The microprocessor is no longer general purpose: why future reconfigurable platforms will win

Conference Paper · November 1997

DOI: 10.1109/ICISS.1997.630241 · Source: IEEE Xplore

Reiner Hartenstein
RPTU - Rheinland-Pfälzische Technische Universität Kaiserslautern Landau
312 PUBLICATIONS **3,189** CITATIONS

The Microprocessor is no more General Purpose: why Future Reconfigurable Platforms will win

Reiner Hartenstein *(Invited Paper)*

Universitaet Kaiserslautern, Postfach 3049, D-67653 Kaiserslautern, Germany
fax: +49 631 205 2640, home fax: +49 7251 14823
http://xputers.informatik.uni-kl.de - hartenst@rhrk.uni-kl.de

Abstract

The paper is a plaidoyer for a radical methodological change in R&D of dynamically reconfigurable circuits. The paper illustrates, that the current main stream approach based on placement and routing is not very likely to obtain the area-efficiency and throughput needed to cope with the emerging crisis cost of future silicon technology generations. The proposed changes include both: architectural principles and fundamental issues in application development support environments. The paper illustrates the feasibility of general purpose programmable accelerators and their commercialization.

The paper highlights computer systems' increasing dependency on add-on accelerators. It shows, why only by a new methodology reconfigurable hardware will overcome its role as a niche technology and become competitive to ASICs and other hardwired accelerators. It illustrates the possible coming crisis of ASIC design based on wasting chip area by placement and routing and discusses the vision of software-only implementation of accelerators.

Preface

The Microprocessor is a Methusalem

Using still machine principles from the 40ies and after more than ten technology generations (fig. 2) the microprocessor is a kind of Methusalem. The so-called "von Neuman" (micro)processor is no more general purpose. More and more add-on accelerators are needed: ICs and boards. The variety of microelectronic (sub)system designs is exploding and product life times are shrinking to sometimes less than a year. Taylored designs are dominant over general purpose designs. A rapidly growing massive human wave is needed to complete and upgrade all these highly complex designs.

A comparable situation has been in the 70ies, when the term "design crisis" has been coined. The variety of MSI and LSI circuit types had exploded: mostly taylored designs for calculators, clocks, radios, TV, etc. This created an increasing desire for a change, later also leading toward a dominance of general purpose designs: microprocessors, microcontrollers, memory and others (fig. 1). The paradigm switch: replacing many designs by procedural programming. Again the variety of specific circuits and boards is exploding, as well as the design manpower needed for product development and redesigning. The dominance of placement and routing as a design stile causes an enormous wasting of chip area: the main reason of the design gap (fig. 2). Due to fab line cost up to one billion US-Dollars and beyond, growing by a factor of two each technology generation (every three years), and slowing down of technology progress expected toward the end of next decade, we are facing a second design crisis.

We're facing a second design crisis

This raises the question: will such a Makimoto wave repeat in the near future? Will the current dominance of specific circuits again lead to focusing more on general purpose circuits? But this cannot be achieved on the basis of the microprocessor. Makimoto predicts (fig. 1), that in the future reconfigurable hardware will dominate over specialized integrated circuits. The paradigm switch: replace many designs by structural programming.

But, being too slow and too area-inefficient, the fine granularity field-programmable circuits currently available commercially are no possible basis of such a new wave. Even to a larger extent as in designs with standard cells or macro cells the percentage of chip area needed just for wiring is by far too high and unavoidable critical paths are too long. But the trend toward coarse granularity reconfigurable platforms [2][3] could make Makimoto's prediction become true.

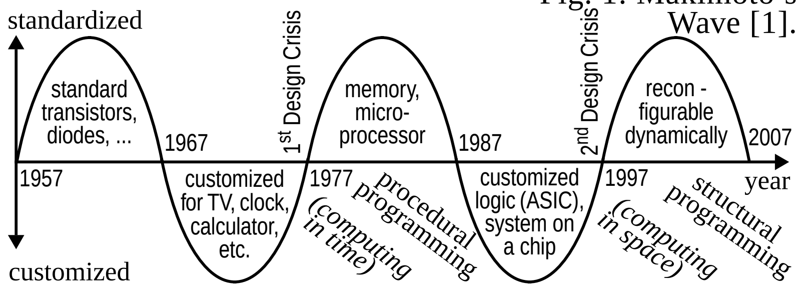


Fig. 1: Makimoto's Wave [1].

Anyway a much better area-efficiency is needed for both, hardwired and reconfigurable ICs, as soon as future generation fabrication equipment will become unaffordable (by the year 2010, or later, or earlier?). The innovation dynamics of the industry may come to an end, probably creating a major crisis of high tech trade and industry. In such a situation the only way to obtain more performance is drastically increased area-efficiency - instead of waiting for a new technology generation (which then eventually may not come). The timely availability and commercial affordability of new switching devices currently is pure speculation. With technology could being no more the innovation driver, a continued dynamics can be driven only by an EDA revolution providing increasing area-efficiency and speed: A new design revolution is a must.

Introduction

Computer designers have lost their constant struggle to find the right balance between speed and generality [4]. Throughput requirements grow faster than speed. Computers sometimes are even too slow to drive their own display. Generality has disappeared. Computers that modify their hardware circuits as they operate are opening a new era in computer design [5][6][7]: the area of dynamically reconfigurable circuits and their applications [8][9]. They excel at pattern recognition, image processing and encryption [10], but also in many other areas. Reconfigurable computing is still a young field, although Gerald Estrin of the University of California at Los Angeles proposed configurable computing already in the late 1960s.

