Configware and Morphware going Mainstream

invited paper

Jürgen Becker
Universitaet Karlsruhe (TH), Germany
http://www.itiv.uni-karlsruhe.de/
becker@itiv.uni-karlsruhe.de

Reiner Hartenstein,
Kaiserslautern University of Technology, Germany
http://hartenstein.de
reiner@hartenstein.de

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.*

1. 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 extreme 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 is *soft "hardware"*, 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.

Booming conferences on morphware and its applications like FPL, the eldest and largest, as well as the adoption of this topic area by congresses like DAC, DATE, ASP-DAC, ISCAS, SPIE, and others 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 80ies.

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 von-

a) platform category	source "running" on platform	machine paradigm
hardware	(hardwired)	
morphware	configware	
ISP**	software	von Neumann
AMP*	flowware	anti machine
rAMP	flowware & configware	

*) anti machine processor (data stream processor).
**) instruction stream processor

b) category of morphware use	granularity (path width)	(re)configurable blocks used
Reconfigurable Logic (fig. 2 b)	fine grain (~1 bit)	CLBs: configurable logic blocks
Reconfigurable Computing	coarse grain (example: 16 or 32 bits)	rDPUs: reconfigurable data path units (e.g. ALU-like)
	multi-granular: supporting slice bundling with carry connect	rDPU slices (example: 4 bits)

Fig. 1: a) Platform categories, b) categories of morphware.

Neumann-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, where a procedural / structural double rail approach is a pre-requisite, providing new chances.

Currently most programmers do not yet really have the background required. This challenge can be met by the new dichotomy of CS. This education gap can be bridged only by a curricular transition from the von-Neumann-only mind set toward this dichotomy of basic computing paradigms. 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. 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.

2. Morphware

Advancing maturity of the area is indicated by a growing consensus on terminology (fig. 1) and acronyms (fig. 2), to clearly distinguish between *Reconfigurable Logic (RL)* and *Reconfigurable Computing* (*RC*, fig. 1), as well as between platform categories (fig. 1). With "*data stream processor*" terminology has a dilemma: "dataflow machine" and DSP (digital signal processing) have been occupied decades ago by other areas. So we prefer the acronym *AMP (anti machine processor)*.

RTR	run time reconfiguration
EM	evolutionary methods
EH	evolvable morphware ("evolvable hardware")
AM	anti machine (DS machine)
AMP	data stream processor*
rAMP	reconfigurable AMP
ISP	instruction stream processor
CPU	"central" processing unit: DPU (**with** instruction sequencer)
DS	data stream

*) no "dataflow machine" [2]

Fig. 2: Acronyms.

asMB	autosequencing Memory Bank
DPU	data path unit (**without** sequencer)
rDPU	reconfigurable DPU
ecDPU	emulation-capable DPU
DPA	data path array (DPU array)
rDPA	reconfigurable DPA
RA	reconfigurable array
RC	reconfigurable computing
RL	reconfigurable logic
FPGA	field-programmable gate array
ASIC	application-specific integrated ckt.
SoC	(an entire) System on a Chip
cSoC	configurable SoC

