

# Fine-Grained Supply Gating Through Hypergraph Partitioning and Shannon Decomposition for Active Power Reduction

Lawrence Leinweber and Swarup Bhunia  
Dept. of EECS, Case Western Reserve University, Cleveland, USA  
[lawrence.leinweber@case.edu](mailto:lawrence.leinweber@case.edu)

**Abstract**— Energy-efficient performance has emerged as the key design objective of high-performance logic circuits to address power-induced reliability concerns and battery life requirements in portable devices. In the sub-65nm technology regime, these problems continue to grow as leakage power becomes the predominant form of power consumption. Among numerous power reduction techniques employed at the circuit and architectural levels, supply gating has been proven to be very effective for standby power reduction. In this paper, we propose application of fine-grained supply gating to large complex circuits for active leakage and dynamic power reduction. A design methodology and associated CAD tool is developed to synthesize combinational logic using hypergraph partitioning and Shannon decomposition, which reduces both leakage and switching power by disabling unused logic dynamically in small clusters of gates. Simulation results for a set of ISCAS-85 benchmarks show that the proposed approach can achieve up to 40% saving in total power in active mode (and up to 37% saving in standby power) with negligible impact on performance and die area for a predictive 32 nm technology.

**Index Terms** – Low Power Design, Supply Gating, Active Power, Hypergraph Partitioning.

## 1. INTRODUCTION

Technological advancement in semiconductor fabrication has led to smaller transistors and higher clock speeds; however, power requirements have not scaled down. Increasing average power and power density have emerged as the major barriers to gigascale integration [2, 4-6]. These issues have significant impacts on electronic system design as increased energy consumption in battery-operated portable devices limits the effectiveness and even the viability of mobile applications. In high-performance systems, on the other hand, increased power dissipation causes high power density, which results in localized hotspots due to limited heat dissipation capability of the package. These localized hotspots cause concerns in terms of reliability of operation. Since logic circuits (such as datapath or controller) experience more activity during operation (compared to embedded memory), they suffer greater junction temperatures. Logic circuits are, therefore, more vulnerable to power-density induced reliability issues. Hence, it is essential to reduce active power in logic circuits.

Power is dissipated in the active mode of operation of a logic circuit in the form of (1) switching power and (2) active leakage power. Among the three major components of leakage (namely, subthreshold leakage, gate leakage, and band-to-band junction tunneling leakage), typically, subthreshold leakage dominates the active mode leakage current. This is because subthreshold leakage has exponential dependence on temperature and high junction temperature in active mode (caused by higher activity) results in high subthreshold leakage. On the other hand, dynamic power of logic elements is controlled by operating voltage/frequency, switching activity per clock cycle, and switching capacitance.

To date, numerous techniques have been developed to reduce the active power in logic circuits. These techniques can be classified

into two broad categories: (1) techniques that target dynamic power reduction and (2) techniques for active leakage reduction. Major techniques in the first category include static or dynamic voltage scaling (which has quadratic impact on power), frequency scaling and clock gating. On the other hand, dual- $V_{th}$  design has been established as an effective active leakage reduction technique. However, it requires a fabrication process supporting high and low- $V_{th}$  transistors on the same die. It can also adversely impact parametric yield under process variations due to increased number of timing-critical paths in a circuit [4].

An effective technique for saving active power in logic circuits is to use *supply gating*, which effectively “gates” the unused sections of a large system to save both switching and leakage power [2, 5, 10-12]. To realize this technique, a NMOS (or PMOS) gating transistor can be inserted in series with the NMOS (or PMOS) network of a logic gate. Stacked transistors reverse bias the NMOS transistors resulting in exponential reduction in sub-threshold leakage [10-12]. Moreover, such a gating technique can also prevent switching of a logic gate, thus saving dynamic power. The effectiveness of supply gating in power saving can be observed from Fig. 1(a) [2] that shows the leakage breakdown of an inverter chain circuit (with and without supply gating) as well as the dynamic switching power. Dynamic switching power is measured in the active mode for a frequency of 1GHz and input switching activity of 20%. It is expected that an extra transistor in series with the charging or discharging path adversely affects circuit delay. Fig. 1(b) shows the impact on delay. From these figures, we can observe that significant saving in total power (>40%) can be achieved with little impact on path delay (<6% for gating transistor width of 6X considering that 50% of logic gates in a path are charging or discharging).