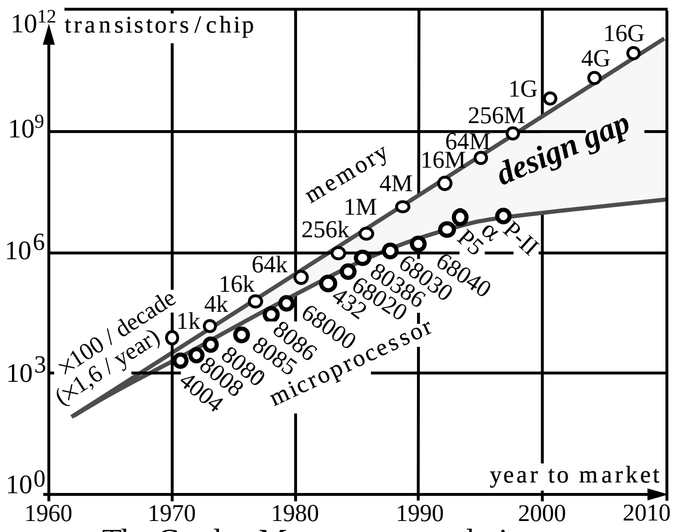


Fig. 2: The Gordon Moore curve w. design gap [11].

The paper advocates a fundamentally new approach to the design of general purpose dynamically reconfigurable accelerator hardware platforms, which are much more area-efficient than the current fine granularity approach. The paper briefly illustrates the feasibility of substantial speed-ups for a variety of application areas. The vision behind this approach is creating and upgrading accelerators by downloading reconfiguration code generated by compilers accepting high level programming language sources — onto a general purpose reconfigurable accelerator.

Such a scenario uses hardware/software co-design supported by partitioning compilers which automatically optimize the hardware/software trade-off (host / accelerator trade-off, supporting also hand-honed improvements). An implementation of such a compiler, having been published earlier, will be briefly recalled. Finally the paper tries to stimulate a discussion on the feasibility to apply a similar approach to the design of hardwired ASICs — for substantially higher area-efficiency and speed.

Resources	% chip area occupied (estimated)	
	bit level granularity	Kress Array
active circuits	1	70
configuration RAM	10	15
wiring	~90	15

Fig. 3: Chip area utilization: FPGA[2] vs. Kress Array[4].

Placement and Routing considered harmful

In both areas, ASIC design and using reconfigurable platforms, the designs obtained usually are highly area-inefficient, and show unavoidable long critical paths. A main reason is the dominance of placement and routing in current EDA environments. The EDA methodology used for placement and routing is highly sophisticated, so that optimization results are poor for complexity reasons, also because of the fast progress in technology. Within the past two decades masses of P&R researchers have provided only marginal improvements. The methodology is incomprehensible to the user, so that hardware specialists are needed to implement a design problem — even in using field-programmable platforms.

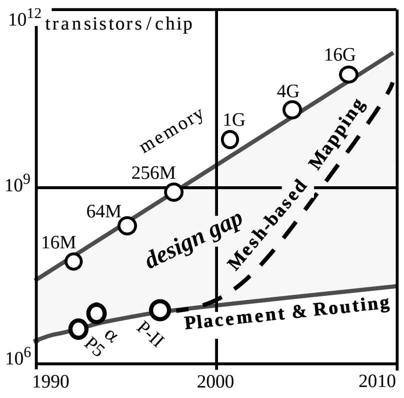


Fig. 4: From traditional placement and routing dominated CAD to Mesh-based Mapping style to bridge the design gap.

High complexity because of fine granularity hardware with up to more than 100,000 reconfigurable, and up to more than a million hardwired gates on a chip permits only a low quality optimization. A main problem is the quality of the placement algorithm used before routing is carried out. It uses connectivity data being of much less relevance than that later needed for good results in routing optimization. This is illustrated by a small design example (fig. 7) [11]. Fig 7 a shows the standard cell result with automatic placement (the area, excluding bonding pads, is taken as 100 %). Fig 7 b shows the result (area: 64.18 %) obtained after carefully hand-honed placement: in total 35.82% area have been saved.

Let us concentrate on the area occupied by the routing channels, which characterizes the influence of placement quality on routing results in a much better way, than the total area. Within the above example the area occupied by routing channels only has been reduced (from 100%) to 41%, i. e. 59% routing area has been saved. This illustrates the bad quality of automatic placement - compared to placement optimized by hand. Fig 7 c shows the area needed for a manually optimized full custom style design (area reduced to 7.04 %), where the total area needed has been reduced by more than an order of magnitude.

