Efficient Encoding for Exact Symbolic Automata-Based Scheduling

Steve Haynal Forrest Brewer
Department of Electrical and Computer Engineering
University of California, Santa Barbara, U.S.A.
haynal@umbra.ece.ucsb.edu, forrest@ece.ucsb.edu

1. ABSTRACT

This paper presents an efficient encoding and automaton construction which improves performance of automata-based scheduling techniques. The encoding preserves knowledge of what operations occurred previously but excludes when they occurred, allowing greater sharing among scheduling traces. The technique inherits all of the features of BDD-based control dominated scheduling including systematic speculation. Without conventional pruning, all schedules for several large samples are quickly constructed.

1.1 Keywords

High-level synthesis, scheduling, BDD, automata

2. INTRODUCTION

The scheduling problem occurs across diverse areas of application from networking to manufacturing to high-level synthesis of digital systems (HLS). Scheduling which assigns operations to time-slots in a synchronous system subject to data and control-flow dependencies as well as resource constraints is a key component of many HLS systems. Consequently, solving this problem efficiently is a direct way to enhance the abilities of such systems.

Most solutions to the scheduling problem fall into two categories: i) heuristics and ii) integer linear programming (ILP). Heuristic schedulers (i.e.[1][9]) find good solutions for large problems quickly but suffer with tightly constrained problems where early pruning decisions exclude candidates leading to superior solutions. ILP schedulers (i.e.[3][4]) exactly solve scheduling but have difficulties with time complexity and control constraint formulation.

Heuristic and ILP scheduling methods produce a single schedule at a time. Finding this schedule, especially an optimal one, often becomes increasingly difficult as more constraints are added to the problem formulation. Symbolic methods, (i.e [2][6][7][8][10]) are often effective in finding exact solutions in highly constrained problem formulations. Furthermore, since all solutions are enumerated, post-process pruning can be used to apply additional constraints which may not have efficient formulation for general schedules. Further, symbolic methods allow much more efficient formulation of control dependencies and environmental timing constraints. However, with symbolic methods, the key to success is to reduce the representation size of the solution sets. Methods to accomplish this include adding additional constraints to the problem, pruning suboptimal candidates early in the search, and efficient encoding techniques.

An exact symbolic scheduling technique was presented in [7][8]. This method uses ROBDDs to describe scheduling constraints and compress solution sets. In this formulation, each operation in a CDFG (Control Data Flow Graph) is assigned a boolean variable for each time-step in the schedule. This variable indicates whether or not the operation is scheduled during that time-step. Constraints, derived from the CDFG and environment, are added to the construction. Guard variables are employed to distinguish control paths. Although this technique performed well, complexity problems arose for lengthy schedules. Worse, since every exact history for all viable traces is kept, the encoding efficiency declines for schedules with many complex alternative histories.

Symbolic ROBDD automata-based schedulers were described in [2][6][10]. In [6], an exact operand scheduling technique is presented for predefined datapaths. This technique allows operands to be lost and later produced again in order to find optimal schedules meeting tight memory constraints. In [2][10], system timing and synchronization requirements are encapsulated in finite-state machine (FSM) descriptions. All constraints are formulated as automaton and product machines are built and traversed. This product machine becomes prohibitively large for practical sized problems. Furthermore, causality (as checked for by causal validation in Section 4.2) is not confirmed.

In this paper we present an exact symbolic automata-based scheduler. Our immediate innovation is an efficient encoding and automaton construction which improves performance of exact symbolic scheduling techniques. Fundamentally, this technique groups together schedules with common although not necessarily identical histories when exploring the schedule solution space. This is accomplished using an encoding which only preserves whether or not an operation has been scheduled but not precisely when. In this way, we minimize the problems of [7] for long schedules and of [2][10] with regards to automata size.

We pursue an automata-based representation since it provides clear potential [2][10] for describing control and protocol-intensive cycle-varying systems. This ability is a key part of our long-term HLS goal. Further, we utilize exact symbolic ROBDD techniques because of their demonstrated success in finding all optimal con-

strained solutions [7][8]. We wish to apply these techniques to constrained critical portions of large scale system designs. Such systems are filled with complex subsystem interactions which may be amenable to Boolean automata representation.

