A Novel ASIC Design Approach Based on a New Machine Paradigm

R. W. Hartenstein, A. G. Hirschbiel, M. Riedmüller, K. Schmidt, M. Weber
Universitaet Kaiserslautern, F.B. Informatik, Bau 12,
Postfach 3049, D - 6750 Kaiserslautern, F. R. G.
phone: (+49-631) 205-2606 *or:* (+49-7251) 3575
fax: (+49-631) 205-3200, *uucp net mail:* abakus@informatik.uni-KL.de

Abstract. This paper introduces a new design methodology for rapid implementation of cheap high performance ASICs. The method described here derives from high level algorithm specifications or from high level source programs not only the target hardware, but - in contrast to silicon compilers - at the same time also the machine code to run it. The new method is based on a novel sequential machine paradigm where execution is used (being by orders of magnitude more efficient) instead of simulation and where programmers may do the design job, rather than real hardware designers. The paper illustrates that for a very large class of commercially important algorithms (DSP, graphics, image processing and many others) this paradigm is by orders of magnitude more efficient than the von Neumann paradigm. Compared to von-Neumann-based implementations acceleration factors of up to more than 2000 have been obtained experimentally. The performance of ASICs obtained by this new methodology mostly is competitive to ASICs designs obtained on the much slower and much more expensive "traditional" way. As a byproduct the new methodology also supports the automatic generation of universal accelerators for co-processor use in work stations etc., such as e. g. to accelerate EDA tools. It is the goal of this paper to explain the highly efficient application of the xputer paradigm, rather than to introduce its hardware implementation. It is the goal of this paper to illustrate the innovative power of this paradigm, and its potential for a major step of progress toward systematically deriving ASIC designs from algorithm specifications.

1. Introduction

For quite a number of commercially important applications extremely high throughput (up to several kiloMIPS) is needed at very low hardware cost. Masses of papers have been published also more recently on the "traditional" VLSI and ASIC design process and its expense and its slowness, so that there is no need to continue this discussion here in this paper, which advocates a completely different approach. For at least another decade this mostly will also be impossible with universally programmable von-Neumann-type computers, even with parallel or concurrent computers systems. The paper presents new approach the goal of which it is to replace hardware design more by a rapid turn-around implementation and debugging technique similar to that known from programming. Because this new approach is mainly based on a novel sequential machine paradigm (xputer machine principles) and also has some loose relations to high level synthesis and systolic array synthesis, this paper does not discuss its differences to traditional VLSI design or ASIC design. Instead, mainly with respect to performance issues, the new methodology is compared to von Neumann computers and parallel computer systems, as well as to application-specific

array processors (ASAPs), such as e. g. systolic arrays.

This paper introduces a new design methodology for rapid implementation of cheap high performance ASICs. The method described here derives from high level algorithm specifications or from high level source programs not only the target hardware, but - in contrast to silicon compilers - at the same time also the machine code to run it. The new method is based on a novel sequential machine paradigm having been published elsewhere [1 - 4]. But within the context of hardware design environments it is a new achievement, where execution is used (being by orders of magnitude more efficient) instead of simulation. All the time before this has been achieved, the (highly inefficient) simulation has been the only practicable way to verify a design. By using a machine paradigm it has become possible, that programmers may do the design job, rather than real hardware designers. The paper illustrates that for a very large class of commercially important algorithms (DSP, graphics, image processing and many others) this paradigm is by orders of magnitude more efficient than the von Neumann paradigm. Compared to von-Neumann-based implementations acceleration factors of up to more than 2000 have been obtained experimentally. The performance of ASICs obtained by this new methodology mostly is competitive to ASICs designs obtained on the much slower and much more expensive "traditional" way. As a byproduct the new methodology also supports the automatic generation of universal accelerators for co-processor use in work stations etc., such as e. g. to accelerate EDA tools.

The introduction discusses the significance of xputers, compared to that of (von Neumann) computers, and shows, why xputers offer a fundamentally new and highly efficient approach to systematically derive customized high performance silicon from algorithm specifications. The second section introduces the hardware operation principles of xputers, which in section no. 3 will be illustrated by several introductory examples of algorithms running on xputers. The fourth section shows why xputers are so much more efficient than computers - why (although being sequential machines) xputers are competitive to most other ASIC solutions. Section no. 5 proposes new directions in R&D on ASIC design methods.

Drawbacks of (von Neumann) computers. Performance problems with vector machines and parallel computer systems - compared to xputers - have been discussed elsewhere [Mch]. Their sustained average performance is by orders of magnitude lower, than the peak rate. The reason is, that communication mechanisms offered by this hardware are not sufficiently powerful and/or too inflexible: the hardware is compiler-hostile, since most of the dense data dependencies of parallel algorithms cannot be mapped onto it. With massive efforts for more than a decade the parallel computing scene achieved world-wide only disappointing results: parallelism here still is more a question, rather than an answer. Probably this scene has headed for the wrong goal. Very high performance (for the enduser) by cheaper software on cheaper universal hardware is a better goal than just a high degree parallelism by multiple von Neumann machine use. We have experienced that the improvement of basic processor principles is a much more promising goal, which can be reached by a new technology platform from computational devices other than (von Neumann) computers.Also data flow machines are optimizer-hostile and their throughput is affected by a number of bottlenecks: enormous addressing overhead, data accessing conflicts and other problems. Even ASAPs, being useful only for systolizable algorithms, have substantial draw-backs: extensive I/O overhead caused by scrambling and unscrambling of data streams.

Very High Acceleration Factors. The approach presented here is much more efficient by avoiding most of such problems. In contrast to the von Neumann paradigm this paradigm accepts a much wider variety of optimization methods, since being supported by a number of innovative hardwired machine fea-

tures, such as: auto-sequencing data memory (minimizing overhead), auto-sequencing register file organization (*scan cache*), reconfigurable rALU (featuring ultra micro parallelism), innovative memory interface, minimized access data trace by ultra micro scheduling. It is also much more efficient and acceleration factors are so high, that also being based on a sequential paradigm its performance is competitive to most other ASIC solutions.

A New Technology Platform. Von Neumann principles are based on sequential code, having been laid down in a RAM (memory) and being scanned by an instruction sequencer. That's why the RAM (random access memory) is the central technology platform of computers already for several decades. The same RAM is used for both, data and program, so that compiler and object code may run on the same hardware. This has been important because of high hardware cost in the past. Such RAM use is also a source of the elegance and the universality of the von Neumann machine paradigm: most of the control structure (instructions) is pushed into the RAM. Only a simple instruction sequencer - a small fraction of the "controller" - remains hardwired (also see fig. 1). This is very important since von Neumann control code usually is massively complex.

Due to the RAM-based machine code needed to run it, the von Neumann machine is highly overhead-prone. It also has a number of "von Neumann bottlenecks" (also see next section), such as the ALU bottleneck: the ALU is completely hardwired (also see fig. 1) such, that only one operator can be executed at a time (no parallelism). Another commercially available technology platform has been tried out with the goal to obtain a more efficient machine paradigm [1 - 5]: *interconnect-reprogrammable media (irM),* which are programmable by combinational code (e.g. see [6, 7]. Until recently such media have mainly been used for "microminiaturized breadboarding" for prototyping relatively simple hardware. In the 80ies irM have become available, which also support highly innovative irM-based methodologies by capabilities like e. g. *retargetting* capability [8 - 10]: due to *code compatibility* irM personalization code can easily be translated into that of real gate arrays (being faster and of higher integration density).

Innovative Applications of interconnect-reprogrammable Media. That's one of the reasons, why in the late 80ies some researchers started to explore the innovative potential of this technology platform. For instance, acceleration media environments have been developed: such as the PAM (programmable active memory) method [11], the hardware subroutine idea [12], and e. g. the ASIC emulator [13, 14], available commercially from Quickturn, Inc. To replace simulators these Quickturn tools derive accelerators from VLSI design netlists. The hardware of the MoM (Map-oriented Machine), a forerunner of xputers, has been the first implementation of a irM-based machine paradigm [1, 15, 16]. This data-procedural paradigm is dual to the control-procedural von Neumann paradigm.the MoM-3 xputer architecture has been developed with the goal to provide a machine paradigm, being as simple, elegant and universal as the von Neumann paradigm - but much more efficient [32]. The complexity of a relatively large *task sequencer* (MoM-1 and MoM-2 [1, 3, 15]) have been pushed from the hardwired part into irM - the central medium.

For xputers irM are (instead of the RAM) the central technology platform (also see fig. 1): source of simplicity, universality, and elegance of hardware principles. This platform provides a reconfigurable ALU (called *rALU*), which is the basis of problem-oriented ultra micro parallelism within compound operators (very low level parallelism at functional level or gate level) [32, 33]. That's why xputer machine code is primarily non-sequential, so that because of the lack of control code a new method of sequencing has to be found: data sequencing (for more details see next sections). But still a little bit of control is needed,

which is called *sparse control*, or, *residual control* [4]. Because of the dominance of data sequencing only very simple controllers are needed, which also can be pushed from hardwired parts (completely, this time) into the irM part (also see fig. 1). That's why - in contrast to computers - the RAM is mainly used for data only (thus no more being also part of the "controller"). The only hardwired part (except external interfaces) by xputer principles is the data sequencer, a part which is not found in **com**puters (also see fig. 1).

A Shortcut on the Way to Customized Silicon. In contrast to the RAM, commercially available versions of interconnect-reprogrammable media just recently obtained properties needed in use for a universal machine paradigm: such as being electrically (re)alterable quickly, incrementally (re)programmable, and having sufficiently high integration density (1990: 20,000 gates/chip, announced: 40,000) and switching speed. Already now (1990) interconnect-reprogrammable media market is a (worldwide) billion US-dollar niche of the integrated circuits market, including >12 vendors from Europe, Japan, Korea and USA. A very interesting property of FPGAs having been made commercially available recently is the ***retargetting*** capability: for particular FPGAs (e.g. Plessey) there are also "real" Gate Arrays (of much higher integration density), which are ***code-compatible*** to these FPGAs. Due to such a technology platform xputers offer a drastically shortened path from algorithm to silicon: the machine code obtained by programming an xputer may be submitted for gate array fabrication. No expensive hardware design process is needed. Xputer feature sufficient throughput to be competitive to (traditional) ASIC design. Since "real" gate arrays have a much higher integration densityn of chip count is obtained by retargetting.

