VLSI DESIGN
2000, Vol. 11, No. 4, pp. 381–396
Reprints available directly from the publisher
Photocopying permitted by license only

An Integrated Approach to Data Path Synthesis for Behavioral-level Power Optimization*

CHAERYUNG PARK[a], TAEWHAN KIM[b,†] and C. L. LIU[c]

[a]*Synopsys Inc., 700 E. Middlefield Rd., Mountain View, CA 94043, USA;* [b]*Department of Electrical Engineering & Computer Science, and Advanced Information Technology Research Center (AITrc), KAIST, Korea;*
[c]*Department of Computer Science, National Tsing Hua University, 101, Sec. 2, Kuang Fu Road, Hsinchu, Taiwan, ROC*

(Received 5 June 1999; In final form 21 December 1999)

This paper presents an integrated approach to data path synthesis which solves three important design problems: scheduling, allocation, and hardware partitioning with power minimization as a key design objective. Based on the rules of thumbs introduced in prior work on synthesis for low power we derive an integer programming formulation for solving the problems. We then, based on the formulation, develop an efficient algorithm which performs scheduling, allocation and hardware partitioning simultaneously so that the effects of them on power consumption are exploited more fully and effectively. Our experimentation results show that the algorithm is quite effective, producing designs with significant savings in power consumption.

Keywords: Behavioral synthesis; Low-power; VLSI; Scheduling; Allocation; Data path

1. INTRODUCTION

Power consumption in VLSI circuits has become an important consideration in circuit design in recent years [1]. In many application domains, we need to use low-power circuits in order to lower the packaging and cooling costs and to extend the battery life. In designing low-power circuits, a number of techniques for reducing internal power dissipation has been proposed. These techniques often focus on reducing the dominant term in the equation for power dissipation in CMOS digital circuits [1]:

$$P = C_L V_{DD}^2 f_p \tag{1}$$

where P denotes the power dissipated in charging and discharging the output capacitive load C_L. V_{DD} is the supply voltage and f_p is the output switching frequency. One way to reduce power consumption is to lower the total capacitive load C_L. In general, there are different types of

* Based on "An Efficient Data Path Synthesis Algorithm for Behavioral-level Power Optimization" by C. Park, T. Kim and C. L. Liu, which appeared In: *Proceedings of 1999 Inter. Symp. On Circuits and Systems*, Vol. I, 294–297, Orlando, Fl, June, 1999 C 1999 IEEE.

† Corresponding author.

functional modules which perform the same computation but have different areas, speeds, capacitive loads and power consumptions. Consequently, for power efficient design it is important to make suitable choices among the different types of functional modules available. Another way to reduce power consumption is to reduce f_p by inhibiting unnecessary circuit switching activity. For example, in the design of the PowerPC 603 the clock signal was disabled thereby inhibiting switching during periods of inactivity in the module [2].

Previous research efforts [3–5] in high level synthesis have mostly focused on speed and/or area optimization using a global periodic clock signal and a single type of functional modules for each operation. Much work on power optimization has focused at the logic level. The power trade-off between different types of adders and multipliers was studied in [6]. In [7] power was minimized by modifying the function of each node in the circuit. [8] employed a re-encoding technique using gated clocks for reducing power in sequential circuits. A high-level synthesis system, HYPER-LP, presented in [9] uses a variety of architectural and arithmetic transformations to optimize the power dissipation.

In this paper, based on the observations from the prior work on synthesis for low power [6, 8, 9, 11] we design a new high-level synthesis algorithm which performs the tasks of scheduling, allocation, and hardware partitioning in an integrated fashion for low power design. Specially, for a given unscheduled data flow graphs, we are to (1) select functional modules from a given general library (2) schedule the operations on the selected functional modules so that groups of functional modules with similar activity patterns can be deactivated and (3) allocate registers to the variables and partition the registers so that groups of registers can be deactivated. Our objectives are to minimize the total hardware cost as well as power consumption. Our algorithm employs the following techniques to reduce power consumption: *functional module selection, selective shutdown of functional modules, and selective shutdown of registers.*

1.1. Functional Module Selection

In general, an operation can be executed on one of several different types of functional modules which have different execution times, areas and power consumptions. For example, Table I shows that a 32-bit addition operation can be executed on a ripple-carry adder (RCA) in 20 ns which consumes 22.7 mW or on a carry lookahead adder (CLA) in 10 ns which consumes 37.3 mW, and a 32-bit multiplication operation can be executed on a Booth multiplier (BOOTH) in 160 ns which consumes 84.0 mW or on an array multiplier (ARR) in 100 ns which consumes 295.6 mW [10]. To reduce power consumption, utilization of functional modules which consume less power is clearly desirable.

As an example, Figure 1 shows a given unscheduled data flow graph. Suppose the duration of a control step is 120 ns, and to carry out a multiplication operation BOOTH takes 2 control steps and ARR takes 1 control step. Schedule A in Figure 2 shows a schedule when only ARR is available. Schedule B in Figure 3 shows a schedule when both ARR and BOOTH are available. Schedule A uses 2 ARRs and consumes 4138.4 mW. On the other hand, Schedule B uses 1 ARR and 2 BOOTHs, and consumes only 2657.2 mW.

TABLE I Delay and average power of operators

	Delay (ns)	Power (mW)
RCA	20.0	22.7
CLA	10.0	37.3
BOOTH	160.0	84.0
ARR	100.0	295.6

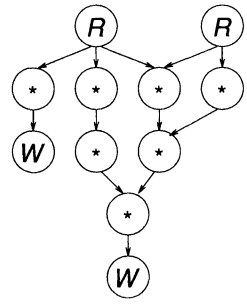


FIGURE 1 An example.

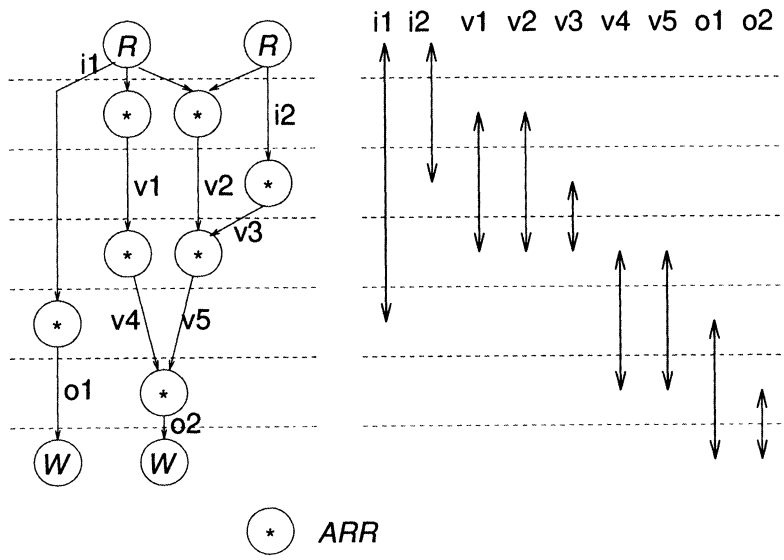


FIGURE 2 Schedule A.

