System Partitioning to Maximize Sleep Time [1]

Amir H. Farrahi *Majid Sarrafzadeh*

Department of Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208, USA

Abstract

Partitioning of a system to maximize exploitable sleep time for low-power synthesis is discussed. The motivation is to deactivate the memory refresh circuitry, apply power down or disable the clock signals during the inactive periods of operation of circuit elements, and thus minimize the power consumption. Since it is impractical to have a separate set of control signals for each circuit element (otherwise, the control itself would consume a lot of power), it is advisable to partition a circuit based on the activity patterns of its elements so that the partitions can be switched into sleep mode for long periods of time. In this paper, we formulate this partitioning problem and show that it is NP-hard. We present **Geo_Part** *, a geometric partitioning heuristic for this problem. An efficient implementation of* **Geo_Part** *using segment tree data structure is discussed. Experimental results are encouraging.*

1 Introduction

Not so long ago, the main objectives in the automation process of designing VLSI chips were: 1. increased processing speed, 2. reduced chip area, and 3. testability. With the modern advances in VLSI and packaging technologies, the average transistor count has shown an increase rate of about one hundred-fold per decade [3], allowing much more complex functionality per chip. The minimization of the average power consumption in modern VLSI circuits is an emerging objective of utmost importance due to a number of reasons, including: longer battery life for portable communication and computing appliances, heat dissipation bottleneck in highly integrated circuits [3], and environmental issues [2]. Because of the importance of the power consumption issue, there has been considerable shift of attention in logic and layout synthesis areas [15, 16] and more recently in high-level synthesis [4, 10] from the delay and area minimization issues towards low power design.

There are three sources of power consumption in CMOS circuits: the charging and discharging of capacitive loads during switchings at gate outputs, the short circuit current which flows during output transitions, and the leakage current. The last two sources should be dealt with and optimized using proper device and circuit design techniques, hence the design automation community has focused on the minimization of the first source, which is frequently referred to as the *switching power* or *dynamic power*. Transition density or average switching rate at different sites in a circuit is introduced in [11] as a quantity to measure the circuit activity, which can be used to estimate the average dynamic power consumption in a digital circuit.

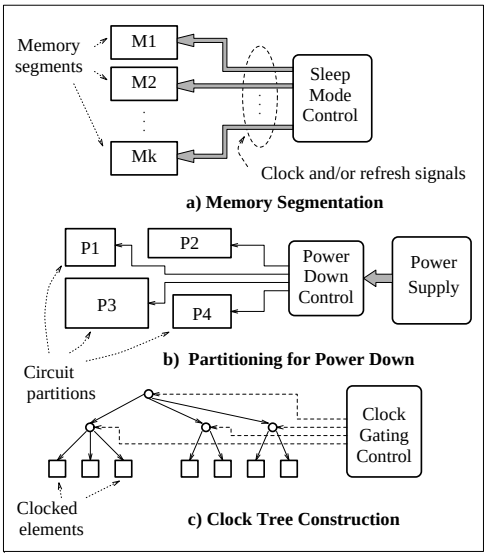

Figure 1: Circuit partitioning to exploit sleep mode

Previous research for low power synthesis of digital circuits has focused on issues such as activity-driven technology decomposition and mapping [15], low power state assignment [8], architectural transformation and reduction of power supply voltage [4], wire and driver sizing [5], and reversible and adiabatic computing [6]. In this paper we study the partitioning problem to exploit sleep mode operation for power minimization in digital circuits. In a general setting, this problem can be viewed as partitioning a set of circuit elements such that the savings in power consumption achieved by switching each partition as a whole into sleep is maximized. Placing a partition into sleep mode assumes different meanings in each setting of the problem. The problem finds many applications in low power design, e.g., memory segmentation, partitioning to power-down portions of the design, and activity-driven clock tree design (see Figure 1).

