THE MIMOLA DESIGN SYSTEM: TOOLS FOR THE DESIGN OF DIGITAL PROCESSORS

PETER MARWEDEL

Institut für Informatik und Praktische Mathematik
Universität Kiel, D-2300 Kiel, W. Germany

ABSTRACT

The MIMOLA design method is a method for the design of digital processors from a very high-level bevavioral specification. A key feature of this method is the synthesis of a processor from a description of programs which are expected to be typical for the applications of that processor. Design cycles, in which the designer tries to improve automatically generated hardware structures, are supported by a retargetable microcode generator and by an utilization and performance analyzer. This paper describes the design method, available software tools and some applications.

1. INTRODUCTION

The increasing complexity of semiconductor chips allows raising the specification level at which chip designs start. We have seen this level rising from the transistor level to the instruction set level. The question of what should be implemented next on a single chip is still an open question. This question has to be answered on an architectural level and decisions can only be taken if the tradeoffs between several alternatives are known. However, there is a lack of tools for that level[1]. Therefore, architectural design decisions are too often based on poor information about the consequences and possible alternatives. For example, the RISC architecture uses a large silicon area for its on-chip registers[2]. It would be interesting to know if this area would have been better used for an on-chip cache, a second ALU or something else.

2. EVOLUTION OF THE MIMOLA DESIGN SYSTEM

The MIMOLA design system has been conceived as a tool for the design of hardware structures. Special care has been given to correctness issues, performance prediction, parallelism and flexible and portable software tools. The basic ideas of this system were first published in 1976, when G. Zimmermann presented a paper at a German conference[3]. There, he described the MIMOLA design method and the design language MIMOLA (= machine independent microprogramming language). This paper stimulated the work on a software system, which supports the design method. This MIMOLA software system 1 (MSS1) was described at the 16th Design Automation Conference[4,5]. The system was used for several designs[6,7,8]. As a result of the experiences we gained with MSS1, work on a new software system, called MSS2, was started in 1980.

In the MSS2 we implemented an improved handling of the control part of computers, an improved hardware allocator and many features which were not present in MSS1 (e.g. generation of diagnostics, detection of parallelism and simulation). In addition, we extended the MIMOLA language[9] and used better software engineering techniques (e.g. a precompiler which allows us to write modular PASCAL programs).

3. DESIGN SPECIFICATION

Design specifications should specify what the final product is supposed to do and not how it is supposed to do it. Therefore there is a tendency from structural design specifications towards functional design specifications. Functional design specifications open a larger design space than structural design specifications do. An additional advantage is the fact that the final hardware is guaranteed to be correct if it is automatically derived from the functional description by a correct design automation system. For an ideal DA system, this final hardware structure had to be optimal for a designated cost function.

The specification of an instruction set is a possible functional design specification. However, it still restricts the design space because it implies the existence of certain registers, address translation hardware, etc. Therefore the design should be specified at a higher level. The application program level is such a level. This

0738-100X/84/0000/0587$1.00 © 1984 IEEE

level is closer to the customers, who want to use the computer for certain application areas. Since more and more programs are written in a high-level language, this approach also opens a larger design space and allows us to utilize the parallelism which is offered by VLSI technology.

Due to the above reasons, application programs form an essential part of our specifications. Type and number of programs have to be selected such that they sufficiently represent the application areas of the projected computers. The operating system, compilers, editors etc. may be included in this set of programs. The programs specify, which type of a processor is to be designed. If an algorithm for the rotation of pictures is used, for example, then a picture rotation processor will be designed. In order to design more general machines, a larger set of programs or algorithms has to be used. The final architecture will be structured according to the overall structure of the applications.

The application programs have to be written in MIMOLA. MIMOLA may be used for the description of algorithms on a PASCAL- (or ADA-) like level, i.e. as a high level language (HLL).

Example:

```
PROGRAM mimola_hll;
 DECLARE
  CONST bias = 7;
  TYPE  .integer = .BIT(31:0);
  VAR   p: .integer,
        q: .integer:=0;
  BEGIN
    p:= bias;
    q:= p "+" q
  END.
(* .BIT(31:0) denotes a
 bitstring of length 32;
 operators are enclosed in " " *)
```

Parallel execution is requested if statements are separated by a comma. This is one of the reasons why MIMOLA may be used as a high-level microprogramming language (HLML).

