INTEGRATION OF LOGIC SYNTHESIS AND HIGH-LEVEL SYNTHESIS INTO THE DIADES DESIGN AUTOMATION SYSTEM

M. Perkowski, M. Driscoll, J. Liu + *, D. Smith + *,
J. Brown *, L. Yang +, A. Shamsapour, M. Helliwell *,
B. Falkowski *, P. Wu * +, M. Ciesielski #, and A. Sarabi,
Department of Electrical Engineering, Portland State University
P.O. Box 751, Portland, Oregon 97207, tel. (503) 464-3806 x 23.
Department of Electrical and Computer Engineering,
University of Massachusetts, Amherst, MA 01003, tel. (413) 545-0401.

Abstract— This paper presents a short description of the high level and logic synthesis stages in the digital design automation system DIADES. High level design, namely data path synthesis and control unit synthesis start from a parallel program-graph, the form of description that includes both the control-flow and the data-flow graph. While the data path is allocated and scheduled, the control unit is designed to be composed of either a microprogrammed units or Finite State Machines. The FSMs are minimized in two dimensions (states and inputs), assigned and realized in logic. Several logic synthesis procedures, respective to various design styles and methodologies, can be used to design combinational parts of state machines, microprogrammed units and data path logic. [1]

I. INTRODUCTION

The VLSI technology with which modern digital systems are designed has advanced at such a tremendous pace within the past few years that the engineer is being outstripped of his ability to design complex state-of-the-art systems. Initially, various CAD tools (circuit analysis, logic minimization, layout, etc.) were made available to aid the designer in concurring such increasingly more difficult design tasks. Technology has now reached a plateau, however, that calls for the realization of design automation systems in which *high-level synthesis is integrated with logic synthesis.*

The main objective of this paper is to present one way in which such integration can be achieved. We would also like to present the new, exciting *practical*

[1]+ Partially supported by Sharp Microelectronics Technology 1988 Research Contract. * Partially supported by 1986, 1987, or 1988 Design Automation Scholarship Award. Diades was also funded from various other sources including Warsaw Technical Univ., Polish Acad. of Sci., NSF, and Honeywell SSED.

opportunities for CAD given by *theoretical research* in some adjacent research areas **(microprogramming, scheduling, Reed Muller Forms, spectral methods).** The DIADES design automation system includes four design stages: *system level design, high-level design, logic synthesis and physical design.* The first one, as well as the entire system and the example of its application are discussed in the companion paper [22]. In this paper, we will, therefore, concentrate only on the high-level and logic synthesis in DIADES. The High-Level Synthesis in DIADES includes two stages: Data Path Synthesis and Control Unit Design.

II. DATA PATH SYNTHESIS

The DIADES data path generator, IMPLEM, assumes that there are no resource restrictions. The descriptive *program graph (p-graph) in language GRAPH88* is, therefore, mapped to a hardware structure in the abstract netlist language STRUCT, exactly as it is written in ADL language [12, 13, 14], or as it results from the previous high-level p-graph transformations [15]. It also does not try to increase the concurrency of operations. The designer, however, can do this manually by modifying his description.

The new version of DIADES includes a *scheduling tool.* After the descriptive format has been converted to p-graph format, the operations in the graph are scheduled for execution in specific control steps. The scheduling tool tries to make use of resources and increase the concurrency of the p-graph. The p-graph is analyzed and the order of execution of operations in it is determined. Those operations that can be executed at the same time are identified. If the resources are *constrained* then the maximum concurrency is not possible and some *trade-offs* are investigated. The Scheduler looks for the best sequence of operations, resulting in the shortest overall control program.

The DIADES Scheduler takes a *global approach.* A program is divided into blocks of straight-line code separated by control points. The control points include if-then-else and while statements. While the *local scheduling* looks only at single blocks of code (so called *linear programs*), the global scheduler can move operations from one block to other blocks. The DIADES Scheduler first constructs an operation/data dependency graph with the help of data synchronization statements contained in the high-level program. Then a classical *state minimization technique* [21, 25, 16] is used to determine the *Maximum Compatible Groups of Operations.* The heuristic rules are applied to limit the number of compatible groups as well as their size. The next step uses more heuristic rules to develop a *tree of possible schedules.* The tree is then searched using classical tree searching methods to determine the optimal schedule. We are currently investigating the heuristic rules to determine which of them give the best results. The output of the Scheduler, as the output of other High-Level Transformers, is still in the p-graph notation. All nodes of the p-graph are now assigned to the specific control steps that satisfy some constraints.

