TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Teemu Pitkänen

Fast Fourier Transforms on Energy-Efficient Application-Specific Processors

Julkaisu 1240 • Publication 1240

Tampere 2014

Tampereen teknillinen yliopisto. Julkaisu 1240
Tampere University of Technology. Publication 1240

Teemu Pitkänen

Fast Fourier Transforms on Energy-Efficient Application-Specific Processors

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB109, at Tampere University of Technology, on the 19th of September 2014, at 10 am.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2014

ISBN 978-952-15-3350-1 (printed)
ISBN 978-952-15-3426-3 (PDF)
ISSN 1459-2045

ABSTRACT

Many of the current applications used in battery powered devices are from digital signal processing, telecommunication, and multimedia domains. Traditionally application-specific fixed-function circuits have been used in these designs in form of application-specific integrated circuits (ASIC) to reach the required performance and energy-efficiency. The complexity of these applications has increased over the years, thus the design complexity has increased even faster, which implies increased design time. At the same time, there are more and more standards to be supported, thus using optimised fixed-function implementations for all the functions in all the standards is impractical. The non-recurring engineering costs for integrated circuits have also increased significantly, so manufacturers can only afford fewer chip iterations. Although tailoring the circuit for a specific application provides the best performance and/or energy-efficiency, such approach lacks flexibility. E.g., if an error is found after the manufacturing, an expensive chip iteration is required. In addition, new functionalities cannot be added afterwards to support evolution of standards.

Flexibility can be obtained with software based implementation technologies. Unfortunately, general-purpose processors do not provide the energy-efficiency of the fixed-function circuit designs. A useful trade-off between flexibility and performance is implementation based on application-specific processors (ASP) where programmability provides the flexibility and computational resources customised for the given application provide the performance.

In this Thesis, application-specific processors are considered by using fast Fourier transform as the representative algorithm. The architectural template used here is transport triggered architecture (TTA) which resembles very long instruction word machines but the operand execution resembles data flow machines rather than traditional operand triggering. The developed TTA processors exploit inherent parallelism of the application. In addition, several characteristics of the application have been identified and those are exploited by developing customised functional units for speeding up the execution. Several customisations are proposed for the data path of the processor but it is also important to match the memory bandwidth to the computation speed. This calls for a memory organisation supporting parallel memory accesses. The proposed optimisations have been used to improve the energy-efficiency of the processor and experiments show that a programmable solution can have energy-efficiency comparable to fixed-function ASIC designs.

PREFACE

The work presented in this Thesis has been carried out in the Department of Computer Systems at Tampere University of Technology, Finland and in part during my research visit at Massachusetts Institute of Technology, Cambridge, MA, USA.

I would like to express my gratitude to my supervisor Prof. *Jarmo Takala* for his support, motivation, and efforts to make this research work possible. My thesis Reviewers, Prof. *Holger Blume* and Dr. *Seppo Virtanen* deserve my deepest gratitude and thanks from their valuable and insightful comments. Because their constructive comments this Thesis has achieved its final form.

Also, I would like to thank Prof. *Arvind* from MIT to make my exchange visit possible and for his help and guidance in the US. Prof. *Mikko Tiusanen* deserve my thanks for valuable comments and support. I would like to thank also Prof. *Olli Silvén* from the University of Oulu for his encouraging support.

My friends and colleagues deserve also special thanks. Especially, the closest colleagues during the research work of this Thesis, Dr. *Perttu Salmela*, Dr. *Jari Heikkinen*, Dr. *Pekka Jääskeläinen*, Dr. *Jarno Tanskanen*, M.Sc. *Lassi Nurmi*, M.Sc. *Vladimir Guzma*, M.Sc. *Otto Esko*, M.Sc. *Heikki Kultala*, M.Sc. *Tero Partanen*, and M.Sc. *Riku Uusikartano*, thank for many fruitful discussions, observant comments, motivation, helpful assistance, constructive comments, and infectious enthusiasm. In addition, I wish to express special thanks to Prof. Arvind's research group at MIT and in particular Dr. *Nirav Dave* and M.Sc. *Myron King*, for their assistance and creative atmosphere.

Yet, there are still far too many people to name, who deserve thanks and with whom this work has been carried out. All the mentioned and many unmentioned people have had a great influence on this Thesis. Without them this work would not have been possible.

This Thesis was financially supported by the Foundation of Advancement of Technology (TES) and the National Technology Agency of Finland (TEKES). The financial support is appreciated and acknowledged gratefully.

Finally, I express the deepest gratitude to the *Pitkänen* family and *Heidi* for the support and motivation and to all the friends for their influence on this Thesis.

Tampere, July 27, 2014

Teemu Pitkänen

TABLE OF CONTENTS

LIST OF PUBLICATIONS

This Thesis consists of an introductory part and five original publications. In the text, these publications are referred to as [P1] to [P5].

[P1] T. Pitkänen, R. Mäkinen, J. Heikkinen, T. Partanen, and J. Takala, ”Low-Power, High-Performance TTA Processor for 1024-Point Fast Fourier Transform,” in *Embedded Computer Systems: Architectures, Modeling, and Simulation: Proc. 6th Int. Workshop SAMOS VI*, S. Vassiliadis, S. Wong, T.D. Hämäläinen, Eds. Berlin, Germany: Springer-Verlag, 2006, vol. LNCS 4017, pp. 227–236.

[P2] T. Pitkänen, R. Mäkinen, J. Heikkinen, T. Partanen, and J. Takala, ”Transport Triggered Architecture Processor for Mixed-Radix FFT,” in *Proceedings of the Fortieth Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, Oct. 29–Nov. 1, 2006, pp. 84–88.

[P3] T. Pitkänen, T. Partanen, and J. Takala, ”Low-Power Twiddle Factor Unit for FFT Computation,” in *Embedded Computer Systems: Architectures, Modeling, and Simulation: Proc. 7th Int. Workshop SAMOS VII*, S. Vassiliadis, M. Berekovic, T.D. Hämäläinen, Eds. Berlin, Germany: Springer-Verlag, 2007, vol. LNCS 4599, pp. 233–240.

[P4] T. Pitkänen, J. Tanskanen, R. Mäkinen, and J. Takala,”Parallel Memory Architecture for Application-Specific Instruction-Set Processors,” *Journal of Signal Processing Systems*, vol. 57, no. 1, pp. 21–32, October, 2009.

[P5] T. Pitkänen and J. Takala, ”Low-Power Application-Specific Processor for FFT Computations,” *Journal of Signal Processing Systems*, vol. 63, no. 1, pp.165–176, April, 2011.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AGU	Address Generation Unit
ALU	Arithmetic Logic Unit
ASIC	Application-Specific Integrated Circuit
ASIP	Application-Specific Instruction-set Processor
ASP	Application-Specific Processor
CMP	CoMPare machine instruction
CORDIC	COordinate Rotation DIgital Computer
CU	Control Unit
DFT	Discrete Fourier Transform
DSP	Digital Signal Processor
FFT	Fast Fourier Transform
FPGA	Field-Programmable Gate Array
FU	Functional Unit
GE	Gate Equivalent
IDFT	Inverse Discrete Fourier Transform
IFFT	Inverse FFT
IP	Intellectual Property
IR	Immediate Register, long

LD	LoaD
LSU	Load/Store Unit
LU	Logic Unit
LUT	Look-Up Table
OFDM	Orthogonal Frequency Division Multiplexing
OPC	OPeration Code
PC	Program Counter
PE	Processing Element
RAM	Random Access Memory
RF	Register File
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
SFU	Special Functional Unit
SHU	SHift Unit
ST	STore
SUB	SUBtract
TCE	TTA-based Codesign Environment
TTA	Transport Triggered Architecture
VLIW	Very Long Instruction Word
WL	Word Length

LIST OF SYMBOLS

d_i	ith bit in a data sequence
N	size of FFT
$x(n)$	discrete time signal at time instant n
$X(r)$	frequency domain representation of $x(n)$
ω_N	Nth principal root of unity
j	imaginary unit
e	Napier's number
$F2$	radix-2 FFT butterfly
$F4$	radix-4 FFT butterfly
s	index of the processing column in FFT algorithm
a_i	representation of bit i in FFT memory address
B_i	sector i in unit circle
M	the number of memory accesses within single cycle

1. INTRODUCTION

The recent advances in the semiconductor fabrication processes and technologies have allowed more complex integrated circuits to be developed. This has allowed more and more functionality to be integrated into a single chip. Nowadays even entire systems are integrated on a single chip, hence the name System-On-Chip (SoC). The high integration level has resulted in reduction of the size and the cost of digital systems. This has expanded the digital systems to whole new markets. The growth has been especially high in embedded systems where the system is designed for a single purpose, i.e., the system is not used for general-purpose computing. Nowadays the line between embedded and general-purpose systems is blurred and modern embedded devices have more computing power than personal computers of yesterday. As many embedded systems are handheld, or even integrated in clothing, the systems are often limited by constraints on size, weight, battery life, and cost. For such devices the area of the chip is an important cost factor but today even more important is the energy-efficiency. Many of the devices are battery powered and the battery life of the device is dependent on the energy drawn from the battery. Low power consumption is often used as a design criteria when designing energy-efficient systems but design for low-power does not necessary imply that the completed design is energy-efficient. A low-power system may have poor performance, thus the time to complete the task is long. A more efficient system may have higher power consumption but it completes the task faster, thus the energy for completing the task is lower implying better energy-efficiency. The design must also fulfil the criteria for performance, i.e., the task is required to be computed in the given time period. The energy-efficiency is defined as the energy consumed for performing the required task. Even the small mobile systems must be able to perform tasks which require high performance. Many design tradeoffs must be made to balance the often conflicting goals of energy-efficiency, low-power, high performance, and small area [6].

To address these design factors application-specific integrated circuits (ASIC) are often used as a part of the system to perform the critical tasks efficiently. As the capabilities of chip manufacturing as well as the the requirements of the applications have increased, the complexity of the designs has exploded, which increases the design time and the risks significantly. One solution for reducing the time to design an ASIC is to use modern design tools, which can be used to convert an algorithm described in a high-level language, e.g., C/C++, to a register-transfer logic (RTL) description, as presented in [9, 83]. There are also modifications developed to the ASIC design flow. For example structured ASICs [93] have predetermined logic layers and the functionality is realised by a routing layer. Such an arrangement lowers the non-recurring engineering (NRE) costs. At the same time, the design time is shortened as there is no need to route all the transistor layers. Although the approach of developing a circuit for one purpose provides potentially the best performance as the design is tailored for the single application, it lacks the flexibility. E.g., if during the development an error is found or specifications change, an expensive and time consuming chip iteration is required.

To address these problems field programmable gate arrays (FPGA) have been developed where the system can be reconfigured if the design is required to be changed leading to lower NRE costs. The main drawback of the FPGA is decreased performance, increased power dissipation, and higher chip cost. Another solution is to use a programmable solution. General-purpose processors can be used but their performance may not be sufficient and, in particular, their energy-efficiency is poor. The best of both worlds can be obtained by combining the programmability with application-specific design, i.e., flexibility and efficiency are obtained by tailoring the processor according to the requirements of the application at hand. Such processors are called application-specific processors (ASP) or application-specific instruction set processors (ASIP). The motivation for the ASIP is to customise the instruction set according to the needs of the given application. In this Thesis, the name ASP is used to refer to processors with a customised instruction set. [12, 25, 63]

It is common that applications of wireless baseband signal processing, multimedia signal processing, on-line monitoring, or sensor signal processing exhibit very similar signal processing characteristics [55]. These application groups are called domains and when a system is designed for an application, it might be designed to support a wider range of applications in this domain. This is called domain-specific design. In a similar fashion, as fixed-function ASICs do not provide flexibility, designs customised for a single application may also show limited flexibility. When increasing the flexibility and supporting several applications from the same domain, i.e., domain-specific design, we obtain an application domain-specific processor (ADSP) [78,94]. In this Thesis, ADSPs are not considered and the thesis will concentrate on ASPs, although similar techniques are used to design ADSPs.

There are several design tools for developing ASPs, e.g., [33, 43, 77]. With the aid of modern ASP toolsets the design time can be significantly lowered and also the risks in the design are lowered, since the result is programmable and can be fixed, if needed, by modifications to the software. The main drawback of ASPs is lower energy-efficiency compared to ASICs. There are several processor architectures, which can be used as a template for customisation. E.g., very long instruction word (VLIW) machines have recently gained considerable popularity especially in digital signal processing (DSP) domain. VLIW has potential for higher performance as it may exploit instruction level parallelism (ILP) in the application by executing operations concurrently in multiple processing elements (PE).

Currently the application area where energy-efficiency and flexibility have great importance is wireless communications. In the future, cognitive radios will require functionality changed fairly often, thus programmable solutions will be preferred. In addition, in the future, almost all communications devices will be battery powered or they scavenge their energy from the environment, thus energy-efficiency is a vital property.

Many recent wireless communication systems are based on orthogonal frequency division multiplexing (OFDM), e.g., IEEE 802.11a/g (WiFi) [41], 802.16 (WiMAX) [42] , 3GPP long term evolution (3G LTE) [1], digital video broadcasting (DVB) [61], digital audio broadcasting (DAB) [48], and ultra wideband (UWB) [67]. OFDM is also used in some wired systems, e.g., very-high-speed digital subscriber line (VDSL). The popular wired internet access asymmetric digital subscriber line (ADSL) [22, 58] is based on VDSL. In OFDM systems [95], digital data is encoded on multiple carrier frequencies. The frequency modulation and demodulation are realised with the aid of one of the most popular tools in DSP, the Fast Fourier transform (FFT). In the transmitter, inverse FFT (IFFT) is used to spread the data over multiple carriers and, in receiver, FFT demodulates information from multiple carriers to time domain as illustrated in Figure 1.

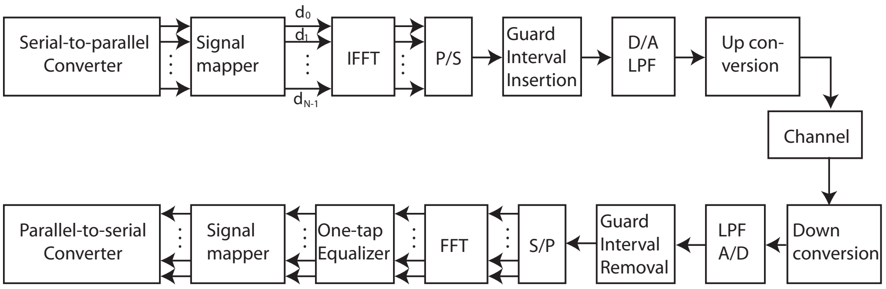

Fig. 1. *Principal block diagram of OFDM system.* $d_0 ... d_{N-1}$ *data bits* 0 *to* $N-1$*; P/S: parallel-to-serial conversion; LPF : low pass filter; D/A: digital-to-analog conversion; A/D: analog-to-digital conversion; S/P : serial-to-parallel conversion.*

The FFT is an efficient method for computing the Discete Fourier Transform (DFT). First, the DFT can be used to determine the frequency spectrum of a signal. This is a direct examination of information encoded in the frequency, phase, and amplitude of the component sinusoids. For example, human speech and hearing use signals with this type of encoding. Second, DFT can be used to determine the frequency response of a system based on the impulse response of the system and *vice versa*. This allows systems to be analysed in the frequency domain, just as convolution allows systems to be analysed in the time domain.Third, the DFT can be used as an intermediate step in more elaborate signal processing techniques.The classic example of this is FFT convolution, an algorithm for convolving signals that is hundreds of times faster than conventional methods [72]. The speedup in previous example is dependent of the convolution size and it will grow with the convolution.

The FFT can be used in various other applications such as sound analysis, magnetic resonance imaging (MRI) [64, 71], compression methods, e.g., in Joint Photographic Experts Group (JPEG) [70, 85] and Moving Picture Experts Group (MPEG) [52, 90], encryption, e.g., in the public key Rivest, Shamir, and Adleman ciphering (RSA) [60, 62], and in Global Positioning System (GPS) [51,91] and Galileo [81,86] acquisition.

1.1 Scope and Objective of Research

The general research problem posed in this Thesis is "how to combine the flexibility and energy-efficiency in application implementation so that the design process is still general and can be supported by tools". Especially, the Thesis considers energy-efficiency of programmable architectures in embedded systems where *a priori* information about the applications is often available. The work considers also scalability, i.e., the resulting methodology must allow designer to trade-off performance against implementation cost. The resulting computing structures must exploit the inherent parallelism in the applications to support scalability. The work does not include reactive general-purpose computations and, therefore, e.g., support for interrupts is not considered.

The solutions proposed in this Thesis are based on the transport triggered architecture (TTA) template.Some earlier work has shown that TTA processors have potential for energy-efficient ASPs [10,38,66]. The main application area in the Thesis is computationally intensive tasks in digital signal processing and, in particular, Fast Fourier transform. The FFT uses complex-valued arithmetic, thus it represents well the current trend in DSP. The Thesis focuses primarily on mixed-radix in-place decimation-in-time (DIT) FFT algorithms and many of the novel contributions related to this algorithm can also be applied to other FFT algorithms.

The objective of the Thesis is to develop effective methods to increase the performance of an ASP in complex-valued DSP computing without major increase in power consumption and still maintaining the flexibility with the aid of programmability. The main claim in this thesis is that a programmable ASP can be designed so that it possesses energy-efficiency comparable to a fixed-function ASIC.

1.2 Main Contributions

In this Thesis, mechanisms are proposed to design energy-efficient programmable solutions for embedded systems. The energy-efficiency is demonstrated with the aid of ASP based on transport triggering paradigm. The customisation is carried out for Fast Fourier transform. Several novel approaches are used for improving the performance for reducing the power consumption of the processor.

A novel energy-efficient coefficient generator is proposed in publication [P3]. Publication [P1] shows that the ASP can perform energy-efficient FFT computation with the aid of hardware supported memory address generation and the exposed data path in the used processor template. Publication [P2] introduces support for variable length FFT; the shown implementation supports all the power-of-two lengths up to 16384. Publication [P4] shows improved energy-efficiency with conflict-free parallel memory scheme for FFT based on two– and four– ported parallel memories. The four–ported parallel memory doubles the throughput while maintaining the energy-efficiency. Publication [P5] presents how the performance can be increased by reducing the software overhead. This improves the energy-efficiency even further.

To summarise, the main contributions are the following:

- novel coefficient generation method for mixed-radix FFT, where the look-up table contains only $\frac{N}{8}+1$ complex-valued coefficients,
- energy-efficient memory organisation for FFT computations based on parallel memory concept,
- energy-efficient operand address generation for mixed-radix FFT, and
- novel processor structure for mixed-radix FFT computations exploiting the exposed data path of transport triggered architecture.

1.3 Author's Contribution

The work presented in this Thesis has been reported in publications [P1–P5]. None of the publications [P1–P5] have yet been used in another academic thesis. The author has been the first author on all of the publications [P1–P5].

The publication [P3] describes the twiddle factor generator for mixed-radix FFT supporting radix-4 and radix-2 computations. The author proposed the principle of twiddle factor generation for radix–4 FFT and implemented the generator. M.Sc. Tero Partanen developed the initial program code for the algorithm. The author also developed and implemented twiddle factor generation for the mixed-radix FFT. This twiddle factor generation is used in publications [P1,P2,P4,P5].

An application-specific processor tailored for FFT supporting one fixed size $N = 1024$ is proposed in [P1]. M.Sc. Risto Mäkinen developed the initial program code for the FFT algorithm and created the first version of the corresponding assembly code. The author was responsible for developing the hardware implementation of the functional units in the processor and connectivity optimisation by programming the final version of the program with parallel assembly.

In [P2], an application-specific processor tailored for FFT supporting power-of-two lengths up to $N = 16384$ is proposed. The author created the mechanisms to support the various transform lengths and implemented the software and hardware parts of the tailored processor.

The author created the hardware implementation of the parallel memory scheme in [P4] where the access scheme for two ported access was developed by Dr. Jarno Tanskanen. The access scheme for the four ported parallel memory was created by the author and the author implemented the required hardware and software parts for the processor.

In [P5], the author developed mechanisms to reduce the software overhead and to speed up the computation. The author provided the required modifications to the hardware units. Finally, the author has carried out the verification and analysis of the developed structures described in [P1–P5].

1.4 Thesis Outline

This Thesis consists of an introductory part where different FFT algorithms are described in Chapter 2. Chapter 3 describes the coefficient generation for the FFT where three different methods for coefficient generation are described. Chapter 4 describes fixed-function FFT processors, programmable FFT processors, and fundamentals of the ASP template used in this work, i.e., Transport Triggered Architecture (TTA). Finally, Chapter 5 concludes the introductory part of the Thesis. This is followed by the second part of the Thesis containing the five original publications.

2. FAST FOURIER TRANSFORM

Discrete Fourier transform (DFT) is the main tool in the class of discrete trigonometric transforms. However, the definition of DFT contains redundancy and several methods have been proposed to avoid the redundant computations. Any method for computing DFT with lower arithmetic complexity than DFT is called Fast Fourier transform (FFT). In this chapter, the most popular FFT methods are discussed. Section 2.1 the describes Discrete Fourier transform and gives an introduction to Fast Fourier transform based on Cooley-Tukey decomposition. In sections 2.2, 2.3, and 2.4, different approaches for exploiting the Cooley-Tukey principle are discussed. Section 2.5 describes properties of FFT algorithms, e.g., in-place computations, which reduces the requirements for memory and, therefore, the energy consumption of the memory. Section 2.6 describes how the operand data can be fetched from the memory and how the storage is efficiently used, i.e., the possibility to use multiple single port memories.

2.1 Discrete Fourier Transform and its Fast Algorithms

Discrete Fourier transform is used to convert a finite sequence of equally-spaced samples to a sequence of coefficients of a finite combination of complex sinusoids. In other words, the time domain representation of an N-point discrete time signal $x(n)$ is converted to frequency domain representation $X(r)$ as follows [56]

$$X(r) = \sum_{n=0}^{N-1} x(n) W_N^{rn},\ r = 0, 1, \cdots, N-1 \tag{1}$$

where the coefficients W_N are defined as

$$W_N = e^{-j2\pi/N} = \cos(2\pi/N) - j\sin(2\pi/N) \tag{2}$$

where j denotes the imaginary unit. As the coefficients W_N are composed of sine and cosine functions, the coefficients W_N^{rn} have symmetry and periodicity properties, which imply that the DFT defined in (1) contains redundancy. By exploiting the underlying properties of the coefficients W_N^{rn} several fast algorithms for DFT have been developed over the years. In general, an algorithm, which has lower arithmetic complexity than DFT, is called a Fast Fourier transform (FFT). The most popular FFT is the Cooley-Tukey algorithm [23] where divide-and conquer paradigm [73] is used to decompose DFT into a set of smaller DFTs. In particular, the Cooley-Tukey principle states that a DFT of length $N = PQ$ can be computed with the aid of P-point DFT and Q-point DFT.

2.2 Radix-p Algorithms

If a factor N, like P or Q above, is not a prime, the previous process can be recursively applied and the larger DFT will be computed with the aid of several smaller DFTs. Especially, when the DFT length is a power of a prime, i.e., $N = p^q$, then the N-point DFT can be computed with the aid of p-point DFTs constructed in q computing stages. As the resulting fast algorithm contains only p-point DFTs, it's is called a radix-p FFT. The most popular approach is radix-2 FFT algorithm where the DFT is decomposed recursively until the entire algorithm is computed with the aid of 2-point DFTs as follows:

$$\begin{aligned} X(r) &= \sum_{n=0}^{\frac{N}{2}-1} x(2n)W_N^{nr} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)W_N^{nr+1} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x(2n)W_{\frac{N}{2}}^{nr} + W_N^r \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)W_{\frac{N}{2}}^{nr},\ r = 0,1,\cdots,N-1. \end{aligned} \tag{3}$$

This equation shows that the coefficients W_N^{nr}, commonly called *twiddle factors*, serve as adjustments in the process of converting small radix DFTs into longer transforms [56].

There are two principal approaches to decomposing the DFT: Decimation-In-Time (DIT) [65] and Decimation-In-Frequency (DIF) [15]. An example of both approaches is illustrated in Figure 2 where signal flow graphs of 8-point FFT derived with both approaches are shown. In the 8-point transform, the computations are carried out in three computing stages where each column contains four 2-point DFTs. The principal computation building block in the radix-2 FFT is 2-point DFT in (3), which is

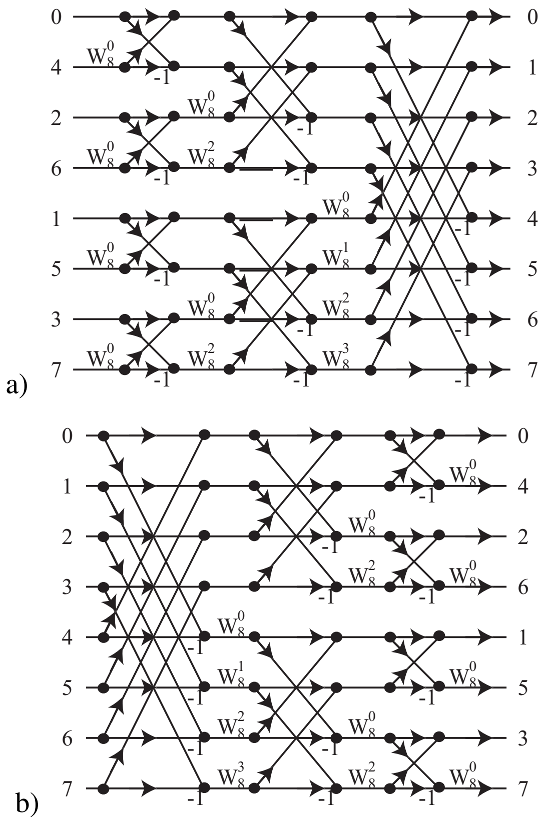

Fig. 2. *Signal flow graphs of 8-point radix-2 FFT: a) decimation-in-time and b) decimation-in-frequency algorithm.*

also called a radix-2 butterfly. In the Figure 2, notation -1 is used to represent subtraction side of the radix-2 butterfly and notation W_N^r shows required coefficient. The Cooley-Tukey butterfly [23], shown in Figure 3(a) is used in DIT algorithms while the Gentleman-Sande butterfly [32] shown in Figure 3(b) is used in DIF algorithms.

This work concentrates on the DIT approach but both approaches result in the same arithmetic complexity and the main difference is the indexing in operand accesses for the FFT butterfly and the order of the computations inside the FFT butterfly. In fixed-point implementations, DIT and DIF approach may result in different quantisation noise performance. However, the differences seem to be small and, according to [13], DIT provides clearly better SNR in radix-2 algorithms.

In the Cooley-Tukey butterfly, one complex multiplication and two complex additions are needed, each stage contains $N/2$ butterflies, and the number of the stages is

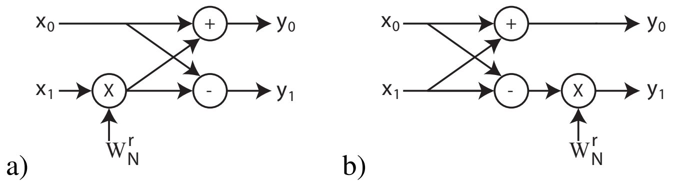

Fig. 3. FFT butterflies according to a) radix-2 DIT algorithm in Figure 2(a) and b) radix-2 DIF algorithm in Figure 2(b).

$\log_2 N$, which gives the total of $\frac{N}{2}\log_2 N$ complex multiplications and $N\log_2 N$ additions for an N-point transform. The complex multiplications use the complex-valued twiddle factors W_N^r as the second input.

2.3 Radix-2^r Algorithms

Traditionally the most popular FFT have been the radix-2 FFTs where computations are based on 2-input, 2-output butterflies depicted in Figure 3. The arithmetic cost of an N-point radix-2 FFT is $(\frac{N}{2}\log_2 N)$ complex multiplications and $(N\log_2 N)$ complex additions. The complexity is often measured in terms of real-valued operations. The complex-valued addition requires two real-valued additions:

$$z_1 + z_2 = (a + jb) + (c + jd) = (a + b) + j(b + d).$$

The complex multiplication in turn requires more operations:

$$z_1 z_2 = (a + jb)(c + jd) = (ac - bd) + j(ad + bc),$$

i.e., four real-valued multiplications and two real-valued additions.

The radix-2 FFT is a special case in the class of radix-2^s FFTs [20]. The arithmetic complexity of FFT can be reduced by using greater than two radix if many of the complex coefficients turn out to be trivial that is (± 1 or $\pm j$). Let us consider the basic equation of the DFT in (1) and divide the original N-point problem to four partial sums by dividing the system to four sub problems where the length of problem

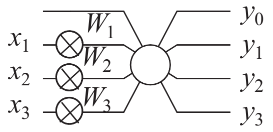

Fig. 4. *Signal flow graph of radix-4 butterfly.*

is $N/4$:

$$\begin{aligned}
X(r) &= \sum_{n=1}^{N-1} x(n)W_N^{rn} \\
&= \sum_{n=0}^{N/4-1} x(4n)W_N^{r(4n)} + \sum_{n=0}^{N/4-1} x(4n+1)W_N^{r(4n+1)} + \sum_{n=0}^{N/4-1} x(4n+2)W_N^{r(4n+2)} \\
&+ \sum_{n=0}^{N/4-1} x(4n+3)W_N^{r(4n+3)}, r = 0,1,\cdots,N-1. \qquad (4)
\end{aligned}$$

This method results in a radix-4 algorithm where computations are based on 4-point DFT. This approach has benefits in terms of arithmetic complexity as 4-point DFT can be computed with trivial coefficients. By combining the twiddle factors, the radix-4 butterfly is defined as:

$$\begin{cases}
y_0 = x_0 + W_1x_1 + W_2x_2 + W_3x_3 \\
y_1 = x_0 - jW_1x_1 - W_2x_2 + jW_3x_3 \\
y_2 = x_0 - W_1x_1 + W_2x_2 - W_3x_3 \\
y_3 = x_0 + jW_1x_1 - W_2x_2 - jW_3x_3
\end{cases} . \qquad (5)$$

This results in arithmetic cost of three complex multiplications and 12 complex additions. The radix-4 butterfly is presented in Figure 4. With the radix-4 algorithm, The computations are carried out in $\log_4(N)$ butterfly stages; e.g., 64-point FFT is completed in three stages while the same transform requires six stages in the radix-2 algorithm. The arithmetic complexity for an N-point radix-4 FFT is $\frac{3N}{4}\log_4 N$ complex multiplications and $3N\log_4 N$ complex additions.

From the implementation point of view, the lower number of arithmetic operations provides potential for faster computation and energy savings. In addition, the latency of computations can be shorter. Besides lower arithmetic complexity, the radix-4 FFT provides also other advantages. The lower number of butterfly computation stages implies that, in memory based systems, less memory accesses are required. This

Table 1. *Arithmetic cost and the number of butterfly stages in different FFT algorithms.*

Radix	# complex multiplications	# complex additions	# stages
2	$\frac{N}{2}\log_2 N$	$N\log_2 N$	$\log_2 N$
4	$\frac{3N}{8}\log_2 N$	$\frac{3N}{2}\log_2 N$	$\frac{1}{2}log_2 N$
8	$\frac{3N}{8}\log_2 N$	$N\log_2 N$	$\frac{1}{3}log_2 N$

speeds up the computations, reduces energy consumption, and relaxes the memory bandwidth requirements.

The arithmetic complexity can further be reduced by selecting on even higher radix. However, in radix-8 and higher, the butterflies will contain non-trivial coefficients and, therefore, the relative arithmetic complexity is not decreasing as much. While radix-8 computations are applicable as they provide some advantages in specific implementation styles, higher radices are seldom used. The arithmetic cost and the number of stages of the different radix systems are presented in Table 1.

The main drawback of the higher radix p is the fact that the length of the transform has to be a power of 2^p, i.e., radix-4 algorithms can only be applied when the transform length is a power of four. When the radix is higher, there are less sequence sizes where the algorithm can be applied. Due to this fact, radix-2 FFT has been popular. It also provides relatively good savings in arithmetic complexity and regularity, which eases the implementation.

2.4 Mixed-Radix FFT

The limitation of radix-2^s FFTs is the fact that it can be applied only to transform lengths that are a power of the radix. This can be overcome by using mixed-radix approach where the DFT decomposition contains several radices, e.g., 32-point FFT can be computed with two radix-4 stages and a single radix-4 stage. An example of mixed-radix FFT is shown in Figure 5 where signal flow graph of a 32-point in-place DIT FFT based on radix-4 and radix-2 is illustrated. The in-place computation saves memory, since each memory block is reused after the butterfly is computed. The memory requirements of the in-place computing is N memory locations. The operand access, i.e., the memory addressing, is discussed in more detail in Section 2.6.