Example:

```
(* same declaration as above *)
BEGIN
  p:= bias,
  q:= bias "+" q
END.
```

Hardware design using RT-level hardware primitives is only possible if the programs are converted into RT-level programs. RT-level programs may also be written in MIMOLA.

Example:

The statement part of the last example could be equivalent to:

```
BEGIN
 SM(15).BIT(31:0):= 7,
 SM(16).BIT(31:0):=
  7 "+" SM(16).BIT(31:0)
END.
(*SM(15) denotes the contents
 of location 15 of memory SM.
 In MIMOLA the names of all
 memories start with a capital S *)
```

The replacement of high-level language elements by equivalent RT-level elements is defined by replacement rules. Declarations of constants, types and variables are shorthands for the definition of such rules. The replacement of PROCEDURE-calls, FOR-loops etc. requires the specification of corresponding rules in the design specification.

Example:

```
REPLACE
   GOTO L&name
WITH
   RP:= L&name
END
(* L&name is a parameter,
 which matches labels.
 RP is the program counter *)
```

Standard replacements are collected in a library.

The simulator of the MSS2 may be used to validate the correct function of RT-level programs.

The final hardware structure need not be structured according to the structure of rarely executed parts of the application programs. Therefore dynamic frequencies of execution are important parameters, when cost/performance tradeoffs are made. Thus estimated dynamic frequencies of execution have to be included in the programs. They may be obtained, for example, by mathematical analysis[10] or by using soft-, hard-, or firmware monitors. In these cases the frequencies have to be added to the programs manually. Our simulator also computes these frequencies and there is a utility program in the MSS2 which copies

frequencies obtained by simulation to the application programs.

The specification may also describe available hardware resources. There are two sets of resources: hardware modules and interconnections. The following is an example of the description of an ALU module[11] (this example makes use of changes to the present syntax):

```
MODULE B74381
  (in left,right: .BIT(3:0);
   in select: .BIT(2:0);
   out result: .BIT(3:0));
BEGIN
   result:= CASE select OF
            0: 0;
            1: right "-" left;
            2: left  "-" right;
            3: left  "+" right;
            4: left  "XOR" right;
            5: left  "OR"  right;
            6: left  "AND" right;
            7: -1
            END
END
```

The resource specification may completely describe a machine. Complete machine descriptions are required if the MSS2 is used for microprogram generation, performance prediction or generation of diagnostics. In the case of a hardware design, the resource specification will be empty or contain just the types of hardware modules. It is then up to the synthesis part of the MSS2 to create the required modules and interconnections.

We have now discussed the four basic parts of the design specification:

1. typical application programs,
2. replacement rules for high-level language elements,
3. dynamic frequencies of execution,
4. description of available hardware resources.

We shall see how this information is processed in order to generate a hardware structure.

4. THE MIMOLA SOFTWARE SYSTEM

In contrast to MSS1, MSS2 consists of a number of independent programs, called components[12]. Fig. 1 shows available components.A common intermediate language (IL) is used for the communication between components. This language is the external re-

