AUTOMATED SYNTHESIS OF MICROPROGRAMMED CONTROL UNITS IN DIADES

Lian Yang, Marek A. Perkowski, David Smith, Ali Shamsapour

Department of Electrical Engineering, Portland State University
P. O. Box 751, Portland, OR 97207, tel. (503) 725-5411

ABSTRACT

This paper presents the automatic synthesis of microprogrammed control units in the DIADES design automation system. The optimization of the microcode is not performed separately in the control unit synthesis system, but is incorporated into the comprehensive process of data path scheduling and allocation and control unit design. Therefore, the optimum standard in this system is good performance of both data path and control unit. A new microprogrammed controller model that is suitable for design automation and formal description is described as well. This new model permits to design and optimize controllers ranging from a very basic microcontroller to a very complex one. The micro-sequencer architecture is retargetable. A Symbolic Intermediate MicroCode (SIMC) is generated as a "high level" intermediate microassembly language. In SIMC, some high level control constructs from the behavioral input language ADL of DIADES and GRAPH88 are preserved and they can be implemented either in hardware or converted into lower level microinstructions. An Object-Oriented HyperCard-based user interface has been designed to simplify the user's access to the system. It includes lessons, exams, helps, animated visualizations, and glossaries.

1. INTRODUCTION

In [3,23,27,31] different approaches to digital systems synthesis are discussed. These systems, however, emphasize the scheduling and resources allocation of the data path, and the control units are generated only in the sense of FSM or control memory contents generation. The micro-architecture of the microprogrammed control units is not discussed in these papers, as well as the various methods of mapping intermediate algorithm descriptions to optimal microprogrammed realizations. The synthesis of the control unit's structure is basically non-existent. The mutual trade-offs between the control unit and the data path are not addressed. The reason for this is probably the fact that in those systems, the control unit structures are predefined.

Papers on the microprogrammed control unit design concentrate mainly on the high-level microprogram compilation or on the process of synthesis of control memory with a *predefined micro-sequencer architecture* [7,8,23].

It can be then concluded, that in the traditional systems the data path and the control unit synthesis are performed **separately.** The reason for this is partly because there lacks a general and uniform intermediate form for both data path and control unit in these systems. Usually, data path synthesis takes place first and its description serves as one of the inputs to the control unit synthesizer. The optimization is separated into the data path optimization and the control unit optimization. This kind of a synthesis system could be called a data path-oriented system. Such systems are found in [6,23]. In the microprogram-based systems, the essential step of control unit synthesis is the *microcode compaction.* As the data path part has been synthesized, microcode compaction consists in extracting the parallelism in the data path as much as possible. This is suitable for a high-level structural description in which data path has been built up before the control unit, but is not suitable for an automated synthesis system in which both the data path and the control unit are synthesized concurrently, in an iterative process, that takes into account evaluation of both data path and control unit quality. In this paper we will present how the above issues are addressed in the comprehensive design automation system DIADES.

