Reiner W. Hartenstein, Jürgen Becker, et al.: Custom Computing Machines vs. Hardware/Software Co-Design: from a globalized point of view; 6th International Workshop On Field Programmable Logic And Applications, FPL'96, Darmstadt, Germany, September 23-25, 1996, Lecture Notes in Computer Science, Springer Press, 1996

Custom Computing Machines vs. Hardware/Software Co-Design: from a globalized point of view

Reiner W. Hartenstein, Jürgen Becker, Rainer Kress

University of Kaiserslautern
Erwin-Schrödinger-Straße, D-67663 Kaiserslautern, Germany
Fax: ++49 631 205 2640, email: abakus@informatik.uni-kl.de

Abstract: The paper gives a generalized survey on Customized Computing with research activities of the emerging new research scenes of Application Specific Instruction Set Processors (ASIPs) and Custom Computing Machines (CCMs). Both scenes have strong relations to Hardware/Software Co-Design. CCMs are mainly based on field-programmable add-on hardware to accelerate microprocessors or computers. The CCM scene tries to make standard hardware more soft for flexible adaptation to a variety of particular application environments. The ASIP scene tries to design an instruction set as an interface between hardware and application closely matching their characteristics.

1 Preface

This paper gives a survey on several fundamentally different approaches to the design of application-specific processors, the application of which we call custom computing (see Table 2). The term „Custom Computing Machine“ (CCM) has been coined by John Gray, now with Xilinx. It has been used for using an Algotronix field-programmable PC extension board to add accelerating extra hardware to a general purpose processor (inside the PC). Later this term has been used within the name of an IEEE conference series „FPGAs for Custom Computing Machines“ [4] [10] [14]. This F-CCM community [12] [27] [34], however, is only one of several different research scenes dealing with application-specific computing setups. An F-CCM design is based on a commercially available von Neumann host microprocessor, which is connected to field-programmable extra hardware for acceleration — in contrast to the VAB-CCM aproach using VLSI extra hardware (VAB stands for VLSI accelerator-based), such as taylored or commodity chips for JPEG, MPEG, FFT and others.
Another research community dealing with application-specific processors has labelled itself ASIP design (Application-Specific Instruction Set Processor design [1] [53]). This approach does not use commercially available processors, nor separately developed hardware extensions. The ASIP scene also does not focus on field-programmable logic devices. The ASIP approach provides a completely application-specific instruction set, whereas custom computing machines based on field-programmable logic (FPL) extensions just use one, or two, or a few specialized instructions added to a standard instruction set.
These two approaches are extremes so far, as designing an ASIP requires the effort to create a complete new microprocessor, whereas the F-CCM scene leaves the host architecture completely untouched, which is a reason of inflexibility of this approach. It would be desirable to have something in the middle, which combines the flexibility of the ASIP approach with the compatibility of using general purpose processors as a kind of host. The solution are procedurally data-driven (PDD) architectures [28] [31] [32] [33] also called transport-triggered architectures [18]. The PDD-CCM scene primarily focuses on reconfigurable (structurally

field-programmable) architectures [21] [28] [29] [44] [45]. The PDD approach also supports using data dependency analysis, such as demonstrated by prototyping systolic algorithms [28].

So far all the CCM scenes mainly cover a kind of hardware/software co-design approach. But an open question is: what is the difference between CCM design and H/S co-design? Why does H/S co-design maintain its own R&D scene [13], apart from the CCM scenes? This question will be discussed briefly.

2 Introduction

Hardware and software have been more or less seen from different view points. A major organizational hurdle has been the enormous difference of prototyping turn-around time between debugging a piece of software (e.g. an algorithm) and debugging hardware. The main reason is the delivery time in hardware prototype manufacturing (e.g. gate arrays), so that most designs cannot be tested earlier than several weeks after completion. This dichotomy of computing resulted in specifying hardware (CAD) before developing software (CASE). It is time to break down the gap between hardware and software and it is time for a marriage between CASE and CAD. One basis for this goal is the availability of increasingly powerful field-programmable logic devices, which is no longer a niche market. Experimental results indicate, that in using such hardware platforms, along with tools driven by recent progress in logic synthesis and formal methods, the turn-around time on the hardware side may be reduced to days or even only hours.

The new research scene focusing on low-cost and flexible solutions achieving a higher throughput for complex applications is the F-CCM scene, which tries to explore the tuning of microprocessors by adding field-programmable hardware in a flexible way, to obtain customizable computers.

