Published in IET Circuits, Devices & Systems
Received on 9th July 2010
Revised on 23rd January 2011
doi: 10.1049/iet-cds.2010.0259

ISSN 1751-858X

Discrete wavelet transform realisation using run-time reconfiguration of field programmable gate array (FPGA)s

C. Desmouliers E. Oruklu J. Saniie

Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, Illinois, USA
E-mail: erdal@ece.iit.edu

Abstract: Designing a universal embedded hardware architecture for discrete wavelet transform is a challenging problem because of the diversity among wavelet kernel filters. In this work, the authors present three different hardware architectures for implementing multiple wavelet kernels. The first scheme utilises 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 run-time configuration of FPGA units for dynamically programming any desired wavelet filter. As a case study, the authors 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 implementations.

1 Introduction

Discrete wavelet transform (DWT) has been widely used in many multimedia applications including video coding and various signal-processing applications. Multimedia standards such as JPEG2000 and MPEG-4 have adopted DWT as its transform coder. Consequently, efficient hardware realisation 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, such as 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 ultrasonic imaging applications [1, 2], different wavelet kernels have shown varying results for target detection, necessitating the inclusion of multiple wavelet hardware units in the embedded system for robust operation. These wavelet kernels included Daubechies (D4, D10), Symmlet (8,10), Coiflet (1,5), Battle-Lemarie (1,5) and they require significantly diverse implementations [2]. Similarly, JPEG2000 standard [3] 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. However, an efficient unified hardware realisation proves to be very difficult to achieve because of disparity in wavelet filter kernels. A multiplierless wavelet design, based on canonical signed digits is proposed in [4] for FPGA implementations but the design is fixed and cannot be changed for other wavelet kernels. In [5], a lifting-scheme-based architecture with a fixed type processing element (PE) 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 memory efficient, pipelined architecture for DWT is presented in [6] but only 5/3 and 9/7 kernels are discussed. A systematic design method is proposed in [7] to construct lifting-based DWT. This method uses the non-uniqueness property of the wavelet lifting factorisation 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 [8] using folding of the systolic PE array. However, the specific lifting factorisation 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 wavelet processing core based on RISC architecture with an instruction set specifically designed to facilitate the implementation of wavelet-based applications is shown in [9] but the wavelet kernel implementation is not universal. In [10], authors describe generic and modular architectures that allow the rapid silicon design of a broad

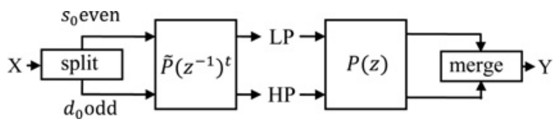


Fig. 1 *One-stage filterbank for signal analysis and reconstruction using polyphase matrices*

range of wavelet systems. This method targets reusable IP cores with fast development cycle but it is not capable of on-chip reconfiguration, limiting its application areas.

In this work, a universal wavelet transform platform based on partial configuration of the FPGA fabric is presented. Dynamic reconfiguration of FPGAs is a flexible technique to achieve spatial mapping of complex algorithms [11, 12]. Recently, it has been successfully applied to image and video-processing applications [13]. Furthermore, a wavelet transform implementation has been proposed in [14], however it only supports two very similar wavelet kernels. In the following sections, we briefly discuss the lifting-based wavelet transform. We the present three adaptive wavelet architectures that support implementation of kernels from different wavelet families. Finally, the FPGA synthesis results are discussed and analysed.

2 Lifting-based wavelet transform

DWT can be implemented directly by convolution. Nevertheless, such an implementation can be computationally very heavy (depending on the wavelet) which is not desirable for high-speed and low-power applications. A lifting-based scheme [15], which can reduce the computational load up to 50%, has been proposed for the DWT [16]. 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.

The DWT implementation using polyphase matrices is shown in Fig. 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, respectively. The corresponding analysis and synthesis polyphase matrices are then defined as

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

and

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

where $\tilde{h}_e(z)$, $\tilde{g}_e(z)$ are even indexed filter coefficients and $\tilde{h}_o(z)$, $\tilde{g}_o(z)$ are odd indexed filter coefficients. If $(\tilde{h}, \tilde{g})$ is a complementary pair then $\tilde{P}(z)$ can always be decomposed into lifting steps as [15]

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

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 factorisation. It is important to note that, 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 very large scale integration (VLSI) structure implementation.

