Memory Segmentation to Exploit Sleep Mode Operation

Amir H. Farrahi *Gustavo E. Téllez* *Majid Sarrafzadeh*

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

Abstract

Sleep mode operation and exploiting it to minimize the average power consumption are of great importance. In this paper, we formulate the memory segmentation/partitioning problem to exploit sleep mode operation and show that the problem is NP-complete. We present polynomial time algorithms for special classes of the problem. Some generalizations of the problem are discussed. Preliminary experiments are conducted to show the effectiveness of the algorithms and applicability of the approach. The experimental data confirm that a careful partitioning allows up to 40% more sleep time which could be exploited to minimize the average power consumption. Directions for further research in this area are presented.

1 Introduction

As a result of advances in VLSI technology and the advent of portable and mobile communication and computing services, the minimization of power consumption in modern circuits is of utmost importance. Due to this importance, there has been considerable shift of attention in the logic and layout synthesis areas [11, 13, 12, 9] and more recently in high-level synthesis [3, 2, 8] from the delay and area minimization issues towards low power design. Recent studies [1] indicate that the clock signal and memory unit in digital computers, each consumes somewhere between 15 to 45 percent of the total power. This suggests good opportunities for savings in power consumption due to these sources. Exploiting sleep mode operation is an attempt to do so.

In general, the term *sleep mode* refers to the mode in which there is no activity in part(s) of the system during certain periods of time. The sleep mode issue can be studied at different fronts, e.g., behavioral level, register-transfer level (RTL), logic level and transistor level.

32nd ACM/IEEE Design Automation Conference ®

In this paper we study the partitioning [1] problem to exploit sleep mode operation for power minimization in digital circuits. The general 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 mode is maximized. A partition can be switched into sleep mode during time interval $I = (l, r)$ if all the elements in that partition are *idle* during I. The set of intervals during which an element m is idle, is referred to as the *idle set* of m. We present a general formulation for this partitioning problem and study its complexity. The problem finds many applications in low power design, e.g. the following (see Figure 1): memory segmentation, partitioning to power-down portions of the design, clock tree construction. We briefly describe how to obtain the information on idle

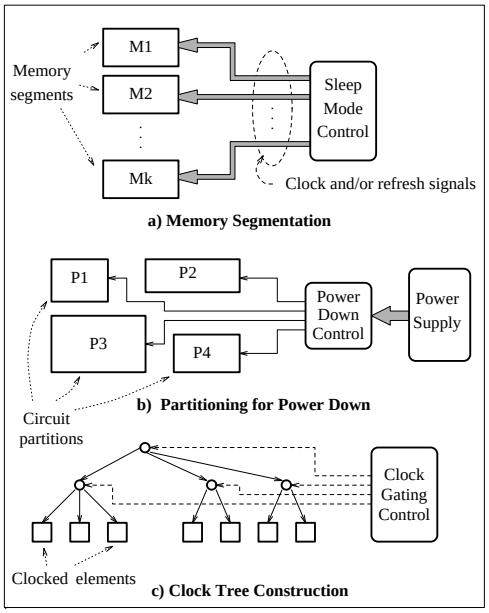

Figure 1: Circuit partitioning to exploit sleep mode

sets, later in the paper. The rest of this paper is organized as follows: Section 2 presents the necessary background. Section 3 briefly describes a methodology to obtain the the idle sets for a set of memory or clocked element in a design (i.e., the data needed for this problem). Section 4 describes the problem formulation. The complexity of the problem is discussed in Section 5. Exact algorithms to solve the general

[1] The terms *segmentation* and *partitioning* are used interchangeably in this paper.

problem are presented in Section 6. Some special classes of the problem are studied in Section 7 and polynomial time algorithms are presented to solve these classes. Section 8 focuses on some generalization of the problem. Preliminary experimental results for the problem are provided in Section 9, and Section 10 concludes the paper summarizing the key features of this study and provides directions for further research in this area.

2 Background

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 [14], 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*. The average dynamic power consumption for a CMOS gate g with load capacitance C_g is given by:

$$P_{av}(g) = 0.5\ C_g\ V_{dd}^2\ D(g), \qquad (1)$$

where $D(g)$ and V_{dd} represent the transition density [2] of the signal at the output of g, and the voltage of the power supply, respectively.

