A Run-Time Reconfigurable 2D Discrete Wavelet Transform Using JBits

Jonathan Ballagh[1], Peter Athanas[1] and Eric Keller[2]

[1]Virginia Tech, 340 Whittemore Hall, Blacksburg, VA 24061

[2]Xilinx Inc., 2300 55^{th} Street, Boulder, CO 80301

ABSTRACT

With the growth in high performance multimedia applications, specialized hardware for certain tasks is desirable. While ASICs provide a solution addressing performance, they are unable to provide an optimal solution for a given problem instance. FPGAs can be used with run-time reconfiguration to dynamically customize a circuit. Optimizations leading to faster circuits and reduced logic can result. The paper discusses the implementation of a run-time parameterizable 2D Discrete Wavelet Transform core using the JBits tool suite. The motivation for such a core is discussed, as well the benefits afforded by dynamic circuit specialization.

Keywords: Wavelets, DWT, Compression, RTR, JBits, FPGA

1. INTRODUCTION

Multimedia applications require a great amount of processing which can benefit from hardware acceleration. While ASICs provide a possible solution, programmable logic provides the ability to be reconfigured and optimized to fit a specific problem instance. Run-Time Reconfigurable (RTR) systems distinguish themselves from traditional static design implementations by performing logic and routing customization just prior to circuit use. FPGAs are well suited for RTR systems because of their reprogrammable nature. While FPGAs provide a time to market advantage and field upgradability, dynamic circuit specialization has not made it into mainstream commercial use. A reason for the lack of commercial success is the scarcity of software design tools that support RTR. It has been shown that specialization of a circuit can increase performance, decrease I/O requirements, and reduce logic resources used [7]. JBits is a tool suite that has radically changed the design approach. JBits provides direct access to the bitstream through a Java application programming interface (API), allowing the configuration of individual resources in a Xilinx® FPGA. This allows for rapid bitstream modifications and provides an environment in which the FPGA design and the software interacting with the hardware reside in the same program. Using RTR with JBits, a Discrete Wavelet Transform was designed. Discrete Wavelet Transforms are becoming increasingly popular in applications such as image compression. The ability to adapt to a given set of data can produce performance and results superior to fixed designs.

2. RUN-TIME RECONFIGURATION

While much research has been done documenting the advantages of run-time reconfigurable (RTR) computing [15–17], software support has lagged behind. The lack of software has made it difficult to fully exploit all the advantages RTR provides. RTR systems are characterized by their ability to dynamically specify and modify a circuit at run-time. It has been shown that RTR can increase performance, decrease I/O requirements, and reduce circuitry [7]. An example optimization is when one of the inputs to a multiplier is fixed to a constant value for a given problem instance. The multiplier can be replaced by a multiplication by a constant [10], which results in smaller resource consumption and faster speeds. Until recently, RTR systems relied on mainstream tools to compile a design into a bitstream. This results in large time, memory, and file overheads. While these systems claim to support RTR, they are more accurately described as run-time customizable. Run-time customization systems differ from RTR systems in that RTR systems are modified several times throughout the execution of the application. Using mainstream tools may require precompiling all of the different instances of the design and then switching the configuration as the application demands [20]. Designs with a large number of contexts or where the contexts are unknown at design time are unsuited to this approach.

Further author information:

J. Ballagh: E-mail: jballagh@vt.edu

P. Athanas: E-mail: athanas@vt.edu

E. Keller: E-mail: eric.keller@xilinx.com

3. JBITS

In the past, design synthesis and implementation tools have acted as the main obstacle preventing efficient RTR of FPGA based hardware designs, by increasing configuration times beyond acceptable limits. The Java-based JBits API eliminates that barrier by providing direct access to the Xilinx configuration bitstream. The development of JBits [13] has provided the necessary tools required for the implementation of effective RTR designs for Xilinx devices. This section provides a brief overview of the JBits API.

A typical JBits program creates logic in a new bitstream, modifies the logic and interconnects in an existing bitstream, or performs an analysis of a bitstream. The JBits API has four main methods. The first two allow a bitstream to be read and written from a file. Another method allows the setting of a resource, i.e. a programmable interconnect point (PIP), to a certain value, i.e. on. The last method allows reading the state of a resource from the bitstream. The rest of the JBits API is a set of constants that define the configurable resources and their settings.

