An Area/Time Optimizing Algorithm in High-Level Synthesis for Control-Based Hardwares

Nozomu Togawa Masayuki Ienaga Masao Yanagisawa Tatsuo Ohtsuki

Dept. of Electronics, Information and Communication Engineering, Waseda University
3-4-1 Okubo, Shinjuku, Tokyo 169-8555, Japan Tel: +81-3-5286-3396 Fax: +81-3-3203-9184
E-mail: togawa@ohtsuki.comm.waseda.ac.jp

Abstract—This paper proposes an area/time optimizing algorithm in high-level synthesis for control-based hardwares. Given a call graph whose node corresponds to a control flow of an application program, the algorithm generates a set of state-transition graphs which represents the input call graph under area and timing constraint. In the algorithm, first state-transition graphs which satisfy only timing constraint are generated and second they are transformed so that they can satisfy area constraint. Since the algorithm is directly applied to control-flow graphs, it can deal with control flows such as bit-wise processes and conditional branches. Further, the algorithm synthesizes more than one hardware architecture candidates from a single call graph for an application program. Designers of an application program can select several good hardware architectures among candidates depending on multiple design criteria. Experimental results for several control-based hardwares demonstrate effectiveness and efficiency of the algorithm.

I. Introduction

Generally, if a control-based application program such as image coding and decoding, protocol processing, and encryption is implemented on an application-specific hardware, bit-wise processes or conditional branches can be executed concurrently and then the application program runs faster compared with being implemented on a micro processor. A high-level synthesis system which synthesizes such control-based hardwares should have the two features: (1) Hardwares for control-based processes including bit-wise processes and conditional branches can be synthesized. (2) More than one design candidates can be provided for a design specification given by a designer so that a designer can select several good designs among the design candidates depending on multiple criteria. Based on this idea, we have been developing a high-level synthesis system for control-based hardwares [5]. Given a behavioral description for an application program in C, the system generates hardware descriptions for the input application program. The system produces more than one hardware architecture candidates, all of which meet the given area and timing constraint. The system is composed of (i) a code optimizer, (ii) an area/time optimizer, and (iii) a hardware generator. The code optimizer generates a call graph and its corresponding control-flow graphs from an input application program. Based on them, the area/time optimizer generates more than one hardware architecture candidates. Finally, the hardware generator generates hardware descriptions.

Let us focus on area/time optimization among the processes (i)–(iii) above. Area/time optimization corresponds to operation scheduling for control-flow oriented hardwares and several researches on it have been reported as in [1]–[3], [6]. These approaches can synthesize general control-based hardwares including conditional branches. However, they only obtain a single solution and then they do not enumerate more than one solutions based on several criteria.

In this paper, we propose an area/time optimizing algorithm in the high-level synthesis system for control-based hardwares. Given a call graph whose node corresponds to a control flow of an application program, the algorithm generates a set of state-transition graphs which represents the input call graph under area and timing constraint. In the algorithm, first state-transition graphs which satisfy only timing constraint are generated and second they are transformed so that they can satisfy area constraint. The proposed algorithm has the following advantages: First the proposed algorithm can deal with control flows such as bit-wise processes and conditional branches since it is directly applied to control-flow graphs. Second the proposed algorithm synthesizes more than one hardware architecture candidates from a single call graph for an application program.

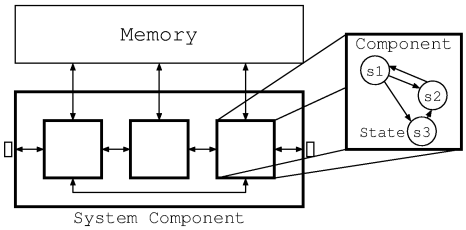

Fig. 1. Hardware model.

II. Definitions

A. Hardware Model

Fig. 1 shows a hardware model [5]. The hardware model is composed of *components* and a *memory*. A component is a state-transition machine and has states in it. Input and output signals of a component are given by primary inputs and outputs, inputs and outputs of another component, and a memory. The common clock signal is also given to each component. A component hierarchically has sub-components. Particularly the uppermost component is called the *system component*. A memory is composed of several memory banks.