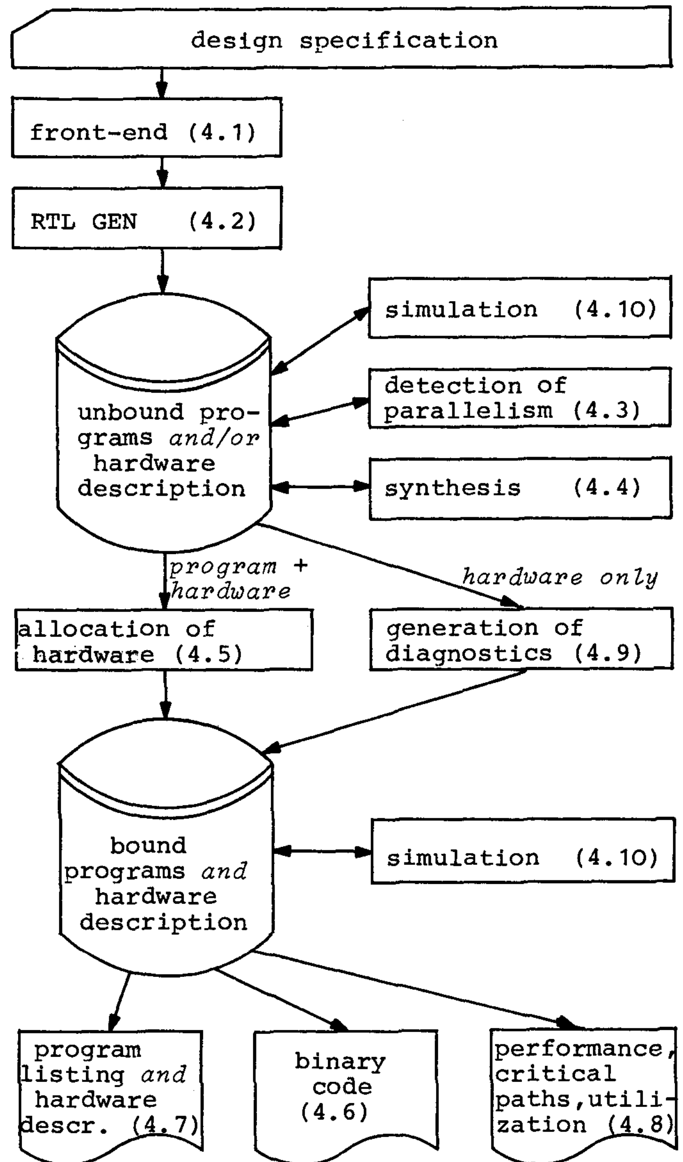

Fig. 1: Components in the MIMOLA software system (MSS2) (numbers refer to the corresponding sections in the text)

presentation of abstract syntax trees which describe the flow of data and control. It made the segmentation of the MSS2 into components possible and thereby allowed us to implement more features than those which were present in the MSS1.

We shall now describe the individual components of the MSS2:

4.1 Translation to intermediate language

The front-end of the MSS2 translates programs from MIMOLA into the intermediate language and performs necessary compile-time checks (syntax check, proper declaration of variables, etc.).

4.2 Mapping to register-transfer level

Component RTLGEN replaces all high-level language elements by an equivalent set of RT-level statements. It also computes the number of bits which are required for the data objects (constants, function boxes etc.). RTLGEN also performs some optimizations (e.g. compile-time evaluation of constant expressions).

4.3 Detection of parallelism

We put special emphasis on the design of machines with instructions similar to those of horizontally microprogrammable machines because we found that these machines offer performance improvements over classical sequential machines[6] and because the control structure of these machines is less complex than for machines with several instruction streams. Furthermore, there is an underlying parallel microarchitecture for most von-Neumann machines as well as for non-von-Neumann machines. Therefore tools for the design of parallel microarchitectures are needed for almost all machines.

Hardware synthesis and code generation are simplified by a component of the MSS2, which detects statements, which may be executed in parallel. All such statements are put into parallel blocks. The transformation is based upon an evaluation of control flow and data dependence.

Example:

The following sequence of statements:

```
SM(15):= 3;
SM(16):= 7 "+" SM(15)
```

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 the second statement ('statement substitution'[13]):

```
SM(15):= 3,
SM(16):= 7 "+" 3
```

(* In MIMOLA, statements in a parallel block are separated by a comma *)

At present we do not unfold FOR-loops or copy procedure bodies. We assume that there is a potential data dependence between references to CALL-BY-REFERENCE procedure parameters and other memory references. This assumption seems to be an important limit to parallelism. This limit could be avoided by global data-flow analysis.
Using the present approach, the detected parallelism normally allows us to keep about 8 memory ports and 4 ALU's busy.

Since parallel execution may also be defined by the designer explicitly, the automatic detection of parallelism is an optional step.