The current *Allocator* of DIADES does not do intelligent allocation. It rather allocates variables to registers and detects only few patters of other units such as adders. When necessary, multiplexers are used to implement the communication channels between the units. The improved Allocator version includes a sophisticated allocation ability. After scheduling, DIADES assigns variables to registers and operations to specific functional units. A lifetime analysis is done on the p-graph variables. The Variable Partitioner program selects the best mapping of variables to registers. Another partitioning algorithm, Operation Partitioner, maps the operations to the functional units. The other objective is to minimize the cost of multiplexers that connect register outputs to functional unit inputs.

After allocation, the design process splits into two separate processes. The data path unit is generated in program IMPLEM and then in MGEN. Specification of detailed connections between the elements is created. The output of IMPLEM is in STRUCT. This format is based on the p-graph. The functional units, registers, and input/output ports are represented by nodes, while the connections are represented by arrows. The STRUCT description is detalized by the program MGEN, which means that abstract descriptions of structural blocks are replaced with lower level descriptions of blocks and their connections. The output format is in netlist format called M-language (TM SCS Systems). For instance, the STRUCT description specifies the "abstract block" *BLOCK1 - COUNTER modulo 5 with loading input A, and clock CL.* The equivalent M-language description will specify all flip-flops and NAND gates of this counter and how they are connected, with an accuracy to each wire and gate input.

III. CONTROL UNIT SYNTHESIS

Two design styles are used in DIADES to design the control unit: the Microprogrammed Units [32, 8], and the Finite State Machines (FSMs) [6, 25, 4, 5, 11, 16, 26]. Both of them are designed starting from (parallel or serial) program graphs in the language GRAPH88 [6], the main internal data representation language in DIADES.

The Microprogrammed Unit Synthesizer, MICUS, generates first SIMC (Symbolic Intermediate MicroCode) language description for retargetability. Microin-

Figure 1: *The block diagram of the* FSM Synthesizer

struction compaction and optimization are necessary for efficient microcode. To translate SIMC further into object microcode a microassembler is used. Ultimately, a .TT formatted truth table of Control Memory for logic minimization is generated. Until now, a simple scheme to generate single-way branching vertical microcode was implemented, but parallel microprogrammed/nanoprogrammed scheme (Baba) with multi-way branching, loop counters, stacks and address arithmetic is under implementation.

The block diagram of the *FSM Synthesizer* is presented on Fig. 1. The p-graph includes the control-flow graph (cf-graph) and some data path specification. The cf-graph can either be mapped to a network of FSMs or microprogrammed units, or converted to a single FSM or microprogrammed unit.

High Level Program Graph Transformations can be optionally executed (see [15, 32, 24]). They serve to optimize the description, investigate area/speed tradeoffs, and also aid in don't care generation from invariants [23]. The state machines generated in the well-known design automation systems are either completely specified or (rarely) have few don't cares. They do not have all of the don't cares of transitions and outputs that could be generated from the knowledge of the system being designed. We introduced, therefore, a method of generating don't cares in FSMs from their high level specifications. The method generates many more don't cares than other methods using constraints that exist on the values of predicate signals and come to the control unit from the data path. For some design examples, our invariant analysis method evaluates the *invariants* and finds combinations of predicate values (predicates are the signals from the data path to the control unit) that have constrained values for some of the control unit states. Such invariants are then used to replace the corresponding next-state transitions in the control unit's state table with the don't care transitions. This is beneficial because a table composed of a large number of don't cares facilitates state minimization, state assignment, and logic minimization [26, 6].

Conversion/decomposition. One of approaches to design the control unit is a conversion of parallel cf-graphs to equivalent sequential graphs and next state machines, assuming their synchronous interpretation [6]. We use an algorithm that reduces the number of internal states generated. This problem is not automatically solved in the existing systems as well. The conversion of a parallel cf-graph to a sequential cf-graph can be used for the verification of the former [6, 26]. Some other methods for the verification of such programs are in [24]. In the general case, however, the ADL programmer is responsible for avoiding all deadlocks, variable conflicts, and other undesired phenomena characteristic to parallel programs. Various methods can be used to realize parallel cf-graphs as sequential circuits. The conventional FSM methods, however, such as minimization, assignment, and realization of excitation functions, can only be applied when the parallel cf-graph has either been **converted** to an equivalent sequential cf-graph or has been **decomposed** to a sequential cf-graph. The cf-graphs are then converted to FSMs and FSM-based design procedures are followed. Program FGEN generates non-disjoint formats of FSMs: .KISS transitions format of the well-known Kiss program, .STAB format of state tables, and .LISS serial cf-graph format for compatibility with some control unit tools under design. All those formats are nondisjoint, which means that cubes can be overlapping. Conversion from nondisjoint to disjoint fromat is done by program FDISC.

