19. Morphware and Configware

Reiner Hartenstein
TU Kaiserslautern
http://hartenstein.de

Abstract. *This chapter introduces morphware as the basis of a second machine paradigm, which mainly has been introduced by the discipline of embedded system design targeting the System on Chip (SoC). But more recently SoC design is adopting more and more computer science mentality and also needs the services of CS professionals. Computing Sciences are going to include the morphware paradigm in its intellectual infrastructure. The time has come to bridge the traditional hardware software chasm. A dichotomy of two machine paradigms is the road map to upgrade CS curricula by evolution, rather than by revolution. This chapter mainly introduces morphware platforms as well as their models and architectures, whereas chapter 5 of this book deals with their applications and algorithms in reconfigurable computing*

19.1. Introduction

Morphware [1] [2] is the new computing paradigm, the alternative RAM-based general purpose computing platform model. The traditional hardware / software chasm distinguishes software running on programmable computing engines (microprocessors) *driven by instruction streams* scanned from RAM, as well as application-specific fixed hardware like accelerators which are not programmable after fabrication. The operations of such accelerators are primarily *driven by data streams*. Such accelerators are needed because of the microprocessor's performance limits caused by the sequential nature of its operation - by the *von Neumann bottleneck*.

John von Neumann's key achievement has been the simple common model called *von Neumann machine paradigm* ([3] [4], von Neumann has not invented the computer.) His model provides excellent guidance in CS education, also by narrowing the almost infinite design space. However, the contemporary common model of computing systems is the cooperation of (micro)processor and its accelerator(s) including an interface between both (figure 1 a). This model holds not only for embedded systems, but also for the

PC needing accelerators not only for running its own display. The accelerators are a kind of slaves. Operating system and other software is running on the microprocessor, which is the host and master of the accelerators. The host may send parameters (like for mode select, start, stop, reset etc.) and receive interrupts and some result data.

The host operation is *instruction-stream-driven*. The instruction stream is managed by the program counter inside the host processor. The accelerator usually has no program counter. Its operations are *data-stream-driven* (see *data stream* interface in fig. 1 a). Not only in terms of efficiency, this model especially makes sense for data-intensive applications, where *multiple data streams* are interfaced to the accelerator (fig. 1). Only a few very sophisticated architectures are difficult to map onto this model. In case of computation-intensive applications with very low data traffic to/from the accelerator a single data stream generated by the host may be sufficient. This model (for details see next section and section 19.3.1 ff.) is as simple as the host's von Neumann (vN) model, what also is important for educational purposes. For details also see section 19.3.1.

By the way, *data-stream-driven computing* (*flowware-based computing*: this term will be defined later) has been used implicitly already by the first programmers. In a von-Neumann-based instruction-stream-driven environment, the less efficient detour over the application control-structures has been the only viable solution. However, by avoiding the von Neumann bottleneck, a data-stream-driven environment permits a much more direct efficient solutions. For more detailed explanations see section 19.3.

vN processor programming is supported by compilers, whereas traditional accelerator development has been and is done with *EDA* tools (electronic design automation [5] - for acronyms see fig. 2).

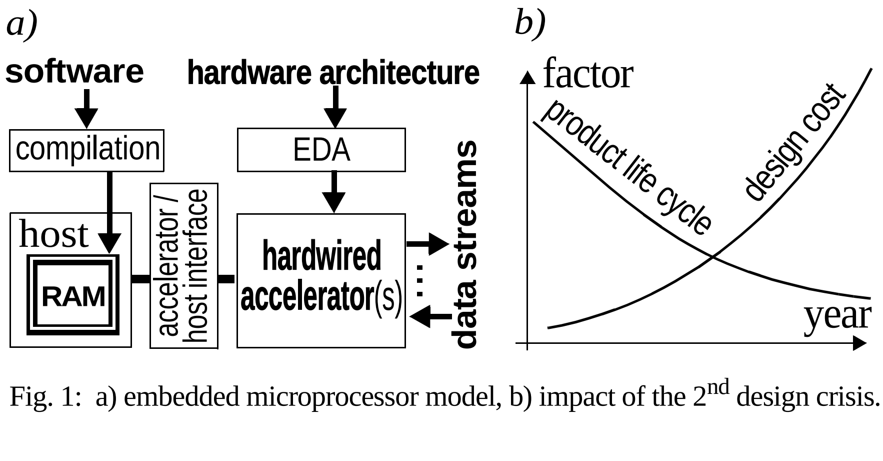


Fig. 1: a) embedded microprocessor model, b) impact of the 2nd design crisis.

More recently, however, such accelerator design is affected by the *2nd design crisis* (figure 1 b). Compared to microprocessor design the SoC (System on Chip) design productivity in terms of gates per day is slower by a factor of about 10^{-4} [6]. Another symptom of increasing design implementation problems and the *silicon technology crisis* is the drastically decreasing number of wafer starts for newer technology fabrication (fig. 3 a) and the still decreasing low number of *ASIC* design starts (fig. 3 c). Another major cost factor of the application-specific silicon needed for accelerators is increasing mask cost (fig. 3 b), driven by growing wafer size and the growing number of masks needed. ASIC (application-specific IC) stands for mask-configurable gate arrays and similar methodologies [7]

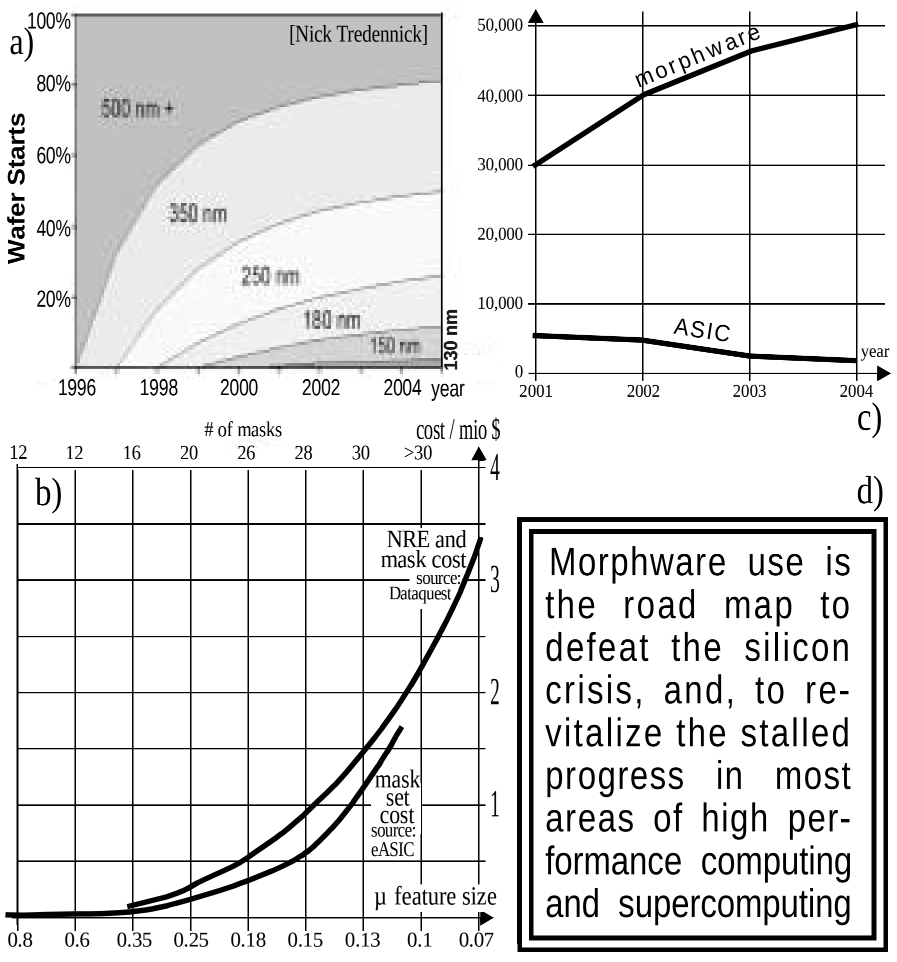
AM	anti machine (DS machine)	ISP	instruction stream processor
AMP	data stream (AM) Processor	LSI	Large Scale ICs
ASIC	application-specific IC	LUT	Look-Up Table
asMB	autosequencing Memory Bank	MCGA	Mask-Configurable Gate Array
BIST	Built-In Self-Test	MPGA	(see MCGA)
CFB	Configurable Function Block	MSI	Medium Scale ICs
CLB	Configurable Logic Block	MW	Morphware
COTS	commodity off the shelf	PC	Personal Computer
CPU	“central” processing unit: DPU	PS	Personal Supercomputer
	(with instruction sequencer)	pSoC	programmable SoC
cSoC	configurable SoC	rDPU	reconfigurable DPU
CW	Configware	rDPA	reconfigurable DPA
DAC	Design Automation Conference	RA	reconfigurable array
DPA	data path array (DPU array)	RAM	random access memory
DS	data stream	rAMP	reconfigurable AMP
DPU	data path unit (without sequencer)	RC	reconfigurable computing
ecDPU	emulation-capable DPU	rGA	reconfigurable gate array
EM	evolutionary methods	RL	reconfigurable logic
EDA	electronic design automation	RTR	run time reconfiguration
EH	evolvable morphware	SoC	(an entire) System on a Chip
	(“evolvable hardware”)	SSI	Small Scale ICs
FPGA	field-programmable gate array	SW	Software
FRGA	field-reconfigurable gate array	SystemC	C dialect f.Hw/Sw co-design
FW	Flowware	UML	Unified Modelling Language
GNU	GNU’s Not Unix (consortium)	Verilog	a popular C-like HDL
HDL	Hardware Description Language	VHDL	VHSIC Design Language (
HPC	High Performance Computing		a HDL)
HW	Hardware	VHSIC	Very High Speed ICs
IC	integrated circuit	VLSI	Very Large Scale ICs
IP	intellectual property	vN	von Neumann

Fig. 2: Acronyms.

needing much less masks than *full custom ICs* (integrated circuits) requiring the full mask set of the fabrication process [8].

19.1.1. Morphware

Illustrated by Makimoto's wave model [9] [10] the advent of morphware is the most important revolution to silicon application after the introduction of the microprocessor [11]. Emerging in the 80ies and now having moved from a niche



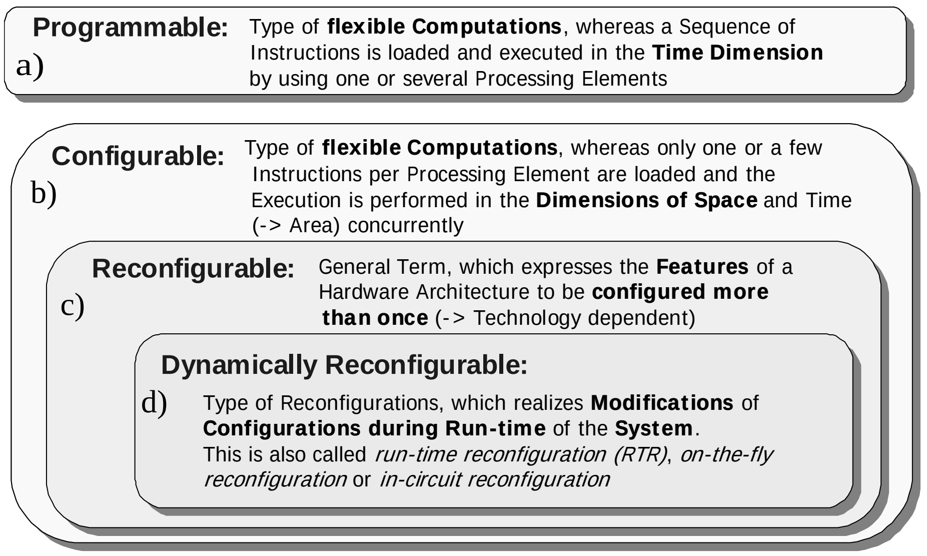
Morphware use is the road map to defeat the silicon crisis, and, to re-vitalize the stalled progress in most areas of high per-formance computing and supercomputing

Fig. 3: The second design crisis (silicon crisis): b) increasing mask set cost and total NRE cost, a) decreasing number of wafer starts, c) growing number of morphware-based design starts [13] vs. declining number of ASIC design starts [13]: demonstrating, that morphware already has reached mainstream status, d) providing the road map on the way out of the silicon crisis.

market to mainstream we have a third class of platforms filling the gap between vN-type procedural compute engines and application-specific hardware. It is *morphware*, the fastest growing segment of the semiconductor market. (for terminology also see figure 5.). The most important benefit of morphware is the opportunity to replace hardwired accelerators by RAM-based reconfigurable accelerators, so that application-specific silicon can be mostly avoided, like well known from running software on the vN-type microprocessor. This will be explained by the following paragraphs. The very high and still increasing number of morphware-based design starts (fig. 3 c) demonstrates the benefit of using replacing morphware platforms instead of ASICs, where the backlog of design starts over morphware has exceeded a factor of more than 10 and is growing further.

von Neumann's key achievement is the simple common model called ***von Neumann machine paradigm***.

Morphware is structurally programmable hardware, where the interconnect between logic blocks and/or functional blocks, as well as the active functions of such blocks can be altered individually by *downloading configware*, down to the *configuration memory (configuration RAM)* of a morphware chip (also compare figure 6 e). So we need two kinds of input sources: traditional *software* for programming instruction streams, and, *configware* for structural reconfiguration of morphware.



Programmable: a) Type of **flexible Computations**, whereas a Sequence of Instructions is loaded and executed in the **Time Dimension** by using one or several Processing Elements

Configurable: b) Type of **flexible Computations**, whereas only one or a few Instructions per Processing Element are loaded and the Execution is performed in the **Dimensions of Space** and Time (-> Area) concurrently

Reconfigurable: c) General Term, which expresses the **Features** of a Hardware Architecture to be **configured more than once** (-> Technology dependent)

Dynamically Reconfigurable: d) Type of Reconfigurations, which realizes **Modifications** of **Configurations during Run-time** of the **System**. This is also called *run-time reconfiguration (RTR)*, *on-the-fly reconfiguration* or *in-circuit reconfiguration*

Fig. 4: Contribution to Terminology [12]: programmable vs. (re)configurable.

Before going into more detail we should do a first step in clarifying the *terminology* around reconfigurability [12]. To highlight the key issues in distinguishing the classical vN paradigm from morphware we should define the term *reconfigurable*, because *reconfiguration* in general has many different meanings. In computing sciences the terms *programmable* refer to the *time domain*, where and *programming* means *instruction scheduling* (fig. 4 a). The term *configurable* introduces the *space domain*, where *configuration* means *the set-up of structures* and pre-adjustment of logic blocks or function blocks (fig. 4 b). *Re-configuration* means, that a platform can be configured again several times for different structures (fig. 4 c), whereas MCGAs (see section 19.2) can be configured only once. Configuration or reconfiguration usually is *impossible during run time*. But *Dynamically reconfigurable* (fig. 4 d) means, that partial reconfiguration may happen at run time. A warning to educators: dynamically reconfigurable or *self-reconfigurable* systems are more bug-prone than others and difficult to explain and difficult to understand.

