Realization of a Programmable Parallel DSP for High Performance Image Processing Applications

Jens Peter Wittenburg   Willm Hinrichs   Johannes Kneip   Martin Ohmacht
Mladen Bereković   Hanno Lieske   Helge Kloos   Peter Pirsch

Laboratorium für Informationstechnologie, Universität Hannover
Schneiderberg 32
30167 Hannover, Germany
Tel: +49 511 762 5018
E-mail: wittenburg@mst.uni-hannover.de

ABSTRACT

Architecture and design of the HiPAR-DSP, a SIMD controlled signal processor with parallel data paths, VLIW and novel memory design. The processor architecture is derived from an analysis of the target algorithms and specified in VHDL on register transfer level. A team of more than 20 graduate students covered the whole design process, including the synthesizable VHDL description, synthesis, routing and backannotation as the development of a complete software development environment. The 175mm^2, 0.5μm 3LM CMOS design with 1.2 million transistors operates at 80 MHz and achieves a sustained performance of more than 600 million arithmetic operations.

1. INTRODUCTION

The ongoing trend towards growing complexity in communication and image processing systems calls for the development of flexible signal processors. Monolithic architectures capable to process a wide range of algorithms and a performance sufficient to meet real time conditions will form the core of future multimedia platforms. An ideal processor platform has to meet a number of sometimes contradicting requirements: For a maximum of efficiency, the design has to be adapted to the target algorithms, should be generic to allow the porting of a high level language compiler - mandatory for the minimization of implementation cost of complex applications. Parallelization potential of the algorithms should be exploited as far as possible to boost performance, but keeping hardware cost as low as possible. Some fundamental problems, as the high bandwidth parallel access to shared memory segments, are not yet satisfactorily solved in currently available DSP architectures.

The target of the design project described in this paper is twofold: First, this design is an innovative processor architecture, proving that a parallel, homogenous DSP architecture can achieve a processing power hitherto addressed only by dedicated designs. This was achieved by an appropriate combination of measures known from standard processors with a novel controlling and memory concept. Second, it served as an ambitious tutorial project for graduate students, which worked on the project for 6-12 months during their final thesis in microelectronics or information technology. During the design process, all aspects of VLSI design from architectural specification, HDL description, synthesis, verification, routing and backannotation were actively performed by the students, making the project a highly valuable contribution towards a more practically oriented university education.

In this paper, we provide an overview of the architecture and the design it has undergone during the four year lasting realization phase. We start with a brief description of the architecture, it's specialities and the basic considerations that lead to it. Section 3. describes the HDL design on RTL and the synthesis on an 0.5μm 3 layer metal CMOS standard cell technology. Section 4. covers verification aspects while section 5. describes the backend design process and the problems students had to cope with during routing, backannotation and final verification of a complex sub-μ design. Section 6. gives an overview of the HiPAR-DSP's software development environment, also developed within the project. Finally, we present the layout and performance data for a selected number of algorithms for the processor, which is currently manufactured.

2. ARCHITECTURE

Fig. 1 shows a block diagram of the HiPAR-DSP architecture. The homogenous core consists of four (in a future version 16) identical 16 bit data paths, each containing a local register file, an ALU, a shift/round unit and a 36 bit multiply/accumulate unit. The data paths are centrally controlled by a RISC control unit, operating on a 96 bit very long instruction word (VLIW), which is read from an instruction cache assigned to the control unit. A two stage memory hierarchy was implemented. First, data paths have concurrent access to shared data in shape of a matrix (Fig. 2), which are stored in an on-chip memory, the matrix memory. Second, each data path may address private data autonomously, which is stored in a cache memory assigned to each data path. An autonomously operating DMA control unit serves all cache misses and may perform prefetch data transfers to the matrix memory. A test and debug interface is used for the *in system* software debugging by accessing all internal registers via the fabric scan chains and test access port, while the processor is halted.