2. THE DIADES DESIGN AUTOMATION SYSTEM.

The research presented here is a portion of the DIADES automated design synthesis system developed at Portland State University. The DIADES system (DIgital Analog DESign automation) is intended to design multi-processor, digital/analog, electrically programmable and dynamically reconfigurable architectures. A simple example of this kind of architecture, based on Xilinx FPGA devices and applied for digital circuits emulation, is presented in [5]. DIADES has its own high level HDL called ADL (Algorithmic Design Language), which is a block-structured, behavioral/functional/structural language. The DIADES system takes ADL as its input and generates the circuit at the structural level as its output. The purpose of the DIADES project is to provide a CAD tool to help the circuit designer to fast prototype and explore various design options and styles quickly and more precisely than in the traditional methodologies. Like source languages of most existing high level synthesis systems which begin with a behavioral description, ADL has a high level syntax, similar to C [29,30]. The user does not have to be concerned with logic level implementation details of the system under design [24,25]. The generated output structural description is in the M language (TM Mentor), which serves as an input to the silicon compiler designed around various logic synthesis and layout tools interfaced to GDT and other Mentor tools. We can therefore call DIADES a **high-level preprocessor for a commercial silicon compiler.** Program in ADL is translated by an object-oriented expandable compiler TAG91 into the Lisp-based language GRAPH88, being an universal internal description form [29,30]. Descriptions in GRAPH88 are called Program Graphs *(p-graphs).* The initial p-graph description is similiar to "register transfer" control flow which permits for parallel operations as well as standard sequential operations, but which can include very complex assignment statements. P-graph includes basically two components: the control flow graph (*cf-graph*) and the data path description. This is different from a traditional data flow graph or a control flow graph because the p-graph combines the control flow description and the data path description into one description file, composed of several nested lists, each of which relates to a single high-level aspect of the circuit (it is somehow similiar to EDIF with this respect). During transformations, more information about structure and timing is added to the p-graph's lists. The parallel constructs of ADL and p-graph are: **FORK** (forking to parallel branches), **SIM** (simultaneous assignment statements execution), **DAND** (coincidential join of parallel branches), **DEXOR** (exclusive join of parallel branches), **DROP** (termination of local parallel branch), and **STOPADL** (global control termination in the parallel graph). These constructs make p-graphs functionally equivalent to parallel program schemata of Karp and Miller and more powerful than *the safe Petri Nets.* The p-graph being the translation from ADL is then subject to optimizing and other transformations. Therefore, the p-graph is a DIADES intermediate form that represents an implementation level of the entire High-Level Synthesis Subsystem.

In DIADES system, we employ powerful design tools in both data path part and control part synthesis [11-21,24-26]. The systematic optimization takes place at the p-graph level, which takes advantage of inherent parallelism of the algorithm itself and simultaneously performs the operation scheduling and allocation at this level. There are two design styles for control unit synthesis that are chosen by the designer: the FSM synthesis and the Microprogrammed Control Unit synthesis. They are implemented in the FSM Synthesizer [17-21] and the Microprogrammed Control Unit Synthesizer (MICUS), respectively. Some initial optimizations respective to both Synthesizers are performed first at the p-graph level. With respect to the target architecture it is very important that several controller and data path design styles are supported. While the controller design style provides various partitionings to modules and devices, the logic synthesis algorithms provide various mappings to configurable logic blocks (CLBs). Therefore, additionally to the well-known "disjunctive logic" synthesis tools we use a variety of EXOR-based and spectral-based tools optimized for our target programmable architectures [20].

3. COMPLEX MICROPROGRAMMED UNIT SCHEME IN DIADES

3.1. A General Microprogrammed Controller Model

This research was supported in part by NSF Grant MIP-9110772.

0-7803-0593-0/92 $3.00 1992 IEEE

In traditional microprogrammed control, microsequencing has been implemented using a variety of implicit and explicit techniques. Due to recent advances in VLSI technology, there is an increasing interest in more complex control sequencing. Devices such as Am29500 series has provided high sequencing capability such as microroutines, nested looping, and multi-way branching. But as far as the high level synthesis is concerned, there has been no literature indicating a mature method to synthesize such complex control schemes. In our system we have developed an approach that makes microprogrammed control unit synthesis process universal and formal, and we propose a general microprogrammed controller scheme. Fig. 1 shows the diagram of a *general microprogrammed controller model*, which is a new concept of a microprogrammed controller. To understand the diagram from Fig. 1, some concepts should be defined at first.

Definition 1. Internal control is the control of the internal behavior of the controller, such as *push, pop,* or *select address resources.* The internal control is also referred to as the *microcontrol.*

Definition 2. Internal controller is the controller that performs internal control functions. Internal controller is also referred to as the *microcontroller.*

Definition 3. Internal function is a set of microcontrol operations determined by the microinstruction format and internal states. The parameter of internal function is like the argument of the conventional function. An internal function along with its parameters determines a sequencing behavior. Its formal expression is: *f(x*) -> y,* in which **f** represents internal function, **x** represents the function parameter set, and **y** the *target address.* In each microinstruction, f and x should be specified and two fields are needed to hold **f** and **x** information.

