Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2013, Article ID 140234, 9 pages
http://dx.doi.org/10.1155/2013/140234

Research Article

Design and Implementation of an Embedded NIOS II System for JPEG2000 Tier II Encoding

John M. McNichols,[1] Eric J. Balster,[1] William F. Turri,[2] and Kerry L. Hill[3]

[1] *Department of Electrical and Computer Engineering, University of Dayton, Kettering Laboratory, Room 341, 300 College Park, Dayton, OH 45469, USA*
[2] *University of Dayton Research Institute, 300 College Park, Dayton, OH 45469, USA*
[3] *Air Force Research Laboratory Sensors Directorate, Wright-Patterson Air Force Base, OH, USA*

Correspondence should be addressed to Eric J. Balster; ebalster1@udayton.edu

Received 12 February 2013; Revised 6 May 2013; Accepted 29 May 2013

Academic Editor: René Cumplido

This paper presents a novel implementation of the JPEG2000 standard as a system on a chip (SoC). While most of the research in this field centers on acceleration of the EBCOT Tier I encoder, this work focuses on an embedded solution for EBCOT Tier II. Specifically, this paper proposes using an embedded softcore processor to perform Tier II processing as the back end of an encoding pipeline. The Altera NIOS II processor is chosen for the implementation and is coupled with existing embedded processing modules to realize a fully embedded JPEG2000 encoder. The design is synthesized on a Stratix IV FPGA and is shown to out perform other comparable SoC implementations by 39% in computation time.

1. Introduction

One of the most recent image compression schemes, JPEG2000, offers a wide range of features and flexibility over the existing JPEG standard [1]. A block diagram of the JPEG2000 encoder is shown in Figure 1. The encoder consists of two main parts: the discrete wavelet transform (DWT) and the embedded block coding with optimal truncation (EBCOT) coder. The wavelet transform takes an image in the spatial domain and transforms it to the wavelet domain. The wavelet domain consists of a frequency representation with the addition of spatial information as well. Once the wavelet transform is completed, the coefficients are scalar quantized if lossy compression is chosen. The quantized wavelet coefficients are then entropy encoded using EBCOT, a two-tier coding algorithm which first divides each wavelet subband into code blocks (typically 32×32 or 64×64). EBCOT is composed of Tier I and Tier II encoders. Tier I produces independent embedded bitstreams for each code block using a context-based arithmetic encoder (MQ coder), the context for which is generated by the bit-plane coder. Tier II then reorders the individual compressed bitstreams and applies rate-distortion slope optimization to form the final JPEG2000 bitstream.

While JPEG2000 offers a number of improvements and additional features over JPEG and other image encoding standards, these benefits come with much greater computational cost. JPEG2000 is approximately 4 times more computationally expensive than the original JPEG [2]. Due to these high costs, it becomes impractical to utilize JPEG2000 in applications which require real-time processing of high-resolution images, such as wide area imagery or medical imagery. To solve this problem, developers continue to turn to hardware implementations to yield the throughput necessary to meet frame rates for high-resolution imagery [3]. Hardware solutions are able to outperform software implemenations through the use of parallel processing and custom designed hardware. A hardware implementation is capable of leveraging the inherent parallelism of the EBCOT block coders to achieve large increases in throughput over typical software implementations. Not only do hardware solutions offer dramatic increases in throughput over their

software counterparts, but they also free host processors to handle other critical tasks.

Most embedded JPEG2000 solutions focus on performance increases in the EBCOT Tier I, either through novel architectures or simply by leveraging the parallelism of multiple block coders. Research focuses on EBCOT Tier I improvement because this is the most computationally expensive module of JPEG2000, as shown in [4, 5].

In [6, 7], architectures for the MQ coder are proposed which consume two context-data (C × D) pairs per clock cycle. Reference [4] takes a different approach, increasing performance by using column-based operation combined with pixel and group-of-column skipping techniques. A number of implementations focus on very large-scale integration (VLSI) architectures for JPEG2000. Some, such as [3], focus on high-speed VLSI implementations by utilizing pass-parallel EBCOT implementations. Others, such as [8], attempt to reduce the on-chip memory requirements for EBCOT Tier I and Tier II while also improving performance.

However, most of these implementations fail to mention the final piece of JPEG2000 which is the formation of the full bitstream, EBCOT Tier II. Generally it is not mentioned at all and assumed to be left for the host processor to handle in software. References [8, 9] propose an architecture for EBCOT Tier II which is focused on reducing memory requirements for bitstream buffering but do not offer high performance. Instead, we propose the use of a softcore coprocessor to serve as a Tier II processing module situated at the back end of an encoding pipeline to realize a fully embedded encoder. The softcore coprocessor offers more flexibility than [8, 9], due to the soft nature of the processor, while also offering adequate performance to meet the demands of high-resolution image compression.

This work couples an Altera NIOS II processor [10] with existing, efficient, and hardware implementations of the other various processing units to create a fully embedded JPEG2000 system on a chip (SoC). The hardware is designed as a tile encoding pipeline in order to efficiently encode high-resolution imagery. The NIOS II processor interfaces with a FIFO containing the independent code block bitstreams produced by a variable number of hardware block coders in order to create the final bitstream. The NIOS II processor handles all of the Tier II processing as well as transferring data back to the host processor.