Supply gating has been explored earlier for leakage saving in the active mode of operation [5]. The basic idea in this case is to turn off a complete logic module when its outputs are not used for several clock cycles. However, many logic gates inside a module (e.g. an ALU) do not functionally contribute to the output values even when performing useful computations. A fine-grained supply gating approach can target saving active power in these “idle” (i.e. functionally non-contributing) logic gates. In [2], authors propose a technique to save active leakage power by cofactoring a logic block using Shannon decomposition and then supply gating the cofactors. Irrespective of the state of the control variable (of cofactoring), only

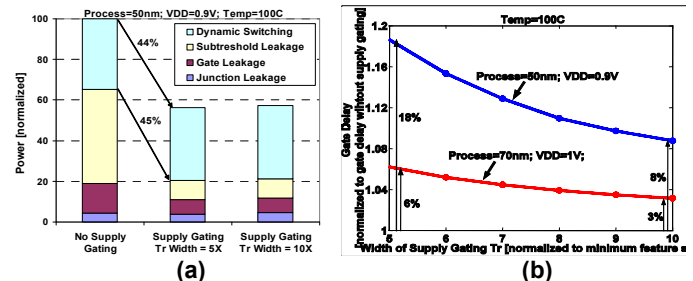


Figure 1: (a) Power reduction with supply gating for 50nm process; (b) corresponding impact on delay [2].

one of the cofactors takes part in the function evaluation at a time. Hence, we can dynamically apply supply gating to a cofactor to save leakage and dynamic power in the active mode of operation. Multiple levels of Shannon decomposition can be applied to increase the power saving. However, the technique proposed in [2] does not scale well for large, complex circuits. Application of Shannon decomposition to large benchmarks results in negligible power saving (or even an increase in power) along with an unacceptable area increase. This is primarily due to the fact that in a large circuit, typically, logic elements are less cohesive, i.e. a very small part of the logic depends on a single control variable. Section 2.2 explains this situation in details with simulation results. To address these issues with the existing supply gating techniques, we propose a fine-grained supply gating technique suitable for large circuits. Instead of directly applying Shannon decomposition to the entire circuit, in the proposed approach, first we apply hypergraph partitioning to generate cohesive partitions of the circuit and then choose only a subset of appropriate partitions for supply gating.

In particular, the paper makes the following contributions:

1. It presents a supply gating methodology for active power reduction that is suitable for large and complex logic circuits. The proposed supply gating scheme is “*fine-grained*” with respect to both circuit and timing granularity. As a by-product, it also results in significant saving in standby power.
2. It proposes a low-complexity hypergraph partitioning approach for clustering the logic gates in a large circuit in a manner that makes the clusters suitable for Shannon expansion.
3. It describes an automatic synthesis tool that implements the proposed methodology with low run-time complexity.

The rest of the paper is organized as follows. Section 2 provides some background information and motivation behind the proposed approach. Section 3 presents the synthesis flow that incorporates fine-grained supply gating. Section 4 presents the simulation setup and comparison of power, performance and area. Section 5 concludes the paper.

## 2. BACKGROUND AND MOTIVATION

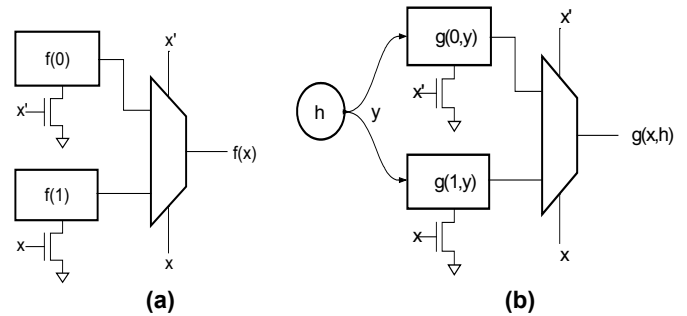
In this section, we will provide a brief overview of Shannon expansion based supply gating described in [2].

### 2.1 Shannon Expansion based Supply Gating

In Shannon expansion, a Boolean expression  $f$  is factored into two terms: one corresponds to control variable = 0 and another to control variable = 1:

$$f(x) = f(0) \cdot \bar{x} + f(1) \cdot x \quad (1)$$

Thus, the logic for  $f(0)$  can be stacked with a supply-gating transistor controlled by  $\bar{x}$  and the logic for  $f(1)$  can be stacked with a transistor controlled by  $x$ . The outputs of the two are combined in a multiplexer controlled by  $x$ , as illustrated in Fig. 2 (a). Thus, the Shannon decomposition technique requires that the circuit be factored into two sub-expressions, or *co-factors*. This is always



**Figure 2: General “Shannon decomposed” function with supply gating; (a) without sharing common logic and (b) with a shared logic block “h”.**