This suggests that a signal has a high contribution to the dynamic power consumption if it has either relatively large load capacitance or relatively high transition density. And these are both true about the clock signal in moderately sized synchronous digital systems. Now consider a scenario in which the access times to a set of dynamic memory elements are known. If we can partition these memory elements such that for long periods of time either of the partitions contains no data, then we can turn off the memory refresh circuitry for that partition during these periods and thus reduce the amount of power consumption. Recent studies [1] indicate that the clock signal and memory each consumes somewhere between 15 to 45 percent of the total power in digital computers. Hence, it would be worthwhile to study the mechanisms and approaches through which the power consumption due to these sources can be optimized. Exploiting sleep mode operation being one of these approaches. Our approach to exploiting sleep mode is to partition the memory such that memory elements (MEs) with similar activity patterns are grouped together so that the clock signals, refresh circuitry, and/or the power supply to each partition can be switched off (via gates or switches) during the periods of time that all the MEs in that partition are idle. Clearly, there is some overhead involved, caused by the extra control logic needed to switch the partitions in and out of sleep mode and the amount of power that switching in and out of sleep mode will consume. This overhead is mainly dependent on the switching pattern and switching frequency of the partitions in and out of sleep mode. The activity patterns for the MEs can be obtained through simultion of the real time operations of the circuit, or using statistical approaches applied at logic level. Given the idle sets of the MEs, the

[2] Average number of transitions per unit time.

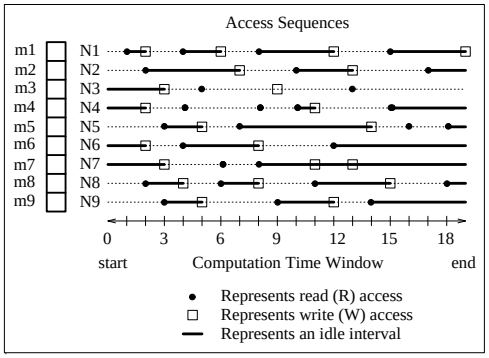

Figure 2: Idle sets for memory from the memory access sequence

question is how to partition the set of MEs to maximize the savings in power consumption achievable by this technique, and that how much power would this technique save us. We believe that there is a high potential of savings in the power consumption using this technique and our paper is an attempt to study this problem. In the next section we discuss how to obtain the idle sets.

3 Obtaining Idle Sets

In this section, we briefly describe methodologies to obtain the idle sets of the memory and clocked elements in a design. Availability of these activity patterns are vital for the partitioning algorithm to be applicable.

3.1 Idle Sets for Memory Elements

Let $M = \{m_1, m_2, ..., m_r\}$ represent the set of dynamic memory elements (MEs) in an application. Assume that the access sequence for each ME $m_i \in M$ during a whole run cycle is given as a sequence of ordered pairs each of the form (t_i, A_i), where t_i corresponds to the access time, and $A_i \in \{R, W\}$ represents the type of access, read (R), or write (W) (see Figure 2). To obtain the access sequence for the MEs, we can use simulation-based tools that take as input an application program and produce statistics on the resource utilization over time and space. Given the access sequence for all the MEs, we can use the following rules to generate, for each ME m_i, the set of intervals during which m_i does not need to be refreshed (see Figure 2), and thus obtain the idle sets for each ME:

- If the first access to m_i is a W access at time t, then there is no need to refresh m_i prior to time t.
- After the last access to m_i, there is no need to refresh m_i.
- If an access to m_i (of any type, R or W) at time t, is followed by a W access at time t', then there is no need to refresh m_i during time interval (t, t').

3.2 Idle Sets for Clocked Elements

Consider the description of a design after the scheduling and allocation steps have been performed. We assume that the functional units have registers at their input. This means that if an FU F is not used for a consecutive set of cyles, then we can gate the clock signal to the registers feeding this FU during this idle time, and thus reduce the power