The Efficiency of Xputers. Code to be generated by xpilers is not sequential because of the use of field-programmable media. That's why xputer operation principles are data-procedural (in contrast to the control-procedural von Neumann hardware) in using a data sequencer, which mainly is hardwired (fig. 1). This xputer paradigm is much less overhead-prone. It combines several measures to improve performance, which cannot be provided on a von Neumann basis: (intra-ALU) low level parallelism, avoiding several kinds of overhead (addressing overhead, control overhead, synchronization overhead, thus substantially reducing code size). This and smart register file (scan cache [1, 3]) and smart memory organization support rich optimization strategies (by compiler) to substantially reduce memory traffic. All this contributes, so that xputers offer drastically more performance with less hardware. With respect to performance most algorithm implementations on xputers are competitive to (traditional) ASIC solutions.

Optimizing Xpilers: a new kind of Silicon Compilers? The programmable part of xputers is concentrated within the ***rALU*** (a reconfigurable ALU permitting internal parallelism), which does not provide a hardwired instruction set. The translator (called *xpiler*) compiles problem-oriented *compound operators* by combinational switching networks into the rALU [3, 4, 18]. Utilizing this rALU softness only exactly those resources are generated, which are needed for a given problem: only the number and size (path width) needed. For very high throughput requirements, however, more resources will be created to achieve sufficiently high parallelism (using a larger rALU if needed). Due to the unusual degree of softness of the rALU the variety of constructs to be generated is substantially larger than for traditional compilers. This is a chance for much more efficient optimization strategies, but also a challenge to the compiler implementer: a new direction of R&D. The new central technology platform raises the question: are Xpilers a new kind of silicon compilers? The answer is "no": Xpilers are much more than just silicon compilers since generating the target hardware **and** the machine code at the same time. Silicon compilers, however, only generate a target hardware.

Technology Transfer Issues. In our opinion it would not make sense to compete with the von Neu-

mann computer in general. There is no chance because of immense infrastructures having grown in decades and commercially available: software (operating system, user interfaces, languages, application programs etc.) and hardware, communication media etc. Because of the data-driven paradigm there is practically no software for xputers available. Because of cheap hardware it also does not make sense to run application and development tools on the same hardware. So to-day it is wise to run only those performance-critical cores of applications on an xputer, where its better performance is effective and really needed.

Xputer Application Environments. For such a strategy mainly two classes of xputer application environments can be identified. In one class a xputer-based programmable universal accelerator serves as a co-processor, where the von-Neumann-type host runs the xpiler, the operating system and all other software and thus provides the universality, flexibility, portability and other general features. In case of high performance applications embedded in cheap mass products, a xputer should be frozen into customized silicon such, that it runs in stand-alone mode to minimize the chip count. Xpilers could be implemented, which accept versions of widely accepted high level languages, such as C, Pascal, OCCAM, etc., to minimize the training effort needed to introduce xputers in both classes of application environments. Also VHDL may be a candidate, since because of its behavioral part it may be used to formulate algorithms.

Relations to High Level Synthesis. Relations between the xputer method and high level synthesis have not yet been thoroughly investigated. But some differences and some common grounds are obvious. Both use finite state machine synthesis as an ingredient: on the way down from program (or micro program) high level synthesis uses a control-driven approach, whereas xputers use a data-procedural approach. That's why we believe, that it would be relatively easy to develop an alternative high level synthesis method, which is dual to "traditional" high level synthesis. Major differences are to be expected in resource allocation and scheduling. Since being data-procedural and thus being based on data dependencies, the xputer method makes use of projection methods similar to thus known from synthesis of regular processing arrays (e. g. systolic array synthesis). Such methods accept high level algorithm specifications which are of higher level than the usual sources to convention al high level synthesis. At least we believe, that the development of an alternative high level synthesis (by adoption of concepts from xputers) is a challenging and promising new R&D area.

2. Xputer Machine Organization

It is the goal of this paper to explain the highly efficient application of the xputer paradigm to ASIC design, rather than to introduce its hardware machine implementation. From this point of view the reader probably will fully understand xputer machine principles only after also reading section no. 3, which illustrates hardware operations by means of simple algorithm examples. This section shows the xputer machine paradigm from three points of view. It shows operations from the view point of a programmer, it illustrates the task of the optimizing xpiler, and, it demonstrates the principles of basic hardware operations. For clarification xputers are compared to **com**puters.

Fig. 2 a illustrates computer hardware principles and their bottlenecks. The *ALU bottleneck:* the ALU is a very narrow bandwidth device: it can carry out only a single simple operation at a time. The *memory bottleneck:* the word-by-word communication channel to the RAM memory has a very narrow bandwidth. *Control flow overhead:* In computers control flow is the primary activator: the instruction counter is the

control state register. The rate of control flow is quite high: for each single data manipulation action at least one preceding control action is needed, which requires at least one memory cycle each. If no emit address nor emit data is used, additional control flow and even data operations are needed for address computation. *Addressing overhead:* a very large percentage of memory cycles is needed to carry out address computations: in some applications values of more than 90% have been found [32].

Fig. 2 b illustrates the (non-von-Neumann) hardware principles of xputers. Also see fig. 3 which compares basic operation principles to those of **com**puters and data flow machines. Instead of an instruction counter (within an instruction sequencer) there is a data counter (within a data sequencer), which substantially reduces control overhead and addressing overhead (for more details see below). The data counter is part of a hardwired address generator (also see fig. 4 c) which provides a powerful repertory of generic address sequences [4]. Instead of a hardwired ALU (fig. 2 a) there is a reconfigurable *rALU* (fig. 2 b), which avoids the ALU bottleneck by permitting intra-rALU parallelism. A smart register file called *scan caches* and a smart memory interface support an optimizing translator called xpiler to reduce memory traffic (for more details see next paragraphs and next section).

Avoiding the ALU Bottleneck. Xputers, however, use a FPM-based *r-ALU* ([3] fig. 4 a), being reconfigurable such, that several highly parallel data paths may be arranged into powerful *compound operators,* which need only a few nanoseconds per execution, due to highly parallel dedicated intra-chip interconnect between smart register files (called *scan caches,* see below) and r-ALU (fig. 2 b and 4 a). This *intra-rALU parallelism* avoids the ALU bottleneck. Usually the r-ALU is configured only during loading, not at run time, so that PLD set-up slowness does not affect performance: dedicated wires are fast and avoid buses' multiplexing overhead [19]. Although 2 ns gate delay FPM ave available commercially, FPM might be slower than traditional ALU technologies. This is more than compensated by its micro parallelism and other xputer features. Soft instruction set computers are not new: well known are dynamically microprogrammable architectures, where, however, flexibility is based on sequential programs stored in RAMs. Also such an inner von Neumann machine suffers from the above von Neumann bottlenecks.

Avoiding Control Overhead. Xputer operations are not control-procedural (like computer), but data-procedural. This paradigm is called *data sequencing* which means, that a smart register file (scan cache) steps a small data window through primary memory space along a prescheduled path called *scan pattern*. Each such step automatically evokes hardwired *auto-xfer* and *auto apply* actions such, that prescheduled compound operators and a prescheduled cache-to-memory traffic pattern are carried out without needing any memory cycles for control. Control cycles are needed only for switching between different schedules. That's why an xputer only needs *sparse control (residual control)*, which is called upon request (for more details see next section).

Avoiding Addressing Overhead. Driven by a auto-sequencing data memory xputers are deterministically data-driven (fig. 3). For auto-sequencing the data memory interface includes a data sequencer, a hardwired data address generator (fig. 2 b, instead of computers' instruction sequencer: fig. 2 a). This hardwired data sequencer provides a repertory of generic data address sequences without any addressing overhead. Such an address sequence makes a scan cache move through data memory space, step by step, scanning a predefined segment of primary memory space along a path, which we call a *scan pattern*.

Reducing Memory Traffic. Of course intra-rALU parallelism and avoiding control overhead and addressing overhead already substantially reduce memory traffic. But there are extra hardware features to

save even more memory cycles. In cooperation with an optimizing translator (called *xpiler*) the *smart register file* (called *scan cache*) and a smart interface to primary memory support strategies to arrange *auto-apply* and *auto-xfer* operations for minimal schedules. Access mode tags minimize the number of memory semi cycles needed in cache/memory traffic. The selection of optimum combinations of data storage maps and scan patterns achieves optimized sweeps through data such, that the right data are found at the right time at the right location. Section no. 3 illustrates this by simple algorithm execution examples.

Xputers versus Data Flow Machines. Because xputers represent a data-procedural paradigm, we should briefly compare it to the other well known data-driven machine paradigm: the principles of data flow machines (fig. 3). There is a fundamental difference between both such, that xputer principles are relatively closer to von Neumann machine principles, than to the principles of data flow machines. There is even a duality between xputer principles and computer principles: both operate deterministically. Computers are deterministically driven by control flow, from where data accessing is evoked (when needed). In xputers the order of causality is reverse to that: xputers are driven by data accessing, from where control is evoked (upon request, i. e. only when needed). Also data flow machines are data-driven, however, not deterministically (like xputers), but by "firing", which is a kind of arbitration process at run time. Xputers support compilation techniques, where as many as possible decisions are made up at compile time, where most overhead is pushed over from precious run time to compile time. This is a good strategy for optimum ASIC synthesis. The principles of data flow machines, however, require a much larger effort on the run time side, which would a bad strategy for optimum synthesis of cheap ASICs. For more detailed comparison see [5]

2.1 The MoM (Map-oriented Machine)

Let's illustrate the role of this data sequencer by a xputer architecture example. The MoM (Map-oriented Machine) having been implemented at Kaiserslautern [1 - 3, 16] features several peculiarities: 2-dimensional data address space (fig. 3 b, c), and a special 2-D optimizer-friendly register file organization. The MoM has 4 such register files which we call *scan windows* or *scan caches* (fig. 4 a), because each such cache operates like a size-adjustable window on the data memory (fig. 4 b). Due to its 2-dimensional memory organization the MoM very well also supports image processing, signal processing and also VLSI layout processing.

