Determining Schedules for Reducing Power Consumption Using Multiple Supply Voltages

Noureddine Chabini[1], El Mostapha Aboulhamid[1], Yvon Savaria[2]
1: *LASSO, DIRO, Université de Montréal C.P.6128, Suc. Centre ville, Montréal, Qc, Canada, H3C 3J7.*
Email:{chabinin, aboulham}@iro.umontreal.ca
2: *GRM, DGEGI, École Polytechnique de Montréal, C.P. 6079, Suc. Centre-ville, Montréal, Qc, Canada, H3C 3A7. Email: savaria@vlsi.polymtl.ca*

Abstract

Dynamic power is the main source of power consumption in CMOS circuits. It depends on the square of the supply voltage. It may significantly be reduced by scaling down the supply voltage of some computational elements in the circuit, with the penalty of an increase of their execution delay. To reduce the dynamic power consumption, without degrading the performance determined assuming that the circuit operates at the highest available supply voltage, the supply voltage of computational elements off critical paths can be scaled down. Defined here as $MinP_{dyn}$, the problem of minimizing the dynamic power consumption, under performance constraints, by scaling down the supply voltage of computational elements on non-critical paths is NP-hard in general. Solving $MinP_{dyn}$ for multi-phase clocked sequential circuits may allow to reduce their power consumption and the required number of registers. Reducing the number of registers also allows to reduce the power consumption, the number of control signals, and the area of the circuit. In this paper, we focus on devising methods to efficiently solve $MinP_{dyn}$ for designs modeled as cyclic or acyclic graphs. More precisely, once the circuit is optimized for timing constraints, then we look for schedules that allow the computational elements of the circuit to operate at the lowest possible supply voltage. We present an integer linear programming formulation for that problem, which we use to devise a polynomial time solvable method and an exact algorithm based on a branch-and-bound technique. Experimental results confirm the effectiveness of the method and power reduction factors as high as 53,84% were obtained. Also, they show that the exact algorithm produces optimal results in a small number of tries, which is due to the rules used to prune useless solutions.

1. Introduction

Design for low power has several motivations, such as prolonging battery life in wearable electronic devices, and reducing the cooling cost for high performance systems. Battery life is becoming a product differentiator in many portable electronic markets [8]. High performance systems are characterized by large power dissipation. This transforms to heat that can lead to system malfunction or that may force reducing system performance.

Dynamic power, P_{dyn}, is the main source of power dissipation in CMOS circuits. P_{dyn} depends on the square of the supply voltage, V_{dd}, as expressed by the following equation [3, 8]:

$$P_{dyn} = KC_{lc} f V_{dd}^2, \quad (1)$$

where K is the switching activity factor, C_{lc} is the loading capacitance, and f is the clock frequency.

Due to the quadratic term in equation (1), dynamic power may significantly be reduced by scaling down the supply voltage. Supply voltage reductions increase execution delays. To avoid decreasing performance, we can use performance determined assuming that the circuit operates at the highest acceptable supply voltage as a target. Then, the supply voltage of computational elements on non-critical paths can be reduced. Defined in this paper as $MinP_{dyn}$, the problem of minimizing the dynamic power consumption, under performance constraints, by scaling down the supply voltage of computational elements on non-critical paths is NP-hard in general [4].

Methods to obtain solutions to $MinP_{dyn}$ have been proposed. In [1], a polynomial-time algorithm has been proposed for performance-constrained non-pipelined designs. Given a predetermined set of supply voltages, the authors [1] schedule the acyclic datapath and assign voltages to the computational elements in order to minimize power. In their approach, it was assumed that the voltage versus delay curve is identical for all computational elements in the circuit. Under this assumption, the problem transforms to a problem where identical computational elements are used. Without this assumption, there is no guarantee that the algorithm produces optimal results.

A technique to reduce power consumption has been proposed in [2], where only two supply voltages are allowed. A depth-first search is used to determine computational elements which may operate at low supply voltage, without violating the timing constraints of the circuit. In this method, it is only after examining all its

successors that a computational element can be allowed to operate at low supply voltage. Nevertheless, significant reductions of power consumption can be obtained by selecting a computational element without first examining all its successors. Also, it is a question of how to choose the node to start the depth-first search in cyclic designs.

To optimally solve $MinP_{dyn}$ under resource constraints, an integer linear program approach for non-pipelined acyclic designs has been provided in [3]. The authors [3] also present a heuristic to solve the problem, but general designs, such as pipelined and/or cyclic datapaths, are not examined.

A dynamic programming technique to solve $MinP_{dyn}$ is proposed in [4]. The authors [4] reported that their method can produce optimal results for designs modeled as tree-like data flow graphs. Suboptimal results are obtained in the general case, such as pipelined design.

