Energy Minimization Using Multiple Supply Voltages*

Jui-Ming Chang and Massoud Pedram
Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA 90089

Abstract

We present a dynamic programming technique for solving the multiple supply voltage scheduling problem in both non-pipelined and functionally pipelined data-paths. The scheduling problem refers to the assignment of a supply voltage level 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 four supply voltage levels on a number of standard benchmarks, an average energy saving of 53% (with a computation time constraint of 1.5 times the critical path delay) can be obtained compared to using one fixed supply voltage level.

1 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 speed of VLSI circuits and systems.

Traditionally, behavioral synthesis attempts to minimize the number of resources to perform a task in a given time or minimize the execution time for a given set of resources. It is necessary to develop behavioral synthesis techniques that target lower power dissipation in the circuit.

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. [ChPo92] 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 becoming common place. 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 non-critical paths to use lower voltages (thus reducing the energy consumption). This scheme tends to result in smaller area overhead compared to parallel architectures. For this scheme to work, we will however need to insert level-shifters between connected modules that operate at different supply voltage levels. The area and energy costs of these level shifters must be taken into account when comparing a multiple-supply voltage design with that of a fixed-supply voltage design.

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 (i.e., total computation time for non-pipelined designs or throughput constraint for pipelined designs). This problem was tackled in [RaSa95] where the authors proposed a multiple supply voltage scheduling approach for minimizing the power consumption while meeting the computation time constraint. The authors assume that they are given delay vs. supply voltage curves for all modules in the design library and propose a an iterative improvement algorithm for solving the problem. The approach is optimal for general directed acyclic graphs. However, the authors make a number of simplifying assumptions (e.g., identical delay vs. supply voltage curves for all modules in the circuit; the assumption that the difference of squares of the consecutive voltages on the delay vs. voltage curve is fixed; the independence of energy consumption of a module from data activity at its inputs). These assumptions enable the authors to reduce the problem of $Min \sum_{i \in modules} E_i$ under given computation time constraint where E_i is the energy consumption of module i to $Max \sum_{i \in modules} d_i$ where d_i is the delay of module i for the corresponding voltage assignment. Relaxing the assumptions mentioned above makes the algorithm proposed in [RaSa95] suboptimal.

Usami and Horowitz [UsHo95] 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 ones on the non-critical path at the lower voltage level. In

*This research was supported in part by SRC under contract no. 94-DJ-559 and ARPA under contract no. F33615-95-C-1627.

this manner, the energy consumption is minimized without affecting the circuit speed.

In this paper, we tackle the problem in its general form. We will show that the multiple-voltage scheduling 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 produces optimal results for trees, but is not optimal for general directed acyclic graphs. We will show that energy minimization given the total computation time or throughput constraints is equivalent to minimizing the average power dissipation. The dynamic programming technique is then generalized to handle functionally pipelined designs. This is the first time that 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 2, we summarize related work for discrete gate sizing (also known as the circuit implementation) problem which is closely related to the multiple-supply voltage scheduling problem addressed here. In Section 3, we describe timing and energy consumption models for non-pipelined designs. In Section 4, 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 5, we extend the approach to functionally pipelined designs. Experimental results and concluding remarks are provided in Sections 6 and 7.

2 Related Problems

Multiple-voltage scheduling problem as described above is closely related to the circuit implementation problem as defined in [LiLi92]. The 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 [LiLi92] that even under a fanout (load) independent delay model, with two implementations per circuit node, equal signal arrival times at inputs, and tree-like circuit structure, the problem of finding a solution where tree area $\leq \alpha$ and signal arrival time $\leq \beta$ is *NP-complete*. Multiple-voltage scheduling problem for low power is therefore *NP-hard* (Proof is by restriction [GaJo79].).

Another similar problem is that of delay constrained technology mapping [ToMo90] [ChPe92] [TsPe94]. Our method for solving multiple voltage scheduling is similar to the method used in delay constrained technology mapping [ChPe92] [TsPe94]. In these works, the authors try to cover a subject graph by a library of pattern graphs with the goal of minimizing area/power while satisfying given timing constraints. The approach consists of two steps: First, the delay functions (which capture arrival time-energy trade-offs) are generated at all nodes of the circuit using a post-order traversal. In the second step, a pre-order traversal is performed to determine the gate mapping of each node based on the user-specified required times at the circuit outputs.

