Universal Wavelet Kernel Implementation Using Reconfigurable Hardware

Christophe Desmouliers, Erdal Oruklu and Jafar Saniie
Department of Electrical and Computer Engineering
Illinois Institute of Technology
Chicago, Illinois 60616
erdal@ece.iit.edu

Abstract

Designing a universal embedded hardware architecture for Discrete Wavelet Transform (DWT) is a challenging problem due to the diversity among wavelet kernel filters. In this work, we present three different hardware architectures for implementing multiple wavelet kernels. The first scheme utilizes fixed, parallel hardware for all the required wavelet kernels whereas the second scheme employs a processing element (PE) based datapath that can be configured for multiple wavelet filters during run-time. The third scheme makes use of partial runtime configuration of FPGA units for dynamically programming any desired wavelet filter. We present FPGA synthesis results for simultaneous implementation of six different wavelets for the proposed methods. Performance analysis and comparison of area, timing and power results are presented for the Virtex-II Pro FPGA implementation.

1. Introduction

The Discrete Wavelet Transform (DWT) has been widely used in many multimedia applications including multimedia coding and signal processing. Multimedia standards such as JPEG2000 and MPEG-4 have adopted DWT as its transform coder. Consequently, efficient hardware realization of DWT for embedded devices has been an important research topic. Although numerous architectures have been proposed for DWT, these have been for the most part influenced by popular wavelet kernels used in multimedia standards such as CDF 9/7, limiting their applicability to other kernel implementations.

DWT does not have a pre-fixed kernel as in other transforms like fast Fourier transform (FFT) or discrete cosine transform (DCT). Therefore, the wavelet kernel can be chosen based on the performance of the wavelet filters with respect to the application type. The clear benefit of using DWT then is the capability of fine-tuning the wavelet filters for more adaptive solutions. For example, in non-destructive testing (NDE) applications [1], different wavelet kernels have shown varying results for ultrasonic flaw detection, necessitating the inclusion of multiple wavelet hardware units in the embedded system. Similarly, JPEG2000 standard [2] requires inclusion of at least two wavelet kernels (9/7) and (5/3) in order to provide two levels of algorithm complexity.

Consequently, it is imperative for adaptive applications to support numerous wavelet kernel implementations. Recently, several reconfigurable wavelet architectures have been proposed to address this issue. In [3], a lifting-scheme based architecture with a fixed type processing element is used for the set of seven filters proposed in JPEG2000. However, these kernels are very similar to each other and the paper does not provide any method for simultaneous implementation of kernels. A systematic design method is proposed in [4] to construct lifting based DWT. This method uses the non-uniqueness property of the wavelet lifting factorization to force four specific types of lifting steps, resulting in a systolic array implementation but reconfiguration and support for more than one type of wavelet kernel is not discussed. A reconfigurable architecture based on the same systematic method is described in [5] using folding of the systolic PE array. However, the specific lifting factorization required in these methods is not optimal for all wavelet families. The final lifting step may require decomposition into smaller lifting steps which can rapidly increase the number of PEs required. A multilevel lifting-based wavelet transform architecture is shown in [6] but the hardware is very complex and not scalable, similar to domain-specific reconfigurable array used in [7].

In the following sections, we briefly discuss the lifting-based wavelet transform. Then we present three adaptive wavelet architectures that support implementation of kernels from different wavelet families. Finally, the FPGA synthesis results are discussed and analyzed.

978-0-7695-3474-9/08 $25.00 © 2008 IEEE
DOI 10.1109/ReConFig.2008.32

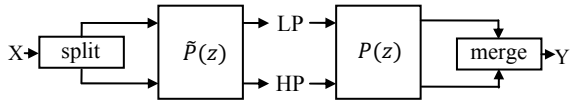


Figure 1. A one-stage filterbank for signal analysis and reconstruction using polyphase matrices

2. Lifting-based wavelet transform

DWT can be implemented directly by convolution. Nevertheless such an implementation can be computationally very complex (depending on the wavelet) which is not desirable for high-speed and low-power applications. A lifting-based scheme [8], which can reduce the computational complexity up to 50% has been proposed for the DWT [9].