The selection of the used radix is a compromise of the butterfly complexity, arithmetic cost, transform sizes to be supported, system requirements, and the design time of the

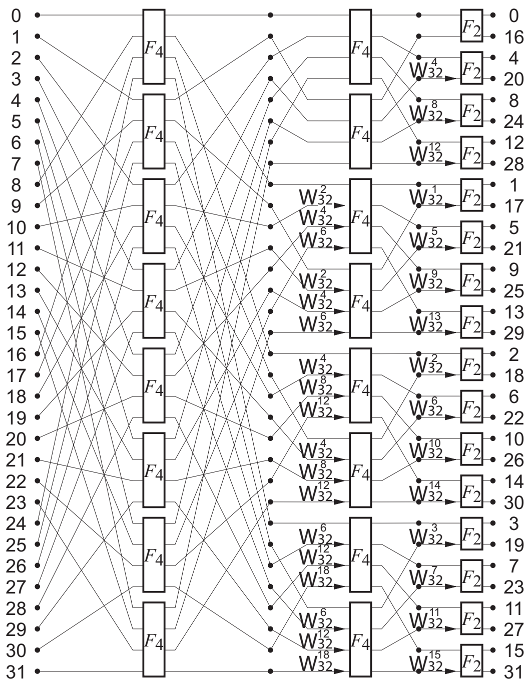

Fig. 5. *Signal flow graph of 32-point mixed-radix DIT in-place FFT algorithm.*

system. In this work, the mixed-radix is selected, which supports all the transform sizes of power of two. When size is not power of four the last stage of the transform is computed with radix-2 algorithm. In the developed system, one radix-4 or two radix-2 butterflies are computed simultaneously, to obtain four input and four output butterflies in either case. The mixed-radix approach is usually used when designed system should support multiple FFT sizes, since the required sizes are not usually all equal to the same power. E.g., if we need to support the IEEE 802.16.1 OFDMA PHY [42], the transform lengths 256 and 2048 need to be supported but radix-4 FFT cannot be used to compute a 2048-point FFT.

2.5 In-Place Computations

The mixed-radix algorithms, radix-2^r FFT algorithms, and radix-p algorithms in general, are block processing algorithms where processing is carried out in processing stages consisting of butterfly computations. This can be seen from the signal flow graphs of the algorithms, e.g., Figure 2 and Figure 5: when input operands are avail-

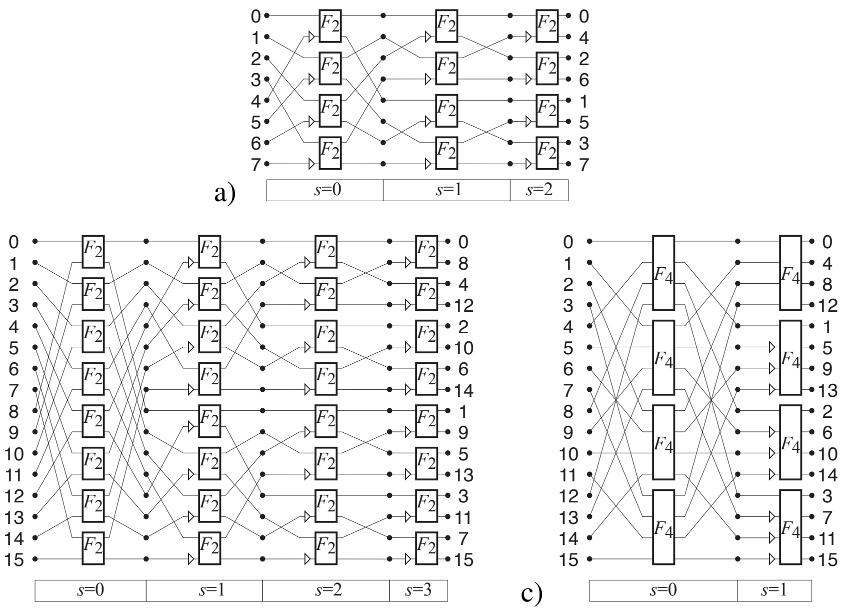

Fig. 6. *Signal flow graphs of in-place DIT FFTs: a) 8-point radix-2, b) 16-point radix-2, and c) 16-point radix-4.*

able the butterfly operations in the first computing stage can be completed, results are stored, and then used as operands for the second butterfly stage. Often in software implementations, double buffering [34] is used, i.e., operands are stored in an array and results are stored to an another array and the role of buffers is exchanged for the next iteration. However, there is no need to reserve additional memory for the computation. Once the operands for a butterfly operations are read, the same memory locations can be used to store the results, i.e., computation can be performed in-place [46]. Exploitation of this property reduces the memory requirements of software implementations significantly.

Real-time FFT is often computed on consecutive blocks in the input data stream but FFT can be computed with overlapped blocks. In such a case, the computation is called sliding FFT (alternatively running FFT) [29]. There are special mechanisms for reducing complexity of sliding FFT computations but this type of computations are not considered in this thesis.

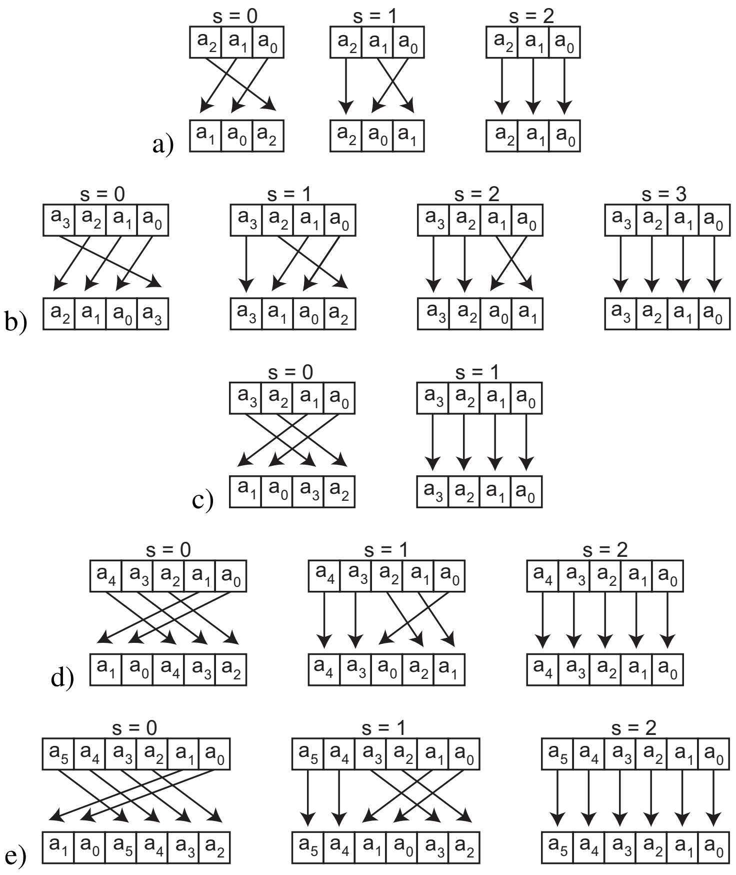

Fig. 7. Operand address generation for in-place FFT: a) 8-point radix-2 FFT at Figure 6a), b) 16-point radix-2 FTT at Figure 6b), c) 16-point radix-4 FFT in Figure 6c), d) 32-point mixed-radix FFT at Figure 5, and e) 64-point radix-4 FFT.

2.6 Permutations and Operand Access

Different FFT algorithms have different operand access patterns. E.g., in Figure 6a), the signal flow graph of an 8-point in-place DIT radix-2 FFT is shown where the access pattern of the operand data for butterfly computations depends on the butterfly column s of the FFT. This implies that results stored in an array from a butterfly stage need to be permuted before processing on the next stage. If the array is stored in memory and the length of array is a power of two, the permutation can be carried out with index manipulation. At bit-level, a linear address $(a_{N-1}, a_{N-2}, \ldots, a_0)$ is rotated to the right but the bit field to be rotated depends on the stage s: the least significant bits $(a_{\log_2 N-s-1}, \ldots, a_0)$ are rotated. The bit-level manipulation is illustrated in Figure

7. The address bit field to be rotated depends clearly on the processing stage s. Figure 7b) and c) show the difference between operand address generation in radix-2 and radix-4 algorithms. In radix-2, one bit rotation to the right is performed while in radix-4, two bits are rotated to the right. Mixed-radix approach is used when the number of bits in the array index is odd, as shown in Figure 7d) where two bit rotation is used for radix-4 butterflies and, in the last stage, no rotation is required for radix-2 butterflies as shown in Figure 5.

In Figure 5, the input is at natural order and the output of the FFT is at bit reversed order. In order to get the output frequencies of the FFT, the output sequences needs to be permuted according to bit reversal. The bit reversal of $N = 2^n$ indexed data is an algorithm that reorders the the data according to a reversing of the bits of the index [31]. E.g., representation of an array $X = (x_0, x_1, \ldots, x_7)$ in bit reversed order is $(x_0, x_4, x_2, \ldots)$ as follows

$$\begin{array}{ll}
x_0 \rightarrow x_0 : 000 \rightarrow 000 & \qquad x_4 \rightarrow x_1 : 100 \rightarrow 001 \\
x_1 \rightarrow x_4 : 001 \rightarrow 100 & \qquad x_5 \rightarrow x_5 : 101 \rightarrow 101 \\
x_2 \rightarrow x_2 : 010 \rightarrow 010 & \qquad x_6 \rightarrow x_3 : 110 \rightarrow 011 \\
x_3 \rightarrow x_6 : 011 \rightarrow 110 & \qquad x_7 \rightarrow x_7 : 111 \rightarrow 111
\end{array}$$

The reversal of bits of the index is clearly seen in the previous example. The previous permutation is related to radix-2 algorithms. In radix-4 algorithms, the same type of reversal in bit-level indices is visible but now the reversal is carried out in 2-bit fields as follows (array with 16 elements):

$$\begin{array}{ll}
x_0 \rightarrow x_0 \ : 0000 \rightarrow 0000 & \qquad x_8 \rightarrow x_1 \ : 1000 \rightarrow 0010 \\
x_1 \rightarrow x_8 \ : 0001 \rightarrow 0100 & \qquad x_9 \rightarrow x_9 \ : 1001 \rightarrow 0110 \\
x_2 \rightarrow x_4 \ : 0010 \rightarrow 1000 & \qquad x_{10} \rightarrow x_5 \ : 1010 \rightarrow 1010 \\
x_3 \rightarrow x_{12} : 0011 \rightarrow 1100 & \qquad x_{11} \rightarrow x_{13} : 1011 \rightarrow 1110 \\
x_4 \rightarrow x_2 \ : 0100 \rightarrow 0001 & \qquad x_{12} \rightarrow x_3 \ : 1100 \rightarrow 0011 \\
x_5 \rightarrow x_{10} : 0101 \rightarrow 0101 & \qquad x_{13} \rightarrow x_{11} : 1101 \rightarrow 0111 \\
x_6 \rightarrow x_6 \ : 0110 \rightarrow 1001 & \qquad x_{14} \rightarrow x_7 \ : 1110 \rightarrow 1011 \\
x_7 \rightarrow x_{14} : 0111 \rightarrow 1101 & \qquad x_{15} \rightarrow x_{15} : 1111 \rightarrow 1111
\end{array}$$

Since bit reversal is invertible, the same permutation applies to sorting out a sequence in bit reversed order as desired for the output frequencies of the FFT.

3. TWIDDLE FACTOR GENERATION FOR FFT

The twiddle factors are an integral part of FFT algorithms and often these coefficients are stored in a lookup table and fetched during computation of the algorithms. While this is a simple and quick method for short transforms, the table size increases super-linearly with the transform size. Therefore, with longer transforms it is more efficient to compute the coefficients at run-time.

In Section 3.1, we introduce the twiddle factors in FFT algorithms. Next we discuss two principal methods for computing the coefficients; polynomial and recursive methods are discussed in section 3.2, methods based on CORDIC algorithm are introduced in section 3.3, and methods exploiting lookup tables are reviewed in section 3.4. Finally in section 3.5, we discuss the proposed method to compute the coefficients based on a lookup table, which supports mixed-radix FFTs consisting of radix-4 and radix-2 computations.

3.1 Twiddle Factors

The FFT algorithms contain complex-valued coefficients known as twiddle factors as shown in (3). They are defined as

$$W_N^k = e^{-j2\pi k/N} = \cos(2\pi k/N) - j\sin(2\pi k/N). \tag{6}$$

These coefficients are actually complex roots of unity evenly spaced in the unit circle on complex plane [20]. The number of different factors depends on the problem size N and the type of the fast algorithm. E.g., in radix-2 algorithms, each butterfly operation use one coefficient thus $\frac{N}{2}\log_2 N$ factors are used in the algorithm. However, there are only $\frac{N}{2}$ different factors in the algorithm as seen in the 16-point radix-2 Fourier transform illustrated in Figure 8 a). In the radix-4 algorithm, there are four twiddle factors in each butterfly:

$$W_N^0, W_N^k, W_N^{2k}, W_N^{3k}, k = 0, \ldots, \frac{N}{4} - 1.$$

As $W_N^0 = 1$, there is a total of $\frac{3}{4}N \log_4 N$ non-trivial twiddle factors while only $(\frac{N}{2} - 1)$ are unique, Figure 8 b) shows coefficients of 16-point radix-4 FFT.

When implementing an FFT algorithm, it is obvious that there is a large number of coefficients, which need to be used, especially when the transform size is large. The twiddle factors can be formed by using trigonometric functions, which are, however, expensive. The coefficients can be computed with fast algorithms, which exploit the trigonometric identities of twiddle factors, e.g., Singleton's method [69]. Then the coefficients can be computed on the fly, when they are needed. Another approach is to exploit lookup tables and read the coefficients from the tables. In many cases, the lookup tables are stored in ROM but, in software implementations, data memory can be used to store the coefficients. However, random access memory consumes more power than read-only memory. In addition, when coefficients are stored to data memory, additional memory accesses are required. In general, on-the-fly computation of twiddle factors requires less area, but consume more dynamic power compared to methods using lookup tables.

3.2 Polynomial and Recursive Twiddle Factor Generation

The twiddle factors can be generated as piecewise polynomial approximation of a function. Polynomial approximation requires multiplications and additions to compute the value of a function with given parameters. It should also be noted that the complexity of the polynomial based algorithm increases significantly with the required output precision. In [28], second order polynomial approximation is combined with Horner's rule to compute the sine and cosine values. The described computation unit contains four multipliers, four adders, and two lookup tables. The tables are used to partition the parameter ranges into regions. In order to increase precision, the number of regions have to be increased, which in turn increases the cost. Similar structure is proposed in [74].

The recursive twiddle factor generation is based on recursive feedback difference equations for sine and cosine functions. This approach is less complex compared

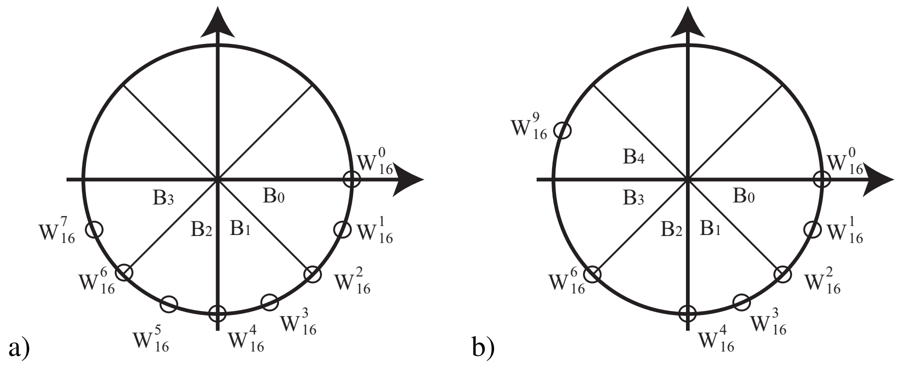

Fig. 8. *Twiddle factors in 16-point FFT: a) radix–2 algorithm and b) radix–4 algorithm.* B_k *refers to sector number.*

to the polynomial, one iteration uses two real-valued multiplications and two real-valued additions to produce a complex-valued result. The drawback of the algorithm is error propagation of the finite numbers due to the feedback structure of the algorithm. In [19], a method to reduce the complexity of error propagation circuit is proposed. The accuracy is improved with a LSB correction table containing $\frac{N}{8}$ 3-bit entries. The area cost is reduced by sharing the same multiplier and adder for both real and imaginary parts. This effectively doubles the latency of coefficient generation. The method uses two lookup tables for cosine and sine values and both tables require $\log_2 N - 2$ entries. The drawback is that the method generates an ordered sequence of twiddle factors, thus it supports only a specific type of FFT algorithms and the reported unit supports only radix-2 DIF FFT.

In these algorithms, the large number of iterations will increase the length of computation kernel. This might increase the need for intermediate storage. i.e., registers. Also the large number of multiplications will increase the power consumption of twiddle factor generation [P3].

3.3 Twiddle Factor Generation Based on CORDIC

The Coordinate Rotational Digital Computer (CORDIC) algorithm is a well known iterative algorithm for performing vector rotations [26, 45, 84]. All of the trigonometric functions can be evaluated by rotating an unit vector in complex plane. This operation is effectively performed iteratively with the CORDIC algorithm. The general rotation transform at iteration t can be given as

$$\begin{cases} X_{t+1} &= X_t \cos\phi - Y_t \sin\phi \\ Y_{t+1} &= Y_t \cos\phi + X_t \sin\phi \end{cases} \tag{7}$$

where (X_{t+1}, Y_{t+1}) is the resulting vector generated by rotation of an angle ϕ from the original vector (X_t, Y_t), i.e., the resulting vector rotates in the unit circle in similar fashion as the twiddle factors. Therefore, the CORDIC algorithm can be used to compute the twiddle factors, generating the sine and cosine values. In particular, CORDIC is used for replacing the twiddle factors with rotation information and, therefore, avoid multiplication with the twiddle factor by replacing it with rotation realised with additions. This operation mode is called rotation mode. The CORDIC can also operate in vectoring mode which returns the rotation angle and the scaled magnitude of the original vector.

The CORDIC multiplier consumes less power compared to a traditional multiplier. E.g., in [92], a pipelined CORDIC unit consumed roughly 20 % less power than the traditional complex-valued multiplier while the area cost was about the same. Recursive CORDIC iteration saves area compared to lookup based twiddle factors (discussed in the next section) but it introduces longer latency. In [92], the rotation angle constants for generating all the twiddle factors for an N-point FFT are stored in a lookup table with $\log_2 N$ entries. In [30], the twiddle factors are generated without pre-calculated coefficients. The CORDIC algorithm is iterative, thus it can be pipelined easily and it lends itself to pipelined FFT architectures. However, the dynamic power consumption with a large number of iterations and/or long pipeline will be in higher than in a lookup table based approach. This will be the case, when longer word widths are used, i.e., increased accuracy calls for more iterations. This can be seen also with short transforms. E.g., in [7], it is shown that multiplier-based FFT consumes significantly less power compared to CORDIC FFT on FPGA. However, the clock frequency of CORDIC implementation can be much higher.

Traditionally the CORDIC has mainly beed used in fixed-function ASICs but it can be used to accelerate computations in a programmable processor as reported in [68]. The authors describe instruction extensions for CORDIC operations and there are separate instructions for vectoring and rotation mode. Four different instructions are needed for a 16-iteration CORDIC operation. The architecture supports SIMD operation and two CORDIC 16-bit operations with 16-bit operands are carried out in parallel. Despite the overhead, the authors report 10x speedup in several matrix operations compared to a baseline architecture without the CORDIC extensions.

3.4 Methods Based on Lookup Tables

The twiddle factors can be stored in a lookup table, which is then indexed according to the selected algorithm. The simplest design, in radix-2, requires two coefficients in one butterfly, where the first twiddle factor is trivial +1 and the second is a non-trivial complex number. Now the memory contains all the twiddle factors in order in the memory thus the number of required coefficients in radix-2 is $\frac{N}{2}\log_2 N$ where N is the transform size, e.g., lookup table has 448 entries when $N = 128$ [11]. The memory can be now accessed with an index obtained from a simple counter. In this approach, the lookup table contains redundancy as many of the coefficients are the same. Such redundancy can be removed by adding an address generation logic before the lookup table. In [47, 88], a method to reduce the number of coefficients in radix-2 algorithms to $\frac{N}{2}$ is proposed. Each twiddle factor is stored only once in the table, i.e., the lookup table contains 64 entries when $N = 128$. Such a table can be used only for a sequential implementation but, in [49], a method is proposed, which allows the $N/2$ entries to be distributed over $2^P, P = 0, 1, ..., log_2\frac{N}{2} - 1$ sub tables such that those can be accessed by 2^P butterfly units simultaneously.

The previous twiddle factor table contains redundancy: as the twiddle factors are equally spaced in unit circle on the complex plane, there is symmetry as illustrated in Figure 8 a). We can note that the real and imaginary parts of the twiddle factors in the sector B_0 and B_1, as seen in Figure 8 a), can be used to obtain the twiddle factors required in sectors B_2 and B_3. The number of lookup table entries in the radix-2 case can be reduced down to $\frac{N}{4} + 1$. Such an approach has been presented in [14, 54, 80]. When $N = 128$ only 33 complex-valued entries are needed. In this method, the twiddle factors from sectors B_0 and B_1 in Figure 8 are stored to the lookup table and the rest of twiddle factors are generated simply by interchanging the real and imaginary parts of the coefficient and changing the sign of the real part of the coefficient before it is assigned to the imaginary part.

The previous representation still contains redundancy in lookup tables: the symmetry of twiddle factors between in different quadrants is exploited but the symmetry between real and imaginary parts of the twiddle factors is not used. This symmetry allows all the twiddle factors to be generated from only one sector, B_0 in Figure 8 a). In [36], this redundancy is avoided by generating twiddle factors for radix-2 FFTs with the aid of $\frac{N}{8}+1$ complex-valued twiddle factors stored in a lookup table, i.e., when $N=128$, the lookup table contains 17 entries. A twiddle factor W_N^k is computed based on exponent k as follows

$$W_N^k = \begin{cases} R_A + jI_A & \text{, when } 0 \le k \le \frac{N}{8} \\ -R_A - jI_A & \text{, when } \frac{N}{8} < k < \frac{N}{4} \\ I_A - jR_A & \text{, when } \frac{N}{4} \le k \le \frac{3N}{8} \\ -R_A + jI_A & \text{, when } \frac{3N}{8} < k < \frac{N}{2} \end{cases} \tag{8}$$

where A is the index to the lookup table consisting the coefficients from the segment B_0:

$$A = \begin{cases} k[n-2:0] & \text{, when } 0 \le k \le \frac{N}{8} \\ \sim k[n-2:0]+1 & \text{, when } \frac{N}{8}+1 \le k \le \frac{N}{4}-1 \\ k[n-2:0] & \text{, when } \frac{N}{4} \le k \le \frac{3N}{8} \\ \sim k[n-2:0]+1 & \text{, when } \frac{3N}{8}+1 \le k \le \frac{N}{2}-1 \end{cases} \tag{9}$$

where $k[a:b]$ denotes the bit field $(k_a, k_{a-1}, k_{a-2}, \dots, k_{b+1}, k_b)$ of bit-level representation of integer k, and $\sim$ denotes bit-wise complement operation. R_A and I_A are the real and imaginary part of the coefficient from the lookup table at the location A, respectively.

In Table 2, the twiddle factors for 32-point radix-2 FFT are listed in 2 decimals according to the sectors. The sine values, which are needed to create all the twiddle factors, are in the sector B_0. The actual twiddle factor requires only negation of sine and cosine values read from the lookup table as defined in (8). According to (9), the index to the lookup table is formed with simple operations: increment and complement. The index generation logic depends only on the length of FFT to be supported.

Table 2. *Twiddle factors in 32-point radix-2 FFT.*

B_0	B_1	B_2	B_3
$w_{32}^0(1.0, 0.0)$	$W_{32}^8(0.0, -1.0)$		
$w_{32}^1(.98, -.20)$	$W_{32}^7(.20, -.98)$	$W_{32}^9(-.20, -.98)$	$W_{32}^{15}(-.98, -.20)$
$w_{32}^2(.92, -.38)$	$W_{32}^6(.38, -.92)$	$W_{32}^{10}(-.38, -.92)$	$W_{32}^{14}(-.92, -.38)$
$w_{32}^3(.83, -.56)$	$W_{32}^5(.56, -.83)$	$W_{32}^{11}(-.56, -.83)$	$W_{32}^{13}(-.83, -.56)$
$w_{32}^4(.71, -.71)$		$W_{32}^{12}(-.71, -.71)$	

3.5 Extension for Lookup Table Methods to Mixed-Radix FFT

Many of the previously discussed methods are applicable only for a limited set of FFT algorithms, e.g., many methods support only radix-2 FFT. If more efficient mixed-radix computations are to be used, more flexible twiddle factor generation methods need to be found. In [P3], a twiddle factor computation method is proposed, which supports radix-4 FFT and mixed-radix FFT. The method exploits lookup tables and $\frac{N}{8}+1$ complex-valued twiddle factors need to be stored for an N-point FFT. When the lookup table has been created for an N-point FFT, the proposed method supports all the the power-of-two transform sizes up to N when computed with mixed-radix algorithms containing radix-4 and radix-2 computations. The Figure 8 b) shows that the method presented in (8) could not be used, the usage of sectors from b_0 to B_3 is not sufficient to generate all coefficients in radix-4 FFT. The main drawback of the lookup approach compared to polynomial methods is the fact that the size of table increases as the transform size increases. However, we target embedded systems where the FFT sizes are not huge. Therefore, the size and power consumption of the lookup tables are reasonable compared to multipliers required in polynomial methods. If huge transforms are computed, the polynomial methods will be more effective.

Apart from the lookup tables, the proposed twiddle factor unit requires only few additional components, mainly two real-valued negation units for changing the sign of sine and cosine values obtained from the lookup table. In addition, the method allows completely pipelined implementation, thus it can generate twiddle factors at each clock cycle unlike some of the other methods discussed earlier.

In the proposed method, the factor W_N^k is computed based on the given power of the principal root of unity, k. Such an approach is flexible as it allows the method to be used with several types of FFT algorithms as there is no need to compute the twiddle factors in a certain fixed order. The different ordering of twiddle factors can be organised simply by providing the powers, k, to the unit. This is also illustrated in [P3] as there is a permutation unit, in which the power parameter k is generated from a linear counter to be used as a operand for the proposed twiddle factor unit. Such permutations are performed at bit-level, thus no arithmetic units are needed and the solution requires a rather small area. Such an additional unit is not needed if radix-4 DIF algorithm is used. This is the same case as used in some polynomial methods, where the twiddle factors are generated in increasing order of powers, k.

4. ARCHITECTURES FOR FFT COMPUTATION

In general, the FFT implementations can be divided to two main classes: fixed-function processors and programmable processors. The fixed-function processors are tailored for a specific application and the functionality cannot be changed at run-time. However, fixed-function FFT processors may allow parameters to be changed, e.g., the size of he the Fourier transform to be computed can be changed at run-time. Fixed-function processors can be implemented either on an FPGA or as an ASIC.

The implementation style depends on the application at hand and the design goals. Traditionally fixed-function circuit is selected when application is known, i.e., there are no changes expected to the application, and in particular, low energy and/or high performance is desired. On the other hand, software implementation is flexible as the functionality can be changed easily. This style is selected when the application is not mature and there is a high probability to change the functionality. The application-specific processor tries to collect the best features of both styles; some of the resources are tailored for the specific application, thus lower power consumption can be expected. With adequate parallelism also higher performance can be assumed. Software implementation allows the functionality to be changed by changing the program.

In this chapter, architectures for computing fast Fourier transform are reviewed. First, fixed-function FFT processors are discussed in Section 4.1. In Section 4.2, more flexible solutions, i.e., FFT on programmable processors, are discussed. The proposed energy-efficient programmable processor for FFT is discussed in Section 4.3. First, the TTA template and the design framework related to TTA are discussed. Finally in Section 4.4, the energy-efficiency of the proposed customised processor is compared to other FFT processors including both fixed-function and programmable solutions.

4.1 Fixed-Function Architectures for FFT

In fixed-function FFT processors, the underlying hardware is either an FPGA or an ASIC tailored specifically for FFT computations. The circuit can be used as an accelerator for a host processor where the main goal of the circuit is to be faster and/or more energy-efficient than the host. One of the advantages of such a configuration is that while the accelerator is carrying out computations, the host is free to execute other tasks. Another possibility is to use the circuit as an individual block of a larger system or in the system-on-chip (SoC), e.g., in an OFDM transceiver.

The signal flow graphs of FFTs derived with Cooley-Tukey decomposition consist of smaller DFTs and multiplications with twiddle factors as seen in Figure 2. It can be seen that the basic building blocks of these algorithms are 2-point DFT and complex-valued multiplication, i.e., FFT butterflies. The butterfly related to radix-2 DIT algorithm in Figure 2(a) is shown in Figure 3(a). The FFT butterfly is related to the radix; radix-2 butterfly is based on 2-point DFT while radix-4 butterfly is based on 4-point DFT.

According to [3] FFT computing architectures are classified into five categories: single memory, dual-memory, pipelined, cache-memory, and array architectures. Here, single memory and dual-memory architectures are combined to memory architecture as illustrated in Figure 9(a). An example of pipeline architecture is shown in Figure 9(b), principal cache-memory architecture is shown in Figure 9(c), and array architecture depicted in Figure 9(d) .

Memory architectures consist one or more processing elements within the FFT column connected to main memory. In pipeline architecture there is a processing element, or elements, dedicated for one butterfly stage in the FFT algorithm. A memory architecture where local cache is connected between main memory and processing elements is called cache-memory architecture.

The array architecture consist of several independent processing elements with local buffers connected together by an interconnection network, many implemented as Network-on-Chip (NoC) [57]. Array systems may use multiple chips [17, 76] or a single chip [50]. Throughput of the array systems can be high as several butterflies in a FFT column are computed in parallel and the data for the next column is fed from local buffers via interconnection to processing elements. However, the cost of having

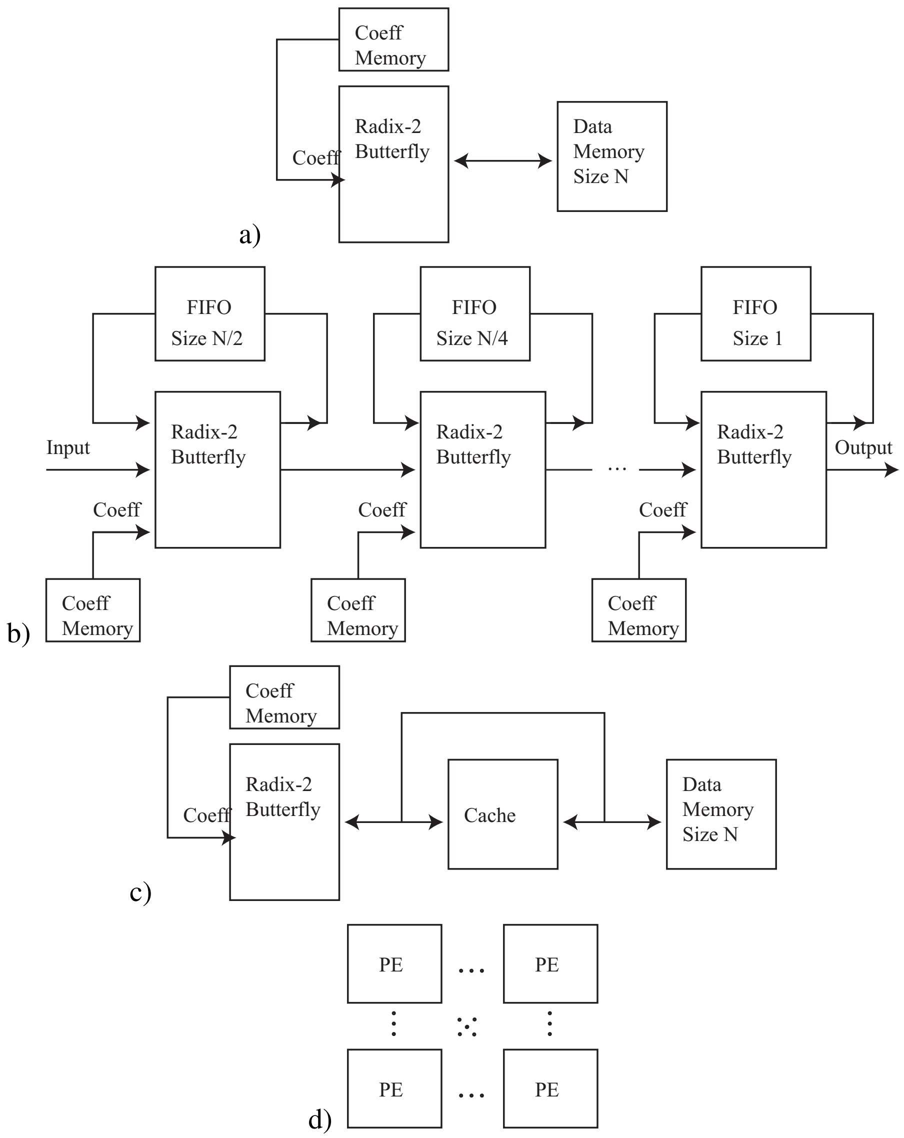

Fig. 9. *Principal architectures for fixed-function FFT processors: (a) memory, (b) pipelined, (c) cache-memory, and (d) array.*

such amount of parallelism and parallel data transports is high.

An architecture is developed by selecting an FFT algorithm with specific features, which then can be exploited for optimising the architecture. The simplest implementation is achieved when using memory architecture with a single processing element; other implementation methods are used when the simplest implementation style cannot satisfy the performance requirements. In the memory architecture, the computation speed of the system is restricted by the memory bandwidth since operands to butterfly computations and results use the same memory. The memory architecture with a single processing element is naturally the cheapest solution.

4.1.1 Memory Architecture

In memory architectures, one or more FFT butterflies from the same column, i.e., data processing for input set defined by FFT radix, is performed in parallel [53]. computing multiple butterflies from the same column in parallel increases the throughput of the system [89]. This increases parallelism and therefore the area cost, i.e., when using radix-2 FFT the maximum number of butterflies in one column is $\frac{N}{2}$. Higher radix can be used to increase parallelism, e.g., in radix-4 FFT, two radix-2 columns are combined. It should also be noted that the complexity of data flow control increases with the parallelism.

Input data for the butterfly is fetched from the data memory and the output data is stored back to it. The pressure for memory accesses increase with the parallelism. The coefficients used in the butterfly computations can be fetched from the data memory, from a separate coefficient memory, or generated on the fly. This topic was described in more details in Section 3.