3 Lifting decomposition for hardware realisation

In order to demonstrate the hardware realisation challenges, six wavelet kernels with different complexities have been implemented in this work as a case study. They correspond to the wavelet transformations given in MATLAB as Daubechies-2 (Db2), Daubechies-3 (Db3), CDF 5/3, CDF 9/7, Spline1.5 and Symlet-4. In this work, only the synthesis filter decomposition will be examined; it can be easily extended to the analysis filters.

The lifting decomposition for each wavelet kernel has been obtained from a C program we developed for lifting factorisation. This program is based on Liftpack software package [17], which is capable of extracting all possible lifting decompositions given for a wavelet kernel. Lifting decomposition is not unique [18] and a very large number of factorisations are possible for a single kernel. It is important to note that our lifting factorisation program is designed to find and generate only the decompositions that are useful for hardware implementation: First, it ensures that the decomposition finishes by a scaling matrix, so it has the form given in (3). Second, only the decompositions which have few numbers of taps and small coefficient values for those taps are selected. More taps means more hardware is required which is not desirable. A small coefficient value is desired because of finite precision operations. (16-bit datapath is used with 12 bits for integer part and four bits for fractional part.) In order to prevent overflow at the output of the multipliers, one of the input operands (in this case the coefficient) is kept small but not close to 0 for precision. To illustrate the lifting-selection criteria, an example is given below.

Using the algorithm given in [18], Coiflet-2 wavelet kernel has more than 5000 different decompositions when kernel polyphase matrix is factorised into lifting steps. However, if only the decompositions that are valid (having the form given in (3)) are selected, there are only 28 possibilities remaining. An example of an invalid decomposition is given in (4). There is no scaling matrix with a constant K for this case (see (4)).

Examples of valid decompositions are given in (5) and (6). Although the lifting structures are identical, the coefficient

$$\tilde{P}(z^{-1})^t = \begin{bmatrix} a_{10}z^{-2} + a_{11}z^{-3} + a_{12}z^{-4} & a_{13}z^{-1} + a_{14}z^{-2} + a_{15}z^{-3} + a_{16}z^{-4} \\ a_{17}z^4 & a_{18}z^5 + a_{19}z^4 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ a_8 z + a_9 & 1 \end{bmatrix} \begin{bmatrix} 1 & a_6 z^{-1} + a_7 z^{-2} \\ 0 & 1 \end{bmatrix}$$
$$\times \begin{bmatrix} 1 & 0 \\ a_4 z + a_5 & 1 \end{bmatrix} \begin{bmatrix} 1 & a_2 z^{-1} + a_3 z^{-2} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ a_1 z & 1 \end{bmatrix} \quad (4)$$

values differ greatly.

$$\tilde{P}(z^{-1})^t = \begin{bmatrix} a_{15} & 0 \\ 0 & 1/a_{15} \end{bmatrix} \begin{bmatrix} 1 & a_{12}z^{-4} + a_{13}z^{-5} + a_{14}z^{-6} \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ a_{10}z^4 + a_{11}z^3 & 1 \end{bmatrix} \begin{bmatrix} 1 & a_8z^{-2} + a_9z^{-3} \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ a_6z^2 + a_7z & 1 \end{bmatrix} \begin{bmatrix} 1 & a_4 + a_5z^{-1} \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ a_2z^2 + a_3z & 1 \end{bmatrix} \begin{bmatrix} 1 & a_1z^{-1} \\ 0 & 1 \end{bmatrix} \tag{5}$$

where $a_1 = -2.530$, $a_2 = 0.075$, $a_3 = -1.184$, $a_4 = -0.057$, $a_5 = 0.665$, $a_6 = -1176.195$, $a_7 = -122.462$, $a_8 = 0.001$, $a_9 = -0.0001$, $a_{10} = 76\,377.547$, $a_{11} = 15\,797.077$, $a_{12} = -0.00001$, $a_{13} = 0.000003$, $a_{14} = -0.0000007$, $a_{15} = 244.326$.

$$\tilde{P}(z^{-1})^t = \begin{bmatrix} a_{15} & 0 \\ 0 & 1/a_{15} \end{bmatrix} \begin{bmatrix} 1 & a_{12}z^{-4} + a_{13}z^{-5} + a_{14}z^{-6} \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ a_{10}z^4 + a_{11}z^3 & 1 \end{bmatrix} \begin{bmatrix} 1 & a_8z^{-2} + a_9z^{-3} \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ a_6z^3 + a_7z^2 & 1 \end{bmatrix} \begin{bmatrix} 1 & a_4z^{-1} + a_5z^{-2} \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ a_2z^2 + a_3z & 1 \end{bmatrix} \begin{bmatrix} 1 & a_1z^{-1} \\ 0 & 1 \end{bmatrix} \tag{6}$$

where $a_1 = -2.530$, $a_2 = 0.075$, $a_3 = 0.3418$, $a_4 = -0.655$, $a_5 = -7.489$, $a_6 = 0.937$, $a_7 = 0.2311$, $a_8 = -0.005$, $a_9 = -1.044$, $a_{10} = 6.338$, $a_{11} = 1.311$, $a_{12} = -0.157$, $a_{13} = 0.032$, $a_{14} = -0.0088$, $a_{15} = 2.225$.