JBits itself provides a very low-level design environment. Designing at this level can be very difficult for all but the most trivial designs. However, by using the object-oriented paradigm afforded by Java, tools can be built upon JBits to provide higher levels of abstraction. An example of some of the tools created is listed below:

XHWIF - The Xilinx Hardware Interface is an API that provides methods to communicate with FPGA based boards. This includes a method for reading bitstreams from the FPGAs. There are methods to write bitstreams to the FPGAs, step a clock, and write to and read from the memories on the board. Essentially, XHWIF provides a universal interface for communicating with different boards.

XHWIF Server - Embedded systems that implement the XHWIF API can communicate with local or remote host systems. An application using the XHWIF API can communicate with a board on a remote computer through a networked connection. The remote computer runs an XHWIF server to handle communication between the local board and the application accessing the server.

VirtexDS - The VirtexDS [8] is a device simulator that accurately models a Xilinx Virtex™ [25] device. Applications can transparently switch between the VirtexDS and an actual FPGA.

JRTR - The JRTR [12] API is an extension of the JBits API to take advantage of the partial reconfiguration support provided by Virtex devices. This interface provides a caching model that automatically tracks changes to configuration data and only the modified data is written to or read back from the device.

BoardScope - BoardScope [21] is an application that graphically displays state information within the FPGA. It makes use of the XHWIF interface, which allows access to a remote board as well as the VirtexDS. For each FPGA on the board, BoardScope displays the flip-flop states and the LUT contents for each CLB. The ability to display the connections to each input or output of a CLB is also provided. Both a command line and graphical interface are provided. An example command is stepping the clock. After the clock is stepped, the devices are read back automatically and the graphical displays are updated. The information is stored in memory for user specified signals and can be displayed in a waveform viewer.

JRoute - Built upon JBits, JRoute [14] is a run-time auto-router with different levels of control. There is also a template router where an application can define a generic route consisting of types of resources that are to be used. JRoute then determines an actual route that follows the template. A list of resources to be connected may also be specified. In cases where a net is no longer needed, a net or part of a net may be unrouted in order to free up resources.

RTPCores - The JBits run-time parameterizable (RTP) core [22] specification provides a means for abstracting away the low level configuration calls, thereby creating an environment similar to traditional hardware design languages (HDLs) while concurrently affording the ability to make bitstream level modifications. Using RTP cores, bitstream calls are encapsulated by core primitives, so that higher abstraction cores can be designed in the manner similar to a traditional structural HDL approach. Net and bus classes specify connections between cores. The JRoute API performs the necessary routing for the defined nets and buses. Because of Java language constructs, such as for loops, parameterization is simplified. In addition to producing smaller, faster circuits, this approach permits the construction of circuits which would not be feasible with compile-time parameterization. The ability to react to command line parameters, device type, user input, real-time input or circuit state provides capabilities for circuit designers far beyond what is available using static circuit design tools such as schematic capture and hardware description languages.

4. WAVELETS

Recently, wavelets have received a great deal of attention, mainly due to their wide appeal in a variety of applications, ranging from image editing and compression to electrocardiogram analysis. The wavelet transform rectifies the shortcomings of the Fourier Transform, providing a representation of a given signal in both time and scale domains [3]. Utilizing this methodology, the wavelet transform has been shown to significantly outperform the discrete cosine transform (DCT) in image compression applications, leading to its inclusion in the JPEG 2000 standard [27]. Wavelets are also aptly suited for image editing and progressive transmission applications since they provide a multiresolution decomposition of a signal.

Wavelet transforms are typically implemented in software. Although software allows for a great deal of flexibility of operation, performance is often insufficient for high-end multimedia applications such as video and sound compression. It is also often desirable to offload redundant computations - such as compression coding - to a hardware accelerator in order to free the processor for other tasks.

Although a number of ASIC DWT architectures have been explored [1,2], they offer little flexibility in the way of operation customization. Although wavelet parameterization can be accomplished through coefficient loading into registers, performance will be suboptimal for certain wavelets since extraneous resources are used regardless of the number of required wavelet coefficients.