Methods based on software pipelining techniques have been recently proposed to derive multi-phase clocked sequential circuits operating at the optimal throughput [5, 7]. Solving $MinP_{dyn}$ for this class of circuits may allow to reduce the power consumption, and to reduce the number of registers in the circuit. Although operation chaining is assumed to reduce the required number of registers and the number of control signals in this class of circuits, further reductions may be obtained by a supply voltage scaling approach. Indeed, registers connected to computational elements on non-critical paths may be omitted, since the execution delay of these elements, after applying supply voltage scaling techniques, may be increased, which may make operation chaining possible. Control signals for these registers will then be saved, and the circuit area will be reduced too.

The approach we propose here is general and can be used at any level of the design hierarchy, but we focus in this paper on solving $MinP_{dyn}$ for the class of circuits mentioned above. These circuits contain feedbacks and can be modeled as cyclic graphs for optimization purposes. To the best of our knowledge, no work has been explicitly proposed yet for solving $MinP_{dyn}$, when the design is modeled as a cyclic graph. In this paper, we present an integer linear programming formulation to $MinP_{dyn}$ that we use to devise polynomial-time solvable method and an exact algorithm based on the branch-and-bound technique. Experimental results confirm the effectiveness of the method and power reduction factors as high as 53,84% were obtained. Also, they show that the exact algorithm produces optimal results in a small number of tries, which is due to the rules used to prune useless solutions.

The rest of the paper is organized as follows. In Section 2, we introduce the notations and definitions used in this work. Section 3 presents the formulation of the problem we tackle. We present a polynomial time solvable method to determine solutions to that problem in section 4, and an exact algorithm based on a branch-and-bound technique in section 5. Section 6 provides experimental results and Section 7 concludes the paper.

2. Preliminaries

2.1. The Cyclic Graph Model

A synchronous sequential circuit is modeled (as in [5, 7]) as a directed cyclic graph $G = (V, E, d, w)$, where V is the set of functional elements in the circuit, and E is the set of edges which represent interconnections between vertices. Each vertex v in V has a non-negative integer execution delay $d(v) \in N$. Each edge $e_{u,v}$, from u to v, in E is weighted with a register count $w(e_{u,v}) \in N$, representing the number of registers on the wire between u and v.

Figure 1 presents an example of a circuit and its directed cyclic graph model. In this figure, large rectangles represent functional elements, and small rectangles represent registers. Wires are oriented to show the propagation direction of the signals. The execution delay of each functional element of this circuit is specified as a label on the left of each large rectangle. This example will be used through this paper, and will serve as an example of initial specification for the problem to optimize. The initial specification is in general a synchronous circuit with a single-phase clock period, and it is assumed to operate at the highest acceptable supply voltage. As an example, the clock period of the circuit in Figure 1 is 6, which is equal to $d(v_4) + d(v_1)$.

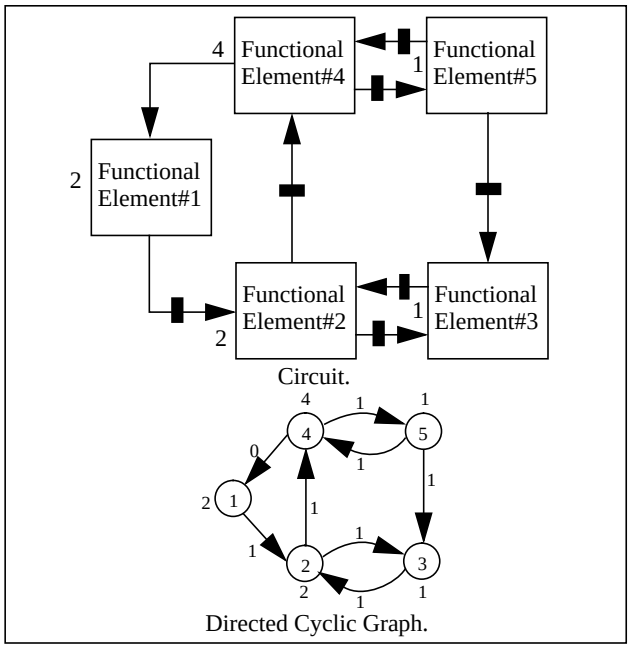


Figure 1 : Sample circuit and its directed cyclic graph model.

