



# Configware and morphware going mainstream

Jürgen Becker <sup>a,\*</sup>, Reiner Hartenstein <sup>b</sup>

<sup>a</sup> Institut fuer Technik der Informationsverarbeitung (ITIV), Universitaet Karlsruhe (TH), D-761288 Karlsruhe (TH), Germany  
<sup>b</sup> Kaiserslautern University of Technology, Germany

## Abstract

The paper addresses a broad readership in information technology, computer science and related areas, and gives an introduction to fine grain and coarse grain morphware, reconfigurable computing, and its impact on classical computer science and business models. It points out trends driven by microelectronics technology, EDA, and the mind set of data-stream-based computing.

© 2003 Elsevier B.V. All rights reserved.

## 1. Preface

This paper introduces a mind set fundamentally different from (yet) current mainstream digital systems. This roadmap paper may be viewed as a kind of digest of a forthcoming book, or, from a full day tutorial usually required to do this efficiently. That is why this paper is hard to read for readers not daily working directly in the center of the new R&D direction advocated here. For acronyms see Fig. 5.

*Computing requirements are rapidly changing* for embedded systems. Doubling every 12 months, performance requirements for wireless communication are growing very fast (Fig. 1): Hansen's law is faster than Moore's law. Also in other embedded applications, like multimedia, telemetry and oth-

ers, performance requirements are growing rapidly. With the growth rate recently slowing down, the integration density of microprocessors is more and more falling back behind Moore's law (Fig. 1). Accelerators occupy most of the silicon chip area. Compared to hardwired accelerators more flexibility is provided by *morphware*, which will be explained later.

*The amount of software written for embedded systems*, doubling every 10 months, is growing faster than for any other application area (Fig. 1), where entertainment electronics and communication, mainly wireless, are the largest segment of the semiconductor market (Fig. 2). Together already in 1999 these fastest growing market segments, where a major part of products is battery-driven, had a market share of 38% (Fig. 2).

*The capacity of batteries* is growing extremely slow (doubling every 30 years: Fig. 1). A new computing performance metrics has to be introduced. Instead of MIPS we use MOPS per milli-Watts (MOPS/mW: Fig. 3). Will future fuel cells soon solve this problem? Maybe for laptops, but not for handys.

\* Corresponding author. Tel.: +49-721-6082502; fax: +49-721-607438.

E-mail addresses: [becker@itiv.uni-karlsruhe.de](mailto:becker@itiv.uni-karlsruhe.de) (J. Becker), [reiner@hartenstein.de](mailto:reiner@hartenstein.de) (R. Hartenstein).

URLs: <http://www.itiv.uni-karlsruhe.de>, <http://www.hartenstein.de>.

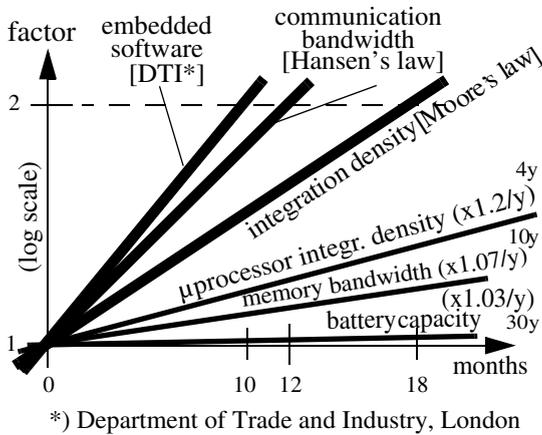


Fig. 1. Moore's Law and other "laws" (ST microelectronics, MorphICs).

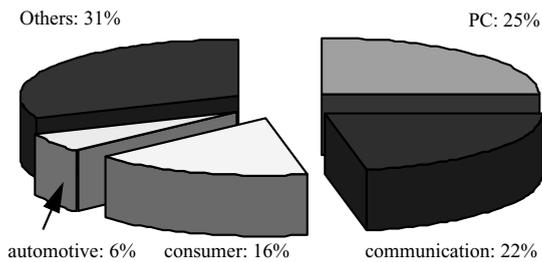


Fig. 2. Semiconductor market sectors in 1999 (Dataquest).

*The memory bottleneck.* Doubling approximately only every 10 years the growth of memory communication bandwidth is extremely slow (Fig. 1). Because of the von Neumann bottleneck, memory bandwidth is an important issue. Avoiding this memory bottleneck not only by using accelerators, but also by innovative computing architectures, or even by breaking the dominance of the von Neumann machine paradigm is a promising goal of new trends in embedded system development and CSE education.

**2. Introduction**

The dominance of the procedural mind set in computer science stems from the general purpose properties of the ubiquitous von Neumann (vN) microprocessor. Because of its RAM-based ex-

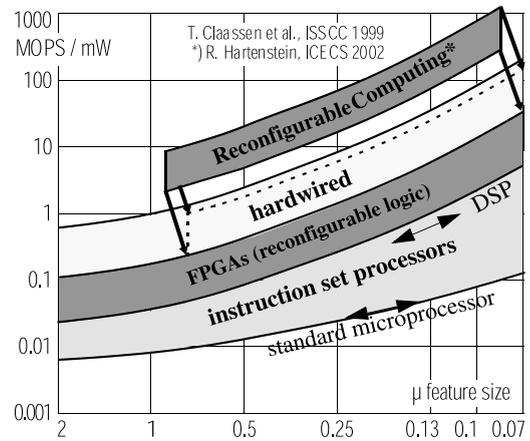


Fig. 3. Energy efficiency vs. flexibility.

treme flexibility no application-specific silicon is needed, so that for software-based products no mass production is needed for profitability. Throughput is the only limitation because of its sequential nature of operation. But now a second RAM-based computing paradigm is heading for mainstream: the application of *morphware* [1], which is also a kind of general purpose platform. Morphware (Fig. 4a), sometimes called *soft hardware*, is programmable by *reconfiguration* of its structure: programming in space—in contrast to vN-based programming in time. Already a single morphware device may provide massive parallelism at logic level or at operator level, often more efficient than vN-based process level parallelism. Morphware has introduced a number of fundamental issues to computing and microelectronics [2] and more will come.

*Booming conferences* on morphware and its applications like FPL [3], the eldest and largest, as well as the adoption of this topic area by congresses like ASP-DAC, DAC, DATE, and others [4] indicate, that morphware is heading from niche to mainstream. This is a viable challenge to CS curricula innovators, an occasion to reconsider vN culture criticism, partly dating back to the 1980s [5].

The vN microprocessor is indispensable. But because of its monopoly our CS graduates are no more professionals.

(a) platform category	source “running” on platform	machine paradigm	(b) category of morphware use	granularity (path width)	(re)configurable blocks used
hardware	(hardwired)		Reconfigurable Logic (fig. 5 b)	fine grain (~1 bit)	CLBs: configurable logic blocks
morphware*	configware		Reconfigurable Computing	coarse grain (example: 16 or 32 bits)	rDPUs: reconfigurable data path units (e.g. ALU-like)
ISP**	software	von Neumann			
AMP***	flowware				
rAMP	flowware & configware	anti machine		multi-granular: supporting slice bundling with carry connect	rDPU slices (example: 4 bits)

\*) term used by DARPA-funded program ACS (adaptable computing systems).  
 \*\*) instruction stream processor  
 \*\*\*) anti machine processor (datastream processor).