Figures 3 c shows a linear scan pattern example (step width / direction vector $\Delta x, \Delta y$): an iterative relative jump). Fig. 3 d shows a *video scan* pattern, e. g. for 2-D filtering or design rule check etc.. Other scan pattern examples are: slanted video scan (fig. 4 e), spiral scan (fig. 4 f), reflect, shift, shuffle (fig. 4 g), butterfly etc. Also special scan patterns to emulate systolic arrays, as well as data-dependent scan patterns (fig. 4 h, e. g. for image preprocessing, design rule check etc.), are available in hardwired form, i. e. free of any overhead. Because of a hardwired *local branching shortcut* the address registers within the data sequencer are directly manipulated by decision data from the rALU, so that no memory cycles are needed nor actions of the sparse controller [3, 16].

Looking back at computers: their control flow has only a single "*scan pattern*" scanning instructions one by one (as long as no branch nor jump is encountered, which we consider to be an ***escape*** from the scan). In contrast to those of xputers this scan pattern is not free of overhead: each step requires its own instruction fetch. Each instruction fetch requires an extra memory access cycle. This especially makes iterative

operations inefficient, since the same instruction is fetched again and again. Loops on a computer cause additional *control overhead* for return jumps and subscript watching and thus cause additional memory access cycles. From this point of view it is obvious, that the computer paradigm is extremely overhead-prone, whereas the xputer paradigm strongly tends to avoid most kinds of overhead.

3. Operation Illustration by Application Examples

The reader probably will fully understand section no. 2 on xputer machine principles only after also reading this section no. 3, which illustrates hardware operations by means of simple algorithm examples. This section shows the xputer machine paradigm from three points of view. It shows operations from the view point of a programmer, it illustrates the task of the optimizing xpiler, and, it demonstrates the principles of basic hardware operations.

For high level programming of xputers we use a simple model which is supported by the *auto-sequencing* data memory, and, which we call *data sequencing*. This paradigm and also the xpiler's tasks will be illustrated here by 2 simple algorithm examples ('xpiler' stands for the xputer's compiler). The first example (a systolic algorithm: fig. 5) is not a good one to demonstrate the merits of xputers over vector machines. But it has been selected for easy illustration of the data sequencing paradigm.

Fig. 5 a shows it textually and fig 5 b its *signal flow graph (SFG)*. From this SFG the xpiler derives (fig. 5 c): an optimum *data map* (fig. 5 d), from this map a *scan cache format* (left of fig. 6 a) and r-ALU *subnet spec* including wiring (fig. 6 a, derived from a single iteration: fig. 5 b), and finally a *scan pattern* (arrows in fig. 6 b). At each step of a scan via ***auto-apply*** and ***auto-xfer*** the scan cache automatically (i.e. without needing a controller action) communicates with the r-ALU subnet currently selected and with primary (data) memory. Note, that by access mode tags only a minimum of memory semi cycles is carried out: read-only tags for all 4 words of this example (fig. 6). In our example 8 steps (x width = 1, y width = 0) are carried out (fig. 6 b shows initial and final cache locations). A register having been configured into the rALU subnet (fig. 6) avoids the additional memory cycles which would have been needed otherwise to save the carry (c[i+1]) and to restore it (c[i]) between iterations of the loop.

Fine Granularity Scheduling. This first example has illustrated the task of the innovative kind of compilers needed for xputer [3, 18]: a kind of fine granularity scheduling (or: *ultra micro scheduling*) of caches and rALU subnets, and, of data words, ready to be auto-sequenced. This is fundamentally different from sequentially piling up sequential code like conventional compilers do it for computers. Later in a section on xputer high performance features a more detailed impression on this scheduling task will be given.

Organization of Residual Control. At the end of the above data sequence example the cache finds a *tagged control word* (TCW) which then is decoded (see right side in fig. 6 b) to change the state of the *residual control logic* (fig.4 a) to select further actions of the xputer. This sparse TCW insertion into data maps we call *sparse control*. Note, that the control state changes only after many data operations (driven by the data sequencer). That's why usually only a very small (residual) controller is needed. That's why we use the term *residual control* or *sparse control* for this philosophy. Note, that xputer operation is data-driven so that TCWs may be encountered only from within a data sequence. The residual controller and the TCW decoder are defined at compile time (by a xpiler) and configured as a subnet within the r-ALU. Only very little rALU space is needed, so that the elegance from pushing control structures into the central

technology platform is cheap (compare section no. 1).

Branching within Data Sequencing. During a scan there is **no** control action: the data counter is not a state register. But to achieve xputer universality the MoM-3 features the following *escapes* for branching out of a scan: *normal escape* (by *end of scan flag* from data sequencer), *delimiter escape* (on TCW encounter), *off-limits escape* (address exceeds memory segment limits), *conditional branch escape* (by decision data from r-ALU), and, *event escape* (by external event flag). Upon off-limits escape, branch escape, or event escape a *remote control word* (RCW) or *remote address word* (RAW) is fetched from a remote memory segment via an *escape cache*. A second decision mechanism (*implicit branching,* because residual control state is not affected because of local branching shortcut by feeding decision flags directly into the address generator) is activated only **within** *data-dependent scans* (i. e. without escape: curve following etc. [3, 16]). Such a data-dependent scan pattern may be left by conditional branch escape or off-limits escape.

To achieve xputer universality also non-generic scans and individual data accessing are needed, implemented by list-directed scan: next data address is read from a TAW (tagged address word) within the data map or from a RAW (in case of an escape). This list mode can be entered directly during a scan upon TAW encounter. If no TCW is found, a TAW does not activate residual control. Reading addresses from primary memory means addressing overhead, so that list-driven sequencing is slower than hardwired scan patterns. But also in this mode of operation the xputer paradigm is still superior to the computer paradigm.

I/O Data Sequencing and Real-time Processing. Xputer I/O is simple: the scan-cache-based data sequencing hardware may be linked to an I/O channel, which is more powerful arrangement than DMA known from computers. The data streaming in are not necessarily just downloaded into a linear array memory segment. Via a suitable scan pattern selection, along with proper scan cache adjustment, the data sequencer may also set up a structured data map already directly during input operation, so that re-shuffling later on will not be needed. Also during output data may be picked (by the data sequencer) from memory in a structured way. Since I/O anyway is somehow real-time-oriented, a similar scheme (also involving the rALU) may be also used for embedding xputers into real-time applications.

Highly Flexible Cost/Performance Ratio. Xputers do not have a hardwired instruction set, so that there is no need for a fixed data word size. That's why xputer word lengths are compiler-defined: data path, cache, and control words. Thus extensible xputer architectures are feasible, upgradable by inserting more irM chips into free r-ALU sockets and more boards into free memory slots. E. g. it is easy to design a VWL memory (Variable Word Length Memory), where data word length could be changed under software control to support VLDW (very large data word) strategies for more parallelism [4, 17]: instead of only a single iteration of the DG (fig. 5 b) this time the r-ALU subnet spec is derived from n iterations (similar to fig. 7 using 4 iterations). (Such an iterative compound operator we call a *super compound operator*.) The new super compound operator is n times as powerful (4 times as powerful i fig. 7). Its use requires a vertical cache format with a VLDW including many subwords for many operands. In our example with (n=4) 16 subwords would be needed for pure VLDW strategy. The scan pattern would be very short, so that the 1-by-1 cache visits only 2 locations with a total speed-up by about a factor of 16. This illustrates the extremely high flexibility of the xputer paradigm with respect to cost/performance trade-off. We have obtained this speed-up (still with a single processor) by an about 4 times larger rALU, a 16 times larger scan cache, and a 16 times wider memory data path.

Compiled Pipelines. Designing xputer architectures with extremely long memory word capability

would cause rALU partitioning (probably causing a high chip count) and packaging problems (very large number of pins, which can be extremely expensive) because of the large VLDW caches needed for this. The same speed-up results can be obtained with substantially smaller word length when pipelining is used (fig. 7). For our example (n=4) we use a 4 subfield word size (instead of 16 subwords), so that - compared to the full VLDW version - only a quarter of the cache width and the memory data path width is needed. At the irM side this solution is much cheaper. A four bank interleaved memory is needed. We again have obtained a total speed-up factor of 16: a factor of 4 by pipelining, and a factor of 4 by quadruple super compound operator.

Non-systolizable Algorithms. The introductory application example in fig. 5 / 6 has been a systolic algorithm, being easy to convert into a data sequencing scheme because of the locality of data communication. In digital signal processing and in other important application areas, however, also non-systolizable algorithms are very important. In contrast to parallel computer systems and VLSI arrays, xputers smoothly accept also non-systolic data sequencing schemes.Fig 8 a/b shows data map and rALU configuration for such an algorithm (a 16 point example constant geometry FFT). Fig. 8 a shows the signal flow graph, which also includes non-local data communication. The grid overlay illustrates the ease of deriving a data map. Fig. 8 b shows, how the r-ALU subnet spec and a 3 cache configuration are easily derived from the DG: just take a single iteration (spider-shaped). Fig. 8 a also shows initial and final locations of the 3 caches being scanned in parallel.

Also here a VLDW or pipelining strategy may be used. Instead of a single spider (iteration) 4 adjacent "spiders" (see fig. 8 c) are picked from the DG. Compared to the above example this VLDW version yields a speed-up factor of about 5. Due to this flexibility and their high performance and their universality xputers may replace specialized digital signal processors. Due to this universality xputers may accelerate also any other kind of parallel algorithms. For mass production applications xputers may also be used in stand-alone mode, so that no host is needed which substantially reduces the total chip count (also see section on embedding and technology issues).

3.1 Xputer use in EDA

Due to their 2-dimensional memory organization and their highly efficient data sequencing paradigm xputers are especially well suitable for image preprocessing, so that no specialized and much more expensive image processing computers or specialized accelerators are needed. Due to its universality also other kinds of parallel algorithms may be accelerated by the same xputer, and, in mass product applications customized stand-alone xputer use substantially reduces the total chip count. In image preprocessing systolizable algorithms (mainly using only very simple scan patterns, like the video scan: fig. 4 d) and methods using data-dependent scan patterns are dominating. This section illustrates xputer use here by electronics design automation examples having been implemented at Kaiserslautern, where integrated circuit layout uses grid-based data structures being quite similar to those, well known from image preprocessing.