3 Energy-delay Curves

3.1 The timing model

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

Each multi-cycle operation starts its execution on the boundary of a c-step, but it may finish its execution within a c-step. 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:

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

where operation j is a predecessor of operation i in the DFG.

3.2 The energy dissipation model

We assume that the dynamic energy dissipation in a functional unit is given by a simple equation as follows:

$$E_{FU\_i} = \frac{1}{2} C_i \cdot V_i^2 \cdot \alpha_i^{FU} \quad (1)$$

where V_i is the supply voltage of functional unit FU_i, C_i is the physical capacitance of the functional unit FU_i, and α^{FU} is the average switching activity at the inputs of FU_i. C_i is calculated for each functional unit using circuit or gate level simulation [Deng94] [BuNa93] and curve fitting. Obviously C_i depends on the module type, input data width, technology and logic style used and internal module structure. Equation (1) is the basis of many macro-modeling techniques for energy estimation and has been used in the works of [PoCh91][SvLi94] [MeRa94]. Power estimation accuracies of 10-15% have been reported for this model (A more accurate power macro-model is presented in [LaRa94].).

To be more precise, 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 1, where E_{act} and E_{est} denote the actual and estimated energy, respectively and Mn denote Mux of bitwidth=n.). Here the first column gives the functional unit name, the second column gives the actual energy dissipation obtained by gate-level simulation (using 10,000 random input vectors), the third column gives the C parameter obtained by curve-fitting, the fourth column gives the average switching activity per input line of the module and finally the last column gives the estimated energy dissipation obtained from equation (1). $V = 5$ volts and C is in units of pF. This table shows that equation (1) provides energy estimations for the modules in our library which are an average 9-10% in error (compared to E_{actual}).

With this macro-modeling, we can calculate the energy consumption of each module alternative under different supply voltages and switching activity. Note that α^{FU} is calculated through behavioral simulation of the given DFG using the set of user-specified (application-dependent) input vectors.

Let $\mathbf{E_{LS\_i}}$ be the energy used by level shifter i in the circuit when its input changes once. $\mathbf{E_{LS\_i}} = \mathbf{E_{LS\_static\_i}} + \mathbf{E_{LS\_dynamic\_i}}$. For a well designed level shifter, such as the one taken from [UsHo95], $\mathbf{E_{LS\_static\_i}} = \mathbf{0}$. The energy consumed in a 16-bit level shifter per voltage level transition is given in Table 2. Entry $(\mathbf{x}, \mathbf{y})$ in this table is the energy used for converting the output of a module which uses supply voltage x to the input of a module which uses

Circuit	E_{act} (pJ)	C (pF)	α^{FU}	E_{est} (pJ)	err %
add8	50.97	9.46	0.419	49.65	2.58
add16	106.76	18.91	0.428	101.2	5.50
add32	210.3	37.82	0.407	192.55	9.20
mult8	553.34	100.16	0.408	510.94	8.30
mult16	2310.6	400.64	0.409	2050.3	12.7
mult32	9024.8	1602.5	0.390	7820.5	15.4
M8:2/1	12.025	1.98	0.5	12.375	3.11
M16:2/1	24.05	3.96	0.5	24.75	7.60
M32:2/1	48.1	7.92	0.5	49.5	2.90
M8:4/1	34.02	5.58	0.5	34.875	2.50
M16:4/1	68.032	11.16	0.5	69.75	2.52
M32:4/1	136.06	22.32	0.5	139.5	2.52

Table 1: Data-Path Circuits and their simulation results under White Noise Model (V=5volts).

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

Table 2: Average **energy** consumption (in units of pJ) of a **16-bit** level shifter per logic transition (all 16-bits are switching) produced by Spice simulation.

supply voltage y. Using this table and the switching activity of the level shifters obtained from behavioral simulation, the dynamic energy consumption of the level shifters used in the design can be easily calculated. The propagation delay through a level shifter for typical load value is less than $1ns$ (which makes it negligible compared to the propagation delay through the modules) (cf. Table 3). From Table 1, we can also see that the energy consumed in multiplexors is relatively small.