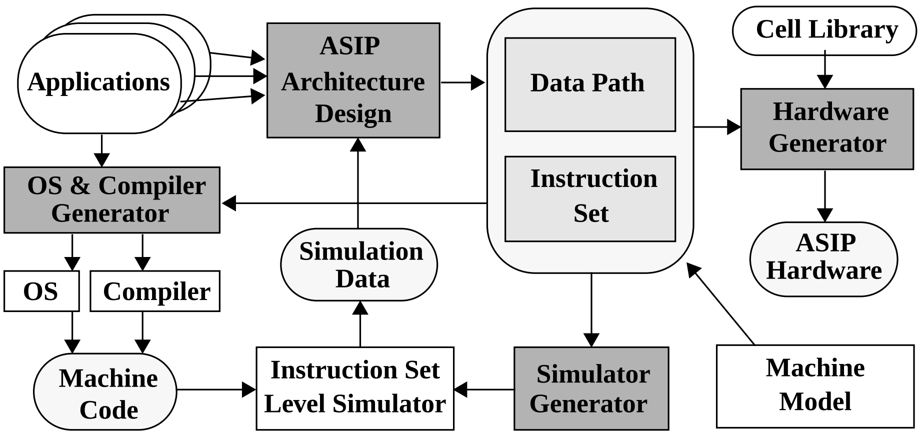
As the tendency towards more complex electronic systems continues, many of these systems are equipped with embedded processors. Essential advantages of these processors include their high flexibility and short design time. Furthermore, they allow an easy implementation of optional product features as well as an easy design correction and upgrading. Reprogrammable processors offer a flexible and low-cost solution for embedded systems with complex algorithms or control intensive applications. The performance of microprocessor-based system depends on how efficiently the application can be mapped to the hardware. One key issue determining the success of the mapping is the design of the instruction set, which serves as an interface between the hardware and the application. How to design an instruction set that closely matches the characteristics of the hardware and of the application is an important design problem. This initiated the new research scene of application-specific instruction set processors (ASIPs).

The paper gives a brief survey on these new ways in customized computing, which have strong relations to Hardware/Software Co-Design. First the ASIP approach is explained briefly. Then two classes of Custom Computing Machines, Enhanced Instruction Set CCMs (EIS-CCMs) and Procedurally data-driven CCMs (PDD-CCMs), are discussed. Finally some performance results are presented.

3 Application Specific Instruction Set Processors (ASIPs)

General purpose processors with their fixed internal architecture usually have been designed to have an extremely efficient layout. Some of them have passed verification procedures, allowing them to be employed in safety-critical applications. In contrast, ASIPs are processors with an application-specific instruction set, where still configurations can take place. Program classes which are executed on an ASIP are seldom changed, while programs executed on a general purpose system are changed frequently. A hardware system can be optimized for fixed programs. Depending upon the application, certain instructions and hardware features are either implemented or not. The definition of ASIPs may include generic parameters, such as compile-time parameters defining, for example, the size of memories and the bitwidth of functional units. ASIPs often operate at low supply voltages and low clock frequencies to minimize power dissipation, while still meeting the required processing performance (data throughput) of their target application(s). Hence, they are popular especially for low-power applications.

Fig. 1. Design environment for ASIP architectures



CAD support for ASIPs is currently not commercially available, but research activities have been reported in this area [2] [8] [11] [30] [43] [48] [53]. Early approaches on automatic instruction set design [8] [11] [30] focused on mainly directly supporting high level languages or increasing the code density. These techniques are not suitable for designing instruction sets for modern pipelined processors, because used instructions are CISC-like . Sato et al. [53] propose an integrated framework (called PEAS-I) for application specific instruction set processors, which generates profiling information from a given set of application benchmarks and their expected data. Based on the profiles, the design system customizes an instruction set from a super set, decides the hardware architecture, and the related software development tools. Despain et al. [43] introduces the ASIA framework, which differs from PEAS-I in terms of the machine model and the design method. Here a pipelined machine model is assumed and the synthesis of the instruction set is performed directly instead of selecting subsets from a super set. Yasuura et al. [2] propose a CAD tool (called COACH) for computer architecture design supported from hardware to system level. From a user specified architecture a compiler of a high level programming language is then generated.
These efforts are mainly situated in the intersection between two existing research disciplines: software compilation and hardware synthesis. The following four main steps in ASIP design can be identified (see Figure 1):

3.1 ASIP architecture design.

In this step a new instruction-set architecture of an ASIP is customized from a super set containing standard arithmetic, memory and control flow instructions. Also a number of specialised instructions are available, e. g. digital filter operations or a full stage of a Viterbi decoding algorithm. This customization of the ASIP´s data path, memory structure and instrucion set is based on generated profiling information from a given set of application benchmarks and their expected data. Constraints on execution speed, area and power dissipation have to be taken into account. The decision of the hardware architecture is derivated from an abstract machine model, which is in [43] and [53] pipelined.

3.2 Operating system (OS) and compiler generation

The task of this step is to generate an operating system and a compiler for an ASIP. The compiler´s output is the assembly code for the target ASIP, which is generated from a high level description, e.g. C program. In contrast to traditional compilers, this retargetable code generation can derive assembly code for a range of different target processors, described e.g. in a hardware description language (instruction patterns, available resources and interconnect). This essential aspect of retargetability makes flexible modifications during ASIP design possible.

3.3 Simulator generation

The purpose of this step is to generate a tool for simulating an assembly code program from the ASIP compiler in an instruction-cycle accurate way. Here preferably the same hardware description serving as input to the compiler generator can be used (with behavioural models for the operations in the instruction set). Optional couplings to VHDL|Verilog based simulators can be useful for verifying complete heterogeneous IC designs containing ASIPs.

