The Expensive von Neumann Paradigm: is its Predominance stil Tolerable?

Reiner Hartenstein, IEEE fellow, TU Kaiserslautern

FPT'09; Sydney, Australia, December 9-11, 2009

Abstract. *Mainly three different highly problematic issues come along with the predominance of the von Neumann (vN) paradigm: a very high and growing energy consumption, the software crisis still not solved after 40 years, and more recently, the multicore programming crisis. Based on a well proven technology meanwhile an alternative programming paradigm is available: reconfigurable computing, promising speed-up factors and energy savings by up to several orders of magnitude. To preserve the affordability of all our computer-based infrastructures we need a world-wide changeover of many applications from software running on von Neumann machines, over to configware running reconfigurable platforms. However, a sufficiently large programmer population qualified for such re-programming jobs is far from existing. To program hybrid systems including both, vN processors and reconfigurable platforms, the paper proposes a twin dichotomy strategy for twin-paradigm basic and advanced training.*

A CPU-centric Flat World

Energy Consumption of Computers. The future of our world-wide total computing ecosystem is facing a mind-blowing and growing electricity consumption, and a trend toward growing cost and shrinking availability of energy sources. The electricity consumption by the internet alone causes more Greenhouse Gas emission than the world-wide air traffic. If current trends continue, a recent study sees an increase by a factor of 30 within 22 years[1]. The growth of the internet by cloud computing[2], video on demand, and other newer services is also illustrated by spectacular new server farms and their overwhelming electricity consumption[3]. But the internet is only a part of our total computer-based infrastructures. Although estimations are under controversial discussion, power consumption has become an industry-wide issue for computing[4]. Also the microprocessor industry's strategy change from (single-core) higher clock speed to multicore[5], the biggest challenge our computer industry faces today, is an energy issue, since a single chip's power dissipation came up to close to 200 watts. Driven by economic factors industry will switch to more energy-efficient solutions in computing. Incremental improvements are on track, but „we may ultimately need revolutionary new solutions" [Horst Simon[4]].

Arthur Schopenhauer: "Approximately every 30 years, we declare the scientific, literary and artistic spirit of the age bankrupt. In time, the accumulation of errors collapses under the absurdity of its own weight."

RH: "Mesmerized by the Gordon Moore Curve, we in CS slowed down our learning curve. Finally, after 60 years, we are witnessing the spirit from the Mainframe Age collapsing under the von Neumann Syndrome."

Computing Model from the Mainframe Age. The predominance of tremendously inefficient software-driven von-Neumann-type (vN) computers, employing a fetch-decode-execute cycle to run programs, is the cause of this massive waist of energy. Under its top-heavy demand, software packages often reach astronomic dimensions (up to 200 million lines of code) and running such packages requires large power-hungry slow extra memory microchips ("The Memory Wall"[6]). A major problem is the predominance of a CPU-centric monoprocessing mind set (table 1).This reminds to the Aristotelian geo-centric flat world model. since the year 2009 is the International Year of Astronomy. At a US state governor summit meeting Bill Gates said, that the US educators teach computing like for the mainframe era, and, that he cannot hire such people. But meanwhile the „CPU" (central processing unit) became less central, needing accelerator support (ASICs) like, for

computer machine model of the mainframe era	resources		sequencer		
	property	programming source	property	programming source	state register
„CPU" instruction set processor	hardwired	-	**programmable**	**Software** (instruction stream)	**program counter**

Table 1: The Computer System Model of the Mainframe Era.

CPU / accelerator symbiosis		resources		sequencer		
		property	programming source	property	program source	state register
1	**ASICs** hardwired accelerators	hardwired	-	hardwired	-	
2	instruction set **CPU** processor	hardwired	-	**programmable**	**Software** (instruction stream)	**program counter**

Table 2: The Post-Mainframe Machine Model: SE not affected © 2009, Reiner Hartenstein

instance, to run its own display. The tail is wagging the dog (table 2). The CPU-centric model (table 1) became obsolete, but so far only for hardware designers. For mainstream software engineering (SE) the practice did not change prior to multicore: where programming microprocessors was based on a sequential mind set. Programmers had it easy. The same old software ran on the new chips much faster: a free ride on Moore's curve.