Among the remaining 28, the desired decomposition shouldhave small number of taps and small finite precision range for coefficients. There are only three such solutions and (6) is one of them. Lifting scheme in (5) has coefficients which are too large (such as a_{10} and a_{11}) for finite precision operations. After running this algorithm for each wavelet kernel, we obtain the lifting decompositions given in Fig. 2. Those decompositions exactly match the ones given by MATLAB.

4 Parallel wavelet kernels implementation

A straightforward approach for applications that require multiple wavelet kernels is to implement all kernels in parallel. This requires a priori knowledge about all the wavelet kernels to be used in the application and also results in a highly redundant system. For performance comparison, six different wavelet kernels are synthesised on a Virtex-II pro FPGA. Each wavelet kernel is implemented using MATLAB Simulink with Xilinx system generator library to generate the vhdl files and the required netlist files. The lifting factorisation and the implementation of these wavelet kernels without the pipelining stages are shown Fig. 2. Here, d_1 and s_1 are the highpass and lowpass filter outputs, respectively. In this work, we have implemented only one level decomposition but it can be easily extended to multi-level decomposition by adding others stages or by reusing the same hardware unit.

For hardware realisation, a hardware/software codesign method has been used on a Virtex-II pro FPGA device using Embedded Development Kit (EDK) 9.1 software from Xilinx. The system overview is given in Fig. 3. The microprocessor is Microblaze (32-bit embedded microprocessor) and it is used for controlling and selecting the necessary wavelet datapath units (see Fig. 2). Microblaze CPU also handles the input and output transfers. For testing purposes, an input signal of 512 samples (each sample is 8-bits) 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) [19]. 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.

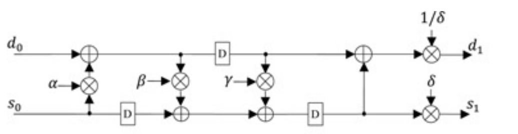
The purpose of the control unit is to enable access to the memory units and generate the necessary addresses when DWT is performed. It also generates control signals when the output data are ready to be written into the first in first out (FIFO) of the FSL socket and when the DWT is completed. The FSL socket is used as an interface between the Microblaze processor and the hardware. It uses four FSL (two slaves and two masters) 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.

5 PE based systolic array implementation

In order to remove the significant hardware redundancy observed in the parallel implementation, a second scheme is proposed. In this approach, a 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 Fig. 2 that a possible PE could be composed of one adder, one multiplier and several registers. The proposed PE is in given Fig. 4. Whereas one multiplexer decides which input will be delayed by the delay chain two other multiplexers select the input that will go to the multiplier or the adder. The delay chain is shown in Fig. 5. It is composed of two registers and a multiplexer.

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 factorisation (except for the scaling factors). For example, Db2 requires four PEs. Since Symlet-4 DWT has eight taps, the maximum number of PEs required will be eight. Eight PEs are connected in series as shown in Fig. 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 at the end are configured in order to have d_1 ready at one clock cycle after s_1 (this is required so that the same control unit can be used for each wavelet kernel). This new module replaces the six DWTs shown in Figs. 2 and 3. GPIO registers have been added to configure the architecture (control signals and coefficient values must be initialised).

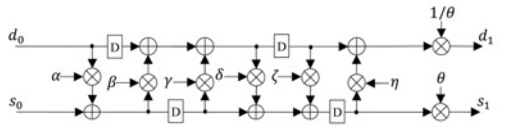
In order to illustrate the reconfiguration of the PE datapath, we examine Db2 (see Fig. 2*a*) wavelet filter implementation.