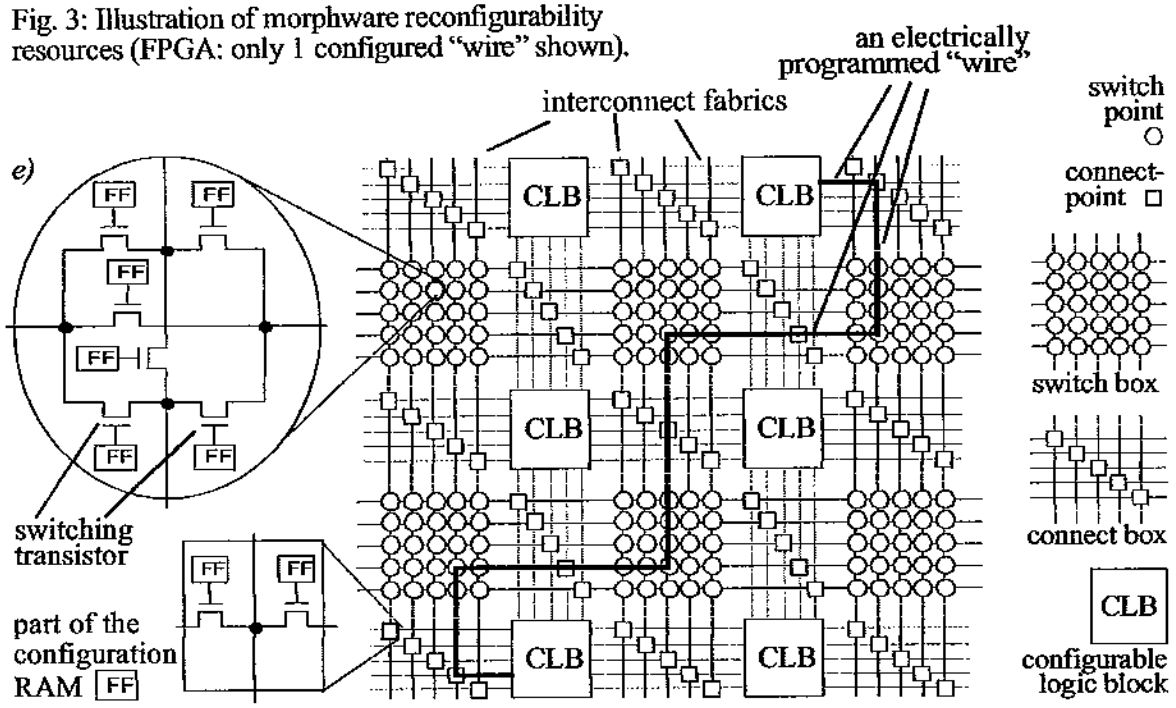
Terminology. Even more important is the terminology from a more global point of view, not to get confused between "programmable," "microprogrammable", "field-programmable" and contradictory "soft hardware", and, to clarify the dichotomy of fundamental models, as well as, to alleviate the merger of RC and classical CS (figure 1 a). Whereas classical CS deals with *software* running on *hardware*, the new branch of CS deals with *flowware* running on *hardware*, was well as with *configware* [2] [3] and *flowware* [4] "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. programming in time. But implementing an algorithm on *morphware* means *structural programming by configware* for structural reconfiguration of the morphware device (mapper output, see fig. 11) and flow ware for scheduling the data streams (scheduler output, see fig. 11). The acronym *FPGA (field-programmable gate array)* 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).

3. Reconfigurable Logic

Figure 3 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.

Fig. 3: Illustration of morphware reconfigurability resources (FPGA: only 1 configured "wire" shown).



CLBs and interconnect fabrics. A minor part of the area is used by *CLBs (configurable logic blocks)*, 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. 3). The state of each switching transistor is controlled by a Flip-flop (FF) which is part of "hidden" *configuration RAM* (not shown in fig. 3), also used to 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. 1 b): *fine grain* morphware (FPGAs) and *coarse grain* morphware. First FPGAs appeared as cheap replacements of MPGAs (Mask Programmable Gate Arrays). Still to-day FPGAs are the reason for shrinking ASIC* (fig. 2) 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 lead to the adoption by the *rapid prototyping*

*) ASIC fabrication cost is much lower (only a few specific masks needed) than that of other integrated circuits (fig. 7)

