BERICHT NR. 1996/11

INTEGRATED SOFTWARE / HARDWARE DEVELOPMENT SYSTEM

Walter H. Burkhardt and Stefan Rust
Universitaet Stuttgart
Institut fuer Informatik
Breitwiesenstr. 20
D-70565 Stuttgart, Germany
e-mail: burkhardt@informatik.uni-stuttgart.de
Fax: 1-49-711-781-6370

Abstract:

The usual design and development method in systems architecture of separate portions of soft- and hardware components introduces high inefficiencies, due to redundancies and seldom used functions on the wrong levels of iplementation. A better approach offers a methodology for an integrated software, firmware, and hardware design. Required are, however, immense computer support and graphical capabilities. A system of this kind is especially valuable for development under constraints of performance, cost, time, space, or other limitations. We present such a system and show its capabilities in the development of a defect-free sorter systems module.

Keywords: Application-Specific-Architecture, Integrated design and developmentsystems, CAD, ASIC, Performance.

1. Introduction.

The economies of integrated circuits permit realization of the required functionality by specialized digital systems [Amm87], where in the future up to one billion circuits per chip can be expected [Hoe78]. Applications exist e.g. in Communications and Signal processing, protocol converters, speech recognition and filter functions, Autoelectronics, control of brakes and ignition, Instrumentation, control of the sensors, Picture processing, pattern recognition, object tracking, and contrast improvement. These applications appear especially in aerospace systems, machine control, and home electronics, etc. Complicated design requirements require most often a compact realization of the steeringand control devices as embedded systems [Axf89, Jun88] in systems on silicon.

Different possibilities exist for the realization of such embedded systems by utilization of existing processors with programming of the sytems functions or by special hardware designs [HNS86]. The first approach has the advantage of fast realizability of the components of the hardware and ease of modification by exchanging the software components. Disadvantages are the relatively high system costs in high volume, low performance, and non-optimal adaptatation to the

problem. The second approach divides into special processors with microprogrammed control and into optimized processors. The first case has the advantage of high systems performance, but with expensive design and lack of adaptability. The second approach (ASICS) can show optimized functionality and hardwired control (RISC) or microprogrammed control (CISC) and software components optimized for the task. Here, high systems performance can be obtained with well-fitted functionality and flexibility for modification. The design of the processor and the definition of the software/ hardware-interface turns out to be expensive. The present system is meant to help with this problem. The project grew out from the necessity of integrating connection networks in multiprocessor systems, similar to [DGKL87], but for a crossbar conneted one. As computers with ever higher performance need special hardware elements, e. g. [ES90], they become then application oriented systems with less usability in other areas.

The effort for the development of solutions to problems in the architecture of application specific integrated circuits (ASICS) increases considerably with the complexity of the functions to be implemented. A method for the design, development, and test is needed with CAD-tools for the support. A preliminary investigation of the requirements has shown that a combined architecture of components in software and in hardware has to be supported for an efficient realization by integrated systems.

A model for integrated specification was created for the specification of such software/hardware systems in contrast to other approaches to this problem. It applies parallel and communicating hierarchical processes for the description on different levels of specification with automatic and semi-automatic transformations between them.

2. Design of Integrated Software/Hardware Systems.

2.1. Available systems and requirements.

Several CAD-software packages are available for different aspects of the hardware design tasks, e.g. placement and routing [BLM86]. Cell generators permit creation of parametrized circuits elements as registers, memories, and functional units [Bur88]. PLA and control circuit generators allow automatic realization of even complicated control circuitry [e.g. MLB88].

The emphasis in systems design has moved so from the design of the layout to the design of the systems. The design of the architecture requires still the highest effort for the chip functions to be implemented [Sch89].

Two existing systems for the synthesis give a first approach to the automation of the architectural design phase [CR89, Sch89]. Another system permits the definition of a hardware structure for the execution of ADA programs [JRS89].

These systems show, however, a relative inflexibility in regard to the architecture and cannot deliver the quality of a hand design for complicated circuitry [Mar89]. Interactive control of the synthesis is needed for much better integration and the applicability of the architectural synthesis in extended designs.