The HiPAR-DSP architecture meets the special requirements of image processing algorithms implemented in parallel. By combining instruction word level parallelism (VLIW), data parallelism (multiple data paths, split ALU capability) and transfer parallelism (autonomous DMA unit), a high degree of concurrent processing can be achieved for various algorithms [1][2]. The SIMD controlling principle leads to low hardware implementation cost compared with MIMD or multithreaded approaches. The required flexibility for more complex algorithms is achieved by the implementation of a three stage autonomous controlling scheme for the data paths, providing as addressing and execution autonomy as the concurrent

35th Design Automation Conference ®
Copyright ©1998 ACM
1-58113-049-x-98/0006/$3.50   DAC98 - 06/98 San Francisco, CA USA

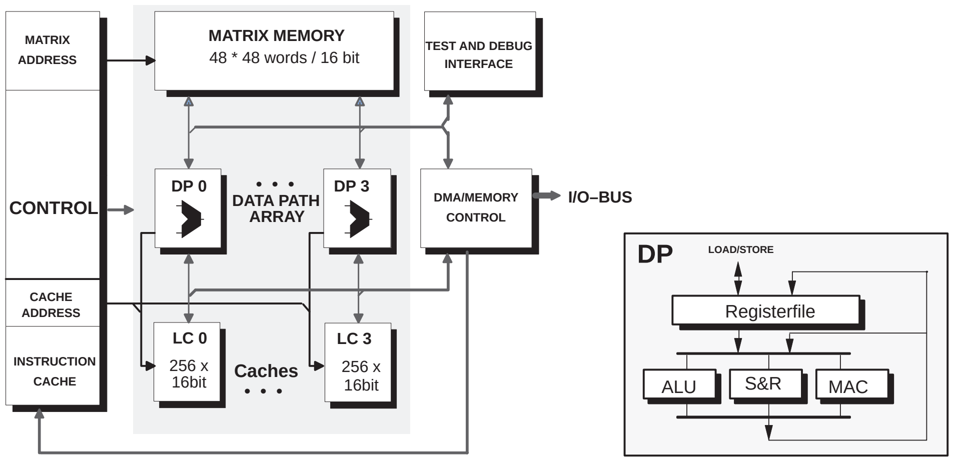

Figure 1: HiPAR-DSP architecture and structure of a data path (S&R: Shift-and-Round unit)

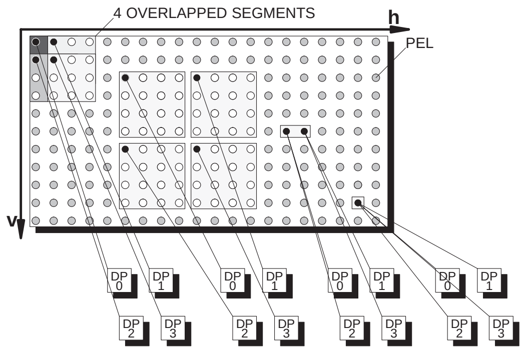

Figure 2: Concurrent access of data paths to the matrix memory. The matrix memory has a virtual 2D address space and provides a conflict free access to the data stored in it.

evaluation of local conditions.

The requirement of a high bandwidth access to shared data is solved by the matrix memory architecture. Internally, the data are distributed to nine single ported memory banks by use of a distribution function that provides a conflict free access to data from any address of it's virtual 2D address space [3]. The generic structure of the processor and the implementation of a stack mechanism made the porting of the GNU C++ compiler feasible. The architecture was derived from a thorough analysis of the targeted image processing algorithms. Available high-level design tools focus on mainly dedicated realizations, but were not of much help in the design of a generic programmable architecture. So, our experience in the parallel implementation of image processing application and in VLSI design were the driving forces leading to the final structure of the HiPAR-DSP.

3. FRONTEND

VHDL was chosen as hardware description language for design entry and verification at RT-Level. On one hand, technically the preference of VHDL to Verilog turned out to be slightly disadvantageous, because of nonexisting VHDL (VITAL) gate level models of the standard cells, which forced us to employ different simulators for functional and gate-level sign-off simulations. On the other hand, in Europe the number of commercial VHDL installation exceeds the number of Verilog installations by far, which justifies the preference of VHDL at European universities.