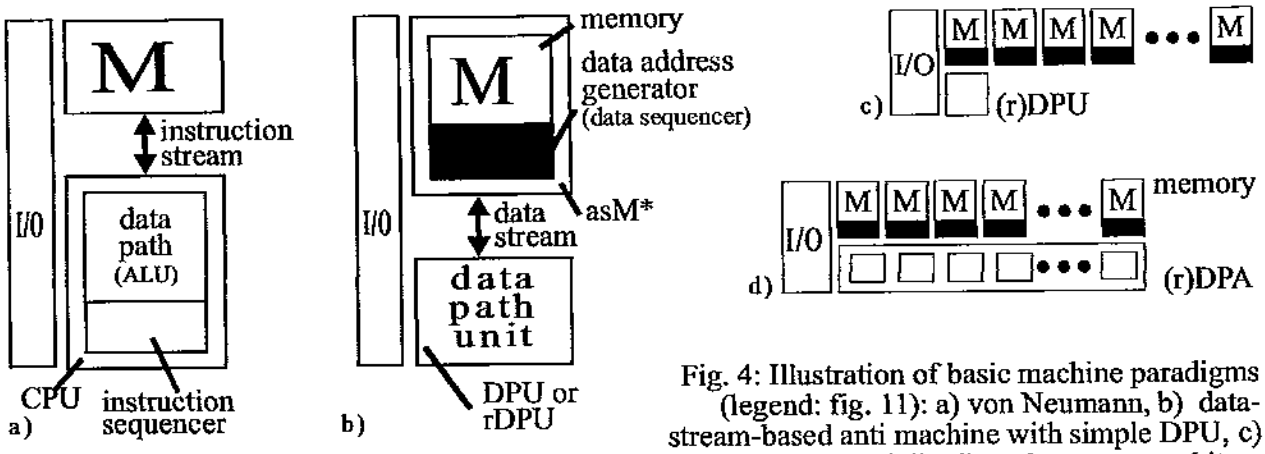


Fig. 4: Illustration of basic machine paradigms (legend: fig. 11): a) von Neumann, b) data-stream-based anti machine with simple DPU, c) with rDPU and distributed memory architecture, d) w. DPU array (DPA or rDPA).

*) auto-sequencing memory

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 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 SoC design (System-on-Chip). 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. 6).

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 3 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 designs less complex emulation boards are used, like Logic emulation PWB (based on Xilinx Virtex, can emulate up to 3 million gates), and, the DN3000k10 ASIC Emulator from the Dini Group.

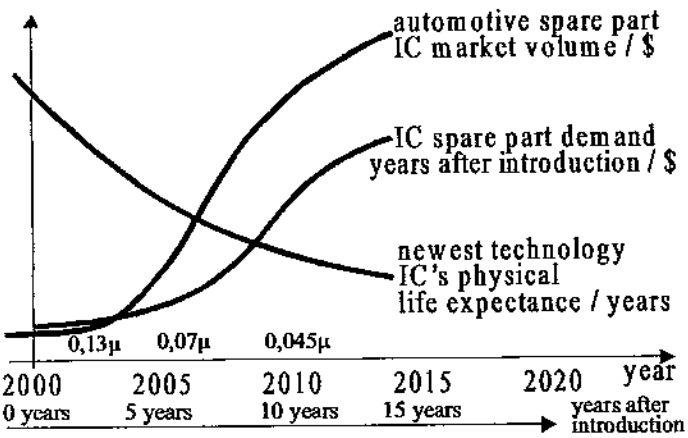


Fig. 5: The emerging automotive IC spare part problem.

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. 5) 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. 5). 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 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. 7).

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 *systems on a chip (SoC)* can be implemented on morphware *(cSoC: configurable SoC)*. Due to Moore's law 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 4 PowerPCs on board, and 24 Conexant 3.125 Gb/s serial link cores providing a total of 75 Gb/s/chip link bandwidth.

Future Giga FPGAs. A future challenge is the giga FPGA with hundreds of millions of gates, possible by future progress of technology. Will this lead to a real general purpose fine grain morphware platform, so that coarse grain morphware is no more needed: by just mapping it onto a giga FPGA? The submicron leakage current problem of future technologies creating too high power dissipation may be solved within a few years: with fully-depleted silicon-on-insulator (SOI) planar transistors, or with new devices like the tri-gate transistor developed by IBM - improving the leakage current by a factor of about 10 to 100. 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 Rabaey et al. obtained an improvement by factors between 5 and 22. have been obtained by Jan Rabaey et al. A reduction of clock frequency by a factor of n yields a reduction of power dissipation by a factor of n^3, when also the optimum technology is selected - if its fab line is still available.