Fig. 4. (a) Platform categories and (b) categories of morphware.

*Toward a dichotomy of basic computing paradigms.* From this starting point Computing Sciences (CS) are slowly taking off to explore new horizons: toward a *dichotomy of basic computing paradigms*, by removing the blinders of the still strong vN-only mind set. Within our CS curricula this dominance is still emphasized, but by a slowly decaying majority. It has been predicted, that by the year 2010 more than 90% of programmers will implement applications for embedded systems (Fig. 1) [6,7], where a procedural/structural double rail approach is a pre-requisite, providing new chances.

*Missing algorithmic cleverness.* Currently most programmers do not yet really have the background required. Typical programmers have severe problems to map an application onto platforms other than relying on state-machine-based mechanisms like vN or similar. This education gap can be easily bridged only by a curricular transition from the von-Neumann-only mind set toward this dichotomy of two basic computing paradigms, where the vN paradigm is taught together with a second machine paradigm which we call anti-machine [8,9]. A rich supply of tools and research results is available to adapt fundamental courses, lab courses and exercises. Not the time needed for such modifications is the problem, since there are a lot of similarities between both branches, like between matter and anti-matter.

*Our educators are the problem.* The vN microprocessor is indispensable. But because of its monopoly our CS graduates are no more professionals. The key problem is, that educators need to be a more open-minded to be able to cope with the few asymmetries between both branches.

A concise and comprehensive basic terminology, an important help to remove barriers, will be summarized by this paper, along with an introduction to morphware, the new computing paradigm, its history, its commercial and scientific backgrounds, as well as to its key issues and future aspects.

### 3. Morphware

Currently the most important motivation to replace hardwired accelerators by morphware is more flexibility. But performance is the key issue. Currently often it is still more difficult to obtain high speed-up by morphware. But there is a wide variety of acceleration factors. For example the migration of a cell metabolism simulator [10] from 1,3 GHz Pentium to 42 MHz implementation on Virtex II (a Xilinx FPGA) brought a factor of 8, and a FIR filter on a 4-by-4 coarse grain reconfigurable array a factor of 9 [11]. Compared to an implementation on a VAX-11/750 a grid-based design rule check [12] with a reprogrammable PLA (which to-day could well be replaced by an FPGA), runs on a different machine paradigm to avoid the vN bottleneck [13] brought a factor by three orders of magnitude—by processing a complex Boolean expression in a single clock cycle and using a generic address generator to implement a 4-by-4 pixel video scan window [14].

In morphware application the lack of algorithmic cleverness is an urgent educational problem.

AM	anti machine (DS machine)	EH	evolvable morphware ("evolvable hardware")
AMP	data stream processor*	FPGA	field-programmable gate array
ASIC	application-specific integrated ckt.	ISP	instruction stream processor
asMB	autosequencing Memory Bank	rDPU	reconfigurable DPU
CPU	"central" processing unit: DPU (with instruction sequencer)	rDPA	reconfigurable DPA
cSoC	configurable SoC	RA	reconfigurable array
DPA	data path array (DPU array)	RAM	random access memory
DS	data stream	rAMP	reconfigurable AMP
DPU	data path unit ( <b>without</b> sequencer)	RC	reconfigurable computing
ecDPU	emulation-capable DPU	RL	reconfigurable logic
EM	evolutionary methods	RTR	run time reconfiguration
EDA	electronic design automation	SoC	(an entire) System on a Chip

\*) no "dataflow machine" [15]

Fig. 5. Acronyms.

*Reconfigurable logic versus reconfigurable computing.* A growing consensus on terminology (Fig. 4) and acronyms (Fig. 5) clearly distinguishes *reconfigurable logic* (RL) from *reconfigurable computing* (RC, Fig. 4), as well as different platform categories (Fig. 4). With *data stream processor* terminology has a dilemma: "dataflow machine" [15] and digital signal processing (DSP) have been occupied decades ago by other areas. So we prefer the acronym *anti-machine* (AM) of *AMP*.

*Terminology.* Even more important is the terminology from a more global point of view, not to get confused between "programmable", "micro-programmable", "field-programmable" and contradictory "soft hardware", and, to clarify the dichotomy of fundamental models, as well as, to alleviate the merger of RC and classical CS (Fig. 4a). To add more confusion some authors even use the term "reconfigurable" (because processors can be addressed) for micro-miniaturized classical parallel computing systems with all its classical CPUs on a single chip. Whereas classical CS deals with *software* running on *hardware*, the new branch of CS deals with *flowware* running on *hardware*, as well as with *configware* [16,17] and *flowware* [18] "running" on *morphware*. This paper gives introductions for a broad readership mainly with a CS background, rather than for hardware specialists.

*Reconfigurability.* Although it is not seriously taught to most CS students, it is not new, that algorithms can be implemented in software, or in hardware. Implementing an application by *software* means *procedural programming*, i.e. pro-

gramming in time. But implementing an algorithm on *morphware* means *structural programming by configware* for structural reconfiguration of the morphware device (mapper output, see Fig. 6) and *flowware* for scheduling the data streams (scheduler output, see Fig. 6). The acronym *field-programmable gate array* (FPGA) indicates, that structural re-programming can be practised anywhere, like e.g. at the customer's site. This also is an important commercial aspect already now, which will have an even greater impact in the future. Following chapter will illustrate the reconfigurability mechanism within morphware platforms by an example of reconfigurable logic (*fine grain morphware*).

#### 4. Reconfigurable logic

Fig. 7 illustrates the basic mechanisms of *RL* implementation on a *morphware* platform, which can be changed at any time after fabrication by downloading configware into the configuration RAM.

*CLBs and interconnect fabrics.* A minor part of the area is used by *configurable logic blocks* (CLBs), which are the logic resources. Major part of the area is covered by a *reconfigurable interconnect fabrics*, providing wire pieces, switch boxes, and connect boxes to connect a pin of a CLB, with a pin of another CLB by programming a "soft wire" (an example shown in Fig. 7). The state of each switching transistor is controlled by a Flip-flop (FF) which is part of "hidden" *configuration RAM* (not shown in Fig. 7), also used to

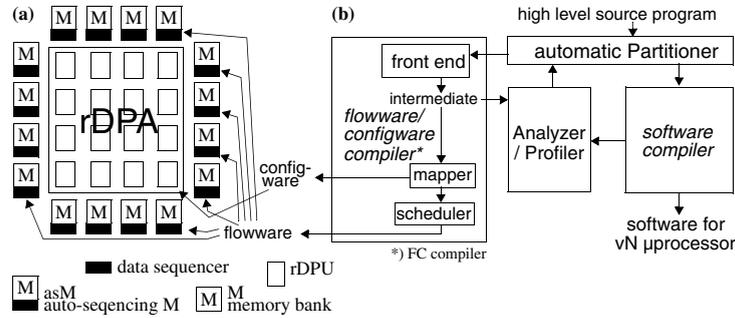


Fig. 6. FC/software co-compilation: (a) anti-machine example and (b) partitioning co-compiler.

