Methodologies for Computer System Design
W.K. Giloi and B.D. Shriver (Editors)
Elsevier Science Publishers B. V. (North-Holland)
IFIP, 1985

THE MIMOLA DESIGN SYSTEM: A DESIGN SYSTEM WHICH SPANS SEVERAL LEVELS

Peter Marwedel
Inst. fur Informatik & Prakt. Mathem.
University of Kiel
Kiel, W. Germany

The MIMOLA design system is a computer-aided system for the design of digital computers. The system uses requirements containing typical ap plication programs of the computer to be de signed. The output of the design process is a register-transfer level description of the com puter. Thus the MIMOLA system covers several design levels. The paper presents the method, the CAD-tools and some applications. CAD tools generate parallel programs from sequential pro grams, synthesize hardware structures, gene rate code and evaluate hardware structures. The designer plays an active role in the design process by bringing in his ideas about design improvements.

1. INTRODUCTION

Nowadays hardware cannot be designed independently of software. The implications of software algorithms are too strong. For example, algorithms for safe operating systems require hardware protection mechanisms and fast processing of pictorial information requires the use of array processors. Functions of operating systems, which were formerly implemented in software, are now implemented in hardware or firmware. Very large scale integration (VLSI) will allow us to design parallel processors which are structured according to the structure of the problem. Therefore we need tools for the design of hardware from a high-level specification of the problem.
Before we describe our tools, we have to think about what our specification should look like. We consider the specification of a machine instruction set as inadequate because the problems, which are to be solved by a computer system, are stated on a much higher level and because the specification of conventional instruction sets will not allow us to fully utilize the possibilities of VLSI. Therefore we start with a higher level specification. Our specification contains the set of problems to be solved by the computer. This set of problems is described by a set of application programs, written in a high-level programming language. This specification opens a large design space and allows us to optimize the instruction set.
A key feature of our system is the automated synthesis of hardware, i.e. the automated selection and interconnection of hardware modules such that the specified higher level function is performed. Synthesis procedures are regarded as a means to overcome the verification problem, i.e. the problem of verifying

low level meets the high-level specification. Simulations at the low level, which have been used for many years, are time-consuming and cannot guarantee the absence of errors.

2. THE MIMOLA DESIGN SYSTEM

The basic ideas of the MIMOLA system go back to 1975, when G. Zimmermann presented a paper at the national GI-conference'. The paper contained a new design method and the definition of the language MIMOLA (= machine independent microprogramming language). During the years that followed, design tools were developed and applied 2'3'u. Currently the system is being extended and used for two designs.
Similar systems have been concurrently developed at the CarnegieMellonUniversity[5] and at the MITE . Huang combined ideas of the Carnegie-Mellon system and our system'.

2.1 The language MIMOLA

A language, which is used at several design levels, has to be able to describe several levels of details. Therefore MIMOLA supports a PASCAL-like level as well as RT-levels[8]. (In the CONLAN project[9], a different approach is used: a family of languages is derived from a common base language).

Example:

A high-level MIMOLA program may contain the sequence

```
TYPE    .INTEGER = .BIT(15:0);
VAR     p,q,r : .INTEGER;
   p:= r;
   q:= p "-" 1
```

The example shows that there are only minor syntactical differences (like double quotes enclosing operators) between PASCAL and high-level MIMOLA programs.

The language MIMOLA allows explicitly binding variables to locations. This may be done either in the declaration part:

```
VAR    p: SM(100).INTEGER
       q: SM(101).INTEGER,
       r: SM(102).INTEGER;
```

or in the statement part:

```
       SM(100):= SM(102);
       SM(101):= SM(100)  "-" 1
```

S ( ) is the 'contents of' operator in MIMOLA. 'M' is the name of the memory and 100, 101 and 102 are addresses. This level is called partially bound RT-level because variables have been bound to memory locations.

It is also possible to define explicitly the functions to be performed and the number of bits to be used e.g.:

```
SM(100.BIT(7:0)/"LOAD").BIT(15:0):=                    (1)
SM(102.BIT(7:0)/"READ").BIT(15:0)
```

Finally, completely bound programs may be written in MIMOLA.

