FACTA UNIVERSITATIS (NIŠ)
SER.: ELEC. ENERG. vol. 20, no. 3, December 2007, 437-459

Digital Signal Processing Designing for FPGA Architectures

Mariusz Rawski, Bogdan J. Falkowski, and Tadeusz Łuba

Abstract: This paper presents the discussion on efficiency of different implementation methodologies of DSP algorithms targeted for modern FPGA architectures. Modern programmable structures are equipped with specialized DSP embedded blocks that allow implementing digital signal processing algorithms with use of the methodology known from digital signal processors. On the first place, however, programmable architectures give the designer the possibility to increase efficiency of designed system by exploitation of parallelism of implemented algorithms. Moreover, it is possible to apply special techniques, such as distributed arithmetic (DA) that will boost the performance of designed processing systems. Additionally, application of the functional decomposition based methods, known to be best suited for FPGA structures, allows utilizing possibilities of programmable technology in very high degree. The paper presents results of comparison of different design approaches in this area.

Keywords: Digital signal processing, DSP algorithm, FPGA architecture, DSP embedded blocks, distributed arithmetic.

1 Introduction

Digital Signal Processing (DSP), thanks to explosive development of wired and wireless networks and multimedia, represents one of the most fascinating areas in electronics. The applications of DSP continue to expand, driven by trends such as the increased use of video and still images and the demand for increasingly

Manuscript received August 14, 2007.

M. Rawski is with Warsaw University of Technology, Institute of Telecommunications Nowowiejska 15/19, 00-665 Warsaw, Poland (e-mail: rawski@tele.pw.edu.pl). B. Falkowski is with Nanking Technological University, School of Electrical and Electronic Engineering, 50 Nanying Avenue, Singapore 639798 (e-mail: efalkowski@ntu.edu.sg). T. Łuba is with Warsaw University of Technology, Institute of Telecommunications, Nowowiejska 15/19, 00-665 Warsaw, Poland (e-mail: luba@tele.pw.edu.pl).

reconfigurable systems such as Software Defined Radio (SDR). Many of these applications combine the need for significant DSP processing efficiency with cost sensitivity, creating demand for high-performance, low-cost DSP solutions. Traditionally, digital signal processing algorithms are being implemented using general-purpose programmable DSP chips. Alternatively, for high-performance applications, special-purpose fixed function DSP chipsets and application-specific integrated circuits (ASICs) are used. Typical DSP devices are based on the concept of RISC processors with an architecture that consists of fast array multipliers. In spite of using pipeline architecture, the speed of such implementation is limited by the speed of array multiplier.

Multiplications, followed by additions, subtractions or accumulations are the basis of most DSP applications. The number of multipliers embedded in DSP processor is generally in the range of one to four. The microprocessor will sequence data to pass it through the multipliers and other functions, storing intermediate results in memories or accumulators. Performance is increased primarily by increasing the clock speed used for multiplication. Typical clock speeds are between tens of MHz to 1GHz. Performance, as measured by millions of Multiply And Accumulate (MAC) operations per second, typically ranges from 10 to 4000.

The technological advancements in Field Programmable Gate Arrays (FPGAs) in the past decade have opened new paths for DSP design engineers. FPGAs, with their newly acquired digital signal processing capabilities, are now expanding their roles to help offload computationally intensive digital signal processing functions from the processor.

FPGAs are an array of programmable logic cells interconnected by a matrix of programmable connections. Each cell can implement a simple logic function defined by a designer's CAD tool. Typical programmable circuit has a large number (64 to over 300,000) of such cells, that can be used to form complex digital circuits. The ability to manipulate the logic at the gate level means that designer can construct a custom processor to efficiently implement the desired function.

FPGAs offer performance target not achievable by DSP processors. However, to achieve the high-performance, FPGA-based designs have come at a cost. Efficient utilization of possibilities provided by modern programmable devices requires knowledge of hardware specific design methods. Designing DSP system targeted for FPGA devices is very different than designing it for DSP processors. Most algorithms being in use were developed for software implementation. Such algorithms can be difficult to translate into hardware. Thus the efficiency of FPGA-based DSP is heavily dependent on experience of the designer and his ability to tailor the algorithm to efficient hardware implementation. Moreover CAD tools for FPGA based DSP design are immature.

FPGA manufacturers have for years now been extending their chips' ability to

implement digital signal processing efficiently, for example by introducing low-latency carry-chain-routing lines that speed-up addition and subtraction operations spanning multiple logic blocks. Such mechanism is relatively efficient when implementing addition and subtraction operations. However, it is not optimal in cost, performance, and power for multiplication and division functions. As a result, Altera (with Stratix), QuickLogic (with QuickDSP, now renamed Eclipse Plus) and Xilinx (with Virtex-II and Virtex-II Pro) embedded in their chips dedicated multiplier function blocks. Altera moved even further along the integration path, providing fully functional MAC blocks called the DSP blocks. This allows design methodologies known from DSP processors to be used.

However DSP-oriented FPGAs provide the ability to implement many functions in parallel on one chip. General-purpose routing, logic and memory resources are used to interconnect the functions, perform additional functions, sequence and, as necessary, store data. This provides possibility to increase the performance of digital system by exploitation of parallelism of implemented algorithms. Moreover, this technology allows also application of special techniques such as distributed arithmetic (DA) [1, 2]. DA technique is extensively used in computing sum of product with constant coefficients. In such a case partial product term becomes a multiplication with constant (i.e. scaling). DA approach significantly increases the performance of implemented filter, by removing general purpose multipliers and introducing combinational blocks that implement the scaling. These blocks have to be efficiently mapped onto FPGA's logic cells. This can be done with the use of advanced synthesis methods such as functional decomposition [3–5].

In the case of applications targeting FPGA structures based on lookup tables (LUTs), the influence of advanced logic synthesis procedures on the quality of hardware implementation of signal and information processing systems is especially important. Direct cause of such a situation is the imperfection of technology mapping methods that are widely used at present, such as minimization and factorization of Boolean function, which are traditionally adapted to be used for structures based on standard cells. These methods transform Boolean formulas from sum-of-products form into multilevel, highly factorized form that is then mapped into LUT cells. This process is at variance with the nature of LUT cell, which from the logic synthesis' point of view is able to implement any logic function of limited input variables. For this reason, for the case of implementation targeting FPGA structure, decomposition is a much more efficient method. Decomposition allows synthesizing the Boolean function into multilevel structure that is built of components, each of which is in the form of LUT logic block specified by truth tables. Efficiency of functional decomposition has been proved in many theoretical papers [6–10]. However, there are relatively few papers where functional decomposition procedures were compared with analogous synthesis methods used in

commercial design tools. The reason behind such a situation is the lack of appropriate interface software that would allow transforming description of project structure obtained outside commercial design system into description compatible with its rules. Moreover, the computation complexity of functional decomposition procedures makes it difficult to construct efficient automatic synthesis procedures. These difficulties - at least partially - have been eliminated in so called balanced decomposition [11, 12].