For designs with a complexity of more than several hundred thousand gates complexity a hierarchical design approach becomes absolutely mandatory. The SYNOPSYS DESIGN COMPILER used for RT-Level synthesis and timing optimization in this project is hardly able to optimize more than 5 to 10 thousand gates at a time, depending on regularity. On physical level, the routing tools enable efficient routing of blocks up to about 30 thousand gates without losses in gate density for the chosen 0.5 μm 3 LM standard cell library. Thus, logic- and physical hierarchy are not necessarily identical. Nevertheless, it is reasonable to take the physical level into account when defining the logic hierarchy, e.g., VHDL-modules to be designed independently. Conversely, logic hierarchies usually provide a certain degree of locality for interconnections, which lead to better routing results and lower delays caused by wire capacitances on the physical level.

Partitioning started from an initial logic hierarchy derived from the processor's block diagram and a black box floorplan based on course estimations of the silicon effort. After first synthesis steps with much more accurate area calculation, we were able to derive a logic partitioning of the processor. All major parts of the architecture were designed independently, while none of the resulting blocks in the floorplan exceeded the size of 30.000 gates. The subunits: data path, controller, data cache controller, instruction cache controller and DMA were VHDL-designed, synthesized and verified as independent design projects by graduate students. A project management of two to three research assistants was responsible for the supervision of design constraints (clock speed and area) and the definition and specification of the interfaces.

Due to the pipelined nature of the architecture, we were able to achieve a clock frequency of about 100 MHz from initial synthesis, with wiring capacities calculated from a heuristic wire load model delivered with the standard cell library. The arithmetic units, especially 32 bit adders and the 17x17 bit multiply & accumulate (MAC) unit required special treatment, since simple synthesized versions of the adders (riple carry) and multipliers (brown array) did not fulfill the timing requirements. Following two measures to increase arithmetic performance were investigated first:

- Instantiation of advanced hierarchical architectures (fast carry lookahead in case of the adders, booth coded wallace tree for the multipliers) from the SYNOPSYS DESIGN WARE libraries.

- Generation of Arithmetic Macro Cells using the COMPASS data path compiler, which allows to generate layout optimized cells from bit slice descriptions.

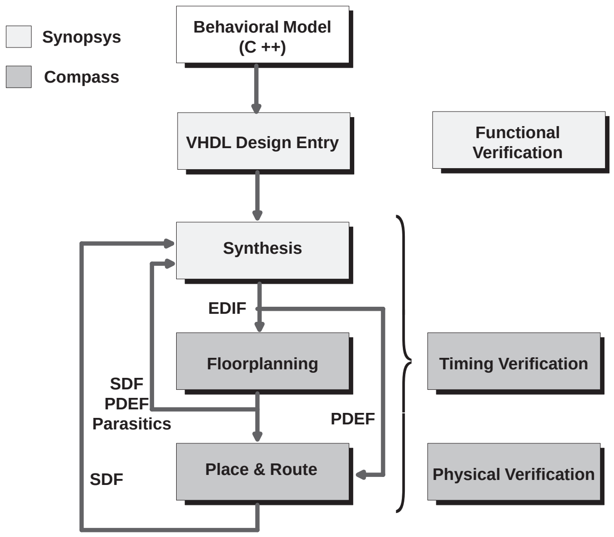

Figure 3: Simplified design flow and interfaces. SYNOPSYS tools (VSS, DESIGN COMPILER) were used for all frontend tasks, while COMPASS tools (PATHFINDER, CHIP TIME, QSIM) were employed for all backend activities [5][6].

Both methods lack the flexibility to insert pipeline stages at user defined places. This leads to inefficient realizations, where a complete second adder has to be introduced in case of the multiply/accumulator To solve these problems we designed own advanced arithmetic architectures by structural VHDL description. Instead of elements from the standard cell library we used generic cells from the SYNOPSYS GTECH library. Then we used DESIGN COMPILER to map and optimize the modules, which allowed us to keep the designflow consistent. This approach not only kept the required flexibility but surprisingly also by far outperformed all DESIGN WARE instantiations for multiplier, 32 bit adders and barrel shifters in terms of effciency (A * T product).