Example:

```
SM_A(I(100).BIT(17:9)/I(1).BIT(18)).BIT(15:0):=        (2)
SM B(I(102).BIT(24:18)/I(0).BIT(19)).BIT(15:0)
```

This means that e.g. instruction bits I.BIT(17:9) contain the value 100. These bits are used at the address input of memory port SM-A.
Multiplexers, ALU's, decoders and busses may also be included in completely bound programs. This is necessary in order to use MIMOLA for lower design levels.
But it is not sufficient to allow procedural descriptions of programs. For automatic generation of hardware tests, for retargetable code generation, for chip layout generation etc. it is necessary to include non-procedural descriptions of hardware structures in the language.
Examples (these examples make use of changes to the implemented syntax)

1. Declaration of hardware modules:

```
MODULES
  B74xyz(IN left,right:.BIT(15:0);
         IN sel:.BIT(1:0);OUT result:.BIT(15:0))
    BEGIN
     result:=CASE sel OF
             0:(left "+" right)delay 40;
             1:(left "-" right) delay 40;              (3)
             2:(left "AND" right)delay 30;
             3:(left "OR" right)delay 30
             END
    END;
```

2. Declaration of paths:

```
CONNECTIONS
  B74xyz_cpu.result  ->B74xyz_io.left delay 7;
```

It has been recognized that languages, which span over several design levels, have to provide a mechanism for passing information, which cannot be expressed by normal syntax rules, between the levels. MIMOLA provides two such extension-mechanisms:

1. Reserved words may be followed by an underscore character and an alphanumeric string. Examples:

   CALL_FORTRAN, RETURN_FROMINTERRUPT, END_CASE

2. Lexical tokens may be followed by 'property-lists' included in angle brackets. Examples:

   CONNECTIONS <test_for_stuckat>, B74381<cost=17>

There are various mappings to be performed during the design process.

For example, variables have to be replaced by memory locations, high-level language elements like FOR-Loops have to be replaced by register-transfer statements and unimplemented operations to be replaced by routines. MIMOLA allows an explicit definition of these normally implicit replacements.

```
Examples('&'-signs denote parameters):
   REPLACE   a   WITH      SM(100).INTEGER END,
   REPLACE "INCR" &a  WITH &a "+" 1        END;
```

CONST-, TYPE- and VAR-declarations are implemented as shorthands for these replacements. Underscore-suffixes of the keyword REPLACE are used for making replacement rules available to newly designed software tools without changing the syntax of MIMOLA.

2.2 Specification of the design problem

In the introduction we mentioned that a design with MIMOLA starts with application programs. Type and number of programs have to be selected such that they sufficiently represent the application areas of the projected computer. In this context, the term 'application programs' includes the operating system, compilers, editors etc.
The programs specify, which type of a processor is to be designed. If signal processing programs are used, signal processors will be synthesized. Similarly, array processing programs will lead to array processors etc. General purpose computers may be designed by using a broad spectrum of applications as input. In general, hardware architectures will be structured according to the structure of the applications.
The specification has to be written in MIMOLA. The task of converting e.g. PASCAL programs to MIMOLA does not present many problems.
Semantics of MIMOLA is predefined only at the RT-level. Semantics of high-level MIMOLA is defined by a set of substitution rules (called macros), which define a mapping from high-level MIMOLA to RT-level MIMOLA. This allows us, for example, to translate both PASCAL and FORTRAN DO-loops into MIMOLA DO-loops and to define the semantics by different mappings to the RT-level. Substitution rules are always required as a part of the specification. However, users do not have to develop these rules themselves but select them from a library.
The functional behaviour of programs is not specified by the programs alone. Normally, information about overflow traps, index checking, protection etc. is implicit. This information has to be made explicit in the problem specification. Index checking, for example, may be included in the substitution rules for arrays.
A simulator is included in the MIMOLA CAD-system. It may be used to check if the programs perform the intended function.
The synthesized hardware may be less optimal for rarely executed parts of the programs. Therefore the CAD-system has to know the relative importance of the different parts of the application programs. To this end, weights or estimated frequencies of execution have to be inserted into the programs. There are various possible sources for this information:
- In certain cases, mathematical analysis is possible'°. - If the programs are executable somewhere, their execution may be

monitored by hard-, soft- or firmware-monitors. -The MIMOLA simulator is able to insert the frequencies into the programs automatically.

The specification is completed by a set of boundary conditions. These boundary conditions may prescribe the number and/,or type of ALU's to be used, the number of memory ports, the instruction fields and the data paths. The amount of restrictions may range from empty to a complete specification of a target architecture (in the last case the synthesis part of the system is not used). Resources (like ALU's and memories) which are to be used, may be completely described in a library. This description may contain low-level information such as the required VLSI-chip area.- This allows us to use elements of a bottom-up design style in our top-down design. For example, the library may contain a TTL-catalogue.

2.3 Transformations applied to the problem description

In order to produce the desired results, the MIMOLA software system (MSS)" applies several transformations to the set of application programs. Fig. 1 shows how these transformations are related.