While similar to [5] in the use of an Altera NIOS II processor, the proposed implementation utilizes the processor as a separate processing module as opposed to a system scheduler/device arbiter. This avoids the scheduling overhead associated with such an implementation while also preventing the other processing modules from becoming limited by the throughput of the NIOS II system. Additionally, the simplicity of using the NIOS II as a separate processing unit yields a system which is much easier to debug and test, since difficulty in debugging in [5] prevented the system from actually being implemented on an FPGA.

The rest of this paper is organized as follows. Section 2 gives a brief overview of the FPGA-based processing architectures used for the front end of the processing pipeline. Section 3 details the selected target platform and the selection of the coprocessor before giving a detailed description of the implementation of the SoC. Section 4 analyzes the performance of the implemented system and discusses the impact of increasing numbers of parallel block coders before comparing the results to other SoC implementations. Finally, Section 5 concludes the paper.

2. JPEG2000 Hardware Modules

2.1. Discrete Wavelet Transform/Quantization. The proposed system implementation uses a lossy CDF 9/7 wavelet transform [11]. The lossy wavelet transform is chosen as it offers additional compression gain over the lossless implementation while still maintaining comparable image quality at lower compression ratios. This implementation is an integer-based approach, utilizing the CDF 9/7 wavelet filter to transform integer input pixels into scaled fixed-point wavelet coefficients. These scaled fixed-point coefficients are then quantized back into integers prior to compression by EBCOT. Running at a clock rate of 100 MHz, this DWT implementation consumes two pixels per clock cycle and takes approximately 7 ms to perform a standard 5-level transform on a 1024 × 1024 tile.

2.2. EBCOT Tier I. EBCOT Tier I is comprised of two main processing modules: the bit-plane coder (BPC) and the MQ coder [1]. The BPC for the proposed implementation is a generic implementation, conforming to the standard, and operates at a clock speed of 100 MHz. While most implementations of the BPC aim to maximize throughput, this design instead focuses on reducing resource utilization and does not make use of any of the optimization techniques proposed in [3, 5–7]. Instead, minimal resource usage is achieved by consolidating the number of memory devices necessary to store code block state data. The MQ coder follows the same design principle as the BPC, with a focus on minimizing hardware resource usage. The MQ coder runs at a clock rate of 200 MHz. The design goal of both the BPC and MQ coders is to maximize the number of Tier I coders which can fit on a device. By achieving high clock rates through resource optimized designs, a Stratix IV device with over 90% resource utilization is capable of yielding throughput in excess of 180 MBytes per second when multiple parallel Tier I coders are coupled with three DWT targets.

3. Proposed System Implementation

3.1. Target Platform. The target platform for the JPEG2000 SoC is a Stratix IV PCIe ×4 development board from GiDEL [12]. The platform selected features a Stratix IV E 530 FPGA with a 512 MB DDR2 memory bank and two DDR2 SODIMMs with up to 4 GB each. The platform has two additional ports for expansion daughter cards offered from GiDEL. A block diagram of the selected platform can be seen in Figure 2. The high performance offered from a PCIe based platform coupled with the flexibility and size of an Altera Stratix IV FPGA provides an ideal platform capable of meeting the demands of a JPEG2000 SoC [12].

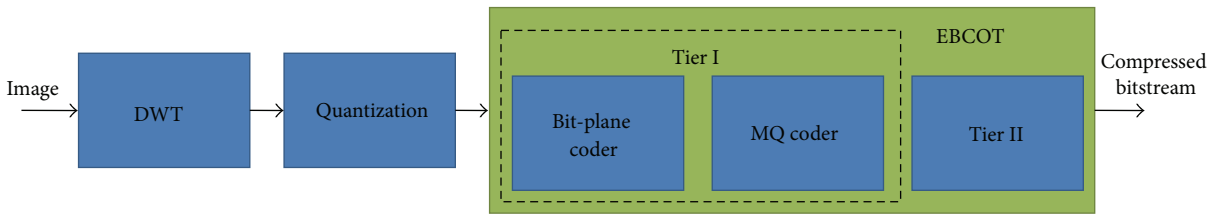

Figure 1: Block diagram of JPEG2000 encoder.