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

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

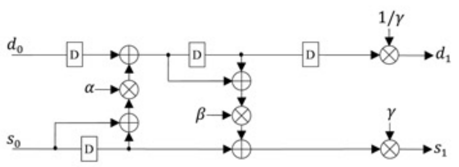
a



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

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

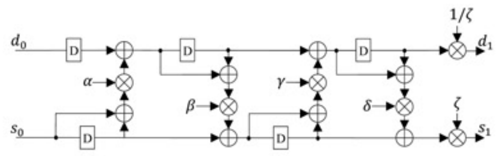
b



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

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

c

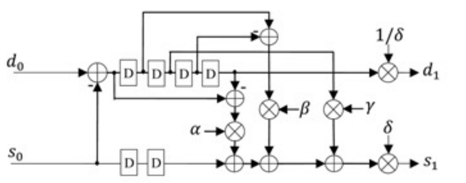


$$\widetilde{P}(z^{-1})^t = \begin{bmatrix}\zeta & 0\\ 0 & 1/\zeta\end{bmatrix}\begin{bmatrix}1 & \delta(1+z^{-1})\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ \gamma(1+z) & 1\end{bmatrix}\begin{bmatrix}1 & \beta(1+z^{-1})\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ \alpha(1+z) & 1\end{bmatrix}$$

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

$\zeta = 1.1496044$

d



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

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

e

Fig. 2 *Lifting factorisation and implementation of wavelet kernels Db2, Db3, CDF 5/3, CDF 9/7, Spline 1.5 and Symlet-4*

a Db2 factorisation
b Db3 factorisation
c CDF 5/3 factorisation
d CDF 9/7 factorisation
e Spline 1.5 factorisation
f Symlet-4 factorisation

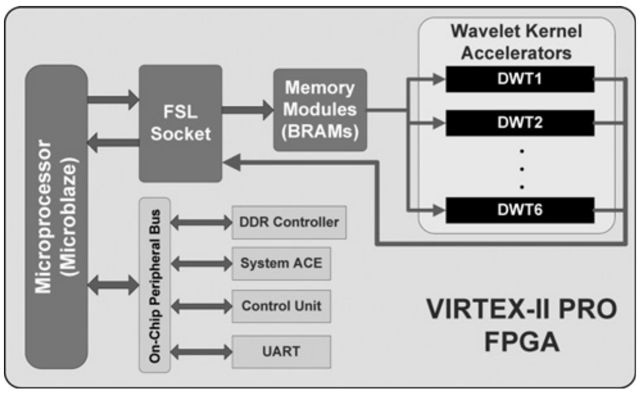


Fig. 3 *System implemented on Virtex-II Pro*

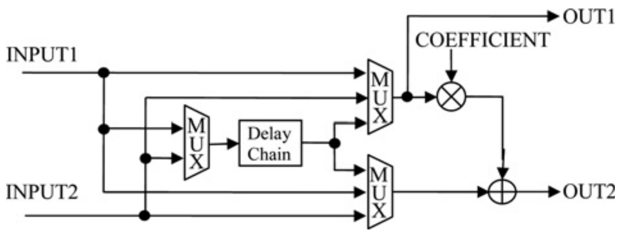


Fig. 4 *PE architecture*

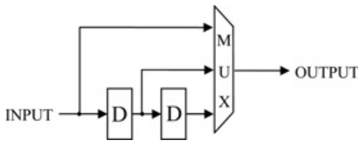


Fig. 5 *Delay chain architecture*

Suppose that d_0 is connected to input1 and s_0 to input2 of the first PE:

- The wavelet kernel requires four PEs that are selected by MUX1.

- *1st PE*: coefficient is α, input1 goes to the adder and input2 goes to the multiplier, no registers are needed.
- *2nd PE*: coefficient is β, input1 goes to the multiplier and input2 goes to the adder, 1 register is needed on adder path.
- *3rd PE*: coefficient is γ, input1 goes to the multiplier and input2 goes to the adder, 1 register is needed on multiplier path.
- *4th PE*: coefficient is 1, input1 goes to the adder and input2 goes to the multiplier, 1 register is needed on multiplier path.
- *Scaling D* will be $1/\delta$ and Scaling S will be δ.

This approach requires fewer resources than the first method; nevertheless, eight PEs will be utilised 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 which is presented in the next section.

6 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) [20]. For this case study, the six DWTs will be replaced by a reconfigurable module which will correspond to only one DWT at a time. The steps required to achieve dynamic partial self-reconfiguration is described below (see Fig. 7):

- *Step 1. Create the processor system*: EDK 9.1 has been used to create the processor system that will correspond to the static module. This module is basically the same as the two previous methods except that the DWT modules are removed but the ports d_0, s_0, d_1 and s_1 (see Fig. 2) are made external. In order to perform dynamic partial self-reconfiguration, the IP core HWICAP [21] provided by Xilinx is added to the system and connected to the on-chip peripheral bus (OPB) bus. Furthermore, a software component that will be executed by the microprocessor is