The *average power* is given by:

$$\mathbf{power} = \frac{\mathbf{E_{FU}} + \mathbf{E_{LS}}}{\mathbf{T_{comp}}} \tag{2}$$

where E_{FU} and E_{LS} are the total energy consumption of all modules and all level shifters and $\mathbf{T_{comp}}$ is the total computation time for one data sample. $\mathbf{T_{comp}}$ is a user-specified constraint. Therefore, if we minimize ($\mathbf{E_{FU}} + \mathbf{E_{LS}}$), we are minimizing the **average power** dissipation in the circuit.

We assume (and *enforce*) that each module is **turned on** only when it is performing an operation, and is **turned off** at all other times.

3.3 Trade-off curves

We assume that for each module in the library, the energy vs. delay curve (representing energy-delay trade-offs for all possible supply voltage assignments to the module) is given. These curves, which are referred as delay curves, are generated and stored in the module library by simulating each module using a pseudo random data sequence. The energy values are however parametrizable with respect to the average data activity at the inputs of the module (cf. α^{FU} in equation (1)). Points on the curve represent various voltage assignment solutions with different trade-offs between the speed and energy.

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

4 The Scheduling Algorithm

We first describe scheduling of DFGs 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.

First, a post-order traversal is used to determine a set of possible output arrival times at the root (primary output) of the tree. Then a pre-order traversal is performed starting from the root to recursively determine the specific solution on each node in the tree based on the given computation time constraint.

We calculate on each node 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.

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 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 post-order traversal by keeping track of the voltages used in the current node and its children (using Table 2 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.

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 $(\mathbf{arrival\_time}, \mathbf{energy})$ pairs corresponding to the composite delay function at the root node defines a set of arrival time-energy trade-offs for the user to choose from.

The user can now use the total computation time constraint $\mathbf{T_{comp}}$ on the root of the tree and perform 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 is computed as $\mathbf{T_{comp}}$ - $\mathbf{t_{delay}}$, where $\mathbf{t_{delay}}$ is the delay of the module alternative of the root that makes the root satisfy arrival time $\leq$ $\mathbf{T_{comp}}$ 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.

Extension to general DFG's with conditional branches can be found in [ChPe96a].

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). [1]

5 Functionally Pipelined Data-path

5.1 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). **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. 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 **multi-cycle**. If the voltage is further lowered, for a small pipeline initiation latency L, an operation may become **multi-frame**.

The **computation time** $\mathbf{T_{comp}}$ 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.

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 $(\mathbf{N-1}) \cdot \mathbf{L} \cdot \mathbf{t_c} + T_{comp}$. Let $\mathbf{E_{LS}}$ 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 $\mathbf{E_{FU}}$ 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

$$\mathbf{power} = \frac{\mathbf{N \cdot (E_{FU} + E_{LS})}}{\mathbf{(N-1) \cdot L \cdot t_c + T_{comp}}} \approx \frac{\mathbf{(E_{FU} + E_{LS})}}{\mathbf{L \cdot t_c}} \quad (3)$$

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

[1] Two operations are said to be compatible if they can be executed by the same module with the same supply voltage level and hence delay if their lifetimes do not overlap. Two operations are said to be mutually exclusive if they cannot not be alive at the same time in a DFG with conditional branches.

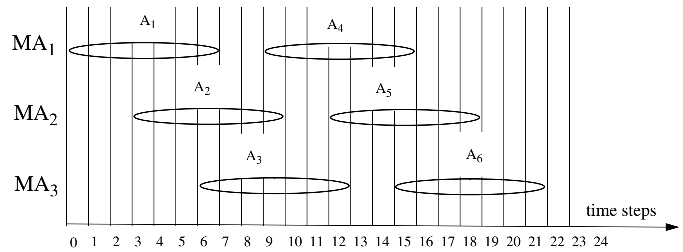

Figure 1: Example to Show the Revolving Schedule.

An algorithm for performing scheduling and allocation for functionally pipelined DFG's is described in [PaPa88]. 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 *multi-cycle* or *multi-frame* operations. The idea is to build a resource allocation table where columns correspond to c-steps, and rows correspond to module instances and entry (i, j) of the table denotes the assignment of operation(s) to module instance i at c-step j. In functional pipelining, two operations that use the same module are said to be *non-overlapping* if their life spans (in terms of the c-steps in which they are alive) are not overlapping or that they are mutually exclusive operations in a conditional DFG. The feasible scheduling algorithm [PaPa88] is basically a modified *list scheduling* with two main ingredients: *urgency* priority and *allocation table*. Forward/backward urgency of an operation is the longest path delay from an operation to the primary outputs/inputs. The main flow of list scheduling is preserved while the urgency priority is used to sort the list of operations and the resource allocation table is used to check for resource conflicts.

