Optimal Simultaneous Module and Multivoltage Assignment for Low Power

DEMING CHEN
University of Illinois, Urbana-Champaign
JASON CONG
University of California, Los Angeles
and
JUNJUAN XU
Synopsys, Inc.

Reducing power consumption through high-level synthesis has attracted a growing interest from researchers due to its large potential for power reduction. In this work we study functional unit binding (or module assignment) given a scheduled data flow graph under a multi-Vdd framework. We assume that each functional unit can be driven by different Vdd levels dynamically during run time to save dynamic power. We develop a polynomial-time optimal algorithm for assigning low Vdds to as many operations as possible under the resource and latency constraints, and in the same time minimizing total switching activity through functional unit binding. Our algorithm shows consistent improvement over a design flow that separates voltage assignment from functional unit binding. We also change the initial scheduling to examine power/energy-latency tradeoff scenarios under different voltage level combinations. Experimental results show that we can achieve 28.1% and 33.4% power reductions when the latency bound is the tightest with two and three-Vdd levels respectively compared with the single-Vdd case. When latency is relaxed, multi-Vdd offers larger power reductions (up to 46.7%). We also show comparison data of energy consumption under the same experimental settings.

Categories and Subject Descriptors: B.5.1 [**Register-Transfer-Level Implementation**]: Design—*Data-path design*; B.5.2 [**Register-Transfer-Level Implementation**]: Design Aids—*Optimization*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Network problems*

A preliminary version of this work was presented in *Proceedings of the 2005 Asia South Pacific Design Automation Conference* (Shanghai, China), 850–855.
This work was partially supported by National Science Foundations (NSF) grants CCR-0306682 and CCR-0096383 and by Altera Corp. under the California MICRO program.
D. Chen and J. Xu were affiliated with the University of California, Los Angeles, at the time of the research for this article.
Authors' addresses: D. Chen, Department of Electrical and Computer Engineering, University of Illinois, Urbana, IL 61801; email: dchen@uiuc.edu; J. Cong, Department of Computer Science, University of California, Los Angeles, Los Angeles, CA 90095; email: cong@uiuc.edu; J. Xu, Synopsis Shanghai, 14-16F Zhaofeng Plaza, 1027 Changning Road, Shanghai, 200050, China; email: Junjuan.Xu@synopsys.com.

General Terms: Algorithms, Design, Theory

Additional Key Words and Phrases: Data path generation, functional unit binding, high-level synthesis, level conversion, low power design, multiple voltage, power optimization, scheduling

1. INTRODUCTION

With the exponential growth of the performance and capacity of integrated circuits, power consumption has become one of the most critical constraining factors in the IC design flow [ITRS 2003]. Excessive power consumption limits the degree of transistor integration on a single chip, requires expensive packaging and cooling systems, shortens battery lifetime for portable devices, and brings on problems of signal integrity. In his keynote speech at DAC'04, Intel CTO Patrick Gelsinger mentioned that delivering performance in power envelop was one of the biggest technology challenges in the future [Gelsinger 2004]. Rigorous low-power design will require power optimization through the entire design flow to achieve maximal power reduction.

There are two major sources of power consumption: dynamic power and static power. Dynamic power is consumed when signal transitions take place at gate outputs. Static power (also called leakage power) is consumed when the circuit is either active or idle. According to Kao et al. [2002], static power may take up to 42% of total power in 90-nm technology. In Li et al. [2003], a similar percentage is reported for certain FPGA architectures in 100-nm technology. Therefore, both dynamic and static power needs to be optimized.

Dynamic power consumption is calculated as $P_d = 0.5 \cdot S \cdot C \cdot V_{dd}^2 \cdot f$, where S denotes the switching activity of the circuit, C denotes the effective capacitance, V_{dd} is the supply voltage, and f is the operating frequency. To lower dynamic power, each of these factors can be reduced. Deploying multiple supply voltages is one of the most effective techniques to reduce dynamic power. This technique has the advantage of reducing power dissipation without sacrificing the performance of the system by assigning high Vdd to critical paths and low Vdd to non-critical paths. Clusters of high-Vdd cells and low-Vdd cells were first explored in Usami and Horowitz [1995]. The work in Takahashi et al. [1998] adopted multiple supply voltages in the real design of a MPEG4 video codec. To reduce static power, power gating is an efficient technique [Duarte et al. 2002; Mutoh et al. 1995]. When there are no useful operations executing on a module, it can be shut down to get rid of both dynamic and static power.

Our work studies power optimization at the behavioral level. The higher the design level is, the more critical the design decisions are for the quality of the final result. The behavioral synthesis process mainly consists of three stages: scheduling, allocation, and assignment. Scheduling determines when a computational operation will be executed; allocation determines how many instances of each type of resources (functional units, registers, or interconnection units) are needed; assignment assigns/binds operations, variables, or data-transfers to these resources. The last process is called functional unit binding when working with operations. Some people use module assignment to refer to the same concept. The number of resources may be limited and the total

time (latency) to finish the operations can be constrained. This makes most of the high-level synthesis problems difficult. The essence of behavioral synthesis with multiple supply voltages is to assign low-Vdd values to as many operations as possible under latency and resource constraints. In Raje and Sarrafzadeh [1995], an optimal solution was given for time-constrained scheduling problem for data-flow graphs under multiple voltages. No resource constraint was considered. In Chang and Pedram [1997], a scheduling algorithm (with binding as a post-processing step) was presented. It considered multiple supply voltages and switching activities in its energy model. Works in Johnson and Roy [1997]; Lin et al. [1997]; and Manzak and Chakrabarti [2002] proposed different heuristics for the time- and resource-constrained scheduling and binding problem under multiple voltages. These works adopted iterative methods to perform the two subtasks simultaneously. However, no switching activity reduction through binding was considered in their formulations. There are quite some works that focus on resource binding alone. Works in Chang and Pedram [1995, 1996] and Lyuh and Kim [2003] minimized switching activity for various resources, such as registers, functional units, and buses, but only single Vdd was considered. There is no optimal algorithm that combines both voltage assignment and resource binding for power reduction.

In this article, we focus on operational binding with voltage assignment, and derive an optimal algorithm to simultaneously assign maximum number of operations to low Vdd levels and minimize total switching activity through functional unit binding for the design. We use a network flow formulation. The solution of the min-cost flow will produce the binding and voltage assignment solutions. All of these are done under latency and resource bounds given by the initial scheduling. In addition, we change the initial scheduling to study power/energy-latency trade-offs, and provide power/energy optimization solutions under different design constraints. We design our architecture model in such a way so that functional units can be driven by different Vdd levels, or get into a sleep mode. Thus, we can target reducing dynamic power through multiple Vdds and reducing static power through power gating. Experimental results show that we can achieve significant amount of power savings compared to the single-Vdd case.

In the following, Sections 2 and 3 provide the details of our architecture model and power model. Section 4 describes our simultaneous multi-Vdd assignment and functional unit binding in detail. Section 5 shows experimental results, and Section 6 concludes this article.

2. ARCHITECTURE MODEL

We use the dual-Vdd case as an example to present our architecture model. It is shown in Figure 1. We insert two PMOS transistors between the high-Vdd (VddH) and low-Vdd (VddL) power rails and a functional unit (FU). The PMOS transistors are like sleep transistors, and the control bits C_1 and C_2 are used to control them so that an appropriate supply voltage can be chosen for the FU. When both transistors are off, the FU is in the sleep mode. This scheme is similar to that used in Li et al. [2004], where each configurable logic block

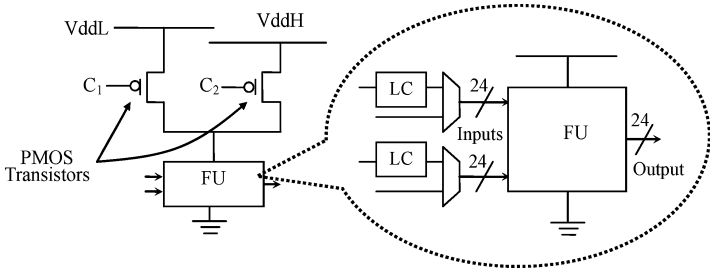


Fig. 1. Proposed architecture scheme for dual supply voltages.

(CLB) in an FPGA is in such an arrangement. We believe functional unit-level granularity for multi-Vdd configuration is natural for high-level synthesis. In addition, we assume that the FU's voltage can be dynamically changed during run time, which dramatically improves the chances for operations to execute under VddL. A more detailed diagram of the FU shows level converters (LC) at the input ports. A VddL signal needs to go through the level converter if it is going to drive a VddH device. Otherwise, the signal can bypass the converter through the MUX. We use the converter design from Chen et al. [2004]. A single level converter contributes 0.08-ns delay and 9.7E-15 Joul energy per switch. The MUX associated with the converter contributes 14 ps delay and about 2.0E-15 Joul energy per switch. All of these data were obtained with 100 nm technology [Chen et al. 2004]. The bit-width of the FU is 24. We assume that we can use an arbitrary number of voltage levels as long as it is realizable and reasonable practically in the architecture design. For example, an architecture with three Vdds will have three power rails and three PMOS transistors for each FU to control the voltage selection. Our main focus is to study the impact of different voltage levels and their combinations on power/energy reduction systematically, while considering both voltage assignment and functional unit binding simultaneously.

According to previous works, the overhead of dual-Vdd power rails and level converters is acceptable compared to the amount of power savings achieved. A new layout style of standard cells for ASIC designs was proposed in Usami et al. [1998], showing that adding a second power grid and level converters increased circuit area by 15%, but saved power by 47%. For FPGA designs, the area overhead of sleep transistors was 24% over the original CLB size with 5% delay overhead, and the power consumption of the sleep transistors could be optimized and become almost ignorable [Li et al. 2004].