3.4 Hardware generation

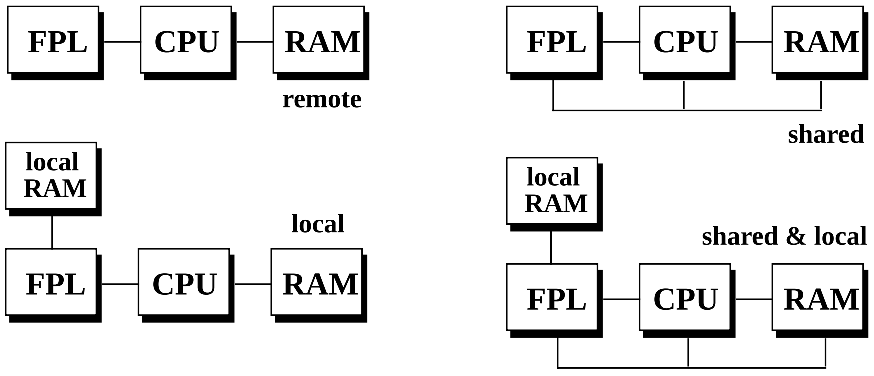
If the simulated ASIP architecture meets all functional and timing requirements, a silicon implemenation satisfying certain clock rate requirements is made in this step. Therefore hardware generators use available libraries with processor cores, bus interfaces, memories, ALUs, (pipelined) arithmetic|relational|special operators, counters, registers, etc. The ASIP may be designed in the form of a parametrisable macro cell that can also be added to a library for designing heterogeneous ICs. After the design of the base structure and the instruction set the testability aspect of the ASIP has to be considered.
Multimedia, mobile and personal communication systems need new integration technologies containing e.g. an ASIP as a core component [26]. Therefore new design techniques are required to support the design of practical systems using ASIPs. In this chapter the most important aspects of an ASIP-based CAD environment has been outlined. These aspects are strongly related to the emerging new discipline of hardware/software co-design.

4 Enhanced Instruction Set Custom Computing Machines (EIS-CCMs)

Custom Computing Machines (CCMs) represent machines where the hardware can be reconfigured and customized on a program-by-program basis, or where VLSI extra hardware is added for acceleration. The commercial relevance of CCMs is based on the large number of required application-specific microprocessors in embedded systems. The growing market of field-programmable logic (FPL) indicates the increasing importance of structural programming. Enhanced instruction set custom computing machines (EIS-CCMs) try to explore the tuning of general purpose microprocessors by adding field-programmable (F–CCMs) or VLSI extra hardware (VAB-CCMs) in a flexible way, to obtain higher acceleration factors for specific applications. A few more powerful operators are added, while EIS-CCMs do not have an own programming paradigm. There is still the strong von Neumann based coupling between instruction sequencer and ALU. This chapter focuses on classifying the F-CCM subclass in detail.

F-CCMs can be classified according to different criteria. From a user's point of view, the following three aspects seem to be most interesting. First, the field-programmable devices have direct access to memory, which is important for the I/O bandwidth to these devices. Second, i the interconnect between the field-programmable devices can be fixed on a printed circuit board or reconfigurable (at least statically), which influences the routability of designs in the FPGAs. And third, how much hardware expertise is required to configure the custom computing machine to the requirements of the application, which corresponds to the user-friendliness of the whole environment.

Fig. 2. Memory organization of EIS-CCM



The memory access can be distinguished to four cases: local and shared; shared; local; and remote. Figure 2 displays the different memory structures of custom computing machines according to this aspect. E.g. Raimbault et al. [51] use shared memory in their general purpose F-CCM called ArMen. Examples for remote memory access are PRISM and PRISM-II by Athanas et al. [6] [7] [55]. The shared and local approach is implemented e. g. in SPLASH [23] [24], SPLASH-2 [3] [4] [25], PeRLe-1 [9] [10] TM-1 [22] and in the commercial version of SPLASH-2 called WILDFIRE [49]. Examples, where the added FPL hardware has only local memory access, are a commercial F-CCM called EVC [16] [50] and Enable++ [42].
The interconnect between the FPGAs can be classified as fixed or configurable. Within these categories there are gradual differences regarding the extend of flexibility in the interconnect. E.g. a crossbar is more flexible than a simple switch between preselected configurations, because the crossbar allows to connect any pin to any other of a second device. Examples for fixed FPGA interconnect are PRISM and PRISM-II [6], [7], [55], whereas the interconnect between the FPGAs in TM-1 [22] and Enable++ [42] is configurable.
Currently, only few custom computing machines exist, which do not require any hardware expertise in programming. Although there are a lot of them, which allow to synthesize designs from derivatives of programming languages, many of them require the addition of compiler directives and/or constraints in the source code and/or when invoking the compiler to guide the synthesis process. Most of these informations are directly related to hardware design (like wordlength of basic types, size or speed constraints, etc.) and unfamiliar to a conventional application programmer. The spectrum of the programmability of custom computing machines ranges from low-level hardware design (schematic entry) to high-level programming (e. g. C), with high-level hardware synthesis somewhere in the middle. The BORG [15], a commercial F-CCM, needs schematic entry to synthesize their functionalities, ArMen [51] uses a high-level hardware description language, whereas PRISM [6], [7] can be programmed by a high level programming language.