Usually memory based systems are used when large Fourier transform sizes are needed but high throughput is not needed [89]. The performance is limited by the memory bandwidth, i.e., the number of available memory ports. Multi-ported memories can be used to increase the memory bandwidth but unfortunately such a solution is expensive both in area and power consumption. A cheaper solution is to use multiple single-port memories but in such parallel memory systems the bandwidth is reduced by memory conflicts. In case of conflicts, the memory accesses need to be serialised. In addition, conflict detection and management increases the area and power dissipation of the system. There are also conflict-free parallel access schemes tailored for FFT, e.g., the method reported in [35].

4.1.2 Pipelined Architectures

In the previous approach, parallelism was increased by computing several butterflies in parallel. However, throughput can also be increased by computing butterflies from multiple columns at the same time and this is the basis for the pipeline architecture [8, 79]. Such an architecture can achieve high throughput. At minimum, one butterfly unit is required for each column in the algorithm. This implies that the number of butterflies is dependent on the Fourier transform size unlike in the previous approach.

Another issue to note is the fact that in pipeline architecture all the twiddle factors in a butterfly column are mapped on to a single multiplier unit. This is very effective if the selected algorithm is such that many columns have constant twiddle factors and in such a case the constant multiplier can be used instead of a general complex-valued multiplier or complex rotator [59]. Different pipeline approaches are presented in [37] and the results indicate that the delay-feedback approach provides the best utilisation and minimises the memory cost. In radix-2 single delay-feedback approach, each butterfly unit requires a memory, i.e., a FIFO type structure, with size of $\frac{N}{radix \times s_i}$. The total size of these memories is $N-1$, which is the same as the memory based systems using in-place computation although the individual memories in pipeline architecture are smaller, however several different address generation schemes are required. Pipeline based systems consume more area due to the parallel resources and are used normally in FFT systems where high throughput is required.

The throughput of the system can be increased by computing multiple butterflies from the same column in parallel as shown for the pipelined systems in [44] or increasing the FFT radix. This requires more memory ports and the area cost is higher due to the increased number of processing elements (PE). In [75], a scalable FFT architecture is proposed, which combines principles from the memory-based and pipelined designs. As illustrated in Figure 10, the architecture carries out iterative computations with the aid of butterfly PEs . The input multiplexer selects either the input or data from the previous iteration. The architecture is based on a constant geometry algorithm where the interconnections between the butterfly columns are perfect shuffle permutations. Such permutations are realised in the architecture with the aid of hard-wired perfect shuffle network followed by temporal permutation based on cascaded switch-exchange units. Each unit contains two delay lines and multiplexers for exchanging the signals. The architecture is scalable in the sense that for N-point FFT

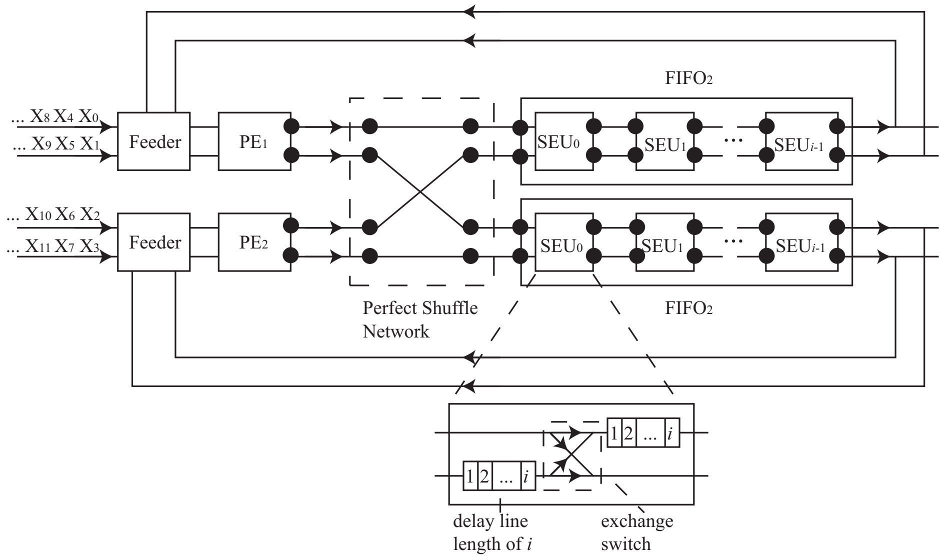

Fig. 10. *Principle organisation of scalable architecture for radix-2 FFT.*

the number of processing elements can be scaled from one to $N/2$. However, the number of processing elements must be a power-of-two. The twiddle factors for the butterfly processing elements are obtained from internal lookup tables.

In [45], an FFT architecture using a CORDIC multiplier is proposed. This pipeline architecture contains four radix-8 butterfly units and one radix-2 unit, thus it supports FFT sizes of 8192, 4096, 1024, 512, 128, and 64. The radix-8 units exploit distributed arithmetic. Each result of the radix-8 butterflies is fed to a CORDIC-multiplier and then stored to temporal memory. Due to the pipeline schedule, the memory usage is not optimal; $\frac{3N}{2} + \frac{9N}{128}$ memory elements are required for N-point Fourier transform. The system has throughput of one sample per cycle and the latency depends on the Fourier transform size. The area of the system is 55k gates without memories. The small area is achieved by using 16-bit internal data representation with 8-bit input data.

An 8-parallel pipelined architecture is proposed in [79] supporting Fourier transform sizes from 64 to 1024. Different transform sizes are based on different algorithms and radix-$2/2^5$radix-$2/2^4/2^5$, radix-$2^2/2^5$, radix-$2^3/2^5$, and radix-$2^4/2^5$ FFT algorithms are used. The locality of the algorithm is exploited when mapping the computations on 8-path or 4-path modules. The first module contains four radix-2 operation sets and the second consists five radix-2 operation sets. An 1024-point Fourier transform uses four data paths and an additional radix-2 stage in the first module. A CORDIC unit with nine pipeline stages is used in twiddle factor generation and complex multiplication. The accuracy of a 9-stage CORDIC unit is sufficient for input with 10 bits of precision. The actual input data format is block floating-point: 10-bit mantissa with shared 4-bit exponent in blocks of eight samples. The proposed scheme processes 512-point FFT in 64 cycles and 1024-point FFT in 256 cycles. The maximum clock frequency is 300MHz, which provides throughput of 2,4 Gb/s for 512-point Fourier transform. The 1024-point Fourier transforms are executed at 5 MHz speed. The processor takes 3.2 mm^2 of area.

4.1.3 Cache-Memory Architecture

In order to reduce the power consumption of a large centralised memory, a smaller memory, i.e., cache, could be used. Cache-memory architecture is based on this idea. The FFT algorithm must be such that it contains a high degree of locality [2] and the algorithm can be divided into independent groups, called epochs, of smaller FFTs [3]. The FFT size is defined by the epoch sizes, $N = N_{eo}N_{e1}\cdots N_{en}$ and each epoch contains $\frac{N}{N_e}$ independent N_e-point FFTs, i.e., this follows the Cooley-Tukey decomposition. For example, 1024-point FFT can be divided to two epochs, which each contain 32 FFTs of size 32. In this case, A memory with 32 entries could be used in inter-epoch computations and memory with 1024 entries is used in intra-epoch computations. The cache is usually so small that it can be implemented as a local register bank [3, 4].

An energy-efficient FFT based on cache-memory architecture is proposed in [16]. The unit uses a memory model where the cache is located between main memory and the FFT butterfly. FFT sizes from 128 to 1024 are supported and the FFT scheme is radix-2 and radix-2^2 using pipelined dual-delay-feedback architecture. This is an example how pipelined architecture model could be combined with a memory-based model, i.e., the butterfly uses pipeline based architecture and rest of the system is memory based. The authors present a 3-epoch system. Four memory accesses are required in each cycle, which would lead to an expensive four-ported implementation. To avoid this, the cache is divided to two banks containing even and odd addresses with data exchange logic if data is required to or from different cache. The FFT scheme is manipulated in order to have only one read and write from the even cache and the second one from the odd cache in a cycle. The main memory is divided in similar fashion to two single port memories. Two twiddle factors are required in each cycle, and those are provided from ROM memories where power consumption savings can be achieved by applying the twiddle factor generation model to the system. The system is optimised based on symbol-error-rate analysis and the word length is set to 13 bits. The multiplier is set to have two modes: full 13x13 bit multiplication and smaller, 13x(9-5) bit multiplication depending on the modulation scheme and the size of the Fourier transform.

4.2 FFT on Programmable Processors

In programmable processors, the underlying hardware is either a general-purpose processor (GPP), digital signal processor (DSP), or an application-specific processor (ASP). Usually GPPs are too power hungry to satisfy the demand on power budget or require too many cycles when driven in low power modes. The GPPs designed for low power dissipation, e.g., for mobile domain, contain typically hardware accelerators for computationally intensive algorithms like FFT. The hardware accelerator can be a DSP, ASP, or a fixed-function accelerator discussed in the previous Section. The accelerator is placed in the same circuit to reduce power consumption and to speed up the data transfers.

One popular GPP processor family in the mobile systems is ARM, which completes 1024-point FFT with 62.5 kcycles (1 GHz ARM cortex A9, Pandaboard) [18]. The DSPs often provide special instructions to speed up FFT computations and to improve energy-efficiency [82]. In software implementations, FFT computations are naturally related to memory architecture discussed in Section 4.1. This gives a lower bound $\frac{N}{\log_{radix} N \frac{M_a}{2}}$ to the cycle count where N is the size of the FFT, $radix$ is the radix of the FFT, and M_a is the number of memory accesses in a cycle.

In [82], the underlying hardware is a two-way exposed pipeline VLIW where eight parallel operations can be executed in parallel. The processor contains two register banks, each register bank provides service to four functional units, i.e., each register and FU combination forms a cluster. Each cluster has memory access, through a L1 cache, of 64-bit with aligned data and 32-bit with non aligned data, giving maximum throughput of 128 bits per cycle. The vendor specific libraries contain optimised FFT functions written in assembly language and the special instructions for speeding up

FFT computations are extensively used in the code. A 1024-point radix-4 FFT with 16-bit operands takes 6526 cycles without L1 cache misses and stalls. Estimating the L1 cache miss and stall cycles gives a total cycle count of around 8000 cycles. [82].

An ASP called SPOCS is presented in [5] where the processor architecture is specially designed for OFDM systems with custom hardware and instructions for FFT. The processor contains one program memory, two data memories, a program control unit, a data processing unit, and address unit, an instruction register and an interrupt controller. The address unit contains a special FFT address generation unit, which generates the address for accessing the input data as well as the twiddle factors. The twiddle factors are stored in the memory. The system supports FFT sizes from 64 to 8192 points but only a version supporting Fourier transform lengths up to 2048 points has been implemented. The processor uses radix-2 computations. The internal word length of the processor is 16 bits.

Another ASP tailored for mixed-radix FFT is proposed in [87]. It uses a two-issue five-stage dynamical-static-hybrid pipeline described in [40]. The ASP can compute two radix-4 and four radix-2 butterflies in every cycle, i.e., eight $32x32$-bit complex-valued multiply-accumulate operations in one cycle. The memory is vectorised and cached with separate vector direct memory access (DMA) unit. The memory word length is 256-bit. The implementation is based on the cache-memory architecture discussed in 4.1.3 and computations are divided to epochs. The epoch computations requires access to memory in columns and rows; the first epoch access the memory by columns and the second epoch by rows. For this purpose the data shuffling within the register file has been introduced. This saved an extra pipeline stage for the dedicated shuffling unit. This also requires dedicated instructions for the register file, in order to shuffle the data correctly. The twiddle factors are stored in the main memory but the redundancy of the twiddle factors is not exploited. The register file is power hungry and since the data path of the processor is not exposed the designer cannot optimise the register transports.

In [34], an ASP supporting radix-2 FFT sizes from 16 to 8092 points is proposed. The processor uses two or three epochs, depending on the size of the Fourier transform. By introducing the third epoch the size of the intra epoch register file could be kept small when the size of the Fourier transformation increases. The ASP has been obtained by adding custom hardware Tensilica's Xtensa template [33] to efficiently compute FFT on the ASP. The Xtensa core contains 128-bit wide data bus, which supports four parallel data reads or stores, i.e., the complex-valued data is 32-bit. The Xtensa core is expanded with two register files, twiddle factor memory, and radix-2 butterfly unit. The butterfly unit can compute four radix-2 butterflies simultaneously, which requires eight input data operands and four twiddle factors. This effectively implies four complex-valued multipliers. The two register files are used to store intermediate results within an epoch and the register files are used with ping-pong approach. Since in-place computations are not used, two register files are required. However, two register files simplify the addressing and butterflies are easier to design to have a uniform structure. The twiddle factor memory contains coefficients for intra epoch computations and the twiddle factors are fetched from the data memory to the local twiddle factor memory at the beginning of each epoch. The twiddle factors for inter-epoch, i.e., twiddle factors between the epochs, are fetched from the main memory. Total of $\frac{N}{8}+1$ coefficients are stored to memory, which are used to compute the twiddle factors on the fly. The twiddle factor generator would save the accesses to the data memory and thus would increase the performance and decrease the energy consumption, although the core area would be increased.

4.3 Proposed Architecture Based on Transport Triggering

In general, ASPs are based on flexible processor templates, which are then tailored according to the requirements of the given application. One such a template is the transport triggered architecture (TTA) [24], which is a class of statically programmed instruction level parallelism (ILP) architectures reminding very long instruction word (VLIW) architecture. It has been widely used in modern day embedded systems due to its modularity and scalability. In the TTA programming model, the program specifies only the data transports to be performed in the architecture. A data transport to a specific port in a functional unit triggers the actual operations. Such a paradigm is opposite to traditional operation triggered architectures.

The TTA was proposed to overcome the limitations of VLIW architectures, in particular, to reduce the register pressure; When the performance of a VLIW architecture is scaled up by adding new functional units (FU), new connections to the register file (RF) are required [21] to provide more operands from the register file to the larger number of functional units. Such connections limit the performance of the system by constraining the operational frequency and increasing power dissipation of the bus

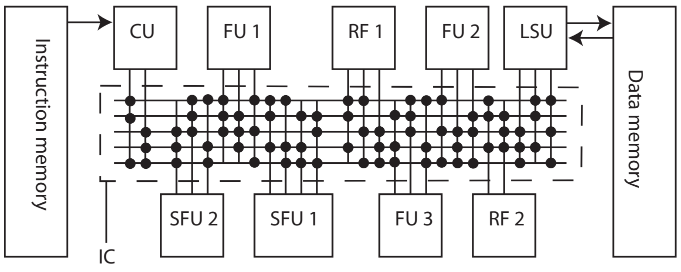

Fig. 11. *Principal organisation of transport triggered architecture consisting of instruction and data memories, control unit (CU), interconnection network (IC), functional units (FU), and register files (RF).*

structure.

The TTA the programming model allows specifying each transport and, therefore, the transports to the register file are visible to the programmer at compile time. The register file can be implemented as a functional unit that has significantly less read and write ports compared to a normal register file in VLIW [24]. Also the complexity of the interconnection network, i.e., the bus connecting FUs and RFs, can be significantly reduced compared to the VLIWs. Derived from the VLIW, the TTA possess modularity and scalability, which allows tailoring the processors by including only the necessary FUs. The application-specific functions can be implemented with user defined special functional units (SFU), which are used in a manner similar to standard functional units, they transporting operands to input port of the function unit.

Figure 11 illustrates the principal organisation of a TTA processor. Control unit (CU) reads the instruction from instruction memory and decodes the instruction as transports within the interconnection network (IC) between FUs, SFUs, and RFs. The control unit can also receive transports from the interconnection network to maintain the return address and perform jumps. The load store unit (LSU) is a special functional unit, which is used to connect memory to the machine. The interconnection network consist connections to the FUs and connections from the FUs. The usual number of connections are two input ports and one for the output but the number of ports are not restricted by the architecture. The register file can contain an arbitrary number of input (write) and output (read) ports. In the figure, the connections to and from the FU or the RF are marked with dots. Apart from customisation of the computing resources and their connections, the word lengths of each resource, FU, RF, SFU or bus, can be set according to the requirements of the given application.

The main drawback of TTA is the wide instruction word like in VLIW. This can be alleviated by using instruction compression as discussed, e.g., in [39] where dictionary-based compression, Huffman coding, and instruction template-based compression were evaluated.

The TTA-based Codesign Environment (TCE) [43] is an application-specific processor design toolset that uses a multi-issue exposed data path processor architecture template. The design flow allows the designer to freely customise the data path resources in the core to exploit the available instruction-level parallelism (ILP) in computation intensive kernels. The design toolset includes a retargetable compiler and architecture simulator, making design space exploration feasible. The input language for the toolset could be OpenCL, C/C++, or parallel assembly. Apart from tailoring the processor architecture with basic data path resources, the designer can also create user-specific instruction set extensions (SFUs) to further accelerate the application at hand [27].

One possible candidate for the SFU is found by chaining up the operations executed; if an operation pattern is repeated in the application, it is a good candidate for a user specific functional unit. Such operation patterns can be, e.g., memory address generation, complex addition, and complex multiplication, as illustrated in [P2–P5]. The SFU can also contain more specialised and complex functions like twiddle factor generator as shown in [P3]. The TTA template supports different latencies, thus the special function units can be pipelined to an arbitrary number of stages.

After the required performance is achieved, the next step is to minimise the resources to reduce power dissipation, while maintaining the performance. When the data path is highly customised for one application there is only small overhead in the area and energy budget compared to fixed-function ASICs [66]. The programmability of the processor is usually limited if heavy customisations are used, i.e., when general-purpose functional units are removed. However, the architecture is still programmable but the performance of general applications may not be high. With modern tools, the design of TTA processor takes less time than designing a fixed-function hardware implementation [P1]. The TCE framework is used for designing the experiments used in this Thesis.

4.4 Energy-Efficiency Comparison

The energy-efficiency is defined as the energy consumed for performing the required task. Power consumption is a usual design metric when designing energy-efficient systems. The power consumption depends on several issues: area due to the static power consumption, computing resources, memories, caches, computation cycles, operating voltage, and operation frequency. In general, the memory architecture with a single butterfly processing element has the lowest power consumption due to the simplicity of the design. Such a system has small area but as it is sequential, it has also the lowest performance. The other architectures possess higher levels of parallelism, which increases the power consumption. The increased parallelism allows lower clock frequency to be used for the same performance. This allows lower supply voltage to be used and, therefore, a parallel solution may even provide better energy-efficiency [79], even though the power consumption is higher.

In this Thesis, the energy-efficiency is measured as the number of 1024-point FFTs performed with energy of 1 mJ. This approach tries to compensate the effect of computational speed but there are other implementation specific parameters, which have a great effect on the result. In order to have meaningful comparisons, the implementations should be normalised. Although exact scaling of the characteristics of an implementation on a specific IC technology to another technology is difficult, even impossible, there are several normalisation methods proposed in the literature. In [79], a normalisation method for IC technologies is proposed, which take into account many aspects of architectures and implementations. The normalised energy consumption of a system, E_{norm}, is defined as follows:

$$E_{norm} = E_{impl} \frac{L_{ref} U_{ref}^2 (\frac{1}{3} W_{ref}^2 + \frac{2}{3} W_{ref})}{L_{impl} U_{impl}^2 (\frac{1}{3} W_{impl}^2 + \frac{2}{3} W_{impl})} \tag{10}$$

where E_{impl} is energy consumption of the system implemented on a specific IC technology, W_{impl} is the word length of the system, U_{impl} is the supply voltage of the implementation, and L_{impl} is feature size of the specific IC technology on which the system has been implemented. The energy consumption of the implemented system is normalised for the same system implemented on reference technology where L_{ref} is the feature size of the reference IC technology, U_{ref} is the supply voltage of reference technology, and W_{ref} is the word length of the reference design.

Table 3. *Energy-efficiency comparison of various normalised FFT implementations measured as the number of executed 1024-point FFTs with energy of 1 mJ. V_{CC}: Supply voltage. WL: Word length. t_{clk}: Clock period. t_{FFT}: FFT execution time.*

Design	Tech. [nm]	Method	WL [bits]	V_{CC} [V]	t_{clk} [MHz]	t_{FFT} [μs]	FFT/mJ
[P5]	130	ASP	16	1.5	250	21	809
[18]	65	GPP	16	1.2	1000	63	1[1]
[82]	130	DSP	16	1.5	720	8	100[1]
[5]	180	ASP	16	1.8	280	37	61[1]
[75]	45	ASIC	32	0.9	650	2	1007
[16]	180	ASIC	13	1.8	51	61	748
[87]	65	ASP	16	1.2	150	6	633[1]
[34]	130	ASP	16	1.5	320	14	1170[1]
[79]	180	ASIC	14	1.8	5	220	755

[1] Energy does not include memories.

We have compared the energy-efficiency of the proposed TTA processor customised for FFT computations from [P5] to several other FFT implementations by using the previous normalisation with the following parameter set:

- L_{ref} = 130 nm,
- U_{ref} = 1.5V, and
- W_{ref} = 16 bits.

The energy-efficiency comparisons are listed in Table 3. As predicted the implementation on a general-purpose processor in [18] has quite poor energy-efficiency. The DSP processor presented in [82] can achieve high performance but the energy-efficiency in high speed mode is lower than execution with lower frequency and core voltage. The high performance is achieved by using manually optimised assembly code. It should be noted that the energy figures for this processor in Table 3 do not include memories.

The ASP presented in [5] introduces special operations to obtain higher performance and improve the energy-efficiency. The special function units are address generation, subtraction butterfly, and addition butterfly. Each butterfly uses two complex-valued multipliers and three complex-valued adders. The twiddle factors are stored in the main memory. It seems that the performance and energy-efficiency could have been improved by using a sophisticated twiddle factor generation mechanism.

The pipeline architecture in [75] uses permutation realised with the aid of delay lines, which introduce high dynamic power as the data travels through the registers in a delay line. The authors do not describe whether the twiddle factor memories and address generators are included in the energy figures.

An energy-efficient pipelined processor is proposed in [79]. The processor uses 10-bit mantissa and 4-bit shared exponent in floating-point arithmetic to gain higher accuracy than fixed-point arithmetic. The processor is capable of running a 512-point FFT at 300 MHz while consuming power of 507 mW. The authors present power figure for the 1024-point FFT at 5 MHz clock. The narrow floating-point word length allows CORDIC pipeline to be shortened as the accuracy obtained depends on pipeline length of CORDIC. If larger word lengths are needed, e.g., to support larger Fourier transforms or to improve the signal to noise ratio, the pipeline depth needs to be increased, which increases the power consumption.

The cache-memory architecture in [16] provides an energy-efficient memory architecture where a small data cache is used to reduce memory accesses. The processor uses 13-bit complex data input and the maximum FFT size is 1024-points. Increasing the maximum size will increase the memory and cache energy consumption and if better accuracy is required the increase in word length would increase the power consumption of memories and arithmetic units. In [87], a small cache memory is used to reduce memory accesses. The energy figure does not include caches or memories. Further energy savings could be reached by using sophisticated twiddle factor generation instead of storing all the twiddle factors to the main memory.

The ASP in [34] has two small caches to reduce memory accesses. The caches are accessed in ping-pong fashion to avoid performance penalty due to stall cycles when off-loading the computation results to the main memory. Two kinds of coefficients are required; intra epoch twiddle factors and N different inter epoch twiddle factors. The intra epoch twiddle factors are stored in a ROM memory and inter epoch twiddle factors are stored to main memory. The implementation uses $\frac{N}{8}+1$ complex-valued coefficients to compute the twiddle factors. The butterfly unit can compute four butterflies in parallel, which implies that there are four complex multipliers, while the proposed design contains only one. The processor supports a maximum of 1024-point FFT and, in order to support larger FFT sizes, the cache sizes or the number of epochs has to be increased. The memories, except intra epoch twiddle factor ROM, and caches are excluded from the energy figure. Finally, the power consumption estimates have been obtained from Tensilica tool and the accuracy of the power estimation is not known.

5. CONCLUSIONS

In this Thesis, several methods for increasing the energy-efficiency of fast Fourier computations on application-specific processor platform based on transport triggered architecture are proposed. The application-specific processor template was selected as the primary target platform for picking the best of the two worlds: flexibility of programmability and efficiency due to customisation. In addition, the design cycle of an ASP with aid of modern design tools is significantly shorter compared to traditional circuit design cycle. The main goal was to improve energy-efficiency without sacrificing performance and for obtaining this goal optimisations both in hardware and software were used. By introducing sufficient parallelism in the architecture, the computationally intensive kernels could be mapped efficiently on the computing resources. The traditional overheads due to iterations were minimised by exploiting software pipelining. High parallelism and in particular high utilisation of the computational resources provides high performance with good energy-efficiency. The proposed system still has flexibility due to programmability.

The main contributions of this Thesis are the following:

- novel coefficient generation method for an N-point mixed-radix FFT where the lookup table contains only $\frac{N}{8}+1$ complex-valued coefficients,
- energy-efficient memory organisation for FFT computations based on parallel memory concept,
- energy-efficient operand address generation for mixed-radix FFT, and
- novel processor structure for mixed-radix FFT computations exploiting the exposed data path of transport triggered architecture.

Memory bandwidth is often the main bottleneck limiting the performance. Increasing the bandwidth by adding more memory ports is extremely costly with traditional methods. One solution is to add single port memories but there the memory access conflicts slow down the execution. Here, the parallel memory access scheme is proposed to reduce the energy penalty for using multiple memory ports and to remove memory access conflicts, i.e., allowing continuous loading of the operands and storing the results. Parallel memory emulates multi-ported memory, i.e., the software designer does not require knowledge of the memory architecture, which might speed up the design cycle. The proposed parallel memory scheme is targeted towards ASP designs where the existing principles are used to construct a conflict-free access scheme. The proposed scheme does not employ any predefined access format signals for the parallel memory and the detailed hardware structure for dynamic conflict resolution is included in the memory interface. One of the sources for energy consumption are memory address computations and an here energy-efficient approach was taken as address computation units were customised for the application. Especially, by operating at bit-level, expensive word-wide arithmetic operations can be avoided and replaced with simple bit-wise logic-operations. The customised operations reduce the overhead as long sequences of arithmetic operations can be avoided. The drawback is, of course, the fact that the address computations are application-specific. but programmability also general address computations in a similar fashion as in any other processor. Another source for energy consumption is a multi-ported register file. In particular, the write ports are expensive. In the exposed data path of the TTA template, register file distribution was used. In addition, in TTA processors the operand transports introduce overhead, thus special arrangements were proposed to lower such overheads. E.g., the proposed multi-input complex-valued adder provides an energy-efficient solution for performing radix-4 butterfly operations without moving the operands several times. The approach is normally used in fixed function circuits and could only be used with exposed data path processors.

Finally, various coefficients in DSP algorithms can also be a source of high energy consumption. When large number of coefficients are needed, hether they are computed on-the-fly, indicating high dynamic power consumption due to computations, or stored in memory, implying energy consumption due to access to it. By exploiting the special properties of the coefficients, energy-efficient approach for application specific architectures can be developed. This was illustrated with the aid of the proposed special function unit for twiddle factor generation. The proposed method supports radix-4 and mixed-radix algorithms while the existing methods are proposed for radix-2 algorithms.

The results of the proposed ASP are characterised in terms of energy consumption per 1024-point FFT computation and the area in terms of the equivalent gates on 130 nm IC process. In conclusion, the FFT can be implemented efficiently, both in terms of energy and performance. The transport triggered architecture provides a good template for energy-efficient computations in the area of digital signal processing.

In order to support all the Fourier transform sizes specified in the 3G LTE specific-ation, further development is required to support non-power-of-two transforms, i.e., support for radix-3 and radix-5 are required. The energy-efficiency could be further improved by reducing memory accesses by introducing a small cache and tweaking the FFT algorithm to maximise the cache usage.

BIBLIOGRAPHY

[1] D. Astely, E. Dahlman, A. Furuskar, Y. Jading, M. Lindstrom, and S. Parkvall, "LTE: The evolution of mobile broadband," *IEEE Communications Magazine*, vol. 47, no. 4, pp. 44–51, Apr. 2009.

[2] B. M. Baas, "An energy-efficient single-chip FFT processor," in *Proc. IEEE Symp. VLSI Circuits*, June 1996, pp. 164–165.

[3] ——, "A low-power, high-performance, 1024-point FFT processor," *IEEE J. Solid State Circuits*, vol. 43, no. 3, pp. 380–387, Mar. 1999.

[4] ——, "A generalized cached-FFT algorithm," in *IEEE Int. Conf. Acoustics Speech Signal Processing*, vol. 5, Mar. 2005, pp. 89–92.

[5] J. H. Baek, S. D. Kim, and M. H. Sunwoo, "SPOCS: Application specific signal processor for OFDM communication systems," *J. Signal Processing Syst.*, vol. 53, no. 3, pp. 383–397, Dec. 2008.

[6] P. Bassett and M. Saint-Laurent, "Energy efficient design techniques for a digital signal processor," in *IEEE Int. Conf. IC Design Techn.*, June 2012, pp. 1–4.

[7] R. Bhakthavatchalu, N. Abdul Kareem, and J. Arya, "Comparison of reconfigurable FFT processor implementation using CORDIC and multipliers," in *IEEE Recent Advances in Intelligent Computational Systems*, Sept. 2011, pp. 343–347.

[8] G. Bi and E. V. Jones, "A pipelined FFT processor for word-sequential data," *IEEE Trans. Acoustics, Speech and Signal Processing*, vol. 37, no. 12, pp. 1982–1985, Dec. 1989.

[9] T. Bollaert, "Catapult synthesis: A practical introduction to interactive C synthesis high-level synthesis," in *High-Level Synthesis*, P. Coussy and A. Morawiec, Eds. Dordrecht: Springer Netherlands, 2008, ch. 3, pp. 29–52.

[10] J. Boutellier, I. Lundbom, J. Janhunen, J. Ylimainen, and J. Hannuksela, "Application-specific instruction processor for extracting local binary patterns," in *Conf. Design and Architectures for Signal and Image Processing*, Oct. 2012, pp. 1–8.

[11] E. O. Brigham, *The Fast Fourier Transform.* Englewood Cliffs, New Jersey: Prentice-Hall, 1974.

[12] F. Catthoor and H. De Man, "Application-specific architectural methodologies for high-throughput digital signal and image processing," *IEEE Trans. Acoustics, Speech and Signal Processing*, vol. 38, no. 2, pp. 339–349, Feb. 1990.

[13] W.-H. Chang and T. Q. Nguyen, "On the fixed-point accuracy analysis of FFT algorithms," *IEEE Trans. Signal Processing*, vol. 56, no. 10, pp. 4673–4682, Oct. 2008.

[14] Y.-N. Chang and K. K. Parhi, "Efficient FFT implementation using digit-serial arithmetic," in *Proc. IEEE Int. Workshop Signal Process. Syst.*, Taipei, Taiwan, 1999, pp. 645–653.

[15] R. Chassaing and D. Reay, *Digital Signal Processing and Applications with the TMS320C6713 and TMS320C6416 DSK.* Wiley-IEEE Press, 2008.

[16] C.-M. Chen, C.-C. Hung, and Y.-H. Huang, "An energy-efficient partial FFT processor for the OFDMA communication system," *IEEE Trans. Circuits and Systems II: Express Briefs*, vol. 57, no. 2, pp. 136–140, Feb. 2010.

[17] T. Chen, G. Sunada, and J. Jin, "Cobra: A 100-MOPS single-chip programmable and expandable FFT," *IEEE Trans. VLSI Systems*, vol. 7, no. 2, pp. 174–182, June 1999.

[18] K.-T. Cheng and Y.-C. Wang, "Using mobile GPU for general-purpose computing: A case study of face recognition on smartphones," in *Proc. Int. Symp. VLSI Design Automation Test*, Apr. 2011, pp. 1–4.

[19] J.-C. Chi and S.-G. Chen, “An efficient FFT twiddle factor generator,” in *Proc. European Signal Process. Conf.*, Vienna, Austria, Sept. 6–10 2004, pp. 1533–1536.

[20] E. Chu and A. George, *Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms*. Boca Raton, FL: CRC Press, 2000.

[21] R. P. Colwell, R. P. Nix, J. J. O’Donnell, D. B. Papworth, and P. K. Rodman, “A VLIW architecture for a trace scheduling compiler,” *IEEE Trans. Computers*, vol. 37, no. 8, pp. 967–979, Aug. 1988.

[22] C. Conroy, S. Sheng, A. Feldman, G. Uehara, A. Yeung, C.-J. Hung, V. Subramanian, P. Chiang, P. Lai, X. Si, J. Fan, D. Flynn, and M. He, “A CMOS analog front-end IC for DMT ADSL,” in *IEEE Int. Solid-State Circuits Conference*, 1999, pp. 240–241.

[23] J. W. Cooley and J. W. Tukey, “An algorithm for the machine calculation of complex Fourier series,” *Mathematics of Computation*, vol. 19, no. 90, pp. pp. 297–301, 1965. [Online]. Available: http://www.jstor.org/stable/2003354

[24] H. Corporaal, *Microprocessor Architectures: From VLIW to TTA*. Chichester, UK: John Wiley & Sons, 1997.

[25] P. Dang, “High performance architecture of an application specific processor for the H.264 deblocking filter,” *IEEE trans. VLSI Systems*, vol. 16, no. 10, pp. 1321–1334, Oct. 2008.

[26] A. M. Despain, “Fourier transform computers using CORDIC iterations,” *IEEE Trans. Computers*, vol. C-23, no. 10, pp. 993–1001, Oct. 1974.