By introducing morphware we obtain a new general model of embedded computers (figure 7 a): *the accelerator has become reconfigurable.* It has been changed from hardware (fig. 1 a) to morphware. It has been mentioned above, that accelerator operation usually is data-stream-based. Because of its *non-von-Neumann machine principles* an accelerator has *no von Neumann bottleneck* and may be interfaced to a larger number of data streams (fig. 23 c, d). With a morphware accelerator (figure 7 a), the host may also use the host/ accelerator interface to organize the reconfiguration process (this will be explained later). Also mixed-type accelerators are possible (figure 7 b): hardware and morphware. However, a few architectures include morphware directly inside the vN microprocessor. Here the morphware is used for *flexible instruction set extensions* [14] [15], a modern version of the vN-only model, where Morphware is connected to the *processor bus* (fig. 8 a). Also most *network processors* use instruction set extensions [16]. This is different from the common model shown in figure 7, where morphware is just connected to the hosts *memory bus* (fig. 8 b).

19.1.2. Two RAM-based machine paradigms

We now have two different RAM-based input source paradigms: one for scheduling (programming) the instruction streams, to be scanned *from RAM program memory <u>during</u> run time* by sequences of instruction fetches, and, the other one for configuring structures by downloading configware code *to the configuration RAM <u>before</u> run time*. Downloading configware code is a kind of pseudo instruction fetch (but here not *at run time*) where, however, such “instructions” or expressions may be much more powerful than microprocessor instructions. The configuration RAM is often called *hidden RAM*, because it is not nicely concentrated into a matrix, like in typical RAM components sold by IC vendors. Physically the individual memory cells in a morphware device are located close to the switch point or connect point they are controlling (see the flipflops *FF* in fig. 10 c and d). Also the addressing method used by morphware for downloading reconfiguration code often is different from that of classical RAM.

Data-stream-driven computing has been used implicitly already by the first programmers.

It has been recognized rather early, that morphware has introduced a fundamentally new machine paradigm. FCCM (Field-reconfigurable Custom Computing Machines [17]), the name of an annual conference series, is an indication. Mostly from academia a major number of experimental computing machines of this kind has been implemented (for a survey covering the years 1995 and earlier see [18]).

As has been mentioned above, using commodity off-the-shelf (COTS) morphware for acceleration can avoid the very costly need for application-specific silicon. Both kinds of platforms support *rapid downloading of patches, upgrades, or even new applications* downto the RAM program memory, even via the internet. The consequence is a change of the business model for accelerators. *Personalization* before *fabrication* typical to hardwired

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

Fig. 5: Terminology.

*) see section 19.3.6 and figure 23.

accelerators can be replaced by the business model of the microprocessor, using *personalization after fabrication* - at the customers site.

It is very important to distinguish, that the personalization source for vN microprocessors is *software*, and, for morphware it is *configware*. Because of the growing importance of configware we currently observe a growing *configware industry* - a kind of emerging competitor to *software industry*. Morphware has become an essential and indispensable ingredient in SoC (System on a Chip) design and beyond. Morphware meanwhile is used practically everywhere, so that this paper has no room for a survey to mention all of them. A good reading source are the proceedings volumes (published by Springer LNCS series [19]) of FPL [20], the annual international conference on Field-Programmable Logic and its applications, is the largest conference in this area.

19.2. Fine grain morphware

Since their introduction in 1984, *Field-Reconfigurable Gate Arrays (FRGAs*, often also called FPGAs), or, *rGAs (reconfigurable gate arrays)* have become the most popular implementation media for digital circuits. For a reading source on the role of rGAs (providing 148 references) also see [21]. The very high and increasing number of design starts on FRGAs demonstrates, that the mask-configurable ASICs have been the looser already years ago (fig. 3 c). The technology-driven progress of FRGAs (for key issues see [22]) is much faster than that of microprocessors. FRGAs with 50 mio system gates are coming soon [23]. It is well known that the

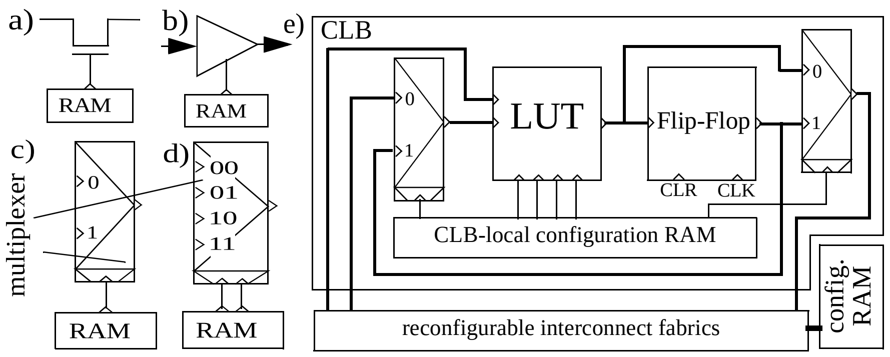


Fig. 6: Programmable switches and blocks used in FRGAs: a) pass transistor, b) tri-state buffer, d) 2-way multiplexer, d) 4-way multiplexer, e) simplified example of a configurable logic block (CLB).

growth rate of the integration density of microprocessors is much slower than Moore’s law. Because of the high degree of layout regularity the integration density of FRGAs, however, is going by the same speed as with Moore’s law [9]. But because of the high percentage of wiring area the transistor density of FRGAs are behind memory two orders of magnitude [9]. However, the number of transistors per chip on FRGAs has overhauled that of microprocessors already in the early 90ies and now is higher by two orders of magnitude [9].

19.2.1. The Role of rGAs

We may distinguish two classes of morphware: *fine grain* reconfigurable morphware, and, *coarse grain* reconfigurable morphware. Reconfigurability of fine granularity means, that the functional blocks have a datapath width of about one bit. This means, that programming at low abstraction level is logic design. Practically all products on the market are *FPGAs* (field-programmable gate arrays, better called *FRGAs* or *rGAs*: (*(field-)reconfigurable gate arrays*), although some vendors prefer different terms as a kind of brand names, like, for instance, PLD (Programmable

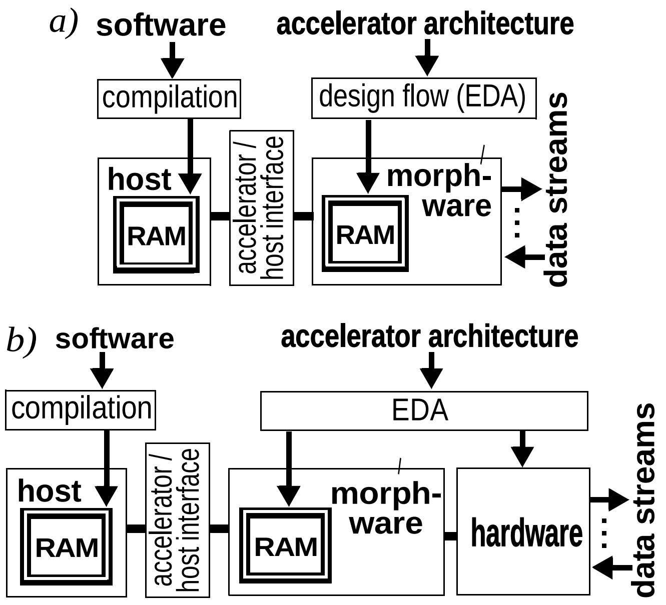


Fig. 7: Traditional embedded computing design flow: a) mo phware-based, b) morphware-hardware-based.

Logic Device, or rLD for reconfigurable logic device). Morphware platforms and their applications have undergone a long sequence of transition. First FPGAs appeared as cheap replacements of *MPGAs* (or *MCGAs*: Mask-Configurable Gate Arrays). Still to-day FRGAs are the reason for shrinking ASIC markets (fig. 3 c), since for FPGAs no application-specific silicon is needed - a dominating cost factor in low production volume products. (ASIC fabrication cost is much lower (only a few specific masks needed) than that of other integrated circuits.) Later the area proceeded into a new model of computing possible with FRGAs. 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* 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.

Meanwhile, morphware is used practically everywhere.

From a terminology point of view the historic acronyms *FPGA* and *FPL* are a bad choice, because programming, i. e. scheduling, is a *procedural* issue in the *time domain*. Also the term *PLD* is a bad choice and should be replaced by *rLD (reconfigurable Logic Device)*. A program determines a *time sequence* of executions. I fact the FP in FPGA and in FPL, acronym for *field-programmable*, actually means field-reconfigurable, which is a structural issue *in the space domain: configuration in space*. In a clearly consistent terminology it would be better to use *FRGA (field-reconfigurable gate array)* or *rGA (reconfigurable gate array)* instead of FPGA. In following parts throughout this chapter the term rGA or FRGA will be used instead o FPGA. For terminology also see fig. 2, fig. 5, and sections 19.2.5 and 19.4.1.

The most important architectural classes of rGAs are [24]: island architecture (Xilinx), hierarchical architecture (Altera), and row-based architecture (Actel). A more historic architecture is *mesh-connected,* sometimes also called *sea of gates* (introduced by Algotronix) [25]. A simple example of a CLB block diagram is shown by figure 6. Its functional principles by multiplexer implementation are shown by figure 9 a and b, where in CMOS technology only 12 transistors are needed for the fully decoded multiplexer (fig. 9 c). The island architecture is illustrated by fig. 10 a. Fig. 10 b shows details of *switch boxes* and *connect boxes*. Fig. 10 c shows the circuit diagram of a *cross point* in a switch box, and, fig. 10 d from within a connect box. The thick wire in fig. 10 b illustrates, how these interconnect resources are configured to connect a pin of one CLB with a pin of another CLB The total configuration of all wires of an application is organized by a *placement and routing* software. Sometimes more interconnect resources are needed than are available, so that for some CLB not all pins can be reached. Due to such *routing congestion* it may happen, that a percentage of CLBs cannot be used.

19.2.2. Commercially available FRGAs

A wide variety of fine grain morphware products is available from a number of vendors, like the market leader Xilinx [26], the second largest vendor Altera [27], and many others. Also a variety of evaluation boards and prototyping boards is offered. COTS (commodity off the shelf) boards for FRGA-based developments are available from Alpha Data, Anapolis, Celoxica, Hunt, Nallatech, and others, to support a broad range of in house developments. As process geometries have shrunk into the deep-submicron region, the logic capacity of FRGAs has greatly increased, making FRGAs a viable implementation alternative for larger and larger designs. FRGAs are available in many different sizes and prices per piece ranging from 10 US-Dollars up to FRGAs with much more than a million usable gates for more than 1000 US-dollars. Xilinx has pre-announced FRGAs with 50 mio system gates for about 2005 [23]. Modern FRGAs support mapping entire systems onto the chip by offering on board all components needed, like several memory banks for user data, one, or several microprocessors like ARM, PowerPC, MIPS, or others, a major number of communication interfaces (WAN, LAN, BoardAN, ChipAN etc.) supporting contemporary standards, up to several GHz bandwidth, JTAG boundary scan circuitry to support testing, sometimes even multipliers.

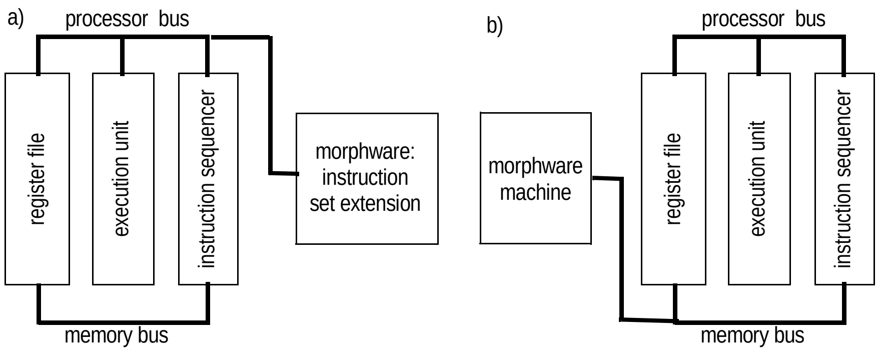
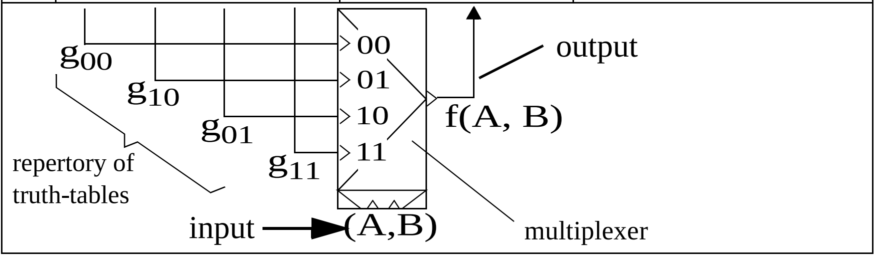
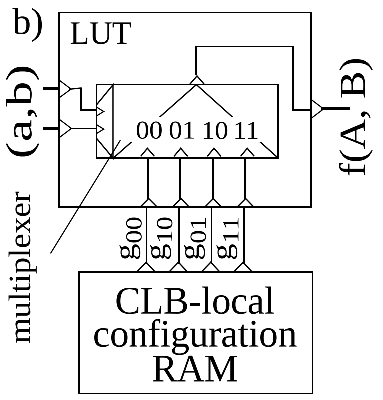


Fig. 8: Alternative morphware applications: a) von Neumann processor with morphware-based instruction set extension, b) von Neumann host with morphware-based co-processor.

Also FRGAs featuring low power dissipation [28] or better radiation tolerance (for aerospace applications) are offered. Several major automotive corporations have contracts with FRGA vendors to develop morphware

a)	configuration bits g_{ab}				f(A, B)	function
#	g_{00}	g_{01}	g_{10}	g_{11}		
0	0	0	0	0	0	constant 0
1	0	0	0	1	A and B	and
2	0	0	1	0	B disables A	if B then 0 else A
3	0	0	1	1	A	identity A
4	0	1	0	0	A disables B	if A then 0 else B
5	0	1	0	1	B	identity B
6	0	1	1	0	A exor B	antivalence
7	0	1	1	1	A or B	or
8	1	0	0	0	not(A or B)	nor
9	1	0	0	1	A coin B	equivalence
10	1	0	1	0	not(B)	negation of B
11	1	0	1	1	B implies A	if A then 1 else ¬ B
12	1	1	0	0	not(A)	negation of A
13	1	1	0	1	A implies B	if B then 1 else ¬ A
14	1	1	1	0	not(A and B)	nand
15	1	1	1	1	1	constant 1





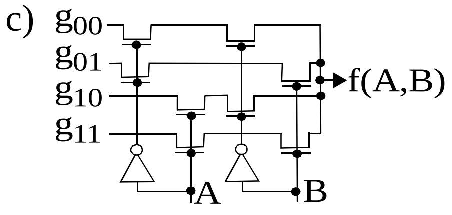


Fig. 9: Illustrating LUT implementation by multiplexer: example for functions of 2 variables: a) illustration of the function generator, b) illustration of LUT (look-up table) block use within CLB (compare figure 6 e), c) multiplexer circuit.

optimized for this branch of industry. Some FRGAs commercially available also support partial column wise reconfiguration, so that different talks may reside in it and may be swapped individually. This may also support *dynamic reconfiguration* (*RTR*: run time reconfiguration), where some tasks may be in the execution state, whereas at the same time other tasks are being reloaded. Dynamic reconfiguration, however, tends to be tricky and difficult to understand and to debug. But static reconfiguration is straight forward and more easy to understand. Because reconfiguration is slow, also multi-context morphware has been discussed, but is not yet available commercially. Multi-context morphware features several alternative internal reconfiguration memory banks, for example 2 or 4 banks, so that reconfiguration can be replaced by an ultra fast context switch to another memory bank.