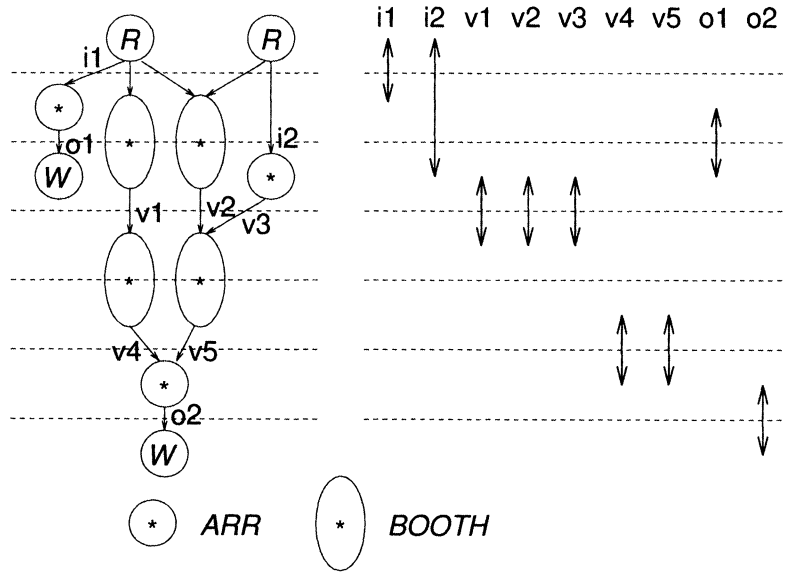


FIGURE 3 Schedule B.

Consequently, Schedule B yields a 35% saving in power consumption.

1.2. Selective Shutdown of Functional Modules

Recent studies [11] indicate that the clock signal in a digital circuit consumes somewhere between 15% to 45% of the total power. The clock signal is distributed globally to all functional modules and memory modules. However, in most cases, some portions of the circuit are inactive during some control steps (the functional modules are not performing any active computation) and the clock signal is not needed for these functional modules during these control steps. However, since every

module is driven by the clock signal, there will be switching activity even in modules that are not performing any active computation, leading to unnecessary power consumptions in *both* the clock signal and the modules. In CMOS circuits, a *clocked functional module* (From now, we assume all functional modules are clocked functional modules.) has registers at its inputs. When the clock signal is on, the contents of the registers are loaded into the functional unit. In other words, if the clock signal is off during time period, there will be no dynamic power consuming activity during that time period [11]. It is therefore desirable that a functional module be driven by a clock signal which is on only during control steps in which the functional module is active.

In order to save power consumption in a clock signal as well as in functional modules, we can regenerate the clock signal so that clock pulses are present only when they are needed. Figure 4 shows a clock regenerator that regenerates the clock signal using a *clock gating* signal which is designed according to the active/inactive control steps of the functional modules. (An *active* control step of a functional module is a control step in which the module is performing active computation. Similarly, an *inactive* control step is one in which the functional module is not performing active computation.) Note that power consumption is reduced both in the functional modules during the inactive control steps and in the clock signal when the corresponding clock pulses are suppressed.

From a power consumption point of view, each functional module should have its own regenerated clock signal. That is, a functional module will be deactivated in all the control steps in which it is not performing any computation. However, in this case, there will be too many distinct clock regenerators and the control logic for clock regenerators will become very complicated. Correspondingly, there also will be too many distinct clock gating signals. Consequently, the clock regenerators and their control logic might become dominant factors of the total power consumption. A more practical approach is to use the same regenerated clock signal for a *group* of functional modules so that not only will the number of clock regenerators be reduced, but also the control logic for clock gating will not become excessively complicated.

In our approach, we use a regenerated clock signal for each *type* of functional modules. Figure 5 shows the clock regenerator logic used in our algorithm. Figure 6 shows active and inactive control steps of ARR and BOOTH for Schedule B. With the regenerated clock signals, Schedule B consumes 1222.8 mW on the functional modules and consumes 70% less power than Schedule A does.

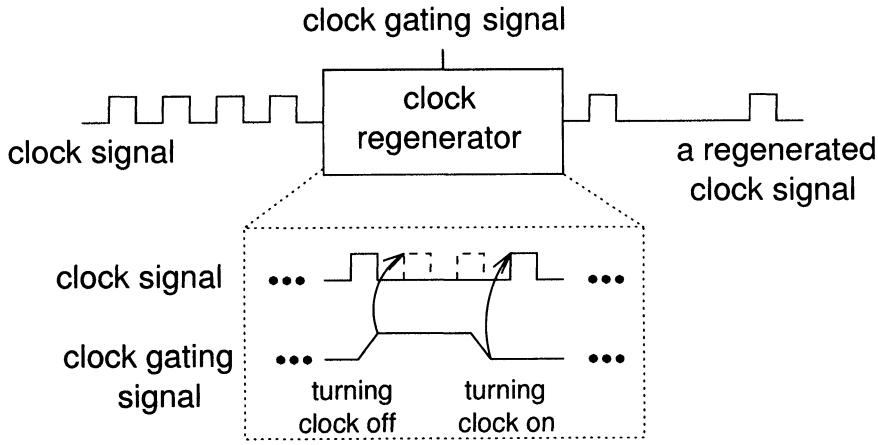


FIGURE 4 Clock regenerator.

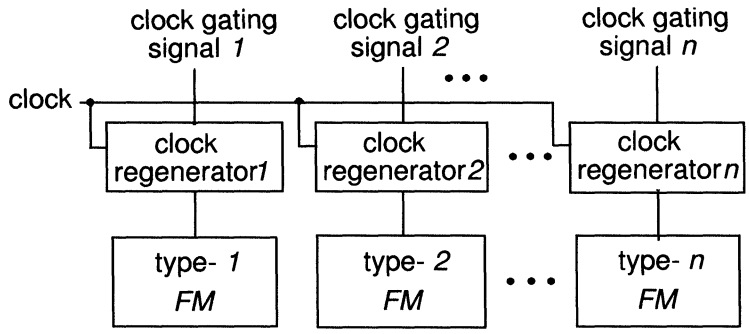


FIGURE 5 Clock regenerator logic.