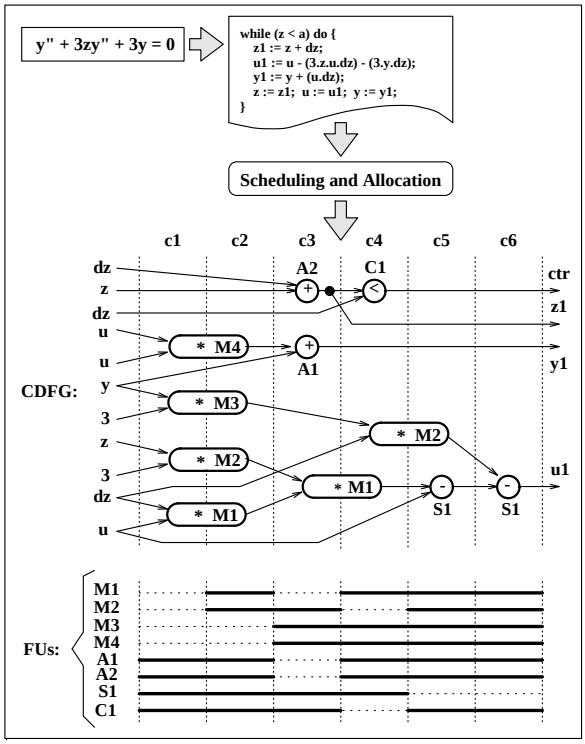

Figure 3: Idle sets for a scheduled and allocated design

consumption due to the clock tree. Furthermore, it guarantees that there would be no dynamic power consuming activity during this time in F. From the scheduled and allocated design we can say that if FU F is assigned to a control step c, then it is active during c. Otherwise, it is idle during this time. This allows us to generated the idle sets for each of the FUs in our design. Figure 3 shows how to obtain the idle sets from a design that solves a differential equation of the form $y'' + 3zy' + 3y = 0$ after scheduling and allocation have taken place. The design contains the following functional units: four multipliers $M1, M2, M3, M4$, two adders $A1, A2$, one subtractor $S1$, and one comparator $C1$. The idle sets for the registers at the inputs of each FU is computed from the *Control-Data Flow Graph* (CDFG) after scheduling and allocation are done. These idle sets are shown at the bottom next to their corresponding FU names. The clock gating problem, and the placement of gates in the clock tree are studied in [10]. A matching-based heuristic is presented for grouping the clocked elements to form the clock tree structure.

4 Formulation of The Problem

Consider a set $M = \{m_1, m_2, ..., m_r\}$ of MEs. We say that ME 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 NISes (Non-overlapping Interval Sets) during all of which m is idle. We assume that the information about the idle sets of elements in M are given as a set $S = \{N_1, N_2, ..., N_r\}$, where $N_i = \{I_{i_1}, I_{i_2}, ..., I_{i_{n_i}}\}$

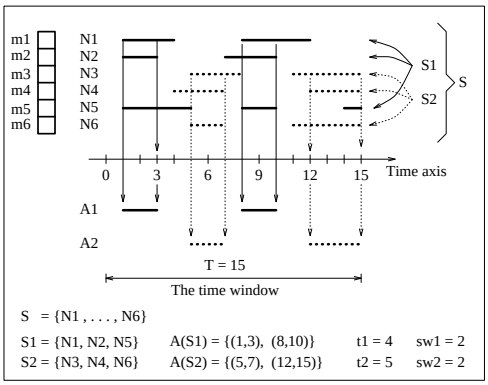

Figure 4: Memory partitioning for sleep mode

is a NIS representing the idle set of m_i (see Figure 4). The notation () denotes an *empty* interval. 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 $I_2 = ()$ (that is, all intervals cover the empty interval). 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 NISes $N_1 = \{I_{11}, I_{12}, ..., I_{1n_1}\}$ and $N_2 = \{I_{21}, I_{22}, ..., I_{2n_2}\}$, denoted as $N_1 \wedge N_2$, is defined as the NIS formed by the pairwise intersection of the intervals one picked from N_1 and the other picked from N_2, that is:

$$N_1 \wedge N_2 = \{I_1 \wedge I_2 \mid I_1 \in N_1,\ I_2 \in N_2,\ I \neq ()\} \quad (2)$$

The intersection of more than two NISes is defined similarly. Figure 4 shows NISes $N_1, N_2, N_3, N_4, N_5, N_6$ and also $N_1 \wedge N_2 \wedge N_5$, $N_2 \wedge N_3 \wedge N_6$. Given NISes N_1, N_2, we say N_1 *covers* N_2 if $N_1 \wedge N_2 = N_2$. The *endpoint set* E_N of a NIS N is defined as the set of endpoints of the intervals in N, that is: $E_N = \{p \mid \exists p : (p, q) \in N \text{ or } (q, p) \in N\}$. 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 NISes, the *internal-intersection* $A(S)$ of S is defined as the intersection of all the NISes 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 NISes 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: $S_1, S_2 \subset S$, 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 $|S_i|$ is used to denote the *cardinality* of set S_i. The gain $G(S_1, S_2)$ of b-balanced bi-partitioning (S_1, S_2) is defined as:

$$G(S_1, S_2) = t_1 + t_2 - a \times (sw_1 + sw_2) \quad (3)$$

where,

$$t_i = D(A(S_i)) ; \quad i \in \{1, 2\} \quad (4)$$

$$sw_i = |A(S_i)| ; \quad i \in \{1, 2\} \quad (5)$$

In (3), 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 resulted from the extra control circuitry needed to supervise sleep mode operation. The parameter a is introduced to control relative significance of savings vs. overhead terms. Figure 4 shows an example of memory partitioning to exploit sleep mode.