19.2.3. Applications

Morphware is used practically everywhere, so that this section can mention only a few examples. For applications also see chapter 5 on “Reconfigurable Computing”. Most early FRGA applications have been rapid prototyping [25] [29] [30], rather than directly implementing products on morphware platforms. *Rapid Prototyping* and *ASIC Emulation* still is important for the development of hardwired integrated circuits. Since in IC design flow simulation may take days or even weeks, a remedy has been *ASIC emulation*, using huge emulation machines called *ASIC emulators.*

Early such machines included racks full of boards equipped with masses of FRGAs of the low density available that time. By acquisitions the 3 major EDA vendors now 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 the Xilinx Virtex FRGA series, can emulate up to 3 million gates), and, the DN3000k10 ASIC Emulator from the Dini Group.

Terminology: reconfigurable vs. programmable.
- The semantics is: structural vs. procedural.

Another morphware application area is scientific HPC (high performance computing), where often the desired performance is hard to reach by “traditional” high performance computing. For instance, the gravitating n-body-problem is one of the grand challenges of theoretical physics and

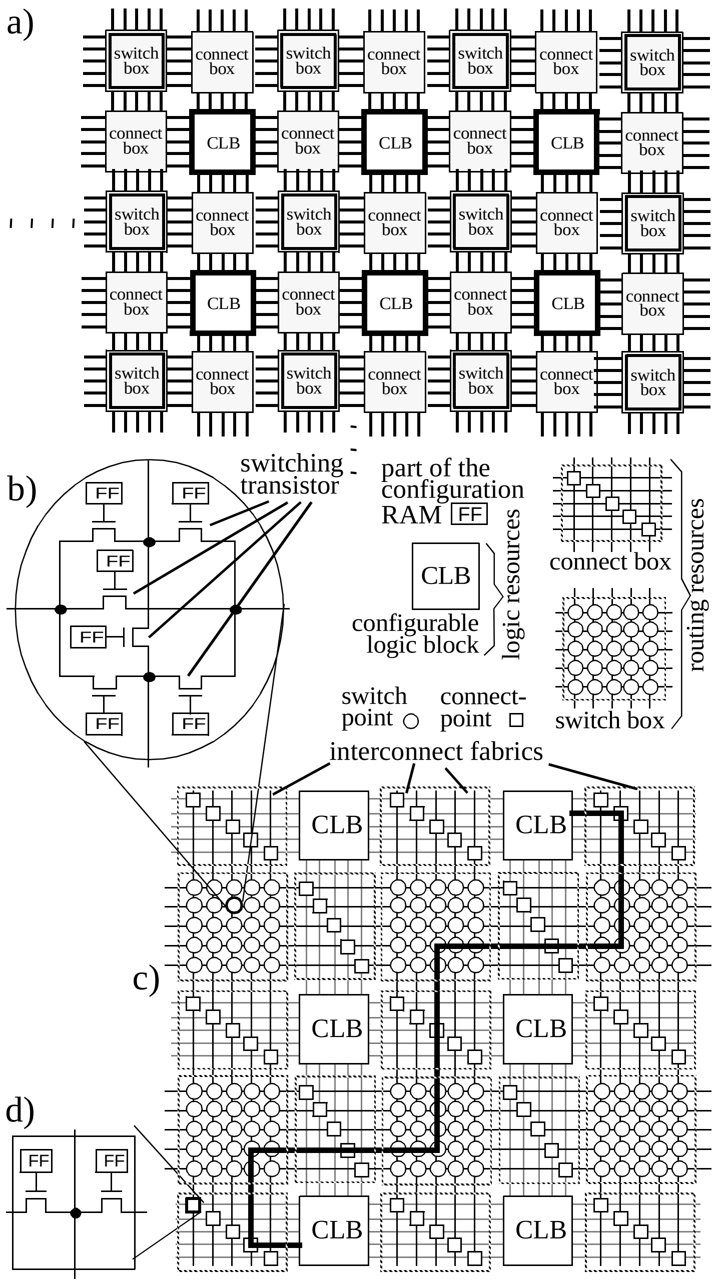


Fig. 10: Illustrating FRGA island architecture fine grain morphware resources: a) global view, b) switch point circuit of a switch box, c) FPGA detailed view (only 1 configured “wire” shown), d) connect point circuit of a connnect box.

astrophysics [31] [32]. 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. For astrophysics the situation improved by the GRAPE special purpose computer [33]. To improve the flexibility a hybrid solution has been introduced with AHA-GRAPE, which includes auxiliary morphware [31]. Other morphware-based machines are WINE II, MDGRAPE [34] MDM (Modular Dynamics Machine) [35] [36] [37] are also used for modeling and simulation in molecular dynamics [31] [33] [38].

Because of the availability of high density FRGAs the scenario has drastically changed. The trend is it to deliver the FRGA-based solution directly to the customer, at least for lower production volumes. Not only micro controllers or simple logic circuits are easy to migrate onto a FRGA platform. Practically everything may be migrated onto morphware. A single FRGA type may replace a variety of IC types. Design and debugging turn-around times can be reduced from several months to weeks or days. Patches or upgrades may take only days, hours, or even minutes, and, may be even carried out at the customer's site, even remotely over the internet or wireless communication, which means a change of the business model - an important benefit for innovative efforts in remote diagnosis and other customer services.

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 [39]. The increasing spare part demand 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. 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

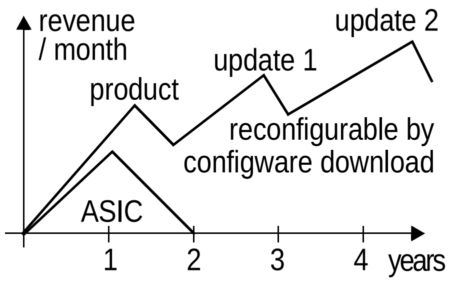


Fig. 11: accelerator longevity [40].

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 FRGAs, since application-specific silicon will not be affordable because of low microchip production volumes in these areas and rapidly increasing mask cost

Fortunately now with FRGAs (field-programmable gate arrays) a new kind of IC platform is available, so that we can switch from hardware to morphware, which can be "re-wired" at run time. Because of their general purpose properties FRGAs are a suitable platform for reverse engineering of unavailable spare parts required. Morphware is the fastest growing segment of the IC market [Dataquest]. Also for industries like automotive, aerospace, military of industrial electronics such a common morphware platform would be a promising route to avoid the very high mask cost, to reduce the number of IC types needed, to accelerate IC time to market, and to solve long term spare part supply problems by retro emulation.

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

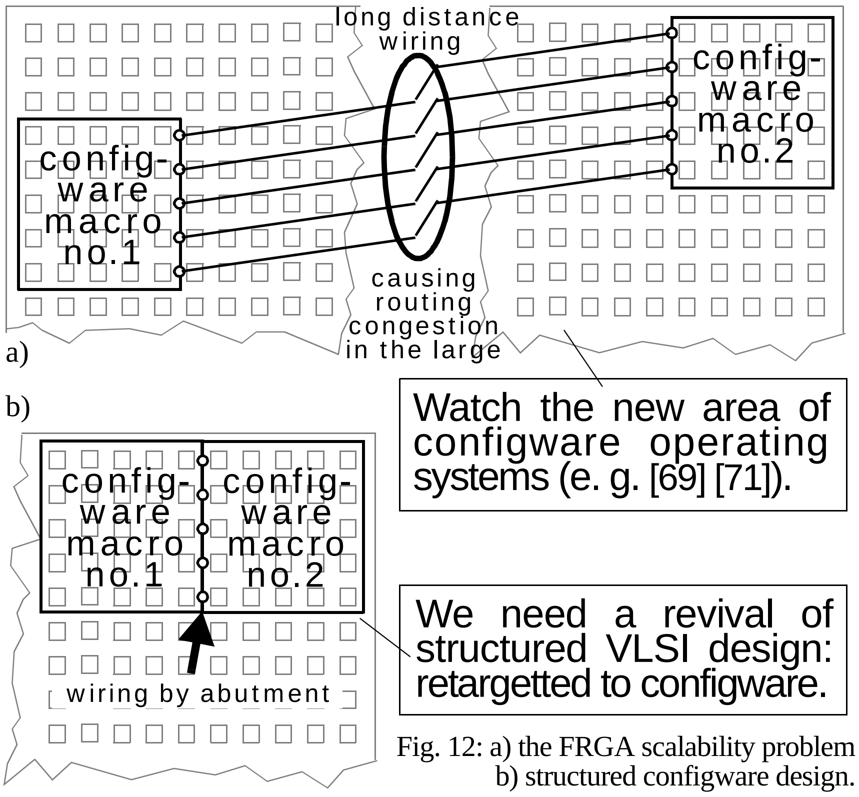
The new business model of Morphware brings a new dimension to digital system development and has a strong impact on SoC design (System-on-Chip). Performance by parallelism is only one part of the story. The time has come to fully exploit morphware flexibility to support very short turn-around time for real-time in-system debugging, profiling, verification, tuning, field-maintenance, and field-upgrades. One of the consequences of the new business model is the adoption of computer science mentality for developing all kinds of electronics products, where patches and upgrades are carried out at the customer's site (figure 11) - even via the internet using run time reconfiguration (RTR). This is also an important remedy of the current embedded system design crisis caused by skyrocketing design cost coincides with decreasing product lifetime, by providing product longevity (figure 1 b).

19.2.4. Application Development support

Morphware is the fastest growing segment of the integrated circuit (IC) market, currently relying on a growing large user base of HDL-savvy designers. A number of books is available which give an introduction to application development using FRGAs [29] [41] [42] [43] [44] [45]. Not only the configware industry is rapidly growing, which offers IP cores [46] and libraries for morphware platforms. Also a rapidly growing branch of the EDA industry offers tools and design environments supporting configware development. Complete design flows from HDL sources like VHDL [47] are offered by Mentor Graphics [48], Synplicity [49], Celoxica [50], and others. A key issue is the integration of IP cores in the design flow. At DAC [51] a task force has been set up to solve standards problems.

A sloppy terminology is a severe problem by torpedoing diffusion and education.

There are also design flows [52] [53] from Matlab sources [54]. Also a tool to generate HDL description from UML has been reported [55]. An emerging trend is going to input sources of higher abstraction levels like the language Handel-C by Celoxica, Precision-C from Mentor Graphics, SystemC [56] [57], a C dialect [58] by Synopsys [59] targeting HW/CW/SW co-design. Matlab indicates a tendency to go to the even higher abstraction level of mathematical formulas. The emerging use of *term re-writing systems (TRS)* for design is another indication of this trend [60] [61] [62] [63].



Watch the new area of configware operating systems (e. g. [69] [71]).

We need a revival of structured VLSI design: retargetted to configware.

Fig. 12: a) the FRGA scalability problem b) structured configware design.

Also a wide variety of vendors is offering tools not covering the entire design flow, like for debugging, timing estimation [64], simulation, verification, placement and routing, and other tasks as well as soft IP cores. Examples are: CoreConnectBus (Xilinx), Parameterizes Processor (Xilinx),

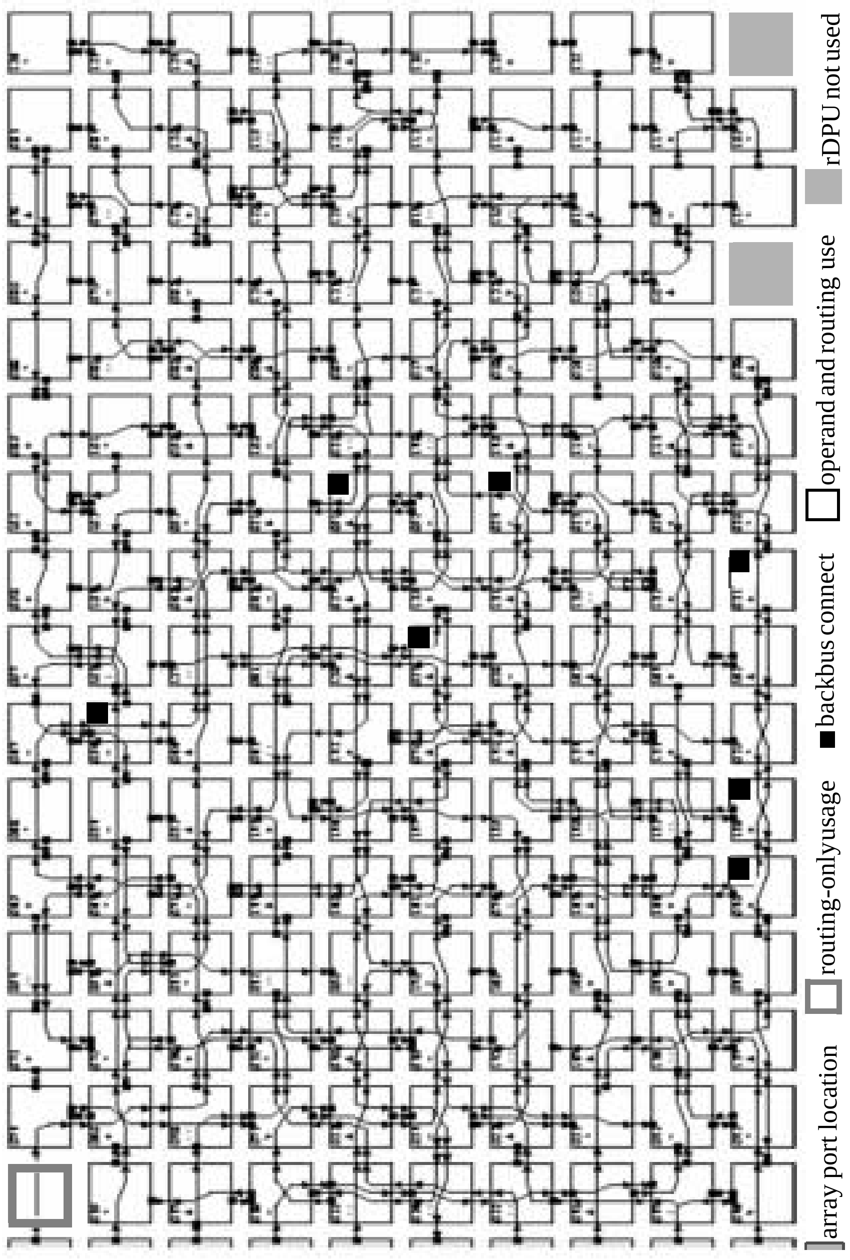


Fig. 13: Example of mapping an application (image processing: SNN filter) onto a (coarse grain) KressArray

IPbus interface (Xilinx), embedded software development tools (Wind River, GNU and others), Integrated Bus Analyzer (Xilinx), board support package for interface software (Xilinx), over 40 processor IP models (Xilinx), [23]. Still a research area are morphware operating systems to load and coordinate multiple tasks to be resident in a single FRGA. PACT has a kind of such an OS for its XPP coarse grain reconfigurable array (see section 19.3), which can be *partly reconfigured* rapidly in parallel while neighboring reconfigurable data path units (rDPUs) are still processing data. Reconfiguration is triggered externally or even by special event signals originating within the array, *enabling self-reconfiguring designs* [65]. In general there is still room for new tools and design flows offering improved quality and designer productivity. Key issues for performance of FRGAs implemented in deep-submicron processes are the following three factors: the quality of the CAD tools used to map circuits into the FRGA, the quality of the FRGA architecture, and the electrical (i.e. transistor-level) design of the FRGA. In order to investigate the quality of different FRGA architectures we need EDA tools capable of automatically implementing circuits in each FRGA architecture of interest.