The software portion of a complicated system presents another problem. The implementation of extensive tasks purely in hardware turns out most often to be rather inefficient. The reason is that the requirements for hardware and especially for systems control grow immensely, and the components of the hardware cannot be utilized sufficiently. A hierarchical implementation of the control circuitry offers much better efficiency and structure. The highest levels of the hierarchy should reside as control programs in a memory. Structures of processors and control circuits appear so with an optimal instruction code of the solution of the problem, and optimal distribution of the software and hardware components [HR88].

Demands increase for the development system in the design of combined software/hardware systems. So, methods are required from software engineering.

The possibility of behavioral descriptions offers high potential for the design, development, and test of integrated circuits, using programming languages.

2.2. Solution proposition.

A systematic transfomation of the behavioral description into an architectural description serves as the basis for the proposed solution here under consideration of the division into software and hardware components. The user will have a hierarchy of algorithmic and rule-based design tool modules for the analysis, transformation, and synthesis of designs. The graphical design environment supports the user at the definition of the behavioral description of the design. It contains an integrated synthesis systems module for the development of the hardware architecture from different hierarchical levels, see Figure 1.

3. Design Steps.

Several steps can be distinguished during the design of a complicated software/hardware system: definition of the functionality of the total system, observation of realtime constraints, division of the funtions into software and hardware components, definition of the software/hardware interface, design of the hardware, firmware, and software components, and system integration and test. Figures 2 and 3 show two types of developmental venue: development with prototype verification, and development with architectural simulation.

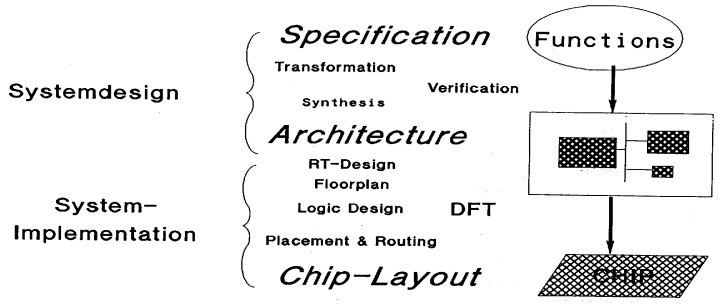

Figure 1: The developmental task.

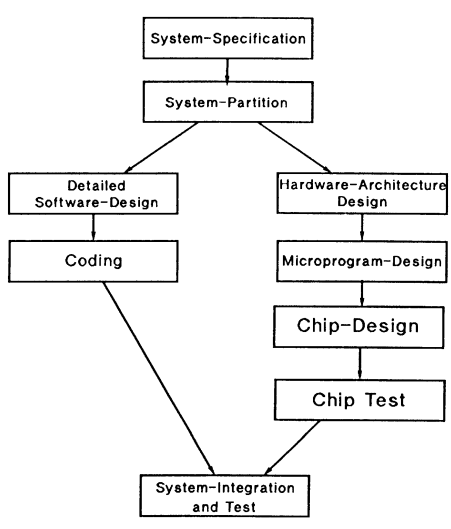

Figure 2: Systems development with prototype verification.

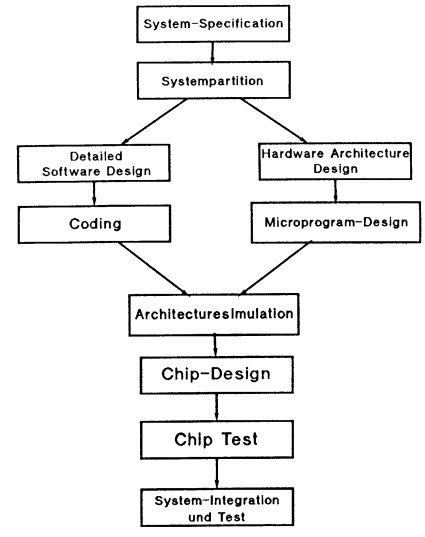

Figure 3: Steps for systems development with architecture simulation

The design of complicated systems forces the designer into a structured approach with hierarchies and modularity, at best with a meet-in-the-middle approach [DLM88], followed by verification and validation. Different phases can be distinguished: study, specification, system design, implementation, integration and test, and production. Iterative cycles appear for all of these.