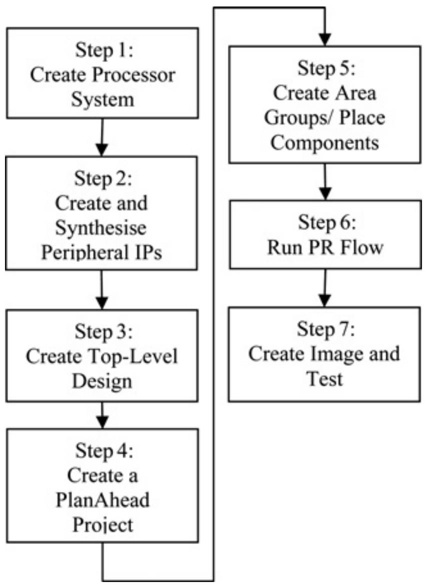

Step 1: Create Processor System
Step 2: Create and Synthesise Peripheral IPs
Step 3: Create Top-Level Design
Step 4: Create a PlanAhead Project
Step 5: Create Area Groups/ Place Components
Step 6: Run PR Flow
Step 7: Create Image and Test

Fig. 7 *Steps required for dynamic partial self-reconfiguration*

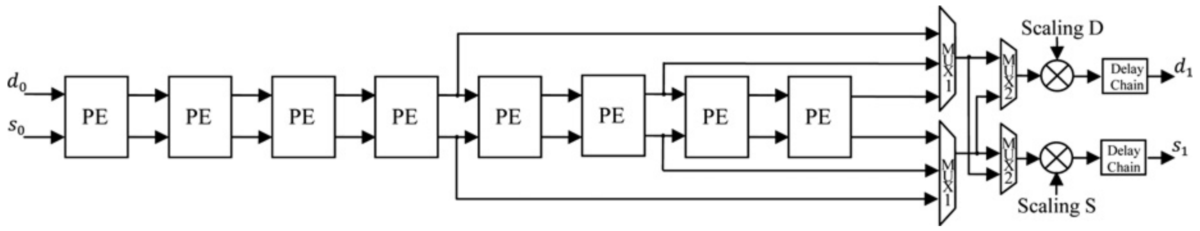


Fig. 6 *PE array for six DWT kernels*

needed. With this software, the user can select which DWT to implement in real time. The partial reconfiguration is initiated by the software using a function provided by Xilinx. Finally, the static module is completed and can be synthesised in order to obtain the required file for the synthesise of the top module.

- *Step 2. Create and synthesise peripheral IPs*: The vhdl and netlist files for each wavelet kernel were already created in the first approach (i.e. parallel implementation of kernels), hence they can be used with minor modifications. First, it is important that the top module of each wavelet kernel has the same name (in our case it is wavelet) because they will be instantiated as a black box in the top module. Each wavelet kernel is synthesised using Integrated Software Environment (ISE) 9.1 software from Xilinx with the option Add I/O Buffers unchecked. This is important because this module will be instantiated within the top module and no buffers are required since its input and output ports will be internal. After synthesis, the Xilinx NGC (netlist with constraint) netlist files are obtained for each wavelet kernel.
- *Step 3. Create top-level design*: The top module (which establishes the interface between static and dynamic parts) must be manually created. The communication between the static module and the dynamic module is done by using Bus Macros. Moreover, the dynamic module requires a clock and clock signals cannot be transferred using Bus Macro. Therefore the digital clock management (DCM) module must be removed from the static part and instantiated on the top-level. The reconfigurable module will operate at the frequency of the static module sys_clk, which is 100 MHz.

Bus Macros are unidirectional and the bus going from the dynamic module to the static module must be disabled during reconfiguration. This is done by using a GPIO register. The reconfigurable module is instantiated as a black box with the name wavelet. Fig. 8 shows the top module of the entire system. Finally, the top-level design is synthesised in ISE9.1 using the vhdl file of the top module that was created and the system.xmp file of the static module. After synthesis, the NGC file of the top-level design is obtained.

- *Step 4. Create a PlanAhead project*: PlanAhead 9.2.7 software from Xilinx has been used to create the implementation files used to reconfigure the FPGA. Within the project file, the NGC file of the top-level design, the NGC file of the wavelet kernel and the NGC file of the Bus Macros given by Xilinx are selected. Finally, the constraint file that has been created in EDK is added and set as partial reconfiguration (PR) project.

