Helsinki University of Technology, Signal Processing Laboratory
Teknillinen korkeakoulu, Signaalinkäsittelytekniikan laboratorio
Espoo 2004 Report 50

APPLICATIONS OF REPROGRAMMABILITY IN ALGORITHM ACCELERATION

Matti Tommiska

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Department of Electrical and Communications Engineering for public examination and debate in Auditorium S3 at Helsinki University of Technology (Espoo, Finland) on the 18th of March, 2005, at 12 o´clock noon.

Helsinki University of Technology
Department of Electrical and Communications Engineering
Signal Processing Laboratory

Teknillinen korkeakoulu
Sähkö- ja tietoliikennetekniikan osasto
Signaalinkäsittelytekniikan laboratorio

Distribution:
Helsinki University of Technology
Signal Processing Laboratory
P.O. Box 3000
FIN-02015 HUT
Tel. +358-9-451 3211
Fax. +358-9-452 3614
E-mail: Mirja.Lemetyinen@hut.fi

ISBN 951-22-7526-0 (Printed)
ISBN 951-22-7527-9 (Electronic)
ISSN 1458-6401

Otamedia Oy
Espoo 2005

Abstract

This doctoral thesis consists of an introductory part and eight appended publications, which deal with hardware–based reprogrammability in algorithm acceleration with a specific emphasis on the possibilities offered by modern large–scale Field Programmable Gate Arrays (FPGAs) in computationally demanding applications.

The historical evolution of both the theoretical and technological paths culminating in the introduction of reprogrammable logic devices is first outlined. This is followed by defining the commonly used terms in the thesis. The reprogrammable logic market is surveyed, and the architectural structures and the technological reasonings behind them are described in detail. As reprogrammable logic lies between Application Specific Integrated Circuits (ASICs) and general–purpose microprocessors in the implementation spectrum of electronics systems, special attention has been paid to differentiate these three implementation approaches. This has been done to emphasize, that reprogrammable logic offers much more than just a low–volume replacement for ASICs.

Design systems for reprogrammable logic are investigated, as the learning curve associated with them is the main hurdle for software–oriented designers for using reprogrammable logic devices. The theoretically important topic of partial reprogrammability is described in detail, but it is concluded, that the practical problems in designing viable development platforms for partially reprogrammable systems will hinder its wide–spread adoption.

The main technical, design–oriented, and economic applicability factors of reprogrammable logic are laid out. The main advantages of reprogrammable logic are their suitability for fine–grained bit–level parallelizable computing with a short time–to–market and low upfront costs. It is also concluded, that the main opportunities for reprogrammable logic lie in the potential of high–level design systems, and the ever–growing ASIC design gap. On the other hand, most power–conscious mass–market portable products do not seem to offer major new market potential for reprogrammable logic.

The appended publications are examined and compared to contemporaneous research at other research institutions. The conclusion is that for relatively wide classes of well–defined computation problems, reprogrammable logic offers a more efficient solution than a software–centered approach, with a much shorter production cycle than is the case with ASICs.

Acknowledgments

I had the pleasure of working with many talented researchers during my career at the Signal Processing Laboratory at Helsinki University of Technology. The most enduring influence on my thinking can be traced to Dr. Jarkko Vuori, who nowadays works as a Professor at Jyväskylä University. I have always been very impressed by his wide experience on both practical and theoretical applications of reprogrammability. Other ex–members of the Signal Processing Laboratory with whom I had the pleasure of working with include Petri Jehkonen, Dr. Mika Loukola, Keijo Länsikunnas, Dr. Jarno Tanskanen and Jukka Tenkama. I would like to thank them for many fond memories of working together.

I would also like to thank the current members of the Signal Processing Laboratory with whom I had the pleasure of working together in our research projects. Thanks are therefore due to Juha Forsten, Jaakko Kairus, Matti Rintamäki, Antti Hämäläinen, Esa Korpela, Kimmo Järvinen, Kati Tenhonen and Sampo Ojala.

I would also like to thank my supervisor professor Jorma Skyttä for giving me complete scientific freedom to pursue my own areas of interest during my research. Furthermore, professor Skyttä took care of all the necessary administrative and bureaucratic tasks related to my dissertation, which helped me a lot during the final months of writing this thesis.

The reviewers, professor Jarmo Takala of Tampere University of Technology and professor Hannu Heusala of University of Oulu, deserve thanks for their valuable comments on a draft version of this thesis.

Our laboratory secretaries Anne Jääskeläinen and Mirja Lemetyinen deserve thanks for their continuous and successful efforts to make things run smoothly at the Signal Processing Laboratory. I have probably also been an occasional nuisance to the computer administrators at the Lab, for which I would like to express my sincere apologies.

I also had the honor of being enrolled for two years in GETA (Graduate School in Electronics, Telecommunications and Automation). I would like to thank GETA director Professor Iiro Hartimo and GETA secretary Marja Leppäharju for their successful efforts in making GETA the best graduate school in Finland.

Needless to say, if I have not included a person's name in the above lists, the omission is entirely due to my poor memory and must not be considered as a personal insult for the omitted person.

I would also like to thanks my current employer, AMD Finland, for their flexibility in the final phases of preparing this doctoral thesis.

The financial support of the Emil Aaltonen Foundation, Ella och Georg Ehrnrooths Stiftelse, Magnus Ehrnrooth Foundation and Nokia Foundation is greatly appreciated.

On a more personal level, I would like to thank my mother Eira and father Pekka for their continuous love and encouragement throughout my whole life. My brothers Esa and Mikko deserve thanks for being just what they are, namely great brothers.

Finally, I would like to express my deepest gratitude to my beloved wife Johanna for her love and support. Our son Atte also had a very important role during the writing of my thesis, namely off–loading my work–related thoughts just by his existence.

Espoo, 2nd of February, 2005

Matti Tommiska

List of Publications

This thesis consists of an overview part and the following publications:

P1 M. Tommiska and J. Vuori, "Hardware Implementation of GA", in *Proceedings of the 2nd Nordic Workshop on Genetic Algorithms*, Vaasa, Finland, Aug. 19–23, 1996, pp. 71–78.

P2 M. Tommiska, M. Loukola, and T. Koskivirta, "An FPGA–based Implementation and Simulation of the AAL Type 2 Receiver", *Journal of Communications and Networking*, vol. 1, no. 1, pp. 63–67, Mar. 1999.

P3 M. Tommiska, "Area–Efficient Implementation of a Fast Square Root Algorithm", in *Conference Proceedings of the Third IEEE International Caracas Conference on Devices, Circuits and Systems*, Cancun, Mexico, Mar. 15–17, 2000, pp. S–18–1 – S–18–4.

P4 M. Tommiska and J. Skyttä, "Dijkstra's Shortest Path Routing Algorithm in Reconfigurable Hardware", in *Proceedings of the 11th Conference on Field–Programmable Logic and Applications (FPL'01)*, G. Brebner and R. Woods, Eds., Belfast, Northern Ireland, UK, Aug. 27–29, 2001, pp. 653–657.

P5 M. Tommiska, J. Tanskanen, and J. Skyttä, "Hardware–Based Adaptive General Parameter Extension in WCDMA Power Control", in *Proceedings of the IEEE 54th Vehicular Technology Conference (VTC'01 Fall)*, Atlantic City, NJ, United States, Oct. 7–11, 2001, vol. 4, pp. 2023–2027.

P6 A. Hämäläinen, M. Tommiska, and J. Skyttä, "6.78 Gigabits per Second Implementation of the IDEA Cryptographic Algorithm", in *Proceedings of the 12th Conference on Field–Programmable Logic and Applications (FPL'02)*, M. Glesner, P. Zipf, and M. Renovell, Eds., Montpellier, France, Sept. 2–4, 2002, pp. 760–769.

P7 K. Järvinen, M. Tommiska, and J. Skyttä, “A Fully Pipelined Memoryless 17.8 Gbps AES-128 Encryptor”, in *Proceedings of the 11th International Symposium on Field–Programmable Gate Arrays (FPGA’03)*, Monterey, CA, United States, Feb. 24–26, 2003, pp. 207–215.

P8 M. Tommiska, “Efficient Digital Implementation of the Sigmoid Function for Reprogrammable Logic”, *IEE Proceedings – Computers and Digital Techniques*, vol. 150, no. 6, pp. 403–411, Nov. 2003.

Contents

Chapter 1

Introduction

Reprogrammable logic[1] has been around for decades, and it has struggled to find its place as a suitable compromise between the flexibility and adaptability of software running on general–purpose processors and the high performance of custom integrated circuits[2]. Reprogrammable logic has been generally regarded as a substitute for custom circuits in low–sized production quantities by the hardware designers, and on the other hand, software designers have been relatively uninformed of the possibility of running their compiled programs on other platforms besides general–purpose processors. Both of these viewpoints are missing a novel trend, which is that reprogrammable logic is emerging as an equivalent and important algorithm implementation platform of its own. This is mainly due to the advantages that the practically instant reprogramming of the entire device provides, both during product development and device operation.

This doctoral consists of an extensive and comprehensive introductory part, and appended eight Publications P1–P8, which describe the results in research projects that the author has participated in over the years. The common theme in the introductory part and the appended Publications are the advantages that reprogrammable logic –based implementations of computationally intensive algorithms have, especially when compared to software–based implementations.

The thesis is organized as follows: Chapter 2 presents an overview of the history and present–day reprogrammable logic technology followed by a retrospective look at reconfigurable computing, and the chapter is concluded by comparing

[1]In this thesis, reprogrammable logic encompasses all digital logic devices with device–wide reprogrammabality. See also Section 2.1.1.

[2]Section 2.3.2 reviews reconfigurable processors, which in an optimal case combine the best features of reprogrammable logic and processors.

microprocessors, reprogrammable logic, and custom integrated circuits.

Chapter 3 presents a detailed look at the architectural details of modern Field Programmable Gate Arrays (FPGAs), including their design challenges and opportunities, and presents examples of high–speed computing platforms based on reprogrammable logic devices.

Chapter 4 presents a summation on the technical, design–related, and economic applicability of reprogrammable logic in efficiently implementing algorithms and constructing large electronic systems. The main points are collected into a SWOT (Strengths, Weaknesses, Opportunities, Threats) matrix for reprogrammable logic.

Chapter 5 presents overviews of application areas covered in Publications P1–P8, and compares the author's results with those of contemporaneous research at other places.

Chapter 6 concludes the thesis by summarizing the main results of the research culminating in this thesis.

1.1 Author's Contribution in the Publications

The author's contribution in the appended Publications P1–P8 is as follows:

- The author performed all AHDL programming for the genetic algorithm, and was responsible for writing chapters 6.2 and 6.3 in Publication P1.

- The author performed all C, MATLAB, and VHDL programming for the AAL Type 2 receiver, and was responsible for writing chapters III–VII in Publication P2.

- The author is responsible for all research and writing in Publication P3.

- The author is responsible for the bulk of research and writing in Publication P4, with professor Skyttä performing supervisory and project management duties.

- The author is responsible for all C and VHDL programming for the GP–extended FIR predictor, and was responsible for writing chapters 1 and 4–6 in Publication P5. Furthermore, the author added significant portions to the MATLAB code provided by researcher Tanskanen.

- The author is responsible for suggesting the usage of diminished–one number representation and Ma's algorithm for the implementation of the multiplication block in the IDEA encryption algorithm, for supervisory and project management duties and for performing all VHDL recoding of the design. The author is responsible for writing most of Publication P6, with researcher Hämäläinen contributing for figures 3 and 4.

- The author is responsible for suggesting the usage of memoryless implementation of the SubBytes transformation in the AES encryption algorithm, for supervisory and project management duties, and for writing chapters 1 and 6–7 in Publication P7.

- The author is responsible for all research and writing in Publication P8.

Chapter 2

Reprogrammable Logic Devices

This chapter begins with a retrospective look at the history of reprogrammable logic up to the early nineties, followed by a clarification of the terminology used in this thesis and an introduction into current reprogrammable logic market. A brief description of reconfigurable computing is included to clarify its relationship to the concepts presented in this thesis. Finally, ASICs and microprocessors are compared to reprogrammable logic devices in terms of performance and flexibility, and particular emphasis is laid on the field of reconfigurable processors.

2.1 History of Reprogrammable Logic

The first descriptions of computing automata capable of reprogramming themselves were put forward by John von Neumann in a series of lectures and unfinished manuscripts dating back to the late 40's and early 50's. After the death of von Neumann in 1957, his works on self–reproducing automata were collected and edited by Arthur Burks, who published them in 1966 [277]. Although von Neumann is generally regarded as the main developer of the conventional serial model of computing, it seems obvious that during the last years of his life, he was more interested in more complicated computing automata.

In 1960, Gerald Estrin of University of California at Los Angeles proposed a variable structure computer system to achieve performance gains in a variety of computational tasks [88] [90]. The central idea was to combine both fixed and variable structure computer organizations, where the variable subsystem could be reorganized into a variety of problem–oriented special–purpose configurations.

The structure of a prototype of a variable structure computer system is also presented in [89].

In 1967, Robert Minnick published a survey of microcellular research [205]. He described both fixed and variable cell–function arrays. In fixed cell–function arrays the switching function of each cell remained fixed, and only the interconnections between cells were programmable. In the case of variable cell–function arrays, the function produced by each cell could also be determined by parameter selection.

In the seventies, interest in and the corresponding financial support for non–serial forms of computation seems to have tapered off. This was most probably caused by the introduction of the first microprocessor —the famous 4004 by Intel— in 1971 and the ever growing number and scope of applications enabled by the expanding market of microprocessors and microcontrollers [184]. For a moment, it seemed that software running on serial microprocessors could fulfill practically all computing needs.

In the eighties, there was a revived interest in both systolic and parallel architectures. This interest was inspired by the advances in semiconductor integration technology, the evolution of system design concepts [201], and by new and more demanding applications of supercomputing. An interesting design combining parallelism with reprogrammability was the Texas Reconfigurable Array Computer (TRAC) [175]. In the TRAC project, adaptability was achieved by the reprogramming of interconnections between individual computing elements.

In the late eighties and early nineties, the first computing platforms with substantial amounts of reprogrammable logic were designed. Noteworthy projects were the Programmable Active Memory (PAM) project at Digital Equipment Corporation's (DEC) Paris Research Laboratory (PRL) [28], and the Splash project at the Supercomputing Research Center in Bowie, Maryland [43] (See also Section 3.3). The speedups achieved by the PAM project were very impressive and as similar results were reported by other research groups at approximately the same time, interest into the possibilities of reprogrammable platforms increased substantially. This was also realized by two influential engineering societies, the Association for Computing Machinery (ACM) and the Institute of Electrical and Electronics Engineers (IEEE). These engineering societies began sponsoring two annual conference series about the applications of FPGAs (Field Programmable Gate Array), namely the ACM/SIGDA International Symposium on Field–Programmable Gate Arrays and the IEEE Symposium on FPGA–Based Custom Computing Machines. In Europe, the first International Workshop on Field–

Programmable Logic and Applications was held in 1991.

Manufacturing reprogrammable logic devices would not be possible without the advances in electronics, because large reprogrammable circuits have enormous silicon overhead; for example a reprogrammable device with 50000 usable gates may have well over a million transistors [258] (See also Figure 3.2). This demonstrates that the reprogrammable logic market has benefited tremendously from advances in semiconductor manufacturing technology.

The earliest electronic computers were built with error–prone vacuum tube technology. In 1959, Jack Kilby invented the monolithic integrated circuit at Texas Instruments, but it was not until the introduction of Intel's 4004 microprocessor in 1971 (see above) that general–purpose computers began to be integrated on the same silicon chip [259].

The first suggestion for an implementable reprogrammable logic device is probably due to Sven Wahlstrom, who in 1967 proposed the inclusion of additional gates to customize an array of integrated circuitry [279]. However, the silicon "real estate" was an extremely scarce resource in those days, and the idea that a large area of an integrated circuit would be dedicated to customizable interconnection structure was regarded as heretic.

In the late seventies, the first Programmable Array Logic (PAL) devices were introduced by Monolithic Memories. These were simple arrays of AND and OR gates, and their reprogramming could not be performed on–the–fly. Initially, PALs and their close cousins, Programmable Logic Arrays (PLAs) were used mainly as "glue logic" and there were not any serious attempts to implement demanding computation applications with them. PALs and PLAs were followed by Programmable Logic Devices (PLDs), first by Simple Programmable Logic Devices (SPLDs), and later by Complex Programmable Logic Devices (CPLDs). Xilinx, which was founded in 1984, introduced the world's first FPGA in 1985 [51][1].

Being the first in the FPGA market, Xilinx has continued to dominate it well into the present time, although its main competitor Altera has occasionally taken Xilinx' place as the market leader. Other noteworthy reprogrammable logic manufacturers include Actel, Atmel, Cypress and Lattice (See also Section 2.1.2). In the nineties and onwards, reprogrammable logic began to be regarded as the third, middle–of–the–road alternative to both ASICs (Application Specific Integrated Circuit) and microprocessors in the implementation of digital circuits and systems. The historical review of reprogrammable logic is summarized in Figure 2.1.

[1]The terminology is explained in Section 2.1.1.

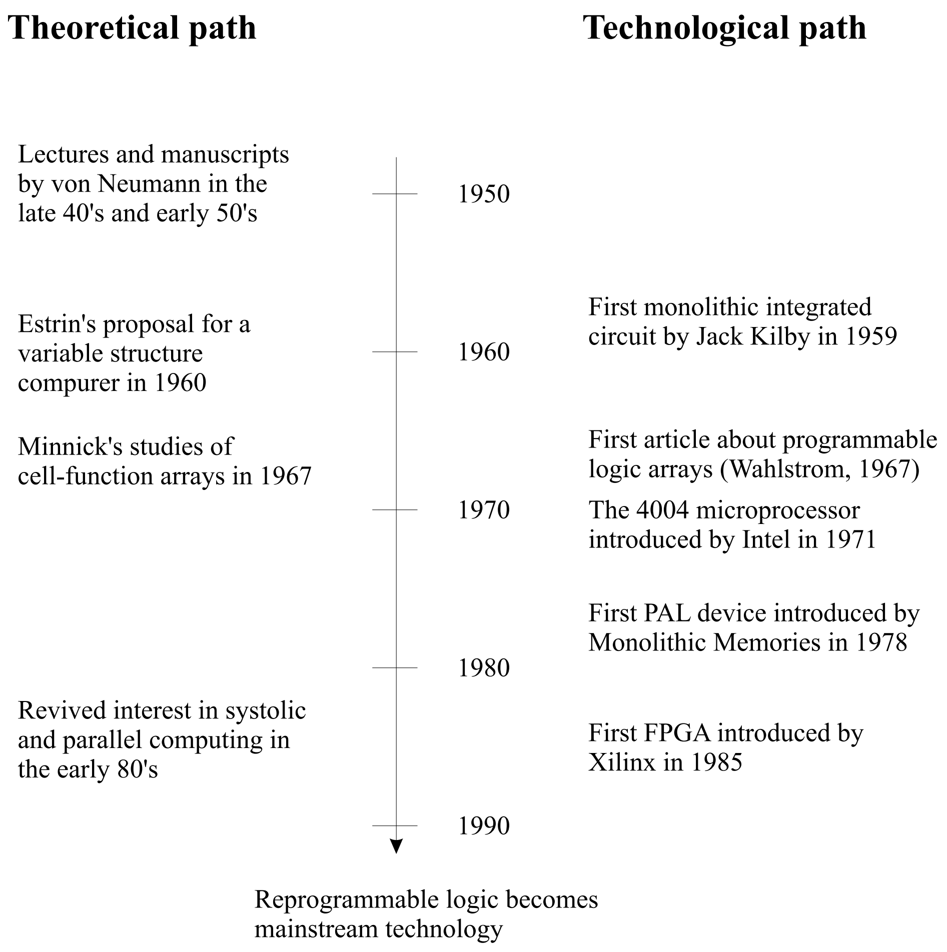
Theoretical path
Technological path
Lectures and manuscripts by von Neumann in the late 40's and early 50's
1950
Estrin's proposal for a variable structure compurer in 1960
First monolithic integrated circuit by Jack Kilby in 1959
1960
Minnick's studies of cell-function arrays in 1967
First article about programmable logic arrays (Wahlstrom, 1967)
1970
The 4004 microprocessor introduced by Intel in 1971
First PAL device introduced by Monolithic Memories in 1978
1980
Revived interest in systolic and parallel computing in the early 80's
First FPGA introduced by Xilinx in 1985
1990
Reprogrammable logic becomes mainstream technology

Figure 2.1: Historical timeline of reprogrammable logic into the nineties.

2.1.1 Reprogrammable Logic Terminology

Defining a universally acceptable terminology for reprogrammable logic is a daunting task, as most manufacturers and researchers (re)define common terms for their own purposes, and this fluidity in terminology has also been pointed out elsewhere [37].

As PLDs and FPGAs were introduced in the previous chapter, it is worth pointing out their main difference, which lies in the internal structure: whereas PLDs are based on wide two–level switching functions with even tens of inputs, FPGAs use a typically 4–input programmable lookup–table (LUT) with an optional output register as the basic structure [42]. Furthermore, FPGAs are volatile SRAM (Static Random Access Memory) –based, whereas PLDs are typically based on EPROM/EEPROM technology ((Electrically) Erasable Programmable Read–Only Memory). The timing characteristics of FPGAs are more dependent on signal routing and their architecture is more hierarchical, than is the case with PLDs. However, the distinctions between FPGAs and (C)PLDs are sometimes blurred, as there is a recent trend to adopt FPGA–based logic blocks into modern CPLDs, e.g. Altera's MAX II product line [192].

For the purposes of thesis, the term "*reprogrammable logic*" is used to encompass all digital logic devices, which incorporate device–wide reprogrammability, and thus both (C)PLDs and FPGAs are included.

The terms *reprogrammability* and *reconfigurability* are used interchangeably in this thesis, as in all practical cases the two terms are equivalent. Furthermore, the term "*FPGA*" is occasionally used interchangeably with "reprogrammable logic (device)". This is to conform with the original references and/or based on the assumption that the topic under discussion is specifically targeted for FPGAs, and not for (C)PLDs.

Reconfigurable computing, the topic of Section 2.2, is defined as the ability to modify the system structure during run–time [73].

The *granularity* of reprogrammable logic architectures is defined as the size and complexity of the basic computing block [62]. Generally, fine–grained commercial architectures are more flexible in implementing logic, but have longer reprogramming times and are less efficient in implementing arithmetic–heavy datapaths.

A system is classified as *partially reprogrammable* (in the paper [178], this is called dynamically reconfigurable, but for the purposes of this thesis, these terms are equivalent) if it can be partially reprogrammed while active [178].

The terms *compile–time reconfiguration* (CTR) and *run–time reconfiguration* (RTR) have been defined in [146], where CTR applications have been defined as systems with a single configuration, whereas RTR applications consist of multiple configurations.

2.1.2 Modern Reprogrammable Logic Devices

In 2003, the size of the global reprogrammable logic market was \$2.6 billion[2], of which Xilinx accounted for 51%, Altera for 32%, Lattice for 8%, Actel for 6%, with the remaining 3% belonging to other manufacturers [296]. Although the reprogrammable logic market of \$2.6 billion is less than the total ASIC market of \$13 billion in 2003 [220], the ratio between new ASIC designs and new FPGA designs is one to three [19] (See also Section 2.3.3), which is a solid argument to regard reprogrammable logic as a mainstream technology (See also Figure 2.1).

The main market segments for reprogrammable logic are communications and consumer electronics, with 37% of the annual volume targeted for production phases, 30% for preproduction phases, 30% for prototyping, and 3% for emulation [183]. Leading reprogrammable logic manufacturers do not own their fabrication facilities ("fabs"), but instead rely on the "fabless" corporation model, which allows them to exploit the most cutting–edge manufacturing technology without having to invest into modern multi–billion dollar fabs. For example, in March 2003 Xilinx became the first reprogrammable logic manufacturer to ship 90–nm line–width reprogrammable logic devices, but the actual production of these devices takes place at fabs operated by United Microelectronics Corporation (UMC) and International Business Machines Corporation (IBM) [297].

A list of representative and popular reprogrammable logic devices is presented in Table 2.1. The distinction between CPLDs and FPGAs is done by the manufacturers. The gathered information originated from [161], and has been updated from the manufacturers' websites. The figures are as given by the manufacturers and they may not be fully comparable with each other[3]. For example, I/O numbers may include pins not only for general–purpose use, but also clock inputs may be included. It has been assumed, that the manufacturers have used the standard 2–input NAND gate requiring four transistors as the definition of one gate,

[2]The total global semiconductor market in 2003 was \$166.4 billion [230], with reprogrammable logic outpacing average market growth [229]

[3]Exact comparison between advertised and estimated gate count figures is discouraged, as there are no independent third party measurements (See also Section 3.1.1).

but this has not been explicitly stated in the data sheets. Especially the largest FPGA families have embedded processors and hardwired multiplier blocks (See also Section 3.1.1, but these characteristics have not been represented in detail in Table 2.1.

2.2 Reconfigurable Computing

At the beginning of the 1990s, the research community began to view the volatility of SRAM–based FPGAs as a strength instead of a liability. It was assumed, that the reprogrammability of FPGAs was in fact the key to new types of applications [132]. One of the first such applications was logic emulation, where reprogrammable systems outperformed software simulation by orders of magnitude. FPGAs were being seen as an ideal building block for multimodal and generic hardware, and it was visioned that a future hardware supercomputer would consist of an array of interconnected FPGAs.

In the mid 1990s, the semiconductor industry was actively looking for a replacement for the traditional sequential microprocessor as the driving force for further revenues [258]. Reprogrammable logic was seen as a promising alternative or at least as a supplement to microprocessor, despite its overhead and unsparing use of silicon real estate. On the theoretical front, there were efforts to define a restricted domain of the general–purpose architectural space focused on *Reconfigurable Computing* [73] (See Section 2.1.1). At around the same time (mid 1990s), the term Configurable Computing Machine (CCM) was used to describe an FPGA system that could be customized to the task at hand [34].

After an initial enthusiasm for reconfigurable computing, realism began to replace over–optimistic predictions. Realizing the potential of reconfigurable computing systems outside research laboratories proved difficult, because these systems relied on manipulating low–level abstractions and thus required high–skilled developers [185]. It was deemed unlikely that reconfigurable computing would make significant inroads against the microprocessor in the foreseeable future. Major drawbacks included a lack of high–level software programming model and a consequent lack of truly automatic mapping tools [273].

Interest in reconfigurable computing meant broadening the scope of implementation platforms from fine–grained FPGAs to coarse–grained and mesh–based architectures and related CAD software. It was argued, that the classical "von Neumann" computer was becoming obsolete and being replaced by a new "soft

Table 2.1: Popular reprogrammable logic devices in the market (July 2004). Updated from [161].

Manufacturer					
	Family	Type	Max. gates (k)	RAM bits (k)	User I/O
Actel	Axcelerator	FPGA	125–2000	29–338	168–684
	ProASIC$^{\mathrm{PLUS}}$	FPGA	75–1000	27–198	158–712
Altera	MAX 3000A	CPLD	0.6–10[a]	–[b]	34–208
	MAX II	CPLD	3.6–32[a]	–[b]	80–272
	Stratix II	FPGA	600–8000[a]	419–9383	365–1173
	Cyclone II	FPGA	180–2800[a]	120–1152	142–622
	Stratix	FPGA	422–3300[a]	920–7428	426–1203
Atmel	ATF15	CPLD	1.5–12	–[b]	36–212
	AT40	FPGA	5–50	2–18	128–384
	AT6000	FPGA	6–30	–[b]	96–124
Cypress	Delta39K	CPLD	30–200	70–480	174–428
	Quantum38	CPLD	48–144	16–48	174–302
Lattice	ispXPLD	CPLD	75–300	128–512	141–381
	ispXGPA	FPGA	139–1250	92–414	176–496
	LatticeECP–DSP	FPGA	400–3000	117–709	224–576
Xilinx	CoolRunner–II	CPLD	0.75–12	–[b]	33–270
	Spartan–III	FPGA	50–600	32–288	182–514
	Virtex–II	FPGA	40–8000	72–3024	88–1108

[a] As Altera no longer publishes equivalent gate count figures, the table uses estimates based on the logic element to gate conversion factor inferrable from Altera's older datasheets.

[b] No dedicated RAM blocks.

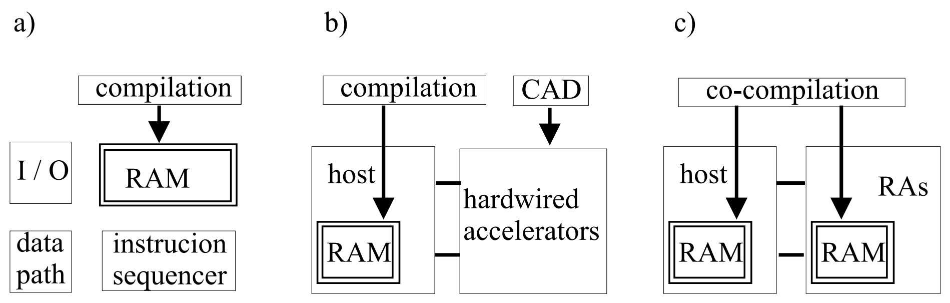

Figure 2.2: A speculative view on computing platforms: a) von Neumann, b) current, c) emerging [129]

machine" paradigm with a host and reconfigurable arrays (RAs) [129]. If this became true, performance is not the only feature that improves, as the flexibility of the host/accelerator system to support turn–around times of minutes instead of months is certainly also interesting. A fully integrated design language allowing co–compilation is naturally an absolute requirement for this to succeed. These speculative topics are described in Figure 2.2.

A good review of the most recent developments in reconfigurable computing is presented in [62], which considers both hardware aspects, design software and run–time reconfiguration. Another recent review [37] summarizes the main differences of reconfigurable logic and traditional processing architectures as spatial vs. temporal computation, configurable vs. fixed datapath, distributed vs. centralized control and distributed vs. centralized resources. Although the reconfigurable computing never materialized, it can be argued that the contemporary incorporation of reconfigurable datapaths in both ASICs and microprocessors can be traced to the same phenomenon in changing design approaches, namely a fresh look at the potential offered by reconfigurable computation elements.