2.2. Periodic Schedules

We define a schedule s [11] as a function $s : N \times V \rightarrow Q$, where $s_n(v) \equiv s(n, v)$ denotes the schedule time of the n^{th} iteration of operation v. In multi-phase flip-flop based circuits, the schedule time is the start time of the operation. A schedule s is called periodic with period P, if:

$$\forall n \in N,\ \forall v \in V: s_{n+1}(v) = s_n(v) + P\,. \qquad (2)$$

When there is no resource constraint, a schedule s is said to be valid if and only if the operations terminate before their results are needed. In this case, we say that data dependencies are satisfied, which is equivalent to the following mathematical inequality:

$$\forall n \in N,\ \forall e_{u,v} \in E : s_{n+w(e_{u,v})}(v) \geq s_n(u) + d(u)\,. \qquad (3)$$

2.3. The Maximum Throughput of Synchronous Sequential Circuits

The throughput, T, of a synchronous sequential circuit is bounded by the inverse of the length, P, of the critical paths in the circuit. Based on data dependencies constraints only, the maximum throughput is [11]:

$$T = Min_{c \in C}\left(\left(\sum_{e_{u,v} \in c} w(e_{u,v})\right) \Big/ \left(\sum_{\forall v \in V \text{ and } e_{u,v} \in c} d(u)\right)\right), \qquad (4)$$

where C is the set of directed cycles in the directed cyclic graph modeling the circuit. Determining the maximum throughput is a Minimal Cost-to-Time Ratio Cycle Problem [6, 9, 10], which can be solved in the general case in $O(|V| \cdot |E| \cdot \log(|V| \cdot d_{max}))$ [6, 10], where $d_{max} = Max_{v \in V}(d(v))$. A possible method to solve this problem is to iteratively apply Bellman-Ford's algorithm for longest paths on the graph $G_P = (V, E, d, w_P)$ derived from G by letting:

$$w_P(e_{u,v}) = d(u) - P \cdot w(e_{u,v})\,, \qquad (5)$$

where $e_{u,v} \in E$ and $P = 1/T$. A binary search may be used to find the minimal value of P for which there is no positive cycle in G_P [10].

For the example in Figure 1, we have that $P = 4$. This value corresponds to the cycle defined by vertices v_1, v_2, and v_4.

2.4. Schedule for a Given Throughput

From equation (2) and inequality (3), we have that:

$$\forall e_{u,v} \in E,\ s_0(v) - s_0(u) \geq d(u) - P \cdot w(e_{u,v})\,. \qquad (6)$$

In the case of periodic schedules, determining a valid schedule of all the instances of each vertex v in V is equivalent to determining $s_0(v)$ for each v in V, which is also equivalent to determining solutions to the system of inequalities described by (6). To solve this system, the graph G_P, previously described, may be used. To find an ASAP schedule, Bellman-Ford's algorithm for longest paths, from a chosen vertex v_x to the others, may be applied on the graph G_P. Finding an ALAP schedule may be done as follows: step 1, a graph G' has to be derived from G_P by inverting the direction of each edge in G_P; step 2, Bellman-Ford's algorithm for longest paths, from the vertex v_x to the others, has to be applied on the graph G', where the weights of its edges are defined by equation (5); finally, step 3, the ALAP schedule is obtained by multiplying each result in step 2 by -1. Relatively to $v_x = v_1$, the ASAP schedules of vertices v_1, v_2, v_3, and v_4 of the circuit in Figure 1 are 0, -2, -4, -4 and -4, respectively. The ALAP schedules of these vertices are 0, -2, 1, -4 and -1, respectively.

2.5. Schedule Graph

As in [7], a periodic schedule, with period P, of vertices of directed cyclic graph modeling a circuit is presented by a schedule graph $G_s = (V, E, d, w_s, P)$. Here V, E and d have the same definition given for the case of the graph G previously defined, and $w_s : E \rightarrow Q$ is a weight function, which associates to each edge $e_{u,v}$ in E the time distance between the schedule times of u and v. Mathematically, $w_s(e_{u,v})$ is defined as follows:

$$\forall e_{u,v} \in E,\ w_s(e_{u,v}) = s_{w(e_{u,v})}(v) - s_0(u)\,. \qquad (7)$$

Because s is periodic with period P, equation (7) may be written as follows:

$$\forall e_{u,v} \in E,\ w_s(e_{u,v}) = s_0(v) - s_0(u) + P \cdot w(e_{u,v})\,. \qquad (8)$$

The graph G_s is consistent if and only if for each edge $e_{u,v}$ in E, $w_s(e_{u,v}) \geq d(u)$. This is derived from (6) and (8). Figure 2 shows a consistent schedule graph, where edges are labeled with w_s values, for the circuit in Figure 1 using the ALAP schedule presented in Section 2.4.

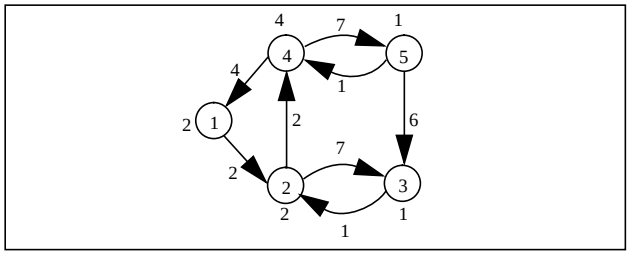
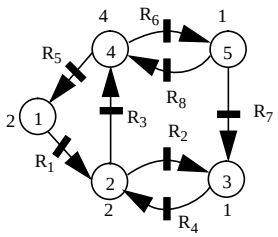


Figure 2 : Schedule graph.