The present development system has at its heart a database in a specially designed language SIL for a design. SIL is an intermediate system level description language, comparable to controlflow/dataflow languages in other high level synthesis systems, but with high graphical capabilities. The user can define initially a system in several ways: by a functional or a structural description with a graphical PMS notation; by the RT-SA/SD method on the register transfer level; or by DACAPO-II [DAC85]; or C-language algorithms, and by combinations of these. All definitions are then compiled to SIL, see Fig. 4.

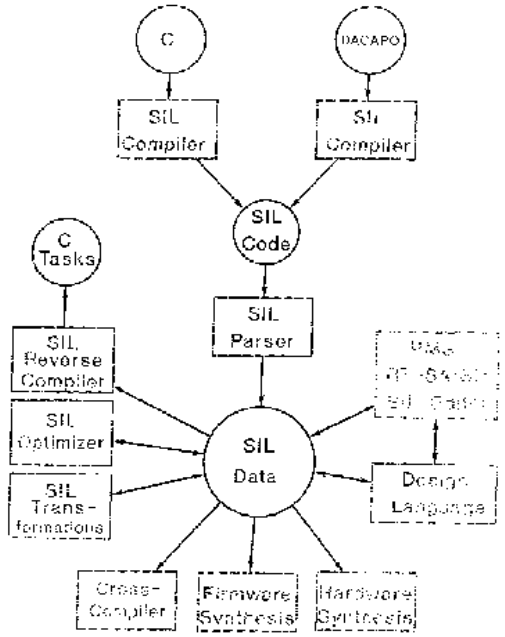

Figure 4: The system design level structure

A design in the SIL database can be optimized or parallelized by transformations as to the control and dataflow by tools as individual modules. Also the design can be transformed from here to the architectural level by translating the SIL description to a software assembler [Bur77]), to firmware, and to hardware by synthesis tool modules. The design can also be manipulated and modified on the architectural level.

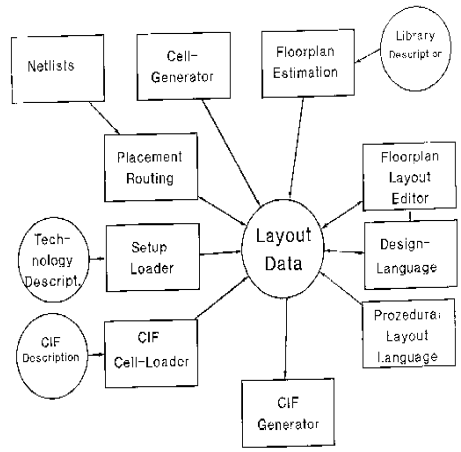

Figure 5: Tools on the architectural level.

Once a design is complete, it can be transformed to the layout level by a generator. Additional modifications can be applied here again, see Fig. 6.

As the design process works iteratively, the different levels can be followed automatically and a better adaptation to the requested tasks achieved.

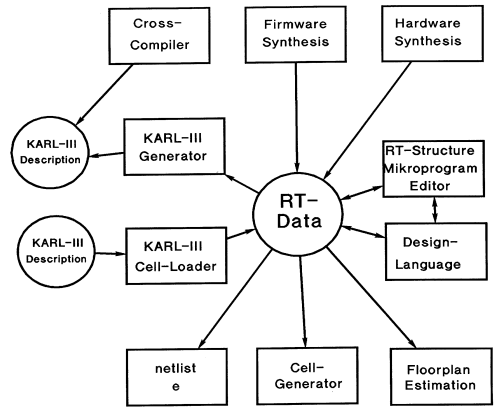

Figure 6: Tools on the layout level.

4. The Concept of the Design Method.

The requirement for explorative designs of integrated software/hardware-systems is supported by three application methods: interactive, guided, or automated methods. Once a design has taken shape, it can be modified interactively with the creation of elements on the design level for concern. Tools exist for the analysis, transformation and synthesis of elements, and the results are processed for graphical representation. The combination of the support functions permits automatic generation of elements for specific applications. The global flow of control can be defined by a rule-based program.

Different degrees of automation are thus achieveable. The simulator generator module can produce automatically a model for the KARL-II simulator. The expertise in the expert module for the design is not predefined rigidly, which would then be restricted, but can be formulated by the designer. The expertise can be stored, documented, and transferred to other designs. Expert knowledge from other designs can be called upon within the system. An adaptable concept for the automation of systems designs is obtained for different requirements. For the stucture of the design system, see Fig. 7.