2.3 Reprogrammable Logic, Processors, and ASICs

In an ideal case, reprogrammable logic combines the best of both worlds: the flexibility of software run on general–purpose processors and the speed of tailored hardware, namely ASICs (Application Specific Integrated Circuit). This tradeoff is illustrated in Figure 2.3, and a more quantified comparison between

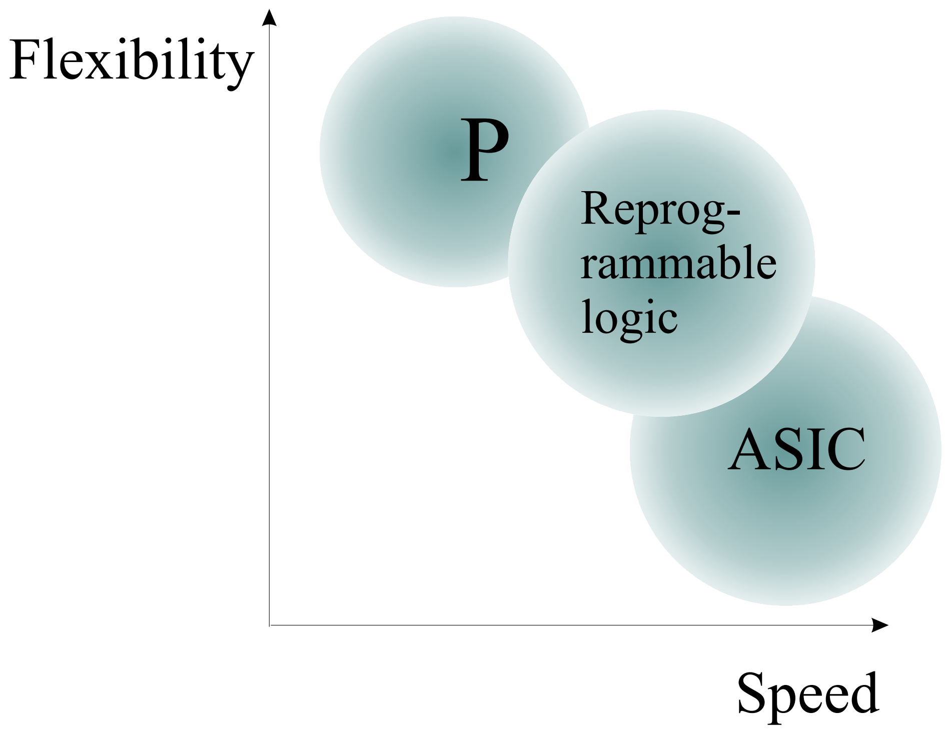

Figure 2.3: Comparing processors, reprogrammable logic and ASICs. There are overlaps between regions, due to designer agility and design schedule requirements.

reprogrammable logic, processors and ASICs is made in the following Sections. Special attention is paid to an emerging and promising field of reconfigurability in processors in Section 2.3.2.

2.3.1 Reprogrammable Logic and Processors

Most current high–performance computing platforms have a processor as the main workhorse, which makes the platform flexible enough to execute a large class of applications. The internal architecture of a processor is fixed, which implies that the sequential instruction decoding and execution, memory access bottleneck, and fixed control architecture limit the achievable performance. Modern processors consist of small execution engines, which are multiplexed heavily to support demanding computations on a relatively small amount of active hardware.

In reprogrammable logic, computations are implemented by spatially com-

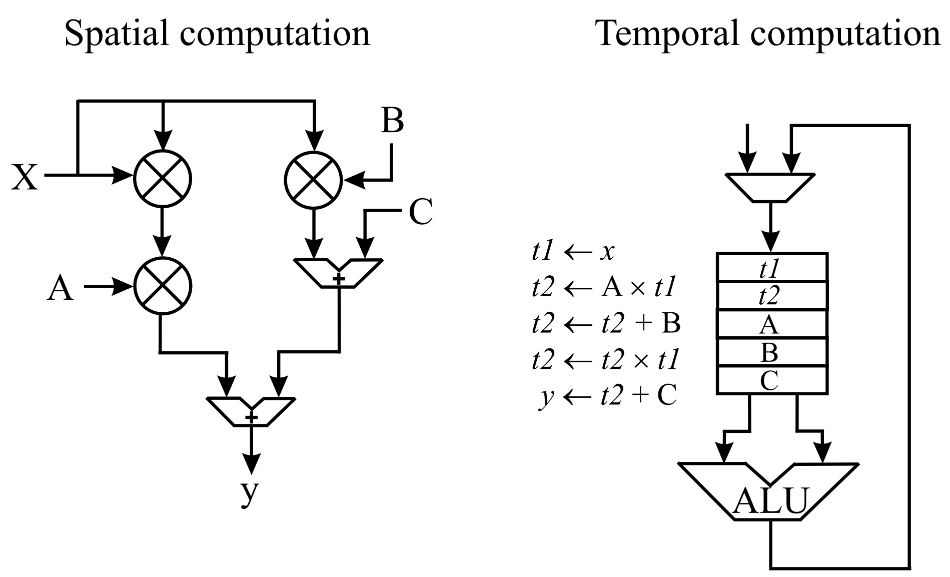

Figure 2.4: Spatial vs. temporal computation for $y = Ax^2 + Bx + C$ [76].

posing primitive operators rather than temporally composing them of primitive instructions as in microprocessors (See Figure 2.4). This way, reprogrammable logic can be thought of as offering higher peak computational density at the cost of lower instruction density. [76]

The continuing decline in the line width of semiconductor manufacturing processes does not necessarily mean that processor performance scales linearly with feature size. Increased power consumption, on–chip interconnection delays, memory access time (the so–called "memory wall"), and escalating heat problems (the so–called "thermal wall") shift the design effort from processor to the entire system. Silicon die area may be of little concern in the future, but there are physical and program behavior limits to taking advantage of additional area resources, when processors with hundreds of millions of gates are being designed. [95]

A probable trend in future microprocessors is to incorporate adaptability to the hardware, where reconfigurable logic will support the spatial composition of regular computations. The major functional blocks traditionally found on microprocessors may well be interconnected in a reconfigurable manner. Furthermore, as the well–known "90/10 rule" implies (90% of runtime is consumed in 10% of code), operations locality may exhibit sufficient regularity for a reconfigurable implementation within a processor [76]. Increasing the performance of microprocessors cannot be achieved by simply adding fixed functional units, as this neither yields the highest desired performance in proportion to the area which the fixed units consume, nor allow the construction of low–cost systems of wide applicability [70].

2.3.2 Reconfigurable Processors and FPGAs as Coprocessors

General–purpose microprocessors for modern desk–top computers dedicate more silicon area for the application–specific accelerators than for the microprocessor core itself [128]. This has blurred the distinctions between conventional microprocessors and reprogrammable logic, which has resulted in intense research into reconfigurable processors, and also commercially successful products have been introduced. Reconfigurable processors can be tuned for a particular computation task after the physical hardware has been designed, for example by adaptively reconfiguring their instruction set. Reconfigurable processors are generally divided into three main classes [21] (See also Figure 2.5):

- Attached processor, where the reconfigurable logic is placed on an I/O bus of the host processor.
- Coprocessor, where the reconfigurable logic is placed next to the processor, and the communication protocol is similar to that used for floating–point coprocessor.
- Reconfigurable functional unit (RFU), where the reconfigurable logic is placed inside the processor, and the processor treats the RFU as if it were one of the standard units on the processor's datapath.

In all three classes, the conventional microprocessor is typically used to support the bulk of the functionality required to implement an algorithm, while the reconfigurable processor is used to accelerate only the most critical computation kernels of the program [134]. Common requirements for all classes of reconfigurable processors are moderately wide datapath, very short reconfiguration times, and the possibility to reconfigure the processor during runtime [128].

The PRISM–I (Processor Reconfiguration through Instruction–Set Metamorphosis) [16] platform is an example of the attached reconfigurable processor model. The proof–of–concept hardware platform consisted of a processor board with a 10 MHz Motorola 68010 processor, and a second board consisting primarily of four

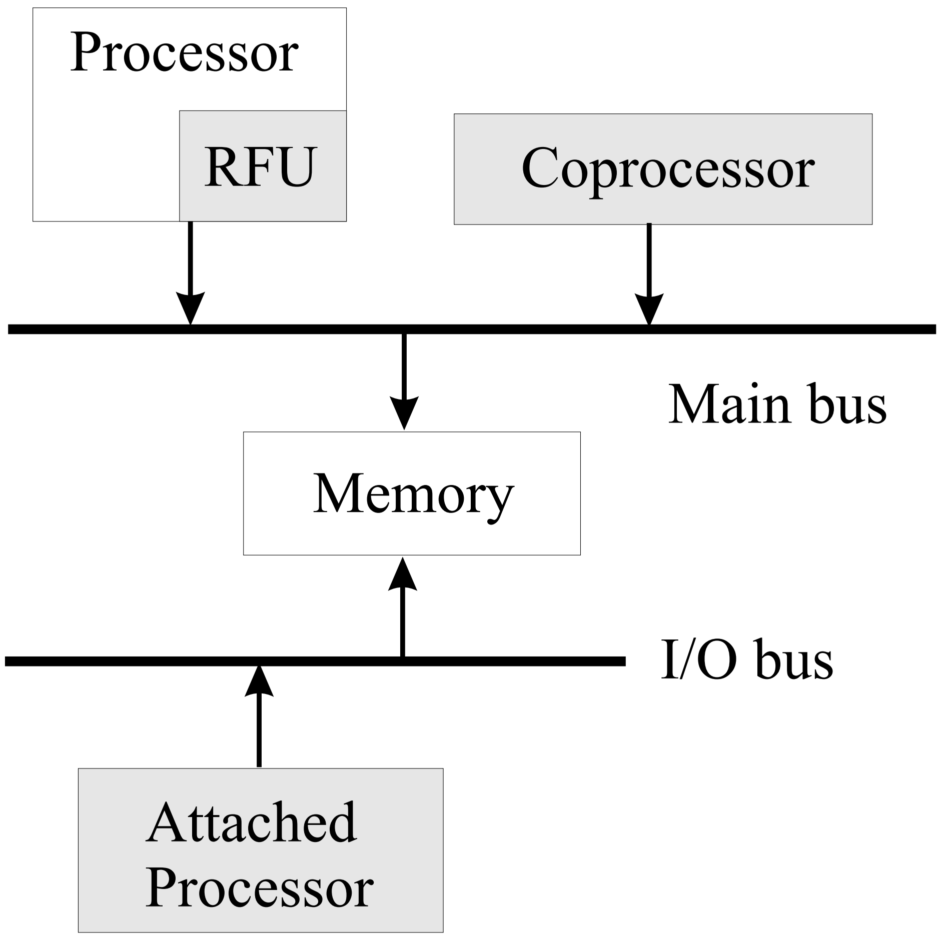

Figure 2.5: Three basic types of reconfigurable processors [21].

Xilinx 3090 FPGAs with a 16–bit bus connecting the two boards. The microprocessor implemented standard functions, and computationally demanding parts were identified and executed on hardware.

Both HARP [167] and Garp [136] can be regarded as reconfigurable coprocessors. The HARP platform consisted of a 32–bit RISC (Reduced Instruction Set Computer) microprocessor (a T805 transputer) with 4 MB of DRAM, closely coupled with a Xilinx XC3195A FPGA with its own local memory. A MIPS processor was combined with reprogrammable logic on the same silicon die in the Garp architecture, which allowed partial reprogramming controlled by the extended MIPS instruction set.

The majority of research into reconfigurable processors has concentrated on reconfigurable functional units (RFUs), where the instruction decoder issues instructions to the reconfigurable unit as if it were one of the standard units of the processor [21]. The Nano reconfigurable Processor [290] was implemented on a Xilinx 3000 series FPGA. Custom instructions were developed to achieve application–specific functionality, and the processor control was implemented within the FPGA instead of using a standard microprocessor. The DISC (Dynamic Instruction Set Computer) [287] reconfigurable processor had a morphable instruction set, which was implemented by partial demand–driven reconfiguration. A drawback of this scheme is the overhead caused by continually reconfiguring instruction modules.

The MorphoSys architecture [168] was targeted at computation–intensive and data–parallel applications, and combined a TinyRISC control processor with an array of reconfigurable cells. The execution model was based on partitioning applications into sequential and data–parallel tasks. The OneChip [50] reconfigurable processor integrated a reconfigurable functional unit into the pipeline of a RISC processor, and supported issuing multiple instructions simultaneously and performing out–of–order execution. The Chimaera [134] reconfigurable processor enabled multi–operand custom instructions be granting direct access to the register file of the host processor, which was also taken into account in the design of an automated compiler. Operating system issues have been taken into consideration from the very beginning in the Proteus [68] architecture, whose objective was to extend an ARM processor with reconfigurable functional units. An interesting application of reconfigurability in processors is presented in [231], where run–time adaptability allows an automatic evolution and refinement of the system to better suit run–time conditions. The so–called polymorphic processor paradigm, which allows the programmer to modify the processor functionality and hardware at will without architectural and design modifications, is the object of research in the MOLEN reconfigurable processor project [270].

Commercial reconfigurable processors are offered, among others, by TenSilica with its configurable and extensible 32–bit Xtensa microprocessor core, and by Triscend (acquired by Xilinx in March 2004) with its A7 and E5 families of customizable microcontrollers. Altera's Excalibur devices integrate an industry–standard ARM922T processor with debugging modules, on–chip memory, and peripherals with an APEX 20KE device–like architecture [91], and Xilinx has integrated the IBM PowerPC 405 core into the Virtex–II Pro device family [274] (See also Section 3.1.1).

Work on reconfigurable processors is still primarily in the research phase, with significant challenges remaining [81], especially in improving current immature hybrid CPU/FPGA programming models [10]. Nevertheless, there are promising application domains for reconfigurable processors, such as communications, multimedia and cryptography [87].

2.3.3 Reprogrammable Logic and ASICs

At the other end of the computing spectrum, ASICs are designed for a specific application. Therefore, they have a superior performance for a highly restricted set of computing tasks, due to their fixed functionality. For example, in certain tightly defined radar signal processing applications, ASICs seem to outperform best available FPGAs by a factor of two [207] [188]. An obvious disadvantage of ASICs is the exclusion of post–design optimizations and feature upgrades. [37]

The fundamental reason for the popularity of ASICs in mass–market applications is well–known: for any algorithm to achieve maximum throughput, it should be implemented in hardware [53]. ASICs also dominate the low–power market segment, where reprogrammable logic is not as competitive (See also Section 3.1.3). When a designer decides to go for an ASIC–based solution, he prefers speed over generality, since ASICs provide precisely defined functionality for a specific task. It has been estimated, that the ratio between new ASIC designs and new FPGA designs is one to three (with FPGA designs in the majority), and that the ratio is growing more and more advantageous for the FPGA manufacturers [19].

Reprogrammable logic and ASICs have connections and similarities in both economical and development respects. Reprogrammable logic may be used in either pre– or post–ASIC production phases, where the small market size makes reprogrammable logic more economical than ASICs. There are also several migration paths for converting an FPGA–based design into an ASIC [23], for example the leading reprogrammable logic manufacturers both offer a service to convert existing FPGA design into structured ASICs, as evidenced by Altera's HardCopy^{TM} [127] and Xilinx' EasyPath^{TM} [275] solutions. Software development environments for both reprogrammable logic and ASICs have much more in common than the push–button approach available in software engineering [164]. Using a Hardware Description Language (HDL) or a low–level schematic entry tool is unheard of in modern software development, whereas they are commonplace in hardware–based design. When the price of development tools for FPGAs and ASICs is compared, the similarities end: FPGA design tools are by far less expensive than corresponding ASIC tools [19].

The semiconductor industry has traditionally regarded ASICs as the only economical alternative for mass–market products, and the first reprogrammable logic devices did not directly encroach on ASIC territory. However, this may be changing, as modern million–gate FPGAs enable Configurable System on Chip (CSoC) products allowing unprecedented levels of system integration. Reprogrammable logic may have several benefits over ASICs, such as reduction in costs, time, and resources relative to ASIC designs, flexibility to implement different functions using the same devices, and wider vendor sources of FPGA technologies than with ASIC designs [207]. Consequently expanding markets, such as networking and wireless communications, are not anymore the exclusive realm of ASICs. At the same time there is also a move away from programming the end product only once, for both economical and technological reasons. The economical reasons include time to market and upgrade revenues, and the technological reasons comprise bug fixes, customization, and remote diagnostics and monitoring. All this motivates to search for new business models for reprogrammable logic. [158]

According to the so–called Makimoto's wave [182], the semiconductor industry swings between standardization and customization, with changes in directions taking place roughly every ten years. Reprogrammable logic represents both extremes of Makimoto's wave, as they are standardized in manufacturing but customized later in application. Coupled with the current shift from the "PC" (Personal Computer) to "DC" (Digital Consumer) products, this means that reprogrammability is a necessity in mass–market applications, which may have a short time to market and a dramatic end of life. [182]

Communications and networking may become the first mass–market application area where reprogrammable architectures find widespread acceptance as a true alternative to ASIC–based solutions. As the complexity of electronics systems outpaces the technology evolution predicted by Moore's law, the integration density advantage of ASICs begins to lose its importance. Evolutionary and adaptive environments require flexible solutions, which are causing a shift in the design paradigm towards a platform–based design [221]. The combined need of flexibility and energy–efficiency benefit from using reprogrammable architectures, as energy–consuming blocks can be turned on only when needed. The potential to integrate application–tailored coarse–grained dynamically reprogrammable architectures has been studied, for example, in the Dynamically Reconfigurable Architecture for Mobile Systems (DReAM) research project [24].

2.3.4 Quantitative Comparisons

There have been surprisingly few published articles with a rigorous investigation of the quantified performance differences of FPGAs, ASICs and microprocessors with a wide spectrum of compared algorithms and applications. Furthermore, design effort and required time for a design cycle are often omitted from these comparisons. This section presents three quantified comparisons of FPGAs, ASICs and microprocessors. Further material can also be found in Chapter 4 and the appended Publications P1–P8.

A quantitative performance comparison of FPGAs, ASICs and digital signal processing (DSP) processors using actual fixed–point DSP applications and CAD tools demonstrated that FPGAs outperform DSP processors by an order of magnitude, and in many cases approach the ASIC performance [215]. The comparisons concentrated on the performance of a multiplier, which is a core building block in almost every DSP application. Single–dimensional Finite Impulse Response (FIR) filters and a Fast Fourier Transform (FFT) calculation were used as benchmarks. It was found, that for FPGAs to achieve a performance increase in DSP applications over DSP processors and ASICs, specialization and parallelism must be employed to the fullest. Specifically, constant coefficient multipliers in FPGAs almost equalled the performance of ASICs.

When floating–point applications were compared in another study [27], different results in DSP performance were obtained when FPGA–based custom computers were compared with general–purpose computers. The conclusion was, that custom computers based only on FPGA execution units improve only little the performance of state–of–the–art workstations. The most important reason for this was, that both the TMS320C40 and Alpha 21064 (state–of–the–art processors of that time) had high–performance hardware support for floating–point arithmetic.

The implementations of a combinatorial search problem in both software and reprogrammable logic were compared in [232]. The particular problem was the search for approximate solutions of overconstrained systems of equations over GF(2), which is of practical interest in cryptanalysis. It was discovered, that careful programming of a modern microprocessor with good compiler support yields comparable performance, such that an FPGA–based solution, albeit faster, may not be worth pursuing. This is especially true for algorithms with high data dependencies and sequential computation flow, whereas algorithms with a high degree of parallelism are better suited for implementation on reprogrammable logic than on microprocessors. Both microprocessors and reprogrammable logic have well–known performance thresholds in problem size; for reprogrammable logic this means exceeding the available logic resources, whereas for a software–based solution, performance degrades after the problem size has exceeded the size of the first level cache.

Chapter 3

Designing with Reprogrammable Logic

The purposes of this Chapter are to describe the architectural characteristics of modern large–scale reprogrammable logic devices, and to survey the advantages and disadvantages of reprogrammability with a particular emphasis on partial reprogrammability, to present examples of reprogrammable computing platforms, and to survey the software design tool issues in reprogrammable logic design.

3.1 Architectural Characteristics

The two main building blocks in reprogrammable logic devices are logic and interconnections, and both these are reviewed in separate subsections. The emphasis is on modern large–scale FPGAs, since they represent the highest level of development in reprogrammable technology. As reprogrammable logic device capacities grow, power consumption is increasingly a matter of grave concern for application developers, and power–related issues are described in their own subsection. The description of reprogrammable logic architectures is concluded by a survey into several non–commercial and unconventional architectural solutions.

3.1.1 Computational Elements

A hierarchical view of a modern FPGA[1], namely Xilinx Virtex–II[2], is presented in Figure 3.1. The following paragraphs highlight the historical and theoretical background for basic logic size, logic block clustering, cascade and carry chains, embedded memory, clock distribution and I/O. It should be remembered, that only a small portion of the silicon area of a modern FPGA is dedicated to active computational elements. A typical rule–of–thumb is a 1:10:100 relationship between action logic, configuration memory, and programmable interconnect [75], which is also presented in Figure 3.2.

For comparison purposes, the ratio of routing to logic cell area (the so–called routing factor) in standard—cell ASIC design depends primarily on the number of metal layers. With two metal layers the routing factor is typically between 1 and 2, whereas with three or more metal layers with over–the–cell routing enabled, the routing factor is usually zero to 1. When test structures are added, it can be estimated that the relative area of core logic in a standard–cell ASIC design varies between 20–80% [243]. On the other hand, examples of processor core areas are given, among others, in [84], and generally processor cores occupy 10–20% of die area in modern general–purpose microprocessors.

Good reviews of the design issues with FPGAs are presented in [59] and [248]. It is emphasized, that the basic building block of an FPGA should be easily arrayable in both horizontal and vertical directions. A regular structure is easier to manufacture than an FPGA with a variable–grain architecture, i.e. where the size of a logic block is not constant in the entire design.

Before a more detailed examination of the inner structure of modern FPGAs begins, a few cautionary remarks are in place. Benchmarking and comparing different FPGA device families and their architectures is challenging, as there are various independent variables that have an effect on the performance. These include, but are not limited to, gate count, types of logic circuits, types of memory circuits, ratio of logic to memory, architecture, and design software tools. The device vendors' own benchmarks and test runs should also be taken with a grain of salt [78]. FPGA performance depends a great deal on the way that CAD tools map circuits on the chip, and also the gate count figures advertised by device manufacturers cannot usually be verified by independent outsiders [42] (See also

[1]CPLDs are not handled separately in this thesis, as they do not represent the high–end spectrum of reprogrammable logic (See also Table 2.1).

[2]There is variation in the hierarchy and internal among different devices and manufacturers, but the depicted Virtex–II device can be said to represent a typical modern FPGA.

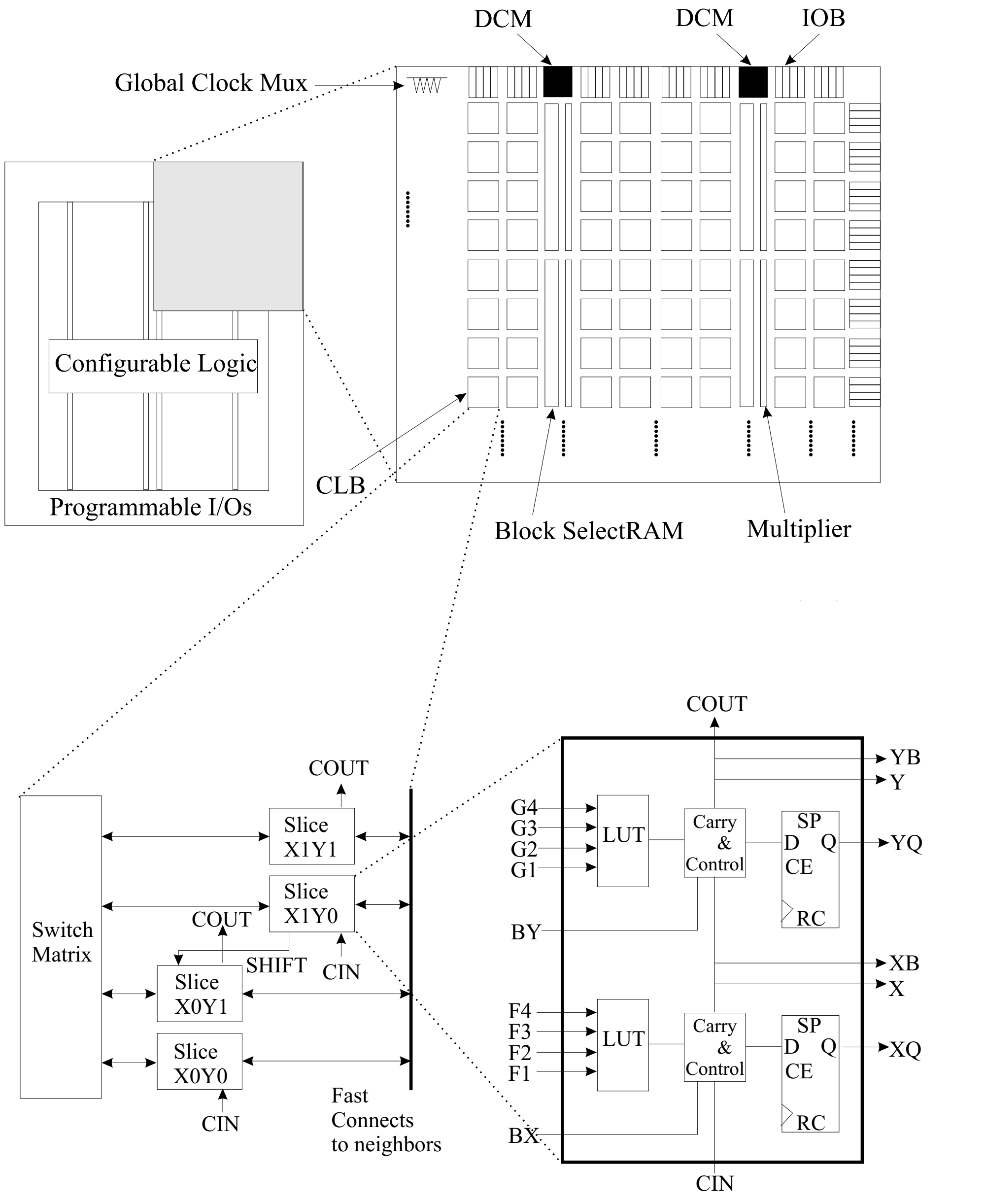

Figure 3.1: Hierarchical structure in Xilinx Virtex–II devices. Adapted from [274].

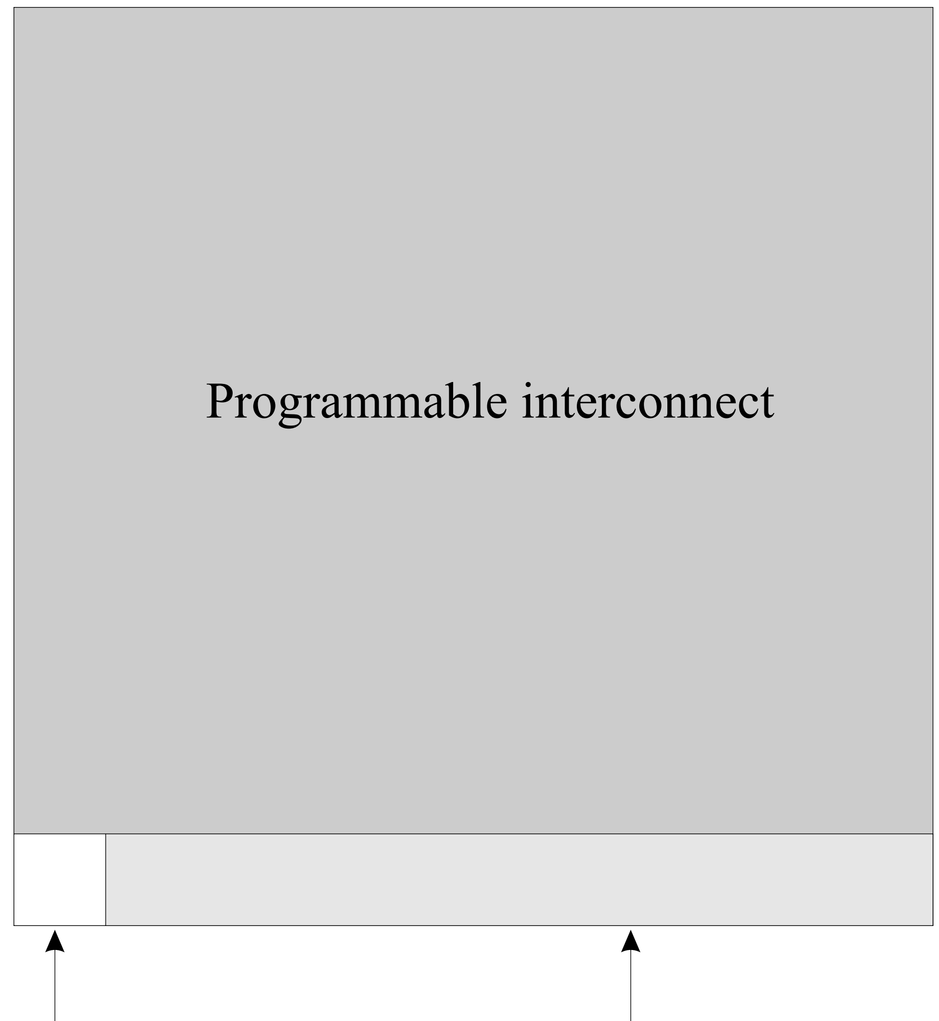

Figure 3.2: The relative area of logic, memory and interconnect [75].

Table 2.1).

The fundamental structure of the logic block was heavily investigated in the early nineties, and the conclusion was to use a four–input lookup–table (LUT) as the best compromise between area requirements, timing delay, and flexible functionality. These findings are relevant also today, as most commercial FPGAs manufacturers use a four–input LUT as the basic building block of their devices. On the silicon level, a LUT is implemented as a programmable static RAM lookup table [223].

The effect of the logic block functionality on the total delay of an FPGA was investigated in [240], where four basic architectures were compared. The experiments indicated that five– or six–input LUTs had the lowest total delay over a set of relevant logic circuits. Multiplexer–based block was close behind, followed by wide–input AND–OR gates and NAND gates. These findings corroborated previously research, which had found that a four– or five–input lookup table achieved the minimum average critical path delay over a large set of design examples [162].

The effect of logic block functionality on area efficiency was the subject of research in [223], where the best results were achieved with three– or four–input LUTs, largely independent of the programming technology. It was also found, that the logic block should contain a D flip–flop, since this reduces the required area in sequential circuits.

Individual logic blocks are clustered together with high–speed local interconnections for three main reasons: less delay, less area, and shorter compilation times [31]. The reasons for these improvements are clear: local interconnect is faster than general–purpose routing, most area is consumed by general–purpose interconnect with its long metal wiring, and placement and routing is the most time–consuming step in mapping a design to an FPGA.