5 Procedurally data-driven Custom Computing Machines (PDD-CCMs)

PDD-CCMs combine the flexible ASIP approach with the compatibility of using general purpose processors as a kind of host in the EIS-CCM class. The PDD-CCM class consist of two subclasses: Parallel reconfigurable ALU Custom Computing Machines (PrA-CCMs) [31] [32] [33], and others [18] [28]. This chapter focuses on describing the PrA-CCM approach in detail.

PrA-CCMs use structural programmable ALUs also suitable for arithmetic applications, which are very flexible and not commercially available. PrA-CCMs realizes a kind of instruction level parallelism comparable to the scene of “normal” microprocessors. The communication structure is organized at compile time, which results in a run-time/compile-time shift of overhead and high speed-ups. The von Neumann paradigm is not feasible for PrA-CCMs, because of the tight coupling between instruction sequencer and ALU including a compact instruction code. One solution for a better paradigm is to sacrifice the instruction sequencer and to use a data sequencer instead. The resulting architecture class, called Xputer, is produrally data-driven and has only a loose coupling between data sequencer and rALU. The memory is now mainly data memory and the data manipulations are transport-triggered. Figure 3 shows the compatible use of an Xputer-based Pra-CCM as co-processor of a usual host environment usable for general purpose and rapid prototyping. The software e.g. reconfiguration tools, operating system, compiler, application development software etc. is running on the host. The stand-alone Xputer version (without host integration) can be used e. g. for embedded systems [36].

Xputer-based accelerators consist of several (up to seven) Xputer modules. The modules are connected to a host computer. Making use of the host simplifies disk access and all other I/O operations. After setup, each Xputer module runs independently from the others and the host computer until a complete task is processed. Each module generates an interrupt to the host when the task is finished. So, the host is free to concurrently execute other tasks in-between. This allows the use of the Xputer modules as a general purpose acceleration board for time critical parts in an application.

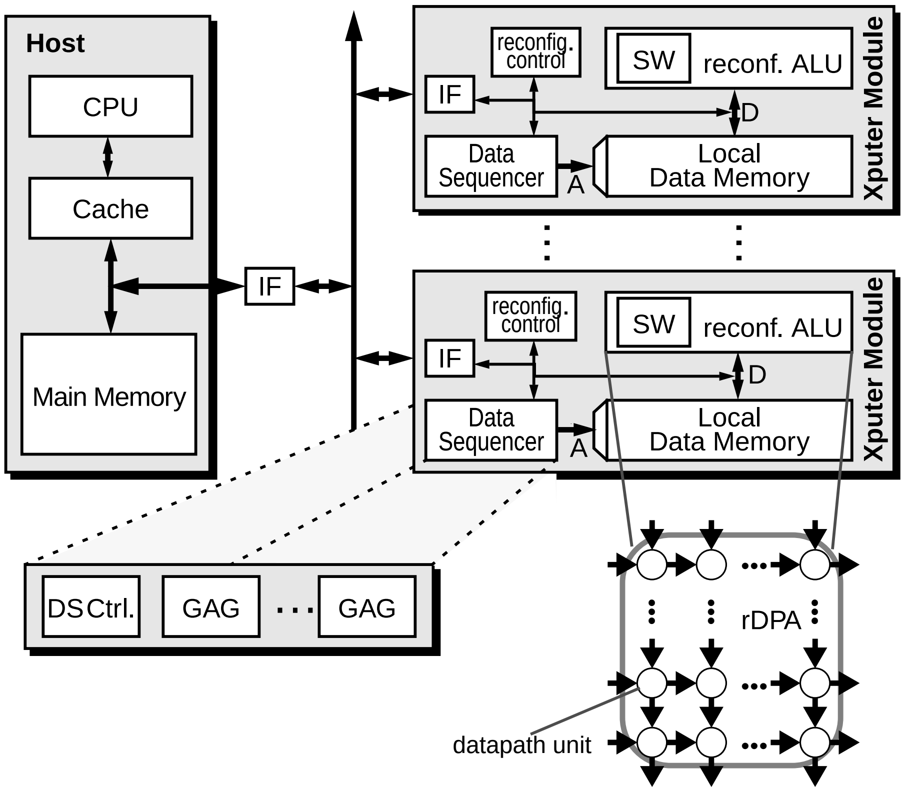
The basic structure of an Xputer module consists of three major parts (figure 3):

- the data memory
- the reconfigurable arithmetic and logic unit (rALU) including several rALU subnets with multiple scan windows
- the data sequencer (DS) comprising several generic address generators (GAGs)

The data memory which is local on each Xputer module, is primarily organized two-dimensional, but can also be interpreted as higher dimensional. It contains the data which has to be accessed or modified during an application. The data is arranged in a special order to optimize the data access sequences. This arrangement is called data map. The scan windows (SW) serve as interface of the rALU subnets to the data memory. A scan window holds a copy of the data out of a local neighbourhood of the data map. Scan windows are adjustable in size during run-time of an application. The main memory of the host and the local memories are logically a single shared memory (figure 3). The host has access to all local memories, and vice versa the Xputer modules have access to the main memory as well as to other local memories. Of course local memory access is much faster than remote access to others via the bus. Additionally, the external bus can be used only by a single Xputer-module at a time.