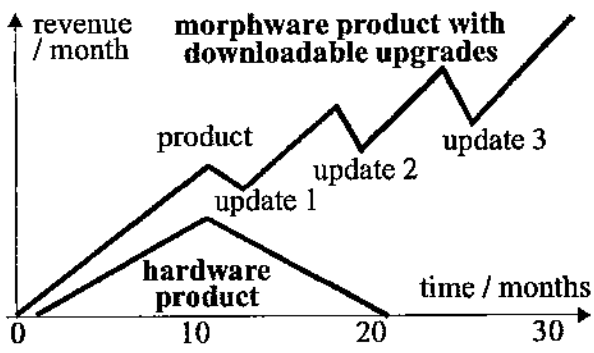


Fig. 6: Morphware product longevity by updates.

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 morphware technology. The labelling „evolutionary" and the „DNA" metaphor helped to start new conference series, and to raise research funds in the EU, Japan, Korea, and the US. Critics disfavor the trend to love genetic algorithms, even where simulated annealing is much more efficient. The scene is still in its visionary phase, where shake-out phenomena are not yet noticeable.

4. Reconfigurable computing

Fine grain morphware lacks area-efficiency. The physical integration density (transistors per chip) of FPGAs is roughly 2 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 4 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 [5]. Coarse grain morphware is also about one order of magnitude more energy-efficient than fine grain (fig. 9 [6] [7]). Whereas RL based on fine grain morphware (FPGAs) uses single bit wide CLBs (fig. 1 b), Reconfigurable Computing (RC) uses *rDPUs (reconfigurable data path units),* which, similar to ALUs, have major path widths, like 32 bits, for instance - or even *rDPAs (rDPU arrays).* Important applications stem from the performance limits of the "general purpose" processor, creating a demand for accelerators.

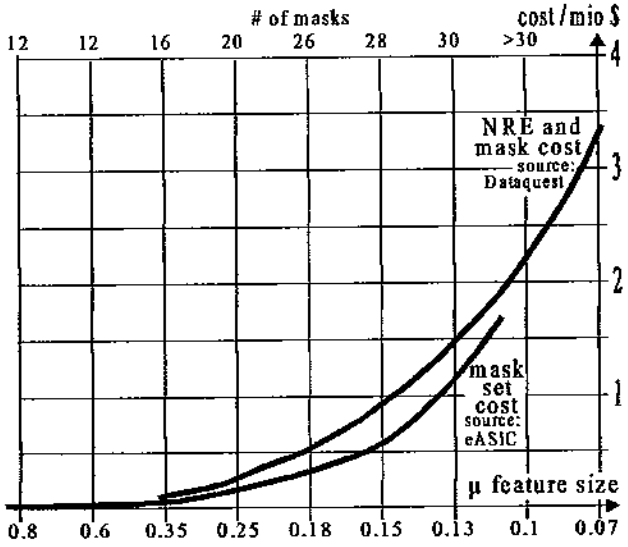


Fig. 7: Increasing mask set cost and total NRE cost.

Coarse-grain Reconfigurable Architectures. Several interconnect topology categories of coarse grain morphware for reconfigurable computing (survey: [5]) 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 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 throughput requirements are growing faster than Moore's law, along with growing flexibility requirements due to unstable standards and multi-standard operation [8]. Currently the requirements can be met only by rDPAs from a provider like PACT (fig. 8 [9]). 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-