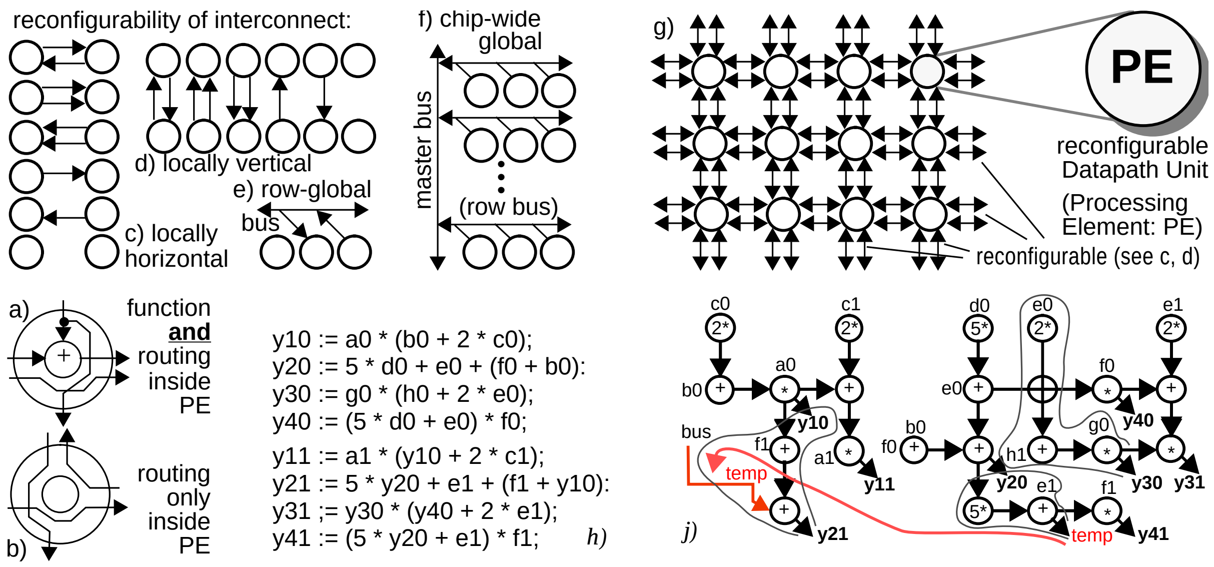
Placement and Routing in FPGAs

A design revolution is a must

Even drastically more harmful than in designs for hardwired accelerators is Placement and Routing as the dominant design style for reconfigurable architectures like FPGAs, where a PE (processing element), also called CLB (configurable logic block), or, LUT (if look-up-table-based), is only a single bit wide and includes just a few gates. For function selection it needs 4 or more flipflops with at least 4 gates per flipflop of configuration RAM. The fine granularity FPGAs commercially available today use only about 1% chip area for active logic circuits and about 90% for wiring [2] (fig. 3), from which a major part is used for reconfigurable routing areas. With severe difficulty commercial FPGAs route in most designs less than 80-90% of the available LUTs — in some cases even only about 50%.

Also massive space for switches, together with associated RAM, is needed for routing resources (also see caricature in fig. 5). The density of the routing switch population has been investigated by Rose and Brown [12]. Brown and Rose estimate that a moderately sized FPGA requires 200 - 400 switches per 4-LUT [12][13]. The distribution of line lengths has been studied by Soon Ong Seo [14]. The merits of hierarchical interconnect are discussed by Agerwal and Lewis [15], Greater word widths increase wiring requirements (but not with the mesh-based approach, see next section), but decreases switching requirements [2]. For a survey see in [2].

Placement and routing is one of the deeper reasons, why reconfigurable circuits commercially available are caught in a niche market, why FPGAs are obsolete, But the dominance of placement and routing is harmful also to the scene of hardwired ASIC design. It will probably be the root of a coming design crisis. This is a challenge to R&D: we need a fundamentally new approach to the ASIC design problem. We need a new design revolution. Could the future development of reconfigurable architectures and their design tools help create new ideas to cope with the coming design crisis? Could mesh-based reconfigurables be the solution? (see next section).



```
y10 := a0 * (b0 + 2 * c0);
y20 := 5 * d0 + e0 + (f0 + b0):
y30 := g0 * (h0 + 2 * e0);
y40 := (5 * d0 + e0) * f0;
y11 := a1 * (y10 + 2 * c1);
y21 := 5 * y20 + e1 + (f1 + y10):
y31 ;= y30 * (y40 + 2 * e1);
y41 := (5 * y20 + e1) * f1;
```

Fig. 6: Illustrating the Kress Array: a-g) architecture example summary: a) intra-PE and operand routing, b) intra-PE routing, c-d) nearest neighbor routing, e) local bus routing, f) global bus routing, g) 4-by-4 mesh interconnect, h) a design problem, j) 4-by-8 PE array solution of (h).

Mesh-based Reconfigurable Circuits.

Currently the mainstream design style sees design cost as a dominant factor. But its immense wasting of chip area cannot be tolerated in the future, when technology progress and the affordability of fabrication reach their limits. The design gap could be substantially narrowed (fig. 4) by extending the use of full custom design style instead of design styles based on channel routers and switchbox routers. This section introduces a new form of such a high density methodology for dynamically reconfigurable circuits. It is a new form of mesh-connected PE array (also called NEWS network) with direct interconnect of a PE only to its north, east, west and south neighbor, having the following advantages:

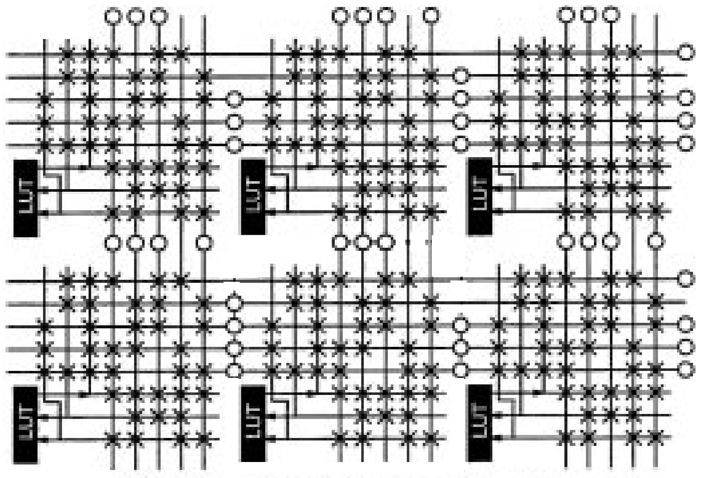


Fig. 5: Interconnect Caricature [2] (o, x : switches).

- coarse granularity permits full custom style processing elements (PEs) instead of single bit CLBs
- a mesh-connected PE array, similar to PE arrays known from 2-dimensional systolic arrays, allows wiring by abutment instead of using routing channels
- because of the absence of routing channels the structure synthesis is carried out by a placement-only algorithm. Kress uses a simulated annealing optimizer [4].

