# Chapter 2
# The Relevance of Reconfigurable Computing

**Reiner Hartenstein**

**Abstract** This chapter introduces the highly promising future role of Reconfigurable Computing (RC) and emphasizes, that it is a critical survival issue for computing-supported infrastructures worldwide and stresses the urgency of moving RC from niche to mainstream. It urges acceptance of the massive challenge of reinventing computing, away from its currently obsolete CPU-processor-centric Aristotelian CS world model, over to a twin-paradigm Copernican model including and interlacing both, software and configware. It gives a flavor of the fundamentals of RC and the massive impact on the efficiency of computing it promises. Furthermore the chapter outlines the educational barriers we have to surmount and the urgent need for major funding on a global scale to run a world-wide mass movement, of a dimension at least as far reaching as the Mead-&-Conway-style VLSI design revolution in the early 1980s. The scenarios are similar: around 1980 an urgently needed designer population has been missing. Now a properly qualified programmer population is not existing. But this time the scenario is much more complex and the problem is more difficult, requiring not only a twin-paradigm approach for programming heterogeneous systems including both: many-core processors and reconfigurable accelerators, but also to find a solution to the parallelism crisis also called the "Programming wall". The presentation of recent R&D advances in RC, especially those ones funded by the EU, are also subject of all other chapters of this book.

R. Hartenstein (✉)
Fachbereich informatik, Technische Universität Kaiserslautern,
Traubenstr. 29, 76532 Baden-Baden, Germany
e-mail: reiner@hartenstein.de

J.M.P. Cardoso and M. Hübner (eds.), *Reconfigurable Computing: From FPGAs to Hardware/Software Codesign*, DOI 10.1007/978-1-4614-0061-5_2,

## 2.1 Introduction

Reconfigurable computing is an important survival issue of the world-wide computing infrastructures. This chapter stresses, that all our computer-based infrastructures worldwide are extremely important to avoid a massive crisis of the global and local economy (Sects. 2.1.1 and 2.2.3). This chapter warns of the future unaffordability of the electricity consumption of the entirety of all computers worldwide, visible and embedded (Sect. 2.2), and, that green computing, although important and welcome, is by far not sufficient to guarantee affordability and not at all to support further progress for future applications of high performance computing (Sect. 2.2.2).

Reconfigurable Computing, the second RAM-based machine paradigm offers drastic reduction of the electric energy budget and speedup factors by up to several orders of magnitude – compared to using the von Neumann paradigm [1], now beginning to loose its dominance. This chapter stresses the urgency of moving Reconfigurable Computing (RC) from niche to mainstream (Sect. 2.2.3) and urges, that we need a worldwide mass movement of a larger format than that of the VLSI design revolution around 1980, where only an urgently needed designer population has been missing [2–6]. This time a properly qualified programmer population is missing. But we need to push the envelope into two different directions. The VLSI design revolution has been the most effective project in the modern history of computing. But this time we need even more. A dual rail effort is needed for simultaneously developing the scene toward parallel programming for manycore architectures and to structural programming for reconfigurable computing (RC), as well as heterogeneous systems including the cooperation of both paradigms.

Currently the dominance of the basic computing paradigm is gradually wearing off with the growth of the area of RC applications – bringing profound changes to the practice of both, scientific computing and ubiquitous embedded systems, as well as new promise of disruptive new horizons for affordable very high performance computing. Due to RC also the desk-top personal supercomputer is near. To obtain the payoff from RC we need a new understanding of computing and supercomputing, as well as of the use of accelerators. For bridging the translational gap, the software/configware chasm, we need to think outside the box.

Section 2.3 tries to introduce (almost) non-expert readers to the flavor of RC. FPLAs, FPGAs, platform FPGAs, fine grain and coarse grain. Section 2.3.1 discusses the differences of RC applications in embedded systems and in supercomputing. It introduces the fundamentals of RC and the massive impact on the efficiency of computing it promises. Furthermore the chapter outlines the educational barriers we have to surmount and the urgent need for major funding on a global scale to run a world-wide mass movement, of a dimension reaching further than the Mead-&-Conway-style microelectronics revolution in the early 1980s. Section 2.2.2 illustrates the Reconfigurable Computing Paradox, and Sect. 2.2.3 Why von Neumann is so inefficient. Section 2.3 tries to convince the reader, why we need to reinvent computing. This chapter advocates to introduce a dual paradigm trans disciplinary education by using Configware Engineering as the counterpart of Software Engineering by new curricula in CS (Computer Science)

and CE (Computer Engineering) for providing an integrating dual paradigm mind set to cure severe qualification deficiencies of our graduates Sect. 2.3.3 tells us what problems we must solve, Sect. 2.3.1 the "parallel programming problem", Sect. 2.3.4 how to introduce RC and Sect. 2.3.5 describes the way toward a new world model of computing. It urges acceptance of the massive challenge to reinvent computing, away from its currently obsolete CPU-processor-centric Aristotelian CS world model, to a twin-paradigm Copernican model. For more details about the problem area see [7].

### *2.1.1 Why Computers Are Important*

Computers are very important for all of us. By many millions of people around the world computers are used in hundreds of application areas featuring ten-thousands of programs with millions of lines of code developed by thousands of man-years by investment volumes up to billions of dollars [8]. Computers running this legacy software are indispensable in our world [9]. Completing their tasks manually would require much more time, or, would be completely impossible, especially if networking is involved. Not only to maintain our global economy we must maintain these important infrastructures. However, threatening unaffordable operation cost by excessive power consumption is a massive future survival problem for our existing cyber infrastructures, which we must not surrender.

### *2.1.2 Unaffordable Energy Consumption by Computing*

It has been predicted that by the year 2030, if current trends continue, the worldwide electricity consumption by ICT (Information and Communication Technology) infrastructures will grow by a factor of 30 [10], reaching much more than the current total electricity consumption of the entire world for everything, not just for computing. The trends are illustrated by an expanding wireless internet, and by a growing number of internet users. as well as with tendencies toward more video on demand, HDTV over the internet, shipping electronic books, efforts toward more cloud computing and many other services. Other estimations claim, that already now the greenhouse gas emission from power plants generating the electricity needed to run the internet is higher than that of the total world-wide air traffic. For more predictions see [11]. The electricity bill is a key issue not only for Google, Microsoft, Yahoo and Amazon with their huge data farms at Columbia River [12]. That's why Google recently submitted an application asking the Federal Energy Regulatory Commission for the authority to sell electricity [13], and has a patent for water-based data centers, using the ocean to provide cooling and power (using the motion of ocean surface waves to create electricity) [14]. Already in the near

future the electricity bill of most facilities will here be substantially higher than the value of their equipment [15]. Already in 2005, Google's electricity bill was about 50 millions US-$ higher than the value of its equipment. Meanwhile the cost of a data center is calculated solely by the monthly power bill, not by the cost of hardware or maintenance [16]. As Google's employee L. A. Barroso said [17]: "The possibility of computer equipment power consumption spiraling out of control could have serious consequences for the overall affordability of computing." Power consumption estimations for an exascale supercomputer (1,018 calculations/s) like expected for about 2,018 range between 200 MW and 10 GW (the double of New York 16 million people energy consumption) [18, 19].

### *2.1.3 Peak Oil: Dig More Coal for Computing?*

Rapidly growing energy prices are predicted since the oil production has reached its peak by about the year 2009 [20–23]. Already currently 80% of crude oil is coming from decline fields (Fig. 2.1). However, the demand is growing because of developing standards of living in China, India, Brazil, Mexico and newly industrializing countries. We need at least "six more Saudi Arabias for the demand predicted for 2030" (Fatih Birol, Chief Economist IEA [20]). I believe that these predictions do not yet consider the growing electricity consumption of computers. Maybe, we will need ten more Saudi Arabias. About 50% of the shrinking oil reserves are under water [24]. In consequence of the Gulf of Mexico oil spill not all deepwater explorations will be allowed and the crude oil prices will go further up. All this will cause a massive future survival problem for our cyber infrastructures, which we must not surrender because this is an important global economy issue. Or, should we dig more coal [22]? It makes sense, to measure computing performance not just by MIPS (million instructions per second), but by MIPS/W instead.

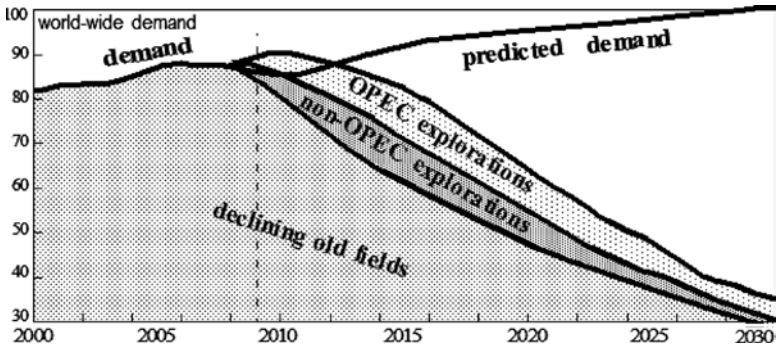


**Fig. 2.1** Beyond Peak Oil: massively declining future crude oil production [22]

### 2.1.4 Green Computing: Important, but Limited

Green Computing tends to use conservative methods to save energy by more efficient modules and components. For example LED flat panel displays need much less power than LCD-based displays. Also much more power-efficient power supply modules are possible. The potential to save power is substantially less than an order of magnitude: maybe, a factor of about 3–5. A scene separate from Green Computing is Low Power Circuit Design, now also called Low Power System on Chip Design (LPSoCD). Its most important conference series are about 30 years old: the PATMOS (oldest) and the ISLPED conference series.

