

U8: 16 x 4-bit FIFO buffer to store the data bits from U7

U9: 16 x 4-bit FIFO buffer to store the sign bits from U7

U10: BPC module

U11: 22 x 13-bit FIFO buffer used to store distortion estimation data

U12: AC module

U13: 16 x 17-bit FIFO buffer used to store compressed data from the AC module

U14: Finite state machine to control the information flow between the various components of the system

U15: Register file containing 32 x 32-bit registers, 16 for read and 16 for write, to interface between the BPC (U5) system and the control bus

U16 : Distortion estimation module

## Design Methodology

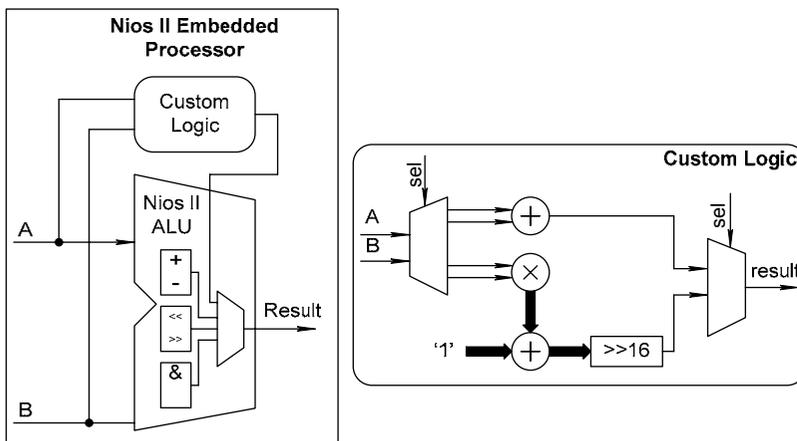
The hardware/software co-design of the JPEG2000 followed these design steps:

1. Alteration to Kakadu to support multithreading.
2. Development of bit-accurate software to verify the functionality of the proposed hardware peripherals.

3. Implementation of hardware peripherals using hardware description languages (HDLs).
4. Use of the ModelSim<sup>®</sup> tool to verify the hardware peripherals' functionality. The testbench vectors are generated using the bit-accurate software.
5. Use of the LeonardoSpectrum<sup>™</sup> tool to synthesize the hardware peripherals.
6. Use of the Quartus<sup>®</sup> II development suite to produce the layout and timing analysis of the hardware peripherals.
7. Post-synthesis simulation in the ModelSim software using the Quartus II timing results.
8. Load Kakadu software into the Nios II integrated development environment (IDE) and develop glue software to interface it to the hardware peripherals. Custom instructions were also added to Kakadu at this point.
9. Build and load of the Quartus II project onto the FPGA.

To improve the performance of the DWT function in Kakadu, we augmented the instruction set in the Nios II processor with two new custom instructions. The block diagram for the custom instructions is shown in Figure 5.

**Figure 5. Nios II Arithmetic Logic Unit (ALU) [4] & Custom Logic Layout to Perform a Lifting Step**



## Design Features

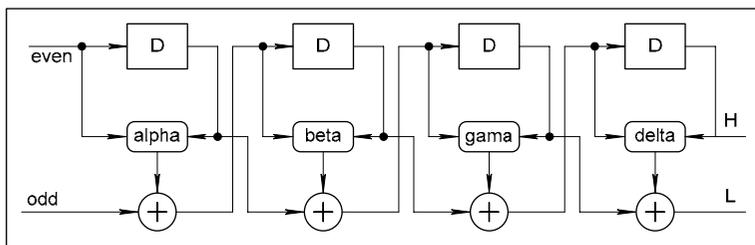
This section describes the project's design features.

### **DWT Custom Instructions**

As can be seen from Figure 6 of the state machine for the CDF 9/7 Lifting DWT implementation, the four lifting steps are very similar; the only difference is the value of the multiplier coefficient. Therefore, it was clear that the DWT would best be implemented using the Nios II processor's ability to add custom logic to its ALU. To perform a lifting step, two custom instructions were needed: one to augment two 16-bit samples into one, and one to perform the lifting step, shown in Figure 6.

Upon compilation, the number of assembly instructions to perform the lifting step decreased from five (in the pure C implementation) to two (using Nios II custom instructions).

**Figure 6. 9/7 DWT Lifting State Machine**



### **Multi-Threading of Kakadu**

The Kakadu software library was originally written as a single-threaded library. While this is adequate for single CPU systems, it makes dispatching multiple code-blocks to multiple BCs quite challenging. To utilize the availability of multiple BCs, the Kakadu library was modified to support threads. The use of threads requires using the  $\mu$ C/OS-II real-time operating system, a useful feature integrated into the Nios II IDE.