5.2 Handling multi-frame 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 $\frac{k}{L} > 1$, 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 $\lceil \frac{k}{L} \rceil$ modules for operation A and use **a revolving schedule** as described next.

Suppose that we have modules $MA_1, MA_2 \ldots MA_{\lceil \frac{k}{L} \rceil}$ for operation A in the DFG. Our revolving schedule assigns operation A on the $m \cdot \lceil \frac{k}{L} \rceil + 1st$ data sample (pipeline initiation) to module MA_1 at *time step* $L \cdot (m \cdot \lceil \frac{k}{L} \rceil)$, assigns operation A on the $m \cdot \lceil \frac{k}{L} \rceil + 2nd$ data sample to module MA_2 at *time step* $L \cdot (m \cdot \lceil \frac{k}{L} \rceil + 1)$, etc., where $m = 0, 1, 2, \ldots$. Example in Fig. 1 shows the revolving schedule of operation A on modules MA_1, MA_2 and MA_3 when the module delay $= 7t_c$ and pipeline latency, $L = 3(t_c)$. Note that A_i is the execution of operation A in pipeline initiation i, and c-step 1 = time steps {1,4,7,...}, c-step 2 = time steps {2,5,8, ...}.

The following results hold (Proofs. can be found in [ChPe96a]).

Theorem 5.1 *The* revolving scheduling *algorithm assigns the operation whose corresponding module delay is* $k \cdot t_c$, *where $\frac{k}{L} > 1$ to $\lceil \frac{k}{L} \rceil$ modules without creating any resource conflict while meeting the throughput constraint of $\frac{1}{L}$ where L is the latency of the functional pipeline.*

Theorem 5.2 *For any module with delay $k \cdot t_c$, where $\frac{k}{L} > 1$, $\lceil \frac{k}{L} \rceil$ 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) Operation delay $k \cdot t_c$ is larger than $L \cdot t_c$. As shown before, here we have no choice but to use $\lceil \frac{k}{L} \rceil$ modules to perform the operation without creating any resource conflict while meeting the global throughput constraint. Recall that each module is on only when it is performing an operation, otherwise, it is off. 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. Consequently, the total energy consumption for processing N data samples is the same regardless of the number of modules used. That is, energy consumption per operation per pipeline initiation remains the same as the energy consumption of the corresponding module per computation. The area however increases by a factor of $\lceil \frac{k}{L} \rceil$. This means that during the post-order traversal step of the dynamic programming, we must calculate the energy cost of a multi-frame operation assigned to module MA_i as the energy cost of one such module per computation (data value). Note that there is no possibility of sharing any of these $\lceil \frac{k}{L} \rceil$ modules with any other operations.

Consider energy calculation for module MA_r. The calculation should be input pattern dependent. The sequence of operands fed to module MA_r can be obtained by sampling the original sequence of operands feeding to operation A at the sampling period of $\lceil \frac{k}{L} \rceil$.

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 with other operations. It is difficult to account for the possibility of module sharing during dynamic programming as at any point during the post-order tree traversal we have only partial information about the operations that have been assigned to modules and cannot modify the dynamic programming cost function to reflect the sharing potential. We 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). We estimate the switching activity at the inputs of a module during the dynamic programming phase assuming that the module is not shared.

5.3 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 [Golu80]. (For a functionally pipelined data-path, a row in the resource allocation table [PaPa88] is a track which is circular in nature, i.e. the Lth c-step in the current frame should come before the first c-step of next frame). The exact solution is obtained by the algorithm proposed in [Stok91] which solves the register allocation problem in cyclic data flow graphs by using a multi-commodity flow formulation. Instead, we have adopted a less expensive heuristic for doing module sharing as described in [ChPe96a].