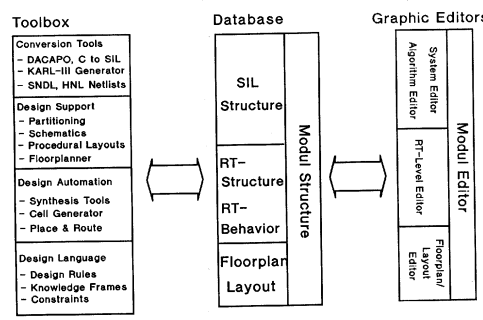

Fig. 7: Structure of the development system.

4.1 Design Levels.

Several levels can be employed in the design of complex systems: the systems or behavioral level for the specification of a software/hardware-system; the architectural level for the design of the systems modules; the floorplan/ layout level for the estimation for the floorplan and the detailed chip design. A design can be verified as to its performance by the automatic generation of simulation models for DACAPO-II (behavioral level) [DAC85], KARL-III (RT-level) [HLW86], and RELAX (layout level).

5. Implementation of the System.

The system has been implemented in a hierarchy of modules that can be called-up from an overall mon-itor. The idea here is that additions, changes and modifications to the system can be made any time, especially with respect to further automation of the design process, without affecting the operability of the system.

6. Sample Application: Sorterchip.

The design of a sorterchip for picture processing was chosen for a model application. The following aspects were of special importance during this development:

1. The development and presentation of the transformations for the specifications development and planning of the system test on the behavioral level. A powerful algorithm was constructed with these transformations that is especially fitted for a chip implementation, due to the regular structure of the sorter,

2. The application of the editors for the interactive graphical definition of the specifications;

3. Representation of different software/hardware implementations and their statistical evaluation, including estimation of the floorplan requirements;

4. Application of the optimizer and synthesis tools for a processor element and omparison to the hand design;

5. Interactive design support for the architectural and logic design and the verification by the architecture simulators;

6. Application of a rule-based design language for a specific user appliplication and automation on the architectural level.

6.1 Initial Specification.

A problem-oriented definition in C of the sorting algorithm did serve as the starting point for a hand design as the same basis as for the development system. The transformations simplified the complex expressions with the addition of new variables. Backtracking had to be used by the optimizer for simplification for an implementation by counters and MS-registers. For-loops offer two possibilities for parallizing: loops without data dependencies are unrolled, and the assignments of some loop positions put into a procedure on a lower level which permits sliceable structures.

6.2 Statistical evaluation and selection of the architecture.

The statistical evaluation of the behavioral specification results in information for the selection of the target architecture. The execution time for a solution in software came to 180 us for sorting, or for a pixel rate of only 6 kHz. Required was a pixel rate of 4 MHz, or a speedup factor needed of $S_{hw}/S_{sw} = 700$. The technological speedup factor S_{tech} for the transition from software to hardware runs up to 2us/20ns = 100, because an assignment in the behavioral description needs about 2 us, but on the Gate Forest level about 20 ns. This means, a parallization factor of at least 7 is required. Possible parallelization is solely limited by the available chip area. The floorplan estimator tool module has allocated some 30% of chip area for the control logic, the registers and drivers. One processor element uses about 10% of the area, so the 7 processor elements became feasible. The time analyzer module for the different sorter functions in use gave no specific peak, all of them are in the 20%-30% range of the total time requirement. This means that all sorting functions should be implemented as hardware elements, no firmware or software used [BM78].

6.3 Evaluation of different implementations.

The pure software solution needs no additional chip area, but requires 180 us for sorting. The use of one sorter slice reduces the sorting time to 28 us, but needs 4 mm^2 chip area. Seven sort slices can be combined unto one Gate Forest chip with 28 mm^2 area and 4 us for sorting time. An implementation of the sorter as a coprocessor reduces the time requirement to .25us, but needs 40 mm square chip area, see Figure 8. The execution time requirements T by the area occupied A follow a falling log-function with 99.97% correlation: $T = 50.25 + 14.04 * \ln A$, see Fig. 9.