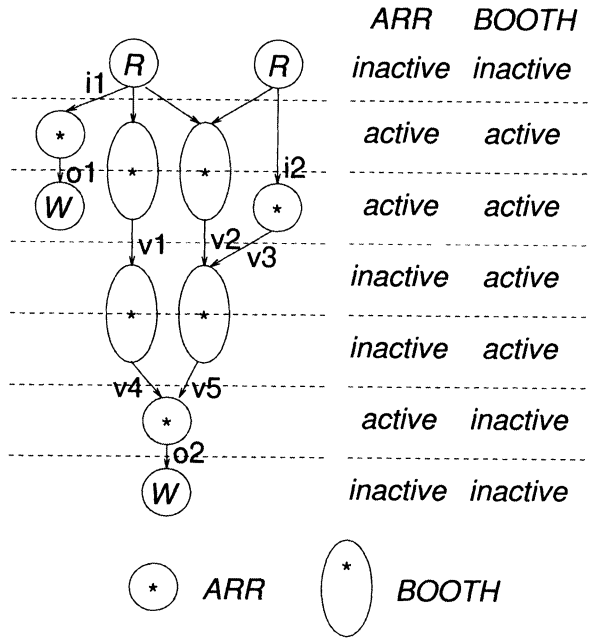


FIGURE 6 Schedule B with clock regenerator.

1.3. Selective Shutdown of Registers

In a dynamic register, information is stored in the form of electric charge which leaks gradually over time. Thus dynamic registers need to be refreshed periodically, usually in each control step.

Consequently, power is consumed in both the refresh circuit and the clock signal. However, data stored in a register might become obsolete, and it is unnecessary to refresh a register after the data stored in the register will no longer be used. Similar to the case of functional modules, by clock gating, we can turn off both the refresh circuit and the clock signal that drives the refresh circuit during control steps in which a register no longer needs to be refreshed.

Again, it is not practical for each register to have its own regenerated clock signal. Consequently, we shall partition the registers into groups and let each group be driven by its own regenerated clock signal.

To partition the registers, we first partition the variables into groups. Each group of variables is

then assigned to the registers to form a group. The partition is to be carried out in such a way that the total number of active control steps in the registers is minimized. (An *active control step* of a register is a control step in which the register contains the value of a live variable. Similarly, an *inactive* control step is one in which the register contains data which is obsolete.) Figure 7(a) shows the life times of 9 variables from Schedule B in Figure 3. Figure 7(b) shows three different ways to partition the variables (which are then assigned to registers). In Partition A, there is only one group of 3 registers. They have only one (common) inactive control step. Thus, the total number of active control steps in the registers is 15. In Partition B, there are 2 groups of registers, $P1$ and $P2$. In group $P1$, there are 2 registers and one (common) inactive control step. In group $P2$, there are 1 register and 2 (common) inactive control steps. The total number of active control steps in the registers is 14. In Partition C, there are also 2 groups of registers, $P1$ and $P2$. In group $P1$, there

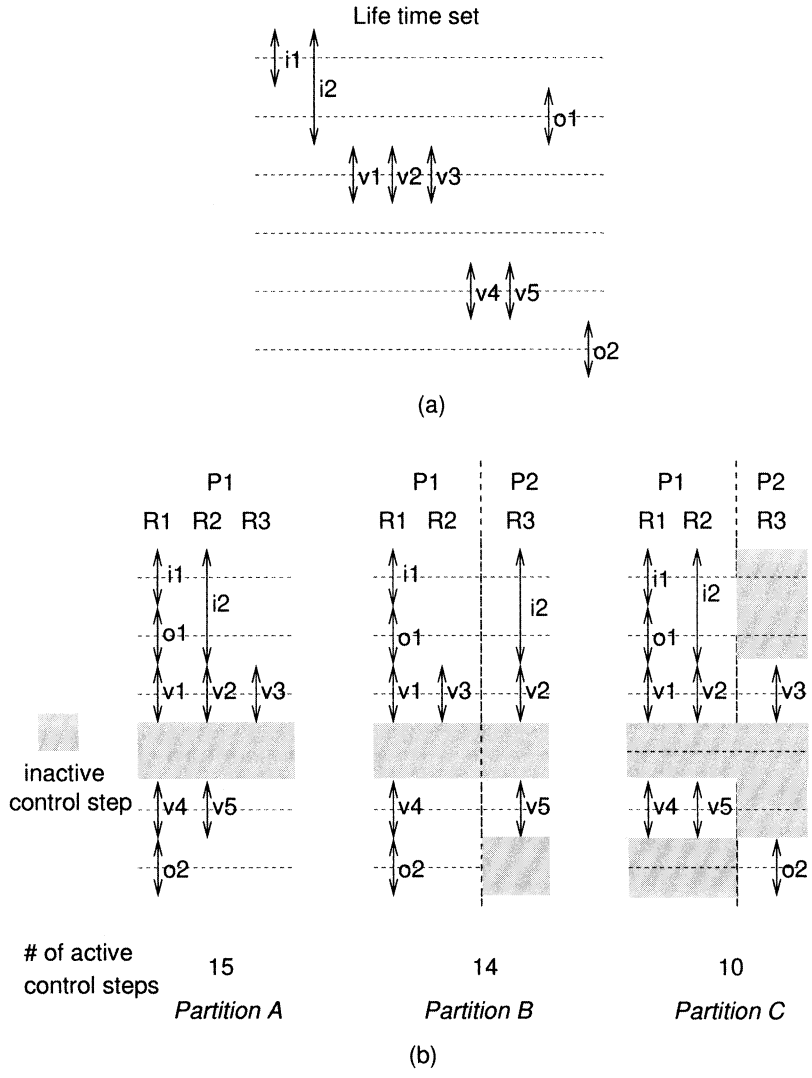


FIGURE 7 Three possible partitioning.

are 2 registers and 2 (common) inactive control steps. In group $P2$, there are 1 register and 4 (common) inactive control steps. The total number of active control steps in the registers is 10. In fact, Partition C achieves a saving of 33% of power consumption in register than using Partition A.

The three techniques mentioned above are closely inter-related. Our algorithm performs the tasks of functional module selection, scheduling, allocation and partitioning simultaneously embodying these techniques. We are given a general library which contains several types of functional modules for each type of operation and the total number of control steps within which all operations in the data flow graph are to be executed. We are to (1) determine an execution schedule for the operations, (2) select the type of functional module for each operation, (3) partition the variables into groups, (4) allocate functional modules and registers and (5) determine the regenerated clock signal for each *type* of functional modules and each group of registers. The hardware cost, the total power consumption, and the total number of active control steps in functional modules and registers will be minimized. We propose a polynomial time algorithm which is an approximation algorithm for solving an integer programming (IP) problem.