In this paper, FPGA based DSP implementation methodologies are discussed. As the example Discrete Wavelet Transform and Discrete Fourier Transform are used.

The wavelet transform has gained much attention in recent years. It is widely used in signal and image processing [13–16]. Discrete wavelet transform (DWT) is one of the useful and efficient signal and image decomposition methods with many interesting properties. Similar to the Fourier transform, this transformation can provide information about frequency contents of signals. However, unlike Fourier transform, this approach is more natural and fruitful when applied to non-stationary signals, like speech and images. The flexibility offered by discrete wavelet transform allows researchers to develop and find the right wavelet filters for their particular application. For example, for the compression of fingerprints, a particular set of bio-orthogonal filters, Daubechies bio-orthogonal spline wavelet filters, is found to be very effective [17]. The computational complexity of the discrete wavelet transform is very high. Hence, efficient hardware implementation is required to achieve very good real-time performance. Application of the DWT requires convolution of the signal with the wavelet and scaling functions. Efficient hardware implementation of convolution is performed as a finite impulse response (FIR) filter. Two filters are used to evaluate a DWT: a high-pass and a low-pass filter, with the filter coefficients derived from the wavelet basis function.

Fourier transform is another most recognized DSP functions. It is deployed in a wide range of communications, radar, and signal intelligence applications. While this transform can be implemented using MAC operation, one of the most efficient methods of performing this transformation is Fast Fourier Transform (FFT) [18]. Simplest and most common form of FFT is the radix-2 ”butterfly” algorithm. Each butterfly consists of multipliers and adders that accept two input points and compute two output points based on suitably chosen coefficients from a sine table.

2 Digital Filters

Digital filters are typically used to modify attributes of signal in the time or frequency domain trough the process called linear convolution [1]. This process is

formally described by following formula

$$y[n] = x[n] * y[n] = \sum_k x[k] f[n-k] = \sum_k x[k] c[k] \tag{1}$$

where the values $c[i] \neq 0$ are called the filter's coefficients.

There are only a few applications (e.g. adaptive filters) where general programmable filter architecture is required. In many cases the coefficients do not change over time - linear time-invariant filters (LTI). Digital filters are generally classified as being finite impulse response (FIR) or infinite impulse response (IIR). According to the names, an FIR filter consists of a finite number of samples values, reducing the above presented convolution to a finite sum per output sample. An IIR filter requires that an infinite sum has to be performed. In this paper implementation of the LTI FIR filters will be discussed.

The output of an FIR filter of order (length) L, to an input time-samples $x[n]$, is given by a finite version of convolution sum

$$y[n] = \sum_{k=0}^{L-1} x[k] c[k] \tag{2}$$

The L-th order LTI FIR filter is schematically presented in Fig. 1. It consists of a collection of delay line, adders and multipliers.

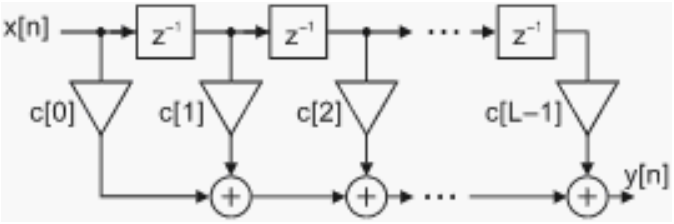

Fig. 1. Direct form FIR filter.

Available digital filter software allows for very easy computation of coefficients of given filter. However, the challenge is in mapping the FIR structure into suitable architecture. Digital filters are typically implemented as multiply-accumulate algorithms with use of special DSP devices. In case of programmable structures direct or transposed forms are preferred for maximum speed and lowest resource utilization. Efficient hardware implementation of filter's structure is possible by optimization of multipliers and adders implementation.

A completely different FIR architecture is based on the distributed arithmetic concept. In contrast to a conventional sum-of-product architecture, in distributed arithmetic the sum of product of a specific bit of input sample over all coefficients is always computed in one step.

3 Fourier Transform

The essence of the Fourier transform of a waveform is to decompose or separate the waveform into a sum of sinusoids of different frequencies. In other words, the Fourier transform identifies or distinguishes the different frequency sinusoids, and their respective amplitudes, which combine to form an arbitrary waveform. The Fourier transform is then a frequency domain representation of a function. This transform contains exactly the same information as that of the original function; they differ only in the manner of presentation of the information [16]. Fourier analysis allows one to examine a function from another point of view, the frequency domain.

The Discrete Fourier Transform (DFT) is described by the following formula:

$$F(k) = \sum_{n=0}^{N-1} f(n) e^{-j\frac{2\pi nk}{N}}, \quad \text{for} \quad 0 \leq k \leq N-1 \tag{3}$$

DFT transforms the sequence of N complex numbers $x_0, \ldots, x_{N-1}$ (time domain samples) into the sequence of N complex numbers $X_0, \ldots, X_{N-1}$ called frequency domain samples.

If $x_0, \ldots, x_{N-1}$ are real numbers, as they often are in practical applications, then the DFT obeys the symmetry $X_k = X^*_{N-k}$, where the * denotes complex conjugation and the subscripts are interpreted modulo N. Therefore, the DFT output for real inputs is half redundant, and one obtains the complete information by only looking at roughly half of the outputs.

Computation of N-point DFT requires N^2 complex valued multiplications ($4 \times N^2$ real valued multiplications). Typical case in digital signal processing is transformation of real valued signals, so the DFT needs only $2 \times N^2$ real valued multiplications. For both cases DFT's computational complexity is $O(N^2)$.

In 1965, IBM researcher Jim Cooley and Princeton faculty member John Tukey developed what is now known as the Fast Fourier Transform (FFT) [18]. It is an algorithm for computing DFT where the computational complexity is of order $O(N \log N)$ for certain length inputs. Now when the length of data doubles, the spectral computational time will not quadruple as with the DFT algorithm; instead, it approximately doubles. Later research showed that no algorithm for computing the DFT could have a smaller complexity than the FFT.