[27] O. Esko, P. Jääskeläinen, P. Huerta, C. S. de La Lama, J. Takala, and J. I. Martinez, “Customized exposed datapath soft-core design flow with compiler support,” in *Proc. Int. Conf. on Field Programmable Logic and Applications*, Milan, Italy, 2010, pp. 217–222.

[28] L. Fanucci, R. Roncella, and R. Saletti, “A sine wave digital synthesizer based on a quadratic approximation,” in *Proc. IEEE Int. Frequency Control Symp. PDA Exhibition*, 2001, pp. 806–810.

[29] B. Farhang-Boroujeny and S. Gazor, “Generalized sliding FFT and its application to implementation of block LMS adaptive filters,” *IEEE Trans. Signal Process.*, vol. 42, no. 3, pp. 532–538, Mar. 1994.

[30] M. Garrido and J. Grajal, “Efficient memoryless CORDIC for FFT computation,” in *Proc. IEEE Int. Conf. Acoustics Speech Signal Process.*, vol. 2, Apr. 2007, pp. 113–116.

[31] M. Garrido, J. Grajal, and O. Gustafsson, “Optimum circuits for bit reversal,” *IEEE Trans. Circuits Syst. II*, vol. 58, no. 10, pp. 657–661, Oct. 2011.

[32] W. M. Gentleman and G. Sande, “Fast Fourier transforms: For fun and profit,” in *Proc. AFIPS ’66 (Fall)*, San Francisco, CA, USA, Nov. 1966, pp. 563–578.

[33] R. Gonzalez, “Xtensa: A configurable and extensible processor,” *IEEE Micro*, vol. 20, no. 2, pp. 60–70, Mar. 2000.

[34] X. Guan, Y. Fei, and H. Lin, “Hierarchical design of an application-specific instruction set processor for high-throughput and scalable FFT processing,” *IEEE Trans. VLSI Systems*, vol. 20, no. 3, pp. 551–563, Mar. 2012.

[35] D. T. Harper III, “Block, multistride vector, and FFT accesses in parallel memory systems,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 2, no. 1, pp. 43–51, Jan. 1991.

[36] M. Hasan and T. Arslan, “FFT coefficient memory reduction technique for OFDM applications,” in *Proc. IEEE ICASSP*, vol. 1, Orlando, FL, May 13–17 2002, pp. 1085–1088.

[37] S. He and M. Torkelson, “A new approach to pipeline FFT processor,” in *Proc. Int. Parallel Process. Symp.*, Honolulu, Hawaii, USA, Apr. 15–19 1996, pp. 766–770.

[38] Y. He, D. She, B. Mesman, and H. Corporaal, “MOVE-Pro: A low power and high code density TTA architecture,” in *Proc. Int. Conf. Embedded Computer Systems*, July 2011, pp. 294 –301.

[39] J. Heikkinen, “Program compression in long instruction word application-specific instruction-set processors,” Ph.D. dissertation, Tampere University of Technology, Finland, Tampere, Finland, 2007.

[40] W.-W. Hu, Y.-P. Gao, T.-S. Chen, and J.-H. Xiao, "The Godson processors: Its research, development, and contributions," *Journal of Computer Science and Technology*, vol. 26, pp. 363–372, 2011. [Online]. Available: http://dx.doi.org/10.1007/s11390-011-1139-2

[41] *IEEE Standard for Information Technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std P802.11-REVmb/D12, Mar. 29 2012.

[42] *IEEE Standard for WirelessMAN-Advanced Air Interface for Broadband Wireless Access Systems*, IEEE Std 802.16.1–2012, 7 2012.

[43] P. Jääskeläinen, V. Guzma, A. Cilio, and J. Takala, "Codesign toolset for application-specific instruction-set processors," in *Proc. SPIE Multimedia on Mobile Devices*, Jan. 2007, pp. 1–11.

[44] M. A. Jaber and D. Massicotte, "A new FFT concept for efficient VLSI implementation: Part ii – parallel pipelined processing," in *Int. Conf. Digital Signal Processing*, July 2009, pp. 1–5.

[45] R. M. Jiang, "An area-efficient FFT architecture for OFDM digital video broadcasting," *IEEE Trans. Consumer Electronics*, vol. 53, no. 4, pp. 1322–1326, Nov. 2007.

[46] H. Johnson and C. Burrus, "An in-order, in-place radix-2 FFT," in *IEEE Int. Conf. Acoustics Speech Signal Processing*, vol. 9, Mar. 1984, pp. 473–476.

[47] S. L. Johnsson, R. L. Krawitz, R. Frye, and D. MacDonald, "A radix-2 FFT on connection machine," in *Proc. ACM/IEEE Conf. Supercomputing*, Nov. 1989, pp. 809–819.

[48] G. Jones, "DAB goes digital [digital audio broadcasting]," in *IEE Seminar Broadcasting Spectrum: The Issues*, June 2005, p. 2 pp.

[49] P.-C. Jui, C.-L. Wey, and M.-T. Shiue, "Low-cost parallel FFT processors with conflict-free ROM-based twiddle factor generator for DVB-T2 applications," in *Proc. IEEE Int. Midwest Symp. Circ. Syst.*, Columbus, OH, Aug. 4–7 2013, pp. 1003–1006.

[50] M.-K. Lee, K.-W. Shin, and J.-K. Lee, “A VLSI array processor for 16-point FFT,” *IEEE J. Solid-State Circ.*, vol. 26, no. 9, pp. 1286–1292, Sep 1991.

[51] C. Li and J. Guan, “Code acquisition method of GPS signals based on WFTA, PFA and Cooley-Tukey FFT,” in *Proc. IEEE Int. Conf. Intelligent Control, Automatic Detection and High-End Equipment*, July 2012, pp. 119 –122.

[52] W. Li, “Overview of fine granularity scalability in MPEG-4 video standard,” *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, no. 3, pp. 301–317, Mar. 2001.

[53] X. Li, Z. Lai, and J. Cui, “A low-power and small area FFT processor for OFDM demodulator,” *IEEE Trans. Consumer Electr.*, vol. 53, no. 2, pp. 274–277, May 2007.

[54] Y. Ma and L. Wanhammar, “A hardware efficient control of memory addressing for high-performance FFT processors,” *IEEE Trans. Signal Process.*, vol. 48, no. 3, pp. 917–921, Mar. 2000.

[55] D. Novo, A. Kritikakou, P. Raghavan, L. Van der Perre, J. Huisken, and F. Catthoor, “Ultra low energy domain specific instruction-set processor for on-line surveillance,” in *IEEE Symp. Application Specific Processors*, June 2010, pp. 30–35.

[56] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 3rd ed. Upper Saddle River, NJ: Pearson, 2010.

[57] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, “Performance evaluation and design trade-offs for network-on-chip interconnect architectures,” *IEEE Trans. Computers*, vol. 54, no. 8, pp. 1025–1040, Aug. 2005.

[58] T.-C. Pao, C.-C. Chang, and C.-K. Wang, “A variable-length DHT-based FFT/IFFT processor for VDSL/ADSL systems,” in *IEEE Asia Pasific Conf. Circuits and Systems*, vol. 1, Dec. 2004, pp. 381–384.

[59] F. Qureshi and O. Gustafsson, “Twiddle factor memory switching activity analysis of radix-2^2 and equivalent FFT algorithms,” in *Proc. IEEE ISCAS*, Paris, France, May 30–June 2 2010, pp. 4145–4148.

[60] D. Rajaveerappa and A. Almarimi, “RSA/shift secured IFFT/FFT based OFDM wireless system,” in *Int. Conf. Information Assurance and Security*, vol. 2, Aug. 2009, pp. 208–211.

[61] U. H. Reimers, “DVB -the family of international standards for digital video broadcasting,” *Proc. IEEE*, vol. 94, no. 1, pp. 173–182, Jan. 2006.

[62] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *ACM Commun.*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[63] J. Rodriguez, O. Lifschitz, V. Jimenez-Fernandez, P. Julian, and O. Agamennoni, “Application-specific processor for piecewise linear functions computation,” *IEEE Trans. Circuits and Systems I: Regular Papers*, vol. 58, no. 5, pp. 971–981, May 2011.

[64] D. Rueckert, L. I. Sonoda, C. Hayes, D. L. G. Hill, M. O. Leach, and D. J. Hawkes, “Nonrigid registration using free-form deformations: Application to breast MR images,” *IEEE Trans. Medical Imaging*, vol. 18, no. 8, pp. 712–721, Aug. 1999.

[65] A. Saidi, “Decimation-in-time-frequency FFT algorithm,” in *Proc. IEEE Int. Conf. Acoustics Speech Signal Processing*, vol. iii, Apr. 1994, pp. 453–456.

[66] P. Salmela, “Implementations of baseband functions for digital receivers,” Ph.D. dissertation, Tampere University of Technology, Finland, Tampere, Finland, 2009.

[67] H. G. Schantz, “Three centuries of UWB antenna development,” in *IEEE Int. Conf. Ultra-Wideband*, Sep 2012, pp. 506–512.

[68] M. Senthilvelan, M. Sima, D. Iancu, M. Schulte, and J. Glossner, “Instruction set extensions for matrix decompositions on software defined radio architectures,” *Journal of Signal Processing Systems*, vol. 70, pp. 289–303, 2013.

[69] R. Singleton, “A method for computing the fast Fourier transform with auxiliary memory and and limited high-speed memory,” *IEEE Trans. Audio Electroacoustics*, vol. 15, no. 2, pp. 91–98, June 1967.

[70] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG 2000 still image compression standard," *IEEE Signal Processing Magazine*, vol. 18, no. 5, pp. 36–58, Sept. 2001.

[71] J. Sled, A. Zijdenbos, and A. Evans, "A nonparametric method for automatic correction of intensity nonuniformity in MRI data," *IEEE Trans. Medical Imaging*, vol. 17, no. 1, pp. 87–97, Feb. 1998.

[72] S. W. Smith, *The scientist and engineer's guide to digital signal processing*. San Diego, CA, USA: California Technical Publishing, 1997.

[73] W. Smith, *Handbook of Real-Time Fast Fourier Transforms (Algorithms to Product Testing)*. Wiley-IEEE Press, 1995.

[74] A. Sodagar and G. Roientan, "A novel architecture for ROM-less sine-output direct digital, frequency synthesizers by using the 2nd-order parabolic approximation," in *Proc. IEEE/EIA Frequency Control Symp. Exhibition*, 2000, pp. 284–289.

[75] A. Suleiman, H. Saleh, A. Hussein, and D. Akopian, "A family of scalable FFT architectures and an implementation of 1024-point radix-2 FFT for real-time communications," in *IEEE Int. Conf. Computer Design*, Oct. 2008, pp. 321–327.

[76] G. Sunada, J. Jin, M. Berzins, and T. Chen, "COBRA: An 1.2 million transistor expandable column FFT chip," in *Proc. IEEE Int. Conf. Computer Design*, Oct. 1994, pp. 546–550.

[77] *Processor Designer: Automating the Design and Implementation of Application-Specific Processors and Programmable Accelerators*, Synopsys, Mountain View, CA, 2012.

[78] Y. Takeuchi, K. Sakanushi, and M. Imai, "Generation of application-domain specific instruction-set processors," in *Int. SoC Design Conference*, Nov. 2010, pp. 75–78.

[79] S.-N. Tang, C.-H. Liao, and T.-Y. Chang, "An area- and energy-efficient multimode FFT processor for WPAN/WLAN/WMAN systems," *IEEE J. Solid-State Circ.*, vol. 47, no. 6, pp. 1419 –1435, June 2012.

[80] Y. Tang, L. Qian, Y. Wang, and Y. Savaria, "A new memory reference reduction method for FFT implementation on DSP," in *Proc. ISCAS*, vol. 4, May 2003, pp. 496–499.

[81] Y. Tawk, A. Jovanovic, J. Leclere, C. Botteron, and P.-A. Farine, "A new FFT-based algorithm for secondary code acquisition for Galileo signals," in *Proc. IEEE Vehicular Technology Conf.*, Sept. 2011, pp. 1 –6.

[82] *TMS320C64x DSP Library Programmer's Reference*, Texas Instruments, Inc., Dallas, TX, Oct. 2003.

[83] S. van Haastregt and B. Kienhuis, "Automated synthesis of streaming C applications to process networks in hardware," in *Proc. Conf. Design Automation Test in Europe*, Nice, France, 2009, pp. 890–893.

[84] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electronic Computers*, vol. EC–8, no. 3, pp. 330–334, Sept. 1959.

[85] G. Wallace, "The JPEG still picture compression standard," *IEEE Trans. Consumer Electronics*, vol. 38, no. 1, pp. 18–34, Feb. 1992.

[86] A. Wang, J. Wang, and B. Xue, "Acquisition of BOC(n, n) with large Doppler," in *Proc. Int. Conf. Wireless Communications, Networking and Mobile Computing*, Sept. 2011, pp. 1 –4.

[87] W. Wang, L. Li, G. Zhang, D. Liu, and J. Qiu, "An application specific instruction set processor optimized for FFT," in *IEEE Int. Midwest Symp. Circuits and Systems*, Aug. 2011, pp. 1–4.

[88] L. Wanhammar, *DSP Integrated Circuits*. San Diego, CA: Academic Press, 1999.

[89] C.-L. Wey, W.-C. Tang, and S.-Y. Lin, "Efficient memory-based FFT architectures for digital video broadcasting (DVB-T/H)," in *Int. Symp. VLSI Design Automation Test*, Apr. 2007, pp. 1–4.

[90] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, July 2003.

[91] L. Yang and X. Xiaoli, “Combining FFT and circular convolution method for high dynamic GPS signal acquisition,” in *Proc. Int. Conf. Electronic Measurement and Instruments*, vol. 2, July 2007, pp. 159–162.

[92] C.-Y. Yu, S.-G. Chen, and J.-C. Chih, “Efficient CORDIC designs for multimode OFDM FFT,” in *Proc. IEEE Int. Conf. Acoustics Speech Signal Process.*, vol. 3, May 2006.

[93] B. Zahiri, “Structured ASICs: Opportunities and challenges,” in *Proc. IEEE Int. Conf. Computer Design*, Oct. 2003, pp. 404–409.

[94] B. Zhang, H. Liu, H. Zhao, F. Mo, and T. Chen, “Domain specific architecture for next generation wireless communication,” in *Design, Automation Test in Europe Conference Exhibition*, Mar. 2010, pp. 1414–1419.

[95] W. Zou and Y. Wu, “COFDM: An overview,” *IEEE Trans. Broadcasting*, vol. 41, no. 1, pp. 1–8, Mar. 1995.

[P1] PUBLICATION 1

T. Pitkänen, R. Mäkinen, J. Heikkinen, T. Partanen, and J. Takala, ”Low-Power, High-Performance TTA Processor for 1024-Point Fast Fourier Transform,” in *Embedded Computer Systems: Architectures, Modeling, and Simulation: Proc. 6th Int. Workshop SAMOS VI*, S. Vassiliadis, S. Wong, T.D. Hämäläinen, Eds. Berlin, Germany: Springer-Verlag, 2006, vol. LNCS 4017, pp. 227–236.

Low-Power, High-Performance TTA Processor for 1024-Point Fast Fourier Transform

Teemu Pitkänen, Risto Mäkinen, Jari Heikkinen, Tero Partanen, and Jarmo Takala

Tampere University of Technology, P.O. Box 553, FIN-33101 Tampere, Finland
{teemu.pitkanen, jari.heikkinen, risto.makinen, jarmo.takala}@tut.fi

Abstract. Transport Triggered Architecture (TTA) offers a cost-effective trade-off between the size and performance of ASICs and the programmability of general-purpose processors. This paper presents a study where a high performance, low power TTA processor was customized for a 1024-point complex-valued fast Fourier transform (FFT). The proposed processor consumes only 1.55 μJ of energy for a 1024-point FFT. Compared to other reported FFT implementations with reasonable performance, the proposed design shows a significant improvement in energy-efficiency.

1 Introduction

Fast Fourier transform (FFT) has an important role in many digital signal processing (DSP) systems. E.g., in orthogonal frequency division multiplexing (OFMD) communication systems, FFT and inverse FFT are needed. The OFMD technique has become a widely adopted in several wireless communication standards. When operating in wireless environment the devices are usually battery powered and, therefore, an energy-efficient FFT implementation is needed. In CMOS circuits, power dissipation is proportional to the square of the supply voltage [1]. Therefore, a good energy-efficiency can be achieved by aggressively reducing the supply voltage [2] but unfortunately this results in lower circuit performance. In this paper, a high performance, low power processor is customized for a 1024-point FFT application. Several optimization steps, such as special function units, code compression, manual code generation, are utilized to obtain the high performance with low power dissipation. The performance and power dissipation are compared against commercial and academic processors and ASIC implementations of the 1024-point FFT.

2 Related Work

Digital signal processors offer flexibility and, therefore, low development costs but at the expense of limited performance and typically high power dissipation. Field programmable gate arrays (FPGA) combine the flexibility and the speed of application-specific integrated circuit (ASIC) [3]. However, FPGAs cannot compete with the energy-efficiency of ASIC implementations. For a specific application, the energy-efficiency between these alternatives can differ by multiple orders of magnitude [4]. In general, FFT processor architectures can be divided into five categories: processors are based on single-port memory, dual-port memory, cached memory, pipeline, or array architecture [5]. In [6], a reconfigurable FFT-processor with single memory based scalable IP core is presented, with radix-2 algorithm. In [7], variable-length FFT processor is designed using pipeline based architecture. It employs radix-2/4/8 single path delay feedback architecture. The proposed processor supports three different transform lengths by bypassing the input to the correct pipeline stage. In [5], cached memory architecture is presented, which uses small cache memories between the processor and the main memory. It offers good energy-efficiency in low voltage mode but with rather low performance. In [8], an energy-efficient architecture is presented, which exploits subtreshold circuits techniques. Again the drawback is the poor performance.

The proposed FFT implementation uses a dual-port memory and the instruction schedule is constructed such that during the execution two memory accesses are performed at each instruction cycle, i.e., the memory bandwidth is fully exploited. The energy-efficiency of the processor matches fixed-function ASICs although the proposed processor is programmable.

3 Radix-4 FFT Algorithm

There are several FFT algorithms and, in this work, a radix-4 approach has been used since it offers lower arithmetic complexity than radix-2 algorithms. The specific algorithm used here is a variation of the in-place radix-4 decimation-in-time (DIT) algorithm and the 4^n-point FFT in matrix form is defined as

$$\begin{aligned} F_{4^n} &= \left[\prod_{s=n-1}^{0} [P^s_{4^n}]^T (I_{4^{n-1}} \otimes F_4) D^s_{4^n} P^s_{4^n} \right] P^{in}_{4^n} \; ; \\ P^s_{4^n} &= I_{4(n-s-1)} \otimes P_{4(s+1),4^s} \; ; \; P^{in}_{4^n} = \prod_{k=1}^{n} I_{4(n-k)} \otimes P_{4^k,4} \; ; \\ P_{K,R}(m,n) &= \begin{cases} 1, \text{iff } n = (mR \bmod K) + \lfloor mR/K \rfloor \\ 0, \text{otherwise} \end{cases} \end{aligned} \tag{1}$$

where $\otimes$ denotes tensor product, P^{in}_N is an input permutation matrix of order N, F_4 is the 4-point discrete Fourier transform matrix, D^s_N is a diagonal coefficient matrix of order N, P^s_N is a permutation matrix of order N, and I_N is the identity matrix of order N. Matrix $P_{K,R}$ is a stride-by-R permutation marix [9] of order K such that the elements of the matrix. In addition, mod denotes the modulus operation and $\lfloor \cdot \rfloor$ is the floor function. The matrix D^s_N contains N complex-valued twiddle factors, W^k_N, as follows

$$D^s_N = \bigoplus_{k=0}^{N/4-1} \operatorname{diag}\left\{ W_{4^{s+1}}^{i(k \bmod 4^s)} \right\} \;,\; i = 0, 1, \ldots, 3 \; ; \; W^k_N = e^{-j2\pi k/N} \tag{2}$$

where j denotes the imaginary unit and $\oplus$ denotes matrix direct sum. Finally, the matrix F_4 is given as

$$F_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix}. \tag{3}$$

4 Transport Triggered Architecture

Transport triggered architecture (TTA) is a class of statically programmed instruction-level parallelism (ILP) architectures that reminds very long instruction word (VLIW) architecture. In the TTA programming model, the program specifies only the data transports (moves) to be performed by the interconnection network [10] and operations occur as “side-effect” of data transports. Operands to a function unit are input through ports and one of the ports is dedicated as a trigger. Whenever data is moved to the trigger port, the operation execution is initiated.

When the input ports are registered, the operands for the operation can be stored into the registers in earlier instruction cycles and a transport to the trigger port starts the operation with the operands stored into the registers. Thus the operands can be shared between different operations of a function unit, which reduces the data traffic in the interconnection and the need for temporary storage in register file or data memory.

A TTA processor consists of a set of function units and register files containing general-purpose registers. These structures are connected to an interconnection network, which connects the input and output ports of the resources. The architecture can be tailored by adding or removing resources. Moreover, special function units with user-defined functionality can be easily included.

5 TTA Processor for Radix-4 FFT

An effective means to reduce power consumption without reducing the performance is to exploit special function units for the operations of the algorithm. These units reduce the instruction overhead, thus they reduce the power consumption due to instruction fetch. Here four custom-designed units tailored for FFT application were used.

The interconnection network consumes a considerable amount of power and, therefore, all the connections from ports of function units and register files to the buses, which are not really needed, should be removed. By removing a connection, the capacitive load is reduced, which reduces also the power consumption. Clock gating technique can be used to reduce the power consumption of non active function units. Significant savings can be expected on units with low utilization.

TTA processors remind VLIW architectures in a sense that they use long instruction words, which implies high power consumption on instruction fetch. This overhead can be significantly reduced by exploiting program code compression.

5.1 Arithmetic Units

Since the FFT is inherently an complex-valued algorithm, the architecture should have means to represent complex data. The developed processor uses 32-bit words and the complex data type is represented such that the 16 most significant bits are reserved for the real part and the 16 least significant bits for the imaginary part. Real and imaginary parts use fractional representation, i.e., one bit for sign and 15 bits for fraction. The arithmetic operations in the algorithm in (1) can be isolated into 4-input, 4-output blocks described as radix-4 DIT butterfly operation defined by the following:

$$(y_0,y_1,y_2,y_3)^T = F_4\left(1,W_1,W_2,W_3\right)^T\left(x_0,x_1,x_2,x_3\right)^T \tag{4}$$

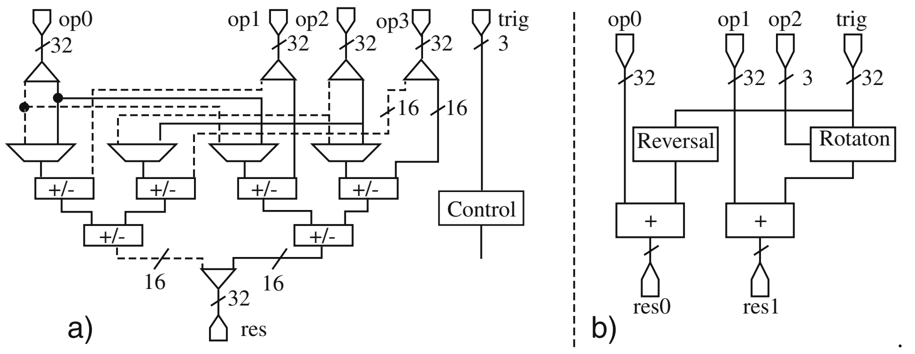

Fig. 1. a) Block diagram of complex adder. Solid lines represent real parts and dotted lines imaginary parts. ports op1-3 are operand ports and trigger port defines the operation. b) Block diagram of data address generator. op0: Base address of input buffer. op1: Base address of output buffer. op2: Butterfly column index. trig: Element index, trigger port. res0: Resulting address after field reversal. res1: Resulting address after index rotation.

where x_i denotes an input operand, W_i is a twiddle factor, and y_i is an output operand. One of the special function units in our design is complex multiplier, CMUL, which is a standard unit containing four 16-bit real multipliers and two 16-bit real adders. When the operand to the CMUL unit is a real one, i.e., multiplication by one, the other operand is directly bypassed to the result register. The CMUL unit is pipelined and the latency is three cycles. The butterfly operation contains complex additions defined by (3). In this work, we have defined a four-input, one-output special function unit, CADD, which supports four different summations according to each row in F_4. The motivation is that, in a TTA, the instruction defines data transports, thus by minimizing the transports, the number of instructions can be minimized. Each of the four results defined by F_4 are dependent on the same four operands, thus once the four operands have been moved into the input registers of the function unit, four results can be computed simply by performing a transport to trigger register, which defines the actual function out of the four possible complex summations. The block diagram of the CADD unit is illustrated in Fig. 1 a).

5.2 Address Generation

The N-point FFT algorithm in (1) contains two type of data permutations: input permutation of length N and variable length permutations between the butterfly columns. In-place computations require manipulation of indices into data buffer. Such manipulations are low-power if performed in bit-level. If the 4^n input operands are stored into a buffer in-order, the read index to the buffer, i.e., operand for the butterfly operation, can be obtained by bit field reversal. This reminds the bit reversal addressing in radix-2 algorithms but, instead of reversing single bits, here 2-bit fields are reversed [11], i.e., a $2n$-bit read index $r = (r_{2n-1}r_{(2n-2}\dots r_0)$ is formed from an element index (a linear counter) $a = (a_{2n-1}a_{2n-2}\dots a_0)$ as

$$r_{2k} = a_{2n-2k-2}\ ;\ r_{2k+1} = a_{2n-2k-1}\ ,\ 0 \le k < n \qquad (5)$$

This operation is implemented simply with wiring. In a similar fashion, the permutations between the butterfly columns can be realized in bit-level simply by rotation of two bits to the right. However, the length of the bit field to be rotated is dependent on the butterfly column index, s, in (1). The $2n$-bit read index $p = (p_{2n-1}p_{(2n-2} \cdots p_0)$ is formed from the element index a as [11]

$$\begin{cases} p_{2k} = a_{(2k+2s) \bmod 2(s+1)}, 0 \leq k \leq s \\ p_{2k+1} = a_{(2k+1+2s) \bmod 2(s+1)}, 0 \leq k \leq s \\ p_{2k} = a_{2k}, s < k < n \\ p_{2k+1} = a_{2k+1}, s < k < n \end{cases} . \tag{6}$$

Such an operation can be easily implemented with the aid of multiplexers. When the generated index is added to the base address of the memory buffer, the final address to the memory is obtained. The block diagram of the developed AG unit is shown in Fig 1 b). The input ports of the AG units are registered, thus the base addresses of input and output buffers need to be store only once into operand ports op0 and op1, respectively. The butterfly column index is stored into operand port op2 and the address computation is initiated by moving an index to trigger port. Two results are produced: output port res0 contains the address according to input permutation and port res1 according to bit field rotation.

5.3 Coefficient Generation

A coefficient generator (COGEN) unit was developed for generating the twiddle factors, which reduces power consumption compared to the standard method of storing the coefficients as tables into data memory. In an radix-4 FFT, there are $Nlog_4(N)$ twiddle factors as defined by (2) but there is redundancy. It has been be shown that all the twiddle factors can be generated from $N/8+1$ coefficients [12] with the aid of simple manipulation of the real and the imaginary parts of the coefficients. The COGEN unit is based on a table where the $N/8+1$ are stored. This table is implemented as hard wired logic for reducing the power consumption. The unit contains an internal address generator, which creates the index to the coefficient table based on two input operands: butterfly column index $(s = 0, 1, \ldots, n-1)$ and element index $(a = 0, 1, \ldots, 4^n - 1)$. The obtained index is used to access the table and the real and imaginary parts of the fetched complex number are modified by six different combinations of exchange, add, or subtract operations depending on the state of input operands. The resulting complex number is placed in the output register as the final twiddle factor.

5.4 General Organization

The general organization of the proposed TTA processor tailored for FFT (FFTTA) processor is presented in Fig. 2. The processor is composed of eight separate function units and a total of 11 register files containing 23 general-purpose registers. The function units and register files are connected by an interconnection network (IC) consisting of 18 buses and 61 sockets. In addition, the FFTTA processor contains a control unit, instruction memory, and dual-ported data memory. The size of the data memory is 2048 words of 32 bits implying that 32-bit data buses are used. There is one 1-bit bus, which is used for transporting the Boolean values.

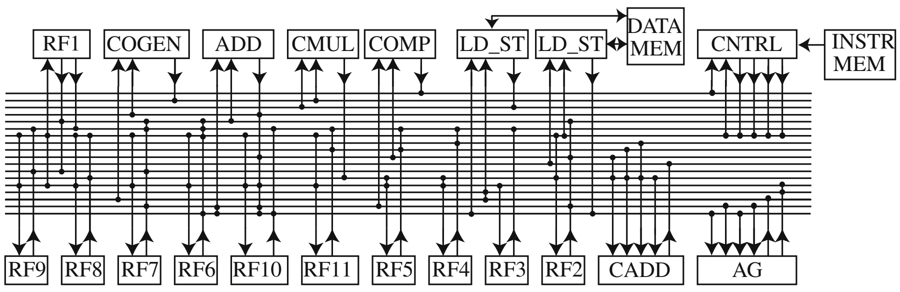

Fig. 2. Architecture of the proposed FFTTA processor. CADD: Complex adder. CMUL: Complex multiplier. AG: Data address generator. COGEN: Coefficient generator. ADD: Real adder. LD_ST: Load-store unit. COMP: Comparator unit. CNTRL: Control unit. RFx: Register files, containing total of 23 general purpose registers.

5.5 Instruction Schedule

In principle, an 4^n-point radix-4 FFT algorithm in (1) contains two nested loops: an inner loop where the butterfly operation is computed $4^{(n-1)}$ times and an outer loop where the inner loop is iterated n times. Each butterfly operation requires four operands and produces four results. Therefore, in a 1024-point FFT, a total of 10240 memory accesses are needed. If a single-port data memory is used, the lower bound for the number of instruction cycles for a 1024-FFT is 10240. If a dual-port memory is used, the lower bound is 5120 cycles.

In order to maximize the performance, the inner loop kernel needs to be carefully optimized. Since the butterfly operations are independent, software pipelining can be applied. In our implementation, the butterfly operations are implemented in a pipelined fashion and several butterflies at different phases of computation are performed in parallel. The developed 1024-point FFT code follows the principal code in Fig. 3.

In initialization, pointers and loop counters, i.e., butterfly and element indices, are set up. The input data is stored in order into data memory buffer. Another 1024-word buffer is reserved for intermediate storage and the final result. There is no separate code performing the input permutation but the address generation unit is used to access the input buffer in correct order with an address obtained from port res0 of AG in Fig.1b). The results of the first butterfly column are stored into the intermediate buffer with an address obtained from port res1 of AG. All the accesses to the intermediate buffer are done by using addresses from port res1 of AG.

In the prologue, the butterfly iterations are started one after each other and, in the actual inner loop kernel, four iterations of butterfly kernels are performed in parallel in pipelined fashion. The loop kernel evaluates also the loop counter. In the epilogue, the last butterfly iterations are completed and the loop counter of the outer loop is evaluated. The kernel contains the functionality of butterfly operations, which requires four triggers for memory reads and memory writes and corresponding address computations, four triggers for complex multiplier and four triggers for CADD unit. Since the branch latency is three cycles, the kernel can actually be implemented with four instructions. However, this approach results in a need for moving variables from an register to another. The reason is that parallel butterfly iterations need more than four intermediate

```
main() {
  initialization(); /* 9 instr. */
  for(stage=0; stage<5; stage++) {
    prologue(); /* 16 instr. */
    for(k=0; k<84; k++)
      kernel(); /* 12 instr. */
    epilogue; /* 21 instr. */
  }
}
```

Fig. 3. Pseudocode illustrating structure and control flow of program code

results, which need to be stored into register files. Since there is no mechanism to dynamically index the register accesses, the only way is to use the register files as first-in-first-out buffers. Such register copies introduce additional power consumption, in particular, since the moves require additional buses and increase the register activity.

The final implementation of the kernel was 12 instructions and by that way, it was possible to keep the intermediate results in a dedicated register without need to copy the values. This resulted significant savings in power consumption at the expense of lengthening the program code by eight instructions. The parallel code for 1024-point FFT contains a total of 58 instructions and the instruction length was 162 bits. The program spends 96% of the execution time in the kernel. The execution of 1024-point FFT takes 5234 instruction cycles, thus the overhead to the theoretical lower bound with dual-port data memory (5120 cycles) is only 2% (114 cycles). This overhead is negligible compared to overheads seen in typical software implementations.

5.6 Code Compression

TTA suffers from poor code density, which is mostly due to minimal instruction encoding that is used to simplify decoding. Minimal instruction encoding leads to long instruction words. The long instruction word consists of dedicated fields, denoted as move slots. Each move slot specifies a data transport on a bus. Each move slot consists of three fields: guard, destination ID, and source ID. The guard provides means for conditional execution. The destination ID specifies the address of a socket that is reading data from a bus. The source ID specifies the address of a socket that is writing data on a bus. In addition to move slots, instruction words may contain dedicated long immediate fields to define large constant values, e.g., for jump addresses.

The poor code density can be improved by compression. Compression also results in reduced power consumption as fewer bits need to be fetched from the program memory. Dictionary-based compression is one of the simplest compression approaches to improve the code density [13]. Dictionary-based program compression stores all unique bit patterns into a dictionary and replaces them in the program code with code words to the dictionary. Given a program with N unique instructions, the length of the code word is $\lceil log_2|N|\rceil$ bits. During execution, the code word, fetched from the program memory is used to obtain the original instruction from the dictionary for decoding.