The main feature of the lifting-based DWT scheme is to decompose the highpass and lowpass filters into a sequence of upper and lower triangular matrices and convert the filter implementation into banded matrices multiplications.

The DWT implementation using polyphase matrices is shown Figure 1. Let $\tilde{h}(z)$ and $\tilde{g}(z)$ be the lowpass and highpass analysis filters, and let $h(z)$ and $g(z)$ be the lowpass and highpass synthesis filters. The corresponding polyphase matrices are then defined as

$$\tilde{P}(z) = \begin{bmatrix} \tilde{h}_e(z) & \tilde{h}_o(z) \\ \tilde{g}_e(z) & \tilde{g}_o(z) \end{bmatrix} \quad (1)$$

and

$$P(z) = \begin{bmatrix} h_e(z) & g_e(z) \\ h_o(z) & g_o(z) \end{bmatrix} \quad (2)$$

If $(\tilde{h}, \tilde{g})$ is a complementary pair then $\tilde{P}(z)$ can always be decomposed into lifting steps as

$$\tilde{P}(z) = \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \prod_{i=1}^{m} \begin{bmatrix} 1 & \tilde{s}_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \tilde{t}_i(z) & 1 \end{bmatrix} \quad (3)$$

where $\tilde{s}_i(z)$ and $\tilde{t}_i(z)$ are Laurent polynomials, K is the scaling factor and m is determined by the wavelet kernel factorization. For a given wavelet kernel, $\tilde{s}_i(z)$ and $\tilde{t}_i(z)$ filters can have any number of taps making it difficult for a regular VLSI structure implementation.

3. Parallel implementation of multiple wavelet kernels

A straightforward approach for applications that require multiple wavelet kernels is to implement all kernels in parallel. As a case study, six wavelet kernels have been implemented in this work and they correspond to the wavelet transformations given in MATLAB as Daubechies-2 (Db2), Daubechies-3 (Db3), CDF 5/2, CDF 9/7, Spline1.5 and Symlet-4. Each wavelet kernel has been implemented using Matlab Simulink with Xilinx System Generator to generate the *vhdl* files and the required netlist files. The lifting factorization and the implementation of these wavelet kernels without the pipelining stages are shown Figure 2 (see next page).

For hardware realization, a hardware/software codesign method has been used on a Virtex-II pro FPGA device using Embedded Development Kit (EDK) 9.1 software. The system overview is given Figure 3. Microblaze CPU is a 32-bit embedded microprocessor and it is used for controlling and selecting the necessary wavelet datapath units (see Figure 2).

For testing purposes, an input signal of 512 samples (8-bit wordlength) is stored on a Compact Flash disk and loaded into two dual-port memories separately for even and odd samples using Fast Simplex Link (FSL) [10]. Using a software accessible register (a General Purpose Input Output (GPIO) register), the desired DWT kernel can be selected. When the DWT operation has been completed, the results are sent back to the processor via FSL, stored in the external memory and displayed on the screen using a terminal screen connected to the FPGA via a serial link.

The purpose of the *control unit* is to enable access to the memory units and generate the necessary addresses when the DWT is performed. It also generates control signals when the output data are ready to be written into the FIFO of the FSL socket

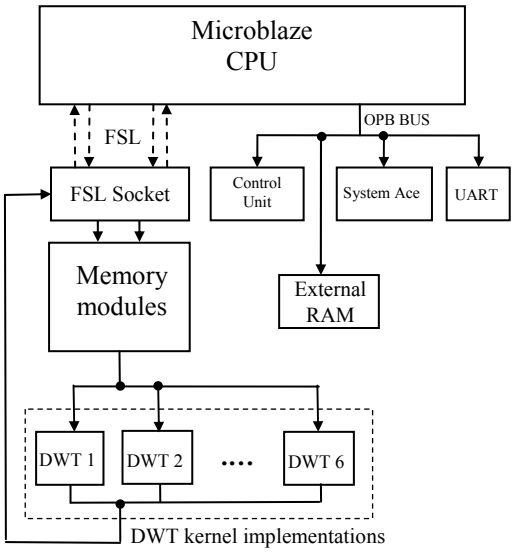
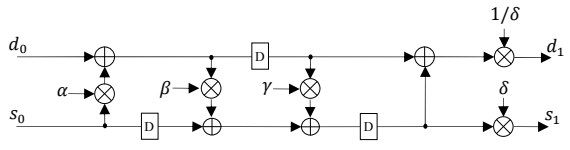