A large amount of input data is typically organized in arrays where the array elements are referenced as operands of a computation in a current iteration. The sequence of data accesses in iterations shows a regularity which allows to describe this sequence by a number of parameters. The hardwired generic address generators (GAGs) of the data sequencer (DS) interpret these parameters and compute generic address sequences to access the data. This results in a controlled movement of the scan windows over the data map, each controlled by a single GAG [52]. The address sequences are called scan patterns. Each time the scan windows move one step, a compound operator of the corresponding rALU is evaluated. Thus the data sequencer represents the main control instance of an Xputer. Data sequencing in general means, that the GAGs address data at correct locations in the data memory by generic scan patterns and load it into the scan windows of the rALU subnets.

Fig. 3. Hardware structure of an Xputer-based PrA-CCM (used as co-processor)



All data manipulations are done in the reconfigurable ALU (rALU). The rALU consists of several rALU subnets. Each subnet is distinguished by its rALU subnet number. Each Xputer-module can simultaneously use three different rALU subnets on its board. The rALU subnets perform the computations on the data which are provided by the scan windows by applying a compiler configured complex operator on that data. That complex operator is called compound operator. Each rALU subnet has four output flags to control the generic address generators (GAGs), e.g. for data-dependent scan patterns. The operations in the rALU are done in parallel to the address computations of the GAGs. In order to have a general purpose interface between the rALU and the GAGs, these components are transport-triggered, which means the availability of data triggers the operation. This allows to be very flexible in implementing different rALU subnets, as the subnets do not need to have the same computation times. Furthermore, it allows to implement a pipelined rALU concept as well as a combinatorial net. After finishing, the rALU signals the end of the computation to the GAGs.

As long as a rALU operates on the local memory of the same Xputer module, the data accesses can be done in parallel to the other Xputer modules. A non local data access has to be done sequentially via the external bus. Using this bus, the data sequencer can also access data in the host memory. The control unit in the data sequencer is responsible for the configuration of the GAGs. The reconfiguration control unit holds the parameter sets and the rALU configuration data for the complete application in its memory.
For the evaluation of standard C programs, word-oriented operators such as addition, subtraction, or multiplication are required. This realization of a reconfigurable architecture for word-oriented operators needs a more coarse grained approach for the logic block architecture of the rALU. The granularity is more at operator level instead of gate level like in commercially available FPGAs. Complete arithmetic and logic operators can be implemented in a single logic block. Thus, in contrary to FPGAs, such a block is called datapath unit (DPU) to show its prior functionality. The datapath unit can be optimized to implement operators faster and more area efficient than FPGAs. A word-oriented structure for the datapath unit is no drawback since random logic or control logic need not to be supported by this architecture. Furthermore, such a structure requires less configuration bits which saves additional area. In addition the timing is more predictable since there are less transits of signals via programming switches. This greatly simplifies the synthesis system since it saves a necessary back-annotation step to determine the exact delay times for simulation. In the current prototype, the reconfigurable datapath architecture (rDPA) serves as rALU [46]. It consists of 72 identical word-oriented DPUs. Further it has a regular architecture also across chip boundaries. This allows to see an rDPA array consisting of several rDPA chips as a large reconfigurable datapath architecture. Thus an rDPA can be easily adapted to the size of the problem. The communication between neighbouring datapath units is synchronized by handshake. This avoids problems with clock skew and simplifies the extension of the rDPA over printed circuit board (PCB) boundaries. The DPUs are connected to each other with short local connections in a regular array. Due to the wide datapath width, a global interconnect is multiplexed in time to save area. In the current prototype, all I/O buses of the rDPA chips are connected together. A given placement of the operators in the DPUs is always routable due to the multiplexing. Together with the rALU controller the rDPA forms the data-driven rALU, as shown in figure 3. The control chip consists of a control unit, a configuration unit, a register file, and an address generation unit for addressing the DPUs. The register file is useful for optimizing memory cycles. The rDPA control unit holds a program to control the different parts of the data-driven rALU. The current version of the rDPA is described in [39].