The most well-known use of the Cooley-Tukey algorithm is to divide the transform into two pieces of size $N/2$ at each step, and is therefore limited to power-of-two sizes. It is called the radix-2 algorithm. Radix-2 divides a DFT of size N into

two interleaved DFTs of size $N/2$ with each recursive stage.

$$\begin{aligned} F(k) &= \sum_{n=0}^{\frac{N}{2}-1} f(2n)e^{-j\frac{2\pi(2n)k}{N}} + \sum_{n=0}^{\frac{N}{2}-1} f(2n+1)e^{-j\frac{2\pi(2n+1)k}{N}} \\ &= \sum_{n=0}^{\frac{N}{2}-1} f(2n)W_N^{(2n)k} + \sum_{n=0}^{\frac{N}{2}-1} f(2n+1)W_N^{(2n+1)k} \\ &= \sum_{n=0}^{\frac{N}{2}-1} f(2n)W_{N/2}^{nk} + \sum_{n=0}^{\frac{N}{2}-1} f(2n+1)W_{N/2}^{nk}W_N^k \\ &= F_{\text{even}}(k) + F_{odd}(k)W_N^k \end{aligned} \tag{4}$$

Radix-2 first computes the Fourier transforms of the even-indexed input samples and of the odd-indexed input samples, and then combines those two results to produce the Fourier transform of the whole sequence. This idea can then be performed recursively to reduce the overall runtime to $O(N \log N)$. This simplified form assumes that N is a power of two; since the number of sample points N can usually be chosen freely by the application, this is often not an important restriction.

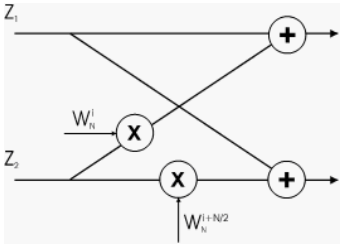

Fig. 2. Schematic diagram of butterfly operation.

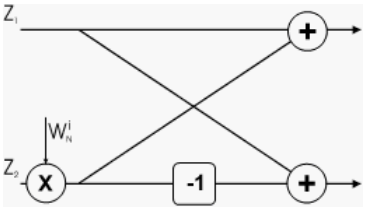

Fig. 3. Simplified butterfly operation.

Basic operation in radix-2 algorithm is called butterfly due to the shape of the dataflow diagram (Fig. 2). Butterfly operation requires two complex multiplications. Since $W_N^{N/2+k} = -W_N^k$, by replacing one addition with subtraction only one complex valued multiplication need to be performed. The simplified butterfly operation is shown in Figure 3.

Figure 4 shows the diagram of an 8-point DFT. In the diagram, the radix-2 decimation-in-time algorithm is used. In the algorithm, the input samples are permuted so that they follow the so called bit-reversed order [19].

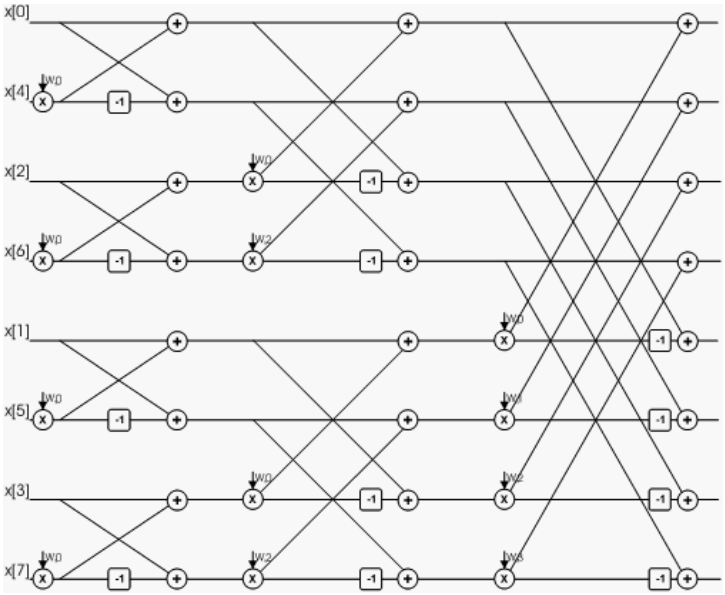

Fig. 4. Schematic diagram of 8-point DFT.

4 Distributed Arithmetic

Distributed arithmetic is a method of computing the sum of products. In many DSP applications, a general purpose multiplication is not required. In case of filter implementation, if filter coefficients are constant in time, then the partial product term $x[n]c[n]$ becomes multiplication with a constant. Then taking into account the fact that the input variable is a binary number:

$$x[n] = \sum_{b=0}^{B-1} x_b[n] \cdot 2^b, \quad \text{where} \quad x_b[n] \in [0,1] \tag{5}$$

the whole convolution sum can be described as shown below.

$$y[n] = \sum_{b=0}^{B-1} 2^b \cdot \sum_{k=0}^{L-1} x_b[k] \cdot c[k] = \sum_{b=0}^{B-1} 2^b \cdot \sum_{k=0}^{L-1} f(x_b[k], c[k]). \tag{6}$$

The efficiency of filter implementation based on this concept strongly depends on implementation of the function $f(x_b[k], c[k])$. The preferred implementation method is to realize the mapping $f(x_b[k], c[k])$ as the combinational module with L inputs. The schematic representation of such implementation is shown in Fig. 5,

where the mapping f is presented as a lookup table that includes all the possible linear combinations of the filter coefficients and the bits of the incoming data samples [1]. The utility programs that generate the lookup tables for filters with given coefficients can be found in the literature.

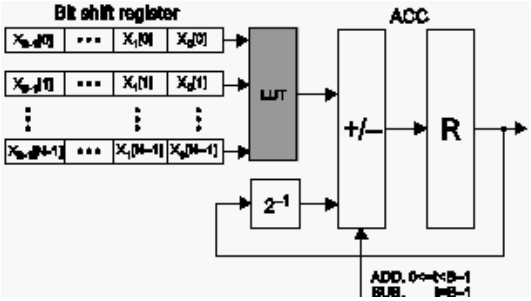

Fig. 5. DA architecture with lookup table.

The hardware description language (HDL) specification of the lookup table can be easily obtained for filter described by its $c[i]$ coefficients. Since the size of lookup tables grows exponentially with the number of inputs the efficient implementation of these blocks becomes crucial to final resource utilization of filter implementation. In the approach presented in this paper, the balanced decomposition has been successfully applied for technology mapping of DA circuits onto FPGA logic cells.