The internal functions include *conditional jumping, looping, forking, calling subroutines,* etc. Each internal function is a way of directing the sequencing activity. The parameters of an internal function can be instantiated as the *branching addresses, loop counts,* etc.

In Fig. 1, the idea of the new scheme is to partitionate the internal function and its parameters into two combinational logic units. Traditionally, these two parts were incorporated in the control memory. This kind of schemes are found in the most current microprogrammed controller synthesis systems [23]. In these schemes, the control storage takes the entire internal and external control tasks, influencing the efficiency of the controler and the cost of the control memory. While in the scheme shown in Fig. 1, the control task of the controller is divided into three parts, each of the three parts has its particular control task, thus they all can be made the most of. This scheme is particularly suitable for complex microprogrammed control units synthesis that have very heavy internal control tasks. The tasks of the three parts are described in detail below:

(1) The *Internal Function Unit (IFU)* generates the control signal for internal control such as *push-stack, loop detect* or *forks invocation,* it selects also the address. The input of this unit is the addressing type field of the microinstruction which resides in the control memory.

(2) The *Internal Function Parameter Unit (IFPU)* generates the parameters needed by internal functions such as *loop count, branching adresses,* and *external mapping.* As for each micoinstruction, there is one or more *internal function parameters* associated to it, which would seem as if the input of this unit were the address of the current microinstruction. But actually, not all internal functions have real parameters. Sequential addressing has no parameter, for example. Thus the inputs of the parameter unit are the encoded parameter numbers of the internal functions. This will be further discussed in section 4.2.

(3) The central part of the controller is the *Control Memory (CM).* It generates the control signals to the data path and directs the internal control. Because the control memory only selects the internal control tasks but does not generate them, its burden is dramatically reduced. At the implementation level, the control memory includes three fields: *external control field, internal function number field, and internal function parameter number field.* These three fields can be further conmonly or separately encoded to reduce the CM size.

A particular feature of this scheme is that all the three internal control units do not necessarily exist for a particular problem. The diagram from Fig. 1 is more a *conceptual scheme* that corresponds to the SIMC internal language than a physical realization. The IFU and IFPU could be set to null for some applications. The key to this conceptual model is to provide a formal description of a microprogrammed control unit that is suitable for different goals.

DIADES provides the program to perform transformations between several induced schemes, which is automatically induced by the SIMC assembler program while evaluating the efficiency and the costs of the possible induced schemes. Basically there are three sorts of induced schemes:

"scheme 1." *Fully Distributed Scheme,* in which the internal controls are very complex and there exist many concurrent conditional jumps. The diagram of this scheme is shown in Fig. 1.

"scheme 2." *Semi-Distributed Scheme,* in which the internal controls are simple but there still exist many concurrent conditional jumps. The IFU would be set to null by the assembler, but the IFPU would be employed to store multi-way branching addresses. The diagram of this scheme is shown in Fig. 5.

"scheme 3." *Classical Scheme,* can be also called a CM-based scheme, in which the internal controls are simple and few multiway jumps exist. The diagram of this scheme is shown in Fig. 7. It stores all internal and external control information in the Control Memory.

In the controller scheme from Fig. 1, the microinstruction in the Control Memory employs a semi-implicit addressing scheme because the microinstruction residing in CM selects the formation of the next address only, but does not include this address in its fields.

This scheme is new to microprogrammed controller synthesis. In [9,10] a highly distributed addressing scheme was proposed, in which the control memory (ROM) stores the external control signals only, and the next address is generated by a sequencing PLA. In the sense of internal control distribution, our scheme is similar to it. But in the scheme from [9], the flexibility is almost lost as the internal control and the external control are totally separated and fixed in hardware.