FPGAs provide a unique solution in which devices have the ability to be reconfigured while also affording clock speeds suitable for many applications. While clock speeds are not comparable to ASICs, RTR can create higher performance circuits in FPGAs [19]. Although this has made FPGAs attractive for many applications, reconfiguration of these devices has been burdened by long compile times, thereby preventing dynamic reconfiguration within reasonable time constraints. JBits offers a solution to this problem by encapsulating low level configuration control in a high level language. Efficient run-time design reconfiguration can be achieved with this approach.

4.1. Discrete Wavelet Transform Background

The *Discrete Wavelet Transform* (DWT) converts a signal from the time domain into the time-scale domain. Although the Fourier Transform provides information about the frequency content of a signal, it does not preserve time information that indicates when those frequencies occur. Based on a multiresolution analysis framework, the DWT captures both time and frequency information [6].

The DWT of a signal can be computed by passing a signal through a 2 channel filter bank. The filter bank is comprised of a high-pass and a low-pass filter. The low-pass filter branch generates the average DWT coefficients from the signal while the high-pass branch generates the detail DWT coefficients.

The coefficients of low-pass filter define the scaling function of the wavelet. The high-pass filter coefficients define the wavelet function. In the case of orthogonal wavelets, the high-pass and low-pass filters constitute a quadrature mirror filter (QMF) pair [4]. If orthogonal wavelets are used in the DWT, the same wavelet is used in the reconstruction process. On the other hand, using biorthogonal wavelets requires a separate wavelet for decomposition and reconstruction of the signal. The QMF relationship of the high-pass and low-pass filters is not required with biorthogonal wavelets.

The decimated output of the high-pass filter constitutes the first octave output. As the filter pair processes the signal, the output is downsampled by a factor of two. Filtering the signal controls the resolution of the signal, while decimating the signal controls the scale. Scale and frequency are inversely proportional such that higher frequencies correspond to lower (i.e. finer) scales, while lower frequencies correspond to higher (i.e. coarser) scales. Because the filters separate the frequency bandwidth, the filter pairs produce different resolutions or levels of detail.

Since the output signal is downsampled by two, the filter outputs can be stored in the original signal space. The average signals are stored in the first half of the space, while detail signals are stored in the latter half. The average signals are then processed again through the same set of filters producing a second set of average and detail coefficients. This DWT decomposition of the signal continues until the desired scale is achieved. Mallat illustrates this process in the Pyramid Algorithm [5,6].

Two dimensional signals, such as images, are transformed using the 2-D DWT. The 2-D DWT operates in a similar manner, with only slight variations from the 1-D transform. Given a 2-D array of samples, the rows of the array are processed first with only one level of decomposition. This essentially divides the array into two vertical halves, with the first half storing the average coefficients, while the second vertical half stores the detail coefficients. This process is repeated again with the columns, resulting in four quadrants within the array defined by filter output. Figure 1 shows a one level decomposition using the 2-D DWT. The filter output that results from two low-pass filters, labeled LL in Figure 1, is then processed again in the same manner. The process is repeated for as many levels of decomposition as are desired.

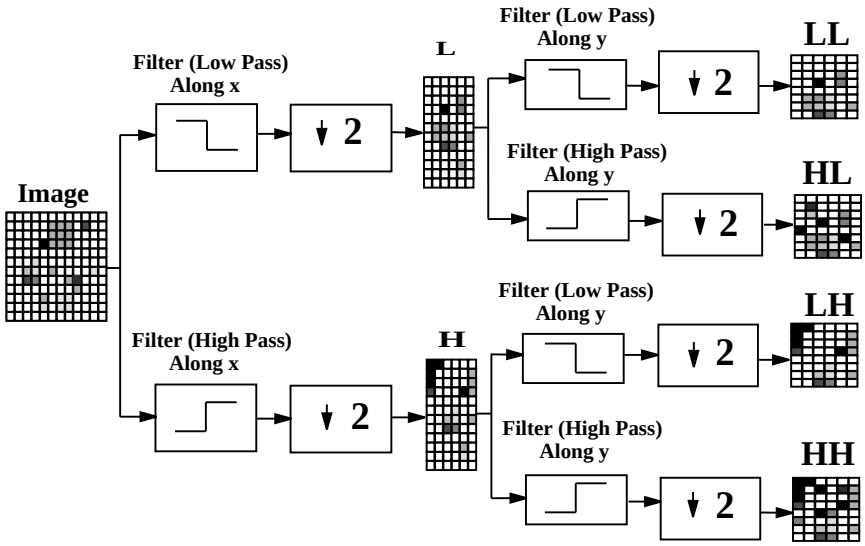

Figure 1. One level of decomposition using the 2D DWT.

