Energy Minimization Using Multiple Supply Voltages

Jui-Ming Chang and Massoud Pedram

***Abstract*— We present a dynamic programming technique for solving the multiple supply voltage scheduling problem in both nonpipelined and functionally pipelined data-paths. The scheduling problem refers to the assignment of a supply voltage level (selected from a fixed and known number of voltage levels) to each operation in a data flow graph so as to minimize the average energy consumption for given computation time or throughput constraints or both. The energy model is accurate and accounts for the input pattern dependencies, re-convergent fanout induced dependencies, and the energy cost of level shifters. Experimental results show that using three supply voltage levels on a number of standard benchmarks, an average energy saving of 40.19% (with a computation time constraint of 1.5 times the critical path delay) can be obtained compared to using a single supply voltage level.**

***Index Terms*— Dynamic programming, energy minimization, functional pipelining, multiple supply voltages, scheduling.**

I. Introduction

ONE driving factor behind the push for low power design is the growing class of personal computing devices as well as wireless communications and imaging systems that demand high-speed computations and complex functionalities with low power consumption. Another driving factor is that excessive power consumption has become a limiting factor in integrating more transistors on a single chip. Unless power consumption is dramatically reduced, the resulting heat will limit the feasible packing and performance of VLSI circuits and systems.

The most effective way to reduce power consumption is to lower the supply voltage level for a circuit. Reducing the supply voltage however increases the circuit delay. Chandraskan *et al.* [1] compensate for the increased delay by shortening critical paths in the data-path using behavioral transformations such as parallelization or pipelining. The resulting circuit consumes lower average power while meeting the global throughput constraint at the cost of increased circuit area.

More recently, the use of multiple supply voltages on the chip is attracting attention. This has the advantage of allowing modules on the critical paths to use the highest voltage level (thus meeting the required timing constraints) while allowing modules on noncritical paths to use lower voltages (thus reducing the energy consumption). This scheme tends to result in smaller area overhead compared to parallel architectures.

Manuscript received September 8, 1996; revised May 15, 1997. This work was supported by ARPA under Contract F33615-95-C1627 and SRC under Contract 97-DJ-559.

The authors are with the Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, CA 90089 USA.

Publisher Item Identifier S 1063-8210(97)09203-2.

There are, however, a number of practical problems that must be overcome before use of multiple supply voltage becomes prevalent. These problems include routing of multiple supply voltage lines, area/delay overhead of required level shifters, and lack of design tools and methodologies for multiple supply voltages. The first issue is an important concern which should be considered by any designer who wants to use multiple supply voltages. That is, there is a trade-off between lower energy dissipation and higher routing cost. The remaining issues (that is, level shifter cost and lack of tools) are addressed in this paper. That is, we will show that the area/delay overhead of level shifters is relatively small and will present an effective algorithm for using multiple supply voltages during behavioral synthesis.

In this context, an important problem is to assign a supply voltage level (selected from a finite and known number of supply voltage levels) to each operation in a data flow graph (DFG) and schedule various operations so as to minimize the energy consumption under given timing constraints. We will refer to this problem as the *multiple-voltage scheduling* problem or the MVS problem for short.

In this paper, we tackle the problem in its general form. We will show that the MVS problem is NP-hard even when only two points exist on the energy-delay curve for each module (these curves may be different from one module to another), and then propose a dynamic programming approach for solving the problem. This algorithm which has pseudo-polynomial complexity (cf., Section IV-C) produces optimal results for trees, but is suboptimal for general directed acyclic graphs. The dynamic programming technique is then generalized to handle *functionally pipelined* designs. This is the first time that the use of multiple supply voltages in a functionally pipelined design is considered. We will present a novel *revolving schedule* for handling these designs.

The paper is organized as follows. In Section II, we summarize related work. In Section III, we describe timing and energy consumption models for nonpipelined designs. In Section IV, we present a dynamic programming approach for solving the multiple-voltage scheduling problem for the tree-like DFG's and then for general DFG's. In Section V, we extend the approach to functionally pipelined designs. Experimental results and concluding remarks are provided in Sections VI and VII.

II. Related Problems

The multiple-voltage scheduling problem (MVS) as described above is closely related to the *circuit implementation problem* as defined in [2]. The latter problem is to minimize the

total gate area in a circuit by selecting a gate implementation for each circuit node while meeting a timing constraint. It was shown in [2] that even under a fanout (load) independent delay model, with two implementations per circuit node, equal signal arrival times at inputs, and chain-like circuit structure, the problem of finding a solution where circuit area $\leq \alpha$ and signal arrival time $\leq \beta$ is *NP-complete*. We will show (cf., Section IV) that the MVS problem for minimum energy is also *NP-complete*.

Another similar problem is that of delay constrained technology mapping [3]–[5]. Our method for solving multiple voltage scheduling is similar to the method used in [4], [5]. In these works, the authors use dynamic programming to cover a subject graph by a library of pattern graphs with the goal of minimizing area/power while satisfying given timing constraints.