Placement and Routing could be the root of the coming Design Crisis

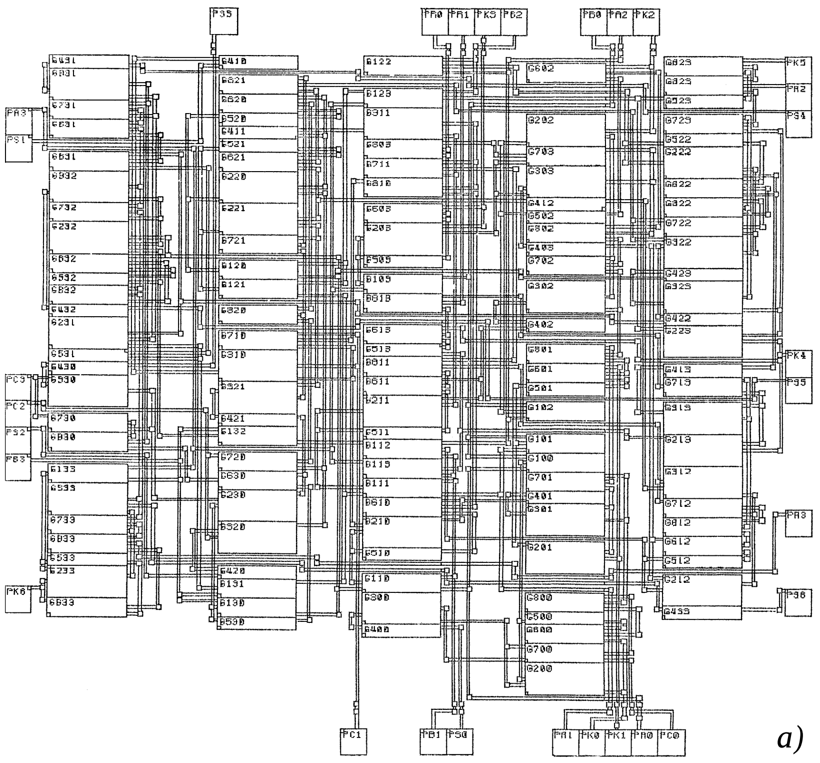
We call this method “mesh-based Mapping”. The application problem is mapped onto a mesh-based reconfigurable architecture, having been designed in full custom design style also using wiring by abutment. Mesh-based reconfigurable is not new: Already in the early 80ies it has been commercialized by Algotronix in Edinburgh — but only at single bit granularity. New, however, is coarse grain mesh-based reconfigurable, which provides the following main advantages over the current main stream solution by placement and routing.

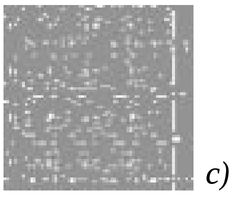
- very high area-efficiency through custom layout style
- much better optimization results through a restricted design space

The bad optimization results of Placement and Routing are also determined by the enormous size of the design solution space, which is a severe complexity problem.

New is also the way, how an application is mapped onto that meshed array. The predefined mesh scheme of the Mesh-based Mapping approach makes the optimization problem much more simple. The synthesis problem has been mainly reduced to a placement problem only, with only a small residual routing problem, which also is solved by placement of routing elements into a particular PE (selecting a PE to aid routing, e. g. see fig. 6 j, where one of 22 PEs is used as a routing resource).

An architecture example of the mesh-based approach is the Kress Array concept [4], which includes a mapper DPSS (data path synthesis system) being a simulated annealing optimizer, as well as a data scheduler to organize and optimize data streams for communication with the host. Instead of only 1% as in fine granularity FPGAs this Kress Array approach uses about 70% chip area for active PE circuitry (fig. 3), which is an improvement by more than an order of magnitude: not very far from almost two orders of magnitude.





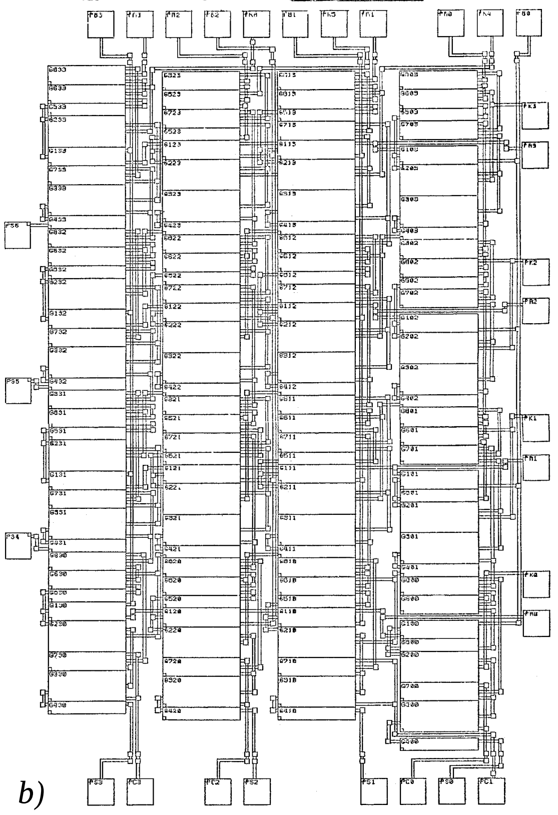


Fig. 7: Illustrating area-efficiency (4 bit integer multiplier example): a) standard cells with automatic placement (area without pads: 100 %), b) standard cells: hand-honed placement (64.18 %), c) full custom (7.04 %).

A Generalization of the Systolic Array

Structured VLSI full custom design style use provides the most powerful and area-efficient VLSI circuit implementations. Typical targets are register files, memories, microprocessor CPUs, ALUs: iterative architectural structures consisting of rows or matrixes of bit slices. But the only very compact structured solution for other applications and including a systematic synthesis method is the systolic array, which provides massively pipelined highly parallel solutions for highly sophisticated application algorithms like systems of equations. Their high performance stems from massively parallel multiple pipelines, running in parallel, also in opposite directions, as well as crossing each other.