5. 2-D DISCRETE WAVELET TRANSFORM CORE

Dynamic circuit specialization of the RTP Wavelet Transform Core affords significant performance increases in image compression applications. It has been shown that the selection of transform wavelet relates to the resulting S/N ratio of a reconstructed image [9]. Therefore, choosing the appropriate wavelet can result in better performance over other wavelets for a particular image. Since the wavelet used in the transformation process can either be lossy or lossless, it is also possible to control the compression ratio achieved with additional circuitry directly through the selection of the wavelet.

An implementation of the 2D Discrete Wavelet Transform, the *DWT2D* core, was designed using the JBits environment for use on Virtex FPGA devices. The system level diagram of the core is shown in Figure 2. The core features a direct implementation in which the same filter circuitry is reused to process both the row and columns of the 2D signal array. The core requires two SRAMs of size W x H bytes, where W represents the image width and H represents the image height. Although more efficient designs have been implemented in FPGAs that require less memory [29], the folded architecture was thought most suitable for a JBits implementation to keep control logic simple. The core is parameterized by image height and width values, coefficients for the high-pass and low-pass filters, and desired filter resolution. Parameterizing the core with filter coefficients allows any wavelet to be used in the transform, assuming ample FPGA resources are available during core instantiation. This section begins with a brief discussion of the *DWT2D* control logic and then discusses the implementation of a sequential *FIRFilter* core as a hierarchical composition of RTP cores.

5.1. Control Logic

The 2D Wavelet Transform core control logic is responsible for performing memory address generation and data multiplexing. Two address generator RTP cores provide the input and output address values for the SRAM memory. The address generator cores also supply the signals necessary for controlling the read/write and tri-state operations of the external memory.

The transform core interfaces to two separate SRAMs. During operation, one bank provides the necessary core input, while the other bank stores the intermediate coefficient data. After processing the image rows, the roles of both memory banks are swapped. This swapping can be realized in hardware using multiplexers and controlling the tri-state enables of the FPGA IOBs. Since each memory bank must be able to read and write data, both input and output address buses are connected to the memory address lines through a multiplexer. In the same regard, a multiplexer ensures that the correct memory data bus acts as input into the high-pass and low-pass filters (Figure 2).

5.2. RTP Sequential FIR Filter Core

Two sequential FIR Filters perform most of the computation for the wavelet transform core. The RTP FIR Filter defines a hierarchy of register, multiply, and adder structures. Each core makes advantageous use of the low level bitstream access

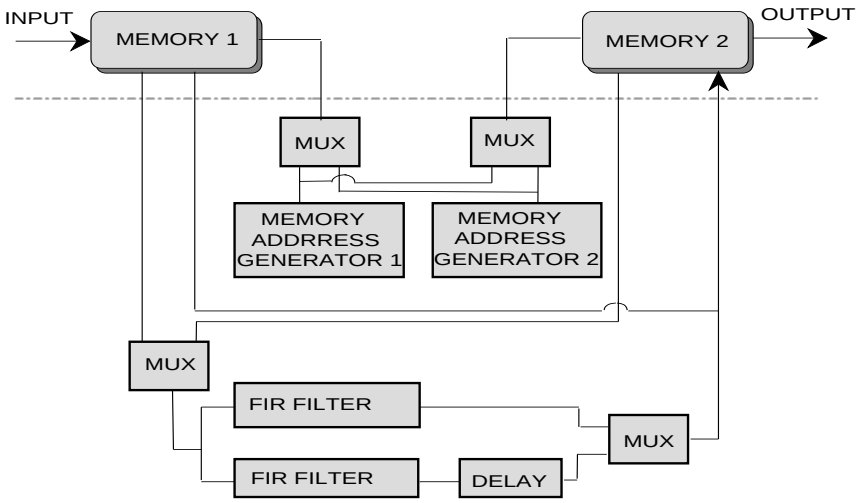

Figure 2. Diagram of a complete DWT core.

afforded by JBits in order to guarantee optimal placement and speed. Reconfiguration of the filter core involves software configuration of the child cores and routing resources in such a way that the highest degree of performance is achieved during operation.