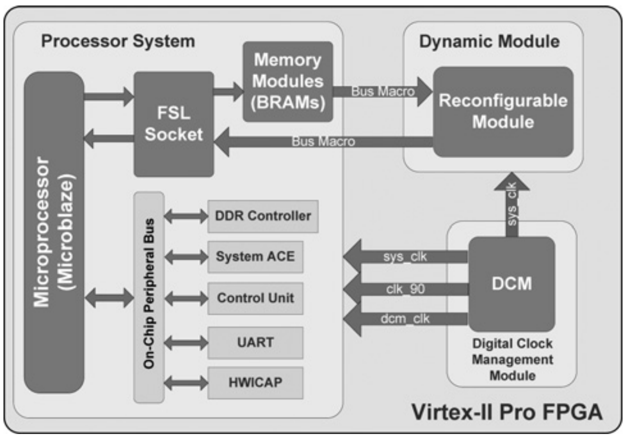


Fig. 8 *Top module of the dynamic partial self-reconfiguration implementation*

- *Step 5. Create area groups/place components*: First, a reconfigurable area is specified as to where the wavelet kernel will be implemented. This area must be large enough so that the wavelet kernel that uses the most resources, fits. In this case, the symlet-4 wavelet kernel would be the largest. Each Bus Macro must then be placed on the edge of the reconfigurable area for data transfers. Finally the DCM module must be placed.
- *Step 6. Run PR flow*: Before PR Implementation flow is run, the user has to set the path to the system_stub.bmm file in order to generate the system_stub_bd.bmm file after the implementation. The system_stub.bmm file describes the block RAM (BRAM) composition (logical) where the program will be stored when the FPGA is configured. The system_stub_bd.bmm file describes actual BRAMs used in the implementation. Routing of the top module after placement and routing (PAR) is shown Fig. 9. Finally, the full bitstream that will be used to initially configure the FPGA is generated. A default wavelet kernel is selected for the full bitstream. Moreover, partial bitstream files for each wavelet kernel are created.
- *Step 7. Create image and test*: For testing purposes, we have created a system.ace file using the full bitstream file and the ELF file (obtained from EDK) containing the information of the C program. This file and all the partial bit files are stored on the compact flash disk. Hence, the board is automatically configured using the system.ace file when it is powered up. A menu is then displayed in the HyperTerminal using the serial link. From this menu, the user can load the input data in the BRAMs, select the wavelet kernel to implement in real time and finally display the results. Of course, a control unit can be designed depending on the application that will specify the DWT kernel to implement in real time. 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 disk, and 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.

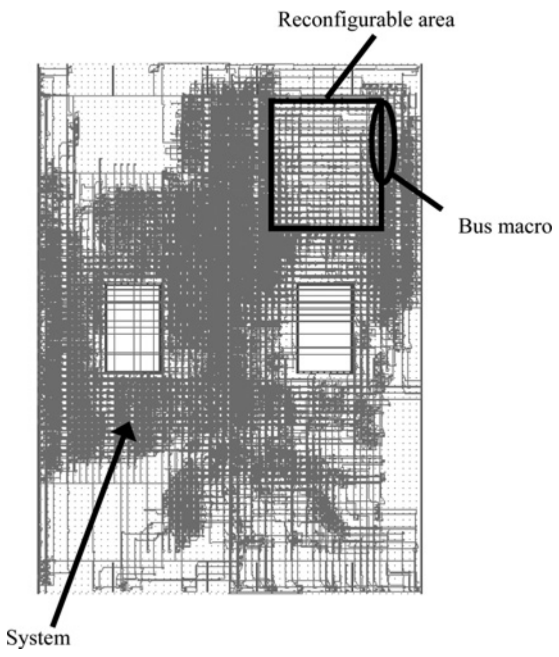


Fig. 9 *Routing of the top module after PAR in PlanAhead*

7 Implementation results

Three different wavelet architectures are synthesised 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, respectively. Power results are obtained using XPower [22]. Furthermore, the reconfiguration time for the partial configuration case is measured. Tables 1–3 display the FPGA resources such as slices, flip-flops (FF), look-up tables (LUT) and multiplier units (Mult.) used in the wavelet transform implementations. In the three cases, the base system includes all the hardware required except the wavelet kernels. For Table 3, the total represents the base system plus the biggest kernel which is Symlet-4. 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 Schemes 1 and 3 (both methods use same implementations for six DWTs) is 300 MHz that corresponds to the maximum speed of the dedicated multiplier. On the other hand, the systolic array implementation can achieve only 150 MHz because of the extra control logic used for reconfiguration.

Table 4 lists the power consumption results. The parallel implementation scheme requires more than double the power compared with Schemes 2 and 3. Partial configuration scheme uses more power than systolic implementation because of extra hardware added for on the fly reconfiguration.