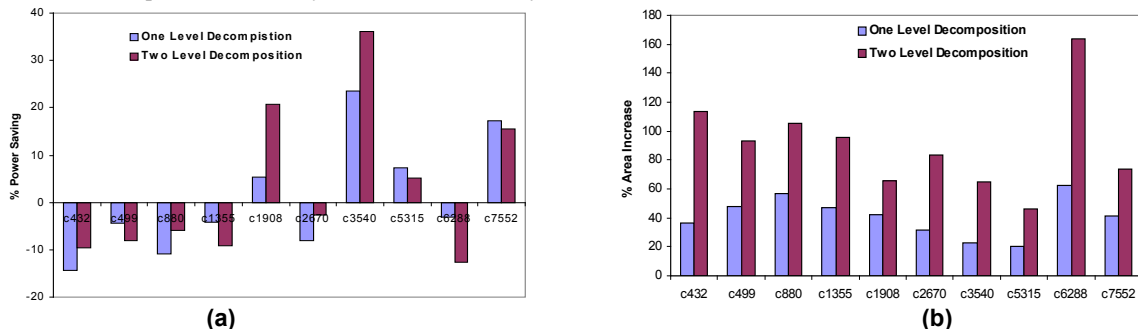
possible but, if not done carefully, can lead to some redundant logic. To prevent this, consider the following equation:

$$g(x, h) = g(0, h) \cdot \bar{x} + g(1, h) \cdot x \quad (2)$$

Here  $g$  has been factored with respect to  $x$ . The functional output of  $h$  (e.g.  $y$ ) is clearly an input to both cofactors of  $g$ . It is important to identify such signals when deriving the co-factors in order to avoid implementing the same logic twice, which wastes area and consumes leakage power in the disabled redundant copy. The implementation is illustrated in Fig. 2(b). The stacking transistors are typically NMOS since they offer lower resistance than PMOS for the same W/L ratio.

### 2.2 Motivation

It is not practical to use Shannon decomposition based supply gating (the way it has been proposed in [2]) on a complex and arbitrarily large circuit. To observe the effectiveness of Shannon decomposition on large circuits, we applied one and two-level Shannon decomposition on the ISCAS-85 benchmarks and plotted the area and estimated power savings. The results are shown in Fig. 3. It is clear that Shannon decomposition for the circuit as a whole causes a huge penalty in terms of area, while resulting in negligible (or even negative) saving in power. Although benchmark c3540 gives up to 36% power saving (for two-level decomposition), however, the corresponding area increase is more than 60%, which is not acceptable. Our investigation shows that there are primarily three reasons for the ineffectiveness of Shannon decomposition for large circuits: (1) For large circuits, the logic elements are not cohesive with respect to any single variable. Thus by choosing a single control variable, we create very small cofactors (that depend on the true and complementary state of the control variable) compared to the shared logic, largely nullifying the power saving strategy. (2) Since for large circuits, Shannon decomposition needs to be performed on a multi-level logic (because collapsing to two-level logic can become exponentially complex), it reduces the scope of logic optimization. In some circuits, a significant part of logic circuit is duplicated between  $f(0)$  and  $f(1)$ , increasing the overall area. (3) For multi-output circuits, there is a large amount of



**Figure 3: Comparison of area (a) and power (b) with one and two-level Shannon decomposition for the entire circuit (without partitioning).**

overhead due to the multiplexers if the number of outputs is large, which is often the case for the benchmarks considered.

Therefore, for an arbitrarily large circuit, we require dividing it into partitions, each of which can be Shannon decomposed independently. In this paper, we propose such a partitioning approach, which enables supply gating large logic circuits for active power reduction. Our approach is to represent the circuit as a *hypergraph* and to apply hypergraph partitioning techniques to it. A *hypergraph* is a generalization of a graph, where the set of edges are replaced by a set of *hyperedges*, each of which can connect any number of vertices. Formally, a hypergraph  $H = (V, E^h)$  is defined as a set of vertices  $V$  and a set of hyperedges  $E^h$ , where each hyperedge is a subset of  $V$ . The number of vertices that this hyperedge connects becomes the cardinality of this subset. It is important to note that, conventional hypergraph partitioning algorithms [3] that try to minimize the cut edges between partitions are not amenable for the proposed supply gating application. We need to develop a new partitioning algorithm to cluster the logic gates in a way that makes the partitions suitable for Shannon decomposition. We refer to this partitioning approach as *Shannon decomposition aware partitioning*. In the next section, we describe such a partitioning algorithm.

### 3. DESIGN FLOW

As mentioned earlier, the main idea is to partition the circuit into small cohesive logic blocks, which are amenable to Shannon cofactoring based supply gating. Fig. 4(a) shows the flowchart describing the major steps in the design process. The top-level design flow, as described in Fig. 4(a), starts with logic optimization and technology mapping for the given logic description followed by representing the circuit as a hypergraph and partitioning of the hypergraph. Circuit partitioning is a key step in the entire process, which controls how well parts of a circuit factor with Shannon decomposition. We can illustrate the advantage of partitioning with a simple example. Let us consider a circuit implementing the following two functions (i.e. a circuit with two outputs):

$$\begin{aligned} o1 &= x \cdot y \cdot z + \bar{x} \cdot y \cdot \bar{z} \\ o2 &= p \cdot q \cdot \bar{r} + \bar{p} \cdot q \cdot r \end{aligned} \quad (3)$$