B. Call Graph, Control-flow Graph, and State-Transition Graph

An application program is described by a set of functions in the C language. Each function corresponds to a component in our hardware model. A *call graph* $G_c = (V_c, E_c)$ represents function calls in an entire application program. Each node $v \in V_c$ in G_c corresponds to each function. If a function v_1 calls a function v_2, G_c has a directed edge (v_1, v_2). Each function $v \in V$ is represented by a control-flow graph. A *control-flow graph* $G_{cf} = (V_{cf}, E_{cf})$ represents a control flow in a function (Fig. 2(a)). Nodes in G_{cf} are grouped into start nodes, end nodes, operation nodes, memory nodes, condition nodes, join nodes, and loop nodes. If there is control flow from a node v_1 to a node v_2, G_{cf} has a directed edge (v_1, v_2). Particularly an incoming edge of a loop node coming from the loop ending node is called a *feedback edge* (Fig. 3(a)).

A control-flow graph can be modified so that it contains no join nodes. This graph is called a *modified control-flow graph* and denoted as $G'_{cf} = (V'_{cf}, E'_{cf})$. For example, if the control-flow graph in Fig. 2(a) is given, the modified control-flow graph in Fig. 2(b) will be obtained. Then a *state-transition graph* $G_{st} = (V_{st}, E_{st})$ is constructed based on a modified control-flow graph as in Fig. 3. A state $v_i \in V_{st}$ in a state-transition graph G_{st} corresponds to a subgraph G^i_{cf} of a modified control-flow graph G'_{cf} and has its *execution start point* which corresponds to a node in G^i_{cf}. For example, the node "`i>0`" is the execution start point in the state $ST2$ in Fig. 3.

A node v_i in a state-transition graph G_{st} is said to be feasible if the following conditions are satisfied for its corresponding subgraph G^i_{cf} in G'_{cf}:

Condition 1: G^i_{cf} has only one execution start point.

Condition 2: G^i_{cf} has no cycles.

If v_i is feasible, v_i can be executed within one clock cycle and

 ISBN 0-7803-5974-7

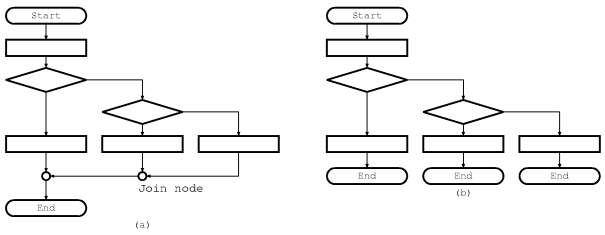

Fig. 2. Control-flow graph (a) and its modified control-flow graph (b).

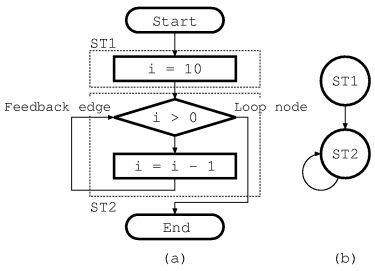

Fig. 3. Modified control-flow graph (a) and its state-transition graph (b).

functioned as one state in a state-transition machine.[1] If all the nodes in a state-transition graph are feasible, the state-transition graph is feasible.

If a modified control-flow graph G'_{cf} for a function f has no cycles, its state-transition graph G_{st} has also no cycles. In this case, we can obtain a directed path p in G_{st} in which the maximum number of states is included. The path p is called a *critical path* of f. The number of states in p is called a *path length* of f and denoted as $l(f)$.

III. Area/Time Optimizing Algorithm

A. Area/Time Optimization Problem

Let $a_{max}(f)$ be a maximum allowable area for a hardware realizing a function f in a call graph $G_c = (V_c, E_c)$. Area constraint of f is given by $a(f) \leq a_{max}(f)$, where $a(f)$ is area for a hardware of f. In our system, area is computed by adding up area for functional units, area for registers, and area for controlling states in a hardware. Area for functional units and area for registers are given from our hardware unit libraries. Area for controlling states is given by $[c_0 \times$(the number of the states in a hardware)$]$, where c_0 is a constant.[2] Similarly, let a_{max} be a maximum allowable area for a hardware of G_c. Area constraint of G_c is given by $a \leq a_{max}$, where a is area for a hardware of G_c. *Area constraint* is given by either $a(f) \leq a_{max}(f)$ for a function f in G_c or $a \leq a_{max}$.

Let $t_{max}(f)$ be a maximum allowable execution time for a function f in a call graph $G_c = (V_c, E_c)$ when an input sequence of data is entered into f. Execution time constraint of f is given by $t(f) \leq t_{max}(f)$, where $t(f)$ is an execution time for f. $t(f)$ is computed by multiplying a clock period by the number of clock cycles to run the input application program. Similarly, let t_{max} be a maximum allowable execution time for G_c. Execution time constraint of G_c is given by $t \leq t_{max}$, where t is an execution time for G_c. *Execution time constraint* is given by either $t(f) \leq t_{max}(f)$ for a function f in G_c or $t \leq t_{max}$.