A cluster is defined as a group of basic logic blocks, that are fully connected by a mux–based crossbar (See also Figure 3.3). Clustering has its origins in VLSI design, where it has been used to construct a natural hierarchy of the circuit elements [239]. FPGA manufacturers have for several years incorporated a cluster–based architecture in their devices.

The effect of cluster size on FPGA performance has been researched in both [5] and [187]. The results of [5] indicate, that with LUT sizes of 4 to 6 a cluster size of between 4 and 10 provides the best area–delay product in an FPGA, when also deep sub–micron electrical effects are taken into account, and detailed simulations are carried out. FPGA architectures with cluster sizes ranging from 1 to 20 were compared in [187]. The best performance was achieved with size 8 clusters,

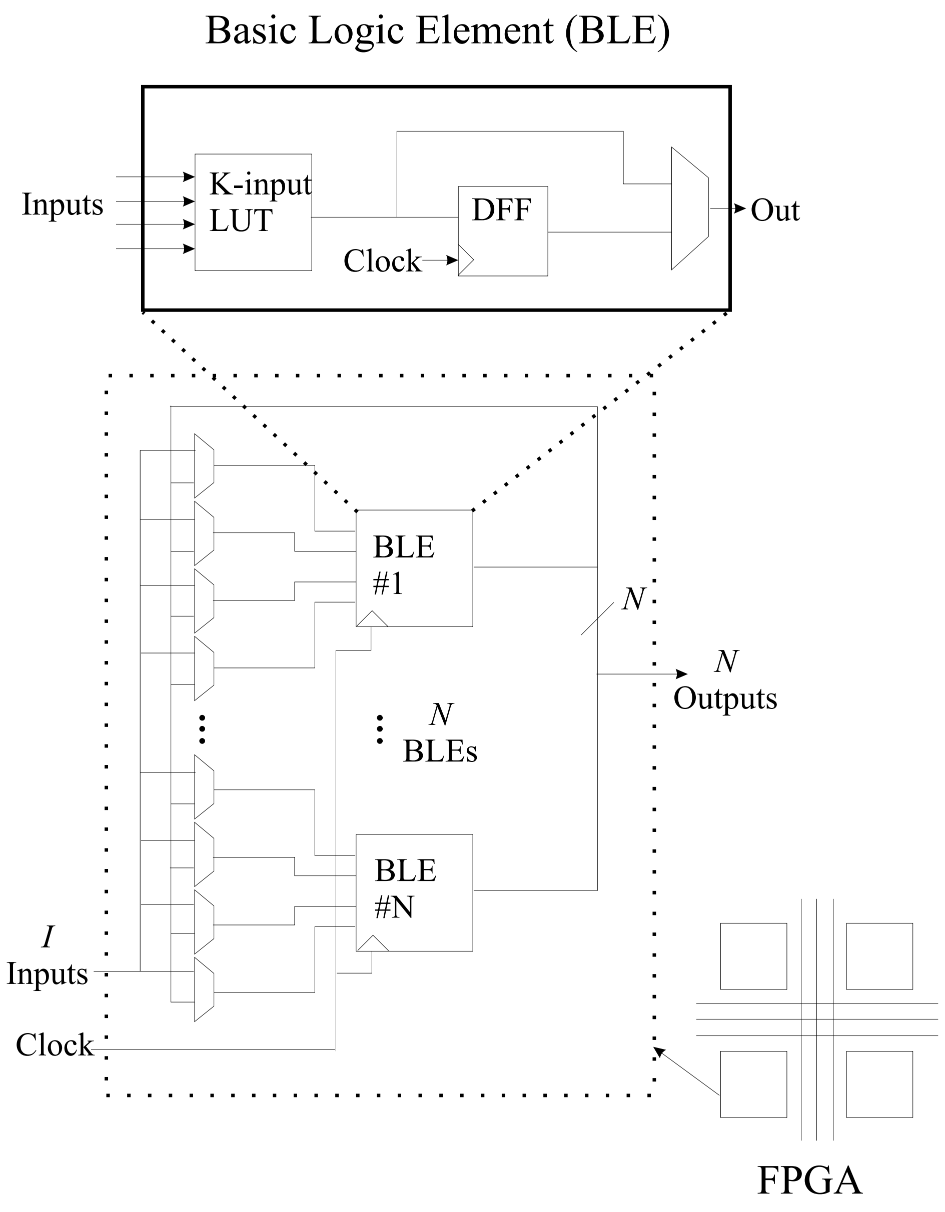

Figure 3.3: Structure of a clustered FPGA [5].

which had 23% less delay and 14% less area than size 1 clusters. As an example of the reduction in compilation times, size 20 clusters required seven times less time than size 1 clusters to compile.

Carry and cascade chains are usually embedded in the basic logic block to facilitate the implementation of adders, comparators, bit shifters, counters and equality checking circuits. They are also used in high–speed connections between adjacent logic cells for functions whose implementations requires more area than provided by a single logic block. An enhanced cascade chain based on a tree structure was proposed in [294], where it was argued that the new structure would reduce delay from linear time to log time in terms of the number of logic cells cascaded. High–performance carry chains based on carry select, carry lookahead, and Brent–Kung adders were proposed in [135], where an average of 3.8 improvement over traditional ripple carry based carry chains was reported.

Modern FPGAs have embedded memory blocks with even millions of bits, which act either as internal memory or as additional combinational logic for functions with wide inputs. The instantiated memory can be configured to support a wide variety of features, and can also have dual–port functionality, and be either a synchronous or synchronous [137]. Flexible amounts of embedded memory help in finding the right balance between logic and memory in an individual design, but the generation of software design algorithms to take advantage of the additional flexibility remains a challenge [138].

Large–scale modern FPGAs require global clock distribution of high quality, and therefore fast clock paths are slowed down to be as slow as the slowest path to achieve low–skew global clock signals. This requires dedicated de–skewing circuitry. The I/O capacity of large FPGAs is limited, since the minimum pad spacing shrinks much slower than the line width of semiconductor manufacturing processes. This means that large capacity FPGAs are often pin limited [263]. Most modern FPGAs have several Phase–Locked Loops (PLLs), which enable the generation of multiple desired internal clock frequencies by PLL synthesis. Furthermore, tens of different electrical interface standards are also supported [6]. Modern FPGAs have also dedicated digital signal processing (DSP) blocks, for example Altera's Stratix devices DSP blocks consist of hardware multipliers, adders, subtractors, accumulators, and pipeline registers enabling up to 333 million samples per second rates [6]. Embedding processors onto modern FPGAs is also gaining popularity, for example Xilinx allows the integration of IBM PowerPC 405 cores onto Virtex-II Pro devices [274], and Altera offers Excalibur devices, where an ARM922T$^{\text{TM}}$ is implemented on an APEX 20KE device-like architecture [91].

Examples of the design choices in commercial FPGA devices over the years are briefly described below. Altera's FLEX 6000 architecture, introduced in 1998, has clusters consisting of eight four–input LUTs. Clusters are called Logic Array Blocks (LABs) and they have common LAB–wide clock and asynchronous control signals [271]. Altera's Stratix architecture, introduced in 2002, has clusters of size 10, and to provide support for digital signal processing, dedicated DSP blocks are available for common signal processing functions. The memory hierarchy of Stratix devices comprises three levels [172]. Variable–grain architectures have been proposed relatively seldom, although they might have advantages over constant–grained architectures in fitting a larger variety of functions on the basic logic block. Examples of variable–grain architectures are Vantis' VF1 FPGA architecture, which had a three–level logic hierarchy with both three– and four–input LUTs at the lowest hierarchical level [4], and Atmel's second generation FPGA architecture, whose lowest hierarchical level consisted of two– and three–input LUTs [157].

Recently, there has been a trend towards adaptive basic building blocks in high–end FPGAs. Altera has developed a novel structure called Adaptive Logic Module (ALM) for its new Stratix II FPGA family. Each ALM contains two adaptive four–inputs LUTs, and with up to eight inputs to the combinatorial logic block, one ALM can implement up to two independent functions, each of varying widths [46].

3.1.2 Interconnections

The internal interconnections in FPGAs are the primary reason for the speed and density gap between FPGAs and Mask Programmable Gate Arrays (MPGAs), since the MPGAs use mask–programmed metal wires and do not suffer from the capacitance, resistance, and size of programmable connections of FPGAs [60]. Interconnections consume most of the chip area and are the dominating factor of the overall circuit delay [234]. First FPGAs had a fully symmetrical structure, with a symmetrical grid of logic blocks and routing channels on all four sides of the logic block. This evolved into hierarchical FPGAs, which have a hierarchical structure of both logic blocks and interconnections. Research proved that hierarchical FPGAs could implement circuits with fewer routing switches and fewer switches in total, which meant lower density and less costs [3]. Reducing the number of programmable interconnect points that a signal must traverse to reach its target offers highly predictable signal delays, where predictability is defined as the accuracy with which interconnect delay is estimated when the gates have been placed on the logic blocks, but routing has not been completed [210].

The traditional measure of good FPGA design has been high gate utilization, i.e. the more logic is fitted into LUTs, the better the design. This view has been challenged in [74], where the relationships between high logic utilization and interconnection efficiency were researched. It was concluded, that high LUT utilization does not necessarily correlate with high area efficiency, as the amount of interconnect needed per LUT varies among designs and within a single FPGA. This means, that all LUTs and all interconnections cannot be used to their full potential at the same, but one resource must be underutilized to fully utilize the other. Furthermore, conventional FPGAs have substantially more configuration bits for interconnections than actually needed. Bloated configuration files could be compressed a great deal, if this were taken into account in design software [72].

The best distribution of routing segment lengths and the best mix of pass transistor and tristate buffer routing switches in island–style FPGA architectures was investigated in [32] and later elaborated on in [234] (See also Figure 3.4). The results indicate, that while most commercial FPGA architectures use mostly length 1 wires (where the length of a wire is measured in terms of the number of logic blocks it passes before being switched), the best performance both in terms of area and delay is achieved with wires of length 4 to 8. The distribution of pass transistors and tristate buffers seems to be about 50–50, with liberal switching between buffered and pass transistor tracks.

Interconnections are used not only between logic blocks, but also between memory blocks and logic. Adding direct programmable interconnections between the internal memories improves routability and speed of FPGAs, with only a small additional cost in required area. [285]

An active area of study is designing an optimal switch block, which is a programmable interconnect connecting each incoming track to a number of outgoing tracks. The flexibility of each switch block is the key to the overall flexibility and routability of the device [191], since the majority of the electrical delay occurs in interconnect and interconnect switching [148]. An analytical framework for switch block design is presented in [169], where the design of a switch block fabric containing wires of any length is described. The theory and design of universal switch blocks is presented in [236], where an algorithm to construct generic universal switch blocks is presented, and the results indicated that universal switch blocks improve routability at the device level. On the other hand, the traditional

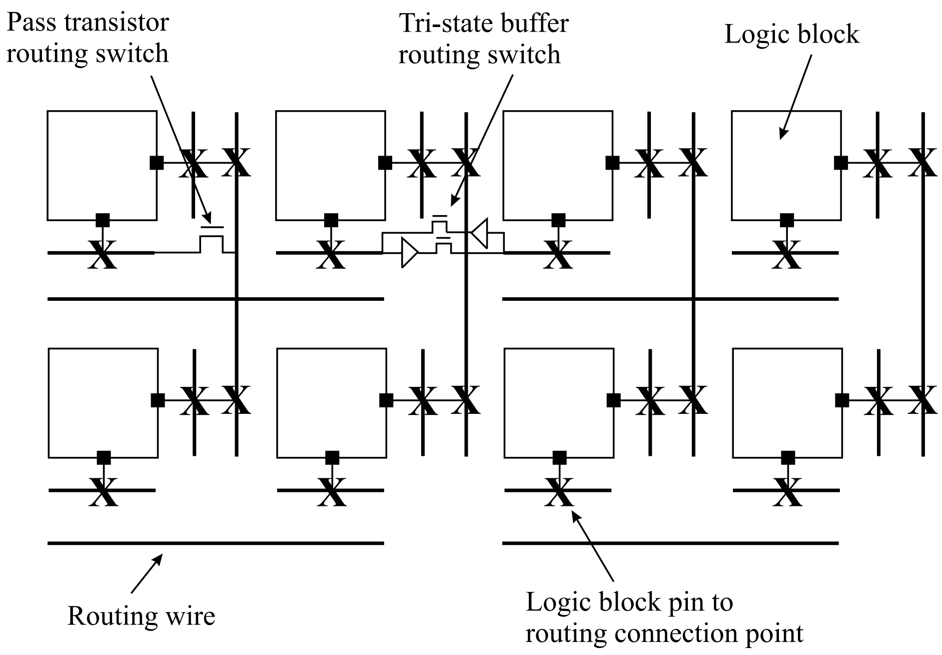
Pass transistor routing switch
Tri-state buffer routing switch
Logic block
Routing wire
Logic block pin to routing connection point

Figure 3.4: FPGA interconnection terminology [32].

Xilinx–style nonuniversal subset switch block was found superior to other proposed architectures in [226].

3.1.3 Power Consumption

Reducing power consumption has several advantages: avoiding expensive packaging, increasing the chip life operation, simplified cooling, and extending the lifetime of battery powered systems [36]. Therefore, reduced power consumption is a key design goal for portable computing and communication devices [1]. It has been projected that power consumption, along with increasing design complexity, will become the most serious design concern in the electronics industry. This applies as well to FPGAs, which consume more power than ASICs during high–speed operations, because the long routing tracks in FPGAs have significant parasitic capacitance which dissipates significant amounts of power during switching activity. Besides this dynamic power consumption, static power is expected to become an increasingly important part of the total power [218].

FPGA power consumption must be estimated at an early design stage step to develop a suitable board design and a power supply unit. P_{EST}, or the total estimated power consumption, of an FPGA consists of three parts:

$$P_{EST} = P_{STAT} + P_{IO} + P_{INT},$$

where P_{STAT} is the static power consumption, P_{IO} is the I/O–related power consumption, and P_{INT} is the internal power consumption [281]. The computed P_{STAT} value is usually in the range of a few microwatts and can be ignored in most power estimations. For FPGAs with a few number of on–chip gates and disproportionately many I/O pins, P_{IO} plays a major role in total power consumption. However, in most cases the power budget is dominated by the internal power consumption P_{INT}, which is primarily caused by the charging and discharging of the capacitance on each internal node that is switched. According to a detailed analysis of dynamic power consumption in Xilinx' Virtex–II FPGAs, P_{INT} stands for 90% of the total power dissipation on the average, with interconnections accounting for 60%, logic for 14%, and clocking for 10% of the total consumption. It was also concluded, that dynamic power dissipation of a Virtex–II CLB (Configurable Logic Block, the basic logic block in Xilinx FPGAs) is 5.9 μW per MHz for typical designs, but may vary significantly depending on the switching activity [233].

Careful design can produce substantial savings in total power consumption. Design pipelining can reduce power consumption by about 25–40%, and an additional saving of nearly 15–45% can be achieved by improving partitioning. These improvements are due to a reduction of the interconnection network influence [36]. Yet another design technique to achieve dynamic power savings is clock gating, which temporarily disables inactive elements in the circuitry. The basic idea is to replace the global clock to pipeline stages with local clocks, which may reduce the total power consumption by about 30% [48].

Present FPGA architectures will totally dominate the total power dissipation in a portable environment with a power budget in the milliwatt range. To overcome this, low–energy FPGA architectures have been proposed. An energy improvement of more than an order of magnitude was achieved by utilizing a hybrid interconnect structure and low–swing circuit design techniques in [102]. Similar techniques, namely a rich local–interconnect network and a dual–voltage scheme, were proposed in [165].

3.1.4 Innovative Reprogrammable Logic Architectures

Commercial fine–grained FPGAs are very efficient for implementing random logic functions, but they have weaknesses in general arithmetic functions, reprogramming and compilation times, forward compatibility, and size constraints [82] [111]. For these reasons, mostly academic research projects have periodically suggested improvements to the mainstream fine–grained four–input LUT –based FPGA architectures, but so far the commercial market has been dominated by traditional solutions.

The goal of Triptych FPGA Architecture [38] and its associated mapping tools [83] was to reduce the cost paid for routing in standard FPGAs. This was accomplished by combining routing and logic in a way that allows a tradeoff between resources on a per–mapping basis. Consequently, the basic building block in the Triptych architecture was called Routing & Logic Block (RLB), which was a capable of performing both routing and logic tasks simultaneously.

All array resources are dedicated to a single function for an entire operation in conventional reprogrammable logic architectures, whereas Dynamically Programmable Gate Arrays (DPGAs) were designed to be multiple context devices [71]. The goal was to increase device utilization by allocating space on chip to store several configurations for each gate or switch. This enabled context switches with minimal overhead. A prototype DPGA was implemented in 1995 in a 3–layer metal, 1.0μm CMOS process. The prototype used four–input LUTs for the basic logic blocks and supported 4 on–chip context memories, with each context fully specifying both the interconnect and LUT functions [251].

The goal of the RaPiD research project was to define a general coarse–grained reconfigurable architecture, consisting of a linear array of functional units which can be configured to form a linear computational pipeline [82]. Another coarse–grained architectural solution resulted in the design of the PipeRench architecture, which was based on pipeline reconfiguration, where individual physical pipeline stages called stripes could be dynamically reconfigured in a single clock period [111].

The quest for FPGAs with faster performance includes research into SiGe heterojunction bipolar transistor (HBT) FPGAs, whose prototype implementations should run in the 1–20 GHz range [105]. This would make FPGAs suitable for even the most demanding high–speed digital signal processing tasks, but the immaturity of the manufacturing process and large power consumption are disadvantages of the SiGe technology. There have also been research efforts to remove the clocking restrictions by designing fully asynchronous FPGAs [133] [293], but the absence of a clock would require a radical re–education of FPGA designers, and therefore the future commercial prospects of asynchronous FPGAs do not look very promising.

3.2 Reprogrammability

The reprogrammability of FPGAs sets them apart from other pieces of hardware, as the functionality of hardware can be changed on–the–fly. Reprogrammability naturally has also disadvantages, as a large portion of on–chip resources have to be dedicated to perform this function. There does not seem to be a general rule for quantifying the costs and benefits of reprogrammability, and more often than not, they have to be analyzed in a case–by–case manner. The purpose of the following subsections is to describe the reprogramming technologies, partial reprogramming, self–reconfiguration and design systems. As mentioned in Section 2.1.1, the terms reprogrammability and reconfigurability are used interchangeably in this thesis to conform with the terminology in the original publications.

3.2.1 Reprogramming Technologies

The reprogrammability of most modern FPGAs is based on SRAM (Static Random Access Memory) technology, meaning that the configuration points in the FPGA are connected to SRAM bits, whose (re)programming also (re)configures the FPGA [62]. The contents of the SRAM–based programming memory determine the interconnections between computational elements and the functions which the computational elements (typically four–input LUTs) perform. Usually, the SRAM stores only one configuration, but if it is capable of storing more, for example four [224], contexts could be switched on a clock–cycle basis. The contents of SRAM are volatile, which means that the FPGA has to be programmed (or "booted"), usually from an external non–volatile memory, every time the power is turned on.

If nonvolatility of an FPGA is desirable, and there is no need to reprogram the device ever, antifuse–based FPGAs are a good alternative. Antifuses are one–time programmable, which create a connection between two points when "blown", while when unblown remain an open circuit [132].

There has been interesting research into optically reprogrammable FPGAs, which would enable ultra fast dynamic reprogrammability and high density of usable gates. The configuration of logic elements would be stored in an on–chip programming memory, which would be dynamically reprogrammed using optical interconnections. A fully optical FPGA would be the ultimate goal, but so far, these ambitious plans have remained on the drawing tables [269].

Most modern device families support only device–wide reprogrammability, which means that the contents of the entire chip are reprogrammed at the same time, meaning that the device is not functional during reprogramming. Partial reprogramming (also called dynamic and run–time reprogramming) has been commercially supported by the discontinued CLAy (Configurable Logic Array) platform of National Semiconductor and similarly discontinued XC6200 device family of Xilinx. Currently, Xilinx' Virtex families support partial reprogrammability. This important concept is further studied in the next subsection.

The perspective of reconfigurable hardware going beyond the traditional electronic systems is investigated in [194], where it is noted that current electronic systems are at a definite disadvantage in terms of *plasticity*, true hardware reconfiguration and especially reconfiguration and evolution of the hardware construction system itself. Microfluidic systems are described in detail and it is concluded, that they provide a potential bridge between biomolecular reactions and reconfigurable electronic hardware.

3.2.2 Partial Reprogrammability

Partial reprogrammability is a theoretically important concept, but the regrettable lack of practical applications and development tools has so far prevented it from maturing into a mainstream computing model. However, due to its theoretical significance and a potential for future breakthrough, partial reprogrammability is reviewed at considerable depth in this Section.

As defined in Section 2.1.1, partial reprogrammability means that a device can be partially reprogrammed while active [178]. This means that the embedded configuration storage circuitry can be updated selectively, without disturbing the operation of the remaining logic. The term *Logic Caching* is introduced to imply the memory hierarchy of active and inactive tasks. This is presented in more detail in Figure 3.5. A four–class categorization of reconfigurable architectures is presented in [118]. General purpose machines are all considered reconfigurable, but differences arise in the way in which reconfiguration is managed. Based on this, reconfigurable machines are grouped into four classes based on the size of reconfigurable units and the presence (or absence) of local memory.

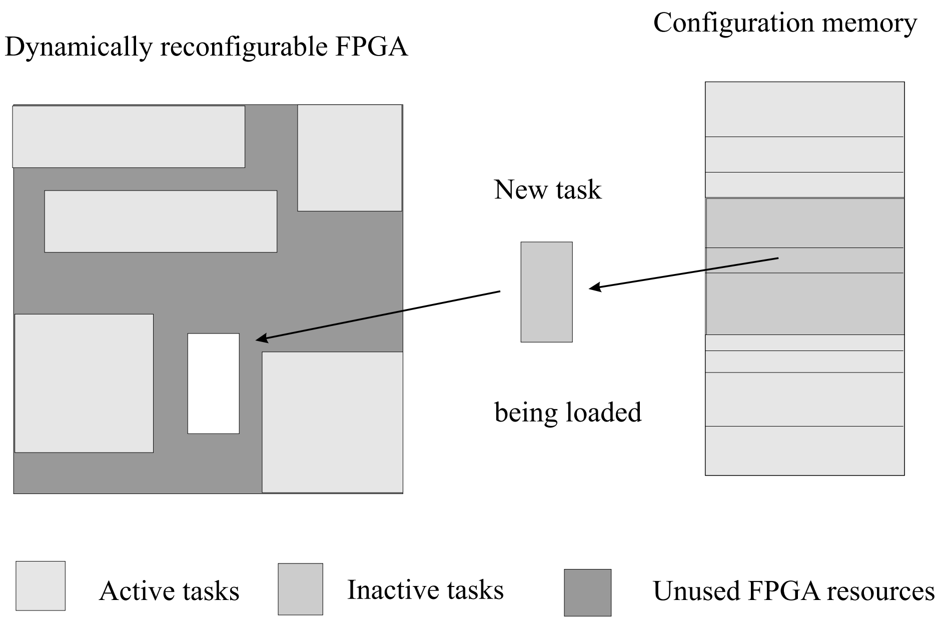

Figure 3.5: Logic caching concept in dynamic reconfiguration of FPGAs [178].

The terms compile–time reconfiguration (CTR) and run–time reconfiguration (RTR) were described in Section 2.1.1. RTR systems can be further divided into global and local classes, with the local RTR applications taking advantage of partial reprogrammability, as they may program any percentage of the reprogrammable resources at any time. A functional density metric is introduced in [289] to balance the advantages of RTR against its associated reconfiguration costs. The goal of RTR is to free up reprogrammable circuitry that would be idle in non–RTR designs, while at the same time making sure that reconfiguration time should not dominate execution time. A Run–Time Reconfiguration Artificial Neural Network (RRANN) is presented as an application that benefits from RTR [289] [86].

The generic identification of circuits which would benefit from partial reprogrammability remains an open research issue. Attempts have been made to formally establish a class of circuits whose performance can be improved by partial reprogramming. In [179], this class of circuits is referred to as programmable, multi–function cores (PMCs), whose control registers are programmed at run–time to select a particular circuit as required by the application. Results of a UART (Universal Asynchronous Receiver Transmitter) case study show area reductions of 21% and speed increase of 14%. Circuits that fix their parameters at run–time might well benefit from partial reprogrammability, as constant coefficient arithmetic units would be used instead of general–purpose arithmetic units. This would lead to substantial savings in area resources [119]. Run–time parameterized circuits for the Xilinx XC6200 device family were presented in [214]. Worst–case delay analysis is a difficult problem in run–time parameterized designs, as a bound on the circuit delays must be determined during design time.

Basic Research on Partial Reprogrammability

Effective utilization of partial reprogrammability requires partitioning the design in time and scheduling the partitioned design into a time–multiplexed reprogrammable logic device. In [262], the architectural framework of a scheduler with a micro register to hold computation results intact between configurations was presented. The scheduling problem in MorphoSys, an integrated coarse–grained multicontext reconfigurable system, has been discussed in detail in [180], where system operation control has been delegated to a tiny RISC (Reduced Instruction Set Computer) processor.

Hardware sequencing, i.e. the managing of the partial reprogramming of the device, has been mostly accomplished with a processor based architecture [288], and an algorithm for automatically adding prefetch operations into reprogrammable applications has been presented [131]. Effective sequencing was also the goal in designing the Virtual Hardware Handler (VHH) [139], where low reconfiguration time has been regarded as a key design parameter. Other research into run–time management of different configurations includes RAGE (Reconfigurable Architecture Group) [44], and structuring the reconfiguration manager into three parts: a monitor, a loader, and a configuration store [235].

Well–known concepts from software engineering, such as data flow computing [35] and pipelining [225] have also influenced research into partial reprogrammability. Data–flow concepts are evident in wormhole run–time reprogramming [34], where the control mechanism for partial reprogrammability is inherently distributed. Multiple computational streams can simultaneously reprogram parts of the device in a data–driven stream processing. The PipeRench architecture has been developed based on the notion, that if reprogrammable systems become practical, they will be predominately applied to pipelineable applications [47]. For this reason, PipeRench supports placing different pipeline stages in different absolute locations of the device, and reprogramming individual pipeline stages in one clock cycle.

Self–Reconfiguration

An interesting subtopic with intriguing new design possibilities in partial reprogrammability is self–reconfiguration, which has obvious connections with the (somewhat overhyped) concept of "evolvable hardware". A reprogrammable device is called self–reconfigurable, if it is able to both read and write the configuration memory [237], which is also called metacomputation. Self–modification of configuration bits has its counterpart in software technology, where a piece of software having access to both its program and data segments has the potential for self–reference, but this is both rare and taboo in disciplined software technology [80]. The on–chip management of dynamically varying circuit modules with the reprogrammable device itself overseeing a static collection of tasks was presented in [41]. When coupled with an operating system, such as Java–based ReConfigME [283], totally self–sufficient reprogrammable devices may be feasible.

A case study in self–reconfiguration was performed in [196], where the reconfiguration control logic and target application executed in parallel on the same FPGA, with the data required for reconfiguration generated on demand. The application in question was a pattern matching with task–dependent data folding on a commercial Xilinx XC6216 device. A Self–Reconfigurable Gate Array (SGRA) capable of context switching (i.e. self–reconfiguration) in a single clock cycle was designed in [238]. The synthesized SGRA was able to store eight configurations.

Design Objectives and Tools in Partial Reprogrammability

The most important objective in designing systems with partial reprogrammability is to maximize the amount of static circuitry that remains unchanged when changing from one configuration to another. This is accomplished by partitioning the application into functional blocks that are, for the most part, common to all configurations [122]. This is the first step in developing a partially reprogrammable system, the second step is to physically map the configurations onto the device.

Verification and visualization of partially reprogrammable systems is orders of magnitude more challenging than in conventional designs, where a one–to–one mapping exists between circuits and device resources. Partially reprogrammable systems map many circuits to shared device resources, each of which can be decomposed into sequences of temporal, one–to–one mappings. This verification problem has been attempted to solve in a proposed CAD framework called Dynamic Circuit Switching (DCS) [222]. Visualizing configuration partitioning, reprogramming overhead effects, spatial overlaps between blocks in different configurations and design execution and configuration schedule has been attempted in the DYNASTY Framework [268].

Design tools for partially reprogrammable systems have been mostly based on the Java$^{\text{TM}}$ programming language, and not on traditional HDLs (Hardware Description Language), such as VHDL or Verilog, for example. First attempts at design tools for partially reprogrammable systems included writing a supervisory program that controlled the platform. The goal of JHDL (Just another Hardware Description Language) [25] was to integrate supervisory control with circuit description, and to allow designers to express dynamically changing circuit organizations using only standard programming abstractions found in object–oriented languages. Starting with JBits$^{\text{TM}}$ [117], Xilinx became the pioneer among reprogrammable logic manufacturers to support partial reprogrammability at the tool level. JBits$^{\text{TM}}$ is a set of Java$^{\text{TM}}$ classes which provide an Application Program Interface (API) into the Xilinx FPGA bitstream. The interface allows all configurable resources to be individually set under software control, since the programming model used by JBits$^{\text{TM}}$ is a two dimensional array of Configurable Logic Blocks (CLBs), the basic reprogrammable unit in Xilinx devices.

Based on JBits$^{\text{TM}}$, RTP (RunTime Parameterizable) Cores were developed [119], which allow cores to be created at runtime and to be used to dynamically modify existing circuitry. Simple and direct support for partial reprogramming was not supported in JBits$^{\text{TM}}$, but it was introduced in JRTR [199], which has been used in frame–based reprogramming of Virtex devices. JBitsDiff [154] is also a Jbits$^{\text{TM}}$ based tool for extracting circuit information directly from the configuration bitstreams to produce pre–routed and pre–placed cores for run–time use. PARBIT (PARtial BITfile Transformer) [143] is a tool to transform configuration bitstreams into partial bitstreams, which is accomplished by reading the configuration frames from the original bitstream and copying only the configuration bits related to the user–defined area to the partial bitstream.