3. CDFG BOOLEAN FORMULATION

We define a CDFG as a directed graph where nodes denote operations, forks or joins and arcs represent dependencies. Fig. 1 shows a simple CDFG. In this example, the directed arcs from operation 1 to the fork and join denotes a control dependency. Consequently, the resolution of the fork and join remains unknown until after operation 1 has been scheduled. It is important to note that operations 3 and 4 can be speculatively executed before the fork is resolved. In this event, operations 3 and 4 must both be scheduled regardless of the control resolution. The right-hand side of Fig. 1 shows this speculative transformation. Finally, the directed arc from operation 1 to operation 2 represents a data dependency. Hence operation 2 can only be scheduled after operation 1.

Figure 1. Simple CDFG before and after speculation.

In our formulation, data and control constraints are extracted from a user supplied acyclic CDFG. These constraints along with user supplied resource restrictions are used to construct a BDD-based boolean relation. Implicit state-traversal techniques are used to determine a valid schedule.

3.1 Encoding

Each operation *j* in the CDFG (excluding forks and joins) is encoded with exactly two (P_j, N_j) boolean variables. Table 1 describes the meaning of this encoding.

TABLE 1: *P* and *N* Variables

P_j	N_j	*Meaning*
0	0	*j* not scheduled previously and will not be by next cycle.
0	1	*j* not scheduled previously but will be by next cycle.
1	0	*j* scheduled previously but result will be lost.
1	1	*j* scheduled previously and result remains available.

This encoding is efficient since each minterm in the relation only contains information regarding what has been scheduled and what may or may not be scheduled in the immediate cycle. For instance, Fig. 2 shows possible valid histories which would allow operation 3 to be scheduled in cycle 4. Instead of enumerating all three possible histories when scheduling operation 3, we simply represent the commonality of these histories (operations 1 and 2 have been scheduled) in one term, $P_1N_1P_2N_2P_3N_3$.

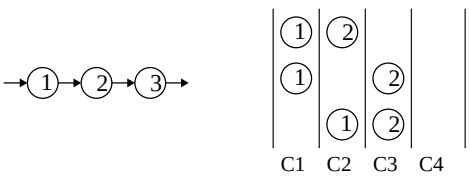


Figure 2. Some possible scheduling histories at cycle C4.

Control values to determine which side of a fork or join is used are produced by specific operations in the CDFG. These operations are encoded as described previously but have an additional pair of variables (P_{Gj}, N_{Gj}) associated with them. These guard variables indicate whether the produced control value is true or false.

3.2 Constraints

Six constraints, described below, are identified from the CDFG or supplied by the user and constructed as ROBDDs. These constraints are constructed in the complemented sense; each constraint describes situations which would not exist in a valid schedule given the encoding of Section 3.1. Once constructed, the product of the complement of all six constraints forms the desired scheduling transition relation.

3.2.1 Dependency Constraints

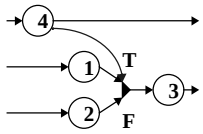
Dependency constraints impose an ordering of operation execution. To illustrate, if operation 2 requires a result produced by operation 1, it can only be scheduled after operation 1. In this case, it would be illegal to have any minterm in the relation containing P_1N_2 . Furthermore, an operation may have more than one dependency. In this case, all dependencies must be resolved before this operation can potentially be scheduled. In general, illegal dependency minterms are enumerated by,

$$\sum_{i \to j} \bar{P}_i N_j \text{ where } i \to j \text{ is a dependency arc in the CDFG.} \qquad (1)$$

A dependency arc in a CDFG may pass through one or more joins. This dependency only applies if the control is unresolved or the control has resolved in its favor. In general, illegal dependencies through joins are,

$$\sum_{i \to j} \left(\sum_{k \in \alpha} P_k P_{Gk} \right) \cap (\bar{P}_i N_j) \qquad (2)$$

where α is the set of all operations producing control values for joins through which the arc $i \to j$ passes. Furthermore, the value of the guard P_{Gk} is assigned the complement value of the join resolution for arc $i \to j$. For example, if $i \to j$ is a dependency arc passing through the true side of a join resolved by operation *k*, then P_{Gk} must be false. Fig. 3 illustrates this situation.



$$\left(\overline{(\bar{P}_4 P_{G4})} \cap \bar{P}_1 N_3 \right) + \left(\overline{(\bar{P}_4 P_{G4})} \cap \bar{P}_2 N_3 \right) \quad \text{is illegal.}$$

Figure 3. Dependency arcs passing through a join.