Let $l_{max}(f)$ be a maximum allowable path length for a function f in a call graph $G_c = (V_c, E_c)$. *Path length constraint* of f is given by $l(f) \leq l_{max}(f)$. If $t_{c,max}$ is given as a maximum allowable clock period, *delay constraint* is given by $t_c \leq t_{c,max}$, where t_c is a clock period given by the maximum delay in a hardware realizing G_c.

Timing constraint refers to execution time constraint, path length constraint, and delay constraint. As timing constraint, not all of them are given but some of them are given depending on an application program. For example, execution time constraint is given in a protocol processing. In a huffman coding, path length constraint $l_{max}(f) = 1$ is given for each function f corresponding to each pipeline stage and also delay constraint is given as timing constraint.

Then area/time optimization problem is defined as follows.

Definition 1 (Area/time optimization problem) *Given a call graph, area constraint, and timing constraint, generate more than one sets of state-transition graphs representing the input call graph under the area constraint and timing constraint.*

B. The Algorithm

The proposed area/time optimizing algorithm is composed of three steps:

Step 1 (Initial state-transition graph generation). Based on each control-flow graph corresponding to a node f in a call graph $G_c = (V_c, E_c)$, generate a feasible state-transition graph in which the number of states is minimum. Assign initial hardware resources to f. For memory nodes, insert empty states so that correct state transition can occur.

Step 2 (Considering timing constraint). Partition a state of the state-transition graph in Step 1 repeatedly so that it can satisfy timing constraint. In this step, a hardware candidate which satisfies timing constraint is obtained but its area can be maximum.

Step 3 (Considering area constraint). While satisfying timing constraint, partition further a state of the state-transition graph in Step 2 or reassign hardware resources repeatedly. Enumerate a set of state-transition graphs as a hardware candidate if it satisfies timing constraint and area constraint.

The proposed algorithm has the two advantages:

(1) We obtain a feasible state-transition graph for each function f in G_c in Step 1 and further it can satisfy timing constraint and area constraint in Steps 2 and 3.

(2) In Step 3, we can enumerate more than one state-transition graphs while satisfying timing constraint and area constraint. A hardware designer can select better designs among them depending on several criteria.

B.1 Initial State-transition Graph Generation (Step 1)

Let $G'_{cf} = (V'_{cf}, E'_{cf})$ be a modified control-flow graph of a function f in a call graph $G_c = (V_c, E_c)$. In Step 1, we generate from G'_{cf} a feasible state-transition graph $G_{st} = (V_{st}, E_{st})$ in which $|V_{st}|$ is minimum. G_{st} is called an *initial state-transition graph*. Then we assign hardware resources to the function f. Hardware resources here can have large area but their execution time is minimum.

Fig. 4 shows an algorithm for Step 1. Fig. 5 shows an algorithm for state-transition graph generation which is called in Fig. 4. For example of state-transition graph generation in Fig. 5, let us assume that a modified control-flow graph of a function f shown in Fig. 6(a) is given. In Fig. 6(b), the feedback edge going into the node c is deleted and the nodes a and c are marked as execution start points ($V_s = \{a, c\}$, Step (0)). At that time, $V(a) = \{a, b, e\}$ and $V(c) = \emptyset$ are obtained since $V_s \neq \emptyset$ (Step (1-1)). $ST1$ for $V(a)$ is generated as a state in a state-transition graph. After generating $ST1$, $V(a)$ is deleted and the node c connecting to $V(a)$ is marked as an execution start point in Fig. 6(c). V_s is updated as $V_s = \{c\}$ (Step (1-3)). By repeating this process, $ST2$ is generated as a state in the state-transition graph. Finally edges are added to the state-transition graph in Fig. 6(d).

Now, we clearly have the following theorem.

Theorem 1 *The algorithm in Fig. 5 gives a feasible state-transition graph.*

Further, we can show that the algorithm in Fig. 5 gives a state-transition graph in which the number of states is minimum.

[1] If a subgraph G^i_{cf} has a memory node, correct state transition cannot always be realized even if it satisfies Conditions 1 and 2. In order to deal with memory nodes, we first construct a feasible state-transition graph and then we insert empty states into it as postprocessing.