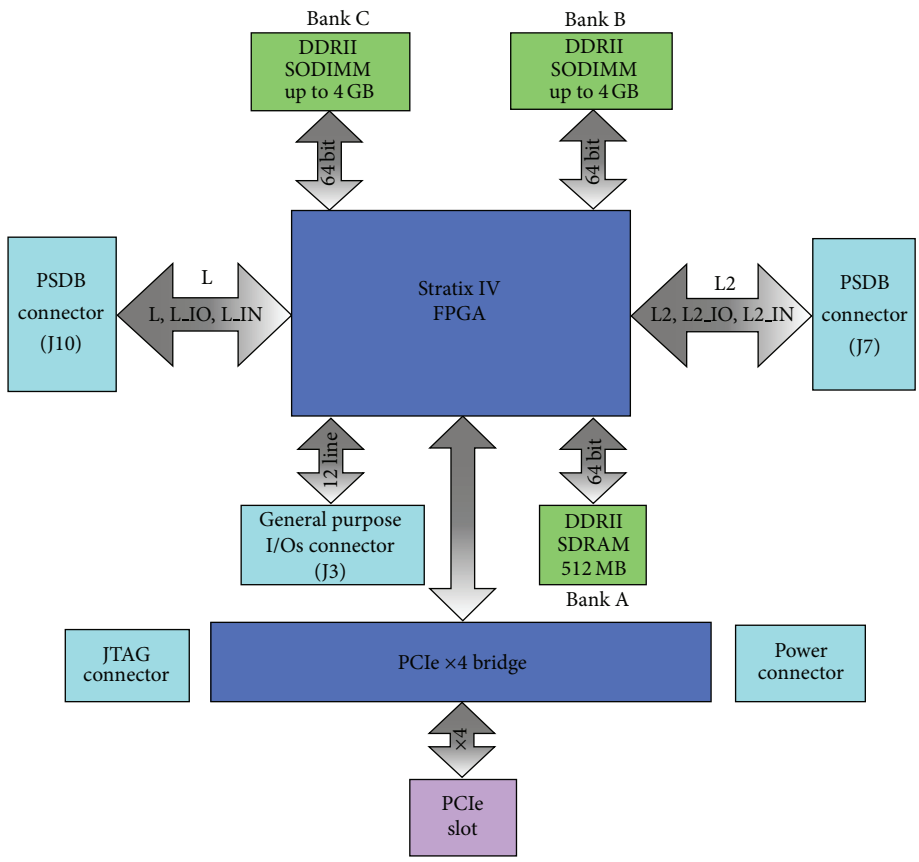

Figure 2: Block diagram of target platform.

3.2. Selection of Softcore Coprocessor. The softcore coprocessor chosen is the NIOS II processing core from Altera [10]. The NIOS II processor features a 32-bit reduced instruction set computing (RISC) architecture that is highly configurable, capable of supporting up to 256 custom instructions and clock speeds near 300 MHz on a Stratix IV device. The NIOS II processor system consists of a NIOS II processor core coupled with on-chip peripherals (DMA controllers, timers, and custom HW interfaces), on-chip memory as well as interfaces to off-chip memory. All of the various peripherals and memories are managed through the Avalon switching fabric which serves as an arbiter between the various masters and slaves within a system. The Avalon switching fabric allows multiple data/instruction masters to communicate directly with multiple slaves devices simultaneously, assuming no two masters are attempting to communicate with the same slave. The NIOS II processing core is chosen given high degree of flexibility and ability to support custom peripherals as well as built-in support for embedded C/C++ development using the NIOS II software development suite [10, 13].

3.3. Hardware Implementation. In order to realize a full JPEG2000 SoC, the EBCOT Tier II processing module must reside in hardware. Most research on JPEG2000 neglects to mention the implementation of Tier II, presumably leaving it to be handled by the host processor. This paper proposes a novel solution to this gap by leveraging an embedded softcore coprocessor to serve as an embedded EBCOT Tier II processing module. While similar in nature to the proposed architecture in [5], which utilizes the NIOS II system as an

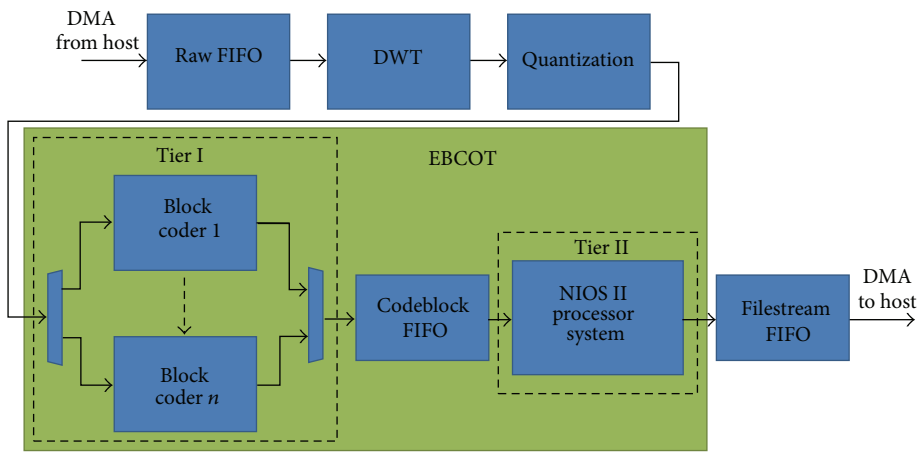

Figure 3: Data flow of proposed JPEG2000 SoC.

arbiter between different hardware modules, this architecture treats the coprocessor as a separate processing module.

The proposed JPEG2000 SoC is implemented on the target platform by integrating the NIOS II processing unit with existing embedded JPEG2000 processing modules. The existing design features a pipelined DWT architecture coupled with a variable number of parallel EBCOT Tier I block coders. The details of the specific architecture are given in Section 2. The coprocessor serves as the final stage of the pipeline, taking the code block streams from the block coders and forming the final JPEG2000 filestream. This eliminates the scheduling overhead associated with arbitration between the various processing modules as in [5]. The dataflow of the proposed implementation can be seen in Figure 3.

Prior to transferring the raw image to the target device, the image is padded and divided into tiles. Tiles are then sent to the target via DMA over the PCIe bus and processed sequentially. Each stage of the pipeline begins once a single tile has been received from the previous stage. Tiling the image reduces the memory requirements for each stage in the pipeline since each stage will need enough memory to store a single tile. Additionally, tile processing enables the use of distributed architectures where a single host leverages multiple target devices to process the tiles of a single image in parallel.