4. VERIFICATION

The processor design started with a high level behavioral model written in C ++. This model served as cycle precise simulator with the possibility to simulate complete algorithms. In addition it enabled access to all logical registers and memories. C ++ has been chosen as modeling language due to the high performance and portability. Because the abstraction level of this model was much too high for direct breakdown to RTL level by refinement, HDL-modeling would not have been of advantage anyway. The behavioral model has been extensively verified against the specification. Graphical user interface and efficient interfaces to object code enabled simulation of several image- and signal processing algorithms (e.g., complex FFT of variable length, grey level histogramming, hough transform) with realistic input data. Additionally synthetic command sequences were generated to verify rarely used special cases. Thus the verification coverage of the behavioral model could be judged as high enough to classify the model as "known good device" for all subsequent verification steps.

Functional verification of the RT-level VHDL-code took place using SYNOPSY's VSS VHDL simulator. Verification strategy followed the typical bottom up approach, starting in the lowest levels of the hierarchy and ending at top level with simulation of the complete architecture using the same stimuli already employed for verification of the behavioral model. The existence of this known good reference model significantly simplified verification on all levels of hierarchy.

As already mentioned, sign-off verification of the employed library using VHDL is impossible due to the lack of suitable standard cell models. COMPASS' proprietary QSIM is the alternative. Unfortunately QSIM does not support flexible testbenches and therefore does not allow text-I/O as we used for functional verification. To solve this problem we inserted certain VHDL tasks dumping logic levels at I/O interfaces (of complete design or submodule) for each executed clock cycle into a file. Such interface dumps representing stimuli and expected results can be converted to a format importable to COMPASS. Even though this seems to be a standard problem, we had to write our own conversion utilities (C++) for this purpose. Another challenge during sign-off simulation was the propagation of unknown logic levels due to uninitialized registers with feedback. Because of incompletely modeled standard cells (a multiplexer always propagating unknown when its select input is undefined - even if all inputs are equal - is the most common case) it can become impossible to initialize such registers. Controlled by attributes synthesis of a working synchronous reset can be forced for problematic registers. However, uninitializable registers can (and usually do) occur after each resynthesis, since implementing resets to all registers would cause non neglectable overhead. The major amount of work during sign-off verification related to those uninitializable registers.

After sign off simulation the final netlist had been fully verified against the behavioral description. This verification also covers all timing information including the asynchronous interfaces of the on chip memories, which represent the only timing of the design not fully synchronous with respect to the master clock.

5. BACKEND

The backend tasks were performed using the CAD-tools of COMPASS DESIGN AUTOMATION. Main feature of the tool is the tight interaction with PASSPORT standard cell libraries also developed by COMPASS and qualified by the ASIC vendor. The netlists generated by the synthesis tool SYNOPSYS were transferred via an EDIF interface to the COMPASS tools. All on-chip RAMs were generated by COMPASS RAM compilers. The resulting macro cells as well as boundary scan and pad cells were added in COMPASS schematic editor to complete the netlist for the HiPAR-DSP. Scan logic for test purposes was automatically inserted by the COMPASS tools following a two step approach. First all D-Flipflops were exchanged by scanable counterparts. The cell order in the scan paths had not defined at that point. This task was performed later after cell placement but before routing. This two step approach minimizes the routing overhead of scan logic significantly.

The final layout of the HiPAR-DSP is shown in Fig. 4. On-chip RAM cells for the matrix memory, data cache and instruction cache are placed close to their corresponding controllers. Each memory is served by the DMA/memory controller. Main silicon area is occupied by the global RISC controller including a base register file and the four data paths with local register files. Due to the wide internal data and address busses of the load/store architecture between data paths, matrix memory and data caches, a large vertical routing channel runs through the chip. The power pads are evenly distributed in the pad ring to handle simultaneously switching outputs. 20 power pad pairs are provided for core supply and six AC/DC pads for padring power.