[2] We set c_0 to be $2551\mu m^2$/state from experiments. Note that our area for a functional unit or a register includes connection area for each of them.

Step 1.1 Pick up each function $f \in V_c$ from a call graph $G_c = (V_c, E_c)$ one by one. For a modified control-flow graph $G'_{cf} = (V'_{cf}, E'_{cf})$ for f, call the algorithm in Fig. 5.

Step 1.2 Assign hardware resources to each f in G_c.

Fig. 4. Initial state-transition graph generation (Step 1).

(Input: Modified control-flow graph $G'_{cf} = (V'_{cf}, E'_{cf})$**)**
(Output: State-transition graph $G_{st} = (V_{st}, E_{st})$**)**

(0) Delete all the feedback edges in G'_{cf} and mark the node pointed by the start node and loop nodes as execution start points. Let $V_s \subset V'_{cf}$ be a set of the marked nodes.

(1) While $V_s \neq \emptyset$, repeat (1-1)–(1-3). If $V_s = \emptyset$, go to (2).

(1-1) For each execution start point $v \in V_s$, let $V(v) = P(v) - \bigcup_{u \in V_s - \{v\}} P(u)$, where $P(v)$ is a set of v and its all succeeding nodes excluding the end node.

(1-2) For a node $v \in V_s$ such that $V(v) \neq \emptyset$, generate a state corresponding to $V(v)$ for G_{st} and delete a set $V(v)$ of nodes from G'_{cf}. Mark as execution start points the nodes connecting to the edges coming from $V(v)$. Include these marked nodes into V_s and exclude v from V_s.

(2) Add edges for G_{st}.

Fig. 5. Algorithm for state-transition graph generation.

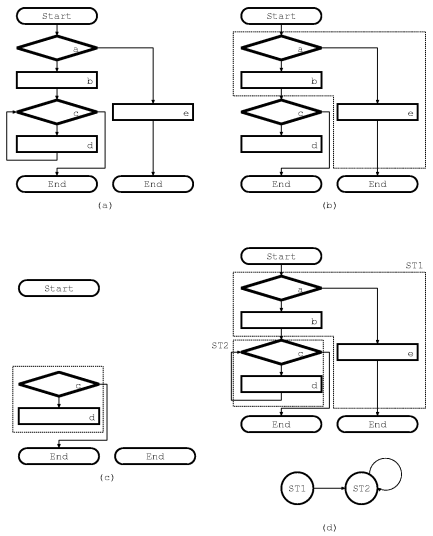

Fig. 6. An example of state-transition graph generation (Fig. 5). (a) Step (0) (the feedback edge is not yet deleted). (b) First iteration of Steps (1-1)–(1-2). (c) Second iteration of Steps (1-1)–(1-2). (d) Step (2).

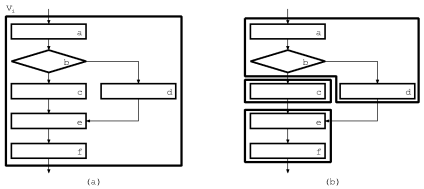

Fig. 7. State partition. (a) The state v_i is partitioned at the node c. (b) After partitioning, three states are obtained ($\{a, b, d\}$, $\{c\}$, $\{e, f\}$).

Theorem 2 *The number of states in a state-transition graph given by the algorithm in Fig. 5 is minimum.*

Proof Since each state in a state-transition graph has only one execution start point, we show that the number of execution start points is minimum in this proof.

Since a loop node has a feedback edge, it must become an execution start point. Clearly the node pointed by a start node is an execution start point. Then all the nodes in V_s of Step (0) must be execution start points in any state-transition graph.

Assume that all the nodes in V_s must be execution start points in a state-transition graph. Let u be a node marked newly in Step (1-2). Since u is not included in $V(v)$ and has an edge coming from $V(v)$, u has at least two preceding nodes which are execution start points, one of which is the node v and another one is a node $w \in V_s - \{v\}$. Thus, if u does not become an execution start point, a state including u has at least two execution start points v and w. u must be an execution start point in a state-transition graph.

Based on the above discussion, V_s always includes the nodes which must become execution start points in any state-transition graph for a given modified control-flow graph. This means that the number of execution start points is minimum and then the number of states in an obtained state-transition graph is minimum. □

Theorems 1 and 2 show that the algorithm in Fig. 5 gives a feasible state-transition graph in which the number of states is minimum. Since we generate an initial state-transition graph using the algorithm in Fig. 5, we only try to partition a state when considering area constraint and timing constraint in Steps 2 and 3.