2.6. Register Placement

In the method based on software pipelining proposed in [7], once the optimal clock period is determined and a schedule time of all the computational elements is computed, then a register placement step is needed in order to preserve the behavior of the original circuit. The placement of registers is derived from a schedule graph G_s, by breaking every path in G_s that is longer than the optimal clock period P. For paths having a length less than P, no register is required because operations chaining is assumed.

For the circuit in Figure 1, applying the algorithm in [7] for register placement on the schedule graph G_s in Figure 2, starting from v_1, gives the placement of registers and their schedules, as presented in Figure 3. The number of registers is 8 and the number of phases is 4.



Register	Schedule
R_1	2
R_2	1
R_3	0
R_4	2
R_5	0
R_6	3
R_7	1
R_8	0

Figure 3 : Register placement and their schedules.

3. Problem Formulation

Let P_h be the power consumed by the circuit, where each computational element operates at the highest supply voltage. The problem $MinP_{dyn}$ we tackle in this paper is defined as follows. Given that acceptable multiple different supply voltages are available, and that only one can be used to drive a computational element, then determine the schedule that leads to the maximum reduction of power consumption, relatively to P_h, under the requirements R_1 and R_2 defined in the following. For R_1, we are locking for a solution where each computational element operates at the lowest possible supply voltage. For R_2, the schedule has the same latency and throughput as when the circuit operates at the highest supply voltages only. Here, latency is defined as the time required to execute the first instance of all the operations.

Before presenting a mathematical formulation to that problem, let us first define some requirements. Based on the supply voltages used, we assume that we have n_i different implementations for each computational element *i*. If supply voltage V_{dd}^k, where $1 \le k \le n_i$, is used, then the computational element *i* has an execution delay $d(i) = d_{i,k}$ and consumes the power $p_{i,k}$. We suppose that supply voltages are sorted from the highest value to the smallest one, and numbered starting from 1. For each $i \in V$, and for each *k* such that $1 \le k \le n_i$, let us denote by $x_{i,k}$ an integer, which is equal to 1 if supply voltage *k* is used and to 0 otherwise. $MinP_{dyn}$ can be formalized as a mixed integer linear programming problem, $MILPMinP_{dyn}$, as presented in Figure 4. In this formulation, the objective function expresses the power consumed by all the computational elements. The variables are $x_{i,k}$'s and the schedule time $s_0(u)$ for each $u \in V$. *P* is the optimal clock period, which is determined as mentioned in Section 2.3 assuming that the circuit operates at the highest supply voltages. The other parameters are as defined in Section 2. Equation (9) ensures that only one of all the available implementations of computational element *i* will be used. Equation (10) is equivalent to (6), where the execution delay of *i* is $(\sum_{k=1}^{n_i} x_{i,k} d_{i,k})$. Equation (11) is used to have a schedule that has a latency less than or equal to that of the case where the circuit operates at the highest supply voltages only. Equation (12) and (13) are used to prune the solution space and to guarantee that the schedule will have the optimal throughput, where ASAP and ALAP schedules are computed assuming that the circuit operates at the highest supply voltages. Equation (14) is equivalent to the definition of $x_{i,k}$'s.

Formulation F_1:

$$Minimize \sum_{\forall i \in V} \sum_{k=1}^{n_i} x_{i,k} p_{i,k}$$

Subject to:

$$\forall i \in V, \sum_{k=1}^{n_i} x_{i,k} = 1 \quad (9)$$

$$\forall e_{i,j} \in E, s_0(j) - s_0(i) \ge (\sum_{k=1}^{n_i} x_{i,k} d_{i,k}) - P \cdot w(e_{i,j}) \quad (10)$$

$$\forall i \in V, s_0(i) + \sum_{k=1}^{n_i} x_{i,k} d_{i,k} \le ALAP_i + d_{i,1} \quad (11)$$

$$\forall i \in V, s_0(i) \ge ASAP_i \quad (12)$$

$$\forall i \in V, s_0(i) \le ALAP_i \quad (13)$$

$$\forall i \in V, k = 1, ..., n_i : x_{i,k} \in \{0, 1\} \quad (14)$$

Figure 4 : A Mixed integer linear programming formulation to $MinP_{dyn}$.

Due to the space limitation, proofs of the following theorems are omitted.