19.2.5. Education

Also education is an important area of application development support - by avoiding a shortage of qualified professionals. In morphware application the lack of algorithmic cleverness is one of the urgent educational problem. For instance, how to implement a high performance application for low power dissipation on 100 datapath units running at 200 MHz, rather than on one processor running at 20 GHz An example is the migration of an application from a very fast digital signal processor to a low power implementation on FRGA yielding speed-up factors between 5 and 22 [66]. The transformation of the algorithm from the software domain to fine grain morphware took an enormous effort from the student in charge of this project, because such algorithmic cleverness is not yet taught within typical curricula.

The data stream paradigm has been around for almost 3 decades. Software used it indirectly by inefficient instruction-stream implementations. By poor synthesis methodology its direct use by systolic arrays remained a niche til the mid' 90ies.

CS education is becoming more and more important for embedded system development because SoC design rapidly adopts CS mentality [67]. The amount of program code implemented for embedded systems doubles every 10 months and will reach 90% of all code being written by the year 2010 [68]. Currently a typical CS graduate with von-Neumann-only mentality does not have the skills needed for HW / CW / SW partitioning decisions, nor the algorithmic cleverness needed to migrate an application from software onto an FRGA. When teaching important skills needed to map applications onto morphware are not included in our CS curricula, this will cause a major disaster. This means, that our graduates are not qualified for the IT labor market of the near future [72].

Terminology is a key issue. It is very important to maintain a clear and consistent terminology. A sloppy terminology is a severe problem by torpedoing diffusion, education, and efforts to bridge communication gaps

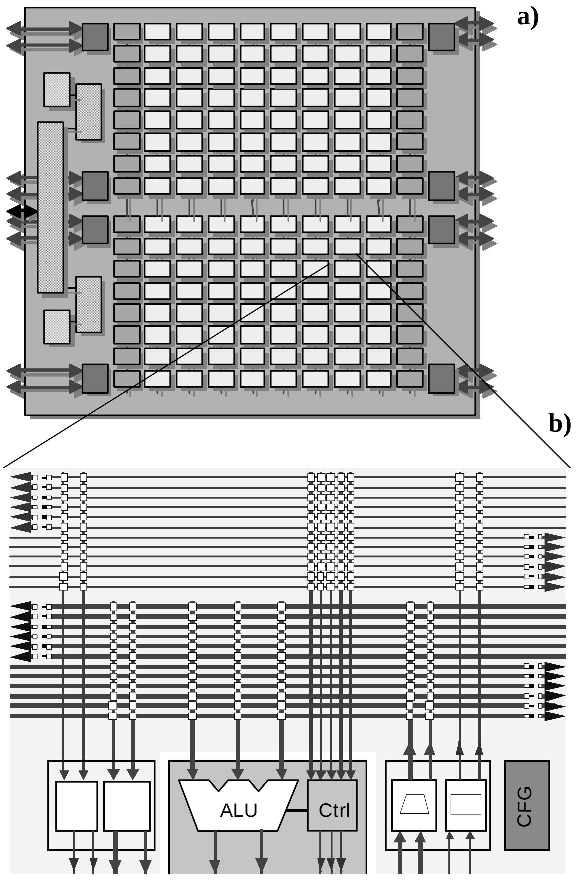


Fig. 14: Configurable XPU (xtreme processing unit from PACT: a) array structure, b) rDPU.

between disciplines. Too many experts using their own non-consensus terminology create massive confusion: their colleagues often do not know what they are really talking about.

I have my own frustrating experience on contradictory terminology when teaching VHDL and Verilog in the same course [73]. For a major part of the terminology the following problems have been confusing the students: for almost each important term of this area there have been usually three different definitions: 1) what the student associates when hearing this term the first time, 2) how it is used by VHDL experts, and, 3) how it is used by Verilog experts.

Terminology should be tightly linked with common models. In both, hardware and software, the *design space* has almost infinite size. Not only students get lost in space without any guidance by models narrowing the design space. A machine paradigm is needed. This has been highly successful for 50 years by the *von Neumann paradigm*. Because of the *dominance of morphware* wee need a new, second, machine paradigm to be used as a general model for guidance by: (1:) its well-defined terminology, (2:) by its simplicity: the *anti machine paradigm* (also see section 19.3.6). The term *reconfigurable* has too many different meanings in many different areas including all-day life. For this reason *morphware* often is much better. Because terminology is so much domain-specific you can guess a person's origin from terminology use. When somebody associates blacksmith with *hardware*, you know, it is not an IT professional. When somebody associates downloading drivers or other software into the RAM of a von Neumann machine with *reconfiguration*, you know that this person is not familiar with morphware and its environment.

19.2.6. Innovative Applications

Terms like *evolvable hardware* (EH, in fact: *evolvable morphware: EM*), *Darwinistic Methods* for system design, or, *biologically inspired system design* point to a newer research area stimulated by the availability of fine grain morphware. For details see chapter 17 “Evolvable Hardware”. Also retro emulation is an innovative application. It is an efficient way of re-engineering unavailable electronics parts for replacement solutions serving 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. But in the future reverse engineering can be avoided in the future, when the implementation of all IC architectures is FRGA-based from the beginning.

The anti machine has no von Neumann bottleneck. No caches are needed.

FRGAs may be good platforms to achieve *fault tolerance* by self-healing mechanisms [74] [75]. Partial re-routing can circumvent wires or CLBs found to be faulty. A NASA single-chip spacecraft is discussed (breaking many paradigms in spacecraft design [76]), which is based on a high-density FRGA, does not need an operating system, and uses such fault tolerance to reduce the

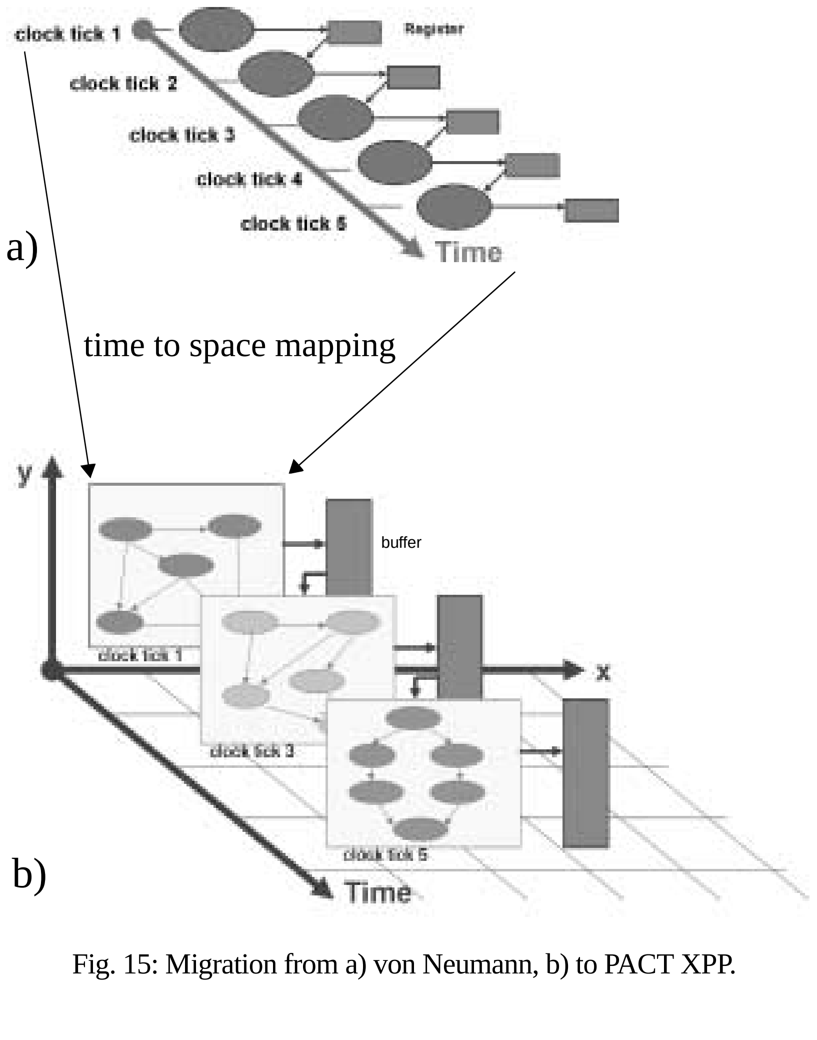


Fig. 15: Migration from a) von Neumann, b) to PACT XPP.

need for radiation hardening. Currently available commercial FRGA architectures insufficiently support such re-arrangements at run time. Some more research is required to obtain better architectural support [74] [77].

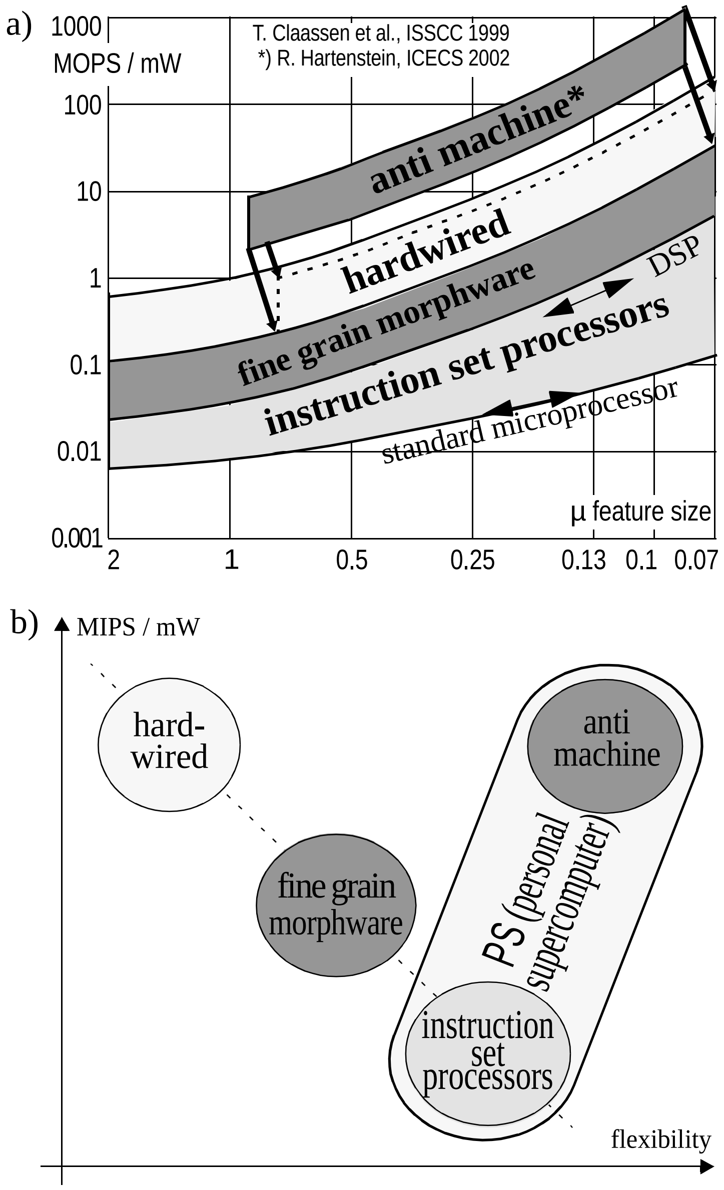


Fig. 16: Performance and Energy efficiency: a) vs. technology generation.b) vs. flexibility.

Another interesting area deals with *soft CPUs*, also called *FRGA CPUs*, i. e. microprocessors implemented by mapping their logic circuits onto an FRGA. Examples are the *MicroBlaze*, a 32 bit Harvard architecture from Xilinx [78], Altera’s *Nios* processor [27], the ESA *SPARC LEON* open source core [79] [80], the *LEON2* processor [81] which is a synthesizable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture, and the *Dragonfly* 8-bit core [78]. Of course, soft processors run about a factor of 3 to 5 slower than their hardwired versions. By the way, designing soft CPUs is a popular subject of lab courses offered by major number of universities.

19.2.7. Scalability and Relocatability

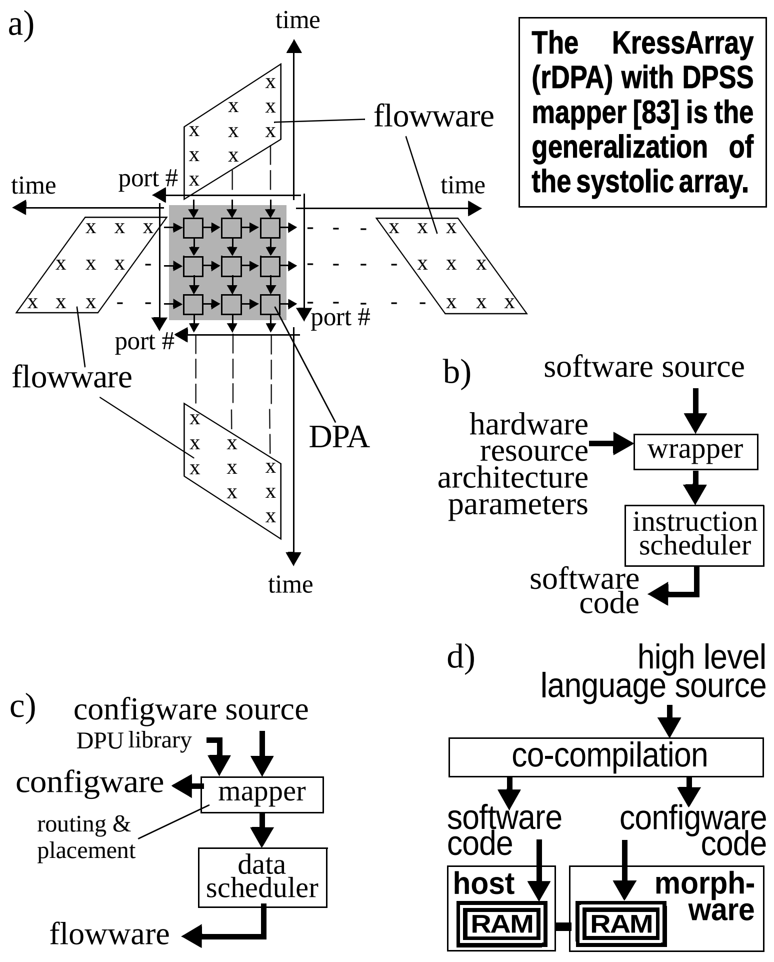
Relocation, even dynamically at run time, of configware macros is subject of the new area of configware operating systems [69] [70] [71]. Some FRGAs are so large, that more than 100 soft CPUs can be mapped onto such a single chip. Will future gigaFRGAs permit mapping practically everything, even including large rDPAs, onto a single morphware chip? This leads to the question of FRGA scalability. For instruction set processors the von Neumann bottleneck guarantees full relocatability of code. Within very large FRGAs, however, relocatability might be limited by routing congestion (fig. 12 a). But *Structured Configware Design* (a design philosophy, derived from structured VLSI design [82]) is a promising approach to solve the relocatability problem (fig. 12 b), so that FRGAs may be universal as microprocessors.

19.3. Coarse Grain Morphware

In contrast to fine grain morphware using CLBs of smallest datapath width (~1 bit), coarse grain morphware uses rDPUs (reconfigurable Data Path Units) with wide data paths, like, e. g. 32 bits wide. Instead of FRGAs we have rDPAs (reconfigurable DPU Arrays). As an example figure 13 shows the result of mapping an image processing application (SNN filter) onto a primarily mesh-based KressArray [83] with 160 rDPUs of 32 bit path width. This array is interfaced to 10 data streams: 9 input streams and 1 output stream. Figure 14 shows some details of the XPU (xtreme processing unit), a commercially available rDPA from PACT AG [84] [85] [86] [87]. Figure 15 illustrates the difference of the execution mechanisms. At vN execution (fig. 15 a) exactly one operation is carried out per clock cycle. Intermediate results are stored in registers. For migration of such an algorithm from vN to a rDPA like PACT XPP (fig. 15 b) a subsequence is mapped from time to space and executed in parallel on the array.