The MVS problem was tackled in [6] where the authors proposed an algorithm for minimizing the energy consumption of a nonpipelined design while meeting the computation time constraint. The authors assume that delay versus supply voltage curves for all modules in the design library are given and propose an iterative improvement algorithm for solving the problem. The approach is optimal for general directed acyclic graphs. However, the authors make a number of simplistic and rather unrealistic assumptions (e.g., the assumption that the difference of squares of the consecutive voltages on the delay versus voltage curve is fixed; the independence of energy consumption of a module from data activity at its inputs; identical latency vs. supply voltage curves for all modules in the circuit including adders and multipliers). The first assumption enables the authors to reduce the problem of $\min\ \Sigma_{i\in \text{ modules}}\ E_i$ under given computation time constraint where E_i is the energy consumption of module i to $\max\ \Sigma_{i\in \text{ modules}}\ d_i$ where d_i is the delay of module i for the corresponding voltage assignment. If the assumptions made in [6] do not hold for a given problem instance, then their proposed algorithm will produce a suboptimal solution without any performance guarantee.

Usami and Horowitz [7] proposed a technique to reduce the energy consumption in a circuit by making use of two supply voltage levels. The idea is to operate gates on the critical paths at the higher voltage level and the gates on the noncritical path at the lower voltage level. In this manner, the energy consumption is minimized without affecting the circuit speed.

Power profiler [8] primarily uses a *genetic search algorithm* to solve the multiple voltage scheduling problem. Johnson and Roy presented an ILP-based formulation for the multiple voltage scheduling problem for nonpipelined design in [9]. Both algorithms have exponential worst case complexity and hence the results are suboptimal for large problem instances where computation time is bounded due to practical considerations. In addition, they do not address conditional branches; nor do they consider functional pipelining. Their energy models do not support input data dependency.

In comparison to previous work, our algorithm is able to find the minimal energy solution for tree-like DFG's under timing constraints, handles general DFG's and functionally pipelined designs, explicitly supports the conditional branches, uses an energy model that takes different input data switching activities into consideration, and has pseudo-polynomial time complexity.

III. Energy-Delay Curves

We assume there are latches on the inputs of all modules to synchronize the input arrival times, and no multiple module activations per cycle occurred.

A. The Timing Model

Let c-step denote a control step (clock cycle), the basic unit of time used in the DFG in behavioral level. When the supply voltage level of a module is lowered, the delay increases. For a given length t_c of a c-step, an operation may thus become a *multicycle operation*.

Let t_i^s be the starting time of operation i, a_i the *output arrival time* of operation i, d_i the execution time (delay) of operation i, t_c the length of a c-step, then we have the following:

$$a_i = t_i^s + d_i$$
$$t_i^s = \max_{(j,i)} \lceil a_j / t_c \rceil \cdot t_c$$

where operation j is a predecessor of operation i.

B. The Energy Dissipation Model

We present in this section two computational models for energy dissipation at behavioral level. Our optimization algorithm is however independent of the specifics of these energy models. More precisely, any energy macromodel whose parameters depend on the input and/or output activity factors can be used here. This includes for example, the power macromodel reported in [10].

We assume that the dynamic energy dissipation in a functional unit is given by this equation

$$E_{FU_i} = F_i(\alpha_{i,1}, \alpha_{i,2}) \cdot V_i^2 \tag{1}$$

where V_i is the supply voltage of functional unit $FU_i, \alpha_1^{FU_i}$ and $\alpha_2^{FU_i}$ are the average switching activities on the first and second input operands of FU_i, respectively, F_i is a function of $\alpha_1^{FU_i}$ and $\alpha_2^{FU_i}$ and in general may be nonlinear. We propose two methods to calculate E_{FU_i}/V_i^2 given the pairs $(\alpha_{i,1}, \alpha_{i,2})$.

The first method is based on look-up table, that is, we store energy dissipation values for various (α_1, α_2) combinations and interpolate to calculate the energy value for a given (α_1^*, α_2^*) combination which is not found in the table. This method can achieve very high accuracy based on the number of entries in the look-up table.

The second method is based on energy macro-modeling using a linear equation with α_1 and α_2 as random variables. More precisely, we use the least square fit to find a plane in the three-dimensional (3-D) space that best fits the set of points $(\alpha_{i,1}, \alpha_{i,2}, E_{FU_i}/V_i^2)$ for each module FU_i. From the least square fit, we obtain

$$F_i(\alpha_{i,1}, \alpha_{i,2}) = C_1 \times \alpha_{i,1} + C_2 \times \alpha_{i,2} + C_3, \tag{2}$$

TABLE I
ENERGY DISSIPATION OF DATA-PATH FUNCTIONAL UNITS

Circuit	E_{est}^{TLU}	E_{act}	err^{TLU}	E_{est}^{Eq2}	err^{Eq2} %
add16	98.86	100.94	2.06	98.97	1.95
mult16	14421	14097	3.26	12633	10.39
M16:2/1	18.27	19.10	4.34	20.38	6.70
M16:4/1	49.66	47.37	4.83	50.73	7.09

TABLE II
AVERAGE ENERGY OF A 16-BIT LEVEL SHIFTER

x \ y	1.5	2.4	3.3	5
1.5	0	38.4	58.4	88.0
2.4	28.0	0	64.0	128.0
3.3	36.0	49.6	0	142.4
5	73.6	88.0	104.0	0

C_i's depend on the module type, the input data width, the technology and logic style used, and the internal module structure. We obtain the C_i values for every module using gate-level simulation and the least square fit. The accuracy of the model can be improved by using more variables.