3.3 FPGA–based Computing Platforms

After the introduction of reprogrammable devices into the semiconductor market, the pent–up demand for computation acceleration platforms could be satisfied. The AnyBoard [267] project at the North Carolina State University is a representative example of the capabilities offered by the first entrants into the reprogrammable rapid–prototyping market, as the initial version was completed in 1990. The heart of AnyBoard consisted of an array of five Xilinx FPGAs, with a total usable gate count of approximately 25000 gates, large by the standards of that time. The AnyBoard hardware had both local and global buses, RAM memory, downloading interface and PC interface with the ISA (Industry Standard Architecture) bus. Applications were developed with dedicated custom software, and AnyBoard was primarily used as a coprocessor to a PC in neural networks, image compressors, motor controllers, systolic convolvers and microsequencers.

More ambitious and longer–lasting FPGA–based computing platforms were the Splash project [43] at Supercomputing Research Center in Bowie, Maryland, and the PAM (Programmable Active Memory) project [278] at Digital Equipment Corporation's Paris Research Laboratory (See also Section 2.1).

The Splash project was initially motivated by a need to accelerate systolic algorithms for DNA string matching. The first Splash version was released to the Supercomputing Research Center user community in June 1989, and it consisted of two boards [107]. The first board contained a 32–stage linear array of Xilinx FPGAs and associated memories, and the second board contained a dual–port memory card. The boards resided in two VME (Versabus Modified for Eurocard) slots of a Sun workstation. Programming Splash was very tedious, as applications had to be described at the gate level and manual partitioning was required. Compared to supercomputers of that time , the computing acceleration achieved with Splash was impressive, as speedup factors of over 300 were reported. Architectural limitations of the first Splash version, most notably small I/O bandwidth and lack of interprocessor communication motivated the design of Splash 2 [13], which had a fully programmable 16x16 crossbar among the Xilinx FPGAs. The programming environment for Splash 2 was based on VHDL [12], which facilitated the porting of applications, such as keyword/dictionary searching, DNA pattern matching, and custom image processing [15]. The impressive speedups achieved with both Splash versions were undoubtedly a strong motivation for the entire reprogrammable logic community, as they demonstrated that custom hardware can outperform high–end supercomputers in dedicated computing tasks.

The PAM (Programmable Active Memory) concept was introduced in 1989 [28], when the Perle–0 prototype board was advertised as a software silicon foundry for a 50K gate array with a 50 milliseconds turn–around time. Over the following six years, four generations of PAM hardware and four generations of PAM programming environments were designed and implemented [29]. The concept of PAM meant in general terms a reprogrammable hardware coprocessor tightly coupled with a host workstation. PAMs took different sizes and forms: as an example, the third generation board, DECPeRLe–1, was composed of an array of 24 Xilinx 3090 FPGAs, 4 MB SRAM, and a fast FIFO–based interface to the I/O bus of a workstation. A customized version of C++ was later used in programming the applications onto PAMs. The PAM project achieved remarkable speedups by a factor of 10 to 1000 over conventional processors in computer arithmetic, cryptography, error correction, image analysis, stereo vision, high–energy physics, thermodynamics, biology and astronomy [278]. Later, the PCI Pamette [232] grew out of the PAM Project, as the popular PCI (Peripheral Component Interconnect) bus was selected to connect the computing platform with its host processor. The programming of Pamette was also performed in C++. with a special class library allowing netlist descriptions to be embedded in user–written C++ code [195].

The one million gate limit in FPGA–based computing platforms was achieved in 1995 by the Teramac configurable hardware system [8] built at the Hewlett–Packard Laboratories at Palo Alto, California. A fully configured Teramac included half gigabytes of RAM, and it consisted of 1728 custom FPGAs implemented in MCMs (Multichip Modules). An entire software tool chain with compiler, netlist filter, global and local partitioner, placer, and router was also designed. The Teramac system tolerated defective resources introduced during manufacturing of its FPGAs, as automatic mechanisms to precisely locate the defects were also developed [66]. Another FPGA–based computing platform that surpassed the one million gate barrier was the Transmogrifier–2 designed at the Department of Electrical and Computer Engineering, University of Toronto [173]. The Transmogrifier–2 was intended as a flexible rapid–prototyping system with relatively high clock rates of approximately 10 MHz, and the hardware resources consisted of 16 boards each with two Altera 10K50 FPGAs. Of the most recent large–scale FPGA–based systems, the Berkeley Emulation Engine (BEE) [54] is representative with its estimated capacity of 10 million gates and attainable system clock frequency of over 60 MHz.

The commercial manufacturers of present FPGA–based computing platforms include the leading programmable logic manufacturers, which all sell development and prototyping kits [7] [295], as well as third–party companies, such as Annapolis Micro Systems [11]. Interesting recent academic research projects into FPGA–based computing platforms include an Internet–connected FPGA–based platform [92], where the required TCP/IP (Transmission Control Protocol, Internet Protocol) protocol stack was implemented directly in VHDL on a Xilinx Virtex FPGA, a stand–alone portable platform iPACE–V1 [159] for real–time applications with a support for partial reprogramming, and Pilchard [171], which interfaces directly into the DIMM (Dual In–line Memory Module) socket in standard PCs.

3.4 Software Tools

A smooth, easy–to–use, and reliable design tool set is necessary for the acceptance of reprogrammable logic in the electronic design community. To accomplish this, major reprogrammable logic manufacturers have invested a lot into their design tools[3], and due to their high quality, most designers do not have to learn the intricacies of converting a design from a high–level description onto silicon.

A typical design process for reprogrammable logic starts from a high–level design description, and includes logic synthesis, technology mapping, placement, and routing [55]. Most often logic synthesis and technology mapping are performed with a third party design software, such as Synplify from Synplicity [249] or FPGA Advantage from Mentor Graphics [96], whereas placement and routing require a backend tool from the FPGA manufacturer, for example Quartus II by Altera [219] or ISE Foundation by Xilinx [150].

3.4.1 Logic Synthesis, Technology Mapping, Placement and Routing

The research in design tools has concentrated on SRAM–based FPGAs with four–input LUTs as their basic architectural element, as this architecture has been most widespread among major reprogrammable logic manufacturers.

Combinational logic synthesis optimizes the logic gate networks by first performing logic optimization, which transforms the gate–level network into another equivalent gate–level network which is more suitable for the subsequent step, and then performing technology mapping, which transforms the gate–level network into a network of logic elements (or cells) in the target technology by covering the network with the cells [64]. First FPGA–based logic synthesis algorithms performed a variety of network transformations and optimizations that were geared specifically towards standard cell architectures, and knowledge of FPGA architectures was not used in the optimization process. Recent research has concen-

[3]According to Wim Roelandts (the CEO of Xilinx), there are more software engineers working at Xilinx than there are hardware engineers [120].

trated on an FPGA–specific logic synthesis approach, which unites multilevel logic transformation, decomposition, and optimization techniques into a single synthesis framework [272]. After logic synthesis has been performed, the technology mapping phase selects the circuit elements used to implement the optimized circuit. As FPGAs are usually constructed of K–input lookup tables (with K typically equalling 4), a single LUT can implement 2^{2^K} different combinational functions. Therefore, ASIC–based approaches to technology mapping, which focus on using circuit elements from a limited set of simple basic gates, do not work well with FPGAs, and different approaches to technology mapping are required for LUT–based FPGAs [98] [61].

The original approach to logic synthesis and technology mapping of finite state machines (FSMs) and sequential circuits was to use the same algorithms as for combinational circuits, and to map the combinational logic between the output registers of logic elements [212], but technology–specific features of the target architectures are nowadays taken into consideration in the FSM partitioning procedures [93]. After the introduction of internal embedded memory blocks into FPGAs, additional challenges and opportunities appeared for technology mapping. The large capacity of embedded memory blocks enables the implementation of complex functions in one logic level without the routing delays associated with a subnetwork of LUTs. Technology mapping algorithms that identified parts of circuits that could be efficiently mapped to an embedded memory block have been designed, such as EMB_Pack [65] and SMAP [284]. The technology mapping procedure was also extended to dual–port memory blocks [142], as many applications require memories that can be accessed simultaneously by two separate subcircuits.

Placement and routing are the two most important steps in physical design for FPGAs, and they are responsible for a major portion of the overall design time [156]. The placement process fixes the locations of the logic blocks onto the FPGA, and the routing process assigns signals to routing resources to successfully route all signals while achieving a given overall performance [200]. As routing resources in FPGAs are discrete and scarce, a simultaneous solution to both minimizing total delay and routing all signals is challenging. A popular placement and routing tool with freely available source code is the VPR (Versatile Place and Route) [30] CAD suite. The router in VPR is based on the Pathfinder negotiated congestion algorithm [200], and the simulated annealing algorithm is used for placement. Most recent research has concentrated on tightly integrating placement and routing, instead of the traditional view of iterating between the two distinct steps [156].

As compile times are important for production software [147], it has been observed that users may be willing to trade certain mapping quality for a reduction of CAD tool runtimes. In this respect, FPGA design differs from ASIC design, where producing the highest quality design results at the cost of significant runtimes is justified by the long fabrication times and large costs, whereas FPGA design allows reasonable compromises in this respect. The tradeoffs between runtime and design were investigated in [209].

3.4.2 High–Level Design with FPGAs

The vast majority of present–day FPGA designs are written in a Hardware Description Language (HDL), such as VHDL or Verilog. The size and complexity of modern circuits make traditional schematic–based design methods unattractive alternatives [94], and it has been estimated, that 5–20 kilogate designs described with schematics or Boolean equations can require several person–months of labor [242]. When using an HDL for FPGAs, the design must be adapted to the available reprogrammable resources, and this requires a solid understanding of the targeted architecture. For example, one–hot encoding of state machines is beneficial for fine–grained LUT–based architectures, and multiplexers are expensive to implement in FPGAs [116]. Proper use of hierarchy is critical in achieving desired device utilization, as the advantages of hierarchical design style include adding structure to the design, easing debugging, providing a mechanism for sharing the design among members of the design team, and facilitating the creation of reusable design libraries. However, no amount of re–designing the hierarchical structure compensates for a poorly written HDL code. A time–tested recommendation by Xilinx has been to partition a large design into modules in the 3000 to 5000 equivalent gate range [94].

An obvious disadvantage of using an HDL in FPGA design is the steep learning curve associated with HDL–based design and the complexity of the associated software tools [177]. A long–standing interest, although of largely academic nature, has been to design a single programming language that would permit the designers to compile parts of a program into instruction sequences for a conventional processor and other parts into circuits for reprogrammable logic [286]. This would allow the user to program at a high level language without having to deal with low–level hardware details, such as scheduling, pipelining, and architectural details [26]. The challenges in constructing a universal programming language are daunting, as hardware designs are parallel in nature, whereas most designers think in sequential von Neumann patterns [141].

Despite the obvious difficulties, there have been several efforts to incorporate high–level programming languages into FPGA design. Less wide–spread high–level programming languages that have been attempted to be used for FPGA design include occam [211], Ruby [121], Pebble [177], and Prolog [26]. A compiler for a subset of occam with only integers as a basic type is described in [211], where completely synchronous circuits with only simple gates and D–type flip–flops are produced. The compiler for Ruby, a relational language for capturing block diagrams parametrically, produces target code in various formats, including device–specific formats such as XNF (Xilinx Netlist File), or device–independent formats such as VHDL [121]. The compiler for Pebble, an alias for Parameterized Block Language, supports run–time reconfiguration. Other design goals included simple syntax and semantics, and providing support for both word–level and bit–level design [177]. The development environment based on Prolog, a structured logic programming language, takes a high–level algorithm description as its input, and produces an EDIF (Electronic Data Interchange Format) netlist through an intermediate hardware description notation [26].

The most intense research efforts into using high–level programming languages for FPGA design have concentrated on the C programming language, since there are millions of C programmers worldwide who might be quick to adopt the language for reprogrammable hardware design [101]. None of the proposed C–like programming languages supports the entire C syntax, as compromises have had to be made to tailor the C programming language to take into account the limited availability of hardware resources. The compiler tools described in [141] and [190] both produce a synthesizable HDL description from a C–like input. The implementation in [141] has an open source license and its syntax is modelled on both Perl and C, with support for while loops, if–else branches, integer arithmetic, parallel statements and register assignments. The C–to–HDL compiler in [190] generates pipelined circuits for recursive programs, and it is based on dividing all operations into cascades of 8–bit wide operations. The Transmogrifier C [101] and NAPA C [109] are examples of C–like programming environments that have been initially developed for a particular hardware environment. The Transmogrifier C language lacks support for multiplication, division, pointers, arrays, structures and recursion, and its output is an XNF file. The NAPA C language and its associated tools target a hybrid architecture with both a RISC processor and an adjacent FPGA. Programming pragmas are used extensively to specify where data is to reside and where computation is to occur. Computing machines with multiple FPGAs are supported by the system described in [216], where a C–like input is first translated into a dataflow graph representation, and then scheduled and partitioned onto the multiple FPGAs. Compiling for more than one FPGA is also supported by the Streams–C system [108], which has been developed to support stream–oriented calculations. The Streams–C language is implemented as a small set of library functions callable from a C language program. A C–like programming environment that has matured into a commercial product is the Handel–C programming language [126] by Celoxica, but based on the company's repeating funding rounds, the acceptance of Handel–C does not seem to be widespread.

Object–oriented programming languages, such as C++ and Java^{TM}, have also been attempted to be used in high–level design for FPGAs. One of the first C++ compilers for FPGAs is described in [151], where a small subset of C++ with some extensions is used as the input programming language. Other C++–based design systems for FPGAs are described in [152] with support for high–performance bit–serial pipelined systems, and in [203] with support for module library generation. Java^{TM}–based design systems for FPGAs are described in [49] and [291], with both systems supporting synthesizing hardware from compiled Java^{TM} byte codes instead of the original Java^{TM} text. This seems promising, as the byte code model is platform independent, executable, and supported by many available tools that compile for the JVM (Java Virtual Machine) model. An ongoing project is Open SystemC^{TM} [255], whose goal is to provide hardware–oriented constructs within the context of C++ as a class library implemented in standard C++.

As MATLAB is a very popular language within the global academic community, it is not surprising that there have been intense research efforts into synthesizing FPGA–based designs directly from MATLAB code. The MATCH compiler [20] produces synthesizable VHDL, and first experimental results indicated that the performance of MATCH compiler generated hardware were on the average five times slower than manually optimized designs [124]. The initially academic project has led to a commercial startup company named AccelChip [2], which currently holds the intellectual property rights for the MATCH compiler.

Chapter 4

Applicability of Reprogrammable Logic

This chapter presents an overview of the common characteristics of applications, that are suitable for implementation on reprogrammable logic. The emphasis is first placed on the technical features of applications, and issues such as computational characteristics, host coupling and the added value brought by reprogrammability are taken into account. Design problems with reprogrammable logic and the semiconductor market issues are reviewed, because they are at least as instrumental as the technical features of applications in shaping up the future usability of reprogrammable logic.

The observations and predictions in this chapter are partly based on the material presented in Chapters 2 and 3. The individual design cases, presented in Publications P1–P8 and reviewed in Chapter 5, will be used as representative examples of applications, that benefit from an FPGA–based implementation. The observations in Section 4.1 concentrate mostly on the advantages of reprogrammable logic over software–based solutions, as the projects described in Publications P1–P8 were academic prototyping research and feasibility studies, and an ASIC implementation was not a realistuc option. Design culture issues are handled separately in Section 4.2, whereas the economic considerations (especially between reprogrammable logic and ASICs) are treated in Section 4.3. The chapter is concluded in Section 4.4, where a SWOT (Strengths, Weaknesses, Opportunities, Threats) analysis on reprogrammable logic is performed. The SWOT chart is reflects the author's own opinions and experience in design projects described in the appended Publications P1–P8.

4.1 Algorithms on Reprogrammable Logic

4.1.1 Granularity

As mentioned in Section 3.1.1, most commercially available reprogrammable logic devices have fine–grained granularity, since they are based on four–input LUTs with optional registers and fast carry chains. There have also been mostly academic designs with more coarse–grained architectural solutions (See Section 3.1.4), which have been primarily intended for byte–width datapath functions, whereas the commercially available finer–grained reprogrammable logic is most suitable for bit–level manipulations and arithmetic.

Most often the major contributor to hardware performance advantage can be traced to fine–grained parallelism [115], and if an algorithm is mapped spatially on a fine–grained reprogrammable architecture, the computational density advantage [75] of FPGAs over conventional sequential processors may be measured in hundreds. As fine–grained reprogrammable logic architectures provide generic logic functionality, they are suitable for implementing datapath circuits that are based on data widths not implemented on conventional processors [62].

Of the applications presented in the following chapter, genetic algorithms (Publication P1), the forward error correction (FEC) in AAL2 (ATM Adaptation Layer Type 2) prototyping (Publication P2), the cryptographic algorithms IDEA (International Data Encryption Algorithm) and AES (Advanced Encryption Standard) (Publications P6 and P7), and the bit–level mapping performed in the sigmoid function calculation (Publication P8) all utilize the underlying fine–grained architecture in an efficient manner. Unconventional word–widths offered by a direct mapping to reprogrammable hardware are used in the network optimization implementations in Publication P4.

4.1.2 Data Characteristics

The main advantages of reprogrammable logic over traditional computer architectures in computing problems with repetitive computation of small or variable–sized arithmetics are twofold: the ability to tailor the datapath to the problem at hand [232] and the ability to take advantage of bit–level and instruction–level parallelism that is not accessible to general–purpose processors [252]. On the other hand, reprogrammable logic has serious disadvantages, especially in implementing variable–length loops and branch control [62], and it is questionable, whether implementing e.g. recursive subroutines on hardware is worth the effort at all [286].

Beneficial algorithmic and data features, from the viewpoint of implementation on reprogrammable logic, include the following (adapted from [252]):

- Non–conventional, e.g. not powers–of–two, word–width is the basic operand size
- Inherent parallelism, with multiple units operating at the same time
- Operation foldability, where multiple basic operations can be combined into a single specialized operation
- Computations can be pipelined
- Constant propagation can be performed
- Input value reusability

Computations suitable for implementation on reprogrammable logic can be divided into two categories: stream–based functions and custom instructions [252], with the stream–based functions processing a large data input stream and producing a large data output stream, whereas custom instructions take only a few inputs and produce a few outputs, possibly after a substantial computation delay.

Of the applications presented in the following chapter, communications protocol implementations (Publication P2), digital signal processing algorithms (Publication P5), and cryptographic algorithms (Publications P6 and P7) are stream–based functions. On the other hand, genetic algorithms (Publication P1), special arithmetic operations (Publications P3 and P8), and shortest path routing algorithm implementation (Publication P4) are better described as custom instructions.

4.1.3 Host Coupling

A large fraction of reprogrammable logic applications utilize a processing fabric attached to a host processor [37], which may control functions to reconfigure the logic, schedule data I/O, and perform external interfacing. Depending on the closeness of the coupling, the reprogrammable logic platform is called either a coprocessor (tight coupling) or an attached reconfigurable processing unit (loose coupling) [62] (See also Section 2.3.2. The looser the coupling is, the more independent the reconfigurable unit is to perform computations, but a price has to paid

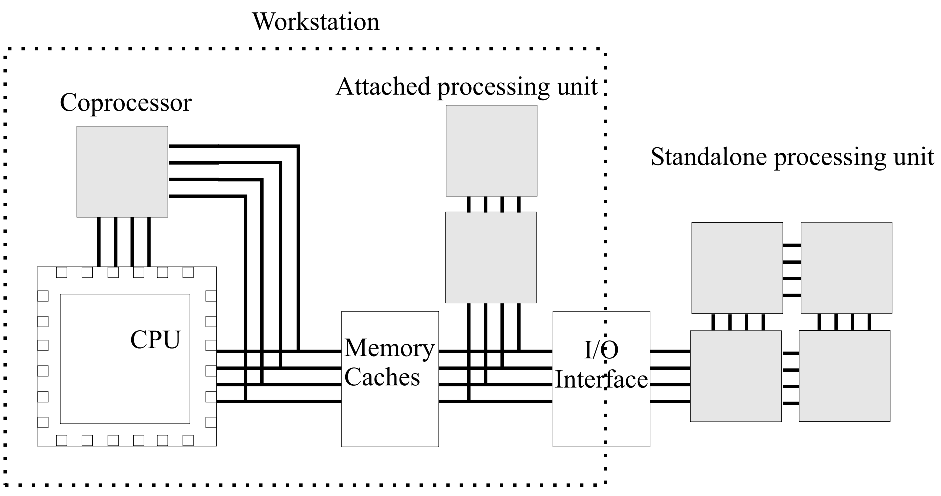

Figure 4.1: Coupling between host and reprogrammable logic (shaded) [62]. Compare also to Figure 2.5.

in the form of a higher delay in communication between the host processor and the reconfigurable hardware. If no host processor is needed, the reprogrammable platform acts as an external stand–alone processing unit, which communicates infrequently (if at all) with other system units. The alternatives for host coupling are summarized in Figure 4.1. The challenge of a system designer is to accurately balance the communications needs versus the computational burden of the algorithm, and thereafter settle on the most suitable host coupling scheme.

Of the applications presented in Publications P1–P8, genetic algorithms (Publication P1) were implemented on a custom–made PCI (Peripheral Component Interconnect) board, which can be regarded as an attached reconfigurable processing unit. The applicability of the special arithmetic operations (Publications P3 and P8) and the shortest path routing algorithm (Publication P4) varies depending on the implementation, but in most cases they are executed on tightly coupled coprocessors, which enables frequent communication of inputs and results between the host and the attached coprocessor. The remaining applications, namely the AAL2 Type 2 receiver (Publication P2), the digital signal processing application (Publication P5), and the stream–based cryptographic applications (Publications P6 and P7) would in most cases be implemented on an external stand–alone processing unit.

4.1.4 Benefits of Reprogrammability

When FPGAs were introduced, the fact that they required (re)programming from an external memory source, and that they lost their configuration once power was removed, was generally considered a liability [33] (See also Section 2.2). However, over the years the additional benefits of short time–to–market and the ability to fix bugs or accommodate late specification changes rose in importance, and nowadays the reprogrammability of FPGAs is regarded as an asset, and not as a liability.

One of the first applications to take advantage of reprogrammability was logic emulation [132], as the designers of custom chips needed to verify that the circuit they were designing actually behaved exactly as desired. A more recent potential application that makes extensive use of (partial) reprogrammability is a multimode handset with integrated services and global roaming ability. A multimode radio requires real–time reconfiguration capability to interface and communicate with heterogeneous networks as the user moves over different geographic regions, and requests different services or if the radio has to adapt to varying channel conditions. Provided that the vast challenges in reconfiguration management and power consumption issues are solved, an FPGA–based configurable computing machine (CCM) platform might show promise for this kind of a software radio handset [245].

Although partial reprogrammability has been extensively researched in the past (See Section 3.2.2), it has never really left the research laboratories, and has remained a largely theoretical concept to this day. The main reasons are a lack of easy–to–use design tools [185] and the failure to establish a commercial presence with resultant business failures [250]. At least for the time being, the main benefits of reprogrammability include ease of design, low upfront costs, and field upgradability (See also Section 4.3). Both design–time and run–time parameterizable designs, e.g. special arithmetic functions for arbitrary bitwidths, benefit from reprogrammability, as the hardware can be adapted to the computation structure at an appropriate design stage [37]. This is also true for designs, whose computation accuracy has to be first determined by simulations.

Reprogrammability has been naturally useful in all the designs described in all Publications P1–P8, but this is especially true for the square root algorithm (Publication P3) and the GP (General Parameter) adaptation (Publication P5). The required bit width of these designs can be decided on–the–fly based on computation accuracy requirements and/or simulation results. Publication P2 also includes discussion on the benefits of hardware–based versus software–based simulation, and the design of the AAL Type 2 described in Publication P2 was partly motivated by its potential use in a logic emulation environment.

4.2 Design Challenges and Opportunities

To gain larger market share, reprogrammable logic should make advances both into the microprocessor–dominated market segment, as well as into the ASIC–dominated high–performance mass–market and low–power segment. The main obstacle for a more widespread acceptance of reprogrammable logic as a new computing paradigm is the inherently sequential design culture of programming, which thoroughly dominates mind set in computer science education. The microprocessor will remain an indispensable tool for implementing digital systems, but it should not have the mental monopoly in programming education. As of today, ordinary programmers have severe problems in mapping an application or algorithm onto hardware platforms, other than relying on microprocessor–based state–machine mechanisms. [22]

A remedy to reduce the emerging productivity gap in embedded system design is to include hardware design languages, like VHDL and Verilog, into mainstream computer science curricula. This would enable ordinary graduating engineers and programmers to have the skills needed for hardware/software partitioning decisions, and to acquire the algorithmic cleverness needed to migrate an application from software onto reprogrammable platforms. A more thorough and long–term solution would be to implant reprogrammable thinking deep into the intellectual infrastructures of computer science education. [130]

Reprogrammable logic has been regarded more as "hardware" than "software" in the electronic design community. This suggestive terminology has not helped to fully understand the potential of programmable logic, and a better distinction than hardware/software is to think in terms of control flow vs. data flow approaches when designing and implementing algorithms. Control flow is exemplified by sequential programs, where instructions manipulate data. This includes microprocessor instruction sequences, and at a higher level, programs coded in imperative and functional languages. Data flow, on the other, is exemplified by interconnected processing elements with data moving between elements, not under some central programmed control. This includes logic circuitry, and at a higher level, interconnected computation units. Another way of comparing these two approaches is to call control flow as "computing in time" and data flow as "computing in space". Traditionally, control flow has been associated with software, and data flow with hardware. The main impact of reprogrammable logic should be that a data flow approach can be made available to the software designer, as an alternative form of programming. This would enable the emergence of a new parallel computing paradigm based on reprogrammable logic and a softening of barriers between thinking either in software or in hardware. [40]

There has been much interest into High–Level Synthesis (HLS) design environments (See Section 3.4.2), but their immaturity and lackluster commercial reception makes it probable, that the majority of FPGA designs will be based on either VHDL or Verilog for the foreseeable future. However, there is a potential for an integrated hardware/software co–design environment based on a single high–level programming language [286], but practical and design cultural problems have so far been insurmountable for a widespread commercial acceptance.

Several appended publications include discussion on the topics mentioned in this subsection, such as co–simulation and co–development environments, and high–level design. Publication P5 describes custom–written conversion functions and batch files in the C programming language and `awk`, which facilitated both importing initial simulation values from MATLAB into Altera's Max+Plus II design software and exporting results from Max+Plus II back into MATLAB for graphical inspection. A corresponding co–development environment is also described in Publication P8, where a MATLAB–based design environment for automatically generating synthesizable VHDL code from the MATLAB output file via the McBoole Logic Minimizer [67] is described. Publication P6 describes the initial design effort to write the HDL code in Handel–C [126], a high–level C–like design language (See Section 3.4.2). Because design performance deteriorated, clock speeds decelerated, and compilation times outgrew practical bounds, the design described in Publication P8 was decided to be entirely rewritten in more resilient VHDL.

4.3 Economic Considerations

In the real world, component and design choices are never made on the technical merits alone, and economic factors are decisive in product design. Putting all the relevant economic variables into a comprehensive equation is challenging, although it has been tried in Altera's FPGA total cost calculator [97]. The relevant economic parameters include, among others, the average annual salary of an engineer, company's gross margin, software tools cost, approximate original selling price of final product, typical end product price deterioration per quarter, forecast final product unit sales in introductory quarter, how much faster the market were accessed if the design were implemented on an FPGA, and how many quarters are required to get to full production.

The preceding listing suggests, that choosing between an ASIC and an FPGA on economic merits is more complicated than a simple glance at the price–per–unit metric would suggest. This was also investigated in [18], where three cases (pure ASIC, pure digital signal processor, pure FPGA) were compared with Monte Carlo statistical analysis for designing a 3G base station, whose ASIC–based design required approximately $100 million development cost and a 36–month design gestation. The final conclusion was something of a non–result, mainly that even after consolidating all the relevant different items (unit costs, power budgets, masking and tooling costs, development costs, amortized time costs, project flexibility etc.), it proved extremely difficult and very situation–specific to tell which approach was the best. Oftentimes economic reviews also overlook two advantageous characteristics of reprogrammable logic, namely shorter time–to–market and field upgradability [158]. Both of these may contribute substantially to the cumulative sales of the end product (See also Figure 4.2).

The debate between reprogrammable logic advocates and ASIC advocates resembles the older debate between assembly–language advocates and high–level–language advocates in software engineering. The debate was "won" on technical merits by the assembly–language advocates by proving better performance and better size. But the war was won by the high–level programming advocates, because the significance of the shift to high–level languages and design concepts and better productivity was missed by the assembly–language advocates. The analogy to the present debate between reprogrammable logic advocates and ASIC advocates is obvious: focusing only on chip size, power consumption, and circuit speed misses the significance of the shift to reprogrammable systems [261]. The main point is no longer about absolute size and speed; it is about what is good enough, and ASICs fail in this respect because they are inflexible and too expensive. The CEO of Synopsys was quoted in September 2002 as saying that "The cost of designing a complex SoC in copper on 0.13 μm is $10.9 million, and we are reaching the point where testing the chip may be more expensive than fabricating it" [186]. It is no coincidence, that the soaring ASIC design costs have started to limit the number of companies that can afford an ASIC–based imple-

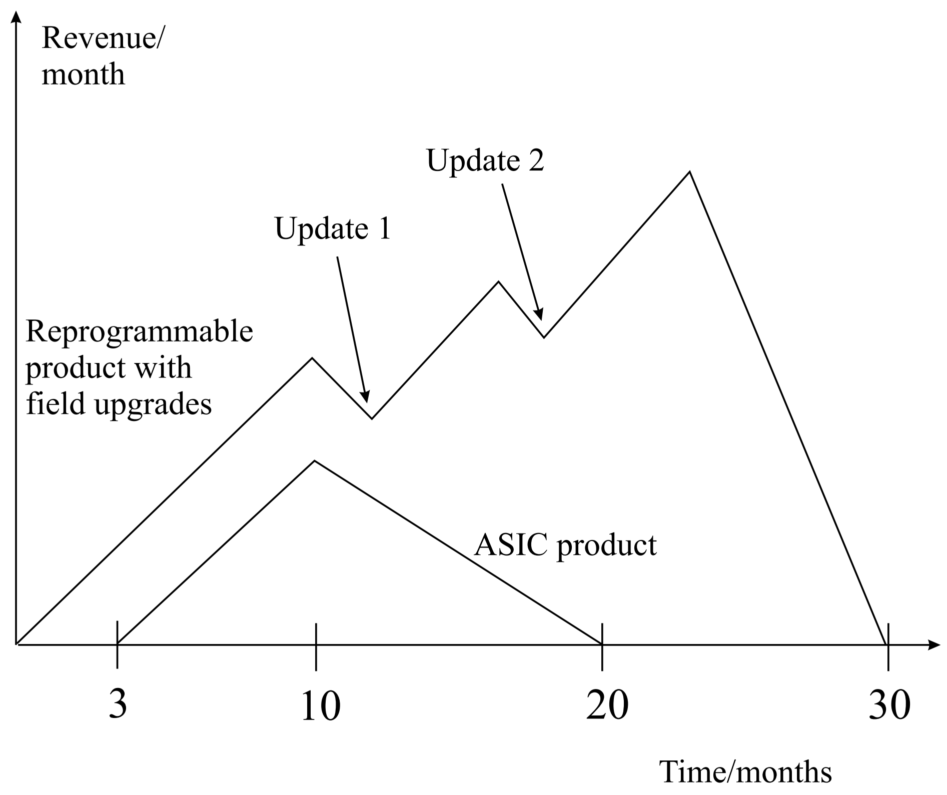

Figure 4.2: Economic benefits of faster time–to–market and field upgradability. Adapted from [158].

mentation for their end product [45]. It has been reported, that standard cell ASIC designs have dropped from 5200 in 1997 to 1400 in 2003, mostly due to rising mask fabrication costs [220].