The systolic array is the most area-efficient and throughput-efficient hardware design style known [16][17][18]. The synthesis methods known for systolic arrays, however, support only a small class of applications. That's why most other applications are implemented by standard cells (also macro cells) routing and placement, which is highly inefficient - in space and time. The systolic array concept permits solutions:

- only from algorithms with regular data dependencies
- with linear pipelines only
- with full length pipelines only (covering the entire array size)
- where all PEs (processing elements) have to be the same
- where parallel skewed data streams to/from many array I/O ports have to be organized

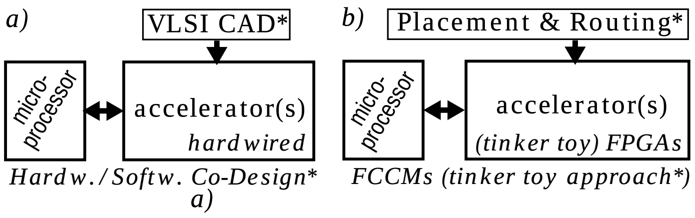


Fig. 9: The microprocessor meanwhile usually needs special add-on accelerators: a) hardwired, b) FPGAs.

*) hardware expert needed for accelerator synthesis

The most severe restriction is, that it may be used only for design problems with regular data dependencies (only for "systolic algorithms"). That's why also from the reconfigurable mesh-connected PE array scheme with wiring by abutment capability usually a severe lack of flexibility is expected. But in fact it can be made dramatically more flexible than expected. The structured regular layout is not the problem with the systolic array. The problem is the mathematical synthesis method for systolic arrays, which uses linear projections yielding only linear pipes and perfectly regular interconnect, as well as permitting a uniform ensemble of PEs only.

Mesh-based Mapping: new is the router-less mapping onto the mesh

The solution is to sacrifice mathematical methods to obtain a generalization of the systolic array to achieve a dramatic increase of flexibility. Although using the same basic layout scheme the Kress Array [4] provides the drastically improved flexibility by different local details (fig. 8). Figure 6 summarizes a newer Kress Array architecture [19]. There are several main differences:

- locally individual PE functionality permitted (e. g. see fig. 6 j)
- locally individual PE use as function, routing element, or both allowed (fig. 6 a and b)
- multiple pipelines may have any free form individually: linear, meandering, zig-zag, spiral, feed back, forks, joins, and others (for examples see fig. 6 j).
- locally individually neighbor PE mesh interconnect configuration (see fig. 6 j).
- also torus-style wrap-around connect is supported [4].
- an additional global bus multiplexed to row buses for reaching all PEs

#	feature	systolic array	Kress Array
1	data paths	multiple pipelines: parallel, opposite directions, orthogonal to e. a.	
2	circuit layout	structured VLSI design (full custom) capable, wiring by abutment, highly area-efficient	
3	applications executable	regular data dependencies only	any — no restrictions
4	interconnect	hardwired	programmable at 5 levels
5	total interconnect pattern	fully connected strictly regular mesh only	no restrictions: globally, locally & PE-internally individual
6	ensemble of PE functions	uniform only	locally individual
7	pipeline shape	linear only	free form: linear, meandering, zig-zag, spiral, feed back, forks, joins, random ...
8	data stream synthesis method	projection (linear only)	optimizing scheduler
9	method of array synthesis, (or, configuration, respectively)	projection (linear only)	simulated annealing optimizer
10	generate data streams running	no systematics published	pre-scheduled / smart interface

Fig. 8: The Kress Array, generalization of the Systolic Array - differences and similari-

■ optimizer used for synthesis instead of linear projection (or placement & routing)

These and more differences to the systolic array are summarized in the table in fig. 8. The Kress array may be used for any of a very wide variety of applications. There are no restrictions with respect to the patterns of data dependencies. The limit (for the degree of parallelization) is only the array size. But the Kress Array is scalable, also beyond chip boundaries.

In total the Kress Array architecture (fig. 6) provides 5 levels of reconfiguration: 1). operand routing inside a particular PE (fig. 6a), 2). PE use as a routing element only (fig. 6 b), 1)/2) combined routing of (1) and (2), 3). configure local inter-PE connect (fig. 6 c and d), 4). row bus routing (fig. 6 e), and, 5). global master bus use (fig. 6 f), The flexibility of this architecture is illustrated by an application example (8 equations in fig. 6 h) and its mapping onto a 4 by 8 array (fig. 6 j).

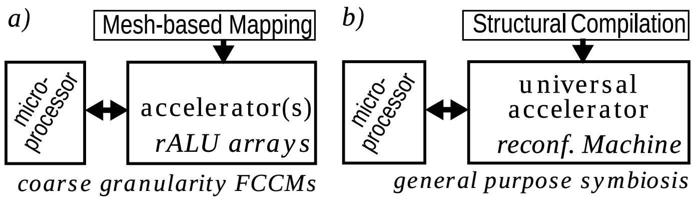


Fig. 10: Gen. purpose host/accelerator(s) symbiosis: a) coarse grain FCCM, b) accel. w. machine paradigm.

In NEWS networks like the Kress Array long distance interconnect (not to direct neighbors) needs costly traverses through one (e. g. row 2, column 6 in fig. 6 j) or several PEs. But the Kress DPSS maximizes pipelining, which minimizes the occurrence of long distance interconnect (compare fig. 6 j). Another problem are input to or output from non-peripheral PEs (inputs a0, f0, f1, g0, h0 and outputs y10 y11, y20, y30, and y40 in fig. 6 j). But in a Kress array these PEs are accessed by a fast bus network (not shown in fig. 6), to that traversing PEs is avoided. So all problems reported from NEWS networks [2] are fixed by Kress array auxiliary architectural features.

Embedding Reconfigurable Accelerators

The so-called "von Neuman" (micro)processor is no more general purpose. More and more add-on accelerators are needed: ICs and boards. The models of high end desk top computers and workstations become obsolete rapidly, not so much because of the processor MIPS, but more because their add-on accelerators become obsolete.