In order to reduce the power consumption of the FFTTA processor and improve the code density, dictionary-based program compression was applied. All the unique instructions of the program code were stored into a dictionary and replaced with indices pointing to the dictionary. This resulted in decrease in the width of the program memory from 162 bits to 6 bits. The decompression, i.e., the dictionary access was supplemented to the control unit without additional pipeline stage. The actual dictionary(8586 bits) was implemented using standard cells.

6 Performance Analysis

In order to analyse the characteristics of the FFTTA processor, the structures of the previous special function units were described manually in VHDL. The structural description of the FFTTA core was obtained with the aid of the hardware subsystem of the MOVE Framework [14], which generated the VHDL description.

Then the FFTTA was synthesized to a 130nm CMOS standard cell ASIC technology with Synopsys Design Compiler. This was followed by a gate level simulation at 250 MHz. Synopsys Power Compiler was used for the power analysis. The obtained results are listed in Table 1. It should be noted that the instruction and data memories take 40% of the total power consumption of 74mW with 1.5V supply voltage. If the supply voltage is reduced to 1.1V, the total power consumption will drop down to about 40 mW. However, this will reduce the maximum clock frequency.

Table 1. Characteristics of 1024-point FFT on FFTTA processor on 130 nm ASIC technology with 1.5V supply voltage

Clock Cycles	**5234**	Execution Time	**20.94 μs**	Power	**74 mW**
Clock Frequency	**250 MHz**	Area	**140 kgates**	Energy	**1.55 μJ**

Table 2 presents how many 1024-point FFT transforms can be performed with energy of 1 mJ. The results are presented for ten different implementations of the 1024-point FFT. For some implementations there are different operating voltage or clock frequency points listed. Spiffee processor [5] employs a high performance architecture and low supply voltages and it's dedicated for the FFT. The StrongArm SA-1100 processor [15] employs custom circuits, clock gating, and reduced supply voltage. The Stratix [16] is an FPGA solution with dedicated embedded FFT logic usign Altera Megacore function. The TI C6416 [17] is a digital signal processor and the Imagine [18] is a media processor. They were both created using pseudo-custom data path tiling. In addition, the TI C6416 employs pass-gate multiplexer circuits. The 1024-point FFT with radix-4 algorithm can be computed in 6002 cycles in TI C6416 when using 32-bit complex words (16 bits for real and imaginary parts) [19]. However, in-place computations cannot be used and the processor has eight memory ports while the FFTTA uses only two. The Intel Pentium-4 [20] is a standard general-purpose microprocessor. Rest of the processors are dedicated for the FFT. The custom scalable IP core Zhao [6], employs single memory architecture with clock gating. The custom variable-length Lin [7] FFT-processor employs radix-2/4/8 single-path delay algorithm. MIT FFT uses subtreshold circuit techniques [8].

Table 2. The number of 1024-point FFTs performed with a unit of energy

Design	Tech. [nm]	Oper. voltage [V]	Clock freq. [MHz]	Exec. time [μs]	FFT/mJ	Design	Tech. [nm]	Oper. voltage [V]	Clock Freq. [MHz]	Exec. time [μs]	FFT/mJ
FFTTA	130	1.5	250	20.9	645		130	1.2	720	8.34	100
	600	1.1	16	330	319	TI C6416	130	1.2	600	10.0	167
Spiffee	600	2.5	128	41	67		130	1.2	300	21.7	250
	600	3.3	173	40	39	MIT FFT	180	0.35	0.01	250000	6452
SA-110	350	2	74	425.7	60		180	0.9	6	430.6	1428
	130	1.3	275	4.7	241	Lin	350	3.3	45.45	22.5	93
Stratix	130	1.3	133	9.7	173		350	2.3	17.86	57	133
	130	1.3	100	12.9	149	Zhao	180	-	20	281.6	43
Imagine	150	1.5	232	16.0	16	Intel P4	130	1.2	3000	23.9	0.8

Compared to other FFT designs the proposed FFTTA processor shows significant energy-efficiency. Only the MIT FFT outperforms the FFTTA. However, due to its long execution time, the MIT FFT is not usable in high performance designs. The performance of the FFTTA processor is still quite feasible although it does not provide the best performance. However, the performance can be scaled, i.e., the execution time can be halved by doubling the resources and memory ports. The memory size remains constant and it can be estimated that the energy-efficiency remains the same in terms of FFTs per energy unit.

7 Conclusions

In this paper, a low-power application-specific processor for FFT computation has been described. The resources of the processor have been tailored according to the needs of the application consisting of eight function units and 11 register files. Several methods for reducing the power consumption of the processor were utilized: clock gating, special function units, and code compression. The processor was synthesized on a 130 nm ASIC technology and power analysis showed that the proposed processor has both high energy-efficiency and high performance.

The described processor has limited programmability but the purpose of this experiment was to prove the feasibility and potential of the proposed approach. However, the programmability can be improved by introducing additional function units and loosening the code compression. In addition, different transform sizes can be supported by modifying the address generators and twiddle factor unit. This modifications are mainly addition of multiplexers, thuse significant increase in power consumption is not expected. In addition, the performance of the processor can be improved by adding computational resources implying need for higher data memory bandwidth

Acknowledgement

This work has been supported by the Academy of Finland under project 205743 and the National Technology Agency of Finland under research funding decision 40153/05.

References

1. Weste, N., Eshraghian, K.: Principles of CMOS VLSI Design: A Systems Perspective. Addison-Wesley, Reading, MA (1985)
2. Chandrakasan, A., Sheng, S., Brodersen, R.: Low-power CMOS digital design. IEEE Journal of Solid State Circuits **27** (1992) 473–483
3. Reeves, K., Sienski, K., Field, C.: Reconfigurable hardware accelerator for embedded DSP. In Schewel, J., Athanas, P.M., Bove, V.M., Watson, J., eds.: Proc. SPIE High-Speed Comp. Dig. Sig. Proc. Filtering Using Reconf. Logic. Volume 2914., Boston, MA (1996) 332–340
4. Chang, A., Dally, W.: Explaining the gap between ASIC and custom power: A custom perspective. In: Proc. IEEE DAC, Anaheim, CA (2005) 281–284
5. Baas, B.M.: A low-power, high-performance, 1024-point FFT processor. IEEE Solid State Circuits **43** (1999) 380–387
6. Zhao, Y., Erdogan, A., Arslan, T.: A low-power and domain-specific reconfigurable fft fabric for system-on-chip applications. In: Proc. 19th IEEE Parallel and Distrubuted Prosessing Symp. Reconf. Logic, Denver, CO (2005)
7. Lin, Y.T., Tsai, P.Y., Chiueh, T.D.: Low-power variable-length fast fourier transform processor. Proc. IEE Computers and Digital Techniques **152** (2005) 499–506
8. Wang, A., Chandrakasan, A.: A 180-mV subthreshold FFT processor using a minimum energy design methodology. IEEE J. Solid State Circuits **40** (2005) 310–319
9. Granata, J., Conner, M., Tolimieri, R.: Recursive fast algorithms and the role of the tensor product. IEEE Trans. Signal Processing **40** (1992) 2921–2930
10. Corporaal, H.: Microprocessor Architectures: From VLIW to TTA. John Wiley & Sons, Chichester, UK (1997)
11. Mäkinen, R.: Fast Fourier transform on transport triggered architectures. Master's thesis, Tampere Univ. Tech., Tampere, Finland (2005)
12. Wanhammar, L.: DSP Integrated Circuits. Academic Press, San Diego, CA (1999)
13. Lefurgy, C., Mudge, T.: Code compression for DSP. Technical Report CSE-TR-380-98, EECS Department, University of Michigan (1998)
14. Corporaal, H., Arnold, M.: Using transport triggered architectures for embedded processor design. Integrated Computer-Aided Eng. **5** (1998) 19–38
15. Intel: StrongARM SA-110 Microprocessor for Portable Applications Brief Datasheet. (1999)
16. Lim, S., Crosland, A.: Implementing FFT in an FPGA co-processor. In: The International Embedded Solutions Event (GPSx), Santa Clara, CA (2004) 230–233
17. Agarwala, S., Anderson, T., Hill, A., Ales, M., Damodaran, R., Wiley, P., Mullinnix, S., Leach, J., Lell, A., Gill, M., Rajagopal, A., Chachad, A., Agarwala, M., Apostol, J., Krishnan, M., Duc-Bui, Quang-An, Nagaraj, N., Wolf, T., Elappuparackal, T.: A 600 MHz VLIW DSP. IEEE J. Solid State Circuits **37** (2002) 1532–1544
18. Rixner, S., Dally, W., Kapasi, U., Khailany, B., Lopez-Lagunas, A., Mattson, P., Owens, J.: A bandwidth-efficient architecture for media processing. In: Proc. Annual ACM/IEEE Int. Symp. Microarchitecture, Dallas, TX (1998) 3–13
19. Texas Instruments, Inc. Dallas, TX: TMS320C64x DSP Library Programmer's Reference. (2003)
20. Deleganes, M., Douglas, J., Kommandur, B., Patyra, M.: Designing a 3 GHz, 130 nm, Intel® Pentium ®4 processor. In: Digest of Technical Papers Symp. VLSI Circuits, Honolulu, HI (2002) 230–233

[P2] PUBLICATION 2

T. Pitkänen, R. Mäkinen, J. Heikkinen, T. Partanen, and J. Takala, ”Transport Triggered Architecture Processor for Mixed-Radix FFT,” in *Proceedings of the Fortieth Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, Oct. 29–Nov. 1, 2006, pp. 84–88.

[P3] PUBLICATION 3

T. Pitkänen, T. Partanen, and J. Takala, ”Low-Power Twiddle Factor Unit for FFT Computation,” in *Embedded Computer Systems: Architectures, Modeling, and Simulation: Proc. 7th Int. Workshop SAMOS VII*, S. Vassiliadis, M. Berekovic, T.D. Hämäläinen, Eds. Berlin, Germany: Springer-Verlag, 2007, vol. LNCS 4599, pp. 233–240.

Low-Power Twiddle Factor Unit for FFT Computation

Teemu Pitkänen, Tero Partanen, and Jarmo Takala

Tampere University of Technology, P.O. Box 553, FIN-33101 Tampere, Finland
{teemu.pitkanen, tero.partanen, jarmo.takala}@tut.fi

Abstract. An integral part of FFT computation are the twiddle factors, which, in software implementations, are typically stored into RAM memory implying large memory footprint and power consumption. In this paper, we propose a novel twiddle factor generator based on reduced ROM tables. The unit supports both radix-4 and mixed-radix-4/2 FFT algorithms and several transform lengths. The unit operates at a rate of one factor per clock cycle.

1 Introduction

Fast Fourier transform (FFT) has gained popularity lately due to the fact that OFDM has been used in several wireless and wireline communication systems, e.g., IEEE 802.11a/g, 802.16, VDSL, and DVB. An integral part of the FFT computation are the twiddle factors, which, in software implementations, are typically stored into RAM memory implying large memory footprint. The twiddle factors can be generated at run-time. A traditional method is to use CORDIC as described, e.g., in [1]. The sine and cosine values are needed in direct digital frequency synthesizers and often the generation is based on polynomials, e.g., in [2]. An other approach is to use a function generator based on recursive feedback difference equation [3,4]. Typically these approaches result in smaller area than memory based approaches. However, since the computation is done at run-time, there is a huge amount of transistor switching implying higher power consumption in CMOS implementations.

Another approach is to store the twiddle factors into a ROM table. In an N-point FFT, there are $N/2$ different twiddle factors and an approach exploiting this property has been reported in [5]. Methods requiring only $N/4$ coefficients to be stored into a table are described in [6,7]. There is, however, even more redundancy since the real and imaginary parts of the factors are sine values and $N/8+1$ complex coefficients are needed to reconstruct all the factors for an N-point FFT [8]. In [9], a coefficient manipulation method is presented where only $N/8+1$ coefficients are needed to generate the twiddle factors. However, the previous methods are designed only for radix-2 algorithms containing more arithmetic operations than radix-4 algorithms.

A twiddle factor generator unit could be used as a special function unit in an application-specific instruction-set processor (ASIP) but it may not increase the performance of the software implementation. Often several instructions are needed to compute the correct index to the unit. Considerable performance increase can be expected, if the unit can also perform the index modifications to avoid additional instructions. However, the indexing of the twiddle factors varies depending on the FFT variant. More detailed discussion on twiddle factor indexing can be found from [10].

In this paper, we propose a low-power twiddle factor unit based on a ROM table. The proposed work differs from the related work such that the proposed unit a) supports radix-4 and mixed-radix-4/2 FFT algorithms, b) supports several transform sizes (power-of-two), and c) integrates index manipulation. By supporting radix-4 algorithms, the performance of FFT computation is increased significantly compared to radix-2 algorithms. In addition, the overhead of address manipulation in software implementation is omitted, which increases the performance even more. The unit can generate factors at a rate of one factor per clock cycle. The proposed unit has already been used in the FFT implementations described in our previous work [11,12] but here the twiddle factor generation is described in detail.

2 FFT Algorithms

In this work, we have used the traditional in-place radix-4 decimation-in-time (DIT) radix FFT algorithm with in-order-input, permuted output as given, e.g., in [13]. In this work, we would like to expose the different permutations, thus we formulate the traditional algorithm in the following fashion:

$$F_{2^{2n}} = R_{2^{2n}} \left[\prod_{s=n-1}^{0} [O^s_{2^{2n}}]^T (I_{2^{(2n-2)}} \otimes F_4) A^s_{2^{2n}} O^s_{2^{2n}} \right] ; \tag{1}$$

$$F_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix} ; \tag{2}$$

$$R_{2^{2n}} = \prod_{k=2}^{n} I_{2^{(2n-2k)}} \otimes P_{2^{2k},4} ; \tag{3}$$

$$O^s_{2^m} = I_{4^s} \otimes P_{2^{(m-2s)},2^{(m-2s-2)}} \tag{4}$$

where j is the imaginary unit, I_n is an identity matrix of order n, and the permutation matrices R_N and O_N are based on stride-by-S permutation matrices [14] $P_{N,S}$ defined as

$$[P_{N,S}]_{mn} = \begin{cases} 1, \text{ iff } n = (mS \bmod N) + \lfloor mS/N \rfloor \\ 0, \text{ otherwise} \end{cases} , \; m,n = 0,1,\ldots,N-1 \tag{5}$$

The matrix A^s_N contains N twiddle factors $W^k_K = e^{j2\pi k/K}$ as follows

$$A^s_N = Q^s_N \left[\bigoplus_{b=0}^{N/4-1} \operatorname{diag}\left(W^0_{4^{s+1}}, W^{\lfloor \frac{b4^{s+1}}{N} \rfloor}_{4^{s+1}}, W^{2\lfloor \frac{b4^{s+1}}{N} \rfloor}_{4^{s+1}}, W^{3\lfloor \frac{b4^{s+1}}{N} \rfloor}_{4^{s+1}} \right) \right] ; \tag{6}$$

$$Q^s_N = \prod_{l=0}^{s} P_{4^{(s-l)},4} \otimes I_{N/4^{(s-l)}} . \tag{7}$$

Examples of signal flow graphs of this algorithm are depicted in Fig. 1a) and 1c).

As the Fig. 1 shows the output data is not in order, thus to give it in order, input permutation is needed at each column and it complicates the index modifications in the coefficient generator.

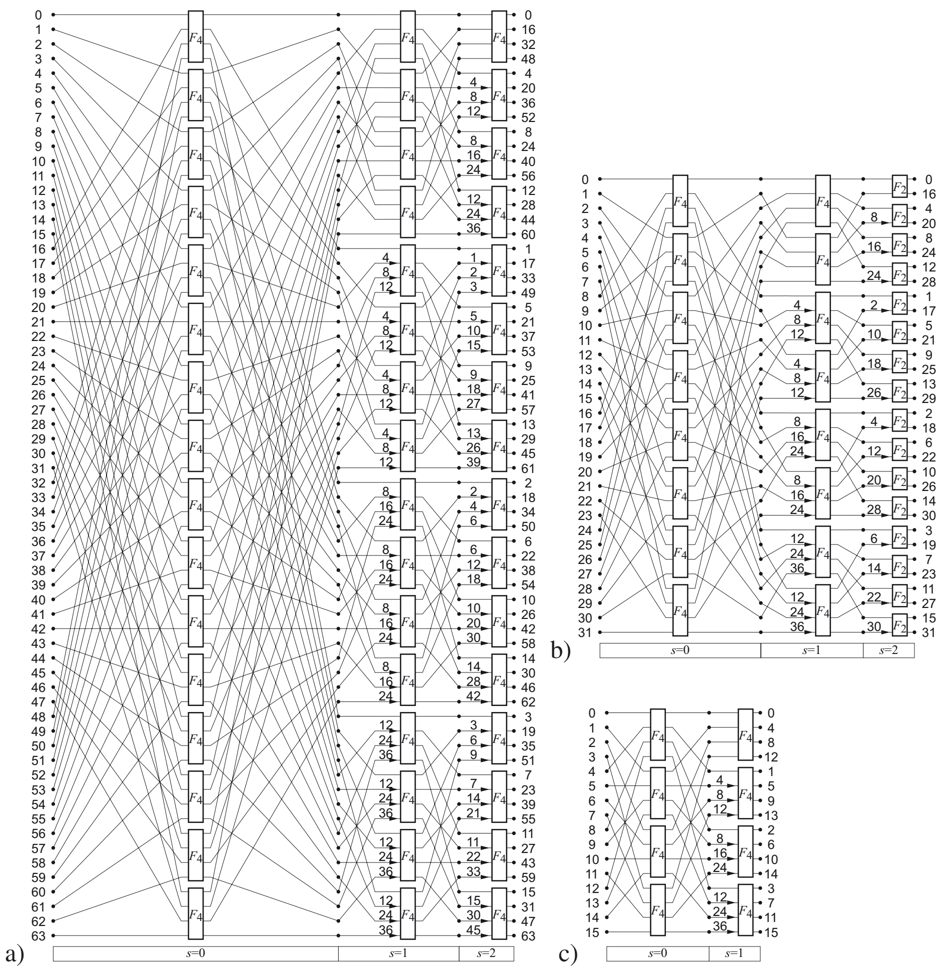

Fig. 1. Signal flow graph of a) 64-point radix-4, b) 32-point mixed-radix, and c) 16-point radix-4 FFT. A constant k in the signal flow graph represents a twiddle factor W_{64}^{k}.

The radix-4 algorithms can be used only when the FFT size is a power-of-four. Power-of-two transforms can be supported by using mixed-radix approach and a mixed-radix-4/2 FFT consists of radix-4 processing columns followed by a single radix-2 column as follows

$$F_{2^{2n+1}} = S_{2^{(2n+1)}} \left(I_{4^n} \otimes F_2\right) B_{2^{(2n+1)}} \cdot \left[\prod_{s=n-1}^{0} [O^s_{2^{(2n+1)}}]^T \left(I_{2^{(2n-1)}} \otimes F_4\right) A^s_{2^{(2n+1)}} O^s_{2^{(2n+1)}} \right] \; ; \; F_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{8}$$

where the matrices O^s_N and A^s_N are defined in (4) and (6), respectively. The matrix S_N is a permutation matrix given as

$$S_N = \left(I_2 \otimes R_{4^n}\right) P_{N,2} \; , N = 2^{2n+1}. \tag{9}$$

The matrix B_N contains the twiddle factors for the radix-2 processing column and it is defined as

$$B_N = Q_N^{\log_4(N/2)} \bigoplus_{b=0}^{N/2-1} \operatorname{diag}\left(W_N^0, W_N^b\right) ,N = 2^{2n+1} \tag{10}$$

where the permutation matrix Q_N^s is defined in (7). Example of signal flow graph of the mixed-radix-4/2 algorithm is shown in Fig. 1b).

3 Twiddle Factor Access

Our objective is to design a unit, which can generate twiddle factors for several power-of-two size transforms. By investigating the structure of the twiddle factors in FFTs of different size, we find that the twiddle factors of a shorter transform are included in the larger transform. Our approach is based on lookup tables (LUT) containing the twiddle factors, thus we need to define the maximum FFT size supported, $N_{max} = 2^{n_{max}}$, and the twiddle factors for shorter transforms can be generated from the same LUT.

The unit generates a twiddle factor based on index from an iteration counter, which may be updated by software, if the unit is used as a special function unit in a processor. When targeting to an application-specific fixed-function FFT processor, the iteration counter is the counter, which used to generate all the control signals in the architecture.

An N-point radix-4 FFT contains $\log_4(N)$ iterations of butterfly columns divided into $N/4$ four-input radix-4 butterfly computations while, in a mixed-radix-4/2 algorithm, $\log_4(N/2)$ iterations of $N/4$ radix-4 computations is followed by $N/2$ two-input radix-2 computations. Therefore, we need $\log_2(N)$ bits to identify each butterfly input in a butterfly column and $\lceil \log_2(\log_4(N)) \rceil$ bits to express the butterfly column, i.e., s in definitions (1) and (8).

The input operands for the unit are the iteration counter and parameter indicating the transform size. Let us denote the $(\lceil \log_2(\log_4(N)) \rceil + \log_2(N))$-bit iteration counter by $c = (c_{\lceil \log_2(\log_4(N)) \rceil + \log_2(N) - 1}, \ldots, c_1, c_0)^T$. The transform size is indicated by parameter $f = \log_2(N_{max}) - \log_2(N)$. The structure of the proposed function unit is discussed in the following sections with an example design supporting FFT sizes of 16, 32, and 64. In this example case, the input parameter f can have values 0, 1, or 2 to indicate FFT sizes 64, 32, or 16, respectively. A 5-bit iteration counter c is used when FFT size is 16 and, for a 64-point FFT, an 8-bit counter is needed. The block diagram of the example design is illustrated in Fig. 2. The input parameters are written into registers f and c and the final twiddle factor is obtained from the output registers.

3.1 Scaling

In order to minimize the bit-level shifts due to different transform sizes, we first shift the iteration counter c to the left by the number of bits indicated by the parameter f. This implies that after the shift we obtain a bit-field where the $\lceil \log_2(\log_4(N_{max})) \rceil + \log_2(N_{max})$ bits indicate the butterfly column s and the $\log_2(N_{max}) = n_{max}$ least significant bits contain the index of the twiddle factor in the column to be generated. Let us denote this part by $d = (d_{n_{max}-1}, \ldots, d_0)^T$. However, the actual index is in the most significant bits index and d contains $(n_{max} - \log_2(N))$ zeros in the least significant bits. The rest of the operation is based on operands s and d.

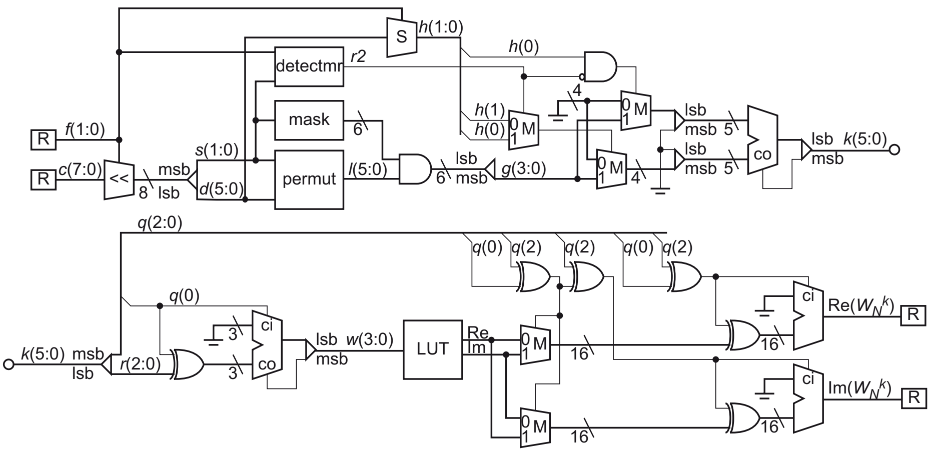

Fig. 2. Block diagram of twiddle factor generator supporting transform sizes of 16, 32, and 64. R: register. M: multiplexor. co: carry out. ci: carry in. <<: left shifter.

3.2 Permutation

The order of twiddle factors depends on the FFT algorithm and, in this work, we concentrate on the in-order input, permuted output FFTs given in (1) and (8). In these particular cases, we need to consider the implementation of matrices A_N^s and B_N^s defined in (6) and (10), respectively.

Our approach is based on index modifications, thus first we need to perform the permutation Q_N^s in (7). The the permutation can be investigated by considering the bit-level rotations as discussed in [15,16]. This shows that the permutation is actually the traditional bit-reversed permutation but here 2-bit fields are used instead of a single bit. It should also be noted that the permutation varies according to the butterfly column s. The permutation in our case is actually independent on the transform size, since we have shifter the index earlier, thus the permuted index, $l = (l_{\log_2(N)}, \ldots, l_1, l_0)$, of an N-point FFT can be expressed in bit-level in matrix form as follows

$$l = \begin{pmatrix} \bar{I}_s \otimes I_2 & \\ & I_{n_{max}-2s} \end{pmatrix} d \;\; ; \;\; \bar{I}_s = \begin{pmatrix} & & & 1 \\ & & \cdot & \\ & \cdot & & \\ 1 & & & \end{pmatrix} \tag{11}$$

where $\bar{I}_m$ is an antidiagonal matrix of order m. The bit-level permutations in a general case are illustrated in Fig. 3. In our example case, the permutations are performed in the block "permut" and the first two, i.e. the maximum butterfly column s is 2, permutations from Fig. 3 are needed.

3.3 Lookup Table Index

Our approach is to store the twiddle factors to a lookup table and the indexing into the table is based on the exponent k in the twiddle factor W_N^k as defined in (6). Different

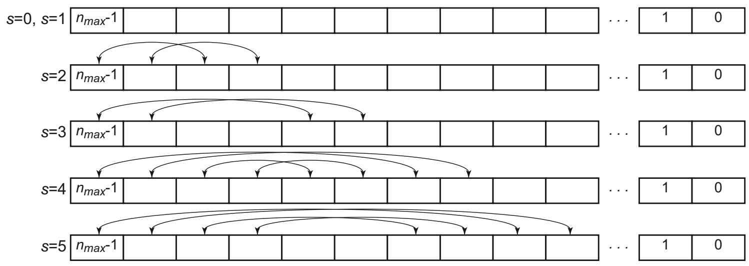

Fig. 3. Examples of bit-level index permutations according to (11)

values of k in FFTs can also be seen in Fig. 1. By exploiting the property $W_{aN}^{a} = W_N$, we can express the twiddle factors as powers of $W_{N_{max}}$. The twiddle factors for radix-4 algorithm are defined in (6) and we can rewrite this equation as follows:

$$A_N^s = Q_N^s \bigoplus_{b=0}^{N} W_{N_{max}}^{(b \bmod 4)\lfloor \frac{\lfloor b/4 \rfloor 4^{s+1}}{N} \rfloor \frac{N_{max}}{4^{s+1}}} \tag{12}$$

where mod is the modulus operation. Here we need an equation for the exponent k for factor $W_{N_{max}}^{k}$, which can be found from the previous. In addition, we have used a shifted index and, therefore, the index b can be replaced with the permuted index l from (11) by the relation $l = bN_{max}/N$, thus we obtain

$$k = (b \bmod 4)\lfloor \frac{\lfloor b/4 \rfloor 4^{s+1}}{N} \rfloor \frac{N_{max}}{4^{s+1}} = \left[\lfloor \frac{lN}{N_{max}} \rfloor \bmod 4 \right] \left[\lfloor \frac{\lfloor l/4 \rfloor 4^{s+1}}{N_{max}} \rfloor \frac{N_{max}}{4^{s+1}} \right]. \tag{13}$$

We may denote the first term as h and the second as g. Then the operation at bit-level representation can defined as follows:

$$k = hg \; ; \; g = \begin{pmatrix} I_{2s} & \\ & 0_{n_{max}-2s-2, n_{max}-2s} \end{pmatrix} l \; ; \; h = (0_{2,n_{max}-f-2}, I_2, 0_{2,f})\, l \tag{14}$$

where $0_{n,m}$ is an $n \times m$ matrix containing zeros and f is the input operand defining the index shift, $f = n_{max} - \log_2(N)$. In the example case in Fig. 2, the block "mask" generates a 6-bit mask, where the $2s$ most significant bits are ones and the rest are zeros. This is used to mask the 6-bit permuted index l. Then the two least significant bits are omitted and the 4-bit result is passed to multiplication with h. Since h is a 2-bit variable, a simple solution is to us adder, where the same operand is fed but the second one is shifted one bit to right, i.e., multiplied by two. Multiplexers can be used to feed either the operand or zero to the adder and these multiplexers are controlled by the multiplicand h.

The 2-bit variable h needs to be extracted from l with the aid of multiplexer controlled with f. Figure 2 indicates that h can be extracted also from d, which shortens the critical path. The block "S" performs the extraction, i.e., $h = (h_1, h_0)^T = (d_{f+1}, d_f)^T$.

In the last butterfly column of mixed-radix-4/2 algorithm, the twiddle factors have a bit different form and by using the fact that, in the last column, $s = \log_4(N/2)$ we may rewrite (10) as follows:

$$B_N = Q_N^s \bigoplus_{b=0}^{N} W_{N_{max}}^{(b \bmod 2)\lfloor \frac{\lfloor b/2 \rfloor 4^{(s+1)}}{N} \rfloor \frac{N_{max}}{4^{(s+1)}}} . \tag{15}$$

By following the procedure used to define the exponent k in radix-4 case, we can define k in this case as follows

$$k = (b \bmod 2)\lfloor \frac{\lfloor b/2 \rfloor 4^{s+1}}{N} \rfloor \frac{N_{max}}{4^{s+1}} = \left[\lfloor \frac{lN}{N_{max}} \rfloor \bmod 2 \right] \left[\lfloor \frac{\lfloor \frac{l}{2} \rfloor 4^{s+1}}{N_{max}} \rfloor \frac{N_{max}}{4^{s+1}} \right] . \tag{16}$$

By comparing this equation to (13), we find that there is a scaling difference in the second term and, if the same hardware is used to generate exponent for both radix-4 and mixed-radix-4/2 twiddle factors, this needs to be compensated. In bit-level representation, this can be defined as

$$k = 2hg \; ; \; h = (0_{1,n_{max}-f-1}, 1, 0_{1,f})\, l \tag{17}$$

where g is obtained as in (14). The example case in Fig. 2 shows a block "detectmr", which is used to detect when mixed-radix algorithm is used and the twiddle factors are for the last butterfly column consisting of the radix-2 butterflies. In this case, the least significant bit of f can be used to detect the mixed-radix transform and the detection of the last butterfly column is detected with the aid of hard-coded detection. Signal "r2" is active-high, which masks the signal "h(1)" since h is only a 1-bit parameter. In addition, the operand g is directed to the lower input of the adder, where the operand is shifted one bit to the left, thus the additional multiplication by two is realized.

3.4 Memory Reduction

Here we propose a method to reconstruct twiddle factors for radix-4 and mixed-radix-4/2 FFT from a ROM table containing $N/8+1$ coefficients. The twiddle factors in 64-point radix-4 FFT are shown in Table 1 and it can be seen that by reordering the coefficients into six blocks, $B0 \ldots B5$, all the twiddle factors can be retrieved from coefficients in block $B0$ containing nine complex coefficients. Since we need to support several transform sizes up to an N_{max}-point FFT, we store $(N_{max}/8+1)$ complex-valued coefficients into a table, $M = (M_0, M_1, \ldots, M_{N/8}) \mid M_k = W_{N_{max}}^k$. The rest of the twiddle factors can be obtained from the table M as follows:

$$W_{N_{max}}^k = \begin{cases} M_k & , \quad k \leq \frac{N_{max}}{8} \\ -jM_{\frac{N_{max}}{4}-k} & , \frac{N_{max}}{8} < k \leq \frac{N_{max}}{4} \\ -jM^*_{k-\frac{N_{max}}{4}} & , \frac{N_{max}}{4} < k \leq \frac{3N_{max}}{8} \\ -M^*_{\frac{N_{max}}{2}-k} & , \frac{3N_{max}}{8} < k \leq \frac{N_{max}}{2} \\ -M_{k-\frac{N_{max}}{2}} & , \frac{N_{max}}{2} < k \leq \frac{5N_{max}}{8} \\ jM^*_{\frac{3N_{max}}{4}-k} & , \frac{5N_{max}}{8} < k \end{cases} \tag{18}$$

where M_k^* is the complex conjugate of M_k.

Table 1. Twiddle factors in 64-point radix-4 FFT. The decimal value is shown as (real,imaginary).

B0	B1	B2	B3	B4	B5
W_{64}^{0} (1.0,0.0)	W_{64}^{16} (.00,-1.0)				
W_{64}^{1} (1.0,-.10)	W_{64}^{15} (.10,-1.0)			W_{64}^{33} (-1.0,.10)	
W_{64}^{2} (.98,-.20)	W_{64}^{14} (.20,-.98)	W_{64}^{18} (-.20,-.98)	W_{64}^{30} (-.98,-.20)		
W_{64}^{3} (.96,-.29)	W_{64}^{13} (.29,-.96)				W_{64}^{45} (-.29,.96)
W_{64}^{4} (.92,-.38)	W_{64}^{12} (.38,-.92)	W_{64}^{20} (-.38,-.92)	W_{64}^{28} (-.92,-.38)	W_{64}^{36} (-.92,.38)	
W_{64}^{5} (.88,-.47)	W_{64}^{11} (.47,-.88)	W_{64}^{21} (-.47,-.88)	W_{64}^{27} (-.88,-.47)		
W_{64}^{6} (.83,-.56)	W_{64}^{10} (.56,-.83)	W_{64}^{22} (-.56,-.83)	W_{64}^{26} (-.83,-.56)		W_{64}^{42} (-.56,.83)
W_{64}^{7} (.77,-.63)	W_{64}^{9} (.63,-.77)			W_{64}^{39} (-.77,.63)	
W_{64}^{8} (.71,-.71)		W_{64}^{24} (-.71,-.71)			

In order to generate correct twiddle factor W_N^k for the given exponent k defined earlier, we need to create an index to the table M. Such a method can be obtained by noting the fact that the twiddle factors are defined by vectors with equal spaced angles along a unit circle, thus when starting from zero angle the indices to the table M increase by one until $k = N/8$. Then the indices decrease until $k = N/4$ and they start to increase again. This behavior results in six regions as shown in Table 1.

