A Roadmap to New Horizons in High Performance Computing

Reiner Hartenstein
Kaiserslautern University of Technology
http://hartenstein.de

Abstract. The paper advocates a dichotomy of machine paradigms for high performance computing, and, for embedded system design to bridge the severe communication gap between classical computing and new directions [1] currently becoming mainstream. The paper drafts ways to update our CS curricula in order to avoid, that our graduates will not be qualified for the IT labor market in a few years from now..

In a growing number of high performance computing (HPC) application areas the desired performance is hard to reach by "traditional" HPC. For instance, the gravitating n-body-problem is one of the grand challenges of theoretical physics and astrophysics [2]. Also hydrodynamic problems fall in the same category, where often numerical modeling can be used only on the fastest available specialized hardware. Analytical solutions exist only for a limited number of highly simplified cases. For interpretation of dense centers of galactic nuclei observed with the Hubble Space Telescope to unite the hydrodynamic and the gravitational approach within one numerical scheme. Until recently this limited the maximum particle number to about a 10^5 even on largest supercomputers available. The situation improved by the GRAPE special purpose computer [3]. To improve the flexibility a hybrid solution has been introduced with AHA-GRAPE, which includes auxiliary morphware (FPGA-based processors) [4]. A commercially much more important morphware usage example is cellular wireless communication, where performance requirements grow faster than Moore's law [5] [6].

CS curricula keep students away. Hybrid solutions combining morphware and classical instruction-stream-based CPUs, becoming more and more important, are currently a domain of hardware professionals experienced in embedded System design. However, the severe communication gap, also caused by obsolete CS curricula, still keeps programmers away from this highly promising area. More details about this problem area are discussed later in this paper.

Procedural versus structural programming. The main reason of the communication gap between this area and classical computing are the different mind sets. procedural thinking (programming in time) on the programmer side, versus structural modelling on the hardware and morphware side (computing in space and time; see fig. 1). This means ***instruction-stream-based computing from software sources*** running on von-Neumann-like CPUs, versus ***data-stream-based computing programmed from flowware sources*** (data scheduling instead of instruction scheduling).

Morhpware: a fundamental paradigm shift. This paper tries to illustrate the introduction of morphware as a new basic computational paradigm, and tries to help to penetrate the communication barrier between the R&D scenes of embedded systems, being familiar with morphware application, and the scientific scenes of high performance computing. Also in embedded systems von-Neumann-like controllers or

domain	procedural	structural		
computing in ...	time only	space and time		
program source	software*	hardwired		reconfigurable
		currently	emerging	
		(hardware +) software **	(hardware +) flowware	configware + flowware
"instruction" fetch	at runtime	before fabrication		at loading time
data "fetch"		at runtime		

*) only one source needed **) software "simulates" flowware

Fig. 1: Programming procedural vs. structural

microprocessors are very important because of their enormous flexibility. Its simple machine paradigm is also an important common model, not only for education.

Software industry's secret of success. The von Neumann paradigm is the driving force behind the success of software industry, by its simplicity helping to focus language design and compilation techniques to a usefully narrow design space, and, by helping to educate masses of programmers. Prerequisite of this success is the fact, that its operational principles are RAM-based, yielding a seemingly almost unlimited flexibility, and scalability. Another driving force is compatibility provided by processor marketing policy and dominance.

Distributed Memories. The rapidly growing new R&D area and IP core market segment of distributed memory [7] have arrived just in time to provide ways to vanquish the consequences of the von Neumann bottleneck by skillfully tailored embedded system architectures. Two main directions can be distinguished: generating special architectures with application-specific address generators [8], or, more flexible architectures with general purpose address generators [9]. Example application areas are accelerators for DSP, multimedia, or wireless communication. From the SoC (System on a Chip) system level point of view such architectures and their address generators are sources and sinks of multiple ***data streams*** (fig. 2), "programmed" by ***flowware*** - in contrast to software which programs ***instruction streams***. For terminology also see fig. 2 [10]. The popularity of distributed memory is also a motivation to herald a new machine paradigm, which is driven by data-streams instead of instruction streams. This will be discussed in detail later in this paper.

