THE INTEGRATED DESIGN OF COMPUTER SYSTEMS WITH MIMOLA

PETER MARWEDEL
UNIVERSITY OF KIEL

Summary

A design method and the corresponding *CAD* tools for the design of digital computers from a very high-level functional specification is presented. Computers are specified by a set of application programs, for which they shall be optimized, and a set of boundary conditions for the type and number of hardware resources. The paper describes *CAD* tools, which synthesize architectures from the specification, evaluate their cost and performance and generate microcode. Several design iterations, in which the designer plays an active role, are used to optimize the design. Two applications of the design method are included in the paper.

1. Introduction

Nowadays hardware cannot be designed independently of software. Too strong are the implications of software algorithms. For example, algorithms for safe operating systems require hardware protection mechanisms and fast nrocessina of pictoral information requires the use of array processors. Functions of operating systems, which were formerly implemented in software, are now implemented in hard- 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 how 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

cation 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 that a manual design at the 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 MI140LA 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[2'3'4]. Currently the system is being extended and used for two desians.

A similar system, starting from a slightly lower level, was concurrently developed at the Carnegie-Mellon University . Huang combined ideas of both systems6

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 +.

A high-level MIMOLA program may contain the sequence

```
TYPE    .INTEGER = .BIT(15:0);
VAR     p,q   : .INTEGER
LO      p:= p/1 - > B(+),
        q:= p/1 - > B(-);
```

+) A different approach is used in the CONLAN project: a family of languages is derived from a common base language.[7]

This means that a block, labelled LO, may be evaluated in parallel. Not yet named function boxes of type B (binary) shall be used to increment/decrement a variable p (Notation for expressions is postfix). The old value of p will be used in the second assignment ('edge triggered assignment').

The same language is able to describe explicitly which memory locations are used:

```
LO   Smain(100) .BIT(15:0): = Smain(100). BIT(15:0)/1->B(+), (1)
     Smain(101) .BIT(15:0): = Smain(100).BIT(15:0)/1->B(-);
```

S ( ) is the CONT operator in MIMOLA. 'main' is the name of the memory and 100 is the address. This level is called partially bound RT-level because variables have been bound to memory locations.

Finally it is possible to express completely bound programs in the language:

```
LO Smain_A(I(100).BIT(17:9)).BIT(15:0) : = Smain_B(I(100).BIT(17:9))
          .BIT(15:0)/F1 ->Balu_0(I(7).BIT(3:0)),...          (2)
```

This means that instruction bits I.BIT(17:9) contain the value 100. They are used at the address input of memory ports Smain_A and Smain_B. The constant 1 is hard-wired (F1). Function box Balu_0 is steered by bits 3 to 0 of the instruction.

MIMOLA also contains constructs for the description of hardware.

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 provided with 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[8].
- 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 progran (cf. Fig. 1).

2.3.1 Translation of MIMOLA programs to 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 9,10

2.3.2 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.

All variables are replaced by references to actual memories.

2.3.3 Program transformations generating a more parallel program

In order to allow synthesis and effective use of parallel hardware, the application programs are made more parallel. The transformations cause blocks to coalesce such that the number of parallel executable statements in the blocks is increased. All resource constraints are ignored during the transformations, they are considered later on. The method is similar to methods used in data flow analysis[11] ('reducible flowcharts').

There may exist data dependences between statements of blocks to be coalesced. If they would lead to additional tests at runtime the blocks are not coalesced at all. Otherwise, data dependences are removed by statement substitution[12]

Example:

```
Blocks L0 and L1:
   L0  S(100):= S(101);
   L1  S(102):= S(100)/S(103) ->B(+);
```

will be coalesced into a single block of assignments executable in parallel:

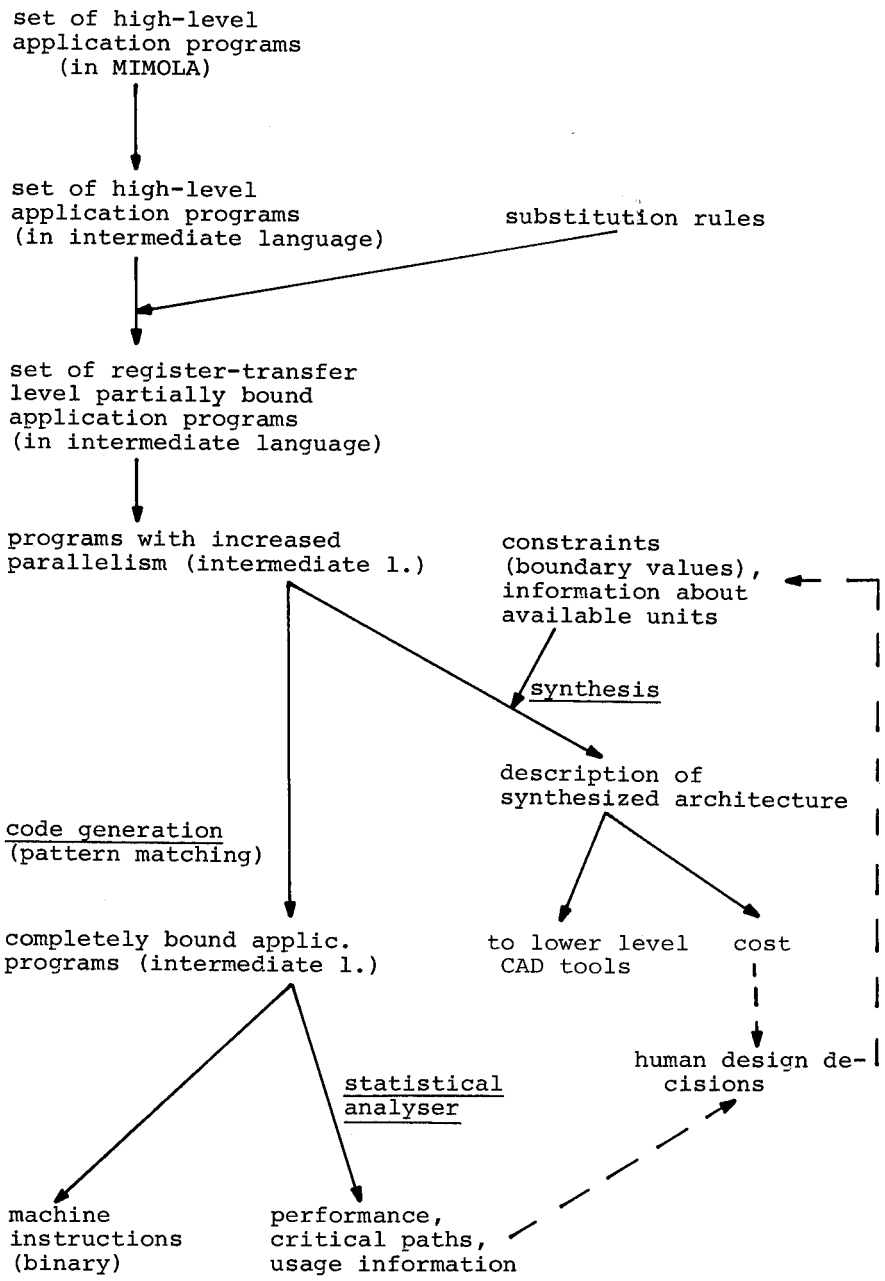

Figure 1 Steps in the design with MIMOLA

```
LO and 1  S(100):= S(101),
          S(102):= S(101)/S(103) ->B(+);
```

2.3.4 Synthesis

Synthesis, as it will be described, generates an RT-level hardware description, containing memories, function boxes and paths. Function boxes are described by their width, their speed and a list of their functions. Synthesis below the RT-level, e.g. synthesis of the function boxes from logic gates[13], is one of the projected extensions to the MIMOLA system.

Hardware is synthesized such that parallel execution of the statements contained in a block is possible for every block, except if the boundary conditions do not allow that much hardware. This means that the fastest architecture, which does not violate the boundary conditions, is synthesized.

Synthesis procedures derive the following architectural parameters from the application programs:

- Number of input and output ports: memories with many ports may be designed. Synthesis procedures compute the largest number of readand write-operations in a block in order to create the required memory ports.
- Function boxes: a number of function boxes is created which is sufficient to execute all functions in each of the blocks in parallel. Available types of function boxes may be declared.
- Paths (wires): the required data paths are created.
- Instructions: it is assumed that hardware units are directly con trolled by instructions having a format similar to that of horizon tal microinstructions. Our system synthesizes instructions by assum ing that all hardware resources should be controlled by independent instruction fields.
  Existing designs proof that the size of the code for these instruc tions may even be less than for conventional machine instructions[14]
  The MIMOLA system will be extended such that more conventional in struction formats may be studied as well.

Synthesis procedures contain a number of optimizations. Other optimizations are done by the designer and are described in section 2.4.

2.3.5 Microcode generation

Microcode generation is a key feature of the MIMOLA system. Its inclusion in the system is a big step towards the integrated design of hardware and soft/firmware. Program development and hardware development are done at the same time, using the same tools and the same language. No inconsistencies between several versions of hardware descriptions may arise. Microcode tools are available before any hardware implementation exists. This, of course, requires a retargetable microcode generator, i.e. a microcode generator that is table driven by the hardware description. The large number of projected target architectures inhibits the design of a separate code generator for every architecture. It is not obvious that such a code generator is feasible. Therefore we shall give some details about our code generator:

The code generator operates on the partially bound intermediate language tree and assigns hardware units to the knots of the tree and data paths to its edges. Function boxes are assigned to logic and arithmetic operations, memory ports to read and write operations. Control information is assigned to the function select inputs of hardware units. If a direct data path is missing (say, a path between a function box and a memory), new knots, representing the required detours (busses, multiplexers), are inserted in the flow trees.