Previous research efforts [3–5] in high level synthesis have mostly focused on speed and/or area optimization using a global periodic clock signal and a single type of functional modules for each operation. In [12, 13] the problem of selecting *one* type of functional modules for each type of operations from a given library was studied. In [14], the scheduling problem for general libraries was studied, combining both integer linear programming (ILP) and list scheduling techniques. In [11], reduction in power consumption in registers using a partitioning approach was studied.

2. NOTATIONS

Our algorithm accepts as input a VHDL description of a data path. We are given the total number of control steps, T, within which execution is to be scheduled. From the VHDL description, a set of operations and a precedence relation over the operations, $\prec$, are derived. Let $\mathcal{O} = \{op_1, op_2, \ldots, op_N\}$ denote the set of operations. The relation $op_i \prec op_j$ means that the execution of op_i must be completed before the commencement of the execution of op_j. A library of functional module types is provided. Each type is specified by its cost, power consumption, execution time, and the operation types which can be executed on such functional modules. Let $\mathcal{L} = \{1, 2, \ldots, L\}$ denote the library of functional module types. For a type-k functional module, let e_k denote its execution time in terms of the number of control steps which is an integer ≥ 1, c_k its cost, and d_k the power consumption per control step.

The set of operations that can be executed on a type-k functional module is denoted $\mathcal{O}(k)$, *i.e.*, $\mathcal{O}(k) = \{op_i | op_i$ can be executed on a type-k functional module$\}$. For a given operation op_i, let $\mathcal{U}(op_i)$ denote the set of types of functional modules on which op_i can be executed, *i.e.*, $\mathcal{U}(op_i) = \{k | op_i$ can be executed on a type-k functional module$\}$. The *time frame* of an operation op_i, $[s_i, t_i]$, indicates the control steps within which the operation is to be executed in order to satisfy the timing constraints (so that execution of all operations will be completed within T control steps).

Let $\mathcal{V} = \{\mathrm{var}_1, \mathrm{var}_2, \ldots, \mathrm{var}_M\}$ denote the set of variables where var_i is the output produced by operation op_i. Let $C(\mathrm{var}_i)$ be the set of operations which have var_i as an input, *i.e.*, $\mathcal{C}(\mathrm{var}_i) = \{op_j | \mathrm{var}_i$ is an input to operation $op_j\}$. The value of each variable is stored in a register. We are given the cost of a register, c_r and the power consumptions per control steps, d_r.

3. PROBLEM FORMULATION

The problem we want to solve is a very complex one. Our major contribution is an efficient approximation algorithm (Section 4) which is

developed from an integer programming (IP) formulation of the scheduling, allocation and partitioning problem. In this section we provide the details of the formulation. For simplicity, we assume that the registers are to be partitioned into two groups. Our formulation can be easily extended to the multiple partitioning.

We introduce first the definitions of several sets of variables. First of all, for each operation op_i, we have a 0–1 integer variable o_{ijkp}, $1 \le i \le N$, $s_i \le j \le t_i - e_k + 1$, $k \in U(op_i)$, $p \in \{1, 2\}$.

$$o_{ijkp} = \begin{cases} 1 & \text{if } op_i \text{ begins its execution at control step } j \text{ on a type-}k \text{ functional module and the output variable var}_i \text{ is stored in a register in Group-}p \\ 0 & \text{otherwise} \end{cases}$$

Second, for each variable var_i, we have a 0–1 integer variable v_{ikp}, $1 \le i \le M$, $s_i \le j \le t_i$, $p \in \{1, 2\}$.

$$v_{ijp} = \begin{cases} 1 & \text{if var}_i \text{ is alive at control step } j \text{ and is stored in a register in Group-}p \\ 0 & \text{otherwise} \end{cases}$$

Third, to determine the total number of functional modules of each type, we introduce the integer variables f_{kj} and F_k, $1 \le k \le L$, $1 \le j \le T$. f_{kj} is the number of type-k functional modules used in control step j and F_k is the total number of type-k functional modules used throughout all control steps.

Fourth, to determine the total number of registers in each group, we have two sets of integer variables, r_{pj} and R_p, $p \in \{1, 2\}$ and $1 \le j \le T$. r_{pj} is the number of variables which are alive at control step j in Group-p. In other words, r_{pj} is the number of registers needed at control step j in Group-p. R_p is the number of registers in Group-p which is the maximum of r_{pj} for $j = 1, \ldots, T$. It should be noted that it is guaranteed that the variables in Group-p can be assigned to R_p registers when the left edge algorithm [15] is used since there are at most R_p variables that are alive in each control step.

Finally, to represent the active/inactive control step of each type of functional modules, we have the 0–1 integer variables, a_{kj} and b_{pj}, $1 \le k \le L$, $p \in \{1, 2\}$.

$$a_{kj} = \begin{cases} 1 & \text{if any type-}k \text{ functional module is active at control step } j \\ 0 & \text{otherwise} \end{cases}$$

Similarly,

$$b_{pj} = \begin{cases} 1 & \text{if any register in Group-}p \text{ is active at control step } j \\ 0 & \text{otherwise} \end{cases}$$

We have the following constraints. First, each operation must be scheduled for execution exactly once. Thus, for each i (corresponding to operation op_i), $1 \le i \le N$, we have

$$\sum_j \sum_k \sum_p o_{ijkp} = 1 \tag{1}$$

Second, the precedence relations over the operations are represented as linear constraints. Let

$$\mathcal{F}(i, j) = \{o_{ij'kp} | s_i \le j' \le j - e_k + 1, \quad k \in U(op_i), \quad p \in \{1, 2\}\}$$

Thus,

$$\sum_{o \in \mathcal{F}(i,j)} o = \begin{cases} 1 & \text{if the execution of } op_i \text{ is completed on or before control step } j \\ 0 & \text{otherwise} \end{cases}$$

Similarly, let

$$\mathcal{I}(i,j) = \{o_{ijkp} | k \in U(op_i), \quad p \in \{1, 2\}\}$$

Thus,

$$\sum_{o\in\mathcal{I}(i,j)} o = \begin{cases} 1 & \text{if the execution of } op_i \text{ is} \\ & \quad \text{initiated at control step } j \\ 0 & \text{otherwise} \end{cases}$$

For two operations op_a and op_b such that $op_a \prec op_b$, we have the following constraints: For each j, $s_a \le j \le t_a - 1$,

$$\sum_{o\in\mathcal{F}(a,j)} o \ge \sum_{o'\in\mathcal{I}(b,j+1)} o' \tag{2}$$

which ensures that the execution of op_b is initiated at control step $j+1$ only if the execution of op_a is finished on or before control step j.