The RTP FIR Filter core is created in a three-step process. Filter input and output buses are passed to the filter constructor during instantiation. The widths of these buses define data path bit widths and filter resolution. The filter core must then be anchored to physical coordinates on the FPGA. This is accomplished by modifying the filter cores offset field, in which CLB row and column coordinates are defined. After placement, an array of doubles representing filter coefficients is passed to the cores *implement* method, in which all child sub cores are instantiated, and the bitstream is configured accordingly.

Assuming a given filter is of order n, then n registers and multipliers are required, as well as $n - 1$ adders. A constant multiplier core is instantiated for each coefficient in the double array and loaded with coefficient value. If a coefficient is a power of two, additional optimization can be achieved through performing a shift directly through the use of routing resources. While delay registers may then be required, Virtex devices have the ability to implement a 16-bit serial shift register in a single LUT. This reduces logic as well as routing resources. An optimal adder tree core is responsible for summing the multiplier outputs. The resulting filter structure is shown in Figure 3. The higher abstraction RTP cores required for filter design include constant coefficient multiplier (KCM), register, and adder tree RTP cores.

5.3. RTP KCM

Because filter coefficients are known at run-time, constant coefficient multipliers are used in the filter design. The KCM [18] provides an implementation of a constant coefficient multiplier that is ideally suited for the Virtex family architecture. A KCM uses pre-computed multiplier results stored in LUT resources in combination with an adder to execute the multiplication operation. The KCM requires fewer resources and results in less latency than variable coefficient multipliers.

The JBits KCM core implementation, shown in Figure 4, is comprised of ROM cores with outputs summed by an RTP adder tree core. Using JBits, the KCM can be reconfigured at run-time to produce any desired coefficient resolution. 16 x n distributed ROM cores are used for the KCM implementation, where n is equal to the coefficient resolution plus the input bus width. During reconfiguration, each of the 16 ROM locations are loaded with the value of coefficient k multiplied by the ROM location index. The product is left shifted by a multiple of four, depending on input lines addressing the ROM.

5.4. RTP Adder Tree

An RTP Adder Tree core allows any number of inputs with varying bit widths to be summed together. The core is comprised of adder cores for summing two inputs and delay cores that provide the required delay in the case of unbalanced trees. The number of adder cores required for a tree with l inputs is simply $l - 1$. The adder tree core exemplifies the advantages of run-time reconfiguration, since an optimal tree is derived for any given number of inputs. Each stage of the adder tree is pipelined,

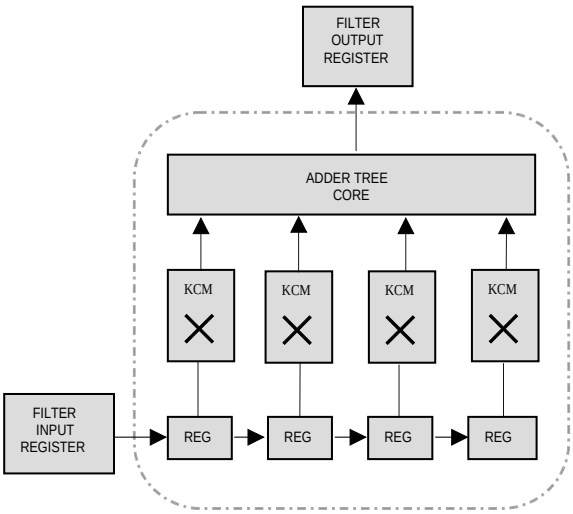

Figure 3. Diagram of a FIR filter.

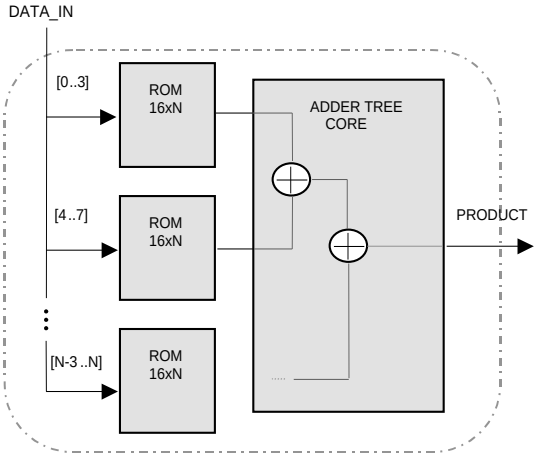

Figure 4. Diagram of the KCM.

thereby allowing for increased clock frequencies. The latency of the core from input to output is $\lceil log_2 l \rceil$. Each adder core is horizontally adjacent to the previous adder. The delay core is adjacent to the adder in which delay is required, and makes use of LUT based shift registers.