The Software Crisis. We are still affected by the „Software Crisis" although this term has been coined already in 1968[7]. The software crisis has been and still is manifested by software difficult to maintain, very inefficient, of low quality, not meeting requirements, and by projects running over-budget and/or over-time, being unmanageable or canceled. The software crisis is still far from being tamed. Dijkstra explained its causes[8] by the overall complexity made possible by growing processor performance, i. e. by Moore's Law. More recently, Microsoft's Nathan Myhrvold even argues that Nathan's Law is driving Moore's Law by the demand software creates as a gas which completely fills its containers. To go even further: this gas is not only filling growing memory microchips (Moore's law), but also growing disc space by Kryder's Law[9], leaving Moore's law behind as a snail's pace. This gas also fills the internet with its communication bandwidth capacity for growing numbers of surfers and of e-mails growing in size, video on demand, internet radio and TV, voice over IP, etc., and growing numbers of smart mobile phones like iPhone, Blackberries, etc. is using it and its server farms[3] and cloud computing space[2] (who's law is this?).

The Multicore Crisis means an intensification of the software crisis by the introduction of vN-type parallelism having been mainly only practiced so far in supercomputing by very expensive interdisciplinary program development teams, often with massive productivity problems. Instead of a single processor core with higher clock speed, the number of (slower low power) cores will double for every microprocessor generation. 32 and more cores per chip are pre-announced. Already six or eight CPUs mostly talk to each other instead of getting work done[5]. The old software may run slower. To take advantage of the additional cores most applications have to be rewritten, unfortunately also introducing new types of bugs[5,10]: multithreading considered harmful[11]. Often the algorithm has to be changed. However, typical programmers are not trained to think toward parallel processing.

The von Neumann Syndrome is the reason of the Softw are Crisis, and the massive energy consumption of computers

The von Neumann Syndrome. A lot of proposals have been made to tame the software crisis like by Disjkstra's famous paper „The goto considered harmful"[12]. But we all agree, that no silver bullet has been found yet. Just about a decade ago Niklaus Wirth's law says[13]: „Software gets slower faster than hardware gets faster." - a plea for lean software. But to obtain „lean software" is extremely difficult under the still pre-dominant instruction-stream-based CPU-centric vN-based sequential-only mind set from the mainframe age, which is characterized by multiple levels of overhead, like e. g. address computation overhead and multiplexing overhead[14]: the universal bus considered harmful[15]. The overhead is growing progressively with the degree of vN-type parallelism. For this cumulation of inefficiencies Prof. Ramamoorthy from UC Berkeley has coined the term „von Neumann Syndrome".

The Term „Software" stands for very memory-cycle-hungry instruction streams: coming with multiple levels of overhead: the von Neumann Syndrome. The vN architecture principles, based on a fully sequential mind set, are the reason of the von Neumann Syndrome: The vN machine is a kind of emulation of a tape machine on RAM (random access memory), where the instruction streams are the emulations of tapes. The von Neumann Syndrome is the reason of the Software Crisis (and the Multicore Crisis), and the massive energy consumption of computers. The only promising

The term „Softw are" stands for extremely memory-cycle-hungry instruction streams — coming with multiple levels of overhead: the von Neumann Syndrome

contemporary computer system machine model		resources		sequencer		
		property	programming source	property	program source	state register
1	**ASICs** hardwired accelerators	hardwired	-	hardwired	-	
2	instruction set **CPUs** processors	hardwired	-	**programmable**	**Software** (instruction stream)	**program counter**
3	reconfigurable **RCUs** accelerators	**programmable**	**Configware** (configuration code)	**programmable**	**Flowware** (data streams)	**data counter(s)**
4	flowware-programmable **FPUs** accelerators[24]	hardwired	-	**programmable**	**Flowware** (data streams)	**data counter(s)**

Table 3: Our Contemporary Computer System Machine Model © 2009, Reiner Hartenstein

solution is a paradigm shift, away from the massively inefficient vN platform, away from software, over to another programming paradigm, not behaving like a gas. We can pick this paradigm from reconfigurable computing (RC). To stress the fundamental difference of this paradigm we should **not** call it „software".

Reconfigurable Computing: the Silver Bullet?

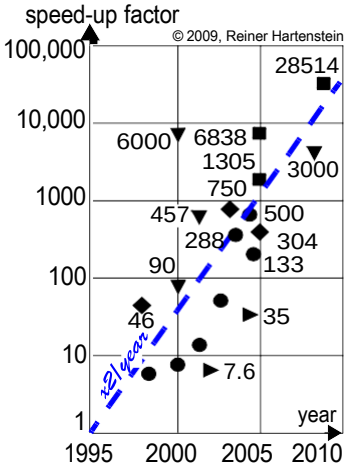

Fig. 1: Some speed-up factors from CPU to FPGA migration[19].

