UNLV | UNIVERSITY LIBRARIES

Electrical and Computer Engineering Faculty Publications

Electrical & Computer Engineering

6-11-2008

Significance of Logic Synthesis in FPGA-Based Design of Image and Signal Processing Systems

Mariusz Rawski
Warsaw University of Technology, rawski@tele.pw.edu.pl

Henry Selvaraj
University of Nevada, Las Vegas, henry.selvaraj@unlv.edu

Bogdan J. Falkowski
Nanyang Technological University, Singapore

Tadeusz Luba
Institute of Telecommunications Warsaw

Follow this and additional works at: https://digitalscholarship.unlv.edu/ece_fac_articles

Part of the Computer Engineering Commons, Electrical and Electronics Commons, Power and Energy Commons, Signal Processing Commons, and the Systems and Communications Commons

Repository Citation

Rawski, M., Selvaraj, H., Falkowski, B. J., Luba, T. (2008). Significance of Logic Synthesis in FPGA-Based Design of Image and Signal Processing Systems. *Pattern Recognition Technologies and Applications: Recent Advances* 265-283. Hershey, PA: IGI Global.
https://digitalscholarship.unlv.edu/ece_fac_articles/303

This Chapter is protected by copyright and/or related rights. It has been brought to you by Digital Scholarship@UNLV with permission from the rights-holder(s). You are free to use this Chapter in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself.

This Chapter has been accepted for inclusion in Electrical and Computer Engineering Faculty Publications by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.

Pattern Recognition Technologies and Applications:
Recent Advances

Brijesh Verma
Central Queensland University, Australia

Michael Blumenstein
Griffith University, Australia

Information Science REFERENCE

INFORMATION SCIENCE REFERENCE
Hershey · New York

Acquisitions Editor: Kristin Klinger
Development Editor: Kristin Roth
Assistant Development Editor: Jessica Thompson
Senior Managing Editor: Jennifer Neidig
Managing Editor: Jamie Snavely
Assistant Managing Editor: Carole Coulson
Copy Editor: Sue Vander Hook
Typesetter: Sean Woznicki
Cover Design: Lisa Tosheff
Printed at: Yurchak Printing Inc.

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue, Suite 200
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

and in the United Kingdom by
Information Science Reference (an imprint of IGI Global)
3 Henrietta Street
Covent Garden
London WC2E 8LU
Tel: 44 20 7240 0856
Fax: 44 20 7379 0609
Web site: http://www.eurospanbookstore.com

Copyright © 2008 by IGI Global. All rights reserved. No part of this publication may be reproduced, stored or distributed in any form or by any means, electronic or mechanical, including photocopying, without written permission from the publisher.

Product or company names used in this set are for identification purposes only. Inclusion of the names of the products or companies does not indicate a claim of ownership by IGI Global of the trademark or registered trademark.

Library of Congress Cataloging-in-Publication Data

Pattern recognition technologies and applications : recent advances / Brijesh Verma and Michael Blumenstein, editors.

p. cm.

Summary: " This book provides cutting-edge pattern recognition techniques and applications. Written by world-renowned experts in their field, this easy to understand book is a must have for those seeking explanation in topics such as on- and offline handwriting and speech recognition, signature verification, and gender classification"--Provided by publisher.

Includes bibliographical references and index.

ISBN-13: 978-1-59904-807-9 (hardcover)

ISBN-13: 978-1-59904-809-3 (e-book)

1. Pattern perception. 2. Pattern perception--Data processing. I. Verma, Brijesh. II. Blumenstein, Michael.

Q327.P383 2008

006.4--dc22

2007037396

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

All work contributed to this book set is original material. The views expressed in this book are those of the authors, but not necessarily of the publisher.

Chapter XII
Significance of Logic Synthesis in FPGA-Based Design of Image and Signal Processing Systems

Mariusz Rawski
Warsaw University of Technology, Poland

Henry Selvaraj
University of Nevada, USA

Bogdan J. Falkowski
Nanyang Technological University, Singapore

Tadeusz Łuba
Warsaw University of Technology, Poland

ABSTRACT

This chapter, taking FIR filters as an example, presents the discussion on efficiency of different implementation methodologies of DSP algorithms targeting modern FPGA architectures. Nowadays, programmable technology provides the possibility to implement digital systems with the use of specialized embedded DSP blocks. However, this technology gives the designer the possibility to increase efficiency of designed systems by exploitation of parallelisms of implemented algorithms. Moreover, it is possible to apply special techniques, such as distributed arithmetic (DA). Since in this approach, general-purpose multipliers are replaced by combinational LUT blocks, it is possible to construct digital filters of very high performance. Additionally, application of the functional decomposition-based method to LUT blocks optimization, and mapping has been investigated. The chapter presents results of the comparison of various design approaches in these areas.

Copyright © 2008, IGI Global, distributing in print or electronic forms without written permission of IGI Global is prohibited.

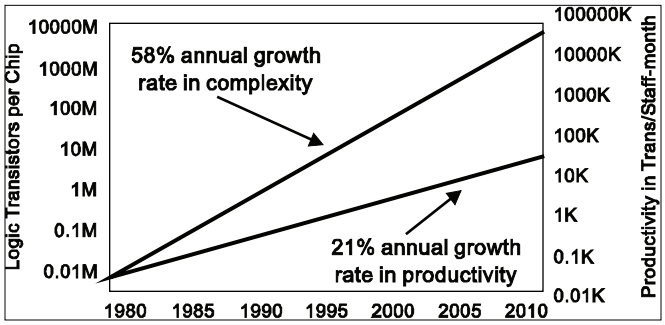
INTRODUCTION

The pattern recognition research field aims to design methods that allow recognition of patterns in data. It has important application in image analysis, character recognition, speech analysis, and many others. A pattern recognition system is composed of sensors gathering observations that have to be classified, a feature extraction part that provides specific information from gathered observation, and a classification mechanism that classifies observation on the basis of extracted features. Feature extraction methods are responsible for reducing the resources required to describe observation accurately. In the case of image analysis, character recognition, or speech analysis, various digital signal-processing (DSP) algorithms are used to detect desired features of digitalized image or speech signal. Efficient implementation of feature extraction-based DSP methods requires specific hardware solutions.

The commercial success of hardware implementations of image processing systems is due in large part to revolutionary development in microelectronic technologies. By taking advantage of the opportunities provided by modern microelectronic technology, we are in a position to build very complex digital circuits and systems at relatively low cost. There is a large variety of logic building blocks that can be exploited. The library of elements contains various types of gates, a lot of complex gates that can be generated in (semi-) custom CMOS design, and the field programmable logic families that include various types of (C)PLDs and FPGAs. The other no less important factors of the success are the automation of the design process and hardware description languages. Modern design tools have enabled us to move beyond putting together digital components in a schematic entry package to start writing code in an HDL specification. However, the opportunities created by modern microelectronic technology are not fully exploited because of weaknesses in traditional logic design methods. According to the International Technology Roadmap for Semiconductors (1997), the annual growth rate in design complexity is equal to 58%, while the annual growth rate in productivity is only 21% (Figure 1). This means that the number of logic gates available in modern devices grows faster than the ability to design them meaningfully. New methods are required to aid design process in a way that possibilities offered by modern microelectronics are utilized in the highest possible degree.