Two-dimensional state minimization is done by program FMINI. Since we can describe or generate an FSM with many don't cares, we are able to investigate some new design methodologies that have not yet been proposed. The minimization of the number of internal states of an incompletely specified FSM, for example, is a well known task but it has not yet been incorporated in the comprehensive design automation systems. We have found it to be useful in some applications. Moreover, this approach can be extended for the minimization of the number of input symbols [25]. Input minimization and state minimization are iterated until no further state joining is possible. We implemented two efficient programs for state minimization. One of them is used for completely specified machines and uses the well known algorithm of Ullman/Hopcroft. The other one, for incompletely specified machines, solves the covering/closure problem [16, 25]. Such a problem is the generalization of the well-known covering problem. The method of two-dimesional minimization of FSM, applied in our system generates an FSM with the combinational input encoder [25]. As any other combinational logic, this encoder is minimized with any of the existing in DIADES logic synthesizers.

Mealy/Moore Transformations. Since we want the user to have some decisions regarding trade-offs between speed and area, as well as shape of the FSMs, we give him additional tools to transform the machines (FTRAN). Each Mealy machine can be converted to an equivalent Moore machine. Similarly, each Moore machine can be converted to an equivalent Moore machine [19].

State Assignment. There are several algorithms for state assignment in DIADES, included in program FASS. Some of them make design with output decoder possible. In order to solve the dilemma between time of design and the quality of

Figure 2: *The logic design subsystem of DIADES*

the design better, we give the designer several algorithms again, so he can use some of them for exact optimization of small machines, and another one for large ones [26, 4, 5, 11]. One of the algorithms is rule-based, one is based on quadratic assignment and one on graph embedding to a hypercube. The last one was used for machines with more than 100 states, inputs and outputs.

Evaluation. To evaluate the variants of the design, several evaluation functions are used that do some approximate measurements on the resultant data of the minimizers as truth tables, netlists or logic equations. It is interesting to observe how much each of the stages contributes to area minimization, as well the symbiosis and the contradiction of the various minimizers.

IV. LOGIC SYNTHESIS

A logic design component exists in any of the comprehensive design automation systems, although in some of them (U.C. Berkeley, IBM) it is more developed than in the others. It was the DIADES assumption from the very beginning to include various tools for logic design. Therefore, the TANT design, multi-level networks and Ashenhurst decomposition were used, even in the earliest versions of the system, together with various heuristic and optimal PLA minimizers [9, 10, 11, 12, 13, 14]. The logic design tools have been modified many times, improved, and replaced in the past as the system was growing and crossing the Atlantic ocean.

The logic design subsystem of the current version of DIADES is shown in Fig. 2. Its inputs, coming from the control unit and data path design subsystems, are truth tables (.TT format), logic equations (.EQN format) and M-language netlists composed of only logic gates.

From the point of view of the entire DIADES system the logic design subsystem performs the following functions:

1. **Logic minimization.** Various logic minimization algorithms are used that minimize two-, three-, and multi- level networks from various kinds of gates, independently on the circuit's technology applied at the next stages.

2. **Decomposition and partitioning of logic.** Since logic minimization algorithms, especially those that attempt to find the exact minimum solutions, can be used only for a limited number of inputs/outputs/terms, various means are used in DIADES to partitionate input logic to functions of smaller dimensions. Another method is to partitionate the logic (array of cubes or netlist)

according to some heuristic graph theoretic and other criteria.

3. **Linking various descriptions.** Pieces of logic description can come from control unit design, state machines embedded in data path, data path, or glue logic. They need to be linked together, and possibly redesigned, to improve area cost, speed, testability, or other criteria.

4. **Format changes.** Interfacing various formats, to make our tools mutually compatible, to make them compatible to outside tools (U.C. Berkeley, U. Washington), and to allow for redesign.