The Impact of Reconfigurable Computing (RC). Meanwhile ASICs used as accelerators have massively lost market shares in favor of reconfigurable accelerators. Now FPGA projects outnumber ASIC projects by 30 to 1 [Gartner, 2009], or by 50 to 1 [another estimation[16]]. FPGA stands for „Field-Programmable Gate Arrays". FPGAs are structurally programmed from ***„configware"*** sources, which are fundamentally different from the instruction-stream-based „software" sources (table 3). FPGAs come with a different operating system world organizing data streams and swapping configware macros for partially and dynamically reconfigurable platforms[17,18]. On „FPGA" Google finds 10 million hits. Introduced in 1984 and now a 5 billion US-$ world market, FPGAs are a well proven technology rapidly heading for mainstream and also used in supercomputing, not only by Cray and Silicon Graphics. FPGAs are also standard gear in many high performance streaming applications like medical imaging, routers, market data feeds, military systems, and others.

Massive Speedup. From CPU software to FPGA configware migrations for a variety of application areas, speedup factors from almost 10 up to more than 3 orders of magnitude have been published (fig. 1) by a number of papers I have referenced[19]. More recently, for instance, a factor of 3000 has been obtained in 3-D image processing for computer tomography by Siemens Corporation. Biology applications[20] showed speed-up factors up to 8723 (Smith-Waterman pattern matching[21]: table 4). Multimedia reports up to 6000 (real-time face detection). Cryptology reports for DES breaking a speed-up factor of 28514[21] (table 4). After migration the same performance requires drastically less equipment. For instance, we may need only one rack or half a rack and no air conditioning, instead of a hangar full of racks. Unlike software, configware is not a gas!

Massively Saving Energy. The energy saving factors tend to be roughly 10% of the speed-up factor: the golden bullet for saving energy? The most spectacular power savings published so far we see in table 4: 779x for DNA and protein sequencing and 3439x for DES breaking[21].

Application	Speedup Factor	Savings		
		Cost	Power	Size
DNA and protein sequencing	8,723x	22x	779x	253x
DES breaking	28,514x	96x	3,439x	1,116x

Table 4. Migration from Beowulf cluster to SGI Altix 4700/RASC[21]

Paradigm	program source(s)	software crisis?	source behaving like a gas (Nathan's law) ?	memory-cycle-hungry slow instruction streams	memory wall effect?
von Neumann	Software	massive	yes	yes, massively	yes, massive
Reconfigurable Computing	Configware and Flowware	no	no	**no**:no instructionfetch at run time	mostly: no

Table 5: Benefit of a Reconfigurable Computing Revolution

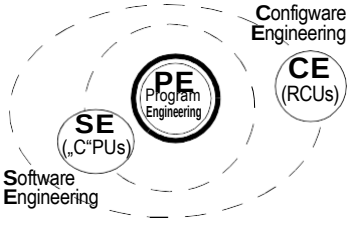

Fig. 2: „heliocentric" CS model.

Not a Gas. FPGA technology is worse than microprocessors: slower clock, and massive reconfigurability overhead. This means massively higher performance with a worse technology **— *The Reconfigurable Computing Paradox.*** The explanation: the overwhelming impact of the Software Crisis, also caused by von Neumann Syndrome. Configware and Flowware are no gases: Nathan's law is invalid (table 5). Reconfigurable platforms have no instruction fetch at run time. By orders of magnitude less memory is needed, which often fits on the same chip: faster by 3 orders of magnitude: no memory wall **—** supported by COTS FPGA chips with up to more than a hundred fast memory blocks on board.