In bit-level, we may generate the index to lookup table by dividing the bit-field k into two parts; the three most significant bits of k are denoted as $q = (k_{n_{max}-1}, k_{n_{max}-2}, k_{n_{max}-3})^T$ and the least significant bits by $r = (k_{n_{max}-4}, \dots, k_1, k_0)^T$. The index to the lookup table is obtained as follows

$$w = \begin{cases} r & \text{, if } q_0 = 0 \\ \sim r + 1 & \text{, otherwise} \end{cases} \tag{19}$$

where $\sim r$ denotes inversion of bits in r. This can be seen in the lower part in Fig. 2. The index w is used to access the lookup table M ("LUT" in Fig. 2) and the obtained complex value M_w needs to be modified according to (18), which shows that the correct twiddle factor can be obtained as follows

$$W_{N_{max}}^{k} = \begin{cases} (-1)^{q_0 \triangledown q_2} \Re(M_w) + j(-1)^{q_0 \triangledown q_1 \triangledown q_2} \Im(M_w) & \text{, if } q_0 \triangledown q_1 = 0 \\ (-1)^{q_0 \triangledown q_2} \Im(M_w) + j(-1)^{q_0 \triangledown q_1 \triangledown q_2} \Re(M_w) & \text{, otherwise} \end{cases} \tag{20}$$

where $\triangledown$ denotes bitwise exclusive-OR operation and $\Re(x)$ and $\Im(x)$ denote real and imaginary part of x, respectively. Figure 2 shows that this modification requires two multiplexors and two real adders with XOR-gates in inputs.

4 Experiments

We have described the proposed twiddle factor unit in VHDL language such that $N_{max} = 2^{14}$, i.e., the unit supports power-of-two FFTs up to 16K, thus lookup table contains 2049 complex-valued coefficients. The inputs to the unit are 17-bit c register and 4-bit

Table 2. Power dissipation and area of twiddle factor unit designs: proposed unit, pipelined (two stages) and non-pipelined, and unit based on ROM table [5]

	pipelined@250MHz		non-pipelined@140MHz		ROM table [5]@250MHz	
	Power [mW]	Area [kgates]	Power [mW]	Area [kgates]	Power [mW]	Area [kgates]
LUT	1.50	12	2.24	15.8	43.00	20.5
Pipeline	0.95	0.3				
Total	3.70	14.3	4.11	18.4	43.00	20.5

f register. The lookup table contains complex-valued coefficients with 16-bit real and imaginary parts, i.e., word width of lookup table is 32 bits.

The design has been synthesized with Synopsys tools onto a 130 nm standard cell technology. Then power estimates have been obtained with Synopsys tools with aid of gate level simulations. The analysis results are listed in Table 2.

The analysis results show that the critical path limits the clock frequency to 140 MHz when no pipelining is exploited. When two pipeline stages are used, the maximum clock frequency is 275 MHz. The lookup table has been designed with hard-wired logic for reducing the power consumption. If the lookup table was implemented as a ROM memory, the power consumption would have been eight times higher, although the area had been half smaller.

For comparison, we have also implemented a unit based on the traditional ROM table approach where $N_{max}/2 = 8192$ coefficients are stored ("ROM table" in Table 2). The method in [9] is not compared since it does not support radix-4 algorithms.

We have also the twiddle factor unit in an ASIP tailored for FFT computations [12] and, in this 32-bit processor containing, e.g., complex multiplier and adder, the twiddle factor unit uses about 23% of the core area (instruction and data memories not included), while the power consumption is only 7% of the total power consumption of the core. However, the unit improved significantly the performance of the FFT software implementation; the unit provides twiddle factor once per instruction cycle without additional address manipulation instructions.

5 Conclusions

In this paper, we have described a twiddle factor unit supporting radix-4 and mixed-radix-4/2 FFT algorithms and several power-of-two FFT sizes. The unit can be used as a special unit in an ASIP architecture or a coefficient generator in application-specific FFT processors. The unit shows significant power savings compared to the popular approach where the twiddle factors are stored into a ROM table.

Acknowledgement

This work has been supported in part by the Academy of Finland under project 205743 and the Finnish Funding Agency for Technology and Innovation under research funding decision 40441/05.

References

1. Wu, C.S., Wu, A.Y.: Modified vector rotational CORDIC(MVR-CORDIC algorithm and its application to fft. In: Proc. IEEE ISCAS, Geneva, Switzerland, vol. 4, pp. 529–532 (2000)
2. Xiu, L., You, Z.: A new frequency synthesis method based on flying-adder architecture. IEEE Trans. Circuits Syst. 50(3), 130–134 (2003)
3. Fliege, N.J., Wintermantel, J.: Complex digital oscillator and FSK modulators. IEEE Trans. Signal Processing 40(2), 333–342 (1992)
4. Chi, J.C., Chen, S.G.: An efficient FFT twiddle factor generator. In: Proc. European Signal Process. Conf., Vienna, Austria, pp. 1533–1536 (2004)
5. Cohen, D.: Simplified control of FFT hardware. IEEE Trans. Acoust., Speech, Signal Processing 24(6), 577–579 (1976)
6. Chang, Y., Parhi, K.K.: Efficient FFT implementation using digit-serial arithmetic. In: Proc. IEEE Workshop Signal Process. Syst., Taipei, Taiwan, pp. 645–653. IEEE Computer Society Press, Los Alamitos (1999)
7. Ma, Y., Wanhammar, L.: A hardware efficient control of memory addressing for high-performance FFT processors. IEEE Trans. Signal Processing 48(3), 917–921 (2000)
8. Wanhammar, L.: DSP Integrated Circuits. Academic Press, San Diego, CA (1999)
9. Hasan, M., Arslan, T.: FFT coefficient memory reduction technique for OFDM applications. In: Proc. IEEE ICASSP, Orlando, FL, vol. 1, pp. 1085–1088 (2002)
10. Chu, E., George, A.: Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms. CRC Press, Boca Raton, FL (2000)
11. Pitkänen, T., Mäkinen, R., Heikkinen, J., Partanen, T., Takala, J.: Low-power, high-performance TTA processor for 1024-point fast Fourier transform. In: Vassiliadis, S., Wong, S., Hämäläinen, T.D. (eds.) SAMOS 2006. LNCS, vol. 4017, pp. 227–236. Springer, Heidelberg (2006)
12. Pitkänen, T., Mäkinen, R., Heikkinen, J., Partanen, T., Takala, J.: Transport triggered architecture processor for mixed-radix FFT. In: Proc. Asilomar Conf. Signals, Systems, and Computers, Pacific Grove, CA (2006)
13. Rabiner, L.R., Gold, B.: Theory and Application of Digital Signal Processing. Prentice Hall, Englewood Cliffs (1975)
14. Granata, J., Conner, M., Tolimieri, R.: Recursive fast algorithms and the role of the tensor product. IEEE Trans. Signal Processing 40(12), 2921–2930 (1992)
15. Akopian, D., Takala, J., Astola, J., Saarinen, J.: Multistage interconnection networks for parallel Viterbi decoders. IEEE Trans. Commun. 51(9), 1536–1545 (2003)
16. Bóo, M., Argüello, F., Bruguera, J., Doallo, R., Zapata, E.: High-performance VLSI architecture for the Viterbi algorithm. IEEE Trans. Commun. 45(2), 168–176 (1997)

[P4] PUBLICATION 4

T. Pitkänen, J. Tanskanen, R. Mäkinen, and J. Takala,”Parallel Memory Architecture for Application-Specific Instruction-Set Processors,” *Journal of Signal Processing Systems*, vol. 57, no. 1, pp. 21–32, October, 2009.

J Sign Process Syst (2009) 57:21–32
DOI 10.1007/s11265-008-0173-y

Parallel Memory Architecture for Application-Specific Instruction-Set Processors

Teemu Pitkänen · Jarno K. Tanskanen · Risto Mäkinen · Jarmo Takala

Received: 14 October 2007 / Revised: 4 January 2008 / Accepted: 4 March 2008 / Published online: 26 April 2008

Abstract Many of the current applications used in battery powered devices are from digital signal processing, telecommunication, and multimedia domains. These applications typically set high requirements for computational performance and often parallelism is the key solution to meet the performance requirements. In order to exploit the parallel processing units, memory should be able to feed the data path with data. This calls for a memory organization supporting parallel memory accesses. In this paper, a conflict resolving parallel data memory system for application-specific instruction-set processors is described. The memory structure is generic and reusable to support various application-specific designs. The proposed memory system does not employ any predefined access format signals for memory addressing. The proposed parallel memory system is attached to an application-specific instruction-set processor core and comparison on area, power, and critical path are shown. The experiments show that significant power savings can be obtained by exploiting the parallel memory system instead of multiport memory.

T. Pitkänen ( ) · J. K. Tanskanen · J. Takala
Tampere University of Technology,
Room TH318, PL553, 33101 Tampere, Finland
e-mail: teemu.pitkanen@tut.fi

J. K. Tanskanen
e-mail: jarno.tanskanen@tut.fi

J. Takala
e-mail: jarmo.takala@tut.fi

R. Mäkinen
Plenware Oy, P.O.B. 13, 33201 Tampere, Finland
e-mail: risto.makinen@plenware.fi

Keywords Parallel memory · Low power · TTA · ASIP · Transport triggered architecture · Application-specific instruction-set processors

1 Introduction

In current embedded systems, the most important applications are from digital signal processing (DSP), telecommunication, and multimedia domains. These applications typically set high requirements for computational performance. Furthermore, these applications are realized on battery powered devices where low power consumption is an critical design issue. In general, algorithms in these domains contain inherent parallelism, which can be exploited in various ways, thus the performance is often obtained by increasing the parallelism in the implementations. Typical algorithms contain fine-grain parallelism thus, in implementations based on application-specific integrated circuits (ASIC), the number of computational resources can be increased, e.g., arithmetic units. In processor based implementations, the same approach can be used, i.e., the number of parallel functional units is increased, which calls for instruction-level parallelism (ILP).

An implementation approach where the flexibility of programmable solution and power-efficiency of an ASIC solution are combined is application-specific instruction-set processor (ASIP). In an ASIP, the architecture is tailored according to the requirements of the given application or application domain, thus the fine-grain parallelism in the algorithm can be exploited in form of ILP. Since the architecture is customized, the power consumption can be expected to be lower than in a processor designed for general-purpose computing.

In order to exploit the parallel functional units in the data path of a processor, the bandwidth of the memory system should match the performance of the data path. Especially in DSP domain, several operands from memory need to be accessed to feed the parallel arithmetic units, which implies that the memory system should either support several accesses per instruction/clock cycle or have multiple ports.

In this paper, a parallel memory system is proposed to be used in ASIP implementations. The memory system is based on several independent single-port memory modules and access logic, which directs several memory access requests to multiple memories and converts the logical addresses to physical addresses for each memory. The power-efficiency of the proposed system is compared to multi-port memories and the results show that significant power savings can be obtained, if the data can be distributed over the multiple memory modules. Although, the proposed design is demonstrated with a single storage scheme, the parallel memory system can be easily modified to support other storage schemes. Furthermore, the data permutation and address computation circuitry might be fitted in the existing pipeline structure of the processor without lowering the clock frequency or increasing the number of pipeline stages.

2 Related Work

The most flexible and simplest approach from the programmer or compiler point of view to increase the memory bandwidth is to use multi-port memories since the same memory space can be accessed simultaneously through several ports. Unfortunately, multi-port memories require larger area, provide slower access time, and higher power consumption than single-port memories. Recently multi-port memories have been considered in field-programmable gate array designs, e.g., in [1, 2], but there the memories are predefined modules with additional logic. Therefore, the cost of memory organization is different than in other gate array technologies where the basic building block is gate.

Another approach to increase the memory bandwidth is to access a single-port memory with a higher frequency than the processing frequency. However, this solution might not scale to when the number of ports is increased and the power consumption typically increases as the access time is shortened. In DSP processors, the memory bandwidth is traditionally increased by using extended Harvard architecture where several data memories are connected to the data path, e.g., in Motorola DSP56000 [3], there were two data memory banks and Hitachi DSPi [4] contained four memory banks. The drawback is that certain memory locations can be accessed only over a dedicated memory port and this is visible to the programmer.

To avoid the cost of multi-port memory structure, several different methods have been used. High-bandwidth multi-port data cache memory systems in multiple-issue processors have been considered in [5–7]. To provide the functionality of an N-port memory, N single-port memory modules with the same data content could be employed. A write operation is always sent to all the memory modules to maintain the data coherence. As a drawback, the memory must be replicated N times and no other accesses can be made during a write operation.

Another approach is to use N single-port memories to emulate the multi-port memory. This is referred to as a parallel memory system. In order to support access to memory location over different ports, additional permutation and address computation circuitry is needed. Unfortunately, there is no single storage scheme, which would allow conflict-free parallel access for all possible access patterns thus designs, which consider conflict resolution multi-port memory system, employ some form of a simple low-order interleaving scheme, e.g., [2, 5, 7, 8]. The parallel memory approach can also be used to create shared memory for multi-processor systems as described, e.g., in [9].

In a parallel memory system, a module assignment function $S(i)$ is a function of the incoming address i from the load/store unit (LSU) and determines the index of the memory module where the data is located. The address for $\mathrm{MM}_{S(i)}$ is determined by the address assignment function $a(i)$. If the parallel memory logic has N LSUs, then N module and address assignment functions, $S(i_k)$ and $a(i_k)$, respectively, are computed simultaneously. Here i_k refers to an address from LSU_k, $0 \leq k < N$. Basically, $S(i)$ determines, how well the memory performs for a given parallel address trace. The most simple and well-known module assignment functions are low-order and high-order interleaving functions. The low-order interleaving, is efficient for parallel access of successive array elements and it is defined by the following equations:

$$\begin{cases} S(i) = i \bmod N \\ a(i) = i/N \end{cases} . \tag{1}$$

The high-order interleaving performs well when several different arrays are accessed in parallel and it is described as follows

$$\begin{cases} S(i) = i/a_{\max} \\ a(i) = i \bmod a_{\max} \end{cases} \tag{2}$$

Table 1 Low-order and high-order interleaving examples.

Low-order						High-order					
i	$S(i)$	$a(i)$	i	$S(i)$	$a(i)$	i	$S(i)$	$a(i)$	i	$S(i)$	$a(i)$
0	0	0	8	0	2	0	0	0	8	2	0
1	1	0	9	1	2	1	0	1	9	2	1
2	2	0	10	2	2	2	0	2	10	2	2
3	3	0	11	3	2	3	0	3	11	2	3
4	0	1	12	0	3	4	1	0	12	3	0
5	1	1	13	1	3	5	1	1	13	3	1
6	2	1	14	2	3	6	1	2	14	3	2
7	3	1	15	3	3	7	1	3	15	3	3

$N = 4$, $a_{\max} = 4$.

where $a_{\max}$ is simply a constant defining the number of memory locations in each memory module. An example of low and high-order interleaving module and address assignment functions are shown in Table 1.

In many cases, the operands for parallel processing can be stored such that conflict-free access to certain patterns is possible, e.g., rows, columns, blocks, forward, and backward-diagonals [10, 11]. These are called the access formats. In general, the module assignment functions used for this purpose are linear [10] or so called XOR-schemes [12, 13]. Multi-skewing scheme [14] provides versatile access formats. Storage schemes supporting stride accesses are presented in [15, 16]. A brief overview of storage schemes is provided in [17].

In previous papers, parallel memories have been considered in different type of architectures and applications: multiprocessor systems performing scientific computations with matrix data, vector machines with stride accesses, and general-purpose computation on multi-issue machines. Popular examples on DSP area have been fast Fourier transform (FFT) computation and video processing. This paper considers parallel memory system in ASIP designs where the existing parallel memory theory can be used to construct specific storage schemes to avoid or significantly reduce the memory conflicts. Furthermore, unlike in many other parallel memory systems, we do not employ any predefined access format signals for the parallel memory addressing. We also present details of the hardware structure for dynamic conflict resolution.

3 Parallel Memory System

As shown in Fig. 1, the parallel memory logic is designed to locate between the load-store units and synchronous single-port memory modules, thus a parallel memory non-optimally emulates a multi-port memory. Unlike in multi-port memories, access conflicts are possible in parallel memories, i.e., one or more single-port memory modules are tried to be accessed more than once during a single cycle parallel memory access. In general, it is not possible to find a generic storage scheme such that conflict-free access is possible for all address traces. In the case of memory conflict, parallel memory hardware recognizes the conflict, locks the processor by sending a lock request to the global control unit of the processor core, performs conflicting accesses sequentially (requiring more cycles), and releases the lock. No modifications to software are required for correct functionality. Because of this locking behavior, the software does not know whether a multi-port or parallel memory system is employed. However, access conflicts increase the cycle count.

A multi-port memory can be replaced by a parallel memory architecture after the application code has been written. In this case, the software has been likely developed assuming an ideal multi-port memory and no attention is paid into the addresses of simultaneous memory accesses. It might be possible not to find a conflict-free parallel memory storage scheme since the address traces can be irregular, not fitting to typical access formats. A better performance in terms of clock

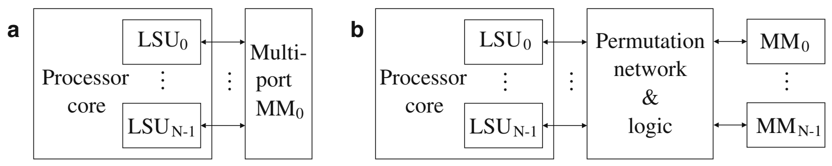

Figure 1 Different data memory configurations: **a** multi-port memory and **b** parallel memory system consisting of single-port memories with permutation network and conflict resolution logic.

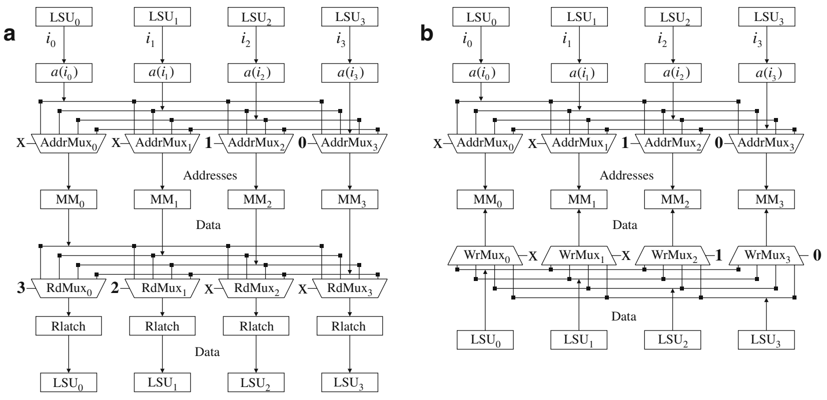

Figure 2 Principal structure of parallel data memory: **a** read (load) and **b** write (store) operation with related address, read data, and write data crossbars. The fixed control signals for the multiplexers refer to the example in Fig. 3.

cycles can be obtained when a conflict-free storage scheme is found for application-specific access formats used in the application code. Especially, the code of the innermost loops, where typically the most of the execution time is spent, should be written such that the addresses of simultaneous memory accesses will not cause memory module access conflicts. This may require manual assembler optimization and regeneration of the hardware for the processor core.

The proposed parallel memory design is generic and re-usable. Application-specific memory functions can be fitted in and generics are employed in the very high speed integrated circuit hardware description language (VHDL) design so that several parallel memory components with different parameters (bus widths, number of ports, and memory functions) could be fitted in the same design. Parallel memory logic provides the needed address computation, interconnection, and conflict resolution logic. The proposed design is a matched memory system, i.e., the number of LSUs and memory modules is the same. In addition, the proposed structure receives multiple addresses simultaneously and no access pattern signaling is used. In traditional designs, the access pattern is defined and the actual addresses to the memory modules are generated separately based on the base address information.

4 Conflict Resolution

Depending on the address i_k and the module assignment function $S(i_k)$, the load or store operation from LSU$_k$ may refer to any memory module. For this reason, permutation network shown in Fig. 1b is needed to route the signals from the LSUs to the memory modules. The read (load) data from the memory modules need also to be permuted when moved to the LSUs. The permutations are often implemented with crossbars and the crossbars related to read and write operations are illustrated in Fig. 2 for $N = 4$. The read data from the memory module need to be saved to registers when several accesses are made to the same memory module.

For each LSU$_k$ and MM$_k$ pair, there is a control unit, which enables the MM$_k$ and drives the crossbar control signals: control for the address (AddrMuxCtrl$_k$), read data (RdMuxCtrl$_k$), and write data (WrMuxCtrl$_k$). This control unit circuitry is shown in Fig. 4. The module assignment functions $S(i_k)$ are solved in parallel. They can be used to control read data crossbar so that $S(i_k)$ drives RdMuxCtrl$_k$ (connected to RdMux$_k$ in Fig. 2) at the correct moment. For various other control purposes, $S(i_k)$ indices are binary coded and used to construct a binary control matrix. This is shown

Figure 3 An example of control matrix. **a** Control matrix. **b** Transposed control matrix. **c** '1' bits served in the first cycle. **d** '1' bits served in the second cycle.

a

MM$_k$	**0**	**1**	**2**	**3**	
LSU$_0$	0	0	0	1	$\leftarrow S(i_0)$
LSU$_1$	0	0	1	0	$\leftarrow S(i_1)$
LSU$_2$	0	0	1	0	$\leftarrow S(i_2)$
LSU$_3$	0	0	0	1	$\leftarrow S(i_3)$

b

LSU$_k$	**0**	**1**	**2**	**3**
MM$_0$	0	0	0	0
MM$_1$	0	0	0	0
MM$_2$	0	1	1	0
MM$_3$	1	0	0	1

c

LSU$_k$	**0**	**1**	**2**	**3**
MM$_0$	0	0	0	0
MM$_1$	0	0	0	0
MM$_2$	0	0	1	0
MM$_3$	0	0	0	1

d

LSU$_k$	**0**	**1**	**2**	**3**
MM$_0$	0	0	0	0
MM$_1$	0	0	0	0
MM$_2$	0	1	0	0
MM$_3$	1	0	0	0

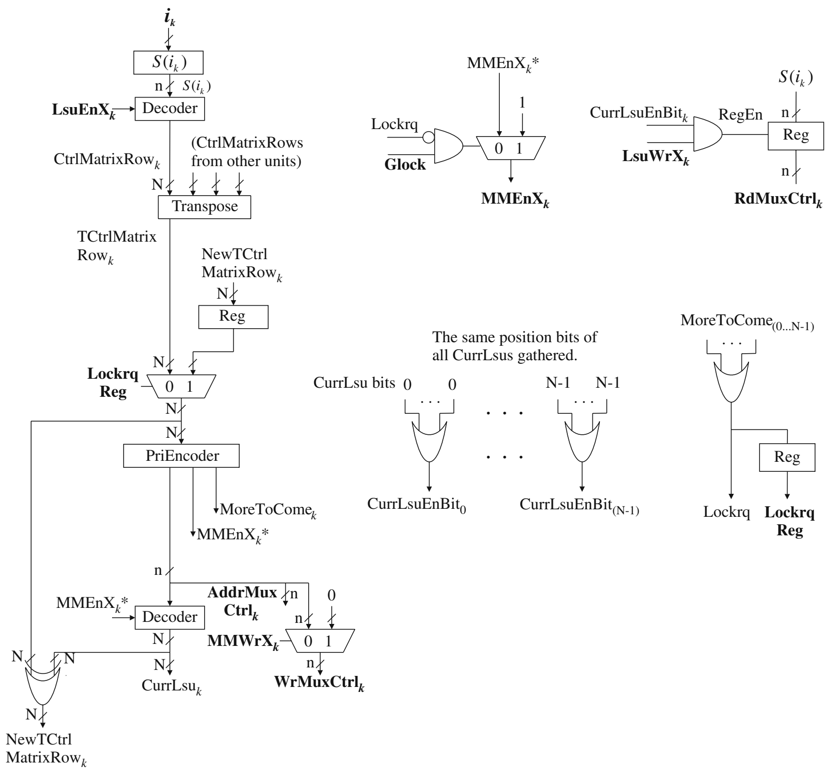

Figure 4 Control unit logic. There is one control unit circuit for each LSU$_k$ and MM$_k$ pair. OR-trees producing CurrLsuEnBit and Lockrq signals combine the data from all the control units. Block input and output signals are written in bold. LsuEnX$_k$, LsuWrX$_k$, and i_k are signals from LSU$_k$. MMEnX$_k$ and MMWrX$_k$ are enable and R/W signals for MM$_k$. (With true multi-port memory, we would simply have LsuEnX$_k$ = MMEnX$_k$ and LsuWrX$_k$ = MMWrX$_k$.) Glock: global lock signal from the processor core. $N = 2^n$: the number of memory ports.

in Fig. 4, where a decoder produces a single control matrix row, CtrlMatrixRow$_k$. An example matrix is shown in Fig. 3a for $N = 4$. It can be seen that LSU$_0$ and LSU$_3$ are accessing MM$_3$, and LSU$_1$ and LSU$_2$ are accessing MM$_2$.

Each control unit needs to know which LSU will access their memory module. This is obtained by transposing the control matrix as illustrated in Fig. 3b. The kth row of the transposed matrix is delivered for the kth control unit. In Fig. 4, the control matrix composed from CtrlMatrixRows from all the control units is transposed and TCtrlMatrixRow$_k$ is obtained for each control unit k. If there are more than one '1' bits on any TCtrlMatrixRow of the transposed matrix, then there are corresponding number of memory conflicts. Parallel control units make always as many parallel accesses as possible. If there is a conflict, a priority encoder (PriEncoder) of the control unit k selects the rightmost bit (LSU) on the TCtrlMatrixRow$_k$ in the first cycle, the next rightmost bit in the second cycle, and so on, until all the '1' bits on the row are served. '1' bits are reduced one by one with the XOR-gates producing NewTCtrlMatrixRow$_k$ in Fig. 4. This loop is enabled by the multiplexer controlled by LockrqReg signal. All the rows of the transposed control matrix are processed in parallel by the control units.

Each priority encoder has MoreToCome signal which tells that there are still '1' bits left. As is shown in Fig. 4, these MoreToCome signals are combined from each control unit to a single bit using an OR-tree. After registering, this bit becomes the lock request signal (LockrqReg) to be sent to the processor. CurrLsu signal of the control unit tells the index of the LSU, which is to be currently served by the memory module. The same position CurrLsu signal bits have been combined by N OR-trees shown in Fig. 4. The resulted CurrLsuEnBits are used to control the state machines and RdMuxCtrl signals.

The transposed control matrices for the first and second memory cycles of the example are shown in Fig. 3c–d. The rows of these matrices are used to control the multiplexers of the address (AddrMuxCtrl), write data (WrMuxCtrl), (write mask), and write signal crossbars. The crossbar multiplexer control signals for the control matrix given in Fig. 3 would be the following (x refers to “don’t care” condition):

When all the accesses are load operations, the multiplexer controls are:

Cycle 1: $\mathrm{AddrMuxCtrl}_{0\ldots3} = \{x, x, 2, 3\}$,
$\mathrm{RdMuxCtrl}_{0\ldots3} = \{x, x, 2, 3\}$.

Cycle 2: $\mathrm{AddrMuxCtrl}_{0\ldots3} = \{x, x, 1, 0\}$,
$\mathrm{RdMuxCtrl}_{0\ldots3} = \{3, 2, x, x\}$.

When all the accesses are store operations, the multiplexer controls are:

Cycle 1: $\mathrm{AddrMuxCtrl}_{0\ldots3} = \mathrm{WrMuxCtrl}_{0\ldots3} = \{x, x, 2, 3\}$.

Cycle 2: $\mathrm{AddrMuxCtrl}_{0\ldots3} = \mathrm{WrMuxCtrl}_{0\ldots3} = \{x, x, 1, 0\}$.

The cases for cycle 2 are shown with a simplified architecture in Fig. 2. A parallel memory access may consist of both load and store operations. The main benefit of using the control matrix is that any kind of module assignment function, $S(i)$, can be included and used to construct the rows of the control matrix. After that, all the memory module and crossbar controls can be obtained automatically.

The lock request signal (LockrqReg) from the parallel memory logic is registered and, therefore, the previous input values have to be saved to registers (i.e., addresses (i_k), memory enables (LsuEnX_k), write enables (LsuWrX_k), write data, and possible write masks), because otherwise they would be overwritten by new values. A simple state machine controls the saving of this data at the correct moment. The saving is not shown in the figures. The previous input values caused memory conflict(s) and the lock request, thus they are needed to resolve the conflict(s) sequentially.

The design contains numerous scalable components. There are three crossbars: address, read data, and write data. When LSUs with sub word support are used, an additional crossbar is needed for the write mask. Each crossbar has N input and output busses with corresponding bus widths. In addition, one smaller crossbar for single-bit memory write signals is needed. There are $(N + 1)$ OR-trees and each OR-tree merges an N-bit input into a single bit. Finally, there are N priority encoders with an N-bit input and $2N$ decoders with a $(\log_2 N)$-bit input.

5 Experiments

In order to demonstrate the power savings, the proposed memory system was integrated to an existing ASIP design. Different type of memory organizations were connected to the core and synthesized on an ASIC technology and various cost estimates were obtained with the aid of simulations. Before going into details of experiments, we shortly describe the reference design used in the experiments.

5.1 Reference ASIP with Dual-Port Memory

The reference ASIP was a processor core tailored for radix-4 FFT computations described in [18]. The processor is based on transport triggered architecture (TTA). In the TTA programming model, the program specifies only the data transports to be performed by the interconnection network and operations occur as “side-effect” of data transports [19]. Operands to a function unit enter through ports and one of the ports is dedicated to act as a trigger. When data is moved to the trigger port, execution of an operation is initiated. A TTA processor consists of a set of function units and general-purpose register files. These structures are connected to an interconnection network, which connects the input and output ports of the resources. The architecture can be modified by adding or removing

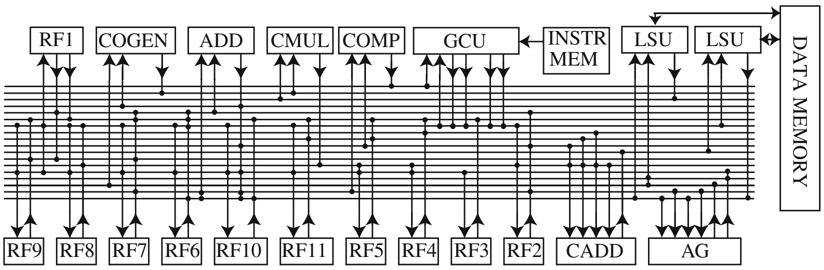

Figure 5 Principal architecture of the reference processor. *ADD*: Real adder. *AG*: Data address generator. *CADD*: Complex adder. *CMUL*: Complex multiplier. *COGEN*: Coefficient generator. *COMP*: Comparator unit. *GCU*: Global control unit. *RFx*: Register files containing a total of 23 general-purpose registers.

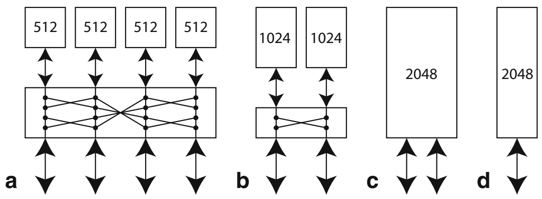

Figure 6 Memory configurations with total size of 2048×32 used in comparison: **a** parallel quad-port memory, **b** parallel dual-port memory, **c** dual-port memory, and **d** single-port memory.

resources and special function units with user-defined functionality can be easily included.

The structural VHDL description of the TTA core was obtained by using the processor generator of the TCE framework [20]. The principal block diagram of the reference core with two LSUs is illustrated in Fig. 5, which also shows that a dual-port memory was used in the reference design. The processor was synthesized and verified on a commercial 130 nm ASIC technology and the critical paths in the core dictated a clock frequency of 250 MHz. This is actually quite close to the practical maximum clock frequency of the used technology.

The original reference design can compute a radix-4 butterfly operation with the throughput of 0.25 butterflies per cycle, which implies that two memory accesses are performed in a clock cycle, thus two LSUs are sufficient. In the reference design, a dual-port memory shown in Fig. 6c was used.

5.2 ASIPs with Parallel Memory System

In order to compare the properties of parallel memory systems, the dual-port memory (2048×32) in the reference ASIP was replaced with a parallel memory logic and two 1024×32 single-port memories as illustrated in Fig. 6b. In this test case, the logic to control the crossbar was designed to implement the storage scheme proposed in [21] where a XOR scheme designed for stride permutation access for 2^r-element arrays distributed over $N = 2^n$ memory modules. In this case, the assignment functions are represented at bit-level, i.e., the module assignment is $S(i) = (m_{(n-1)}, \ldots, m_1, m_0)^T$ and the address assignment $a(i) = (a_{(r-n-1)}, \ldots, a_1, a_0)^T$. If the address from a LSU is represented in bit-level as an array $i = (i_{(r-1)}, \ldots, i_1, i_0)^T$, the assignments are defined with the following equation pair:

$$\begin{cases} a_j = i_{(j+n)},\ j = 0, 1, \ldots, r-n-1 \\ m_j = \bigoplus_{k=0}^{l_{r,n}(j)} i_{((kn+j) \bmod r)},\ j = 0, 1, \ldots, n-1 \end{cases};$$

$$l_{r,n}(j) = \lfloor (r + n - \gcd(n, r \bmod n) - j - 1)/n \rfloor \qquad (3)$$

where $gcd(.)$ is the greatest common denominator, $\lfloor \cdot \rfloor$ denotes floor operation, and $\bigoplus$ denotes bit-wise XOR operation.