3.2.2 Resource Constraints

In practical digital system designs, there are only a fixed number of function unit resources available to perform a given task. Consequently, only *limit* operations of a given resource may be scheduled in any cycle. For example, if only one ALU is available and operations 1, 2 and 3 each require an ALU, it would be illegal to schedule any two or more of these operations in a single cycle. Hence, for every combination of *limit*+1 operations *i...k* from the set ρ of all type *resource* operations,

$$\sum_{i \ldots k \in \rho} \bar{P}_i N_i \ldots \bar{P}_k N_k \quad \text{is illegal.} \qquad (3)$$

3.2.3 History Constraints

As an initial simplification, we require that once an operation has been scheduled, its result will always be available in the future. This excludes encodings shown in row 3 of Table 1. In general,

$$\sum_{i \in operations} P_i N_i \text{ is illegal.} \quad (4)$$

3.2.4 Fork Constraints

Once a control value is resolved, a complete schedule must bifurcate into two traces with one trace scheduling a true control resolution and the other a false resolution. To ensure this bifurcation, we impose the constraint,

$$\sum_{i \in control} P_i (P_{Gi} \bar{N}_{Gi} + \bar{P}_{Gi} N_{Gi}) \text{ is illegal.} \quad (5)$$

3.2.5 Exclusion Constraints

If a control value has been resolved, it is unnecessary to schedule any operations unreachable under the resolved control for a particular trace. For instance, in Fig. 4, if operation 1 has been scheduled, then it is no longer necessary to schedule operations 2 or 4 in any trace with a false control resolution. Alternatively, any trace with a true control resolution must still schedule operations 2 and 4 but not operation 3. It is important to note that to support speculative execution, we must not exclude operation 2 or 4 entirely from any trace with a false control resolution but only exclude them from being scheduled in any false control resolution trace that hasn't yet scheduled them. In general, illegal minterms are,

$$\sum_{i \in \alpha} \left(P_i \bar{P}_{Gi} \cap \left(\sum_{j \in \beta} \bar{P}_j N_j \right) + P_i P_{Gi} \cap \left(\sum_{k \in \omega} \bar{P}_k N_k \right) \right) \text{ where,} \quad (6)$$

α is the set of all control value producing operations, β is the set of all operations unreachable in the CDFG if condition *i* is false and ω is the set of all operations unreachable in the CDFG if condition *i* is true.

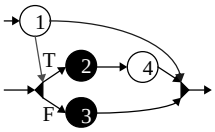


Figure 4. Exclusive operations once control is known.

3.2.6 Immediacy Constraints

It is desirable to have a constraint which implies an operation must be scheduled in the immediate cycle after another operation. This allows for multicycle and pipelined unit. Each stage of this type of unit is represented by a (P_i, N_i) pair. For multicycle units all stages are classified under the same resource constraint. For pipelined units each stage is classified under its own resource constraint. The general illegal form of this constraint is,

$$\sum_{i \to j} P_i \bar{N}_j \text{ where } i \to j \text{ is an immediate dependency.} \quad (7)$$

4. ENSEMBLE SCHEDULES

The final relation of Section 3.2 can be viewed as δ in an automata defined by the four-tuple (V, δ, $S_i(V)$, $S_f(V')$) where V is the finite, non-empty set of states, δ: $V \to V'$ is the next-state function and $S_i(V)$ and $S_f(V')$ are sets of initial and final states respectively. This viewpoint allows symbolic reachable state analysis techniques to be employed to determine exact valid ensemble schedules. A present state consists of a P_i vector and a next state consists of a N_i vector. Although we leverage symbolic reachable state analysis techniques, we are *not* bound to reachable state analysis in the sense that construction of schedules implies finding shortest paths between pairs of initial and final states. We need not fully explore the state space, and may use generated results to modify the transition relation during the scheduling process.

We define an ensemble to be a cycle-ordered set of set $\{S_0(V), S_1(V), ..., S_j(V)\}$. Let $S_0(V)$ be assigned the state with no knowledge of any scheduled operation (all P_i = 0). The set of reachable states on the *j*th iteration of the clock may be determined from this starting point by iteratively computing,

$$S_j(V') = \exists_{v \in V} [S_{j-1}(V) \cap \delta(V, V')] . \quad (8)$$

4.1 Completeness

Completeness is achieved when at some cycle *j*, there is a set $T \subseteq S_j(V)$ such that each $t \in T$ has scheduled the termination operation. (A termination operation, P_t which depends on all paths exiting the CDFG, is added for convenience.) Furthermore, all CDFG-imposed control paths must be scheduled by at least one $t \in T$. Although completeness is necessary for the existence of an ensemble schedule, it is not sufficient. A complete ensemble must contain a set of traces which are both complete and form a causal (deterministic) schedule. Note that given resource limits, even a complete set of traces on some cycle does not guarantee that any ensemble schedule is causal and can terminate on that cycle.