3. POWER MODEL AND ANALYSIS

3.1 Resource Characterization

We use delay and power data extracted from Chen et al. [2003] for adders and multipliers driven by VddH = 1.3v. The data was obtained through an FPGA evaluation tool *fpgaEva_LP* [Li et al. 2003] under 100-nm technology. We add in several more VddL values to extend the voltage domain of our study. The characterization data for the functional units driven by different VddL values

Table I. Characterization of FUs for Various Supply Voltages

Characterization Items	Adder/Subtractor				
	VddH	VddL1	VddL2	VddL3	VddL4
Voltage Level (v)	1.3	1	0.8	0.7	0.5
Exe Delay (ns)	6.1	8	10.6	12.7	23.3
Exe Cycle	1	2	2	3	4
Power (w)	0.016	0.0095	0.006	0.0046	0.0024
E per Switch (J)	3.20E-10	1.89E-10	1.20E-10	9.28E-11	4.73E-11
	Multiplier				
	VddH	VddL1	VddL2	VddL3	VddL4
Voltage Level (v)	1.3	1	0.8	0.7	0.5
Exe Delay (ns)	14.6	19.2	25.3	30.5	55.8
Exe Cycle	3	4	5	5	9
Power (w)	0.246	0.146	0.093	0.071	0.036
E per Switch (J)	4.90E-09	2.90E-09	1.86E-09	1.42E-09	7.25E-10

are obtained through scaling. The threshold voltage for the transistors stays as a constant $V_{th} = 0.25$v. Therefore, as the voltage scales down, the delays of the resources become longer.[1] Meanwhile, both dynamic and leakage power scales down as well.[2] The clock period is set as 6.5 ns, that is, the delay of each cycle (control step) in the schedule takes 6.5 ns.

Table I shows the details. *Exe Cycle* represents the number of cycles for the operation to finish one 24-bit addition or multiplication. *E per Switch* is the energy consumed by the adder or multiplier when the output of the FU has a full voltage swing from logic 0 to 1. Notice that we use the data related to FPGA only because these data are available in recent publications. Our work can be applied to the ASIC design flow as well.

3.2 Power Gating and Voltage Switching

Next, we derive the conditions of applying power gating and compute the power overhead to charge an FU from VddL to VddH. According to the data presented in Li and He [2001], the circuit controlled by a sleep transistor needs at least one cycle to shut down and another cycle to come back alive. The maximum turn-on charging current can reach up to 87% larger than the normal switching current. Therefore, the turn-on power overhead (dynamic power) is at least equal to the dynamic power consumed during the normal operation. We can quantify this overhead by the following formula:

$$P_{overhead} = \frac{Ratio_{signal\_restore}}{0.5 \cdot SA_{FU}} \cdot DynamicPower, \tag{1}$$

where $Ratio_{signal\_restore}$ is the percentage of signals that are to be restored to logic high to power up the FU, and SA_{FU} is the switching activity for the FU, which

[1]Delay of the resource is proportional to $\frac{V_{dd}}{(V_{dd}-V_{th})^{\alpha}}$ [Gonzalez 1997]. We use $\alpha = 1.6$ in this work.

[2]Dynamic power scales down through the term V_{dd}^2. Leakage power scales down due to the scaling of V_{DS} (drain/source potential difference) and V_{GS} (gate/source potential difference) while V_{th} being maintained as a constant [Anderson and Najm 2004]. We consider this effect in our power model. When a functional unit stays idle but is not shut down (to be explained later), it will be driven by the lowest possible voltage level available in the architecture to reduce leakage power.

counts signal switching of both 0→1 and 1→0. We assume that, on average, half of the signals are to be restored to logic high in the FU, that is, $Ratio_{signal\_restore} = 0.5$. We can obtain SA_{FU} through simulations on our designs. $P_{overhead}$ captures the power overhead due to a full swing of logic 0 to 1. Since power gating only saves static power (assuming no signal switches for idle FUs), we need to guarantee that the static power saved will surpass the turn-on power overhead before turning off the FU. Thus, we define the following formula to calculate the number of sleep cycles for a FU to start saving power through power gating:

$$SleepCycle = \left\lceil \frac{P_{overhead}}{StaticPower} \right\rceil + 2. \tag{2}$$

The number 2 at the end counts in one cycle to turn off the FU and one cycle to turn on the FU. By this formula, it will need 9 (13) cycles for our adder (multiplier) to remain idle to guarantee that turning off the FU will save power.[3] Charging energy can be calculated as follows [Li et al. 2003]:

$$E(V_1 \rightarrow V_2) = \frac{C}{2}(V_1 - V_2)(V_1 + V_2 - 2V_{dd}). \tag{3}$$

C is load capacitance; V_1 is the initial value of gate output with a rising transition; V_2 is the final voltage. V_2 = VddH in our case. Plug in our VddL and VddH values, the charging energy is relatively small. For example, charging from 0.8v to 1.3v is only 15% of the charging energy compared to that from GND to 1.3v. Our *Exe Cycle* numbers assigned to the VddL operations provide enough cushion time.[4] Since the charging from VddL to VddH can be done in a much shorter time than that from GND to VddH (turn-on time), we don't need an extra cycle when the FU's voltage changes from VddL to VddH or vice versa, by taking advantage of the cushion time available.

3.3 Switching Activity Estimation

We use an efficient simulation-based switching activity calculator, which is similar to Bogliolo et al. [1999]. We perform simulation just once at the beginning and estimate the switching activity between every pair of operations if this pair of operations can be bound into a single functional unit. We can also compute switching activities for any legal binding solution afterwards without repeating simulations. We take a scheduled design so each operation in the design is already assigned to a certain control step.

Two operations are comparable if they can be bound to the same functional unit (to be formally defined later). We define $C_{in}(O_1, O_2)$ as the *input toggle count* from operation O_1 to operation O_2 when these two operations are bound into a functional unit W. It represents the input transitions when W switches

[3]Leakage power in the total power consumption is 23% for adders and 16% for multipliers in our characterization. Average SA_{FU} is equal to 0.5 in our case. The adder's *SleepCycle* = ceiling [0.77/(0.5*0.23)] + 2 = 9. The SleepCycle of the multiplier is similarly calculated.

[4]For example, 0.8v addition only needs 10.6 ns. There is a 2*6.5–10.6 = 2.4ns cushion time between the end of the addition and the start of a new cycle. This assumes that an operation can cross multiple clock cycles through proper controller design.

the execution from O_1 to O_2. Let $(I_1 \rightarrow I_2 \ldots \rightarrow I_K)$ be a set of primary input vectors for the design, $C_{in}(O_1, O_2)$ can be calculated as follows:

$$C_{in}(O_1, O_2) = \sum_{j=1}^{K} D_H\left(I_1^j, I_2^j\right), \tag{4}$$

where $D_H(X, Y)$ represents the Hamming Distance between bit vectors X and Y. I_1^j is the bit vector on the input ports of W when executing O_1 under the primary input vector I_j (I_j propagates through the design and generates new bit vectors for the internal operational nodes), and I_2^j is the bit vector on W when executing O_2 under the same primary input vector I_j. Notice that W has two ports. We use $C_{in}(O_1, O_2)$ to represent the input toggle counts of both ports for simplicity reason. Similarly, we can calculate the *output toggle count* $C_{out}(O_1, O_2)$ for W while executing O_1 and O_2. The switching activity for binding O_1 and O_2 together is estimated below:

$$S_{12} = \frac{C_{in}(O_1, O_2) + C_{out}(O_1, O_2)}{3 \times Bit\_width \times K}, \tag{5}$$

where Bit_width is the input vector width of W(set as 24 in our study).

We now present the method to estimate the switching activity on the design after functional unit binding is done. For each functional unit, a set of operations are assigned to it in a certain order. For functional unit W, let $(O_1 \rightarrow O_2 \ldots \rightarrow O_N)$ be the *operation set* in the execution order. We still have $(I_1 \rightarrow I_2 \ldots \rightarrow I_K)$ as primary input vectors. $C_{in}(O_i, O_{i+1})$ and $C_{in}(O_N, O_1)$ are defined as follows:

$$C_{in}(O_i, O_{i+1}) = \sum_{j=1}^{K} D_H\left(I_i^j, I_{i+1}^j\right) \tag{6}$$

$$C_{in}(O_N, O_1) = \sum_{j=1}^{K-1} D_H\left(I_N^j, I_1^{j+1}\right) \tag{7}$$

where $1 \leq i < N$. $C_{in}(O_N, O_1)$ is the toggle count when W switches operation from O_N back to O_1 when a new input vector arrives on the primary inputs. The switching activity of the inputs on W is defined as

$$S_{in} = \frac{\sum_{i=1}^{N-1} C_{in}(O_i, O_{i+1}) + C_{in}(O_N, O_1)}{2 \times Bit\_width \times (N \times K - 1)}. \tag{8}$$

A matrix of C_{in} can be constructed and used for looking up when calculating S_{in} after every binding solution. For two comparable operations O_i and O_j, there will be two entries $[O_i, O_j]$ and $[O_j, O_i]$ in the pre-calculated matrix. Suppose O_i is scheduled before O_j, the value of $[O_i, O_j]$ is from Eq. (6) and the value of $[O_j, O_i]$ is from (7). After binding, the *operation set* is known for every functional unit. According to the execution order of the *operation set*, every C_{in} value is looked up in the matrix, and the input switching activity can be calculated based on Eq. (8). The toggle count and the switching activity of the output of W are similarly calculated.