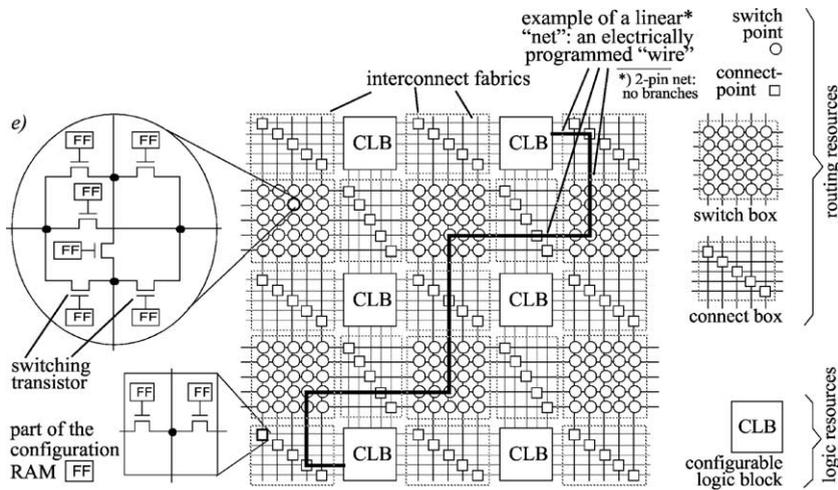


Fig. 7. Illustration of morphware reconfigurability resources (FPGA: only 1 configured "wire" shown).

program the CLBs to select the particular logic function of each. By new configware all this can re-programmed anywhere and at any time.

*The history of fine grain morphware.* The domain of morphware platforms and their applications has undergone a long sequence of transitions, where we may distinguish different subareas (Fig. 4b): *fine grain* morphware (FPGAs) and *coarse grain* morphware. First FPGAs appeared as cheap replacements of Mask Programmable Gate Arrays (MPGAs). Still today FPGAs are the reason for shrinking ASIC<sup>1</sup> (Fig. 5) markets, since for FPGAs no application-specific silicon is needed—

a dominating cost factor in low production volume products. Later the area proceeded into a new model of computing possible with FPGAs. Next step was making use of the possibility for debugging or modifications the last day or week, which also leads to the adoption by the *rapid prototyping* community which also has lead to the introduction of *ASIC emulators* faster than simulators. Next step is direct *in-circuit execution* for debugging and patching the last minute.

*Fastest growing market.* Morphware is the fastest growing segment of the integrated circuit (IC) market, currently relying on a growing large user base of HDL-savvy designers. Cost differences between volume FPGAs and volume ASICs are shrinking. Driven by a growing large user base innovations occur more and more rapidly. FPGA

<sup>1</sup> ASIC fabrication cost is much lower (only a few specific masks needed) than that of other integrated circuits (Fig. 8).

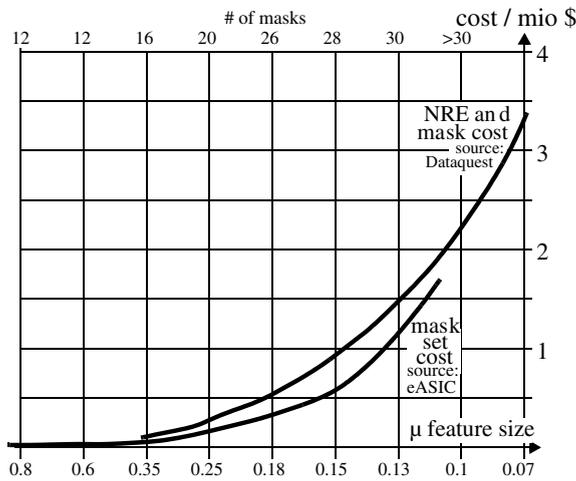


Fig. 8. Increasing mask set cost and total NRE cost.

vendors are heading for the forefront of platform-based design.

**New business models.** Morphware brings a new dimension to digital system development and has a strong impact on system-on-chip (SoC) design. Performance by parallelism is only one part of the story. The time has come to fully exploit morphware flexibility to support very short turn-around time for real-time in-system debugging, profiling, verification, tuning, field-maintenance, and field-upgrades. The consequence is a new business model for all kinds of electronics products, where patches and upgrades are carried out at the customer's site—even via the internet using run time reconfiguration (RTR). This is an important remedy of the current embedded system design crisis caused by skyrocketing design cost coincides with decreasing product lifetime, by providing product longevity (Fig. 9).

**Rapid prototyping and ASIC emulation.** Since in integrated circuit design flow simulation may take days or even weeks, the next step has been *ASIC emulation*, using huge emulation machines called ASIC emulators. By acquisitions the three major EDA vendors offer ASIC emulators, along with compilers: Cadence has acquired Quickturn, Synopsys has acquired IKOS, and Mentor Graphics bought Celaro, also offering such service over the internet. Another R&D scene and market segment is calls itself *rapid prototyping*, where for smaller

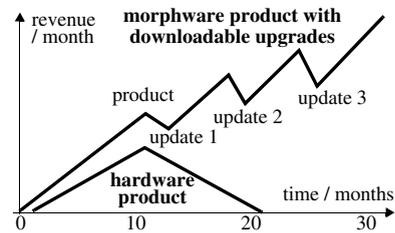


Fig. 9. Morphware product longevity by updates.

designs less complex emulation boards are used, like logic emulation PWB (based on Xilinx Virtex, can emulate up to three million gates), and, the DN3000k10 ASIC Emulator from the Dini Group.

**Retro emulation.** A future application of emulation may serve to solve the long term microchip spare part problem in areas like industrial equipment, military, aerospace, automotive, etc. with product lifetimes up to several decades. The increasing spare part demand (Fig. 10) stems from increasing amount of embedded systems, limited life time of microchip fab lines (mostly less than 7–10 years), and decreasing life time of unused microchips (Fig. 10). When a modern car with several dozens of embedded microchips needs electronic spare parts 10 or 15 years later, the microchip fab line is no more existing, and major percentage or all of the parts kept in a spare parts storehouse have faded away. To keep an old fab line alive, which would deliver long-lasting robust products at low NRE cost, seems to be an illusion. *Retro emulation* might be the only viable solution, where

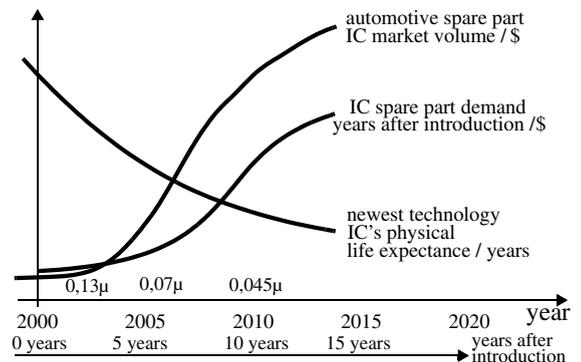


Fig. 10. The emerging automotive IC spare part problem.

reverse engineered products are emulated on FPGAs, since application-specific silicon will not be affordable because of low microchip production volumes in these areas and rapidly increasing mask cost (Fig. 8).