Placement & route of the HiPAR-DSP were carried out using the COMPASS PATHFINDER tool. Starting from the final floorplan, placement was controlled by high-level timing constraints (e.g., targeted clock frequency). Therefore the placement tool has to employ its own static timing analysis engine. Clock tree compilation takes place after the placement has been completed. Since the position of the registers is not finally fixed at an earlier level, this ensures the best possible delay estimation to achieve a prop-

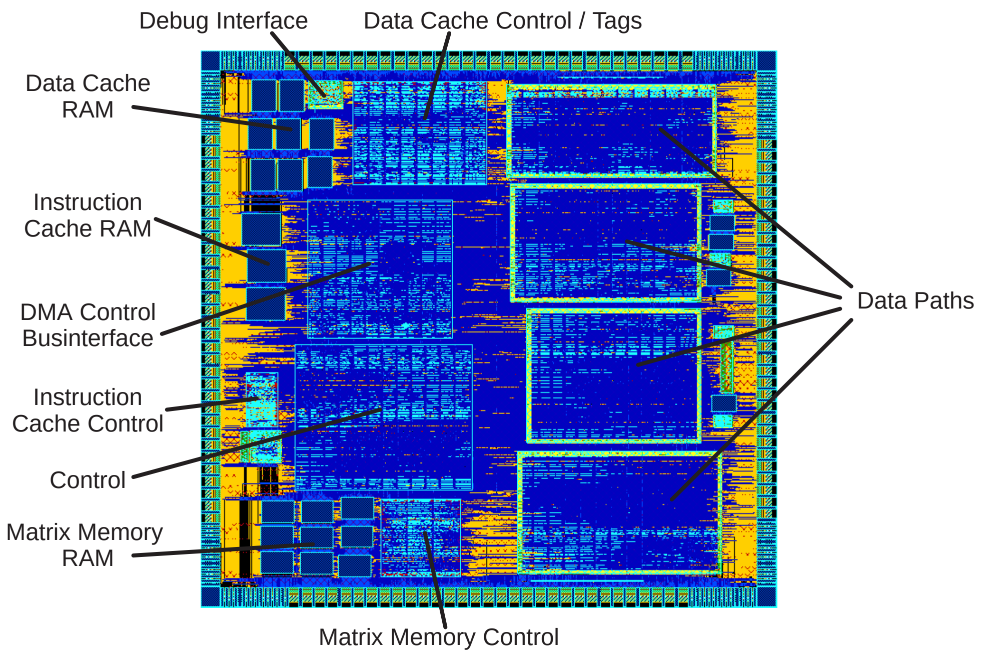

Figure 4: Layout of the HiPAR-DSP

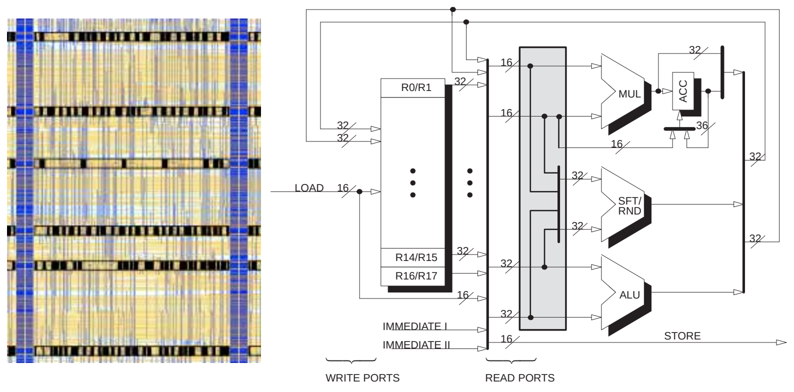

Figure 5: **a)** Typical segment of registerfile layout. Only the narrow horizontal strips represent active cell areas

b) Schematic overview of data paths using less register file ports. The number of register file read ports has been limited to two 32bit- and three 16bit read ports.

erly balanced clock tree. After clock tree compilation and 2nd step of scan path insertion, the bin based PATHFINDER router is invoked to connect the placed cells. A completely flat routing approach showed best results in terms of gate density, since global optimization saved block wiring overhead. Routing of the DSP required about one week CPU-effort on a 200 MHz Ultra SPARC Workstation equipped with 1 GByte main memory.