To validate our energy model, we present some results for the set of data-path modules used in our library which are implemented in a 1-μ technology (cf., Table I) using the two methods presented above. Table I presents energy values in pJ at $V = 5$ V when the input sequence has average activities of $\alpha_1 = 0.5, \alpha_2 = 0.1$ (random data for one operand and biased data for another operand). It is clear from these results that the table look-up method (with 100 entries) remains accurate over the range of α value whereas the curve fitting method becomes inaccurate for small α.

We have also assumed that the range of V_i is such that the major source of energy consumption is the capacitive charging/discharging; that is, E_i/V_i^2 remains constant as V_i is scaled down. This may not be true when static standby current becomes important at very low voltages.

With this macromodeling, we can calculate the energy consumption of each module alternative under different supply voltages and switching activities. Note that $\alpha_1^{FU_i}$ and $\alpha_2^{FU_i}$ are calculated by using behavioral simulation of the given DFG using the set of user-specified (application-dependent) input vectors.

Let $\boldsymbol{E_{LS\_i}}$ be the energy used by level shifter i in the circuit when its input changes once. The energy dissipation (in pJ) in a 16-bit level shifter per voltage level transition is given in Table II (all 16 bits are switching). The propagation delay through a level shifter taken from [7] for typical load value is less than $1ns$ (which makes it negligible compared to the propagation delay through the modules) (cf., Table IV). Note that at most one level shifter will be used after any module. We can absorb the delay costs (1 ns) for level shifters into the delay of the functional units they follow, because in the module library, the minimum module delay is at least 20 times larger than the level shifter delay. Multiplexors will be used to route data in for nonoverlapping operations that share the same module sequentially. From Table I, we can also see that the energy consumed in multiplexors is relatively small compared to energy dissipation in adders and multipliers. In any case, multiplexors are needed with or without multiple supply voltages.

We assume (and *enforce*) that each module is **active** only when it is performing an operation, and is in the **sleep mode** at all other times. The sleep mode can be achieved by clock gating or use of flip-flops with enable/disable.

C. Tradeoff Curves

We calculate on each node of the DFG a delay function (or delay curve) where each point on that curve relates the accumulated energy consumed on the subtree rooted at that node (or operation) and the output arrival time of the node when a certain module (with certain supply voltage level and hence delay) is used to perform that operation. Different module alternatives for the same operation give rise to different points on the delay curve. The accumulated energy is the sum of energy consumed in all modules in that subtree (including the root of that subtree) plus all energy consumed in the necessary level shifters.

The delay function is therefore represented by a set of ordered pairs of real positive number (t, e), where a piecewise linear function $e = f(t)$ can be constructed which describes the set of all possible energy-delay trade-off solutions.

Without loss of generality, we assume that for each module in the library, the C_i values [cf., (2)] are stored in a table like Fig. 1(a). During dynamic programming, we have to calculate the energy-delay trade-off points for each instance of the module. At that time, the input operand activities (α_1, α_2) are known from a behavioral simulation of the DFG; the information shown in Fig. 1(a) can be thus used to generate the energy delay curve shown in Fig. 1(b) and (c) for any given pair of (α_1, α_2). Points on the curve represent various voltage assignment solutions with different tradeoffs between the speed and energy.

We only keep *noninferior* points on each curve. Point p^* is a noninferior point if and only if there does not exist a point $P = (t, e)$ such that either $t \leq t^*, e < e^*$ or $t < t^*, e \leq e^*$.

IV. THE SCHEDULING ALGORITHM

We first describe scheduling of DFG's which are **trees**. The goal here is to obtain a minimum energy solution that binds the operations in DFG to modules in the library while satisfying a computation time constraint.

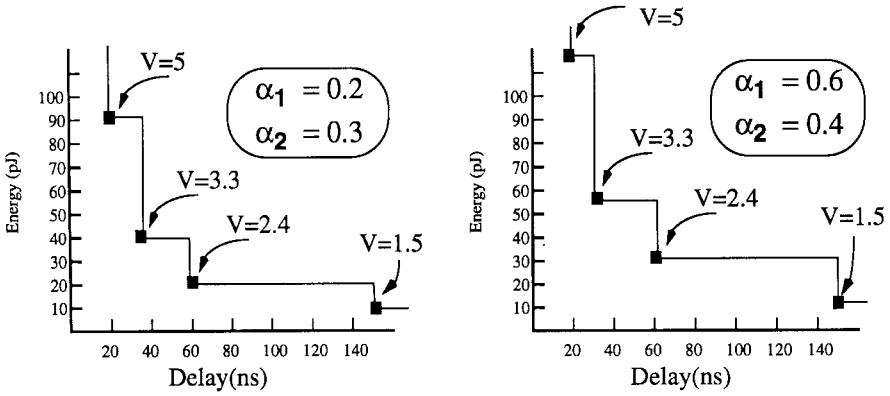
It is a simple exercise to formulate this problem as an integer linear programming problem (ILP). However, the ILP formulation does *not* take advantage of the *problem structure* and is in general very difficult and inefficient to solve. Instead, we use a *dynamic programming* approach as described next.