3.2. Symbolic Intermediate MicroCode (SIMC) Formatting

In a simple microprogrammed scheme, the internal control function is very primitive and there is no need for the microcontroller. One or two MUXs would be enough for such a sequencing scheme. As far as complex control structures are concerned, the microcontrol functionality is much more complicated and a lot of parallelism exists. In this case, the task of the microinstruction (MI) is divided into two parts: *microsequence control* and *data path control.* The former is referred to as the *internal control* and the latter the *external control.* These internal functions and their microcontrol operations are listed in Fig. 2.

The purpose of SIMC is to symbolically represent all the internal functions from Fig. 2 in a clear, and easy to be translated, way. SIMC format is as follows:

(fname fparameter op)

In above, "fname" specifies the microfunction name for a MI. "fparameter" specifies the parameter that the specified microfunction takes. For example, (**Loop 3 op1**) specifies a **Loop** MI and the Loop count is **3,** which designates to execute the following loop body **3** times. **op1** is symbolic operation number of the data path control signals this MI holds.

As we should notice, there is no explicit branch and condition information specified in SIMC. In fact, the *fparameter* field of SIMC includes a lot of information related to condition numbers, jump addresses and branching addresses. Fig. 3 lists all kinds of SIMC formats together with their semantics. As we see in Fig. 3, SIMC has only explicitly declared the sequencing behavior of an MI. The data path operations are represented abstractly by *op* numbers. This is a reasonable solution since the semantics of *op* is defined detailly in the p-graph. When assemblying SIMC into the object microcode, everything the assembler needs to do is to simply map the semantics of each **op** to the respective encoded binary number.

3.3. Translating GRAPH into SIMC

SIMC can be used as an assembly programming language for some applications. A program that translates p-graphs into SIMC has been build. Theoretically, each p-graph node corresponds to a single microinstruction,which can be executed in a single cycle. The difference is, that in p-graph, the explicit addressing is employed. In SIMC, semi-implicit addressing scheme is employed. "semi" means, that when a branching is encountered, explicit branching addresses should be still declared. To translate p-graph into SIMC, the program simply locates each p-graph node in the virtual CM space and converts the p-graph node into the SIMC word, one by one. A depth-first searching algorithm was designed to perform this task. Employing this algorithm, the redundant join MIs are minimized, some other techniques to remove redundant jump MIs are additionally used.

4. SYNTHESIS OF THE COMPLEX MICROPROGRAMMED CONTROL UNIT (CMC)

In a control unit, two basic tasks are performed: generating control signals to data path and generating the next control state (for microprogrammed

control unit the next MI address). The task of generating next address in microprogrammed control unit is referred to as *microsequencing*. Because of the increasing requirements of high level microprogramming and the development of VLSI technology, more and more powerful microsequencers have been proposed. In [1], some multi-functional microsequencer chips are described.

In automated synthesis systems, microprogrammed controllers are widely employed. Most of them are, however, very simple and predefined microsequencer schemes are used. In [4] concurrent modular controller synthesis is addressed, but not quite compatible with our approach. Most high level control constructs are translated into simple two-way branching schemes or to the constructs corresponding to its fixed microsequencer schemes. In DIADES system, the following reasons ecourage us to implement complex microprogrammed control units.

1. ADL has powerful control statements such as loop, case, subroutine and forking. A fixed microsequencer scheme cannot take advantage of high level control capabilities of ADL.
2. DIADES is a performance oriented system, which means that the best performance of the algorithm is pursued. Therefore, the control unit scheme should be very flexible in order to match the particular algorithm.
3. A new, flexible, formal microprogrammed control unit model is proposed, which is suitable for efficient synthesis of complex microprogrammed controllers.