2.3.1 Translation of MIMOLA programs into an intermediate language

In order to simplify all following transformations, MIMOLA programs are translated to an intermediate language which closely mirrors the flow of control and of data. The intermediate language is similar to intermediate tree languages used in compiler development projects"' 13.

2.3.2 Mapping of the programs to the RT-level

This transformation replaces all high-level language elements, such as CALL, FOR and WHILE, by assignments and simple conditions. This replacement uses the already mentioned substitution rules. These substitution rules provide the

```
REPLACE  &a  "/"  &b  WITH  Mdivide(&a,&b)  END
```

replaces the divide operation by a call to a routine called Mdivide. These replacements are useful especially in design iterations when the statistical evaluation (cf. 2.3.8) found that the divide hardware is rarely used.
Binding variables to memory locations may be done explicitly in the application programs, by standard substitution rules collected in a library or by an automatic optimizing memory allocator. Thus the system designer is free to choose between different addressing mechanisms. For example, the library may provide substitution rules for static allocation and for dynamic allocation using the index register-technique or Tanenbaum's proposal"`. Therefore it is possible to study hardware implications of the different techniques. Static allocation is useful even for PASCAL application programs. We are just designing an array processor for algorithms written in PASCAL. In that particular case, recursive procedures will never be used.

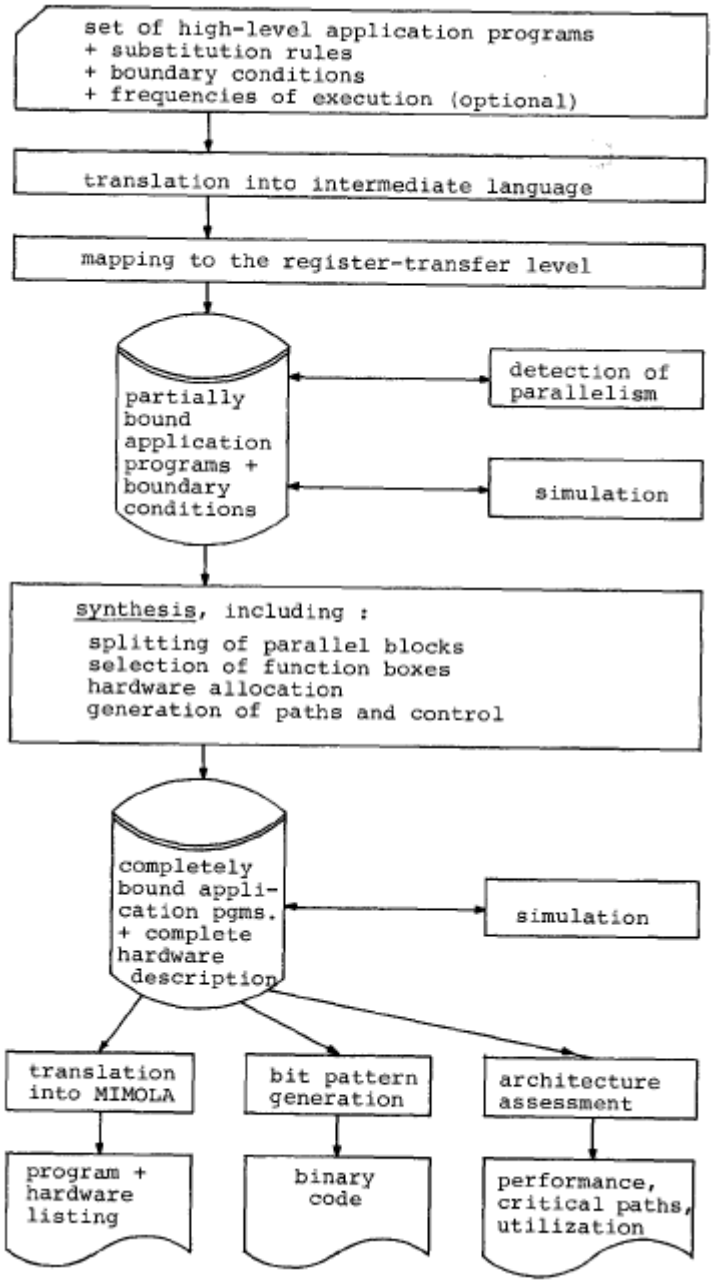

Fig.1 Synthesis of hardware structures with MSS

2.3.3 Detection of parallelism

We put special emphasis on the design of machines with instructions similar to horizontally microprogrammable machines since we found out that these machines offer performance improvements over classical sequential machines '5 and because the control part of these machines seems to be less complex than for machines with several instruction streams. Furthermore, there is an underlying microarchitecture for most of the sequential machines and therefore tools for the design of parallel microarchitectures are needed even if sequential machines are to be designed.
In order to faciliate the tasks of hardware synthesis and of code generation for parallel machines, application programs are made more parallel. This means that blocks of statements are coalesced such that the number of statements executable in parallel is increased.
This transformation is based upon control flow and data dependence. It is not easy to decide if data dependence exists on the registertransfer level at compile time.
In our design system, two memory references are considered as datadependent, if memory names and the expressions for the calculation of effective addresses are equal. They are considered as independent, if memory names are different or if they belong to different nonformal variables or if the effective addresses differ by a non-zero constant. In all other cases, memory references are considered as potentially dependent.

Examples:

a[1,i], a[1,i] : data dependent