4.4 Synthesis

The synthesis component of the MSS2 generates an RT-level hardware description for a computer such that the parallel execution of the statements in parallel blocks is possible. To this end we compute the following architectural parameters:

- Number of input and output ports: memories with many ports may be designed. Synthesis procedures compute the maximum number of read- and write-operations in a block in order to create the required memory ports.
- Number of function boxes: function boxes are created such that parallel execution of all functions in each of the blocks is possible.
- Paths (wires): the required number of paths is generated.
- Instruction format: it is assumed that the hardware units are directly controlled by an individual field of the instruction ('direct encoding'). The required fields are created. The resulting code is compact[6].

In order to increase the number of operations executable in parallel, we do not break conditional statements down into conditional jumps and unconditional assignments.

Example:

The conditional statement

```
IF condition THEN SM(O):= 1 FI
```

may be executed in parallel to others if the hardware of Fig. 2 is used:

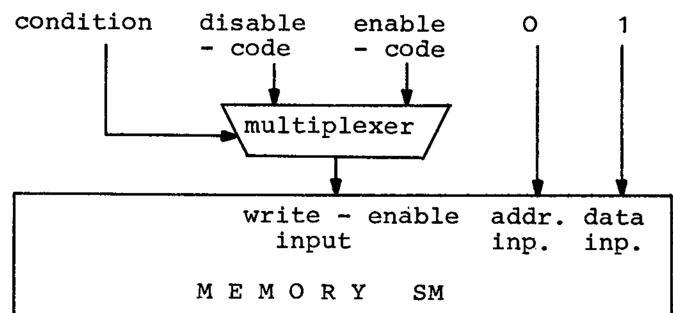

Fig. 2 Hardware implementation of conditional assignment

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 that of the description of MODULE B74381 above. No attempt is made to automatically generate a gate-level description.

The CMU-DA system and the MIMOLA synthesis system have many features in common. The input to our synthesis procedures corresponds to Carnegie-Mellon's value_trace and our synthesis algorithm is similar to the EMUCS algorithm[14]. In contrast to the CMU system, however, we do generate the data and control paths at the same time and found this to be quite valuable.

Details about the synthesis algorithm used in MSS1 are described in a paper by Zimmermann[15]. This algorithm has been adapted to MSS2. Currently we are working on an algorithm using more 'global' knowledge about the application programs.

Our synthesis procedures are designed to produce a single processor with a single clock. If a set of asynchronously coupled processors (e.g. a pipelined machine) is to be designed, each member of this set must be designed independently.

The instruction format of the designed machines is a horizontal microinstruction format. It remains to be seen if conventional instruction formats should be generated and if this may be done automatically.

4.5 Generation of bound programs

In order to allow performance predictions and code generation for the designed hardware, application programs are bound to hardware resources. This is done by the allocator of the MSS2. It's task corresponds to that of a compiler. In contrast to normal compilations however, we do not only generate binary instruction patterns but we transform programs such that every used hardware resource is shown. This simplifies for example the generation of resource utilization statistics.

Example:

Fig. 3 shows the graphic respresentation of the assignment

SM(15.BIT(7:0)).BIT(31:0):= 7.BIT(31:0)

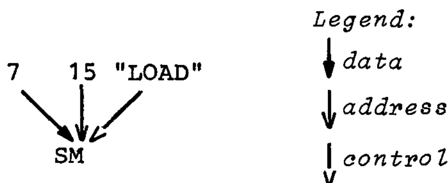

Fig. 3 unbound assignment (required number of bits not shown)

Fig. 4 shows the graphic respresentation of a corresponding bound assignment:

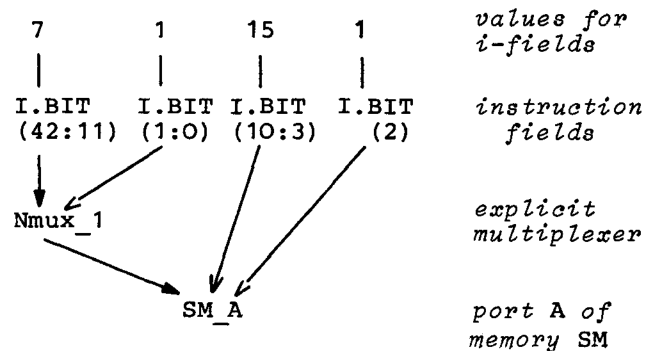

Fig. 4 bound assignment

Bound programs must be generated for a large set of hardware structures. This is possible only if the corresponding component of the MSS2 is retargetable (see Ganapathi[16] for a discussion of retargetable compiler code generation). We don't speak of a compiler-compiler because the compiler-compile phase is unimportant in our system. The task of writing an allocator is complicated because it has to be retargetable, allow rapid changes of the target and handle parallelism.