As soon this is completed the next chunk of parallelized code is executed. Intermediate results may be communicated by a buffer (see fig. 15 b).

Usually a rDPA is a pipe network, and is not a multiprocessor or multicomputer network, since DPUs do not show a program counter (for details see later sections of this chapter). Coarse grain morphware has been a research area for more than a decade (for a survey see [88] [89]). Since it plays an important role in wireless communication [90] [91], *software-defined radio* [92] and multimedia processing, not only performance, but also MIPS / mW



The KressArray (rDPA) with DPSS mapper [83] is the generalization of the systolic array.

Fig. 17: Compilation: a) a systolic array example (matrix multiplication) for illustration of flowware and its role, b) compilation for von-Neumann platforms, c) configware / flowware compilation for morphware platforms, d) software / configware co/compilation.

are key issues. Figure 16 shows, that FRGAs just fill the efficiency gap and the flexibility gap between hardwired platforms and instruction set processors. Coarse grain arrays, however, almost reach the efficiency of hardwired platforms (fig. 16), when mesh-based architectures using wiring by abutment are used, so that no separate routing areas are needed [9]. Also configuration memory being an order of magnitude smaller than for FRGAs contributes to this area/power efficiency [9].

Breaking away from the current mind set requires more than traditional technology development and infusion. It requires managerial commitment to a long-term plan to explore new thinking [96].

19.3.1. Pipe Networks and Flowware

We have to distinguish two different domains of programming in time: *instruction scheduling* and *data scheduling*. The programming code for von-Neumann-like devices is an *instruction schedule*, compiled from *software* (fig. 17 b). The programming code for resources like systolic arrays and other DPA (arrays of DPUs) is a *data schedule*, which can be compiled from *flowware* defining, which data item has to appear at which port at which time. Such data schedules manage the flow of *data streams*. This is illustrated by figure 17 a, showing a typical *data stream* notation introduced with *systolic arrays* more than 20 years ago.

The first *flowware-based* paradigm, the systolic array, got stuck in a niche for a long (throughout the 80ies and beyond) because of the wrong synthesis method - until the *super systolic array* made it viable for morphware. This will be explained later. A systolic array [93] [94] [95] is a pipe network. The term *systolic* reminds to the multiple data streams clocked into and out of such a pipe network and its similarity to the heart and the blood streams entering and leaving it. Its DPUs never have instruction sequencers. The mode of DPU operation is transport-triggered by data items. If synchronization is done by handshake instead of clocking a systolic array may be also called *wavefront array.*

The traditional systolic array could be used only for applications with strictly regular data dependencies, because array synthesis methods used linear projections or algebraic methods resembling linear projections. Such synthesis methods yield only strictly uniform arrays with linear pipes. The Data Path Synthesis System (DPSS) [83], however, used simulated annealing instead (the mapper in fig. 17 c), which removed the traditional application limitations, enabling the synthesis of *super-systolic arrays* featuring also any kind of non-uniform arrays with any free form pipes like zigzag, spiral, completely irregular, and many others. By this drastically improved flexibility also reconfigurable arrays (rDPAs) make sense. The KressArray Xplorer including a mapper has been implemented as a design space explorer to optimize rDPU and rDPA architectures [97] [98] [99]. For more details on Xplorer see section Figure 19.3.4.

19.3.2. Data streams and flowware languages

More recently data-stream-based computing has been popularized by a number of academic projects like SCCC [100], SCORE [101] [102], ASPRC [103] BEE [104] [105], KressArray [97] [98] and more [106]). The

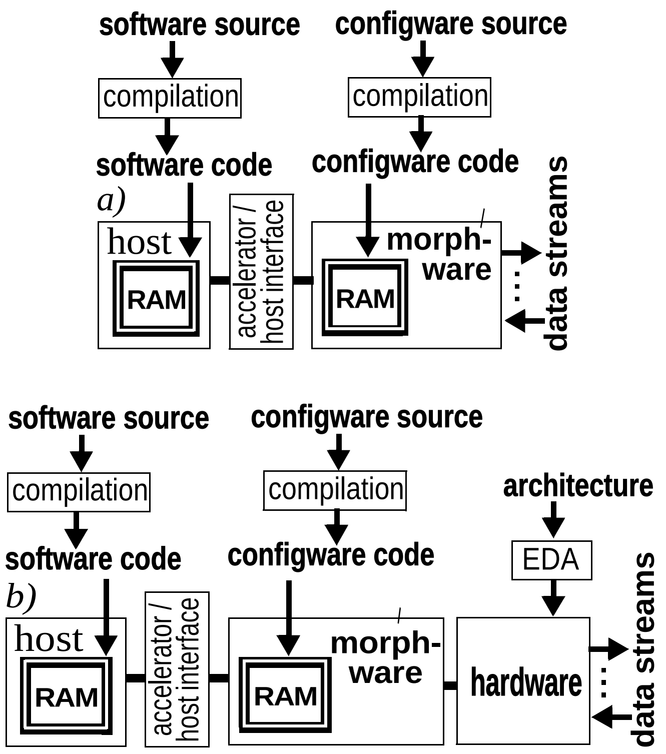


Fig. 18: Modern embedded computing design flow: a) software / configware co-design, b) software / configware / hardware co-design.

specifications of data streams can be expressed by *flowware language*. Data streams are created by executing flowware code on *auto-sequencing memory modules (asM)*. Figure 19 a shows a distributed memory array of such asM modules driving data streams from/to the rDPA surrounded by the asMs. All enabling architectural resources for flowware execution are available [107] [108] [110] [111]. The new R&D discipline of application-specific distributed memory architectures [107] has arrived just in time to provide a methodology of architectural resources for processing flowware. Two alternative memory

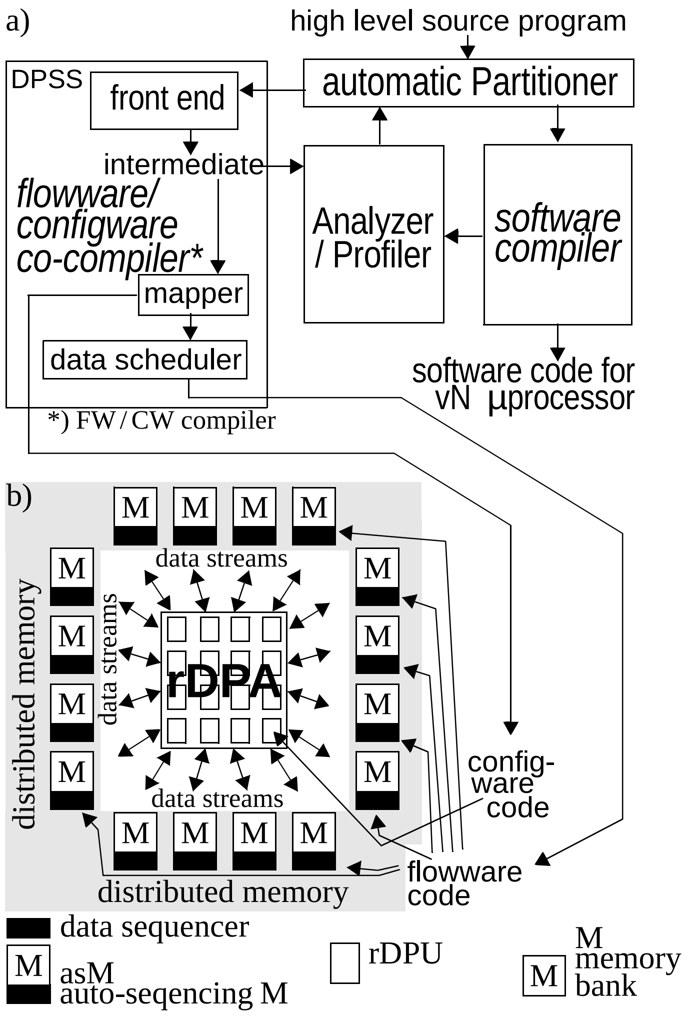


Fig. 19: Flowware / Configware / Software Co-Compilation: a) Becker's partitioning Co-Compiler, b) its anti machine target example.

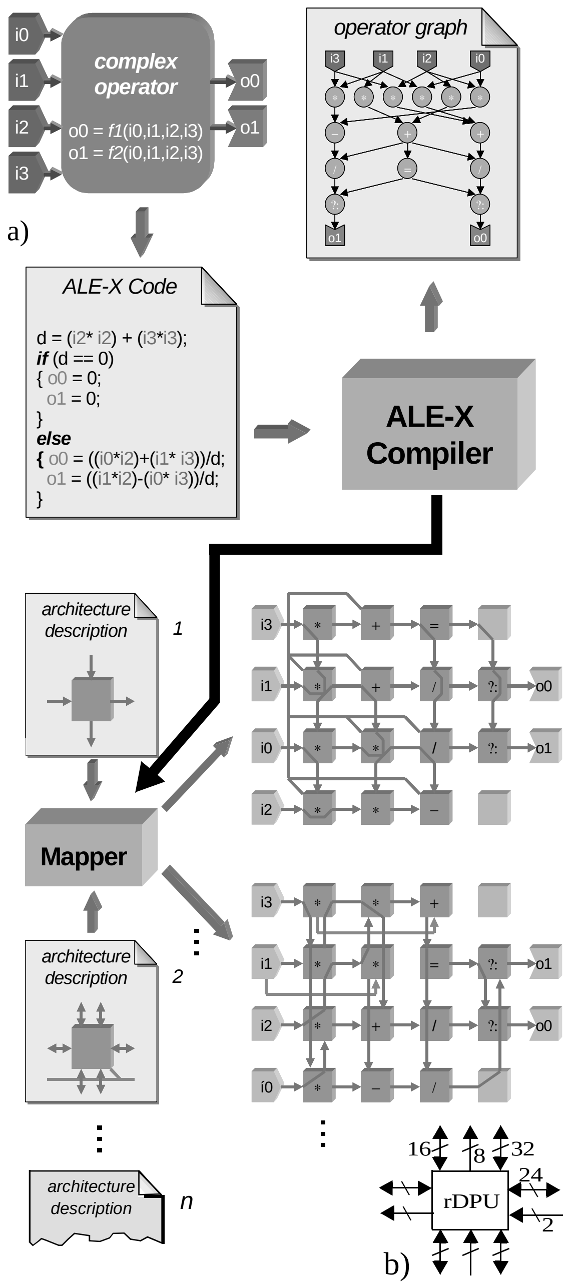
implementation methodologies available [107] [112] [113], either specialized memory architecture using synthesized address generators (e. g. APT by IMEC [107]), or, flexible memory architectures using programmable general purpose address generators [109] [114]. Performance and power efficiency are supported especially by sequencers, which do not need memory cycles even for complex address computations [107], having been used also for a smart memory interface of an early anti machine architecture [114] [115].

Flowware may also be described by higher level flowware languages [116], which are similar to high level software languages like C (fig. 21). Both languages have jumps, loops and nested loops. The main differences between software and flowware is, that flowware semantics is based on one or several data counters, whereas software refers to only a single program counter. Because of multiple data counters flowware also features parallel loops, which is not supported by software languages. Flowware is much more simple, because it does not need to express data manipulation.

Because of the wrong synthesis method the systolic array, first flowware-based paradigm, got stuck in a niche for long - until the super systolic array made it viable for morphware.

For good morphware application development support an integrated synthesis system is useful, which efficiently supports configware / flowware co-design, like, for instance, DPSS [83] (figure 19 b), so that the user does not need to care about configware / flowware interaction. A well-designed dual-paradigm language covering both [116], the flowware paradigm and the configware paradigm, and supporting the communication between both segments, would be useful for designer productivity. Examples for multiple-scope languages are already hardware languages like VHDL [47] or Verilog [43], which support the co-description of hardware and software constructs also alleviate handling of hardware / software communication. The strong trend within EDA toward higher abstraction levels, heralded by new languages like System-C [56] [57] [58] and others, opens a way toward integrated co-design frameworks coordinating all three paradigms covering hardware, morphware, software, configware and flowware.

The flowware-based common model of data-stream-based computing may be used for both, for hardware and morphware. There is in principle no difference, whether DPAs are hardwired or reconfigurable (rDPAs). The only important difference is binding time of placement and routing: before fabrication (hardware), or, after fabrication (morphware, compare fig. 27).



```
d = (i2* i2) + (i3*i3);
if (d == 0)
{ o0 = 0;
  o1 = 0;
}
else
{ o0 = ((i0*i2)+(i1* i3))/d;
  o1 = ((i1*i2)-(i0* i3))/d;
}
```

Fig. 20: KressArray Xplorer (design space explorer [97]): a) simplified example to illustrate platform space exploration by finding an optimized array depending on rDPU architecture (1 or 2), b) KressArray family rDPU example architecture illustrating flexibility.

19.3.3. Coarse Grain Arrays

Because the number of CFBs is by orders of magnitude smaller than that of CLBs in FRGAs, mapping takes only minutes or less instead of hours. Since computational data paths have regular structure potential, full custom designs of *reconfigurable datapath units* (rDPUs) are drastically more area-efficient. Coarse-grained architectures provide operator level CFBs, and very area-efficient datapath routing switches. A major benefit is massive reduction of configuration memory and configuration time, and drastic complexity reduction of the P&R (placement and routing) problem. Several architectures will be briefly outlined (for details see [88]).

Primarily mesh-based architectures arrange their PEs mainly as a rectangular 2-D array with horizontal and vertical connections which supports rich communication resources for efficient parallelism. and encourages nearest neighbor links between adjacent PEs Typically, also longer lines are added with different lengths for connections over distances larger than 1. The *KressArray* [83] is primarily a mesh of rDPUs physically primarily connected through wiring by abutment. *MATRIX* [117] is a multi-granular array of 8-bit CFBs (Basic with vN microprocessor core. *RAW (Reconfigurable Architecture Workstation)* [118] provides a 4 by 4 array RISC multi processor architecture of NN-connected 32-bit modified MIPS R2000 microprocessors. The *DReAM* Array *(Dynamically Reconfigurable Architecture for Mobile Systems* [119]) for next generation wireless communication.

Some RAs are based on one or several linear arrays, like *RaPiD (Reconfigurable Pipelined Datapath)* [120] and *PipeRench* [121]. Architectures using Crossbars are: *PADDI (Programmable Arithmetic Device for DSP)* uses a central reduced crossbar (difficult to rout) and a 2 level hierarchy of segmentable buses. *PADDI-1* [122] [123], *PADDI-2* [124]. The *Pleiades* Architecture [66] is a kind of generalized low power *PADDI-3*.

19.3.4. Compilation Techniques

A first step in introducing morphware-oriented compilation techniques in application development for embedded systems is the replacement of EDA (fig. 7 a and b) by compilation also for the morphware part (fig. 18 a and b). This step of evolution should be accompanied by a clean model having been introduced in the course of history. Partly synchronized to Tsugio Makimoto’s Wave model [9] [10] Nick Tredennick summarizes the history of silicon application [125] by 3 phases (figure 22 a - c). Hardwired components like SSI, MSI, and LSI circuits *cannot be programmed*:

resources fixed and algorithms fixed (figure 22 a). The introduction of the microprocessor changes this to: resources fixed, but algorithms variable (fig. 22 b). We need *only one programming source*: *software* (fig. 22 e). The advent of morphware has introduced: both are variable, resources and algorithms (fig. 22 c). We need *two programming sources: configware* to program the resources, *and*, *flowware* to program the data streams running through the resources (fig. 22 f). An early implementation is the DPSS (fig. 17 c, also see section 19.3.1).