Several methods are known for LPSoCD, such as: Active Body Bias (ABB), Adaptive Voltage Scaling (AVS), Dynamic Voltage Scaling (DVS), Multiple Supply Voltages (MSV), Multi-Threshold CMOS (MTCMOS), Power Gating (PG), Power Gating with Retention (RPG), etc. [16]. However, the order of magnitude of the benefit to be expected from this subarea LPSoCD is rather low. By MSV in using 3 Vdds the power reduction ratio at best is about 0.4 [16]. LPSoCD is a matter of ASIC design, e. g. of hardwired accelerator design. Only 3% of all design starts are ASIC designs (Fig. 2.2) with a trend leading further down. But in fact, low power design is also used for developing better power-efficient FPGAs – to the benefit of Reconfigurable Computing. But we need a much higher potential of saving energy because "Energy cost may overtake IT equipment cost in the near future" [9]. "Green Computing has become an industry-wide issue: incremental improvements are on track" [23], But "we may ultimately need revolutionary new solutions." [25] Let me correct this statement by "we will ultimately also need revolutionary solutions (like reconfigurable computing), since we need much higher efficiency."

### 2.1.5 Massively Saving Energy by RC

The idea of saving energy by using Reconfigurable Computing is not new [26, 27]. Being very important to massively reduce the energy consumption of computing, by up to several orders of magnitude, Reconfigurable Computing is extremely

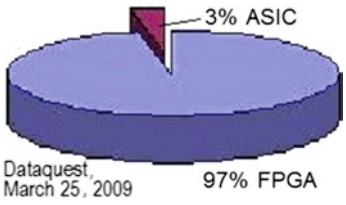


**Fig. 2.2** FPGA to ASIC design start ratio

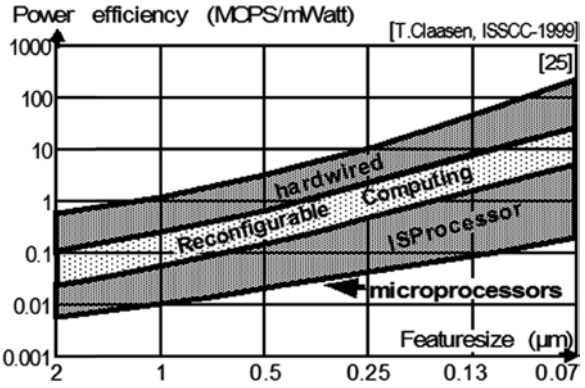


**Fig. 2.3** Better power efficiency by accelerators

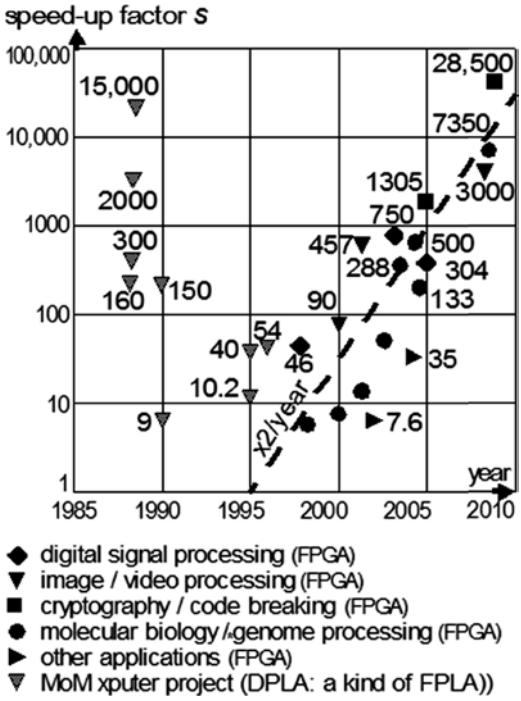


**Fig. 2.4** Speed-up factors

important for the survival of the world economy. Already a partial paradigm shift promises to save electricity by orders of magnitude. Dozens of papers (ref. in [28]) have been published on speed-ups obtained by migrating applications from software running on a CPU, over to configware for programming FPGAs [28]. It has been reported already more than a decade ago, that for a given feature size,

microprocessors using traditional compilers have been up to 500 times more power hungry than a pure hardware mapping of an algorithm in silicon [27] (Fig. 2.3). Speedup factors up to 4 orders of magnitude have been reported from software to FPGA migrations [26–49]. Here the energy saving factor is roughly about 10% of the speedup factor, i.e., still up to >3 orders of magnitude.

Figure 2.4 shows a few speedup factors picked up from literature, reporting a factor of 7.6 in accelerating radiosity calculations [46], a factor of 10 for FFT (fast Fourier transform), a speedup factor of 35 in traffic simulations [47]. A speedup by a factor of 304 is reported for an R/T spectrum analyzer [48]. For digital signal processing and wireless communication, as well as image processing and multimedia, speed-ups by 2 to almost 4 orders of magnitude have been reported. In the DSP area for MAC operations a speedup factor of 100 has been reported compared to the fastest DSP on the market (2004) [49]. Already in 1997, a speedup between 7 and 46 has been obtained over the fastest DSP [26]. In the multimedia area we find factors ranging from 60 to 90 in video rate stereo vision [34] and from 60 to 90 in real-time face detection [35], and of 457 for hyperspectral image compression [36]. In communication technology we find a speedup by 750 for UAV radar electronics [37]. For cryptography speed-ups by 3 to >5 orders of magnitude have been obtained. For a commercially available Lanman/NTLM Key Recovery Server [50] a speedup of 50–70 is reported. Another cryptology application reports a factor of 1,305. More recently for DES breaking a speed-up by × 28,514 has been reported [51] (Table 2.1).

For Bioinformatics applications [52] (also see [29]) speed-ups have been obtained by 2–4 orders of magnitude. Compared to software implementations sensational speed-up factors have been reported for software to FPGA migrations. A speedup of up to 30 has been shown in protein identification [30], by 133 [31] and up to 500 [32] in genome analysis. The Smith-Waterman algorithm, which is used for protein and gene sequence alignment, is basically a string-matching operation that requires a lot of computational power [52]. Here another study demonstrates speed-ups of 100x using Xilinx Virtex-4 hardware matched against a 2.2 GHz Opteron [53]. A speedup by 288 has been obtained with Smith-Waterman at the National Cancer Institute [33]. More recently a speed-up higher by more than an order of magnitude has been obtained here [45]. The CHREC project (supported by 24 industry partners [54]) reports running Smith-Waterman on a Novo-G supercomputer, a cluster of 24 Linux servers, each housing four Altera Stratix-III E260 FPGAs. According to this CHREC study, a four-FPGA node ran 2,665 times faster than a single 2.4 GHz Opteron core [55]. Another Smith-Waterman DNA sequencing application that would take 2.5 years on one 2.2 GHz Opteron is reported to take only 6 weeks for 150 Opterons running in parallel. Using 150 FPGAs on NRL's Cray XD1 (speedup by 43) is reported to further reduce this time to 24 h, which means a total speedup of 7,350X over a single Opteron [42]. These are just a few examples from a wide range of publications [29–51] reporting substantial speedups by FPGAs.

Recently not only energy saving factors have been reported, roughly one order of magnitude lower than the speed-up. More recently has been reported [51] for DES

**Table 2.1** Recent speed-up/power save data from software to configware migration [51]

| SGI Altix 4,700 w. RC 100 RASC vs. Beowulf cluster | Speed-up factor | Save factor Power | Cost | Size |
|---|---|---|---|---|
| DNA and protein sequencing | 8,723 | 779 | 22 | 253 |
| DES braking | 28,514 | 3,439 | 96 | 1,116 |

breaking (a crypto application): 28,500 (speed-up) vs. 3,439 (saving energy) and for DNA sequencing 8,723 (speed-up) vs. 779 (saving energy) etc. (Table 2.1). This paper also reports factors for saving equipment cost (up to x96) and equipment size (up to 1,116, see Table 2.1). No hangar full of equipment is needed when FPGAs are used in Scientific Computing. The Pervasiveness of FPGAs is not limited to embedded systems, but is also spread over practically all areas of scientific computing, where high performance is required and access to a supercomputing center is not available or not affordable. The desk-top supercomputer is near.

### *2.1.6 A Mass Movement Needed as Soon as Possible*

This subsection emphasizes that RC is a critical survival issue for computing-supported infrastructures worldwide and stresses the urgency of moving RC from niche to mainstream. It urges acceptance of the massive challenge of reinventing computing, away from its currently obsolete CPU-processor-centric Aristotelian CS world model, over to a twin-paradigm Copernican model. A massive software to configware migration campaign is needed. First this requires clever planning to optimize the effort versus its expected results. Which software packets should be migrated first. All this requires massive R&D and education efforts taking many years. Lobbying for the massive funding should be started right now. We should address politicians at all levels: community level, state level, national level, and European Union level.

To explain all this to politicians is very difficult. Since politicians always watch the sentiment of their voter population, we efficiently have to teach the public, which is a challenge. Without a strong tailwind from the media a successful lobbying seems to be almost successless. All this has to be completed as soon as possible, as long as we can still afford such a campaign. To succeed with such a challenging educational campaign the foundation of a consortium is needed for running an at least Europe-wide project.