Routing between adders follows the hierarchical tree structure shown in Figure 5. Using this approach, routing distances between adder levels is minimized.

6. SIMULATION

Using the Virtex Device Simulator, JBits provides a bitstream level debugging environment that supports RTR. External SRAM memory models were designed and interfaced with the simulator through the SimulatorClient and SimulatorServer objects [23]. The models provided a generic memory interface, including data, address, CE, and R/W signals. Through the TCP/IP communication channel with the device simulator, the models were able to probe and stimulate internal states of the simulated Virtex device. The models allowed the SRAM values to be set and read back at any point during execution. Interfacing two of these SRAM models to the VirtexDS provided a complete debugging environment for the *DWT2D* core. To completely simulate

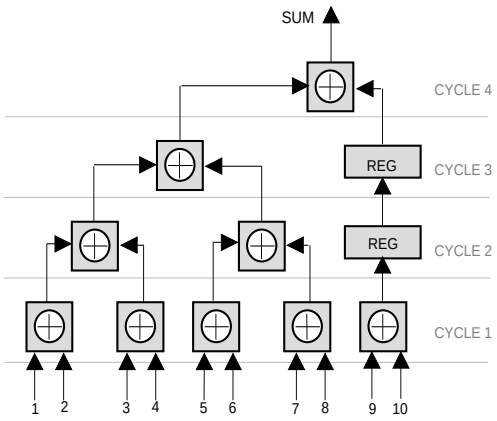

Figure 5. Layout of an Adder Tree with 10 inputs.

10,000 cycles with the VirtexDS and external SRAM models took 416.5 seconds*. The long execution time is a result of using a TCP/IP connection for the communication. Performance would be greatly improved if the external hardware models were more tightly integrated with the simulator.

7. I/O INTERFACING

A set of JBits classes have recently been introduced the enable RTR and design interface support to FPGA IOB resources [24]. These classes provide RTP core abstractions of the targeted FPGA hardware platforms and IOBs. By using these abstractions, IOBs can be added into a JBits design in the same manner as other cores, using net and bus signals to provide the interconnections. A user constraints file (UCF) is parsed to create a mapping between net names attached to the IOB cores and their associated physical pins on the device. The translation table obtained from the UCF file is used to map each top-level net to a physical IOB pin end-point. The endpoints are then attached to the nets so that they can be routed with JRoute. These classes eliminate device-dependent interfacing calls from JBits designs, allowing for platform portability.

For testing, the DWT core was interfaced to the SLAAC-1V board [28]. This PCI accelerator board contains three Virtex XCV1000s, memory, and switching components. The JBits RTR I/O classes were used to connect the DWT core to two SRAM banks of the SLAAC-1V. A SRAM RTP core was created that abstracted SLAAC-1V SRAM memories, and automatically configured the appropriate IOBs to support I/O with the SRAM data and control signals.

8. RESULTS

Shown in Table 1 is the performance of the wavelet transform with different numbers of taps using 12-bit FIR filters. The notation n/m filter bank refers to the number of taps in the low-pass filter, n, and the number of taps in the high-pass filter, m. Shown are the cases for a typical number coefficients use in wavelet based compression. Also shown in Table 1 is the amount of time it takes to run the design through the JBits design flow. Table 2 gives the performance of different configurations of the FIR filter used in the implementation of the DWT core. The frequencies for FIR filters with 12 bit resolution range from 147.514 MHz for 9 taps to 176.429 MHz for 2 taps. The performance of a JBits implemented FIR filter is comparable to a commercially available FPGA implementation such as by Xilinx [26].

A SLAAC-1V board was used, which has NBT SRAM that runs at 100 MHz. This caused the bottleneck in the system to be the DWT core. Within the DWT core, the control logic for memory addressing proved to be the critical path, running at 84.154 MHz for each filter configuration. At this frequency, a 512 x 512 image can be processed with a 4 level 2-D DWT at 120.86 frames per second. Multiple DWT cores can be implemented on a single FPGA to achieve higher processing rates.