The role of microprocessors may also change in a design environment dominated by reprogrammable logic: the microprocessors will assume the role of supervisors, and they will manage systems tasks, much the same way as car drivers manage their automobiles' tasks but do not do the work of propelling all that mass [260]. The biggest drawback to achieving power–conscious design with modern reprogrammable logic is their power–hungry SRAM configuration memory, which should be replaced with more efficient nonvolatile memory, such as magnetoresistive memory, ferroelectric memory, and ovonic unified memory [103].

4.3.1 Suitable Comparison Metrics

The most serious proposal for adopting industry–wide impartial comparison benchmarks for reprogrammable logic devices was put forward by the Programmable Electronics Performance Corporation in 1993 [193]. The PREP Benchmarks included a set of ten benchmark circuits and an implementation methodology for measuring both logic capacity and speed performance in programmable logic devices. The PREP benchmarks were developed to evaluate different architectures and devices using common logic functions and a standard methodology. Nowadays, the PREP benchmarks are a bit outdated, but a new substitutive set of benchmarks designs has not been universally adopted, and there is a tendency among reprogrammable logic manufacturers either tend obfuscate or to oversimplify the quantitative characteristics of their products, whichever seems more advantageous for their purposes.

If comparing FPGAs to other FPGAs is challenging, it is even more difficult to agree on an universally acceptable mechanism for comparing FPGAs to either ASICs or microprocessors (See also Section 2.3). However, certain general conclusions can be drawn from recent research in this area. For example, it has been found, that the peak FPGA floating–point performance is growing significantly faster than peak floating–point performance for microprocessors [265]. The main reasons for this inequality in performance improvement are the high degree of hardware configurability in FPGAs, and the dataflow nature of computation on FPGAs. In quantitative terms in floating–point performance, the microprocessors are following a well known corollary of Moore's law [1] (doubling in performance every 18 months), whereas FPGA performance is increasing by a factor of four every two years. When FPGAs are compared to ASICs, it has been proposed that the flexibility of reprogrammable hardware should be taken into account as yet another metric besides the more traditional metrics, such as area, performance, and power [63]. If the flexibility of a design is defined as the ability of a design to implement particular types of circuits from a given application domain, the reprogrammability of FPGAs comes into its own when comparing them to ASICs with a fixed architecture.

It has also been argued that traditional cost functions, such as area, power, clock speed, and design efficiency, do not adequately represent the performance as conceived by the end–user. Features as dependability, scalability, product–level inter–generation compatibility and effective lifetime, should also be taken into account. It has been proposed, that reprogrammable IC architectures do well if these "softer" performance characteristics are also taken into account. [266]

Suitable comparison metrics are also discussed in Publication P8, where the usage of *gate* as a measurement of area efficiency in FPGAs is criticized, and the more suitable Logic Element (LE) [2] is proposed as a better metric for area usage in FPGAs.

4.4 SWOT Analysis on Reprogrammable Logic

A SWOT analysis is a tool used to evaluate the strengths, weaknesses, opportunities, and threats involved in a product [3]. Strengths and weaknesses are internal characteristics of the product, whereas opportunities and threats originate from outside the product. Based on the discussion in this and preceding chapters, a SWOT matrix for reprogrammable logic is proposed in Figure 4.3.

[1]The observation made in 1965 by Gordon Moore [208], co–founder of Intel, that the number of transistors per square inch on integrated circuits had doubled every year since the integrated circuit was invented. Moore's law has held surprisingly well for almost four decades.

[2]LE is Altera's terminology for the basic building block, that is a four–input LUT and a presettable register, but its usage is justified, as the underlying structure of most modern FPGAs is sufficiently similar.

[3]Originally, SWOT analysis was used for projects or business ventures, but over the years its usage has extended to organizations and products. Although SWOT analysis has been criticized for oversimplifying complex issues, its advantage is obvious: by explicitly pinpointing weaknesses and threats, it may help in turning them into opportunities, and ultimately into strengths.

Internal	Strengths: - Reprogrammability - Fast turn-around time - Field upgradability	Weaknesses: - Power consumption - High unit costs for mass-market products
External	Opportunities: - High-level design - Exploiting the ASIC "design gap"	Threats: - Design culture - Overhyped expectations

Figure 4.3: SWOT matrix for reprogrammable logic.

The strength of reprogrammable logic is self–evident from the name itself: *Reprogrammability*, which enables fast turn–around times and consequently shorter time–to–market than with ASICs. Reprogrammability enables also remote field upgrades to products, which would otherwise require costly manual maintenance work. As mentioned in Section 4.1.4, partial reprogrammability has had a rough start in the commercial world, and the more conventional occasional device–wide reconfiguring still remains the dominant motif for choosing reprogrammable logic as an implementation platform for digital designs.

The weakness in modern reprogrammable logic is twofold: power consumption and high unit costs for mass–market applications. The relatively high power consumption of FPGAs (See Section 3.1.3) has thus far precluded their usage in portable battery–powered environments, but this may be slowly changing with the introduction of less power–hungry reprogrammable devices [260]. The other weakness, higher unit costs, becomes obvious by glancing at Figure 3.2, where it is evident that reprogrammable dedicates an enormous amount of high–prices silicon real estate for non–logic functions [258]. However, the break–even point in production quantities between ASICs and FPGAs is moving upwards, caused both by rising ASIC design and production startup costs (See Section 4.3), and by the diminishing product life cycle lengths, which emphasize shorter time–to–market [18]. However, the real bonanza of mobile phones and related products, seems to remain outside the reach of reprogrammable logic for the time being due to their huge production quantities and stringent power consumption requirements, despite the tantalizing appeal of multimodal reconfigurable mobile handsets (See also Section 4.1.4).

The main opportunities for a reprogrammable logic breakthrough are the emergence of "push–button" high level design systems, and the growing design gap in ASICs. Although there has been a lot of research into high–level design languages and systems (See Section 3.4.2), commercially viable solutions have been practically non–existent. However, if a breakthrough in high–level design occurs, the attractiveness of reprogrammable logic will certainly increase in the eyes of more software–oriented designers. The other main opportunity for reprogrammable logic is a corresponding threat to ASICs, namely the growing design gap, which means that while the complexity of the designs is increasing from year–to–year, the productivity of the integrated–circuit designer is not keeping pace [221]. Besides the afore–mentioned design gap, other design advantages of reprogrammable logic over ASICs include speed (especially the place and route of large–scale ASIC designs can be very time–consuming), ease (there is no need for post–

layout back–annotation, as in ASICs), the need to design extensive test vectors for ASICS, and the large price gap between ASIC and FPGA design software (See Section 2.3.3. These reasonings make ASICs the design choice of ever fewer and bigger companies, which can invest the required amounts of workforce and money for high–end ASIC design, thus leaving the small–to–medium sized design vulnerable for reprogrammable logic –based alternatives.

Two external threats to reprogrammable logic can be identified: existing design cultures and overhyped expectations [4]. The existing dichotomy between software and hardware designers is not beneficial for reprogrammable logic, as the present–day computing science curricula present mostly only the traditional von Neumann–type microprocessor as the implementation platform of choice [22]. Softening of barriers between hardware design and software design [40] should be an educational goal, but the general trend in undergraduate education has been a divergence between electrical engineering and computer science [155]. An example of overhyped expectations for reprogrammable logic was the fever for *reconfigurable computing* (See Section 2.2), or the idea of creating chips that can (self–)modify their function dynamically, or on the fly as they are operating [250]. The expectations and projections for reconfigurable computing were sometimes wildly off–the–mark, and business failures [5] were an unfortunate consequence. The adoption of reprogrammable logic as an equivalent alternative to both ASICs and microprocessors is most likely to be an evolutionary, not revolutionary process.

[4]Overhyping, or raising unwarranted expectations which are not later fulfilled, is admittedly a rather general threat for every new product and/or technology, but it can be argued that reprogrammable logic has been more vulnerable to overhyping than most other technical fields.

[5]An example of business failures was the Silicon Valley –based Chameleon Systems, which raised 70 million dollars before going bankrupt in 2002 [250].

Chapter 5

Applications in Selected Fields

Each of the individual sections in this chapter begins with an overview of the application area and its relevance. After that, the results presented in the author's publications P1–P8 are compared to contemporary research efforts elsewhere, and the main contributions based on the discussion and categorization in Chapter 4 are highlighted.

The author's publications are not covered in a strict order by the publication date, as they have been reordered in compliance with a more meaningful application–centered model. Although it is impossible to present an entire spectrum of FPGA–friendly case studies, Publications P1–P8 can be said to represent a fairly extensive survey of the possibilities offered by reprogrammable logic in modern digital design.

5.1 Optimization Methods

Optimization can be loosely defined as the study of maximizing and minimizing functions subject to specified boundary conditions or constraints. Certain optimization problems are computationally intractable, and require heuristic or optimization methods, whereas other problems can be solved by deterministic, i.e. non–heuristic, methods. In the following two subsections, FPGA–based implementations of both a heuristic optimization method (genetic algorithm) and a deterministic optimization method (shortest path algorithm) are presented.

5.1.1 Genetic Algorithms

A genetic algorithm (GA) is an optimization method that is based on natural selection methods found in the Darwinian theory of evolution [110]. Genetic algorithms are regarded as a robust general–purpose optimization technique with real–world applications in adaptive equalization, character and number recognition, digital filter design, image compression, robot control and VLSI design [256].

The basic operations of genetic algorithms are the selection of population members for the next generation, mating these members via a crossover of chromosomes, and performing mutations on the chromosomes to preserve population diversity so as to avoid convergence to local optima. The fitness of each member in the new generation is determined using a fitness function, whose results influence the selection process for the next generation [228]. The success of a genetic algorithm is determined by the right combination of crossover rates, mutation rates, and population size [241]. The basic operations of genetic algorithms display inherent parallelism, atomic bit–level operations, unconventional word lengths, and pipelinability, thus making them ideal candidates for direct implementation on reprogrammable hardware. However, FPGA–based implementations of genetic algorithms have been relatively scarce, probably due to the unfamiliarity with reprogrammable devices in the genetic algorithm research community. In the late nineties, approximately around the time of Publication P1, there were a few other research projects on FPGA–based genetic algorithms in the academic community, and they are briefly reviewed in the following paragraphs.

The Hardware Genetic Algorithm (HGA) [228] was designed to work as a coprocessor with the CPU of a PC, and it implemented the core functions of genetic algorithms on a separate FPGA–based computing platform. The HGA was tested against software–based genetic algorithms running on a Silicon Graphics 4D/440 with four MIPS R3000 CPUs, each running at 33 MHz, and the average speedup factors ranged between 12.75 and 18.74. It was estimated, that for complex problems, the HGA might accelerate computations by three orders of magnitude.

Accelerating the chip partitioning problem, a computationally intensive NP–complete problem with applications also in FPGA design, with genetic algorithms was the topic of research in [241]. The most time–demanding computations in a software–based implementation were required in the fitness function evaluation, which required over 95% of the execution time. The computational kernels of the genetic algorithm for the chip partitioning problem were implemented on a separate Armstrong III multicomputer with both conventional microprocessors and three Xilinx XC4010 FPGAs, and the results indicated an average speedup factor of 8.32 over a pure software–based implementation of the same algorithm on a SUN SPARCStation 20 Model.

Genetic algorithms were employed with the Splash II platform to accelerate the NP–complete family of travelling salesman problems [114], which involved finding the shortest path through a collection of n cities, visiting each city exactly once and returning to the starting city. A candidate tour was coded into a chromosome, with the fitness function indicating the length of the tour. The FPGA–accelerated genetic algorithm outperformed a software implementation executing on a 125 MHz HP PA–RISC workstation by a speedup factor ranging between 6.8 and 10.6. The most important factor for this performance difference was found to be fine–grained parallelism, which enabled operator pipelining [115]. The other contributors for the performance difference included hard–wired control, and flexibility in custom address generation.

Significant speedups were reported in [189], where a computation platform with two Altera's EPF10K100 FPGAs and external SRAM memory was used to implement genetic algorithms for the knapsack problem and the graph partitioning problem. The FPGA–based system achieved a speedup factor ranging between 50 and 130 compared to 200 MHz Ultra–Sparc workstation. Interesting research results were also reported in [163], where Xilinx XC6216 was used to accelerate the fitness measurement task of genetic programming, which is an extension to genetic algorithms with the goal of automatically creating computer programs.

Publication P1 describes a high–level hardware design language (AHDL, Altera Hardware Description Language) –based implementation of a genetic algorithm, whose performance is compared to both a 120 MHz Pentium based Linux system and a HP C110 workstation. The speedup factor over software–based genetic algorithms was roughly 200, and it was estimated that the speedup factor could be increased to over 3200 by further parallelizing the algorithm and utilizing extra instances of the acceleration card. The results achieved in Publication P1 are in line or exceed corresponding research results reviewed in previous paragraphs. Additional contributions of Publication P1 include distributed internal memory utilization, PCI (Peripheral Component Interconnect) bus interfacing, and random noise generation with the help of a noise diode.

5.1.2 Shortest Path Graph Algorithms

Graphs are central data structures, and they have wide applicability in many fields of engineering. Formally, a graph G is defined by (V, E), where V is a finite nonempty set of vertices, and E is a finite nonempty set of edges connecting pairs of distinct vertices from V [282]. Since the terminology of graph theory has not been standardized, vertices are often called nodes and edges are often called arcs, branches or links. Graph algorithms are used, for example, in computing critical placement and routing patterns in circuit and hardware design, in deducing connectivity in networks, in finding shortest paths in telecommunications and Internet routing, in computing the closure of relations in databases, and in managing storage in operating and runtime systems [144]. To achieve acceptable performance in time–critical computations, graph algorithms must be fast and the data structures, which represent graphs, must be effective.

The efficient representation of graph structures in reprogrammable hardware was investigated in [144], where dynamic graph structures were implemented as dynamic graph processors (DGPs), which maintain a collection of gates (representing the vertices) and wires connecting the gates (representing the edges) as a graph circuit. DGPs allow a low–cost and dynamic insertion and deletion of vertices and edges without the need to recompile and refit the circuit structure, and speedups of more than three orders of magnitude were reported in computing a graph's transitive closure compared to an effective software algorithm running on a contemporary fast 500 MHz DEC Alpha processor. The DGP structure was extended in [204] in three ways: streamlining the implementation via tri–state logic to increase density and graph size, scaling the hardware arrays to larger graphs by splitting them into multiple contexts, and embedding the graph accelerator circuits in a module generation environment. The resulting improved graph structure was called HArdware Graph ARray (HAGAR).

The problem of finding the shortest path plays a central role in the design, analysis, and operation of networks. Most routing problems can be solved as shortest path problems once an appropriate cost is assigned to each link in the network. For example, in communications networks the cost of a link could reflect its available bandwidth, delay or bit error ratio. Formally, given a graph $G = (N, E)$ with a positive cost d_{ij} for all edges $(i, j \in N)$, start node S and a set P of permanently labelled nodes, the shortest path problem is defined as finding the shortest path from start node S to every other node in the graph. If the shortest path between the start node S and a given node is sought, the shortest path algorithm may terminate as soon as the shortest path to it has been found.

Shortest path problems are categorized as the single–destination shortest path problem (SDSP) and the all–pairs shortest path problem (APSP). Of the commonly used shortest path algorithms, the Bellman–Ford algorithm and Dijkstra's algorithm are used for the SDSP problem, whereas the Floyd–Warshall algorithm is suitable for the APSP [181].

Dynamic computation structures (DCS) were introduced in [17] to directly map graph instances onto reconfigurable architectures, and a front–end compiler was implemented to parallelize the entire inner loop of the Bellman–Ford shortest path algorithm. The speedups ranged between 10 and 52 for the shortest path problem, compared to execution on a SPARCStation 10 processor. The Bellman–Ford shortest path algorithm was also implemented on reprogrammable hardware in [69], where the research emphasis was on decreasing the compilation time by synthesizing high–level graph designs to a specific domain and adapting to the input graph instance at run–time. In comparison to software implementations, the asymptotic run–time speedup was estimated at 3.75. Automating the design flow was a central theme also in [181], where Xilinx' System Generator was used to generate the VHDL code automatically for the Floyd–Warshall shortest path algorithm, but explicit speedup figures compared to a software–only implementation were not reported.

Publication P4 describes the first published FPGA–based implementation of Dijkstra's shortest path routing algorithm, as extensive literature searches have not found publications on the same topic. The average speedup factor compared to a contemporary microprocessor–based implementation ranged between 23.58 and 67.41, and the speedup factor increased as graph sizes grew. This can be attributed to the parallelism in comparator bank implementation, which along with efficient usage of embedded memory and a discussion on the suitability of reconfigurable computing in network routing are the main contributions of Publication P4.

5.2 Protocol Prototyping and Verification

Designing and operating communications networks is greatly facilitated by a layered protocol stack, of which the International Standards Organization (ISO) Open Systems Interconnect (OSI) Reference Model is the best known. The ISO/OSI Reference Model defines seven layers of communications types (starting from the top: application, presentation, session, transport, network, data link and physical), and the interfaces among them. Each layer depends on the services provided by the layer below it, all the way down to the physical network hardware. The ISO/OSI Reference Model processing is normally performed in software, but with the continuous rise in data rates, hardware–based protocol processors or accelerators are increasingly being deployed [104]. This is also of interest to the reprogrammable logic industry, since the reprogrammability and fast time–to–market of FPGAs makes them an attractive alternative for accelerating network protocol processing.

A dynamically reconfigurable FPGA–based architecture, called the Programmable Protocol Processing Pipeline (P4), was described in [123] to act as a protocol booster. The P4 could be inserted and deleted from network protocol stacks on an as-needed basis, with a dynamical downloading and activation of processing elements, whose architecture consisted of a pool of Altera's FLEX8000 device family FPGAs. The P4 was designed to operate on streams of Asynchronous Transfer Mode (ATM) cells at 155 Mb/s at the edge of the boosted portion of a network cloud. Case study measurements indicated that the P4 protocol booster performed with minimal processing delay in Forward Error Correction (FEC) acceleration. An accelerator for the topmost layer in the ISO/OSI Reference Model, layer seven, was described in [202]. The accelerator was implemented on Xilinx Virtex XCV1000E FPGA target chip, and its key tasks included tree lookup for routing, checksum calculation, packet classification, and pattern matching. The overall execution time was decreased by as much as 20 times, and the reprogramming overhead ranged from 0.2 μs to 3 ms.

Reprogrammable devices have also been extensively used in the design of networking elements and protocols. A case study of decreasing the simulation times for ATM switching fabrics was described in [257], where discrete event simulation was accelerated by mapping the model of a physical ATM switch onto a computing system comprised of a Xilinx XC6216 Reconfigurable Processing Unit (RPU). The performance improvement in simulation times ranged between 200 and 1100. Reprogrammable architectures are widely used in prototyping experiments, for example in designing Internet routers [140]. A framework to process both ATM cells and Internet Protocol (IP) packets directly in hardware has been proposed in [39], where an 8–port ATM switch was equipped with FPGA–based Field Programmable Port Extenders (FPX). Since the lowest layers could be processed directly in hardware, the prototype IP router could operate at an impressive line speed of 2.4 Gb/s. Another example of a modularly reprogrammable networking platform is the Transmutable Telecom System [206], which consisted of five reconfigurable boards and a main microprocessor–based master board. Different networking functions were packed on individual boards to provide flexibility, and the system could be completely reconstructed by using application–dependent board arrangements.

A growing trend in the design of networking equipment is to make them secure against malicious attacks by third parties, and due to their ability to effectively parallelize bit–level operations at the hardware level as opposed to tedious software–oriented approach, reprogrammable logic devices have been used in accelerating security–related functions. One of first examples of this design trend is described in [197], where an FPGA–based Firewall Inline Processor (FIP) is used, based on an access list, to either accept or reject connections in an ATM firewall. Internet Protocol (IP) characterization, or the process of classifying IP packets into categories depending on information in the header, has also been accelerated with FPGAs [79]. The characterization was performed with the Content Addressable Memories (CAM) in a Xilinx XCV1000 FPGA, and depending on the details of the implementation, between 3.4 and 7.1 million searches per second were achieved. Software–based real–time network monitors do not function properly in the gigabits per second transmission speed range, and research efforts have concentrated on using reprogrammable hardware to accelerate network packet filtering. A high–level design methodology to automatically synthesize FPGA circuits from network monitor descriptions is described in [160], and the open–source firewall software Snort [244] has been accelerated with FPGA–based solutions [57] [106], with both implementations performing real–time network intrusion detection at approximately 2 Gb/s. An automated JHDL–based [25] module generator to automatically extract rules from the Snort rule database and to generate corresponding circuits for Xilinx FPGAs is described in [145]. The FPGA–based intrusion detection module exceeded the performance of a software–based system by over 600 times.

5.2.1 Simulation Acceleration of an ATM Adaptation Layer Type 2 Receiver

Publication P2 describes an FPGA–based implementation and simulation of the ATM Adaptation Layer Type 2 (AAL2) [153] receiver, and the features of hardware and software–based simulation approaches are compared. The AAL Type 2 Receiver was written entirely in VHDL, and it was implemented on a special hardware acceleration card with a custom–written reconfiguration utility. The

FPGA–based AAL Type 2 receiver exceeded the performance of a similar AAL Type 2 receiver written in the C programming language by 70 times, and the simulation speed of a MATLAB–based AAL Type 2 receiver model was outperformed by a factor of 55000. The speedups are based on the ratios simulated transmission rates, which are displayed in Table 1 in Publication P2 ($70 \approx 55$ Mbps $\div$ 790 kbps and $55000 = 55$ Mbps $\div$ 1 kbps). The biggest contribution to the speedup was the concurrent implementation of the Header Error Check (HEC) module, where utilizing the reprogrammable logic architecture in an efficient manner was successful. Another contribution of Publication P2 is an analysis on the advantages of hardware–based simulation, and it was concluded, that simulating a communications block directly in hardware both speeds up the simulations and decreases the development time.

As the project described in Publication P2 had direct industrial funding, the design was motivated by its potential use in a logic emulation environment.

5.3 Arithmetic Algorithms

The development of cost–effective arithmetic operators for reprogrammable logic requires a pairing of high–speed algorithms with a solid understanding of the given target technology [176]. Arithmetic algorithms have usually been optimized for microprocessors, and most often they do not map well onto reprogrammable logic without modifications. A special class among arithmetic operations is formed by iterative algorithms, which use only shifts and adds, thus making them an attractive choice by avoiding costly multiplications and divisions. The skills required to design effective special arithmetic operations for reprogrammable logic are in high demand, as it seems that the majority of today’s hardware designs are done by engineers with little or no background in hardware efficient algorithms [9].

5.3.1 Square Root Operator

Square root is a basic arithmetic operation in scientific calculations and computer graphics, among other applications [174]. The square root operation remains a serious design challenge, because of the dependence among the iteration steps and the relatively high complexity of the result–digit generation function, which typically produces one digit of the result per iteration [253]. When arithmetic operations are implemented as a part of complex computation systems on reprogrammable logic, special care must be taken to design as small units as possible, since this enables fitting a maximum amount of special arithmetic on the reprogrammable device.

A digit–recurrence square root implementation for FPGAs is described in [176]. The implemented algorithm computes one result digit per clock cycle, and a cycle time of 22.3 ns was achieved in simulations for Xilinx XC4010 FPGA devices. Another square root implementation on FPGAs is reported in [253], which also computes one result digit per clock cycle, but since the result digits are produced in signed–digit form, an on–the–fly conversion scheme to two's complement form is required. The FPGA area resource requirements for Xilinx XC3000 FPGA device family architectures were estimated at $26 + 9.5n$ CLBs (Configurable Logic Block), where n equals the number of bits in the radicand. A more recent implementation of the square root algorithm is reported in [217], where a 32–bit square root based on the non–restoring square root algorithm was implemented on Altera's FLEX10K20 device. The implementation required 161 Logic Cells (LCs), and achieved a maximum clock rate of 21.36 MHz.

In addition to the fixed–point FPGA–based square root implementations described in the previous chapter, floating–point square root operations have also been implemented. Both a low–cost iterative and a high–throughput pipelined version of a single precision floating point square root operator were reported in [174], and the results were refined in [280]. However, it was also noted that floating–point arithmetic has just recently become feasible with FPGAs [280].

The area–efficient and iterative square root algorithm in Publication P3 offers significant speed advantages compared to other published FPGA–based square root operations, because the algorithm produces two result digits per a clock cycle of 35 ns (1/28.57 MHz), and thus converges twice faster than the other implementations reported in [176], [217] and [253]. Furthermore, Publication P3 emphasizes the need for FPGA designers to have skills both in arithmetic algorithms and the underlying reprogrammable target technology, which is one of the main points in also [9]. The third main contribution of Publication P3 lies in the ability to adapt a relatively old and little–known software algorithm for hardware usage by parallelizing it as much as possible.

5.3.2 Sigmoid Function

Artificial neural networks (ANNs) have found applicability in providing solutions in pattern recognition, signal processing, and time series analysis problems, among others. Software–based simulations are useful for investigating the capabilities of neural network models and creating new algorithms, hardware implementations are essential for taking full advantage of the inherent parallelism of neural networks [99].

FPGA–based realization of ANNs [1] with a large number of neurons is a challenging task [300], and careful planning of every computational element is required. This is especially true for the sigmoid function (See Equation 5.1), which is the most common activation function in ANNs [301]:

$$y = \frac{1}{1 + \varepsilon^{-x}} \tag{5.1}$$

In practice, the sigmoid function cannot be implemented with a straightforward division and exponentiation, as these two operations are prohibitively demanding both in terms of speed and area. Publication P8 compares the speed, accuracy, and required area resources of four previously published piecewise linear and one piecewise second order approximation of the sigmoid function with a novel purely combinational approximation SIG–Sigmoid. The main contributions of Publication P8 are the introduction of an FPGA–based metric for comparing the performance of sigmoid function approximations, the automated design method for minimizing two–level logic functions for combinational approximations, and the final design which is the most accurate published FPGA–based sigmoid function approximation with tight area usage constraints. Due to its compactness and accuracy, the SIG–Sigmoid approximation plays an important role in designing massive fully connected artificial neural networks [125].

5.4 Digital Signal Processing

Digital signal processing (DSP) has pushed the limits of computation power, especially in terms of real–time applications. Nowadays, digital signal processing is used extensively in various engineering fields, for example, in communications technology, image and video processing, audio and speech processing,

[1]Due to their size limitations, CPLDs have not been as intensively researched as FPGAs as an implementation platform

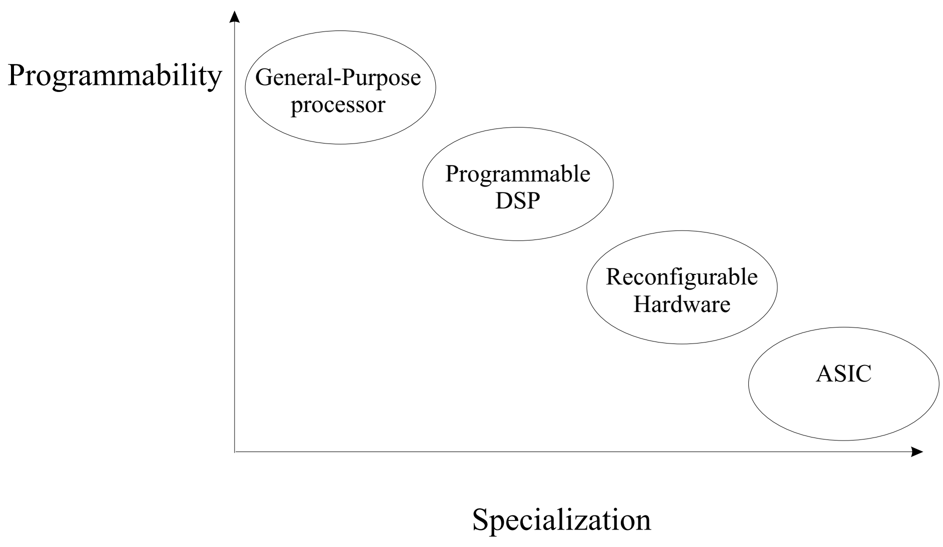

Figure 5.1: Digital Signal Processing Implementation spectrum [254] (Compare also to Figure 2.3).

target recognition and radar technology. FPGA–based DSP designs have been extensively reported in the published literature, and it can be safely assumed, that digital signal processing is the most significant application area for reprogrammable devices. However, despite the widely varying fields of target applications, digital signal processing systems exhibit the same basic computational characteristics [254]. Over the years, the Multiply and Accumulate (MAC) operation has remained the basic DSP operation, and a wide range of arithmetic functions, such as Fast Fourier Transform (FFT), convolution, and digital filtering algorithms all require an efficient MAC implementation, for which SRAM–based FPGAs are well suited [113]. Modern FPGA devices also have several DSP–specific features, which is a testimony to the commercial importance of digital signal processing for reprogrammable logic manufacturers. The DSP implementation spectrum can be regarded as a two–dimensional coordinate area with specialization and programmability as the horizontal and vertical axis (See Figure 5.1).