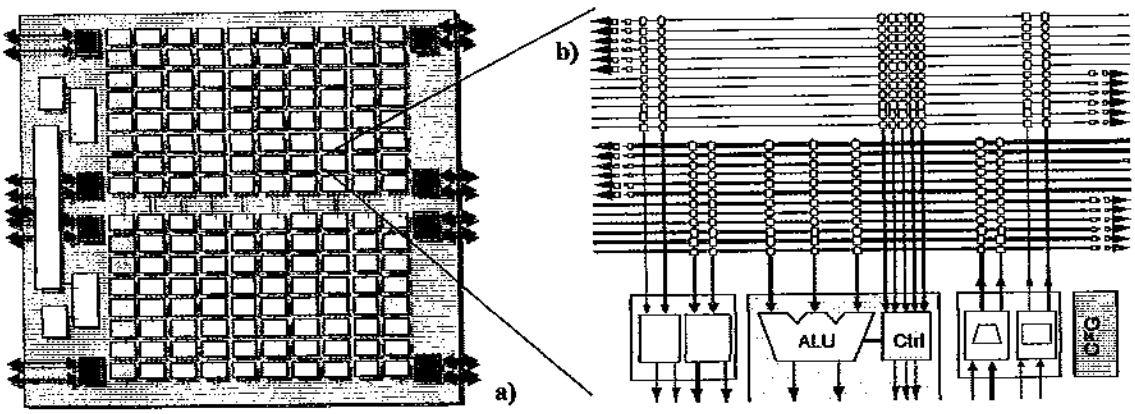


Fig. 8: Configurable XPU (xtreme processing unit) from PACT: a) array structure, b) rDPU.

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.

Domain-specific approach. Another solution is a domain-specific approach [5], where rDPU architecture and other features are optimized for a particular application domain. A design space explorer may help to derive an optimum DPU array architecture from a benchmark or domain-typical set of applications within a few days [10]. A desirable new solution for the far future could be the soft array approach mapping DPAs onto a very large FPGAs, providing highest flexibility. However, this would be feasible only with new EDA flows for DPAs or rDPAs.

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.

5. Data-stream-based Computing

The world of traditional instruction-stream-based informatics is based on computing in the time domain, where a program schedules the instructions for execution. The classical model locates instruction sequencer and datapath in the same CPU (fig. 4 a). Due to morphware a second basic model has emerged, so that we now have a dichotomy of models: instruction-stream-based computing vs. data-stream-based computing. There is a lot of similarities, so that each of the 2 models is a kind of mirror image of the other model - like with matter and antimatter.