Finally, we have the following set of equations. First, for each variable var_i we need to determine whether var_i is alive at control step j and the group to which var_i belongs. var_i is alive at control step j and is included in Group-p if operation op_i completes its execution before control step j and its output is included in Group-p, *and* any one of the operations which has var_i as an input has not been initiated at control step j. For each $p \in \{1, 2\}$, let

$$\mathcal{F}_p(i,j) = \{o_{ij'kp} | s_i \le j' \le j - e_k + 1, \\ k \in U(op_i)\}$$

Thus,

$$\sum_{o\in\mathcal{F}_p(i,j)} o = \begin{cases} 1 & \text{if the execution of } op_i \text{ is completed} \\ & \quad \text{on or before control step } j \\ & \quad \text{and its output variable } \text{var}_i \text{ is} \\ & \quad \text{included in Group-}p \\ 0 & \text{otherwise} \end{cases}$$

$$\mathcal{I}^+(i,j) = \{o_{ij'kp} | j \le j' \le t_i,\ k \in U(op_i), \quad p \in \{1,2\}\}$$

Thus,

$$\sum_{o\in\mathcal{I}^+(i,j)} o = \begin{cases} 1 & \text{if the execution of } op_i \text{ is initiated} \\ & \quad \text{at or after control step } j \\ 0 & \text{otherwise} \end{cases}$$

So we have the following equations:

$$v_{ijp} = F_p(i,j) \cap \bigcup_{op_{i'}\in C(i)} \mathcal{I}^+(i', j+1) \tag{3}$$

After the determination of values of the variables o_{ijkp} and v_{ijp}, we can compute the values of f_{kj}, F_k, r_{pj} and R_p as:

$$f_{kj} = \sum_{\{i | op_i \in O(k)\}} \left(\sum_{j'=j-e_k+1}^{j} x_{ij'k} \right) \tag{4}$$

$$F_k = \max_j f_{kj} \tag{5}$$

$$r_{pj} = \sum_i v_{ijp} \tag{6}$$

$$R_p = \max_j r_{pj} \tag{7}$$

Finally, after the values of f_{kj} and r_{pj} have been determined, we know the active/inactive control step of each type of functional modules and each group of registers. Therefore the values of a_{kj} and b_{pj} can be determined as:

$$a_{kj} = \begin{cases} 0 & \text{if } f_{kj} = 0 \\ 1 & \text{otherwise} \end{cases} \tag{8}$$

$$b_{pj} = \begin{cases} 0 & \text{if } r_{pj} = 0 \\ 1 & \text{otherwise} \end{cases} \tag{9}$$

In other word, if any type-k functional module is used in control step j ($f_{kj} \ge 1$), all type-k functional modules will be active at control step j and a_{kj} equals to 1. b_{pj} is computed similarly.

The objective function is

$$F = \alpha F_f + \beta F_r$$

where

$$F_f = \gamma \sum_k c_k F_k + \delta \sum_k \sum_j a_{kj} d_k F_k + \omega \sum_k \sum_j a_{kj}$$

$$F_r = \gamma \sum_p c_r R_p + \delta \sum_p \sum_j b_{pj} d_r R_p + \omega \sum_k \sum_j b_{pj}$$

and α, β, γ, δ and ω are weighting factors. F_f is a weighted sum of hardware cost, power consumption and clock regenerators for the functional modules. Similarly, F_r is a similar weighted sum for the registers.

4. APPROXIMATION ALGORITHM

This section presents an approximation algorithm for solving the IP problem formulated in Section 3, which yields an approximation solution to the problem of module selection, scheduling, allocation and register partitioning.

In a feasible solution to the ILP, each 0–1 variable o_{ijkp} will assume the value 0 or 1. In our approximation algorithm, we will determine the values of the 0–1 variables in a step-by-step fashion by examining intermediate solutions in which some of the variables o_{ijkp} assume non-integral values.

At the beginning, we set the value of one of these variables o_{ijkp} to 1. Such a choice might lead to the determination of the 0–1 values of other 0–1 variables because of the constraints in (1) and (2).

Let us use the example in Figure 8 to illustrate the idea.

We have the following constraints when the total number of control steps is given to be 5.

Constraints of the form (1):

(1) $o_{1111}+o_{1112}+o_{1211}+o_{1212}=1$
(2) $o_{2211}+o_{2212}+o_{2221}+o_{2222}+o_{2311}+o_{2312}+o_{2321}+o_{2322}+o_{2411}+o_{2412}=1$
(3) $o_{3211}+o_{3212}+o_{3221}+o_{3222}+o_{3311}+o_{3312}=1$
(4) $o_{4311}+o_{4312}+o_{4411}+o_{4412}+o_{4511}+o_{4512}=1$
(5) $o_{5311}+o_{5312}+o_{5321}+o_{5322}+o_{5411}+o_{5412}=1$
(6) $o_{6411}+o_{6412}+o_{6511}+o_{6512}=1$

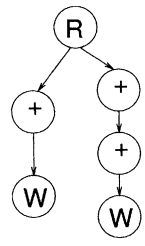
Constraints of the form (2):

(7) $o_{1111}+o_{1112} \geq o_{2211}+o_{2212}+o_{2221}+o_{2222}$
(8) $o_{1111}+o_{1112} \geq o_{3211}+o_{3212}+o_{3221}+o_{3222}$
(9) $o_{2211}+o_{2212} \geq o_{4311}+o_{4312}$
(10) $o_{2211}+o_{2212}+o_{2311}+o_{2312}+o_{2221}+o_{2222} \geq o_{4411}+o_{4412}$
(11) $o_{3211}+o_{3212} \geq o_{5311}+o_{5312}+o_{5321}+o_{5322}$
(12) $o_{5311}+o_{5312} \geq o_{6411}+o_{6412}$

Suppose we set o_{1111} to 1. Because of constraint (1), we have $o_{1112}=o_{1121}=o_{1122}=0$. On the other hand, suppose we set o_{1211} to 1. Again, because of constraint (1), we have $o_{1111}=o_{1112}=o_{1212}=0$. Since $o_{1111}+o_{1112}=0$, because of precedence constraints (7) and (8), we have $o_{2211}=o_{2212}=o_{2221}=o_{2222}=0$ and $o_{3211}=o_{3212}=o_{3221}=o_{3222}=0$. Since $o_{3211}+o_{3212}=0$, because of constraint (11), we have $o_{5311}=o_{5312}=o_{5321}=o_{5322}=0$. Since $o_{5311}+o_{5312}=0$, because of constraint (12), we have $o_{6411}=o_{6412}=0$.

On the other hand, suppose we set o_{2211} to 1. Because of constraint (2), we have $o_{2212}=o_{2221}=o_{2222}=o_{2311}=o_{2312}=o_{2321}=o_{2322}=o_{2411}=o_{2412}=0$. Since $o_{2211}=1$, because of constraint (7), we have $o_{1111}+o_{1112}=1$, and consequently, $o_{1211}=o_{1212}=0$.