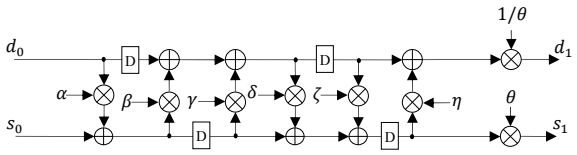
Figure 3. System implemented on Virtex-II Pro



$$\tilde{P}(z) = \begin{bmatrix} \delta & 0 \\ 0 & 1/\delta \end{bmatrix} \begin{bmatrix} 1 & 0 \\ z & 1 \end{bmatrix} \begin{bmatrix} 1 & \beta z^{-1} + \gamma \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \alpha & 1 \end{bmatrix}$$

where $\alpha = -1.7320508,\ \beta = -0.0669873, \gamma = 0.4330127,$
$\delta = 1.93185165$

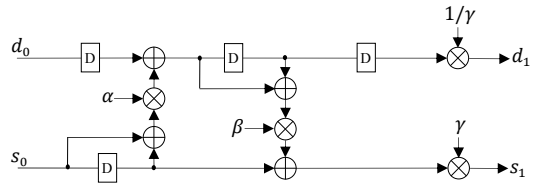
(a). Db2 factorization



$$\tilde{P}(z) = \begin{bmatrix} \theta & 0 \\ 0 & 1/\theta \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \eta & 1 \end{bmatrix} \begin{bmatrix} 1 & \delta + \zeta z \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \delta + \beta z^{-1} & 1 \end{bmatrix} \begin{bmatrix} 1 & \alpha \\ 0 & 1 \end{bmatrix}$$

where $\alpha = -0.4122866,\ \beta = -1.565136, \gamma = 0.3523876,$
$\delta = 0.028459, \zeta = 0.492152, \eta = -0.3896204,$
$\theta = 1.918203$

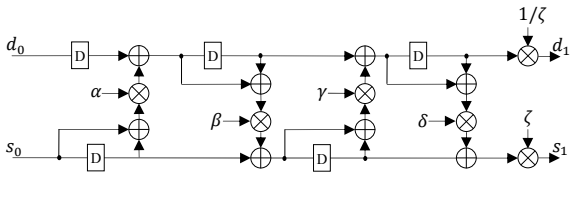
(b). Db3 factorization



$$\tilde{P}(z) = \begin{bmatrix} \gamma & 0 \\ 0 & 1/\gamma \end{bmatrix} \begin{bmatrix} 1 & \beta(1+z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \alpha(1+z^{-1}) & 1 \end{bmatrix}$$

where $\alpha = -0.5,\ \beta = 0.25, \gamma = 1.4142136$

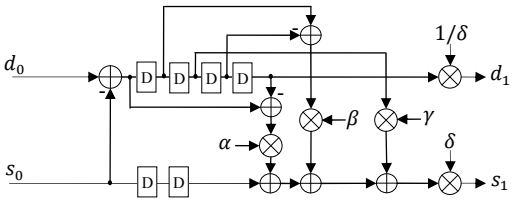
(c). CDF 5/2 factorization



$$\tilde{P}(z) = \begin{bmatrix} \zeta & 0 \\ 0 & 1/\zeta \end{bmatrix} \begin{bmatrix} 1 & \delta(1+z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \gamma(1+z^{-1}) & 1 \end{bmatrix} \begin{bmatrix} 1 & \beta(1+z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \alpha(1+z^{-1}) & 1 \end{bmatrix}$$

where $\alpha = -1.5861343,\ \beta = -0.052980, \gamma = 0.882911,$
$\delta = 0.443507, \zeta = 1.1496044$