4.2 Causal Validation

Trace validation ensures that each trace is part of some ensemble schedule. Consider the CDFG of Fig. 5 with a one adder (solid circle) constraint. It is possible to hoist the addition operation 2 past the true fork and schedule it and operation 1 in one cycle. Likewise, another trace could hoist the false add (operation 3) past the fork and schedule it and operation 1 in one cycle. Together at cycle *j*=1, both these traces form a set T satisfying the conditions for completeness. It should be clear that this resulting ensemble schedule is not causal since two additions cannot be scheduled speculatively in the same cycle given a single adder constraint. Unfortunately, after removal of such traces, ensemble scheduling sets may no longer be complete. Thus, validation must continue until a fixed point is reached and all traces belong to some valid ensemble schedule.

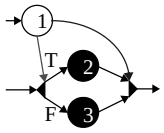


Figure 5. Trace validation ensures causal solution sets.

An ensemble is trace validated using the algorithm of Fig. 6. This algorithm explicitly describes trace validation only progressing backward through time although there is an additional symmetric portion for forward validation. Intuitively, this algorithm ensures that at each time-step, there is a modified transition relation which allows condition producing operations to be scheduled if and only if there are transitions with matching common histories for both true and false resolutions of the condition. Thus to be causal, operations speculatively executed assuming a true outcome must also have been speculatively executed assuming a false outcome. Forward and backward validation is performed on the entire ensemble set until a complete pass with no pruning of that time-step's transition relation occurs. This algorithm originated in [7] and is shown here modified for an automata formulation.

$S'_j(V')$==T;

do {

incomplete=FALSE;

for each time-step *j* to 0 {

$\delta'(S_{j-1}(V), S'_j(V')) = S_{j-1}(V) \cap \delta(V, V') \cap S'_j(V')$

for each conditional *k* {

$\delta_{validated}'(S_{j-1}(V), S'_j(V')) = \delta' \overline{P_k N_k} + \forall_{N_{Gk}} \delta' P_k N_k$

if $\delta_{validated}'$ is empty then exit;

}

if $\delta_{validated}' \neq \delta'$ then incomplete==TRUE;

$S'_{j-1}(V) = \exists_{v \in V'} [S_{j-1}(V) \cap \delta_{validated}' \cap S'_j(V')]$

}

if incomplete==FALSE then exit;

(Symmetric Forward Portion for time-steps 0 to j)

} while (incomplete==TRUE);

Figure 6. Trace validation algorithm.

4.3 Scheduling Instances

A complete and validated ensemble implicitly contains every schedule of cycle length *j*. It is possible to greedily pick a single schedule from this set. Beginning with $S_0(V)$, we choose a state at random. This present state maps to next states in $S_1(V)$ via a validated transition relation δ for that time-step. We greedily pick a valid next state with maximum P_i=1 implying peak utility. This process continues until the termination operation has been scheduled at time-step *j*. If the picked present state-next state mapping implies that a condition is resolved, then two traces, one for a true resolution and one for a false resolution, must be propagated forward from this point. Trace validation ensures that there will always be two such traces with opposite resolution to choose when a condition is resolved.

Greedy schedule selection is not the only possible selection method. Since a complete and validated ensemble implicitly contains every schedule of cycle length *j*, it is possible to pick a schedule that better suits the designers needs. For example, schedule selection methods which simplify control or minimize power could be applied at this point. Furthermore, there is no need to stop at cycle *j* once completeness and trace validation have been verified. Additional $S_j(V)$ may be added to the ensemble without the need to recheck completeness and trace validation. In this way, a schedule of specific length and exact construction can be found to suit a designer's requirements.

4.4 Schedules with Cycle-Length Constraints

The scheduling technique described in Section 4.3 uses no cycle-length constraint. If a cycle-length constraint is added, then ALAP bounds can be applied. These ALAP constraints prune the number of traces in $S_j(V)$ sets near the end by removing traces failing ALAP bounds. Furthermore, since traces have been pruned, it is possible to apply trace validation early for additional pruning.