19.3.5. Co-compilation

Separate compilation of software and configware (fig. 18 a and b) gives only a limited support to reach the goal of good designer productivity. Especially to introduce *software / configware / flowware co-design* to CS professionals and CS curricula we need *co-compilation techniques* to support application development at high abstraction levels. Figure 17 c shows the typical structure of a *software / configware partitioning co-compiler* (fig. 17 c), where the configware part (DPSS in fig. 19 b) includes both, *configware code generator* and a *flowware code generator. CoDe-X* has been an early implementation of a compiler of this kind, which is a *partitioning co-compiler* (fig. 19 b and c), accepting C language input (pointers are not supported),

language category	Software Languages	Flowware Languages
sequencing managed by	read next instruction, goto (instruction address), jump (to instruction address), instruction loop, nesting, ***<u>no</u>*** *parallel loops*, escapes, instruction stream branching	read next data item, goto (data address), jump (to data address), data loop, nesting, *parallel loops*, escapes, data stream branching
data manipu-lation	yes	not needed
state register	program counter	single or multiple data counter(s)
Instruction fetch	memory cycle overhead	no overhead
address com-putation	massive memory cycle overhead	drastically reduced overhead

Fig. 21: Software languages versus flowware languages.

which partitions source input to run on a symbiosis of a host and a rDPA [126] [127] [128]. This partitioner (fig. 19 c) is based on the identification of usability of *loop transformations* [129] [130] [131] [132] [133] [134]. This partitioner is implemented via *simulated annealing*. An additional *analyzer / profiler* (fig. 19 c) is used for further optimization. Figure 19 b shows the flowware / configware compiler (a version of the DPSS) as explained above, which is used as a subsystem inside the CoDe-X co-compiler. Figure 20 a gives some DPSS details. *ALE-X* is an intermediate form derived from the C language.

It is time to bridge the hardware / software chasm.
We need a Mead-&-Conway-like edu rush [135].

A newer version of DPSS includes *KressArray Xplorer* (fig. 20 a), a design space explorer to optimize KressArray DPU and rDPA architectures [98] [99]. Mapping based on architecture description 1 yields another array configuration than from architecture description 2. Figure 20 b illustrates he high flexibility of the KressArray family concept accepted by Xplorer. Path width and mode of each nearest neighbor connect can be individually selected. Also a wide variety of second level of *back bus interconnect* resources (not shown in the figure) featuring highly parallel buses or bus segments. Other design space explorers are: *DSEs (Design Space Explorers*, survey: [88]) use automatic guidance systems or design assistants giving advice during the hardware (and morphware) design flow, like by DPE (Design Planning Environment) [136], *Clio* [137] (both for VLSI) and DIA (for ASICs) [138]. *Platform Space Explorers (PSEs)* are used to find an optimum vN processor array, like by DSE [139], *ICOS (Intelligent Concurrent Object-oriented Synthesis)* [140], and *DSE for Multimedia Processors (DSEMMP)* [141].

19.3.6. A dichotomy of two Machine Paradigms

Traditionally hardware experts have been needed for morphware application development (fig. 7 a, compare section 19.2.4). Because of the rapid growth of the amount of code to be implemented for embedded systems [68] also CS graduates are needed to cope with the amount of work to be done. This is hardly possible without moving to higher abstraction levels. Because it focuses the design space like known for software from the *von Neumann paradigm*, a second machine paradigm is needed as a simple guideline to implement flowware (and configware). This *anti machine paradigm* is summarized by fig. 23 b thru d. In contrast to the von Neumann paradigm (fig. 23 a) the sequencer (data counter) has moved to the memory (as part of asM, an auto-sequencing memory bank), whereas the DPU of the anti machine has no sequencer (fig. 23 b). The anti machine paradigm [141] also supports multiple data streams by multiple asMs providing multiple data counters (fig. 23 c, d). That's why the anti machine has no von Neumann bottleneck. It does not need caches because of multiple data streams. Caches do not help because new data mostly have new values. The enabling technologies for the anti machine architecture implementations are available [107] [108] [110] [111] [112] [113] [114] [142] [143] [144]. Figure 26 a shows details of an anti machine mapped onto a KressArray, and fig. b mapped onto a PACT XPP array. The anti machine paradigm is useful for both, morphware-based

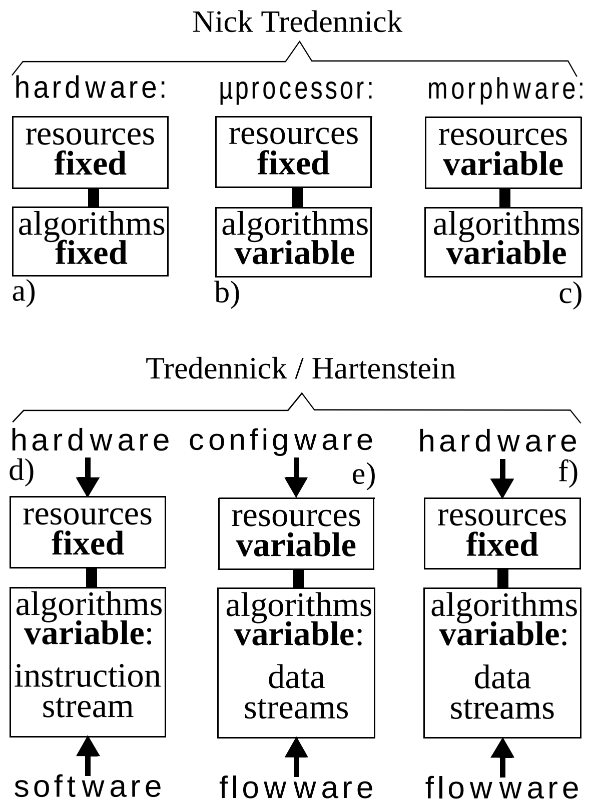
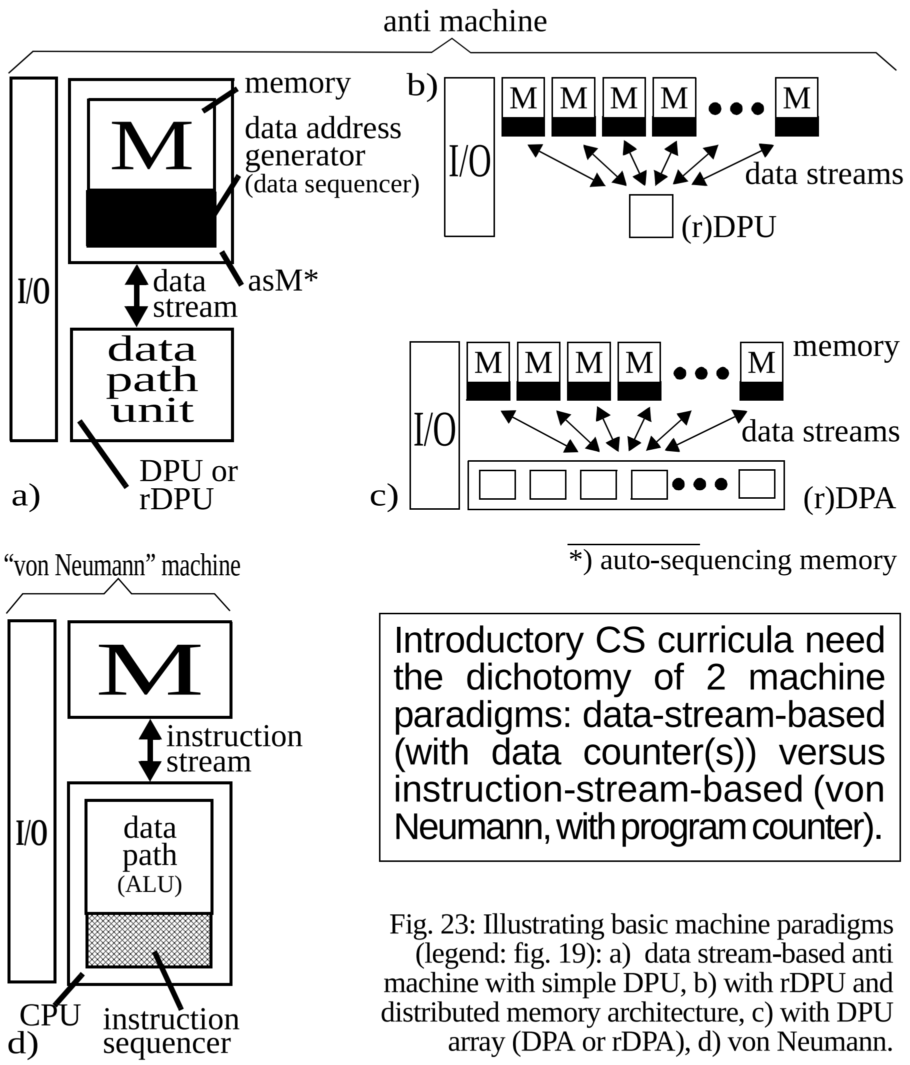


Fig. 22: Nick Tredennick's digital system classification scheme: a) hardwired, b) programmable in time, c) reconfigurable; d) von-Neumann-like machine paradigm, e) reconfigurable anti machine paradigm, f) Broderson's hardwired anti machine.

machines and hardwired machines ([145] etc.). The anti machine should not replace von Neumann. We need both machine paradigms. We need morphware to strengthen the declining vN paradigm.

The anti machine is not a *dataflow machine* [146]. This term cannot be used for the anti machine, because it had been established by an old research area (dead, meanwhile) having worked on an arbitration-driven machine, which checks for each operator, wether all its operands are available. In case of a reject, this operator can be re-submitted later. Such a machine operation is indeterministic and for an algorithm the total order of execution cannot be predicted. The execution of the vN machine and the anti machine, however, are



Introductory CS curricula need the dichotomy of 2 machine paradigms: data-stream-based (with data counter(s)) versus instruction-stream-based (von Neumann, with program counter).

Fig. 23: Illustrating basic machine paradigms (legend: fig. 19): a) data stream-based anti machine with simple DPU, b) with rDPU and distributed memory architecture, c) with DPU array (DPA or rDPA), d) von Neumann.

deterministic. However, the dataflow languages having come along with this indeterministic paradigm [147] could also be useful sources for the anti machine.

19.4. The impact of Morphware on Computing Sciences

The growth rate of algorithmic complexity ([148] see (3) in fig. 24) is higher than that of Moore’s law (1), whereas the growth rate of microprocessor integration density (2) is far behind Moore’s law). The improvement of computational efficiency in terms of mA needed per MIPS (5) has slowed down and goes toward a saturation. The performance requirements for wireless communication (4) is jumping up by huge steps from device generation to device generation. Also in a number of other application areas, like multimedia, or, scientific computing for instance (section fig. 19.2.3), suffer from similar growth of requirements. Traditional HPC needs too much power: about 100W per gigaFLOPS [55]. Coming microprocessor generations promise only marginal performance improvements (fig. 25). A highly promising alternative is the microprocessor interfaced to a suitable coarse grain array (fig. 17 d), maybe for converting a PC into a PS (personal supercomputer). But such a PS will be accepted by the market only, when it comes along with a good co-compiler (fig. 19 b and c), the feasibility of which has been demonstrated [126] [127] [128].

The future of the microprocessor is no more very promising: only marginal improvements can be expected for performance area efficiency (fig. 25). Power dissipation is going worse, generation by generation. The intel Itanium 2 on 130 nm technology with 410 million transistors dissipates 130 Watts at 1,3 Volts operating voltage [91], compared to 130 Watts at 1,6 Volts for the first itanium. Traditional HPC (High Performance Computing) using such or similar microprocessors needs about 100W per gigaFLOPS [55]. Pipelined execution units within vN machines yield only marginal benefit for the price of sophisticated speculative scheduling strategies. Multi-threading needs substantial overhead required for any kind of multiplexing [149]. All these bad messages add to old limitations like the vN bottleneck [9] [147] [150] [151] [152] [153] [154]. Because of the increasing weakness of the microprocessor we need a new computing paradigm as an auxiliary resource to cooperate with the microprocessor (fig. 16 b). Morphware came just in time. Future acceptance of stand-alone operation of morphware is not very likely. Adding a rDPA and a good co-compiler to a microprocessor (fig. 17 d) enables the PC to become a PS (personal supercomputer).

Static reconfiguration is straight forward and easy to understand. But dynamic reconfiguration tends to be tricky and difficult to understand and to debug.

SoC design rapidly adopts CS mentality [67]. The amount of program code implemented for embedded systems doubles every 10 months and will reach 90% of all code being written by the year 2010 [68]. Currently a typical CS graduate with von-Neumann-only mentality does not have the skills needed for HW / CW / SW partitioning decisions, nor the algorithmic cleverness needed to migrate an application from software onto an FRGA. There is a trend to convey Co-design of embedded computing systems from the domain of hardware expertise over to CS methodologies. To cope with this challenge to CS curricula the new anti machine paradigm and new compilation methods are needed.

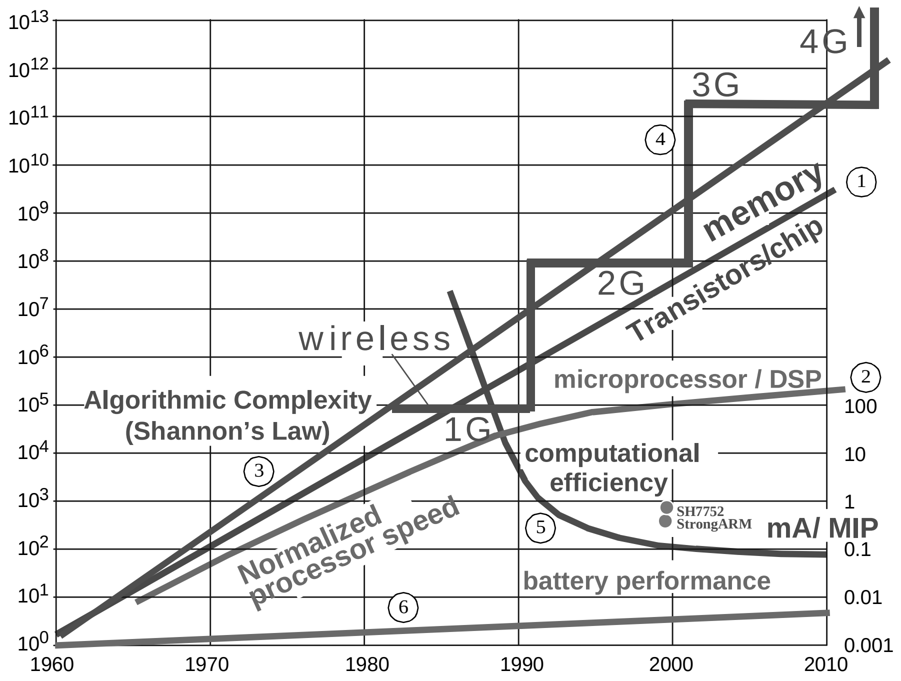


Fig. 24: Computational requirements growing faster than Moore's law.