In recent years, digital filtering has been recognized as a primary digital signal processing (DSP) operation. With advances in technology, digital filters are rapidly replacing analogue filters, which were implemented with RLC components. Digital filters are used to modify attributes of signal in the time or frequency domain through a process

Figure 1. Difference in growth of device complexity and productivity



called linear convolution. Traditionally, digital signal filtering 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. Digital filters are implemented in such devices as multiply-accumulate (MAC) algorithms (Lapsley, Bier, Shoham & Lee, 1997; Lee, 1988, 1989). However, the technological advancements in field programmable gate arrays (FPGAs) in the past decade have opened new paths for DSP design engineers.

Digital filtering plays an extremely important role in many signal and image processing algorithms. An excellent example is wavelet transform, which has gained much attention in recent years. Discrete wavelet transform (DWT) is one of the useful and efficient signal and image decomposition methods with many interesting properties (Daubechies, 1992; Falkowski, 2004; Falkowski & Chang, 1997; Rao & Bopardikar, 1998; Rioul & Vetterli, 1991). This transformation, similar to the Fourier transform, can provide information about frequency contents of signals. However, unlike Fourier transform, this approach is more natural and fruitful when applied to nonstationary signals, like speech, signal, 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, in fingerprints compression, a particular set of bio-orthogonal filters—Daubechies bio-orthogonal spine wavelet filters—is found to be very effective (Brislawn, Bradley, Onyshczak & Hopper, 1996). 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.

Progress in the development of programmable architectures observed in recent years has resulted in digital devices that allow building very complex digital circuits and systems at relatively low cost in a single programmable structure. FPGAs are an array of programmable logic cells interconnected by a matrix of wires and programmable switches. Each cell performs a simple logic function defined by a designer's program. An FPGA has a large number (64 to more than 300,000) of these cells available to use as building blocks in complex digital circuits. The ability to manipulate the logic at the gate level means that a designer can construct a custom processor to efficiently implement the desired function. FPGA manufacturers have for years been extending their chips' ability to implement digital-signal processing efficiently; for example, by introducing low-latency carry-chain-routing lines that sped addition and subtraction operations spanning multiple logic blocks. Such a 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.

Programmable technology makes it possible to increase the performance of a digital system by implementing multiple, parallel modules in one chip. This technology allows also the application of special techniques such as distributed

arithmetic (DA) (Croisier, Esteban, Levilion & Rizo, 1973; Meyer-Baese, 2004; Peled & Liu, 1974). DA technique is extensively used in computing the sum of product in filters with constant coefficients. In such a case, partial product term becomes a multiplication with a constant (i.e., scaling). DA approach significantly increases the performance of an 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 such advanced synthesis methods as functional decomposition (Rawski, Tomaszewicz, & Łuba, 2004; Rawski, Tomaszewicz, Selvaraj, & Łuba, 2005; Sasao, Iguchi, & Suzuki, 2005).

In the case of applications targeting FPGA structures based on look-up 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 a sum-of-products form into a multilevel, highly factorized form that is then mapped into LUT cells. This process is at variance with the nature of the 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 a multilevel structure that is built of components, each of which is in the form of the LUT logic block specified by truth tables. Efficiency of functional decomposition has been proved in many theoretical papers (Brzozowski & Łuba, 2003; Chang, Marek-Sadowska & Hwang, 1996; Rawski, Jóźwiak & Łuba, 2001; Scholl, 2001). However, there are relatively few papers in which 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 a transforming description of project structure obtained outside a commercial design system into a description compatible with its rules. Moreover, the computation complexity of functional decomposition procedures makes it difficult to construct efficient automatic synthesis procedures. These difficulties have been eliminated at least partially in so-called balanced decomposition (Łuba, Selvaraj, Nowicka & Kraśniewski, 1995; Nowicka, Łuba & Rawski, 1999).

BASIC THEORY

In this chapter, only such information necessary for an understanding of this chapter is reviewed. More detailed description of functional decomposition based on partition calculus can be found in Brzozowski and Łuba (2003).

Cube Representation of Boolean Functions

A Boolean function can be specified using the concept of cubes (e.g., input terms, patterns) representing some specific subsets of minterms. In a minterm, each input variable position has a well-specified value. In a cube, positions of some input variables can remain unspecified, and they represent "any value" or "don't care" (–). A cube may be interpreted as a p-dimensional subspace of the n-dimensional Boolean space or as a product of $n–p$ variables in Boolean algebra (p denotes the number of components that are "–"). Boolean functions are typically represented by truth tables. A truth table description of a function using minterms requires 2^n rows for a function of n variables. For function from Table 1, a truth

table with $2^6 = 64$ rows would be required. Since a cube represents a set of minterms, application of cubes allows for much more compact description in comparison with minterm representation. For example, cube 0101–0 from row 1 of the truth table from Table 1 represents a set of two minterms {010100, 010110}.

For pairs of cubes and for a certain input subset B, we define the compatibility relation COM as follows: each two cubes S and T are compatible (i.e., $S, T \in \mathrm{COM}(B)$) if and only if $x(S) \sim x(T)$ for every $x \subseteq B$. The compatibility relation ~ on {0, –, 1) is defined as follows: 0 ~ 0, – ~ –, 1 ~ 1, 0 ~ –, 1 ~ –, – ~ 0, – ~ 1, but the pairs (1, 0) and (0, 1) are not related by ~. The compatibility relation on cubes is reflexive and symmetric, but not necessarily transitive. In general, it generates a "partition" with nondisjoint blocks on the set of cubes representing a certain Boolean function F. The cubes contained in a block of the "partition" are all compatible with each other.

"Partitions" with nondisjoint blocks are referred to as blankets (Brzozowski & Łuba, 2003). The concept of blanket is a simple extension of ordinary partition, and typical operations on blankets are strictly analogous to those used in ordinary partition algebra.

Representation and Analysis of Boolean Functions with Blankets

A blanket on a set S is such a collection of (not necessarily disjoint) subsets B_i of S, called blocks, that:

$$\bigcup_i B_i = S$$

The product of two blankets β_1 and β_2 is defined as follows:

$$\beta_1 \bullet \beta_2 = \{ B_i \cap B_j \mid B_i \in \beta_1 \text{ and } B_j \in \beta_2 \}$$

For two blankets we write $\beta_1 \leq \beta_2$ if and only if for each B_i in β_1 there exists a B_j in β_2 such that $B_i \subseteq B_j$. The relation ≤ is reflexive and transitive.