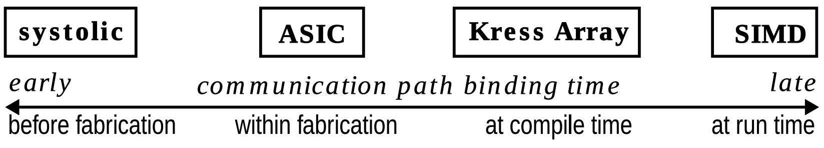


Fig. 11: Communication binding time: Kress Array vs. others.

But the processor is needed for controlling and monitoring, as a supervisor, for implementing some glue logic which does not fit onto the accelerator very well, for same last minute patches, for running commodity software available commercially, the operating system, user interfaces, and other utilities. But the standard block scheme of contemporary computing systems is determined by the host / accelerator(s) symbiosis (fig. 9 and 10), whereas stand-alone microprocessors are hardly found anymore. Since both are needed, processor and accelerator(s) we now usually have a host/accelerator symbiosis as a quasi standard scheme.

		MoM-2 Xputer architecture		VAX-11/750		
	1	2	3	4	5	6
	Algorithm	speed-up factor	data manipulation		addressing overhead	control flow overhead
1	grid-based design	2 300	99.7 %	~7 %	93 %	~1 %
2	rule check (DRC)	~15 000*				
3	electrical rule check	>300				
4	Lee routing	>160	99 %	~6 %	92 %	~3 %
5	2-D FIR filter 3x3	>300	98 %	28 %	58 %	14 %

*). versus non-optimized VAX version of DRC

Fig. 12: MoM-2 speed-up and overhead comparison.

In ASIC design and hardware/software Co-design the accelerator is hardwired (fig. 9 a). The FCCM scene [20] uses fine granularity reconfigurable accelerators (fig. 9 b). Also rALU arrays like the Kress array (fig. 10 c), and accelerators based on a machine paradigm (fig. 10 d, also see section "Automated Hardware/Software Co-Design") may be embedded by such a scheme.

Hardware/software Co-Design [21] traditionally deals with hardwired accelerators. But since hardwired vs. reconfigurable is only a minor difference, H/S co-design and F-CCM are close together and should be merged: A Generalization of Hardware/Software Co-Design.

Downloading the Accelerator

For traditional accelerator implementation, hardwired or FPGA-based (fig. 9), hardware experts are needed, so that in fact even the reconfigurable accelerator is an application-specific solution. But rALU arrays also support numeric problems by direct compilation from high level language expressions [4] fig. 6 h and j), so that much less hardware expertise is needed.

But the really universal accelerator we obtain by embedding it into a machine paradigm. So we may distinguish the following two classes of coarse granularity reconfigurable platforms (fig. 10):

- **without** a machine paradigm: synthesis is more a CAD problem (fig. 10 a: synthesis is design)
- for a reconfigurable **machine (**fig. 10 b): synthesis is a compilation problem (a programming problem)

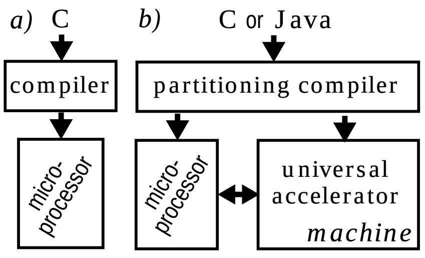


Fig. 13: Compilation method: a) traditional, b) by automatic hardware /software co-design.

Using a machine paradigm, e. g. an accelerator machine, makes compilers much easier to develop and implement, also by technology transfer from classical compilation techniques (fig. 13 a). But there is one major difference. The cocalled "von Neumann" paradigm does not support reconfigurable ALUs because of the tight coupling between instruction sequencer and data path Only non-von-Neumann paradigm is feasible), which uses data sequencers instead of an instruction sequencer [22]. This Xputer paradigm permits at least as many architectures as the Computer paradigm. An Xputer is not a data flow machine, not debuggable by tracing nor checkpointing, since being indeterministically sequenced by arbitration [22]. Xputer operation is deterministic: Xputers are step by step debuggable.

With the MoM-2 Xputer substantial speed-ups have been obtained experimentally already about 10 years ago. Some of them are in fig. 12. From the 3 orders of magnitude speed-up for the DRC algorithm (row 1) already one order of magnitude has been obtained by the data sequencer (which does not eat memory cycles for computing complex generic address sequences). Column 5 shows the very high percentage of VAX computation time elapsed for addressing overhead because of intense need for memory cycles.

Downloading accelerators is feasible: the new culture is an education problem.

Currently the 3rd generation MoM Xputer architecture is being implemented [22]. For algorithms from a number of different application areas speed-up factors around 1 and occasionally up to 4 orders of magnitude have been obtained experimentally or by estimation [23].

A number of fundamental effects contribute to these speed-up figures, like generating generic address sequences without needing memory cycles ([24] avoiding addressing overhead), software to hardware migration (mainly by loop transformations [25]), run time to compile time migration, compared to parallel computing (fig. 11), and others. For a comparison of platforms w. r. to speed-up and slow-down also see (fig. 14). For the MoM a compilation environment has been implemented [25], which in fact permits downloading of accelerators from C programming language sources.

For quite a number of high performance applications a computer with such a general purpose accelerator machine as a co-processor would permit just downloading the reconfiguration code instead of inserting a chip or board. For an upgrade you would not need a new computer or a new extension board: you just download the new accelerator.

Automated Hardware/Software Co-Design

Parallel computing scenes ignore, that "von Neumann" is no communication paradigm.

Today stand-alone microcomputers without an accelerator (fig. 13 a) are exceptions, like in trivial embedded microcontroller applications. The paper has shown, that the microprocessor/accelerator(s) symbiosis is dominant in computer applications. From this view the classical compiler is a kind of obsolete (fig. 13 a). The availability of reconfigurable universal accelerator machines (fig. 10 b, like the Xputer [22]) creates a demand for innovative compilation techniques, such as e. g. a partitioning compiler accepting input from programming language sources (fig. 13 b). An example is the CoDe-X partitioning compiler [25], which accepts C programs and automatically partitions this application into a task for the host and one for the accelerator machine.