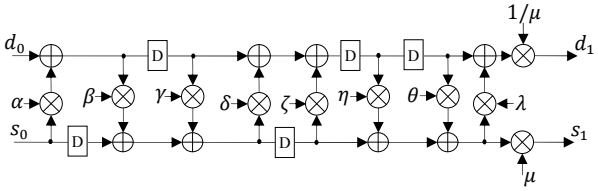
(d). CDF 9/7 factorization



$$\tilde{P}(z) = \begin{bmatrix} \delta & 0 \\ 0 & 1/\delta \end{bmatrix} \begin{bmatrix} 1 & \alpha(z^2 - z^{-2}) + \beta(z - z^{-1}) + \gamma \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$$

where $\alpha = 0.01171875,\ \beta = -0.0859375, \gamma = 0.5,$
$\delta = 1.41421356$

(e). Spline 1.5 factorization



$$\tilde{P}(z) = \begin{bmatrix} \mu & 0 \\ 0 & 1/\mu \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \lambda z^{-1} & 1 \end{bmatrix} \begin{bmatrix} 1 & \theta z + \eta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \zeta z + \delta & 1 \end{bmatrix} \begin{bmatrix} 1 & \gamma + \beta z^{-1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \alpha & 1 \end{bmatrix}$$

where $\alpha = 0.391147,\ \beta - 0.12439, \gamma = -0.339244,$
$\delta = -1.4195148, \zeta = 0.1620314, \eta = 0.43128342,$
$\theta = 0.145983077,\ \lambda = -1.0492552, \mu = 1.5707$

(f). Symlet-4 factorization

Figure 2. Lifting factorization and implementation of wavelet kernels Db2, Db3, CDF5/2, CDF9/7, Spline1.5 and Symlet-4

and when the DWT is completed. Through this control unit, the desired DWT datapath is selected.

The *FSL socket* is used as an interface between the Microblaze processor and the hardware. It uses 4 FSL (2 slave and 2 master) links. The slave links are used to download the input signal into the two memory modules. When a *Load* signal is asserted, if there is any data in the FIFO of the link, they are written into the memory. When a *Start* signal is asserted, the output data of the DWT are written into the FIFO. Finally, using drivers created by EDK software, these data are stored in the external memory and displayed on the screen to verify that results are correct.

4. PE based systolic array implementation

In order to remove the significant hardware redundancy observed in the direct implementation method (discussed in Section 3), a second scheme is proposed. In this approach, wavelet transform architecture can be reconfigured depending on the type of DWT required. This is accomplished using PEs in a

systolic array structure. It can be observed from Figure 2 that a possible PE could be composed of one adder, one multiplier and registers. The proposed PE is given Figure 4. One multiplexer decides which input will be delayed by the delay chain (DC) and two other multiplexers to select which input will go to the multiplier or the adder. The delay chain is shown Figure 5. It is composed of two registers and a multiplexer.

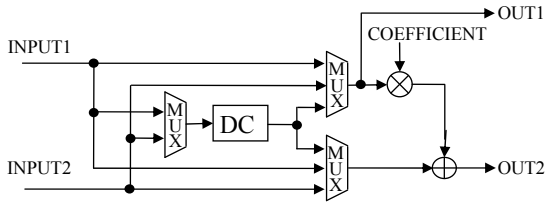


Figure 4. Processing element (PE)

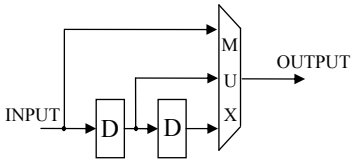


Figure 5. Delay chain

For this case study, the wavelet hardware architecture is required to implement the six DWTs selected. The maximum number of PEs required corresponds to the number of taps in the lifting factorization (except for the scaling factors). For example, *Db2* requires four PEs. Since *Symlet-4* DWT has 8 taps, the maximum number of PEs required will be 8. Eight PEs are connected in series as shown in Figure 6. The length of PE chain (which corresponds to the number of filter taps) is determined by MUX1. The output samples are multiplied by the scaling factor. The delay chains (DC) at the end are configured in order to have d_1 ready one clock cycle after s_1 (this is required for the control unit). This new module replaces the six DWTs shown in Figure 2 and Figure 3.