The allocator has been implemented as two components of the MSS2. The first component tries to find possible bindings for the application programs, i.e. tries to find function boxes for the operators and memory ports for read and write operations. For each (unbound) statement there may exist several bound versions of the statement (the concept of versions has been introduced by Mallett[17]). Therefore this first component of the allocator has been called version-generator. If expressions cannot be evaluated in one instruction, the version-generator generates the required assignments to temporaries.

The second component of the allocator tries to select and pack versions such that the required execution time is minimal. Therefore this component is called compaction-component. Compaction is complicated by a possibly large number of versions.

4.6 Generation of binary instructions

Instruction bit patterns may be easily obtained by scanning bound programs for instruction fields (cf. Fig. 4).

4.7 Translation to MIMOLA

MIMOLA is general enough to express bound programs and therefore bound programs may be translated from the intermediate lan-

guage into MIMOLA in order to improve readability.

4.8 Timing and utilization analysis

The structure description may contain a specification of delay times. These times are copied to the bound programs by the allocator. Analysis of these times is performed by a separate component of the MSS2. This component computes critical paths, utilization frequencies of hardware resources and (using the frequencies of execution) the expected runtime of bound programs[18]. This information is very useful for design iterations.

4.9 Generation of diagnostics

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 feasible[19]. Our system generates tests for stuck-at errors for the 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.

4.10 Simulation

Simulation is possible for unbound and bound RT-level programs. It may be used in order to validate their correct function and in order to compute dynamic frequencies of execution.

5. Design Iterations

We believe that the ideas of human designers will be needed in the design process for many years. Therefore our system is a computer-aided system, not a fully automatic system. When all program transformations are completed, the designer has to take care about cost performance tradeoffs. To this end, the designer uses the information which has been computed by the timing and utilization analyzer and tries to improve the design by changing the description of the generated hardware. Possible design changes include:

- Deletion of function boxes if other function boxes provide the same function,
- Replacement of infrequently used hardware functions by software routines,
- Deletion of data paths,
- Reduction of the instruction length by using coding techniques,
- Replacement of conditional assignments by conditional jumps,
- Adjustment of the speed of resources.

The modified hardware description is then used as a part of a new design description. This new description is used for binding the programs to the modified hardware. This time, temporaries may be required and the resulting bound program probably will be slower. Analysis of bound programs allows assessing the design decisions. The process may be repeated until the design space is explored and a final decision is made.

Note that binding programs for user-defined hardware structures is necessary during design iterations. This feature of the MSS2 is not present in other synthesis systems. This feature also allows us to use the MSS2 as a microprogram generation system.

6. APPLICATIONS

The design of the MSS1 and the MSS2 have been accompanied by applications of the design method in order to prove the usefulness of the method and in order to get a fast response from design-tool users.

6.1 Design of a processor for scientific computations

The first design started in 1976, when only few tools were available. The goal of this design was the development of a fast processor for scientific computations. These we considered to be represented by IBM's scientific subroutine package (SSP). A large portion of the SSP was translated from FORTRAN into MIMOLA. Manual detection of parallelism was used during this translation. Dynamic frequencies of execution were added to the programs. These could be easily estimated because in most cases they were closely related to matrix dimensions. A first pass through the software system generated a highly parallel, fast (and expensive) hardware structure. The joint distribution of operator usage served as a criterion for the selection of adequate multi-function units (e.g. ALU's) from a TTL-catalogue. The number of parallel read- and write-operations served as a criterion for the selection of multiport memories. The program was then manually bound. A second pass through the software system was then used in order to compute the utilization of data paths. After a manual reduction of the number of data paths, the hardware structure was complete and was built up in hardware. This hardware uses a 112-bit instruction in order to control 5 ALU's, 9 memory ports and the program counter. It is about 25 times faster than a mid-range minicomputer[6].