4.1. CMC Sequencer Scheme

In Fig. 1, the conceptual model of the CMC controller is displayed. Its microsequencer scheme varies from a very complex one including stacks, loop-counters, incrementers, and buses, shown in Fig. 4. It can be also very simple, like that shown in Fig. 5. Since **FORK** control construct exists in some applications supporting distributed systems, the Figure's 1 model can be expanded to a distributed controller model, in which several parallel subcontrollers *(nano-controllers)* are controlled by a main controller (*microcontroller* - [2]) The diagram is shown in Fig. 7. In high level language, the distributed control flow is derived by using forking. As we see this microprogrammed control scheme employs distributed internal controllers.

4.2. Microassemblying SIMC into the Object Microcode

SIMC includes all information on both internal and external control. At the implementation level, three units are employed to accomplish the control tasks. Therefore, an instruction in SIMC will be actually partitioned into three parts. Microassembly SIMC means to generate the *contents of the control memory*, the *internal function unit*, and the *internal function parameter unit* (**CM, IFU, IFPU,** respectively). Remember that both IFU and IFPU could be set to null, i.e. classic microsequencer scheme might be included.

The steps of the microassembly process are:

a. *Architecture Selection:* at this stage, the internal function and their parameters are accounted and different sequencer schemes are evaluated and compared. Then a microsequencer architecture is selected.

b. *SIMC Translation:* SIMC is translated into the object microcode within the selected architecture. At this stage, the truth tables of the three functional units, i.e. the IFU, IFPU and CM are formed.

Fig. 8 shows the relation between the SIMC fields and the contents of the three units. In Fig. 8, **f1, f2, f3** are the eventual fields in CM. Fields **f1** and **f2** are the *microfunction number* and the *microfunction parameter number*, respectively. The numbers are encoded as the inputs of the IFU and the IFPU PLAs, respectively (for simplification, we refer to here and below to PLAs, however the reader has to bear in mind that any multi-level or ESOP (Exclusive Sum of Products) synthesizer [20] from DIADES or Mentor's Autologic can be used to realize these logic blocks). The IFU and the IFPU are generated dynamically by scanning the SIMC words. Basically, the rules of microassembly are:

1. The content of IFU is dependent on what microcontrol functions the current SIMC has. A **full MFU PLA** is defined as the **MFU PLA** in which occur all possible internal functions listed in Fig. 1. The **current IFU PLA** is generated by cutting the rows and columns that correspond to the not occuring internal functions.
2. The content of the IFPU is the list of all *fparameter values* specified in SIMC. Its input is the *encoded fparameter number* that was generated by the assembler at the assemblying time.
3. The content of CM is divided into three fields **f1, f2, f3.** Field **f1** represents the *internal function type and serves as the invoking input of IFU.* Field **f2** represents the microfunction parameter number and serves as the invoking input of the IFPU PLA. For example, a MI (2 9 2) has the following meaning:
   (1) Its IF number is 2, assume it represents the conditional jump addressing type.
   (2) Its IFP number is 9, it is an index to the *associative target address* that is stored in the IFPU. In this case, the target address is determined by the *index number* and the *machine status* that is also the input of the IFPU.
   (3) Its external operation number is 2, which is an index to the activities of this microinstruction. As we mentioned earlier, the semantics of the operation is represented in the six-tuples of the p-graph.

5. THE HYPERCARD ENVIRONMENT

It is difficult for the new users of complex CAD systems to start using the system since they are overwhelmed by both the complexity of the new architectural and design concepts, as well as the amount and complexity of documentation - *"How to teach CAD tools to 40 students, having a single copy of a 20 volume tools documentation?"* Hopefully, new software products, such as the HyperCard from Apple, provide excellent tools for writing all kind of graphics database and learning environments, that can be created to help the novice or occasional CAD system users. Our current HyperCard Environment [22,28] of DIADES covers DIADES, ADL, and microprogramming, but we plan to expand it.