A. Post-Order Traversal

A post-order traversal of the tree is performed, where for each node n and for each module alternative at n, a new delay function is produced by appropriately *adding* the delay

16-bit Adder		Regression		
Voltage	Delay (ns)	Coefficients (pF) C_1	C_2	C_3
5.0	20.4	3.02	3.14	2.14
3.3	36.1	3.78	3.46	2.02
3.3	48.3	3.02	3.14	2.14
2.4	60.2	3.02	3.14	2.14
1.5	149.75	3.02	3.14	2.14

(a)



(b) (c)

Fig. 1. Our module library using the second method in Section III-B for a 16-bit adder and the energy versus delay curves.

functions at the children of node n. Adding must occur in the common region among all delay functions in order to ensure that the resulting merged function reflects feasible matches at the children of n. Note that the energy consumed in *level shifters* is computed during the postorder traversal by keeping track of the voltages used in the current node and its children (using Table II and switching activity information). The delay function for successive module alternatives at the same node n are then merged by applying a *lower-bound merge* operation on the corresponding delay functions. See [11] for details of operations.

The delay function addition and merging are performed recursively until the root of the tree is reached. The resulting function is saved in the tree at its corresponding node. Thus each node of the tree will have an associated delay function. The set of (t, e) pairs corresponding to the composite delay function at the root node defines a set of arrival time-energy tradeoffs for the user to choose from.

B. Pre-Order Traversal

The user starts with the total computation time constraint $\boldsymbol{T_{\mathrm{comp}}}$ at the root of the tree and performs a pre-order traversal to determine the specific point on each curve associated with each node of the tree. The timing constraints of children at the root are computed as $\boldsymbol{T_{\mathrm{comp}} - t_{\mathrm{delay}}}$, where $\boldsymbol{t_{\mathrm{delay}}}$ is the delay of the module alternative for the root which satisfies arrival time $\leq \boldsymbol{T_{\mathrm{comp}}}$ and has the minimum energy. This module selection and timing constraint propagation technique is applied recursively at all internal nodes during the pre-order traversal.

C. Complexity Analysis

Theorem IV.1: The MVS problem is *NP-complete*.

Proof: By restricting our DFG into a chain and allowing only two implementations for each operation in the chain, our problem is identical to the circuit implementation problem which is known to be NP-complete [2]. ■

Definition IV.1: Let $\mathcal{I}$ be an instance of a computational problem. Then $|\mathcal{I}|$ is the problem size and $\max(\mathcal{I})$ is the largest integer appearing in $\mathcal{I}$.

Definition IV.2: An algorithm $\mathcal{B}$ for a problem Π is pseudo-polynomial if it solves any instance $\mathcal{I}$ of Π in time bounded by a polynomial in $|\mathcal{I}|$ and $\max(\mathcal{I})$.

Let's scale delay values for all modules under different voltage assignments to become integers. Furthermore, let's denote the maximum computation time for a tree-like DFG (using the worst-case integer delay values on any path) by $T_{\max}$. Clearly, $T_{\max}$ is bounded from above by an integer M. Let $|\mathcal{I}| = n$ where n is the number of nodes in the DFG.

The MVS problem Π is a *number problem* [12] because there exists no polynomial p such that M is less than or equal to $p(n)$. This implies that we can develop an algorithm for solving Π with a pseudo-polynomial time complexity (Π is *not* NP-complete in the strong sense).

Theorem IV.2: Our dynamic programming algorithm provides a pseudo-polynomial time algorithm for exactly solving the MVS problem for tree-like DFG's.

Proof: The number of energy-delay points on each node in the tree is bounded from above by M. The algorithm thus has a time complexity of $n \cdot M$. Delay function merging and

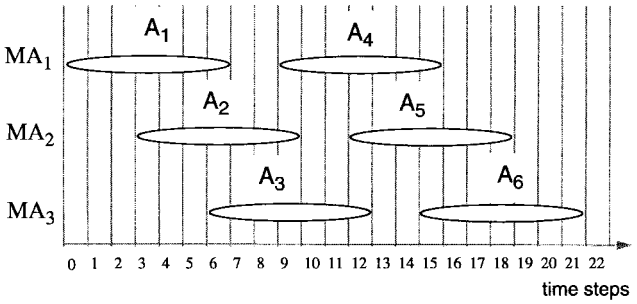


Fig. 2. An example to show the revolving schedule.

adding can be done in polynomial time in the number of points on the curves involved in the operations. Therefore, our algorithm for solving the MVS problem runs in pseudo-polynomial time because its time complexity is bounded above by a polynomial function of n and M as defined above.

Corollary IV.1: If the tree is node-balanced (its height is logarithmic in the number of its leaf nodes), then our dynamic programming algorithm runs in polynomial time.

D. Extension to General DFG's

The delay functions for nodes of a *general DFG* are computed by a post-order traversal as was the case for a tree-like DFG. The key question is how to add up the energy cost of children of a node during the post-order step.

We have adopted a heuristic whereby the energy value of a multiple fanout point is divided by its fanout count when its is propagated upward in the DFG. This heuristic is also adopted in technology mapping programs such as MIS [13] or ad-mapper [4] and tends to produce good results.

General DFG's contain *conditional branches*. We use nodes D and J to indicate the distribute and join nodes in order to express the conditional branches. For each D and J pair (which serve as *synchronization points*), there were two subgraphs which represent the "true" and "false" conditions, respectively. We treat the two subgraphs as if they are two simultaneous (parallel) subgraphs and apply dynamic programming technique on each subgraph except for the following. During the postorder traversal, when we come to a D node, we do not divide the cost of the subgraph rooted at D by two (in case of a single branch). Furthermore, when we come to a J node, we weight the cost of each branch by the probability that the branch is taken and then add the weighted branch costs to obtain the cost of the J node.