Theorem 1 : *The problem in Figure 4 has always a solution that is not necessarily optimal.*

Theorem 2 : *The optimal solution to the problem in Figure 4 has the maximum throughput, P.*

Since no efficient method is reported yet, in the literature, for solving large and general mixed integer linear programming problems, then devising heuristics to solve them is of great interest. In the subsequent section, we will present how we can transform the $MILPMinP_{dyn}$ to obtain efficient methods for solving $MinP_{dyn}$.

4. A Linear Programming Formulation to Determine a Solution to $MinP_{dyn}$

Scaling down the supply voltage of a computational element reduces its power consumption, but increases its execution delay. Based on this fact, replacing

$$Minimize \sum_{\forall i \in V} \sum_{k=1}^{n_i} x_{i,k} p_{i,k} \quad (16)$$

in the formulation in Figure 4, by

$$Maximize \sum_{\forall i \in V} \sum_{k=1}^{n_i} x_{i,k} d_{i,k}, \quad (17)$$

may produce significant power saving over the power consumed when the circuit operates at the highest supply voltages. Let $NMILPMinP_{dyn}$ be the resulting formulation.

Let

$$d(i) = \sum_{k=1}^{n_i} x_{i,k} d_{i,k}. \quad (18)$$

A linear programming formulation, $LPMinP_{dyn}$, can be derived from $NMILPMinP_{dyn}$ as presented in Figure 5. In this formulation, (19) is derived using (17) and (18). Inequality (20) is obtained using (10) and (18). Inequality (21) is determined using (11) and (18). Inequalities (22) and (23) are equivalent to (12) and (13), respectively. Inequalities (24) and (25) are equivalent to the fact that the execution delay of any implementation of the computational element i is between $d_{i,1}$ and d_{i,n_i}, as defined in Section 3.

Formulation F_2:

$$Maximize \sum_{\forall i \in V} d(i) \quad (19)$$

Subject to:

$$\forall e_{i,j} \in E,\ d(i) + s_0(i) - s_0(j) \le P \cdot w(e_{i,j}) \quad (20)$$

$$\forall i \in V,\ s_0(i) + d_i \le ALAP_i + d_{i,1} \quad (21)$$

$$\forall i \in V,\ -s_0(i) \le -ASAP_i \quad (22)$$

$$\forall i \in V,\ s_0(i) \le ALAP_i \quad (23)$$

$$\forall i \in V,\ -d(i) \le -d_{i,1} \quad (24)$$

$$\forall i \in V,\ d(i) \le d_{i,n_i} \quad (25)$$

Figure 5 : A linear programming formulation to determine a solution to $MinP_{dyn}$.

Once the problem in Figure 5 is solved, the supply voltage under which the computational element i must operate is V^m_{dd}, where m is an integer such that $d_{i,m} \le d(i) < d_{i,m+1}$. The schedule of computational elements may have to be recomputed using, for instance, methods provided in [5].

Due to the space limitation, proofs of the following theorems are omitted.

Theorem 3 : *The problem in Figure 5 has always a solution.*

Theorem 4 : *The optimal solution to the problem in Figure 5 has the maximum throughput, P.*

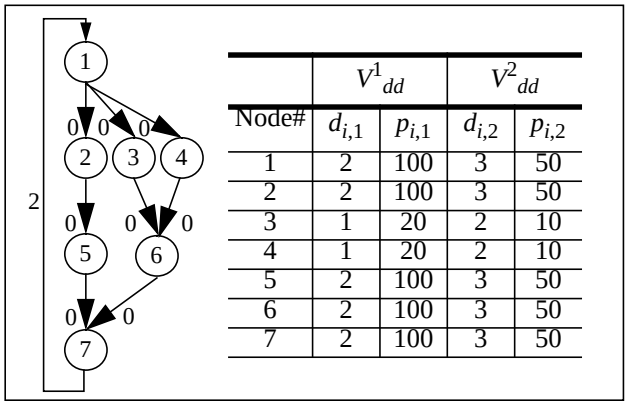
Theorem 5 : *The problem in Figure 5 can be solved in polynomial time.*

In the case where the computational elements exhibit significant differences in the power reduction to delay increase tradeoff, the objective function in Figure 5 may not provide the best power reduction. Let us first give an example and then show how we can improve the objective function to deal with such situation. For the example in Figure 6, the critical path is formed by nodes 1, 2, 5 and 7. The paths 1, 3, 6, 7, and 1, 4, 6, 7 are not critical. Indeed, the maximum throughput is 1/4 and the ASAP schedules for nodes 1, 2, 3, 4, 5, 6 and 7 are 0, 2, 2, 2, 4, 3 and 6, respectively. Their ALAP schedules are 0, 2, 3, 3, 4, 4 and 6, respectively. Critical nodes have the ASAP schedules equal to their ALAP schedules. Now, to scale down the supply voltage of the nodes on non-critical paths, we have two possibilities: scaling down the supply voltage of nodes 3 and 4 or of node 6. If the first case is chosen, the power consumed will be reduced by 20 units; whereas in the second case, it will be reduced by 50 units. However, by examining the average power consumed per unit of time, c_i, for nodes 3, 4, and 6, we remark that for nodes 3 and 4, we have that $c_4 = c_3 = (p_{3,1} + p_{3,2})/(d_{3,1} + d_{3,2}) = 10$, and for node 6, we have that $c_6 = (p_{6,1} + p_{6,2})/(d_{6,1} + d_{6,2}) = 30$. Hence, injecting these coefficients into the objective function would favor slowing node 6 over nodes 3 and 4. Based on that fact, formula (19) can be replaced by:

$$Maximize \sum_{\forall i \in V} c_i \cdot d(i), \quad (26)$$

where c_i's are defined as follows:

$$c_i = (\sum_{k=1}^{n_i} p_{i,k})/(\sum_{k=1}^{n_i} d_{i,k}). \quad (27)$$



	V^1_{dd}		V^2_{dd}	
Node#	$d_{i,1}$	$p_{i,1}$	$d_{i,2}$	$p_{i,2}$
1	2	100	3	50
2	2	100	3	50
3	1	20	2	10
4	1	20	2	10
5	2	100	3	50
6	2	100	3	50
7	2	100	3	50

Figure 6 : Example to explain how Figure 5 can be improved.

5. An Exact Algorithm to Solve $MinP_{dyn}$

To solve optimally $MinP_{dyn}$, we provide an exact algorithm, *ExactAlgo*, based on a branch-and-bound technique as presented in Figure 8. Let $config_G$ be the set of supply voltages at which the computational elements of G operate. Initially, $config_G$ holds the highest supply voltages only. *ExactAlgo* has as input a graph G, an integer i that controls the termination of the algorithm, an upper bound, *UpperBound*, on the power consumed by the circuit, and the optimal clock period, P, of the circuit. Initially, the circuit operates at the highest supply voltages, which allows to determine *UpperBound*, P and ASAP and ALAP schedules for computational elements. These schedules are saved as *ASAP_save* and *ALAP_save*. Example of implicit tree to be explored by *ExactAlgo* is presented in Figure 7. *ExactAlgo*

has to be called using $i = 1$. At each time *ExcatAlgo* is called, one and only one computational element is switched to the next low supply voltage, which leads to the creation of a new node in the searching tree. In order to be accepted, the configuration of the graph at that node must satisfy the following conditions: (1) the graph is still working at the maximum throughput *P*, (2) ASAP and ALAP schedules are still bounded by the values of *ASAP_save* and *ALAP_save*, and (3) each computational element *i* terminates its execution no later than $d_{i,1} + ALAP\_\text{save}_i$. If one of these constraints is not satisfied, then the computational element is switched back to the last admissible supply voltage; in this case, the process of trying another supply voltage for *i* is stopped, and we then examine the next computational element of the node where we are; if all the computational elements of that node are examined, then *ExactAlgo* backtracks or it terminates if it passed all the computational elements of the first node at which the algorithm started to execute. If a node is accepted, then a new node is created in the searching tree in order to examine the next computational element. This new node has as parent the computational element whose supply voltage was just updated at the accepted node. In *ExactAlgo*, an early check of the acceptance of switching the supply voltage to the next one is done by verifying if $d_{i,k} \le d_{i,0} + ALAP\_\text{save}_i - ASAP\_\text{save}_i$, which is derived from inequality (11) assuming that $s_0(i) = ASAP\_\text{save}_i$.

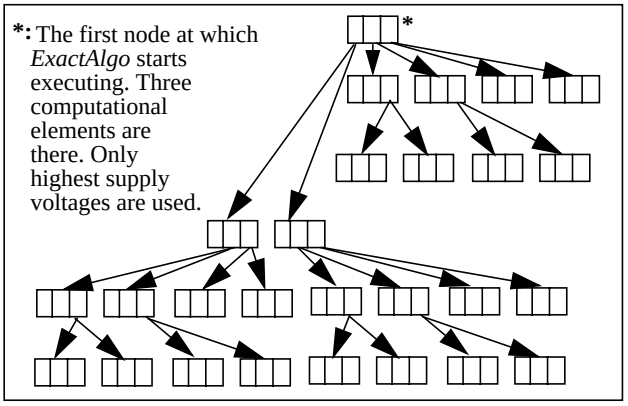


Figure 7 : Example of implicit tree to be explored by *ExactAlgo*. Three supply voltages are assumed.