In this case, the number of ports, N, was two, thus the scheme reduces to a parity bit computation of an address i_k from the LSU$_k$. The computed bit $S(i_k) = m_0$ defines which one of the memory modules should be accessed. An address for the module $S(i_k)$ is simply defined by the most significant bits of the address i_k as $a(i_k) = i_k/N$.

As an example, an excerpt of the data address trace is shown in Table 2. The first 10 clock cycles represent the prolog code and the actual iteration kernel starts at cycle $cc = 10$. To save the memory storage, the computation is performed in in-place fashion, which can be seen form the memory trace: operand load from a certain address is followed by result store into the same address 10 cycles later. During the iteration kernel the full memory bandwidth is exploited. In each clock cycle, both memory modules are accessed as indicated by the column S, which in this case equals to the parity bit of the corresponding address i_k. The conflict-free operation was verified with RTL-level simulation.

Table 2 Example of data address trace for the 1024-point radix-4 FFT implementation with two load/store units.

cc	LSU$_0$			LSU$_1$		
	rw	i_0	S	rw	i_1	S
0	r	1024	1	r	1028	0
1						
2	r	1032	0	r	1036	1
3						
4	r	1025	0	r	1029	1
5						
6	r	1033	1	r	1037	0
7						
8	r	1026	0	r	1030	1
9						
10	r	1034	1	r	1038	0
11	w	1024	1	w	1028	0
12	r	1027	1	r	1031	0
13	w	1032	0	w	1036	1
14	r	1035	0	r	1039	1
15	w	1025	0	w	1029	1
16	r	1040	0	r	1044	1
17	w	1033	1	w	1037	0
18	r	1048	1	r	1052	0
19	w	1026	0	w	1030	1
20	r	1041	1	r	1045	0
21	w	1034	1	w	1038	0

cc Clock cycle, *rw* read/write, i_0 and i_1 addresses, S memory module index.

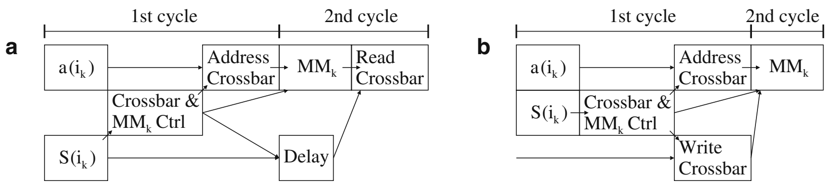

Figure 7 Parallel memory pipeline timing: **a** read and **b** write operation.

Another test case was created by extending the reference core by adding more functional units such that the butterfly computations were performed with the throughput of 0.5 butterflies per clock cycle. Therefore, higher memory bandwidth was needed and four LSUs were included. This resulted in almost twofold speedup; a 1024-point FFT was computed in 2648 cycles compared to the two load/store case, 5208 cycles. In this case, the parallel memory system contained four 512×32 single-port memory modules and parallel memory logic as depicted in Fig. 6a. The module and address assignment functions are still from Eq. 3 but now the module assignment reduces to computation of two bit parity and address assignment produces one bit narrower results.

5.3 Memory Access Pipeline

The LSUs in the reference ASIP were pipelined containing three stages and, therefore, the proposed parallel memory system was easy to interface to the core. The LSU signals to and from the memory modules are registered requiring two clock cycles, and the memory access itself requires one clock cycle. The input signals for the synchronous memory modules (i.e., address, enable, write enable, write data (and write mask)) are read in the rising clock edge. Thus, because there is not much logic between the registered LSUs and input/output ports of the memory modules, the timing budget was met. The principal timing diagram of the parallel memory between the input and output registers of LSU for read and write operations are illustrated in Fig. 7. The control logic, address crossbar, and write crossbar are located in the 1st cycle slot between the LSUs and memory modules. The read crossbar is connected between the outputs of the memory modules and LSUs. During a read clock cycle, the data appears to the memory module data output bus after a related memory access delay. Then the data travels through the read crossbar to the data input registers of the LSUs.

5.4 Comparison

All the test cases were synthesized to a 130 nm, 1.5 V CMOS standard cell ASIC technology with Synopsys Design Compiler. This was followed by a gate level simulation at 250 MHz. Synopsys Power Compiler was used for the power analysis. The obtained results are listed in Table 3. The cost figures listed in "Memory" in parallel memory cases include also the control logic and crossbar.

When comparing the two first cases (dual-port memory case and 2-port parallel memory case, which contain the same reference core with two LSUs where the same code is executed), it can be seen that the dual-port memory requires larger area (factor of 3.5) than the parallel memory system. The power consumption of dual-port memory case is almost double compared to parallel memory, thus the power-efficiency of the parallel memory system is significantly better than in dual-port memory.

Table 3 Characteristics of 1024-point radix-4 FFT implementations on TTA.

		Area [kgates]	Power [mW]	Energy/FFT [μJ]	Cycles/FFT
Dual-port memory, Fig. 6c	Memory	102.4	27.2	0.57	
reference core with	Core	33.1	43.6	0.91	5208
2 LSUs	Total	135.5	70.8	1.48	
Parallel memory, Fig. 6b	Memory	29.7	15.7	0.33	
reference core with	Core	33.1	43.6	0.91	5208
2 LSUs	Total	62.8	59.3	1.24	
Parallel memory, Fig. 6a	Memory	37.0	29.0	0.31	
extended core with	Core	66.3	88.0	0.93	2648
4 LSUs	Total	103.3	117.0	1.24	

All designs synthesized and analyzed at 250 MHz clock frequency.

Table 4 Characteristics of different 2048 × 32 memory implementations based on 130 nm ASIC technology with 250 MHz clock frequency.

Structure	# ports		Organization	Area [kgates]	Power [mW]	Energy/FFT [μJ]	Cycles/ FFT	Delay [ns]
Parallel		Memory	4*(512 × 32)	32.0	23.5	0.25		
Fig. 6a	4	Logic & crossbar		5.0	5.5	0.06	2648	2.10
		Total	2048 × 32	37.0	29	0.31		
Parallel		Memory	2*(1024 × 32)	27.8	13.9	0.29		
Fig. 6b	2	Logic & crossbar		1.9	1.8	0.04	5208	2.17
		Total	2048 × 32	29.7	15.7	0.33		
Dual-port Fig. 6c	2	Total	2048 × 32	102.4	27.2	0.57	5208	2.48
Single-Port Fig. 6d	1	Total	2048 × 32	25.9	10.8	0.45	10328	1.83

We can also analyze the scalability of the system by comparing the last two test cases in Table 3 (in extended core, the computational resources are doubled compared to reference core). The results show that the area and power consumption of the parallel memory scale quite nicely. The power-efficiency seems to even improve but that is due to the fact that we have doubled the resources but the schedule, e.g., execution time, is not half of the original, thus the relative time per memory access is longer than in the reference case. It should be noted, that this kind of scaling was possible in our reference application, FFT, since it is regular. Some other applications may not allow this kind of scalability.

In order to compare the power consumption of parallel-memory and multi-port memory, we analyzed the area and power consumption of 2048 × 32 memory realized as parallel memory, dual-port, and single-port memory as shown in Fig. 6. The results are listed in Table 4, which presents the area, power, and energy dissipation acquired through gate level simulation of 1024-point FFT at 250 MHz. The only multi-port memory configuration is the dual-port memory since the used technology library did not contain any memories with larger number of ports.

The results show that multi-port memories are expensive both in terms of area and power. Adding a second port results in almost four times larger area and three times higher power consumption. When adding a second port with parallel memory approach, the area and power are increased by factors of 1.2 and 1.5, respectively. A 4-port parallel memory increases area and power by factors of 1.4 and 2.8, respectively, compared to a single-port case. The energy figures in Table 4 show that a 2-port parallel memory is even more energy-efficient than single-port memory. This is due to the fact that execution time is half of the execution time in single-port case.

The largest dual-port memory in the technology library was 4096 words, thus we repeated the previous analysis for 4096 × 32 memory organizations and the results are listed in Table 5. It can be noted that the parallel memory requires only 26% of the area of corresponding dual-port memory and uses 52% of the power. In addition, when the memory size is doubled (2 to 4 kW), the power consumption of dual-port memory increases by a factor of 1.61, while the corresponding dual-port parallel memory increases by a factor of 1.43. Correspondingly doubling the memory capacity of

Table 5 Characteristics of different 4096 × 32 memory implementations based on 130 nm ASIC technology with 250 MHz clock frequency.

Structure	# ports		Organization	Area [kgates]	Power [mW]	Energy/FFT [μJ]	Cycles/ FFT	Delay [ns]
Parallel		Memory	4*(1024 × 32)	55.7	27.2	0.29		
	4	Logic & crossbar		5.0	5.5	0.06	2648	2.23
		Total	4096 × 32	60.7	32.7	0.35		
Parallel		Memory	2*(2048 × 32)	51.8	21.6	0.45		
	2	Logic & crossbar		1.9	1.8	0.04	5208	2.86
		Total	4096 × 32	53.9	22.4	0.49		
Dual-port	2	Total	4096 × 32	204.3	43.4	0.90	5208	2.71

a 4-port parallel memory increases power consumption by factor of 1.13. Finally, the energy figures in Tables 4 and 5 show that the dual-port memory consumes 1.7-1.8 times more energy than the 2-port parallel memory.

It should be noted that the memory structures shown here are created for the same pipelining structure as used in the original core designed for dual-port memory, thus the structures are not always optimal, e.g., in Table 5, the 4 kW 2-port parallel memory is slower than the corresponding 4-port memory. However, all the structures fulfill the clock constraint, 4.0 ns.

In parallel memory systems, the additional logic and crossbar introduce additional delay, which may limit the access time and, therefore, the maximum clock frequency. We have listed the critical paths of the memory designs in Tables 4 and 5. The timing aspects of parallel memories have already been considered, e.g., in [17, 22], and the results indicate that the cost of additional logic and crossbar is not dominant when then number of ports is not large, i.e., less than 16. It can be assumed that, in ASIP implementations, this is the case. In addition, if the critical path in parallel memory becomes the limiting factor, the load and store operations need to be pipelined more efficiently.

The previous discussion indicates that the area and power costs of multi-port memory are significantly larger than in parallel memory case. The costs also seem to increase faster as the number of ports is increased. This is one of the reasons that multi-port memories ASIC technology libraries are often limited only to dual-port. Finally, the ASIC technology used in our experiments can be classified as a low-power IC technology and the static power consumption is not as significant as in future technologies. Therefore, the previous power consumption comparisons between multi-port and parallel memory will be different in the future. However, the static power consumption is related to the area, thus it can be expected that, in the future, parallel memory system will be even more energy-efficient compared to multi-port memories.

6 Conclusion

In this paper, a conflict resolving parallel data memory for ASIP designs was proposed. The proposed memory system does not employ any predefined access format signals for the parallel memory addressing. Such a memory system can be exploited in application-specific designs where the address trace can be highly regular and predictable, thus there can be a good change to find a well performing storage scheme. The existing parallel memory theory can be used to construct application-specific storage schemes. The proposed parallel memory system contains conflict-detection logic, thus it is general-purpose. However, if the storage scheme does not suit to the instruction schedule, i.e., the parallel accesses in the address trace are done into the same memory module, performance is degraded as the memory accesses need to be serialized.

Although the experiments used in this paper were performed with a TTA processor, the parallel memory system is applicable for any ASIP supporting multiple LSUs and memory wait states.

Our experiments on low power ASIC technology show that significant power savings can be obtained by using parallel memory system instead of multi-port memory. Furthermore, the area comparison implies that the savings will be even higher in the future when static power consumption will become dominant.

Acknowledgements This work has been supported in part by the Academy of Finland under project 205743 and the Finnish Funding Agency for Technology and Innovation under research funding decision 40441/05.

References

1. Sawyer, N., & Defossez, M. (2002). Quad-port memories in Virtex devices. Xilinx application note, XAPP228 (v1.0) (September 24).
2. Ang, S. S., Constantinides, G., Cheung, P., & Luk, W. (2006). A flexible multi-port caching scheme for reconfigurable platforms. In K. Bertels, et al. (Eds.), *ARC 2006, LNCS* (Vol. 3985, pp. 205–216). New York: Springer.
3. Kloker, K. L. (1986). The Motorola DSP56000 digital signal processor. *IEEE Micro, 6*(6), 29–48 (December).
4. Kaneko, K., Nakagawa, T., Kiuchi, A., Hagiwara, Y., Ueda, H., Matsushima, H., et al. (1987). A 50ns DSP with parallel processing architecture. In *IEEE int. solid-state circuits conference, digest of technical papers.* (pp. 158–159) (February).
5. Sohi, G. S., & Franklin, M. (1991). High-bandwidth data memory systems for superscalar processors. In *Proc. 4th int. conf. architectural support for programming languages and operating systems,* Santa Clara, CA, U.S.A. (pp. 53–62) (April 8–11).
6. Juan, T., Navarro, J. J., & Temam, O. (1997). Data caches for superscalar processors. In *Proc. 11th int. conf. supercomputing,* Vienna, Austria, (pp. 60–67) (July 7–11).
7. Rivers, J. A., Tyson, G. S., Davidson, E. S., & Austin, T. M. (1997). On high-bandwidth data cache design for multi-issue processors. In *Proc. 30th ann. ACM/IEEE int. symp. microarchitecture, research triangle park,* NC, U.S.A. (pp. 46–56) (December 1–3).
8. Zhu, Z., Johguchi, K., Mattausch, H. J., Koide, T., Hirakawa, T., & Hironaka, T. (2003). A novel hierarchical multi-port cache. In *Proc. 29th European solid-state circuits conf.,* Estoril, Portugal (pp. 405–408) (September 16–18).
9. Patel, K., Macii, E., & Poncino, M. (2004). Energy-performance tradeoffs for the shared memory in multi-processor systems-on-chip. In *Proc. IEEE int. symp. circuits*

and systems, Vancouver, British Columbia, Canada (Vol. 2, pp. 361–364) (May 23–26).
10. Budnik, P., & Kuck, D. J. (1971). The organization and use of parallel memories. *IEEE transactions on computers, C-20*(12), 1566–1569 (December).
11. Kim, K., & Prasanna, V. K. (1993). Latin squares for parallel array access. *IEEE Transactions on Parallel and Distributed Systems, 4*(4), 361–370 (April).
12. Frailong, J. M., Jalby, W., & Lenfant, J. (1985). XOR-schemes: A flexible data organization in parallel memories. In *Proc. int. conf. parallel process.* (pp. 276–283), (August 20–23).
13. Liu, Z., & Li, X. (1995). XOR storage schemes for frequently used data patterns. *Journal of Parallel and Distributed Computing, 25*(2), 162–173 (March).
14. Deb, A. (1996). Multiskewing – a novel technique for optimal parallel memory access. *IEEE Transactions on Parallel and Distributed Systems* 7(6), 595–604 (June).
15. Rau, B. R. (1991). Pseudo-randomly interleaved memory. In *Proc. 18th ann. int. symp. computer architecture,* Toronto, Ontario, Canada. (pp. 74–83) (May 27–30).
16. Seznec, A., & Lenfant, J. (1995). Odd memory systems: a new approach. *Journal of Parallel and Distributed Computing, 26*(2), 248–256 (April).
17. Tanskanen, J. K., Creutzburg, R., & Niittylahti, J. T. (2005). On design of parallel memory access schemes for video coding. *Journal of VLSI Signal Processing, 40*(2), 215–237 (June).
18. Pitkänen, T., Mäkinen, R., Heikkinen, J., Partanen, T., & Takala, J. (2006). Low-power, high-performance TTA processor for 1024-point fast fourier transform. In S. Vassiliadis, et al. (Eds.), *Embedded computer systems: Architectures, modeling, and simulation: Proc. 6th int. workshop SAMOS 2006. LNCS,* (Vol. 4017, pp. 227–236). New York: Springer.
19. Corporaal, H. (1997). *Microprocessor architectures: From VLIW to TTA*. Chichester: Wiley.
20. Jääskeläinen, P., Guzma, V., Cilio, A., Takala, J. (2007). Codesign toolset for application-specific instruction-set processors. In *Proc. SPIE - multimedia on mobile devices.* 05070X–1–10.
21. Takala, J. H., Järvinen, T. S., & Sorokin, H. T. (2003). Conflict-free parallel memory access scheme for FFT processors. In *Proc. IEEE int. symp. circuit syst.* Bangkok, Thailand (Vol. 4., pp. 524–527) (May 25–28).
22. Aho, E., Vanne, J., & Hämäläinen, T. D. (2006). Parallel memory implementation for arbitrary stride accesses. In *Proc. int. conf. embedded comput. syst. architectures modeling simulation,* (pp. 1–6) Samos, Greece (July).

Teemu Pitkänen received his M.Sc.(EE) degree from Tampere University of Technology, Tampere, Finland (TUT) in 2005. From 2002 to 2005, he worked as a Research Assistant and currently he works towards Dr.Tech as researcher in the Institute of Digital and Computer Systems at TUT. His research interest include paraller architectures, minimization of energy dissipation and design methodologies for DSP systems.

Jarno K. Tanskanen was born in Joensuu, Finland in 1975. He studied analog and digital electronics in the Department of Electrical Engineering, and computer architecture in the Department of Information Technology at Tampere University of Technology, where he received his M.Sc. and Dr.Tech degrees in 1999 and 2004. From 1998 to 2007, he was a researcher at TUT. He is currently working as an ASIC/field-programmable gate array designer at SWECO Industry Oy in Oulu.

Risto Mäkinen received his M.Sc. (EE) degree from Tampere University of Technology, Tampere, Finland (TUT) in 2006. From 2003 to 2006, he worked as a Research Assistant in the Institute of Digital and Computer Systems at TUT. Currently, he is a Software Designer in the Telecom Unit of Plenware, Tampere, Finland.

Jarmo Takala received his M.Sc. (hons) (EE) degree and Dr.Tech. (IT) degree from Tampere University of Technology, Tampere, Finland (TUT) in 1987 and 1999, respectively. From 1992 to 1996, he was a Research Scientist at VTT-Automation, Tampere, Finland. Between 1995 and 1996, he was a Senior Research Engineer at Nokia Research Center, Tampere, Finland. From 1996 to 1999, he was a Researcher at TUT. Currently, he is Professor in Computer Engineering at TUT and Head of Department of Computer Systems of TUT. His research interests include circuit techniques, parallel architectures, and design methodologies for DSP systems.

[P5] PUBLICATION 5

T. Pitkänen and J. Takala, ”Low-Power Application-Specific Processor for FFT Computations,” *Journal of Signal Processing Systems*, vol. 63, no. 1, pp.165–176, April, 2011.

J Sign Process Syst (2011) 63:165–176
DOI 10.1007/s11265-010-0528-z

Low-Power Application-Specific Processor for FFT Computations

Teemu Oskari Pitkänen · Jarmo Takala

Received: 9 July 2009 / Revised: 2 September 2010 / Accepted: 3 September 2010 / Published online: 17 September 2010

Abstract In this paper, a processor architecture tailored for radix-4 and mixed-radix FFT computations is described. The processor has native support for power-of-two transform sizes. Several optimizations have been used to improve the energy-efficiency of the processor and experiments show that a programmable solution can possess energy-efficiency comparable to fixed-function ASICs.

Keywords Discrete Fourier transform · Application-specific integrated circuit · Digital signal processors · Parallel architecture

1 Introduction

Quite often implementations of Fast Fourier transform (FFT) are based on Cooley-Tukey radix-2 FFT algorithms due to the regularity and principal simplicity of the computations. However, FFT algorithms with higher radices can be more efficient as many of the twiddle factors become trivial multiplications. Especially, radix-4 algorithms have been popular since the 4-point discrete Fourier transform (DFT) can be computed without multiplications.

In order to gain efficiency with higher radices, the basic DFTs need to be computed with efficient algorithms but advantage is gained only in cases where the length of the input sequence is a power of the radix. A solution supporting power-of-two transform lengths with lower complexity than radix-2 is mixed-radix FFT where the DFT is decomposed to a structure containing DFTs of several sizes.

Even lower arithmetic complexity can be achieved with split-radix FFT algorithms, where the number of arithmetic operations is close to the theoretical minimum. Typically in split-radix algorithms even and odd indexed outputs are computed differently, thus the regularity of the Cooley-Tukey FFT in control flow is not achieved.

FFT implementations can be divided to two categories: memory based, e.g., [25, 31] or pipelines, e.g., [11, 26, 30]. Pipelined FFTs are usually used in short FFTs since as the size of the FFT increases, more intermediate storage is needed and memory is more efficient storage for large amount of data than registers and delay elements used in pipelines. The basic principle of the both methods are shown in Fig. 1: (a) shows memory based computations with one processing element (PE) and a single port memory and, in (b), a two ported memory with one PE is illustrated, with one radix-2 PE up to four simultaneous access to the memory can be used efficiently. In (c), a parallel architecture is shown with multiple PEs and memory ports and, in (d), basic principle of pipelined method is depicted where delay elements are used in the feedback loop. The memory based designs are considered to be smaller and require less power. On the other hand, pipelined designs offer higher throughput. Nowadays memories are also used in pipelined designs to reduce power dissipation, especially when large transform is needed [30]. In memory based designs, the throughput can be improved by

T. O. Pitkänen ( ) · J. Takala
Department of Computer Systems, Tampere University of Technology, P.O. Box 553, 33101 Tampere, Finland
e-mail: teemu.pitkanen@tut.fi

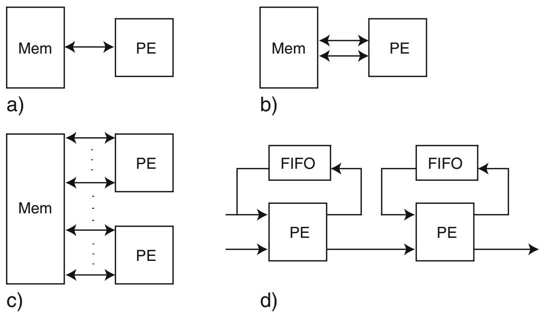

Figure 1 Basic principles of FFT computation methods: **a** single PE with single port memory, **b** single PE with dual ported memory, **c** parallel column architecture, and **d** pipelined architecture.

adding more processing elements and more memory accesses, i.e., parallel architectures, but this of course increases area and power consumption.

Traditionally FFT has been implemented as a fixed-function VLSI circuit since it provides better energy-efficiency and performance than software implementations. Such implementations have recently been reported, e.g., in [31, 33], based on radix-2 FFT and examples of radix-4 FFTs can be found, e.g., from [7, 9] and mixed-radix algorithms are reported in [12, 16]. Recently, software implementations have became preferable due to the flexibility but the energy-efficiency of programmable architectures has been poor compared to dedicated hardware structures. However, the energy-efficiency can be improved by customizing the architecture towards to the application domain.

In this paper, we describe a processor architecture tailored for FFT computations. Several optimizations have been used to improve the energy-efficiency of the processor. This paper is based on the principles reported in our earlier papers [18, 19, 21] and shows that a programmable solution can possess energy-efficiency comparable to fixed-function ASIC. The processor is tailored for radix-4 and mixed-radix FFT algorithms and supports several transform lengths.

2 Related Work

General-purpose processors (GPP) are used in FFT calculation, but the main limitation of the GPPs in the mobile environment is the energy consumption. The main reason for this is high cycle count, which requires high clock frequency to achieve required throughput. The Intel Pentium-4 [5] represents a standard general-purpose microprocessor. StrongArm SA-1100 [10] can be considered as general-purpose processor for mobile devices as it employs custom circuits, clock gating, and reduced supply voltage. Better solution in energy wise are general-purpose DSP processors. The FFT computation takes significantly less cycles compared to GPPs, thus the clock frequency can be orders of magnitude less for the same or better throughput. Examples of such general-purpose DSP processors are TI TMS320C6416 [28], which is a VLIW machine, and Imagine [24] designed for media applications. Both the processors utilize pseudo-custom data path tiling. In addition, pass-gate multiplexer circuits are exploited in C6416.

Pure hardware solution can achieve highest usage of the parallelism available in the algorithm, but there are different solutions how much parallelism is exploited, since it requires more functional units and memory ports. The energy consumption can be reduced by using low supply voltage [1] or even using sub threshold supply voltage [29]. With ultra low voltages the throughput of the system is heavily reduced. Dynamic power consumption can be reduced by exploiting the inherent properties of algorithms. E.g., one of the four twiddle factors in radix-4 butterfly or half of the twiddle factors in radix-2 butterflies is trivial 1 thus multiplication is not needed. In [30], multiplications by $\sqrt{2}/2$ and $-j$ are avoided. In [11], a method to reduce the use of general multipliers is presented by balancing the binary tree of the FFT calculation.

In memory based designs, the memory bandwidth needs to match the throughput, which often calls for additional memory ports. However, the parallel memory ports are typically expensive in terms of area and energy, in particular the write ports. Instead of using multi-port memories, we can use several single-port memories. In [25], two single-port memories are exploited and four single-port memory are used in [32] but the FFT computations were limited to radix-2 algorithms, the supported FFT sizes are 256 and 1,024. Also a parallel memory scheme can be used where memory emulates the multi-port memory with some access scheme [17], access scheme can be conflict-free. Power consumption of the memories are important issue also in pipeline based designs, where in [30] authors present a method of using shift registers instead of the memories in the feedback loop. This has a major impact on the power dissipation. Twiddle factor storage has major effect to the power dissipation and it can be reduced by using coefficient generator, which exploits the symmetry of sine and cosine functions. In [11, 30], the twiddle factors are computed on-online, which is area-efficient solution but unfortunately computations increase the dynamic power consumption.

Application-specific processors (ASP) are increasing in popularity as they offer a good midpoint solution between DPS and custom hardware. One example is SPOCS [2], which contains special instructions for FFT computations reducing the implementation complexity and cycle count. SPOCS processor consist: program control unit, address unit with two address generation units, specialized FFT addres generation unit, and address register, optimized data processing unit with two MACs, one ALU, barrel shifter and register file, program memory and two dual ported data memories.

In this paper, we propose a processor customized for FFT computations. The processor is programmable, thus it is ASP. We exploit several of the optimizations used also in ASIC implementations, which typically cannot be exploited in software implementations or introduce too much additional overhead, e.g., we use complex data type with native support, address computations often requiring multiplication are performed at bit-level where mainly rotations are needed, nested loops are avoided with the aid of specialized address generation units, multiplications with trivial constants 1 is avoided, and a parallel memory with conflict-free parallel access scheme is used to emulate two-ported memory. We have created a twiddle factor generator which exploits the redundancy in the factors and we use ROM table approach, which consumes significantly less power compared online computations. Finally, the processor is based on transport triggered paradigm where the programming model reminds static data flow. This architecture template provides some additional advantage in energy-efficiency. We exploit efficiently radix-4 computations and the resulting instruction schedule is tight and exploits fully the available memory bandwidth, which is seldom the case in software implementations.

3 FFT Algorithms

In this paper, we have used the in-place decimation-in-time radix-4 FFT algorithm with in-order-input, permuted output as given, e.g., in [23] and corresponding mixed-radix algorithm. The radix-4 algorithm can be formulated as follows:

$$F_{2^{2n}} = R_{4^n}\left[\prod_{s=n-1}^{0} (I_{4^s} \otimes F_4 \otimes I_{4^{n-s-1}})\,(T_{4^{s+1},4^n} \otimes I_{4^{n-s-1}})\right]; \tag{1}$$

$$F_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix}; \tag{2}$$

$$R_{4^n} = \prod_{k=n-2}^{0} I_{4^k} \otimes P_{4^{(n-k)},4} \tag{3}$$

where $\otimes$ denotes tensor product, I_N is identity matrix of order N, and R_N is a permutation matrix based on stride-by-S permutation matrix of order N, $P_{N,S}$, defined as [6]

$$[P_{N,S}]_{mn} = \begin{cases} 1, \text{ iff } n = (mS \bmod N) + \lfloor mS/N \rfloor \\ 0, \text{ otherwise} \end{cases}, \quad m, n = 0, 1, \ldots, N-1\,. \tag{4}$$

The matrix $T_{k,N}$ contains twiddle factors $W_N^m = e^{j2\pi m/N}$ as follows

$$\begin{aligned} T_{k,N} = \operatorname{diag}\big(&\omega(0)^0, \omega(0)^1, \omega(0)^2, \omega(0)^3, \\ &\omega(1)^0, \omega(1)^1, \omega(1)^2, \omega(1)^3, \ldots, \\ &\omega(k/4-1)^2, \omega(k/4-1)^3\big); \end{aligned} \tag{5}$$

$$\omega = R_{N/4}\left(W_N^0, W_N^1, \ldots, W_N^{(N/4-1)}\right)^T \tag{6}$$

An example of signal flow graph of this algorithm is depicted in Fig. 2a. As the radix-4 algorithm supports only power-of-four transform sizes, we use mixed-radix approach to support also power-of-two FFTs. Again we use a mixed-radix algorithm with in-order input, permuted output, consisting of radix-4 and radix-2 computations as follows

$$\begin{aligned} F_{2^{2n+1}} &= S_{2^{(2n+1)}}\,(I_{4^n} \otimes F_2)\,U_{2^{(2n+1)}} \\ &\times \left[\prod_{s=n-1}^{0} (I_{4^s} \otimes F_4 \otimes I_{2^{2n-2s-1}})\left(T_{4^{s+1},4^n} \otimes I_{2^{2n-2s-1}}\right)\right]; \end{aligned} \tag{7}$$

$$F_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}; \tag{8}$$

$$S_{2^{2n+1}} = (I_2 \otimes R_{4^n})\,P_{2^{2n+1},2} \tag{9}$$

where twiddle factors for the last radix-2 processing column are defined as

$$U_N = S_N^T\left(I_{N/2} \oplus \operatorname{diag}\left(W_N^0, W_N^1, \ldots, W_N^{N/2-1}\right)\right) \tag{10}$$

where $\oplus$ denotes matrix direct sum. An example of signal flow graph of the mixed-radix algorithm based on radix-4 and radix-2 computations is shown in Fig. 2b.

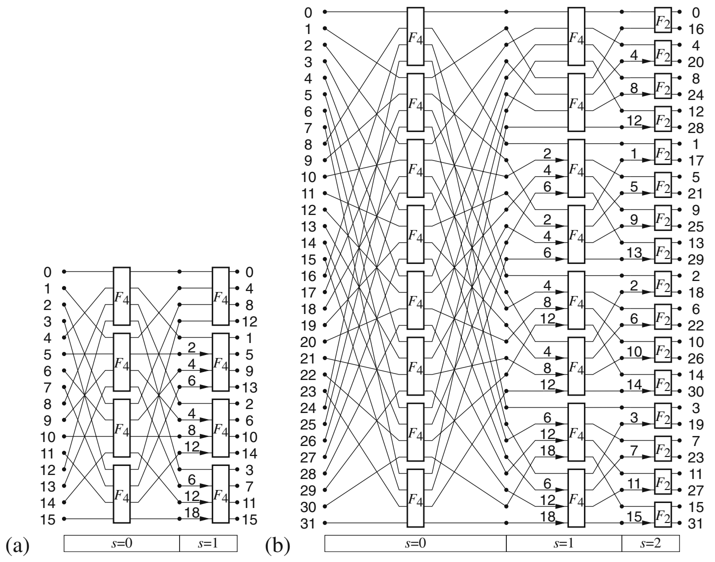

Figure 2 Signal flow graph of **a** 16-point radix-4 FFT and **b** 32-point mixed-radix FFT. A constant k in the signal flow graph represents a twiddle factor W_{32}^{k}.

4 Transport Triggered Architecture

Transport triggered architecture (TTA) [4], is a class of statically programmed instruction-level parallelism architectures reminding very long instruction word (VLIW) architectures. In TTA programming model, the program specifies only the operand transfers and operations are performed as a side-effect of the data transfer. Operands to a function unit are fed through input ports and one of the ports acts as a trigger; whenever data is written to the trigger port, function unit starts executing the operation.

A TTA processor consists of a set of function units and register files containing general-purpose registers. The input and output ports of these resources are connected together with an interconnection network as illustrated in Fig 3. The data transports are carried out by buses in the interconnection network. The processors in Fig 3 contains six buses implying that six parallel data moves can be performed in each cycle. The operands can be moved directly from the output port of an function unit to an input port of another function unit without the need to store the result to register file. The interconnection network can be fully connected, i.e., all the ports of all the function units are connected to the all the buses. Quite often the network is optimized in such a way that only a sub set of all possible connections are used as shown in Fig. 3; the ports of the resources are connected only to some of the buses (black dots). Instructions specify each individual transport on each bus, i.e., instruction contains a bit field for each bus, which specifies which output port and input port are connected to the bus during one instruction cycle. As the instruction contains several fields for defining parallel moves, the instruction format reminds very long instruction word (VLIW) machines but here the instructions control the interconnection network and specify the data moves rather than controlling directly the function units as in VLIW machines. As the instructions control only data transports, the architecture has only one instruction, move. The drawback of the TTA is really wide instruction word and poor code density, both can be improved with code compression [8].