The NIOS II processing system creates the independent tile stream for each tile it receives from the pipeline before placing it in a FIFO for DMA back to the host via PCIe. The host system receives the tile streams from the device and applies the main headers to form a valid JPEG2000 filestream. While the NIOS II could easily be configured to add the main headers, this task is left for the host processor in order to maintain architectural flexibility in the event that more target devices are added to the system.

3.4. NIOS II System Implementation. This NIOS II processor operates within the entity created by Altera's System on a Programmable Chip (SoPC) builder [14]. This tool allows for seamless integration between the softcore processor and other hardware peripherals through the Avalon switching fabric. In addition, the SoPC builder allows for integration of multiple processing blocks running at different clock rates, with SoPC handling all of the arbitration between clock domains. The SoPC system used in the proposed design features a NIOS II fast core, running at a clock rate of 290 MHz. This NIOS II core is the fastest of the three offerings from Altera, offering high clock speeds and a number of additional features over the other two cores. A block diagram showing the implemented NIOS II system is shown in Figure 4. The clock rates and interface types of all modules shown are detailed in Table 1.

The NIOS II core is coupled with three different memory controllers. One is a DDRII SDRAM controller from Altera, running at 200 MHz, which interfaces directly with one of the two DDRII SODIMMs available on the target platform (Bank B or C) seen in Figure 2. The SDRAM serves two functions in the system. First, the SDRAM address space serves as a buffer to store incoming code blocks which are read out of the code block FIFO seen in Figure 3. Referred to as the input data buffer, code blocks are buffered in this address space prior to processing by Tier II. Second, the SDRAM address space is used to store the completed JPEG2000 filestream for each tile prior to transfer back to the host. This is referred to as the filestream buffer in subsequent sections.

Besides the off-chip SDRAM, there are also two separate on-chip memory controllers, one which controls a 50 kByte bank and the other which controls a 30 kByte bank. The 50 kByte bank is used to hold the executable code and data sections in addition to the program stack and heap during execution. The other 30 kByte bank is configured as "Tightly Coupled Data" memory. The details of this implementation are elaborated on in Section 3.5. Both of the on-chip memory banks are configured to run on the same clock as the NIOS II core.

Two custom interfaces are designed to communicate with the code block and filestream FIFOs as seen in Figure 3. These interfaces are used to couple the code block and filestream FIFOs with two DMA controllers to allow streaming of data in/out of the two FIFOs. Each DMA controller is comprised of two modules: a dispatcher and a read/write master. The