In this paper we formulate the partitioning problem to maximize sleep time, while at the same time controlling the total number of switchings of the partitions into and out of sleep mode and show that the problem is NP-hard. Because of the geometric nature of the problem, direct application of graph partitioning schemes to this partitioning problem is not possible. We present **Geo_Part** , a geometric algorithm based on iterative improvement techniques [9] for this problem. We present an efficient implementation of this algorithm using the segment tree data structure [12], and discuss its time complexity. We also discuss upper bounds on total achievable sleep time for a given instance of the problem. Experiments for memory segmentation were conducted on a number of sorting, matrix multiplication, and DSP algorithms to show

the effectiveness of our approach.

2 Problem Formulation

Let $M = \{m_1, m_2, ..., m_r\}$ represent the set of circuit elements (CEs). We say that CE m is *idle* during time interval $I = (l, r)$, $l < r$, if m can be switched into sleep mode during I. Intervals $I_1 = (l_1, r_1)$ and $I_2 = (l_2, r_2)$ are *non-overlapping* if $l_1 \geq r_2$ or $l_2 \geq r_1$. A set of intervals are non-overlapping if they are pairwise non-overlapping. The *idle set* N_m of m consists of a set of non-overlapping intervals or an NIS (Non-overlapping Interval Set) during all of which m is idle. We assume that the idle sets of elements in M are given as a set $S = \{N_1, N_2, ..., N_r\}$, where $N_i = \{I_{i_1}, I_{i_2}, ..., I_{i_{n_i}}\}$ represents the idle set of m_i (see Figure 2). An *empty* interval is denoted by $()$. Given intervals $I_1 = (l_1, r_1)$ and $I_2 = (l_2, r_2)$, we say I_1 *covers* I_2 if either $l_1 \leq l_2 < r_2 \leq r_1$, or $I_2 = ()$. The *length* $L(I)$ of an interval $I = (l, r)$ is defined as the quantity $r - l$ (or 0 if $I = ()$). The *intersection* of two intervals I_1 and I_2, denoted as $I_1 \wedge I_2$, is defined as the longest interval covered by both I_1 and I_2 (or empty if the two intervals do not overlap). The intersection of more than two intervals is defined similarly. The intersection of two NISs $N_1 = \{I_{11}, I_{12}, ..., I_{1n_1}\}$ and $N_2 = \{I_{21}, I_{22}, ..., I_{2n_2}\}$, denoted as $N_1 \wedge N_2$, is defined as the NIS formed of the non-empty pairwise intersections of the intervals one picked from N_1 and the other picked from N_2, that is:

$$N_1 \wedge N_2 = \{I_1 \wedge I_2 \mid I_1 \in N_1,\ I_2 \in N_2,\ I_1 \wedge I_2 \neq ()\} \qquad (1)$$

The intersection of more than two NISs is defined similarly. The *endpoint set* E_N of NIS N is defined as the set of endpoints of the intervals in N, that is: $E_N = \{p \mid \exists q : (p, q) \in N \text{ or } (q, p) \in N\}$. The *duration* $D(N)$ of a NIS $N = \{I_1, I_2, ..., I_k\}$ is defined as the sum of the lengths of the intervals contained in it, that is: $D(N) = \sum_{i=1}^{k} L(I_i)$. Given a set $S = \{N_1, N_2, ..., N_k\}$ of NISs, the *internal intersection* $A(S)$ of S is defined as the intersection of all the NISs in S, that is: $A(S) = \wedge_{N_i \in S} N_i$. The endpoint set E_S of S is defined as the union of the endpoint sets of the NISs in S, that is: $E_S = \{p \mid \exists N \in S : p \in E_N\}$. Given a set S, (S_1, S_2) is a *bi-partitioning* for S if: $S_1, S_2 \subset S$, $S_1 \cap S_2 = \phi$ and $S_1 \cup S_2 = S$. The bi-partitioning (S_1, S_2) is *b-balanced* if $|S_1| \geq b$ and $|S_2| \geq b$, where the notation $|S_i|$ denotes the cardinality of set S_i, and is called the *size* of partition S_i. Note that a bi-partitioning (S_1, S_2) of S defines a corresponding bi-partitioning (M_1, M_2) of M, and vice versa, where $M_i = \{m_j \mid N_j \in S_i\}$. That is M_i is the set of CEs whose idle sets are partitioned into S_i. The *density* of a partition S_i at a given point p, is the number of NISs in S_i that contain some interval containing p. The gain $G(S_1, S_2)$ of b-balanced bi-partitioning (S_1, S_2) of S is defined as: $G(S_1, S_2) = f(t_1, t_2, sw_1, sw_2)$ where $t_i = D(A(S_i))$ and $sw_i = |A(S_i)|$. are referred to as the *sleep times*, and the *switchings* of partitions S_1 and S_2, respectively. Note that the internal intersection of a partition S_i of S, can be thought of as the idle set of the corresponding partition M_i of M, i.e., the set of maximal intervals during which all the CEs in M_i are idle. Hence, we will use the terms idle set and internal intersection of a partition interchangeably in the rest of this paper. Any interval in the idle set of a partition is an *idle interval* or *sleep interval* of that partition.