5 Balanced Functional Decomposition

There are several approaches to FPGA-based logic synthesis. The most common approach relies on breaking of the synthesis process into two phases: a technology independent one, and a technology mapping phase. The technology independent phase attempts to generate an optimal abstract representation of the logic circuit. For the combinational logic, the abstract representation is a Boolean network, i.e. a structure of a directed acyclic graph $G(V,E)$ where each node $v \in V$ represents an arbitrarily complex single-output logic function.

The second phase of logic synthesis maps the design onto cells of a user specified target library, and performs technology dependent optimizations taking the given constraints into account. For FPGAs the constraints are specific because their structures differ from the structures of the standard ASIC technologies. The architecture based on LUTs is the prevalent one among many FPGA architectures.

LUT-based FPGAs consist of an array of LUTs, each of which can implement any Boolean function with up to k (typically 4 or 5) inputs. A Boolean network can be directly realized by a one-to-one mapping between nodes and LUTs if every node in the network is feasible, i.e. has up to k input variables. Thus in FPGA-based technology mapping the functional decomposition algorithm is usually applied to multi-output functions which result from a node clustering process in a Boolean network [8].

A serial decomposition of the Boolean function $F(X) = Y$ is defined as follows (Fig. 6). Let $X = A \cup B$ be the set of input variables, Y the set of output variables and $C \subseteq A$. There exists a serial decomposition of F if $F = H(A, G(B,C)) = H(A,Z)$, where G and H denote functional dependencies $G(B,C) = Z$ and $H(A,Z) = Y$, and Z is the set of output variables of G. If, in addition, $C = \Phi$, then H is called a disjoint decomposition of F.

The functional decomposition algorithms are usually incorporated into a multilevel synthesis environment [8], where the nodes are created and then, each of the nodes is treated as a Boolean function to be decomposed. In other words, each such node then constitutes an input to the decomposition algorithm.

A completely different approach to FPGA-based technology mapping was introduced by Łuba and Selvaraj [20], where the concept of parallel decomposition was introduced and effectively applied in the so called balanced decomposition method. Based on redundant variable analysis of each output of a multi-output function, parallel decomposition separates F into two or more functions, each of which has as its inputs and outputs a subset of the original inputs and outputs. Although in their method (recently improved in [21]), the crucial point of the whole mapping process is again created by the serial decomposition algorithm, the parallel decomposition based on argument reduction process plays a very important role. Thanks to this algorithm the functional decomposition procedure can start directly with a two-level, espresso based specification. Thus the method itself allows to develop a uniform, autonomous tool for decomposition based technology mapping of FPGAs. The influence of these improvements which partly rely on application of argument reduction algorithm mentioned above on the results of FPGA-based technology mapping will be shortly described below.

Consider a multi-output function F. Assume that F has to be decomposed into two components, G and H, with disjoint sets Y_G and Y_H of output variables (Fig. 7). This problem occurs, for example, when we want to implement a large function using components with a limited number of outputs. Note that such a parallel decomposition can also alleviate the problem of an excessive number of inputs of f. This is because, for typical functions, most outputs do not depend on all input variables. Therefore, the set X_G of input variables on which the outputs of Y_G depend, may be smaller than X. Similarly, the set X_H of input variables on which the

outputs of Y_H depend may be smaller than X. As a result, components G and H have not only fewer outputs, but also fewer inputs than F. The exact formulation of the parallel decomposition problem depends on the constraints imposed by the implementation style. One possibility is to find sets Y_G and Y_H such that the combined cardinality of X_G and X_H is minimal. Partitioning the set of outputs into only two disjoint subsets is not an important limitation of the method, because the procedure can be applied again for components G and$H$.

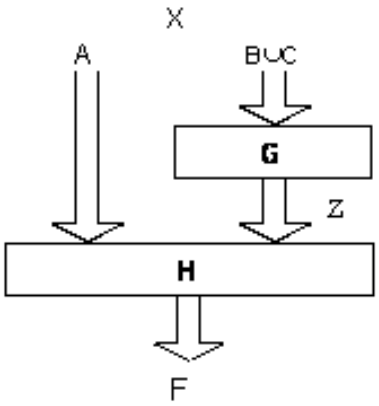

Fig. 6. Schematic representation of serial decomposition.

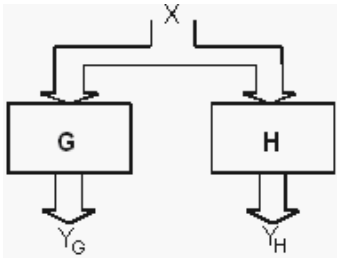

Fig. 7. Schematic representation of parallel decomposition.

Example 1. The influence of the parallel decomposition on the final result of the FPGA-based mapping process will be explained with the function F given in Table 1, for which cells with 4 inputs and 1 output are assumed (this is the size of Altera's FLEX FPGAs).

Table 1. Truth table of functions F

.type fr	0001001110 01
.i 10	0110000110 01
.o 2	1110110010 10
.p 25	0111100000 00
0101000000 00	0100011011 00
1110100100 00	0010111010 01
0010110000 10	0110001110 00
0101001000 10	0110110111 11
1110101101 01	0001001011 11
0100010101 01	1110001110 10
1100010001 00	0011001011 10
0011101110 01	0010011010 01

As F is a ten-input, two-output function, in the first step of the decomposition either parallel or serial decomposition can be applied. If we first apply serial decomposition (Fig. 8), then the algorithm extracts function g with inputs x_1, x_3, x_4, and x_6, thus the next step deals with seven-input function h, for which again serial decomposition is assumed, now resulting in block g, with 4 inputs and 2 outputs (implemented by 2 Logic Cells -LC). It is worth noting that the obtained block g takes as its input variables x_0, x_2, x_5, and x_7, which, fortunately, belong to primary variables, and therefore the number of levels is not increased in this step. In the next step we apply parallel decomposition. Parallel decomposition generates two components, both with one output but 4 and 5 inputs, respectively. The first one forms a logic cell. The second component is subject to two-stage serial decomposition shown in Fig. 8. The obtained network can be built of 7 (4 to 1) cells, where the number of levels in the critical path is 3.