Example 1: Blanket-Based Representation of Boolean Functions

For function F from Table 1, the blankets induced by particular input and output variables on the set of function F's input patterns (cubes) are as follows:

$$\beta_{x1} = \{\overline{1, 2, 3, 4, 5, 6, 8, 9}; \overline{3, 6, 7, 9, 10}\},$$
$$\beta_{x2} = \{\overline{5, 6, 7, 8, 10}; \overline{1, 2, 3, 4, 6, 7, 9, 10}\},$$
$$\beta_{x3} = \{\overline{1, 2, 3, 4, 8, 9}; \overline{5, 6, 7, 8, 10}\},$$
$$\beta_{x4} = \{\overline{2, 3, 5, 8, 9, 10}; \overline{1, 2, 4, 5, 6, 7, 8}\},$$
$$\beta_{x5} = \{\overline{1, 2, 3, 5, 6, 7, 8, 10}; \overline{1, 4, 5, 6, 8, 9, 10}\},$$
$$\beta_{x6} = \{\overline{1, 2, 3, 4, 7, 8, 9, 10}; \overline{3, 4, 5, 6, 7, 9, 10}\},$$
$$\beta_{y1} = \{\overline{1, 2, 3, 4, 5, 6, 7}; \overline{8, 9, 10}\},$$

The product of two blankets β_1 and β_2:

$$\beta_{x2x4} = \beta_{x2} \bullet \beta_{x4} =$$
$$\{\overline{5, 8, 10}; \overline{5, 6, 7, 8}; \overline{2, 3, 9, 10}; \overline{1, 2, 4, 6, 7}\},$$
$$\beta_{x2x4} \leq \beta_{x2}.$$

Information on the input patterns of a certain function F is delivered by the function's inputs and

Table 1. Boolean function $F(x_1, x_2, x_3, x_4, x_5, x_6)$

	x_1	x_2	x_3	x_4	x_5	x_6	y_1
1	0	1	0	1	–	0	0
2	0	1	0	–	0	0	0
3	–	1	0	0	0	–	0
4	0	1	0	1	1	–	0
5	0	0	1	–	–	1	0
6	–	–	1	1	–	1	0
7	1	–	1	1	0	–	0
8	0	0	–	–	–	0	1
9	–	1	0	0	1	–	1
10	1	–	1	0	–	–	1

used by its outputs with respect to the blocks of the input and output blankets. Knowing the block of a certain blanket, one is able to distinguish the elements of this block from all other elements, but is unable to distinguish between elements of the given block. In this way, information in various points and streams of discrete information systems can be modeled using blankets.

Serial Decomposition

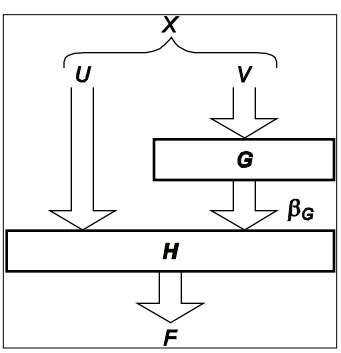
The set X of a function's input variable is partitioned into two subsets: *free variables* U and *bound variables* V, such that $U \cup V = X$. Assume that the input variables $x_1,...,x_n$ have been relabeled in such way that:

$U = \{x_1,...,x_r\}$ and
$V = \{x_{m-s+1},...,x_n\}$.

Consequently, for an n-tuple x, the first r components are denoted by x^U, and the last s components, by x^V.

Let F be a Boolean function, with $n > 0$ inputs and $m > 0$ outputs, and let (U, V) be as previously indicated. Assume that F is specified by a set $\mathcal{F}$ of the function's cubes. Let G be a function with s inputs and p outputs, and let H be a function with $r + p$ inputs and m outputs. The pair (G, H) represents a serial decomposition of F with respect to (U, V), if for every minterm b relevant to F, $G(b^V)$ is defined, $G(b^V) \in \{0, 1\}^p$, and $F(b) = H(b^U, G(b^V))$. G and H are called blocks of the decomposition (Figure 2).

Figure 2. Schematic representation of the serial decomposition



Theorem 1: Existence of Serial Decomposition (Brzozowski & Łuba, 2003)

Let β_V, β_U, and β_F be blankets induced on the function's F input cubes by the input subsets V and U, and outputs of F, respectively.

If there exists a blanket β_G on the set of function F's input cubes such that $\beta_V \leq \beta_G$, and $\beta_U \bullet \beta_G \leq \beta_F$, then F has a serial decomposition with respect to (U, V).

Proof of Theorem 1 can be found in Brzozowski and Łuba (2003).

As follows from Theorem 1, the main task in constructing a serial decomposition of a function F with given sets U and V is to find a blanket β_G that satisfies the condition of the theorem. Since β_G must be $\geq \beta_V$, it is constructed by merging blocks of β_V as much as possible.

Two blocks B_i and B_j of blanket β_V are compatible (merge able), if blanket γ_{ij} obtained from blanket β_V by merging B_i and B_j into a single block satisfies the second condition of Theorem 1; that is, if $\beta_U \bullet \gamma_{ij} \leq \beta_F$. Otherwise blocks B_i and B_j are incompatible (unmergeable). A subset δ of blocks of the blanket β_V is a compatible class of blocks if the blocks in δ are pairwise compatible. A compatible class is maximal if it is not contained in any other compatible class.

From the computational point of view, finding maximal compatible classes is equivalent to finding maximal cliques in a graph $\Gamma = (N, E)$, where the set N of nodes is the set of blocks of β_V and set E of edges is formed by set of compatible pairs.

The next step in the calculation of β_G is the selection of a set of maximal classes, with minimal cardinality, that covers all the blocks of β_V. The minimal cardinality ensures that the number of

blocks of β_G, and hence the number of outputs of the function G, is as small as possible.

In certain heuristic strategies, both procedures (finding maximal compatible classes and then finding the minimal cover) can be reduced to the graph coloring problem.

Calculating β_G corresponds to finding the minimal number k of colors for graph $\Gamma = (N, E)$.

Example 2

For the function from Table 1 specified by a set $\mathcal{F}$ of cubes numbered 1 through 10, consider a serial decomposition with $U = \{x_2, x_4, x_5\}$ and $V = \{x_1, x_3, x_6\}$.