5. **Redesign.** To improve logic descriptions by redesigning linked/partitioned/macrogenerated descriptions using other tools.

6. **Technology mapping.** To map to the CMOS library cells used for the given technology (SCS).

Truth tables (arrays of cubes) are used as inputs and outputs to programs for PLA minimization. Currently Espresso from U.C. Berkeley is used, together with PalMini [7], Umini [1], and KUAI-Exact [31, 23]. The last program permits to design PLAs with input and output decoders, and will use new input pairing scheme, where primary inputs can be shared between the two-input, four-output decoders and three-input, eight-output decoders. The decomposition program Decomp uses truth tables as input and generates decomposed truth tables [20]. We use both Ashenhurst and generalized (Sasao's like) decomposition, that we generalized for the case of multi-output functions with don't cares. State assignment methods are used for encoding minimized multi-valued symbols of connections between decomposed PLAs. Each of the PLAs generated in decomposition or partitioning can be realized with any logic design program discussed below, or mapped to M-language standard cells. We found that conversion from nondisjoint cube format to a disjoint one is an useful pre-processing step for many algorithms (program Disjoint).

Truth tables are also inputs to three programs that realize various multi- level circuits. TantMini [10] minimizes the TANT networks. Classical TANT networks, introduced by Mc Cluskey and Gimpel were single output, three-level NAND (NOR) networks where all primary inputs were positive. Our algorithm assumes certain extensions to the classical model: the TANT network can have an arbitrary subset of inputs positive or negative, the network is multi-output and don't cares are allowed. The algorithm generates the exact minimum solution, so functions with not more than six inputs/outputs are allowed. We are currently working on a new program that will apply Espresso-like expansion/reduction/reshaping loop to generate TANT-implicants, which will allow for obtaining minimal solutions (not exact) for larger functions.

Program NegMini [17] minimizes two-level networks from negative gates (negative unate functions). It has the same drawbacks as TantMini, so its new version is needed in the future as well. It seems to us now that exact minimization is impossible to achieve for networks of more than six variables, using more than two levels of AND and OR gates.

Another internal form of Boolean function representation is a *Generalized Reed Muller Form (GRM)* (it is an array of cubes as well, but represents EXOR of products of literals). The program Exorcism [3] finds minimal (not exact) solutions for networks up to 30 inputs/outputs. It generates Mixed-Polarity Generalized Reed Muller Forms. Such networks are very good with respect to their testability. The new program of Helliwell, Rmini, generates an exact minimum solution for Mixed-Polarity Generalized Reed Muller Forms (can be applied to 6 variables). A program to generate exact optimum Fixed-Polarity Generalized Reed Muller Forms as well as quasi-optimal fixed-polarity forms, Firm, is also under preparation [27]. RM2M translates GRM forms to M netlist.

Yet another general purpose format for Boolean functions is *Walsh spectrum* [2], which is created here for incompletely specified *multiple-valued input functions.* Program tt2Walsh does forward and program Walsh2tt does the inverse spectral transform. Again, disjoint cube represenation of SPF was found useful, and even fundamental. Spectrum is then used by the program Spede to find generalized spectral-based decomposition for selected types of standard cells.

The netlist in M includes gates that result from macrogeneration of data-path segments of higher level descriptions in language STRUCT. Such gates can have inputs equal logic 0 or logic 1. This calls for applying to them recursive logic transformations based on Boolean algebra, like $A * 1 = A$, or $A + 1 = 1$. Such transformations are executed in program Implem3, being a part of EXPO. The optimized network can then be optimized or redesigned with other programs.

Linking is performed by the program Link, while partitioning using program Partition. Program Eqn2tt translates .EQN format files to .TT format files, program Tt2eqn translates .TT format files to .EQN format files. Program Eqn2m translates .EQN format files to .M format files, program M2eqn translates .M format files to .EQN format files. Such format conversion programs enable linking, partitioning, and redesign of any kind of logic files.

The technology mapping is done by the heuristic, tree-searching, rule-based program EXPO [18, 29, 30].

M language format, together with truth tables and logic equations embedded in it is an input to SCS layout system, and related cell and macrocell design tools.

V. SYSTEM INTEGRATION

There are several basic principles of integration that we wanted to obey while designing the system.

1. Communication of all programs is through *user-readable* and editable ASCII files. All formats are relatively easy to learn.