3.4 Overall Power Estimation

After voltage assignment and binding for the operations, we estimate the switching activity for each FU. Both dynamic power and static power are estimated and accumulated when the FU is active. Static power of the FU is estimated and accumulated when the FU is idle without power gating. The effect and overhead of power gating are counted when it is applied. The effect of power reduction due to voltage scaling is calculated. We also consider the power overhead due to voltage switching on a FU and the power overhead of level converters. One thing worth mentioning is that we do not count the power overhead of multiple power rails because it is hard to quantify without a real layout of the chip.

4. OPTIMAL VOLTAGE ASSIGNMENT WITH FUNCTIONAL UNIT BINDING

4.1 Problem Formulation

We define the problem of optimal voltage assignment with functional unit binding (**optVF** problem) as follows:

Inputs. A scheduled data-intensive design (its operations and data dependencies can be represented by a data flow graph); a set of predefined voltage levels; estimated switching activities between the operations; a set of functional units (resource constraints); and a latency constraint.

Objective. Assign voltage levels to all the operations and bind these operations to the set of functional units so that the total number of operations driven by low-Vdd levels is maximized under the resource and latency constraints with minimized total switching activity.

We assume that the initial scheduling result of the input design fulfills latency and resource constraints. During voltage assignment and binding, we do not perform rescheduling of the operations. Therefore, the objective is to carry out voltage assignment and functional unit binding in such a way so that these constraints are still honored while minimizing power. In this section, our main focus is to present an optimal algorithm to achieve our objective. We also apply power gating as a post-processing procedure and examine its effectiveness on leakage power reduction. In the next, Section 4.2 presents some definitions and problem reduction. Section 4.3 presents a network flow formulation to solve the *optVF* problem for the dual-Vdd case. Section 4.4 extends our optimal solution into the multiple-Vdd case. Section 4.5 presents a simple power gating approach.

4.2 Definitions and Problem Reduction

Given a data flow graph (DFG), $G = (V, A)$, set V corresponds to operations and set A corresponds to data flowing between operations. An edge $a = (x, y)|x, y \in V, a \in A$ indicates there is a data dependency between operations x and y. Scheduling assigns operations to control steps so that the overall execution latency meets a certain time constraint, and the number of resources used

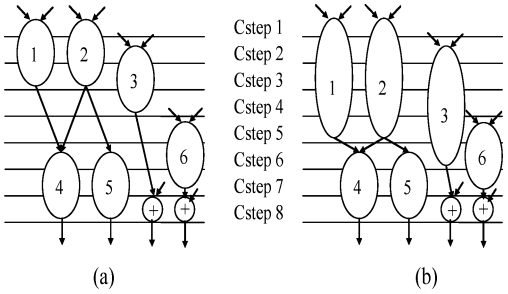


Fig. 2. Example of extendable operations.

also meets a certain resource constraint. After scheduling, the lifetime of each operation in the DFG is the time during which the operation is active, defined as an interval [*startTime*, *endTime*]. A *comparability graph* $G_c = (V_c, A_c)$ for these operations can then be constructed for addition and multiplication separately. V_c corresponds to all the operations of the same type, and there is a directed edge $a_c = (v_i, v_j)|a_c \in A_c$ between two vertices if and only if their corresponding lifetimes do not overlap, and operation v_i comes before v_j. In such a case, we call operations v_i and v_j *comparable* with each other, and they can be bound into a single FU without lifetime conflicts. Let s_{ij} denote the weight of edge a_c, which represents the cost when we bind v_i and v_j into the same FU. This cost is the switching activity between these two operations when v_j executes right after v_i on the FU, which is estimated by equation (5) in Section 3.

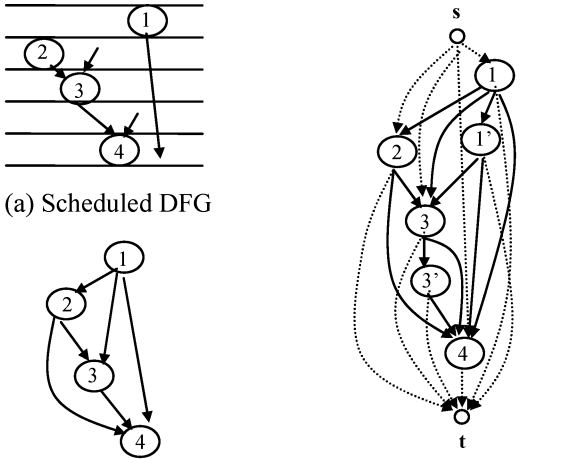
We first examine the dual-Vdd case. We show our problem formulation and solution, and prove its optimality. We then extend our formulation into multiple Vdds. First of all, we call our high Vdd VddH, and our low Vdd VddL. In addition, we introduce two definitions. An operation O is *extendable* if O can be assigned to VddL, and the extended execution delay of O will not violate the overall latency constraint, and in the same time, the data dependencies between O and other operations are still valid. In other words, O will still generate its data in time so that the data can flow to all the other operations that require it. If O is assigned VddL in the final solution, we say O is *extended*. Its startTime stays the same as before but its endTime is increased. Due to the resource constraint, not all extendable operations can be extended eventually. Figure 2 shows an example. Figure 2(a) shows a scheduled DFG with 6 multiplications and 2 additions. The *Exe Cycle* is 3 cycles for VddH and 5 cycles for VddL for the multiplication. Latency constraint is 8 control steps, and the number of available multipliers is 3. We will examine multiplication nodes. Node 6 is not extendable because of the data dependency. Nodes 4 and 5 are not extendable due to the latency constraint. Nodes 1, 2 and 3 are extendable, which are shown in Figure 2(b). However, only two can be extended to meet the resource constraint. If operations 1 and 3, or 2 and 3 are chosen to be extended, although resource constraint is fulfilled for control step 5, it will be violated in step 6 because node 3 is no longer comparable with nodes 4, 5 and 6 after its extension (their lifetimes overlap at control step 6). Therefore, we need an efficient way to assign VddL to as many operations as possible within the

constraints. Suppose M_e is the maximum number possible of extended operations given resource and latency constraints, and the total number of extendable operations is T_e, we have $M_e \leq T_e$ for a design. It is easy to see that there may be different sets of M_e operations and each of such sets fulfills the constraints. Which set of M_e operations to extend will influence power reduction because different extensions will change the original G_c into a different new comparability graph since the lifetimes of the M_e operations in G_c have changed. Let G'_c denote the new comparability graph due to M_e extensions. G'_c has the same node set V_c but a different A_c. Notice that although we process multiplications and additions separately, the optimality of our solution is not changed by this separation. This is because that we simulate our switching activities on the whole design and we honor the data dependencies of the whole design when we extend nodes. We have to bind additions and multiplications separately because an addition cannot be bound with a multiplication.

Given a comparability graph $G_c = (V_c, A_c)$, our objective for solving the **optVF** problem becomes the following two related optimization goals: (1) find a node subset $V_L \subset V_c$ and $|V_L| = M_e$ so the extensions of V_L nodes will give the best new comparability graph G_B among all the G'_c graphs in terms of power reduction and meet the constraints; (2) find an edge subset in G_B that covers all the vertices in V_c in such a way that the sum of the edge weights in the subset is the minimum, and all the vertices can be bound into no more than k FUs. The first goal is voltage assignment, and the second goal is FU binding for reducing switching activity. We can see these two goals are intertwined because we cannot achieve the first goal without achieving the second goal or vice-versa. The second goal of the objective can be formulated as a traditional clique partitioning problem. Each clique corresponds to the operations that are to be bound into a single FU. Although clique partitioning problem is NP-hard for general graphs, it is shown that we can find the minimum number of cliques required to bind all the nodes in polynomial time when working with comparability graphs [De Micheli 1994]. In our work, k is the minimum number of FUs required. Early works proposed optimal solutions to compute maximum *k-covering* in weighted transitive graphs [Sarrafzadeh and Lou 1993] and maximum weighted *k-cofamily* in partially ordered sets [Cong and Liu 1991] through network flow formulations. Both works found various applications across many optimization fields. Comparability graphs belong to transitive graphs [De Micheli 1994] and can also be represented using partially ordered sets [Chen and Cong 2004a]. Therefore, there are previous works that used network formulation to solve various binding problems on comparability graphs. In the next section, we will discuss more details of these early works, and then present our simultaneous voltage and functional unit binding solution by computing the min-cost k-flow in a flow network.

4.3 Network Flow Formulation for the Dual-Vdd Case

Various binding algorithms have been proposed previously for reducing circuit power through network flow formulation. In Chang and Pedram [1995], an optimal low-power register binding algorithm to reduce total switching activity



(a) Scheduled DFG

(b) Comparability graph

(c) N_G with two Vdds

Fig. 3. An example showing the formulation accommodating two Vdds.

was presented. However, it did not guarantee using the minimum number of k resources during the binding process. In other words, its network-flow solution might not cover all the nodes with k resources in the comparability graph. In Chang and Pedram [1996], the same authors formulated functional unit binding as a multi-commodity flow problem to reduce switching activity. The inter-frame binding constraints made the problem hard (to be discussed later). In Chen and Cong [2004a], a register binding algorithm was presented to reduce total MUX connections in the design by computing the min-weighted k-cofamilies. It showed consistent positive impact on area, delay and power optimizations due to reduced interconnect usage. In Lyuh and Kim [2003], a single-commodity network flow was used to solve the bus binding problem with improved run time. It then presented a heuristic to fulfill the inter-frame binding constraints and showed promising results. None of these works considered dual Vdds in their formulations. In this work, we will build voltage assignment into our formulation and show that we can assign the maximum number of operations to VddL under latency and resource constraints and achieve min-power functional unit binding simultaneously. We always guarantee that we use no more than k resources.