a[1,i], a[2,i] : data independent

a[1,i], a[1,j] : potentially data dependent

There are two types of data dependence and potential data dependence: (potential) source data dependence and (potential) destination data dependence (cf. Kuck[16]). Given two statements S_i and S_j and S_i being a dynamic predecessor of S_j, then S_j is (potentially) source data dependent on S_i iff there exists a memory location which is read by S_j and (potentially) written by S_i. S_j is (potentially) destination data dependent on S_i iff there exists a location, which is (potentially) written by S_i and S_j.

If potential data dependence exists, our present design system will not put them into a single block. Otherwise, run-time tests for data dependence would be required. Normally, these run-time tests would either require additional hardware or slow down the application. A future version of our CAD tools might include the possibility to use run-time testing for data dependence.

If there is no potential data dependence, source data dependence is removed by statement substitution and both statements are put into a single block (except when both are destination data dependent: then the first statement is deleted).

Example:

The following sequence of statements:

```
SM(100):=SM(102); SM(101):=SM(100) "-" 1
```

may be executed in parallel if the data dependence is removed by copying the right side of the first statement to the right side of

```
second statement ('statement substitution'[16]):
        SM(100):=SM(102), SM(101):= SM(102) "-" 1
(* In MIMOLA statements in a parallel block are separated a by a
comma *)
```

Our design system does not only coalesce straight line sequences but also coalesces blocks with more than one dynamic successor with the following blocks. At present, FOR-loops are not automatically converted into parallel execution of all instances of the loop and procedure calls are not modified. Both transformations would be required for multi-instruction stream processors " .

2.3.4 Simulation

Application programs may be validated by using simulations. In addition, the simulator is able to compute dynamic frequencies of execution. These frequencies may be added to the programs. Figure 2 describes some details of the simulation subsystem:

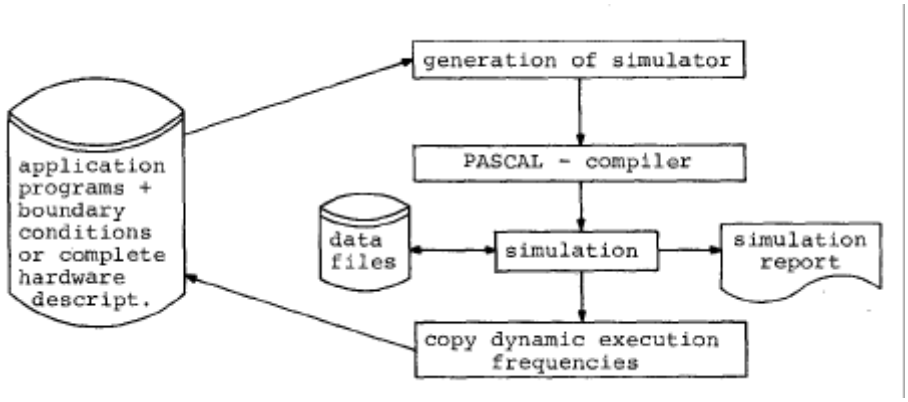

Fig. 2 Simulation - subsystem (Detail of fig. 1)

2.3.5 Synthesis

Synthesis procedures generate an RT-level description of the synthesized computer. This description specifies used ALU's, memories, path-switching circuits and their interconnections on a level corresponding to the description of MODULE B74xyz above. Synthesis procedures try to generate a hardware, which allows as many parallel operators as can be used by the program. However, boundary conditions may be used to limit the number of memory ports, ALU's etc. The synthesis part of the MSS tries to generate the fastest hardware structure which does not violate the given boundary conditions.
Algorithms, which would generate the optimum hardware in a single step would be too complex. Therefore synthesis has to be broken down into a series of steps such that each step has a manageable complexity. We decided to use the following steps:

1. Computation of the number of resources which would be necessary to execute each parallel block in a single step.
2. Interactive definition of the number of allowed resources.
3. Splitting of parallel blocks into sequences of blocks such that each block may be executed in a single step. During this transformation it may be necessary to split complex statements and therefore temporary locations, holding intermediate results, are generated.
4. Arithmetic/logic operations within each block are counted and the result is used to compute a sufficient number of function boxes such that the total cost for function boxes is minimal. Costs may be declared for predefined function box types. The optimization uses an integer linear programming algorithm';.
5. The next step assigns hardware resources to operations in the program. Memory ports are assigned to read and write operations, function boxes to arithmetic/logic operations and instruction fields to constants. The assignment is done such that the minimum number of new data paths is required for each scanned block. At the end of this step, a complete description of the hardware structure is known. In addition, the application programs have been bound to the available hardware. This means that the programs describe which part of the hardware structure is used to perform a certain operation (section 2.1 contains an example of a bound statement).

2.3.6 Bit pattern generation

Instruction bit patterns may be easily obtained by scanning completely bound programs for instruction bit definitions (see section *2.1).*

2.3.7 Translation into MIMOLA

A readable form of bound programs may be obtained by a translation from the intermediate language into MIMOLA. MIMOLA is general enough to allow this translation.

2.3.8 Architecture assessment and preparation of design decisions

We assume that for quite a number of years it will not be possible to autmatically generate sufficiently optimized designs. The ideas of human designers will be needed to improve the design. Our system allows the human designer to bring in his own ideas by modifying the design and to iterate the design process. Our design system aids the designer by providing data upon which design decisions may be based. Furthermore, our system assesses the value of the designer's decisions. Both functions are provided by the timing-evaluation processor of the MSS. This processor

- searches for bottle necks such as critical data paths,
- searches for resources, which, with respect to their cost, are insufficiently used,
- searches for resources which could be a substitute for insufficiently used resources (computation of joint probabilities of resource usage),
- computes an estimated performance.

A good estimate for the performance is the expected runtime of the

programs contained in the specification. Runtime is computed by computing the time to complete each of the generated instructions and then multiplying by the estimated frequencies of execution (cf. section 2.2):

$$\text{runtime} = \sum_{\substack{\text{all} \\ \text{instructions}}} \text{duration of instruction} \times \text{frequency of instruction}$$

Runtime is easily computed by using the completely bound intermediate language trees since these contain execution frequencies and because the delay times declared in hardware descriptions like (3) may be copied to the trees.
Three versions for the computation of instruction durations are implemented in the MSS[19]:
1. Constant duration: The duration is the same for all instructions and is equal to the time required to complete the slowest instruction.
2. Instruction-dependent duration: For each of the instructions duration is computed separately, using worst-case data. The duration is either implicitly contained in the instruction or compiled by the compiler and stored in a separate field of the instruction (the Am2925 of AMD is designed to support just this mode of operation 20). The use of instruction dependent durations significantly reduces
runtime if long as well as short instructions exist. 3. Instruction- and data dependent duration: This mode of operation may be implemented by adding 'data-valid' tokens to all data lines. If all tokens reached their destination, the next instruction may be started (we assume that there is only one instruction stream). A similar mode of operation is used in selftimed systems[21].

2.3.9 Design decisions

As mentioned in the previous section, design decisions are necessary for optimizing the design. The designer's task is to use the information provided by the system (e.g. runtime), to decide which changes should be done, to change some of the inputs to the MSS and the iterate the design process.
The different levels which are affected by the designers decisions demonstrate very well how the MIMOLA design method spans over several levels:
1. Changes of the delay time: These changes could be a result of critical path analysis. They affect only the timing characteristics of the RT-level modules, not their function. For example, Schottky-TTL could be used instead of the Low-Power Schottky TTL. Or a path could be moved from polysilicon to metal. Thus, these changes are changes at a low level (possibly the layout level). 2. Changes of the function select codes: These changes only affect the internal structure of the RTL modules, still their function is not modified. The level is somewhat higher than the previous one. 3. Changes of the functions of the RTL-modules: In certain cases, functions should be added or deleted. For example, joint distributions for the use of functions may indicate that a special "DECREMENT" function in module 1 is not required because module 2 is able to subtract and is always idle when module 1 is decrementing. These changes affect only the RTL modules but not the system itself.
4. Changes of the number of invocations of the hardware resource