Table 1. Acceleration factors of EIS-CCMs as reported by the authors

algorithm	CCM		reference		acceleration factor
	machine	speed	machine	speed	
long multiplication [9]	DECPeRLe-1	66 Mbit/s	Cray II or Cyper 170/750		16
linear convolution (16-point) [17]	EVC1	5 s	SPARC10 using Matlab	95 s	10
JPEG, 704x576 RGB image, 24 bits / pixel [52]	MoM-3	64 ms	SPARC10/51	1.51	24
Ising 128 lattice, 1000 iterations [52]	MoM-3	3.85 s	SPARC10/51	1208 s	331

Table 1. Acceleration factors of EIS-CCMs as reported by the authors (Continued)

algorithm	CCM		reference		acceleration factor
	machine	speed	machine	speed	
Radiation Tracker TRT [42]	Enable++ (estimation)	9 s execution time	modern RISC (Power PC, Alpha, Sparc10		180
DNA sequence comparison [41]	Splash2 (16 modules)	43000 M CUPS (cells updated per second)	MP-1	32 M	1344
			CM-2	5.9 M	7288
			Sun 10/30GX	1.2 M	35833
Grid-based Design Rule Check[31]	MoM-1		optimized VAX 11/750		>2000
			VAX 11/750, directly mapped from MoM-1		15000

For a previous Xputer prototype, called MoM-2 [38] [40] [56], a shared memory was used and its functionality was programmed by a hardware description language. In the current prototype MoM-3 [35] [37] [46] [52] [54] shared and local memory access is possible, and the hardware can be programmed by a static subset of C. The integration of the MoM-3 as embedded accelerator into a usual host environment is realized by the parallelizing compilation framework CoDe-X [37], which accepts X-C source programs. X-C (Xputer-C) is a C extension. The X-C source input is partitioned in a first level into a part for execution on the host (host tasks, also permitting dynamic structures and operating system calls) and a part for execution on the Xputer (Xputer tasks). Parts for Xputer execution are expressed in a X-C subset, which lacks only dynamic structures, but includes language features to express scan patterns [5] [54]. At second level this input is partitioned in a sequential part for the DS, and a structural part for the rDPA. Experienced users may use special MoPL library functions [5] to take full advantage of the high acceleration factors possible by the Xputer paradigm. Used for H/S Co-Design this partitioning approach introduces data sequencing as a data-driven backbone paradigm to this discipline [13]. A backbone paradigm has not been available for co-design before.

Table 2. Survey on different CCM (Custom Computing Machines) R&D scenes.

backbone paradigm	EIS-CCM		ASIP-CCM	PDD-CCMs		(Hardware Software Co-Design)
	VAB-CCM	F-CCM		PrA-CCM (Xputer-based)	others	
von Neumann paradigm use	host only	host only	total system			host only
tinker toy (no backbone paradigm av.)	extra hardware	extra hardware	(none)			extra hardware
data sequencing (transport-triggered)				total system	total system	total system
systolic data streams				data part		

6 Performance

ASIPs have optimizing criterias such as performance, area and power consumption. Dependent on the application different criterias are focused on and the corresponding trade-offs are optimized with the given abstract hardware models.
For some custom computing machines performance comparisons have been published. Some of them are listed in table 1 from figures given by the authors. Since they compare different algorithms run on different machines, these figures are not directly comparable. Furthermore, they are based on the informations supplied by the respective authors, so that all inaccuracies are in their responsibility. Nevertheless, they may give an impression on the benefits and the limitations of custom computing machines, if interpreted carefully.

7 Conclusions

The paper has given a brief survey on research and most important requirements in customized computing, including a classification scheme of Custom Computing Machines. The viewed approaches are strongly related to the emerging discipline of Hardware/Software Co-Design, which is an interdisciplinary research activity. These concepts should be applied to the new design problems of today. Performance aspects have also been covered. Implementation completion data, performance figures, and other relevant information have been taken from publications by the project teams without verification.
Designing Custom Computing Machines also means practicing Co-Design. Custom Computing Devices is almost a synonym of Hardware/Software Co-Design. The main difference is, that Co-design focuses on partitioning, whereas most CCM methods focus on a particular technology platform. ASIP design means designing appication-specific hardware, rather then H/S co-design Co-Design focuses on the hardware/software partitioning problem. I. e., designing a processor and a compiler for it is not really a partitioning problem. This means, that all CCM approaches are Co-Design approaches, except ASIP development.
We have seen, that the availability of a backbone paradigm is essential to integrate the wide design space of myriads of architectures into a general methodology. Such a backbone paradigm is available only for AISP-CCMs and for PDD-CCMs. For the ASIP approach the von Neumann paradigm is an excellent backbone paradigm, but the EIS-CCM approach is not supported by von Neumann, so that the configurable add-ons tend to be a very wide variety of tinker toy structures. But PDD adds backbone to CCM and Co-Design.

References

1. A. Postula, D. Abramson, P. Logothethis: Synthesis for Prototyping of Application-specific Processors; Proc. 3rd Asia Pacific Conf. on Hardware Description Languages (APCHDL’96), Bangalore. India, Jan. 1996
2. H. Akaboshi, H. Yasuura: COACH: A Computer Aided Design Tool for Computer Architects; IEICE Trans. Fundamentals, Vol. E76-A, No. 10, Oct. 1993
3. J. M. Arnold, D. A. Buell, E.G. Davis: Splash-2; 4th Annual ACM Symposium on Parallel Algorithms and Architectures (SPPA ‘92), pp. 316-322, ACM Press, San Diego, CA, June/July 1992
4. J. M. Arnold: The Splash 2 Software Environment; IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'93, IEEE Computer Society Press, Napa, CA, April 1993
5. A. Ast, J. Becker, R. W. Hartenstein, R. Kress, H. Reinig, K. Schmidt: Data-procedural Languages for FPL-based Machines; 4th Int. Workshop on Field Programmable Logic and Appl., FPL’94, Prague, Sept. 7-10, 1994, Lecture Notes in Computer Science, Springer, 1994
6. P. Athanas: A Adaptive Machine Architecture and Compiler for Dynamic Processor Reconfiguration; Ph.D. Thesis, Brown University, May 1992