6. Experimental Results

To test the effectiveness of the approach we propose, we experimented the linear programming formulation in Figure 5 and the exact algorithm in Figure 8 on some benchmarks used in the literature and on Figures 1 and 6. Circuits from the ISCAS89 benchmark suite have been used to test the efficiency, in terms of reducing power consumption and the run-time, of the linear programming formulation. We use four supply voltages: 5V, 3.3V, 2.4V, and 1.5V. To determine the delay $d_{i,k}$ and the power consumed $p_{i,k}$ for each computational element *i*, when the supply voltage V_{dd}^k is used, we proceed as follows. We use the rule [2] $d_{i,k} = (V^k_{dd}/(V^k_{dd} - V_{th})^2) \cdot ((V^1_{dd} - V_{th})^2 / V^1_{dd}) \cdot d_{i,1}$, where V_{th} is the threshold voltage. V^1_{dd} is assumed known and V_{th} is fixed at 0.7V, which is a typical value used in the literature. $p_{i,k}$'s are determined assuming that the fanout of the computational element *i* has the same loading capacitance. By using equation (1), we have that:

$$p_{i,k} = KC_0 f . (Fanout_i \cdot (V^k_{dd})^2) . \qquad (28)$$

For Figure 5 with formula (26), we use the LP_Solve tool [12] to solve the linear program for each circuit. The linear program is automatically generated by a module we coded in C++ and integrated in the tool we developed in [5]. Results are summarized in Table 1, where the first column gives the circuit name, the second column gives the power consumed (divided by KC_0f) in the case of using the highest supply voltage only (P_h) and in the case of using multiple supply voltages (P_m), the third column presents the power saving factor defined as $((P_h - P_m)/P_h) \times 100$, and the last column gives the run-time in second.

ExactAlgo is coded in C++ and integrated in the tool we developed in [5]. Table 2 reports the obtained results. Column 1, 2 and 3 are the same as for Table 1. Column 4 presents the number of tries done divided by the size, k^n, of the searching space, where *k* is the number of the different available supply voltages and *n* is the number of the computational elements in the circuit.

As Table 1 reports, significant reductions of power consumption are obtained in a short run-time using the linear programming formulation. Indeed, saving factors ranging from 5.3% to 53.84% are obtained in less than 166.81s using an UltraSparc 10 with 1GB RAM. As summarized by Table 2, optimal power consumptions using multiple supply voltages are obtained in a small number of tries in a very large searching space. For instance, for the circuit FOWDEF, we have that $n = 34$ and $k = 4$; nevertheless, only 22399 possibilities are examined (in 44.01 s) to find the schedule with the minimal power consumption.

7. Conclusions

The problem, $MinP_{dyn}$, of minimizing the power consumption, under performance constraints, by supply voltage scaling is NP-hard in general. To the best of our knowledge, no work has been explicitly reported yet in the literature to tackle this problem in the case of cyclic designs.

Software pipelining has recently been used to derive multi-phase clocked sequential circuits operating at the maximum throughput. To preserve the functionality of the original circuit, registers are placed after the optimal clock period *P* and a schedule of the computational elements of the circuit are determined. During registers placement, only

paths that have a length greater than P are broken since operation chaining is assumed.

Solving $MinP_{dyn}$ for the class of multi-phase clocked circuits is very interesting since it may allow to reduce the number of registers and hence to save the power they consume, to reduce the number of the control signals, and to reduce the area of the circuit.

We have provided a mixed integer linear formulation to $MinP_{dyn}$, and use it to devise a polynomial time solvable method and an exact algorithm based on a branch-and-bound technique. Experimental results confirmed the effectiveness of the method and showed that power reduction factors as high as 53.84% can be obtained in short run-times. They also confirmed the efficiency of the exact algorithm. Indeed, optimal results are obtained in a small number of tries in a very large searching space.

Table 1 : Results using Figure 5.

	(Power Consumed)/ (KC_0f)		Gain = (Column #2- Column# 3)/ Column# 2	Run-time (sec)
	Using the highest supply voltages	Using multiple supply voltages		
Figure 1	205	176	14.14%	0.01
Figure 6	232	202	12.93%	0.01
SOIIR Filter [11]	129	114	11.62%	0.01
Polynomial Div. [11]	309	251	18.77%	0.01
Correlator [7]	283	268	5.3%	0.01
FOWDEF [15]	1306	1181	9.57%	0.01
S344	6225	4146	33.39%	0.2
S641	8457	5980	29.28%	0.17
S731	9758	6549	32.88%	0.22
S5378	64847	47234	27.16%	14.37
S9243	46905	22712	51.57%	12.38
S1238	25017	17896	28.46%	1.81
S1423	24533	11323	53.84%	1.87
S1488	33360	28144	15.63%	3.64
S1494	33868	28431	16.05%	5.93
S13207	116500	62255	46.56%	79.76
S15850	139403	69383	50.22	166.81

Table 2 : Results using *ExactAlgo*.