*cSoC and general purpose chip.* The most important business factor of morphware is its general purpose property, so that application-specific silicon with its extremely high NRE cost can be avoided. With the continuous technology progress even entire SoC can be implemented on morphware *configurable SoC* (cSoC). Due to Moore's law (Fig. 1) the FPGA vendors offer more and more products having microcontrollers like ARM, MIPS, PowerPC, or other RISC architectures, memory, peripheral circuitry and others, together with the FPGA on board of the same chip. A Xilinx FPGA, for example, has four PowerPCs on board and 24 Conexant 3.125 Gb/s serial link cores providing a total of 75 Gb/s/chip link bandwidth.

*Power efficiency.* More recently the research is heading toward low power FPGAs from the algorithms side. By transfer of an algorithm from a high performance DSP to an experimental low power FPGA a performance improvement by factors between 5 and 22 has been obtained by Jan Rabaey et al. [19]. A reduction of clock frequency by a factor of  $n$  yields a reduction of power dissipation by a factor of  $n^3$  [20], when also the optimum technology is selected—if its fab line is still available.

*Evolvable morphware.* The terms *evolvable morphware*, and *evolutionary methods* (EM, also called Darwinistic Methods), and biologically inspired electronic systems stand for a new application area of morphware in research—a resurrection of cybernetics or bionics, stimulated by the new availability of morphware technology, where the effect of chromosomes within evolution is emulated by continuously changing configware [51]. The labelling “evolutionary” and the “DNA” metaphor helped to start and popularize new conference series [52], and to raise research funds in the EU, Japan, Korea, and the US. Critics disfavor the trend to love genetic algorithms in other applications areas, even where simulated annealing is more efficient, like e.g. for simultaneous placement and routing which could be viewed as a kind of

mutational morphing [26]. The new morphware-driven EM scene is still in its visionary phase.

## 5. Reconfigurable computing

Fine grain morphware lacks area/power-efficiency (Fig. 3). The physical integration density (transistors per chip) of FPGAs is roughly two orders of magnitude worse than the Gordon Moore Curve. Due to reconfigurability overhead roughly about only one percent of these transistors deserve the real application, so that the logical integration density is about four orders of magnitude behind Gordon Moore. For very high throughput requirements RC using coarse grain morphware is the drastically more powerful and more area-efficient alternative, also providing a massive reduction of memory and time needed for configuration [21]. Coarse grain morphware is also about one order of magnitude more energy-efficient than fine grain (Fig. 3 [22–24]). Whereas RL based on fine grain morphware (FPGAs) uses single bit wide CLBs (Fig. 4b), reconfigurable computing (RC) uses *reconfigurable datapath units* (rDPUs), which, similar to ALUs, have major path widths, like 32 bits, for instance—or even *rDPU arrays* (rDPAs). Important applications stem from the performance limits of the “general purpose” processor, creating a demand for accelerators.

*Coarse-grain reconfigurable architectures.* Several interconnect topology categories of coarse grain morphware for reconfigurable computing (survey: [21]) may be distinguished: universal RAM-based topologies of *regular arrays* (URA), customized *regular RAM-based arrays* (CRA), hierarchical *irregular RAM-based macro-blocks* (IMB). Another class of resources for reconfigurable computing is called *multi-granular morphware*, where several nibble pathwidth rDPU slices (4 bits, for instance) with slice bundling capability including carry signal propagation can be configured to be merged into datapath units (DPUs) with a path width of multiples of the slice path width (e.g. 16, 20, or 24 bits).

*Commercial architectures.* Especially in application areas like multimedia, wireless telecommunication, data communication and others, the

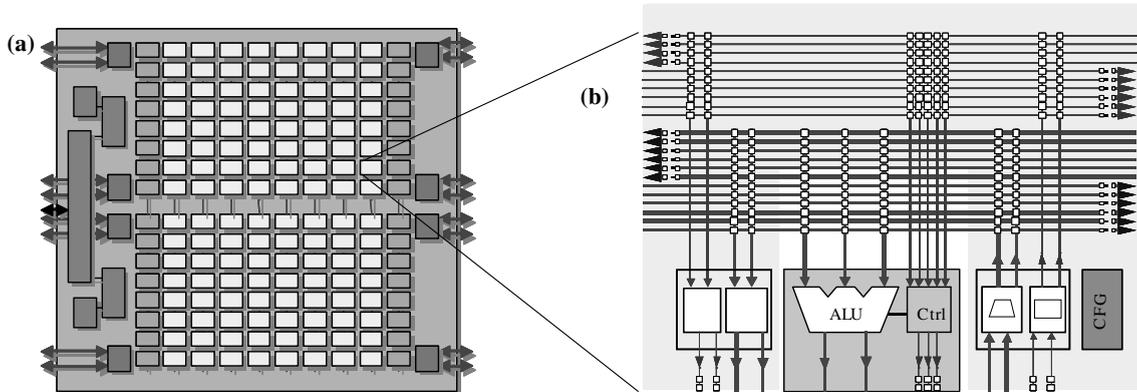


Fig. 11. Configurable XPU (xtreme processing unit) from PACT: (a) array structure and (b) rDPU.

throughput requirements are growing faster than Moore's law (growth of required bandwidth: Fig. 1), along with growing flexibility requirements due to unstable standards and multi-standard operation [27]. Currently the requirements can be met only by rDPAs from a provider like PACT (Fig. 11 [28]). Since a general purpose rDPA still seems to be an illusion, such applications need domain-specific architectures, so that specific silicon is needed, commercially feasible to very high production volume products like in consumer electronics.

*Different routes to DPAs.* In design and implementation of embedded systems using DPAs different business models are possible: using hardware DPUs and DPAs designed all the flow down to physical layout and tape-out directly or from a library, or morphware using rDPUs and rDPAs. With morphware using a fully universal rDPA of universal rDPUs is not area-efficient nor resource-efficient, unless done with a multi-granular platform which permits configuring narrow path rDPUs into wider compound rDPUs. This supports a business model, where personalization is carried out after fabrication, and many different design can be implemented onto the same platform.

*The domain-specific rDPA approach* is another solution [21], where rDPU architecture and other features are optimized for a particular application domain. This may provide very high area efficiency by structured VLSI design [25] of the DPUs supporting regular array layout and wiring by abutment, like the cells of the KressArray family [26].

A design space explorer may help to derive and compare within a few days several alternative DPU and array architectures by a benchmark or a domain-typical set of applications [29]. A desirable new solution for the far future could be the soft array approach mapping DPAs onto a very large FPGAs, providing highest flexibility (*structured configurable design*: see Section 10).

*The same models for hardware and morphware.* All four routes have in common, that mainly the same mapping design flow part may be used for all of them. With the tendency toward data-stream-based DPAs there is in principle no difference, whether the DPU array is hardwired or reconfigurable—except the binding time of placement and routing: before, or, after fabrication.