## 2.2 Reconfigurable Computing

This section introduces a flavor of Reconfigurable Computing, its history, its more recent developments and, the massive impact on the efficiency of computing it promises. It is not easy to write this section such that it may be (almost) readable for non-experts – like the booklet FPGAs for Dummies which may help a little bit [56]. A Classical application for reconfigurable computing subsystems is the use as an accelerator to

support the CPU ("central processing unit"). According to the state of the art in the 1990s, having been the tail wagging the dog, this typically was and is a non-von-Neumann accelerator [57]. But we have to distinguish two kinds of such accelerators: made from hardwired logic or from field-programmable logic. These two kinds are distinguished by binding time of their functionality: (1) before fabrication for fixed logic or hardwired logic devices (HWD) vs. (2) after fabrication for (field-)programmable logic devices (PLD). The term "field-programmable" indicates, that by reconfiguration the functionality can be changed also at the users site by receiving new configuration code: from some memory, or, even over the internet.

First field-programmable blocks from the early 1980s have been so-called FPLAs featuring very area-efficient layout similar as known from ePROM memory for the price of being able to compute only Boolean functions in sum-of-product form. Very high speed-up could be obtained by matching hundreds of boolean expressions within a single clock cycle instead of computing them sequentially by a microprocessor. Together with a reconfigurable address generator [58] this brought a speed-up by factor up to 15,000 [59–63] for a grid-based design rule checker – already in the early 1980s. Via the multi project chip organization of the E.I.S. project such a FPLA (which was called DPLA) has been manufactured on a multi-project chip of the multi university E.I.S. project: the German contribution to the Mead-&-Conway VLSI design revolution. This DPLA has the capacity of 256 first FPGAs (field-programmable gate array) just appearing on the market (by Xilinx in 1984). This again demonstrates the massive area-inefficiency of FPGAs contributing to the Reconfigurable Computing Paradox (see Sect. 2.3.2) and the very early high speed-ups (Fig. 2.4).

The usual acronyms (Table 2.2) FPLA and FPGA are highly confusing being really not intuitive. From the straight-forward language feeling there does not seem to be any difference between "logic" in logic array (LA) and "gate" in Gate Array (GA). What is really different with FPGAs? In fact, FPGAs feature much more flexibility by introducing CLBs and routable wiring fabrics for interconnect between CLBs (Fig. 2.5). In contrast to FPLAs, the CLB in FPGAs allows for instance to select one of 16 logic functions from simple LUTs (look-up tables, Fig. 2.6). However, PLAs [64, 65] are not routable and allow only to implement Boolean functions in sum-of-product form.

Beyond such fine grained reconfigurability the progress of Moore's law leads to higher abstraction levels with "coarse-grained reconfigurability" featuring also CFBs (configurable function blocks), which may be adders, multipliers and/or many other functions. The next step is coarse-grained "platform FPGAs", which also include one or several microprocessors, like the PowerPC in earlier platform FPGAs from Xilinx.

### *2.2.1 Embedded Systems vs. Supercomputing*

A growing trend is the use of FPGAs in embedded systems: ERC (Embedded Reconfigurable Computing). Originally there has been a feeling that FPGAs are too slow, power-hungry and expensive for many embedded applications. This has

**Table 2.2** List of acronyms

| Acronym | Meaning | Acronym | Meaning | Acronym | Meaning |
|---|---|---|---|---|---|
| ALU | Arithmetic/logic unit | ESL | Electronic system-level design | MIPS | Mio instructionsp.second |
| ASM | Auto-sequencing memory | EU | European Union | MoPL | Map-oriented PL |
| CE | Computer engineering | FIR | Finite impulse response | PC | Personal computer |
| CS | Computer science | FPGA | Field-programmable GA | PL | Programming language |
| CHREC | NSF Center for High-Performance Reconfigurable Computing | FPL | Field-programmable logic | PLA | Programmable LA |
| CLB | Configurable logic block | FPLA | Field-programmable PLA | PLD | Programmable logic device |
| CPU | vN central processing unit | GA | (Routable) gate array | PROM | Programmable ROM |
| DNA | Deoxyribonucleic acid | GPP | General purpose processor | PU | Processing unit |
| DPA | Detapath array | GPGPU | General-purpose computation on graphics processing units | RAM | Random access memory |
| DPU | Detapath unit | GPU | Graphic processing unit | RC | Reconfigurable computing |
| DSP | Digital signal processing | HPC | High performance computing | rDPA | Reconfigurable DPA |
| EDA | Electronics design automation | HPRC | High performance RC | rDPU | Reconfigurable element |
| EPLA | E-Programmable LA | HWD | Hard-wired device | rE | Reconfigurable element |
| EPP | Extensible programmable platform | ICT | Information and communication technology | ROM | Read-only memory |
| ePROM | E-Programmable PROM | IEA | Intern'l energy agency | R&D | Research & development |
| ERC | Electrical rules checker | IT | Information technology | SMP | Symmetric multiprocessor |
| ESA | Embedded software automation | LA | (Compact) logic array | TM | Transactional memory |
| | | LUT | Look-up table | VLSI | Very large scale integrated (circuits) |
| | | MAC | Multiply/accumulate unit | vN | von Neumann |

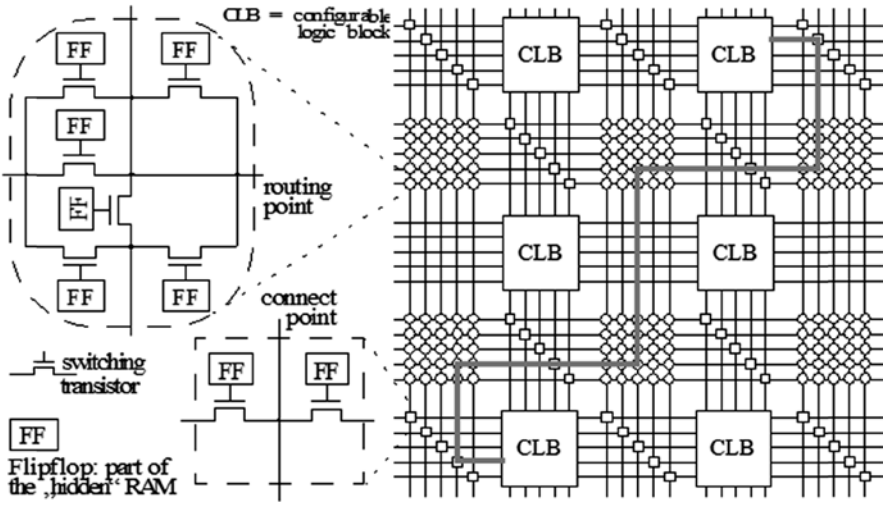


**Fig. 2.5** Interconnect fabrics example of a routable GA; grey line: example of one routed wire connecting 2 CLBs with each other

| # | $g_{ab}$ | | | | f(A, B) |
|---|---|---|---|---|---|
| | $g_{00}$ | $g_{01}$ | $g_{10}$ | $g_{11}$ | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | A and B |
| 2 | 0 | 0 | 1 | 0 | B disables A |
| 3 | 0 | 0 | 1 | 1 | A |
| 4 | 0 | 1 | 0 | 0 | A disables B |
| 5 | 0 | 1 | 0 | 1 | B |
| 6 | 0 | 1 | 1 | 0 | A exor B |
| 7 | 0 | 1 | 1 | 1 | A or B |
| 8 | 1 | 0 | 0 | 0 | not(A or B) |
| 9 | 1 | 0 | 0 | 1 | A coin B |
| 10 | 1 | 0 | 1 | 0 | no (B) |
| 11 | 1 | 0 | 1 | 1 | B implies A |
| 12 | 1 | 1 | 0 | 0 | not (A) |
| 13 | 1 | 1 | 0 | 1 | A implies B |
| 14 | 1 | 1 | 1 | 0 | not(A and B) |
| 15 | 1 | 1 | 1 | 1 | 1 |

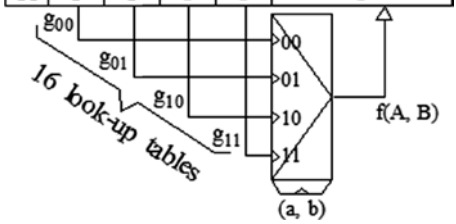


**Fig. 2.6** LUT example

changed. With low power and a wide range of small packages, particular FPGAs can be found in the latest handheld portable devices, including smartphones, eBooks, cameras, medical devices, industrial scanners, military radios, etc.

But embedded designers just don't like FPGAs with CPUs inside [66]. FPGAs in this context have been very much seen as a hardware engineer's domain, with the softies allowed in to play at some late stage [67]. Xilinx pre-announced a new family of devices "going beyond the FPGA". This "Extensible Programming Platform (EPP)" has a hardwired area with a top-end twin-core ARM Cortex-A9M processor unit and with a NEON multimedia processor, memory interfacing, peripherals, and a programmable fabric [67]. Instead of communicating across an FPGA, the two processors are connected by 2,500 wires, providing much capacity for an AMBA-AXI bus and other communications protocols. Xilinx was stressing that this approach recognizes the increasingly dominant role of software in systems and is pushing EPPs as a way to first define the system in software and then carry out software and hardware design in parallel.

EPPs make the processor the centre of the device with the programmable fabric as an extra. And this, argues Xilinx, now puts the software engineer first with the hardies following behind. In EPPs the FPGA logic and the CPU will be programmable separately. FPGA configuration will be handled by the processor(s) directly, not by a serial ROM. In other words, you have to tell the FPGA you want it configured. That's very un-FPGA-like [66]. That's EPP-like. The approach of using both a processor and programmable fabric allows design to start at high level and the system to be implemented as software [67].