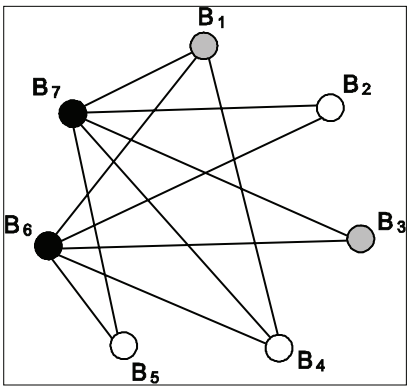
We find:

$$\beta_U = \beta_{x2\,x4\,x5} = \beta_{x2} \bullet \beta_{x4} \bullet \beta_{x5} =$$
$$\{\overline{5, 8, 10}; \overline{5, 6, 7, 8}; \overline{2, 3, 10}; \overline{1, 2, 6, 7}; \overline{9, 10}; \overline{1, 4, 6}\},$$
$$\beta_V = \beta_{x1\,x3\,x6} = \beta_{x1} \bullet \beta_{x3} \bullet \beta_{x6} =$$
$$\{\overline{1, 2, 3, 4, 8, 9}; \overline{3, 4, 9}; \overline{8}; \overline{5, 6}; \overline{3, 9}; \overline{7, 10}; \overline{6, 7, 10}\},$$
$$\beta_F = \beta_{y1} = \{\overline{1, 2, 3, 4, 5, 6, 7}; \overline{8, 9, 10}\},$$

For:

$$\beta_V = \{\overset{B_1}{\overline{1, 2, 3, 4, 8, 9}}; \overset{B_2}{\overline{3, 4, 9}}; \overset{B_3}{\overline{8}}; \overset{B_4}{\overline{5, 6}}; \overset{B_5}{\overline{3, 9}}; \overset{B_6}{\overline{7, 10}}; \overset{B_7}{\overline{6, 7, 10}}\}$$

Figure 3. Incompatibility graph of β_V's blocks



the following are the unmergeable pairs: (B_1, B_4), (B_1, B_6), (B_1, B_7), (B_2, B_6), (B_2, B_7), (B_3, B_4), (B_3, B_6), (B_3, B_7), (B_4, B_6), (B_4, B_7), (B_5, B_6), and (B_5, B_7). Using the graph coloring procedure, we find that three colors are needed here (Figure 3).

Nodes B_1, B_3 are assigned one color, nodes B_2, B_4, B_5 are assigned a second color, and a third color is assigned for nodes B_6, B_7. The sets of nodes assigned to different colors form the blocks of β_G.

$$\beta_G = \{\overline{1, 2, 3, 4, 8, 9}; \overline{3, 4, 5, 6, 9}; \overline{6, 7, 10}\}$$

It is easily verified that β_G satisfies the condition of Theorem 1. Thus, function F has a serial decomposition with respect to (U, V).

Since β_G has 3 blocks, to encode blocks of this blanket, two encoding bits g_1 and g_2 have to be used. Let us assume that we use the encoding:

$$\beta_G = \{\overset{00}{\overline{1, 2, 3, 4, 8, 9}}; \overset{01}{\overline{3, 4, 5, 6, 9}}; \overset{10}{\overline{6, 7, 10}}\}.$$

To define a function G by a set of cubes, we calculate all the cubes, $r(B_i)$, assigned to each block B_i of β_V. The relationship between blocks of β_V and their cube representatives, $r(B_i)$, relies on containment of block B_i in blocks of β_{xj} from $x_j \in V$.

Denoting blocks of β_V from Example 2 as B_1 through B_7, we have $r(B_1) = 000$. This is because $B_1 = \{1, 2, 3, 4, 8, 9\}$ is included in the first blocks of β_{x1}, β_{x3} and β_{x6}. For $B_2 = \{3, 4, 9\}$, we have: B_2 is included in the first block of β_{x1}, in the first block β_{x3} and in both blocks of β_{x6}. Hence, $r(B_2) = 00–$. Similarly, $r(B_3) = 0–0$, $r(B_4) = 011$, $r(B_5) = –0–$, $r(B_6) = 11–$, $r(B_7) = 111$.

Finally, the value of function G is obtained on the basis of containment of blocks B_i in blocks of β_G. Block B_1={1, 2, 3, 4, 8, 9} of blanket β_V is contained in block β_G that has been encoded with 00. Since $r(B_1) = 000$, we have $G(r(B_1)) = G(x_1 = 0, x_3 = 0, x_6 = 0) = 00$. Similarly, $G(r(B_3)) = 00$, $G(r(B_4)) = 01$, $G(r(B_6)) = 10$ and $G(r(B_7)) = 10$. However, block $B_2 = \{3, 4, 9\}$ is contained in two blocks of β_G (one

encoded "00" and the second "01"). The representative "00–" of this block has nonempty product with representative "000" of B_1 and representative "0–0" of B_3, which was assigned output "00." To avoid conflicts, we must subtract cubes "000" and "0–0" from cube "00–." The result is cube "001" that may be assigned output "01." The same applies to block B_5. The representative "–0–" of this block has nonempty product with representative of B_1 and B_3, which was assigned output "00." We must subtract cubes "000" and "0–0" from cube "0–0," and the result in the form of cube "10–" may be assigned output "01".

Truth table of function *G* is presented in Table 2a. To compute the cubes for function H, we consider each block of the product $\beta_U \bullet \beta_G$. Their representatives are calculated in the same fashion. Finally, the outputs of *H* are calculated with respect to β_F (Table 2b).

The process of functional decomposition consists of the following steps:

- Selection of an appropriate input support *V* for block *G* (input variable partitioning)
- Calculation of the blankets β_U, β_V and β_F
- Construction of an appropriate multiblock blanket β_G (corresponds to the construction of the multivalued function of block *G*)
- Creation of the binary functions *H* and *G* by representing the multiblock blanket β_G as the product of a number of certain two-block blankets (equivalent to encoding the multivalued function of block *G* defined by blanket β_G with a number of binary output variables)

In a multilevel decomposition, this process is applied to functions *H* and *G* repetitively, until each block in the obtained network in this way can be mapped directly to a logic block of a specific implementation structure (Łuba & Selvaraj, 1995).

The selection of an appropriate input variable partitioning is the main problem in functional decomposition (Rawski, Jóźwiak & Łuba, 1999a; Rawski, Selvaraj & Morawiecki, 2004). The choice of sets *U* and *V* from set *X* determines the construction of an appropriate blanket β_G, which satisfies Theorem 1. The existence of such a blanket β_G implies the existence of a serial decomposition. Blankets β_V, β_G, $\beta_U \bullet \beta_G$, and β_F constitute the basis for the construction of subfunctions *H* and *G* in serial decomposition. In other words, knowing β_V, β_U, and β_F, and having β_G, one can construct particular subfunctions *G* and *H*.