	(Power Consumed)/(KC_0f)		Gain = (Column #2- Column #3)/ Column #2	(Number of tries)/ (The size of the searching space)
	Using only the highest supply voltages	Using multiple supply voltages		
Figure 1	205	176	14.14%	9/1024
Figure 6	232	202	12.93%	20/16384
SOIIR Filter [11]	129	114	11.62%	8/256
Polynomial Div. [11]	309	209	32.36%	741/262144
Correlator [7]	283	268	5.3%	13/65536
FOWDEF [15]	1306	1144	12.4%	22399/(4^{34})

References

[1] S.Raje and M.Sarrafzadeh, "Variable Voltage Scheduling", *In Proc. of the Int. Sym. on Low Power Design*, Monterey, CA, Aug., 1995.

[2] K.Usami and M.Horowitz, "Clustered Voltage Scaling Technique for Low-Power Design", *In Proc. Int. Workshop on Low Power Design*, 1995.

[3] Y.-R.Lin and A.-C.-H. Wu, "Scheduling Techniques for Variable Voltage Low Power Designs", *ACM Transactions on Design Aut. of Elec. Sys.*, Vol.2, N0.2, April, 1997.

[4] J.-M.Chang and M.Pedram, "Energy Minimization Using Multiple Supply Voltages", *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, Vol.5, No.4, Dec., 1997.

[5] N. Chabini, E.-M. Aboulhamid and Y. Savaria, "Reducing Register and Phase Requirements for Synchronous Circuits Derived Using Software Pipelining Techniques", *Proceedings of the IEEE Computer Society Annual Workshop on VLSI*, Orlando, Florida, April 19-20, 2001.

[6] N. Chabini, E.-M. Aboulhamid and Y. Savaria, "A Fast Method for Determining an Efficient Bound on the Optimal Solution of the Cost-to-Time Ratio Problem", *To appear in the Proceedings of the SCI/ISAS*, Orlando, Florida, July 22-25, 2001. `http://www.iiis.org/SCI/`

[7] F.-R. Boyer, E.-M. Aboulhamid, Y. Savaria and M. Boyer, "Optimal Design of Synchronous Circuits Using Software Pipelining Techniques", *ACM Transactions on Design Aut. of Electronic Systems*, Vo. 7, Num. 2, 2002.

[8] Gary K. Yeap, *Practical Low Power Digital VLSI Design*, Kluwer Academic Publishers, ISBN: 0-7923-8009-6.

[9] S.-H. Gerez, S.-M.-H. de Groot, and O.-E. Herrmann, "A Polynomial-Time Algorithm for the Computation of the Iteration-Period Bound in Recursive Data- Flow Graphs", *IEEE Trans. on Cir. and Syst.-1*, No. 1, Vo. 39, 1992.

[10] E.-L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Reinhart, and Winston, New York, NY, 1976.

[11] I.-E. Bennour, *Estimation de la performance et méthodes d'allocation dans la synthèse de systèmes numériques, Thèse de doctorat*, DIRO, Université de Montréal, 1996.

[12] The LP_Solve Tool: `ftp://ftp.ics.ele.tue.nl/pub/lp_solve/`

[13] L.-G. Khachian, "A Polynomial Algorithm in Linear Programming", *Soviet Math. Doklady*, Vo. 20, 1979.

[14] N. Karmakar, "A New Polynomial-Time Algorithm for Linear Programming", *Combinatorica*, Vo. 4, 1984.

[15] S. Y. Kung, H. J. Whitehouse and T. Kailath, *VLSI and Modern Signal Processing*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1985, pp. 259-60.


```
Function ExactAlgo (Graph: G = (V, E, d, w) , integer i,
                    float UpperBound, rational P) {
  if (i > |V|) then return;
  else {
    integer j;
    for (j=i; j<= |V|; j++) {
      for (k=2; k<=NumberOfVdd; k++) {
        if (d_{i,k} >d_{i,0} + (ALAP_save_i - ASAP_save_i))
        then { k = NumberOfVdd + 1; break;}
        else {
          save_d_i = d_i;
          d_i = d_{i,k};
          determine the optimal period, P_opt, for the updated graph G;
          determine the ASAP and ALAP schedules for the updated graph G;
          if (P_opt > P or
           ∃x ∈ V : [ASAP_x,ALAP_x] ⊄ [ASAP_save_x,ALAP_save_x] or
           ∃x ∈ V : (ALAP_x + d_x > ALAP_save_x + d_{x,1})
          ) then
          {k = NumberOfVdd + 1; break;}
          else {
            determine the power, power, consumed by the updated graph G;
            ExactAlgo ( G = (V, E, d, w) , i+1, UpperBound, P);
            if (power < UpperBound) then {
              UpperBound = power;
              save the present configuration of G into config_G ;
              // config_G initialy holds the highest supply voltages
            }
          }
          d_i = save_d_i;
        }
      }
    }
  }
}
```

Figure 8 : An algorithm based on a branch-and-bound technique.