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 288 KB), the reconfiguration time is almost the same for each configuration. The first option is to load the bitstream directly from the compact flash disk during run-time reconfiguration. This is the worst case because of the access time of the compact flash disk. A better option is to load all the bitstream in the external memory just after the board has been configured with the full bitstream. Hence, during run-time reconfiguration, the bitstream can be loaded from the memory reducing the reconfiguration time. Table 5 lists the reconfiguration time for different cases:

Table 1 Parallel implementation scheme

	Slices	FF	4 input LUT	Mult.
Db2	327	577	85	5
Db3	556	1041	136	8
CFD 5/3	294	529	68	4
CDF 9/7	551	1025	102	6
Spline 1.5	528	993	85	5
Symlet-4	689	1297	170	10
base system	4512	3248	3392	3
total	7457	8710	4038	41

Table 2 Systolic implementation scheme

	Slices	FF	4 input LUT	Mult.
PE array	1609	2817	1408	10
base system	5363	4523	3519	3
total	6972	7340	4927	13

Table 3 Partial configuration scheme

	Slices	FF	4 input LUT	Mult.
Db2	327	577	85	5
Db3	556	1041	136	8
CFD 5/3	294	529	68	4
CDF 9/7	551	1025	102	6
Spline 1.5	528	993	85	5
Symlet-4	689	1297	170	10
base system	4623	3262	3574	3
total	5312	4559	3744	13

Table 4 Power consumption

	Power (mW)
parallel implementation	2437
systolic implementation	1128
partial configuration scheme	1393

Table 5 Reconfiguration time

	Reconfiguration time (ms)
Case 1	1458.8
Case 2	674.9
Case 3	346.9
Case 4	146

Case 1. Load bitstream from compact flash disk and C program executed from external memory.
Case 2. Load bitstream from compact flash disk and C program executed from BRAMs.
Case 3. Load bitstream from external memory and C program executed from external memory.
Case 4. Load bitstream from external memory and C program executed from BRAMs.

A key benefit of the partial reconfiguration scheme for wavelets is its adaptability where the specific kernel implementation is not implied. Although direct-mapped implementation of the lifting scheme (similar to [23]) was used in the case study; other techniques can be used for improved performance (see [24] for state-of-the-art VLSI lifting-wavelet architectures). Recent techniques such as folding [25], flipping [26], recursive [27] and dual-scan [28] architectures can be easily adopted. It is important to note that the proposed partial reconfiguration scheme is a template-based approach where multiple kernel templates are stored in memory. When a specific kernel use is requested, FPGA hardware is partially reconfigured 'on-demand'. Each kernel template can be optimised for area [25], speed [26] or even multi-resolution analysis [27] depending on the application needs.

Furthermore, the main objective of template-based, on-demand wavelet architectures is optimising simultaneous implementation of multiple wavelets. Hence, the performance (in terms of hardware and throughput) advantage will scale up as the number of different wavelets employed within the application increases.

8 Conclusion

In summary, for systems that 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.

Currently, 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.

9 References

1 Oruklu, E., Saniie, J.: 'Dynamically reconfigurable architecture design for ultrasonic imaging', *IEEE Trans. Instrum. Meas.*, 2009, **58**, (8), pp. 2856–2866

2 Oruklu, E., Saniie, J.: 'Ultrasonic flaw detection using discrete wavelet transform for NDE applications'. IEEE Ultrasonics Symp., 2004, vol. 2, pp. 1054–1057

3 JPEG2000 Committee Drafts. Available at http://www.jpeg.org/public/fcd15444-1.pdf

4 Kotteri, K.A., Bell, A.E., Carletta, J.E.: 'Design of multiplierless, high-performance, wavelet filter banks with image compression applications', *IEEE Trans. Circuits Syst. – I: Regular Papers*, 2004, **51**, (3), pp. 483–494

5 Andra, K., Chakrabarti, C., Acharya, T.: 'VLSI architecture for lifting-based forward and inverse wavelet transform', *IEEE Trans. Signal Process.*, 2002, **50**, (4), pp. 966–977

6 Wu, B., Lin, C.: 'A high-performance and memory-efficient pipeline architecture for the 5/3 and 9/7 discrete wavelet transform of JPEG2000 codec', *IEEE Trans. Circuits Syst. Video Technol.*, 2005, **15**, (12), pp. 1615–1628

7 Huang, C., Tseng, P., Chen, L.: 'Efficient VLSI architectures of lifting-based discrete wavelet transform by systematic design method'. Proc. IEEE Int. Symp. on Circuits Systems (ISCAS), 2002, Vol. 5, pp. 565–568