Table 2a. Function G of the serial decomposition

	x_1	x_3	x_6	g_1	g_2
1	0	0	0	0	0
2	0	0	1	0	1
3	0	–	0	0	0
4	0	1	1	0	1
5	1	0	–	0	1
6	1	1	–	1	0
7	1	1	1	1	0

Table 2b. Function H of the serial decomposition

	x_2	x_4	x_5	g_1	g_2	y_1
1	1	1	–	0	0	0
2	1	–	0	0	0	0
3	1	0	0	0	0	0
4	1	0	0	0	1	0
5	1	1	1	0	0	0
6	1	1	1	0	1	0
7	0	1	1	0	1	0
8	–	1	–	0	1	0
9	–	1	–	1	0	0
10	–	1	0	1	0	0
11	0	–	–	0	0	1
12	1	0	1	0	0	1
13	1	0	1	0	1	1
14	–	0	–	1	0	1

The input variables of block G and their corresponding blankets, and the output blanket β_G of block G define together the multivalued function of block G. The structure of β_G obviously influences the shape of the subfunctions G and H (Figure 2). Blanket β_G determines the output values of function G. Each value of this multivalued function corresponds to a certain block of the blanket β_G. Considering the number of values of the multivalued function of a certain subsystem in decomposition is therefore equivalent to considering the number of blocks in blanket β_G of this subsystem. A minimum of $\lceil \log_2 q \rceil$ binary variables is required for encoding q values. Thus, if q denotes the number of blocks in β_G, then the minimum required number of binary outputs from G is equal to $k = \lceil \log_2 q \rceil$.

Since function H is constructed by substituting in the truth table of function F the patterns of values of the primary input variables from set V (bound variables) with the corresponding values of function G, it is obvious that the choice of β_G influences the subfunction H. The outputs of G constitute a part of the input support for block H. Thus, the size of block G and the size of block H both grow with the number of blocks in blanket β_G. The minimum possible number of blocks in β_G strongly depends on the input support chosen for block G, because β_G is computed by merging some blocks of β_V, this being the blanket induced by the chosen support.

Function H is decomposed in the successive steps of the multilevel synthesis process. This is why blanket β_G has a direct influence on the next steps of the process. The structure of the blanket β_G determines the difficulty of the successive decomposition steps and influences the final result of the synthesis process (characterized by the number of logic blocks and number of logic levels). The number of blocks in blanket β_G is the most decisive parameter. The strong correlation of the number of blanket β_G's blocks with the decomposition's quality has been shown in Rawski, Jóźwiak, and Łuba (1999b), and this number can be used as a criterion for testing individual solutions.

In multilevel logic synthesis methods, the serial decomposition process is applied recursively to functions H and G obtained in the previous synthesis steps until each block of the resulting net can be mapped directly to a single logic block of a specific implementation structure (Łuba, 1995; Łuba & Selvaraj, 1995). In the case of look-up table FPGAs, the multilevel decomposition process ends when each block of the resulting net can be mapped directly into a configurable logic block (CLB) of a specific size (typically the CLB size is from 4 to 6 inputs and 1 or 2 outputs). Although algorithms of multilevel logic synthesis can also use parallel decomposition in order to assist the serial decomposition (Łuba et al., 1995), the final results of the synthesis process strongly depend on the quality of the serial decomposition.

Parallel Decomposition

Consider a multiple-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. 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.

Example 3

Consider the multiple-output function given in Table 3. The minimal sets of input variables on which each output of F depends are:

y_1: $\{x_1, x_2, x_6\}$
y_2: $\{x_3, x_4\}$
y_3: $\{x_1, x_2, x_4, x_5, x_9\}$, $\{x_1, x_2, x_4, x_6, x_9\}$
y_4: $\{x_1, x_2, x_3, x_4, x_7\}$
y_5: $\{x_1, x_2, x_4\}$
y_6: $\{x_1, x_2, x_6, x_9\}$

An optimal two-block decomposition, minimizing the card X_G + card X_H (where card X is the cardinality of X), is $Y_G = \{y_2, y_4, y_5\}$ and $Y_H = \{y_1, y_3, y_6\}$, with $X_G = \{x_1, x_2, x_3, x_4, x_7\}$ and $X_H = \{x_1, x_2, x_4, x_6, x_9\}$. The truth tables for components G and H are shown in Table 4.

The algorithm itself is general in the sense that the function to be parallel decomposed can be specified in compact cube notation. Calculation of the minimal sets of input variables for each individual output can be a complex task. Thus, in practical implementation, heuristic algorithms are used, which support calculations with the help of so-called indiscernible variables.

Table 3. Function F

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	y_1	y_2	y_3	y_4	y_5	y_6
1	0	0	0	1	1	1	0	0	0	0	0	0	0	–	0
2	1	0	1	0	0	0	0	0	0	0	0	–	1	0	1
3	1	0	1	1	1	0	0	0	0	0	1	1	0	1	1
4	1	1	1	1	0	1	0	0	0	0	1	1	1	1	0
5	1	0	1	0	1	0	0	0	0	0	0	0	–	0	1
6	0	0	1	1	1	0	0	0	0	1	1	0	1	0	0
7	1	1	1	0	0	0	0	0	0	1	0	–	0	1	0
8	1	0	1	1	0	1	0	0	0	1	1	0	0	–	1
9	1	0	1	1	0	1	1	0	0	–	1	0	1	–	1
10	1	1	1	0	0	0	0	1	0	1	0	1	0	1	–
11	0	0	0	1	1	1	0	0	1	0	0	1	0	–	1
12	0	0	0	1	1	0	0	0	1	–	–	1	0	0	0

Table 4a. Function G of parallel decomposition

	x_1	x_2	x_3	x_4	x_7	y_2	y_4	y_5
1	0	0	0	1	0	0	0	0
2	1	0	1	0	0	0	1	0
3	1	0	1	1	0	1	0	1
4	1	1	1	1	0	1	1	1
5	0	0	1	1	0	1	1	0
6	1	1	1	0	0	0	0	1
7	1	0	1	1	1	–	1	–

Table 4b. Function H of parallel decomposition

	x_1	x_2	x_4	x_6	x_9	y_1	y_3	y_6
1	0	0	1	1	0	0	0	0
2	1	0	0	0	0	0	0	1
3	1	0	1	0	0	0	1	1
4	1	1	1	1	0	0	1	0
5	0	0	1	0	0	1	0	0
6	1	1	0	0	0	1	1	0
7	1	0	1	1	0	1	0	1
8	0	0	1	1	1	0	1	1
9	0	0	1	0	1	–	1	0

Balanced Functional Decomposition

The balanced decomposition is an iterative process in which, at each step, either parallel or serial decomposition of a selected component is performed. The process is carried out until all resulting subfunctions are small enough to fit blocks with a given number of input variables.

Example 4

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 5, for which cells with four inputs and one output are assumed (this is the size of Altera's FLEX FPGAs).

As *F* is a 10-input, two-output function, in the first step of the decomposition, particularly in automated mode, serial decomposition is performed. The algorithm extracts function *g* with inputs numbered 1, 3, 4, and 6; thus, the next step deals with seven-input function *H*, for which again serial decomposition is assumed, now resulting in block *G*, with four inputs and two outputs (implemented by two cells). It is worth noting that the obtained block *G* takes as its inputs variables denoted 0, 2, 5, and 7, which, fortunately, belong to primary variables, and therefore the number of levels is not increased in this step as it is shown in Figure 4a. In the next step, we apply parallel decomposition. Parallel decomposition generates two components, both with one output but four and five inputs, respectively. The first one forms a cell (Figure 4b). The second component is subject to two-stage serial decomposition as shown in Figure 4c. The obtained network can be built of seven (four to one) cells, where the number of levels in the critical path is three.