The main contributor for driving DSP technology forward has been the continuous improvement in the performance of programmable digital signal processors, which typically perform multiple instructions per clock cycle, and which are based on the Harvard Architecture with dedicated memory buses between the processor and separate code and data memories. However, the overall process is typically performed in a three–step series of memory–read, process, and memory–write instructions. This implies that the digital signal processor becomes less efficient when an algorithm is dependent on two or more of the past, present, and/or future state conditions. For this reason, the performance of digital signal processors degrades significantly with each additional MAC operation [113], whereas large FPGAs do not display a deterioration in DSP performance with additional MAC operations, provided that all required multiplications and additions can be instantiated in the same device with sufficient area resources. It has even been argued, that instruction–based digital signal processors will not be efficient enough for growing processing requirements, and that they will be replaced by reconfigurable systems [261].

Designing efficient FPGA–based implementations of core digital signal processing functions requires an intimate knowledge of the target architecture. For example, delay models and cost analyses developed for ASIC technology are not useful in designing and implementing fixed–point adders in FPGAs [298]. A popular design technique for the MAC function in digital filter design is to use Distributed Arithmetic (DA) techniques, instead of conventional arithmetic methods [113]. This is because distributed arithmetic makes extensive use of lookup tables (LUTs), which are the basic building blocks in modern FPGAs. The full flexibility of a general–purpose multiplier is also not required in many DSP applications, and only a limited range of values is needed on one of the multiplier inputs. Various techniques exist to perform constant–coefficient multiplication in an efficient manner, which enable multiplying two numbers by a series of shifts, additions and/or subtractions, thus requiring less area and executing faster than a general–purpose multiplier [264]. When designing demanding real–time applications, manual routing between neighboring arithmetic blocks is often needed to achieve required performance. The design steps in designing highly pipelined implementations include first describing a parallel pipelined architecture, then decomposing the signal processing into small computational blocks that fit into a single logic element with local communications and synchronous registers used for every logic element output, followed by placement of the logic elements starting with the most constrained interconnection regions, and working manually outward to the rest of the design [276]. Manual placement and routing of individual arithmetic functions is necessary in very demanding applications, but DSP engineers tend to be relatively unfamiliar with hardware design, and this may have slowed the wider adoption of FPGAs as the implementation platform of choice for DSP functions [149].

System integration, dynamic reprogramming, and high–level compilation are the main trends in determining the future of digital signal processing with FPGAs [254]. Wireless technologies are the major application area driving the move towards adaptive computing, and FPGAs are becoming viable alternatives to previous full–custom implementations for high–speed classes of DSP applications. An example of new application areas for FPGAs is radar front–end technology, where the trend is to bring the A/D–converter closer to the radar antenna elements, and process the incoming signals digitally instead of analog approaches [188]. FPGAs allow front–end radar processing systems to move away from costly custom chip with long development times, new fabrication runs for design changes, and the inability to prototype hardware without fabrication [207]. A tantalizing vision in the wireless world is offered by the great flexibility of FPGAs, which could enable designers to service multiple standards with the same communications platform [77].

5.4.1 General Parameter Extension for Polynomial FIR Predictors

In Publication P5, an FPGA–based adaptive general parameter (GP) extension [14] to polynomial FIR predictors was proposed for Rayleigh–distributed fading signal prediction. The implementation process included a custom–written MATLAB–to–VHDL conversion utility, analyzing finite wordlength effects, finetuning internal calculation accuracy, and taking into account specific target architecture features in implementing constant–coefficient multipliers. Publication P5 also includes a review of the suitability of FPGAs for digital signal processing tasks. As the implemented GP extension for polynomial FIR predictors has a sample rate of 19.6 million samples per second, it has several real–time applications in environments where low–complexity adaptation is required.

5.5 Cryptographic Algorithms

The goal of cryptography is to keep communication secure, so that eavesdroppers cannot decipher the transmitted messages. There are two kinds of cryptosystems: symmetric and asymmetric. Symmetric cryptosystems use the same session key (secret key) to encrypt and decrypt a message, and asymmetric cryptosystems use one key (the public key) to encrypt a message and a different key (the private key) to decrypt it [227]. When it comes to FPGA–based implementations, especially symmetric cryptosystems have been fairly thoroughly investigated, and it has been stated that the art of cryptographic algorithm implementation is reaching maturity [292]. FPGA–based implementations require both architectural and algorithmic optimization steps [247], which means that an effective FPGA–based cryptosystem requires expert knowledge on both cryptographic theory and design practice with FPGAs.

Reprogrammable devices have several advantages for implementing cryptographic algorithms [85] [292]:

- Algorithm Agility, which refers to the switching of cryptographic algorithms during operation of the targeted application. Algorithm agility allows also the deletion of broken algorithms, choosing algorithms according to personal preferences, and adding new algorithms.

- Algorithm Upload, which allows the upgrading of a fielded device with a new encryption algorithm, if, the current algorithm was broken, or a cryptographic standard has expired.

- Architecture Efficiency, which means that in certain cases the hardware architecture is more efficient if it is designed for a specific set of parameters.

- Resource Efficiency, which means that the same device can initiate communication with an asymmetric algorithm, and after the parameters of a symmetric cryptosystem have been agreed on, the device may perform run–time reconfiguration and continue functioning in symmetric encryption mode.

- Algorithm Modification, which allows the customization of certain parts of a standardized algorithms for proprietary purposes, for example with custom substitution boxes.

- Throughput, which means the proven performance speedup over general–purpose processors in numerous publications of FPGA–based cryptosystems.

- Cost Efficiency, which refers to the generally low– to medium–sized target market, where the large initial manufacturing startup costs included in ASIC–based designs make FPGAs a better alternative.

Specific design techniques for finetuning the performance of FPGA–based implementations of cryptosystems include combining sequences of logical operations into a single operator by setting the lookup tables appropriately, and implementing rotation, substitution and shifting in parallel at the bit–level. It should be noted, that general–purpose processors, in particular if programmed in a high–level language, are very inefficient at performing operations of this type [252]. In many respects, cryptographic algorithms are prime candidates for FPGA–based implementations, as they exhibit vast parallelism, are often pipelineable, make extensive use of bit–level operations on varying word lengths, and benefit from reprogrammability.

5.5.1 The International Data Encryption Algorithm

The International Data Encryption Algorithm (IDEA) was proposed by Lai and Massey in 1990 [166]. The design of IDEA was based on the concept of mixing operations from different algebraic groups, and IDEA operates with three group operations on pairs of 16–bit subblocks, namely bit–by–bit XOR (exclusive OR) of two 16–bit subblocks, addition of integers modulo 2^{16}, and multiplication of integers modulo $2^{16} + 1$, of which the multiplication is by far the most demanding operation to implement both in hardware and in software. In IDEA, both the plaintext and ciphertext are 64–bit blocks, and the secret key is 128 bits long. Encryption and decryption are essentially the same process, but only with different key subblocks. It has been stated, that prior to the introduction of AES (Advanced Encryption Standard), IDEA may have been the most secure symmetric cryptographic algorithm available to the public [227].

The first IDEA implementation on a VLSI chip at the Integrated Systems Laboratory at the ETH, Zurich, achieved a throughput of 115 Mb/s [166]. IDEA achieved particular fame due to its usage in the Pretty Good Privacy (PGP) secure E–mail package, and IDEA was also proposed to be used as a general design benchmark for reprogrammable devices [52], as it is computationally challenging but not impossible to implement, and has clearly understandable performance metrics.

Prior to Publication P6, representative compact and high–speed implementations of IDEA included, a bit–serial implementation described in [170], whose estimated performance on a Xilinx XCV1000 device was 2 Gb/s, and the bit–parallel implementation described in [56], whose throughput achieved 5.25 Gb/s on a Xilinx XCV1000 device. The bit–parallel implementation also included custom–written software to customize the FPGA reprogramming bitstream for different key schedules.

The main contribution of Publication P6 is its speed, since the throughput at 6.78 Gb/s represented the fastest published FPGA–based implementation of IDEA at that time. Other contributions of Publication P6 include implementing a fully pipelined algorithm with both inner and outer loop pipelining on a single Xilinx XCV1000 device, the efficient usage of the diminished–one number system, area–efficient implementation of the modulo (2^{16}) multiplication, and a comparison of the relative virtues of the Handel–C high–level design environment versus VHDL–based design.

After Publication P6, IDEA was implemented on a Xilinx XCV600 device, and this design achieved a throughput of 8.3 Gb/s [112]. The key to high throughput was replacing all the operational units involving the key with its constant–operand equivalents by partial reconfiguration, whose overhead was 4 ms. However, only few devices support partial reconfiguration, and the scheme requires a controlling microprocessor. Another recent implementation of IDEA achieved a throughput of 6 Gb/s by utilizing the embedded multipliers for the modulo (2^{16}) multiplication algorithm, but it seems that this scheme did not produce any area–saving advantages [213].

5.5.2 The Advanced Encryption Standard Algorithm

The two decades old Digital Encryption Standard (DES), with its short keylength of only 56 bits, became obsolete in the nineties, and The National Institute of Standards and Technology (NIST) of the United States initiated a selection process to develop a new Federal Information Processing Standard (FIPS) for the Advanced Encryption Standard (AES). After a careful and open selection process, which culminated in the final comparison process between five finalists, the Rijndael algorithm was selected in October 2000 as the new AES algorithm, and the standard became official in 2001.

The AES algorithm is a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits. A single round in the AES algorithm has four operations, namely byte swapping, constant Galois field multiplication, key addition, and an S–box substitution, which finds the multiplicative inverse of a byte in $GF(2^8)$. In terms of required logic resources, the S–box substitution is the dominant element of the AES round function [85].

Representative FPGA–based high–speed implementations of the AES algorithm prior to Publication P7 include the design reported in [100] with a throughput of 12.2 Gb/s, an implementation in a Xilinx FPGA with exceptionally large internal memory [198] with a throughput of 7 Gb/s, and an implementation with a throughput of 14 Gb/s, due to a design methodology which attempted to rigorously limit the critical path inside a Xilinx slice [246].

The main contribution of Publication P7 is its speed, since the throughput at 17.8 Gb/s represented the fastest published FPGA–based implementation of the AES algorithm at that time. Other contributions of Publication P7 include implementing the S–box substitution without internal memory, thus taking into account the architectural limitations and constraints of target device family, and a thorough comparison of the open literature on published high–speed FPGA–based implementations of the AES algorithm.

After Publication P7, efficient implementation of the AES algorithm have been a subject of intensive research in the FPGA design community. Notable recent achievements include a very small–sized AES core fitting into only 222 Xilinx slices and 3 Xilinx BlockRAMs [58], and probably the fastest current FPGA–based AES implementation with a throughput of up to 23.57 Gb/s [299].

Chapter 6

Results of the Thesis

This thesis presents a comprehensive overview of the possibilities and limitations of present–day reprogrammable logic. The results of the applicability of reprogrammable logic are presented in Chapter 4, where Section 4.1 presents results on the technical features of algorithms suitable for implementation on reprogrammable logic, Section 4.2 discusses the design issues in contemporary reprogrammable logic design, and the economic merits and drawbacks of reprogrammable logic are reviewed in Section 4.3. Chapter 4 ends with a summarized SWOT matrix of reprogrammable logic in Section 4.4.

The results presented in Chapter 4 are corroborated by a wide range of representative case studies presented in Publications P1–P8, which are overviewed and compared to contemporary research in Chapter 5. More specifically, the technical merits of the appended publications P1–P8 are as follows:

- Publication P1 describes an FPGA–based genetic algorithm (GA), which outperformed a similar software–based GA by a factor of 200.
- Publication P2 describes an FPGA–based simulation acceleration of ATM Adaptation Layer (AAL) Type 2 receiver, which outperformed a similar software–based implementation by a factor of 70.
- Publication P3 describes a compact square root operator for FPGAs, which converges more rapidly than other contemporary approximations.
- Publication P4 describes the first published FPGA–based implementation of Dijkstra's shortest path routing algorithm, which outperformed a similar software–based algorithm by a factor from 23.58 to 67.41.

- Publication P5 describes an FPGA–based implementation of adaptive general parameter (GP) extension to polynomial Finite Impulse Response (FIR) predictors with a sample rate of 19.6 million samples per second.

- Publication P6 describes the fastest[1] FPGA–based implementation of the International Data Encryption Algorithm (IDEA) with a throughput of 6.78 Gb/s.

- Publication P7 describes the fastest[1] FPGA–based implementation of the Advanced Encryption Standard (AES) with a throughput of 17.8 Gb/s.

- Publication P8 presents an extensive review of previous approaches to approximate the sigmoid function, and describes SIG–Sigmoid, which is the most accurate sigmoid function approximation with tight area constraints for reprogrammable logic.

The author's contribution in Publications P1–P8 has been presented in Section 1.1.

[1]At the date of publication.

Bibliography

[1] A. Abnous, K. Seno, Y. Ichikawa, M. Wan, and J. Rabaey, “Evaluation of a Low-Power Reconfigurable DSP Architecture,” in *Proceedings of the 1998 Fifth Reconfigurable Architectures Workshop (RAW’98)*, Orlando, FL, United States, Mar. 30, 1998, pp. 55–60.

[2] AccelChip Homepage. Visited on the 20th of November, 2004. [Online]. Available: http://www.accelchip.com

[3] A. A. Aggarwal and D. M. Lewis, “Routing Architectures for Hierarchical Field Programmable Gate Arrays,” in *Proceedings of the 1994 IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD’94)*, Cambridge, MA, United States, Oct. 10–12, 1994, pp. 475–478.

[4] O. Agrawal, H. Chang, B. Sharpe-Geisler, N. Schmitz, B. Nguyen, J. Wong, G. Tran, F. Fontana, and B. Harding, “An Innovative, Segmented High Performance FPGA Family with Variable-Grain-Architecture and Wide-gating Functions,” in *Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field-Programmable Gate Arrays (FPGA’99)*, Monterey, CA, United States, Feb. 21–23, 1999, pp. 17–26.

[5] E. Ahmed and J. Rose, “The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density,” in *Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field-Programmable Gate Arrays (FPGA’00)*, Monterey, CA, United States, Feb. 9–11, 2000, pp. 3–12.

[6] *Stratix Device Handbook*, Altera Corporation, San Jose, California, United States, Oct. 2003.

[7] Altera Development Kits. Visited on the 20th of November, 2004. [Online]. Available: http://www.altera.com/products/devkits/kit-dev_platforms.jsp

[8] R. Amerson, R. J. Carter, W. B. Culbertson, P. Kuekes, and G. Snider, "Teramac—Configurable Custom Computing," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'95)*, Napa Valley, CA, United States, Apr. 19–21, 1995, pp. 32–38.

[9] R. Andraka, "A Survey of CORDIC Algorithms for FPGA Based Computers," in *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field-Programmable Gate Arrays (FPGA'98)*, Monterey, CA, United States, Feb. 22–25, 1998, pp. 191–200.

[10] D. Andrews, D. Niehaus, and P. Ashenden, "Programming Models for Hybrid CPU/FPGA Chips," *IEEE Computer*, vol. 37, no. 1, pp. 118–120, Jan. 2004.

[11] Annapolis Micro Systems Homepage. Visited on the 20th of November, 2004. [Online]. Available: http://www.annapmicro.com

[12] J. M. Arnold, "The Splash 2 Software Environment," in *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines (FCCM'93)*, Los Alamitos, CA, United States, Apr. 5–7, 1993, pp. 88–93.

[13] J. M. Arnold, D. A. Buell, and E. G. Davis, "Splash 2," in *Proceedings of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'92)*, San Diego, CA, United States, June 29– July 1, 1992, pp. 316–322.

[14] A. Ashimov and D. Syzdykov, "Identification of High Dimensional System by the General Parameter Method," in *Preprints of the 8th Triennial World Congress of International Federation of Automatic Control*, Kyoto, Japan, Apr. 1981, pp. 32–37.

[15] P. M. Athanas and A. L. Abbott, "Real-Time Image Processing on a Custom Computing Platform," *IEEE Computer*, vol. 28, no. 2, pp. 16–25, Feb. 1995.

[16] P. M. Athanas and H. F. Silverman, "Processor Reconfiguration Through Instruction-Set Metamorphosis," *IEEE Computer*, vol. 26, no. 3, pp. 11–18, Mar. 1993.

[17] J. Babb, M. Frank, and A. Agarwal, “Solving Graph Problems with Dynamic Computation Structures,” in *High-Speed Computing, Digital Signal Processing, and Filtering Using Reconfigurable Logic, Volume 2914 of Proceedings of SPIE*, J. Schewel, P. M. Athanas, V. M. Bove, and J. Watson, Eds., Boston, MA, United States, Nov. 20–21, 1996, pp. 225–236.

[18] R. Baines and D. Pulley, “A Total Cost Approach to Evaluating Different Reconfigurable Architectures for Baseband Processing in Wireless Receivers,” *IEEE Communications Magazine*, vol. 41, no. 1, pp. 105–113, Jan. 2003.

[19] R. Ball, “ASICs Do Battle with FPGAs – a Survey by Electronics Weekly and Celoxica into ASIC and FPGA Design Trends,” *Electronics Weekly*, Dec. 17, 2003.

[20] P. Banerjee, N. Shenoyand, A. Choudhary, S. Hauck, C. Bachmann, M. Haldar, P. Joisha, A. Jones, A. Kanhare, A. Nayak, S. Periyacheri, S. Walkden, and D. Zaretsky, “A MATLAB Compiler for Distributed, Heterogeneous, Reconfigurable Computing Systems,” in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM’00)*, Napa Valley, CA, United States, Apr. 17–19, 2000, pp. 39–48.

[21] F. Barat and R. Lauwereins, “Reconfigurable Instruction Set Processors: A Survey,” in *Proceedings of the 11th International Workshop on Rapid System Prototyping (RSP’00)*, Paris, France, June 21–23, 2000, pp. 168–173.

[22] J. Becker and R. Hartenstein, “Configware and Morphware Going Mainstream,” *Journal of Systems Architecture*, vol. 49, no. 4–6, pp. 127–142, Sept. 2003.

[23] J. Becker, A. Kirschbaum, F.-M. Renner, and M. Glesner, “Perspectives of Reconfigurable Computing in Research, Industry and Education,” in *Proceedings of the 8th International Workshop on Field-Programmable Logic and Applications (FPL’98)*, R. W. Hartenstein and A. Keevallik, Eds., Tallinn, Estonia, Aug. 31– Sept. 3, 1998, pp. 39–48.

[24] J. Becker, T. Pionteck, and M. Glesner, “DReAM: A Dynamically Reconfigurable Architecture for Future Mobile Communication Applications,” in *Proceedings of the 10th International Workshop on Field-Programmable*

Logic and Applications (FPL'00), R. W. Hartenstein and H. Grünbacher, Eds., Villach, Austria, Aug. 27–30, 2000, pp. 312–321.

[25] P. Bellows and B. Hutchings, "JHDL – an HDL for Reconfigurable Systems," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'98)*, Napa Valley, CA, United States, Apr. 15–17, 1998, pp. 175–181.

[26] K. Benkrid, D. Crookes, A. Benkrid, and S. Belkacemi, "A Prolog-Based Hardware Development Environment," in *Proceedings of the 12th International Workshop on Field-Programmable Logic and Applications (FPL'02)*, M. Glesner, P. Zipf, and M. Renovell, Eds., Montpellier, France, Sept. 2–4, 2002, pp. 370–380.

[27] N. W. Bergmann and J. C. Mudge, "Comparing the performance of FPGA-based custom computers with general-purpose computers for DSP applications," in *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines (FCCM'94)*, Napa Valley, CA, United States, Apr. 10–13, 1994, pp. 164–171.

[28] P. Bertin, D. Roncin, and J. Vuillemin, "Introduction to Programmable Active Memories," DEC Paris Research Laboratory," PRL Research Report 3, June 1989.

[29] P. Bertin and H. Touati, "PAM Programming Environments: Practice and Experience," in *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines (FCCM'94)*, Napa Valley, CA, United States, Apr. 10–13, 1994, pp. 133–138.

[30] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," in *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications (FPL'97)*, W. Luk, P. Y. K. Cheung, and M. Glesner, Eds., London, United Kingdom, Sept. 1–3, 1997, pp. 213–222.

[31] ——, "How Much Logic Should Go in an FPGA Logic Block?" *IEEE Design and Test of Computers*, vol. 15, no. 1, pp. 10–15, Jan.-Mar. 1998.

[32] ——, "FPGA Routing Architecture: Segmentation and Buffering to Optimize Speed and Density," in *Proceedings of the 1999 ACM/SIGDA Seventh*

International Symposium on Field-Programmable Gate Arrays (FPGA'99), Monterey, CA, United States, Feb. 21–23, 1999, pp. 59–68.

[33] R. Bielby. Digital Consumer Convergence Demands Reprogrammability, Xcell Journal Online, Q1 2002. Visited on the 20th of November, 2004. [Online]. Available: http://www.xilinx.com/publications/products/strategic/xc_digcon.htm

[34] R. Bittner and P. Athanas, "Wormhole Run-Time Reconfiguration," in *Proceedings of the 1997 ACM/SIGDA Fifth International Symposium on Field-Programmable Gate Arrays (FPGA'97)*, Monterey, CA, United States, Feb. 9–11, 1997, pp. 79–85.

[35] R. A. Bittner Jr., P. M. Athanas, and M. D. Musgrove, "Colt: An Experiment in Wormhole Run-Time Reconfiguration," in *High-Speed Computing, Digital Signal Processing, and Filtering Using Reconfigurable Logic, Volume 2914 of Proceedings of SPIE*, J. Schewel, P. M. Athanas, V. M. Bove, and J. Watson, Eds., Boston, MA, United States, Nov. 20–21, 1996, pp. 187–194.

[36] E. I. Boemo, G. G. de Rivera, S. López-Buedo, and J. M. Meneses, "Some Notes on Power Management on FPGA-Based Systems," in *Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications (FPL'95)*, W. Moore and W. Luk, Eds., Oxford, United Kingdom, Aug. 29– Sept. 1, 1995, pp. 149–157.

[37] K. Bondalapati and V. K. Prasanna, "Reconfigurable Computing Systems," *Proceedings of the IEEE*, vol. 90, no. 7, pp. 1201–1217, July 2002.

[38] G. Borriellp, C. Ebeling, S. Hauck, and S. Burns, "The Triptych FPGA Architecture," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 3, no. 4, pp. 473–482, Dec. 1995.

[39] F. Braun, J. W. Lockwood, and M. Waldvogel, "Reconfigurable Router Modules Using Network Protocol Wrappers," in *Proceedings of the 11th International Workshop on Field-Programmable Logic and Applications (FPL'01)*, G. Brebner and R. Woods, Eds., Belfast, Northern Ireland, United Kingdom, Aug. 27–29, 2001, pp. 254–263.

[40] G. Brebner, “Field-Programmable Logic: Catalyst for New Computing Paradigms,” in *Proceedings of the 8th International Workshop on Field-Programmable Logic and Applications (FPL’98)*, R. W. Hartenstein and A. Keevallik, Eds., Tallinn, Estonia, Aug. 31– Sept. 3, 1998, pp. 49–58.

[41] G. Brebner and O. Diessel, “Chip-Based Reconfigurable Task Management,” in *Proceedings of the 11th International Workshop on Field-Programmable Logic and Applications (FPL’01)*, G. Brebner and R. Woods, Eds., Belfast, Northern Ireland, United Kingdom, Aug. 27–29, 2001, pp. 182–191.

[42] S. Brown and J. Rose, “FPGA and CPLD Architectures: A Tutorial,” *IEEE Design and Test of Computers*, vol. 13, no. 2, pp. 42–57, Summer 1996.

[43] D. A. Buell, J. M. Arnold, and W. J. Kleinfelder, *Splash 2: FPGAs in a Custom Computing Machine*. IEEE Computer Society Press, 1996.

[44] J. Burns, A. Donlin, J. Hogg, S. Singh, and M. D. Wit, “A Dynamic Reconfiguration Run-Time System,” in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM’97)*, Napa Valley, CA, United States, Apr. 16–18, 1997, pp. 66–75.

[45] D. Bursky, “We Must Hold the Line on Soarign ASIC Design Costs,” *Electronic Design*, vol. 50, no. 21, p. 22, Oct. 14, 2002.

[46] ——, “Adaptive Logic Molds Faster, More Efficient FPGAs,” *Electronic Design*, vol. 52, no. 6, p. 48, Mar. 15, 2004.

[47] S. Cadambi, J. Weener, S. C. Goldstein, H. Schmit, and D. E. Thomas, “Managing Pipeline-Reconfigurable FPGAs,” in *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field-Programmable Gate Arrays (FPGA’98)*, Monterey, CA, United States, Feb. 22–25, 1998, pp. 55–64.

[48] O. Cadenas and G. Megson, “A Clocking Technique with Power Savings in Virtex-Based Pipelined Designs,” in *Proceedings of the 12th International Workshop on Field-Programmable Logic and Applications (FPL’02)*, M. Glesner, P. Zipf, and M. Renovell, Eds., Montpellier, France, Sept. 2–4, 2002, pp. 322–331.

[49] J. M. Cardoso and H. C. Neto, "Macro-Based Hardware Compilation of Java$^{\text{TM}}$ Bytecodes into a Dynamic Reconfigurable Computing System," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'99)*, Napa Valley, CA, United States, Apr. 21–23, 1999, pp. 2–11.

[50] J. E. Carrillo and P. Chow, "The Effect of Reconfigurable Units in Superscalar Processors," in *Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field-Programmable Gate Arrays (FPGA'01)*, Monterey, CA, United States, Feb. 11–13, 2001, pp. 141–150.

[51] W. S. Carter, K. Duong, R. H. Freeman, H.-C. Hsieh, J. Y. Ja, J. E. Mahoney, L. T. Ngo, and S. L. Sze, "A User Programmable Reconfigurable Logic Array," in *Proceedings of the IEEE 1986 Custom Integrated Circuits Conference*, Rochester, NY, United States, May 12–15, 1986, pp. 233–235.

[52] E. Caspi and N. Weaver, "IDEA as a Benchmark for Reconfigurable Computing," BRASS Research Group, University of California at Berkeley, CA, United States, Tech. Rep.

[53] S. Casselman, "Virtual Computing and The Virtual Computer," in *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines (FCCM'93)*, Los Alamitos, CA, United States, Apr. 5–7, 1993, pp. 43–48.

[54] C. Chang, K. Kuusilinna, B. Richards, and R. W. Brodersen, "Implementation of BEE: a Real–Time Large–Scale Hardware Emulation Engine," in *Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field-Programmable Gate Arrays (FPGA'03)*, Monterey, CA, United States, Feb. 24–26, 2003, pp. 91–99.

[55] K.-C. Chen, J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar, "DAG-Map: Graph-Based FPGA Technology Mapping for Delay Optimization," *IEEE Design and Test of Computers*, vol. 9, no. 3, pp. 7–20, Sept. 1992.

[56] O. Cheung, K. Tsoi, P. H. W. Leong, and M. Leong, "Tradeoffs in Parallel and Serial Implementations of the International Data Encryption Algorithm IDEA," in *Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems (CHES'01)*, Çetin Kaya Koç, D. Naccache, and C. Paar, Eds., Paris, France, May 14–16, 2001, pp. 333–347.

[57] Y. H. Cho, S. Navab, and W. H. Mangione-Smith, "Specialized Hardware for Deep Network Packet Filtering," in *Proceedings of the 12th International Workshop on Field-Programmable Logic and Applications (FPL'02)*, M. Glesner, P. Zipf, and M. Renovell, Eds., Montpellier, France, Sept. 2–4, 2002, pp. 452–461.

[58] P. Chodowiec and K. Gaj, "Very Compact FPGA Implementation of the AES Algorithm," in *Proceedings of the 5th International Workshop on Cryptographic Hardware and Embedded Systems (CHES'03)*, C. D. Walter, Çetin Kaya Koç, and C. Paar, Eds., Cologne, Germany, Sept. 8–10, 2003, pp. 319–333.

[59] P. Chow, S. O. Seo, D. Au, T. Choy, B. Fallah, D. Lewis, C. Li, and J. Rose, "A 1.2μm CMOS FPGA using Cascaded Logic Blocks and Segmented Routing," in *FPGAs, edited from the Oxford 1991 International Workshop on Field Programmable Logic and Applications*, W. R. Moore and W. Luk, Eds., Oxford, United Kingdom, 1991, pp. 91–102.

[60] P. Chow, S. O. Seo, J. Rose, K. Chung, G. Páez-Monzón, and I. Rahardja, "The Design of an SRAM-Based Field-Programmable Gate Array—Part I: Architecture," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 2, pp. 191–197, June 1999.

[61] A. Chowdhary and J. P. Hayes, "General Technology Mapping for Field-Programmable Gate Arrays Based on Lookup Tables," *ACM Transactions on Design Automation of Electronic Systems*, vol. 7, no. 1, pp. 1–32, Jan. 2002.

[62] K. Compton and S. Hauck, "Reconfigurable Computing: a Survey of Systems and Software," *ACM Computing Surveys (CSUR)*, vol. 34, no. 2, pp. 171–210, 2002.

[63] ——, "Flexibility Measurement of Domain–Specific Reconfigurable Hardware," in *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field-Programmable Gate Arrays (FPGA'04)*, Monterey, CA, United States, Feb. 22–24, 2004, pp. 155–161.

[64] J. Cong and Y. Ding, "Combinational Logic Synthesis for LUT Based Field Programmable Gate Arrays," *ACM Transactions on Design Automation of Electronic Systems*, vol. 1, no. 2, pp. 145–204, Apr. 1996.

[65] J. Cong and S. Xu, “Technology Mapping for FPGAs with Embedded Memory Blocks,” in *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field-Programmable Gate Arrays (FPGA’98)*, Monterey, CA, United States, Feb. 22–25, 1998, pp. 179–188.

[66] W. B. Culbertson, R. Amerson, R. J. Carter, P. Kuekes, and G. Snider, “Defect Tolerance on the Teramac Custom Computer,” in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM’97)*, Napa Valley, CA, United States, Apr. 16–18, 1997, pp. 116–123.

[67] M. Dagenais, V. Agarwal, and N. Rumin, “McBoole, A New Procedure for Exact Logic Minimization,” *IEEE Transactions on Computer–Aided Design*, vol. CAD–5, no. 1, pp. 229–238, Jan. 1986.

[68] M. Dales, “Initial Analysis of the Proteus Architecture,” in *Proceedings of the 11th International Workshop on Field-Programmable Logic and Applications (FPL’01)*, G. Brebner and R. Woods, Eds., Belfast, Northern Ireland, United Kingdom, Aug. 27–29, 2001, pp. 623–627.

[69] A. Dandalis, A. Mei, and V. K. Prasanna, “Domain Specific Mapping for Solving Graph Problems on Reconfigurable Devices,” in *Proceedings of the 1999 Sixth Reconfigurable Architectures Workshop (RAW’99)*, San Juan, Puerto Rico, United States, Apr. 12, 1999.