## 6. Data-stream-based computing

The world of traditional instruction-stream-based informatics is based on computing in the time domain, where *software* schedules the instructions for execution. The classical machine model, which we call vN paradigm, locates instruction sequencer and datapath in the same CPU (Fig. 12a). We need only one program source for vN machines (Fig. 13a), the *software* which provides the implementation of the instruction stream executed at run time. For hardwired data-stream-based computing platforms, however, we need another programming source. Instead of *software*

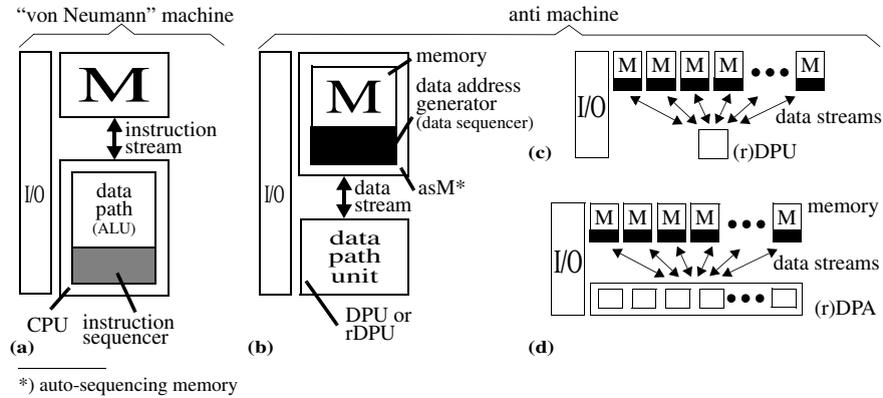


Fig. 12. Illustrating basic machine paradigms (legend: Fig. 6): (a) von Neumann; (b) data-stream-based anti-machine with simple DPU; (c) with rDPU and distributed memory architecture and (d) w. DPU array (DPA or rDPA).

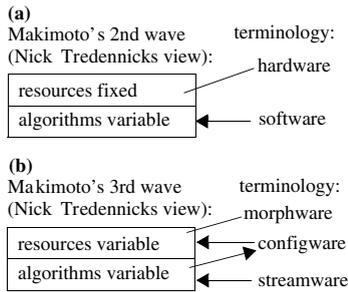


Fig. 13. Terminology explained [36,37].

we need *flowware* to implement data streams instead of an instruction stream. For a reconfigurable data-stream-based platform, however, we need two different “programming sources” (Fig. 13b): a *configware* source to configure (before run time) the resources, and *flowware* to determine, what has to be executed at run time.

*Why we need this second machine paradigm.* Due to the availability of morphware a second basic machine model is needed, since the instruction-stream-based vN model does not support changing datapath units. We have to introduce a data-stream-based second model which we call *anti-machine* so that we now have a dichotomy of two models: instruction-stream-based (vN) computing vs. data-stream-based computing. There are a lot of similarities between both models, so that each of the 2 models is a kind of mirror image of the other model—like with matter and anti-matter

[22]. The instructions are the electrons circulating around the CPU of the vN machine, whereas data streams circulating around the (r)DPU or (r)DPA are the positrons of the AM.

*Similarities and asymmetries.* Data-stream-based computing, the counterpart of instruction-stream-based vN computing, however, uses a data counter instead of a program counter (example in Fig. 12b). However, there are some asymmetries, like predicted by Paul Dirac for anti-matter. Fig. 6a shows the block diagram of data stream machine with 16 autosequencing memory banks. The basic model allows this example machine to have 16 data counters, whereas a vN machine cannot have more than one program counter. The partitioning scheme of the data stream machine model assigns a sequencer (address generator) always to a memory bank (Fig. 12b), never to a DPU. This modelling scheme goes fully conform with the area of embedded distributed memory design and management (see Section 6.2).

### 6.1. Flowware

Data streams, also efficiently supporting loop-level parallelism [30], have been popularized throughout the 1980s by systolic arrays, later by the super systolic array (pioneered by the Kress-Array [26], the generalization of the systolic array [31]), and by projects like Streams-C Configurable Computing (SCCC) [32], Stream Computations

Organized for Reconfigurable Execution (SCORE) [33], Adapting Software Pipelining for Reconfigurable Computing (ASPRC), Biggascale Emulation Engine (BEE) [34], the KressArray Xplorer [29] and many others. In a similar way like instruction streams can be programmed from *software sources*, also data streams can be programmed, but from *flowware sources*. High level programming languages for flowware [35] and for software join the same language principles and have a lot in common, no matter, whether finally the program counter or a data counter is manipulated. The data schedule generated from a flowware source determines, which data object has to enter or leave which DPA port (or DPU port) at which time, so that the embedded distributed autosequencing memory has to generate the data streams.

*Two programming sources.* vN machine needs just *software* as the only programming source, since its hardwired resources are not programmable. A reconfigurable data-stream-based machine, however, needs two programming sources: *con-figureware* to program (reconfigure) the operational resources, and, *flowware* to schedule the data streams. Fig. 6b illustrates the structure of a compiler [16,26] generating the code of both sources from a high level programming language source: phase 1 performs routing and placement to configure the rDPA, resource, and phase 2 generates the flowware code needed to program the autosequencing distributed memory accordingly.

## 6.2. Embedded memory

Increasing use of data-stream-based architectures coming with the growing embedded system market has recently stimulated distributed embedded memory as a growing market segment and important R&D area. Because of using distributed memory the AM has no vN bottleneck. Together with application-specific distributed memory architecture synthesis also flowware implementation deserves performance and power dissipation optimization [14]. Good flowware may be also obtained after optimized mapping an application onto rDPA, where both, data sequencers and the application can be mapped (physically, not conceptually) onto the same rDPA [21].

The anti-machine has no von Neumann bottleneck.

*Address generators.* To solve the memory communication bandwidth problem the AM paradigm (data-stream-based computing) is much more efficient than “von Neumann”. Two alternative methodologies are available [14]: specialized architectures using synthesized address generators (e.g. APT by IMEC [14]), or, flexible architectures with programmable general purpose address generators [13], which do not need memory cycles even during complex address computations [14]—an important performance issue. E.g. a kind of image processing (for a grid-based design rule check) on a VAX 11/750 moving a 4-by-4 pixel window in a video scan mode took 94% of computation time for address computation [12].

## 7. Data-stream-based vs. concurrent computing

Software processor solutions are inefficient relative to hardwired solutions (Fig. 3). Fundamental flaws are: time multiplexing a single piece of logic, data memory/processor traffic overhead, control flow overhead, and intra-processor pipelining control overhead. Compared to real logic functions the overhead going into caches and other auxiliary hardware is several orders of magnitude higher [34]. Processor chips are almost all memory, because the architecture is wrong. The metric for what is a good solution has been wrong all the time.

*Contra concurrent computing.* Arrays or other ensembles of CPUs are difficult to program, and often the run-time overhead is too high, except for a few special application areas favored by Amdahl's Law. Amdahl's Law explains just one of several reasons of inefficient resource utilization. Each CPU comes with a central vN bottleneck, whereas a DPU may have many ports active simultaneously. Like with a systolic array, DPU array computing means parallelism by an application-specific pipe network. There is no CPU. There is nothing “central”. Data-stream-based computing with (r)DPAs provides a drastically more efficient way to cope with memory commu-

nication bandwidth problems than classical concurrent computing.