When enough data has been generated by the DWT, a row of code-blocks is dispatched via the function call 'encode\_row\_of\_blocks()' [2]. At this point, the library was modified to support threads. The library was supplied with a thread pool, where each thread is capable of encoding a single code-block. Each thread is attached to a single BC hardware resource and is responsible for initiating the DMA transfer and for collecting the compressed data and rate distortion information. When a thread completes, it is restarted with a new code-block until the row of code-blocks is exhausted.

### **Hardware Peripheral**

The main reason for the high computational cost of JPEG2000 on a general-purpose processor is due to the bit-oriented processing during block coding. This motivates the use of a custom hardware accelerator for the BC. A major feature of the Nios II SOPC Builder is that it supports the creation and utilization of custom hardware peripherals. Thus, it presents an ideal platform for our design.

The BC peripheral consists of two Avalon slave interfaces and four sub-modules. The first slave interface is used to receive block samples via DMA to remove the processor overhead involved in transferring sample data into the BC. The second slave interface accesses a register file, and is used to control the peripheral as well as access status information and compressed data. The four sub-modules perform bit-plane reorganization, bit-plane coding, distortion estimation, and AC. They are outlined below.

### **Bit-Plane Reorganization**

As the BPC operates on bit planes, sample data must be converted to this format before being sent to the BPC. The bit-plane reorganizer scans through stripe columns and forms a 4-bit word that contains the bit value of the four samples in that stripe column for the current bit plane. These are then stored in a FIFO buffer ready for sending to the BPC. This system must operate at twice the rate of the BPC to ensure data is always available.

### BPC Module

- Generic: Handles all modes of BPC operation for all nominal code-block dimensions
- High processing throughput: Generates an average of 1.1 C x D/clock-cycle (Existing generic BC architectures only generate 0.7CxD/clock-cycle) [5]
- Based on two-state memory system [6]
- Uses proposed optimal two sub-bank memory architecture for internal memories [7]
- Minimum memory cost (16 Kbits dual port RAM) currently reported for a generic BPC architecture
- Efficient intermediate buffer: The BPC has a varying C x Ds per clock-cycle output (anywhere between 0 to 10 C x Ds per clock-cycle) depending on image statistics. Since our AC module has a maximum input rate of two C x Ds per clock-cycle, we use an intermediate buffer [8] to integrate the BPC and AC module. The buffer is optimized for its hardware cost versus throughput performance using real image statistics.

### AC Module

The BPC is capable of producing multiple C x D pairs per clock cycle. Although an AC capable of coding a single pair per cycle can have enough throughput to cope with the C x D rate of the BPC, this would require the AC to have a separate, faster clock domain. To mitigate this complexity, an AC was designed that could consume two C x D pairs per cycle, while operating at the same frequency as the BPC [9].

### Distortion Estimation

The PCRD algorithm requires estimated distortion values associated with each truncation point (coding passes) [9]. The distortion estimation for a truncation point depends on the sample values and their distribution among coding passes, a factor that cannot be simply pre-calculated.