[70] A. DeHon, “DPGA–Coupled Microprocessors: Commodity ICs for the Early 21st Century,” in *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines (FCCM’94)*, Napa Valley, CA, United States, Apr. 10–13, 1994, pp. 31–39.

[71] ——, “DPGA Utilization and Application,” in *Proceedings of the 1996 ACM/SIGDA Fourth International Symposium on Field-Programmable Gate Arrays (FPGA’96)*, Monterey, CA, United States, Feb. 11–13, 1996, pp. 115–121.

[72] ——, “Entropy, Counting, and Programmable Interconnect,” in *Proceedings of the 1996 ACM/SIGDA Fourth International Symposium on Field-Programmable Gate Arrays (FPGA’96)*, Monterey, CA, United States, Feb. 11–13, 1996, pp. 73–79.

[73] ——, “Reconfigurable Architectures for General–Purpose Computing,” Ph.D. dissertation, Massachusetts Institute of Technology, 1996.

[74] ——, "Balancing Interconnect and Computation in a Reconfigurable Computing Array (or, why you don't really want 100% LUT utilization)," in *Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field-Programmable Gate Arrays (FPGA'99)*, Monterey, CA, United States, Feb. 21–23, 1999, pp. 69–78.

[75] ——, "The Density Advantage of Configurable Computing," *IEEE Computer*, vol. 33, no. 4, pp. 41–49, Apr. 2000.

[76] A. DeHon and J. Wawrzynek, "The Case for Reconfigurable Processors," Preliminary Copy for Berkeley Reconfigurable Architectures, Systems, and Software Group, University of California at Berkeley, Computer Science Division, Jan. 31, 1997.

[77] C. Dick and F. J. Harris, "Configurable Logic for Digital Communications: Some Signal Processing Perspectives," *IEEE Communications Magazine*, vol. 37, no. 8, pp. 107–111, Aug. 1999.

[78] B. Dipert, "Lies, Damn Lies, and Benchmarks: The Race for the Truth Is On," *EDN*, vol. 44, no. 11, pp. 54–68, May 27, 1999.

[79] J. Ditmar, K. Torkelsson, and A. Jantsch, "A Dynamically Reconfigurable FPGA–Based Content Addressable Memory for Internet Protocol Characterization," in *Proceedings of the 10th International Workshop on Field-Programmable Logic and Applications (FPL'00)*, R. W. Hartenstein and H. Grünbacher, Eds., Villach, Austria, Aug. 27–30, 2000, pp. 19–28.

[80] A. Donlin, "Self Modifying Circuitry - A Platform for Tractable Virtual Circuitry," in *Proceedings of the 8th International Workshop on Field-Programmable Logic and Applications (FPL'98)*, R. W. Hartenstein and A. Keevallik, Eds., Tallinn, Estonia, Aug. 31– Sept. 3, 1998, pp. 200–208.

[81] N. Dutt and K. Choi, "Configurable Processors for Embedded Computing," *IEEE Computer*, vol. 36, no. 1, pp. 120–123, Jan. 2003.

[82] C. Ebeling, D. C. Cronquist, and P. Franklin, "RaPiD – Reconfigurable Pipelined Datapath," in *Proceedings of the 6th International Workshop on Field-Programmable Logic and Applications (FPL'96)*, R. W. Hartenstein and M. Glesner, Eds., Darmstadt, Germany, Sept. 23–25, 1996, pp. 126–135.

[83] C. Ebeling, L. McMurchie, and S. H. S. Burns, "Mapping Tools for the Triptych FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 3, no. 4, pp. 473–482, Dec. 1995.

[84] EE382 Processor Design lecture notes, Stanford University. Visited on the 20th of November, 2004. [Online]. Available: http://www.stanford.edu/class/ee382/

[85] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA–Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 4, pp. 545–557, Aug. 2001.

[86] J. G. Eldredge and B. L. Hutchings, "Run-Time Reconfiguration: A Method For Enhancing the Functional Density of SRAM-Based FPGAs," *The Journal of VLSI Signal Processing*, vol. 12, no. 1, pp. 67–86, Jan. 1996.

[87] R. Enzler, M. Platzner, C. Plessl, L. Thiele, and G. Tröster, "Reconfigurable Processors for Handhelds and Wearables: Application Analysis," in *Reconfigurable Technology: FPGAs and Reconfigurable Processors for Computing and Communications III, Volume 4525 of Proceedings of SPIE*, J. Schewel, P. M. Athanas, P. B. James-Roxby, and J. T. McHenry, Eds., Denver, CO, United States, Aug. 21–22, 2001, pp. 135–146.

[88] G. Estrin, "Organization of Computer Systems – the Fixed Plus Variable Structure Computer," in *Proceedings of Western Joint Computer Conference*, New York, NY, United States, May 1960, pp. 33–40.

[89] ——, "Reconfigurable Computer Origins: the UCLA Fixed-Plus-Variable (F+V) Structure Computer," *IEEE Annals of the History of Computing*, vol. 24, no. 4, pp. 3–9, Oct.-Dec. 2002.

[90] G. Estrin and C. Viswanathan, "Organization of a "Fixed-Plus-Variable" Structure Computer for Computation of Eigenvalues and Eigenvectors of Real Symmetric Matrices," *Journal of the ACM*, vol. 9, no. 1, pp. 41–60, Jan. 1962.

[91] Excalibur Device Overview. Visited on the 20th of November, 2004. [Online]. Available: http://www.altera.com/literature/ds/ds_arm.pdf

[92] H. Fallside and M. J. Smith, “Internet Connected FPL,” in *Proceedings of the 10th International Workshop on Field-Programmable Logic and Applications (FPL’00)*, R. W. Hartenstein and H. Grünbacher, Eds., Villach, Austria, Aug. 27–30, 2000, pp. 48–57.

[93] K. Feske, S. Mulka, M. Koegst, and G. Elst, “Technology-Driven FSM Partitioning for Synthesis of Large Sequential Circuits Targeting Lookup-Table Based FPGAs,” in *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications (FPL’97)*, W. Luk, P. Y. K. Cheung, and M. Glesner, Eds., London, United Kingdom, Sept. 1–3, 1997, pp. 235–244.

[94] C. A. Fields, “The Proper Use of Hierarchy in HDL-Based High Density FPGA Design,” in *Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications (FPL’95)*, W. Moore and W. Luk, Eds., Oxford, United Kingdom, Aug. 29– Sept. 1, 1995, pp. 168–177.

[95] M. J. Flynn and A. A. Liddicoat, “Technology Trends and Adaptive Computing,” in *Proceedings of the 11th International Workshop on Field-Programmable Logic and Applications (FPL’01)*, G. Brebner and R. Woods, Eds., Belfast, Northern Ireland, United Kingdom, Aug. 27–29, 2001, pp. 1–5.

[96] FPGA Advantage. Visited on the 20th of November, 2004. [Online]. Available: http://www.mentor.com/fpga-advantage

[97] FPGA vs. ASIC Project Cost Calculator. Visited on the 20th of November, 2004. [Online]. Available: http://www.altera.com/products/devices/cost/cst-cost_step1.jsp

[98] R. J. Francis, “Technology Mapping for Lookup–Table Based Field Programmable Gate Arrays,” Ph.D. dissertation, University of Toronto, 1992.

[99] R. Gadea, V. Herrero, A. Sebastia, and A. Mocholi, “The Role of the Embedded Memories in the Implementation of Artificial Neural Networks,” in *Proceedings of the 10th International Workshop on Field-Programmable Logic and Applications (FPL’00)*, R. W. Hartenstein and H. Grünbacher, Eds., Villach, Austria, Aug. 27–30, 2000, pp. 785–788.

[100] K. Gaj and P. Chodowiec, "Fast Implementation and Fair Comparison of the Final Candidates for Advanced Encryption Standard Using Field Programmable Gate Arrays," in *Topics in Cryptology – CT–RSA 2001, Proceedings of the Cryptographer's Track at RSA Conference 2001*, D. Naccache, Ed., San Francisco, CA, United States, Apr. 8–12, 2001, pp. 84–99.

[101] D. Galloway, "The Transmogrifier C Hardware Description Language and Compiler for FPGAs," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'95)*, Napa Valley, CA, United States, Apr. 19–21, 1995, pp. 136–144.

[102] V. George, H. Zhang, and J. Rabaey, "The Design of a Low Energy FPGA," in *Proceedings of the 1999 International Symposium on Low Power Electronics and Design (ISLPED'99)*, San Diego, CA, United States, Aug. 16–17, 1999, pp. 188–193.

[103] L. Geppert, "The New Indelible Memories," *IEEE Spectrum*, vol. 40, no. 3, pp. 48–54, Mar. 2003.

[104] I. Glover and P. Grant, *Digital Communications*. Prentice Hall Europe, 1998.

[105] B. S. Goda, J. F. McDonald, S. R. Carlough, T. W. Krawczyk Jr., and R. P. Kraft, "SiGe HBT BiCMOS FPGAs for Fast Reconfigurable Computing," *IEE Proceedings – Computers and Digital Techniques*, vol. 147, no. 3, pp. 189–194, May 2000.

[106] M. Gokhale, D. Dubois, A. Dubois, M. Boorman, S. Poole, and V. Hogsett, "Granidt: Towards Gigabit Rate Network Intrusion Detection Technology," in *Proceedings of the 12th International Workshop on Field-Programmable Logic and Applications (FPL'02)*, M. Glesner, P. Zipf, and M. Renovell, Eds., Montpellier, France, Sept. 2–4, 2002, pp. 404–413.

[107] M. Gokhale, W. Holmes, A. Kopser, S. Lucas, R. Minnich, D. Sweely, and D. Lopresti, "Building and Using a Highly Parallel Programmable Logic Array," *IEEE Computer*, vol. 24, no. 1, pp. 81–89, Jan. 1991.

[108] M. Gokhale, J. Stone, and J. Arnold, "Stream-Oriented FPGA Computing in the Streams-C High Level Language," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'00)*, Napa Valley, CA, United States, Apr. 17–19, 2000, pp. 49–56.

[109] M. B. Gokhale and J. M. Stone, "NAPA C: Compiling for a Hybrid RISC/FPGA Architecture," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'98)*, Napa Valley, CA, United States, Apr. 15–17, 1998, pp. 126–135.

[110] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison–Wesley Publishing Company, 1989.

[111] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. R. Taylor, "PipeRench: A Reconfigurable Architecture and Compiler," *IEEE Computer*, vol. 33, no. 4, pp. 70–77, Apr. 2000.

[112] I. Gonzalez, S. López-Buedo, F. J. Gómez, and J. Martínez, "Using Partial Reconfiguration in Cryptographic Applications: An Implementation of the IDEA Algorithm," in *Proceedings of the 13th International Workshop on Field-Programmable Logic and Applications (FPL'03)*, P. Y. K. Cheung, G. A. Constantinides, and J. T. de Sousa, Eds., Lisbon, Portugal, Sept. 1–3, 2003, pp. 194–203.

[113] G. R. Goslin, "A Guide to Using Field Programmable Gate Arrays FPGAs for Application-Specific Digital Signal Processing Performance," in *High-Speed Computing, Digital Signal Processing, and Filtering Using Reconfigurable Logic, Volume 2914 of Proceedings of SPIE*, J. Schewel, P. M. Athanas, V. M. Bove, and J. Watson, Eds., Boston, MA, United States, Nov. 20–21, 1996, pp. 321–331.

[114] P. Graham and B. Nelson, "A Hardware Genetic Algorithm for the Traveling Salesman Problem on Splash 2," in *Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications (FPL'95)*, W. Moore and W. Luk, Eds., Oxford, United Kingdom, Aug. 29– Sept. 1, 1995, pp. 352–361.

[115] ——, "Genetic Algorithms In Software and In Hardware — A Performance Analysis Of Workstation and Custom Computing Machine Implementations," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'96)*, Napa Valley, CA, United States, Apr. 17–19, 1996, pp. 216–225.

[116] M. Gschwind and V. Salapura, "A VHDL Design Methodolgy for FPGAs," in *Proceedings of the 5th International Workshop on Field-Programmable*

Logic and Applications (FPL'95), W. Moore and W. Luk, Eds., Oxford, United Kingdom, Aug. 29– Sept. 1, 1995, pp. 208–217.

[117] S. Guccione, D. Levi, and P. Sundararajan, "JBits: Java–Based Interface for Reconfigurable Computing," in *Proceedings of Second Annual Military and Aerospace Applications of Programmable Devices and Technologies (MAPLD'99)*, The Johns Hopkins University, Laurel, Maryland, United States, Sept. 1999.

[118] S. A. Guccione and M. Gonzalez, "Classification and Performance of Reconfigurable Architectures," in *Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications (FPL'95)*, W. Moore and W. Luk, Eds., Oxford, United Kingdom, Aug. 29– Sept. 1, 1995, pp. 439–448.

[119] S. A. Guccione and D. Levi, "Run-Time Parameterizable Cores," in *Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications (FPL'99)*, P. Lysaght, J. Irvine, and R. Hartenstein, Eds., Glasgow, United Kingdom, Aug. 30– Sept. 1, 1999, pp. 215–222.

[120] J. Guedj, Ed., *Inside the Minds: The Semiconductor Industry.* Aspatore Books, 2002.

[121] S. Guo and W. Luk, "Compiling Ruby into FPGAs," in *Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications (FPL'95)*, W. Moore and W. Luk, Eds., Oxford, United Kingdom, Aug. 29– Sept. 1, 1995, pp. 188–197.

[122] J. D. Hadley and B. L. Hutchings, "Design Methodologies for Partially Reconfigured Systems," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'95)*, Napa Valley, CA, United States, Apr. 19–21, 1995, pp. 78–84.

[123] I. Hadžić and J. M. Smith, "P4: A Platform for FPGA Implementation of Protocol Boosters," in *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications (FPL'97)*, W. Luk, P. Y. K. Cheung, and M. Glesner, Eds., London, United Kingdom, Sept. 1–3, 1997, pp. 438–447.

[124] M. Haldar, A. Nayak, N. Shenoy, A. Choudhary, and P. Banerjee, "FPGA Hardware Synthesis from MATLAB," in *Proceedings of Fourteenth International Conference on VLSI Design*, Bangalore, India, Jan. 3–7, 2001, pp. 299–304.

[125] A. Hämäläinen, M. Tommiska, and J. Skyttä, "FPGA–Based Implementation of a 59–Neuron Feedforward Neural Network with a 17.1 Gbps Interlayer Throughput," in *Proceedings of the International Conference on Artificial Intelligence (IC-AI'04)*, vol. I, Las Vegas, Nevada, United States, June 21–24, 2004, pp. 181–187.

[126] HANDEL-C Language Overview. Visited on the 20th of November, 2004. [Online]. Available: http://www.celoxica.com/techlib/files/CEL-W0307171KDD-47.pdf

[127] HardCopy Devices: ASIC Gain without the Pain. Visited on the 20th of November, 2004. [Online]. Available: http://www.altera.com/products/devices/hardcopy/hrd-index.html

[128] R. W. Hartenstein, M. Herz, T. Hoffmann, and U. Nageldinger, "On Reconfigurable Co-Processing Units," in *Proceedings of the 1998 Fifth Reconfigurable Architectures Workshop (RAW'98)*, Orlando, FL, United States, Mar. 30, 1998, pp. 67–72.

[129] R. Hartenstein, "A Decade of Reconfigurable Computing: a Visionary Retrospective," in *Proceedings of Design, Automation and Test in Europe. Conference and Exhibition 2001*, Munich, Germany, Mar. 13–16, 2001, pp. 642–649.

[130] ——, "The Digital Divide of Computing," in *Proceedings of the First Conference on Computing Frontiers (CF'04)*, S. Vassiliadis, J.-L. Gaudiot, and V. Piur, Eds., Ischia, Italy, Apr. 14–16, 2004, pp. 357–362.

[131] S. Hauck, "Configuration Prefetch for Single Context Reconfigurable Coprocessors," in *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field-Programmable Gate Arrays (FPGA'98)*, Monterey, CA, United States, Feb. 22–25, 1998, pp. 65–74.

[132] ——, "The Roles of FPGAs in Reprogrammable Systems," *Proceedings of the IEEE*, vol. 86, no. 4, pp. 615–639, Apr. 1998.

[133] S. Hauck, G. Borriello, S. Burns, and C. Ebeling, "Montage: An FPGA for Synchronous and Asynchronous Circuits," in *Proceedings of the 2nd International Workshop on Field-Programmable Logic and Applications (FPL'92)*, H. Grünbacher and R. W. Hartenstein, Eds., Vienna, Austria, Aug. 31– Sept. 2, 1992, pp. 44–51.

[134] S. Hauck, T. W. Fry, M. M. Hosler, and J. P. Kao, "The Chimaera Reconfigurable Functional Unit," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 2, pp. 206–217, Feb. 2004.

[135] S. Hauck, M. M. Hosler, and T. W. Fry, "High-Performance Carry Chains for FPGA's," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 2, pp. 138–147, Apr. 2000.

[136] J. R. Hauser and J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'97)*, Napa Valley, CA, United States, Apr. 16–18, 1997, pp. 12–21.

[137] F. Heile and A. Leaver, "Hybrid Product Term and LUT Based Architectures Using Embedded Memory Blocks," in *Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field-Programmable Gate Arrays (FPGA'99)*, Monterey, CA, United States, Feb. 21–23, 1999, pp. 13–16.

[138] F. Heile, A. Leaver, and K. Veenstra, "Programmable Memory Blocks Supporting Content-Addressable Memory," in *Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field-Programmable Gate Arrays (FPGA'00)*, Monterey, CA, United States, Feb. 9–11, 2000, pp. 13–21.

[139] J.-P. Heron, R. Woods, S. Sezer, and R. H. Turner, "Development Of a Run-Time Reconfiguration System With Low Reconfiguration Overhead," *The Journal of VLSI Signal Processing*, vol. 28, no. 1/2, pp. 97–113, May 2001.

[140] J. R. Hess, D. C. Lee, S. J. Harper, M. T. Jones, and P. M. Athanas, "Implementation and Evaluation of a Prototype Reconfigurable Router," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'99)*, Napa Valley, CA, United States, Apr. 21–23, 1999, pp. 44–51.

[141] C. H. Ho, P. H. Leong, K. H. Tsoi, R. Ludewig, P. Zipf, A. G. Ortiz, and M. Glesner, "Fly — A Modifiable Hardware Compiler," in *Proceedings of the 12th International Workshop on Field-Programmable Logic and Applications (FPL'02)*, M. Glesner, P. Zipf, and M. Renovell, Eds., Montpellier, France, Sept. 2–4, 2002, pp. 381–390.

[142] W. K. C. Ho and S. J. E. Wilton, "Logical-to-Physical Memory Mapping for FPGAs with Dual-Port Embedded Arrays," in *Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications (FPL'99)*, P. Lysaght, J. Irvine, and R. Hartenstein, Eds., Glasgow, United Kingdom, Aug. 30– Sept. 1, 1999, pp. 111–123.

[143] E. L. Horta, J. W. Lockwood, and S. T. Kofuji, "Using PARBIT to Implement Partial Run-Time Reconfigurable Systems," in *Proceedings of the 12th International Workshop on Field-Programmable Logic and Applications (FPL'02)*, M. Glesner, P. Zipf, and M. Renovell, Eds., Montpellier, France, Sept. 2–4, 2002, pp. 182–191.

[144] L. Huelsbergen, "A Representation for Dynamic Graphs in Reconfigurable Hardware and its Application to Fundamental Graph Algorithms," in *Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field-Programmable Gate Arrays (FPGA'00)*, Monterey, CA, United States, Feb. 9–11, 2000, pp. 105–115.

[145] B. L. Hutchings, R. Franklin, and D. Carver, "Assisting Network Intrusion Detection with Reconfigurable Hardware," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'02)*, Napa Valley, CA, United States, Apr. 21–24, 2002, pp. 111–120.

[146] B. L. Hutchings and M. J. Wirthlin, "Implementation Approaches for Reconfigurable Logic Applications," in *Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications (FPL'95)*, W. Moore and W. Luk, Eds., Oxford, United Kingdom, Aug. 29– Sept. 1, 1995, pp. 419–428.

[147] M. Hutton, K. Adibsamii, and A. Leaver, "Timing-Driven Placement for Hierarchical Programmable Logic Devices," in *Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field-Programmable Gate*

Arrays (FPGA'01), Monterey, CA, United States, Feb. 11–13, 2001, pp. 3–11.

[148] M. Hutton, V. Chan, P. Kazarian, V. Maruri, T. Ngai, J. Park, R. Patel, B. Pedersen, J. Schleicher, and S. Shumarayev, "Interconnect Enhancements for a High-Speed PLD Architecture," in *Proceedings of the 2002 Tenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'02)*, Monterey, CA, United States, Feb. 24–26, 2002, pp. 3–10.

[149] J. Hwang and J. Ballagh, "Building Custom FIR Filters Using System Generator," in *Proceedings of the 12th International Workshop on Field-Programmable Logic and Applications (FPL'02)*, M. Glesner, P. Zipf, and M. Renovell, Eds., Montpellier, France, Sept. 2–4, 2002, pp. 1101–1104.

[150] ISE Foundation. Visited on the 20th of November, 2004. [Online]. Available: http://www.xilinx.com/products/design_resources/design_tool/index.htm

[151] C. Iseli and E. Sanchez, "A C++ Compiler for FPGA Custom Execution Units Synthesis," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'95)*, Napa Valley, CA, United States, Apr. 19–21, 1995, pp. 173–179.

[152] T. Isshiki and W. W.-M. Dai, "Bit-Serial Pipeline Synthesis for Multi-FPGA Systems with C++ Design Capture," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'96)*, Napa Valley, CA, United States, Apr. 17–19, 1996, pp. 38–47.

[153] *B-ISDN ATM Adaptation Layer specification: Type 2 AAL*, ITU Telecommunication Standardization Sector Recommendation I.363.2, Nov. 2000.

[154] P. James-Roxby and S. A. Guccione, "Automated Extraction of Run-Time Parameterisable Cores from Programmable Device Configurations," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'00)*, Napa Valley, CA, United States, Apr. 17–19, 2000, pp. 153–161.

[155] J. Kairus, J. Forsten, M. Tommiska, and J. Skyttä, "Bridging the Gap Between Future Software and Hardware Engineers: A Case Study Using the

Nios Softcore Processor," in *Proceedings of the 2003 Frontiers in Education Conference (FIE'03)*, Boulder, Colorado, United States, Nov. 5–8, 2003, pp. F2F1–F2F5.

[156] P. Kannan and D. Bhatia, "Tightly Integrated Placement and Routing for FPGAs," in *Proceedings of the 11th International Workshop on Field-Programmable Logic and Applications (FPL'01)*, G. Brebner and R. Woods, Eds., Belfast, Northern Ireland, United Kingdom, Aug. 27–29, 2001, pp. 233–242.

[157] S. Kaptanoglu, G. Bakker, A. Kundu, I. Corneillet, and B. Ting, "A New High Density and Very Low Cost Reprogrammable FPGA Architecture," in *Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field-Programmable Gate Arrays (FPGA'99)*, Monterey, CA, United States, Feb. 21–23, 1999, pp. 3–12.

[158] T. Kean, "It's FPL, Jim – But Not as We Know It! Opportunities for the New Commercial Architectures," in *Proceedings of the 10th International Workshop on Field-Programmable Logic and Applications (FPL'00)*, R. W. Hartenstein and H. Grünbacher, Eds., Villach, Austria, Aug. 27–30, 2000, pp. 575–584.

[159] J. Khan, M. Handa, and R. Vemuri, "iPACE–V1: A Portable Adaptive Computing Engine for Real Time Applications," in *Proceedings of the 11th International Workshop on Field-Programmable Logic and Applications (FPL'01)*, G. Brebner and R. Woods, Eds., Belfast, Northern Ireland, United Kingdom, Aug. 27–29, 2001, pp. 69–78.

[160] M. Kirimura, Y. Takamoto, T. Mori, K. Yasumoto, A. Nakata, and T. Higashino, "Design and Implementation of FPGA Circuits for High Speed Network Monitors," in *Proceedings of the 12th International Workshop on Field-Programmable Logic and Applications (FPL'02)*, M. Glesner, P. Zipf, and M. Renovell, Eds., Montpellier, France, Sept. 2–4, 2002, pp. 393–403.

[161] E. Korpela, "Design of a Generic Reconfigurable Computing Platform," Master's thesis, Helsinki University of Technology, 2004.

[162] J. L. Kouloheris and A. E. Gamal, "FPGA Performance Versus Cell Granularity," in *Proceedings of the IEEE 1991 Custom Integrated Circuits Con-*

ference (CICC'91), San Diego, CA, United States, May 12–15, 1991, pp. 6.2.1–6.2.4.

[163] J. R. Koza, F. H. Bennett, III, J. L. Hutchings, S. L. Bade, M. A. Keane, and D. Andre, "Evolving Computer Programs using Rapidly Reconfigurable Field-Programmable Gate Arrays and Genetic Programming," in *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field-Programmable Gate Arrays (FPGA'98)*, Monterey, CA, United States, Feb. 22–25, 1998, pp. 209–219.

[164] R. Kress, "Configurable Computing: The Software Gap," in *Proceedings of the 1997 Fourth Reconfigurable Architectures Workshop (RAW'97)*, Geneva, Switzerland, Apr. 1, 1997.

[165] E. Kusse and J. Rabaey, "Low-Energy Embedded FPGA Structures," in *Proceedings of the 1998 International Symposium on Low Power Electronics and Design (ISLPED'98)*, Monterey, CA, United States, Aug. 10–12, 1998, pp. 155–160.

[166] X. Lai and J. L. Massey, "A Proposal for a New Block Encryption Standard," in *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology*, Aarhus, Denmark, May 21–24, 1990, pp. 389–404.

[167] A. Lawrence, A. Kay, W. Luk, T. Nomura, and I. Page, "Using Reconfigurable Hardware to Speed up Product Development and Performance," in *Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications (FPL'95)*, W. Moore and W. Luk, Eds., Oxford, United Kingdom, Aug. 29– Sept. 1, 1995, pp. 111–118.

[168] M.-H. Lee, H. Singh, G. Lu, N. Bagherzadeh, F. J. Kurdahi, E. M. C. Filho, and V. C. Alves, "Design and Implementation of the MorphoSys Reconfigurable Computing Processor," *The Journal of VLSI Signal Processing*, vol. 24, no. 2/3, pp. 147–164, Mar. 2000.

[169] G. G. Lemieux and D. M. Lewis, "Analytical Framework for Switch Block Design," in *Proceedings of the 12th International Workshop on Field-Programmable Logic and Applications (FPL'02)*, M. Glesner, P. Zipf, and M. Renovell, Eds., Montpellier, France, Sept. 2–4, 2002, pp. 122–131.

[170] M. Leong, O. Cheung, K. Tsoi, and P. H. W. Leong, "A Bit-Serial Implementation of the International Data Encryption Algorithm IDEA," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'00)*, Napa Valley, CA, United States, Apr. 17–19, 2000, pp. 122–131.

[171] P. Leong, M. Leong, O. Cheung, T. Tung, C. Kwok, M. Wong, and K. Lee, "Pilchard – A Reconfigurable Computing Platform with Memory Slot Interface," presented at the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01), Rohnert Park, CA, United States, Apr. 29– May 2, 2001.

[172] D. Lewis, V. Betz, D. Jefferson, A. Lee, C. Lane, P. Leventis, S. Marquardt, C. McClintock, B. Pedersen, G. Powell, S. Reddy, C. Wysocki, R. Cliff, and J. Rose, "The Stratix® Routing and Logic Architecture," in *Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field-Programmable Gate Arrays (FPGA'03)*, Monterey, CA, United States, Feb. 24–26, 2003, pp. 12–20.

[173] D. M. Lewis, D. R. Galloway, M. van Ierssel, J. Rose, and P. Chow, "The Transmogrifier-2: A 1 Million Gate Rapid-Prototyping System," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 2, pp. 188–198, June 1998.

[174] Y. Li and W. Chu, "Implementation of Single Precision Floating Point Square Root on FPGAs," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'97)*, Napa Valley, CA, United States, Apr. 16–18, 1997, pp. 226–232.

[175] G. J. Lipovksi and M. Malek, *Parallel Computing: Theory and Comparisons*. John Wiley & Sons, 1987.

[176] M. E. Louie and M. D. Erceg, "A Digit-Recurrence Square Root Implementation for Field Programmable Gate Arrays," in *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines (FCCM'93)*, Los Alamitos, CA, United States, Apr. 5–7, 1993, pp. 178–183.

[177] W. Luk and S. McKeever, "Pebble: A Language for Parameterized and Reconfigurable Hardware Design," in *Proceedings of the 8th International Workshop on Field-Programmable Logic and Applications (FPL'98)*, R. W.

Hartenstein and A. Keevallik, Eds., Tallinn, Estonia, Aug. 31– Sept. 3, 1998, pp. 9–18.

[178] P. Lysaght and J. Dunlop, "Dynamic Reconfiguration of FPGAs," in *More FPGAs, edited from the Oxford 1993 International Workshop on Field Programmable Logic and Applications*, W. R. Moore and W. Luk, Eds., Oxford, United Kingdom, 1993, pp. 82–94.

[179] J. MacBeth and P. Lysaght, "Dynamically Reconfigurable Cores," in *Proceedings of the 11th International Workshop on Field-Programmable Logic and Applications (FPL'01)*, G. Brebner and R. Woods, Eds., Belfast, Northern Ireland, United Kingdom, Aug. 27–29, 2001, pp. 462–472.

[180] R. Maestre, F. J. Kurdahi, M. Fernández, R. Hermida, N. Bagherzadeh, and H. Singh, "A Framework for Reconfigurable Computing: Task Scheduling and Context Management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 6, pp. 858–873, Dec. 2001.

[181] S.-T. Mak and K.-P. Lam, "Serial-Parallel Tradeoff Analysis of All-Pairs Shortest Path Algorithms in Reconfigurable Computing," in *Proceedings of the IEEE International Conference on Field–Programmable Technology (FPT'02)*, Hong Kong, China, Dec. 16–18, 2002, pp. 302–305.

[182] T. Makimoto, "The Rising Wave of Field Programmability," in *Proceedings of the 10th International Workshop on Field-Programmable Logic and Applications (FPL'00)*, R. W. Hartenstein and H. Grünbacher, Eds., Villach, Austria, Aug. 27–30, 2000, pp. 1–6.

[183] D. Maliniak, "Characteristics of Device Families: Tradeoffs Abound in FPGA Design," *Electronic Design*, vol. 51, no. 27, p. 48A, Dec. 4, 2004.