Note that in this case, there is no cohesion between the logic cone for  $o1$  and that for  $o2$ . Thus considering the entire circuit for Shannon decomposition with respect to a single control variable (e.g.  $x$  or  $p$ ) brings half of the logic into the shared cofactor (Fig. 2(b)). Since only half of the logic will be part of  $f0$  or  $f1$ , the upper bound on power saving for one-level decomposition is 25%. On the other hand, if we partition the circuit into two blocks, one comprising the logic cone for  $o1$  and the other for  $o2$ , and then apply Shannon decomposition to each block separately, we can have perfect cofactoring (with no shared logic), that gives an upper bound of 50% on power saving. The partitioning algorithm developed here addresses this issue and aims at grouping the vertices into multiple partitions in a manner that minimizes shared logic. Similar to the general hypergraph partitioning problem, the formulation of the proposed partitioning problem is NP-complete [9], which is solved here by an heuristic-based method.

Once the partitioning is completed, each partition is considered for Shannon decomposition. However, due to circuit topology and sub-optimality of the partitioning solution, not all partitions will be amenable for Shannon decomposition. To distinguish partitions with respect to their potential for power saving and impact on area and circuit delay, we develop a cost metric and judiciously select a subset of partitions for Shannon decomposition. Note that this decomposition can be performed in a recursive fashion. The cofactors and shared logic generated in the first-level decomposition

can be subjected to further decomposition depending on the cost metric and the area/delay constraints.

Following the Shannon decomposition of the partitions, the next important step is extracting the shared logic. As proposed earlier in [2], it is possible to derive shared logic from a two-level representation (*sum of product*) of circuit functions. This is accomplished by identifying the product terms which are independent of the control variable and grouping them in the shared logic. However, this scheme does not work for large circuits due to the exponential increase in product terms in a function with the number of circuit inputs. In our experiment, even for one of the small ISCAS-85 benchmarks (*c432*) the collapsing routine in SIS [1] ran more than a day on a Sun workstation and failed to produce a two-level representation. We have developed an efficient approach for shared logic extraction, which is suitable for large benchmarks with an arbitrary number of inputs.

Supply gating transistors (NMOS) are inserted in the cofactors ( $f0$  and  $f1$ ) derived after Shannon decomposition. For one-level decomposition we need two gating transistors, where each gating transistor is shared across the entire cofactor logic. The size of the gating transistor is chosen in a way to minimize the performance impact. We assume 50% of the logic gates will switch on an average (statistically). Considering a  $5\lambda$  (where  $\lambda$  is the minimum feature size) wide gating transistor for an inverter, the shared gating transistor size is assumed to be  $N\lambda$  for a cofactor with  $N$  gates.

#### 3.1 Shannon Decomposition

Unlike the decomposition process considered in [2] (which works on two-level sum-of-product representations), we perform Shannon decomposition of the partitions from the multi-level logic description. We select the control variable for decomposition in two rounds, firstly by noting the number of logic gates affected by the true and the complementary form of each input variable. Secondly, we assign logic '0' (for  $f0$ ) and logic '1' (for  $f1$ ) to each candidate variable and subject the netlist to constant propagation (by using the "sweep" command in SIS [1]). We count the total number of logic gates in both  $f0$  and  $f1$ . The winning control variable is the input

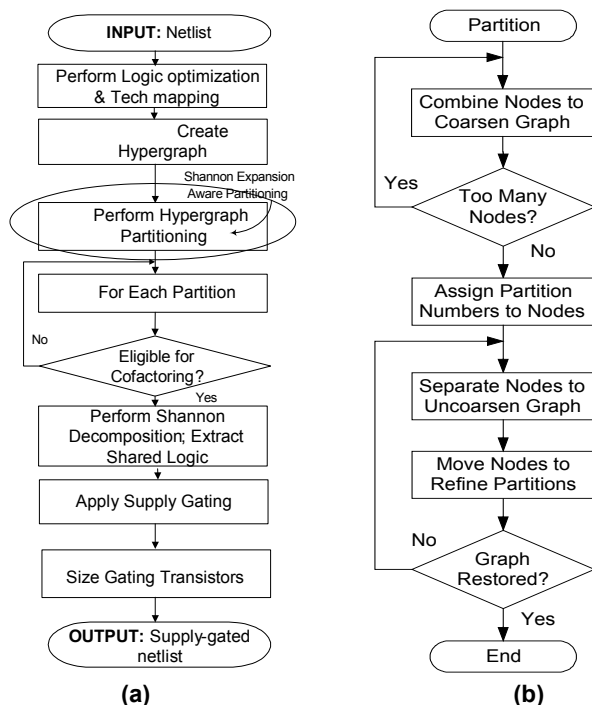


Figure 4: Flowchart of the algorithm: a) the overall approach; b) the steps in hypergraph partitioning.

that gives the least number of gates in this second round. For the selected control variable, we perform full synthesis to derive the two cofactors.