Grid-based Layout is no more obsolete. Apropos grid-based layout: in the late 70s and the early 80ies grid-based layout has been popular within the academic scene [20 - 24]. After having become almost obsolete later on, its popularity now is again increasing slowly. Currently here seems to be a tendency for design rules based on a 0.25 µ grid size - even for designs not intensionally grid-based. Also many routing algorithms are grid-based. There are more examples. A very important argument is the fact, that

for grid-based algorithms dramatically large acceleration factors have been obtained (fig. 11), such as, for example, a factor of >2000 for a CMOS design rule check [2, 5, 17]. For more details see next paragraphs.

Pattern Matching Applications on Xputers. We use pattern matching examples to illustrate image preprocessing capabilities of xputers, such as applicable also to integrated circuit layout verification and routing using grid-based design rules [20, 21]. A DRC may be carried out by a finite state machine [22] or combinational logic [15]. Such algorithms run very fast on ASIC hardware which, however, have to be reimplemented for changed design rules and for portation. Due to very large primary memories modern work stations also conventional software implementation is feasible which, however, is very inefficient because of sequential processing of the very large number of reference patterns. But to measure acceleration factors such implementations are needed. The MoM-DE environment with tools like a reference pattern generator and the PISA [15] package facilitate comparative performance measurement by convenient generation of such pattern matching algorithms (fig. 9 b).

In contrast to computers, here the performance of xputers is competitive to ASIC solutions. E. g. for a grid-based design rule check (DRC) the MoM xputer has been programmed such, that a single video scan over the layout is sufficient (fig. 9 a). Substantial acceleration is obtained also for other kinds of grid-based layout processing, such as Lee routing [1, 3], ERC (electrical rules check [23]), compaction [24], fault extraction [25], etc. Reference patterns are configured combinationally into the r-ALU as a single very powerful compound function linked with a video scan sequence within a 2-dimensional bit map memory segment. A single read-modify-write data loop is performed per cache location without using decision data (figure 9 a). Experimental results in grid-based DRC with 4-by-4 cache are acceleration factors of up to more than 2000 (CMOS design rules [26]).

The extremely high acceleration factor is due to mainly two reasons: all (hundreds of) reference patterns (a few examples in fig. 9 b) are bundled by a huge compound Boolean operator (massive ultra micro parallelism) and caching completely avoids addressing overhead (an analysis of the VAX version of this algorithm has shown about 93% CPU time for addressing [32]). Also MoM on-cache shift and access mode flag features (fig. 9 c) contribute to the high performance by minimized storage access time. Access mode flags ({"read-only", "write-only" or "ignore"}, see shaded areas in fig. 9 c) save memory semi cycles. During a video scan and similar fill scans shift paths within a scan cache (see fig. 9 c) are used to avoid, that variables travel several times forth and back along the memory communication channel (only for the "overflow" memory semi cycles are needed).

Lee Routing. Also the Lee routing algorithm [27 - 29] is an image preprocessing example (for acceleration factors also see fig. 11). But this time data-dependent scan patterns are dominating, such as e. g. in curve following: decision data from the r-ALU influence data sequencer operation telling to which nearest neighbor location to go next (fig. 10 b,d,f). In Lee routing the cache (size 3-by-3) first performs a spiral scan around the start cell S, propagating a wavefront around S (fig. 10 a) until the target T is found (fig. 10 c). This is an example of a data-dependent scan pattern (compare fig. 10 b). When T has been found, a (hardwired) data-driven escape is started (fig. 10 d) after switching cache size to 1-by-1 and activating another rALU subnet. Back-tracking from T (fig. 10 f) generates the wire (fig. 10 e). Also this scan pattern is exited by a data-driven escape (conditional branch, see last line in fig. 10 f). Note, that also the data-dependent scan patterns are hardwired, which prevents address computation overhead by direct r-ALU / sequencer interaction. For the Lee algorithm (without obstacles: 160 reference patterns) an acceleration factor >160 has been achieved. For the case with obstacles (where the number of reference patterns need-

ed is substantially higher) acceleration factors of about 800 have been obtained experimentally.

Xputer Hardware Features supporting Optimization by Xpiler. Since xputer data path width is not hardwired a low path width may save rALU space. This is one of the reasons of xputers' high acceptance of a wide variety of optimization strategies. Further minimizations yield from memory accessing strategies, possible with xputers only - but not with computers. On-cache shift paths (compare fig. 9 c for 4-by-4 example) minimize the number of memory access cycles needed to 1 per word (of the bit map) multiplied by the y size of the scan cache (4 in fig. 9: compared to a dumb read-modify.-write scheme this yields a speed-up factor of 4).

4. Why Xputers are so Efficient

In contrast to the parallel processing scene, the global goal of which is restricted to the coordination of many von Neumann machines, the xputer methodology profits from attacking the problem from several sides. We believe, that the primary goal would be very high performance as cheap as possible, rather than just a special kind of parallelism (von-Neumann-type). The consequence is it, also to search for better basic processor principles (better than von Neumann). The xputer shows, that this can be achieved on the new technology platform of interconnect-reprogrammable media (irM) from computational devices other than (von Neumann) computers. Also cheaper software (substantially smaller code size) and cheaper universal hardware is obtained as a byproduct. This has been possible only by the marriage between different disciplines: (procedural) computational paradigms and structural synthesis (by interconnect programming due to irM use), which almost is a synonym for hardware design, VLSI design, or ASIC synthesis.

Technology-independent Performance Evaluation. Fig. 11 summarizes acceleration factors having been obtained experimentally. Encouraging performance results (fig. 1) have been obtained experimentally on the MoM-1 xputer architecture at Kaiserslautern [1, 3] showing, that in several important applications a single xputer processor may even outperform large parallel computer systems. For a computer-to-xputer performance comparison it does not make sense to use MIPS, since an xputer does not have a hardwired "instruction set". To investigate the efficiency of machine principles we prefer the technology-independent measure of *acceleration factor* obtained experimentally (fig. 1) from two equivalent implementations of the same algorithm (fig. 12 b): one from a computer (VAX-11/750) and one from the technologically comparable MoM-1 xputer [1, 3]).

Having explained introductory sequencing examples (in previous sections) we may obtain deeper insight into xputer performance issues more easily. That's why this section is more a summary. Xputer performance stems from a number of different phenomena and concepts, part of which cannot be practised on a von Neumann processor basis. The most important roots of xputer efficiency are:

- the data sequencing paradigm per se is substantially less overhead-prone
- high hardware flexibility permits a wider variety of optimization strategies
- special xputer architecture features for performance or to support optimization

The data-procedural xputer paradigm obviously is by far less over-head-prone, than the control-procedur-

al von Neumann paradigm. This has been illustrated by the above examples. Control flow overhead is almost completely avoided. Experiments have shown, that addressing overhead is substantially reduced not only by hardwired address generator (also see the pattern matching example in next chapter). Not yet all mechanisms of overhead reduction in xputer programs are well understood: we propose basic research also covering overhead mechanisms of the von Neumann paradigm. Dedicated intra-rALU interconnect being typical for xputers avoids using buses, which are slow and cause *multiplexing overhead* [19]. (Flexibility of microprogrammable von Neumann computers is more restrictive and much less efficient: it is based on a hardwired micro instruction set and fixed micro word formats, ALU flexibility causes overhead since being bus-oriented, the inner machine is overhead-prone since being a von Neumann machine.)

Much wider varieties of optimization strategies than possible with computers can be efficiently mapped onto xputer hardware. Compound operators' parallelism due to rALU softness reduces memory access by substantially minimizing the number of stored intermediate variables. Often the r-ALU's flexible data path width facilitates better utilization of r-ALU space (e. g. trade a more powerful compound operator for less path width). Without this flexibility of the rALU resource the highly performance-relevant smart register file (scan cache) concept would not be practicable.

Smart Register File. The scan cache is another important hardware feature reducing control overhead by auto-apply and auto-xfer operation (for details see above). It supports optimization by adjustable format and its reconfigurable interconnect to the rALU. This efficiently supports optimization of fine granularity scheduling by a rich choice of highly efficient combinations of scan patterns and data maps (for more details see above). Some MoM hardware features having [1, 16, 32] further support the reduction of memory access time, such as: access mode tags, very high hit rate interleaved memory use, the MoM 2-dimensional cache with multidirectional shift paths, and others (for more details see fig 9 c or the paragraph on Pattern Matching Applications). Xputer cache use is fully deterministic (in contrast to computer caches permitting only probabilistic strategies - based on run time decisions - which yield relatively low hit rates). Due to very fine granularity data scheduling (defined at compile time) this mostly yields 100% hit rates since it provides the right data from the right location at the right time

The Xputer: a Communication Machine. Let's have a second look at xputer communication mechanisms - from a higher level point of view. Multiple scan caches may communicate with each other through a shared rALU (see application example in fig. 8). The shared rALU as a communication channel is practically free of overhead and mostly uses intra-chip connectivity. For parallel operation of several scan caches multiple address generators are coordinated under a common clock. These cooperating address generators produce scan patterns being independent of each other, so that also non-locally regular *compound scan patterns* can be generated efficiently. For a very large class of commercially important algorithms (DSP, image processing, graphics, and many other areas) this paradigm is by orders of magnitude more efficient than the von Neumann paradigm.

That's why xputers are by orders of magnitude more efficient than computers for **all** algorithms with regular data dependencies: g-algorithms ("g" stands for "generic"). Fig. 13 characterizes algorithms with respect to performance of their xputer implementation. Systolizable algorithms (only locally regular data dependencies permitted) are only a small subclass of g-algorithms. Even for completely irregular algorithms (spaghetti structures) xputers are substantially more efficient than computers. This is important for stand-alone application of xputers. That's the reason why the performance of ASICs obtained by the new methodology being introduced here mostly is competitive to ASIC designs obtained through the much

slower and much more expensive "traditional" way. Compared to von-Neumann.based implementations acceleration factors up to more then 2000 have been obtained experimentally (fig. 11).