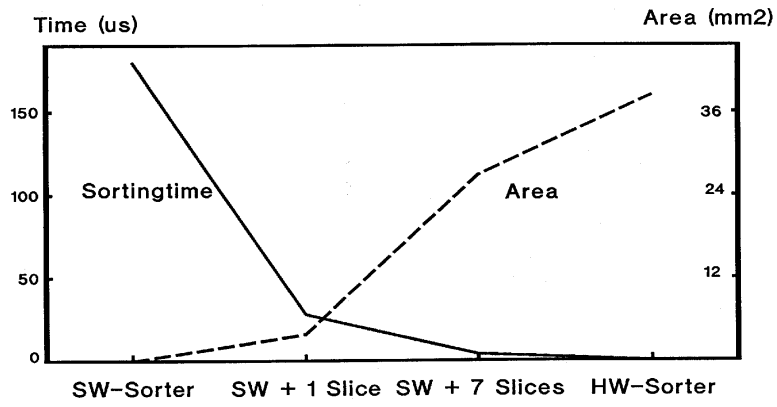

Fig. 8: Trade-off Hardwre/Software.

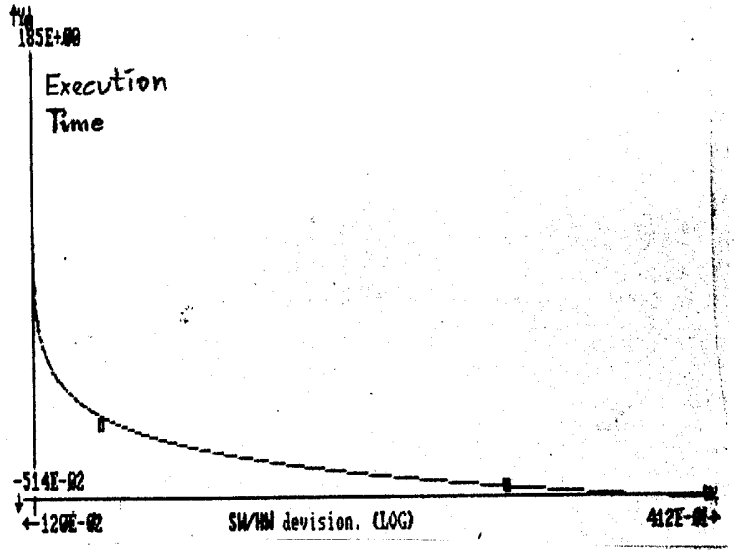

Fig. 9: Execution time by chip area.

The speedup S increases exponenially as a function of the area A with correlation of 96.82% by the function: S = 1.66 * exp(0.14 * A), see Fig. 10. This means, that a trade-off in chip area has the capability of an exponential increase in computational power.

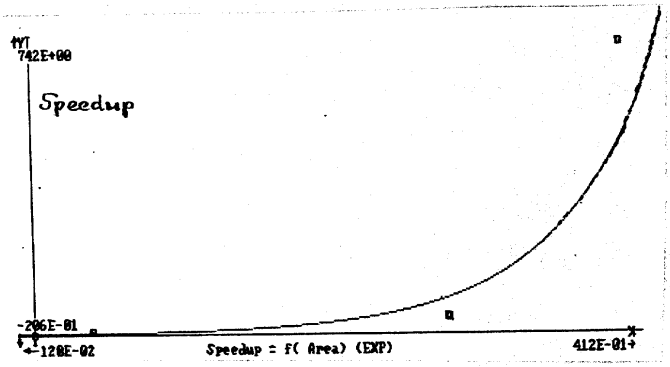

Fig. 10: Speedup by chip area

6.4 Optimizer Performance.

The optimizing tool module from the system has reduced the number of the sorter elements from originally 970 to 540. This gives an improvement of 44%.

6.5 Development time comparison.

The design of the sorter architecture by hand had required 17 man weeks. But with this development system, it was reduced to 6 man weeks, or the significant improvement of 65% with no discernible decrease in the performance of the obtained product.

6.6 Chip implementation.

The above design of the sorter chip was manufactured unchanged in the ACMOS 2 Gate Forest technology [BHK88]. The design uses 18000 active ransistors, enclosed in a pingrid case, and clocks at 5 MHz, see Fig. 11. It operates defectfree and without any flaws or problems.