The host/accelerator partitioning is mainly carried out by identifying loops suitable for parallelizing transformation into code downloadable to the MoM accelerator machine (MoM is an Xputer architecture). The CoDe-X implementation includes 5 different loop transformation methods: strip mining [26][27], loop fusion [26][28], loop splitting [26][29]), loop interchanging [30], and, loop unrolling [27]. Within CoDe-X these loop transformations are controlled by resource parameters for optimum adaptation to the amount of rALU array resources available, like in hardware / software co-design.

Parallel Computing versus Reconfigurable

Parallel computing scenes are in a crisis [5]. Parallel computing provides very high throughput for a class of applications featuring regularly structured data parallelism. Since there are RISC core cells available (e.g. from ARM) so small, that 16 or more (soon 32 or 64) of them would fit onto a single chip.

Parallel computing people may use their R&D results to go reconfigurable. Adopting the new culture their brain hurts.

The question is: why not have a single-chip parallel computing system? But this is not a way out of this crisis. Such a solution would cover only a niche area in massively parallel computing. But for most other parallelized applications the parallel computing scene has failed - despite of the huge human wave of researchers having spent unbelievably high amounts of research funds for decades.

One of the most difficult problems in massively parallel computing is the late binding of communication paths, which often leads to massive fine granularity switching overhead. It would not make sense to retry with similar solutions on the surface of a single chip - or a few chips. The main problems of massively parallel computing research have been:

- the illusion about the feasibility of a general purpose massively parallel architecture
- ignoring, that the so-called “von Neumann” paradigm is not a communication paradigm
- how to cope with the switching overhead at run time, coming with many applications

Many results from the parallel computing R&D scenes may be adapted to be used on with the parallelism on reconfigurable hardware platforms. Examples are loop transformations which have been modified and implemented modified for use on reconfigurable Xputer platforms [26][27][28][29][30].

accelerator platform		speed-up method ***slow down reason***	main speed-up mechanism ***main slow-down mechanism***
hardwired accelerators		software to hardware migration	loop parallelization, others ....
reconfigurable	reconfigurable accelerators (FPGAs) (tinker toy approach)	run time to compile time migration	early communication binding
		software to hardware migration	loop transformation, vectorization, pipelining
		fine granularity placement & routing	***area-inefficient and slow***
	FPAAs /w. Machine Paradigm (Xputers): additional speed-up	minimization of processor / memory communication	addressing overhead migration, avoiding contro overhead, data stream optimization
		coarse granularity	full custom cap./ high density
multiple “von Neumann” processors		concurrency	parallel processes ***run time switching overhead***

Fig. 14: Speed-up and slow-down mechanisms distinguishing accelerator platforms from each other,

Parallel Computing Conferences add sessions and/or workshops on reconfigurable computing, like RAW (Reconfigurable Architectures Workshop [32]) at IPPS (Int'l Parallel Processing Symposium [33]). An internet newsgroup *comp.arch.rpu* on reconfigurable processing units is being installed in addition to the traditional Newsgroup *comp.arch.fpga* on fine grain reconfigurable architectures and application development support tools. Reconfigurable computing gains attention of the main stream press [10]. But most researchers from parallel computing scenes still hesitate to adapt their background to the new culture: their brain hurts. This is the major reason of the software gap of the new culture [31]. The new culture is an education problem.

Reconfigurable beats parallel computing also by run time to compile time migration.

Toward a Break-through

The new development trends obviously are in favor of reconfigurable computing. A new generation of researchers is looking at coarse granularity reconfigurable architectures [2][3], driving the scene out of its fine granularity niche. The area now receives more and more popularity. As the first main stream periodical Scientific American published an article on the subject [10]. More are likely to follow.

R&D in parallel computing is in a crisis and many researchers will have to look for other new challenges. All vendors except one are out of business. But parallel computing R&D people still hesitate to look at reconfigurable platforms - the more efficient basis of parallelism. But this will probably change in the near future. Funding agencies start shifting funds from parallel computing over to reconfigurable computing (DARPA and others). So most trend indicators signal excellent chances to move reconfigurable platforms out of their R&D and market niche, mainly still determined by FPGAs and glue logic applications.

FPGAs are obsolete

Configurable Computing is a new culture. To the classical world of procedural programming for computing in time it adds the world of structural programming for computing in space (fig. 15). Systolic arrays are found at the intersection of both worlds: there synthesis methods include time and space in the same formula. The systolic array scene also provides formal mappings between both worlds.

The integration of both worlds into a duality of computing is only a matter of time and education. Know-how is available. Scientific and technological potential is available. Down-loading accelerators is feasible: the new culture is only an education problem. The only problem is knowledge diffusion. Parallel computing people may use R&D results to go reconfigurable. For the break-through we need an academic user group from many application areas. For a first attempt to gain publicity could be a Troyan horse type commercialization, where the customer does not need to know the implementation method used inside the product.

For the break-through we need an academic user group from many application areas.

Conclusions

FPGAs are obsolete (although their market is growing). The main reason is the tremendous wasting of chip area due to fine grain placement and routing — which current reconfigurable architectures have in common with currently dominant ASIC design styles.

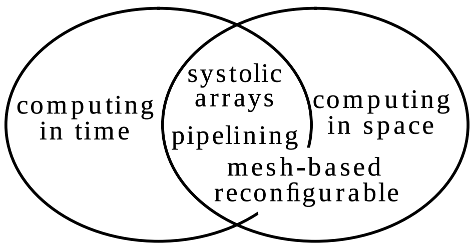


Fig. 15: The dual world of Computing.

The paper has shown a way toward substantially higher area-efficiency of reconfigurable circuits, which is needed for a new technology platform to achieve the paradigm switch to overcome the limitations of placement and routing. The paper has illustrated, that the much more area-efficient and powerful coarse granularity mesh connected scheme known from systolic arrays may be generalized to be used almost without restrictions for almost any kind of algorithm or other application. The author believes, that this approach has the potential to make dynamically reconfigurable platforms more important than the so-called "von Neumann" microprocessor.