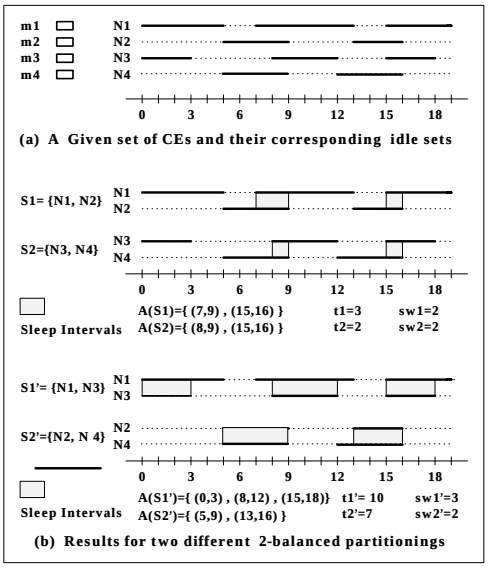

Figure 2: Partitioning to maximize sleep time

In general, the function f, describing the gain, should be an increasing function of t_1 and t_2, and a decreasing function of sw_1 and sw_2, e.g., $f = t_1 + t_2 - a \times (sw_1 + sw_2)$. In this definition, the term $t_1 + t_2$ accounts for the savings in power consumption due to sleep mode operation of partitions S_1, S_2, and the term $a \times (sw_1 + sw_2)$ accounts for the overhead in power consumption resulted from the extra control circuitry needed to supervise sleep mode operation. The parameter a is introduced to control relative significance of saving vs. overhead terms. We can now formulate our partitioning problem **STMP** (Sleep Time Maximization Problem):

- **Instance:** Ordered pair (S, b), where $S = \{N_1, N_2, ..., N_r\}$ is a set of NISs representing the idle sets of a set of CEs, and b is a positive integer.
- **Objective:** Find a b-balanced bi-partitioning (S_1, S_2) of S that maximizes the bi-partitioning gain $G(S_1, S_2)$.

Note that **STMP** is formulated as an optimization problem here. Using a transformation from min-cut graph partitioning problem [7] we can show that the decision version of **STMP** is NP-complete. We state this result as the following theorem(proof omited for brevity):

Theorem 1. *The problem* **STMP** *is NP-hard.*

3 An Iterative Improvement Heuristic

Iterative improvement heuristic is a widely used technique in graph partitioning algorithms [9]. Generally, the heuristic starts with a random or other initial balanced partition and then goes through a series of iterations to gradually improve the partitioning solution. In each iteration, the algorithm keeps moving one or more vertices from one partition to the other. The vertices to be moved are chosen such that the partitioning gain is maximized. The partitioning gain is defined as the net reduction in the number of edges connecting a vertex in one partition to a vertex in the other partition, that is the reduction in the number of *cut* edges. Once a vertex is moved, it is locked in the destination partition, and will not be considered for future moves in the current iteration.