5. New Directions in ASIC Design Tools and Methods

This section introduces *xpilers* ("compilers" for xputers), being important synthesis tools of the new ASIC design methodology proposed by this paper. Innovative optimization techniques used in xpilers are important to obtain the xputer efficiency needed for competitiveness to ASICs derived by traditional ASIC design methods. MoM-DE introduced later in this section is an example implementation of an xpiler.

Resource Optimization by Xputers' Compilers. The programmable part of xputers is concentrated within the rALU (a reconfigurable ALU permitting internal parallelism), which does not provide a hardwired instruction set. The translator compiles problem-oriented compound operators by combinational switching networks into the rALU. Utilizing this rALU softness only exactly those resources are generated, which are needed for a given problem: only the number and size (path width) needed. For very high throughput requirements, however, more resources will be created to achieve sufficiently high parallelism (having a larger rALU if needed).

Code to be generateed by xpilers is fundamentally different from code for computers: one difference is the use of interconnect-reprogrammable media. Another difference of the target hardware is it, that its operation principles are *data-procedural* (in contrast to the control-procedural von Neumann hardware). This requires a new class of (data-procedural) programming languages and a fundamentally new kind of compiler. Due to the unusual degree of softness of the rALU the variety of constructs to be generated is substantially larger than for traditional compilers. This opens up a way to much more efficient optimization strategies, but also challenges the compiler implementer: a new direction of R&D.

Application Development Support. MoM-DE (fig. 12 b) is an example implementation of an *xpiler*. MoM-DE is the MoM application development environment is running on a host (a μVAX [5]) featuring a self-explanatory syntax-driven editor for a high level language MoPL (Map-oriented Programming Language), roughly a Pascal extension (fig. 12 b). MoPL sources are accepted by the MoMpiler the "code generator" of which includes a commercial PLD programming tool needed for r-ALU personalization.

MoPL includes a sublanguage PaDL, which efficiently supports pattern matching applications in general. An optimizing reference pattern generator has been implemented [San], which accepts VLSI layout design rules [16]. Fig. 13 b shows an example: 10 reference patterns needed to detect the violation of minimum poly-to-poly separation by 2 lambda. For inclusion of other kinds of pattern matching applications an interactive graphic pattern editor has been implemented [18] for easy editing, modification, inspection and surveying of sets of reference patterns.

A Shortcut on the Way to ASIC Implementation. A very interesting property of FPGAs having been made commercially available recently is the ***retargetting*** capability: for particular FPGAs (e.g. Plessey) there are also "real" Gate Arrays (of much higher integration density), which are ***code-compatible*** to these FPGAs [8 - 10] (also see section no 1 of this paper). Due to such a technology platform xputers offer a drastically shortened path from algorithm to silicon: the machine code obtained by programming a (programmable) xputer may be submitted for gate array fabrication of an xputer frozen into customized silicon (fig. 12 a). No expensive hardware design process is needed. Xputer feature sufficient throughput

to be competitive to most (traditional) ASIC designs.

The first commercially available application of interconnect-reprogrammable media in VLSI design has been ASIC emulation from netlist sources [13, 14]: replacing simulation since being a more efficient way of ASIC verification. In contrast to xputers, however, ASIC emulation does not provide a new design paradigm: the netlist is imported: the result of a separate (conventional) hardware design process. Xputers have a programming paradigm: a very high level model of parallel algorithms. Running an implementation on an xputer is execution - but not emulation. Because of irM retargetting capabilities [8 - 10] the xputer paradigm may provide a complete ASIC design process including a new kind of ***very high level synthesis***.

Xputer Compilers: a new kind of Silicon Compilers? A rALU is made up from FPGAs raises the question: are xputers' compilers (also called Xpilers) a new kind of silicon compilers? The answer is: xpilers are much more than just silicon compilers. Xpilers generate the target hardware and the machine code at the same time. Silicon compilers, however, only generate a target hardware, but not the machine code, nor the compiler needed to generate code for this target hardware, nor the application program needed as a source for such a compiler. Another important difference is the source language level. Xpilers accept sources of much higher abstraction level, so that their degree of automation is much higher than that of silicon compilers. A silicon compiler accepts a hardware description at RT or functional level only. An xpiler, however, accepts programs written in a high level programming language, or even higher level algorithm specifications, an abstraction level even higher than that o high level programming languages.

By the first xpiler, the MoM-DE [18], the well-known techniques of time / space mapping have been retargetted from data streams and PE arrays to (static) memory data maps and scan patterns and thus have been generalized: from systolizable algorithms to g-algorithms (algorithms also with non-locally regular data dependencies). This has been made possible because the xputer paradigm eliminates the locality problem known from PE arrays (e. g. systolic arrays).

6. Conclusions

With xputers an innovative computational machine paradigm has been introduced and implemented which achieves for a very large class of parallel algorithms drastically much better performance and hardware utilization and drastically more (compiler-) optimizer-friendliness than the von Neumann paradigm. Acceleration factors up to more than 2000 have been obtained experimentally with a simple monoprocessor. For many commercially important high performance applications xputers may outperform large parallel computer systems or ASIC solutions (also see fig. 13).

Due to convenient conversion into a gate array the xputer also provides an alternative ASIC design methodology to design much cheaper hardware in substantially shorter turn-around time. Due to xputer universality also other kinds of parallel algorithms and glue framework may run on the same xputer, and, in mass product applications a stand-alone use is possible, which substantially reduces the total ASIC chip count. For xputer architectures an extremely low amount of xputer-specific hardware is needed, not being performance-critical, so that it is easy to keep up within the technology race.

An exciting new R&D scene has been opened up: promising and challenging. Not really a new theory, but

a new mix of backgrounds is needed, derived from computing, ASIC synthesis methods, algorithms and applications. Due to a marriage between ASIC design and computational paradigms xpilers have opened up a new methodology of systematic ASIC synthesis from algorithm specifications, which substantially simplifies the design process. In contrast to silicon compilers, xpiler generate both at the same time: target hardware and their machine code. The new paradigm is as universal as the von Neumann paradigm, so that it also provides a methodology of universal accelerators.

7. Acknowledgements

Early versions of MoM concepts have been developed within the multi university E.I.S. project, having been jointly funded by the German Federal Ministry of Technology, and by the Siemens-AG, Munich, Germany, coordinated by the GMD Schloß Birlinghoven. We also acknowledge the good cooperation with Elfriede Abel, Herwig Heckl, Gustl Kaesser from GMD and Klaus Woelcken (now with the Commission of the European Communities). We also appreciate valuable ideas from Klaus Singer of **ELTEC** GmbH at Mainz, Germany. Last but not least we appreciate the contributions of more than 30 students.

8. Literature

[1] R. W. Hartenstein, A. G. Hirschbiel, M. Weber: MoM - Map Oriented Machine, in: Chiricozzi, D'Amico: Parallel Processing and Applications, North Holland, Amsterdam / New York 1988.

[2] A. Ast, et al.: Using Xputers as Inexpensive Universal Accelerators in Digital Signal Processing; Int'l Conf. on New Trends in Signal Processing, Communication and Control, Ankara, Turkey, July 1990, North Holland 1990

[3] R. Hartenstein, A. Hirschbiel, M. Weber: MoM - a partly custom-designed architecture compared to standard hardware; Proc. IEEE Comp Euro '89, Hamburg, FRG, IEEE Press, 1989

[4] R. Hartenstein, A. Hirschbiel, M. Riedmüller, K. Schmidt, M. Weber: A Novel Paradigm of Parallel Computation and Its Use to implement Simple High-Performance Hardware; Proc. InfoJapan '90 - International Conference on Information Technology, commemorating the 30th Anniversary of the Information Processing Society of Japan (ISPJ), Oct 1 - 5, 1990, Tokyo, Japan,

[5] R. Hartenstein et al.: Xputers: an Open Family of non-von-Neumann Architectures; Proc. GI/ITG Conf. on the Architecture of Computing Systems, Munich, 1990; VDE-Verlag Berlin 1990

[6] M. Bolton: Digital Systems Design with Programmable Logic; Addison-Wesley, 1990

[7] K. Schmidt: Programmierbare Logik-Bausteine, Architekturen und ihre Programmierung; Hand-out,m Proseminar SS 1990, Fachbereich Informatik, Universität Kaiserslautern, 1990

[8] N.N. (Plessey): ERA Product Overview; Plessey Semiconductors, Publication no. P2350, October 1989

[9] N.N. (Plessey): QuickGate (Product Overview); Plessey Semiconductors, Publication no. P2425, May 1990

[10] N.N. (Plessey): ERA 60100 (Product Data); Plessey Semiconductors, Publication no. P2321, April 1990

[11] P. Bertin, D. Roncin, J. Vuillemein: Introduction to Programmable Active Memories; Proc. 3rd Int'l

Conf. on Systolic Arrays, Kilarney, Ireland, May 1989

[12] N.N. (Plessey): Hardwired Subroutines; report, Plessey Semiconductors

[13] M. D'Amour, et al.: ASIC Emulation cuts Design Risk; High Performance Systems, Oct. 1989

[14] P. A. Kaufman: Wanted: Tools for Validation, Iteration; Computer Design, Dec. 1989

[15] R. W. Hartenstein, R. Hauck, A. G. Hirschbiel, W. Nebel, M. Weber: PISA - A CAD package and special hardware for pixel oriented layout analysis, Proc. ICCAD 1984, Santa Clara 1984.

[16] R. W. Hartenstein, A. G. Hirschbiel, M. Weber: MoM - Map Oriented Machine; in: Ambler et al.: (Prepr. Int'l Worksh. on) Hardware Accelerators, Oxford 1987, Adam Hilger, Bristol 1988.