The goal of the Environment is to help students to prototype, integrate, and test their systems faster. They will be able to consider numerous variants, while under the present conditions, they have enough time to investigate a single possibility only. Moreover, the Environment will help in preparation of the design documentation and group communication, which are weak aspects of students' work. When finished, the modules linked in the HyperCard Environment will be of various kinds: **(1)** Descriptions of electronic hardware modules on many levels of specification (like counters, pipelined digital signal processors, microprogrammed controllers) and with a complete explanation (including graphics and animation). For instance, while reading a text on microprogramming the user can click the button to see a schematic of the controller, which can be next animated to show moving patterns of data flow from various logic blocks to the control register. **(2)** Simulation and test generation information for them. **(3)** Software design tools, formatting and interfacing programs. **(4)** Man-pages of user manuals with graphics. **(5)** Libraries of VLSI cells ready to be used in designs, with complete electrical, simulation, and testing information. **(6)** "Problem-solving" scenarios of standard approaches to solve problems. **(7)** "Brainstorming guides" to help investigate design problem spaces in order to find new designs. **(8)** Midterm Exams and a Final Exam. **(9)** Glossaries.

We believe that our Environment generates students' curiosity and excitement. It provides a learning environment in which the material can be learned quicker, in depth, and be more retainable. *Our goal of designing the environment is much more* than the simulation or the classical Computer Aided Instruction. We want to create an environment in which the students will be guided to learn by solving problems and by designing.

What do we mean by learning by designing? An experienced designer initially has some very vague ideas about his design when he starts to work on a new project. However, he knows a bunch of tricks, helpful rules, he has experience, he knows many existing designs, blocks and how they can interact, he disposes catalogs and schematics. He can modify existing designs, he can combine or simplify them. When the design is ready he can simulate it with simulators on many levels of accuracy or verify them algorithmically. He can use many design programs such as those for Boolean minimization of various microcontroller logic blocks in the design process. The goal of the Environment is to guide the inexperienced user through the "scenarios" helping him to find new ideas. Student is able to investigate various variants of microprogrammed architectures with an ease of playing video games. The main principle is that of objects and operations on them - everything is done on the "showing and selecting" principle.

6. CONCLUSION AND FUTURE WORK

In this paper, we presented the automated synthesis process of a complex microprogrammed controller. The program for microprogrammed control unit synthesis has been written in Common Lisp and was successfully tested on several ADL examples. The work on expansion of this scheme and

high level transformations is in progress, and work on better integration of the Environment is also necessary.