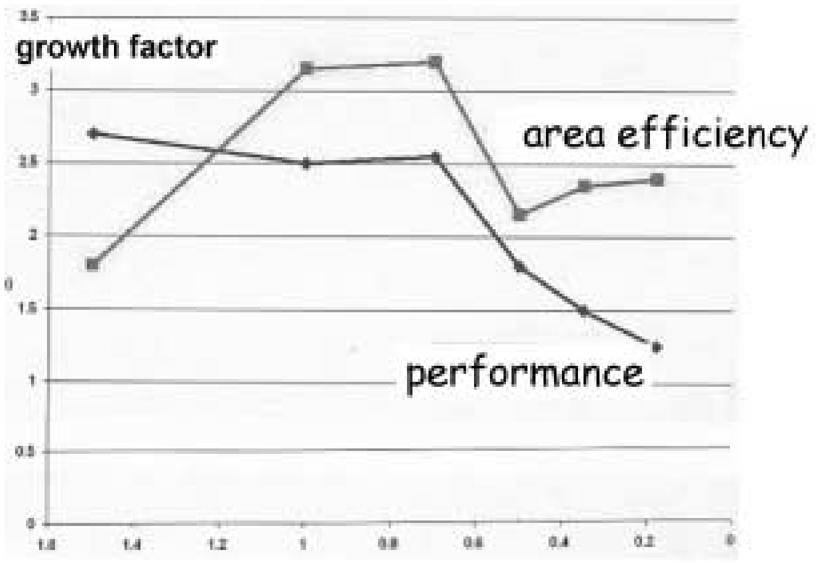


Fig. 25: Pollack's Law [intel]

The hardware / software chasm in professional practice and in education causes a damage amounting to billions of EURO each year worldwide. It is the main reason of the productivity gap in embedded system design. Meanwhile it is widely accepted, that morphware is a new computing paradigm. Morphware provides the enabling fundamentals to cope with this crisis. It is time to bridge the hardware / software chasm. We need Mead-&-Conway-like rush [135]. We are already on the road. Scientific Computing more and more uses Morphware. The international HPC conference IPDPS is coming along with the rapidly growing Reconfigurable Architectures Workshop (RAW [155] [156]). The number of attendees from the HPC scenes coming to conferences like FPL [20] and RAW is rapidly increasing. Special interest groups of professional organizations are changing their scope, like e. g. PARS [32] [157] [158] [159].

There is sufficient evidence that morphware is breaking through as a new computing paradigm. Breaking away from the current mind set requires more than traditional technology development and infusion. It requires managerial commitment to a long-term plan to explore new thinking [96]. Morphware has just achieved its break-through as a second class of RAM-based programmable data processing platforms - counterpart of the RAM-based von Neumann paradigm. Morphware combines very high flexibility by programmability, with the performance and efficiency of hardwired accelerators.

19.4.1. Reconfigurable Computing versus Parallel Processing

A comprehensive treatment of important issues in parallel computing is provided by The Sourcebook for Parallel Computing [150], a key reference giving a thorough introduction to parallel applications, software technologies,

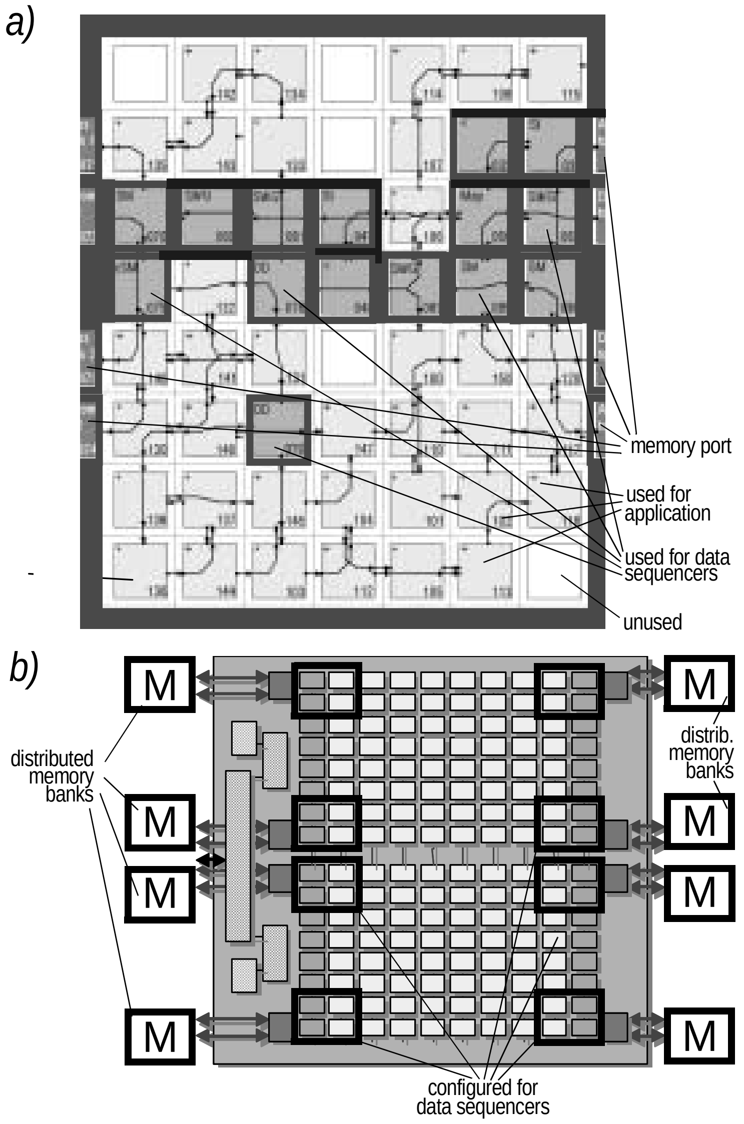


Fig. 26: Anti machine mapped: a) onto KressArray: synthesizable data sequencers mapped by KressArray Xplorer together with an application (linear filter) onto a KressArray, b) onto PACT XPP (other application example, distributed memory shown).

enabling technologies, and algorithms. Classical parallelism by concurrent computing has a number of disadvantages over the parallelism by anti machines having no von Neumann bottleneck, what is discussed elsewhere [105] [114] [151] [152]. In Parallel Computing, unfortunately, the scaling of application performance often cannot match the peak speed the resource platforms seem to provide, and the programming burden for these machines remains heavy. The applications must be programmed to exploit parallelism in the most efficient way possible. Today, the responsibility for achieving the vision of scalable parallelism remains in the hands of the application. Amdahls law explains just one of several reasons of inefficient resource utilization [153]. vN-type processor chips are almost all memory, because the architecture is wrong [105]. Here the metric for what is a good solution has been wrong all the time [105].

Reconfigurable versus parallel computing is also a very important issue for terminology - to avoid confusion. At circuit level all transistors look the same. So the question is how to distinguish switching within a reconfiguration fabrics from other switching activities in an IC. The anti machine model introduces in section 19.3.6 is a good guideline for definition of the term *reconfigurable*. Switching during run time of instruction-stream-based operations, like e. g. addressing the register file, is no reconfiguration. Also switching inside a memory address decoder is not reconfiguration. What about microprogramming? Is it reconfiguration? A microprogrammable instruction set processor can be modelled by the nested machine model showing, that also a microinstruction stream is an instruction stream [149]. This means, that running microcode *is not* *reconfiguration - it is execution* of a micro instruction stream. The following definitions help to avoid confusion. An important difference between Reconfigurable Computing and Concurrent Computing is determined by the binding time (fig. 27). Another important criterion is it, wether the code semantics is *structural* or *procedural*.

- Routing data, addresses, and instructions during run time ***is not*** *reconfiguration*.
- Loading to the program memory of an instruction-stream-driven device ***is not*** *reconfiguration*. It is procedural-style *programming* (instruction scheduling).
- Changing before their run time the effective *structure* of data paths and other resources: ***this definitely is*** *reconfiguration*
- depending on the method used, dynamic reconfiguration (RTR) may be a hybrid, where parts of the system are running to manage the reconfiguration of other parts. (This chapter has already mentioned, that RTR is a quite difficult subject.)

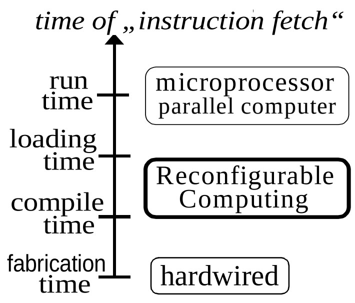


Fig. 27: “Instruction Fetch”.

Within Reconfigurable Computing systems the “instruction fetch” (i. e. set-up of all computational resources and the set-up of all related communication paths) happens *before run time* (fig. 27), what we call *reconfiguration,* because it changes the effective structure of data paths and similar resources. Within Concurrent Computing systems, however, the instruction fetch and set-up of all related communication paths happens *during run time* (fig. 27), which we *do not call reconfiguration.* The main difference with respect to performance is the amount of switching activity at run time, which is low for reconfigurable systems and high for the instruction-stream-driven parallel computing. Depending on the application and the architecture, massively parallel concurrent systems may heavily suffer from communication congestion at run time. Because run time is more precious than compilation time, this migration of switching activities over to compile time or leading time is a welcome performance property of the morphware paradigm. Unfortunately the distinction between parallel and reconfigurable computing is blurred by some projects labelled “reconfigurable”, but, in fact, are dealing with classical parallel computing on a single chip.

19.4.2. New taxonomy needed

We now live in a time exhibiting a shortage of analysts writing good and comprehensive surveys. What is currently missing and should soon be subject of research is all-embracing taxonomy of architectures and algorithms, covering both areas, classical parallel computing and supercomputing, as well as reconfigurable computing. We need a taxonomy of architectures providing guidance in designing modern high performance computing systems using resources from both areas, or, to decide, which area’s resources provide the more promising alternatives. We also need an all-embracing taxonomy algorithms, to support migration of applications or parts of applications from one area to the other area, for instance, from a vN platform to fine grain morphware, or to coarse grain morphware, or mixed platforms. Such a taxonomy of algorithms should also survey the amount of interconnect resources needed by vN to morphware migration. Depending on the algorithm class, the interconnect requirements may show extremely wide variety. Some kinds of algorithms may be very easy to convert into pipelines, whereas others, like for instance the parallelized Viterbi algorithm, may require enormously complex interconnect structures. A new taxonomy should be developed rapidly, which supports the algorithmic cleverness needed for a good morphware-based designer productivity, and for retrieving high quality design solutions.

We need a new taxonomy of architectures and algorithms.

We should not hesitate to reform CS and CSE curricula for avoiding a disqualification for the job market of the near future. Introductory undergraduate programming lab courses should not support the development of a procedural-only mind set. Such courses should rather be a guide to the world of embedded systems requiring the algorithmic cleverness for partitioning an application problem into cooperating software, flowware, and configware blocks. The exercises of such courses should feature a varieties of tasks including several subtasks of different nature, like: (1) software implementation of the problem, (2) its flowware implementation, (3) partitioning the problem into (3 a) a software part and (3 b) a flowware part and (3 c) develop the interface needed for its dual-paradigm co-implementation.

19.5. Conclusions

Morphware has become an essential and indispensable ingredient in SoC (System on a Chip) design and beyond. Already HDLs like VHDL (which is an Ada dialect), Verilog (a C dialect), or others, are languages at higher abstraction levels, and should be taught also to CS students.

The hardware / software chasm in professional practice and in education causes a damage amounting to billions of EURO each year worldwide. It is the main reason of the productivity gap in embedded system design. Meanwhile it is widely accepted, that morphware is a new computing paradigm. Morphware provides the enabling fundamentals to cope with this crisis.

But most current work on reconfigurable systems is specialized and is not motivated by long term aspects - wearing blinders limiting the view to particular applications, architectures, or tools. The long term view, however, shows a heavy impact of reconfigurable computing onto the intellectual infrastructures of CS and CSE. This chapter has drafted a road map for upgrading CS and CSE curricula and for bridging the gap between procedural and structural mentality. The impact of morphware on CS helps to achieve this by evolution, rather than by revolution. You all should be evangelists for the diffusion of the visions needed to go this road out of the current crisis.

19.6. Literature

[1] http://www.darpa.mil/ipto/programs/pca/vision.htm

[2] http://morphware.net/

[3] A. Burks, H. Goldstein, J. von Neumann: Preliminary discussion of the logical design of an electronic computing instrument; US Army Ordnance Department Report 1946.

[4] Goldstein, H., von Neumann, J., and Burks, A.: Report on the mathematical and logical aspects of an electronic computing instrument; Princeton Institute of advanced study, 1947.

[5] D. Jansen et al.: The Electronic Design Automation Handbook; Kluwer, 2003

[6] P. Gillick: State of the art FPGA development tools; Reconfigurable Computing Workshop, Orsay, France, Sept. 2003

[7] M. J. Smith: Application Specific Integrated Circuits; Addison Wesley 1997

[8] D. Chinnery, K. Keutzer: Closing the Gap between ASIC & Custom; Kluwer 2002

[9] R. Hartenstein (invited paper): The Microprocessor is no more General Purpose; Proc. IEEE International Symposium on Innovative Systems (ISIS), Austin, Texas, 1997

[10] T. Makimoto (keynote): The Rising Wave of Field-Programmability; Proc. FPL 2000, Villach, Austria, August 27 - 30, 2000; Springer Verlag, Heidelberg/New York, 2000

[11] Faggin, F., Hoff, M., Mazor, S., and Shima, M.: The history of 4004; IEEE Micro, Dec. 1996

[12] J. Becker (Invited Tutorial): Reconfigurable Computing Systems; Proceedings Escola de Microeletrônica da SBC - Sul (EMICRO 2003), Rio Grande, Brasil, September 2003

[13] B. Lewis, Gartner Dataquest, October 28, 2002

[14] P. Athanas: An adaptive machine architecture and compiler for dynamic processor reconfiguration; Ph. D. thesis, Brown University, Providence, Rhode Island, 1992.

[15] S. Vassiliadis, S. Wong and S. Cotofana: The MOLEN rm-coded Processor; Proc. FPL 2001

[16] M. Iliopoulos, T. Antonakopoulos: Reconfigurable Network Processors Based on Field-Programmable System Level Integrates Circuits; Proc. FPL 2000

[17] http://www.fccm.org

[18] R. Hartenstein: Custom Computing Machines; DMM’95, Smolenice, Slovakia, 1995

[19] http://www.springer.de/comp/lncs/

[20] http://fpl.org

[21] S. Hauck: The Role of FPGAs in Reprogrammable Systems; Proc. IEEE, 1998

[22] V. Betz, J, Rose, A. Marquardt (editors): Architecture and CAD for Deep-Submicron FPGas; Kluwer, 1999

[23] S. Hoffmann: Modern FPGAs, Reconfigurable Platforms and their Design Tools; Proc. REASON summer school; Ljubljana, Slovenia, August 11-13, 2003

[24] D. Soudris et al.: Survey of existing fine grain reconfigurable hardware platforms; Deliverable D9, AMDREL consortium (Architectures and Methodologies for Dynamically Reconfigurable Logic); 2002

[25] J. Oldfield, R. Dorf: Field-Programmable Gate Arrays: Reconfigurable Logic for Rapid Prototyping and Implementation of Digital Systems; Wiley-Interscience, 1995

[26] http://www.xilinx.com

[27] http://www.altera.com

[28] V. George, J. Rabaey: Low-Energy FPGAs: Architecture and Design; Kluwer, 2001

[29] Z. Salcic, A. Smailagic: Digital Systems Design and Prototyping Using Field Programmable Logic; Kluwer, 1997

[30] J. Hamblen, M. Furman: Rapid Prototyping of Digital Systems; Kluwer, 2001

[31] R. Männer, R. Spurzem et al.: AHA-GRAPE: Adaptive Hydrodynamic Architecture - GRAvity PipE; Proc. FPL 1999

[32] G. Lienhart: Beschleunigung Hydrodynamischer N-Körper-Simulationen mit Rekonfigurierbaren Rechensystemen; Joint 33rd Speedup and 19th PARS Workshop; Basel, Switzerland, March 19 - 21, 2003

[33] N. Ebisuzaki et al.; 1997 Astrophysical Journal, 480, pp. 432,

[34] T. Narumi, R. Susukita, H. Furusawa, T. Ebisuzaki: 46 Tflops Special-purpose Computer for Molecular Dynamics Simulations: WINE-2; Proc. 5th Int'l Conf. on Signal Processing, pp. 575-582, Beijing, 2000.