A network $N_G = (s, t, V_n, E_n, C, K)$ is constructed based on the comparability graph $G_c = (V_c, A_c)$. This is an extension to the one used in Chang and Pedram [1995], and we will introduce extra vertices to provide voltage assignment consideration. First, there are source vertex s and sink vertex t. The additional edges are added from s to every vertex in V_c, and from every vertex in V_c to t. Second, for each extendable vertex v in V_c, there is an extra node v' connecting to v. There are additional edges between v' to the vertices comparable with it (these vertices are still comparable to node v after v is extended), and an additional edge between v' to t. N_G has the cost function C and the capacity K defined on each edge in E_n. Figure 3 shows an example. Figure 3(a)

is a simple scheduled DFG with all additions. Figure 3(b) is the corresponding comparability graph. Figure 3(c) is the graph N_G for Figure 3(b). Here an extended node will take 2 cycles. The edges connecting to the source or the sink vertices use dashed lines to differentiate them from other edges. Notice node $1'$ is only connected to node 3 and 4 because node 1 is no longer comparable with node 2 after its extension.

Let V_e denote the set of all the extendable nodes in V_c. We have $V_e \subset V_c$. We use the symbol $\rightarrow$ to represent that two vertices are comparable with each other. Formally, the network $N_G = (s, t, V_n, E_n, C, K)$ is defined as the following:

$$
\begin{aligned}
&V_n = V_c \cup \{s, t\} \cup \{v' | v \in V_e\} \\
&E_n = A_c \cup \{(s, v), (v, t) | v \in V_c\} \cup \{(v, v'), (v', t) | v \in V_e\} \\
&\qquad \cup \{(v_i', v_j) | v_i' \rightarrow v_j; i \neq j; v_i \in V_e; v_j \in V_c\} \\
&C(s, v) = 0 | v \in V_c \\
&C(v, t) = 0 | v \in V_c \\
&C(v', t) = 0 | v \in V_e \\
&C(v_i, v_j) = -L \times (1 - s_{ij}) | v_i \rightarrow v_j; i \neq j; v_i, v_j \in V_c \\
&C(v_i', v_j) = -L \times (1 - s_{ij}) | v_i' \rightarrow v_j; i \neq j; v_i \in V_e; v_j \in V_c \\
&C(v, v') = -T | v \in V_e \\
&K(e_n | e_n \in E_n) = 1,
\end{aligned}
$$

where C is the cost assigned on the edges and K is the capacity on the edges. s_{ij} is the switching activity on the edge (v_i, v_j). L is a positive constant and is set to 100. L is used to scale the costs into integer numbers. To maximize the number of extended operations, we need to guarantee that $C(v_i, v_i') + C(v_i', v_j) < C(v_i, v_j)$. That is the reason that $C(v, v')$ is set as $-T$, where $T = L \times |V_c|$. Value T guarantees that v will be extended if it is the only extendable node within resource constraint as an extreme case, no matter what the values of $C(v_i, v_j)$ are for the edges (to be shown later). Notice $s_{ij} < 1$ always. Therefore, we set the cost $C(v_i, v_j)$ as a negative value. The smaller s_{ij} is, the smaller $C(v_i, v_j)$ will be. Notice N_G captures all the possible configurations of G_c'.

Our algorithm uses the min-cost flow solution in the network to generate the voltage and module assignments. It is necessary to allow only a unit flow to go through each node $v \in V_c$. To guarantee this, we apply a node-splitting technique, which is similar to that used in Chang and Pedram [1995]. We duplicate every vertex $v \in V_c$ in N_G into another node v^d. There is an edge from v to v^d. If there is an edge $(v_i, v_j) \in A_c$, there is an edge (v_i^d, v_j) in the new network, named N_G^d. $C(v_i^d, v_j)$ is the same as $C(v_i, v_j)$. The original edge (v_i, v_j) is removed from N_G^d. Meanwhile, node v' will be connected to v^d instead of v. All the edges are assigned with a capacity of 1. In addition, we assign cost $C(v, v^d) = -X$, where X is a positive constant and $X \geq 2T$. We can show that this cost assignment will guarantee that all the nodes in V_c will be covered when the min-cost flow in N_G^d generates the binding and voltage assignment solution. Figure 4 shows an example.

LEMMA 1. *A flow f, with $|f| = 1$, in the network N_G corresponds to a clique χ in the original comparability graph G_c with voltage assignment. An edge (v_i, v_j) in the flow indicates operations v_i and v_j will be bound into the same*

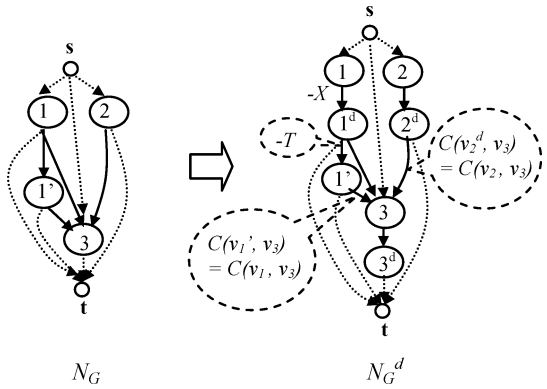


Fig. 4. A simple N_G and its split graph N_G^d.

FU W. An edge (v, v') in the flow indicates operation v will be assigned to VddL when executing in W.

PROOF. A unit flow from source to sink represents a sequence of operations that are comparable with one another. Therefore, they form a clique and can be bound into the same FU. When an edge (v, v') is in the flow, v will be assigned to VddL. This is true by the construction of the network N_G. □

LEMMA 2. *A flow f, with $|f| = k$(ak−flow), that passes through every node $v \in V_c$ by a unit flow is equivalent to finding k disjoint paths (or chains) in N_G, thus generating k cliques in G_c covering all the operational nodes, which is a legal binding solution with voltage assignments.*

PROOF. Since every node only allows a unit flow to pass, the flow with value k will generate k disjoint paths in N_G (except nodes s and t). Each path represents one group of operations that are comparable with one another. By Lemma 1, each path corresponds to one clique χ in the original compatibility graph G_c with voltage assignment. k disjoint paths correspond to a partition of the graph G_c into k cliques. If the k flow passes all the nodes in V_c, the resulted k cliques will cover all the nodes in V_c as well. Thus, a legal binding solution is generated where each clique can be bound into a separate FU with voltage assignments. □

LEMMA 3. *Due to cost assignments, the following results hold:*

(1) *Given any legal binding solution, let S be the total sum of costs from $C(v_i^d, v_j)$ $(i \neq j)$ in the solution, we will have $-S < T$.*
(2) *If three nodes are comparable with one another, for example, $v_1 \rightarrow v_2 \rightarrow v_3$, the cost of binding v_1, v_2, and v_3 together into one FU is always smaller than just binding $v_1$and v_3 together even when v_1 is extendable.*

PROOF

(1) A legal binding solution is equivalent to forming k disjoint chains. Suppose $|V_c| = n$. It means that there will be $(n-k)$ edges (v_i^d, v_j) or $(v_i', v_j)|v_i, v_j \in V_c$

$(i \neq j)$ to form these k chains in N_G^d (suppose a chain contains x edges, it will contain $x+1$ vertices). For any $C(v_i^d, v_j)$, we have $-C(v_i^d, v_j) \leq L$. The total cost on these edges is S. Therefore, we have $S = (n-k) \times C(v_i^d, v_j)$. Thus, $-S \leq (n-k) \times L < n \times L = T$.

(2) We have $C(v, v^d) = -X$, where $X \geq 2T$. If v_1 is not extendable, the cost of binding three variables together $C(v_1, v_1^d) + C(v_1^d, v_2) + C(v_2, v_2^d) + C(v_2^d, v_3) + C(v_3, v_3^d)$ is smaller than the cost of binding v_1 and v_3 together, which is $C(v_1, v_1^d) + C(v_1^d, v_3) + C(v_3, v_3^d)$. If v_1 is extendable, we still have $C(v_1, v_1^d) + C(v_1^d, v_2) + C(v_2, v_2^d) + C(v_2^d, v_3) + C(v_3, v_3^d) < C(v_1, v_1^d) + C(v_1^d, v_1') + C(v_1', v_3) + C(v_3, v_3^d)$. □

THEOREM 1. *The min-cost flow f, with $|f| = k$ (min-cost k-flow) on the network N_G^d gives the largest number of extended operations in the design with the minimum total switching activity on k functional units for the circuit represented by G_c under the dual-Vdd framework.*

PROOF. We first introduce some observations using G_c and N_G. By Lemma 2, we know that we need k cliques (or disjoint chains) covering all the nodes in G_c to form the binding solution. First of all, this is possible due to Dilworth's theorem[5] [Dilworth 1950] because the comparability relation on V_c nodes makes V_c a partially ordered set and the subset of V_c, containing the largest number of mutually noncomparable nodes, has cardinality k. Suppose $|V_c| = n$. It means that there will be $(n-k)$ edges $(v_i, v_j)|v_i, v_j \in V_c$ $(i \neq j)$ to form these k disjoint chains, which can be found by a k-flow in N_G(k different unit-flows). Let us denote these $(n-k)$ edges as set E_c. Different k-flow solutions will give different E_c but $|E_c| = n-k$ always.[6] In addition, let M_e be the maximum number of nodes that can be extended without violating the constraints. After M_e nodes are extended, there are still k disjoint chains from N_G and corresponding E_c edges (containing less (v_i, v_j) edges and at most M_e (v_i', v_j) edges $|v_i, v_j \in V_c$) on these k chains. The additional edges on the k-flow are M_e VddL extension edges, $(v, v')|v \in V_c$. We first show that our solution will cover all the V_c nodes through disjoint k-chains, and then we show that our solution is optimal.