An iteration stops when all vertices are locked in their partitions. At the beginning of each new iteration, all vertices are unlocked. The algorithm continues running one iteration after another until there is no improvement in an entire iteration. As the algorithm proceeds, the best visited partitioning solution is recorded, and it is reported at the end as the final solution. We can employ similar technique, having in mind that the vertices are replaced by NISs and the gain of a bi-partitioning is as defined earlier. The geometric flavor of the algorithm, however, makes the choice of best possible move more complicated. We call the resulting algorithm **Geo_Part** .

4 A Segment Tree Implementation

In order to obtain a fast implementation we should be able to try a tentative move and compute its resulting gain efficiently. To do so, we have to devise a mechanism that allows updating (or computing from scratch) the values of t_1, t_2, sw_1, sw_2 after each tentative move, in an efficient way.

Segment tree (ST) data structure [12] was introduced as a data structure to handle intervals on an axis, whose endpoints belong to a fixed set of points, called the *endpoint-set* of the ST. An important property of the ST data structure is that it allows representation if each interval using at most $O(logP)$ *canonical* intervals, where P is the size of the endpoint set of the ST, furthermore insertion and deletion of an interval into a ST takes $O(logP)$ time.

An slight modification of the ST data structure can be used to maintain each of our partitions. The endpoint-set of the STs for each partition would be the same as the endpoint-set of the **STMP** instance. To add an NIS to a partition, we will insert all its intervals into the ST, and to delete an NIS from a partition, we will delete all its intervals from the ST. Such implementation simplifies the tasks of calculating the sleep time and switching attributes of the partitions once an NIS is added or removed from a partition. An involved analysis of the running time of **Geo_Part** using the proposed ST data structure leads to the following results (the proofs and discussions are omited for brevity):

Theorem 2. *Each tentative move of* **Geo_Part** *can be done in $O((sw + |N_i|)\ log\ P)$ time, where sw is the maximum among the switchings of the partitions before or after the move, and $|N_i|$ is the size of the tentatively moved NIS.*

Theorem 3. *Each iteration of* **Geo_Part** *can be implemented in $O((|S|^2\ sw + n|S|)\ log\ P)$ time, where sw is the maximum switching of either partition encountered during the course of the iteration and n is the total number of intervals in the given* **STMP** *instance.*

5 Upper Bounds on Total Sleep Time

In order to have a feeling how far we are from the optimal solution, we need some reliable, yet realistic upper bounds on the total achievable sleep time. In this section we present two upper bounds on the total sleep time of the partitions in any b-balanced bi-partitioning of a given **STMP** instance. Note that the minimum of these is itself a valid upper bound. Consider a given **STMP** instance (S, b). Note that if $|S| < 2b$, the upper bound on the total sleep time of each b-balanced bi-partitioning of S would trivially be zero. Therefore we assume $|S| \geq 2b$.

Lemma 1. *Let $D_i, i = 1 \ldots |S|$ represent the duration of N_i, and let $D_1 \leq D_2 \leq \ldots \leq D_{|S|}$. Then $U_1(S, b) = D_1 + D_{|S|-(b-1)}$ is an upper bound on the total sleep time of each b-balanced bi-partitioning of S.*

Lemma 2. *Let N_0, N_1, N_2 represent the NISs during which the density of S is less than b, at least b but less than $2b$, and at least $2b$, respectively. Then $U_2 = D(N_1) + 2D(N_2)$ is an upper bound on the sleep time of a b-balanced bi-partitioning of S.*

6 Application to Memory Segmentation

We say that a DRAM cell m is idle during interval $I = (l, r)$ if m need not be refreshed during I. Let $M = \{m_1, m_2, ..., m_r\}$ represent the set of memory elements (MEs) used in an application. Assume that the access sequence for each ME $m \in M$ during a whole run cycle is given as a sequence of ordered pairs, each of the form (n_i, A_i), where n_i corresponds to the access time, and $A_i \in \{R, W\}$ represents the type of access, read (R), or write (W). Given the access sequence for all the MEs, and our definition for an idle interval, we realize that, a DRAM cell is idle:

- After its final access time,
- Before each write access until the closest read access (or the start of computation)

Using these two rules we can calculate the idle set for each memory element form its access pattern.