[17] R. W. Hartenstein, A. G. Hirschbiel, M. Weber: The Machine Paradigm of Xputers: and its Application to Digital Signal Processing Acceleration; ICPP-90 Int'l Conf. on Parallel Processing, 1990; IEEE Press, Wash., D.C., 1990

[18] M. Weber: (Ph. D. thesis), Univ. Kaiserslautern, October 1990

[19] R.W.Hartenstein, G.Koch: The Universal Bus Considered Harmful; in: R.Hartenstein, R.Zaks: The Microarchitecture of Computing Systems; North Holland, Amsterdam/New York 1975;

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

[21] R. F. Lyon: Simplified Design Rules for VLSI Layout; Lambda, 1st quarter 1981

[22] R. Eustace, A. Mukopadhyay: Deterministic Finite Automaton Approach to Design Rule Check for VLSI; Proc. DAC 1982

[23] C. M. Baker, C. J. Terman: A Tool for Verifying Integrated Circuit Designs; Lambda 1st Qu. 1980

[24] D. Boyer, N.Weste: Virtual Grid Compaction using the most recent Layer Algorithms; ICCAD 1983

[25] I. Stamelos et al.: A Multi-Level Test Pattern Generation and Validation Environment; International Test Conference 1986, IEEE Press 1986

[26] G. Zimmer: Lambda Designregeln für das EIS-Projekt; report; IMS Duisburg, F.R.G., 1986.

[27] C. Y. Lee: An Algorithm For Path Connections And Its Applications. IEEE TrC-10 (Sept. 1961)

[28] M. A. Breuer and K. Shamsa: A Hardware Router. In: Journal of Digital Systems, 4, 4, 1981.

[29] I.Velten: Implementierung des Lee-Algorithmus auf der MoM, Dipl.Thesis, Univ..Kaiserslautern, 1987

[30] R. W. Hartenstein, A. G. Hirschbiel, M. Weber: Mapping Systolic Arrays onto the Map-Oriented Machine (MoM), Proc. 3rd Int'l Conf. on Systolic Arrays, Kilarney, Ireland, May 1989.

[31] R. Hartenstein, K. Lemmert, SYS3 - A CHDL-Based CAD System for the Synthesis of Systolic Architectures, Proceedings IFIP CHDL '89, North Holland, Amsterdam / New York 1989.

[32] A. G. Hirschbiel: (Ph. D. thesis), Univ. Kaiserslautern, 1990

[33] T. Mayer: POLU (Problem Oriented Logic Unit), Diplomarbeit, Universität Kaiserslautern, 1989.

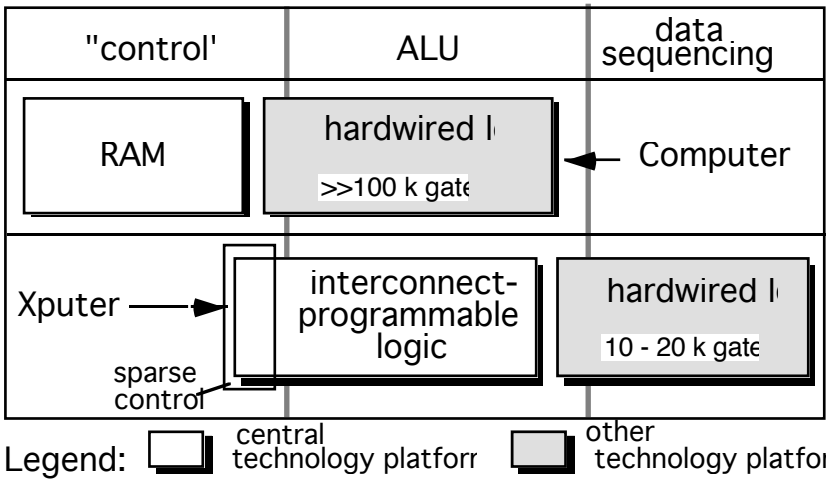


Fig. 1. Central technology platform: computers versus xputers.

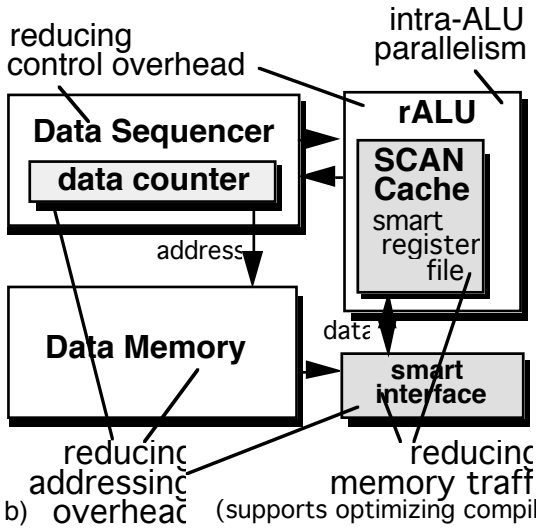


Fig. 2. Basic Structures of: a) computers, and: b) xputers.

	control-procedural	**data-procedural**	**data-driven by "firing"**
the role of:	(von Neumann) computers	xputer s	data flow machines
data flow	derived (by pointers) from control flow	primary activator scheduled at compile time	primary activator determined at run time
control flow	primary activator	derive - only when needed: sparse control residual control	(not existing explicitely)

Fig. 3 Operating principles: xputers versus computers and data flow machines

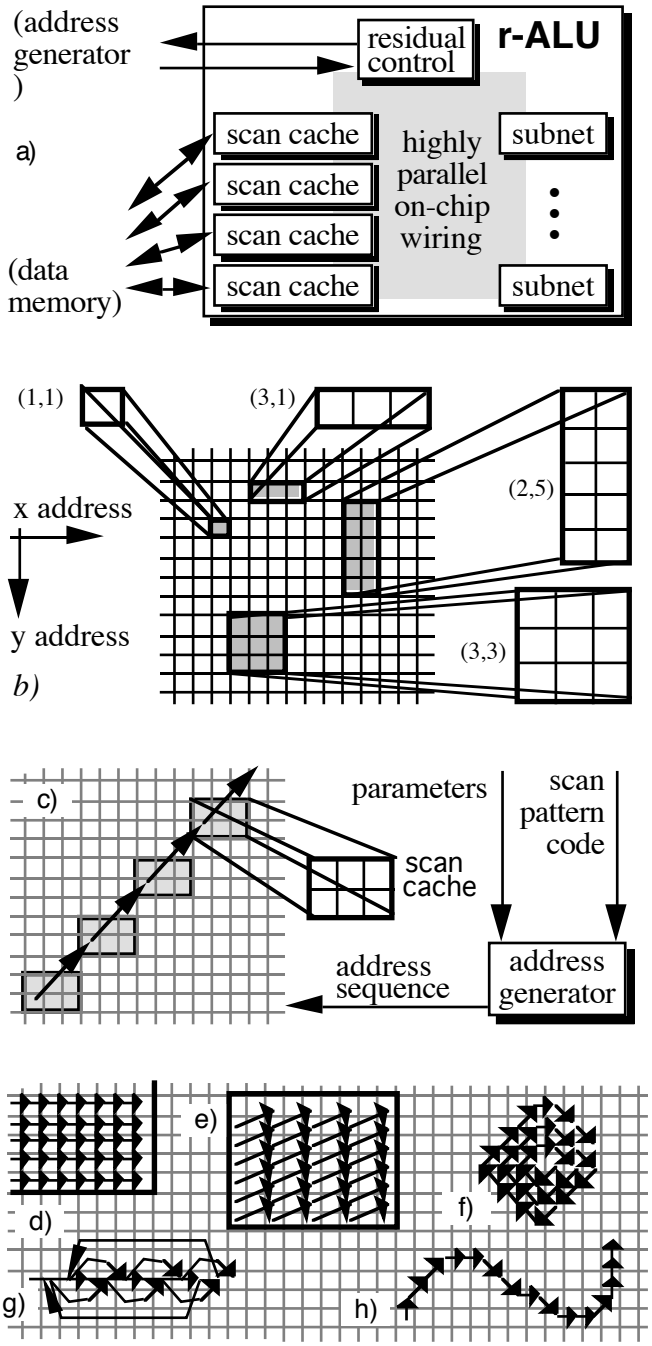


Fig. 4. The MoM (Map-oriented Machine): a) basic structure of the rALU (reconfigurable ALU), b) illustrating 2-dimensional memory space and size-adjustable scan caches (smart register files), c) example of a linear scan pattern, other scan pattern examples: d) video scan, e) slanted video scan, f) spiral scan, g) shuffle scan , h) data-driven scan.