To reduce power and area overhead, next we perform Common Subexpression Elimination (CSE) on the multi-level representation of the cofactors. We treat the outputs from the cofactors as separate (by naming them separately, e.g.  $ol_{f0}$  and  $ol_{f1}$ , for output  $ol$ ), and merge the cofactors into a single netlist. Then we factor the merged netlist to separate the common cubes (using SIS command “ $fx$ ”). The factored netlist is analyzed by traversing from cofactor outputs toward inputs to separate the logic elements belonging to each cofactor. For example, while scanning the fan-in cone for  $ol_{f0}$ , we mark the logic gates that belong to the fan-in cone for this output in cofactor  $f0$ . The logic gate outputs used by both cofactors are included in the shared logic block. For each of the two cofactors, a supply-gating transistor is placed in series with the cofactor’s logic gates. Each output (e.g.  $ol$ ) is produced by combining cofactor outputs (e.g.  $ol_{f0}$  and  $ol_{f1}$ ) in a multiplexer, whose selector is the control variable.

### 3.2 Partitioning Algorithm

Fig. 4(b) shows a high level flowchart for partitioning, which consists of two major steps called *coarsening* and *uncoarsening*. In *coarsening*, nodes are drawn together to form a coarser, simpler graph. A series of progressively simpler hypergraphs are produced to represent the circuit at progressively higher levels. Coarsening continues until at the highest level each node represents one partition. Then during *uncoarsening*, the lower level nodes are assigned to their partitions and some are reassigned temporarily to other partitions in search of a better configuration.

In each step of coarsening, a new, simpler hypergraph is produced which is an abstraction of the previous one. This is carried out by selecting a hyperedge and drawing together all the nodes connected to that hyperedge into a group. The group is represented as a single node in the simpler hypergraph. This is done repeatedly until all the nodes have been drawn together. Hyperedges are selected for drawing together based on their cost. Lowest cost edges have highest priority. Then the vertices attached to each edge are drawn together, but only if they form a group disjoint from any other group. This process is repeated until there are no edges left. Finally, any widowed vertex is included in the higher level graph.

Edges that become parallel, joining the same set of new vertices, are combined and the number of lower level edges represented is summed. An edge that joins a subset of the vertices that are joined by another edge is similarly combined into the larger edge. The highest level of coarsening is reached when the number of vertices has reduced so that each vertex represents approximately the desired number of lowest level vertices. At that point, each vertex represents a partition.

Fig. 5 illustrates coarsening using an atypical hypergraph and cost function. Given the hypergraph shown in (a) and an edge cost function that favors drawing together edges that join the most vertices, there is an edge that joins 5 vertices, creating the node ACDFG in the next higher level graph shown in (b). Then the algorithm searches for the next best edge disjoint from ACDFG, which is HJ. There are no more edges disjoint from these two edges, but 3 vertices remain so the algorithm is run again for these vertices and their connecting edges. The only remaining edge is BE, forming the third node in the higher level graph (b). Finally, node ‘I’ widowed in (a) is copied to (b). Typically, a graph has many more nodes so coarsening is carried out to many levels, not shown in this example. If the desired number of vertices per partition was 2.5, then (b) would represent the highest level graph, and uncoarsening would consist of grouping vertices from the original graph as indicated by the dashed lines in (a). The important part of uncoarsening is refinement, in which some vertices are moved

between partitions for better clustering. Depending on the cost functions, this might move nodes C and F to the partition with I, as indicated in (c) which also has an edge structure slightly different from (b). At each step of uncoarsening, the partition numbers from the vertices at the next higher level that represent a group at the lower level are assigned to the lower level vertices. Then the cost of the entire hypergraph partitioning is evaluated and recorded as the best partitioning encountered before the refinement is performed again. In each iteration of the refinement, a vertex is selected at random and the relative cost of putting it in each partition is evaluated.

### 3.3 Cost Functions

We use two cost functions for evaluating the quality of a partitioning: cost for hyperedges and cost for vertices in each partition. Conventional hypergraph partitioning algorithms use a cost function for hyperedges, which is essentially the  $(K-1)$  metric. The cost of a hyperedge is the number of partitions,  $K$ , that it straddled, which therefore, would require  $K-1$  wires to join. These algorithms are more appropriate for mapping logic onto a fixed resource like an FPGA. For the proposed supply gating technique, the size of the partition is not important. The cost function has to be chosen such that it avoids leaving orphan vertices in partitions (which is not suitable for Shannon decomposition), so we selected a cost function as below, which considers a tunable parameter,  $vopt$ , to indicate the ideal number of vertices per partition:

$$cost(nvert) = (vopt / nvert)^2 + (nvert / vopt)^2 \quad (4)$$

For the hyperedge function, the original  $(K-1)$  metric was scaled down by  $vopt$ , which is the number of vertices in a partition and multiplied by a new parameter,  $kvpe$ , which is designed to be the cost of a vertex relative to a hyperedge. The hyperedge cost is:

$$cost = kvpe \cdot (K - 1) / vopt \quad (5)$$

This leaves a cost per edge, which summed over all edges is a unitless result. As with the cost of vertices, neither has any units of vertices or hyperedges, so they can be summed together to get the overall cost of a partitioning. The hyperedge cost function is still dependent on the assumption that the number of vertices and cut hyperedges are related independent of partition size. But what if cut hyperedges go as the square root of vertices? It would still be possible to get scale-independent values of  $kvpe$  and  $vopt$  although it would change the meaning of  $kvpe$ . And there were some tests that suggested the relationship was linear.

### 3.4 Annealing Algorithm

The partitioning algorithm was improved with the inclusion of an *annealing strategy* that helps to shift the local cost minima towards a globally optimal solution. The refinement process still consists of a series of trials of selecting a vertex at random and evaluating the cost of moving it to each other partition. The decision is whether to leave the vertex in its original partition or to move it to the best alternative partition.

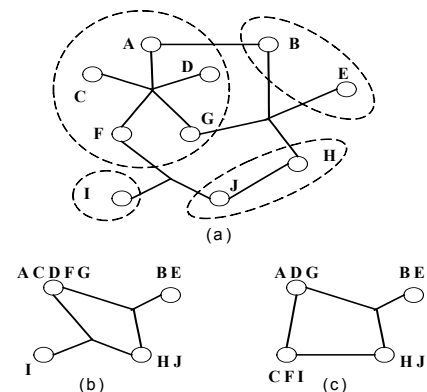


Figure 5: Hypergraph partitioning example.

An important property of the strategy implemented is that it does not require any adjustable parameters, except the number of iterations to run the refinement process. The strategy scales the cost of the alternative partitioning relative to the current one. A negative result represents a state that is always accepted. A positive result is given further consideration. Thus, a data point  $x$  depends on the cost of the current partitioning,  $curr_{cost}$ , and the cost of the alternative,  $new_{cost}$ :

$$x = (new_{cost} - curr_{cost}) / curr_{cost} \quad (6)$$

The  $x$  values have a mean of zero, but some measurement of the variability is needed so the strategy can avoid selecting very high cost configurations. A weighted average of the squared values is used as the variance, so that the weight of older values decays exponentially. The choice of the weighting factor determines how long older values have significance. The reciprocal of this weighting factor is the number of iterations before the significance of a value has diminished to 37% of its original amount. That was selected to be the square root of the number of iterations for the entire refinement process. The variance  $var$  depends on the weighting factor  $delta$  and the previous value of the variance  $oldvar$ .  $delta$  depends on the number of iterations  $ntries$ :

$$delta = 1 / \sqrt{ntries} \quad (7)$$

$$var = delta \cdot (x^2 - oldvar) + oldvar$$

The values are scaled by the square root of the variance, producing a z-score:

$$zscore = x / \sqrt{var} \quad (8)$$

The probability density function for a normal distribution is used to relate the z-score with the probability that the higher cost alternative will be accepted. The inverse cumulative Gaussian distribution function, probit, can be approximated by a straight line with slope  $\sqrt{2\pi} \approx 2.5$  so that at 0.5, the error is zero and at 0.75, the error is 7%. Using a distribution that is only the positive half of a normal distribution, this constant becomes 1.25.

So the wanted probability,  $prob$ , depends on the total number of iterations,  $ntries$ , and the current iteration number,  $n$ . The value of the probit function,  $probit$ , depends on  $prob$ :

$$prob = 0.5 \cdot (ntries - n) / ntries \quad (9)$$

$$probit = 1.25 \cdot prob$$

The decision to accept the alternative partitioning is made by the following expression. The alternative is accepted when the expression is true. The right-hand side gives the probability that the partitioning with the z-score should be accepted given the average probability,  $prob$ , and its probit function value,  $probit$ . The left-hand side of the expression is a random number,  $0 \leq r < 1$ :

$$r < prob \frac{zscore}{probit} \quad (10)$$

## 4. RESULTS

We implemented the entire synthesis flow in a program using C++. A set of simulations was carried out with ISCAS-85 [7] benchmarks to determine the improvement in power, area and delay for the proposed approach.