7. P. Athanas, H Silverman: Processor Reconfiguration Through Instruction-Set Metamorphis; IEEE Computer, Vol. 26, pp. 11 - 18; March 1993
8. J. P. Bennett: A Methodology for Automated Design of Computer Instruction Sets; Ph. D. thesis, University of Cambridge, Computer Laboratory, 1988
9. P. Bertin, D. Roncin, J. Vuillemin: Programmable Active Memories: a Performance Assessment; DIGITAL PRL Research Report 24, DIGITAL Paris Research Laboratory, March 1993
10. P. Bertin, Hervé Touati: PAM Programming Environments: Practice and Experience; IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'94, IEEE Computer Society Press, Napa, CA, April 1994
11. P. Bose, E. Davidson: Design of Instruction Set Architectures for Support of High Level Languages; Proc. of the 11th Annual Int. Symposium on Computer Architecture, 1984
12. N. N.: Brigham Young University Reconfigurable Logic Lab Bibliography; e-mail: wirthlin@fpga.ee.byu.edu, 1995
13. K. Buchenrieder: Hardware/Software Co-Design; IT Press, Chicago 1995
14. D. A. Buell, K. Pocek: Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'95, IEEE Computer Society Press, Napa, CA, April 1995
15. P. K. Chan: A Field-Programmable Prototyping Board: XC4000 BORG User's Guide; UCSRC-CRL-94-18, April 1994
16. S. Casselman, J. Schewel, M. Thornburg: H.O.T. (Hardware Object Technology) Programming Tutorial; Release 1, Virtual Computer Corporation, January 1995
17. H.A. Chow, S. M. Casselman, H. M. Alunuweiri: Implementation of a Parallel VLSI Linear Convolution Architecture Using the EVC1; IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'95, IEEE Computer Society Press, Napa, CA, April 1995
18. Henk Corporaal: Transport Triggered Architectures; Ph. D. thesis, Tech. University of Delft, Holland, 1995
19. Henk Corporaal, Paul van der Arend: MOVE32INT, a Sea of Gates realization of a high performance Transport Triggered Architecture; Microprocessing and Microprogramming vo. 38, pp. 53-60, North-Holland, 1993
20. Henk Corporaal: Evaluating Transport Triggered Architectures for scalar applications; Transport Triggered Architecture; Microprocessing and Microprogramming vo. 38, pp. 45-52, North-Holland, 1993
21. B. K. Fawcett: FPGAs as Configurable Computing Elements; Int'l Worksh. on Reconfigurable Architectures, @ ISPS'95 - 9th Int'l Parallel Processing Symposium, Santa Barbara, 24. - 29. April 1995
22. D. Galloway, D. Karchmer, P. Chow, D. Lewis, J. Rose: The Transmogrifier: The University of Toronto Field-Programmable System; Technical Report CSRI-306, Computer Sys. Res. Inst., Univ. of Toronto, Canada, June 1994
23. M. Gokhale, B. Holmes, A. Kopser, D. Kunze, D. Lopresti, S. Lucas, R. Minich, P. Olsen: SPLASH: A Reconfigurable Linear Logic Array; International Conference on Parallel Processing, pp. I 526-I 532, 1990
24. M. Gokhale, W. Holmes, A. Kopser, S. Lucas, R. Minnich, D. Sweely, D. Lopresti: Building and Using a Highly Parallel Programmable Logic Array; IEEE Computer, Vol. 24, No. 1, IEEE Computer Society Press, January 1991
25. M. Gokhale, R. Minnich: FPGA Computing in a Data Parallel C; FCCM'93, IEEE Workshop on FPGAs for Custom Computing Machines, Napa, CA, April 1993, IEEE CS Press 1993
26. G. Goossens, J. van Praet, D. Lanneer, W. Geurts, F. Thoen: Programmable Chips in Consumer Electronics and Telecommunications; NATO ASI in Hardware/Software Co-Design, Tremezzo, Italy, June 19-30, 1995
27. S. A. Guccione: List of FPGA-based Computing Machines; guccione@ccwf.cc.utexas.edu, last updated: June 2, 1995
28. R. Gupta: Cosynthesis of Hardware and Software for Digital Embedded Systems; Kluwer 1995
29. R. Gupta, G. de Micheli: Hardware/Software Co-Synthesis for Digital Systems; IEEE D&T of Computers, Sept. 1993
30. F. M. Haney: ISDS - A program that designs computer instruction sets; Fall Joint Computer Conference, 1969
31. R. Hartenstein, A. Hirschbiel, M. Weber: MoM - a partly custom-designed architecture compared tzo standara hardware; Proc. COMP EURO, Hamburg, Germany, 1989; IEEE Press 1989