Displacing the CPU from the Center

Now Accelerators are Programmable. Again our common computer system model has changed[22] (table 3: rows 1-3): accelerators have become programmable[22]. „Software" is replaced by „ Program". This Copernican CS world model (fig. 2) has consequences also from the SE point of view: we now have two different kinds of programmable platforms: instruction set processors („C"PUs) and reconfigurable computing units (RCUs). In contrast to a CPU programmed only by ***software***, a RCU needs two program sources (table 3: row 3): ***„configware"*** and ***„flowware"***, which can be compiled together (fig. 3) or separately (fig. 3 a, b). Both sources are not at all instruction-stream-based. ***Configware*** is not procedural nor imperative. ***Flowware*** has been derived from the ***data stream*** definition for systolic arrays already in the late 70ies. The term „flowware" makes sense to avoid confusion with the term „dataflow"[23] and with a variety of data sophisticated local stream definitions. We now have to interface to each other three different programming sources: traditional „software" for programming the CPUs, configware for programming the accelerators, and flowware to program the data streams (fig. 3). By the way, flowware may also be used without configware (fig. 3 b) for hardwired machines (table 3: row 4), e. g. like BEE[24].

By orders of magnitude more performance with worse technology: The Reconfigurable Computing Paradox — caused by the von Neumann syndrome

Our Contemporary Computer System Model (table 3: rows 1-3) now includes 3 programming paradigms: 1 structural programming paradigm: ***configware***, as well as 2 procedural programming paradigms: ***software*** for programming the CPU to schedule instruction streams, and ***flowware*** to schedule data streams, mainly running through the accelerators. The CPU model is the ***von Neumann machine paradigm*** for sequencing by program counter. But flowware is based on sequencing by data counters from/to memory. This counterpart and twin brother of the von Neumann paradigm is the ***Flowware machine paradigm***. Adding to software two more program sources (configware and flowware) looks more complicated than it really is. The dichotomy of this twin paradigm model (table 6) helps a lot to make the new methodology easily graspable. By going from hardwired accelerators to programmable (reconfigurable) accelerators the traditional hardware software chasm within CS education is turning into new horizons of configware / software interfacing. Why does software engineering still ignore this highly potent silver bullet candidate?

Programming beyond Software

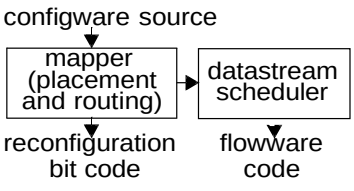

Fig. 3: Configware Co-compiler.

It's a Twin-Paradigm World. Most primitives of a software language and of a flowware language are mainly the same (the twin-paradigm dichotomy: table 6). Different is only the semantics: A software language deals with sequencing a program counter. A flowware language programs one or more data counters (generalized DMA = direct memory access): for implementing data streams. This is the only asymmetry: only one program counter (located in the CPU), whereas datastream machines may have several data counters running in parallel (located in asM data memory blocks: **a**uto-**S**equencing **M**emory - generalized of DMA). Two exceptions make flowware languages

#	Language Features	Software Languages	Flowware Languages
1	sequencing managed by	read next instruction, goto (instruction address), jump (to instruction address), instruction loop, and nesting, escapes instruction stream branching	read next data item, goto (data address), jump (to data address), data loop, and nesting, escapes data stream branching
2	parallelism	***no*** *parallel loops*	***yes,*** *parallel loops* (by multiple data counters)
3	data manipulation	yes	***no*** (hardwired or synth. from configware language)
4	state register	single **program counter** (located in CPU)	1 or more **data counter(s)** (located in ***asM*** memory)
5	instruction fetch	memory cycle overhead	***no*** memory cycle overhead
6	address computation	massive memory cycle overhead (depending on application)	reconfigurable address generator(s) in asM: ***no*** memory cycle overhead

Table 6. Software Languages versus Flowware Languages.

more simple than software: 1) no data manipulation, since being set up by reconfiguration via configware (or being hardwired); 2) parallelism inside loops, since a datastream machine may have several data counters.

Putting old Ideas into Practice. We need to rearrange undergraduate courses, following the advice of David Parnas: „The biggest payoff will not come from new research but from putting old ideas into practice and teaching people how to apply them properly". The flowware paradigm is very old stuff, based on the data stream definition from systolic arrays published in the late 70ies. We all need to extend our horizon to rediscover old stuff. From the relativity dichotomy, early time to space mapping examples are two old simple rules of thumb: a) loops turn into pipelines [1979 and later], b) a decision box turns into a demultiplexer [1967][25]. In the 70ies, when hardware description languages came up, a celebrity said: „A decision box turns into a demultiplexer[25]. This is so simple. Why did it take 30 years to find out?". It's the tunnel view perspective of SE. We need to look through 1 or 2 more tunnels

Now accelerators are programmable: the hardware/software chasm should be turned into configware/ software interfacing

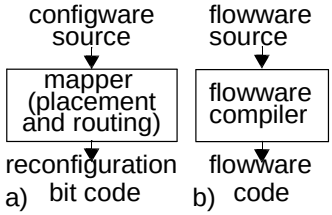

Fig. 3: Separate compilers.