*JDK1.2.2 with HotSpot was used on a 1 GHz Pentium III with 1 GB RAM running Windows 2000

	5/3 Filter Bank	2/2 Filter Bank	9/7 Filter Bank	6/6 Filter Bank
DWT Frequency	84.154 Mhz	84.154 Mhz	84.154 MHz	84.154 Mhz
JBits to Bitstream	12.978 sec	11.909 sec	15.642 sec	13.910 sec
Filter Configuration	2.524 sec	1.242 sec	5.258 sec	3.575 sec
CLBs	450	280	770	600

Table 1. The performance of the Discrete Wavelet Transform Core with 12 bit FIR filters. JBits to bitstream is the time it takes to execute a compiled java program that uses JBits to implement a DWT and connect it up. Included in this time is reading in a bitstream, implementing the DWT core, and writing the bitstream to a file. Filter Configuration is the time to instantiate both filters.

Taps	2	3	5	6	7	9
FIR Freq. (MHz)	176.429	172.980	164.880	157.356	151.745	147.514

Table 2. Performance of FIR filters with 12-bit resolution.

9. CONCLUSIONS AND FUTURE WORK

JBits is a unique design environment supporting run-time reconfiguration. Through the example of a wavelet transform core, it has been shown that JBits is a design environment with capabilities that other design environments cannot provide. JBits provides the low-level control that is generally not available with other tools. Through the use of RTP cores, JBits can be used as an HDL while also permitting run-time parameterization.

Wavelets are becoming increasingly popular in a variety of sound and image processing applications. A wavelet transform core that is parameterizable and fully optimized for the given wavelet has been designed. The individual components each make use of run-time reconfiguration to achieve an optimized circuit.

Although the paper presents a fully realizable RTP DWT core, additional optimizations may improve speed and decrease resource usage. Using JBits, a portion of the design can be modified without affecting the rest of the circuit. Layout floorplanning is a possible route of exploration to increase design performance. By experimenting with different layouts, there may be trade offs in area usage and maximum clock frequency. Work is in progress to create more efficient constant multipliers. [11] discusses a radix-4 booth encoded multiplier that has the advantage of reducing the number of configuration frames necessary to change the constant. Finally, the DWT core can be used in conjunction with quantizer and entropy coder cores to create a full RTR image compression system.

10. ACKNOWLEDGEMENTS

This work was supported by DARPA in the Adaptive Computing Systems (ACS) program under contract DABT63-99-3-0004.

REFERENCES

1. C. Chakrabarti and C. Mumford. “Efficient Realizations of Encoders and Decoders Based on the 2-D Discrete Wavelet Transform,” *IEEE Trans. of VLSI Systems*, pp. 289-298, 1999.
2. C. Chakrabarti, M. Vishwanath, and R. Owens, “Architectures for Wavelet Transforms: A Survey,” *J. VLSI Signal Processing Syst.*, pp.171-192, Nov. 1996.
3. I. Daubechies, “The wavelet transform, time-frequency localization and signal analysis,” *IEEE Trans. Inform. Theory*, vol. 36, pp. 961-1005, Sept. 1990.
4. A. Croisier, D.Esteban, and C. Galand. “Perfect channel splitting by use of interpolation/decimation tree decomposition techniques,” *International Conference on Information Science and Systems*, 1976.
5. S. Mallat. “Multifrequency channel decompositions of images and wavelet models,” *IEEE Trans. Acoustics Speech and Sig. Proc*, 37(12):2091-2110, Dec 1989.
6. S. Mallat. “A theory for multiresolution signal decomposition: The wavelet representation,” *IEEE Trans. Pattern Analysis and Mach. Intell*, 11(7):674-693, July 1989.
7. S. Guccione and D. Levi. “The Advantages of Run-Time Reconfiguration.” *Configurable Computing Technology and its uses in High Performance Computing, DSP and Systems Engineering*, Proc. SPIE Photonics East, J. Schewel (Ed.), SPIE - The International Society for Optical Engineering, pages 87-92, Bellingham, WA, September 1999