[184] M. S. Malone, *The Microprocessor: A Biography*. Springer-Verlag, 1995.

[185] W. H. Mangione-Smith, B. Hutchings, D. Andrews, A. DeHon, C. Ebeling, R. Hartenstein, O. Mencer, J. Morris, K. Palem, V. K. Prasanna, and H. A. Spaanenburg, "Seeking Solutions in Configurable Computing," *IEEE Computer*, vol. 30, no. 12, pp. 38–43, Dec. 1997.

[186] D. Manners, "Good News for Chip Firms... Or Is It?" *Electronics Weekly*, p. 18, Sept. 25, 2002.

[187] A. Marquardt, V. Betz, and J. Rose, “Speed and Area Tradeoffs in Cluster-Based FPGA Architectures,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 1, pp. 84–93, Feb. 2000.

[188] D. R. Martinez, T. J. Moeller, and K. Teitelbaum, “Application of Reconfigurable Computing to a High Performance Front-End Radar Signal Processor,” *The Journal of VLSI Signal Processing*, vol. 28, no. 1–2, pp. 63–83, May 2001.

[189] T. Maruyama, T. Funatsu, and T. Hoshino, “A Field-Programmable Gate-Array System for Evolutionary Computation,” in *Proceedings of the 8th International Workshop on Field-Programmable Logic and Applications (FPL’98)*, R. W. Hartenstein and A. Keevallik, Eds., Tallinn, Estonia, Aug. 31– Sept. 3, 1998.

[190] T. Maruyama and T. Hoshino, “A C to HDL Compiler for Pipeline Processing on FPGAs,” in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM’00)*, Napa Valley, CA, United States, Apr. 17–19, 2000, pp. 101–110.

[191] M. I. Masud and S. J. Wilton, “A New Switch Block for Segmented FPGAs,” in *Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications (FPL’99)*, P. Lysaght, J. Irvine, and R. Hartenstein, Eds., Glasgow, United Kingdom, Aug. 30– Sept. 1, 1999, pp. 274–281.

[192] MAX II: The Lowest–Cost CPLD Ever. Visited on the 20th of November, 2004. [Online]. Available: http://www.altera.com/products/devices/cpld/max2/mx2-index.jsp

[193] D. McCarty, D. Faria, and P. Alfke, “PREP® Benchmarks for Programmable Logic Devices,” in *Proceedings of the IEEE 1993 Custom Integrated Circuits Conference*, San Diego, CA, United States, May 9–12, 1993, pp. 7.7.1–7.7.6.

[194] J. S. McCaskill and P. Wagler, “From Reconfigurability to Evolution in Construction Systems: Spanning the Electronic, Microfluidic, and Biomolecular Domains,” in *Proceedings of the 10th International Workshop on Field-Programmable Logic and Applications (FPL’00)*, R. W.

Hartenstein and H. Grünbacher, Eds., Villach, Austria, Aug. 27–30, 2000, pp. 286–299.

[195] T. McDermott, P. Ryan, M. Shand, D. Skellern, T. Percival, and N. Weste, “A Wireless LAN Demodulator in a Pamette: Design and Experience,” in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM’97)*, Napa Valley, CA, United States, Apr. 16–18, 1997, pp. 40–45.

[196] G. McGregor and P. Lysaght, “Self Controlling Dynamic Reconfiguration: A Case Study,” in *Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications (FPL’99)*, P. Lysaght, J. Irvine, and R. Hartenstein, Eds., Glasgow, United Kingdom, Aug. 30– Sept. 1, 1999, pp. 144–154.

[197] J. T. McHenry, P. W.Dowd, F. A. Pellegrino, T. M. Carrozzi, and W. B. Cocks, “An FPGA–Based Coprocessor for ATM Firewalls,” in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM’97)*, Napa Valley, CA, United States, Apr. 16–18, 1997, pp. 30–39.

[198] M. McLoone and J. V. McCanny, “Single-Chip FPGA Implementation of the Advanced Encryption Standard Algorithm,” in *Proceedings of the 11th International Workshop on Field-Programmable Logic and Applications (FPL’01)*, G. Brebner and R. Woods, Eds., Belfast, Northern Ireland, United Kingdom, Aug. 27–29, 2001, pp. 152–161.

[199] S. McMillan and S. A. Guccione, “Partial Run-Time Reconfiguration Using JRTR,” in *Proceedings of the 10th International Workshop on Field-Programmable Logic and Applications (FPL’00)*, R. W. Hartenstein and H. Grünbacher, Eds., Villach, Austria, Aug. 27–30, 2000, pp. 352–360.

[200] L. McMurchie and C. Ebeling, “PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs,” in *Proceedings of the 1995 ACM/SIGDA Third International Symposium on Field-Programmable Gate Arrays (FPGA’95)*, Monterey, CA, United States, Feb. 12–14, 1995, pp. 111–117.

[201] C. Mead and L. Conway, *Introduction to VLSI Systems*. Addison-Wesley, 1980.

[202] G. Memik, S. O. Memik, and W. H. Mangione-Smith, "Design and Analysis of a Layer Seven Network Processor Accelerator Using Reconfigurable Logic," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'02)*, Napa Valley, CA, United States, Apr. 21–24, 2002, pp. 131–142.

[203] O. Mencer, "PAM-blox II: Design and Evaluation of C++ Module Generation for Computing with FPGAs," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'02)*, Napa Valley, CA, United States, Apr. 21–24, 2002, pp. 67–76.

[204] O. Mencer, Z. Huang, and L. Huelsbergen, "HAGAR: Efficient Multi-context Graph Processors," in *Proceedings of the 12th International Workshop on Field-Programmable Logic and Applications (FPL'02)*, M. Glesner, P. Zipf, and M. Renovell, Eds., Montpellier, France, Sept. 2–4, 2002, pp. 915–924.

[205] R. C. Minnick, "A Survey of Microcellular Research," *Journal of the ACM*, vol. 14, no. 2, pp. 203–241, 1967.

[206] T. Miyazaki, K. Shirakawa, M. Katayama, T. Murooka, and A. Takahara:, "A Transmutable Telecom System," in *Proceedings of the 8th International Workshop on Field-Programmable Logic and Applications (FPL'98)*, R. W. Hartenstein and A. Keevallik, Eds., Tallinn, Estonia, Aug. 31– Sept. 3, 1998, pp. 366–375.

[207] T. J. Moeller and D. R. Martinez, "Field Programmable Gate Array Based Radar Front-End Digital Signal Processing," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'99)*, Napa Valley, CA, United States, Apr. 21–23, 1999, pp. 178–187.

[208] G. E. Moore, "Cramming more Components onto Integrated Circuits," *Electronics*, vol. 38, no. 8, pp. 114–117, Apr. 19, 1965.

[209] C. Mulpuri and S. Hauck, "Runtime and Quality Tradeoffs in FPGA Placement and Routing," in *Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field-Programmable Gate Arrays (FPGA'01)*, Monterey, CA, United States, Feb. 11–13, 2001, pp. 29–36.

[210] E. S. Ochotta, P. J. Crotty, C. R. Erickson, C.-T. Huang, R. Jayaraman, R. C. Li, J. D. Linoff, L. Ngo, H. V. Nguyen, K. M. Pierce, D. P. Wieland, J. Zhuang, and S. S. Nance, "A Novel Predictable Segmented FPGA Routing Architecture," in *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field-Programmable Gate Arrays (FPGA'98)*, Monterey, CA, United States, Feb. 22–25, 1998, pp. 3–11.

[211] I. Page and W. Luk, "Compiling Occam into FPGAs," in *FPGAs, edited from the Oxford 1991 International Workshop on Field Programmable Logic and Applications*, W. R. Moore and W. Luk, Eds., Oxford, United Kingdom, 1991, pp. 271–283.

[212] P. Pan and C. L. Liu, "Technology Mapping of Sequential Circuits for LUT-based FPGAs for Performance," in *Proceedings of the 1996 ACM/SIGDA Fourth International Symposium on Field-Programmable Gate Arrays (FPGA'96)*, Monterey, CA, United States, Feb. 11–13, 1996, pp. 58–64.

[213] Z. Pan, S. Venkateshwaran, S. T. Gurumani, and B. Wells, "Exploiting Fine-Grain Parallelism of IDEA Using Xilinx FPGA," in *Proceedings of the 16th International Conference on Parallel and Distributed Computing Systems (PDCS-2003)*, Reno, NE, United States, Aug. 13–15, 2003.

[214] R. Payne, "Run-Time Parameterised Circuits for the Xilinx XC6200," in *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications (FPL'97)*, W. Luk, P. Y. K. Cheung, and M. Glesner, Eds., London, United Kingdom, Sept. 1–3, 1997, pp. 161–172.

[215] R. J. Petersen and B. L. Hutchings, "An Assessment of the Suitability of FPGA-Based Systems for Use in Digital Signal Processing," in *Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications (FPL'95)*, W. Moore and W. Luk, Eds., Oxford, United Kingdom, Aug. 29– Sept. 1, 1995, pp. 293–302.

[216] J. B. Peterson, R. B. O'Connor, and P. M. Athanas, "Scheduling and Partitioning ANSI-C Programs onto multi-FPGA CCM Architectures," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'96)*, Napa Valley, CA, United States, Apr. 17–19, 1996, pp. 178–187.

[217] K. Piromsopa, C. Arporntewan, and P. Chongstitvatana, “An FPGA Implementation of a Fixed–Point Square Root Operation,” in *Proceedings of the International Symposium on Communications and Information Technology 2001 (ISCIT 2001)*, ChiangMai, Thailand, Nov. 14–16, 2001, pp. 587–589.

[218] K. K. Poon, A. Yan, and S. J. Wilton, “A Flexible Power Model for FPGAs,” in *Proceedings of the 12th International Workshop on Field-Programmable Logic and Applications (FPL’02)*, M. Glesner, P. Zipf, and M. Renovell, Eds., Montpellier, France, Sept. 2–4, 2002, pp. 312–321.

[219] Quartus II Software. Visited on the 20th of November, 2004. [Online]. Available: http://www.altera.com/products/software/products/quartus2/qts-index.htm%l

[220] R. A. Quinnell, “Looking for the Sweet Spot,” *Electronics Design Chain*, vol. 3, pp. 16–20, Spring 2004.

[221] J. M. Rabaey, “Silicon Platforms for the Next Generation Wireless Systems – What Role Does Reconfigurable Hardware Play?” in *Proceedings of the 10th International Workshop on Field-Programmable Logic and Applications (FPL’00)*, R. W. Hartenstein and H. Grünbacher, Eds., Villach, Austria, Aug. 27–30, 2000, pp. 277–285.

[222] D. Robinson and P. Lysaght, “Verification of Dynamically Reconfigurable Logic,” in *Proceedings of the 10th International Workshop on Field-Programmable Logic and Applications (FPL’00)*, R. W. Hartenstein and H. Grünbacher, Eds., Villach, Austria, Aug. 27–30, 2000, pp. 141–150.

[223] J. Rose, R. J. Francis, D. Lewis, and P. Chow, “Architecture of Field-Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency,” *IEEE Journal of Solid-State Circuits*, vol. 25, no. 5, pp. 1217–1225, Oct. 1990.

[224] S. M. Scalera and J. R. Vázquez, “The Design and Implementation of a Context Switching FPGA,” in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM’98)*, Napa Valley, CA, United States, Apr. 15–17, 1998, pp. 78–85.

[225] H. Schmit, “Incremental Reconfiguration for Pipelined Applications,” in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing*

Machines (FCCM'97), Napa Valley, CA, United States, Apr. 16–18, 1997, pp. 47–55.

[226] H. Schmit and V. Chandra, "FPGA Switch Block Layout and Evaluation," in *Proceedings of the 2002 Tenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'02)*, Monterey, CA, United States, Feb. 24–26, 2002, pp. 11–18.

[227] B. Schneier, *Applied Cryptography*. John Wiley & Sons, 1996.

[228] S. D. Scott, A. Samal, and S. Seth, "HGA: A Hardware-Based Genetic Algorithm," in *Proceedings of the 1995 ACM/SIGDA Third International Symposium on Field-Programmable Gate Arrays (FPGA'95)*, Monterey, CA, United States, Feb. 12–14, 1995, pp. 53–59.

[229] Semic Research Corporation: ASIC Market Forecast and Analysis. Visited on the 20th of November, 2004. [Online]. Available: http://www.semico.com/studies/docs/toc303.pdf

[230] Semiconductor Industry Association: Total Semiconductor World Market Sales & Shares. Visited on the 20th of November, 2004. [Online]. Available: https://www.sia-online.org/downloads/market_shares_94-present.pdf

[231] S. Seng, W. Luk, and P. Y. Cheung, "Run-Time Adaptive Flexible Instruction Processors," in *Proceedings of the 12th International Workshop on Field-Programmable Logic and Applications (FPL'02)*, M. Glesner, P. Zipf, and M. Renovell, Eds., Montpellier, France, Sept. 2–4, 2002, pp. 545–555.

[232] M. Shand, "A Case Study of Algorithm Implementation in Reconfigurable Hardware and Software," in *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications (FPL'97)*, W. Luk, P. Y. K. Cheung, and M. Glesner, Eds., London, United Kingdom, Sept. 1–3, 1997, pp. 333–343.

[233] L. Shang, A. S. Kaviani, and K. Bathala, "Dynamic Power Consumption in Virtex®-II FPGA Family," in *Proceedings of the 2002 Tenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'02)*, Monterey, CA, United States, Feb. 24–26, 2002, pp. 157–164.

[234] M. Sheng and J. Rose, "Mixing Buffers and Pass Transistors in FPGA Routing Architectures," in *Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field-Programmable Gate Arrays (FPGA'01)*, Monterey, CA, United States, Feb. 11–13, 2001, pp. 75–84.

[235] N. Shirazi, W. Luk, and P. Y. Cheung, "Run-Time Management of Dynamically Reconfigurable Designs," in *Proceedings of the 8th International Workshop on Field-Programmable Logic and Applications (FPL'98)*, R. W. Hartenstein and A. Keevallik, Eds., Tallinn, Estonia, Aug. 31– Sept. 3, 1998, pp. 59–68.

[236] M. Shyu, G.-M. Wu, Y.-D. Chang, and Y.-W. Chang, "Generic Universal Switch Blocks," *IEEE Transactions on Computers*, vol. 49, no. 4, pp. 348–359, Apr. 2000.

[237] R. Sidhu and V. K. Prasanna, "Efficient Metacomputation Using Self-Reconfiguration," in *Proceedings of the 12th International Workshop on Field-Programmable Logic and Applications (FPL'02)*, M. Glesner, P. Zipf, and M. Renovell, Eds., Montpellier, France, Sept. 2–4, 2002, pp. 698–709.

[238] R. Sidhu, S. Wadhwa, A. Mei, and V. K. Prasanna, "A Self-Reconfigurable Gate Array Architecture," in *Proceedings of the 10th International Workshop on Field-Programmable Logic and Applications (FPL'00)*, R. W. Hartenstein and H. Grünbacher, Eds., Villach, Austria, Aug. 27–30, 2000, pp. 106–120.

[239] A. Singh and M. Marek-Sadowska, "Efficient Circuit Clustering for Area and Power Reduction in FPGAs," in *Proceedings of the 2002 Tenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'02)*, Monterey, CA, United States, Feb. 24–26, 2002, pp. 59–66.

[240] S. Singh, J. Rose, P. Chow, and D. Lewis, "The Effect of Logic Block Architecture on FPGA Performance," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 3, pp. 281–287, Mar. 1992.

[241] N. Sitkoff, M. Wazlowski, A. Smith, and H. Silverman, "Implementing a Genetic Algorithm on a Parallel Custom Computing Machine," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*

(FCCM'95), Napa Valley, CA, United States, Apr. 19–21, 1995, pp. 180–187.

[242] K. Skahill, *VHDL for Programmable Logic*. Addison-Wesley, 1996.

[243] M. J. S. Smith, *Application–Specific Integrated Circuits*. Addison-Wesley, 1997.

[244] Snort^{TM}: The Open Source Network Intrusion Detection System. Visited on the 20th of November, 2004. [Online]. Available: http://www.snort.org/

[245] S. Srikanteswara, R. C. Palat, J. H. Reed, and P. Athanas, "An Overview of Configurable Computing Machines for Software Radio Handsets," *IEEE Communications Magazine*, vol. 41, no. 7, pp. 134–141, July 2003.

[246] F.-X. Standaert, G. Rouvroy, J.-J. Quisquater, and J.-D. Legat, "A Methodology to Implement Block Ciphers in Reconfigurable Hardware and its Application to Fast and Compact AES RIJNDAEL," in *Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field-Programmable Gate Arrays (FPGA'03)*, Monterey, CA, United States, Feb. 24–26, 2000, pp. 216–224.

[247] ——, "Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs," in *Proceedings of the 5th International Workshop on Cryptographic Hardware and Embedded Systems (CHES'03)*, C. D. Walter, Çetin Kaya Koç, and C. Paar, Eds., Cologne, Germany, Sept. 8–10, 2003, pp. 334–350.

[248] A. Stansfield and I. Page, "The Design of a New FPGA Architecture," in *Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications (FPL'95)*, W. Moore and W. Luk, Eds., Oxford, United Kingdom, Aug. 29– Sept. 1, 1995, pp. 1–14.

[249] Synplify Pro®. Visited on the 20th of November, 2004. [Online]. Available: http://www.synplicity.com/products/synplifypro/index.html

[250] D. Takahashi, "Reconfigurable Computing Has Had Problems: Reconfigurable Chips Make a Comeback," *Electronic Business*, vol. 29, no. 11, p. 20, Aug. 2003.

[251] E. Tau, I. Eslick, D. Chen, J. Brown, and A. DeHon, “A First Generation DPGA Implementation,” in *Proceedings of the Third Canadian Workshop on Field–Programmable Devices*, Montreal, Canada, May 29– June 1, 1995, pp. 138–143.

[252] R. R. Taylor and S. C. Goldstein, “A High-Performance Flexible Architecture for Cryptography,” in *Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems (CHES’99)*, Çetin Kaya Koç and C. Paar, Eds., Worcester, MA, United States, Aug. 21–24, 1999, pp. 231–245.

[253] V. Tchoumatchenko, T. Vassileva, and P. Gurov, “An FPGA–Based Square–Root Co–Processor,” in *Proceedings of the 22nd EUROMICRO Conference*, Prague, Czech Republic, Sept. 2–5, 1996, pp. 520–525.

[254] R. Tessier and W. Burleson, “Reconfigurable Computing for Digital Signal Processing: A Survey,” *The Journal of VLSI Signal Processing*, vol. 28, no. 1, pp. 7–27, June 2001.

[255] The Open SystemC^{TM}Initiative. Visited on the 20th of November, 2004. [Online]. Available: http://www.systemc.org

[256] J. Torresen, “Possibilities and Limitations of Applying Evolvable Hardware to Real-World Applications,” in *Proceedings of the 10th International Workshop on Field-Programmable Logic and Applications (FPL’00)*, R. W. Hartenstein and H. Grünbacher, Eds., Villach, Austria, Aug. 27–30, 2000, pp. 230–239.

[257] A. Touhafi, W. Brissinck, and E. F. Dirkx, “Simulation of ATM Switches Using Dynamically Reconfigurable FPGAs,” in *Proceedings of the 8th International Workshop on Field-Programmable Logic and Applications (FPL’98)*, R. W. Hartenstein and A. Keevallik, Eds., Tallinn, Estonia, Aug. 31– Sept. 3, 1998, pp. 461–465.

[258] N. Tredennick, “Technology and Business: Forces Driving Microprocessor Evolution,” *Proceedings of the IEEE*, vol. 83, no. 12, pp. 1641–1652, Dec. 1995.

[259] ——, “Microprocessor-Based Computers,” *IEEE Computer*, vol. 29, no. 10, pp. 27–37, Oct. 1996.

[260] N. Tredennick and B. Shimamoto, “Go Reconfigure,” *IEEE Spectrum*, vol. 40, no. 12, pp. 36–40, Dec. 2003.

[261] ——, “The Inevitability of Reconfigurable Systems,” *ACM Queue: Tomorrow’s Computing Today*, vol. 1, no. 7, pp. 34–43, Oct. 2003.

[262] S. Trimberger, “Scheduling Designs into a Time–Multiplexed FPGA,” in *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field-Programmable Gate Arrays (FPGA’98)*, Monterey, CA, United States, Feb. 22–25, 1998, pp. 153–160.

[263] S. Trimberger, K. Duong, and B. Conn, “Architecture Issues and Solutions for a High–Capacity FPGA,” in *Proceedings of the 1997 ACM/SIGDA Fifth International Symposium on Field-Programmable Gate Arrays (FPGA’97)*, Monterey, CA, United States, Feb. 9–11, 1997, pp. 3–9.

[264] R. H. Turner, R. Woods, and T. Courtney, “Multiplier-less Realization of a Poly-phase Filter Using LUT-based FPGAs,” in *Proceedings of the 12th International Workshop on Field-Programmable Logic and Applications (FPL’02)*, M. Glesner, P. Zipf, and M. Renovell, Eds., Montpellier, France, Sept. 2–4, 2002, pp. 192–201.

[265] K. Underwood, “FPGAs vs. CPUs: Trends in Peak Floating–Point Performance,” in *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field-Programmable Gate Arrays (FPGA’04)*, Monterey, CA, United States, Feb. 22–24, 2004, pp. 171–180.

[266] T. Valtonen, J. Isoaho, and H. Tenhunen, “The Case for Fine-Grained Re–configurable Architectures: An Analysis of Conceived Performance,” in *Proceedings of the 12th International Workshop on Field-Programmable Logic and Applications (FPL’02)*, M. Glesner, P. Zipf, and M. Renovell, Eds., Montpellier, France, Sept. 2–4, 2002, pp. 816–825.

[267] D. E. Van den Bout, J. N. Morris, D. Thomae, S. Labrozzi, S. Wingo, and D. Hallman, “Anyboard: An FPGA-Based, Reconfigurable System,” *IEEE Design and Test of Computers*, vol. 9, no. 3, pp. 21–30, Sept. 1992.

[268] M. Vasilko, “Design Visualisation for Dynamically Reconfigurable Systems,” in *Proceedings of the 10th International Workshop on Field-Programmable Logic and Applications (FPL’00)*, R. W. Hartenstein and H. Grünbacher, Eds., Villach, Austria, Aug. 27–30, 2000, pp. 131–140.

[269] M. Vasilko and D. Ait-Boudaoud, "Optically Reconfigurable FPGAs: Is This a Future Trend?" in *Proceedings of the 6th International Workshop on Field-Programmable Logic and Applications (FPL'96)*, R. W. Hartenstein and M. Glesner, Eds., Darmstadt, Germany, Sept. 23–25, 1996, pp. 270–279.

[270] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte, "The MOLEN Polymorphic Processor," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1363–1375, Nov. 2004.

[271] K. Veenstra, B. Pedersen, J. Schleicher, and C. Sung, "Optimizations for a Highly Cost-Efficient Programmable Logic Architecture," in *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field-Programmable Gate Arrays (FPGA'98)*, Monterey, CA, United States, Feb. 22–25, 1998, pp. 20–24.

[272] N. Vemuri, P. Kalla, and R. Tessier, "BDD-Based Logic Synthesis for LUT-Based FPGAs," *ACM Transactions on Design Automation of Electronic Systems*, vol. 7, no. 4, pp. 501–525, Oct. 2002.

[273] J. Villasenor and B. Hutchings, "The Flexibility of Configurable Computing," *IEEE Signal Processing Magazine*, vol. 15, no. 5, pp. 67–84, Sept. 1998.

[274] Virtex-II Complete Data Sheet. Visited on the 20th of November, 2004. [Online]. Available: http://direct.xilinx.com/bvdocs/publications/ds031.pdf

[275] Virtex-II Series EasyPath Solutions. Visited on the 20th of November, 2004. [Online]. Available: http://www.xilinx.com/xlnx/xil_prodcat_product.jsp?title=v2_easypath

[276] B. von Herzen, "Signal Processing at 250 MHz using High–Performance FPGAs," in *Proceedings of the 1997 ACM/SIGDA Fifth International Symposium on Field-Programmable Gate Arrays (FPGA'97)*, Monterey, CA, United States, Feb. 9–11, 1997, pp. 62–68.

[277] J. von Neumann, *Theory of Self-reproducing Automata*, A. W. Burks, Ed. University of Illinois Press, 1966.

[278] J. E. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, and P. Boucard, “Programmable Active Memories: Reconfigurable Systems Come of Age,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 4, no. 1, pp. 56–69, Mar. 1996.

[279] S. Wahlstrom, “Programmable Logic Arrays — Cheaper by the Millions,” *Electronics*, vol. 40, no. 25, pp. 90–95, Dec. 1967.

[280] X. Wang and B. E. Nelson, “Tradeoffs of Designing Floating-Point Division and Square Root on Virtex FPGAs,” in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM’03)*, Napa Valley, CA, United States, Apr. 9–11, 2003, pp. 195–204.

[281] K. Weiß, C. Oetker, I. Katchan, T. Steckstor, and W. Rosenstiel, “Power Estimation Approach for SRAM-based FPGAs,” in *Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field-Programmable Gate Arrays (FPGA’00)*, Monterey, CA, United States, Feb. 9–11, 2000, pp. 195–202.

[282] M. A. Weiss, *Data Structures and Algorithm Analysis in C.* The Benjamin/Cummings Publishing Company, Inc., 1992.

[283] G. B. Wigley, D. A. Kearney, and D. Warren, “Introducing ReConfigME: An Operating System for Reconfigurable Computing,” in *Proceedings of the 12th International Workshop on Field-Programmable Logic and Applications (FPL’02)*, M. Glesner, P. Zipf, and M. Renovell, Eds., Montpellier, France, Sept. 2–4, 2002, pp. 687–697.

[284] S. J. E. Wilton, “SMAP: Heterogeneous Technology Mapping for Area Reduction in FPGAs with Embedded Memory Arrays,” in *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field-Programmable Gate Arrays (FPGA’98)*, Monterey, CA, United States, Feb. 22–25, 1998, pp. 171–178.

[285] S. J. E. Wilton, J. Rose, and Z. G. Vranesic, “Memory-to-Memory Connection Structures in FPGAs with Embedded Memory Arrays,” in *Proceedings of the 1997 ACM/SIGDA Fifth International Symposium on Field-Programmable Gate Arrays (FPGA’97)*, Monterey, CA, United States, Feb. 9–11, 1997, pp. 10–16.

[286] N. Wirth, “Hardware Compilation: Translating Programs into Circuits,” *IEEE Computer*, vol. 31, no. 6, pp. 25–31, June 1998.

[287] M. J. Wirthlin and B. L. Hutchings, “A Dynamic Instruction Set Computer,” in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM’95)*, Napa Valley, CA, United States, Apr. 19–21, 1995, pp. 99–107.

[288] ——, “Sequencing Run-Time Reconfigured Hardware with Software,” in *Proceedings of the 1996 ACM/SIGDA Fourth International Symposium on Field-Programmable Gate Arrays (FPGA’96)*, Monterey, CA, United States, Feb. 11–13, 1996, pp. 122–128.

[289] ——, “Improving Functional Density Through Run-Time Circuit Reconfiguration,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 2, pp. 247–256, June 1998.

[290] M. J. Wirthlin, B. L. Hutchings, and K. L. Gilson, “The Nano Processor: a Low Resource Reconfigurable Processor,” in *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines (FCCM’94)*, Napa Valley, CA, United States, Apr. 10–13, 1994, pp. 23–30.

[291] M. J. Wirthlin, B. L. Hutchings, and C. Worth, “Synthesizing RTL Hardware from Java Byte Codes,” in *Proceedings of the 11th International Workshop on Field-Programmable Logic and Applications (FPL’01)*, G. Brebner and R. Woods, Eds., Belfast, Northern Ireland, United Kingdom, Aug. 27–29, 2001, pp. 123–132.

[292] T. Wollinger, J. Guajardo, and C. Paar, “Security on FPGAs: State–of–the–Art Implementations and Attacks,” *ACM Transactions in Embedded Computing Systems*, vol. 3, no. 3, pp. 534–574, Aug. 2004.

[293] C. G. Wong, A. J. Martin, and P. Thomas, “An Architecture for Asynchronous FPGAs,” in *Proceedings of the IEEE International Conference on Field–Programmable Technology (FPT’03)*, Tokyo, Japan, Dec. 15–17, 2003, pp. 170–177.

[294] N.-S. Woo, “Revisiting the Ccascade Circuit in Logic Cells of Lookup Table Based FPGAs,” in *Proceedings of the 1995 ACM/SIGDA Third International Symposium on Field-Programmable Gate Arrays (FPGA’95)*, Monterey, CA, United States, Feb. 12–14, 1995, pp. 90–96.

[295] Xilinx Development/Reference Board Search. Visited on the 20th of November, 2004. [Online]. Available: http://www.xilinx.com/xlnx/xebiz/board_search.jsp

[296] Xilinx Investor Factsheet, 1Q'04. Visited on the 20th of November, 2004. [Online]. Available: http://media.corporate-ir.net/media_files/IROL/75/75919/factsheet/facts%heet.pdf

[297] Xilinx Press Release March 31, 2003: Xilinx Ships World's First 90nm Programmable Chips. Visited on the 20th of November, 2004. [Online]. Available: http://www.xilinx.com/prs_rls/xil_corp/033790nm_chips.html

[298] S. Xing and W. W. H. Yu, "FPGA Adders: Performance Evaluation and Optimal Design," *IEEE Design and Test of Computers*, vol. 15, no. 1, pp. 24–29, Jan.-Mar. 1998.

[299] J. Zambreno, D. Nguyen, and A. N. Choudhary, "Exploring Area/Delay Tradeoffs in an AES FPGA Implementation," in *Proceedings of the 14th International Workshop on Field-Programmable Logic and Applications (FPL'04)*, J. Becker, M. Platzner, and S. Vernalde, Eds., Antwerp, Belgium, Aug. 30– Sept. 1, 2004, pp. 575–585.

[300] J. Zhu and P. Sutton, "FPGA Implementations of Neural Networks – a Survey of a Decade of Progress," in *Proceedings of the 13th International Workshop on Field-Programmable Logic and Applications (FPL'03)*, P. Y. K. Cheung, G. A. Constantinides, and J. T. de Sousa, Eds., Lisbon, Portugal, Sept. 1–3, 2003, pp. 1062–1066.

[301] J. M. Zurada, *Introduction to Artificial Neural Systems*. West Publishing Company, 1992.