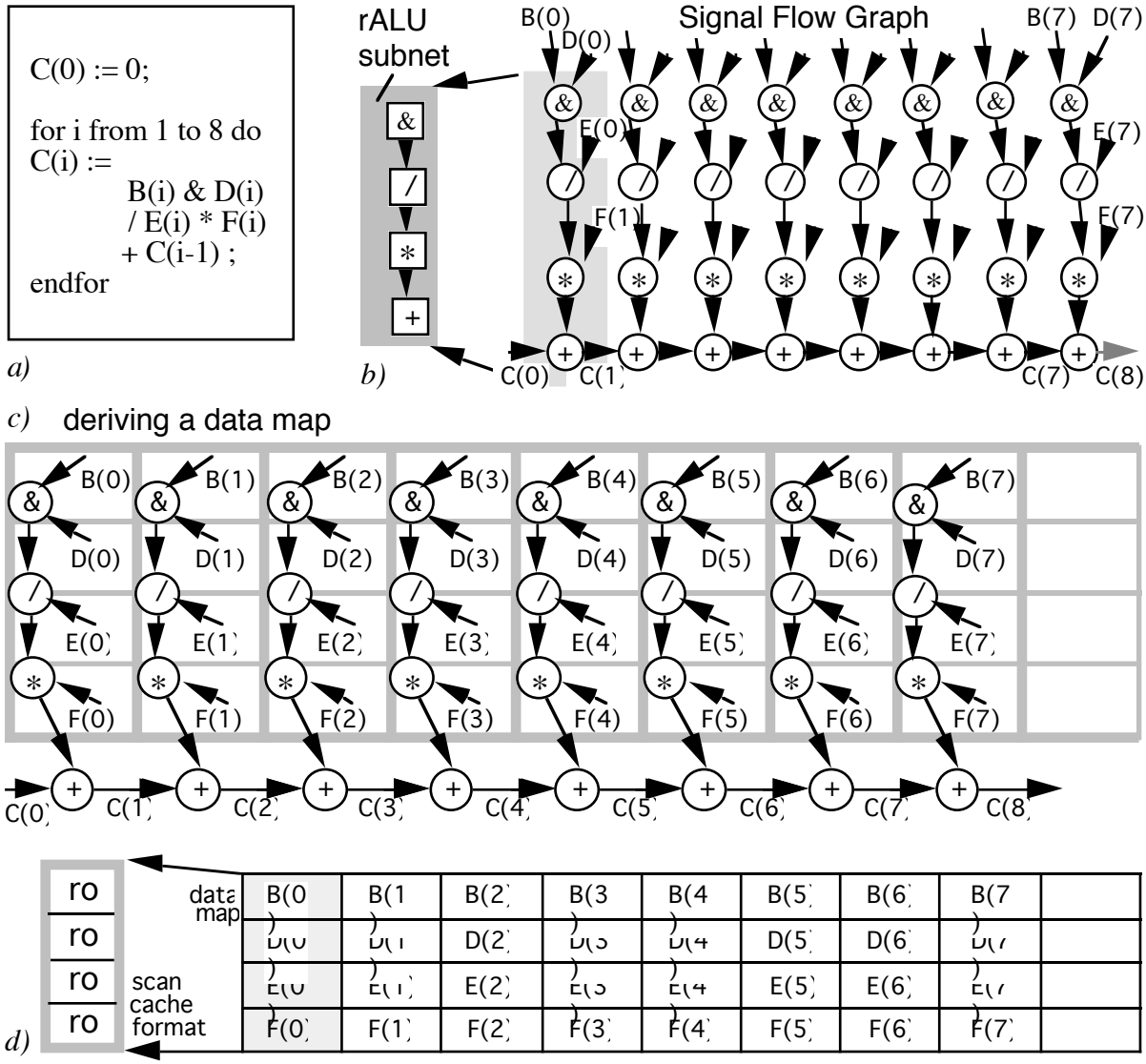


Fig. 5. Illustrating the xputer implementation of a systolizable algorithm: a) textual source, b) signal flow graph representation (SFG) - also illustrating the derivation of a compound operator (rALU subnet), c) illustrating derivation of a data map from SFG, d) data map and deriving a scan cache format.

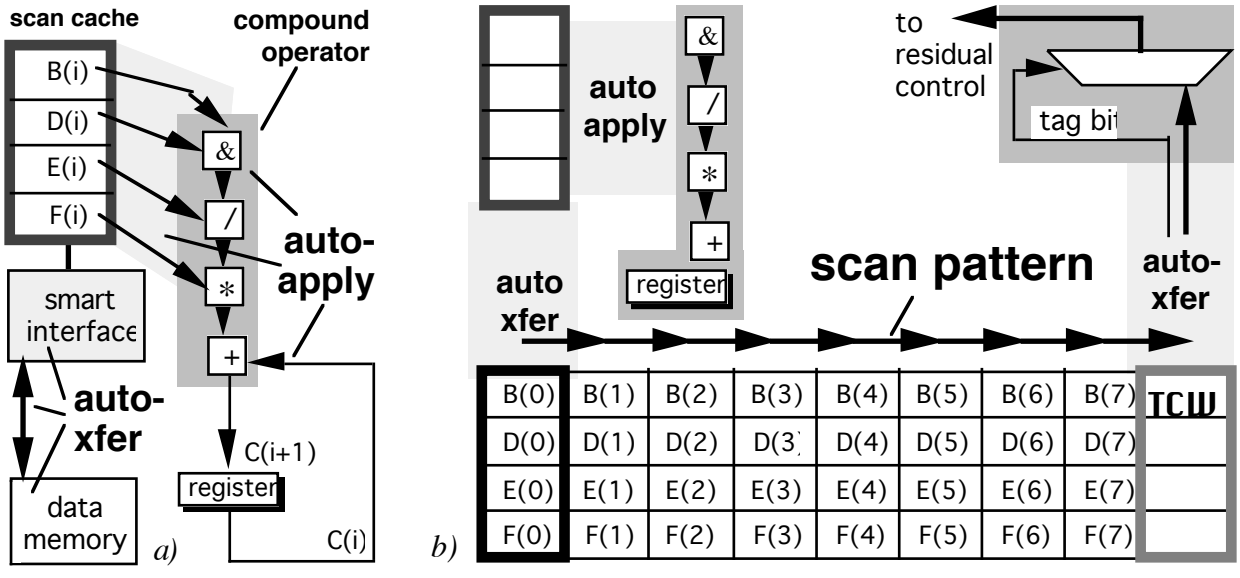


Fig. 6. Illustrating operation principles for example in fig. 5: a) interconnect for auto-apply

and auto-xfer operation, b) illustrating overhead-free scan operation including linkage to residual control at end of scan.

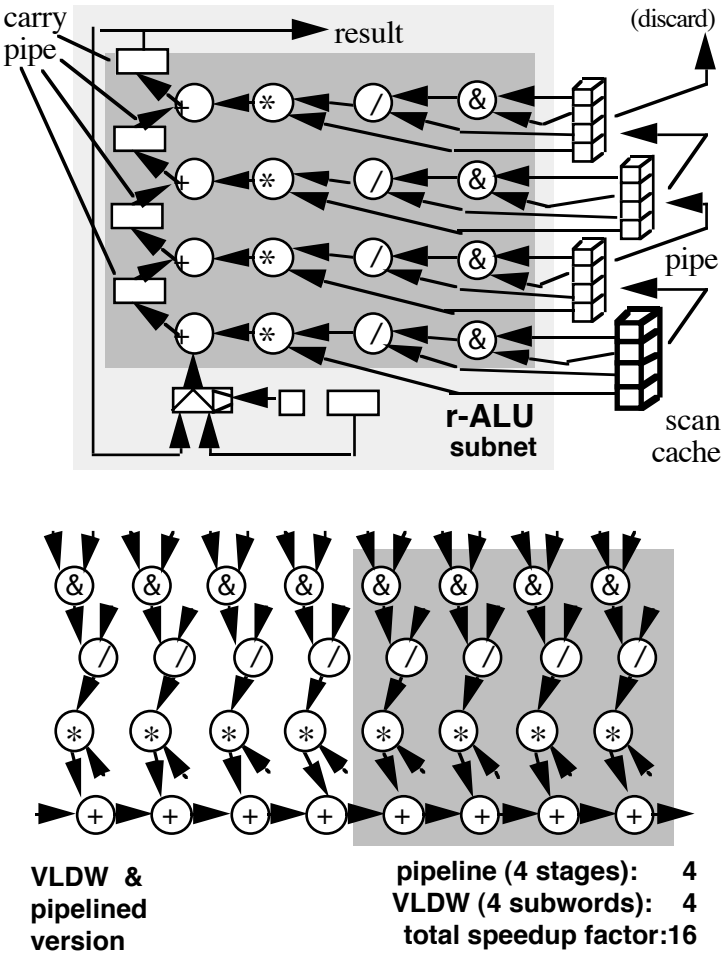


Fig. 7. Illustrating the derivation of combined VLDW and pipelined version xputer implementation from example in fig. 5.

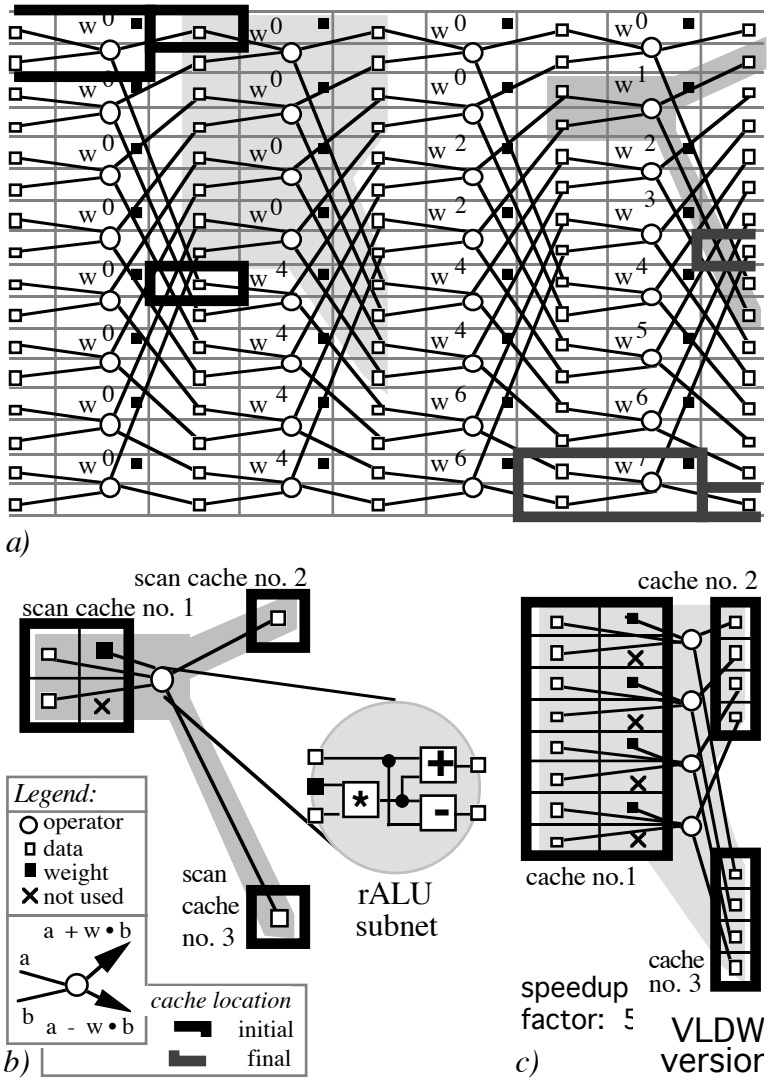
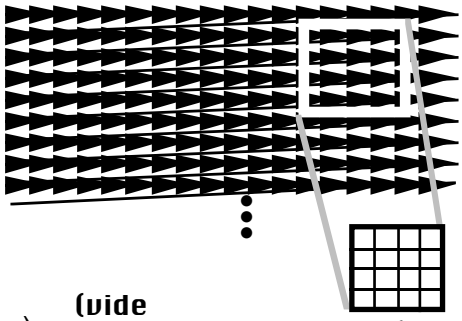
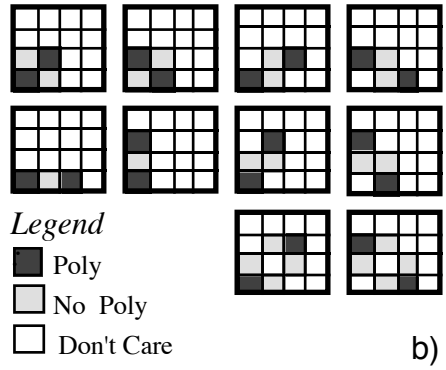


Fig. 8. Illustration of the xputer implementation of a non-systolizable g-algorithm, a constant geometry FFT example: a) deriving data map and 3 scan cache sizes from the SFG, b) deriving a compound operator, c) super compound operator of a VLDW implementation.



design rules	no. of reference patterns
Mead & Conway nMOS	256
IMS CMOS	800

d)



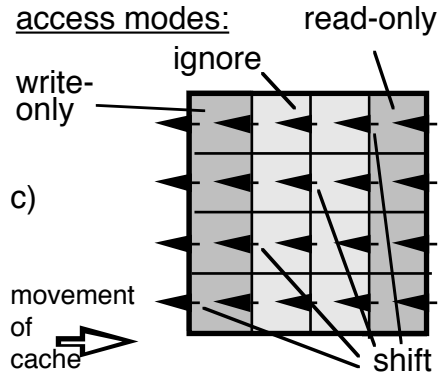


Fig. 9. Pattern matching xputer applications in EDA: a) video scan for grid-based design rule check, b) numbers of reference patterns needed, c) example of a design rule (minimum poly-to-poly separation by 2 units), d) performance-relevant smart register file features of the MoM.