In order to illustrate the reconfiguration of the PE datapath, we examine *Db2* (see Figure 2(a)) wavelet filter implementation:

The wavelet requires four PEs that are selected by MUX1:

1st PE: coefficient is α, inputs inverted, no registers needed.

2nd PE: coefficient is β, inputs inverted, 1 register needed on adder path.

3rd PE: coefficient is γ, no inversion, 1 register needed on multiplier path.

4th PE: coefficient is 1, inputs inverted, 1 register needed on multiplier path.

Scaling D will be $1/\delta$ and Scaling S will be δ. Finally two registers are needed for the DC of d_1 and 0 for s_1.

This approach will require fewer resources than the first method; nevertheless, eight PEs will be implemented whereas only four are required for the case of *Db2* kernel. In order to reduce the hardware resources and vastly improve the kernel support, dynamic partial self-configuration of FPGA devices can be used.

5. Dynamic partial self-reconfiguration

In this approach, dynamic partial self-reconfiguration is needed in order to reduce the hardware required for the implementation of multiple DWTs. Depending on the desired wavelet kernel, the FPGA will reconfigure itself using the Internal Configuration Access Port (ICAP) [11]. For this case study, the six DWTs will be replaced by a reconfigurable module which will correspond to only one DWT.

In EDK 9.1, the DWT modules are not added to the project but the ports $d_0, s_0, d_1,$ and s_1 are made external. In order to perform dynamic partial self-reconfiguration, the IP core HWICAP [12] provided by Xilinx is added to the system and connected to the OPB Bus. Moreover, since the DWT module requires a clock and clocks can't be connected from the system to the module through bus macros, the DCM modules must be removed from the system and added to the top module. Hence the output signals from the DCM must be made external in the system. Figure 7 shows the top module of the whole system.

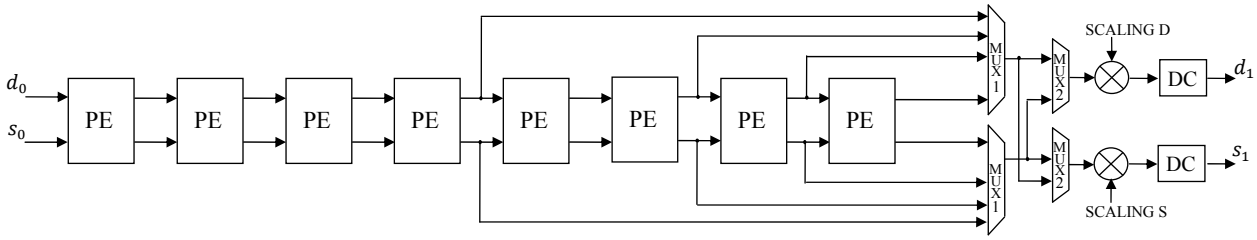


Figure 6. Processing element array for six DWT kernels

In this case study, the reconfigurable module will operate at the frequency of the system sys_clk which is 100 MHz. This clock is also required for the system; in addition we need clk90 and dcm_lock generated by the DCM and used by the controller of the external memory. The *vhdl* code of the top module has been created by instantiating the system in EDK, adding the DCM module, adding the reconfigurable module instantiated as a black box with name *wavelet* and adding the bus macros. The bus macro coming from the reconfigurable part must be disabled during reconfiguration. This is done by adding a GPIO register to the system and making the output external.

After the netlists are created in EDK, the top module is synthesized using ISE 9.1 by importing the *vhdl* file and the *system.xmp* file (generated by EDK). All reconfigurable modules (the six DWTs we have chosen) must have the same top module name (in this case, *wavelet*) and must be synthesized in ISE with the option *Add I/O Buffers* unchecked. After synthesis, PlanAhead 9.2.7 software is used to create the static and partial bitstreams. A reconfigurable area of the FPGA is chosen and the DCM module and bus macro are placed. Figure 8 shows the routing of the system.