types: These changes affect the speed of the microarchitecture. Its functional capabilities are unchanged as long as there is at least one resource of each type. 5. Changes of the interconnections: Rarely used connections may be deleted if they are not absolutely necessary (for example, a bus which increases parallelism may be deleted; an interrupt request line may not be deleted). These changes possibly affect the functional capabilities of the microarchitecture. 6. Changes in the mapping of the programs to the RT-level: Example: Using Tanenbaum's proposal instead of the index-register technique (display-technique) for addressing variables, implementing certain functions in software instead of hardware. In this case, the mapping to the RT level has to be repeated. 7. Changes in the application programs, or even in the algorithms Example: It may turn out that a particular sorting algorithm does not contain enough parallelism to obtain the desired speed. In this case the hardware design cycle has to be reiterated from the beginning. The synthesized architecture also has to be evaluated in terms of cost. This evaluation of course is technology-dependent. An older version of the MSS computes costs of realizations in TTL. This computation is based upon a paper by Phister [22] and described in a paper by Zimmermann [3]. For VLSI it is necessary to predict the required chip area. Some research is going on in this direction [23],

2.3.10 Retargetable code generation

To a certain extent generated hardware structure may be modified by changing the boundary conditions and initially design iterations should be done by using the synthesis part of the MSS. However, as ideas about the hardware structure become more and more precise, only minor changes to the generated hardware description are to be made (e.g. a single path is to be removed). After such minor design changes the hardware structure still is fully specified. Therefore another run through the synthesis part of the MSS would make no sense. However it would make sense to bind the application programs to the modified hardware structure. There are several reasons for this:
1. After illegal design changes the binding process would fail and therefore represents a means for detecting such errors, 2. Bound programs are useful for architecture assessments, 3. Bound programs are required for obtaining binary code.
Because of these reasons we designed a code generator which is able to bind application programs to hardware structures defined by a hardware description. Such a code generator is called a retargetable code generator. The code generator was written such that it accepts a wide range of changes to generated hardware descriptions. Therefore descriptions of many of the available hardware structures are accepted by the code generator and it may be used by its own, namely for generating code for (micro-programmable) architectures which have not been designed with the MSS. Maschines based on the **AMD2900** series of chips may be described nicely, for example.
There has been some work on retargetable code generators [13,24,25,26, 12,28]. However, these approaches could not be used **for our system,** because with the exception of Baba's MPG system the systems are unable to **produce parallel microcode. In addition, our design itera-**

tions require that the target machine may be changed in less than a day. Changes of the target machine normally are less frequent and require e. g. one month of work 25. On the other hand, speed of compilation is of secondary importance. Baba's system would be somewhat hard to use since it employs a hardware description which is not directly related to the resources at the register-transfer level, namely data paths, adders etc. Some hardware design systems are able to produce code for the architectures they synthesize 7,29. However, they are unable to produce code for user defined architectures and therefore don't allow a large range of manual design modifications.
Our code generator works similar to the last step of the synthesis part of the MSS in that it assigns hardware to the program. In contrast to the synthesis part however, it does not create new hardware resources and uses a larger set of program transformations. This larger set of program transformations is required i.e. in order to bind programs to user-defined architectures. Transformations using neutral elements of operations, the law of commutativity, and 'reflected' operations (e.g. "<=" and ">=" form a pair of reflected operations) are standard. Others may be defined by the user. There is a 'tiny expert system' built into the code generator which tries to apply useful transformation rules. During the first phase of code generation, our system computes the different possibilities which exist for the execution of statements. According to Mallet30, the different possibilities for assigning hardware to the knots of the partially bound treesare called versions. Example: the completely bound assignment statement (2) is one possible version of the partially bound assignment statement (1).
The second phase selects a version for every partially bound statement and packs versions into microinstructions. Several algorithms for microinstruction packaging are known 30. We use a modified LINEAR algorithm.

The code generator is included in Fig. 3.