Finally time complexity of the algorithm in Fig. 5 is estimated. Let n be the number of nodes in a given modified control-flow graph and m be the maximum number of nodes marked as execution start points. The iteration of Steps (1-1)–(1-2) is repeated m times and Step (1-1) requires $O(mn)$ time. Since Step (1-1) is the most time consuming in the iteration, time complexity of the algorithm in Fig. 5 becomes $O(m^2 n)$.

B.2 Considering Timing Constraint (Step 2)

First we expect that an initial state-transition graph for each function f in a call graph $G_c = (V_c, E_c)$ satisfy path length constraint since it has the minimum number of states. However, it does not always satisfies delay constraint and execution time constraint. Since initial hardware resources have minimum execution time, Step 2 considers to partition a state in an initial state-transition graph and tries to satisfy delay constraint and execution time constraint.

Let $G'_{cf} = (V'_{cf}, E'_{cf})$ be a modified control-flow and $G_{st} = (V_{st}, E_{st})$ be a state-transition graph of f. Let $G^i_{cf} = (V^i_{cf}, E^i_{cf})$ be a subgraph in G'_{cf} corresponding to a state $v_i \in V_{st}$ and $v_s \in V^i_{cf}$ be its execution start point. *Partition* of the state v_i at a node $v_t \in V^i_{cf} - \{v_s\}$ is to run Steps (1)–(2) of Fig. 5 initializing $V_s = \{v_s, v_t\}$. Consequently, the state v_i is partitioned such that both v_i and v_s are execution start points. For example, if the state v_i in Fig. 7(a) is partitioned at the node c, we obtain Fig. 7(b). In Fig. 7(b), v_i is partitioned into three states where the nodes a, c, and e become execution start points in order to make the node c an execution start point. If a state is partitioned into several states, path length of each state will be smaller and then we expect that delay constraint as well as execution time constraint will be satisfied.

In Step 2, for each node v_t in a control-flow graph, we try to partition a state including v_t at v_t. Then if the node $v_{t,min}$ gives the minimum clock period or minimum execution time, we partition a state including $v_{t,min}$ at $v_{t,min}$ and update hardware resource assignment. This process is continued until delay constraint and execution time constraint are satisfied.

B.3 Considering Area Constraint (Step 3)

After Step 2, a state-transition graph for a function f in a call graph $G_c = (V_c, E_c)$ satisfies timing constraint. Then in Step 3, we consider to reduce hardware resources while satisfying timing constraint. Hardware resources are reduced by (1) decreasing the number of operations executed concurrently (Figs. 8(a) and (b)) or (2) exchanging/discarding some functional units (Figs. 8(a) and (c)). (1) is realized by partition of a state described in

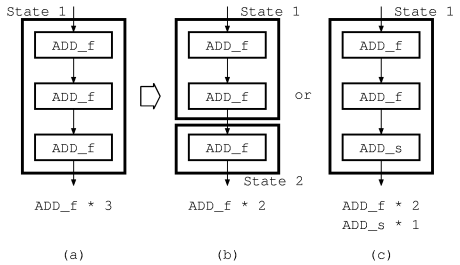

Fig. 8. Process of Step 3. (a) Three fast adders (ADD_f) are originally assigned to State 1. (b) If State 1 is partitioned into State 1 and State 2, we need only two adders and can reduce area. (c) If we exchange the third fast adder for a slow adder (ADD_s), we can also reduce area.

the previous subsection. (2) is easily done by just exchanging or discarding some functional units.

In Step 3, for each node v_t in a control-flow graph, we try to partition a state including v_t at v_t. We also try to exchange functional units or discard a functional unit. Then if the node $v_{t,min}$ or the functional unit op_{min} gives the minimum area, we partition a state including $v_{t,min}$ at $v_{t,min}$ or exchange/discard op_{min}, and we update hardware resource assignment. If area constraint is satisfied, output a current state-transition graph for a function. This process is continued until timing constraint is no longer satisfied.

Based on this process, we can enumerate more than one state-transition graphs satisfying timing constraint and area constraint.