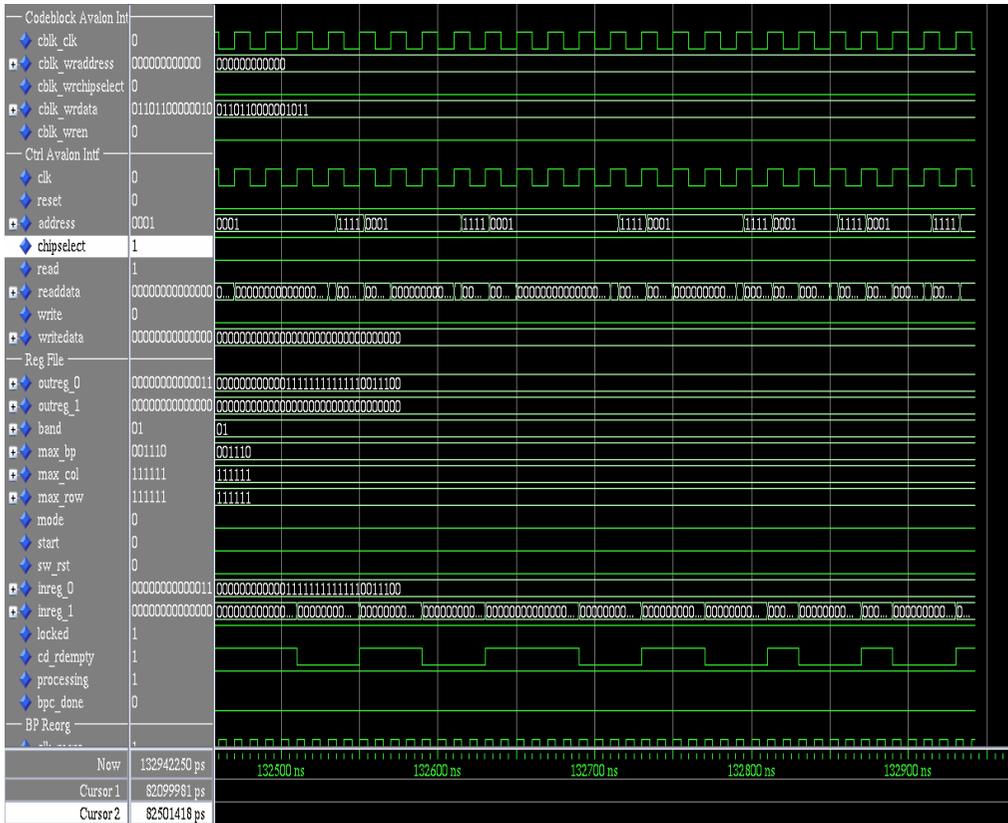
We designed a novel hardware module for distortion estimation that uses one fractional bit (in comparison to five fractional bits as used by Kakadu). Our simulation results show that this results in average 0.02-dB PSNR degradation for a given target rate (in comparison to Kakadu's reported architectures, which achieve an average 0.3-0.7 dB PSNR degradation [10]).

### Implementation Results

Figures 7 and 8 show simulation waveforms of our system. The following points in time are of particular interest:

1. 82,100 ns: Sample DMA finishes.
2. 82,200 ns: Register file access asserts operating parameters and starts system.
3. 82,400 ns: Bit plane reorganization started.
4. 132,550 ns: Register file access checking status and reading compressed data bytes.
5. 132,500 ns: Onwards demonstrates normal operation with BPC providing contexts to the AC.

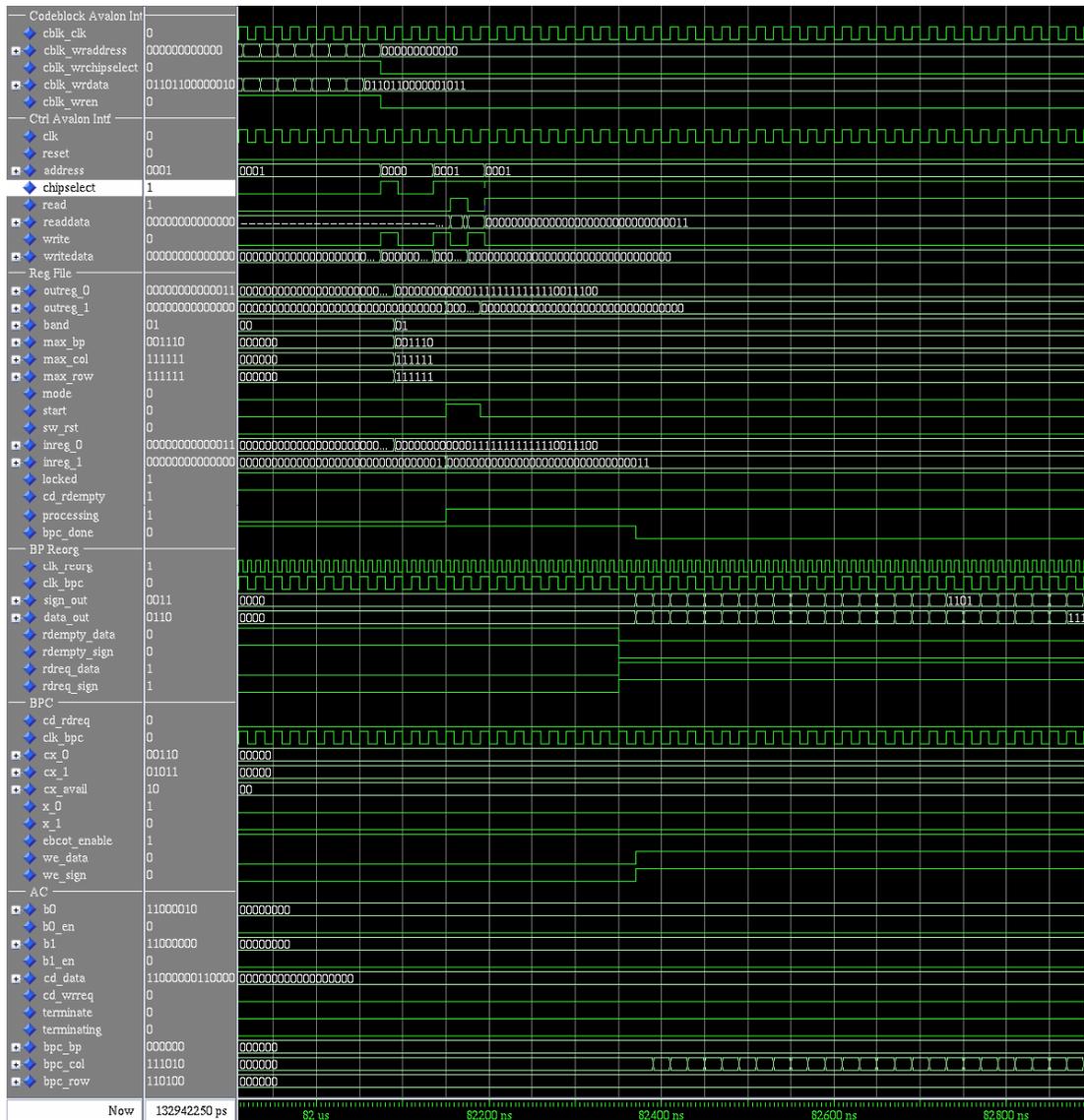
Figure 7. The Simulation Flow Showing BC-Avalon Bus Transfer