The min-cost k-flow from N_G^d will cover all the nodes in V_c by k disjoint chains. N_G^d is generated by splitting each node $v \in V_c$ in N_G. First we will have k disjoint chains because we have a k-flow and each $v \in V_c$ only allows one unit flow to pass due to the unit capacity assigned for the edge (v, v^d) after splitting v. Next, we can show that if a k-flow does not cover all the nodes it will not be the min-cost k-flow. Suppose node $v_x \in V_c$ is not covered in current flow solution, and $|E_{c1}| = n-k-1$. There will be another feasible k-flow that covers all the

[5]This theorem indicates that a partially ordered set P can be partitioned into k-disjoint chains covering all the elements if P contains at least one subset Y, where $|Y| = k$; every pair of elements in Y are non-comparable with each other; and k is the largest number for such kind of subsets in P. Please refer to Chen and Cong [2004a] for the definition of partially ordered sets.

[6]This is true when every node is at least comparable with one other node in the graph. The proof still holds when there are nodes that are not comparable with any other nodes (their lifetimes conflict with all the other nodes). Then, each of these nodes just occupies its own FU in the binding solution.

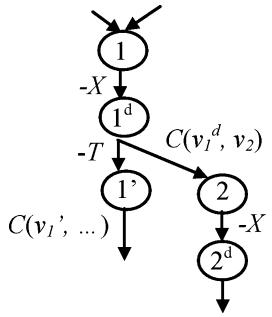
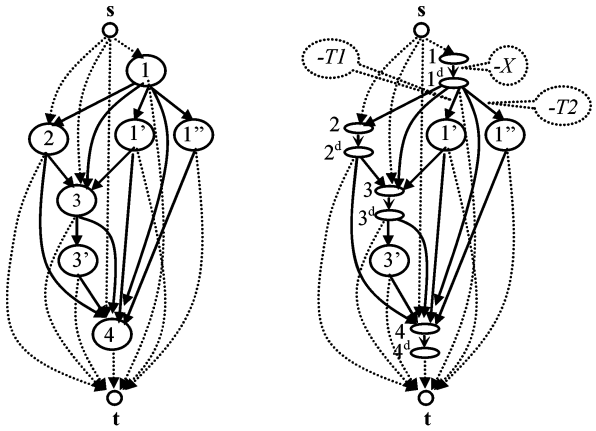


Fig. 5. An example showing that node covering has higher priority than VddL extension.

nodes including v_x. The cost of the new flow will be smaller than before because $-X$ is added to the current cost by covering v_x. This cost reduction surpasses any possible cost increases on the new $(n-k)$ edges if these edges have more total cost than the old E_{c1} edges. This is because $X \geq 2T = 2 \times L \times |V_c| > L \times n > -C(v_i^d, v_j) \times (n-k)$ (Lemma 3). Thus, the old flow is not the min-cost k-flow. Notice that $X \geq 2T$ guarantees that covering all V_c nodes has higher priority than VddL extensions. Figure 5 shows an example. If node 1 is extended, it cannot be bound with node 2 anymore due to lifetime conflict. In such a case, binding node 1 and 2 together takes priority than the extension of node 1. This guarantees that the flow will cover all the nodes first to fulfill the resource constraint before node extensions. Lemma 3 (result 2) addresses this precisely.

The min-cost k-flow will extend M_e nodes, which is the maximum ever possible within the resource constraint, and return the minimum total switching activity thereafter. As we show before, we still have a feasible solution by having M_e nodes extended, that is, all the V_c nodes are still covered through $(n-k)$ number of E_c edges. We can show that if just $M_e - 1$ nodes are extended, it will not be the min-cost k-flow following a similar argument as used before. Suppose there are $(M_e - 1)$ nodes extended. The total cost on the E_c edges reflects the total amount of switching activity. Now, we can extend one more node and still have a feasible k-flow. After this extension, the cost on new E_c edges can at most increase by $-S$, where $-S < T$ (Lemma 3). Thus, the total cost now will be smaller due to the new extension. Therefore, the min-cost k-flow has to extend M_e nodes. Given this is true, the min-cost k-flow indeed returns a set E_c with the minimum total cost on the E_c edges, and thus provides the optimal solution. □

Theorem 1 is optimal in the sense that it will always find the best set of M_e nodes (also the largest possible) to extend, and achieve the minimum switching activity for binding together with these low-Vdd extensions simultaneously. Notice that this theorem holds when we ignore the inter-frame constraints presented in Chang and Pedram [1996], which capture the switching activity in the cyclic executions of the DFG, that is, the switching activity when a new set of vectors arrives on the inputs of the FUs to start execution from the beginning of the DFG again (represented by Eq. (7) in Section 3). However, we count these switches in our power estimation to make our experimental results more accurate (Eq. (8) in Section 3). Our formulation can be easily extended to



(a) N_G with three Vdds (b) $N_G{}^d$

Fig. 6. An example showing the formulation accommodating three Vdds.

consider inter-frame constraints by building a multicommodity flow network as shown in Chang and Pedram [1996]. The min-cost multicommodity flow solution will provide the largest extended-operation number and the minimum switching activity with interframe constraints. Since our goal is to show that we can achieve optimality under multi-Vdd consideration, multicommodity flow is not the focus of this work. We do plan to add this extension in the future.

Our task then becomes finding the min-cost k-flow in the network N_G^d. It can be obtained through capacity scaling and successive shortest path computation and has running complexity $O(|E| \log k\ (|E|+|V|\log|V|))$. After we obtain the min-cost k-flow, each edge with a unit flow in N_G^d, (v_i^d, v_j), represents that operations v_i and v_j should be bound together into the same FU and $v_i$is operating under VddH. Each edge (v_i', v_j) represents v_i and v_j should be bound together and v_i is operating under VddL. If a flow passes $s \Rightarrow v \Rightarrow v^d \Rightarrow [v'] \Rightarrow t$, it represents that v is occupying a single FU just by itself. It operates either under VddH or VddL (when v' exists).

4.4 Extension for Multiple Vdds

In this section, we show how to build more Vdds into our network flow formulation and still achieve optimal solution. We will use three Vdds as an example but the same principle applies to more numbers of Vdds. We call our high Vdd VddH, and our low Vdds VddL_1 and VddL_2. We have $\text{VddH} > \text{VddL}_1 > \text{VddL}_2$. To support a second low Vdd, we can use new v'' nodes connecting to v nodes in N_G. v''nodes will be similarly processed as v' nodes as in the dual-Vdd case, and their associated costs can be designed and assigned. The min-cost flow will decide either picking v' or v'' nodes in its solution. Figure 6(a) shows the graph N_G with $\text{VddH} = 1.3\text{v}$, $\text{VddL}_1 = 0.8\text{v}$, and $\text{VddL}_2 = 0.5\text{v}$ for the comparability graph shown in Figure 3(b). The *exe cycles* for the operations driven by these voltages are 1, 2, and 4 respectively (Table I). Figure 6(b) shows the corresponding N_G^d for this example.

As shown in Figure 6(b), the cost for edge $v^d \rightarrow v'$, $C(v^d, v') = -T_1$, and the cost for edge $v^d \rightarrow v''$, $C(v^d, v'') = -T_2.T_1$ is equal to $L \times |V_c|$ as in the dual-Vdd case. $T_2 = T_1 \times (VddL_1^2/VddL_2^2) = 2.56T_1$ for the voltage levels we use in this example. Therefore, when an operation is executing under VddL_2, its dynamic power will be reduced by 2.56X compared to the case where it is executing under VddL_1 due to these two different voltage scaling. To guarantee that the solution will still cover all the operation nodes, we set $X \geq 2T_2$. All the other costs and capacities are similarly assigned as in the dual-Vdd case. After we obtain the min-cost k-flow, each edge with a unit flow in N_G^d, (v_i^d, v_j), represents that operations v_i and v_j should be bound together into the same FU and v_i is operating under VddH. Each edge (v_i', v_j) or (v_i'', v_j) represents v_i and v_j should be bound together and v_i is operating under VddL_1 or VddL_2 respectively. If we have a series of low Vdd values such as $\mathrm{VddL}_1 > \mathrm{VddL}_2 > \ldots > \mathrm{VddL}_{n-1} > \mathrm{VddL}_n$, we will define a series of corresponding T values so that they are in the following relationship:

$$
\begin{aligned}
&T_1 = L \times |V_c| \\
&T_2 = T_1 \times \left(VddL_1^2/VddL_2^2\right) \\
&\cdots \\
&T_{n-1} = T_{n-2} \times \left(VddL_{n-2}^2/VddL_{n-1}^2\right) \\
&T_n = T_{n-1} \times \left(VddL_{n-1}^2/VddL_n^2\right)
\end{aligned}
$$

Then, we set $X \geq 2T_n$.We then build our network N_G^d by adopting n different v'-type nodes, such as v' and v'' nodes in the three-Vdd case. We connect these v'-type nodes to v^d as long as the delay extensions of these v'-type nodes do not violate data dependency and the latency constraint, that is, they are extendable. We then assign the T values to the edges of v^d to v'-type nodes respectively as we do for the three-Vdd case. We have the following theorem:

THEOREM 2. *Given a set of voltage levels and the power and delay values for the resources driven by these voltages, the min-cost k-flow f on the network N_G^d gives the largest total number of extended operations guided by voltage scaling and a functional unit binding solution with the minimum total switching activity on k functional units.*

PROOF. Simple extension of Theorem 1. □

This theorem guarantees that our algorithm is able to search the combined solution space of different voltage assignments and functional unit bindings and find an optimal solution. It will get the largest total number of extended operations with different voltage levels to achieve the maximum power reduction through voltage scaling and simultaneously minimize the total switching activity of the design to reduce dynamic power.