A Dual Dichotomy Approach. Programming education should use a dual dichotomy approach which alleviates dual-rail teaching: 1.) The ***„Paradigm Dichotomy"*** (vN Software machine vs. Flowware machine), and, 2.) the ***„Relativity Dichotomy"*** (time domain vs. space domain). The Relativity Dichotomy methodology mainly based on old ideas, helps to understand both kinds of parallelism: software parallelism (concurrent sequential processes) and Flowware parallelism (by parallel data paths within the pipe networks inside FPGAs or other RPUs or rDPAs (reconfigurable DataPath Arrays)), as well as mappings between both domains and in hybrid systems covering both domains.

From Software Engineering to Program Engineering (PE). Optimizing FPGAs to run specific applications efficiently, changing the configurations takes just milliseconds. FPGAs can talk to a x86 CPU without host intervention, directly over the high performance native processor bus, like Intel's Front Side Bus (FSB). Several vendors offer FPGAs with several CPUs and many very fast memory blocks on board. Programming such hybrid systems requires qualifications for both: SE and RC. Calling for a methodology to alleviate software/configware co-education is subject of this paper. Most tools currently offered do not yet entirely shield the programmer from hardware design issues or explicit parallel programming, nor do they use a standard programming environment. Software/Configware Co-compilers

Program Engineering — the Generalization of Software Engineering: a Dual Dichotomy Approach.

#	scene	problem	interactions	claims	solution
1	VLSI design revolution	VLSI designer population missing	(semiconductor) technology vs. computer science	technology claimed **it is** their job (but couldn't follow Moore's law)	separating: create VLSI design education scene
2	proposed SE revolution	innov. programmer population missing	Software Engineering (SE) vs. Reconfigurable Computing (RC)	SE claims, it is **not** their job	merging: include RC into SE education

Table 7: Similar scenario: SE revolution vs. Mead-&-Conway VLSI design revolution.

automatically sorting out CPU and FPGA code have been demonstrated a long time ago in academia[26]. Recently OpenDF has been published - a flowware-based toolset for RC and Multicore Systems developed by an industry/ university consortium[27]. Players on the markets for FPGA program development tools and expansion modules are the EDA market leaders and companies like Celoxica, DRC Computer, Impulse Accelerated Technologies, Mitrionics, Nallatech, XtremeData and many others. Major progress can be expected from projects integrating a tool flow with multi-granular twin-paradigm prototyping platforms, like from the MORPHEUS consortium including quite a number of major players in IT industry[28]. A tool mainstream standard is not yet accepted. C or C-like languages are candidates. We need a generalization of Software Engineering. Yes, we can. The traditional hardware/software chasm has become intolerable.

A scenario like the VLSI design revolution: the most effective move in the history of modern computer science —

A Highly Promising Scenario

Conclusion. The future predominance of the von Neumann programming paradigm will be intolerable soon. We need the dual dichotomy approach, since the vN paradigm will not disappear completely. Depending on the algorithm properties and its cost of migration, it does not make sense to migrate everything. A sufficiently large programmer population, qualified for the dual dichotomy approach of PE, is far from existing. We need a revolution[4].

Modelled on the Mead-&-Conway VLSI design revolution. Our scenario resembles the VLSI design revolution, the most effective project in the history of modern computer science (table 7: row 1). Originally the semiconductor technology experts claimed to master circuit design with the left hand. However, along the Moore curve this turned into the design crisis with a missing qualified designer population. VLSI design education has been founded as a separate discipline outside technology, supported by a new text book, especially written for people without a technology background[29]. Within about 3 years these courses have been introduced by more than a hundred universities world-wide.

We have a similar promising scenario (row 2 in table 7). A sufficiently large RC-qualified programmer population is missing. In contrast to the VLSI revolution, the SE community claims, that RC is not their job, and the solution will be merging instead of separating (last 2 columns in table 7). We again need such an innovative education effort: professors back to school! We need a world-wide mass movement qualifying most programmers for dual dichotomy programming, ready for a world-wide change-over of many applications, what can create a lot of jobs at least for a decade. This would also be a highly promising issue in climate and energy policy world-wide. New horizons in massively saving energy and high performance computing will be opened up. We urgently need to motivate the opinion leaders in software engineering and in curriculum recommendation task forces.