Xilinx's first attempt at this was an FPGA with a processor inside. This time around, it's a processor with an FPGA grafted on. That's not just semantic hair-splitting: it's the big difference between these chips and the old ones. The new chips will boot up and run just like normal microprocessors, meaning there's no FPGA configuration required at all [68].

EPPs are a result of the new research topic Network-on-Chip (NoC) [69], which is a new paradigm for designing the on-chip hardware communication architecture based on a communication fabric, also including on-chip routers. NoC CAD tool flows also support mapping applications into NoC.

Apart from ERC (Embedded Reconfigurable Computing) we have another reconfigurable computing scene: HPRC (High Performance Reconfigurable Computing), This last one is a relatively new area, but has attracted a lot of interest in recent years, so much so that this entire new phrase has been coined to describe it [71]. HPRC uses FPGAs as accelerators for supercomputers [72]. Large HPC vendors are already supplying machines with FPGAs ready-fitted, or have FPGAs in their product roadmaps. What are the benefits of using FPGAs in HPC? Also here the first and most obvious answer is performance. HPC is renowned as that area of computing where current machine performance is never enough. A problem yet to be solved here is programmer productivity [73, 75]. It is an educational challenge, that programmers with the needed mix of skills are hardly available. Will FPGAs have a tough road ahead in HPC?

### 2.2.2 The Reconfigurable Computing Paradox

Technologically FPGAs are much less efficient than microprocessors [68, 70]. The clock speed is substantially lower. The routable reconfigurable wiring fabrics cause a massive wiring area overhead. There is also massive other overhead: reconfigurability overhead, since of 200 transistors e. g. maybe about five or even less of them (Fig. 2.9 in [70]) serve the application, whereas the other 195 are needed for reconfigurability (Fig. 2.6). Often there is also routing congestion, so that not all CLBs can be used, what causes further degradation of efficiency. Software to configware migration yield massive improvements in speed and power consumption, although FPGAs are a dramatically worse technology. We call this the "Reconfigurable Computing Paradox". by orders of magnitude better performance with a drastically worse technology? What is the reason? It's the von Neumann paradigm's fault. The next subsection goes into details.

### 2.2.3 Why von Neumann Is So Inefficient

The von Neumann paradigm has been criticized often [78–81]. Peter Newman had for 15 years each month the highly critical „computers at risk" back pages of Communications of the ACM [80]. Nathan's law (by Nathan Myhrvold, a former CTO of Microsoft) said that software is a gas, which fills any available storage space: on-chip memory, extra semiconductor memory located outside the processor chip, as well as hard disks. A lady (I forgot her name) said that it even fills the internet. Nicklaus Wirth's pre-manycore interpretation of Moore's law is, that "software is slowing faster than hardware is accelerating" [78].

Why is von Neumann so inefficient? It is the von Neumann syndrome [82] caused by the fact, that instruction streams are very memory-cycle-hungry. We can distinguish two different reasons: algorithmic complexity required by the von Neumann paradigm, and, architectural issues. There are also other attempts to explain at least particular symptoms of this syndrome (Fig. 2.8 [83]) [84]. A well known architectural problem is the memory wall [85, 86] (Fig. 2.7): the access time to RAM outside the processor chip is slower by a factor of about 1,000, than to on-chip memory [86]. This difference is growing by 50% every year. It is a dramatic software engineering issue, that multiple levels of instruction stream overhead leads to massive code sizes which hit the memory wall [86]. However, rDPUs and rDPAs do not suffer from Nathan's law, since at run time no instruction streams are running through.

How data are moved is a key issue. CPUs usually move data between memories requiring instruction streams (first line, Table 2.3). This means the movement of data is evoked by execution of instructions due to the von Neumann paradigm. Also the execution of operations inside a CPU requires reading and decoding of instructions (Fig. 2.8 gives an idea of the overhead of the main components for contemporary CPUs). However, after a full migration to static reconfigurable computing an

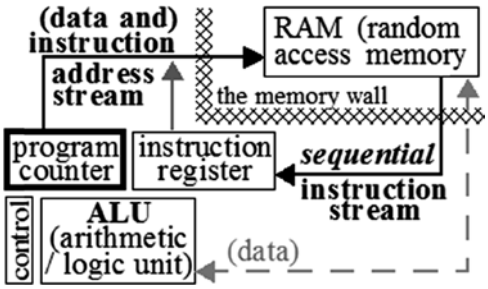


**Fig. 2.7** von Neumann principles

**Table 2.3** Twin paradigm fundamental terminology

| # | Term | Controlledby | Machine paradigm | State register Type | Location |
|---|---|---|---|---|---|
| 1 | Software | Instruction streams | von Neumann | Program counter | in CPU |
| 2 | Configware | (Configuration memory) | None | None | (Hidden) |
| 3 | Flowware | Reconfigurable address generator | Data stream machine | Data counter | In ASM memory block |

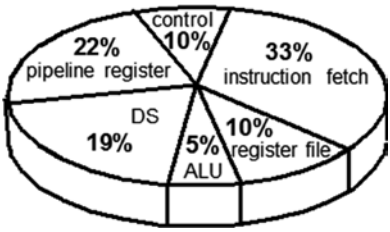


**Fig. 2.8** All but ALU is overhead: ×20 efficiency [83]

algorithm is run by data streams only. Instead of a hardwired program counter reconfigurable data counters are used which do not require instruction sequences for address computation.

Also, how data are moved inside the data paths is a key issue, and pipe network structures to interconnect rDPUs avoid moving data through memory blocks because data are moved directly from DPU to DPU [87]. This means, that operation execution inside a DPU (not having a program counter) is "transport-triggered" (second line, Table 2.3). It is triggered via handshake by the arrival of the data item, not needing an instruction to call it. Not looking at dynamically reconfigurable systems ([88] only for advanced courses) we see, that reconfigurable fabrics don't perform any instruction sequencing at run time.

Beyond such fine grained reconfigurability the progress of Moore's law leads to higher abstraction levels with "coarse-grained reconfigurability" featuring also

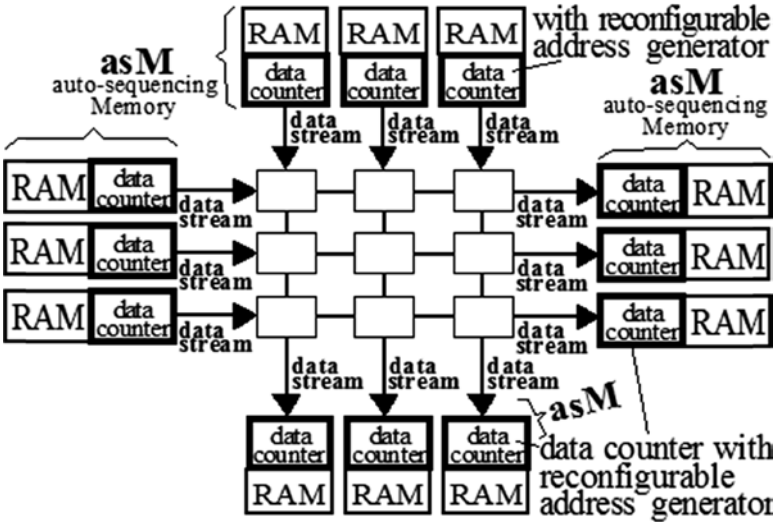


**Fig. 2.9** Example for data stream processor principles

CFBs (configurable function blocks), which may be adders, multipliers and/or many other functions. The next step is coarse-grained "platform FPGAs", which also include one or several microprocessors, like the PowerPC in earlier platform FPGAs from Xilinx.

But by a migration sometimes also the amount of data streams may be minimized by changing the algorithm. Here an illustration example for reducing the algorithmic complexity is given by the migration of the well-known O(n2) complexity bubble sort algorithm away from von Neumann. The algorithmic complexity turns from O(n2) into O(n) [8]. In a similar manner, other well-known algorithmic methods can be transformed to explore parallelism and locality, like in dynamic programming as presented in [88]. The combination of these effects leads to massive speed-up and massive saving of energy.

Of course, the data entering or leaving such an array (Fig. 2.9) have to be stored. The datastream machine paradigm uses auto-sequencing Memory blocks (asM). Each asM has a reconfigurable address generator and data counter inside. so that no instruction streams are needed for address computation. All this data streams can be programmed via data-imperative languages [90], being a kind of sisters of classical instruction-imperative programming languages. Data-imperative languages are easy to teach since both classes of imperative languages use the same primitives (Table 2.4). But there is one exception: data-imperative languages also support parallelism inside loops (Table 2.4). This also contributes to the benefit by reconfigurable computing. The simultaneous use of both classes of languages we call "twin-paradigm approach" (Table 2.4 and Fig. 2.10).