After the value of a 0–1 variable is set to 1, and the values of some of the other 0–1 variables are determined according to the constraints (1) and (2), the remaining 0–1 variables o_{ijkp} are related by



	type	hardware cost	power consumption	execution time	operations
CLA	1	2	4	1	+
RCA	2	5	1	2	+
Register		10	7		

FIGURE 8 An example.

the constraints in (1) which are of the form

$$\sum_{j,k} o_{ijkp} = 1$$

The variables in the equalities will be assigned equal fractional values. For the example in Figure 8, after setting o_{1111} to 1, we have

$$o_{1112} = o_{1211} = o_{1212} = 0$$
$$o_{2211} = o_{2212} = o_{2221} = o_{2222} = o_{2311} =$$
$$o_{2312} = o_{2321} = o_{2322} = o_{2411} = o_{2412} = 1/10$$
$$o_{3211} = o_{3212} = o_{3221} = o_{3222} =$$
$$o_{3311} = o_{3312} = 1/6$$
$$o_{4311} = o_{4312} = o_{4411} = o_{4412} =$$
$$o_{4511} = o_{4512} = 1/6$$
$$o_{5311} = o_{5312} = o_{5321} = o_{5322} =$$
$$o_{5411} = o_{5412} = 1/6$$
$$o_{6411} = o_{6412} = o_{6511} = o_{6512} = 1/4$$

On the other hand, after setting o_{2121} to 1, we have

$$o_{1111} = o_{1112} = 1/2, \quad o_{1211} = o_{1212} = 0$$
$$o_{2212} = o_{2221} = o_{2222} =$$
$$o_{2311} = o_{2312} = o_{2321} = o_{2322} = o_{2411} = o_{2412} = 0$$
$$o_{3211} = o_{3212} = o_{3221} = o_{3222} =$$
$$o_{3311} = o_{3312} = 1/6$$
$$o_{4311} = o_{4312} = o_{4411} = o_{4412} =$$
$$o_{4511} = o_{4512} = 1/6$$
$$o_{5311} = o_{5312} = o_{5321} = o_{5322} =$$
$$o_{5411} = o_{5412} = 1/6$$
$$o_{6411} = o_{6412} = o_{6511} = o_{6512} = 1/4$$

After determining the values of the variables o_{ijkp}, the values of the other variables f_{kj}, F_j, r_{pj}, R_p, a_{kj} and b_{pj} are computed according to the Eqs. in (3) to (9).

We then compute the value of the objective function F. For the example in Figure 8, we obtain the values of F corresponding to all possible choices of setting one of the variables o_{ijkp} to 1. (The values of the weighting factors in F are chosen to be: when $\alpha = 1$, $\beta = 2$, $\gamma = 3$, $\delta = 1$ and $\omega = 1$.)

setting $o_{1111} = 1$: $F = 148.92$
setting $o_{1112} = 1$: $F = 148.92$
setting $o_{1211} = 1$: $F = 152.26$
setting $o_{1212} = 1$: $F = 152.26$
setting $o_{2211} = 1$: $F = 175.19$
setting $o_{2212} = 1$: $F = 175.19$
$\vdots$
setting $o_{5411} = 1$: $F = 161.72$
setting $o_{5412} = 1$: $F = 161.72$
setting $o_{6511} = 1$: $F = 128.16$
setting $o_{6512} = 1$: $F = 128.16$

On the basis of the value of F, the value of one of the variables o_{ijkp} will be set to 1. In other words, among all variables o_{ijkp}, the one that produces the minimum value of F will be set to 1 which together with the values of other variables assigned accordingly constitutes an intermediate solution. For the example in Figure 8, o_{6511} is set to 1. In this case, we obtain an intermediate solution:

$$o_{1111} = o_{1112} = o_{1211} = o_{1212} = 1/4$$
$$o_{2211} = o_{2212} = o_{2221} = o_{2222} = o_{2311} = o_{2312} =$$
$$o_{2321} = o_{2322} = o_{2411} = o_{2412} = 1/10$$
$$o_{3211} = o_{3212} = o_{3221} = o_{3222} =$$
$$o_{3311} = o_{3312} = 1/6$$
$$o_{4311} = o_{4312} = o_{4411} = o_{4412} =$$
$$o_{4511} = o_{4512} = 1/6$$
$$o_{5311} = o_{5312} = o_{5321} = o_{5322} =$$
$$o_{5411} = o_{5412} = 1/6$$
$$o_{6511} = 1, \quad o_{6411} = o_{6412} = o_{6512} = 0$$

The step can now be repeated. Among all 0–1 variables that assume non-integral values in the intermediate solution, one of them is set to 1. The

corresponding values of other variables and the value of F are then computed. Finally, a solution is obtained when all 0–1 variables assume 0–1 values (and all other variables assume integer values). For the example in Figure 8, we have

$$o_{1111} = 1, \quad o_{1112} = o_{1211} = o_{1212} = 0$$
$$o_{2211} = 1, \quad o_{2212} = o_{2221} = o_{2222} = o_{2311} =$$
$$o_{2312} = o_{2321} = o_{2322} = o_{2411} = o_{2412} = 0$$
$$o_{3221} = 1, \quad o_{3211} = o_{3212} = o_{3222} =$$
$$o_{3311} = o_{3312} = 0$$
$$o_{4311} = 1, \quad o_{4312} = o_{4411} = o_{4412} =$$
$$o_{4511} = o_{4512} = 0$$
$$o_{5411} = 1, \quad o_{5311} = o_{5312} = o_{5321} =$$
$$o_{5322} = o_{5412} = 0$$
$$o_{6511} = 1, \quad o_{6411} = o_{6412} = o_{6512} = 0$$

which corresponds to the schedule shown in Figure 9.

A summary of the approximation algorithm is shown below. The algorithm has complexity $O((LTNP)^2)$ in the worst case, where L is the number of types of functional modules in the library, T is the number of control steps, N is the number of operations and P is the number of groups the registers are partitioned into.

Algorithm *IP_solver*( ):
repeat
/* *Determine an intermediate solution corresponding to each* */
/* *variable* o_{ijkp} *whose value is not integral* */
for *each variable* o_{ijkp} *whose value is not integral*

- *Set* $o_{ijkp} = 1$;
- *Compute the values of the other variables according to Eq.* (1) *to* (9);
- *Compute the values of the objective function F*;

end for
- *Select the intermediate solution that yields a minimum value of F*;

until (*all variables assume integral values*)

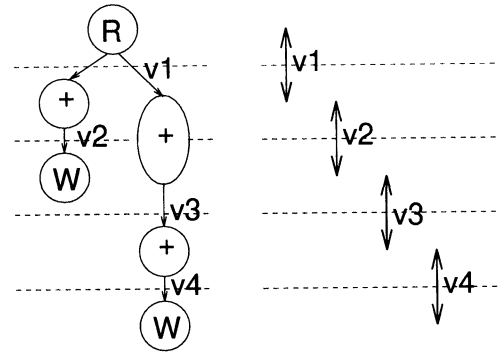


FIGURE 9 The resultant schedule for the example.