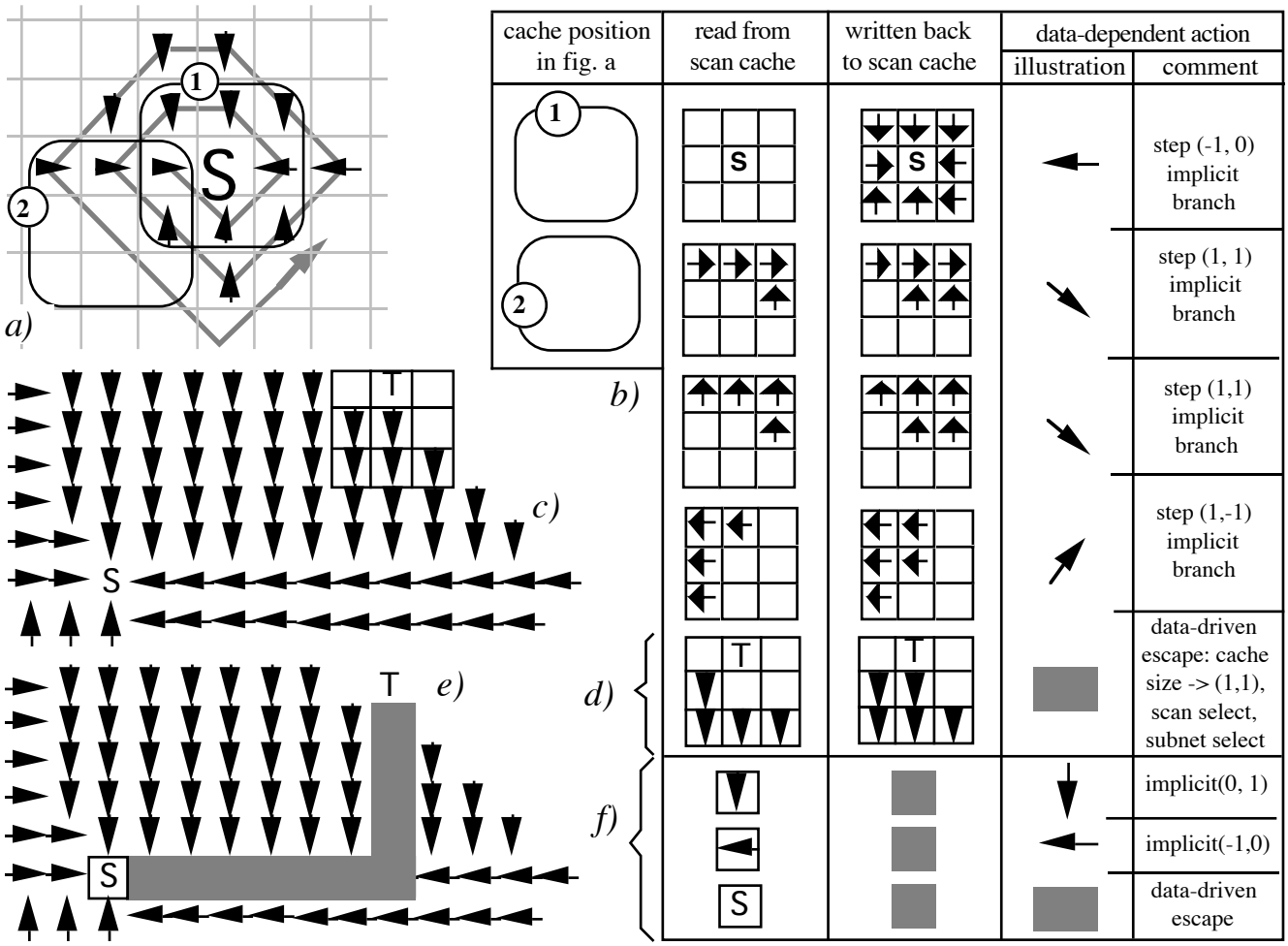


Fig. 10. Lee Routing example: a) wave propagation, b) specific rALU operation repertory, c & d) target T found, e & f) drawing the wire during backtracking.

application area	application example	acceleration factor
EDA (grid-based layout processing)	CMOS design rule check circuit extraction Lee Routing without obstackles with obstackles	> 2000 > 300 > 160 > 800
others	2-dimensional filtering (3 by 3) vector-matrix multiplication	> 300 150

Fig. 11. Acceleration factors obtained experimentally on the MoM.

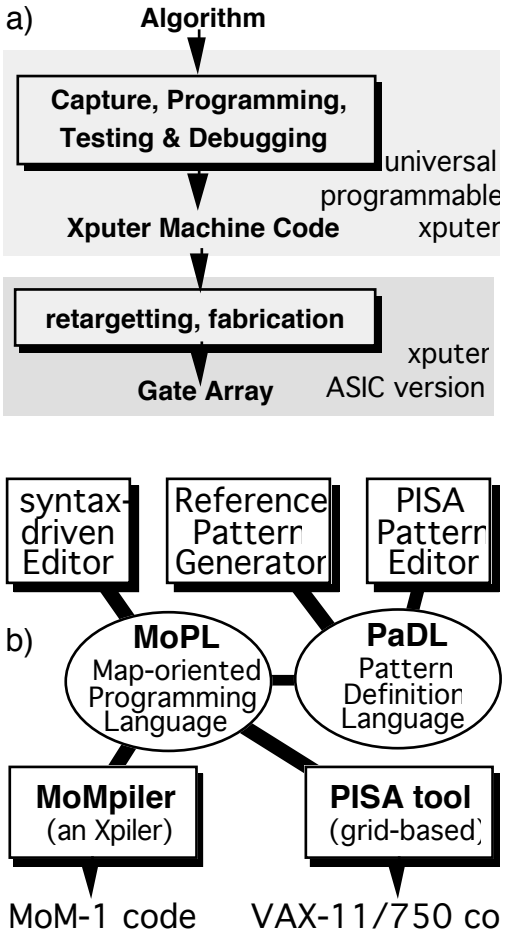


Fig. 12. Xpilers: a) use for ASIC synthesis, b) MoM-DE structure.

typpe of algorithms		acceleration factors (orders of magnitude)	
		systolic arrays	xputers (mono-processor)
non-g--Algorithms (no regular data dependencie		**0**	**0.5**
g-Algorithms (regular dat dependencies)	**systolizable algorithms** (locally regular dat dependencies)	**2**	**3**
	non-systolizable g-Algorithms (globally regular dat dependencies)	**0**	**2.5**

Fig. 13. Classes of parallel algorithms and xputer efficiency.

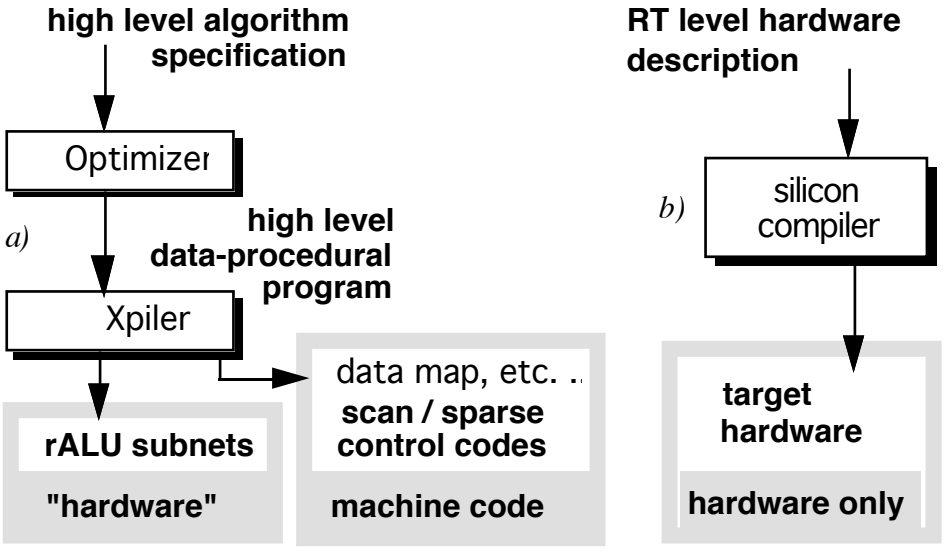


Fig. 14. Xpiler versus silicon compiler: a) xpiler operations, b) silicon compiler operations.

Please, send galley proofs to:

Prof. Reiner W. Hartenstein
Universitaet Kaiserslautern, Bau 12,
Postfach 3049,
DW - 6750 Kaiserslautern,
Germany

phone: (+49-631) 205-2606
or: (+49-7251) 3575
fax: (+49-631) 205-3200,
uucp net mail: abakus@informatik.uni-KL.de