6.2 Design of a processor with an IBM-370 instruction set

In a second application, a machine with an IBM-370 compatible instruction set was de-

signed. A functional description of this instruction set was written in MIMOLA. Dynamic frequencies of execution were known from benchmarks and measurements. The initial pass through the software system again generated a highly parallel, fast and expensive machine. Several manual transformations of the bound programs resulted in a less expensive but even faster machine. These transformations were influenced by the generated timing/utilization statistics. The machine was slightly faster than a SIEMENS 7.750. The design of a faster machine would have been possible. Surprisingly, the size of the microcode for our machine amounts to only 15 % of the SIEMENS microcode.

This example shows that the design space was too restricted to allow the design of machines which are faster than existing ones. However, the design method was useful even in a case where the level of the design specification was lower than anticipated. This example also shows that a significant reduction of the design time can be achieved. This design was done by one student, who had no previous hardware design experience, as his masters thesis[7].

6.3 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 combines concepts of compiler construction and hardware oriented concepts. 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, R. Jöhnk, G. Krüger, L. Nowak and a large number of students made their contributions to the MIMOLA design system.

REFERENCES

[1] J.S. Mayo, keynote session 20th Design Autom. Conf., in: W. Myers, "Extend design automation systems", Computer, Aug. 1983, pp. 100-103.

[2] D.A. Patterson, C.H. Sèquin, "A VLSI RISC", Computer, Sept. 1982, pp. 8-22-

[3] G. Zimmermann, "Eine Methode zum Entwurf von Digitalrechnern mit der Programmiersprache MIMOLA", Informatik-Fachberichte, Vol. 5, Springer, 1976.

[4] G. Zimmermann, "The MIMOLA Design System: A Computer Aided Digital Processor Design Method", Proc. 16th Design Autom. Conf., 1979, pp. 53-58.

[5] P. Marwedel, "The MIMOLA Design System: Detailed Description of the Software System", Proc. 16th Design Automation Conf., 1979, pp. 59-63.

[6] P. Marwedel, "The Design of a Subprocessor with Dynamic Microprogramming with MIMOLA", Informatik-Fachberichte, Vol. 27, Springer, pp. 164-177, 1980.

[7] G. Krüger, "Entwurf einer Rechnerzentraleinheit für den Maschinenbefehlssatz des SIEMENS Systems 7.000 mit dem MIMOLA-Rechnerentwurfssystem", Diploma Thesis, University of Kiel, 1980.

[8] G. Zimmermann, "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 Descr. Lang., 1979, pp. 33-39.

[9] R. Jöhnk and P. Marwedel, "MIMOLA Language Reference Manual, Revision 2", Bericht des Instituts f. Informatik u. Prakt. Mathem., Kiel, 1984 (in print).

[10] D.E. Knuth, "The Art of Computer Programming", Addison Wesley, 1975.

[11] Texas Instruments, engineering staff, "The TTL Data Book for Design Engineers", Texas Instruments, 1977.

[12] R. Jöhnk, G. Krüger and P. Marwedel, "MIMOLA Software System 2 User's Guide", on-line documentation (available on request).

[13] D.J. Kuck, "The Structure of Computers and Computations", Wiley, 1978, p. 111.

[14] C.Y. Hitchcock and D.E. Thomas, "A Method of Automatic Data Path Synthesis", Proc. 20th Design Autom. Conf., 1983, pp. 484-489.

[15] G. Zimmermann, "MDS - The MIMOLA Design Method", Journal of Digital Systems, Vol. 4, 1980, pp. 337-369.

[16] M. Ganapathi, C.N. Fischer and J.L. Hennessy, "Retargetable Compiler Code Generation", acm computing surveys, Vol. 14, 1982, pp. 573-592.

[17] P.W. Mallett, "Methods for Compacting Microprograms", Ph.D. Thesis, University of Southwestern Louisiana, Lafayette, 1978.

[18] P. Marwedel, "Statistical Studies of Horizontal Microprograms", Proc. 5th Int. Conf. on Computer Hardware Description Languages, Kaiserslautern, 1981.

[19] K.-W. Lai, "Functional Testing of Digital Systems", Report CMU-CS-148, Carnegie-Mellon University, Pittsburgh, 1981.