The RISC-typical registerfiles were one of the most critical elements concerning die size and clock frequency [4]. Especially the number of the register file ports necessary to distribute data according to the VLIW concept of the data paths, caused large hardware costs. Full custom optimized blocks of any kind were not available. The enormous amount of wiring between the standard cell rows of the data path layout is shown in Fig. 5a. In a straight forward implementation register file ports according to in- and output of arithmetic/logic- and I/O-units would result in five 32bit and three 16bit read ports respectively two 32bit and one 16bit write ports. Achieving acceptable register file size and speed while not drastically decreasing performance required careful evaluation of each possible measure. A proper balance had to be found between software requirements, performance and silicon area. The result is shown in Fig. 5b. The general purpose character has been kept with the exception that the two16bit operands of the MAC must not be both even or odd. Some further combinations of concurrent operations are no longer possible (e.g., ALU operation and 32bit Shift) but are rarely found in the targeted applications anyway. VLIW restrictions can be easily handled by the compiler. The improvements in size and speed for the implemented register file are shown in Table 1.

Even though floorplanning and thus estimation of wiring delays has been carried out from the beginning of the frontend design, achievement of timing convergence turned out to be the most challanging part of backend design. Several steps of resynthesis had to be carried out in addition to the timing driven cell placement:

Table 1: Size and speed of port unlimited and implemented register file based on $0.5\mu m$ 3LM CMOS. The average gate density for typical standard cell design is about 4000 gates / mm^2

number of ports	**area per data path**	**gate density**	**max clock speed**
2 32bit + 1 16bit WRITE 5 32bit + 3 16bit READ	$35mm^2$	910 $\frac{gates}{mm^2}$	55 MHz
2 32bit + 1 16bit WRITE 2 32bit + 3 16bit READ	$17mm^2$	1600 $\frac{gates}{mm^2}$	80 MHz

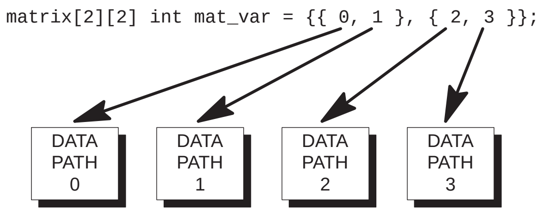

Figure 6: Matrix type variable definition and distribution to data paths

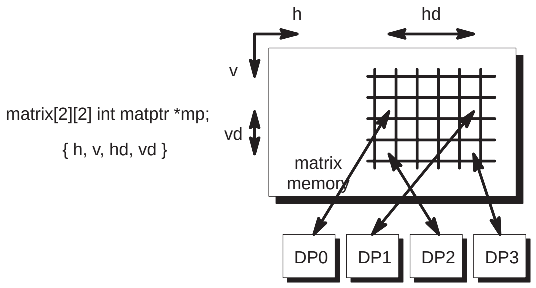

Figure 7: Access to matrix memory

- Resynthesis using custom wire load models generated from the estimated floorplan
- Reoptimization (rebuffering) of the design using delay information backannotated from an initial placement. Therefore clustering information (PDEF), annotated timing (SDF) and parasitics (RC-Delays) have to be backannotated into the SYNOPSYS DESIGN COMPILER. The synthesis tools then start from the annotated timing and uses clustering and parasitic information, where nets have to be changed. Due to the different hierarchy in the SYNOPSYS domain (e.g. missing memories, pads, scan paths) and differing naming conventions between the tools, backannotation across the interface between COMPASS and SYNOPSYS is no trivial task.
- In place optimization again using backannotated delay data. For final placement, positions of inserted buffer trees were forward annotated to the placement tool, which is able to place such buffers without having to discard the existing placement.