Several other kinds of constraints including heuristic constraints are also applicable. However, the goal is to keep the smallest ROBDD sizes for the problem. With the various constraints experimentally tried, including ALAP, it turns out that since the initial encoding is relatively efficient, such pruning often reduces the efficiency of the scheduler by increasing the complexity of the representation even though the number of traces is reduced. Future work will be needed to sensibly apply appropriate constraints.

5. RESULTS

A tool was developed to demonstrate the feasibility of our scheduling technique. This tool utilized an in-house BDD package [5] and was run on a 141MHz SPARC Ultra with 416MB of memory. Results are described for several DFGs and CDFGs found in the literature. In all cases we are only applying the constraints described in Section 3.2 - no additional pruning strategies, heuristic or otherwise have been included. Furthermore, no prior knowledge of the cycle-length is assumed and hence no ALAP bounds are applied. In most cases, the efficiency of our encoding alone allows us to outperform similar symbolic techniques. All times are reported in seconds with the lower time indicating runtime without BDD ordering (preordered) and the higher time indicating runtime with BDD ordering (sifting). These times are inclusive of our entire scheduling process. (Constraint construction and other problem setup costs are not left out.) All operations are single cycle except for multipliers which are two-cycle and in some cases two-cycle pipelined.

5.1 DFG Results

The elliptic wave filter (EWF) and fast discrete cosine transform (FDCT) are widely accepted DFG benchmarks. Table 2 presents our results for various configurations of these benchmarks. EWF-1 is the standard 34 operation single iteration of the elliptic wave filter. EWF-3 is three and EWF-6 is six iterations of the elliptic wave filter unrolled. Here the efficiency of our encoding becomes apparent. Even though there are now as many as 204 operations and schedule lengths of up to 104 cycles, we are still able to produce all exact solutions to this problem in reasonable time. The small cycle-length gain achieved by loop unrolling suggests that this benchmark is tightly constrained with many scheduling traces sharing common histories and hence well suited for our encoding. A more challenging case is EWF-2x2 (136 operations) with two copies of the elliptic wave filter in parallel each unrolled twice. FDCT-1 (42 operations) is also a formidable benchmark with its inherent parallelism. FDCT-1x2 (84 operations) adds an even higher degree of parallelism by requiring two copies of FDCT be scheduled under the same resource constraints.

Our results compared to other symbolic techniques [10][8] show a ~100 speedup for frequently reported benchmarks EWF-1 and FDCT-1. Although [8] reports some exact solutions for EWF-3, we are unaware of any reports for exact solutions to EWF-6, EWF-2x2 or FDCT-1x2. Our formulation performs surprisingly well on DFGs. We attribute this to the fact that no validation step must be done since a DFG schedule consists of a single execution trace. Furthermore, the benchmarks indicate that our method works better under highly resource constrained situations which limit the possible combinations of common histories for scheduling traces. For example, FDCT-1 with a 1 ALU and 1 multiplier constraint finds a result in less time than when run with a 1 ALU and 2 multiplier constraint even though the final schedule is longer.

5.2 CDFG Results

Table 3 shows results for commonly referenced CDFGs KIM (24 operations, 2 conditions) and MAHA (18 operations, 6 conditions). All schedules for these benchmarks are found in just a few seconds. More challenging CDFGs, ROTOR and S2R from [7],

are shown in Table 4. The ROTOR (25 operations, 3 conditions) example performs a rotation of coordinate axes by angle θ. The interesting aspect of ROTOR is its constraint on trigonometric function lookup. Only one single-port memory lookup table containing sine values for arguments $0 \leq \theta \leq 90$ is available. Consequently, three conditionals and up to eight control paths are required to generate all required trigonometric values for rotation. The S2R (42 operations, 6 conditions) example translates spherical coordinates into Cartesian coordinates and basically consists of two modified ROTORs in parallel. Again, a resource constraint of one single-port memory lookup table is enforced. Our results are directly comparable to those in [7]. Our formulation shows the best improvement in cases with longer schedules. For example, the 12 cycle ROTOR result is achieved ~10 times faster (on identical computers) with our formulation.