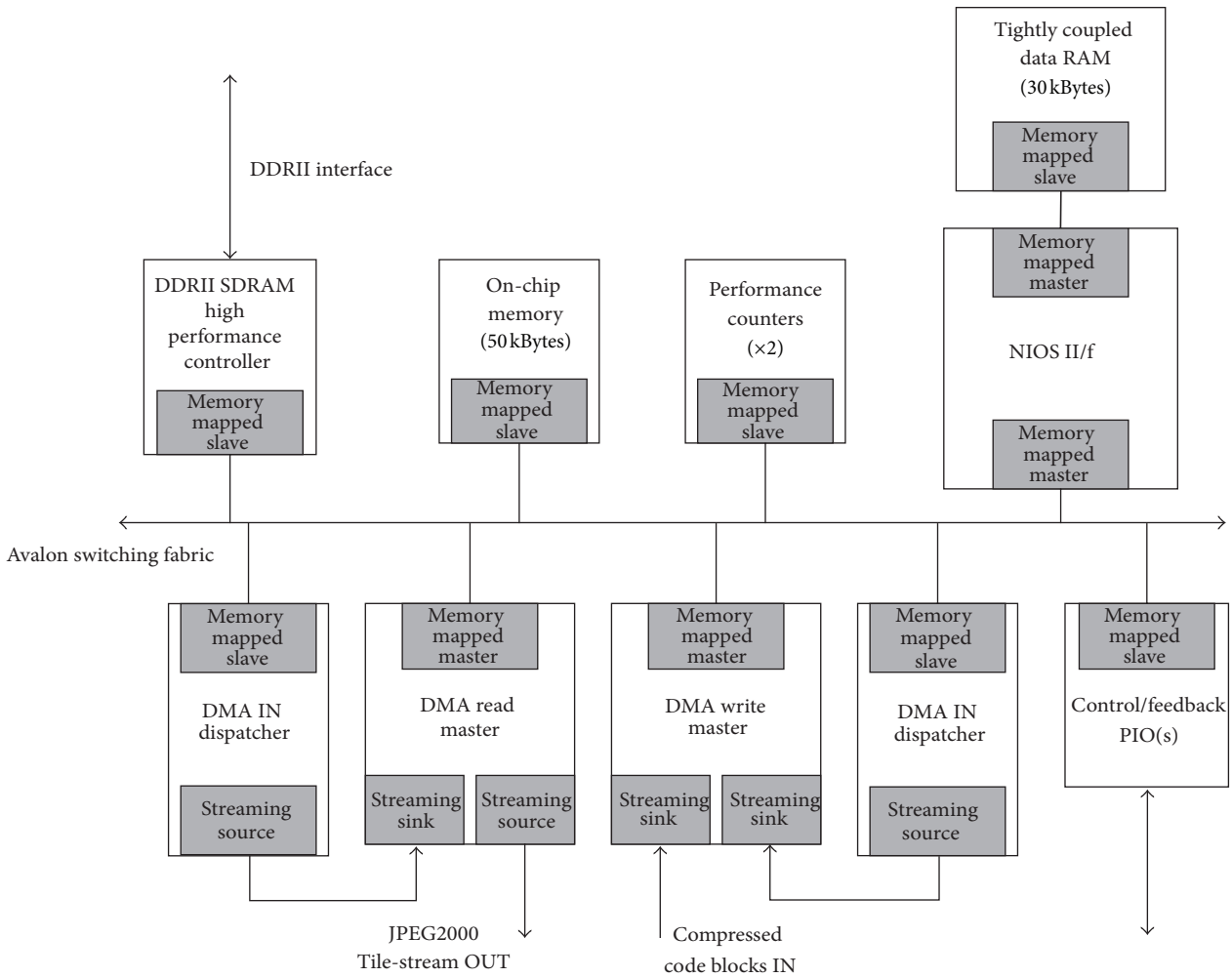

Figure 4: Block diagram of implemented NIOS II SoPC system.

Table 1: Hardware modules present in the NIOS II SoPC system.

Component	Clock (MHz)	Interface
NIOS II/f	290	Memory mapped
DDRII SDRAM controller	200	Memory mapped
On-chip RAM	290	Memory mapped
Tightly-coupled RAM	290	Memory mapped
DMA dispatcher (IN)	100	Memory mapped, streaming
DMA write master	100	Memory mapped, streaming
DMA dispatcher (OUT)	200	Memory mapped, streaming
DMA read master	200	Memory mapped, streaming
Feedback/control PIOs	100	Memory mapped
Performance counters	290	Memory mapped

dispatcher receives read/write commands from the system as a memory mapped slave. These commands are then passed to the connected read/write master which performs the memory transfer. This interaction is shown in Figure 4. Since the read/write masters are separate modules, the processor is free to complete other tasks while the data transfer is pending.

The SoPC system also has a number of parallel I/O (PIO) ports which provide direct communication with the hardware. The PIOs present in the system serve two main functions. First, PIOs enable the system to receive interrupts from the Tier I module upon completion of a tile. Second, PIOs are used to read/write registers which are in turn

accessed by the host device for control or feedback. All of the PIOs operate at the same 100 MHz clock rate.

Finally, there are two performance counters, clocked at 290 MHz, present in the system. These counters are used to profile the system performance, the results of which are detailed in Section 3.5. Two counters are necessary to enable profiling of nested functions.

3.5. Optimizations. A number of optimizations are made to the system in order to increase throughput. As the implementation couples a coprocessor with existing, optimized, and processing modules, these optimizations are focused on the NIOS II processing core. Figure 5 shows the impact of these optimizations on the processing time for a single 1024×1024 tile. Optimizations are applied on top of each other in a sequential manner to demonstrate the combined impact of all optimizations on the average processing time. Therefore, the final data point is the average processing time resulting from the combination of all of the previous optimizations. High-resolution images are compressed and the Tier II processing time is then divided by the number of tiles to yield an average Tier II processing time.

Typically, floating point operations are much more computationally expensive than integer operations. This expense is compounded when using a RISC architecture such as the NIOS II processor. While most of the Tier II algorithm is performed using integer calculations, a logarithm is required to calculate the length of a codeword segment. Codeword segments are used to signal the number of bytes contributed to a packet by a code block. The number of bits required for to store the codeword is given by

$$\text{bits} = L\,\text{block} + \lfloor \log_2 (P) \rfloor, \tag{1}$$

where L block is the state variable for the current code block and P is the number of coding passes contributed to the current code block [1]. As this calculation is necessary for each code block, the computational cost is high. Software profiling reveals that the calculation of (1) takes over 70% of the total Tier II processing time on the NIOS II processor.

Two approaches are taken to reduce the computation cost of this calculation. The NIOS II processing core supports the addition of custom instructions, such as user created HW implementations of specific operations. Additionally, the tools include a number of premade instructions, including custom floating point (FP) instructions [10, 15]. The custom FP hardware is enabled on the NIOS II core, and significant improvement is seen in system throughput, yielding a 66% decrease in processing time. However, this calculation still takes over 35% of the total Tier II time.

To further reduce the processing burden of the binary logarithm, the standard library call is replaced with a custom implementation, referred to as a lookup table (LUT) implementation. The binary logarithm of a number can be thought of as the number of bits required to represent that number in binary. Since only the floored result is used, this calculation can be performed using only logical right shifts and addition. Pseudocode for the algorithm is shown in Algorithm 1. The implementation described here is for a 32-bit unsigned integer and increments the result based on a series of comparisons to powers of two. First, if the input's most significant bit (MSB) is in the upper 16 bits, the output is incremented by 16, since at least 16 bits are required to represent the input. The upper 16 bits of the input are then used for subsequent comparisons by shifting the input right by 16. Then, if the MSB is contained in the upper half of the remainder, the result is incremented by 8 and the upper 8 bits are used for the next comparison. This procedure is repeated 3 more times, operating on the upper half of the remainder of the input word. Figure 5 shows that using the LUT implementation yields better performance than the custom FP hardware and is used in the final implementation in favor of the custom instructions. This implementation removes the need for costly floating point arithmetic, instead of leveraging inexpensive bit shifts and addition. The use of an LUT implementation reduces the calculation time to 3% of the total Tier II time, down from over 70%.