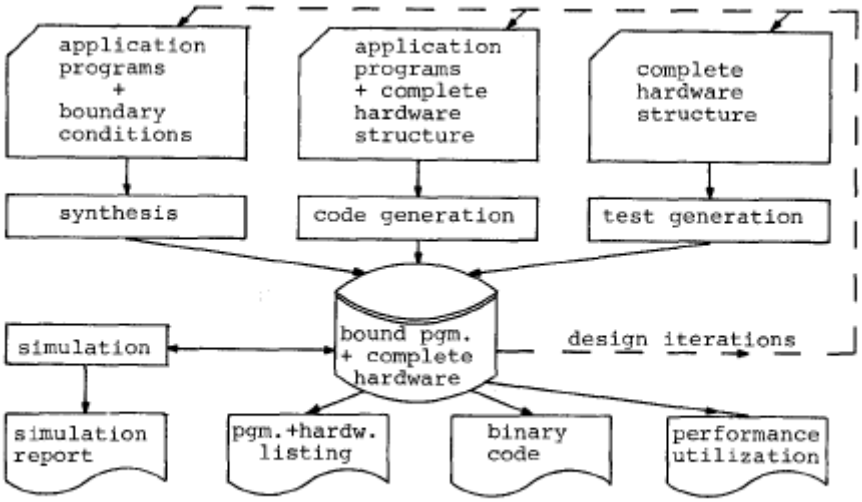

Fig.3 Global view of the MIMOLA Software System

2.3.11 Test Generation
The increasing complexity of VLSI chips calls for an improved test generation and for easily testable designs. It is well known that the automatic generation of tests from a hardware description is feasible31. Our system generates tests for stuck-at errors for declared interconnections and for the functions of hardware modules. Hardware structures, for which no test can be generated automatically should be modified before the design continues.

3. APPLICATIONS

3.1 Scientific Subroutine Package

The first application of the design method was the design of a processor for scientific computations. These were represented by IBM's scientific subroutine package (SSP). A major part of this package was translated into MIMOLA. After two design iterations (using an early version of the MSS) it was decided to implement an architecture containing five function boxes in hardware. The architecture is about as complex as the MODCOMP II minicomputer (equivalent to a midrange PDP-11). It is about 25 times faster as the MODCOMP 15.

3.2 Redesign of a SIEMENS 7.000-type machine

In a second application, we used a functional description of the instruction set of a SIEMENS 7.000-type machine as a design specification. Although the specification level is lower than we intended, the design method proved to be useful.
A functional description of the instruction set was written in MIMOLA. Weights were known from benchmarks. After several design iterations, a machine was obtained, which was slightly faster than a SIEMENS 7.750. The design of a machine being as fast as a SIEMENS 7.760 is feasible. A surprising result was the reduction of the size of the microcode to 15 $ of the SIEMENS design.
This example shows that the design space was too small to improve the performance of the SIEMENS architecture. However, the design was done by one student as his master thesis. A reduction of the design time is obvious. The the sis32 not only describes the architecture but also contains the complete microcode. This is an example of the integrated design of hard- and firmware.

3.3 Design of a processor for graphic layout

Modern electronic equipment is now used for the reproduction of fotos in magazines and books. During the layout of pages, fotos are frequently modified. Scale changes, rotations and colour corrections are frequent transformations. Current processors are either too slow or too expensive to allow these transformations to be done interactively. As a part of a recently completed thesis33, these transformations have been written in MIMOLA. Hardware structures for these programs were obvious and could be specified manually. The **MSS was** used in order to bind these programs and to predict performance. Results indicate that a reduction of runtimes from half an hour (on a special purpose hardware) to less than a

3.4 Design of a processor specified by an operating system
In another application, the kernel of an operating system has been translated into MIMOLA. In this application a hardware monitor was used in order to obtain execution frequencies. The designed machine will be built up in hardware and is expected to be *14* times faster at twice the cost of a commercial minicomputer (using estimated manufacturing costs).

CONCLUSION

A design system has been described which may be a stepping stone for the development of tools for the design of VLSI computers. The design system uses concepts of compiler construction (e.g. code generation) and hardware oriented concepts (e.g. function boxes). It is supported by a language which is able to describe software and hardware. Computers, which were designed with the MIMOLA system, bear comparisons with manual designs.

ACKNOWLEDGEMENT

This paper would have been impossible without the ideas of G. Zimmermann. In addition, many students contributed to the MIMOLA