Computing Architecture Crisis. Commercial break-throughs of the microprocessor mainly stem from the progress of semiconductor technology but have hardly affected the blinders limiting the scope of conference series like ISCA (Int'l Symp. on Computer Architecture) or MICRO, which remained almost 100% only von-Neumann-based all the time. Embedded computing systems and PCs demonstrate, that more and more most silicon real estate is occupied by hardwired accelerators, to support the von Neumann processor. The crisis of computer architecture is obvious, not only from the program statistics history of the ISCA series of annual international conferences [11] (fig. 3). But also this accelerator hardware itself more and more moves into a crisis.

The 2nd Hardware Design Crisis. Also hardware accelerator design enters a severe crisis, its 2nd crisis [12]. Exploding design cost is increasing much faster than circuit complexity. Moore's Law is becoming an increasingly misleading predictor of future developments. In a number of microelectronics application areas the continuing technology progress sometimes

platform		program source	machine paradigm
hardware		(not programmable)	(none)
morphware	fine grain reconfigurable	***configware***	
	coarse grain reconfigurable		
	reconfigurable data-stream-based computing	***configware & flowware***	anti machine
hardwired processor	data-stream-based computing	***flowware***	
	instruction-stream-based computing	software	von Neumann

Fig. 2: Terminology.

creates more problems, than it solves. More and more circuit design cleverness is required because each new technology generation comes with additional new parasitics.

A roadmap around the design crisis. A route to overcome the design and cost crisis is to introduce morphware platforms (terminology: fig. 2), where no specific silicon is needed, which saves mask cost and other NRE cost. Mapping an application onto an FPGA (field-programmable gate array) is a design activity at logic level (gate level), Since in fine grain morphware single bit CLBs (configurable logic blocks) maybe configured into a logic gates. The FPGA market has reached a volume of about 7 billion US-dollars worldwide. Practically everything may be migrated onto morphware.

A second RAM-based platform. Now we have a second RAM-based platform: morphware, where structure definition is RAM-based, instead of the instruction execution schedules as known from von Neumann principles. It turns out, that von Neumann does not support such soft hardware. This is the basis of a emerging disruptive trend which has (almost) all prerequisites to repeat the success story of the software industry: by a configware industry needing the support of a revolutionary new machine paradigm.

Flexibility and time to market. Another advantage of morphware is FPGA flexibility. Since a logic gate is universal, also FPGAs are general purpose. A single FPGA type may replace a variety of IC types. Whereas turn-around times for changing hardwired IC designs takes up to several months, it is by orders of magnitude shorter on FPGAs, because personalization is done after fabrication, even at the customer's site. Patches may take only days, hours, or even minutes. So FPGAs have the potential for debugging or upgrading the last minute - even at the customer's site, or, remotely over the internet [13] or even by wireless communication [14].

Soft CPUs. Configware providers meanwhile offer CPUs as soft IP cores (configware versions of CPUs or micro controllers) 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 such soft CPU IP cores can be also generated from VHDL or Verilog sources originally targeted at a hardwired implementation.

Old fab line + algorithmic cleverness. More recently the research is heading toward low power FPGAs from the algorithms side. Jan Rabaey et al. have shown, that by transfer of an algorithm from a DSP to an experimental low power FPGA an improvement of factors between 5 and 22 have been obtained [15] [16]. When also the optimum technology is selected, a

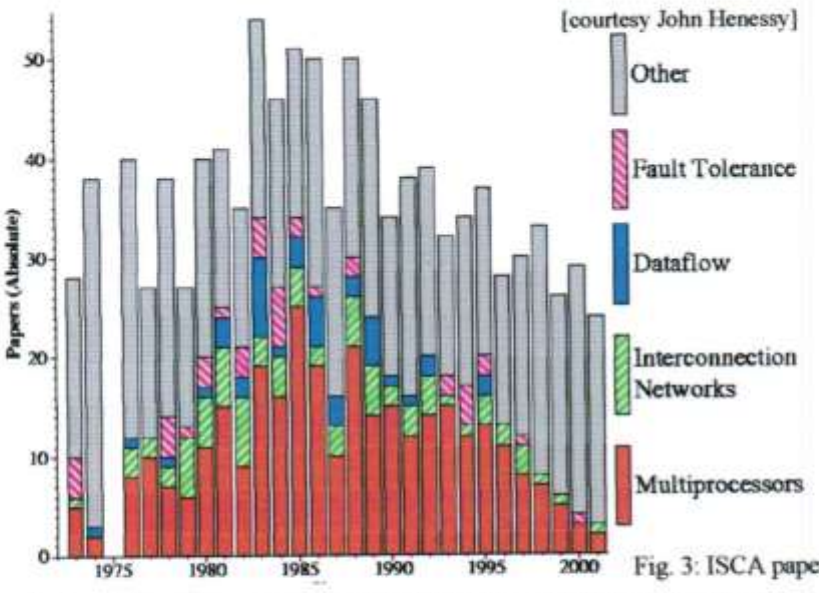

Fig. 3: ISCA paper statistics.