32. R. Hartenstein, M. Riedmüller, K. Schmidt, M. Weber: A Novel ASIC Design Apoproach based on a New Machine Paradigm; IEEE Journal of Solide State Circuits, July 1991
33. R. Hartenstein, A. Hirschbiel, K. Schmidt, M. Weber: A novel Paradigm of Parallel Computation and its Use to implement Simple High-Performance Hardware; Future Generation Computing Systems 7 (1991/92), p. 181 - 198
34. R. Hartenstein et al.: Custom Computing Machines (invited opening keynote); Proc. DMM'95 - Int'l Symp. on Design Methodologies in Microelectronics, Smolenice Castle, Slovakia, September 1995
35. R. Hartenstein, J. Becker, R. Kress, H. Reinig: High-Performance Computing Using a Reconfigurable Accelerator; CPE Journal, Special Issue of Concurrency: Practice and Experience, John Wiley & Sons Ltd., 1996
36. R. Hartenstein, J. Becker, R. Kress: An Embedded Accelerator for Real Time Image Processing; 8th EUROMICRO Workshop on Real Time Systems, L'Aquila, Italy, June 1996
37. R. Hartenstein, J. Becker, R. Kress: Two-Level Hardware/Software Partitioning Using CoDe-X; Int. IEEE Symp. on Engineering of Computer Based Systems (ECBS), Friedrichshafen, Germany, March 1996
38. R. Hartenstein, A. Hirschbiel, M. Weber: Mapping Systolic Arrays onto the Map-oriented Machine (MoM); 3rd Int. Conf. on Systolic Arrays, Killarney, Ireland, Prentice-Hall, May 1989
39. R. Hartenstein, R. Kress: A Datapath Synthesis System for the Reconfigurable Datapath Architecture; Asia and South Pacific Design Aut. Conf., ASP-DAC'95, Nippon Convention Center, Makuhari, Chiba, Japan, Aug. 29 - Sept. 1, 1995
40. A. Hirschbiel: A Novel Processor Architecture Based on Auto Data Sequencing and Low Level Parallelism; Ph.D. Thesis, University of Kaiserslautern, 1991
41. D. T. Hoang: Searching Genetic Databases on Splash 2; Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'93, Napa, CA, pp. 185-191, April 1993
42. H. Högl, H. Kugel, J. Ludvig, R. Männer, K. H. Noffz, R. Zoz: Enable++: A second generation FPGA processor; FCCM'95, IEEE Workshop on FPGAs for Custom Computing Machines, Napa, CA, April 1995, IEEE CS Press 1995
43. Ing-Jer Huang, A. Despain: Synthesis of Application Specific Instruction Sets; IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 14, No. 6, June 1995
44. A. Koch, U. Golze: A Universal Co-processor for Workstations; in: W. Moore, W. Luk (eds.): More FPGAs, Abbington EE&CS Books 1993,
45. A. Koch, U. Golze: A Universal Co-processor for Workstations; Proc. FPL'93 - Field-programmable Logic and Applications, Oxford, UK 1993
46. R. Kress: A Fast Reconfigurable ALU for Xputers; Ph.D. Thesis, University of Kaiserslautern, 1996
47. P. Marwedel: Code Generation for Embedded Processors: An Introduction; Proc. of IFIP International Workshop on Logic and Architecture Synthesis, Grenoble, France, Dec. 1995
48. P. Marwedel, G. Goossens (ed.): Code generation for embedded processors, Kluwer Academic Publ. 1995
49. N. N.: WILDFIRE Custom Configurable Computer WAC4010/16; Document # 11502-0000, Rev. C, Annapolis Micro Systems, Inc., April 1995
50. N.N.: EVC-1 Info 1.1; Virtual Computer Corporation, 1994
51. F. Raimbault, D. Lavenier, S. Rubini, B. Pottier: Fine grain parallelism on a MIMD machine using FPGAs; FCCM'93, IEEE Worksh. on FPGAs for Custom Computing Machines, Napa, CA, April 1993; IEEE CS Press 1993
52. H. Reinig: A Scalable System Architecture for Custom Computing; Ph.D. Thesis (in prep.) Kaiserslautern, 1996
53. J. Sato, M. Imai, T. Hakata, A. Y. Alomary, N. Hikichi: An Integrated Design Environment for Application Specific Integrated Processor; Proc. of IEEE Int'l Conf. on Computer Design: ICCD 1991, pp. 414-417, Oct. 1991
54. K. Schmidt: A Program Partitioning, Restructuring, and Mapping Method for Xputers; Ph.D. Thesis, Kaiserslautern 1994
55. A. Smith, M. Wazlowski, L. Agarwal, T. Lee, E. Lam, P. Athanas, H. Silverman, S. Ghosh: PRISM-II: Compiler and Architecture; FCCM'93, IEEE Worksh. on FPGAs for Custom Computing Machines, IEEE CS Press 1993
56. M. Weber: An Application Development Method for Xputers; Ph.D. Thesis, Univ. of Kaiserslautern, 1990