4.5 Power Gating

We follow a simple power gating scheme. After we obtain the binding solution, we search through the operations bound in each FU and find whether the FU is idle for a certain period of time (*idle_cycle*) that is longer than *SleepCycle*

(Section 3) between two consecutive operations. If this is the case, we count the static power saved during the number of cycles $= idle\_cycle - SleepCycle$. This simple scheme is used because our main goal in this work is to reduce dynamic power. If static power reduction is the main goal, we can modify our network flow formulation so the cost on an edge represents the idle cycles between the two operations on the edge. We expect that the max-cost flow solution from the network can dramatically increase the total idle time spent by functional units.

5. EXPERIMENTAL RESULTS

Our experimental results include two major parts. We first show improved results of our simultaneous voltage assignment and binding algorithm compared to a heuristic that separates voltage assignment from binding. We then examine the power saving potentials of multi-Vdd over single-Vdd architectures and study the impact of different voltage levels and their combinations on power and energy reduction. To obtain an initial scheduling result that is suitable for voltage assignment, we adopt a heuristic algorithm from Lin et al. [1997] to perform the resource- and time-constrained scheduling to maximize the number of extended operations. The main idea in Lin et al. [1997] is to iteratively make an operation extended, and then use a list scheduling algorithm to validate the choice. The choice is reversed if the extension violates constraints. This heuristic will generate voltage assignment along the way. Although dramatic increases of extended operations are observed, this algorithm does not guarantee to extend the optimal number of operations for the schedule it produces.

5.1 Optimality Study

We will use dual-Vdd case to show the advantages of our algorithm. Since there is no previous algorithm that combines voltage assignment and switching activity reduction simultaneously, we will compare our algorithm, named *opti-mvdd*, with an experimental flow *sep-flow* set up by ourselves. *sep-flow* has two stages. First, it obtains the initial voltage assignment from the scheduling result as done in Lin et al. [1997]. All the nodes with VddL assignment will be extended and a corresponding new comparability graph is built. Second, we minimize the switching activity on the new comparability graph as if we are working for the single-Vdd case. We use the binding algorithm presented in Chang and Pedram [1995] for this stage because the algorithm gives an optimal binding solution to reduce switching activity without considering inter-frame constraints. However, its resource usage may exceed the minimum required number k. For *opti-mvdd*, we use the same schedule but ignore all the voltage assignments because *opti-mvdd* will generate the optimal voltage assignment and binding simultaneously. We use VddH as 1.3v and VddL as 0.8v in this experiment.[7] To simulate the DFG for switching activity estimation on the edges, we use 1000 consecutive random input vectors.

[7]These two values form the best combination in works Chen and Cong [2004b] and Chen et al. [2004], which falls into the optimal VddL/VddH ratio range as indicated in Hamada et al. [1998]. The optimal ratio should be in the range of 0.6–0.7.

Table II. Experimental Results of Our Algorithm *opti-mvdd* (with Two Vdds: 1.3v and 0.8v) vs. a Heuristic Algorithm *sep-flow*

Bench Marks	Total Nodes	Ext'able Nodes	*sep-flow* Extended	*opti-mvdd* Extended	*sep-flow* Power (W)	*opti-mvdd* Power (W)	*opti-mvdd vs. sep-flow*
air	422	211	79	89	4.4445	3.5015	−21.2%
chem	342	76	36	39	4.2972	3.896	−9.3%
dir	127	32	23	23	2.0245	1.743	−13.9%
honda	107	25	19	19	2.2949	1.9036	−17.1%
lee	49	15	15	15	0.7596	0.5861	−22.8%
mcm	94	10	8	8	2.7628	2.7565	−0.2%
pr	42	7	6	6	1.4163	1.4027	−1.0%
u5ml	565	183	119	124	3.3751	2.9798	−11.7%
wang	48	8	4	6	1.4583	1.3226	−9.3%
Ave.							**−11.8%**

We carry out experiments based on a set of real-life benchmarks from Srivastava and Potkonjak [1995], including several different DCT algorithms, such as *pr, wang, lee*, and *dir*, and several DSP programs, such as *mcm, honda, chem,* and *u5ml*. Both *opti-mvdd* and *sep-flow* have the power gating feature. The initial scheduling uses the tightest latency and resource bounds. Table II shows the results. We observe that the number of extendable nodes in the design usually is larger than the number of extended nodes. *opti-mvdd* always produces larger or equal number of extended operations than *sep-flow* does. The power values of *opti-mvdd* are consistently better than those of *sep-flow* (11.8% better on average). This is due to two reasons: (1) the initial voltage assignment of *sep-flow* is not optimal. Even for the cases where it extends the maximum number of operations, its choices may not be good because there is no switching activity considered; (2) binding of *sep-flow* sometimes exceeds the resources required. For example, *sep-flow* uses one more multiplier than *opti-mvdd* does for design *lee*.

5.2 Impact of Multi-Vdd on Power and Energy Consumption

To examine how multi-Vdd architecture itself helps on power/energy reduction and gain some insights on power/energy-latency trade-offs, we carry out a series of experiments to compare *opti-mvdd* with an algorithm *opti-hvdd*. *opti-hvdd* only considers the single high Vdd. It uses the same network formulation as presented in Section 4, but without extendable nodes (the v'-type nodes). The nodes in V_c are still split with cost assignment $C(v, v^d) = -X$. It will provide an optimal solution to minimize switching activity within the resource constraint for the single-Vdd case. To examine different trade-off scenarios, we change our initial scheduling to work with different latency bounds. The relaxed latency will be $(1+\alpha)$*$CriticalPath$, where α is the relaxation percentage, and *CriticalPath* is the minimum number of clock cycles a scheduled DFG needs without any relaxation, that is, its smallest critical path length.[8] For example, suppose *CriticalPath* is 10 cycles for a design, $\alpha = 0.5$ will relax the latency of the

[8]Scheduling with the tightest latency may require a large number of resources. Therefore, latency relaxation is a common practice.

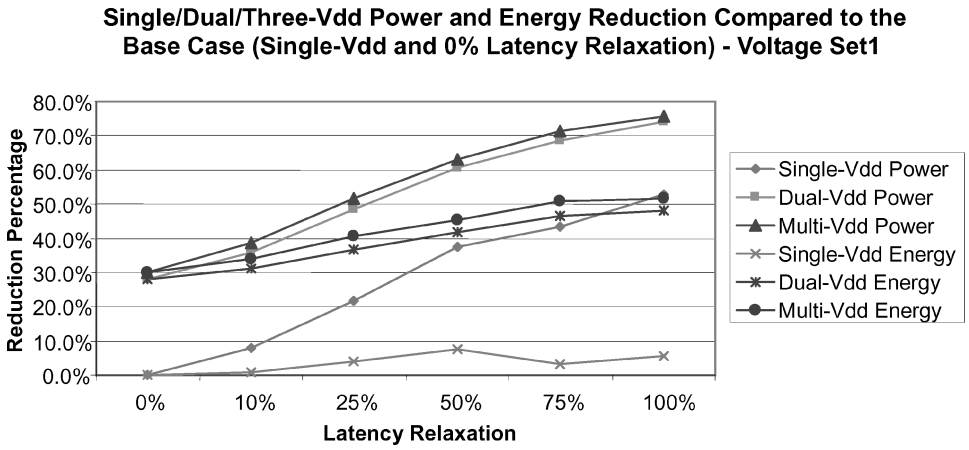


Fig. 7. Power and energy reduction results comparing to the base case of *opti-hvdd*; single-Vdd is 1.3v; dual-Vdd is 1.3v/0.8v; and three-Vdd is 1.3v/0.8v/0.5v.

design to 15 cycles. We still use the heuristic scheduling algorithm from Lin et al. [1997]. The scheduling algorithm will take the new latency constraint and generate the schedule accordingly.

For practical reasons, the largest number of voltage combinations in our experiment includes three Vdds. We first study the following voltage combination (Voltage Set1): VddH = 1.3v, VddL_1 = 0.8v, and VddL_2 = 0.5v. The value of VddH/VddL_1 is almost equal to the value of VddL_1/VddL_2 for this set of voltages. Figure 7 collects the results for single-Vdd, dual-Vdd and three-Vdd configurations. The value of α is shown on the x-coordinate. The power and energy reduction percentages are average values over the benchmarks. We use the power and energy values of the *single-Vdd + no-latency-relaxation* as the comparison base and show the reduction percentages of other configurations over this base case. We first observe that we can achieve power and energy reduction of 28.1% over the base case just by doing dual-Vdd when there is no latency relaxation. The largest power reduction for dual-Vdd is 74% when latency is relaxed by 2X (100%). On the other hand, the energy reduction is 48% for the same 2X relaxation. The percentage is smaller compared to that of power reduction because of the increased computation latency. The power curve shows that dual-Vdd can provide larger power savings compared to trivial techniques, such as frequency scaling. For example, if the frequency of the design is slowed down by 50% for the single high-Vdd case, that is, the delay of each clock cycle becomes 13ns now, and the overall computation latency is also relaxed by 2X as a result. However, its power reduction is bounded above by 50%. Actual number will be determined by the percentage of the dynamic power in the total power consumption. For our adders and multipliers, this bound becomes 42%, which is much smaller than 74% as shown in Figure 7. Next, we observe that three-Vdd actually does not provide much power or energy gain for this set of voltages. Figure 8 provides some hints why this is the case, where the distributions of voltages to the operations are shown for every relaxation point. The numbers