E. Module Sharing After Scheduling

It is difficult to account for the possibility of module sharing during dynamic programming. An attempt to consider sharing during the module assignment and scheduling phase will violate the principle of optimality that is the basis for using dynamic programming. This is because the dynamic programming cost at the root of a subtree cannot be determined independently of the rest of the tree (which is not yet mapped), so the optimal solution cannot be obtained by merging optimal solutions for the corresponding subproblems.

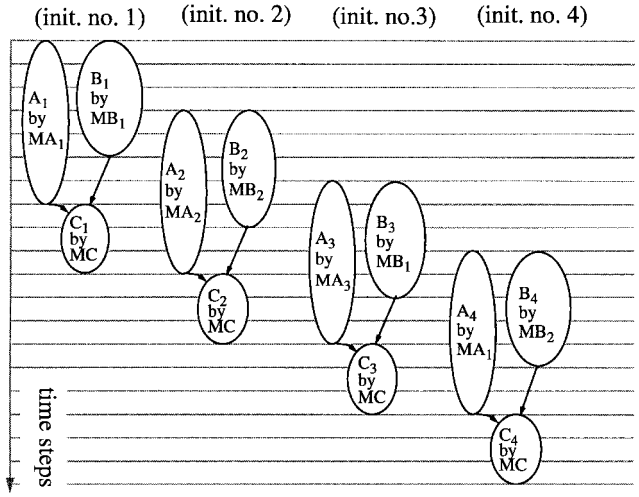


Fig. 3. Revolving schedule for a simple DFG.

After scheduling is completed, a module allocation and binding algorithm is applied whose goal is to exploit the possibility for sharing modules among compatible operations. This algorithm uses conventional techniques to detect operation compatibility and mutual exclusiveness of operations (as in parallel branches).

We use a scheme similar to that of [14] for minimum energy module binding using a max-cost network flow algorithm. Details can be found in [15].

V. Functionally Pipelined Data-Path

A. Background

In a functionally pipelined design, several instances of the execution of a data flow graph are overlapped in time. The time domain is discretized into *time steps* (for a given length of a time step). Unlike a structural pipelining, there is no physical (but logical) stages in a functional pipeline. Structural pipelining implies the use of pipelined modules, such as four-stage pipelined multiplier. Both functional and structural pipelining are aimed to increase the throughput of computation. *Latency* L is defined as the number of time steps between two consecutive pipeline initiations. A *control step* or c-step is a group of time steps that overlap in time (cf., Fig. 3). For a given latency L, c-step i corresponds to time steps $i + (m \cdot L)$, where m is an integer. We denote the L consecutive c-steps in a pipeline initiation as a *frame*. When the supply voltage level of a module is lowered, its delay increases and the operation assigned to the module may become *multicycle*. If the voltage is further lowered, for a small pipeline initiation latency L, an operation may become *multiframe*.

The *computation time* $\boldsymbol{T_{\mathrm{comp}}}$ of a functionally pipelined data-path is defined as the total time needed to process one data sample. Normally, a functionally pipelined circuit has to meet some throughput and/or computation time constraints. Throughput constraint is often more important than the computation time constraint in a functionally pipelined design.

Suppose we are given $\mathbf{N}$ input samples to be processed by a functionally pipelined data-path. Let T_{comp} be the computation

time and t_c be the length of a c-step. Then total time used is equal to $(N-1)\cdot L\cdot t_c + T_{\text{comp}}$. Let E_{LS} be the energy used by all of the level shifters in the circuit *per pipeline initiation* (or the energy used to process only one data sample) and E_{FU} be the average energy used by all of the modules per pipeline initiation. Then total energy used is $N\cdot(E_{FU}+E_{LS})$ and

$$\text{ave. pwr} = \frac{N\cdot(E_{FU}+E_{LS})}{(N-1)\cdot L\cdot t_c + T_{\text{comp}}} \approx \frac{(E_{FU}+E_{LS})}{L\cdot t_c}. \tag{3}$$

In our problem, the latency, L and t_c are assumed to be given. Therefore, when we minimize $(E_{FU}+E_{LS})$ which is the average total *energy* used by all modules and level shifters per pipeline initiation, we are indeed minimizing the average *power* dissipation.

An algorithm for performing scheduling and allocation for functionally pipelined DFG's is described in [16]. This technique known as the *feasible scheduling* deals with single cycle operations and operations that can be chained together in one c-step, but not multicycle or multiframe operations.

B. Handling Multiframe Operations

Our goal is to obtain a minimum energy functionally pipelined data-path realization while meeting the global throughput constraint (which is described by two parameters t_c and L). Suppose there is a module MA with delay equal to $k\cdot t_c$ where $(k/L)>1$, which is capable of performing an operation A in the DFG. To sustain the initiation rate of one data sample per $L\cdot t_c$, we use $\lceil(k/L)\rceil$ modules for operation A and use **a revolving schedule** as described next.