reduction of clock frequency by a factor of n yields a reduction of power dissipation by a factor of n^3 [17]. The only unsolved question is, how long this technology will be still available.

Configware industry is emerging as a counterpart to software industry. Part of the configware is provided by the FPGA vendors for free. But the number of independent configware houses (soft IP core vendors) and design services is growing. **A** good designer productivity and design quality cannot be obtained without good configware libraries with soft IP cores from various application areas.

New business model. Like micro-processor usage, also programming reconfigurable platforms is RAM-based, but by structural programming instead of procedural programming. Now both, host and accelerator are RAM-based and as such also available on the same chip: a new approach to SoC design. Also morphware is RAM-based and supports personalization at the customer's site, which means a change of the business model, providing a flexibility, which is not possible by classical ASICs. Since the program code is fundamentally different from software, it is called **„Configware"** (fig. 2 [10]). Now for patches and upgrades both can be downloaded: software and configware.

Reconfigurable Computing. From a decade of world-wide research on Reconfigurable Computing another breed of reconfigurable platforms is an emerging commercial competitor to FPGAs. Whereas *RL* based on fine grain morphware (FPGAs) uses single bit wide CLBs, *Reconfigurable Computing* (RC) based on coarse-grained morphware, uses rDPUs (reconfigurable data path units), which, similar to ALUs, have major path widths, like 32 bits, for instance - or even rDPAs (rDPU arrays) [18] [19]. In contrast to logic design level of mapping onto fine grain morphware, mapping applications onto coarse-grained morphware belongs to functional abstraction levels - using compilation techniques instead of logical synthesis. Important applications in areas like telemetries, multimedia and communication, are derived from the performance limits of the "general purpose" vN processor. For very high throughput requirements RC morphware is the

drastically more powerful and more area-efficient alternative, also about one order of magnitude more energy-efficient than fine grain, and has drastically reduced reconfigurability overhead [18]. Commercial versions are available from PACT Corp. [20]

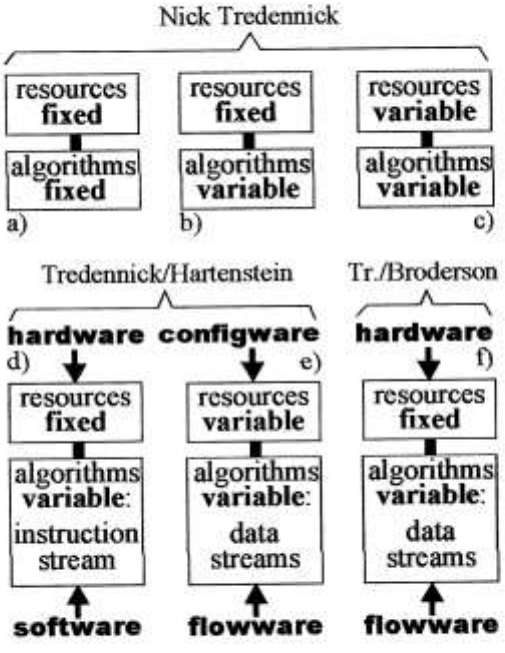

Fig. 4: Nick Tredennick's digital system classification scheme: a) hardwired, b) programmable in time, c) reconfigurable; d) von-Neumann-like machine paradigm, e) reconfigurable anti machine paradigm, f) Broderson's hardwired anti machine.