	5.0V		3.3V	
	(ns)	(pJ)	(ns)	(pJ)
Module	delay	Energy	delay	Energy
mult16	100	2504	175.20	1090.7
add16	20	181	35.05	51.4
sub16	20	181	35.05	51.4
	2.4V		1.5V	
mult16	286.80	576.9	717.03	225.3
add16	57.36	27.2	143.40	10.6
sub16	57.36	27.2	143.40	10.6

Table 3: Module Energy (in pJ) under a pseudo-random white noise data model at $\alpha^{\mathbf{FU}} = \mathbf{0.5}$

6 Experimental Results

We use the module library with voltage, energy, and delay as specified in Table 3. We performed our new methods on various benchmarks including a Test DFG, AR Filter, Elliptical Wave Filter[GeEl92], Discrete Cosine Transform, Robotic Arm Controller, 2nd-order Adaptive Transversal Filter [Hayk91] and Differential Equation Solver [CaWo91].

Our experimental results are shown in Table 4. E^1 and E^4 are the average energy obtained when libraries contain modules running at $\{5V\}$ and $\{5V, 3.3V, 2.4V, 1.5V\}$, respectively. Column corresponding to $\frac{E^i_{LS}}{E^i}$ is the percentage of energy consumed in level shifters over the total energy. Resules show that although the power consumed in level shifters is not negligible, it is not large either. Note that we can delete level shifters for step-down voltage conversions as described in [UsHo95]. In our experiments, however we inserted the level shifters for both step-up and step-down conversions.

In Table 4, †corresponds to the critical path delay of the DFG and $t_c = 30$ ns. This table shows that an average energy savings of 5.83%, 53.46% and 69.53% is achieved when using 4 supply voltage levels with total computation time (T_{comp}) set to T_{crit} (the longest path delay in the DFG), $1.5T_{crit}$ and $2T_{crit}$.

Energy saving for the case of $T_{comp} = T_{crit}$ is very much circuit-dependent. For the AR filter circuit, $\frac{E^4}{E^1}$ ratio is as low as 0.82 while for the $FDCT$ circuit, this ratio is 1. Energy saving potential increases substantially when $T_{comp} > T_{crit}$. For example, $\frac{E^4}{E^1}$ ratio for $T_{comp} = 2T_{crit}$ goes down to 0.28 and 0.29 for the AR filter and $FDCT$ circuits, respectively.

In the functionally pipelined case, we can easily achieve lower average energy at any throughput constraint by using appropriate t_c and L. Longer computation time constraint will result in a solution that uses lower voltages and thereby lower average energy. However, this causes more operations

Cir-cuit	T_{comp} (ns)	(pJ) E^1	(pJ) E^4	% $\frac{E^4_{LS}}{E^4}$	% E^4/E^1
Test DFG	320 †	5303	5303	0	100
	480	5030	2507	4.82	47.28
	640	5303	1846	4.66	34.81
ARF	510 †	41481	33895	0.9	81.71
	765	41481	17512	6.55	42.21
	1020	41481	11360	7.25	27.38
EWF	690 †	22965	21855	2.84	95.16
	1035	22965	12262	8.44	53.39
	1380	22965	7223	9.71	31.45
FDCT	240 †	16618	16618	0	100
	360	16618	8202	7.28	49.35
	480	16618	4849	10.72	29.18
RobC	660 †	46725	43024	0.86	92.08
	990	46725	19815	3.82	42.40
	1320	46725	13674	4.98	29.26
2ATF	180 †	13110	13084	0.62	99.8
	270	13110	6291	8.81	47.98
	360	13110	3743	14.88	28.55
Dif-Eq	300 †	15614	14123	1.44	90.45
	450	15614	6750	8.04	43.23
	600	15614	5106	9.44	32.70
Avg.	T_{cr}	-	-	0.95	94.17
	$1.5T_{cr}$	-	-	6.82	46.54
	$2T_{cr}$	-	-	8.81	30.47

Table 4: Experimental Results on Various Benchmarks.

to become multi-cycle or multi-frame 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.

For the same computation time constraint, using smaller t_c results in lower energy (but not power) dissipation compared to using larger t_c. The reason is that the multi-cycle operations start at the boundary of time steps. Using large t_c increases the *"dead time"* (time interval between the end of operation and the beginning of next time step) of a multi-cycle operation. Larger t_c thus tends to require higher voltages to meet the total computation time constraint.