**Table 2.4** Imperative language twins: program counter vs. data counters

| | Instruction stream languages | Data stream languages |
|---|---|---|
| sequencing primitives | Read next instruction goto (instruction address) jump (to instruction address) instruction loop instruction loop nesting escapes instruction stream branching | Read next data item goto (data address) jump (to data address) data loop data loop nesting escapesdata stream branching |
| asymmetry | *Loops not internally parallel* | *Yes: loops internally parallel* |

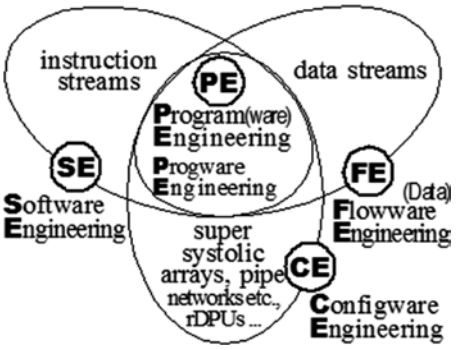


**Fig. 2.10** New CS world model image

## 2.3 Why We Need to Reinvent Computing

We cannot afford to relinquish RC. We will urgently need this technology to cope with threatening unaffordable operation cost by excessive power consumption of the entirety of all von Neumann computers world-wide. We need to migrate many application packages from software over to configware. This is a challenge to reinvent computing to provide the qualifications needed since disruptive developments in industry have caused the many-core programming crisis. Intel's cancellation the Tejas and Jayhawk processors indicated in May 2004 the end of frequency scaling's dominance to improve performance. "Multicore computers shift the burden of software performance from VLSI designers over to software developers" [89]. For Gary Smith from GS-EDA the three biggest disruptions are not only (1) many-core silicon, but also (2) non-vN architectures, and (3) parallel software, and the center of gravity shifts from EDA to programming (not "software" how Gary named it [93], compare Table 2.3 and Fig. 2.10).

To use manycore we need to rewrite our software: our biggest problem. RTL verification moves up to ESL. EDA Industry and ESA need to merge. Especially for Embedded Software Automation (ESA) we need tools to develop parallel software. He calls for an approach of using both, processor and programmable fabric, allows design to start at high level and the system to be implemented as progware (programware, see Fig. 2.10), e. g. with tools like LabView, MatLab, or others.

From Xilinx and ARM hardware and software IP and tool chains are available useful to speed up time-to-market and reducing risk. It's easier to take advantage of accelerators than to try to integrate and program more processors.

To rewrite the software the qualified programmer population is not existing: a huge challenge to provide new educational approaches to qualify for heterogeneous systems including both, parallel software and configware. This requires much more than just bridging the traditional hardware/software chasm in education [91]. We need robust and fast implementations of adequate compilers and design tools, e. g. automated by formal techniques based on rewriting [92]. The biggest payoff will come from Putting Old ideas into Practice and teaching people how to apply them properly [94]. Dimensionality-rooted scaling laws favor reconfigurable spatial computing over temporal computing. Time to space mapping even dates back to the early 1970s and even the late 1960s, years before the first hardware description languages came up [91, 95, 96]. "The decision box (in the flow chart) turns into a demultiplexer. This is so simple! Why did it take 30 years to find out?" [97].

The impact is a fascinating challenge to reach new horizons of computer science. We need new generations of talented innovative scientists and engineers to start the second history of computing.

### *2.3.1 The "Parallel Programming Problem"*

Parallel computer programs are difficult to write: performance is affected by data dependencies, race conditions, synchronization, and parallel slowdown. The problem is: how to optimize parallel Computing despite Amdahl's Law? The "parallel programming problem" has been addressed high performance computing for more than 25 years with very disappointing results [100–102]. Programming languages research has stalled for several decades [103]. Informal approaches are not working. For the multi-core era, we must adopt a systematic approach informed by insight into how programmers think [104].

I do not agree. We have to teach them how to think. We have to teach programmers how to "think parallel", how to find concurrent tasks, how to synchronize free from deadlock and race conditions, how to schedule tasks at the right granularity, and, how to solve the data locality problem. Perhaps a new visual programming paradigm is required [68]. What are the right models (or abstractions) to avoid typical problems [105]: multi-core version applications running slower, problems with race conditions, and strategies for migrating code to multi-core. We see a promising new horizon: a model-based twin-paradigm methodology to master the hetero of all 3: Single-core, Multicore, and Reconfigurable Computing.

We need a different way of thinking. "The shift to multicore processor architectures really is stressing existing programming models" said Richard C. Murphy at Sandia aiming at redesigning memory systems to move computation as close to memory as possible to eliminate the traditional load-store approach where big

systems use more resources moving data around than for actually computing [106]. Since Linpack doesn't measure performance for actual problems in many application areas, Sandia has proposed Graph500 as a new rating system for testing skills in analyzing large, graph-based structures that link huge numbers of data points [107]. Studies show that moving data around (not computations) will be the dominant energy problem.

### *2.3.2 Why FPGAs Should Win*

Most ASIC design world-wide has stopped [108]. Only 3% of all design starts are ASIC designs (Fig. 2.3) from mega-funded companies with gigantic-volume products that can afford latest generation custom SoC development, and niche players that continue doing ASIC design with older-generation processes. The enormous 97% gap can be filled best by hybrid FPGA/hard-core devices – by FPGA companies with the required technology and infrastructure to sell and support them.

"FPGAs have become incredibly capable with respect to handling large amounts of logic, memory, digital-signal-processor (DSP), fast I/O, and a plethora of other intellectual property (IP)" [109]. At 28-nm, FPGAs deliver the equivalent of a 20–30-million gate application-specific integrated circuit (ASIC). At this size, traditionally used FPGA design tools begin to break down and can no longer design and verify these devices in a reasonable amount of time.

This positions FPGA companies in the best place they've ever been – a place from where they could capture huge segments of the standard parts and ASSP business with semi-standard parts that include FPGA fabric for application-specific customization. Instead of today's still-very-general-purpose FPGAs, we'll see more devices with a narrower application focus without too much general-purpose overhead on the die. With each passing process generation, the cost of that overhead shrinks, and the cost of developing the traditional alternative solutions goes up.

Industry abandoned the "pure" FPGA [110]. Countless failed FPGA start-ups have proven that the magic is not in the fabric. The real keys are tools, IP, and support for enabling the customer/designer to get the fabric to do what they want as easily as possible, and with minimal risk.

It turns out that the solution is a mixture of FPGA fabric and hard logic coming from the FPGA companies. Instead of putting FPGA fabric in our custom SoC designs, we are getting custom SoC in our FPGAs. Today's FPGA are hybrid parts with optimized hardwired cells (like multipliers, memory, IO and even processors) and FPGA fabric living on the same die.

It is an important factor that there is an order of magnitude more software engineers than hardware guys. Usually it is the software community that selects the processor, not the hardware team. To gain the approval of software engineers the FPGA vendors realized that promoting "a processor with FPGA accelerators" is more attractive than an "FPGA with a processor inside".

The FPGA business and the processor business looks like "chip" business, but actually are more "tools, software, IP, and services" businesses. With the coming together of the embedded processing world [111] and the FPGA world, we will see if FPGA companies like Xilinx can be convincing enough in their ability to support the embedded software developer, or if companies like Intel can be convincing enough in their ability to support the FPGA designer.

"Customers are increasingly turning to FPGAs and expert 3rd party providers to design progressively complex products within shrinking time to market budgets." As the industry is developing more complex designs on programmable solutions, competent and trusted providers are required to deliver key IP, software, and services to meet the tight delivery schedules of today's system companies and to allow customers to find the right qualified 3rd party provider easier and faster than before – avoiding, that for SoC designers it's a nightmare using IPs delivered by 3rd parties or internal IP teams? An Electronics IP core, a semiconductor intellectual property core, or IP block is a reusable unit of logic, cell, or chip layout design that is the (legal) intellectual property of one party.

### *2.3.3 Problems We Must Solve*

Furthermore this chapter outlines the educational barriers we have to surmount and the urgent need for major funding on a global scale to run a world-wide mass movement, of a dimension as far reaching as the Mead-&-Conway-style microelectronics revolution in the early 1980s. Problems We Must Solve:

1. A mass migration from software to configware for the benefit of massively saving energy, of much higher performance, and, of gaining high flexibility.
2. developing a most promising migration priority list.
3. to reeducating the programmer population for such a mass movement campaign [68], and upgrading our highly obsolete curricula for three reasons:
   (a) to realize that parallel programming qualifications are a must,
   (b) to resolve the extreme shortage of programmers qualified for RC, and
   (c) twin paradigm programming skills are a must to program hetero systems (like modern FPGAs featuring all 3: reconfigurable fabrics, hardwired function blocks, and CPUs).

As a consequence we need innovative undergraduate programming courses [98] which also teach a sense for locality, not only needed for classical parallel programming, is already coming along in RC with time to space mapping required to structurally map an application to the datastream side of the twin paradigm approach. This means, that teaching the structural programming of RC also exercises the sense of locality needed for traditional parallel programming. The extension of the non-sequential part of education should be optimized not to scare away undergraduate students. Twin-paradigm lab courses should be model-based, may be MathWorks-supported, mainly at the abstraction level of pipe networks [99].

### 2.3.4 How to Introduce Reconfigurable Computing

Since software has to be rewritten anyway, this is the occasion for the twin-paradigm approach to massively reduce the energy consumption of our computing infrastructures.

Meanwhile FPGAs are also used everywhere for high performance in scientific computing, where this is really a new computing culture – not at all a variety of hardware design. Instead of H/S codesign we have here software/configware co-design (SC co-design), which is really a computing issue. This major new direction of developments in science will determine how academic computing will look in 2015 or even earlier. The instruction-stream-based mind set will loose its monopoly-like dominance and the CPU will quit its central role – to be more an auxiliary clerk, also for software compatibility issues.