7.1. The key to massive parallelization

However, today for DPA synthesis or mapping applications onto rDPAs linear projection is no more used, but simulated annealing instead, to avoid the limitation to regular data dependencies [26]. This (“super systolic”) generalization of the systolic array also supports inhomogeneous irregular arrays [29]—in contrast to classical systolic arrays.

*Hardwired DPAs.* On hardwired DPU array basis the BWRC [34] has trumped a “chip in a day” design methodology by direct mapping of algorithms onto high-level, pre-characterized macros wired together from a Simulink dataflow diagram. An automated flow goes through module generation, synthesis and layout. With system level optimization BWRC got rid of difficult problems by using relatively low clock rates—but for gaining a factor of about 100 in area efficiency.

8. Configware compilers

We already had “silicon compilers”, which, however, had been datapath layout generators, rather than compilers. But taking into account, that we now have two classes of RAM-based programmable platforms, classical instruction stream processors and morphware, we should distinguish two classes of compilers; classical software compilers (SW compilers), and, flowware/

configware compilers (FC compilers [26,38], compare Fig. 6 and Fig. 15, which explains the need for two sources).

*Co-compilation.* When reconfigurable accelerators are involved instead of the hardwired accelerators, hardware/software co-design turns into FC/software co-design. Using compilation techniques for both sides we turn co-design into FC/software co-compilation. Using coarse grain morphware (rDPAs) as accelerators changes the scenario: implementations onto both, host and accelerator(s) are RAM-based, which allows turn-around times of minutes for the entire system, instead of months, and, supporting a migration of accelerator implementation from IC vendor to customer, who usually does not have hardware experts. This creates a demand for compilers accepting high level programming language (HLL) sources [26,38]. Partly dating back to the 1970s and 1980s know-how is available from the classical parallelizing compiler scene, like software pipelining, and, loop transformations and other concepts [39–44].

*Automatic partitioning.* Currently, not only for hardware/software co-design, but also for software/configware design, the compiler is a more or less isolated tool used for the host only. But most accelerators are still implemented by CAD. *Software/configware partitioning is still done manually*, requiring massive hardware expertise, particularly when hardware description language (HDL) and similar sources are used. Partitioning compilation [26,32] and co-compilation from HLL sources [45] still stems from academic efforts, as well as the first automatic *co-compilation* from HLL sources including automatic software/configware

machine category	“von Neumann” machine	anti machine
general property	procedural sequencing: deterministic*	
driven by	single instruction stream	data <b>stream</b> (s) ( <b>no</b> “dataflow”*)
engine principles	instruction sequencing	data sequencing
state register	program counter	(multiple) data counter(s)
communication path set-up	at run time	at load time
data path	resource	single ALU
	operation	sequential
morphware support	no	yes

\*) arbitration-driven the “dataflow” machine is deterministic [15]

Fig. 14. Comparing the properties of machine paradigms.

	(a) instruction set processor	(b) data stream processor	
		hardware	morphware
machine paradigm	von Neumann (vN)	anti machine	
reconfigurability support	no	yes	
programming	procedural	no	structural (super “instruction” fetch*)
		data scheduling	
program source	software	flowware	flowware & configware
“instruction” fetch	at run time	before run time (at loading time)	
execution at run time	instruction schedule	data schedule	
operation spini	nstruction flow	data stream(s)	
operation resources	CPU	DPU, or, DPA	rDPU, or, rDPA
	hardwired	hardwired	reconfigurable
parallelism	only by multiple machines	by single machine or multiple machines	
state register	program counter	one or more data counter(s)	
state register located	within CPU	outside DPU or DPA	outside rDPU or rDPA

\*) **before** run time

Fig. 15. Asymmetry between machine and anti-machine paradigms.

partitioning (Fig. 6a) by identifying parallelizable loops, having been implemented for the data-stream-based MoM (Map-oriented Machine) [13].

OS. Also operating systems for morphware to provide run time management are an interesting issue, especially for scheduling problems, multi-tasking on morphware coprocessors, dynamic re-configuration, like for exploiting operation level parallelism through dynamically reconfigurable datapaths, embedded memory management, I/O organization, etc.

### 8.1. Machine paradigms and other general models

Machine paradigms are important models to alleviate CS education and for understanding implementation flows or design flows. The simplicity of the vN paradigm helped a lot to educate zillions of programmers. Fig. 12a shows the simplicity of the block diagram, which has exactly one CPU and exactly one RAM module (memory M). The instruction sequencer and the DPU are merged to be the CPU (central processing unit), whereas the RAM (memory M) does not include a sequencing mechanism. Other important attributes are the RNI mode (read next instruction) and a branching mechanism for sequential operation (computing in the time domain). Fig. 14 compares both machine paradigms. Since compilers based on the “von Neumann” machine paradigm do not support

morphware we need the data-stream-based machine paradigm for the rDPA side, (based on *data sequencer* [14]).

*The AM paradigm* for morphware [46] and even for hardwired AMs the data-stream-based AM paradigm is the better counterpart (Fig. 12b) of the vN paradigm (Fig. 12a). Instead of a CPU the AM has only a DPU (datapath unit) without any sequencer, or a rDPU (reconfigurable DPU) without a sequencer. The AM model locates data sequencers on the memory side (Fig. 12b). AMs do not have an instruction sequencer. Unlike “von Neumann” the AM has no vN bottleneck, since it also allows multiple sequencers (Fig. 12c) to support multiple data streams interfaced to multiple memory banks, which is typical to data-stream-based computing (Fig. 12c) allowing operational resources much more powerful than ALU or simple DPU: major DPAs or rDPAs (Fig. 12d).

*General purpose anti-machine.* The AM is as universal as the vN machine. The anti-programming language is as powerful as vN-based languages. But instead of a “control flow” sublanguage a “data stream” sublanguage like *MoPL* [35] recursively defines *data goto*, *data jumps*, *data loops*, *nested data loops*, and *parallel data loops*. For the AM paradigm all execution mechanisms are available to run such an anti-language. Its address generator methodology includes a variety of escape mechanisms needed to interrupt data streams

by decision data or tagged control words inserted in the data streams [46].

The AM model, where the DPUs are transport-triggered by arriving data, goes conform with the new and rapidly expanding R&D area of embedded memories [14], including application-specific or programmable data sequencers. Fig. 14 compares the properties of both paradigms.

## 9. Configware industry

*RAM-based success story.* The secret of success of software industry is based on RAM, vN paradigm, compatibility, relocatability, and, scalability. Due to the simplicity of the von Neumann machine paradigm zillions of programmers can be educated. The processor can be fabricated in volume since all personalization (programming) is downloadable to scalable RAM. Relocatability of machine code is provided by the machine paradigm. Compatibility as a quasi-standard is achieved by business strategy.

*Configware industry,* emerging as a counterpart to software industry. Being RAM-based configware industry is taking off to repeat the success story known from software. Tsugio Makimoto has predicted this more than 10 years ago [36,47]. Like software, also configware may be downloaded over the internet, or even via wireless channels. FPGA functionality can be defined and even upgraded later at the customer's site—in contrast to the hardware it replaces: configware use means a change of the business model—providing shorter time to market and (FPGA) product longevity (Fig. 9). Many system-level integrated products without reconfigurability will not be competitive.