### 4.1 Test Setup

The program implemented for the verification of the proposed approach reads the original circuit in BLIF [1] format and produces a SPICE netlist for both the original circuit as well as supply-gated one. The SPICE netlist was used to estimate the total active power. Baseline conditions for power simulations included predictive 32 nm transistor technology [8], 0.9 V power supply and 110°C temperature. The outputs were loaded with fan-out-of-four (FO4) inverters and the clock cycle time was 5 ns (200 MHz frequency). Input signals consisted of 1000 weighted random vectors per test

(100 for *c6288*) with 20% input activity and 20 ps rise and fall times. The SIS logic synthesis tool [1] was used for synthesis of both original circuit and the Shannon decomposed partitions. Power dissipation for the synthesized circuits was measured using LTSPICE circuit simulator. Area results were computed by the program. Delay was measured using Synopsys Design Compiler. The percent change in a parameter was reported according to the following formula:

$$\% \Delta x = (x_{org} - x_{mod}) / x_{org} \cdot 100\% \quad (11)$$

Thus, a positive number indicate improvements, whereas a negative number indicates degradation.

### 4.2 Test Results

To select the optimal vertices for partitioning, we noted the trend in power saving with partition size for each benchmark by running the partitioning tool for 31 different  $vopt$  values and obtained the  $vopt$  versus power saving plot. From this plot, we

**Table 1: Power, area, and delay for optimal partition sizes (32nm, 0.9V, 110°C, 200MHz)**

Circuit	Power (%)	Area (%)	Delay (%)	$Vopt$
c432	19.8	-5.4	-1.5	11
c880	22.2	1.9	9.6	15
c1908	25.6	16.5	2.8	10
c499	20.5	23.7	-9.8	6
c1355	20.5	22.8	-7.6	7
c2670	15.2	5.3	11.8	30
c5315	4.8	4.8	5.0	16
c7552	5.4	-4.7	-10.3	22
c6288	42.0	16.0	10.7	5

**Table 2: Recursive Shannon decomposition (32nm, 0.9V, 110°C, 200MHz)**

Circuit	Power (%)	Area (%)	Delay (%)	Total partitions	# of Shannon decomposed partitions		Execution Time (sec.)
					L = 1	L = 2	
c432	18.7	-6.8	3.4	9	9	13	220.77
c880	23.6	2.3	13.7	22	22	29	38.78
c1908	27.7	24.3	9.0	52	51	38	50.11
c499	26.6	34.6	4.0	57	57	37	44.95
c1355	27.0	35.7	0.6	61	58	34	46.88
c2670	17.4	12.0	19.1	81	73	62	129.52
c5315	5.3	6.0	13.8	108	105	140	335.37
c7552	7.8	-3.2	-2.8	97	89	139	434.62
c6288	40.4	14.7	9.1	469	427	87	613.46

**Table 3: Comparison of power reduction across different technology nodes (110°C, 200MHz)**

Circuit	Power Reduction (%)		
	Technology node		
	32nm	45nm	65nm
c432	18.7	10.5	4.3
c880	23.6	18.6	14.8
c1908	27.7	21.2	18.0
c499	26.6	20.3	16.6
c1355	27.0	21.9	19.5
c2670	17.4	13.8	8.8
c5315	5.3	2.8	0.5
c7552	7.8	3.1	1.2
c6288	40.4	42.8	43.2

**Table 4: Comparison of power reduction at different operating frequencies (32nm, 0.9V, 110°C)**

Circuit	Power Reduction (%)			
	Cycle Time (ns)			
	2	5	10	1000
c432	9.3	<b>18.7</b>	23.7	<b>31.5</b>
c880	18.8	<b>23.6</b>	26.5	<b>30.1</b>
c1908	21.8	<b>27.7</b>	31.2	<b>36.6</b>
c499	19.9	<b>26.6</b>	30.3	<b>35.5</b>
c1355	22.2	<b>27.0</b>	29.7	<b>33.8</b>
c2670	8.7	<b>17.4</b>	20.4	<b>22.8</b>
c5315	4.6	<b>5.3</b>	11.0	<b>15.1</b>
c7552	4.5	<b>7.8</b>	10.1	<b>14.1</b>
c6288	44.1	<b>40.4</b>	36.9	<b>25.6</b>

obtained the best *vopt* values by fitting a curve. Table 1 gives power, area and delay results for the optimal number of vertices per partition for each benchmark circuit. As the results suggest, the proposed method achieves a significant saving (up to 42%) in power for the benchmark circuits. Note that Shannon decomposition can result in improved delay in some cases due to reduced load at the intermediate nodes, as observed in [2]. The maximum area and delay overhead are within tolerable limits (<5%).

Table 2 gives results for two level recursive Shannon decomposition. In most cases, the power results were better compared to one-level decomposition. Overall, the power savings showed a 22% average improvement over the benchmarks considered. Table 2 also gives the total number of partitions, the number that were Shannon decomposed to one level, and the number of those that were Shannon decomposed recursively, to a second level. When a partition is Shannon decomposed, as many as three subcircuits are produced (Fig. 2(b)), which become candidates for a second level of Shannon decomposition, thereby increasing the number of candidates for second level decomposition compared to the first level one. Table 2 also presents execution time of the Shannon decomposition program, including time to run the logic synthesis using SIS. These runs were carried out using the GNU C++ compiled program on a Red Hat Linux system with a 1.4 GHz AMD Athlon processor. Execution time varied from 39 seconds to 10 minutes (dominated by SIS run). Total memory required for both programs was about 10 MB for the largest benchmark.