The same function decomposed with parallel decomposition, the first step shown in Figure 5, leads to a completely different structure. Parallel decomposition applied directly to function *F* generates two components, both with six 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 (Figure 5a). The second component can be decomposed serially as well; however, with the number of outputs of the extracted block, *G* equals two. Therefore,

Table 5.

```
type fr
.i 10
.o  2
.p 25
0101000000   00
1110100100   00
0010110000   10
0101001000   10
1110101101   01
0100010101   01
1100010001   00
0011101110   01
0001001110   01
0110000110   01
1110110010   10
0111100000   00
0100011011   00
0010111010   01
0110001110   00
0110110111   11
0001001011   11
1110001110   10
0011001011   10
0010011010   01
1010110010   00
0100110101   11
0001111010   00
1101100100   10
1001110111   11
.e
```

Figure 4. Decomposition of function F obtained with a strategy, where serial decomposition is performed at first

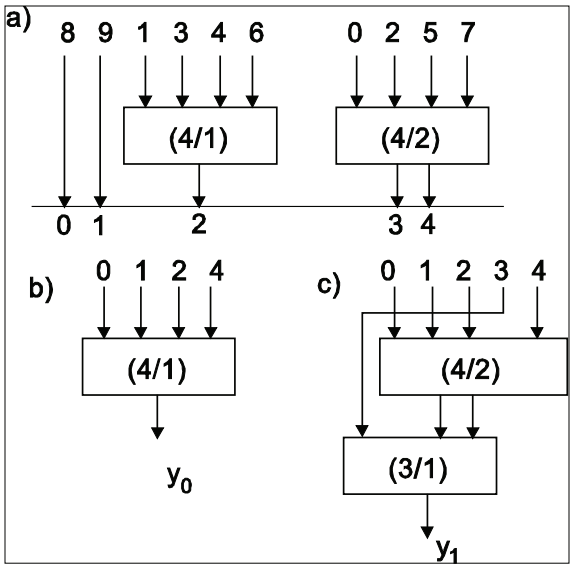
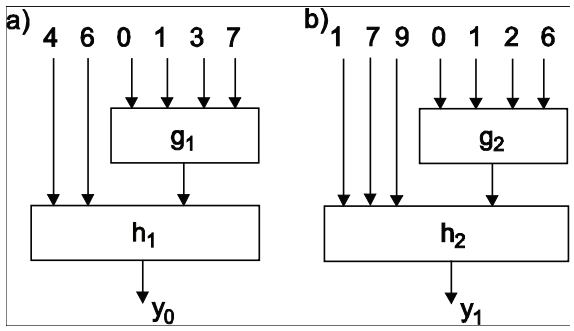


Figure 5. Decomposition of function F with a strategy, where parallel decomposition is performed at first



to minimize the total number of components, a nondisjoint decomposition strategy can be applied. The truth tables of the decomposed functions g_1, h_1, g_2, h_2, are shown in Table 6. The columns in the tables denote variables in the order shown in Figure 5; for example, the first left-hand side column in Table 6b denotes variable numbered 4, the second variable numbered 6, and the third denotes variable g_1. Such a considerable impact on the structure results from the fact that the parallel decomposition simultaneously reduces the number of inputs to both the resulting components, leading to an additional improvement in the final representation.

The idea of intertwining parallel and serial decomposition has been implemented in a program called DEMAIN. DEMAIN has two modes: automatic and interactive. It can also be used for the reduction of the number of inputs of a function when an output depends on only a subset of the inputs. From this point of view, DEMAIN is a tool specially dedicated to FPGA-oriented technology mapping.

Given a function F with n inputs and m outputs, and a logic cell (LC) with C_{in} inputs and C_{out} outputs, a decomposition process is carried out by the following steps:

1. If $n \leq m$, use parallel decomposition. Continue iteratively for each of the obtained components.
2. If $n > m$, try disjoint serial decomposition with the number of block G inputs equal to the number C_{in} of LC inputs, and the number of block G outputs equal to the number C_{out} of LC outputs. If such a serial decomposition is found, find the corresponding H. Continue iteratively with $F = H$. Otherwise, try to find G with fewer inputs than C_{in} and/or fewer outputs than C_{out}. If such a G is found, find H. Continue iteratively with $F = H$. In case a G that fits in one cell cannot be found, try a larger G. This step is repeated until decomposition with a function G larger than the cell exists. Find H and continue with $F = H$. Function G will have to be decomposed later.

The decomposition is carried out until all resulting subfunctions are small enough to fit into logic cells available in the assumed implementation technology.

Table 6.

a) function g_1	b) function h_1
0110 1	-01 0
1101 1	011 1
1000 1	111 0
0010 1	100 1
0000 0	0-0 0
0101 0	110 0
1100 0	
0100 0	
0011 0	
1011 0	
1111 0	
c) function g_2	**d) function h_2**
0110 1	10-1 0
0011 1	-101 1
0100 1	-111 1
1000 1	0011 0
0101 1	0001 1
1100 0	1-00 0
0010 0	0000 0
1010 0	1110 1
1110 0	1010 0
0001 0	0100 1
0111 0	0010 1
1111 0	

DIGITAL FILTERS

Digital filters are typically used to modify the attributes of a signal in the time or frequency domain through a process called linear convolution (Meyer-Baese, 2004). This process is formally described by the following formula:

$$y[n] = x[n] * f[n] = \sum_k x[k] \cdot f[n-k] = \sum_k x[k] \cdot c[k] \quad (1)$$

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). As the names imply, an FIR filter consists of a finite number of sample values, reducing the previously presented convolution to a finite sum per output sample. An IIR filter requires that an infinite sum has to be performed. In this chapter, 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:

$$y[n] = \sum_{k=0}^{L-1} x[k] \cdot c[k] \quad (2)$$

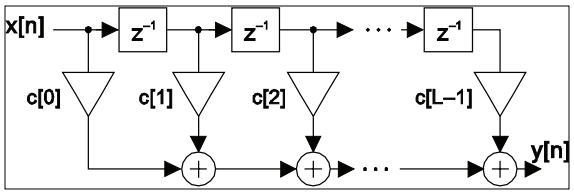
The L-th order LTI FIR filter is schematically presented in Figure 6. It consists of a collection of delay line, adders, and multipliers.

Much available digital filter software enables very easy computation of coefficients for a given filter. However, the challenge is mapping the FIR structure into suitable architecture. Digital filters are typically implemented as multiply-accumulate (MAC) algorithms with the use of special DSP devices.