*Also the configware market* is taking off for main-stream. Because FPGA-based designs become more and more complex, even entire systems on a chip, a good designer productivity and design quality cannot be obtained without good configware libraries with soft IP cores from various application areas.

*EDA is the key enabler.* For the customer EDA is the key enabler to obtain high quality FPGA-based products with good designer productivity. A good morphware architecture is difficult to utilize,

if it is not efficiently supported by the EDA environment(s) available. But EDA still often has problems with designer productivity and quality of designs. Zero maintenance bullet-proof self-supporting tools for masses of designers, with a low EDA budge are still missing.

*EDA software market.* Currently most of the configware (reusable soft IP cores) is provided by the FPGA vendors for free as a service to their customers: the top FPGA vendors are the key innovators. But the number of independent configware houses (soft IP core vendors) and design services is growing. Also a separate EDA software market, comparable to the compiler and OS market in computers, separate from the morphware is already existing, since Cadence, Mentor Graphics and Synopsys just jumped into it by closing the back end gap down to creating configware code.

*Structured configware design.* The main problem of configware industry in competing with software industry is the lack of FPGA-based compatibility, scalability, and code relocatability. Re-compilation and re-debugging is required for another FPGA type. Rather than by new FPGA architectures a future solution will be feasible by a new EDA approach, which supports structured configware design based on soft macro cells supporting wiring by abutment made possible by placement strategies being successful also for rDPAs (compare Section 5).

## 10. Soft CPUs and rDPAs

Configware providers meanwhile offer CPUs as soft IP cores also called FPGA CPU, or, soft CPU, to be mapped onto an FPGA, like MicroBlaze (32 bits, 125 MHz, Xilinx), the Nios (multi-granular, Altera), Leon (32 bit RISC, SPARC V8 compatible, public domain). Using the usual FPGA design flow the soft CPU IP cores can be generated from VHDL or Verilog originally targeted at a hardwired implementation.

*An application example* is a configurable system-on-chip (CSoC) consisting of a soft Leon core, an XPP-array of suitable size, global and local memory cores and efficient multi-layer Amba-based communication interfaces having been

synthesized and analyzed onto 0.13  $\mu\text{m}$  UMC CMOS (8 layer) copper standard cell process at University of Karlsruhe [49]. The obtained synthesis and optimization results were used within the first silicon-right 0.13  $\mu\text{m}$  STMicro CMOS Chips from PACT [53]. The actual applications currently mapped and analyzed onto this XPP64A resp. XPP16 ( $4 \times 4$  ALU PAE array) are different MPEG algorithms, W-LAN (Hiperlan-2), Software Radio including RAKE-Receiver and Baseband Filtering. Implementing a 16 tap FIR filter on a XPP 16 array yielded a speed-up by a factor of 4 and a power efficiency improvement (MOPS/mW) by a factor of 16—compared to a state of the art digital signal processor (SC140 Starcore, [53]).

*The Giga FPGA.* Already now 32 MicroBlaze or 64 Nios can be mapped onto a single FPGA. Within a few years FPGAs with more than 100 million gates will be available commercially, onto which more than a 100 soft processor cores can be mapped, leaving plenty of area to other on-chip resources, so that a coarse grain morphware like from PACT [48] [49,50] can be mapped onto it [38], maybe in 5–10 years from now. The performance disadvantage of lower FPGA clock frequency can be fixed by a higher degree of parallelism obtained through algorithmic cleverness.

## 11. Conclusions

The paper has given an introduction to reconfigurable logic and reconfigurable computing, and its impact on classical computer science. It also has pointed out future trends driven by technology progress and EDA innovations. It has tried to highlight, that deep submicron allows SoC implementation, and the silicon IP business reduces entry barriers for newcomers and turns infrastructures of existing players into liability.

Many system-level integrated future products without reconfigurability will not be competitive. Instead of technology progress better architectures by reconfigurable platform usage will often be the key to keep up the current innovation speed beyond the limits of silicon. It is time to revisit past results from morphware-related R&D to derive promising commercial solutions and curricular

updates in basic CS education. Exponentially increasing CMOS mask costs demand adaptive and re-usable silicon, which can be efficiently realized by integrating morphware of different granularities into cSoCs, providing a potential for short time-to-market (risk minimization), multi-purpose/-standard features including comfortable application updates within product life cycles (volume increase: cost decrease). This results in the fact that several major industry players are currently integrating reconfigurable cores/datapaths into their processor architectures and system-on-chip solutions.

## References

- [1] Available from <<http://morphware.net/>>.
- [2] W. Mangione-Smith et al., Current issues in configurable computing research, IEEE Comput. (1997).
- [3] Available from <<http://fpl.org>>.
- [4] Available from <<http://www.aspdac.com/>, <http://www.dac.com/>, <http://www.date-conference.com/>>.
- [5] P. Naur, Computing: A Human Activity—Selected Writings From 1951 To 1990, ACM Press/Addison-Wesley, New York, 1992.
- [6] H. Hansson, DTI prediction, Department of Trade and Industry (DTI), London, UK 2001.
- [7] F. Rammig, Eingebettete Systeme, in: 10th Anniversary Workshop, Fraunhofer EAS Dresden, April, 2002.
- [8] R. Hartenstein, Reconfigurable computing: urging a revision of basic CS curricula, in: Proceedings of the 15th International Conference on Systems Engineering (ICSENG02), Las Vegas, USA, 6–8 August 2002 (invited paper).
- [9] R. Hartenstein, Data-stream-based computing: Antimaterie der Informatik, in: 60th Semester Anniversary Workshop of Prof. Reusch, University of Dortmund, 18–19 July 2002 (invited paper).
- [10] Y. Osana et al., Implementation of ReCSiP: a ReConfigurable Cell Simulation Platform, FPL, 2003.
- [11] R. Enzler et al., Virtualizing hardware with multi-context reconfigurable arrays, FPL, 2003.
- [12] W. Nebel et al., PISA, a CAD Package and Special Hardware for Pixel-oriented Layout Analysis, ICCAD, 1984.
- [13] M. Weber et al., MOM—map oriented machine, in: E. Chiricozzi, A. D'Amico (Eds.), Parallel Processing and Applications, North-Holland, Amsterdam, 1988.
- [14] M. Herz, Hartenstein, M. Miranda, E. Brockmeyer, F. Catthoor, Memory organization for data-stream-based reconfigurable computing, ICECS, 2002 (invited paper).
- [15] D. Gajski et al., A second opinion on dataflow machines, IEEE Comput. February (1982).
- [16] J. Becker et al., Parallelization in co-compilation for configurable accelerators, in: Proceedings of ASP-DAC, 1998.
- [17] Available from <<http://configware.org>>.