Flowware. We can distinguish two categories of RC platforms [18] [19]: microminiaturized versions of traditional parallel computing systems (not subject of this paper), and, datapath networks driven by data streams. Data streams have been popularized by systolic and super-systolic arrays, and projects like SCCC [21], SCORE [22], ASPRC [23], and, BEE [24]. In a similar way like instruction streams can be programmed from *software sources*, data streams are programmed from *flowware sources* (also see fig. 2). High level programming languages for flowware [25] and for software join the same language principles. A von Neumann machine needs *just software* as the only programming source, since its hardwired resource is not programmable. A reconfigurable data-stream-based machine, however, *needs two programming sources*: *configware* to program (to reconfigure) the operational resources, and, *flowware* to schedule the data streams (see fig. 2).

The Makimoto / Tredennick model. To fully understand the role of morphware within modern SoC design it is useful to model the history of IC application. Tredennick's classification scheme (fig. 4 a, b, and, c) [18] goes conform with Makimoto's wave model [26] [27]. This also explains why morphware needs two different programming sources (fig. 5 b): configware and flowware, whereas instruction-driven platforms need only a single source: software (fig. 5 a). This refines Tredennick's scheme into the one which is illustrated by fig. 4 d and e. The data-stream-based anti machine paradigm ([28] [29] fig. 5 a, and, fig. 4 e) is needed, because the instruction-stream-based von Neumann paradigm only supports hardwired platforms, but no morphware platforms [18]. Morphware has no instruction streams at run-time, since configuration which replaces the instruction fetch, is carried out before run time [18]. Even run time configurable systems distinguish between configuration mode and run time mode. Configuration cannot be derived from software, because it is a structural issue and not a procedural one. It is interesting, that even hardwired

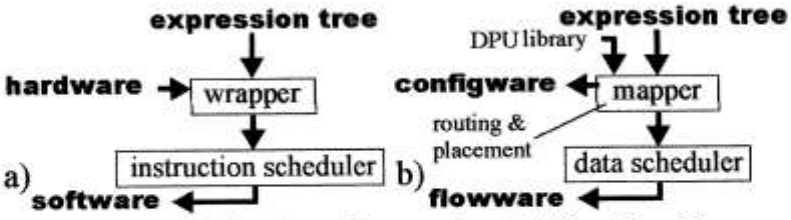

Fig. 5: Compilation: a) von-Neumann-based, b) for anti machines.

machine platforms can be driven by data-streams [24]. This is modelled by the Tredennick / Broderson Scheme (fig. 4 f) - another refinement of Tredennick's scheme.

The dichotomy of common architectural models. Going one abstraction level down from the Makimoto / Tredennick models yields a dichotomy of common architectural models based on the two machine paradigms: the von Neumann machine (fig. 6 a) and the anti machine. All models consist of two blocks: a functional block (FB) and a memory block (MB). On the von Neumann side (fig. 6 a), exactly one sequencer resides in the FB and the MB has only one memory bank. The interface between FB and MB is the "von Neumann bottleneck" not permitting any parallelism (fig. 6 a). On the anti machine side (fig. 6 b, c, and d) one (fig. 6 b,) or several sequencers (fig. 6 c, and d) reside in the MB, whereas the FB has no sequencers. Note the asymmetry between machine and anti machine (compare fig. 6 a with fig. 6 b). The anti machine side has a much wider design space. It also supports also multiple banks and multiple sequencers within the FB, like one datapath with multiple memory banks (fig. 6 c), or multiple datapaths with multiple memory banks (fig. 6 d).

Data-stream-based Computing. The models having been introduced above are important models to alleviate understanding implementation flows. The *Anti Machine* Paradigm [28] [29] [30] (based on *data sequencers* [7] [9] [31]) is for morphware [32] [33] and even for hardwired data-stream-based machines the better counterpart (fig. 6 b) of the von Neumann paradigm (based on an instruction sequencer, fig. 6 a). Instead of a CPU the anti machine has only a DPU (datapath unit) or a rDPU (reconfigurable DPU) without a sequencer (fig. 6 b). The anti machine model locates data sequencers on the memory side (fig. 6 b). Unlike "von Neumann" the anti machine has no von Neumann bottleneck, since it also allows multiple sequencers (fig. 6 c) to support multiple data streams interfaced to multiple memory banks (fig. 6 c), allowing operational resources much more powerful than ALU or simple DPU: major DPAs or rDPAs (fig. 6 d). There is a lot of similarities, so that each of the two models is a kind of mirror image of the other model - like matter and antimatter.