Once the partial bitstreams are created, they are loaded on the Compact Flash disk with a system.ace file created from the full bitstream. Hence the board is automatically configured when it is powered up.

In order to send the partial bitstreams to the ICAP interface for reconfiguration, a *C* function provided by Xilinx is used. It opens the bitstream of our choice located on the compact flash, sends it to the storage buffer (512 words depth) of the HWICAP module. When the buffer is full, the data are sent to the ICAP interface. This continues until the whole bitstream has been loaded.

For testing, the user can load an input signal located on the Compact Flash disk into the memory modules. Then, the user can select the DWT kernel from a menu displayed on a terminal screen which is connected to the FPGA board via a serial link. The results are then verified for transform output.

6. Implementation results

Three different wavelet architectures are synthesized on the same FPGA device and compared in terms of area, power, and frequency. Area and timing results are obtained using ISE 9.1 and EDK 9.1 software. Power results are obtained using XPower. Furthermore, the reconfiguration time for the partial configuration case is measured.

Tables I - III displays the FPGA resources such as slices, flip-flops (FF), look-up tables (LUT) and multiplier units (Mult) used in the wavelet transform

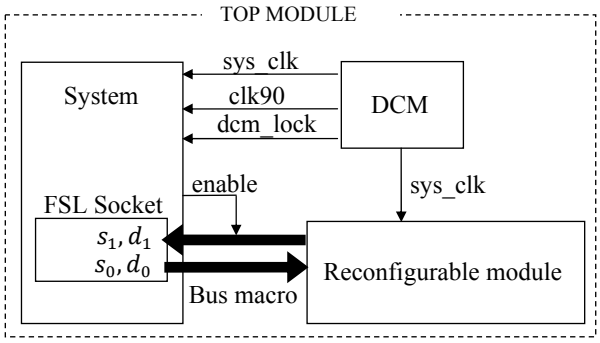


Figure 7. Top module of the dynamic partial self-reconfiguration implementation

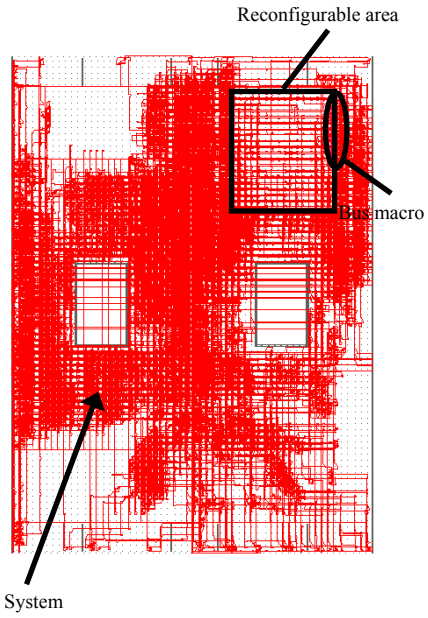


Figure 8. Routing of the top module after PAR in PlanAhead

implementations. It can be seen that the dynamic partial self-reconfiguration scheme requires least hardware logic. Although systolic array implementation results are acceptable, the hardware resources would be significantly higher given a more exhaustive set of wavelets.

The maximum achievable frequency for Scheme 1 and 3 (both methods use same implementations for six DWTs) is 300 MHz which corresponds to the maximum speed of the dedicated multiplier. On the other hand, the systolic array implementation can achieve only 150 MHz due to the extra control logic for reconfiguration. In all configurations, the datapath wordlength is 16-bits. If more precision is required for the DWT operation, partial configuration scheme can also support higher wordlengths without any major architectural change.