7. LITERATURE

[1] Andrews, M., "Principles of Firmware Engineering in Microprogram Control" *Computer Science Press,* 1980. [2] Baba, T., "Microprogrammable Parallel Computer MUNAP and Its Applications ", *The MIT Press 1987.* [3] Brewer, F.D. and D. D. Gajski, "Knowledge Based Control in Micro-Architecture Design", *Proc. 24th DAC, ACM and IEEE,* June 1987, pp. 203-209. [4] Bruck, R., Kleinjohann, B., Kachofer, T., and F.J. Ramming. "Synthesis of Concurrent Modular Controllers form Algorithmic Descriptions", *Proc. 23th DAC, ACM and IEEE,* June 1986. [5] Butts M.R., and J.A. Batcheller, "Method of Using Electronically Reconfigurable Logic Circuits", *U.S. Patent 5,036,473, July 30, 1991.* [6] Director, S.W., Parker, A. C., Siewiorek, D. P., and D. E. Thomas, "A Design Methodology and Computer Aids for Digital VLSI Systems", *IEEE Trans. on Circuits and Systems,* Vol. 28, No. 7, 1981. [7] Hopkins, W. C., Horton, M. J., and C. S. Arnold, "Target- Independent High-Level Microprogramming", *Proc. 18th Ann. Workshop on Microprogr.,* ACM SIGMICRO Newsletter, 1985, Vol. 16, pp. 137-143. *Design and Test of Computers,* **Vol. 2,** No. 4, 1985, pp. 33 - 43. [8] Nixon, J. F., and Schacchi,, S. R., and R. I. Winner, "A Microarchitecture Description Language For Retargeting Firmware Tools", *Proc. 19th Ann. Workshop on Micropr.,* ACM SIGMICRO Newsletter, 1985, 16, pp. 34-43. [9] Papachristou, Ch. A., "Hardware Microcontrol Schemes Using PLAs", *Proc. 14th Ann. Workshop on Microprogr.,* ACM SIGMICRO, 1981, Vol. 16, pp. 3-16. [10] Papachristou, Ch. A., "A Microsequencer Architecture With Firmware Support for Modular Microprogramming" *Proc. 15th Ann. Workshop on Microprogr.,* ACM SIGMICRO, 1982, Vol. 16, pp.105,113. [11] Perkowski, M., "A system for automatic design of digital systems". *Proc. of the FCIP Symp. INFORMATICA 74,* Bled, Yugoslavia, 7-12 Oct. 1974, paper 4.4. [12] Perkowski, M, "ADL - Source Language of the System for Automatic Design", in R. Marczyński (ed.) *Organization of digital computers and microprogramming,* Polish Scientific Publishers (PWN), Łódź 1976. Vol. 1, pp. 167-180. [13] Perkowski, M., and K. Jankowski, "Validation and Optimization of Control Automaton Programs in the System for Automatic Design", *Proc. of Int. Symp. - Fault Diagnosis of Digital Networks and Fault - Tolerant Computing,* Wisla, Poland, 1976, pp. 104-112. [14] Perkowski, M, "A Method of Validation of Parallel Programs in the System for Automatic Design of Black-Oriented Digital Systems", *Proc. 2nd IFAC Symposium on Discrete Systems,* Dresden, Germany, 14-19 March, 1977, Vol.2, pp.71-88. [15] Perkowski, M, "Automatischer Entwurf von MOS-LSI-digitalen Schaltungen in System DIADES", *Messen, Steuern, Regeln,* Vol. 6, 1979, pp. 346-350 (in German). [16] Perkowski, M, "Digital Devices Design by Problem-Solving Transformations", *J. Comp. Artif. Intell.,* Vol. 1, No. 4, 1982, pp. 343-365. [17] Perkowski, M.A., Smith, D., Driscoll, M., Liu, J., and J.E. Brown: "DIADES - A High-Level Synthesis System", *Proc. of the 1989 ISCAS,* May 9-11, 1989, pp. 1895-1898. [18] Perkowski, M.A., Driscoll, M., Liu, J., Smith, D., Brown, J., Yang, L., Shamsapour, A., Helliwell, M., Falkowski, B., and A. Sarabi: "Integration of Logic Synthesis and High-Level Synthesis into the DIADES Design Automation System", *Proc. of the 1989 ISCAS,* May 9-11, Portland, OR, 1989, pp. 748-751. [19] Perkowski, M.A., and J. Liu: "Generation of Finite State Machines from Parallel Program Graphs in DIADES", *Proc. of the ISCAS'90,* New Orleans, 1-3 May 1990, pp. 1139-1142. [20] Perkowski, M.A., "A Multi-Level Logic Optimization Program which Generates Optimal Mix of AND, OR and EXOR Gates", *PSU Report,* 1990. [21] Perkowski, M.A., Zhao, W., and D.Hall, "Concurrent Two-Dimensional State Minimization and State Assignment of Finite State Machines", *Proc. of the IEEE VLSI Design '92 Conference, Bangalore, India, 4-7* January 1992. [22] Shamsapour, A., "Hypercard-Based Learning Environment for DIADES", *M.S. Thesis,* PSU, 1990. [23] Sun, L. F., Liaw, J. M., and T. M. Parng, "Automated synthesis of microprogrammed controllers in digital systems", *IEE Proceedings, Computers and Digital Techniques,* Volume 135, Part E, No. 4, July 1988. [24] Smith, D., "Introduction to DIADES system", PSU DIADES Research Group Report, 1991. [25] Smith, D., "Forthcoming Ph.D. Dissertation", *PSU EE Dept,* 1992. [26] Smith, D., Perkowski, M.A., and K. Stanton: "A Distributed Processor Ensamble Methodology for the PSU-BOT", *Record of Northcon'91,* Portland, 1-3 October 1991, Session S1, paper 5. [27] Tricky, F. H., "A High Level Hardware Compiler", *IEEE Trans. on CAD,* DAC-6, 2 (March 1987), 259-269. [28] Von Pressentin, M., Csopenszky,, M. and M. Sand, "Portland State University: Teaching Electrical Enginering Using HyperCard", *Wheels for the Mind,* Vol. 4, No. 3., pp. 17,95-98, 1988. This is a short version, complete PSU Report also available. [29] Yang, L., and M.A. Perkowski, "Object-Oriented Design of an Expandable Hardware Description Language for a High-Level Synthesis System", *Proc. of the Intern. Conf. on System Sciences,* Jan. 7-10 1992, Stouffer Waiohai Beach Resort, Kauai, Hawaii. [30] Yang, L., "Object-Oriented Design of a Hardware Description Language for the DIADES Silicon Compiler System", *M.S. Thesis,* Dept. EE., PSU, 1990. [31] Zimmermann, G. "MDS-The Mimola Design Method," *Journal of Digital Systems* Vol. 4, No. 3, 1980, 337-369.