2. In order to understand trade-offs better there are *several algorithms* for the same design/optimization problems.

3. The standard languages and the organization of the system permit not only for the variant driven synthesis mentioned in p.2, but they also permit the user to select one of several paths through the design flow in order to *create his own methodologies.* For instance, he can try or avoid some design stages, like Mealy-Moore or other p-graph or FSM transformations.

4. There are *several interface formats,* both on the input, on the output, and inside the system, in order for easier interfacing DIADES to other systems as well as to exchange examples with other systems. Several internal formats, like GRM forms or Walsh spectrum, allow for efficient implementations of new methodologies.

5. To provide some consistency, for most of the design tasks, the multi-purpose behavioral/functional/structural language ADL has been created. It is, however, not considered to be an "HDL language achievement" in itself, but rather a flexible and powerful notation for *quick prototyping of description notations* as they become useful for the synthesis programs when they are being added to DIADES.

6. The role of *don't cares and multiple-valued input functions* is systematically emphasized on all design stages. An attempt to execute all stages for all kinds of machines, and integrate these stages required solving many problems that are not known from the literature. For instance, how to minimize state tables with nondisjoint columns [19, 6].

7. It is our belief that *powerful and flexible control unit design techniques* are the key to the future successes of comprehensive design automation systems.

It is there, in the control unit design, where everything: the system level design, microprogramming, micro-architecture design, state machine design, data path synthesis, and logic design comes together, and also in the most complex and mutually related ways. Control unit design takes most of the design time. Therefore, DIADES considers control unit design to be the most important immediate and long-term task.

8. Other issues are *design for speed* and *design for testability.* We are working on several ways of achieving these goals in DIADES, few of them are implemented now. They include algorithms to design logic networks with few levels, but other than PLAs. This goal also causes our interest in designs that have many EXOR gates, like fixed or mixed Reed Muller Forms.

9. For such complex systems as DIADES, the teaching of the potential users is a key issue. As one of the industrial CAD authorities mentioned: *"teaching the engineer how to use these tools is becoming a most important issue from the practical point of view".* In the DIADES Group we are trying to address this problem by developing a HyperCard-based Help/Tutorial/CAI environment to teach about DIADES for Macintosh II [26], as well as video taped tutorials.

References

[1] Ciesielski, M.J,, Perkowski, M.A., and S. Yang, "Multiple-Valued Minimization Based on Graph Coloring", submitted to DAC'89.

[2] Falkowski, B.J., "Spectral Methods for Synthesis of Easily Testable Multiple-Valued Input Logic Circuits", Ph.D. Dissertation proposal, PSU, 1989.

[3] Helliwell, M., and M. Perkowski, "Fast Algorithm for Minimization of Mixed-Polarity Generalized Reed-Muller Forms", Proc. 25th DAC, 1988.

[4] Lee, E.B., and M. Perkowski, "A New Approach to Structural Synthesis of Automata" Univ. Minnesota, EE Dept., Report, 1982.

[5] Lee, E.B., and M.A. Perkowski, "Concurrent Minimization and State Assignment of Finite State Machines", Proc. 1984 Intern. Conf. on Syst. Man, and Cyb., IEEE, Halifax, Oct. 9-12, 1984.

[6] Liu, J., and M.A. Perkowski, "Generation of Finite State Machines from Parallel Program Graphs in DIADES", submitted to DAC'89.

[7] Nguyen. L.B.,, Perkowski, M.A., and N.B. Goldstein, "PALMINI - fast Boolean minimizer for personal computers", Proc. 24-th DAC, pp. 615-621, June 28 - July 1, 1987.

[8] Perkowski, M., "A system for automatic design of digital systems". Proc. FCIP Symp. INFORMATICA 74, Bled, Yugoslavia, 7-12 Oct. 1974, paper 4.4.

[9] M. Perkowski: "An example of heuristic programming application in the three-level combinational logic design". Proc. 3rd Symp. on Heuristic Methods. Polish Cybern. Soc., Warsaw, 25 Sept., 1976, Vol. 1. pp. 105-132.

[10] Perkowski, M., "Synthesis of multioutput three level NAND networks". Proc. on the Seminar on Comp. Aided Design., Budapest 3-5, Nov. 1976, pp.238-265.

[11] Perkowski, M.A., and A. Zasowska, "Minimal area MOS asynchronous automata", Proc. Intern. Symp. Applied Aspects of Automata Theory, Varna, Bulgaria, 14-19 May 1979, pp. 284-298.