TABLE 2: DFG Results

Benchmark	*Cycles*	*ALUs*	*Multipliers*	*CPU Time*
EWF-1	28	1	1	0.9/2.8
EWF-1	17	3	3	0.5/1.9
EWF-1	28	1	1*	1.1/3.5
EWF-1	17	3	2*	0.5/2.4
EWF-3	82	1	1	15.5/53.8
EWF-3	49	3	3	11.1/45.5
EWF-3	52	2	2*	12.2/55.1
EWF-3	53	2	1*	12.4/63.4
EWF-6	163	1	1	78.0/362
EWF-6	97	3	3	60.5/326
EWF-6	103	2	2*	64.3/364
EWF-6	104	2	1*	64.7/397
EWF-2x2	104	1	1	230/444
EWF-2x2	38	3	3	150/364
FDCT-1	34	1	1	8.5/22.7
FDCT-1	26	1	2	20.3/34.5
FDCT-1	18	2	2	9.0/23.2
FDCT-1x2	66	1	1	957/1067
FDCT-1x2	52	1	2	1343/1446
*Two-cycle pipelined multiplier.				

TABLE 3: CDFG Results

Benchmark	*Cycles*	*Add.*	*Sub.*	*Comp.*	*CPU Time*
KIM	6	2	1	1	0.9/3.3
KIM	8	1	1	1	1.5/3.5
MAHA	4	2	3	-	2.0/3.8
MAHA	5	1	1	-	1.5/2.4

TABLE 4: Rotor and S2R CDFG Results

Benchmark	*Cycles*	*ALUs*	*Multipliers*	*CPU Time*
ROTOR	12	1	(ALU)**	3.2/6.6
ROTOR	7	2	(ALU)**	3.4/6.8
ROTOR	10	1	2*	3.0/6.1
ROTOR	8	2	2*	3.8/6.9
S2R	14	1	(ALU)**	147/176
*Two-cycle pipelined multiplier. **ALU resource used for multiplication.				

6. CONCLUSIONS

This paper presented and demonstrated an efficient encoding for exact symbolic automata-based scheduling. The encoding preserves knowledge of what operations occurred previously but excludes when they occurred, allowing greater sharing among scheduling traces. This encoding, along with a novel automaton representation, allows us to use symbolic state traversal techniques to find all exact solutions to DFG and CDFG benchmarks found in the literature.

7. REFERENCES

[1] R. Camposano, “Path-Based Scheduling for Synthesis”, *IEEE Trans. CAD/ICAS*, vol. 10, no. 1, pp. 85-93, Jan. 1991.

[2] C. N. Coelho Jr, G. De Micheli, “Dynamic Scheduling and Synchronization Synthesis of Concurrent Digital Systems under System-Level Constraints”, *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 175-181, 1994.

[3] C. H. Gebotys and M. I. Elmasry, “Global Optimization Approach for Architectural Synthesis”, *IEEE Trans. CAD/ICAS*, vol. 12, no. 9, pp. 1266-1278, Sep. 1993.

[4] C.-T. Hwang and Y.-C. Hsu, “A Formal Approach to the Scheduling Problem in High Level Synthesis”, *IEEE Trans. CAD/ICAS*, vol. 10, no. 4, pp. 464-475, Apr. 1991.

[5] HomeBrew C++ BDD package
URL: ftp://bears.ece.ucsb.edu/pub/HomeBrew.tar.gz.

[6] C. Monahan, F. Brewer, “Scheduling and Binding Bounds for RT-Level Symbolic Execution”, *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 230-235, 1997.

[7] I. Radivojevic and F. Brewer, “A New Symbolic Technique for Control-Dependent Scheduling”, *IEEE Trans. CAD/ICAS*, vol. 15, no. 1, pp. 45-57, Jan. 1996.

[8] I. Radivojevic and F. Brewer, “On Applicability of Symbolic Techniques to Larger Scheduling Problems”, *Proc. European Design and Test Conf.*, pp. 48-53, 1995.

[9] K. Wakabayashi and H. Tanaka, “Global Scheduling Independent of Control Dependencies Based on Condition Vectors”, *Proc. 29th ACM/IEEE Design Automation Conf.*, pp. 112-115, 1992.

[10] J. C.-Y. Yang, G. De Micheli, and M. Damiani, “Scheduling and Control Generation with Environmental Constraints based on Automata Representations”, *IEEE Trans. CAD/ICAS*, vol. 15, no. 2, pp. 166-183, Feb. 1996.