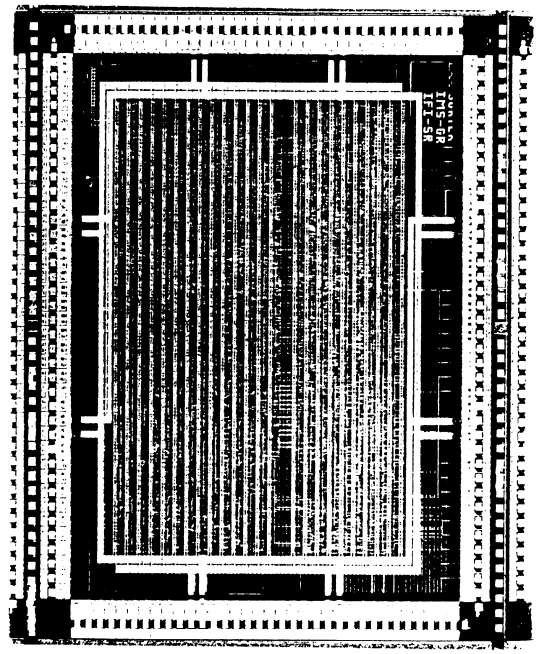

Fig.11: Sorter-Chip.

Bibliography.

[Amm87] P. Ammon: ASIC - Was, wie, wann. Elektronik, Nr. 3, p. 141,(1987).

[Axf89] T. Axford: Concurrent Programming Wiley 1989.

[BHK88] M. Beunder, B. Hoefflinger, and J. Kernhof: New directions in semi-custom arrays. IEEE J. of Solid-State Circuits, Vol 23, pp. 728-735,(1988).

[BLM86] U. G. Baitinger, et. al.: Das Forschungs- und Entwicklungsprojekt MEGA. Elektronik, Nr. 2, p. 61-64, December 1986.

[BM78] W. H. Burkhardt and H. E. Maier: MICOS: A microprogrammed hierarchical operating system nucleus and its performance comparison. Proc. 11th Annual Microprogramming Workshop 1978, p. 33.

[Bur77] W. H. Burkhardt: Universal Micocomputer Systems Software. Proc. Fall Compcon 1977, p. 209-211.

[Bur88] M. R. Burich: Design of Module Generators and Silicon Compilers. In Silicon Compilation, North Holland, 1988.

[CR98] R. Camposano and W. Rosenstiel: Synthesising Circuits from behavioral Descriptions. IEEE Trans. on Com-puter-Aid-Design, Vol. CAD-8,(2,1989), 2.

[DAC85] DACAPO-II, Version 3.0, User Manual, Dosis GmbH, Dortmund, 1985.

[DGK87] J. Dickey, A. Gottlieb, R. Kenner and Y-S.Liu: Designing VLSI network nodes to reduce memory traffic in a shared memory parallel computer. Circuits, Systems and Sig-nal Processing 6, 217-38 (1987).

[DLM88] P. Dutzy et. al.: Vom Handentwurf zur Struktursynthese. Elektronik, VOl. 12, (6,1988), 114.

[ES90] Evans & Sutherland: "ESV - Series", 1990.

[HLW86] R. W. Hartenstein et. al.: KARL-III Language Reference Manual, Universitõt Kaiserslautern, Fachbereich Infor-matik, 1986.

[HNS86] E. Hoerbst et. al.: VENUS: Entwurf von VLSI-Schaltungen, Springer 1986.

[Hoe78] B Hoefflinger (Ed.): Großintegration, Oldenbourg 1978.

[HR88] R. W. Hartenstein and W. Ryba: Parti-tionisierungsschemata für Rechnerstrukturen. In Design Methodologies für VLSI and Computer Architectures North Hol-land 1988.

[RS89] JRS: Integrated Design Automation System (IDAS) SIG-MICRO 21, (1989), 11-17.

[Jun88] T. Juntunen: Real-Time Structured Analysis in System Level Design of Embedded ASICs. In Microprocessing and Microprogramming, 24,(1988), 449-454.

[Mar89] P. Marwedel: Improving the Performance of High-Level Synthesis. In Microprocessing and Microprogramming, 27,(1989), 381-387.

[MLB88) H. Mahler et. al.: Processor Control Part Synthesis Using Effective Partitioning Algorithms. In Microprocessing and Microprogramming, 23,(1988).

[Sch89] D. Schmid: Systemsynthese. In Proc. Tagung Mikroelektronik Stuttgart 1989 p. 9-15.