5. EXPERIMENTAL RESULTS

We tested our program on a number of benchmark examples. Our algorithm described in Section 4 was implemented in C and executed on a Sun Sparc20 workstation. Example df.5 is the *differential equation* from [3] and the given number of control steps is 5. Example df.7 is the same *differential equation* example except that the given number of control steps is 7. Examples df2.10 and df2.13 are obtained by unrolling the *differential equation* example twice and setting the number of control steps to 10 and 13, respectively. Examples ar.8 ar.11 are the AR-Lattice Filter from [16] with the given number of control steps set to 8 and 10, respectively. Examples ewf.18 and ewf.20 are the elliptic wave filter from [16] with the given number of control steps set to 18 and 20, respectively.

Table II summarizes the results produced by our algorithm for the examples df.5, df.7, df2.10, df2.13 ar.8, ar.11 ewf.18 and ewf.20, using the

Library lib1

Type	Name	Cost	Power	Execution time	Operations
1	rca	5	1	1	+, −, >
2	cla	2	4	1	+, −, >
3	booth	15	4	2	x
4	arr	6	15	1	x
	register	10	7		

TABLE II Results over the library lib1

	Functional Modules							Registers					Total	
Example	Rca	Cla	Booth	Arr	# of active steps	Cost	P_{fm}	$R1$	$R2$	# of active steps	Cost	P_{reg}	F	Time (sec)
df.5	1	0	0	2	13	17	125	2	2	20	40	140	379	2.15
df.7	1	0	1	1	22	26	81	3	1	24	40	168	381	4.62
df2.10	3	0	0	2	30	27	198	3	2	50	50	350	692	29.17
df2.13	1	0	0	1	22	11	190	4	1	61	50	427	739	39.45
ar.8	2	0	0	4	30	34	252	2	4	36	60	252	692	17.10
ar.11	2	0	4	0	44	70	140	2	4	44	60	308	708	24.05
ewf.18	3	0	0	2	55	27	195	4	4	128	80	896	1305	157.63
ewf.20	3	0	2	0	58	45	106	5	3	149	80	1043	1399	242.40

TABLE III Total power consumptions

	Hardware (functional modules + registers)			
Example	Cost	P_{fm}	P_{reg}	P_{total}
df.5	54	170	252	422
df.7	50	133	392	525
df2.10	76	380	616	996
df2.13	58	247	686	933
ar.8	88	544	336	880
ar.10	76	418	462	880
ewf.18	90	414	1596	2010
ewf.20	110	460	1470	1930

	Register_only				
Example	Cost	P_{fm}	P_{reg}	P_{total}	Save
df.5	83	164	189	353	16.4
df.7	56	152	154	306	41.8
df2.10	68	252	350	602	39.6
df2.13	104	370	336	706	24.2
ar.8	88	288	252	540	38.7
ar.10	190	248	224	472	46.4
ewf.18	115	320	833	1153	42.7
ewf.20	115	320	868	1188	38.5

	Functional_module_only				
Example	Cost	P_{fm}	P_{reg}	P_{total}	Save
df.5	88	84	252	336	21.4
df.7	71	95	336	431	18.0
df2.10	113	155	539	694	30.3
df2.13	108	151	784	935	0.0
ar.8	94	252	336	588	33.2
ar.10	130	140	420	560	36.4
ewf.18	153	124	1596	1719	14.5
ewf.20	158	125	1911	2036	−5.5

	Power + hardware				
Example	Cost	P_{fm}	P_{reg}	P_{total}	Save
df.5	57	125	140	265	37.3
df.7	66	81	168	249	52.6
df2.10	77	198	350	548	45.0
df2.13	61	190	427	617	33.9
ar.8	94	252	252	504	42.8
ar.10	130	140	308	448	49.1
ewf.18	107	195	896	1091	45.8
ewf.20	125	106	1043	1149	40.5

library lib1 when $\alpha = 1$, $\beta = 1$ $\gamma = 2$, $\delta = 1$ and $\omega = 1$. The columns rca, cla, booth, and arr give the numbers of functional modules used. The column # of active steps in Functional Modules gives the sum of the numbers of functional modules in each active control step. The columns cost and P_{fm} give the total cost and total power consumption of the functional modules. The columns $R1$ and $R2$ give the number of registers in each group after the registers are partitioned into two groups. The column # of active steps in Registers gives the sum of the numbers of registers in each active control steps. The columns cost and P_{fm} give the total cost and total power consumption of the registers. The column F gives the value of the objective function F. The column time gives the CPU time in seconds.

We also compare the results for different choices of values for the weighting factors in the objective function. The results are summarized in Table III.

TABLE IV Comparison of our algorithm and ILP on df.5

	Functional modules							Registers					Total	
Example	Rca	Cla	Booth	Arr	# of active steps	Cost	P_{fm}	$R1$	$R2$	# of active steps	Cost	P_{reg}	F	Time (sec)
Ours	1	0	0	2	13	17	125	2	2	20	40	140	379	2.15
ILP	1	0	0	2	13	17	125	2	2	20	40	140	379	4208

For the case *Hardware* (*functional_modules+registers*), we set $\alpha=1$, $\beta=1$, $\gamma=1$ and $\delta=\omega=0$. Consequently, the result minimizes the cost of the functional modules and registers only and does not take power consumption into consideration. For the case *Functional_module_only*, we set $\alpha=1$, $\beta=0$, $\gamma=2$ and $\delta=\omega=1$. Consequently, the result minimizes the cost and power consumption in the functional modules only and does not consider the cost and power consumption in the registers. Similarly, for the case *Register_only*, we set $\alpha=0$, $\beta=1$, $\gamma=2$ and $\delta=\omega=1$. Consequently, the result minimizes the cost and power consumption in the registers only. For the case *Power+Hardware*, we set $\alpha=1$, $\beta=1$, $\gamma=2$, $\delta=1$ and $\omega=1$. Consequently, the result minimizes the cost and power consumption in both functional modules and registers. The column P_{fm} gives the power consumption in the functional modules and the column P_{reg} gives the power consumption in the registers. The column P_{total} gives the total power consumption in the functional modules and the registers. The column *save* shows the percentage reduction in power consumption when compared with the case *Hardware*. As an illustration, let us examine the results for the example ar.8 closely. If we want to minimize the total hardware cost, we obtain a design which consumes 544 mW in the functional module and 336 mW in the registers. The total hardware cost is 88. If we want to minimize the power consumption in the functional modules only, we obtain a design which consumes only 252 mW in the functional modules and 336 mW in the registers. The total hardware cost has risen to 94. If we want to minimize the power consumption in the registers only, we obtain a design which consumes 288 mW in the functional modules but only 252 mW in the registers. The total hardware cost is 88. In *Power+Hardware*, when we take everything into consideration, we obtain a 42.8% saving in power consumption. The hardware cost is 57. Indeed, as is shown in Table III we can achieve up to 52.6% reduction in power consumption and 43.4% on the average.