An introduction to Reconfigurable Computing (RC) [90, 112, 113] should regard the background to be expected from the reader. This chapter of the book mainly addresses a bit IT-savvy people in the public and its mass media, as well as "software engineers". Here an introduction is difficult, since in both communities people typically know nothing or almost nothing about RC. To move RC from its niche market into mainstream massive funding is needed for R&D and to reinvent programming education. To yield the attention of media and the politicians we need a highly effective campaign by mass media.

RC should urgently become mainstream. Several reasons have prevented RC from truly becoming mainstream [114]. The execution model is inherently different from the traditional sequential paradigm were we can reason about state transition sequences much better than in a hardware or a concurrent execution model. As a consequence, the development and validation of tools is substantially a traditional hardware mind set.

Tools are limited and above all fairly bridle. This means programmers must master the details of not only software development but also of hardware design. Such a set of skills is also not taught as part of major electrical engineering courses severely constraining the pool of engineering with the "right" mindset for programming RC to a selected few. Moreover the recent evolution of FPGAs and to some extent coarse-grain RC architecture make programmer and performance portability difficult at best.

One of the objectives of the REFLECT project (Chap. 11) is lowering the barrier of access of RC to the average programmers, by retaining the "traditional" imperative programming mindset in a high-level language such as MATLAB and rely on the concepts of Aspects to provide a clean mechanism (at the source code level) for the advanced user to provide key information for a compilation and synthesis tool to do a good job in mapping the computation to hardware. The approach should be by no means fully automatic [114]. Instead, we have the programmer involved but controlling the high-level aspects of the mapping while the tools takes care of the low-level, error-prone steps.

We extend the "traditional" imperative programming mindset (for software) by a twin-paradigm imperative mind (subject of Sect. 2.4.4.) also including an imperative datastream programming methodology (for "flowware" – for terminology see Table 2.3) [90]. We obtain an almost fully symmetric methodology: the only asymmetry is intra-loop parallelism, possible for data streams, however not for instruction

streams (Table 2.4). The semantic difference of these machine paradigms is the state register: the program counter (located with the ALU) for running the instruction streams in executing software, and data counter(s) (located in memory block(s) [60, 61]) for running data streams in executing flowware.

### *2.3.5 Toward a New World Model of Computing*

The traditional CPU-centric world model of the CS world is obsolete. It resembles the old Aristotelian geo-centric world model. Its instruction-stream-based software-only tunnel view perspective hides structural and data stream aspects – massively threatening the progression of system performance, where we have to confront a dramatic capability gap. We need a generalized view, comparable to the Copernican world model not being geo-centric. We need a hetero model which also includes structures and data streams and supports time to space mapping, since scaling laws favor reconfigurable spatial computing over temporal computing. Exercising time to space mapping, also by programming data streams and by software to configware migration, provides important skills: e. g. locality awareness, understanding and designing efficient manycore architectures and their memory organization being essential to cope with bottlenecks caused by bandwidth problems.

This new direction has not yet drawn the attention of the curriculum planner within the embedded systems scene. For computer science this is the opportunity of the century, of decampment for heading toward new horizons, and, to preserve the affordability of its electricity consumption. This should be a wake-up call to CS curriculum development. Each of the many different application domains has only a limited view of computing and takes it more as a mere technique than as a science on its own. This fragmentation makes it very difficult to bridge the cultural and practical gaps, since there are so many different actors and departments involved. We need the new CS world model to avoid the capability gap caused by that fragmentation. Computer Science should take the full responsibility to merge Reconfigurable Computing into CS curricula for providing Reconfigurable Computing Education from its roots. CS has the right perspective for a trans-disciplinary unification in dealing with problems, which are shared across many different application domains. This new direction would also be helpful to reverse the current down trend of CS enrolment.

Not only for the definition of the term “Reconfigurable Computing” (RC), it makes sense to use a clear terminology – not only to improve education about how to reinvent computing. It is a sluttish use of terms if “soft” or “software” is used for everything, which is not hardware. The term “software” should be used only for instruction streams and their codes. However, we generalize the term “programming” (Fig. 2.6) such, that procedural programming (in time domain) creates sequential code, like instruction streams (software), or data streams, which we call “flowware”, and, that “structural programming” (programming in space) creates “structural code”, which we call “configware”, since it can be used for the configuration of FPGAs (Field-Programmable Gate Arrays) or other reconfigurable platforms. Summary: Table 2.3.

**Table 2.5** Confusing terms which should not be used

| Term | Once introduced for |
|---|---|
| Dataflow | Indeterministic exotic machines |
| Firmware | Nested von Neumann machines |
| Microcode | |
| Microprogram | |
| Software or "soft" | No use other than for instruction streams |

This established terminology reveals (see Table 2.3 for the terms we should use and Table 2.5 for the terms that usually make some confusion), that a software to configware migration means a paradigm shift, away from the traditional programmer's CPU-centric world model of computing, resembling the geo-centric Aristotelian world model. To reinvent computing we need a multi paradigm hetero system world model of computing science (Fig. 2.9), which models the co-existence of, and the communication between: (1) the traditional imperative software programming language mind set with the CPUs running by software (instruction streams), (2) the reconfigurable modules to be structurally programmed by configware, and (3) an imperative datastream programming language mind set with [90] data stream machines programmed by flowware for generating and accepting data streams (asM in Table 2.3 stands for "auto-sequencing Memory", also containing the data counter inside a reconfigurable address generator). We obtain an almost fully symmetric methodology: the only asymmetry is intra-loop parallelism, possible for data streams, however not for instruction streams (Table 2.4). The semantic difference of these machine paradigms is the state register: the program counter (located with the ALU) for running the instruction streams in executing software, and the data counter(s) (located in memory block(s) [60–66]) for running data streams in executing flowware.

Figure 2.10 illustrates this triple-paradigm "Copernican" world model replacing the von-Neumann-only-centric obsolete "Aristotelian" narrow tunnel view perspective of classical software engineering, which hides almost everything, which is not instruction-stream-based. (The term "supersystolic" in Fig. 2.10 stands for the generalization of the systolic array: non-linear and non-uniform pipes are allowed like spiral, zigzag and any excessively irregular shapes.) This generalized model will help us to come up with a new horizon of programmer education which masters overcoming the hardware/software chasm, having been a typical misconception of the ending first history of computing. The impact is a fascinating challenge to reach new horizons of research and development in computer science. We need a new generation of talented innovative scientists and engineers to start the beginning second history of computing, not only for the survival of our important computer-based cyber infrastructures, but also for developing and integrating exciting new innovative products for the transforming post PC era global information and communication markets [90].

Not yet discussed in this paper is the accelerator role of GPUs (graphics processors [115]) which for some authors seem to be the FPGA's competitor w. r. to speed-up and power efficiency [116–118]. Meanwhile the very busy hype on the accelerator use of GPGPU seems to be over-exaggerated [117]. FPGAs from a new

Xilinx 28 nm high-performance, low-power process, developed by Xilinx and TSMC-optimized for high performance & low power are massively better off than GPUs. E.g., for the Smith-Waterman algorithm the following normalized performance is reported: 584 for FPGA, 25 for GPU, and, 1 for GPP [119]. Since a compute-capable discrete GPU can draw much more than 200 W, other authors call this massive power draw a serious roadblock to the adoption, not only in embedded systems, but even for data centers [120].

But going hetero by interweaving instruction stream parallelism and structural parallelism is a massive challenge requiring to master many difficult problems. The existence of thousands of languages did not prevent a stall of language research in the past two decades [104]. Being speaker in seven tutorials at Supercomputing 2010 [121] Tim Mattson of Intel complaints about what he calls "choice overload" and calls to arms. Another design tool problem is hitting the moving target of the complex value chain in SoC design: the rapidly growing segment of the electronics industry called "Electronics Intellectual Property" or "Electronics IP", where currently the designers have a nightmare using IPs delivered by 3rd parties or internal IP teams [122]. Masses of highly qualified new kinds of jobs must be created to meet the fascinating challenges of reinventing computing sciences, following the wide horizon of the new world model [121].

## 2.4 Conclusions

This chapter has emphasized that Reconfigurable Computing (RC) is a critical survival issue for computing-supported infrastructures worldwide and has stressed the urgency of moving RC from niche to mainstream. Since a qualified programmer population does not exist we need to use Reconfigurable Computing to Reinvent Computing (R2R) and to Rewrite Textbooks for R2R (RT4R2R), and many of us should become Reinvent Computing Evangelists (RCE). We urgently need a worldwide mass movement of R&D and education to be more massively funded and supported than the Mead-&-Conway VLSI design revolution in the early 1980s, which so far has been the most effective project in the history of modern computing science. This chapter urges acceptance of the massive challenge of reinventing computing, away from its currently obsolete CPU-processor-centric Aristotelian CS world model, to a twin-paradigm Copernican model.

For energy cost reasons, a massive software to configware migration campaign is needed. First this requires clever planning to optimize all its aspects. We also need to develop plans deciding, which software packets need to be migrated, and, which of them should be migrated first. All this requires many years, probably a decade of massive R&D and education efforts. We cannot afford to hesitate. Lobbying for the massive funding should be started right now. We should address politicians at all levels: community level, state level, national level, and European Union level. To explain all this to politicians is very, very difficult. Since politicians always watch the sentiment of their voter population, we efficiently have to

teach the public, which is a dramatic challenge. Without the support by a strong tailwind from the media a successful lobbying does not seem to have any chance. All this has to be completed as soon as possible, as soon as we can still afford such massive activities. To succeed with such a challenging educational campaign the foundation of a powerful consortium to be funded at all levels is needed for running an at least Europe-wide project.