Suppose that we have modules $MA_1, MA_2\cdots MA_{\lceil k/L\rceil}$ for operation A in the DFG. Our revolving schedule assigns operation A on the $m\cdot\lceil k/L\rceil + 1st$ data sample (pipeline initiation) to module MA_1 at *time step* $L\cdot(m\cdot\lceil k/L\rceil)$, assigns operation A on the $m\cdot\lceil k/L\rceil + 2nd$ data sample to module MA_2 at *time step* $L\cdot(m\cdot\lceil k/L\rceil+1)$, etc., where $m = 0,1,2,\cdots$. Fig. 2 illustrates the result for $k=7$ and $L=3$.[1]

Theorem V.1: The *revolving scheduling* algorithm assigns the operation whose corresponding module delay is $k\cdot t_c$, where $(k/L)>1$ to $\lceil k/L\rceil$ modules without creating any resource conflict while meeting the throughput constraint of $1/L$ where L is the latency of the functional pipeline.

In the following, we show that the revolving schedule is the best possible schedule in terms of the number of the module instances used.

Theorem V.2: For any module with delay $k\cdot t_c$ where $(k/L)>1, \lceil k/L\rceil$ is the theoretical lower bound on the number of modules that have to be utilized in order to perform the corresponding operation with the pipeline latency of L without creating any resource conflict.

We next discuss how the dynamic programming approach has to be modified for the functionally pipelined designs. We consider three cases.

[1] All proofs can be found in [11].

1) Operation Delay $k\cdot t_c$ is Larger than $L\cdot t_c$: As shown before, here we have no choice but to use $\lceil k/L\rceil$ modules to perform the operation without creating any resource conflict while meeting the global throughput constraint. Recall that each module is active only when it is performing an operation, otherwise, it is in the sleep mode. In any time interval, given t_c and L, the total number of operations is the same regardless of the number of modules used to execute those operations. The total energy consumption for processing N data samples can be calculated as follows. Let the input vectors feeding to a module MA be denoted by V_1, V_2, V_3, V_4, etc. Suppose the corresponding operation becomes multiframe and thus we need to duplicate the module to MA_1 and MA_2. The input sequence feeding to MA_1 is V_1, V_3, etc., whereas that feeding to MA_2 is now V_2, V_4, etc. Obviously, the input activities for MA_1 and MA_2 are different from that of MA. However, the activities for MA_1 and MA_2 can still be calculated based on behavioral simulation results as long as we know how the data is multiplexed to either MA_1 or MA_2. This is known before the dynamic programming step based on the delay of MA and L. Next, the energy dissipation of module MA averaged over one time frame is calculated as the *arithmetic mean* of the energy dissipations of MA_1 and MA_2 under their respective input sequences. This is obviously valid only if we guarantee to shut off MA_1 or MA_2 when they are not in use. The area for module MA however increases by a factor of $\lceil k/L\rceil$.

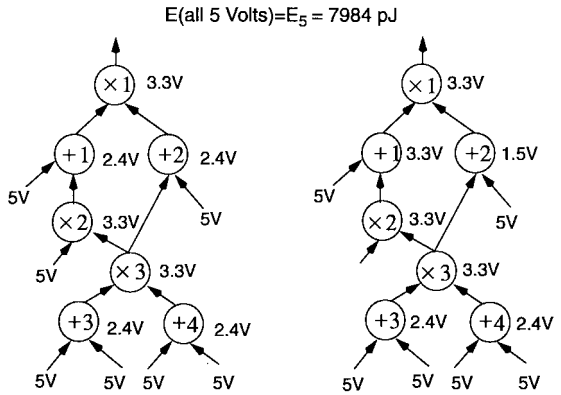
In Fig. 3 we show a very simple DFG with three operations A, B, and C. Suppose operations A, B, and C were assigned voltages during the scheduling step and the resulting delays for tentative modules MA, MB, and MC are, $7t_c, 5t_c$, and $3t_c$, respectively. Fig. 3 shows the result after using the revolving schedule. A_1 represents operation A performed in the pipeline initiation 1, etc.

2) Operation Delay $k\cdot t_c = L\cdot t_c$: We need to use exactly one module to perform the operation. No other operation can share the module. The energy cost of the operation is that of the corresponding module per data value.

3) Operation Delay $k\cdot t_c < L\cdot t_c$: We use one module per operation, however, the module may be shared. We again relegate the sharing issue to a post-processing phase where the scheduling solution obtained by dynamic programming approach is further modified to increase module sharing (thus reducing area cost of the design).

C. Module Sharing After Scheduling

Our goal is to minimize the resources after the scheduling has been done. The problem can be formulated as a minimal coloring of a circular arc graph [17]. (For a functionally pipelined data-path, a row in the resource allocation table is a track which is circular in nature, i.e., the Lth c-step in the current frame comes before the first c-step of next frame). The exact solution is obtained by the algorithm proposed in [18] which solves the register allocation problem in cyclic data flow graphs by using a multicommodity flow formulation. Instead, we have adopted a less expensive heuristic for doing module sharing as described next.



(a) (b)

Fig. 4. A small example. (a) Result obtained by our algorithm Total Energy $E_a = 3875$ pJ, $E_a/E_5 = 48.53\%$. (b) Result obtained by exhaustive search Total Energy $E_b = 3816$ pJ, $E_b/E_5 = 47.79\%$.

TABLE III
MODULE LIBRARY USED FOR FIG. 4 ($\alpha_1 = \alpha_2 = 0.5$)

	5.0V		3.3V		2.4V		1.5V	
	ns	*pJ*	*ns*	*pJ*	*ns*	*pJ*	*ns*	*pJ*
Mod.	del	Ene	del	Ene	del	Ene	del	Ene
mu16	100	2504	175	1090	287	577	717	225
ad16	20	118	35.1	51.4	57	27	143	10.6
su16	20	118	35.1	51.4	57	27	143	10.6

The resource conflicts in a functionally pipelined data-path can be detected in a straight forward manner. See [11] for details.