The coming configware industry? The success of software industry is RAM-based, supporting rapid personalization by procedural programming at the customer's site. Already a single morphware device provides massive parallelism at logic level or at operator level, which usually is much more efficient than process level parallelism possible only with multiple von Neumann (vN) processors being affected by a severe crisis [11]. But this paradigm shift is still widely ignored: FPGAs and RC do not repeat the RAM-based success story of the software industry. There is not yet really a major configware industry, since mapping applications onto FPGAs mainly uses hardware synthesis methods. Because of lacking awareness of this paradigm switch there is not yet a configware industry.

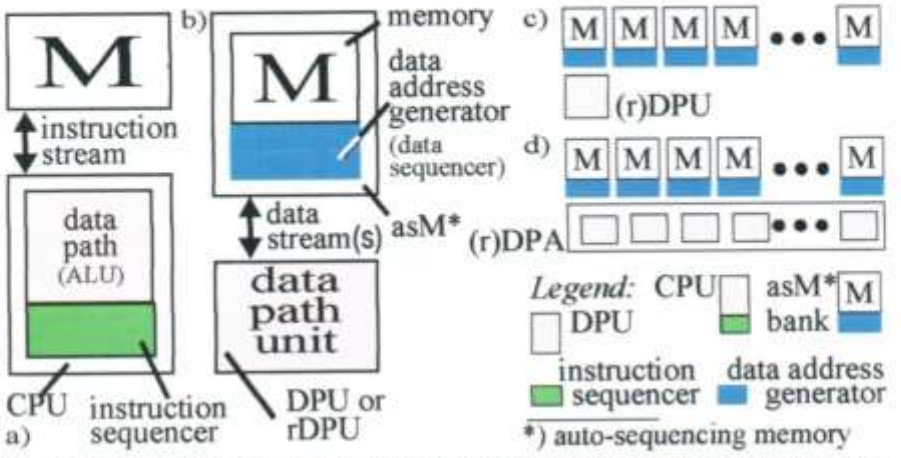

Fig. 6: Basic machine paradigms: a) von Neumann, b) data-stream-based anti machine w. simple DPU, c) w. rDPU and distributed memory, d) w. DPU array (DPA or rDPA)

A new mind set is needed. However, the new trend is coming along with the challenge to overcome the incising limitations of users' dominant "procedural-only" mind set of computing science in the von Neumann era. There is a lot of similarities between the worlds of von Neumann and its anti machine paradigm. But like in particle physics, there are asymmetries - making it a bit inconvenient to teach the coming dichotomy of computing sciences. A new mind set is needed to overcome our inconvenience and to trigger a mass movement.

No, we are not ready for the break-through. It has been predicted, that by the year 2010 about 95% of all programmers will implement applications for embedded systems. However, the algorithmic cleverness and other important skills are missing. Not only computing sciences are not yet ready. Our basic curricula do not teach, that hardware and software are alternatives, and do not teach how hardware / software partitioning is carried out. Our basic curricula still widely ignore the existence of reconfigurable platforms and their use for embedded digital system design.

Missing algorithmic cleverness. In low power IC design (integrated circuit design), the leakage current and other technological phenomena, getting much worse with each new technology generation, are a major design problem. For instance, it may be better to implement a high performance application on 100 processors running at 200 MHz than on one processor running at 20 GHz. But from the EDA (electronic design automation) and programming point of view it is extremely difficult to achieve, that only 100 processors are needed to implement this parallel version. The algorithmic cleverness is missing - in practice and in our curricula. Algorithms and data structure educators should extent their area far beyond pointers, queues and stacks.

Qualification Deficit. We have a lack of experience using FPGAs for computationally intense applications, lack of algorithmic cleverness to translate into morphware, immature FPGA-based design tools. We urgently need a consensus on terminology, and need to fight, that reconfigurable computing and morphware are not treated as a contaminant which when entering computing disciplines only meets troops of anti bodies ready to keep us out. We have

to take the responsibility to take care, that CS graduates are not unqualified to meet the requirements of the labor market around the year 2010 and later.

All ingredients are available to update our CS curricula. Research results from the past 30 years are ready to be used from a wide variety of areas, and just need to be integrated into existing curricula. On the roadmap to reach this goal a second machine paradigm will be very helpful to bridge the gap of mind sets: instruction streams versus data streams. We need a special interest group on a dichotomy of machine paradigms to create awareness, and, to provide a strong pressure group to remove obstacles from the road to a very powerful new direction in high performance computing.