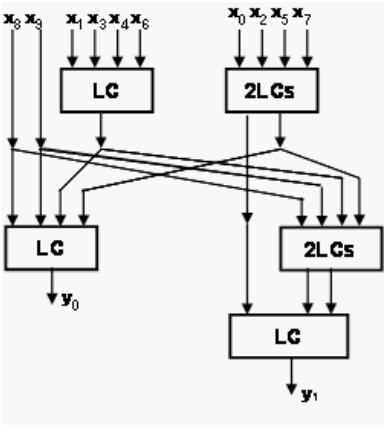

Fig. 8. Decomposition of function F where serial decomposition is performed first.

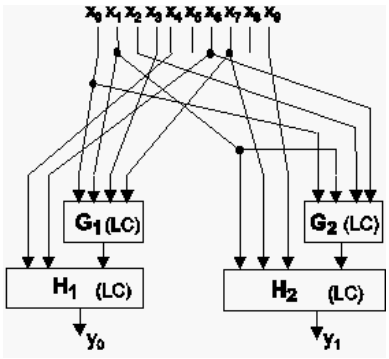

Fig. 9. Decomposition of function F where parallel decomposition is performed at first.

Decomposition of the same function such that the parallel decomposition is applied in the first step leads to completely different structure (Fig. 9). Parallel decomposition applied directly to function F, generates two components both with 6 inputs and one output. Each of them is subject to two-stage serial decomposition. For the first component, a disjoint serial decomposition with four inputs and one output can be applied. The second component can be decomposed serially as well, however with the number of outputs of the extracted block G equals to two. Therefore, to minimize the total number of components, a non-disjoint decomposition strategy can be applied. The truth tables of the decomposed functions G_1, H_1, G_2, H_2, are shown in Table 2. The columns in the table denote variables in

the order shown in Fig. 9. For example, the first left hand side column in Table 2b denotes variable x_4, the second variable x_6, and the third denotes variable g_1 (output of G_1). The above considerable impact on the structure results from the fact that the parallel decomposition simultaneously reduces the number of inputs to both resulting components, leading to additional improvement of the final representation.

Table 2. Truth tables of decomposition components.

a) function G_1	b) function H_1	c) function G_2	d) function H_2
0110 1	−01 0	0110 1	10−1 0
1101 1	011 1	0011 1	−101 1
1000 1	111 0	0100 1	−111 1
0010 1	100 1	1000 1	0011 0
0000 0	0−0 0	0101 1	0001 1
0101 0	110 0	1100 0	1−00 0
1100 0		0010 0	0000 0
0100 0		1010 0	1110 1
0011 0		1110 0	1010 0
1011 0		0001 0	0100 1
1111 0		0111 0	0010 1
		1111 0	

It is worth noticing that the same function synthesized directly by commercial tool, e.g. Quartus can be mapped onto 32 logic cells.

The serial and parallel decompositions are intertwined in a top-down synthesis process to obtain the required topology. At each step, either parallel or serial decomposition is performed, both characterized by operation input parameters. In the case of serial decomposition the related parameter G_{in} and G_{out} denotes the number of G block inputs and outputs, respectively. In the case of parallel decomposition the related parameter G_{out} represents the number of G block outputs. Intertwining of serial and parallel decomposition strategies opens up several interesting possibilities in multilevel decomposition. Experimental results show that the right balance between the two strategies and the choice of operation parameters severely influence the area and depth of the resultant network.

Example 2. The influence of the right balance on the final result of the FPGA-based mapping process will be explained with the function F representing DA logic of a certain wavelet filter with the following filter coefficients [1495,−943, −9687, 18270, −9687, −943, 1495].

As F is a seven-input, sixteen-output function, in the first step of the decomposition both the parallel and serial decomposition can be applied. Let us apply parallel decomposition at first (Fig.10). Parallel decomposition with $G_{out} = 1$ generates two components: the first one with 6 inputs and 1 output, and the second

with 7 inputs and 15 outputs. This is illustrated by two arrow marks with the common starting point going to different directions. The smaller component is subject to two-stage serial decomposition resulting in block *G* with 4 inputs and 1 output and block *H* with 3 inputs and 1 output (both *G* and *H* blocks are implemented by 2 cells). Two brackets (4,1), (3,1), which are given on the bottom side of the arrow mark, show the number of inputs and outputs for functions $G(4,1)$ and $H(3,1)$, respectively. The second component is again decomposed in parallel yielding (7,7) and (7,8) components. For the (7,8) component serial decomposition is assumed, now resulting in block *G* with 4 inputs and 2 outputs (implemented by 2 logic cells), thus the next step deals with six-input function H, which can be directly implemented in ROM. In the next iterative step parallel decomposition is applied to split the (7,7) component into (7,3) and (7,4) blocks. It is sensible to implement the (7,4) block in ROM. The second block is decomposed serially yielding $G(4,3)$ and $H(6,3)$. As *G* block can be implemented by 3 logic cells, the next step deals with function *H*. Parallel decomposition applied to function *H* generates two components. Each of them is subject to two-stage serial decomposition. The obtained network can be built of 14 logic cells and 2 M512 ROMs.

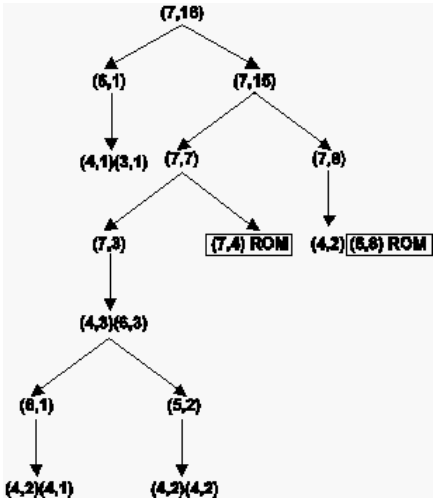

Fig. 10. Decomposition process for the ahp (7,16) filter.

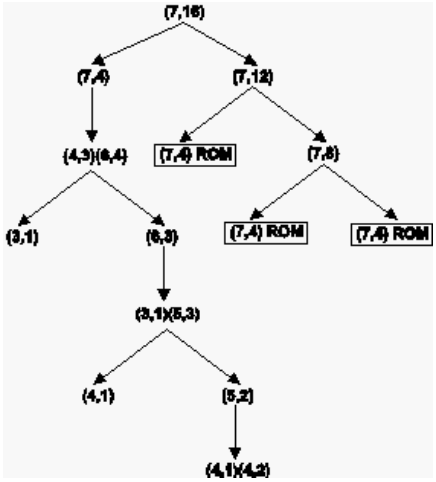

Fig. 11. Decomposition process for the ahp (7,16) filter.