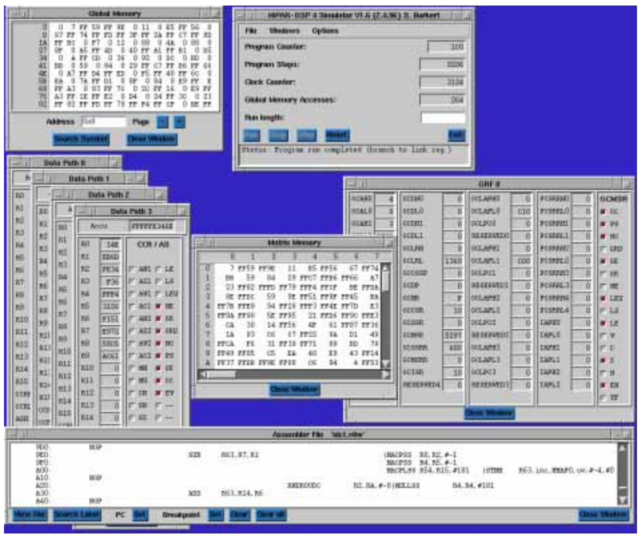

Figure 8: Screen shot of the interactive simulation environment

6. SOFTWARE DEVELOPMENT ENVIRONMENT

Starting with the first specification phase of the HiPAR-DSP, concepts for a high level language development environment have been created. Resulting from this tight cooperation, the HiPAR-DSP architecture provides some features, e.g., efficient operations on a parallel, cache buffered stack, supporting high level programmability for imperative languages like C++ or PASCAL. During further specification phases and the beginning of the implementation phase, we developed an optimizing assembler and adapted a C++ Compiler for the HiPAR-DSP.

The compilation process consists of three phases. The frontend of the development systems consists of an extended port of the GNU-C++-Compiler, which supports parallel operations and matrix memory access using language extensions. The scalar variable types like int, long, etc. have been extended to compound, matrix types shown in Fig. 67, consisting of a set of scalar variables ordered in shape of a matrix. Expressions using variables of this type result in parallel operation of all data paths on different components of this type. This allows existing C or C++ code to be easily ported to the HiPAR-DSP by simply adapting variable types in time critical code sections to the new features of the system.

During the second phase of the compilation process, an optimizing assembler automatically generates VLIW assembler instructions from the sequential code exploiting the VLIW parallelism and the micropipelines, using advanced inter basic block scheduling and software pipelining. In the last phase, machine readable binary code is generated.

The correct behavior of programs developed for the HiPAR-DSP can be verified using an interactive simulator, which is shown in Fig 8. The simulator operates clock cycle based and allows the user to inspect and modify all programmable features of the HiPAR-DSP architecture. Besides standalone debugging, in-circuit debugging using the JTAG interface is supported, allowing rapid application development and testing in the target environment.

The compiler, assembler and debugger, combined with the so far developed application library form a powerful development platform that enables the user to create applications efficiently in a way he is used from industrial design.

Table 2: Performance data for selected algorithms. Data from cycle true simulation of assembler coded programs, cycle time from backannotated layout data.

ALGORITHM	**DESCRIPTION**	**PROCESSING TIME** ($f_{clock} = 80MHz$)
Normalized Correlation	512 x 512 input image size 32 x 32 mask size	2.25 s
Gray level Histogram	512 x 512 image size, 256 gray level	2.75 ms
complex FFT	256 x 2 samples (r,i)	35 μs
3x3 FIR-Filter	non-separable, 512x512 pixels	21.4 ms
Edge Detection	512 x 512 pixels, vertical and horizontal sobel filters and non maximum suppression	11.5 ms
Connected Component Labeling	512 x 512 pixels	25.7 ms
Skeletonization	512 x 512 pixels	21.2 ms
Parallel Search	1024 elements	625 ns