IV. Experimental Results and Conclusions

The proposed algorithm has been implemented on Sun Ultra Workstation in the C language and applied to a vending machine controller, an x25 protocol processor [4], and a huffman coder (the numbers of control-flow graph nodes for them are 360, 37, and 5964, respectively). The vending machine controller is composed of three functions and neither area constraint nor timing constraint is given to it. The x25 protocol processor is composed of a single function. We give it execution time constraint of 200ns (t_{max} = 200ns). We give it no area constraint. The huffman coder is composed of four functions. We give it path length constraint of $l_{max}(f) = 1$ for each function f and delay constraint of 200ns ($t_{c,max}$ = 200ns). We give it no area constraint. We use hardware resource libraries synthesized by Synopsis Design Compiler using VDEC libraries (0.35μm technology).[3]

Figs. 9, 10, and 11 show experimental results for the vending machine controller, the x25 protocol processor, and the huffman coder. Figs. 9 and 10 show the relation between area and execution time when a sequence of input data is entered into each application program. Fig. 11 shows the relation between area and clock period. The run time of the proposed algorithm to obtain the results is a maximum of 15 minutes. The proposed algorithm obtains more than one hardware architecture candidates varying area, execution time, and clock period.

Further, we wrote a hardware description manually for the x25 protocol processor and obtained its hardware. In Fig. 10, the manual design is shown as an x mark. The manual design obtains only one result but the proposed algorithm obtains several results, some of which are superior to the manual design.

The experimental results demonstrate that the proposed algorithm searches wider design space than manual design and then obtains more than one hardware architecture candidates from a hardware specification.

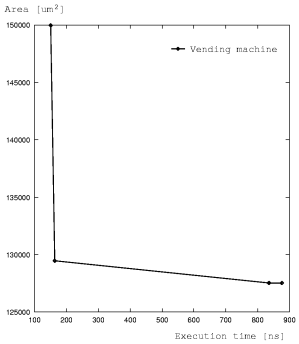

Fig. 9. Experimental results on the vending machine controller.

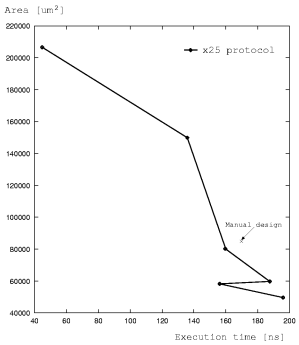

Fig. 10. Experimental results on the x25 protocol processor.

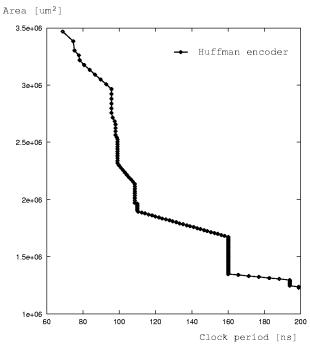

Fig. 11. Experimental results on the huffman coder.

Acknowledgment

The authors would like to thank S. Nakamoto and T. Yoda of Waseda University for their valuable discussions and comments as well as their implementations. This research was supported in part by Grant-in-Aid for Scientific Research Nos. 10650345 and 10750303 from the Ministry of Education, Science, and Culture of Japan.

[3]The libraries in this study have been developed in the chip fabrication program of VDEC, the University of Tokyo with the collaboration by Hitachi Ltd. and Dai Nippon Printing Corporation.

References

[1] R. A. Bergamaschi, S. Raje, I. Nair, and L. Trevillyan, "Control-flow versus data-flow-based scheduling: Combining both approaches in an adaptive scheduling system," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 5, no. 1, pp. 82–100, 1997.

[2] C. N. Coelho, Jr. and G. De Micheli, "Analysis and synthesis of concurrent digital circuits using control-flow expressions," *IEEE Trans. Comput.-Aided Des. Integrated Circuits & Sys.*, vol. 15, no. 8, pp. 854–876, 1996.

[3] G. Lakshminarayana, K. S. Khouri, and N. K. Jha, "Wavesched: A novel scheduling technique for control-flow intensive designs," *IEEE Trans. Comput.-Aided Des. Integrated Circuits & Sys.*, vol. 18, no. 5, pp. 505–523, 1999.

[4] A. S. Tanenbaum, *Computer Networks*, Prentice Hall, Englewood Cliffs, N. J., 1989.

[5] N. Togawa, M. Yanagisawa, and T. Ohtsuki, "A high-level synthesis system for control-based hardwares and its applications," in *Proc. IPSJ Design Automation Symposium '99*, pp. 189–194, 1999 (in Japanese).

[6] J. C.-Y. Yang, G. De Micheli, and M. Damiani, "Scheduling and control generation with environmental constraints based on automata representations," *IEEE Trans. Comput.-Aided Des. Integrated Circuits & Sys.*, vol. 15, no. 2, pp. 166–183, 1996.