7 Experimental Results

We implemented **Geo_Part** using C and tested its performance on a number of test cases. Two matrix multiplication algorithms (mtrx_mult1, mtrx_mult2), a number of sorting algorithms (hpsrt, qsrt, shellsrt) and a set of DSP routines (four1, realft, avevar, moment) were selected from [13] as test cases for memory segmentation in low power synthesis of ASIC designs. To generate the idle sets for the memory elements, we used a profiling tool that provides the utilization of the registers, memory, and various computational units of a processor executing a program. The idle sets were generated from the utilization information provided by the profiling tool, using the methodology presented in Section 6.

In order to estimate the percentage of savings in power consumption, R, using our partitioning technique in memory segmentation we used the data sheets for a number of DRAM memories [1]. Note that a partition runs in standby mode, consuming significantly lower power, when switched into sleep mode. Let P, P' represent the power consumption with and without exploiting sleep mode using our partitioning technique, then using $Power = \frac{Energy\ \ consumption}{Time}$, and ignoring the leakage current and the power consumption due to the sleep mode control unit, we have:

$$P = \frac{P_o\,[2T - (t_1 + t_2)] + P_s(t_1 + t_2)}{T}$$

$$P' = \frac{[(P_oT)\ +\ (P_oT)]}{T} = 2P_o$$

where P_o and P_s represent the power consumption of each partition of the memory in operation and standby modes,

Test Case	$\|S\|$	T	#Iter.	%Power saving	Upper bound
avevar	1007	152929	5	**34.27**	36.43
banmul	281	325224	11	**60.48**	61.49
four16	53	110715	3	**24.19**	47.57
hpsrt	403	489393	2	**19.01**	21.02
moment	1215	73913	5	**19.49**	25.58
mtrx_mult1	97	37240	5	**8.54**	16.94
mtrx_mult2	97	37809	3	**10.46**	12.96
qsrt	205	244870	3	**21.26**	22.22
realft	53	111143	2	**3.55**	46.99
shellsrt	403	433286	2	**7.97**	13.34
Average				**20.02**	30.54

Table 1: Results for power minimization in memory unit

respectively, and T is the computation time. The percentage of saving in the power consumption, R, will then be given by:

$$R = 100 \times \frac{P' - P}{P'} = 100 \times \frac{(t_1 + t_2)(P_o - P_s)}{P_o} \qquad (2)$$

The data sheets indicate that for a given memory chip, we typically have $\frac{P_o}{P_s} > 25$, therefore the percentage of saving in the power consumption using the proposed partitioning technique would be at least:

$$R_{min} = 48 \times \frac{t_1 + t_2}{T} \qquad (3)$$

Table 1 summarizes the results of our experiments. The first column is the name of the test case. The next two columns represent the statistics for each test case, that is the memory size, and the computation time (in cycles) of the algorithm. The fourth column represents the number of iterations in **Geo_Part** algorithm on each test case. The fifth column represents the percentage of savings $R_{min} = 48 \times \frac{t_1+t_2}{T}$ in power consumption using our partitioning approach, and the last column shows the upper bound for the percentage of savings in power consumption computed as $min\{R_{U_1}, R_{U_2}\}$, where $R_{U_i} = 48 \times \frac{U_i}{T}$ are the normalized values for the upper bounds we obtained in Section 5. It is notable that for most test cases, our algorithm follows the upper bound on the sleep time quite closely. This means that our results are very close to the optimal. In the other cases, where our result achieves sleep time that is far from the given upper bound, an explanation maybe because the upper bounds themselves are not tight enough. The estimated savings on the power consumption using our technique ranges from 3.55% to 60.48%, averaging 20.02%, which is quite encouraging.