VI. EXPEIRMENTAL RESULTS

We first present the result obtained by our algorithm on a small DAG (not a tree) and the result obtained by exhaustive search. We assume four voltages are available and that all primary inputs carry 5-V signals. The module library is shown in Table III. The energy consumed by the level shifters is shown in Table II. In this example, the length of a c-step is 30 (ns) and a total computation time constraint $T_{\text{comp}} = 700$ (ns). The results of dynamic programming algorithm and exhaustive search are shown in Fig. 4. Note our new method can handle a very large graph (more than thousands of nodes) in seconds, but the exhaustive search (and the ILP formulation) which can be used to obtain the true optimal solution can only handle a small example ($\leq$20 nodes) in a reasonable amount of time. The two solutions obtained are different, but the results show that our solution which is only 1% away from the optimal solution.

In the remainder of this section, we present detailed results of our algorithm on a number of standard benchmarks including a test DFG, AR filter, elliptical wave filter, discrete cosine transform, robotic arm controller, second-order adaptive transversal filter and differential equation solver.

We use the table look-up method presented in Section III-B for energy calculation. Our module library is shown in Table IV. For the sake of giving energy behavior of our library modules, the energy values in this table are reported for $\alpha_1 = \alpha_2 = 0.5$, but as shown in Section III-B, we calculate the energy values for any other α_1, α_2 pairs as they become necessary.

TABLE IV
MODULE ENERGY (IN pJ) FOR $\alpha_1^{FU} = \alpha_2^{FU} = 0.5$

	5.0V				3.3V	
	(ns)	(pJ)	(ns)	(pJ)	(ns)	(pJ)
Mod.	del	Ener.	del	Ener.	del	Ener.
mult16	103.7	16829	132.0	13265	181.20	7330.9
add16	20.4	130.65	-	-	36.14	56.91
sub16	20.4	130.65	-	-	36.14	56.91
	3.3V		2.4V		1.5V	
mult16	-	-	295.4	3877.5	721.15	1514.6
add16	48.31	61.40	60.27	30.10	149.75	11.76
sub16	-	-	60.27	30.10	149.75	11.76

TABLE V
EXPERIMENTAL RESULTS ON VARIOUS BENCHMARKS. †: CORRESPONDS TO THE CRITICAL PATH DELAY OF THE DFG. $t_c = 30$ ns AND $L = 3$

Cir-cuit	T_{comp} (ns)	(pJ) E^1	(pJ) E^3	% $\frac{E^3_{LS}}{E^3}$	% E^3/E^1
Test DFG	321†	33985	33985	0	100
	481	26856	20942	0.6	77.98
	642	26856	11532	1.4	42.94
AR Filter	510†	256578	219131	0.1	85.40
	765	213804	129214	0.6	60.43
	1020	213804	78073	1.5	36.52
EWF	690†	130747	130933	0.3	100.1
	1035	109360	60728	1.9	55.53
	1380	109360	34031	4.1	31.12
FDCT	240†	102738	102738	0	100
	360	81351	46018	1.3	56.56
	480	81351	25150	3.2	30.92
Robot Ctrl.	650†	297627	278995	0.1	93.74
	975	240594	127061	0.5	52.81
	1300	240594	85650	0.7	35.60
2nd ATF	180†	74106	74113	0.2	100
	270	66977	37681	1.7	56.26
	360	66977	20455	3.4	30.54
Diff Eq.	300†	94500	88507	0.3	93.65
	450	80242	47451	1.5	59.13
	600	80242	31086	2.1	38.74
Avg.	T_{cr}	-	-	0.1	96.12
	1.5 T_{cr}	-	-	1.2	59.81
	$2T_{cr}$	-	-	2.3	35.20

Note that this table, for example, shows that we have five $mult16$ implementations, two at 5.0 V and three at lower supply voltages. Difference between the two which operate at 5 V is their architectures (parallel multiplier versus Wallace tree multiplier).

Our experimental results are shown in Table V. In this table, E^1 is the energy dissipation corresponding to the supply voltage of 5 volt. E^3 is the average energy obtained when the libraries contain modules with {5, 3.3, 2.4 V} voltage levels.[2] The columns corresponding to E^i_{LS}/E^i are the percentage of energy consumed in level shifters over the total energy. The results show that although the power consumed in level shifters is not negligible, it is not large either. Note that we

[2] In [11], we also report results obtained by using only two or four supply voltage levels.

can delete level shifters for step-down voltage conversions as described in [7]. In our experiments, however we inserted the level shifters for both step-up and step-down conversions.

Table V shows that an average energy saving of 3.88, 40.19 and 64.8% is achieved when using 3 supply voltage levels with total computation time ($T_{\rm comp}$) set to $T_{\rm crit}$ (the longest path delay in the DFG), 1.5 $T_{\rm crit}$ and 2 $T_{\rm crit}$.