In order to compare the result produced by our approximation algorithm with the optimal result, we should solve the IP in Section 3. However, since the formulation is cast as a non-linear integer programming problem, the computational effort will be substantial even for small problem instances. Hence, we first generate an integer linear programming (ILP) formulation from the integer programming (IP) formulations in Section 3. Since the objective function F is a quadratic function, we approximate it by a linear form using La Grange first order conditions. We then used the LINDO package on an IBM3081 to solve the ILP problem. Table IV shows a comparison of the results of example df.5 produced by our algorithm and by ILP. For df.5, ours produced the same results as LINDO did and took much less time, 2.15 *vs.* 4208 seconds. It should be noted that we can only test example df.5 since the ILP formulation for the other examples are too big and LINDO was not able to generate any solution.

6. CONCLUSIONS

We presented an integrated approach to the problem of solving scheduling, allocation and hardware partitioning with the power consumption as one of key design objectives. We first

proposed an integer programming formulation for solving the problem, from which we derived an efficient approximation algorithm. Unlike previous approaches for low power in which scheduling and allocation are performed independently, our approach combined scheduling, allocation and partitioning together to exploit the effects of them on power consumption more effectively. The experimental results confirmed that our algorithm is quite effective and robust.

Acknowledgements

The work of Taewhan Kim was partially supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc).

References

[1] Raghunathan, A., Jha, N. K. and Dey, S. (1998). *High-Level Power Analysis and Optimization*, Norwel, MA: Kluwer Academic Publishers.

[2] Gary, S., Ippolito, P., Gerosa, G., Dietz, C., Eno, J. and Sanchez, H. (1994). "PowerPC 603, A Microprocessor for Portable Computers", *IEEE Design and Test of Computers*, **11**(4), 14–23.

[3] Paulin, P. G. and Knight, J. P., "Scheduling and Binding Algorithms for High-Level Synthesis", *Design Automation Conference*, pp. 1–6, June, 1989.

[4] Cloutier, R. and Thomas, D. E., "The Combination of Scheduling, Allocation and Mapping in a Single Algorithm", *Design Automation Conference*, pp. 71–76, June, 1990.

[5] Gebotys, C. H. and Elmasry, M. O. (1992). "Optimal Synthesis of High-Performance Architectures", *IEEE Journal of Solid State Circuits*, **27**(3), 389–397.

[6] Callaway, T. and Swartzlander, E., "Optimizing Arithmetic Elements for Signal Processing", *IEEE Workshop on VLSI Signal Processing*, pp. 91–100, October, 1992.

[7] Iman, S. and Pedram, M. (1994). "Multi-Level Network Optimization for Low Power", *International Conference on Computer-Aided Design*, pp. 372–377.

[8] Benini, L., Siegel, P. and DeMicheli, G. (1994). "Saving Power by Synthesizing Gated Clocks for Sequential Circuits", *IEEE Design and Test of Computers*, **11**(4), 32–41.

[9] Chandrakasan, A., Potkonjak, M., Mehra, R., Rabaey, J. and Broderson, R., "Optimizing Power Using Transformations", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **14**(1), 12–31, January, 1995.

[10] Martin, R. and Kight, J. (1995). "Power-Profiler: Optimizing ASICs Power Consumption at the Behavioral-level", *Design Automation Conference*, pp. 42–47.

[11] Farrahi, A., Tellez, G. and Sarrafazadeh, M., "Memory Segmentation to Exploit Sleep Mode Operation", *Design Automation Conference*, pp. 36–41, June, 1995.

[12] Jain, R., Parker, A. and Park, N. (1988). "Module Selection for Pipelined synthesis", *Design Automation Conference*, pp. 542–547.

[13] Jain, R. (1990). "Mosp: Module Selection for Pipelined Designs with Multi-Cycle Operations", *International Conference on Computer-Aided Design*, pp. 212–215.

[14] Timmer, A. H., Heijligers, M. J., Stok, L. and Jess, J. A. (1992). "Module Selection and Scheduling using Unrestricted Libraries", *International Conference on Computer-Aided Design*, pp. 547–551.

[15] Kurdahi, F. and Parker, A. (1987). "REAL: A Program for Register Allocation", *Design Automation Conference*, pp. 210–215.

[16] Paulin, P. and Knight, J. (1989). "High-Level Synthesis Benchmark Results using a Global Scheduling Algorithm", *Logic and Architecture Synthesis for Silicon Compilers*, North-Holland, pp. 211–228.

Authors' Biographies

Chaeryung Park received the B.S. and M.S. degrees in computer engineering from Seoul National University, Korea in 1987 and 1989, respectively. She received the Ph.D. degree in Computer Science from the University of Illinois at Urbana-Champaign in 1995. She joined Synopsys in 1995 and has developed High-level synthesis tools and Logic-level synthesis tools. Her current research interests include System-On-a-Chip design flow development and formal verification.

Taewhan Kim received the B.S. degree in Computer Science and Statistics and the M.S. degree in Computer Science from Seoul National University, Seoul, Korea in 1985 and 1987, respectively, and received the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 1993. From 1993 to 1998, he worked at Lattice Semiconductor Corporation and Synopsys. Inc., where he was involved in CPLD and behavioral synthesis. Since August, 1998, he has been with the Department of Computer Science at KAIST. His research area includes behavioral and logic synthesis, and combinatorial optimizations.

C. L. Liu obtained his B.Sc. degree at the National Cheng Kung University in Taiwan in 1956. He obtained his M.S. degree and E.E. degree in 1960 and his D. Sc. degree in 1962 at the Massachusetts Institute of Technology. He was on the faculty of

the Massachusetts Institute of Technology and the University of Illinois at Urbana-Champaign. He is now a Professor of omputer Science and President of the National Tsing Hua University in Taiwan. His areas of research interest are Design and Analysis of Algorithms, Computer Aided Designs of Integrated Circuits, and Combinational Mathematics.