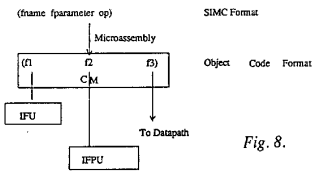
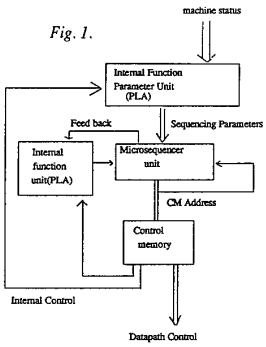


Fig. 8.

Fig. 1.



Internal Functions	Internal Controls
Call	push(stack), ps<=ps+1(stack pointer),Jump
Return	pop(stack), ps<=ps-1, Addr-regis.<=top-stack
Loop	push, push, continue
End-loop	zero-test, jump or continue
Fork	control transfer to nanoprogram unit
Jump	Addr-regis.<=Jump-address
Cjump	Addr-regis.<=targeted-address

Fig. 2.

Function & Parameters	Semantics
(L (Call F-name))	call subroutine
(L (Return))	(subroutine) return
(L (Loop L-count))	do following MIs L-count times
(L (End-l))	Loop ends, test loopcounter
(L (If (c*) (b*)))	conditional jump (single or multi)
(L (Goto L))	unconditional jump
(L (Para fork1.. forkn))	parallel execution of forks (n is not limited)
(L (Sequ op))	sequential addressing
(L (Map))	external address mapping
(Head)*	head of a segment of MI

* Head is not a function, it's a tag of a MI segment such as subroutine, fork.

Fig. 3.

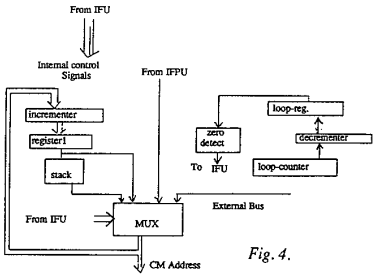
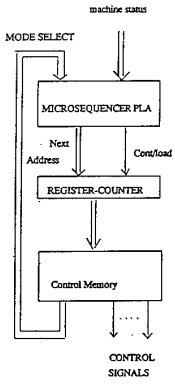


Fig. 4.

Fig. 5.



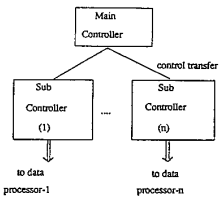


Fig. 6.

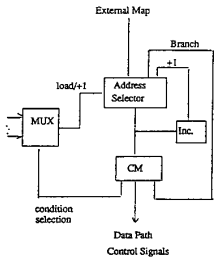


Fig. 7.