8 Tseng, P., Huang, C., Chen, L.: 'Reconfigurable discrete wavelet transform architecture for advanced multimedia systems'. Proc. IEEE Workshop on Signal Process Systems (SIPS), 2003, pp. 137–141

9 Lee, S., Lim, S.: 'VLSI design of a wavelet processing core', *IEEE Trans. Circuits Syst. Video Technol.*, 2006, **16**, (11), pp. 1350–1361

10 Masud, S., McCanny, J.V.: 'Reusable silicon IP cores for discrete wavelet transform applications', *IEEE Trans. Circuits Syst. – I: Regular Papers*, 2004, **51**, (6), pp. 1114–1124

11 Blodget, B., Bobda, C., Huebner, M., Niyonkuru, A.: 'Partial and dynamically reconfiguration of Xilinx Virtex-II FPGAs', in 'Field programmable logic and applications' (Springer, Berlin/Heidelberg, 2004)

12 Raaijmakers, S., Wong, S.: 'Run-time partial re-configuration for removal, placement and routing on the Virtex-II Pro'. Proc. Int. Conf. on Field Programmable Logic and Applications, (FPL 2007), 2007, pp. 679–683

13 Craven, S., Athanas, P.: 'Dynamic hardware development', *Int. J. Reconfigurable Comput.*, 2008, article id 901328

14 Zhang, X., Rabah, H., Weber, S.: 'Cluster-based hybrid reconfigurable architecture for auto-adaptive SoC'. Proc. 14th IEEE Int. Conf. on Electronics, Circuits and Systems, ICECS, 2007, pp. 979–982

15 Sweldens, W.: 'The lifting scheme: A new philosophy in biorthogonal wavelet constructions'. Proc. SPIE, Wavelet Applications in Signal and Image Processing III, 1995, vol. 2569, pp. 68–79

16 Reichel, J.: 'On the arithmetic and bandwidth complexity of the lifting scheme'. Proc. Int. Conf. on Image Processing, 2001, vol. 3, pp. 198–201

17 Fernandez, G., Periaswamy, P., Sweldens, W.: 'LIFTPACK: a software package for wavelet transforms using lifting'. Proc. SPIE, Wavelet Applications Signal Image Processing IV, 1996, vol. 2825, pp. 396–408

18 Daubechies, I., Swelden, W.: 'Factoring wavelet transforms into lifting steps'. Technical report, Bell Laboratories, 1996

19 FSL, (Fast Simplex Link) bus: Xilinx application note, available at http://www.xilinx.com/support/documentation/ip_documentation/fs_v20.pdf

20 French, M., Anderson, E., Kang, D.: 'Autonomous system on chip adaptation through partial runtime reconfiguration'. Proc. IEEE Symp. on Field-Programmable Custom Computing Machines, 2008, pp. 77–86

21 HWICAP: Xilinx product specification, available at: http://www.xilinx.com/support/documentation/ip_documentation/opb_hwicp.pdf

22 Xpower: Xilinx product, available at: http://www.xilinx.com/products/design_tools/logic_design/verification/xpower.htm

23 Liu, C.C., Shiau, Y.H., Jou, J.M.: 'Design and implementation of a progressive image coding chip based on the lifted wavelet transform'. Proc. 11th VLSI Design/CAD Symp., 2000, pp. 49–52

24 Acharya, T., Chakrabarti, C.: 'A survey on lifting-based discrete wavelet transform architectures', *J. VLSI Signal Process.*, 2006, **42**, pp. 321–339

25 Lian, C.J., Chen, K.F., Chen, H.H., Chen, L.G.: 'Lifting based discrete wavelet transform architecture for JPEG2000'. IEEE Int. Symp. on Circuits and Systems, 2001, pp. 445–448

26 Huang, C.T., Tseng, P.C., Chen, L.G.: 'Flipping structure: an efficient VLSI architecture for lifting-based discrete wavelet transform', *IEEE Trans. Signal Process.*, 2004, **52**, (4), pp. 1080–1089

27 Liao, H., Mandal, M.K., Cockburn, B.F.: 'Novel architectures for lifting-based discrete wavelet transform', *Electron. Lett.*, 2002, **38**, (18), pp. 1010–1012

28 Liao, H., Mandal, M.K., Cockburn, B.F.: 'Efficient architectures for 1-D and 2-Dlifting-based wavelet transform', *IEEE Trans. Signal Process.*, 2004, **52**, (5), pp. 1315–1326