[35] T. Narumi, R. Susukita, T. Koishi, K. Yasuoka, H. Furusawa, A. Kawai, T. Ebisuzaki: 1.34 Tflops Molecular Dynamics Simulation for NaCl with a Special-Purpose Computer: MDM; SC2000, Dallas, 2000

[36] T. Narumi, A. Kawai, T. Koishi: An 8.61 Tflop/s Molecular Dynamics Simulation for NaCl with a Special-Purpose Computer: MDM; SC2001, Denver, 2001

[37] T. Narumi, R. Susukita, T. Ebisuzaki, G. McNiven, B. Elmegreen: Molecular dynamics machine: Special-purpose computer for molecular dynamics simulations; Molecular Simulation, vol. 21, pp. 401-415, 1999

[38] T. Narumi: Special-purpose computer for molecular dynamics simulations; Ph D dissertation, University of Tokyo, 1998

[39] T. Thurner: Trends in der Automobile-Elektronik; GI/ITG FG AH - Zielplan-Workshop at FDL 2003, Frankfurt / Main, Germany, September 22, 2003

[40] T. Kean (invited keynote): It‘s FPL, Jim - but not as we know it! Market Opportunities for the new Commercial Architectures; Proc. FPL 2000

[41] R. Zeidman: Designing with FPGAs and CPLDs; CMP Books, 2002

[42] U. Meyer-Baese: Digital Signal Processing with Field Programmable Gate Arrays (With CD-ROM); Springer Verlag, 2001

[43] K. Coffman: Real World FPGA Design with Verilog; Prentice Hall, 1999

[44] R. Seals, G. Whapshott: Programmable Logic: PLDs and FPGAs: McGraw-Hill, 1997

[45] G. Martin, H. Chang (editor): Winning the SoC Revolution: Experiences in Real Design; Kluwer, 2003

[46] G. Ou, M. Potkonjak: Intellectual Property Protection in VLSI Design; Kluwer 2003

[47] P. J. Ashenden: The Designer's Guide to VHDL, 2nd Edition, Morgan Kaufmann, 2001

[48] http://www.mentor.com/fpga/

[49] http://www.synplicity.com/

[50] http://www.celoxica.com/

[51] http://www.dac.com

[52] http://www.mathworks.com/products/connections/ product_main.shtml?prod_id=304

[53] http://www.celoxica.com/methodology/matlab.asp

[54] http://www.mathworks.com/

[55] I. Jones: DARPA funded Directions in Embedded Computing; Reconfigurable Computing Workshop, Orsay, France, Sept. 2003

[56] T. Grötker et al.: System Design with System-C; Kluwer, 2002

[57] http://www.synopsys.com/products/cocentric_systemC/ cocentric_systemC_ds.html

[58] http://www.systemc.org/

[59] http://www.synopsys.com/

[60] J. Hoe, Arvind: Hardware Synthesis from Term Rewriting Systems; Proc. VLSI'99 Lisbon, Portugal

[61] M. Ayala-Rincón, et al.: Efficient Computation of Algebraic Operations over Dynamically Reconfigurable Systems Specified by Rewriting-Logic Environments, Proc. 23rd SCCC. IEEE CS press 2003

[62] M. Ayala-Rincón, et al.: Architectural Specification, Exploration and Simulation Through Rewriting-Logic; Colombian Journal of Computation, Vol 3(2):20-34, 2003.

[63] M. Ayala-Rincón et al.: Using Rewtiting-Logic Notation for Functional Verification in Data-Stream-based Reconfigurable Computing; Proc. FDL 2003 (Forum on Specification and Design Languages), Frankfurt / Main, Germany, September 23-26, 2003

[64] P. Bjureus et al.: FPGA Resource and Timing Estimation from Matlab Execution Traces; 10th Int'l Workshop on Hardware/Software Codesign, Estes Park, Colorado, May 6-8, 2002

[65] V. Baumgarten, G. Ehlers, F. May, A. Nückel, M. Vorbach, M. Weinhardt: PACT XPP - A Self-Reconfigurable Data Processing Architecture; The Journal of Supercomputing, vol. 26, no. 2, Sept. 2003, pp. 167-184, Kluwer Academic Publishers

[66] J. Rabaey: Reconfigurable Processing: The Solution to Low-Power Programmable DSP, Proc. ICASSP 1997

[67] http://public.itrs.net/Files/2002Update/2002Update.htm

[68] N. N., Department of Trade and Industry (DTI), London, UK, 2001

[69] H. Simmler et al.: Multitasking on FPGA Coprocessors; Proc. FPL 2000

[70] H. Walder, M. Platzner: Reconfigurable Hardware Operating Systems: From Design Concepts to Realizations; Proc. ERSA 2003

[71] H. Walder M. Platzner: A Runtime Environment for Reconfigurable Hardware Operating Systems; Proc. FPL 2004

[72] R. Hartenstein (invited paper): Reconfigurable Computing: urging a revision of basic CS curricula; Proc. 15th Int'l Conf. on Systems Engineering (ICSENG02), Las Vegas, USA, 6-8 Aug. 2002

[73] course ID=27 in: http://vlsi1.engr.utk.edu/~bouldin/COURSES/HTML/courselist.html

[74] C. Stroud et al.: BIST-Based Diagnosis of FPGA Interconnect; Proc.IEEE Int'l Test Conf., 2002

[75] P. Zipf: A Fault Tolerance Technique for Field-Programmable Logic Arrays; Dissertation, Univ. Siegen, Germany, 2002

[76] http.//directreadout.gsfc.nasa.gov

[77] M. Abramovici, C, Stroud: Improved BIST-Based Diagnosis of FPGA Logic Blocks; Proc. IEEE Int'l Test Conf. 2000

[78] http://www.xilinx.com/events/docs/esc_sf2001_microblaze.pdf

[79] http://www.leox.org/

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

[81] http://www.gaisler.com/leonmain.html

[82] C. Mead, L. Conway: Introduction to VLSI Systems Design; Addison-Wesley, 1980

[83] R. Kress et al.: A Datapath Synthesis System (DPSS) for the Reconfigurable Datapath Architecture; Proc. ASP-DAC'95

[84] http://pactcorp.com

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

[86] 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

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

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

[89] W. Mangione-Smith et al.: Current Issues in Configurable Computing Research; IEEE Computer, Dec 1997

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

[91] M. Sauer: Issues in Concept Development for Embedded Wireless SoCs; GI/ ITG FG AH - Zielplan-Workshop; Frankfurt / Main, Germany, Sept 22, 2003

[92] A. Wiesler, F. Jondral: A Software Radio for Second and Third Generation Mobile Systems; IEEE Trans. on Vehicular Technology, Vol. 51, No. 4, July 2002

[93] N. Petkov: Systolic Parallel Processing; North-Holland; 1992

[94] M. Foster, H. Kung: Design of Special-Purpose VLSI Chips: Example and Opinions. ISCA 1980

[95] H. T. Kung: Why Systolic Architectures? IEEE Computer 15(1): 37-46 (1982)

[96] http://directreadout.gsfc.nasa.gov

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

[98] U. Nageldinger: Coarse-grained Reconfigurable Architectures Design Space Exploration; Dissertation, 2001 -- downloadable from [99]

[99] http://xputers.informatik.uni-kl.de/papers/publications/ NageldingerDiss.html

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

[101] T. J. Callahan: Instruction-Level Parallelism for Reconfigurable Computing; FPL’98

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

[103] T. Callahan: Adapting Software Pipelining for Reconfigurable Computing; CASES 2000

[104] H. Kwok-Hay So, BEE: A Reconfigurable Emulation Engine for Digital Signal Processing Hardware; M.S. thesis, UC Berkeley, 2000

[105] C. Chang, K. Kuusilinna, R. Broderson: The Biggascale Emulation Engine; FPGA 2002

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

[107] M. Herz et al. (invited paper): Memory Organization for Data-Stream-based Reconfigurable Computing; ICECS 2002,

[108] M. Herz, et al.: A Novel Sequencer Hardware for Application Specific Computing; Proc. ASAP‘97

[109] H. Reinig et al.: Novel Sequencer Hardware for High-Speed Signal Processing; Proc. Design Methodologies for Microelectronics, Smolenice, Slovakia, Sept.1995

[110] M. Herz: High Performance Memory Communication Architectures for Coarse-grained Reconfigurable Computing Systems; Ph. D. thesis, Kaiserslautern, 2001 -- downloadable from: [111]

[111] http://xputers.informatik.uni-kl.de/papers/publications/HerzDiss.html

[112] F. Catthoor et al.: Data Access and Storage Management for Embedded Programmable Processors; Kluwer, 2002

[113] F. Catthoor et al.: Custom Memory Management Methodology Exploration of Memory Organization for Embedded Multimedia Systems Design; Kluwer, 1998

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

[115] A. Hirschbiel et al.: A Flexible Architecture for Image Processing; Microprocessing and Microprogramming, vol 21, pp 65-72, 1987

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

[117] E. Mirsky, A. DeHon: MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources; Proc. IEEE FCCM'96, Napa, CA, USA, April 17-19, 1996.

[118] E. Waingold et al.: Baring it all to Software: RAW Machines; IEEE Computer, September 1997, pp. 86-93.

[119] J. Becker et al.: Architecture and Application of a Dynamically Reconfigurable Hardware Array for Future Mobile Communication Systems; Proc. FCCM'00, Napa, CA, USA, April 17-19, 2000.

[120] C. Ebeling et al.: RaPiD: Reconfigurable Pipelined Datapath; Proc. FPL'96

[121] S. C. Goldstein et al.: PipeRench: A Coprocessor for Streaming Multimedia Acceleration; Proc. ISCA'99, Atlanta, May 2-4, 1999

[122] D. Chen and J. Rabaey: PADDI: Programmable arithmetic devices for digital signal processing; VLSI Signal Processing IV, IEEE Press 1990.

[123] D. C. Chen, J. M. Rabaey: A Reconfigurable Multiprocessor IC for Rapid Prototyping of Algorithmic-Specific High-Speed DSP Data Paths; IEEE J. Solid-State Circuits, Vol. 27, No. 12, Dec. 1992.

[124] A. K. W. Yeung, J.M. Rabaey: A Reconfigurable Data-driven Multiprocessor Architecture for Rapid Prototyping of High Throughput DSP Algorithms; Proc. HICSS-26, Kauai, Hawaii, Jan. 1993.

[125] N. Tredennick: Technology and Business: Forces Driving Microprocessor Evolution; Proc. IEEE Dec. 1995

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

[127] J. Becker: A Partitioning Compiler for Computers with Xputer-based Accelerators, Ph. D. Dissertation, University of Kaiserslautern 1997 - downloadable from [128]

[128] http://xputers.informatik.uni-kl.de/papers/publications/BeckerDiss.pdf

[129] L. Lamport: The Parallel Execution of Do-Loops; C. ACM 17,2, Febr. 1974

[130] D. Loveman: Program Improvement by Source-to-Source Transformation; J. ACM 24,1, Jan. 1977

[131] W. Abu-Sufah, D. Kuck, D. Lawrie: On the Performance Enhancement of Paging Systems Through Program Analysis and Transformations; IEEE-Trans. C-30(5), May 1981

[132] U. Banerjee: Speed-up of Ordinary Programs; Ph.D. Thesis, University of Illinois at Urbana-Champaign, DCS Report No. UIUCDCS-R-79-989, Oct. 1979.

[133] J. Allen, K. Kennedy: Automatic Loop Interchange'; Proc. ACM SIGPLAN'84, Symp. on Compiler Construction, Montreal,Canada, SIGPLAN Notices 19, 6, June 1984

[134] J. Becker, K. Schmidt: Automatic Parallelism Exploitation for FPL-based Accelerators; Hawaii Int'l. Conf. on System Sciences (HICSS'98), Big Island, Hawaii,1998

[135] http://xputers.informatik.uni-kl.de/staff/hartenstein/eishistory_en.html

[136] D. Knapp et al.: The ADAM Design Planning Engine, IEEE Trans CAD, July 1991

[137] J. Lopez et al.: Design Assistance for CAD Frameworks; Proc. EURODAC'62, Hamburg, Germany, Sept. 7-10, 1992

[138] L. Guerra et al.: A Methodology for Guided Behavioral Level Optimization; Proc. DAC'98, San Francisco, June 15-19, 1998

[139] C. A. Moritz et al.: Hot Pages: Software Caching for RAW Microprocessors; MIT, LCS-TM-599, Cambridge, MA, Aug. 1999.

[140] P.-A. Hsiung et al.: PSM: An Object-oriented Synthesis Approach to Multiprocessor Design; IEEE Trans VLSI Systems 4/1, March 1999

[141] J. Kin et al.: Power Efficient Media Processor Design Space Exploration; Proc. DAC'99, New Orleans, June 21-25, 1999http://anti-machine.org

[142] K. Schmidt et. al.: A Novel ASIC Design Approach Based on a New Machine Paradigm; J. SSC 1991 - invited reprint from Proc. ESSCIRC 1990

[143] W. Nebel et al.: PISA, a CAD Package and Special Hardware for Pixel-oriented Layout Analysis; ICCAD 1984

[144] 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, - invited reprint fr. Proc. InfoJapan'90 (Int'l Conf. memorating the 30th Anniversary Computer Society of Japan), Tokyo, Japan, 1990

[145] C. Chang et al: The Biggascale Emulation Engine (Bee); summer retreat 2001, UC Berkeley

[146] D. Gajski et al.: A second opinion on dataflow machines; Computer, Febr. 1982

[147] J. Backus: Can programming be liberated from the von Neumann style? A functional style and its algebra of programs; Communications of the ACM, August 1978, 20(8):613-641.

[148] J. Rabaey (keynote): Silicon platforms for the next generation wireless systems; Proc. FPL 2000

[149] G. Koch et al.: The Universal Bus Considered Harmful; Proc. 1st EUROMICRO Symposium on the microarchitecture of computing systems; Nice, France, 1975; North Holland, 1975

[150] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, A. White (editors): The Sourcebook of Parallel Computing; Morgan Kaufmann 2002

[151] Arvind et al.: A critique of Multiprocessing the von Neumann Style; Proc. ISCA 1983

[152] G. Bell (keynote): All the Chips Outside: The Architecture Challenge; Proc. ISCA 2000

[153] G. Amdahl: Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities; AFIPS Conference Proceedings, 1967 (30).

[154] J. Hennessy: ISCA25: Looking Backward, Looking Forward; Proc. ISCA 1999

[155] http://www.ece.lsu.edu/vaidy/raw04/

[156] http://xputers.informatik.uni-kl.de/raw/index_raw.html

[157] http://www.iti.uni-luebeck.de/PARS/

[158] http://www.speedup.ch/

[159] http://www.hoise.com/primeur/03/articles/monthly/AE-PR-04-03-61.html