Accelerator architectures based on a machine paradigm are useful to partially adopt traditional compilation techniques accepting high level programming language sources — also providing automatic host/accelerator partitioning. The so-called “von Neumann” paradigm does not support reconfigurable data paths. Data sequencing is the only way toward deterministic reconfigurable “machines”, which are debuggable by checkpointing.

The Microprocessor will not become obsolete: as the tail wagging the dog

The Microprocessor is no more general purpose. But the microprocessor will not become obsolete. Its role will be to support future reconfigurable computing platforms by running glue logic, software being not performance-critical, and bloatware (software which needs masses of primary memory and hard disk storage space). But with very powerful future reconfigurable accelerator platforms the microprocessor will be the tail wagging the dog.

Literature

[1] D. Manners, T. Makimoto: Living With The Chip; Chapman & Hall, 1995

[2] A. DeHon: Reconfigurable Architectures for General Purpose Computing; report no. AITR 1586, MIT AI Lab, 1996

[3] N. N.: Task Force on Reconfigurable Computing; Position Papers, HICSS’97 - report to appear in IEEE Computer.

[4] R. Kress et al.: A Datapath Synthesis System for the Reconfigurable Datapath Architecture; Asia and South Pacific Design Automation Conference, ASP-DAC'95, Makuhari, Chiba, Japan, Aug. 29 - Sept. 1, 1995

[5] R. W. Hartenstein (invited paper): High-Performance Computing: Über Szenen und Krisen; GI/ITG Workshop on Custom Computing, Schloß Dagstuhl, Germany, June 1996

[6] R. W. Hartenstein, (invited paper & opening key note): Custom Computing Machines - An Overview; Workshop on Design Methodologies for Microelectronics, Smolenice Castle, Smolenice, Slovakia, Sept. 1995

[7] R. Kress et al.: Customized Computers: a generalized survey; Proc. Workshop on Field-programmable Logic and Applications (FPL’96), Darmstadt, Germany, 1996

[8] annual international Workshops on Field-programmable Logic and Applications (FPL); see Lecture Notes on Computer Science, Springer Verlag

[9] annual IEEE Workshops on FPGAs (FPGA)

[10] J. Villasenor, W. H. Mangione-Smith: Configurable Computing; Scientific American, June 1997 - also se URL: http://www.sciam.com/0697issue/0697villasenor.html

[11] R. W. Hartenstein: Wozu noch Mikrochips?; ITpress Verlag, 1994

[12] J. Rose, S. Brown: Flexibility of Interconnection Structures for Field-Programmable Gate Arrays; IEEE J. SSC 26,3 (March 1991)

[13] S. Brown, R. Francis, J. Rose, Z. Vranesic: Field-programmable Gate Arrays; Kluwer 1992

[14] Soon Ong Seo: A High Speed Field-Programmable Gate Array Using Programmable Minitiles; Master Thesis, Univ. of Toronto. Ontario, Canada, 1994

[15] A. Agerwal, D. Lewis: Routing Architectures for Hierarchical Field-Programmable Gate Arrays; Proc. IEEE ICCD 1994

[16] Y. S. Kung: VLSI Array Processors; Prentice-Hall 1988

[17] N. Petkov: Systolische Algorithmen und Arrays; Akademie-Verlag 1989

[18] N. Petkov: Systolic Parallel Processing (Advances in Parallel Computing, Vol 5); North Holland 1993

[19] J. Becker et al.: An Embedded Accelerator for Real World Computing; Proc. IFIP Int’l Conference on Very Large Scale Integration (VLSI'97), Gramado, RGDS, Brazil, August 26-29, 1997

[20] IEEE annual Conferences on FPGA-based Custom Computing Machines (FCCM); each April, Napa, Ca, U.S.A.

[21] K. Buchenrieder: Hardware/Software Co-Design; ITpress Verlag, 1995

[22] RJ. Becker et al.: A General Approach in System Design Integrating Reconfigurable Accelerators; Proc. IEEE 1996 Int’l Conf. on Innovative Systems in Silicon (ISIS’96); Austin, Texas, USA, Oct. 9-11, 1996

[23] World Wide Web pages about Xputers and related topics; http://xputers.informatik.uni-kl.de

[24] M. Herz, et al.: A Novel Sequencer Hardware for Application Specific Computing; Proc. 11th Int'l. Conf. on Application-specific Systems, Architectures and Processors, (ASAP‘97), Zurich, Switzerland, July 14-16, 1997

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

[26] L. Lamport: The Parallel Execution of Do-Loops; C. ACM 17,2, p. 83-93, Febr. 1974

[27] D. B. Loveman: Program Improvement by Source-to-Source Transformation; J. ACM 24,1, p. 121-145, Jan. 1977

[28] W. A. Abu-Sufah, D. J. Kuck, D. H. Lawrie: On the Performance Enhancement of Paging Systems Through Program Analysis and Transformations; IEEE-Trans. C-30(5), p. 341-356, May 1981

[29] 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.

[30] J. R. Allen, K. Kennedy: “Automatic Loop Interchange”; Proc. ACM SIGPLAN’84, Symp. on Compiler Construction, Montreal, Canada, SIGPLAN Notices 19, 6, p. 233-246, June 1984

[31] R. Kress (invited paper): The Software Gap; in: R. W. Hartenstein, V. Prasanna (Eds.): Reconfigurable Architectures; ITpress Verlag, 1997

[32] R. Hartenstein, V. Prasanna (Eds.): Reconfigurable Architectures; ITpress Verlag, 1997

[33] V. Prasanna et al. (Eds.): Proc. Int'l Parallel Processing Symposium 1997 (IPPS-97), Geneva, Switzerland, April 1997