Table I. Direct Implementation Scheme

	Slices	FF	4 input LUT	Mult.
Db2	327	577	85	5
Db3	556	1041	136	8
CFD5/2	294	529	68	4
CDF9/7	551	1025	102	6
Spline 1.5	528	993	85	5
Symlet 4	689	1297	170	10
Base System	9000	10843	8304	41

Table II. Systolic implementation scheme

	Slices	FF	4 input LUT	Mult.
PE Array	1609	2817	1408	10
Base System	7118	8032	9020	13

Table III. Partial configuration scheme

	Slices	FF	4 input LUT	Mult.
Db2	327	577	85	5
Db3	556	1041	136	8
CFD5/2	294	529	68	4
CDF9/7	551	1025	102	6
Spline 1.5	528	993	85	5
Symlet 4	689	1297	170	10
Base System	4745	3695	7248	3

Table IV. Power consumption

	Power (mW)
Direct implementation	2437
Systolic implementation	1128
Partial configuration scheme	1393

Table IV lists the power consumption results. The direct implementation scheme requires more than double compared to Schemes 2 and 3.

The reconfiguration time of the board depends on the size of the DWT bitstream. A timer connected to the OPB Bus has been used to measure the reconfiguration time. Since all the partial bitstreams have similar size (about 288KB), the reconfiguration time is almost the same (between 1.3 - 1.5 seconds). If the external memory is used instead of Compact Flash, the reconfiguration time can be reduced significantly.

7. Conclusion

In summary, for systems which require multiple wavelet kernels and with limited power and area, both systolic PE array scheme and dynamic partial self-reconfiguration scheme provide feasible solutions. Only the necessary hardware is implemented on the chip and new modules can be programmed or configured depending on the needs of the application.

The main drawback of the partial configuration scheme is the reconfiguration time which is proportional to the size of the hardware that needs to be loaded. All the partial bitstreams must be created offline. A further study could combine partial self-reconfiguration and PE-based wavelet kernels. In this method, the FPGA device can instantiate the number of PEs required for a particular wavelet kernel and reconfigure itself accordingly; removing the necessity of offline storing of wavelet kernel bitstreams.

References

[1] E. Oruklu and J. Saniie, "Dynamically reconfigurable architecture design for ultrasonic imaging", *IEEE Transactions on Instrumentation and Measurement*, in print, 2008.

[2] JPEG2000 Committee Drafts [Online]. Available: http://www.jpeg.org/public/fcd15444-1.pdf .

[3] K. Andra, C. Chakrabarti and T. Acharya, "VLSI architecture for lifting-based forward and inverse wavelet transform," *IEEE Transactions On Signal Processing*, vol. 50, no. 4, pp. 966-977, April 2002.

[4] C. Huang, P. Tseng and L. Chen, "Efficient VLSI architectures of lifting-based discrete wavelet transform by systematic design method," *IEEE International Symposium on Circuits and Systems, ISCAS*, vol.5, pp. 565-568, 2002.

[5] P. Tseng, C. Huang and L. Chen, "Reconfigurable discrete wavelet transform architecture for advanced multimedia systems", *IEEE Workshop on Signal Processing Systems (SIPS),* pp. 137-141, August 2003.

[6] P.Y. Chen, "VLSI implementation for one-dimensional multilevel lifting-based wavelet transform," *IEEE Transactions on Computers*, vol. 53, no. 4, pp. 386-398, April 2004.

[7] S. Baloch, I. Ahmed, and T. Arslan, "Domain-specific reconfigurable array targeting discrete wavelet transform for system-on-chip applications", *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05),* vol. 4, 2005.

[8] W. Sweldens, "The lifting scheme: A new philosophy in biorthogonal wavelet constructions," *Proceedings of SPIE, Wavelet Applications in Signal and Image Processing III*, vol. 2569, pp. 68–79, 1995.

[9] J. Reichel, "On the arithmetic and bandwidth complexity of the lifting scheme," *2001 International Conference on Image Processing*, vol.3, pp. 198-201, October 2001.

[10] Fast Simplex Link (FSL) bus, *Xilinx application note*, http://www.xilinx.com/support/documentation/ip_documentation/fsl_v20.pdf.

[11] M. French, E. Anderson, D. Kang, "Autonomous system on chip adaptation through partial runtime reconfiguration," *IEEE Symposium on Field-Programmable Custom Computing Machines,* April 2008.

[12] HWICAP, *Xilinx product specification*, http://www.xilinx.com/support/documentation/ip_documentation/opb_hwicap.pdf.