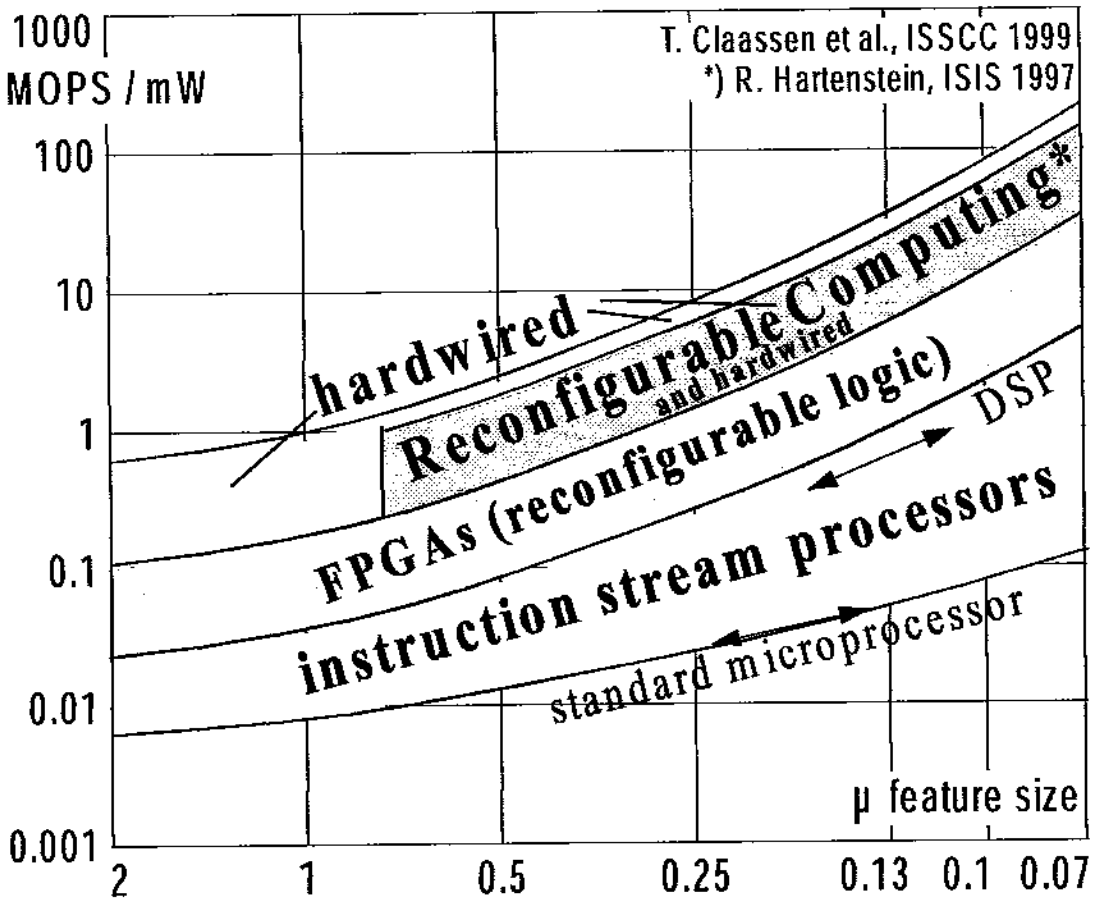


Fig. 9: Energy efficiency vs. flexibility including Reconfigurable computing.

Similarities and Asymmetries. Data-stream-based computing, the counterpart of instruction-stream-based von Neumann computing, however, uses an instruction counter instead of a program counter (example in fig. 4 b). However, there are some asymmetries, like predicted by Paul Dirac for antimatter. Figure 11 a 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, where as a von Neumann machine cannot have more that one program counter. The partitioning scheme of the data-stream machine model assigns a sequencer (address generator) always to a memory bank (fig. 4 b), never to a DPU. This modelling scheme goes fully conform with the area of embedded distributed memory design and management (see section5.2).

5.1 Flowware

Data streams, also efficiently supporting loop-level parallelism [11], have been popularized throughout the 80ies by systolic arrays, and later by the super systolic array (pioneered by the

	(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 spin	instruction 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

Fig. 10: Asymmetry between machine and anti machine paradigms. *) **before** run time

KressArray [12], the generalization of the systolic array [13]), as well as by projects like SCCC (Streams-C Configurable Computing [14]), SCORE (Stream Computations Organized for Reconfigurable Execution [15]), ASPRC (Adapting Software Pipelining for Reconfigurable Computing), BEE (Biggascale Emulation Engine [16]), the KressArray Xplorer [10] 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 [17] and for software join the same language principles and have a lot in common, no matter, wether 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 to generate the data streams.

Two programming sources. Von Neumann machine need 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: *configware* to program (reconfigure) the operational resources, and, *flowware* to schedule the data streams. Figure11 b illustrates the structure of a compiler [2] [12] 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.

5.2 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. Together with application-specific embedded memory architecture synthesis also flowware implementation deserves performance and power dissipation optimization [18]. 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 [5].

Address generators. To solve the memory communication bandwidth problem the anti machine paradigm (data-stream-based computing) is much more efficient than "von Neumann". Two alternative methodologies are available [18]: specialized architectures using synthesized address generators (e. g. APT by IMEC [18]), or, flexible architectures with programmable general purpose address generators [19], which do not need memory cycles even during complex address computations [18].

6. Data-Stream-based vs. concurrent Computing

Software processor solutions are inefficient relative to hardwired solutions (fig. 9). 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 [16]. 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. Amdahls law explains just one of several reasons of inefficient resource utilization. Each CPU comes with a central von Neumann 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's no CPU. There's nothing "central". Data-stream-based computing with (r)DPAs provides a drastically more efficient way to cope with memory communication bandwidth problems than classical concurrent computing.

6.1 The key to massive parallelization

However, to-day 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 [12]. This *("super systolic")* generalization of the systolic array also supports inhomogenous irregular arrays [10] - in contrast to classical systolic arrays.

Chip in a day. On hardwired DPU array basis the BWRC [16] 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 data-flow 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.

7. Configware Compilers

When reconfigurable accelerators are involved instead of the hardwired accelerators, Hardware / Software Co-Design turns into Configware / Software Co-Design. Using compilation techniques for both sides we turn co-design into co-compilation.

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. Partly dating back to the 70ies and 80ies know-how is available from the classical parallelizing compiler scene, like software pipelining, and, loop transformations.