1. Literature

1. http://fpl.org
2. G. Lienhart: Beschleunigung Hydrodynamischer N-Körper-Simulationen mit Rekonfigurierbaren Rechensystemen; Joint 33rd Speedup and 19th PARS Workshop, Basel, Switzerland, March 19 - 21, 2003
3. N. Ebisuzaki et al.; 1997 Astrophysical Journal, 480, pp. 432,
4. R. Männer et al.: AHA-GRAPE: Adaptive Hydrodynamic Architecture - GRAvity PipE; FPL 1999
5. J.Becker: Configurable Systems on Chip; Proc. IEEE ICECS 2002
6. J. Rabaey (keynote): Silicon platforms for the next generation wireless systems; FPL 2000
7. M. Herz et al.: (invited): Memory Organization for Data-Stream-based Reconfigurable Computing; Proc. ICECS 2002,
8. F. Catthoor et al.: Custom Memory management Methodology; Kluwer 1998
9. M. Herz, et al.: A Novel Sequencer Hardware for Application Specific Computing; ASAP'97
10. http://morphware.net - http://configware.org - http://flowware.net - http://data-streams.org
11. J: Hennessy (keynote): Int'l Symp. on Computer Architecture (ISCA) Barcelona, Spain. June 1998
12. R. Hartenstein (keynote): Are we really ready for the Break-through?; Proc. RAW 2003
13. S. Guccione et al.: JBits Java-based interface for reconfigurable computing; Proc. MAPLD 1999
14. A. Dawood, N. Bergmann: Enabling Technologies for the Use of Reconfigurable computing in Space; 5th Int'l. Symp. on Signal Processing and its Applications, vol. 2,
15. V. George, J. Rabaey: Low-Energy FPGAs; Kluwer, 2001
16. J. Rabaey: Reconfigurable Processing: The Solution to Low-Power Programmable DSP, ICASSP 1997
17. M. Flynn et al.: Deep Submicron Microprocessor Design Issues; IEEE Micro July-Aug '99
18. R. Hartenstein (embedded tutorial): A Decade of Research on Reconfigurable Architectures - a Visionary Retrospective; DATE 2001, Munich, March 2001
19. R. Hartenstein (invited embedded tutorial): Coarse Grain Reconfigurable Architecture; ASP-DAC 2001
20. http://pactcorp.com
21. J. Frigo, et al.: Evaluation of the streams-C C-to-FPGA compiler: an applications perspective; FPGA 2001
22. E. Caspi et al.: Stream Computations Organized for Reconfigurable Execution (SCORE): FPL '2000
23. T. Callahan et al.: Adapting Software Pipelining for Reconfigurable Computing; CASES 2000
24. C. Changet al.: The Biggascale Emulation Engine; summer retreat 2001, UC Berkeley
25. A. Ast, et al.: Data-procedural Languages for FPL-based Machines; Proc. FPL 1994
26. T. Makimoto (keynote): The Rising Wave of Field-Programmability; Proc. FPL 2000
27. R. Hartenstein (invited paper): The Microprocessor is no more General Purpose; ISIS 1997
28. R. Hartenstein (keynote): Disruptive Trends by Custom Compute Engines. Proc. FPL 2002
29. R. Hartenstein (keynote): Reconfigurable Computing: urging a revision of basic CS curricula; Proc. ICSENG'02, Las Vegas, USA, 6-8 August, 2002
30. R. Hartenstein et al.: A Novel Paradigm of Parallel Computation and its Use to Implement Simple High Performance Hardware; Future Generation Computer Systems 7 91/92, p. 181-198, - invited reprint from Proc. InfoJapan'90, Tokyo, Japan, 1990
31. J. Becker et al.: A General Approach in System Design Integrating Reconfigurable Accelerators; Proc. IEEE ISIS 1996
32. R. Hartenstein et. al.: A Novel ASIC Design Approach Based on a New Machine Paradigm; IEEE J.SSC, 26/7, July 1991 - invited reprint from Proc. ESSCIRC 1990
33. R. Hartenstein (invited): Reconfigurable Computing and its Impact; 2003 intel ORCC workshop