8. S. McMillan, B. Blodget, and S. Guccione. “VirtexDS: A Virtex Device Simulator.” To be presented at SPIE 2000, Boston MA, November 2000.
9. S. Saha and R. Vemuri, “Adaptive Wavelet Coding of Multimedia Images,” in *Proc. ACM Multimedia*, 1999 Orlando, Florida.
10. F. de Dinechin and V. Lefevre, “Constant multipliers for FPGAs,” in *Second International Workshop on Engineering of Reconfigurable Hardware/Software Objects (ENREGLE 2000)*, Las Vegas, Nevada, June 2000.
11. T. Courtney, R. Turner, and R. Woods, “Multiplexer Based Reconfiguration for Virtex Multipliers.” *Proceedings of the 10th International Workshop on Field-Programmable Logic and Applications*, Lecture Notes in Computer Science 1896, 2000.
12. S. McMillan and S. Guccione, “Partial Run-Time Reconfiguration Using JRTR,” *Proceedings of the 10th International Workshop on Field-Programmable Logic and Applications*, Lecture Notes in Computer Science 1896, 2000.
13. S. Guccione and D. Levi, “XBI: A Java-based interface to FPGA hardware,” *Configurable Computing Technology and its uses in High Performance Computing, DSP and Systems Engineering*, Proc. SPIE Photonics East, J. Schewel (Ed.), SPIE - The International Society for Optical Engineering, Bellingham, WA, November 1998.
14. E. Keller, “JRoute: A Run-Time Routing API for FPGA Hardware,” *7th Reconfigurable Architectures Workshop*, Lecture Notes in Computer Science 1800, pp 874-881, Cancun, Mexico, May, 2000.
15. W. Luk, N. Shirazi and P. Cheung, “Compilation tools for run-time reconfigurable designs,” *IEEE Symposium on FPGAs for Custom Computing Machines*, pp 56-65, April, 1997.
16. H. Schmit, “Incremental Reconfiguration for Pipelined Applications,” *IEEE Symposium on FPGAs for Custom Computing Machines*, pp 47-55, April, 1997.
17. J. Burns, A. Donlin, J. Hogg, S. Singh, and M. deWit, “A Dynamic Reconfiguration Run-Time System,” *IEEE Symposium on FPGAs for Custom Computing Machines*, pp 66-75, April, 1997.
18. K. Chapman, “Fast Integer Multipliers fit in FPGAs,” *Electronic Design News*, May 12, 1994.
19. C. Patterson, “High Performance DES Encryption in Virtex FPGAs using JBits,” *IEEE Symposium on FPGAs for Custom Computing Machines*, Napa Valley, CA, April, 2000.
20. S. Scalera and J. Vazquez, “The Design and Implementation of a Context Switching FPGA,” *IEEE Symposium on FPGAs for Custom Computing Machines*, Napa Valley, CA, April, 1998.
21. D. Levi and S. Guccione, “BoardScope: A Debug Tool for Reconfigurable Systems,” *Configurable Computing Technology and its uses in High Performance Computing, DSP and Systems Engineering*, Proc. SPIE Photonics East, J. Schewel (Ed.), SPIE - The International Society for Optical Engineering, Bellingham, WA, November 1998.
22. S. Guccione and D. Levi, “Run-Time Parameterizable Cores,” *Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications*, Lecture Notes in Computer Science 1673, pp 215-222, 1999.
23. J. Ballagh, P. Athanas, and E. Keller, “Java Debug Hardware Models using JBits,” *8th Reconfigurable Architectures Workshop*, San Francisco, CA, Apr. 27, 2001.
24. J. Ballagh, P. Athanas, S. McMillan, “Defining a Run-Time Reconfigurable FPGA I/O Interfacing Methodology, *1st Engineering of Reconfigurable Systems and Algorithms*, Las Vegas, NV June, 2001.
25. Xilinx, Inc. *The Programmable Logic Data Book*, 1999.
26. Xilinx, Inc. World Wide Web page, http://www.xilinx.com, 2001.
27. Joint Photographic Experts Group, World Wide Web page http://www.jpeg.org/JPEG2000.htm, 2000.
28. Information Sciences Institute, SLAAC project, World Wide Web page, http://www.east.isi.edu/projects/SLAAC/, 2001.
29. A. Benkrid, D. Crookes, and K. Benkrid, “Design and Implementation of a Generic 2-D Biorthogonal Discrete Wavelet Transform on an FPGA,” *IEEE Symposium on FPGAs for Custom Computing Machines*, Rohnert Park, CA, April, 2001.