7 Conclusion

We presented a dynamic programming approach for assigning voltage levels to the modules in non-pipelining and functionally pipelined data-paths. One can reduce the average power consumption by using a single lowered supply voltage. With only a single lower supply voltage, if the computation time constraint is violated, then one has to use pipelining or parallelism on whole or part of the circuit to recover performance. Although this is also one way of trading the chip area for power, the area penalty is 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.

References

[BuNa93] R. Burch, F. Najm, P. Yang and T. Trick, "A Monte Carlo approach for power estimation". In IEEE Trans. on VLSI page 63-61, March 1993.

[CaWo91] R. Camposanoand and W. Wolf, "High-level VLSI synthesis", PP. 256, Kluwer Academic Publishers, 1991.

[ChPo92] A. Chandrakasan, M. Potkonjak, J. Rabaey, and R.W. Brodersen,"HYPER-LP: A System for Power Minimization Using Architectural Transformations", In Proceedings of the ICCAD 1992.

[ChPe96a] J.-M. Chang and M. Pedram, "How to Minimize Energy Using Multiple Supply Voltages", CENG Technical Report 96-13, Computer Engineering Division, Dept. of EE-Systems, Univ. of Southern California, 1996.

[ChPe92] K. Chaudhary and M. Pedram, "Computing the area versus delay trade-off curves in technology mapping", In Proceedings of the IEEE Transactions on CAD, v14, n12, Dec 1995.

[Deng94] C. Deng, Power Analysis for CMOS/BiCMOS circuits. In Proceedingss of the 1994 International Workshop on Low Power Design, pages 3-8, April 1994.

[GaJo79] M. Garey and D. Johnson, "Computers and Intractability: A Guide to the Theory of NP-completeness", W. H. Freeman and Company, 1979.

[GeEl92] C. Gebotys and M. Elmasry, "Optimal VLSI Architectural Synthesis", Kluwer Academic Publishcation, pp. 148, 1992.

[Golu80] M. Golumbic, "Algorithmic Graph Theory and Perfect Graphs", Academic Press, 1980.

[Hayk91] S. Haykin, "Adaptive Filter Theory", 2nd edition, Chap5, 6, and 8, Printice Hall, 1991.

[LaRa94] P. Landman and J. Rabaey, "Black-Box Capacitance Models for Architectural Power Analysis", In Proceedings of the 1994 International Workshop Low Power Design, April 1994.

[LiLi92] W.-N. Li, A. Lim, P. Agrawal and S. Sahni, "On The Circuit Implementation Problem", In Proceedings of the 29th DAC, June 1992.

[MeRa94] R. Mehra and J. Rabaey. "Behavioral Level Power Estimation and Exploration.", In Proceedings of the 1994 International Workshop on Low Power Design, pp. 197-202, Apr, 1994.

[PoCh91] R. Powell and M. Chau, "A Model for Estimating Power Dissipation in a Class of DSP VLSI Chips", In IEEE Trans. on Circuits and Systems, Vol 38, No. 6, June 1991.

[PaPa88] N. Park, A. Parker,"Sehwa: A Software Package for Synthesis of Pipelines from behavioral Specifications", In IEEE Trans. on CAD, vol 7, no. 3, March 1988.

[RaSa95] S. Raje, M. Sarrafzadeh, "Variable Voltage Scheduling", In Proceedings of the 1995 International Workshop Low Power Design, 1995

[Stok91] L. Stok, "Architectural Synthesis and Optimization of Digital Systems", Ph.D Dissertation, Eindhoven University of Technology, 1991.

[SvLi94] C. Svensson and D. Liu, "A Power Estimation Tool and Prospects of Power Savings in CMOS VLSI Chips". In Proceedings of the 1994 International Workshop on Low Power Design, pp. 171-176, Apr 1994.

[ToMo90] H. Toutai, W. Moon, R. Brayton, and A. Wang. "Performance-oriented technology mapping". In Proceedings of the Sixth M.I.T. Conference on Advanced Research in VLSI, pp. 79-97, Apr. 1990.

[TsPe94] C.-Y. Tsui, M. Pedram, and A. Despain, "Power Efficient Technology Decomposition and Mapping Under and Extended Power Consumption Model", in IEEE trans. on CAD, vol. 13, No. 9, Sep 1994.

[UsHo95] K. Usami and M. Horowitz, "Clustered Voltage Scaling Technique for Low-Power Design", in Proceedings of the 1995 International Workshop on Low Power Design, pp. 3-8, 1995