As mentioned in Section 3.4, the 30 kByte bank of on-chip memory is configured as "Tightly Coupled" memory (TCM). The NIOS II core can be configured to have additional data master ports for any number of TCMs, which must be on-chip. TCMs bypass the NIOS II cache and provide guaranteed low-latency memory access to specially designated instructions or data [10, 13]. These instructions or data are designated as tightly coupled through specific linker commands at compile time. In this implementation, specific data structures which are frequently accessed during Tier II are designated as tightly coupled to guarantee consistent performance. While Figure 5 shows a minor improvement in processing time, using TCMs to bypass the cache can be useful for avoiding data corruption while parallel processing is performed.

The downfall of utilizing SDRAM to serve as a tile buffer is that memory accesses to the device are considerably slower than accessing on-chip RAM. During the Tier II processing, multiple reads and writes are performed within this memory space for each code block processed. Additionally, each code block must be copied from the input data buffer to the filestream buffer. This copy alone has a detrimental impact on the overall throughput of the system. To address this issue, a third DMA controller is added to the system which masters only the SDRAM controller, allowing the system to schedule nonblocking memory copies within the SDRAM address range. The addition of this DMA controller hides the latency associated with large memory copies, allowing the processor to continue processing the next set of instructions while the transfer completes. Introduction of the third DMA controller results in a 48% decrease in processing time as shown in Figure 5 when compared to the previous implementation without the additional DMA controller. Care is taken to ensure that all pending memory copies have completed prior to writing out the completed filestream.

The final optimization made to the NIOS II system is to ensure that the system acts as a pipeline in order to maximize throughput of the system. Initially, the system is designed without the code block FIFO seen in Figure 3. Instead, the NIOS II system is directly coupled with the Tier I

```
begin
   result = 0
   if input >= 65536 then input >>= 16; result+ = 16; fi
   if input >= 256 then input >>= 8; result+ = 8; fi
   if input >= 16 then input >>= 4; result+ = 4; fi
   if input >= 4 then input >>= 2; result+ = 2; fi
   if input >= 2 then input >>= 1; result+ = 1; fi
   return result;
end
```

Algorithm 1: A lookup table implementation of a floored binary logarithm of a 32-bit unsigned integer.

output, reading code blocks as they become available. While a simplified approach, the downfall is that the Tier I processor must wait for the code block to be read before proceeding to the next code block. In order to eliminate this idle time, the code block FIFO in Figure 3 is added. Tier I simply writes to the FIFO and signals the Tier II module when a full tile has been buffered. Since the code block FIFO resides in SDRAM, a large 16 MByte FIFO is used which is capable of buffering multiple tiles in the event Tier II falls behind. Tier II then reads entire tiles as they become available, instead of reading each code block. Pipelining has two distinct impacts on the system throughput. First, the Tier I encoder is now free to process code blocks as fast as possible, therefore increasing the system throughput. Additionally, the interrupt latency associated with posting read requests is reduced since there is only one read request per tile, instead of one request per code block. In total, four optimizations yield a 91% reduction in Tier II processing time with the NIOS II system.

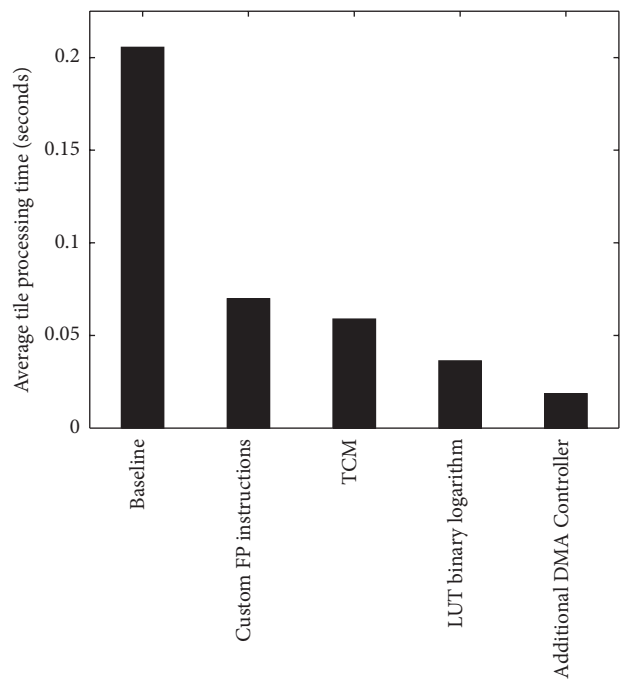

Figure 5: Combined impact of optimizations on average processing time for single 1024 × 1024 tile (optimizations are combined to yield lowest processing time).

4. Analysis of Results

4.1. Performance and Analysis. The performance of the JPEG2000 SoC is measured using three 2048 × 2560 ISO test images "Café," "Woman," and "Bike." For all tests, each image is compressed and the respective processing times of all three images are averaged together. First, the overall performance of the system is measured with varying numbers of parallel block coders in order to determine the optimum number of block coders and the corresponding throughput of the system. The number of parallel block coders is increased from 1 to 20. The impact of an increased number of parallel block coders on the image processing time as well as the Tier I and Tier II times is shown in Figure 6.