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 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. Compilation from HLL sources [12] [14] still stem from academic efforts, as well as the first automatic *co-compilation* from HLL sources including automatic software/configware

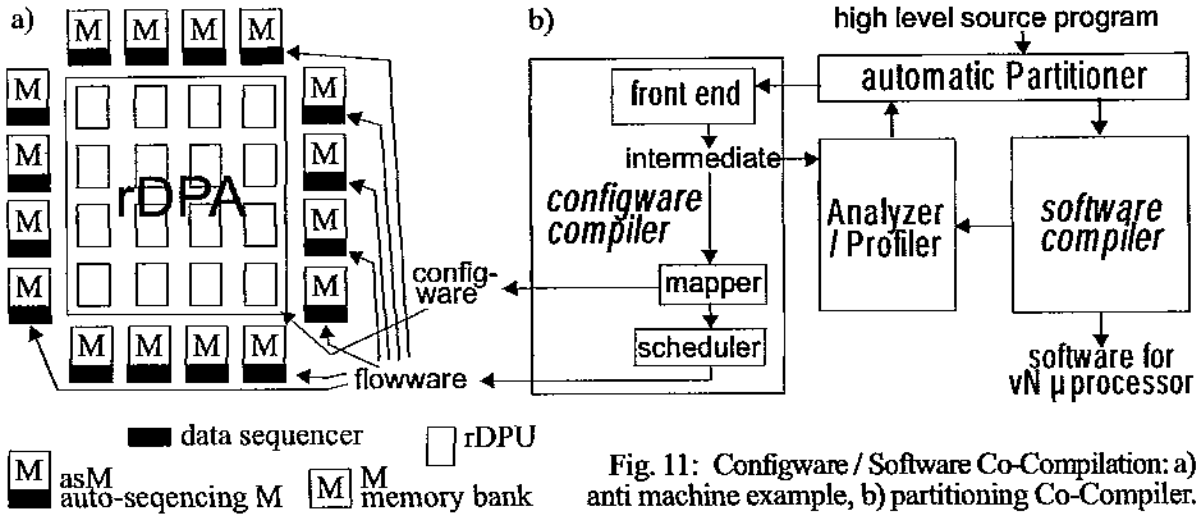


Fig. 11: Configware / Software Co-Compilation: a) anti machine example, b) partitioning Co-Compiler.

partitioning (fig. 11 a) by identifying parallelizable loops, having been implemented for the data-stream-based MoM (Map-oriented Machine) [19].

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

7.1 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 von Neumann paradigm helped a lot to educate zillions of programmers. Figure 4 a 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 (datapath unit) 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.) Figure 12 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* [18]).

machine category		von Neumann	anti machine
general property		procedural sequencing: deterministic	
driven by:		single instruction stream	data **stream**(s) (**no** "*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	array of ALUs and others
	operation	sequential	parallel
morphware support		no	yes

Fig. 12: Machine paradigms: comparing the properties of machine paradigms.

The *Anti Machine* Paradigm for morphware [20] and even for hardwired anti machines the data-stream-based anti machine paradigm is the better counterpart (fig. 4 b) of the von Neumann paradigm (fig. 4 a). Instead of a CPU the anti machine has only a DPU (datapath unit) without any sequencer, or a rDPU (reconfigurable DPU) without a sequencer. The anti machine model locates data sequencers on the memory side (fig. 4 b). Anti machines do not have an instruction sequencer. Unlike "von Neumann" the anti machine has no von Neumann bottleneck, since it also allows multiple sequencers (fig. 4 c) to support multiple data streams interfaced to multiple memory banks, which is typical to data-stream-based computing (fig. 4 c) allowing operational resources much more powerful than ALU or simple DPU: major DPAs or rDPAs (fig. 4 d).

General purpose anti machine. The anti machine is as universal as the von Neumann machine. The anti programming language is as powerful as von-Neumann-based languages. But instead of a "control flow" sublanguage a "data stream" sublanguage like *MoPL* [17] recursively defines *data goto, data jumps, data loops, nested data loops,* and *parallel data loops*. For the anti machine 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 [20].

The anti machine model, where the DPUs are transport-triggered by arriving data, goes conform with the new and rapidly expanding R&D area of embedded memories [18], including application-specific or programmable data sequencers. Figure 12 compares the properties of both paradigms.