If we change the size of smaller component in the first step of parallel decomposition, i.e. (7,4) instead of (6,1) as in Fig.10, then the implementation needs 3 M512 ROMs and 9 LCs. The structure is shown in Fig.11. However, if we decide on serial decomposition to decompose (7,16), instead of parallel decomposition as

in Fig.10 and 11, the implementation needs only 3 ROMs. The structure is shown in Fig.12.

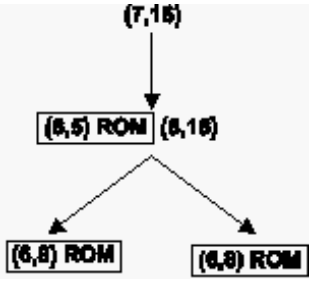

Fig. 12. Decomposition process for the ahp (7,16) filter.

Balanced decomposition was implemented as software package called DEMAIN [12]. Recently the package was improved to help designers to deal with large truth tables. All described methods of truth tables transformations can be performed easily, and results are shown immediately on the screen for further work. It is designed for performing manual operations on functions, and therefore is meant to be highly user friendly, as well as cross-platform compatible. After choosing the operation, a dialog pops up which can be used to input the parameters of the operation. After the actual operation is performed, its results are displayed in the project window.

6 Synthesis of FIR Filters

Below the experimental results of FIR filter implementation with different design methodologies are presented. For experiments, filter found in [22] as well as Daubechies' dbN, coifN, symN and 9/7-tap bio-orthogonal filters have been chosen.

In the first experiment filter with length (order) 15 has been chosen. It has 8-bit signed input samples and its coefficients can be found in [22]. For the comparison the filter has been implemented in Stratix EP1S10F484C5, Cyclone EP1C3T100C6 and CycloneII EP2C5T144C6 structures with use of Altera QuartusII v5.1 SP0.15.

Table 3 presents the comparison of implementation results for different design methodologies. Column falling under the "MAC" label present the results obtained by implementing multiply-and-accumulate strategy with use of logic cell resources, without utilization of embedded DSP blocks. Multipliers, as well as accumulator were implemented in logic cells of circuit. This implementation, due to its serial character, requires 15 clock cycle to compute the result. It requires relatively large

amount of resources, while delivering the worst performance in comparison to other implementations.

Table 3. Implementation results for different design methodologies. Chip: S - Stratix EP1S10F484C5, C - Cyclone EP1C3T100C6, CII - CycloneII EP2C5T144C6.

Chip		MAC	MULT block	DSP block	Parallel	DA	DA decomposed
S	LC	421	287	247	402	1013	569
	DSP	0	2	4	30	0	0
	F_{max}[MHz]	80.44	86.01	105.34	58.97	87.6	84.86
C	LC	421	421	421	2226	1013	569
	DSP[a]	–	–	–	–	–	–
	F_{max}[MHz]	77.03	77.03	77.03	61.0	80.4	78.37
CII	LC	403	271	271	637	1014	569
	DSP	0	2	2	26	0	0
	F_{max}[MHz]	89.92	102.43	102.43	76.49	84.11	82.61

[a] DSP blocks are not present in this device family

Next column - “MULT block” - holds the implementation results of method similar to “MAC” with such difference that multipliers were implemented in dedicated DSP embedded blocks. It can be noticed that the performance of the filter increased at the cost of utilization of additional resources in form of DSP embedded blocks. Results in column falling under “DSP block” were obtained by implementing the whole MAC unit in embedded DSP block. Further increase in performance could be noticed, but still 15 clock cycles have to be used to compute the result.

Results given in “Parallel” column were obtained by implementing filter in parallel manner. In this case results are obtained in single clock cycle. Even though the maximal frequency of this implementation is less than in previous ones, it outperforms these implementations due to its parallel character.

Application of DA technique results in increase of performance since maximal frequency has increased. However in this approach more logic cell resources have been used, since multipliers have been replaced by large combinational blocks and no DSP embedded modules ware utilized.

Finally results presented in column under “DA decomposed” label demonstrate that application of DA technique combined with advanced synthesis method based on balanced decomposition allows obtaining the circuit that not only outperforms any other implemented circuit but also reduces the necessary logic resources. The balanced decomposition was applied to decomposed combinational blocks of DA implementation.

In Table 4, the experimental results of Daubechies’ dbN, coifN, symN and 9/7-tap bio-orthogonal filter banks are presented. Filters 9/7 are in two versions: (a)

analysis filter and (s) synthesis filter. Filters dbN, coifN, symN are similar for analysis and synthesis (a/s). All filters have 16 bit signed samples and have been implemented with the use of DA concept in the fully parallel way. Balanced decomposition software was also added to increase efficiency of the DA tables' implementations.

Table 4. Implementation results of filters with and without decomposition.

Filter	Order	Without decomposition		With decomposition	
		LC	F_{max}[MHz]	LC	F_{max}[MHz]
db3, a/s low-pass	6	1596	278.63	1345	254.26
db4, a/s low-pass	8	3747	212.9	2891	201,73
db5, a/s low-pass	10	10057	169.81	7377	119.39
db6, a/s low-pass	12	–[a]	–	31153	–[b]
9/7, a low-pass	9	3406	206.61	1505	212.86
9/7, s low-pass	7	1483	273.37	881	263.5
9/7, a high-pass	7	2027	253.29	1229	223.16
9/7, s high-pass	9	4071	180.93	1616	189.47
coif6, a/s low-pass	6	1133	283.45	1041	260.62
coif12, a/s low-pass	12	–[a]	–	1614	196.85
sym8, a/s low-pass	8	3663	212.72	2249	197.94
sym12, a/s low-pass	12	–[a]	–	2313	198.61
sym14, a/s low-pass	14	–[a]	–	2345	200.24
sym16, a/s low-pass	16	–[a]	–	2377	206.83

[a] Too long compilation time (more than 24 hours)
[b] Does not fit in EP1S10F484C5

Table 4 presents the result for filter implementations using Stratix EP1S10F484C5 device, with a total count 10570 of logic cells. In the implementation without decomposing the filters, the new method was modeled in AHDL and Quartus2v6.0SP1 was used to map the model into the target structure. In the implementation using decomposition, the DA tables were first decomposed using automatic software. Quartus system was then applied to map the filters into FPGA.

The application of the balanced decomposition concept significantly decreased the logic cell resource utilization and at the same time increased the speed of the implementation.