## References


1. John von Neumann: First Draft of a Report on the EDVAC; University of Pennsylvania, June 30, 1945
2. P. Wallich: Profile: Lynn Conway—Completing the Circuit; Scientific American Magazine, December 2000.
3. L. Conway: The MPC Adventures: Experiences with the Generation of VLSI Design and Implementation Methodologies; Microprocessing and Microprogramming - The Euromicro Journal, Vol. 10, No. 4, Nov 1982.
4. National Research Council: Funding a Revolution; National Academies Press; January 8, 1999
5. N. N.: "The book that changed everything", Electronic Design News, Feb. 11, 2009
6. D. Kilbane: Lynn Conway - A trailblazer on professional, personal levels." Electronic Design, October 23, 2003
7. R. Hartenstein: The Paramountcy of Reconfigurable Computing; in: Young-Choon Lee, Albert Zomaya (editors): Energy Aware Distributed Computing Systems; Wiley, 2011
8. R. Hartenstein: The Grand Challenge To Reinvent Computing; XXX Congress of the SBC, 20–23 July 2010, Belo Horizonte, MG, Brazil,
9. http://wiki.answers.com/QWhy_are_computers_important_in_the_world
10. G. Fettweis: ICT Energy Consumption - Trends and Challenges; WPMC'08, Lapland, Finland, 8 –11 Sep 2008
11. D. Estrin et al: "Internet Predictions," IEEE Internet Computing, vol. 14, no. 1, pp. 12–42, Jan./Feb. 2010, doi:10.1109/MIC.2010.12
12. R. H. Katz: Tech Titans Building Boom; IEEE Spectrum, February 2009
13. R. Nuez: Google is Going into the Electricity Business; The Huffington Post, JUNE 6, 2010; http://www.huffingtonpost.com/ramon-nuez/google-is-going-into-the_b_417035.html
14. http://en.wikipedia.org/wiki/Pelamis_Wave_Energy_Converter
15. A. Zomaya et al.: Interweaving Heterogeneous Metaheuristics Using Harmony Search; Int'l Parallel and Distr. Processing Symp., May 23–29, Rome, Italy
16. J. Rabaey: Low Power Design Essentials; Springer Verlag, 2009
17. L. A. Barroso: The Price of Performance; ACMqueue, Oct 18, 2005 - http://queue.acm.org/detail.cfm?id=1095420
18. Wu-chun Feng: On the Second Coming of Green Destiny?; International Conference on Energy-Aware HPC; Sep 16–17, 2010, Hamburg, Germany,
19. Peter Kogge (editor): ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems; Sep 28, 2008 - http://www.cse.nd.edu/Reports/2008/TR-2008-13.pdf
20. W. Blendinger: Post Peak – Abstieg vom Peak Oil; ASPO annual conference, Berlin, 18 Mai 2010
21. K. Aleklett: Post Peak – The Future of the Oil-Production; ASPO annual conference, Berlin, 18 Mai 2010
22. J. S. Gabrielli de Azevedo: Petrobras e o Novo Marco Regulatório; São Paulo, Dec 1, 2009
23. F. Birol: Leave oil before it leaves us; THE INDEPENDENT (UK) March 2, 2008 - http://www.youtube.com/watch?v=m377Is4tGF0

24. H. Ginsburg: Unterm Wasser liegt die Zukunft; Wirtschaftswoche 23, June 7, 2010
25. H. Simon: Leibniz-Rechenzentrum, TU Munich, 2009, Garching, Germany
26. J. Rabaey: Reconfigurable Processing: The Solution to Low-Power Programmable DSP, Proc. ICASSP 1997
27. T. Claasen: High Speed: Not the Only Way to Exploit the Intrinsic Computational Power of Silicon; ISSCC-1999, pp. 22–25, Feb. 1999
28. R. Hartenstein: Why we need Reconfigurable Computing Education; 1st Int'l Workshop on Reconfigurable Computing Education (RC education 2006), March 1, 2006, KIT Karlsruhe Insitute of Technology, Germany
29. Y. Gu, et al.: FPGA Acceleration of Molecular Dynamics Computations; FCCM 2004 http://www.bu.edu/caadlab/FCCM05.pdf
30. A. Alex, J. Rose et al.: Hardware Accelerated Novel Protein Identification; Proc. FPL 2004, Aug. 29 - Sep 1, 2004, Antwerp, Belgium,
31. N. N. Nallatech, press release, 2005
32. H. Singpiel, C. Jacobi: Exploring the benefits of FPGA processor technology for genome analysis at Acconovis; ISC 2003, June 2003, Heidelberg, Germany; http://www.hoise.com/vmw/03/articles/vmw/LV-PL-06-03-9.html
33. N. N. (Starbridge): Smith-Waterman pattern matching; National Cancer Inst., 2004
34. A. Darabiha: Video-Rate Stereo Vision on Reconfigurable Hardware; Master Thesis, Univ. of Toronto, 2003
35. R. McCready: Real-Time Face Detection on a Configurable hardware Platform; Master thesis, U. Toronto
36. T. Fry, S. Hauck: Hyperspectral Image Compression on Reconfigurable Platforms; IFCCM 2002
37. P. Buxa: Reconfigurable Processing Design Suits UAV Radar; COTS J Oct 2005 http://www.srccomp.com/ReconfigurableProcessing_UAVs_COTS-Journal_Oct05.pdf
38. http://helios.informatik.uni-kl.de/RCeducation/
39. R. Porter: Evolution on FPGAs for Feature Extraction; Ph.D. thesis; Queensland U. of Technology, Brisbane, Australia, 2001,
40. E. Chitalwala: Starbridge Solutions to Supercomputing Problems; RSSI Reconf. Syst. Summer Inst., July 11–13, 2005, Urbana-Champaign, IL, USA
41. S. D. Haynes, P. Y. K. Cheung, W. Luk, J. Stone: SONIC - A Plug-In Architecture for Video Processing; FPL 99
42. M. Kuulusa: DSP Processor Based Wireless System Design; Tampere Univ. of Technology, Publ. No. 296; http://edu.cs.tut.fi/kuulusa296.pdf
43. B. Schäfer et al.: Implementation Of The Discrete Element Method Using Reconfigurable Computing; 15th ASCE Engineering Mechanics Conf., June 2–5, 2002, New York
44. G. Lienhart: Beschleunigung Hydrodynamischer N-Körper-Simulationen mit Rekonfigurierbaren Rechensystemen; Joint 33 rd Speedup and 19th PARS Workshop; Basel, Switzerland, March 19–21, 2003
45. O. O. Storaasli, D. Strenski: Experiences on 64 and 150 FPGA Systems; Reconfig. Syst. Summer Inst., July 7–9, 2008, Urbana-Champaign, IL, USA
46. A. A. Gaffar and W. Luk: Accelerating Radiosity Calculations; FCCM 2002
47. M. Gokhale et al.: Acceleration of Traffic Simulation on Reconfigurable Hardware; 2004 MAPLD Int'l Conf., Sep 8–10, 2004, Washington, D.C., USA
48. J. Hammes, D. Poznanovic: Application Development on the SRC Computers, Inc. Systems; RSSI Reconfigurable Systems Summer Institute, July 11–13, 2005, Urbana-Champaign, IL, USA
49. W. Roelandts (Keynote): FPGAs and the Era of Field Programmability; Proc. FPL 2004, Aug. 29 - Sep 1, 2004, Antwerp, Belgium,
50. F. Dittrich: World's Fastest Lanman/NTLM Key Recovery Server Shipped; Picocomputing 2006
51. K. Gaj, T. El-Ghazawi: Cryptographic Applications; RSSI Reconfigurable Systems Summer Institute, July 11–13, 2005, Urbana-Champaign, IL, USA http://www.ncsa.uiuc.edu/Conferences/RSSI/presentations.html