Table 3: Implementation data of the HiPAR-DSP

Parallel Data Paths	4 parallel data paths (13/32 bit integer)
On-chip memories	9*0.5 kbyte matrix-memory, 4*0.5kbyte data cache, 3kbyte instruction cache (Σ:9.5kbyte)
Technology	Atmel 0.5 μm CMOS 3LM standard cell with full custom memories
Transistors	1.2 million
Area	175 mm^2
Power consumption	$\sim$ 5 watt (estimated)
Clock frequency	80 MHz (typically), 50 MHz (worst case)

7. DESIGN RESULTS AND PERFORMANCE DATA

The design process started in 1993 with first architectural considerations. The project was intended to last 3 years until silicon. With specification completed end of 1993, the HDL design lasted until end of 1994. A lot of time was lost due to two unexpected changes in manufacturer and technology, partly requiring overworking of large sections of the HDL code (which is not so manufacturer-independent as one might believe, if efficient synthesis has to be achieved). A high effort of time went into the setup of a working back end design flow, which was flawed with tool bugs and incompatibilities. The result was a project delay of more than 18 months. There were 22 student thesis written within the project. About half of the students worked on the VLSI design, the other half on the software environment and algorithm implementations. Together with an average of 3 scientists working on the project, the total effort was about 32 men year, not too much for a design of that size and the huge software package developed for it. Table 3. gives an overview of the final chip data.

We have started practical programming of image processing algorithm cores on the C++ processor model already at a very early stage of the design to evaluate potential for the further optimization of instruction set and controlling scheme. The resulting code also formed an important part of the later verification on RTL and gate level. Table 2. shows an overview of the HiPAR-DSP's performance, gained from these optimized assembler written implementations. The performance of C++ written programs varies with the experience of the programmer and the use of the extended processor specific data types. In average, about 1.5 times slower execution is achieved with C++ implemented software.

8. CONCLUSION

A novel programmable DSP architecture has been proposed and implemented. The innovative parallel architecture reaches high performance while keeping a flexibility comparable to other commercially available DSPs. All design phases - beginning from the programming of a first behavioral processor model followed by VHDL implementation, synthesis, placement, route and backannotation and finally ending with the generation of GDS-II mask tapes - have been carried out at the university. The complexity of the architecture required state of the art ASIC technology, design flow and EDA-Tools. Various difficulties in the establishment of the design flow have been mastered. Most recent features especially of the backend tools turned out to be utterly necessary to fulfil the demands of deep submicron design. Currently the HiPAR-DSP is in production phase at the Atmel Fab in Rousset, France. First samples will be available by time of DAC'98. The project impressively demonstrates the possibility to design programmable digital signal processors under university conditions. Complexity, performance and tool support is at least comparable to those of commercially available counterparts.

References

[1] K. Rönner, J. Kneip: "Architecture and Applications of the HiPAR Video Signal Processor", Transactions on Circuits and Systems on Video Technology, 2/1996

[2] J. Kneip, M. Ohmacht, K. Rönner, P. Pirsch: "Architecture and C++-programming environment of a highly parallel image signal processor", Microprocessing and Microprogramming, Vol. 41 (1995), pp. 391-408.

[3] J. Kneip, K. Rönner, P. Pirsch: "A Data Path Array with Shared Memory as Core of a High Performance DSP", Proc. Int. Conf. Application Specific Array Processors (ASAP)'94, pp. 271-282, Aug. 1994.

[4] J. P. Wittenburg, M. Ohmacht, W. Hinrichs, J. Kneip, P. Pirsch: "HiPAR-DSP: A Parallel VLIW RISC Processor for Real Time Image Processing Applications", Proc. Int. Conf. on Algorithms And Architectures for Parallel Processing (ICA3PP)'97, pp. 155-162, Melbourne Dec. 1997

[5] Synopsys Online Documentation, Version 1997.01, Synopsys Inc.

[6] Compass Online Documentation, Version V9R2, Avant! Inc.