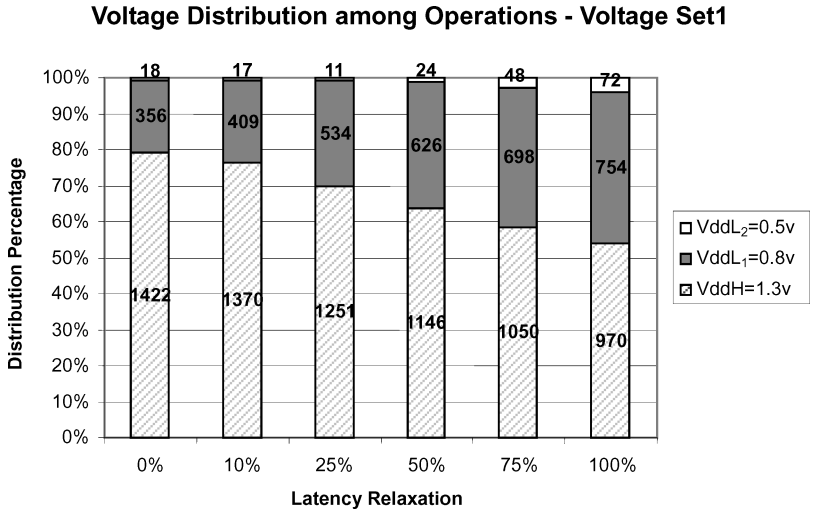


Fig. 8. Node numbers with different voltage assignments for Voltage Set1.

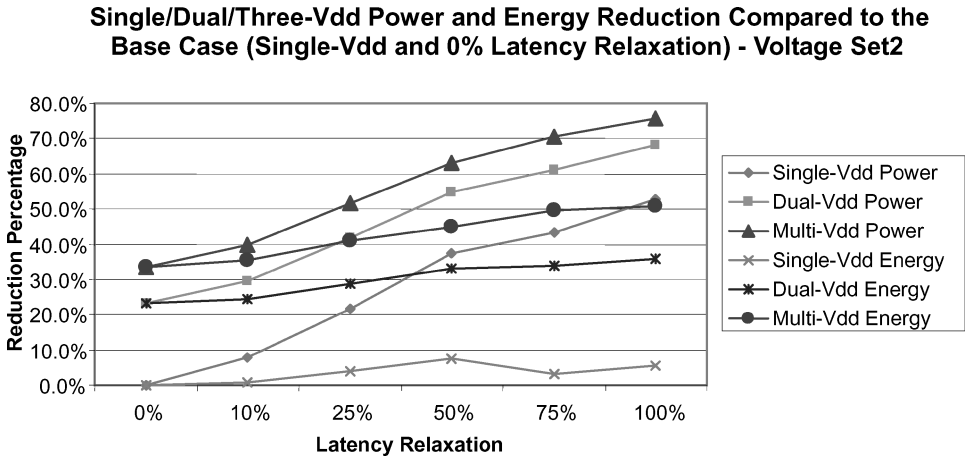


Fig. 9. Power and energy reduction results comparing to the base case of *opti-hvdd*; single-Vdd is 1.3v; dual-Vdd is 1.3v/1.0v; and three-Vdd is 1.3v/1.0v/0.7v.

on the bars indicate the number of operations assigned with the particular voltages. The total number of operations is all the same for the different relaxation points. The numbers are contributed from all the benchmarks.

Figure 8 shows that only a few of operations are able to execute under 0.5v. This is because the execution time of 0.5v is much longer, especially for multipliers (9 cycles). As a result, not many operations can take advantage of this low voltage setting especially when the latency constraint is tight. With this observation, we try another voltage combination (Voltage Set2): VddH = 1.3v, $\text{VddL}_1 = 1.0$v, and $\text{VddL}_2 = 0.7$v. Figure 9 shows the results. We have two observations from Figure 9. First, the dual-Vdd case offers smaller power savings compared to the dual-Vdd case in Figure 7, mainly because that the low Vdd of

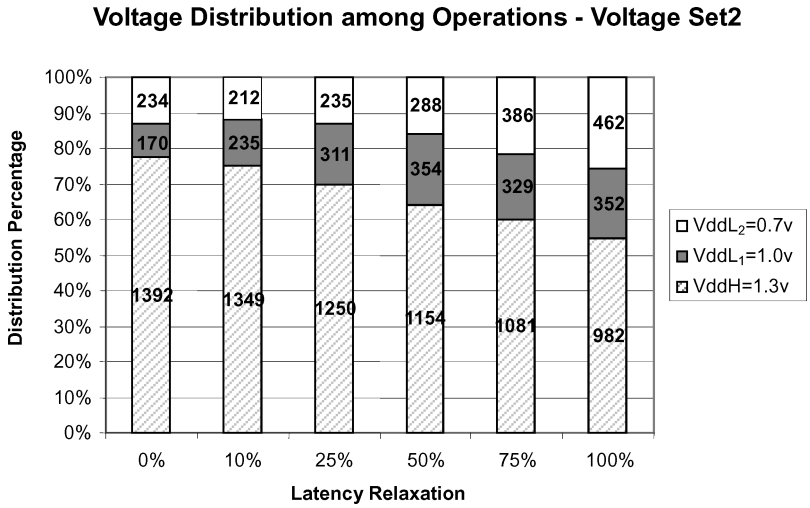


Fig. 10. Node numbers with different voltage assignments for Voltage Set2.

1.0v does not reduce as much power as 0.8v does. Second, three-Vdd case offers larger power savings compared to the three-Vdd case in Figure 7 when latency relaxation percentages are small. For larger relaxation percentages, both settings offer similar power reductions. Figure 10 shows some explanations. We can observe that our solution extends a good number of 0.7v operations because it saves 2.04X more power than 1.0v operations do. Furthermore, the total number of extended nodes (top two portions in each bar) is larger than the corresponding number shown in Figure 8 especially for low relaxation points. This is because that the low-Vdd operations in Voltage Set2 use less execution cycles than those in Voltage Set1 do, and this aspect shows stronger impact when the latency is tight, thus enables more low-Vdd extensions to save more power. This indicates that different voltage configurations can be selected for different power and latency requirements.

The direct reason that latency relaxation can help on power and energy is that the maximum number of extended operations increases significantly along the relaxation so more operations can be executed with low Vdds. Both Figure 8 and Figure 10 show this trend. Another reason is that the number of resources required becomes smaller due to latency relaxation because more operations can share common resources when the schedule becomes longer. This is obvious from the single-Vdd curve in Figure 7 and Figure 9. To find out how much savings are due to the multi-Vdd scheme itself, we compare the power consumptions between *opti-mvdd* and *opti-hvdd* for each individual relaxation point.[9] Figure 11 and Figure 12 show the results for Voltage Set1 and Set2. We can observe that multi-Vdd alone saves power significantly especially when relaxation is larger. Energy comparison shows the same trends.

[9]Both *opti-mvdd* and *opti-hvdd* use the same resource numbers for each relaxation point, and compare their power consumptions based on that relaxation point. This is different from the data in Figures 7 and 9, where everything is comparing to a base point.

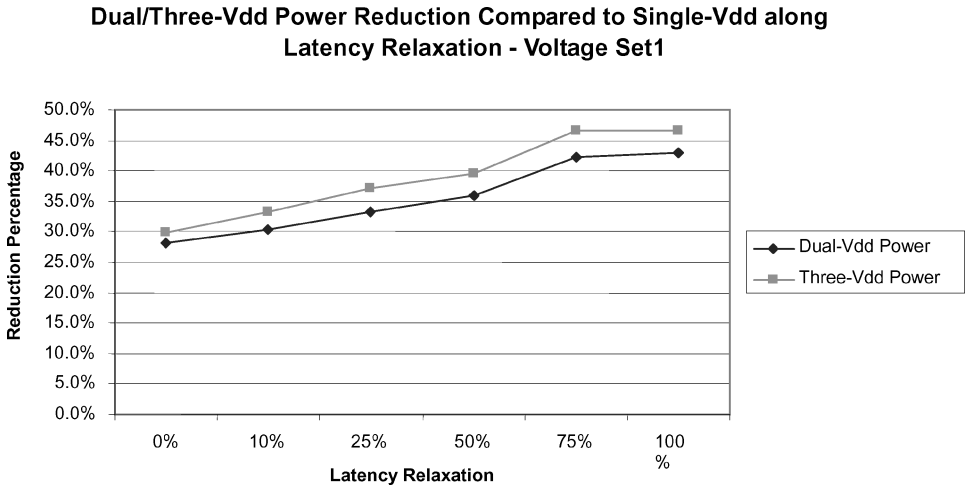


Fig. 11. Power reduction results comparing *opti-mvdd* to *opti-hvdd* for each latency relaxation point respectively; dual-Vdd is 1.3v/0.8v, and three-Vdd is 1.3v/0.8v/0.5v.