The total HW time in Figure 6 shows that, as expected, additional block coders have a large impact on the average image processing time. This is especially true when the block coder count is increased from 1 to 6, resulting in a 67% drop in average processing time. Processing time continues to decrease as the block coder count is increased beyond six with a minimal processing time of 0.22 seconds achieved when 18 block coders are present. Negligible change in performance is seen with block coder counts beyond 18. The steps seen in the total HW time are attributed to lack of saturation of the DMA controller between steps, resulting in jumps in performance when data is presented to the DMA controller at a faster rate.

The overall throughput of the system is limited by the Tier II processing time. Figure 6 shows the time spent performing Tier I and Tier II processing, as well as the total time, for a variable number of block coders. Figure 6 shows that the Tier II processing remains constant at 0.121 sec while the Tier I time decreases as more block coders are added, as expected. As the Tier I processing time approaches the Tier II time, the total processing time begins to flatten out, with little change beyond 18 block coders. Additional block coders have little impact on the total processing time since it has no impact on the Tier II processing time.

Figure 6 shows that the proposed architecture scales well with an increasing number of parallel block coders. This compares favorably with the SoC architecture presented in [5], which does not scale as well as the proposed architecture.

Table 2: Hardware resource comparison for a system with 4 block coders.

System	Block coders	LCs	Memory	Clock (MHz)
[5] DWT/TierI	4	15,268	622,976	50
DWT/TierI	4	13,123	637,952	100/200
DWT/TierI	18	43,690	1,417,472	100/200
TierII (NIOS)	N/A	10,996	923,008	290

Figure 7 shows a comparison between the performance of the proposed architecture and the architecture presented in [5]. The image processing time from the fastest implementation of [5] is overlaid onto the total HW time from Figure 6 for one to ten block coders, using the same set of ISO test images. Results are compared from one to ten block coders since [5] only provides results up to ten coders. It is clear that while [5] outperforms at lower block coder counts, these gains are erased once the count is increased beyond five encoders. At this point the [5] architecture has plateaued while the proposed architecture continues to improve. With 10 parallel block coders, the proposed implementation outperforms [5] by 39%. When the system is scaled to 18 block coders, the proposed design outperforms [5] by 58%. By allowing the other processing modules to operate outside the contexts of the SoPC system, the proposed architecture is able to take full advantage of multiple parallel block coders without the limitations necessarily imposed by the Avalon switching fabric. While extremely effective at integrating multiple different peripherals into a single system, the scheduling and arbitration overhead associated with the NIOS II processing system impose restrictions on the system throughput. By creating a pipelined architecture which utilizes the NIOS II processing core as a separate unit we are able to leverage the flexibility of the NIOS II system while still maintaining the speed of a pipelined encoder.

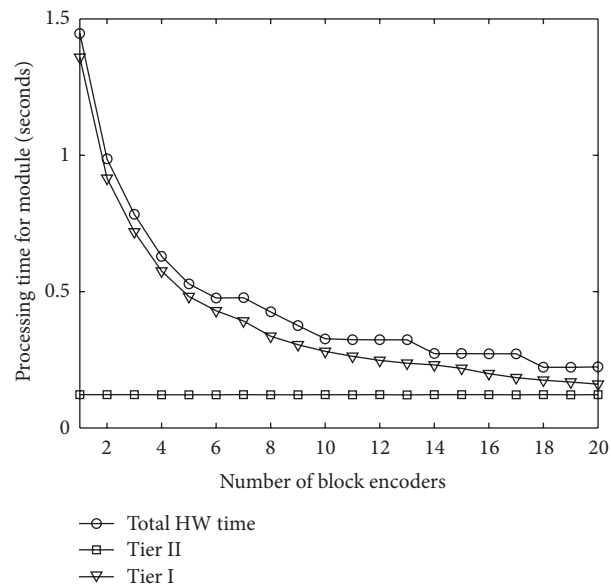

Figure 6: Performance profile of encoding pipeline with parallel block coders.

4.2. Hardware Synthesis Results. The proposed implementation is synthesized on a Stratix IV FPGA using Altera Quartus 10.1. For the purposes of comparing the proposed design to [5], the design is synthesized with 4 parallel block coders. With this encoder count, [5] slightly outperforms the proposed system, but these gains are quickly erased with additional block coders (Figure 7). The hardware costs of the proposed system and [5] are shown in Table 2. Costs for both the 4 and 18 block encoder implementations are shown. The hardware costs for the proposed system are split into two categories: the DWT and Tier I modules and the NIOS II system which performs Tier II. The results are presented in this manner to provide an accurate comparison to [5], which only simulates the NIOS II system, so the hardware costs only reflect the DWT and Tier I modules.

Table 2 shows that the hardware resource costs of the implemented NIOS II system are minimal compared to the other processing modules, whose costs increase as more block coders are added. However, the NIOS II system does have a high memory cost. This is due to the 80 kBytes (50 kByte and 30 kByte banks) of on-chip RAM used along with the instruction and data caches built into the processing core. These large memory modules are necessary to run more complex code requiring larger stack and heap regions in memory. However, due to the flexibility of the NIOS II system, these costs could be shifted off-chip by utilizing more of the SDRAM. However, since the target platform utilizes a Stratix IV FPGA [12] with a large amount of on-chip memory, this is not an issue for the proposed design.