52. R. Jacobi, M. Ayala-Rincón, L. Carvalho, C. Llanos, R. Hartenstein: Reconfigurable systems for sequence alignment and for general dynamic programming; Genetics and Molecular Research 2005
53. M. Feldman: In Fermi's Wake, a Place for FPGAs? HPCwire, Oct. 15, 2009
54. http://www.chrec.org/~george/tmp/ADG.jpg
55. M. Feldman: The FPGA crowd reacts; HPCwire Oct. 15, 2009
56. N. Conner: FPGAs for Dummies - FPGAs keep you moving in a fast-changing world; Wiley, 2008
57. R. Hartenstein (invited paper): The Microprocessor is no more General Purpose: why Future Reconfigurable Platforms will win; Proc. Int'l Conf. on Innovative Systems in Silicon, ISIS'97, Austin, Texas, USA, Oct 8–10, 1997
58. M. Herz et al.: Memory Organisation for Stream-based Reconfigurable Computing; IEEE ICECS 2002, Sep 15–18, 2002, Dubrovnik, Croatia; http://www.michael-herz.de/publications/AddressGenerators3.pdf
59. http://xputer.de/fqa.html#anchor81257
60. R. Hartenstein, A. G. Hirschbiel, M. Weber: MOM-map-oriented machine - a partly custom-designed architecture compared to standard hardware; Proc. IEEE CompEuro, Hamburg, Germany, May 1989
61. R. Hartenstein, A. Hirschbiel, M.Weber: A Novel Paradigm of Parallel Computation and its Use to Implement Simple High Performance Hardware; Proc. InfoJapan'90, Tokyo, Japan, 1990
62. R. Hartenstein et al. (invited reprint): A Novel Paradigm of Parallel Computation and its Use to Implement Simple High Performance Hardware; Future Generation Computer Systems, no. 7, pp. 181–198 (North-Holland)
63. M. Weber et al.: Automatic Synthesis of Cheap Hardware Accelerators for Signal Processing and Image Preprocessing; 12. DAGM-Symp. Mustererkennung (Pattern Recognition), Oberkochen-Aalen, Germany 1990
64. http://tams-www.informatik.uni-hamburg.de/applets/hades/webdemos/42-programmable/10-pla/pla.html
65. http://www.eecs.berkeley.edu/IPRO/Software/Description/platools.html
66. J. Turley: How Many Times Does CPU Go Into FPGA? Embedded Technology Journal - June 8, 2010
67. Selwood: EPP - A Platform to Bridge a Gap? Emb. Technology J. June 8, 2010
68. J. Turley: Whither Embedded Part Deux; Emb. Technology J. May 25, 2010 http://www.techfocusmedia.net/embeddedtechnologyjournal/feature_articles/20100525-esc2/
69. Jih-Sheng Shen, Pao-Ann Hsiung (editors): Dynamic Reconfigurable Network-On-Chip Design: Innovations for Computational Processing and Communication; Information Science Pub, April 2010
70. R. Hartenstein: Implications of Makimoto's Wave; http://hartenstein.de/ImplicationsMakimotosWave2.pdf
71. R. Baxter et al.: High-Performance Reconfigurable Computing – the View from Edinburgh; AHS 2007, Aug 5–8, 2007, Edinburgh, UK
72. http://www.chrec.org/
73. E. El-Araby et al.: Exploiting Partial Runtime Reconfiguration for High-Performance Reconfigurable Computing. ACM Transactions on Reconfigurable Technology and Systems (TRETS) 1(4): (2009)
74. E. El-Araby et al.: Productivity Of High-level Languages On Reconfigurable Computers: An HPC Perspective; IEEE Int'l Conf. on Field-Programmable Technology (ICFPT 2007), Japan, December 2007.
75. E. El-Araby et al.: Comparative Analysis Of High Level Programming For Reconfigurable Computers: Methodology And Empirical Study; III Southern Conf. on Programmable Logic (SPL2007), Mar Del Plata, Argentina, Feb 2007
76. J. Backus: Can programming be liberated from the von Neumann style? C.ACM Aug 1978
77. E. W. Dijkstra: The Goto considered harmful; Comm ACM, March 1968

78. N. Wirth: A Plea for Lean Software, IEEE Computer, 28, 2, (Feb. 1995)
79. Arvind et al.: A critique of Multiprocessing the von Neumann Style; ISCA 1983
80. Peter G. Neumann 1985–2003: 216x "Inside Risks" (18 years in the back cover of each issue of C_ACM)
81. R. Hartenstein, G. Koch: The universal Bus considered harmful; Workshop on the Microarchitecture of Computer Syst, June 23–25, 1975, Nice, France
82. R. Hartenstein: The von Neumann Syndrome; Stamatis Vassiliadis Memorial Symp., Sep 2007, Delft, NL - http://helios.informatik.uni-kl.de/staff/hartenstein/Hartenstein-Delft-Sep2007.pdf
83. R. Hameed et al.: Understanding Sources of Inefficiency in General-Purpose Chips; 37th ISCA, June 19–23, 2010, Saint Malo, France
84. T. C. Chen et al., "Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder," Circuits and Systems for Video Technology, IEEE Transactions on, vol.16, no.6, pp. 673–688, June 2006
85. S. McKee: Reflections on the memory wall; Computing Frontiers, Ischia, Italy, 2004
86. J. L. Hennessy, D. A. Patterson, Computer Architecture: a Quantitative Approach, Morgan-Kaufman, San Mateo, CA, 1990
87. Kiran Bondalapati, Viktor K. Prasanna: Mapping Loops onto Reconfigurable Architectures; 8th Int'l Workshop on Field-Programmable Logic and Applications, Sep 1998, Tallinn, Estonia
88. M. Huebner, D. Goehringer, J. Noguera, J. Becker: Fast dynamic and partial reconfiguration Data Path with low Hardware overhead on Xilinx FPGAs; Proc. RAW 2010, Atlanta, USA, April, 2010
89. J. Larus: Spending Moore's Dividend; Comm ACM, May 2009
90. Juergen Becker, et al.: Data-procedural Languages for FPL-based Machines; FPL'94, Prague, September 7–10, 1994, Prague, Czechia.
91. C. Bell et al: The Description and Use of Register-Transfer Modules (RTM's); IEEE Trans-C21/5, May 1972
92. M. Ayala, C. Llanos, R. Jacobi, R. Hartenstein: Prototyping Time and Space Efficient Computations of Algebraic Operations over Dynamically Reconfigurable Systems Modeled by Rewriting-Logic; ACM TODAES, 2006
93. http://webadmin.dac.com/knowledgecenter/2010/documents/SMITH-VIRTUAL-ABK-FINAL2.pdf
94. D. Parnas (keynote): Teaching for Change; 10th Conf Softw Engrg Education and Training (CSEET), April 13–16, 1997, Virginia Beach, VA, USA
95. Fig. 13, R. Hartenstein: The History of KARL and ABL; in: J. Mermet: Fundamentals and Standards in Hardware Description Languages; Kluwer, 1993
96. M. Barbacci: The ISPS Computer Description Language; Carnegie-Mellon Univ., Dept. of Computer Science, 1977
97. N. N. (I forgot the celebrity's name): Computer Magazine, 1970 or 1971
98. R. Hartenstein (keynote): Reconfigurable Computing: boosting Software Education for the Multicore Era; IV Southern Programmable Logic Conference (SPL 2010), Porto Galinhas Beach, Brazil, 24–26 March 2010
99. Dick Selwood: Drag-and-Drop vs. HDL? FPGA and Programmable Logic J.; http://www.techfocusmedia.net/fpgajournal/feature_articles/20100803-ni/
100. K. Asanovic et al.: A view of the parallel computing landscape. Comm. ACM 52,10 (2009)
101. 4th Workshop on Programmability Issues for Heterogeneous Multicores (MULTIPROG 2011), January 23, 2011, Heraklion, Crete, Greece - http://multiprog.ac.upc.edu/
102. S. Moore: Multicore Is Bad News For Supercomputers; IEEE Spectrum,
103. M. Broy, R. Reussner: Architectural Concepts in Programming Languages; Computer Oct 2010
104. D. Selwood: Showing off in San Jose; Embedded Technology J., June 8, 2010
105. M. Anderson: Understanding Multi-Core in Embedded Systems; The PTR Group, Falls Church VA, USA, June 15, 2010

106. Rick Merritt: Next computing target—exascale systems; EEtimes, Nov. 15, 2010,
107. Neal Singer: New standard for supercomputing proposed; SANDIA LAB NEWS, Nov 5, 2010, http://www.sandia.gov/LabNews/101105.htm
108. Kevin Morris: Why FPGAs Will Win; FPGA and Programmable Logic Journal Update (techfocus media). 30 Nov 30, 2010
109. Garrison Jeff: What! How big did you say that FPGA is? (Team-design for FPGAs) (EETimes Design Article). 27 Sept. 2010
110. S. Z. Ahmed, G. Sassatelli, L. Torres, L. Rougé: Survey of new Trends in Industry for Programmable Hardware; 20th FPL, Milano, Italy, Aug 31 - Sep 2, 2010
111. Alberto Sangiovanni-Vincentelli: Quo Vadis, SLD? Proc. IEEE, March 2007
112. N. Voros, R. Nikolaos, A. Rosti, M. Hübner (editors): Dynamic System Reconfiguration in Heterogeneous Platforms - The MORPHEUS Approach; Springer, 2009
113. Ch. Bobda: Introduction to Reconfigurable Computing - Architectures, Algorithms, Applications; Springer Verlag, 2007
114. personal communication from authors of other chapters of this book
115. http://gpgpu.org/
116. V. W. Lee et al.: Debunking the 100X GPU vs. CPU myth; 37th ISCA, June 19–23, 2010, Saint-Malo, France,
117. R. Vaduc et al.: On the Limits of GPU Acceleration; USENIX Workshop HotPar'2010, June 14–15, 2010, Berkeley, CA, USA
118. R. Bordawekar et al.: Believe it or Not! Multicore CPUs can Match GPUs for FLOP-intensive Applications! IBM Research Report, April 23, 2010
119. P. Lysaght: The Programmable Logic Perspective; FPL-2010, Sep 2010, Milano, Italy
120. Thomas Scogland, Heshan Lin, Wu-chun Feng: A First Look at Integrated GPUs for Green High Performance Computing; In'l Conf. on Energy-Aware HPC; Sep 16–17, 2010, Hamburg, Germany, http://ena-hpc.org/index.html
121. N. N.: A Call to Arms for Parallel Programming Standards; HPCwire, Nov 16, 2010, http://sc10.supercomputing.org/?pg=tutorials.html
122. Russ Henke: The State of IP; EDACafé, August 16, 2010; http://www10.edacafe.com/nbc/articles/view_weekly.php?articleid=850357

http://www.springer.com/978-1-4614-0060-8

Reconfigurable Computing
From FPGAs to Hardware/Software Codesign
(Eds.)J. Cardoso; M. Hübner
2011, XV, 296p. 107 illus., Hardcover
ISBN: 978-1-4614-0060-8