Efficient hardware implementation of a filter's structure in programmable devices is possible by optimizing the implementation of multipliers and adders. In modern programmable structures, specialized embedded blocks can be used to implement multipliers, increasing the performance of the designed system. Moreover, in the case of Altera's devices, a whole MAC unit can be implemented in embedded DSP block, making the design methodology very similar to the one used in the case of DSP processors.

In the case of programmable devices, however, direct or transposed forms are preferred for maximum speed and lowest resource utilization.

Figure 6. Direct form FIR filter



This is because the approach enables exploitation of prevalent parallelism in the algorithm.

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

DISTRIBUTED ARITHMETIC METHOD

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

$$x[n] = \sum_{b=0}^{B-1} x_b[n] \cdot 2^b, \quad \text{where } [x_b \;\; n] \in [0,1] \tag{3}$$

The whole convolution sum can be described as shown next:

$$y[n] = \sum_{b=0}^{B-1} 2^b \cdot \sum_{k=0}^{L-1} x_b[k] \cdot c[k] = \sum_{b=0}^{B-1} 2^b \cdot \sum_{k=0}^{L-1} f(x_b[k], c[k]) \tag{4}$$

The efficiency of filter implementation based on this concept strongly depends on the 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 signed DA filter structure is shown in Figure 7, 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 (Meyer-Baese, 2004). The utility programs that generate the look-up tables for filters with given coefficients can be found in the literature.

In the experiments presented in this chapter, a variation of DA architecture has been used. It increases the speed of a filter at the expense of additional LUTs, registers, and adders. The basic DA architecture for computing the length L sum of products accepts one bit from every L input word. The computation speed can be significantly increased by accepting for the computation more bits per word. Maximum speed can be achieved with a fully pipelined parallel architecture, as shown in Figure 8. Such an implementation can outperform all commercially available programmable signal processors.

The HDL specification of the look-up table can be easily obtained for the filter described by its $c[i]$ coefficients. Since the size of look-up tables grows exponentially with the number of inputs, efficient implementation of these blocks becomes crucial to the final resource utilization of filter implementation. Here, advanced synthesis methods based on balanced decomposition can be successfully applied for technology mapping of DA circuits onto FPGA logic cells.

RESULTS

Experimental results for FIR filter implementation with different design methodologies are presented in this section. Filter with length (order) 15 has been chosen for the experiment. It has eight-bit signed input samples, and its coefficients can

Figure 7. DA architecture with look-up table (LUT)

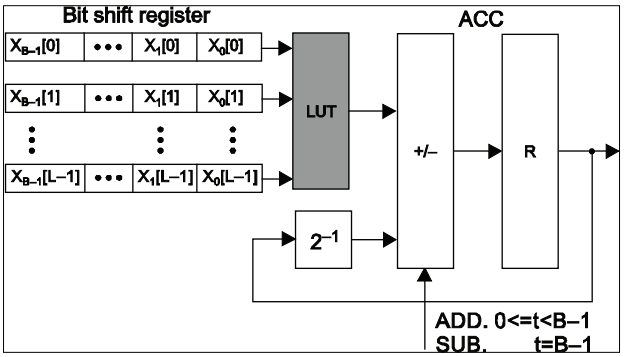
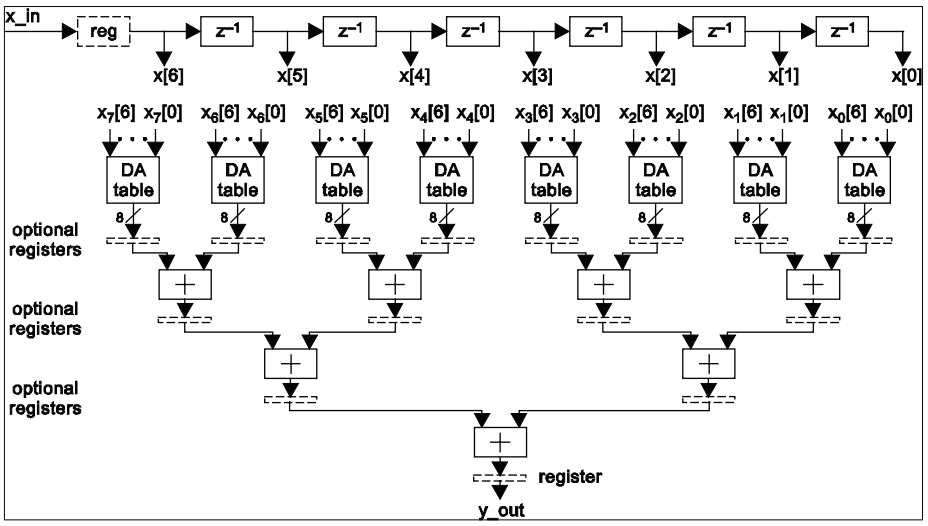


Figure 8. Parallel implementation of a distributed arithmetic scheme



be found in Goodman and Carey (1977). For comparison, the filter has been implemented in Stratix EP1S10F484C5, Cyclone EP1C3T100C6, and CycloneII EP2C5T144C6 structures using Altera QuartusII v5.1 SP0.15.

Table 7 presents the comparison of implementation results for different design methodologies. The column falling under the "MAC" label presents the results obtained by implementing the multiply-and-accumulate strategy with the use of logic cell resources; without utilization the embedded DSP blocks. Multipliers as well as accumulators were implemented in a circuit of logic cells. This implementation, due to its serial character, requires 15 clock cycles to compute the result. It requires a relatively large amount of resources, while delivering the worst performance in comparison to other implementations.

The next column, "MULT block," holds the implementation results of a method similar to "MAC" with a difference where multipliers were implemented in dedicated DSP-embedded blocks. It can be noticed that the performance of the filter increased at the cost of additional resources in the form of DSP-embedded blocks. Results in the column falling under "DSP block" were obtained by implementing the whole MAC unit in the embedded DSP block. Further increase in

Table 7. Implementation results for different design methodologies. Chip: S – Stratix EP1S10F484C5; C – Cyclone EP1C3T100C6, CII – CycloneII EP2C5T144C6 1) DSP blocks are not present in this device family.

Chip		MAC	MULT Block	DSP Block	Parallel	DA	DA Decomposed
S	LC	448	294	225	402	997	567
	DSP	0	2	4	30	0	0
	fmax [MHz]	74.6	85.06	107.30	53.3	65.14	78.75
C	LC	429	436	429	961	997	567
	DSP [1)]	–	–	–	–	–	–
	fmax [MHz]	77.85	79.31	77.85	57.71	72.56	70.87
CII	LC	427	294	275	670	952	567
	DSP	0	2	2	26	0	0
	fmax [MHz]	82.62	97.5	105.59	67.02	78.21	81.46

performance could be noticed, but still 15 clock cycles have to be used to compute the result.