8 Conclusion

In this paper we studied system partitioning problem to maximize sleep time, which can be exploited to minimize the power consumption. The motivation is to de-activate the memory refresh circuitry, apply power down or just disable the clock signals during the inactive periods of operation of circuit elements. We formulated this partitioning problem in this paper and showed that it is NP-hard. We also presented a geometric heuristic algorithm based on the iterative improvement scheme for this problem, and discussed an efficient implementation of this algorithm using a segment tree data structure to maintain each partition. We conducted experiments for memory segmentation in low power synthesis of ASICs, on a number of numerical and DSP algorithms. The experiments indicate that the estimated power consumption due to the memory unit is decreased from 3.55 to 60.48 percent, with an average of 20.02 percent. Our future research in this area will be in the following directions: *i)* achieve tighter upper bounds, *ii)* design fast sleep time estimation algorithms to provide quick feedback to scheduling and allocation tasks in compilers and high-level synthesis. This can be used to perform scheduling and allocation targeted for higher sleep time, which could be exploited to reduce power consumption, *iii)* study the more general problem of multi-way partitioning, in which the algorithm should compute the optimal number of partitions as well as the contents of each partition, *iv)* experiment with other applications. In some applications the connectivity of the circuit elements should also be taken into account when doing the partitioning. Partitioning algorithms should be developed to allow trade-offs between routing area, delay, and power consumption.

9 Acknowledgement

The authors would like to thank Jian Chen at EECS Department, Northwestern University for his great cooperation in extracting the idle sets for the test cases.

References

[1] "Dynamic RAMs and Memory Modules". Motorola Inc., Phoenix, AZ, 1993.

[2] In International Workshop on Low Power Design, April 1994.

[3] H. B. Bakoglu. "Circuits, Interconnections, and Packaging for VLSI". Addison-Wesley Publishing Co., 1990.

[4] A. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. Broderson. "Optimizing Power Using Transformations". IEEE Transactions on Computer Aided Design, 14(1):12–31, January 1995.

[5] J. Cong, C. Koh, and K. Leung. "Wiresizing with Driver Sizing for Performance and Power Optimization". In International Workshop on Low Power Design, pages 81–86, April 1994.

[6] J.S. Denker, S.C. Avery, A.G. Dickinson, A. Kramer, and T.R. Wik. "Adiabatic Computing with the 2N-2N2D Logic Family". In International Workshop on Low Power Design, pages 183–186, April 1994.

[7] M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP–completeness. Freeman, 1979.

[8] G. Hachtel, M. Hermida, A. Pardo, M. Poncino, and F. Somenzi. "Re-Encoding Sequential Circuits to Reduce Power Dissipation". In International Conference on Computer-Aided Design, pages 70–73, 1994.

[9] S. Hauck and G. Boriello. "An Evaluation of Bipartitioning Techniques". In Chapel Hill Conference on Advanced Research in VLSI, 1995.

[10] R. Mehra and J. Rabaey. "Behavioral Level Power Estimation and Exploration". In International Workshop on Low Power Design, pages 197–202. IEEE/ACM, 1994.

[11] F. Najm. "Transition Density: A New Measure of Activity in Digital Circuits". IEEE Transactions on Computer Aided Design, 12(2):310–323, 1992.

[12] F. P. Preparata and M. I. Shamos. Computational Geometry: An Introduction. Springer-Verlag, 1985.

[13] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. Numerical Recipes in C. Cambridge University Press, 1988.

[14] V. Tiwari, P. Ashar, and S. Malik. "Technology Mapping for Low Power". In Design Automation Conference, pages 74–79. ACM/IEEE, 1993.

[15] C. Tsui, M. Pedram, and A.M. Despain. "Technology Decomposition and Mapping Targeting Low Power Dissipation". In Design Automation Conference, pages 68–73. ACM/IEEE, 1993.

[16] H. Vaishnav and M. Pedram. "A Performance Driven Placement Algorithm for Low Power Designs". In EURO-DAC, 1993.

[17] S.G. Younis and Jr. T.F. Knight. "Asymptotically Zero Energy Split-Level Charge Recovery Logic". In International Workshop on Low Power Design, pages 177–182, April 1994.