## Conclusion

The profile of the pure software implementation justifies our decision to provide dedicated hardware for the BPC and DWT, as these functions are at the top of the profile list. The Nios II processor provides an ideal platform for integrating dedicated hardware, as it provides the ability to include both custom instructions and peripherals. Our implementation results show that the inclusion of a DWT step instruction will improve the speed by a factor of 1.04, while the inclusion of dedicated block coding hardware can improve speed by a factor of 2.6. It is interesting to note that providing multiple BCs in parallel provides only minimal improvement over a single hardware BC as a consequence of Amdahl's law.

Figure 8. Simulation Flow Showing DMA Transfer in Place between Bit-Plane Reorganizer & BPC



## References

- [1] “JPEG2000 part i final committee draft version 1.0 ISO/IEC JTC1/SC29/WG1N1646R,” March 2000.
- [2] D. Taubman, “Kakadu software- a comprehensive framework for JPEG2000.” <http://www.kakadusoftware.com/>.
- [3] D. S. Taubman and M. W. Marcellin, *JPEG2000 Image Compression Fundamentals, Standards and Practice*. Norwell, Massachusetts 02061 USA: Kluwer Academic Publishers, 2002.

- [4] “Nios documentation.” <http://www.altera.com/literature/manual/mnlniossft.pdf>.
- [5] A. K. Gupta, S. Nooshabadi, and D. Taubman, “Concurrent symbol processing capable VLSI architecture for bit plane coder of JPEG2000,” *IEICE Transactions on Information and Systems, Special Section on Recent Advances in Circuits and Systems*, vol. E88-D, pp. 1878 – 1884, 2005.
- [6] Y.-T. Hsiao, H.-D. Lin, K.-B. Lee, and C.-W. Jen, “High-speed memory-saving architecture for the embedded block coding in JPEG2000,” *IEEE International Symposium on Circuits and Systems*, vol. 5, pp. V-133 – V-136, May 2002.
- [7] A. K. Gupta, S. Nooshabadi, and D. Taubman, “Optimal 2 sub-bank based memory architecture for bit plane encoder of JPEG2000,” *IEEE International Conference of Circuits and Systems (ISCAS’05)*, 2005.
- [8] A. K. Gupta, S. Nooshabadi, and D. Taubman, “Efficient VLSI architecture for buffer used in EBCOT of JPEG2000 encoder,” *IEEE International Conference of Circuits and Systems (ISCAS’05)*, 2005.
- [9] M. Dyer, D. Taubman, and S. Nooshabadi, “Improved throughput arithmetic coder for JPEG2000,” *IEEE international conference on Image Processing (ICIP’04)*, 2004.
- [10] Y. Chang, H. Fang, C. Lian, and L. Chen, “Novel pre-compression rate distortion optimization algorithm of JPEG2000,” *SPIE Proc. of Visual Communication and Image Processing*, vol. 5308, pp. 1353–1361, 2004.

Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, mask work rights, and copyrights.