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

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

Finally, results presented in the column "DA decomposed" demonstrate that the application of the DA technique combined with an advanced synthesis method based on balanced decomposition results in a circuit that not only outperforms any other implemented circuit but also reduces the necessary logic resource. The balanced decomposition technique was applied to decompose the combinational blocks of the DA implementation.

In Table 8, the experimental results of Daubechies' dbN, coifN, symN, and 9/7-tap biorthogonal 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 distributed arithmetic concept in the fully parallel way. Balanced decomposition software was also added to increase efficiency of the DA tables' implementations.

Table 8 presents the result for filter implementations using Stratix EP1S10F484C5 device, with a total count of 10,570 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 automatic software was used to initially decompose DA tables, and then the Quartus system was 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.

Table 8. 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	–**	–	31153	–*
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	–**	–	1614	196,85
sym8, a/s low-pass	8	3663	212,72	2249	197,94
sym12, a/s low-pass	12	–**	–	2313	198,61
sym14, a/s low-pass	14	–**	–	2345	200,24
sym16, a/s low-pass	16	–**	–	2377	206,83

** does not fit in EP1S10F484C5*

*** too long compilation time (more than 24 hours)*

CONCLUSION

The modern programmable structures deliver the possibilities to implement DSP algorithms in dedicated embedded blocks. This makes designing of such an 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 cannot 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 profit 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 chapter 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 look-up table FPGA structure. This chapter shows that it is possible to apply the balanced decomposition method for the synthesis of FPGA-based circuits directed toward area or delay optimization.

ACKNOWLEDGMENT

This work is 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

Brislawn, C.M., Bradley, C.B.J., Onyshczak, R., & Hopper, T. (1996). The FBI compression standard for digitized fingerprint images. *Proceedings of the SPIE Conference 2847*, Denver, Colorado, 344–355.

Brzozowski, J.A., & Łuba, T. (2003). Decomposition of Boolean functions specified by cubes. *Journal of Multiple-Valued Logic and Soft Computing*, *9*, 377–417.

Chang, S.C., Marek-Sadowska, M., & Hwang, T.T. (1996). Technology mapping for TLU FPGAs based on decomposition of binary decision diagrams. *IEEE Trans on CAD, 15*(10), 1226–1236.

Croisier, A., Esteban, D., Levilion, M., & Rizo, V. (1973). *Digital filter for PCM encoded signals*. US Patent No. 3777130.

Daubechies, I. (1992). *Ten lectures on wavelets*. SIAM.

Falkowski, B.J. (2004). Compact representation of logic functions for lossless compression of grey scale images. *IEEE Proceedings, Computers and Digital Techniques, 151*(3), 221–230).

Falkowski, B.J., & Chang, C.H. (1997). Forward and inverse transformations between Haar spectra and ordered binary decision diagrams of Boolean functions. *IEEE Trans on Computers, 46*(11), 1271–1279.

Goodman, D.J., & Carey, M.J. (1977). Nine digital filters for decimation and interpolation. *IEEE Trans on Acoustics, Speech and Signal Processing, 25*(2), 121–126.

Lapsley, P., Bier, J., Shoham, A., & Lee, E. (1997). DSP processor fundamentals. New York: IEEE Press.

Lee, E. (1988). Programmable DSP architectures: Part I. *IEEE Transactions on Acoustics, Speech and Signal Processing Magazine*, 4–19.

Lee, E. (1989). Programmable DSP architectures: Part II. *IEEE Transactions on Acoustics, Speech and Signal Processing Magazine*, 4–14.

Łuba, T. (1995). Decomposition of multiple-valued functions. *Proceedings of the 25th International Symposium on Multiple-Valued Logic*, Bloomington, Indiana, (pp. 256–261).

Łuba, T., & Selvaraj, H. (1995). A general approach to Boolean function decomposition and its applications in FPGA-based synthesis. *VLSI Design, Special Issue on Decompositions in VLSI Design, 3*(3-4), 289–300.

Łuba, T., Selvaraj, H., Nowicka, M., & Kraśniewski, A. (1995). Balanced multilevel decomposition and its applications in FPGA-based synthesis. In G. Saucier, & A. Mignotte (Eds.), *Logic and architecture synthesis.* Chapman & Hall.

Meyer-Baese, U. (2004). *Digital signal processing with field programmable gate arrays. Second edition*. Berlin: Springer Verlag.

Nowicka, M., Łuba, T., & Rawski, M. (1999). FPGA-based decomposition of Boolean functions: Algorithms and implementation. *Proceedings of the Sixth International Conference on Advanced Computer Systems*, Szczecin, Poland, 502–509.

Peled, A., & Liu, B. (1974). A new realization of digital filters. *IEEE Transactions on Acoustics, Speech and Signal Processing, 22*(6), 456–462.

Rao, R.M., & Bopardikar, A.S. (1998). *Wavelet transform: Introduction to theory and applications*. Addison-Wesley.

Rawski, M., Jóźwiak, L., & Łuba T. (1999a). Efficient input support selection for sub-functions in functional decomposition based on information relationship measures. *Proceedings of the EUROMICRO'99 Conference*, Milan, Italy.

Rawski, M., Jóźwiak, L., & Łuba, T. (1999b). The influence of the number of values in sub-functions on the effectiveness and efficiency of the functional decomposition. *Proceedings of the EUROMICRO'99 Conference*, Milan, Italy.

Rawski, M., Jóźwiak, L., & Łuba T. (2001). Functional decomposition with an efficient input support selection for sub-functions based on information relationship measures. *Journal of Systems Architecture, 47*, 137–155.

Rawski, M., Selvaraj, H., & Morawiecki, P. (2004). Efficient method of input variable partitioning in functional decomposition based on evolutionary algorithms. *Proceedings of the DSD 2004 Euromicro Symposium on Digital System Design, Architectures, Methods and Tools*, Rennes, France, (pp. 136–143).

Rawski, M., Tomaszewicz, P., & Łuba, T. (2004). Logic synthesis importance in FPGA-based designing of information and signal processing systems. *Proceedings of the International Conference on Signal and Electronics Systems*, Poznań, Poland, (pp. 425–428).

Rawski, M., Tomaszewicz, P., Selvaraj, H., & Łuba, T. (2005). Efficient implementation of digital filters with use of advanced synthesis methods targeted FPGA architectures. *Proceedings of the Eighth Euromicro Conference on DIGITAL SYSTEM DESIGN*, Portugal, (pp. 460–466).

Rioul, O., & Vetterli, M. (1991). Wavelets and signal processing. *IEEE Signal Processing Magazine,* 14–38.

Sasao, T., Iguchi, Y., & Suzuki, T. (2005). On LUT cascade realizations of FIR filters. *Proceedings of the Eighth Euromicro Conference on DIGITAL SYSTEM DESIGN*, Portugal, (pp. 467–474).

Scholl, C. (2001). *Functional decomposition with application to FPGA synthesis.* Kluwer Academic Publishers.