- [18] Available from <<http://flowware.net>>.
- [19] J. Rabaey, Reconfigurable processing: the solution to low-power programmable DSP, in: Proceedings of ICASSP, 1997.
- [20] M. Flynn et al., Deep submicron microprocessor design issues, IEEE Micro July–August (1999).
- [21] R. Hartenstein, A decade of research on reconfigurable architectures, DATE, 2001.
- [22] R. Hartenstein, The microprocessor is no more general purpose, in: Proceedings of the ISIS 1997 (invited).
- [23] R. Hartenstein, Trends in reconfigurable logic and reconfigurable computing, in: ICECS 2002 (invited).
- [24] A. DeHon, The density advantage of configurable computing, IEEE Comput. April (2000).
- [25] C. Mead, L. Conway, VLSI Systems Design, Addison-Wesley, Reading, MA, 1980.
- [26] R. Kress et al., A datapath synthesis system for the reconfigurable datapath architecture, in: ASP-DAC'95.
- [27] J. Becker, T. Pionteck, M. Glesner, An application-tailored dynamically reconfigurable hardware architecture for digital baseband processing, SBCCI, 2000.
- [28] Available from <<http://pactcorp.com>>.
- [29] U. Nageldinger et al., Generation of design suggestions for coarse-grain reconfigurable architectures, FPL, 2000.
- [30] B. Mei et al., Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling, DATE, 2003.
- [31] Available from <<http://kressarray.de>>.
- [32] J. Frigo et al., Evaluation of the streams-C C-to-FPGA compiler: an applications perspective, FPGA, 2001.
- [33] E. Caspi et al., Extended version of: stream computations organized for reconfigurable execution (SCORE), FPL, 2000.
- [34] C. Chang, K. Kuusilinna, R. Broderson, G. Wright, The biggascale emulation engine, FPGA, 2002.
- [35] A. Ast et al., Data-procedural languages for FPL-based machines, FPL, 1994.
- [36] T. Makimoto, The rising wave of field-programmability, in: Proceedings of the FPL 2000, Villach, Austria, 27–30 August 2000, Springer Verlag, Heidelberg, 2000 (keynote).
- [37] N. Tredennick, Technology and business: forces driving microprocessor evolution, in: Proceedings of the IEEE, December, 1995.
- [38] J. Cardoso, M. Weinhardt, From C programs to the configure-execute model, DATE, 2003.
- [39] L. Lamport, The parallel execution of Do-Loops, C. ACM 17 (2) (1974) 83–93.
- [40] D.B. Loveman, Program improvement by source-to-source transformation, J. ACM 24 (1) (1977).
- [41] W.A. Abu-Sufah, D.J. Kuck, D.H. Lawrie, On the performance enhancement of paging systems through program analysis and transformations, IEEE-Trans. C30 (5) (1981).
- [42] U. Banerjee, Speed-up of ordinary programs, Ph.D. Thesis, University of Illinois at Urbana-Champaign, DCS Report No. UIUCDCS-R-79-989, October 1979.
- [43] J.R. Allen, K. Kennedy, Automatic loop interchange', in: Proceedings of ACM SIGPLAN'84, Symposium on Compiler Construction, Montreal, Canada SIGPLAN Notices 19 (6) (1984).
- [44] J. Becker, K. Schmidt, Automatic parallelism exploitation for FPL-based accelerators, in: Hawaii International Conference on System Sciences (HICSS'98), Big Island, Hawaii, 1998.
- [45] J. Becker, A partitioning compiler for computers with Xputer-based accelerators, Ph.D. Dissertation, University of Kaiserslautern, 1997.
- [46] K. Schmidt et al., A novel ASIC design approach based on a new machine paradigm, J. SSC (1991).
- [47] D. Manners, T. Makimoto, Living with the Chip, Chapman & Hall, London, 1995.
- [48] V. Baumgarte et al., PACT XPP—a self-reconfigurable data processing architecture, ERSA, 2001.
- [49] J. Becker, M. Vorbach, Architecture, memory and interface technology integration of an industrial/academic configurable system-on-chip (CSoC), IEEE Computer Society Annual Workshop on VLSI (WVLSI 2003), Tampa, Florida, USA, February, 2003.
- [50] J. Becker, A. Thomas, M. Vorbach, V. Baumgarte, An industrial/academic configurable system-on-chip project (CSoC): coarse-grain XPP/Leon-based architecture integration, in: Design, Automation and Test in Europe Conference (DATE 2003), Munich, March 2003.
- [51] M. Perkowski et al., Learning hardware using multiple-valued logic, IEEE Micro May/June (2002).
- [52] Available from <[http://evonet.dcs.napier.ac.uk/evoweb/news\\_events/conferences/index.html](http://evonet.dcs.napier.ac.uk/evoweb/news_events/conferences/index.html)>.
- [53] M. Vorbach, J. Becker, Reconfigurable processor architectures for mobile phones, in: Reconfigurable Architectures Workshop (RAW 2003), Nice, France, April, 2003.



**Juergen Becker** received the Diploma degree in 1992, and his Ph.D. (Dr.-Ing.) degree in 1997, both at Kaiserslautern University, Germany. From 1997 to 2001 Dr. Becker was Assistant Professor at the Institute of Microelectronic Systems at Darmstadt University of Technology, Germany, where he taught CAD algorithms for VLSI design and did research in Systems-on-Chip (SoC) integration and reconfigurable technologies for mobile communication systems. Since 2001 Juergen Becker is full professor for

embedded electronic systems and head of the Institute for Information Processing (ITIV) at the Universitaet Karlsruhe (TH). His actual research is focused on industrial-driven SoCs with emphasis on adaptive embedded systems, e.g. dynamically reconfigurable hardware architectures. This includes corresponding hardware/software co-design and co-synthesis techniques from high-level specifications, as well as low power SoC optimization. Prof. Becker is managing Director of the International Department at Universitaet Karlsruhe (TH) and Vice Department Director of Electronic Systems and Microsystems (ESM) at the Computer Science Research Center (FZI) at the Universitaet Karlsruhe (TH). He is author and co-author of more than 100 scientific papers, published in peer-reviewed international journals and conferences. Prof. Becker is active in several technical program and steering committees of international conferences and workshops. He is a Member of the german GI and Senior Member of the IEEE. Prof. Becker is Chair of the GI/ITG Technical Committee of Architectures for VLSI Circuits.



niques. Supported by

**Dr.-Ing. Reiner W. Hartenstein** is professor of CS&E at Kaiserslautern University of Technology. He is IEEE fellow, and member of ACM, GI, and ITG. All his academic degrees are from the EE Department at Karlsruhe University, where he later became Associate Professor of CS. He has worked in character recognition, image processing, digital and hybrid systems, computer architecture, hardware description languages, VLSI design methods, reconfigurable computing and related compilation techniques. Supported by various funding agencies his group's

achievements are: the hardware design language KARL and its graphical companion language ABL, the Xputer machine paradigm, the KressArray family—a reconfigurable generalization of systolic arrays, and related structured configware design tools, configware/software co-compilation, and others. He has helped to organize numerous international conferences as a program chair (10 times), industrial chair, steering committee member, general chair, program committee member and/or reviewer. He is founder of the PATMOS international workshop series, and of the German Multi University E.I.S.-Project (Mead-&-Conway style VLSI-Design), and co-founder of IFIP working group 10.2 (now 10.5), the FPL International Conference Series, and of EUROMICRO. He has authored, co-authored, or co-edited 14 books and almost 400 papers.