FPLD devices have very complex structure. They combine PLA-like structures as well as FPGA's and even memory-based structures. In many cases designers cannot utilize all of these possibilities such as complex architectures provide due to the lack of appropriate synthesis methods. Embedded memory arrays make possible an implementation of memory like blocks such as large registers, FIFO's, RAM or ROM modules [1].

These memory resources make up considerably large part of the devices. For

example, EP20K1500E devices provide 51 840 logic cells and 442 Kbit of SRAM. Taking under consideration the conversion factors of logic elements and memory bits to logic gates (12 gates/logic element and 4 gates/memory bit) it turns out that embedded memory arrays make up over 70% of all logic resources. Since not every design consists of such modules as RAM or ROM, in many cases these resources are not utilized. However, such embedded memory blocks can be used for implementation of DA blocks in a way that requires less resources than the traditional cell-based implementation. This may be used to implement "non-vital" sequential parts of the design, saving logic cell resources for more important sections. Since the size of embedded memory blocks is limited, such an implementation may require more memory than is available in a device. To reduce a memory usage in ROM-based DA implementations, astructure with combinational logic partially implemented in the ROM and partially implemented in logic cells was proposed.

In Table 5, the experimental results of Daubechies' 9/7-tap bio-orthogonal filter banks are presented. All filters have 16 bit signed samples and have been implemented with the use of DA concept. Balanced decomposition software was also added to increase efficiency of the DA tables' implementations.

Table 5. Implementation results of 9/7 filters.

Filter	Order	LC	ROM	FF	bits	F_{max}
alp	9	236	7xM512, 1xM4K	181	8192	133.51
alp dec	9	248	1xM4K	181	4096	140.51
ahp	7	204	4xM512	149	2048	155.04
ahp dec	7	210	2xM512	153	1024	157.53
slp	7	204	4xM512	149	2048	155.04
slp dec	7	211	2xM512	153	1024	161.21
shp	9	236	7xM512, 1xM4K	181	8192	133.51
shp dec	9	246	1xM4K	181	4096	134.25

Table 5 presents the results for filter implementations using Stratix EP1S10F484C5 device. In the implementation without decomposing the filters, the method was modeled in AHDL and Quartus2v6.0SP1 was used to map the model into the target structure. In the implementation using decomposition (denoted dec), DEMAIN software was used to initially decompose DA tables and then Quartus system was applied to map the filters into FPGA.

Filters 9/7 are in two versions: (a) analysis filter and (s) synthesis filter. Low pass and high pass filters are denoted as lp and hp, respectively. The implementation of filters is characterized by the number of logic cells (LC) and Flip-Flops (FF), memory bits, the number of memory modules (ROM) and operating frequency. In all cases, decomposition reduces the sizes of memory and the number of memory modules. For example, implementation of ahp filter requires 204 LCs and 4 M512

embedded memories if performed by Quartus software. Application of DEMAIN tool allows DA logic of this filter to be implemented with 2 M512 memories and 11 LCs. This allows implementing the whole filter with 210 LCs and 2 M512 memories.

7 Synthesis of DFT

It has been shown that no algorithm for computing the DFT can have a smaller complexity than the FFT. Thus most FPGA implementations are based on this approach. With the introduction of specialized DSP blocks embedded into programmable architectures the efficiency of FFT is limited by the speed of hardware multipliers of DSP modules.

However, programmable architectures provide possibility to increase the performance of digital system by exploiting the parallelism of the implemented algorithms. DFT transforms the sequence of N complex numbers $x_0,\ldots,x_{N-1}$ into the sequence of N complex numbers $X_0,\ldots,X_{N-1}$. Each output sample is computed as sum of products of input samples with constant coefficients. Implementation of DFT based on DA concept in FPGA structure requires computation of each output sample with the DA unit presented in Fig. 5. Since in practical applications most frequently DFT of real valued input samples is required, the implementation can benefit from the symmetry $X_k = X^*_{N-k}$. Therefore, the DFT output for real inputs can be obtained by only looking at roughly half of the outputs.

Each DA unit contains a number of DA tables, which are combinational circuits, and an adder tree. Since adder tree can be efficiently implemented using low-latency carry-chain-routing lines of the FPGA device, the implementation quality of DA unit (and the whole DFT) mostly depends on the quality of DA tables' implementation.

Below the implementations' comparison of 16-points DFT of real valued 12 bits input samples are presented. For comparison three design methodologies were chosen:

- FFT_LC – radix-2 FFT; implementation in logic cell resources (LC) only,
- FFT_DSP – radix-2 FFT; implementation with use of logic cell resources, as well as embedded DSP modules for fast hardware multiplication,
- DFT_DA – distributed arithmetic based implementation.

For the implementations, device EP2C35F672C6 from Altera's CycloneII family was chosen. The implementations were performed using Quartus 6.0 SP1 system. To efficiently utilize possibilities provided by DSP embedded blocks of CycloneII device Library Parameterized Modules (LPM) were used in HDL description of FFT_LC and FFT_DSP algorithms.

Logic synthesis methods implemented in Quartus CAD system do not allow efficient mapping of DA tables into logic cells. Compilation of large DA tables of DA implementation of DFT required to much time and resulted in large logic cell resource utilization. Thus decomposition based methods, which are best suited for FPGA architectures, were used to optimize DA table implementation.

Table 6 presents the results of DFT implementation using FFT radix-2 algorithm in logic cells only (row labeled FFT_LC) and with utilization of embedded DSP blocks (row labeled FFT_DSP). Row labeled DFT_DA presents the result of DFT implementation based on DA concept. Columns of Table 6 present the logic cell resource and embedded DSP blocks usage. Numbers in brackets show the percentage of total chip resources utilization. In the table the maximal frequency and achieved throughput are also presented.

Table 6. Implementation results of DFT.

	Resource usage		Clock frequency [MHz]	Throughput [Mbit/s]
	[#LC]	[#DSP]		
FFT_LC	4723 (14%)	–	43.51	522.12
FFT_DSP	1554 (5%)	70(100%)	48.93	587.16
DFT_DA	7222 (22%)	–	74.36	892.32

The classical implementation of FFT required 4723 logic cells, which constitutes 14% of total logic cells available. The throughput of this implementation is 522.12 Mbit/s. It can be noticed that the utilization of embedded DSP blocks in FFT_DSP implementation decreased the number of needed logic cell, and at the same time increasing the throughput to 587.16 Mbit/s. Utilization of 70 DSP blocks, which is 100% of available blocks, reduced the logic cell utilization from 14% to 5%.