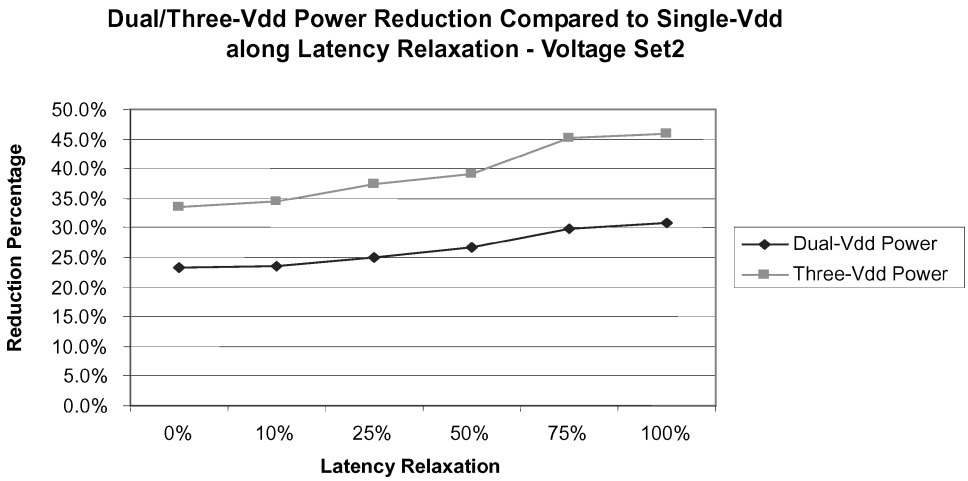


Fig. 12. Power reduction results comparing *opti-mvdd* to *opti-hvdd* for each latency relaxation point respectively; dual-Vdd is 1.3v/1.0v, and three-Vdd is 1.3v/1.0v/0.7v.

To understand how much power is saved due to power gating, we collect some data using 100% latency relaxation. Power gating can provide up to 7% power savings for adders for certain designs. However, power gating does not offer much savings overall and is almost negligible on average for our designs. We believe this is due to the following reasons. First, power gating being applied to a DFG does not have much opportunities comparing to the case where it is applied to a CDFG (control data flow graph). For example, in a CDFG, once a conditional branch is not taken, the functional units executing the operations specifically following that branch can be completely turned off. This is not considered in DFG optimizations. Second, we treat power gating as a post

processing step after dynamic power optimization as explained in Section 4. Therefore, the optimization space is not large. Third, our *SleepCycle* is long because the leakage power percentage in our resources is not that significant. We plan to study power gating in the future that will take leakage power reduction as our first priority. We also plan to extend our work into CDFG domain.

6. CONCLUSIONS AND FUTURE WORK

In this article, we presented high-level synthesis techniques for optimizing power and energy considering multi-Vdd assignment and switching activity reduction simultaneously. We developed a polynomial-time optimal algorithm and showed that our algorithm was consistently better than an optimization flow that separated voltage assignment from functional unit binding. In addition, we studied the impact of different voltage levels and their combinations on power and energy reduction. We also studied power-latency tradeoffs and power-gating potentials. Our results showed significant power and energy reduction compared to the scheme where only a single high Vdd was used. Future works include CDFG extension with a focus of leakage power reduction. Problem formulations with chained operations (in a single control step) and pipe-lined FUs are also under consideration.

REFERENCES

ANDERSON J. AND NAJM, F. 2004. Low-power programmable routing circuitry for FPGAs. In *Proceedings of the IEEE International Conference on Computer-Aided Design* (San Jose, CA). IEEE Computer Society Press, Los Alamitos, CA, 602–609.

BOGLIOLO, A., BENINI, L., RICCÓ, B., AND DE MICHELI, G. 1999. Efficient switching activity computation during high-level synthesis of control-dominated designs. In *Proceedings of the International Symposium on Low Power Electronics and Design* (San Diego, CA). 127–132.

CHANG, J. AND PEDRAM, M. 1995. Register allocation and binding for low power. In *Proceedings of the Design Automation Conference* (San Francisco, CA). 29–35.

CHANG, J. AND PEDRAM, M. 1996. Module assignment for low power. In *Proceedings of the EURO-Design Automation Conference* (Geneva, Switzerland). 376–381.

CHANG, J. AND PEDRAM, M. 1997. Energy minimization using multiple supply voltages, *IEEE Trans. VLSI Syst. 5*, 4, 436–443.

CHEN, D. AND CONG, J. 2004a. Register binding and port assignment for multiplexer optimization. In *Proceedings of the Asian Pacific Design Automation Conference* (Yokohama, Japan). 68–73.

CHEN, D. AND CONG, J. 2004b. Delay optimal low-power circuit clustering for FPGAs with dual supply voltages. In *Proceedings of the International Symposium on Low Power Electronics and Design* (Newport Beach, CA). 70–73.

CHEN, D., CONG, J., AND FAN, Y. 2003. Low-power high-level synthesis for FPGA architectures. In *Proceedings of the International Symposium on Low Power Electronics and Design* (Seoul, Korea). 134–139.

CHEN, D., CONG, J., LI, F., AND HE, L. 2004. Low-power technology mapping for FPGA architectures with dual supply voltages. In *Proceedings of the International Symposium on Field Programmable Gate Arrays* (Monterey, CA). 109–117.

CONG J. AND LIU, C. 1991. On the k-layer planar subset and topological via minimization problems. *Trans. CAD 10*, 459–463.

DE MICHELI, G. 1994. *Synthesis and Optimization of Digital Circuits.* McGraw-Hill, New York.

DILWORTH, R. 1950. A decomposition theorem for partially ordered set. *Ann. Math 51*, 161–166.

DUARTE, D., TSAI, Y., VIJAYKRISHNAN, N., AND IRWIN, M. 2002. Evaluating run-time techniques for leakage power reduction. In *Proceedings of International Conference on VLSI Design*. 31–38.

GELSINGER, P. 2004. Giga-scale integration for tera-ops performance: Opportunities and new frontiers. *41st DAC Tuesday Keynote*.

GONZALEZ, R., GORDON, B., AND HOROWITZ, M. 1997. Supply and threshold voltage scaling for low power CMOS. *IEEE J. Solid-State Circ. 32*, 8, 1210–1216.

HAMADA, M., TAKAHASHI, M., ARAKIDA, H., CHIBA, A., TERAZAWA, T., ISHIKAWA, T., KANAZAWA, M., IGARASHI, M., USAMI, K., AND KURODA, T. 1998. A top-down low power design technique using clustered voltage scaling with variable supply-voltage scheme. In *Proceedings of the Custom Integrated Circuits Conference.* 495–498.

ITRS, INTERNATIONAL TECHNOLOGY ROADMAP FOR SEMICONDUCTORS. 2003. Semiconductor Industry Association.

JOHNSON, M. AND ROY, K. 1997. Datapath scheduling with multiple supply voltages and level converters. *Trans. Des. Automat. Elect Syst. 2*, 3, 227–248.

KAO, J., NARENDRA, S., AND CHANDRAKASAN, A. 2002. Subthreshold leakage modeling and reduction techniques. In *Proceedings of the International Conference on Computer-Aided Design* (San Jose, CA). 141–148.

LI, F., CHEN, D., HE, L., AND CONG, J. 2003. Architecture evaluation for power-efficient FPGAs. In *Proceedings of the International Symposium on Field Programmable Gate Arrays* (Monterey, CA). 175–184.

LI, F., LIN, Y., AND HE, L. 2004. FPGA power reduction using configurable dual-Vdd. In *Proceedings of the Design Automation Conference* (San Diego, CA). 735–740.

LI, F. AND HE, L. 2001. Maximum current estimation with consideration of power gating. In *Proceedings of the International Symposium on Physical Design* (Sonoma, CA). 106–111.

LIN, Y. R., HWANG, C. T., AND WU, A. 1997. Scheduling techniques for variable voltage low power design. *Trans. Des. Automat. Electron. Syst. 2*, 2, 81–97.

LYUH, C. G. AND KIM, T. 2003. High-level synthesis for low-power based on network flow method. *Trans. VLSI Syst. 11*, 3, 364–375.

MANZAK, A. AND CHAKRABARTI, C. 2002. A low power scheduling scheme with resources operating at multiple voltages. *Trans. VLSI Syst. 10*, 1, 6–14.

MUTOH, S., DOUSEKI, T., MATSUYA, Y., AOKI, T., SHIGEMATSU, A., AND YAMADA, J. 1995. 1-V power supply high-speed digital circuit technology with multithresholdvoltage CMOS. *J. Solid-State Circ. 30*, 8, 847–854.

RAJE, S. AND SARRAFZADEH, M. 1995. Variable voltage scheduling. In *Proceedings of the International Symposium on Low Power Design* (Dana Point, CA). 9–14.

SARRAFZADEH, M. AND LOU, R. D. 1993. Maximum k-covering of weighted transitive graphs with applications. *Algorithmica 9*, 1, 84–100.

SRIVASTAVA, M. B. AND POTKONJAK, M. 1995. Optimum and heuristic transformation techniques for simultaneous optimization of latency and throughput. *Trans. VLSI Syst. 3*, 1, 2–19.

TAKAHASHI, M. HAMADA, M., NISHIKAWA, T., ARAKIDA, H., FUJITA, T., HATORI, F., MITA, S., SUZUKI, K., CHIBA, A., TERAZAWA, T., SANO, F., WATANBE, Y., USAMI, K., IGARASHI, M., ISHIKAWA, T., KANAZAWA, M., KURODA, T., AND FURUYAMA, T. 1998. A 60mW MPEG4 video codec using clustered voltage scaling with variable supply-voltage scheme. *J. Solid-State Circ. 33*, 11, 1772–1780.

USAMI, K. AND HOROWITZ, M. 1995. Clustered voltage scaling for low-power design. In *Proceedings of the International Symposium on Low Power Design* (Dana Point, CA). 3–8.

USAMI, K., IGARASHI, M., MINAMI, F., ISHIKAWA, T., KANAZAWA, M., ICHIDA, M., AND NOGAMI, K. 1998. Automated low-power technique exploiting multiple supply voltages applied to a media processor. *J. Solid-State Circ. 33*, 3, 463–472.

Received January 2005; revised October 2005; accepted November 2005