Energy saving for $T_{\rm comp} = T_{\rm crit}$ is very much circuit-dependent. That is, the energy saving is higher in circuits where the number of noncritical nodes is large. For the AR filter circuit, E^3/E^1 ratio is as low as 0.85 while for the FDCT circuit, this ratio is 1. Energy saving potential increases substantially when $T_{\rm comp} > T_{\rm crit}$, e.g., E^3/E^1 ratio for $T_{\rm comp} = 1.5\,T_{\rm crit}$ goes down to 0.60 and 0.57 for the AR filter and FDCT circuits, respectively.

In the functionally pipelined case, we can achieve lower average energy for a given throughput constraint (which is described by two parameters t_c and L) by using a larger computation time because larger $T_{\rm comp}$ will result in a solution that uses lower voltages and thereby lower average energy. Note that the throughput of the functional pipeline remains the same. However, this causes more operations to become multicycle or multiframe operations which will increase the number of modules used to achieve the same throughput constraint. Thus the computation time constraint indirectly controls the chip area.

VII. Conclusion

We presented a dynamic programming approach for assigning voltage levels to the modules in nonpipelining and functionally pipelined data-paths. The average power consumption can be reduced by using a single lowered supply voltage. If the computation time constraint is violated with only a single lower supply voltage, then pipelining or parallelism on whole or part of the circuit to recover performance has to be used. Although this is one way of trading the chip area for power, the area penalty is generally much higher. With a given computation time constraint, when multiple voltages are used, our algorithm will lower the supply voltages of operations which are not on the critical path while keeping the supply voltages of operations on the critical path at a maximum. The computation time constraint is thus achieved at lower area overhead.

The use of $\lceil k/L \rceil$ modules for a multiframe operation was necessary to maintain the throughput while reducing the average energy consumption in the data-path. This, however, increases the controller and multiplexor cost. The multiplexor cost can be easily obtained from Table I. An energy cost model for controller can be developed as a function of the total number of functional units used in the circuit. For example, by assuming that the energy cost of the controller scales with the $\log_2$ of the number of modules of a given type used in the circuit. Therefore, during the post-order traversal, we can add a term reflecting the extra energy consumed in the controller when using $\lceil k/L \rceil$ modules to implement an operation. Thus the accumulated energy consumed in each node in the DFG will include the energy consumed in the controller.

References

[1] A. Chandrakasan, M. Potkonjak, J. Rabaey, and R. W. Brodersen, "HYPER-LP: A system for power minimization using architectural transformations," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1992.

[2] W.-N. Li, A. Lim, P. Agrawal, and S. Sahni, "On the circuit implementation problem," in *Proc. IEEE-ACM Design Automation Conf.*, 1992.

[3] H. Toutai, W. Moon, R. Brayton, and A. Wang, "Performance-oriented technology mapping," in *Proc. M.I.T. Conf. Adv. Res. VLSI*, 1990.

[4] K. Chaudhary and M. Pedram, "Computing the area versus delay trade-off curves in technology mapping," *IEEE Trans. Computer-Aided Design*, vol. 14, Dec. 1995.

[5] C.-Y. Tsui, M. Pedram, and A. Despain, "Power efficient technology decomposition and mapping under and extended power consumption model," *IEEE Trans Computer-Aided Design*, vol. 13, Sept. 1994.

[6] S. Raje and M. Sarrafzadeh, "Variable voltage scheduling," in *Proc. Int. Workshop Low Power Design*, 1995.

[7] K. Usami and M. Horowitz, "Clustered voltage scaling technique for low-power design," in *Proc. Int. Workshop Low Power Design*, 1995.

[8] R. Martin and J. Knight, "Power profiler: Optimizing ASIC's power consumption at the behavioral level," in *Proc. IEEE–ACM Design Automat. Conf.*, 1995.

[9] M. Johnson and K. Roy, "Low-power data-path scheduling under resource," in *Proc. IEEE Int. Conf. Computer Design*, 1996.

[10] P. Landman and J. Rabaey, "Black-box capacitance models for architectural power analysis," in *Proc. Int. Workshop Low Power Design*, 1994.

[11] J.-M. Chang and M. Pedram, "How to minimize energy using multiple supply voltages," Univ. of Southern California, Los Angeles, Tech. Rep. CENG 96-13, 1996.

[12] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman, 1979.

[13] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Technology mapping in MIS," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1987, pp. 116–119.

[14] J.-M. Chang and M. Pedram, "Low power register allocation and binding," in *Proc. IEEE-ACM Design Automat. Conf.*, 1995.

[15] ———, "Power efficient register assignment," Univ. of Southern California, Los Angeles, Tech. Rep. CENG 95-03, 1995.

[16] N. Park and A. Parker, "Sehwa: A software package for synthesis of pipelines from behavioral specifications," *IEEE Trans. on Compter-Aided Design*, vol. 7, Mar. 1988.

[17] M. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*. New York: Academic, 1980.

[18] L. Stok, "Architectural synthesis and optimization of digital systems," Ph.D. dissertation, Eindhoven Univ. of Technol., The Netherlands, 1991.

Jui-Ming Chang received the B.S. degree in electrical engineering from National Taiwan University, Taiwan, in 1989 and the M.S. degree in computer engineering from University of Southern California (USC), Los Angeles. His major was VLSI CAD. He is currently working towards the Ph.D. degree candidate at USC.

His research interests include CAD of VLSI circuits (specializing in behavioral and system level synthesis of VLSI systems targeting low power), discrete, and combinatorial optimization.

Massoud Pedram for photograph and biography, see this issue, p. 351.