Table 2 shows that the proposed DWT and Tier I designs are comparable in cost to [5] with 4 parallel block coders. The main difference is that the proposed design is capable of higher clock speeds, with the DWT and Tier I running at 100 MHz and 200 MHz, respectively. This results in a DWT capable of processing one pixel every 7 ns as opposed to 20 ns per pixel in [5]. The higher performance of the DWT prevents the parallel block coders from becoming starved as they do in [5], which yields increased performance up to 18 block coders as opposed to [5], which peaks at 4 block coders due to starvation of the block coders. Instead, the proposed design is limited by the throughput of the NIOS II system.

5. Conclusion

This paper proposed a fully embedded JPEG2000 SoC which utilized an Altera NIOS II processor as the embedded EBCOT

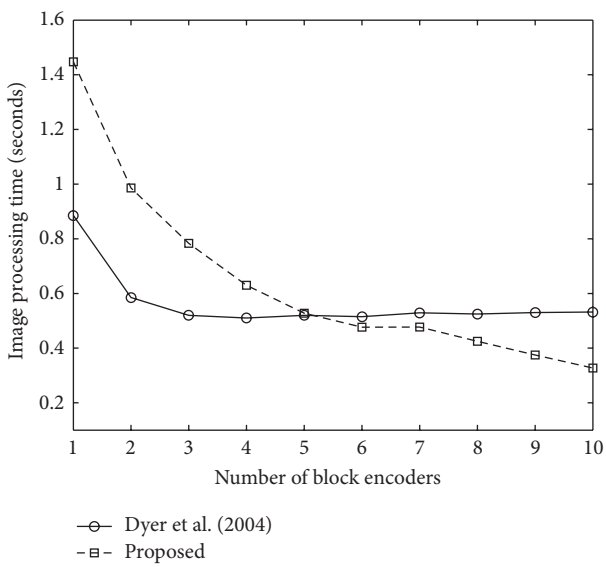

Figure 7: Performance comparison to other SoC implementations.

Tier II processing module. The proposed system is synthesized on a Stratix IV FPGA and yields a 39% performance increase over other JPEG2000 SoC implementations with the same number of parallel block coders. While [5] offers a more flexible and reconfigurable design, the pipelined architecture of the proposed design allows for a design capable of scaling to higher numbers of parallel block coders with comparable increases in system throughput. While limited by the performance of the NIOS II performance, future implementations could mitigate this with optimizations to the Tier II algorithm. Additional NIOS II processing cores could also be added to the system to share the processing load, assuming the availability of adequate hardware resources.

In addition to a high performance and scalable design, the proposed system also demonstrates the feasibility of utilizing an embedded softcore processor as a dedicated processing unit within a pipeline. Ease of reconfiguration and support for a variety of peripherals allowed for seamless integration of the NIOS II system into an existing encoding pipeline. The proposed design also demonstrates techniques for optimizing the performance of software running on the NIOS II through the use of custom instructions and additional peripherals.

References

[1] ISO/IEC 1. 29. 15444-1, "JPEG, 2000 Part I Final Committee Version 1. 0," 2004.

[2] D. Santa-Cruz, R. Grosbois, and T. Ebrahimi, "JPEG 2000 performance evaluation and assessment," *Signal Processing*, vol. 17, no. 1, pp. 113–130, 2002.

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

[4] K.-F. Chen, C.-J. Lian, H.-H. Chen, and L.-G. Chen, "Analysis and architecture design of EBCOT for JPEG-2000," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '01)*, pp. II765–II768, Sydney, Australia, May 2001.

[5] M. Dyer, S. Nooshabadi, and D. Taubman, "Design and analysis of system on a chip encoder for JPEG2000," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 2, pp. 215–225, 2009.

[6] N. R. Kumar, W. Xiang, and Y. Wang, "An FPGA-based fast two-symbol processing architecture for JPEG 2000 arithmetic coding," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '10)*, pp. 1282–1285, Dallas, Tex, USA, March 2010.

[7] M. Dyer, D. Taubman, and S. Nooshabadi, "Improved throughput arithmetic coder for JPEG2000," in *Proceedings of the International Conference on Image Processing (ICIP '04)*, pp. 2817–2820, October 2004.

[8] L. Liu, N. Chen, H. Meng, L. Zhang, Z. Wang, and H. Chen, "A VLSI architecture of JPEG2000 encoder," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 11, pp. 2032–2040, 2004.

[9] L. Liu, Z. Wang, N. Chen, and L. Zhang, "VLSI architecture of EBCOT Tier-2 encoder for JPEG2000," in *Proceedings of the IEEE Workshop on Signal Processing Systems—Design and Implementation (SiPS '05)*, pp. 225–228, November 2005.

[10] Altera Corporation, "NIOS II Processor Reference Handbook," 2010.

[11] E. J. Balster, B. T. Fortener, and W. F. Turri, "Integer computation of lossy JPEG2000 compression," *IEEE Transactions on Image Processing*, vol. 20, no. 8, pp. 2386–2391, 2011.

[12] GiDEL, "ProceIV Data Book," 2011.

[13] Altera Corporation, "NIOS II Software Developer's Handbook," 2011.

[14] Altera Corporation, "SOPC Builder User Guide," 2010.

[15] Altera Corporation, "NIOS II Custom Instruction User Guide," 2011.