REFERENCES

1. Zimmermann, G., Eine Methode zum Entwurf von Digitalrechnern mit der Programmiersprache MIMOLA, Informatik Fachberichte, Vol. 5, Springer, 1976
2. Zimmermann, G., The MIMOLA Design System: A Computer Aided Digital Processor Design Method, Proc. 16th Design Autom.Conf., 1979, pp. 53-58
3. Zimmermann, G., Cost Performance Analysis and Optimization of Highly Parallel Computer Structures: First Results of a Structured Top Down Design Method, Proc. 4th Int.Conf.on Computer Hardware Description Languages, 1979, pp.33-39
4. Marwedel, P., The MIMOLA Design System: Detailed Description of the Software System, Proc. 16th Design Automation Conf., 1979, pp. 59-63
5. Hager, L.J. and Parker, A.C., Automated Synthesis of Digital Hardware, IEEE Trans. Comp., 31, 2 (1982), pp. 93-109
6. Miranker, G.S., The Use of Conflict in the Translation and Optimization of Hardware Description Languages, Ph.D. Thesis, MIT, 1979
7. Huang, C.-L., Computer-Aided Logic Synthesis Based on a New Mufti-level Hardware Description Language, Ph.D. Thesis, State University of New York at Binghampton, 1981
8. J6hnk, R. and Marwedel, P., MIMOLA Language Reference Manual, Revision 2, Report of the Institut f. Informatik and Prakt. Mathem., Kiel, 1984 (in preparation)
9. Piloty, R., Barbacci, M., Borrione, D., Dietmeyer, D., Hill, F. and Skelly, P., CONLAN - A Construction Method for Hardware Description Languages, Proceedings Nat.Comp.Conf., Vol. 49, 1980
10. Knuth, D.E., The Art of Computer Programming, Addison Wesley, 1975
11. Jöhnk R., Krager, G. and Marwedel, P., MIMOLA Software

12. Cattell, R.G.G., Formalization and Automatic Derivation of Code Generators, Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, 1978
13. Schmidt, U. and Völler R., Die formale Entwicklung der maschinenunabhängigen Zwischensprache CAT, Informatik Fachberichte, Vol. 50, Springer, pp. 57-64
14. Tanenbaum, A.S., Implications of Structured Programming for Machine Architecture, CALM, Vol. 21, 1978, pp. 237-245
15. Marwedel, P., The Design of a Subprocessor with Dynamic Microprogramming with MIMOLA, Informatik Fachberichte, Vol. 27, Springer, pp. 164, 1980

16. Kuck, D.J., The Structure of Computers and Computations, p. 111, Wiley, 1978
17. Plas, A. et al., LAU System Architecture: A Parallel Data Driven Processor Based on Single Assignment, Int.Conf. on Parallel Processing, 1976, pp. 293302

18. Langmaack, H., Gomory I, Collected Algorithms of the ACM,Algorithm 263A,1978
19. Marwedel, P., Statistical Studies of Horizontal Microprograms, Proc. 5th Int. Conf. on Computer Hardware Description Languages, Kaiserslautern, 1981
20. Advanced Micro Devices Bipolar Microprocessor Logic and Interface Data Book, 1981
21. Seitz, C.L., System Timing, in: Mead, C.A. and Conway, L., Introduction to VLSI Systems, Addison-Wesley, 1980

22. Phister, M., Analyzing Computer Technology Costs, Computer Design, Sept. 1978
23. Anceau, F., Working Material, Advanced Course on VLSI Architecture, Bristol, 1982
24. Baba, T. and Hagiwara, The MPG System: A Machine-Independent Efficient Microprogram Generator, IEEE Trans. on Computers, Vol C30, 6 (1981)
25. Landwehr, R., Jansohn, H.-St. and Goos, G., Erfahrungen mit einem automatischen Code-Generator, Conf. on Implementation of PASCAL-like languages, Kiel, 1981
26. Glanville, R.S., A Machine Independent Algorithm for Code Generation and its Use in Retargetable Compilers, Ph.D. Thesis, University of California, Berkeley, 1978
27. Miranker, G.S., The Use of Conflict in the Translation and Optimization of Hardware Description Languages, Ph.D. Thesis, MIT, 1977
28. Ganapathi, M., Fischer, C.N. and Hennessy, J.L., Retargetable Compiler Code Generation, acm computing surveys, Vol. 14, 1982, pp. 573-592
29. Snow, E.A., Siewiorek, D.P. and Thomas, D.E., A Technology-Relative ComputerAided Design System: Abstract Representations, Transformations, and Design Tradeoffs, Proc. 16th Design Automation Conf., 1978, pp. 220-226
30. Mallett, P.W., Methods for Compacting Microprograms, Ph.D. Thesis, University of Southwestern Louisiana, Lafayette, 1978
31. Lai, K.-W., Functional Testing of Digital Systems, Report CMU-CS-81-148, Carnegie-Mellon University, Pittsburgh, 1981
32. Krüger, G., Entwurf einer Rechnerzentraleinheit fur den Maschinenbefehlssatz des SIEMENS Systems 7.000 mit dem MIMOLA-Rechnerentwurfssystem, Diploma Thesis, University of Kiel, Kiel, 1980
33. Schulz, T., Entwicklung schneller Prozessoren zur Bildbearbeitung, Diploma Thesis, University of Kiel, Kiel, 1983