Using the definition of Mallett[15], the different possibilities to assign hardware to the knots are called versions.

If the number of hardware resources is not sufficient for the execution of one block in one instruction, the required number of instructions is generated.

The code generator uses three phases:

During the first phase, all possible versions are computed. This is done by matching the patterns of the partially bound programs with the hardware patterns. Example: the completely bound assignment statement (2) is one possible version of the first, partially bound assignment statement in (1).

If a fast, specialized hardware exists for a certain pattern of the program, a special replacement rule may be included in the hardware description. This rule describes the program pattern and the fast version to be generated.

Resource conflicts are ignored during the step.

The second phase excludes all versions for which resource-conflicts and data-dependences lead to contradictions. For example, data dependences may require parallel execution of two completely bound assignments and resource conflicts may inhibit parallel execution. If the list of versions for a part of the input program becomes empty, the architecture is not capable of executing the program. This should not happen, if synthesis is correct.

The third phase selects a version for every partially bound operation and packs versions into microinstructions. Several algorithms for microinstruction packaging are known 15. Earlier versions of the MIMOLA system use an algorithm similar to the LINEAR algorithm. These versions use a lookahead of up to five microoperations[16]. Currently we are implementing a BRANCH AND BOUND-type algorithm. This algorithm is specially designed to minimize the execution time of the microprogram. During a BRANCH AND BOUND search the estimated freauencies of execution and the hardware delay times are used to select a combination of versions with optimized execution time. These versions are packed into microinstructions such that no resource conflicts occur and all data dependences are taken care of.

For common subexpressions the list of versions is identical. Therefore they do not generate resource conflicts and can be evaluated concurrently. Hence, recognition of common subexpressions is a by-product.

The output of the code generator is a tree, which represents the flow of data and of control in the target architecture. Binary microcode may be obtained by picking the instruction bits like apples. The flow trees, however, are very useful for several other purposes, because they do not only describe the flow of data and control, but they also contain hardware delay times and instruction frequencies. Therefore these trees are extremely valuable for performance evaluation.

2.3.6 Architecture assessment and preparation of design decisions

For quite a number of years computer design systems will be computeraided systems, not fully automatic systems. It will not be possible to forgo the ideas of human designers. The CAD-system has to provide a basis for design decisions and has to assess the value of human design decisions. Both functions are provided by the statistical analyser of the MIMOLA system. In order to prepare design decisions, the analyser -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.

The analyser also computes an estimated performance. A good estimate is the expected run-time of the programs contained. in the specification. This run-time is easily computed from the completely bound intermediate language trees because these contain instruction frequencies and hardware delay times.

Computation of costs uses default assumptions about the cost of RT-modules. However, costs may also be defined explicitly. This is useful if the design of the RT-modules has been continued to lower levels and precise information about the number of gates etc. is available.

2.4 Design Decisions

Synthesis procedures try to synthesize the fastest possible hardware. When all program transformations are completed, the designer has to take care about cost/performance trade-offs. To this end, the designer uses the information computed by the statistical analyser and tries to improve the cost/performance ratio by changing the boundary conditions and iterating the design. Changing the boundary conditions requires nothing more but editing a generated hardware description. Possible design changes include:

- Deletion of function boxes if other function boxes provide the same function.
- Reduction of the length of the instruction using coding techniques.
- Replacement of infrequently used hardware functions by routines.
- Deletion of data paths.
- Adjustment of the speed of the resources.
- Replacement of synthesized units by available units.

After several design iterations, cost/performance relations like Fig. 2 are obtained and an architecture may be selected.