However the best performance of 892.32 Mbit/s is achieved when DA concept is used. This DFT realization required 53% more logic cells in comparison to FFT_LC implementation but the performance was increased by as much as 71%.

The efficiency of DA based implementation strongly depends on logic synthesis quality. In the paper decomposition based synthesis methods developed by authors were used to implement DA tables, since Quartus CAD system was unable to map them in reasonable time. Development of more sophisticated synthesis methods directed to DA implementation may give much more efficient DFT modules.

8 Conclusions

The modern programmable structures deliver the possibilities to implement DSP algorithms in dedicated embedded blocks. This makes designing of such algorithm an easy task. However the flexibility of programmable structures enables more advanced implementation methods to be used. In particular, exploitation of parallelism in the algorithm to be implemented may yield very good results. Additionally, the application of advanced logic synthesis methods based on balanced decomposition, which is suitable for FPGA structure leads to results that can not be achieved with any other method.

The presented results lead to the conclusion that if the designer decides to use the methodology known from DSP processor application, the implementation quality will benefit from the utilization of specialized DSP modules embedded in the programmable chip. However, best results can be obtained by utilizing the parallelism in implemented algorithms and by applying advanced synthesis methods based on decomposition. Influence of the design methodology and the balanced decomposition synthesis method on the efficiency of practical digital filter implementation is particularly significant, when the designed circuit contains complex combinational blocks. This is a typical situation when implementing digital filters using the DA concept.

The most efficient approach to logic synthesis of FIR filter algorithms discussed in this paper relies on the effectiveness of the functional decomposition synthesis method. These methods were already used in decomposition algorithms; however they were never applied together in a technology specific mapper targeted at a lookup table FPGA structure. This paper shows that it is possible to apply the balanced decomposition method for the synthesis of FPGA-based circuits directed towards area or delay optimization.

Acknowledgements

This paper was supported by Ministry of Science and Higher Education financial grant for years 2006-2009 (Grant No. SINGAPUR/31/2006) as well as Agency for Science, Technology and Research in Singapore (Grant No.0621200011).

References

[1] U. Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays*. Berlin: Springer-Verlag, 2004.

[2] A. Peled and B. Liu, "A new realization of digital filters," *IEEE Trans. on Acoustics, Speech and Signal Processing*, vol. 22, no. 6, pp. 456–462, June 1974.

[3] M. Rawski, P. Tomaszewicz, H. Selvaraj, and T. Łuba, "Efficient implementation of digital filters with use of advanced synthesis methods targeted fpga architectures," in *Proc. of Eighth Euromicro Conference on Digital System Design (DSD 2005)*, Porto, Portugal, Aug. 2005, pp. 460–466.

[4] M. Rawski, P. Tomaszewicz, and T. Łuba, "Logic synthesis importance in fpga-based designing of information and signal processing systems," in *Proc. of International Conference on Signal and Electronics Systems*, Poznań, Poland, 2004, pp. 425–428.

[5] T. Sasao, Y. Iguchi, and T. Suzuki, "On lut cascade realizations of fir filters," in *Proc. of Eighth Euromicro Conference on Digital System Design (DSD 2005)*, Porto, Portugal, Aug. 2005, pp. 467–474.

[6] J. T. Astola and R. S. Stanković, *Fundamentals of Switching Theory and Logic Design*. Dordrecht: Springer, 2006.

[7] J. A. Brzozowski and T. Łuba, "Decomposition of boolean functions specified by cubes," *Journal of Multiple-Valued Logic and Soft Computing*, vol. 9, pp. 377–417, 2003.

[8] S. C. Chang, M. Marek-Sadowska, and T. T. Hwang, "Technology mapping for tlu fpgas based on decomposition of binary decision diagrams," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 10, pp. 1226–1236, Oct. 1996.

[9] M. Rawski, L. Jóźwiak, and T. Łuba, "Functional decomposition with an efficient input support selection for sub-functions based on information relationship measures," *Journal of Systems Architecture*, vol. 47, pp. 137–155, 2001.

[10] C. Scholl, *Functional Decomposition with Application to FPGA Synthesis*. Kluwer: Academic Publishers, 2001.

[11] T. Łuba, H. Selvaraj, M. Nowicka, and A. Kraśniewski, "Balanced multilevel decomposition and its applications in fpga-based synthesis," in *Logic and Architecture Synthesis*, G. Saucier and A. Mignotte, Eds., 1995.

[12] M. Nowicka, T. Łuba, and M. Rawski, "Fpga-based decomposition of boolean functions: Algorithms and implementation," in *Proc. of Sixth International Conference on Advanced Computer Systems*, Szczecin, Poland, 1999, pp. 502–509.

[13] B. J. Falkowski, "Haar transform: Calculation, generalizations, and applications in logic design and signal processing," in *Proc. of International Workshop on Transforms and Filter Banks (2nd IWTFB)*, Brandenburg, Germany, Mar. 1999, pp. 101–120.

[14] ——, "Compact representations of logic functions for lossless compression of grey scale images," *IEE Proc., Computers and Digital Techniques, United Kingdom*, vol. 151, no. 3, pp. 221–230, May 2004.

[15] R. M. Rao and A. S. Bopardikar, *Wavelet Transform: Introduction to Theory and Applications*. Addison-Wesley, 1998.

[16] O. Rioul and M. Vetterli, "Wavelets and signal processing," *IEEE Signal Processing Magazine*, vol. 8, no. 4, pp. 14–38, Oct. 1991.

[17] C. M. Brislawn, C. B. J. Bradley, R. Onyshczak, and H. T., "The fbi compression standard for digitized fingerprint images," in *Proc. of SPIE Conference 2847*, Denver, USA, 1996, pp. 344–355.

[18] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, pp. 297–301, 1965.

[19] R. G. Lyons, *Understanding Digital Signal Processing*. Upper Saddle River: Prentice Hall, 2004.

[20] T. Łuba and H. Selvaraj, "A general approach to boolean function decomposition and its applications in fpga-based synthesis," *VLSI Design*, vol. 3, no. 3-4, pp. 289–300, 1995.

[21] P. Tomaszewicz, M. Nowicka, B. J. Falkowski, and T. Łuba, "Logic synthesis importance in fpga-based designing of image signal processing systems," in *Proc. of the 14th International Conference on Mixed Design of Integrated Circuits and Systems (MIXDES 2007)*, Ciechocinek, Poland, June 2007, pp. 141–146.

[22] D. J. Goodman and M. J. Carey, "Nine digital filters for decimation and interpolation," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 25, no. 2, pp. 121–126, 1977.