[12] M. Perkowski, "Automatischer Entwurf von MOS-LSI-digitalen Schaltungen in System DIADES". Messen, Steuern, Regeln, No. 6, 1979, pp. 346-350 (in German).

[13] M. Perkowski, "Automatic design of digital MOS LSI circuits in system DIADES", Measurement, Automatics, Control, Nr. 6, pp. 226-228, 1979 (in Polish).

[14] Perkowski, M, "A system for automatic design of digital systems." Magyar Tudomanyos Akademia. Szamitastechnikai Es Automatizalesi Kutato Intezete. Budapest Tanulmanyok 99/1979, pp. 93-112. Hungary, 1979 (in Russian).

[15] Perkowski, M, "Digital Devices Design by Problem-Solving Transformations", J. Comput. Art. Intell. (Pocitace a Umela Int.), Vol. 1, No. 4., 1982, pp. 343-365.

[16] Perkowski, M., and N. Nguyen, "Minimization of Finite State Machines in SuperPeg". Proc. Midwest Symp. Circ. Syst., Luisville, Kentucky, 22-24 Aug. 1985.

[17] Perkowski, M., "Minimization of Two-Level Networks from Negative Gates", Proc. Midwest 86 Symp. Circ. Syst., Lincoln, Nebraska, 1- 12 Aug. 1986.

[18] Perkowski, M., Ming, L.J., and A. Wieclawski: "An Expert System for Optimization of Multi-Level Logic", IASTED Conference, Appl. Sim. and Modeling, ASM '87, Santa Barbara, CA, May 26 - 29, 1987.

[19] Perkowski, M., Uong, H., and H. Uong: "Automatic Design of Finite State Machines with Electronically Programmable Devices", Northcon '87, Portland 1987, paper 13/4.

[20] M. Perkowski, J.E. Brown: "An Unified Approach to Designs Implemented with Multiplexers and to the Decomposition of Boolean Functions", Proc. ASEE Nat. Conf., Portland, Oregon, June 19-23, 1988.

[21] Perkowski, M., Liu, J., and J. Brown, "Quick Software Prototyping: CAD Design of Digital CAD Algorithms", In G. Zobrist (ed.), "Progress in Computer Aided VLSI Design", Ablex Publishing Corp., 1989 in print.

[22] Perkowski, M.A., Smith, D., Driscoll, M., Liu, J., and J.E. Brown, "DIADES - A High-Level Synthesis System", Proc. 1989 ISCAS.

[23] Perkowski, M.A., Wu, P., and K. Pirkl, "KUAI-EXACT: A New Approach for Multi-Valued Logic Minimization in VLSI Synthesis", Proc. 1989 ISCAS.

[24] Perkowski, M.A., and J.E. Brown, Automatic Generation of Don't Cares for the Controlling Finite State Machine from the Corresponding Behavioral Description, submitted to DAC'89.

[25] Perkowski, M.A., and W. Zhao, Two-Dimensional Minimization of Controlling Finite State Machines: Input State and Internal State Minimization, submitted to DAC'89.

[26] Perkowski, M.A., Brown, J.E., Liu, J., and Zhao, W., "Design of Control Units as Networks of Finite State Machines in the High-Level Design Automation System DIADES," submitted to "Progress in Computer Aided VLSI Design," 1989.

[27] Sarabi, A., "Application of AI Methods for Fixed Polarity Reed Muller Forms Minimization", M. Sc. Th., Dept. EE., PSU, 1989.

[28] Shamsapour, A., "HyperCard Based Environment to Teach about High-Level VLSI Design Automation System", M.Sc. Thesis, Dept. EE, PSU, 1989.

[29] Wieclawski, A., and M. Perkowski, "Optimization of Negative Gate Networks Realized in Weinberger-like Layout in a Boolean Level Silicon Compiler", Proc. 23rd DAC, Albuquerque, June 25-27, 1984.

[30] Wieclawski, A., and M. Perkowski, "A Cost Function for Layout in A Boolean Level Silicon Compiler", Proc. Midwest 86 Symp. Circ. Syst., Lincoln, Nebraska, 1- 12 August 1986.

[31] Wu, P., "Minimization of Multi-valued Input, Multi-level Boolean Functions", Ph.D. Th. in preparation.

[32] Yang, L., Perkowski, M., and D. Smith, "Automated Synthesis of Microprogrammed Control Units in DIADES", submitted to DAC'89.