8. Configware industry

RAM-based success story. The secret of success of software industry is based on RAM, von Neumann 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 ten years ago. 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. 6). 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 useless, 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.

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.

Not really scalable. 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. A future solution is feasible by a new EDA approach, rather than by new FPGA architectures.

9. 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 orVerilog originally targeted at a hardwired implementation.

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 hundred soft processor cores can be mapped, leaving plenty of area to other on-chip resources, so that a coarse grain morphware like from PACT [21] [22] [23] can be mapped onto it [24], 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. A Configurable System-on-Chip (CSoC) consisting of a Leon core, an XPP-array of suitable size, global and local memory cores and efficient multi-layer Amba-based communication interfaces is synthesized onto 0.13 µm UMC CMOS (8 layer) copper standard cell process at Universitaet Karlsruhe [22]. The actual applications currently mapped and analyzed onto this CSoC are different MPEG algorithms, W-LAN (Hiperlan-2), Software Radio including RAKE-Receiver and Baseband Filtering, whereas the corresponding non-confidential performance/cost/power trade-offs can be given in a final paper version.

10. 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.

11. Literature

[1] http://morphware.net/

[2] J. Becker et al.: Parallelization in Co-Compilation for Configurable Accelerators; Proc. ASP-DAC'98

[3] http://configware.org/

[4] http://flowware.net/

[5] R. Hartenstein: A Decade of Research on Reconfigurable Architectures; DATE 2001

[6] R. Hartenstein (invited): The Microprocessor is no more General Purpose; Proc. ISIS 1997

[7] R. Hartenstein (invited): Trends in Reconfigurable Logic and Reconfigurable Computing; ICECS 2002

[8] J. Becker, T. Pionteck, M. Glesner: An Application-tailored Dynamically Reconfigurable Hardware Architecture for Digital Baseband Processing; SBCCI 2000

[9] http://pactcorp.com

[10] U. Nageldinger et al.: Generation of Design Suggestions for Coarse-Grain Reconfigurable Architectures; FPL 2000

[11] B. Mei et al.: Exploiting Loop-Level parallelism on Coarse-Grained Reconfigurable Architectures Using Modulo Scheduling; DATE 2003

[12] R. Kress et al.: A Datapath Synthesis System for the Reconfigurable Datapath Architecture; ASP-DAC'95

[13] http://kressarray.de/

[14] J. Frigo, et al.: Evaluation of the streams-C C-to-FPGA compiler: an applications perspective; FPGA 2001

[15] E. Caspi, et al.: Extended version of: Stream Computations Organized for Reconfigurable Execution (SCORE): FPL '2000

[16] C. Chang, K. Kuusilinna, R. Broderson, G. Wright; The Biggascale Emulation Engine; FPGA 2002

[17] A. Ast, et al.: Data-procedural Languages for FPL-based Machines; FPL'94

[18] M. Herz, Hartenstein, M. Miranda, E. Brockmeyer, F. Catthoor (invited paper): Memory Organization for Data-Stream-based Reconfigurable Computing; ICECS 2002,

[19] M. Weber et al.: MOM - Map Oriented Machine; in: E. Chiricozzi, A. D'Amico (editors): Parallel Processing and Applications, North-Holland, 1988

[20] K. Schmidt et. al.: A Novel ASIC Design Approach Based on a New Machine Paradigm; J. SSC 1991

[21] V. Baumgarten, et al.: PACT XPP - A Self-Reconfigurable Data Processing Architecture; ERSA 2001

[22] J. Becker, A. Thomas, M. Vorbach, G. Ehlers: Dynamically Reconfigurable Systems-on-Chip: A Core-based Industrial/Academic SoC Synthesis Project; IEEE Workshop Heterogeneous Reconfigurable SoC; April 2002, Hamburg, Germany

[23] J. Becker, M. Vorbach: An Industrial/Academic Configurable System-on-Chip Project (CSoC): Coarse.grain XPP/Leon-based Architecture Integration; DATE 2003

[24] J. Cardoso, M. Weinhardt: From C Programs to the Configure-Execute Model; DATE 2003