Table 3 shows the effects of technology scaling on the results obtained for the proposed algorithm using two other technology models [8]. These results show that the proposed technique is scalable and effectiveness of the technique increases with scaling. This is due to the fact that leakage power becomes the more important component of total active power below 65 nm. To analyze the impact of supply gating on active power reduction at different clock frequency, we also compared power consumption at four different clock frequencies. Table 4 reports the power reduction results for each frequency. For most circuits, with increasing frequency, Shannon decomposition saves less power because dynamic power becomes the dominant component in total active power. Note that the benchmark *c6288* shows a reverse trend. This is because in *c6288*, dynamic power saving dominates.

Table 4 also includes results for a cycle time of 1000 ns = 1  $\mu$ s, which is a measure of the total saving in static power, since at this frequency dynamic power is negligible. An important additional advantage of the proposed gating technique is that it not only reduces total active power, but it automatically reduces standby mode power dissipation as well. This is because, irrespective of the state of the inputs, only one of the cofactors will remain ON, while the others will remain supply-gated, thus saving leakage power.

It is worth noting that the active power reduction is a function

of input switching activity as well as activity of the control variable. In general lesser switching activity will increase the contribution of leakage power in total power and thus improve power saving. Although we have considered 20% input switching activity, in real circuits, it may be much lower. To investigate the switching activity of a large sequential benchmark, we computed average transitions at each internal node of an ISCAS-89 benchmark *s35932* (~18K gates) for 20% activity at input. Our simulations show that about 65% nodes have an average transition less than 10% and about 57% nodes have average transition less than 5%. Thus, the proposed technique can result in increased power saving since the combinational logic blocks are likely to receive large number of inputs with activity well below 20%.

It can be noted that some of partitions in the test circuits are not allowed to undergo Shannon decomposition (as shown in Table 2) due to unacceptable overhead in delay or area, although they can render considerable improvement in power. This issue can be addressed by integrating the proposed approach with gate sizing to improve power saving. Simultaneous gate sizing and supply gating can leverage on the fact that a partition that cannot be subjected to supply gating due to delay violation can be upsized to compensate for any delay increase. Similarly, a partition can be downsized to accommodate an area increase if it has available delay slack.

## 5. CONCLUSION

We have presented a fine-grained supply gating methodology for large and complex logic circuits to reduce active leakage and switching power. We have shown that with good partitioning, efficient Shannon decomposition, and extraction of shared logic, it is possible to significantly reduce power consumption in scaled technologies with little impact on area and performance. The proposed technique becomes more important as leakage claims a greater share of power consumption with transistor threshold voltage scaling. The technique can be completely automated and can be integrated with existing logic synthesis tools. Along with active power reduction, as a by-product, it also achieves up to 37% automatic reduction in standby power. Finally, the technique can be combined with other low power design techniques such as gate sizing or supply voltage scaling to improve active power reduction.

## References

- [1] E.M. Sentovich et al., "SIS: A System for Sequential Circuit Synthesis", *Memorandum No. UCB/ERL M92/41*, UCB, 1992.
- [2] S. Bhunia et al., "A novel synthesis approach for active leakage power reduction using dynamic supply gating", *DAC*, 2005.
- [3] G. Karypis et al., "Multilevel hypergraph partitioning: applications in VLSI domain", *IEEE TVLSI*, 1999, pp: 69-79.
- [4] A. Agarwal et al., "Effectiveness of Low Power Dual-Vt Designs in Nano-Scale Technologies under Process Parameter Variations", *ISLPED*, 2005.
- [5] J.W. Tschanz et al., "Dynamic Sleep Transistor and Body Bias for Active Leakage Power Control of Microprocessors", *JSSCC*, 2003.
- [6] D. Sylvester and H. Kaul, "Power-Driven Challenges in Nanometer Design", *IEEE Design and Test of Computers*, 2001.
- [7] M. Hansen et al., "Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering", *IEEE Design and Test*, pp. 72-80, 1999.
- [8] <http://www.eas.asu.edu/~ptm>, University of Arizona, Tempe.
- [9] M.R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", *CA:Freeman*, 1979.
- [10] B.H. Calhoun, F.A. Honore, and A.P. Chandrakasan, "A leakage reduction methodology for distributed MTCMOS", *IEEE JSSC*, 2004.
- [11] A. Abdollahi, F. Fallah, and M. Pedram, "An Effective Power Mode Transition Technique in MTCMOS Circuits", *DAC*, 2005.
- [12] H.S. Won et al., "An MTCMOS design methodology and its application to mobile computing", *ISLPED*, 2003.