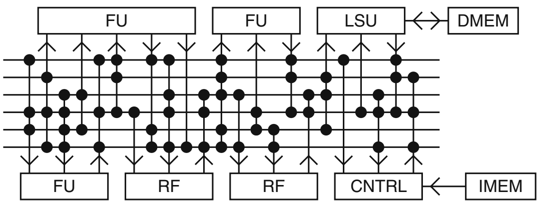

Figure 3 TTA processor organization. *FU* Function unit. *RF* Register file. *LSU* Load-store unit. *CNTRL* Control unit. *DMEM* Data memory. *IMEM* Instruction memory. *Dots* represent connections between buses and ports of function units.

The function units follow the transport triggering paradigm; each unit trigger the operation execution when an operand is moved to a trigger operand register shown in the principal block diagram of a function unit in Fig. 4. A single trigger register is the minimum requirement and additional operands can be fed through one or more operand registers; unit in Fig. 4 is a two-operand unit having a trigger port "tdata" with trigger register and operand port "o1data" with operand register. As the operation starts execution when data is written to the trigger register, all the other operands need to be transferred to operand registers in the same or earlier cycle. Operands in input registers can also be shared between consecutive operations in a function unit, i.e., there is no need to transfer the operands again to the registers if the same operands are used in the next operation. This reduces the traffic in the interconnection and the need for temporary storage in register files or data memory.

Function units can also support several operations and the actual operation to be executed is indicated with an opcode attached to the trigger move instruction. Function units can be pipelined and different operations may even have different pipeline depths. The pipeline is synchronized with a control register chain initiated from trigger load signal "tload" as depicted in Fig. 4. The control signal from register chain can also be used for clock gating as shown in Fig. 4, which effectively reduces power consumption of pipeline stages during idle cycles. Function units can produce one or more results and the output ports can be registered, which allows the result to be moved to the destination on a later instruction cycle. However, new results will overwrite the previous, thus results need to be moved before new results arrive. Finally, TTA processors support also wait states, which means the the pipelines of the function units needs to be locked during such cycles. Each function unit has a global lock signal "glock" as illustrated in Fig. 4.

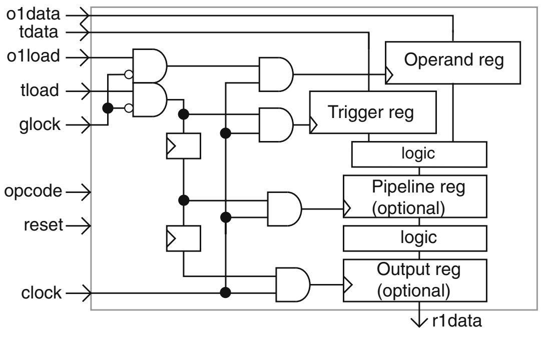

Figure 4 Principal structure of a two-operand, one-result function unit.

The processor architecture can be tailored for a specific application by selecting suitable resources to the processor and optimizing the interconnection network according to the traffic generated by the application. Significant improvements can be obtained by using special function units which perform application-specific operations. Such units are the key to energy-efficient implementations; customization of the function units according to the specific characteristics of the application is effective means to reduce power consumption without reducing the performance. Such units also reduce the instruction overhead, thus they reduce the power consumption due to the instruction fetch.

As TTA processors are statically programmed, there are no run-time predictions, which try to improve the performance in general-purpose computing but unfortunately consume power. When customizing a processor for a specific application, we can exploit all the a priori information and efficient instruction scheduling can be created off-line. In TTA case, the target of scheduling is to allocate operand moves over the buses such that operands are in the operand registers when operand is moved to trigger port. In addition, the schedule has to guarantee that results are moved from result ports before new results arrive. Development tools for customizing TTA processors and retargetable compiler for the customized TTA processors can be found from [27].

5 Building Blocks for FFT Computations

In order to customize the TTA processor for FFT computations, we need to first identify the specific features of the application, which can be exploited when defining the functional units and organization of the processor. The first issue is the fact that FFT exploits complex-valued arithmetic, thus an efficient implementation should support complex data type. Here we simply split the native 32-bit word of the processor template into two parts representing real and imaginary part of complex number. The 16-bit parts are represented as signed fractional numbers. We also need arithmetic units supporting the previous complex data type, thus a complex-valued multiplier and complex-valued adder are included. The complex multiplication unit provides product of two operands with one exception. As +1 is one of the twiddle factors but it cannot be represented with the normalized fractional representation, we have allocated a special twiddle factor bit pattern to represent this value. When the multiplier detects the value, it passed the other operand to the output, i.e., performs multiplication by +1.

The complex add unit is tailored for FFT butterfly computations. The unit has five input ports as illustrated in Fig. 5. The unit computes any of the four-operand (inputs o1–o4 in the figure) additions defined by 4-point DFT, F_4, in Eq. 2. The unit can supports also complex-valued two-operand add/subtract, which is needed in radix-2 butterfly used in our case when realizing mixed-radix algorithm. The operation code is used as the trigger port in the complex add unit, thus once the four complex-valued operands are moved to the input registers, four results can be triggered by simply transferring an opcode to trigger port without the need retransmit all the operands.

In order to save memory storage, in-place computations are used, i.e., results from butterfly computations are stored to memory locations where the operands were read. This implies that for an N-point FFT only N-memory locations are used. The FFT computations contain also special access patterns. This can be seen in Fig. 2: the operands to butterfly computations are not accessed in linear order from the memory array. By considering the index computations in bit-level, it can be found that the operand index can be obtained from the linear address with the aid of simple rotations. In N-point FFT with radix-4 algorithm, we have an n-bit ($n = \log_2 N$) linear index $a = (a_{n-1}, \ldots, a_1, a_0)$. The operand index in the first processing column is obtained by rotating the the linear index two bits to the right; $idx_0 = (a_1, a_0, a_{n-1}, \ldots, a_2)$. In the second column, the rotation is performed for $(n-2)$ least significant bits; $idx_1 = (a_{n-1}, a_{n-2}, a_1, a_0, a_{n-3}, \ldots, a_2)$. In general, in the processing column s (the first column is indexed as zero as indicated in Fig. 2), the $(n-2s)$ least significant bits are rotated two bits to the right. The operation is illustrated in Fig. 6a and b. In 64-point

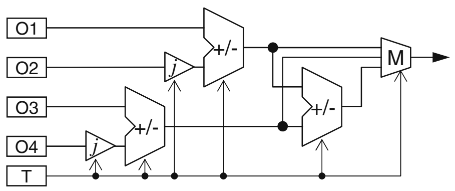

Figure 5 Block diagram of complex adder unit.

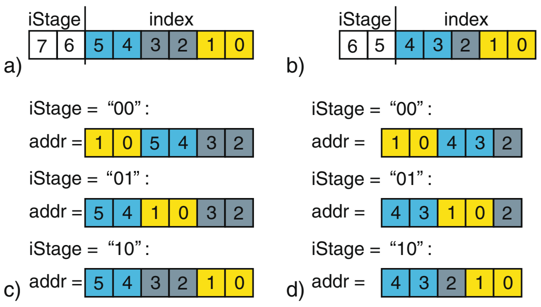

Figure 6 Example of operand address generation. Radix-4 algorithm $N = 64$: **a** 8-bit iteration counter and **b** format of operand index at different processing columns and mixed-radix algorithm $N = 32$: **c** 7-bit iteration counter and **d** operand index format.

transform, there are three processing radix-4 processing columns, thus we have a 2-bit stage counter, "iStage". A 6-bit linear index is seen in Fig. 6a (bits 5–0). In the first column, the 6-bit field is rotated two bits to the right as depicted in Fig. 6b. In the second column, the 4-bit field is rotated, while in the last column (iStage = 2) a 2-bit field is rotated two bits to the right, i.e., original linear index is used.

A mixed-radix algorithm is used when n is odd, i.e., N is not a power-of-four. In such a case, the index is obtained in similar fashion as illustrated in Fig. 6c and d and the only difference is that the field to be rotated has an odd number of bits. In the last column, a 1-bit field is rotated two bits to the right, i.e., again the linear index is used. Based on these principles, we designed a special unit, which rotates the linear index obtained from iteration counter and adds the operand index to the base address of the memory array. The unit requires the transform length and processing column index as additional parameters. The processing column index can be obtained from the iteration counter, i.e., the most significant bits of iteration counter as illustrated in Fig. 6, thus the operand address generation unit contains three input ports as illustrated in Fig. 7. The base address of the memory array and transform length are initialized once, thus the iteration counter is the

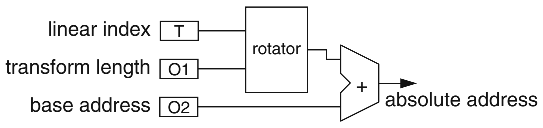

Figure 7 Operand address generation unit.

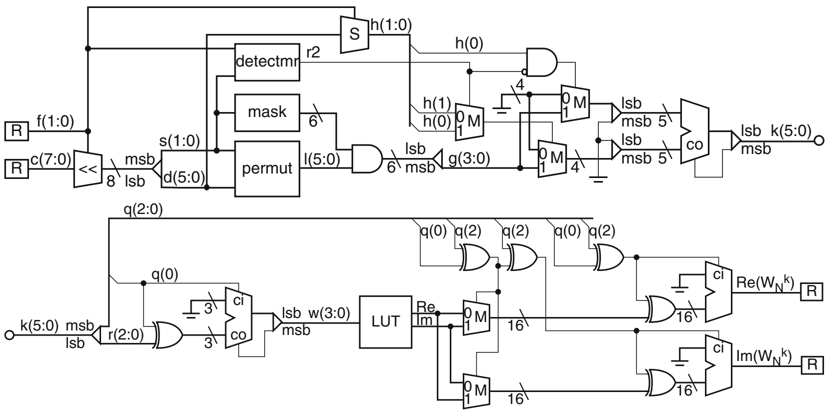

Figure 8 Block diagram of twiddle factor generator supporting transform sizes of 16, 32, and 64. *LUT* Lookup table (nine complex-valued words). *R* register. *M* multiplexor. *co* carry out. *ci* carry in. << left shifter.

only operand transferred to the unit during the kernel execution.

Another memory related issue in FFT computations is the twiddle factors. Here we have designed a special unit, which computes the twiddle factors based on complex-valued coefficients stored in a lookup table. Several methods have been proposed to minimize the lookup tables for generating twiddle factors but mainly for radix-2 algorithms. We have developed a method for minimizing the coefficient tables for radix-4 FFT such that also mixed-radix algorithms are supported. For N-point FFT, we need to store $N/8 + 1$ complex-valued coefficients in a table and four real-valued adders and some control logic are needed to produce the twiddle factors as illustrated in Fig. 8. When the lookup table is designed for an N-point FFT, the units can generate twiddle factors for all the smaller power-of-two transform sizes. More detailed description of the twiddle factor unit can be found from [20].

High performance FFT implementation requires also high memory bandwidth implying need for multiple memory ports. We have avoided the multi-port memories due to their high power consumption and used parallel independent single-port memories instead. Parallel memory organization requires that the operands to be accessed in parallel are distributed to different memory modules, which requires suitable conflict-free access scheme to be used in the parallel memory control logic. As there is no general-purpose conflict-free parallel scheme, this approach is not used in processors. However, in this work, we are tailoring the processor for FFT computations, thus we have a priori information about the access patterns and here we use the simple parity scheme proposed in [3]. The parallel memory organization provides energy-efficiency compared to multi-port memories as we discussed earlier in [22].

6 Processor Organization

The proposed processor is based on TTA template and the special function units described in the previous section were included in the processor as shown in the organization in Fig. 9. The instruction unit (iu) fetches and decodes the instructions from the instruction memory and generates control signals. The immediate unit is used for extracting immediate data from instructions. The interconnection network consists of one Boolean-valued bus mainly for transporting results from comparison unit to be used as parameter for conditional execution. The interconnection contains 17 buses for transporting 32-bit words. The network is optimized

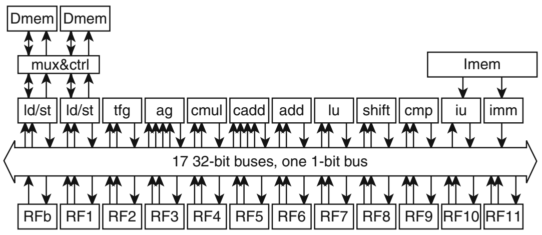

Figure 9 Block diagram of FFT processor. *Dmem* data memory. *Imem* instruction memory. *ld/st* load-store unit. *tfg* twiddle factor unit. *ag* address generator. *cmul* complex multiplier. *cadd* complex adder. *add* adder. *lu* logical unit. *shift* shifter. *cmp* compare unit. *iu* Instruction unit. *imm* immediate unit. *RF* register file. *RFb* Boolean register file.

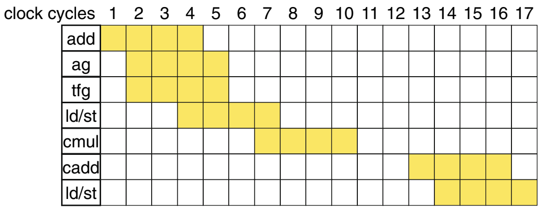

Figure 10 Reservation table of computing a single radix-4 butterfly.

(not fully connected) and several of the buses reduced to point-to-point connections.

The schedule of the FFT computation is constructed with the aid of software pipelining. The reservation table for a single radix-4 butterfly computation is illustrated in Fig. 10 where the register files used for temporary storage are left out for clarity. The resources are listed in vertical axes and the first unit "add" is used to increment the iteration counter. The updated counter value is moved to address generator, which computes the memory address of the operand, and twiddle factor generator, which determines the corresponding factor. The memory address is transferred to load-store unit, which fetches the operands from memory. The memory address is also stored to register file as it is needed later to store the final result. The twiddle factor and operand are transferred to complex multiplier and the product is stored to register file. These operations are performed four times thus all the operands and twiddle factors for a single butterfly are available. Once the four products are available, they are transferred to complex adder and four different additions in radix-4 butterfly (or two add/subtract operations from radix-2 computations in case of the last butterfly column in mixed-radix algorithm) are performed in consecutive instruction cycles. Each result from complex add unit is directly transferred to the second load-store unit. At the same time, the memory address is read from register file, thus the units can store the result to memory. The reservation table shows that the execution of one butterfly takes 18 cycles.

When pipelining several butterfly computations as depicted in the reservation table in Fig. 11, we see a perfect overlap once the pipeline has been filled at cycle #14. All the resources are fully utilized and two memory accesses over two load-store units occur in every cycle indicating that the memory bandwidth matches the performance of the arithmetic resources. During the kernel computations, all the arithmetic units and load/store units have utilization of 100%, thus no energy is spent on idle operations. Based on this reservation table, we can select a kernel function to be iterated in our FFT implementation. The iteration kernel is shown in Fig. 11 consisting of the cycles #16–#32. the cycles #1–#15 represent prologue and cycles #33–#47 are epilogue finalizing the iterations. The selected kernel of 16 cycles consists of one control point of exiting the kernel, which is determined by comparing the iteration counter.

The pseudo code of the schedule is presented in Fig. 12. In the initializations, the end condition for iterations is computed and the used algorithm is determined: pure radix-4 or mixed-radix. Mixed-radix computations require that during the last butterfly column, radix-2 butterflies are computed instead of radix-4. While typically such an alternative computation requires branching, here the kernel is a basic block without branching as we exploit conditional execution. This implies that the software pipelining can be fully exploited without any need to flush the operation pipeline. It should also be noted that the code contains only a single loop as illustrated by the code in Fig. 13. In general, software implementations of FFT contain three nested loops, which indicates overhead of three loops. As we exploit only a single iteration counter, the control overhead is significantly reduced. This can be done as there are special units for managing memory related index computations: both the operand/result access and twiddle factor access.

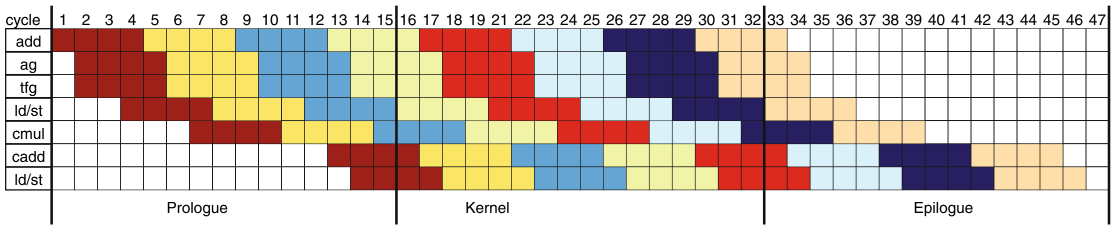

Figure 11 Reservation table for FFT computation.

```
FFT(N,*baseAddress) {
 if (mod4(N) != 0)
   radix2 = 1;
 else
   radix2 = 0;
 for(i = 0; i < N*⌈log4(N)⌉; i +=4) {
    if (i == (⌈log4(N)⌉-1)*N)
     lastStage = 1;
   else
     lastStage = 0;
   for(j = 0; j < 4; j++) {
     twiddle = tfg(N,i+j,radix2,lastStage);
     address[j] = ag(N,*baseAaddress,i+j);
     A[j] = twiddle * (*address[j]);
   }
   if (lastStage == 1 and radix2 == 1){
     res[0] = A[0] + A[1];
     res[1] = A[0] - A[1];
     res[2] = A[2] + A[3];
     res[3] = A[2] - A[3];
   } else {
     res[0] = A[0] + A[1] + A[2] + A[3];
     res[1] = A[0] - im*A[1] - A[2] + im*A[3];
     res[2] = A[0] - A[1] + A[2] - A[3];
     res[3] = A[0] + im*A[1] - A[2] - im*A[3];
   }     for(j = 0; j < 4; j++) {
     *address[j] = res[j];
   }
 }
}
```

Figure 12 Pseudocode for FFT computation.

The efficiency of the implementation can be considered by noting that the theoretical lower bound for a 1,024-point radix-4 FFT when using a dual-port memory is 5,121 memory cycles, i.e., $1{,}024 \log_4(1{,}024) + 1$ parallel reads and writes to dual-port memory. In the proposed processor with two single-port memory modules as illustrated in Fig. 9, the 1,024-point FFT requires 5,160 cycles. This indicates that the looping overheads are minimal.

As the TTA processor reminds VLIW, it suffers the same drawback as VLIW: long instruction word implies high power consumption in instruction fetch. In order to reduce the power consumption of program memory and improve the code density, dictionary-based program compression was applied to the program memory [8]. All the unique instructions of the program code were stored into a dictionary and replaced with indices pointing to the dictionary. This resulted in decrease in the width of the program memory from 167 bits down to 7 bits. The actual dictionary was implemented using hardwired standard cells.

```
main() {
  initialization(); /* 7 to 41 instructions */
  prologue(); /* 15 instr. */
  for(idx=0; idx < (N⌈log4N⌉)/16-1; idx++) {
    kernel(); /* 16 instr. */
  }
  epilogue; /* 15 instr. */
}
```

Figure 13 Pseudocode illustrating the structure and control flow of the program code.

7 Experiments

We have designed a customized TTA processor such that the twiddle factor generator unit supports power-of-two FFTs up to 16K, i.e., the lookup table in the twiddle factor unit contains 2,049 complex-valued coefficients. The native word width of the processor is 32 bits, which allows complex values to be represented with 16-bit real and 16-bit imaginary parts and, therefore, the word width of the lookup table in twiddle factor unit is 32 bits. In addition, the unit has five pipeline stages and the lookup table was implemented as hardwired logic. Clock gating was applied to reduce the power consumption of the function units during idle cycles. This provides power savings on units with low utilization.

The design has been described with VHDL language and synthesized with Synopsys tools onto a 130 nm standard cell technology. Estimates for power consumption are obtained with Synopsys tools with the aid of gate level simulations. The characteristics of the results are listed in Table 1. The twiddle factor unit uses about 23% of the core area and 7% of the

Table 1 Characteristics of the proposed processor synthesized on 130 nm ASIC technology.

Supported FFT sizes	64–16,384
Cycle count	207–114,722
Execution time	828 ns–459µs @ 250 MHz
Power consumption	60–73 mW @ 1.5 V, 250 MHz
Max. clock freq.	255 MHz
Area (kgates)	
Core	37.5
Imem	2
Dmem	240
Total	279.5
1,024-point FFT	
Cycle count	5,160
Power consumption	60.4 mW @ 1.5 V, 250 MHz
	29.8 mW @ 1.1 V, 140 MHz
8,192-point FFT	
Cycle count	57,396
Power consumption	68.7 mW @ 1.5 V, 250 MHz

power consumption, thus twiddle factor unit improves the energy-efficiency of FFT computations. The most significant power savings compared to our previous results from [19] are due to data memories as here we have exploited two parallel single-port memories instead of dual-port memories. The power consumption of the data memories was halved, which is significant since the memories represent half of the total power consumption.

The energy-efficiency of FFT implementations is often compared by measuring how many 1,024-point FFTs can be computed with unit of energy, thus we selected some examples from the literature representing FFTs with different implementation technologies. The comparison results are listed in Table 2. The Intel Pentium-4 [5] represents a standard general-purpose microprocessor. StrongArm SA-1100 [10] can be considered as general-purpose processor for mobile devices. Representatives of general-purpose DSP processors are TI TMS320C6416 and Imagine [24] designed for media applications. FFT implementations on C6416 are reported in [28]. It should be noted that cycle count of 6002 is obtained with eight memory ports while the proposed processor uses only two. The SPOCS [2] is an ASP with instruction extensions tailored for FFT computations which supports FFT from 64- to 8,196-points, but the implemented system would only support FFT to 2,048-points.

Spiffee processor [1] is tailored for FFT computations and energy-efficiency is obtained by using low supply voltages. In [13], an FPGA solution with dedicated embedded FFT logic is reported. In [33], a custom scalable IP core is reported, which employs single memory architecture with clock gating supporting FFT sizes from 16- to 1,024-points, while in [14] a custom variable-length FFT-processor employing radix-2/4/8 single-path delay algorithm is described, supporting FFT lenghts from 512- to 2,048-points. Highly optimized VLSI implementation of FFT using sub threshold circuit techniques is described in [29]. Three staged Radix-8 architecture with memory divided to eight banks capable of 8,192-point FFT is described in [15].

Table 2 Energy-efficiency comparison of various FFT implementations measured as the number of 1024-point FFTs performed with energy of 1 mJ.

Design	Tech. (nm)	Method	V_{CC} (V)	t_{clk} (MHz)	t_{FFT} (µs)	FFT/mJ
1,024-point FFT						
Proposed	130	ASP	1.5	250	21	809
	130	ASP	1.1	140	37	910
[13]	130	FPGA	1.3	100	13	149
	130	FPGA	1.3	275	5	241
[28]	130	DSP	1.2	720	8	100
	130	DSP	1.2	300	22	250
[5]	130	CPU	1.2	3,000	24	1
[24]	150	DSP	1.5	232	160	16
[29]	180	ASIC	0.9	6	430	1,428
	180	ASIC	0.35	0.01	250,000	6,452
[33]	180	FPGA	–	20	282	43
[14]	350	ASIC	3.3	45	23	93
	350	ASIC	2.3	18	57	133
[10]	350	CPU	2.0	74	426	60
[2]	180	ASP	1.8	280	37	31
[1]	600	ASIC	3.3	173	105	39
	600	ASIC	1.1	16	330	319
[25]	65	ASIC	–	1,400	4	–[a]
[32]	180	ASIC	1.8	139	41	–[a]
[11]	180	ASIC	1.8	20	51	480[b]
[26]	45	ASIC	0.9	654	2	2,731
[30]	180	ASIC	1.8	50	21	105
8,192-point FFT						
Proposed	130	ASP	1.5	250	230	63
[15]	180	ASIC	1.8	20	717	55[b]
[31]	180	ASIC	–	22	908	35
[12]	250	ASIC	–	12	1,198	4
[11]	180	ASIC	1.8	20	410	60[b]
[30]	180	ASIC	1.8	50	164	7

V_{CC} supply voltage. t_{clk} clock frequency. t_{FFT} FFT exection time.

[a] Power consumption is not available.

[b] Energy does not include memories.

In [25], switch case fabric is used, which uses two single-port RAM and one ROM memories for each radix-2 processing element. Unfortunately, no power figures are available although quite fast processing time is achieved. Four single-port RAM memories are used in [32] combined with one radix-2 processing element. A scalable pipelined architecture is presented in [26] where two or four processing elements can be used with perfect shuffle permutation and data reordering. High energy-efficiency is achieved by using modern technology and low supply voltage. In [11], a single delay feedback (SDF) pipelined FFT using balanced binary tree is described, which supports different FFT sizes. Only the power dissipation of the core is presented excluding the memories, the system supports FFT sizes from 1,024- to 8,192-points. Another SDF pipelined FFT with SRAMs as delay elements is presented in [30], with support for FFT lengths from 512- to 8,192-points. The use of memories decrease the area and has major impact on the power dissipation compared to implementations using shift registers. This is beneficial when the FFT size is large since the number of elements in delay element increases with the FFT size.

The comparison in Table 2 shows the energy-efficiency of the proposed processor. The efficiency is in the level of fixed-function ASIC implementations although the proposed implementation is programmable.

8 Conclusions

In this paper, a low-power application-specific processor tailored for FFT computation was proposed. The processor is tailored for FFT and several methods for reducing the power consumption of the processor were utilized: clock gating, special function units, and code compression. The processor was synthesized on a 130 nm ASIC technology and power analysis showed that the proposed processor has both high energy-efficiency and high performance. As the proposed processor is highly optimized for FFT, the programmability is limited. However, this is always the case in application-specific implementations where flexibility is traded against efficiency. The programmability of the processor can be easily improved by introducing additional function units and loosening the code compression. In addition, the performance of the processor can be improved by adding computational resources.

Acknowledgement This work has been supported in part by the Academy of Finland under funding decision 205743.

References

1. Baas, B. M. (1999). A low-power, high-performance, 1024-point FFT processor. *IEEE Journal of Solid State Circuits, 43*(3), 380–387.
2. Baek, J. H., Kim, S. D., & Sunwoo, M. H. (2008). SPOCS: Application specific signal processor for OFDM communication systems. *Journal of Signal Processing Systems, 53*(3), 383–397.
3. Cohen, D. (1976). Simplified control of FFT hardware. *IEEE Transactions on Acoustics, Speech, and Signal Processing, 24*(6), 577–579.
4. Corporaal, H. (1997). *Microprocessor architectures: From VLIW to TTA*. Chichester: Wiley.
5. Deleganes, M., Douglas, J., Kommandur, B., & Patyra, M. (2002). Designing a 3 GHz, 130 nm, Intel Pentium4 processor. In *Digest technical papers symp. VLSI circuits* (pp. 230–233). Honolulu, HI.
6. Granata, J., Conner, M., & Tolimieri, R. (1992). Recursive fast algorithms and the role of the tensor product. *IEEE Transactions on Signal Processing,40*(12), 2921–2930.
7. Han, W., Erdogan, A. T., Arslan, T., & Hasan, M. (2008). High-performance low-power FFT cores. *ETRI Journal, 30* (3), 451–460.
8. Heikkinen, J., & Takala, J. (2007). Effects of program compression. *Journal of Systems Architecture,53*(10), 679–688.
9. Hung, C. H., Chen, S. G., & Chen, K. L. (2004). Design of an efficient variable-length FFT processor. In *Proc. IEEE ISCAS* (Vol. 2, pp. 833–836). Vancouver, Canada.
10. Intel: StrongARM SA-110 microprocessor for portable applications brief datasheet (1999).
11. Lee, H. Y., & Park, I. C.: Balanced binary-tree decomposition for area-efficient pipelined FFT processing. *IEEE Transactions on Circuits and Systems, 54*(4), 889–900.
12. Li, X., Lai, Z., & Cui, J. (2007). A low-power and small area FFT processor for OFDM demodulator. *IEEE Transactions on Consumer Electronics, 53*(2), 274–277.
13. Lim, S. Y., & Crosland, A. (2004). Implementing FFT in an FPGA co-processor. In *Proc. int. embedded solutions event*(pp. 230–233). Santa Clara, CA.
14. Lin, Y. T., Tsai, P. Y., & Chiueh, T. D. (2005). Low-power variable-length fast Fourier transform processor. *IEE Proceedings on Computer and Digital Techniques, 152*(4), 499–506.
15. Lin, Y. W., Liu, H. Y., & Lee, C. Y. (2004). Dynamic scaling FFT processor for DVB-T applications. *IEEE Journal of Solid-State Circuits, 39*(11), 2005–2013.
16. Liu, G., & Feng, Q. (2007). ASIC design of low-power reconfigurable FFT processor. In *Int. conf. ASIC* (pp. 44–47). Guilin, China.
17. Patel, K., Macii, E., & Poncino, M. (2004). Energy-performance tradeoffs for the shared memory in multi-processor systems-on-chip. In *Proc. IEEE ISCAS* (Vol. 2, pp. 361–364). Vancouver, BC, Canada.
18. Pitkänen, T., Mäkinen, R., Heikkinen, J., Partanen, T., & Takala, J. (2006). Low-power, high-performance TTA processor for 1,024-point fast Fourier transform. In S. Vassiliadis, S. Wong, & T. D. Hämäläinen (Eds.), *Embedded computer systems: Architectures, modeling, and simulation: Proc. 6th int. workshop SAMOS 2006. LNCS* (Vol. 4017, pp. 227–236). Berlin: Springer.
19. Pitkänen, T., Mäkinen, R., Heikkinen, J., Partanen, T., & Takala, J. (2006). Transport triggered architecture processor for mixed-radix FFT. In *Conf. record asilomar conf. signals syst. comput.* (pp. 84–88). Pacific Grove, CA.

20. Pitkänen, T., Partanen, T., & Takala, J. (2007). Low-power twiddle factor unit for FFT computation. In S. Vassiliadis, M. Bereković, & T. D. Hämäläinen (Eds.), *Embedded computer systems: Architectures, modeling, and simulation: Proc. int. workshop SAMOS 2007, LNCS*. (Vol. 4599, pp. 273–282). Berlin: Springer.
21. Pitkänen, T., & Takala, J. (2009). Low-power application-specific processor for FFT computations. In *Proc. IEEE ICASSP* (pp. 593–596). Taipei, Taiwan.
22. Pitkänen, T., Tanskanen, J. K., Mäkinen, R., & Takala, J. (2009). Parallel memory architecture for application-specific instruction-set processors. *Journal of Signal Processing Systems, 57*(1), 21–32.
23. Rabiner, L. R., & Gold, B. (1975). *Theory and application of digital signal processing*. Englewood Cliffs: Prentice Hall.
24. Rixner, S., Dally, W. J., Kapasi, U. J., Khailany, B., Lopez-Lagunas, A., Mattson, P. R., et al. (1998). A bandwidth-efficient architecture for media processing. In *Proc. ann. ACM/IEEE int. symp. microarchitecture* (pp. 3–13). Dallas, TX.
25. Saleh, H., Mohd, B. J., Aziz, A., & Swartzlander Jr., E. E. (2007). Contention-free switch-based implementation of 1024-point radix-2 Fourier transform engine. In *Proc. IEEE int. conf. comput. design* (pp. 7–12). Lake Tahoe, CA, USA.
26. Suleiman, A., Saleh, H., Hussein, A., & Akopian, D. (2008). A family of scalable FFT architectures and an implementation of 1024-point radix-2 FFT for real-time communications. In *Proc. IEEE int. conf. comput. design* (pp. 321–327). Lake Tahoe, CA, USA.
27. Tampere University of Technology (2008). TTA-based codesign environment. http://tce.cs.tut.fi/.
28. Texas Instruments, Inc., Dallas, TX: TMS320C64x DSP Library programmer's reference (2003).
29. Wang, A., & Chandrakasan, A. (2005). A 180-mV subthreshold FFT processor using a minimum energy design methodology. *IEEE Journal of Solid-State Circuits, 40*(1), 310–319.
30. Wang, S. S., & Li, C. S. (2008). An area-efficient design of variable-length fast Fourier transform processor. *Journal of Signal Processing Systems, 51*(3), 245–256.
31. Wey, C. L., Lin, S. Y., Tang, W. C., & Shiue, M. T. (2007). High-speed, low cost parallel memory-based FFT processors for OFDM applications. In *IEEE int. conf. electronics circ. syst.* (pp. 783–787). Marrakech, Marocco.
32. Yang, Y. X., Li, J. F., Liu, H. N., & Wey, C. L. (2007). Design of cost-efficient memory-based FFT processors using single-port memories. In *IEEE int. SOC conf.* (pp. 321–327). Hsin Chu, Taiwan.
33. Zhao, Y., Erdogan, A. T., & Arslan, T. (2005). A low-power and domain-specific reconfigurable FFT fabric for system-on-chip applications. In *Proc. IEEE par. distributed process. symp. reconf. logic*. Denver, CO.

Teemu Pitkänen received his M.Sc. (EE) degree from Tampere University of Technology, Tampere, Finland (TUT) in 2005. From 2002 to 2005, he worked as a Research Assistant and currently he works towards Dr.Tech. as researcher in the Institute of Digital and Computer Systems at TUT. His research interest include paraller architectures, minimization of energy dissipation and design methodologies for digital signal processing systems.

Jarmo Takala received his M.Sc. (hons) (EE) degree and Dr. Tech. (IT) degree from Tampere University of Technology, Tampere, Finland (TUT) in 1987 and 1999, respectively. From 1992 to 1996, he was a Research Scientist at VTT-Automation, Tampere, Finland. Between 1995 and 1996, he was a Senior Research Engineer at Nokia Research Center, Tampere, Finland. From 1996 to 1999, he was a Researcher at TUT. Currently, he is Professor in Computer Engineering at TUT and Head of Department of Computer Systems of TUT. His research interests include circuit techniques, parallel architectures, and design methodologies for digital signal processing systems.

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FI-33101 Tampere, Finland

ISBN 978-952-15-3350-1
ISSN 1459-2045