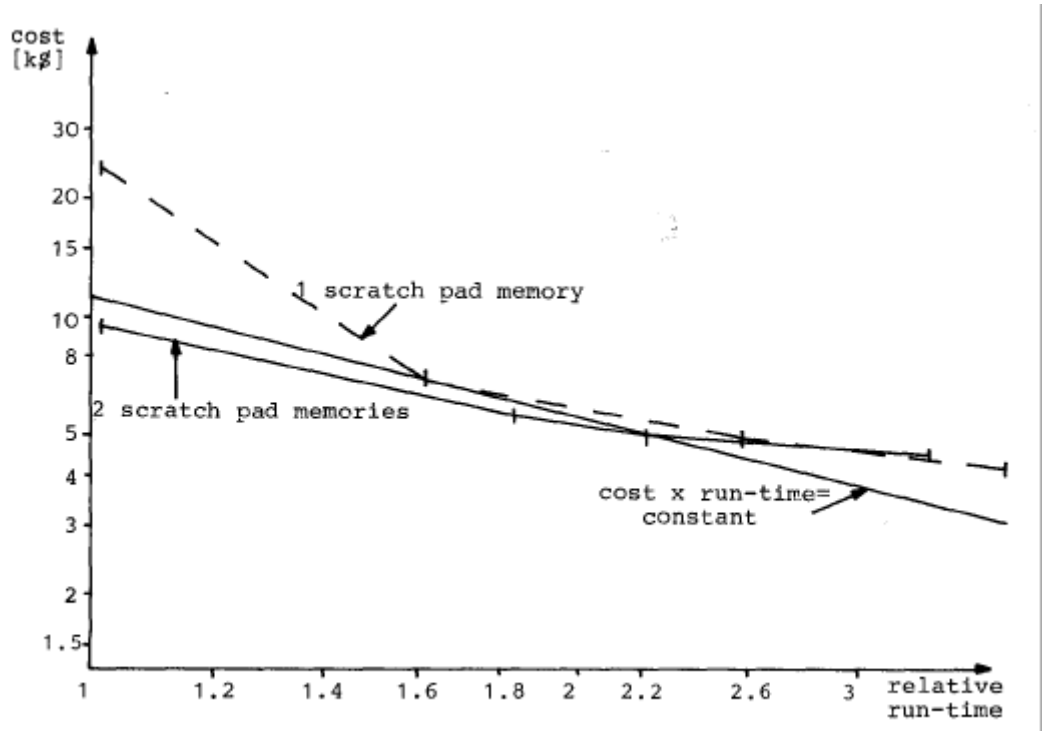

Fig. 2 Relations between cost and performance

3. Applications

3.1 Scientific Subroutine Package

The first application of the desian 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 to MIMOLA, estimated weights for the blocks were added. Fig. 2 shows cost/performance relations for a subset of this package. The dotted lines are valid for architectures with a main memory and one scratch pad memory, the solid line is valid for architectures with a main memory and two scratch pad memories. The number of ports for the scratch pad memories is fixed whereas the number of ports of the main memory varies. It is larger for the fast architectures on the left hand side. Obviously architectures with two scratch pad memories are more economical. They are cheaper because less main memory ports are required to obtain a certain performance.

An architecture with two scratch pads has been implemented in hard-

ware and is operating. The architecture contains two ALU's, an adder, a multiply/divide unit, a comparator and many data paths. Its complexity is similar to the complexity of a MODCOMP II minicomputer (equivalent of a mid-range PDP-11) to which it is coupled. For fixed point operations it is about 25 times faster than the minicomputer. Good utilization of the implemented parallelism has been observed. The code is more compact than for the MODCOMP II or the PDP-10[16].

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 thesis 17 does not only describe the architecture but also contains the complete microcode. This is an example of the integrated design of hard- and firmware.

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 software system.

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 TopDown 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. Huang, C.-L., "Computer-Aided Logic Synthesis Based on a New Multilevel Hardware Description Language", Ph.D. Thesis, State University of New York at Binghampton, 1981
7. Piloty, R., Barbacci, r-1., Borrione, D., Dietmeyer, D., Hill, F. and Skelly, P., "CONLAN - A Construction Method for Hardware Description Languages", Proceedings Nat. Comp. Conf. ,Vol.49, 1980
8. Knuth, D.E., "The Art of Computer Programming", Addison Wesley, 1975
9. Cattell, R.G.G., "Formalization and Automatic Derivation of Code Generators", Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, 1978
10. Schmidt, U. and Voller, R., "Die formale Entwicklung der maschinenunabhängigen Zwischensprache CAT", Informatik Fachberichte, Vol. 50, Springer, pp. 57-64
11. Hecht, M.S., "Flow Analysis of Computer Programs", North Holland, 1977
12. Kuck, D.J., "The Structure of Computers and Computations", p. 111, Wiley, 1978
13. Abraham, J.A. and Gajski, D.D., "Design of Testable Structures Defined by Simple Loops", IEEE Trans. Comp., 30, 11(1981), pp. 875-884
14. Marwedel, P., "The Design of a Subprocessor with Dynamic Microprogramming with MIMOLA", Informatik-Fachberichte, Vol. 27, Springer, pp. 164-177
15. Mallett, P.W., "Methods for Compacting Microprograms", Ph.D. Thesis, University of Southwestern Louisiana, Lafayette, 1978
16. Marwedel, P., "Hardware Allocation for Horizontal Microinstructions in the MIMOLA Software System", Report 5/80, Institut fur Informatik and Praktische Mathematik, Kiel, 1980
17. Krüger, G., "Entwurf einer Rechnerzentraleinheit fur den Maschinenbefehlssatz des SIEMENS Systems 7.000 mit dem MIMOLA-Rechnerentwurfssystem", Diploma Thesis, University of Kiel, Kiel, 1980