Literature

1. G. Fettweis et al.: ICT Energy Consumption – Trends and Challenges; Proc. WPMC'08, Lapland, Finland, 8-11 Sep 2008
2. P. Ross: Cloud Computing's Killer App: Gaming; IEEE Spectrum, March 2009
3. R. Katz: Tech Titans Building Boom; IEEE Spectrum, February 2009
4. H. Simon: The Greening of HPC - Will Power Consumption Become the Limiting Factor in HPC?; HPC User Forum, Oct 13-14, 2008, Stuttgart, Germany
5. S.Moore: Multicore Is Bad News For Supercomputers; IEEE Spectrum, Nov. 2008
6. W. Wulf, S. McKee: Hitting the Memory Wall: Implications of the Obvious; Dept of CS, Univ. of Virginia, December 1994
7. F. L. Bauer: conference chair's introductory remarks; first NATO Software Engineering Conference, 1968, Garmisch, Germany.
8. E. Dijkstra (Turing award lecture): The Humble Programmer; Comm. ACM, 15,10 (Oct 1972)
9. C. Walter: Kryder's Law; Scientific American Magazine, August 2005
10. E. A. Lee: The Problem with Threads; IEEE Computer, 39(5), May 2006.
11. J. Udell: Loose coupling and data integrity; Jon's Radio: Jon Udell's Radio Blog, Aug 6, 2002 - http://radio.weblogs.com/0100887/2002/06/19.html
12. E. Dijkstra: The GOTO Considered Harmful; Comm. ACM 11,3, March 1968
13. N. Wirth: A Plea for Lean Software; IEEE Computer; 28,2, Feb. 1995
14. R. Hartenstein (conference opening keynote): Reconfigurable Computing and the von Neumann syndrome; 7th ICA3PP, June 11-14, 2007, Hangzhou, China
15. G. Koch et al.: The Universal Bus Considered Harmful; proc EUROMICRO, June 1975, Nice, France,
16. K. Morris: Kicking a Dead Horse - FPGAs Going the Distance Against ASIC; FPGA and Structured ASIC Journal, April 7, 2009 http://www.fpgajournal.com
17. K. Paulsson, M. Huebner, J. Becker: Exploitation of dynamic and partial hardware reconfiguration for on-line power/performance optimization; FPL 2008
18. J. Becker, M. Huebner (invited talk): Run-time Reconfigurabilility and other Future Trends; SBCCI 2006, Sept 2006, Ouro Preto, Brazil
19. R. Hartenstein: Why we need Reconfigurable Computing Education; Int'l Workshop on Reconfigurable Computing Education, March 1, 2006, Karlsruhe, Germany
20. R. Jacobi, M. Ayala, C. Llanos et al.: Reconfigurable systems for sequence alignment and for general dynamic programming; j. GMR Genetics and Molecular Research 4,3 2005
21. T. El-Ghazawi et al.: The promise of high-performance reconfigurable computing; IEEE Computer, vol.41, no.2, pp.69-76, Feb. 2008
22. C. Bobda: Introduction to Reconfigurable Computing: Architectures, Algorithms and Applications; Springer Verlag, 2007
23. D. Gajski et al.: A second opinion on data-flow machines and languages; IEEE Computer, Feb. 1982.
24. C. Chang et al: The Biggascale Emulation Engine (Bee); summer retreat 2001, UC Berkeley
25. C. Bell et al: The Description and Use of Register-Transfer Modules (RTM's); IEEE Trans-C21/5, May 1972
26. J. Becker et al.; CoDe-X: A Novel Two-Level Hardware/Software Co-Design Framework; 9th Int'l Conf. on VLSI Design, Bangalore, India, Jan. 1996
27. S. Bhattacharyya et al.: OpenDF, Dataflow Toolset f. Reconfigurable Hardware & Multicore Systems; Multicore Computing (MCC-08), Ronneby, Sweden, 27-28 Nov, 2008
28. N. Voros, A. Rosti, M. Huebner (Eds.): Dynamic System Reconfiguration in Heterogeneous Platforms - The MORPHEUS Approach; Springer Verlag, June 2009
29. C. Mead, L. Conway: Introduction to VLSI Systems; Addison-Wesley, 1980