We can now formulate our problem **P1** as stated below. Note that the problem is formulated as a decision problem although it can also be formulated as an optimization problem.

Instance: Ordered quadruple (a, b, c, S). Where a is a positive number, b, c are positive integers, and $S = \{N_1, N_2, ..., N_r\}$ is a set of NISes of the form: $N_i = \{I_{i1}, I_{i2}, ..., I_{in_i}\}$.

Objective: Determine whether there exists a b-balanced bi-partitioning (S_1, S_2) of S such that: $G(S_1, S_2) \geq c$.

5 NP-Completeness

In this section we discuss the complexity of **P1** and show that it is NP-complete. We present a transformation from the *MIN-CUT INTO BOUNDED SETS* (**MCP**) problem [6], which is an NP-complete problem. Given graph $G = (V, E)$, positive integer $B < |V|$, positive integer K, the **MCP** problem asks whether there exists a B-balanced bi-partitioning (V_1, V_2) of V such that the number of edges in E with one endpoint in V_1 and the other endpoint in V_2 is no more than K. We showed that the following lemma holds [3].

Lemma 1. **MCP** is polynomial-time transformable to **P1**.

It is easy to see that **P1** is in NP. A non-deterministic polynomial time algorithm just needs to guess a bi-partition (S_1, S_2) of S and then check in polynomial time that the gain $G(S_1, S_2)$ of this bi-partition satisfies $G(S_1, S_2) \geq c = K$. Hence, we have the following result:

Theorem 1. **P1** is NP-complete.

6 An Exact Algorithm

The fact that **P1** is NP-complete, rules out the possibility of existence of a polynomial time algorithm for **P1** unless P=NP [6]. The general strategy in such circumstances is to work at two fronts: Towards the theoretical end, the complexity of special sub-classes of the general problem that are potentially solvable in polynomial time are studied. Towards the practical end, heuristic approaches are developed to solve the problem sub-optimally but in polynomial time. Occasionally, it has been observed that formulation of an exact solution to a general NP-complete problem provides valuable insights on how to go about solving special classes of the problem, or how to design practical heuristic algorithms for the problem.

In this section we present an algorithm (PARTITION_EXACT) to solve **P1**. The outline of this algorithm is shown in Figure 5. This algorithm forms the foundation of polynomial time algorithms that we will present in later sections to solve special classes of **P1**. It tries all possible pairs of NISes N, M that are potential internal-intersection of two partitions S_1 and S_2 of a b-balanced bi-partitioning of S, and reports the pair M, N among such partitioning with maximum gain. Let $p = |E_S|$ represent the cardinality of the endpoint set of S, then there are $p(p-1) + 1 = O(p^2)$ intervals (including the empty interval) with endpoints picked from E_S. Therefore there are no more than 2^{p^2} ways to choose either of N and M. Thus the algorithm PARTITION_EXACT would have the time complexity $O(2^{2p^2} f(p, r))$, where $f(p, r)$ is the time needed to perform steps 6 and 7 of the algorithm. Using efficient geometrical techniques we can achieve $f(p, r) = O(r + p)$, obtaining time complexity $O(2^{2p^2}(p + r))$ for PARTITION_EXACT algorithm. The following observation allows us to bound the search space in this algorithm:

Observation 1. Let I_{max} represent the longest interval in a given **P1** instance (a, b, S), and let (S_1, S_2) be a bi-partitioning of S, then no intervals in the internal-intersection of S_1 or S_2 could be possibly longer than I_{max}.

This means that we need to consider only the NISes consisting of intervals with length at most equal to the longest interval in the given **P1** instance.

```
ALGORITHM     PARTITION_EXACT
  Input:    (a, b, S = {N1, N2, ..., Nr})
  Output:   b-balanced bi-partitioning (S1, S2) of S such that
            Ga(S1, S2) is maximized ;
BEGIN
1.  Es = endpoint set of S ;
2.  Max_gain = -1 ;
3.  Int_set = Set of intervals with endpoints in Es ;
4.  For each NIS  N  consisting of intervals picked from Int_set {
5.      For each NIS  M  consisting of intervals picked from Int_set {
6.          If there exists a b-balanced bi-partitioning (P1, P2) of S
              with  N , M  being thier auto-intersections {
7.            Get such partitioning  (P1, P2) of maximum gain ;
8.            If (G(P1, P2) > Max_gain) {
9.                S1 = P1 ;
10.               S2 = P2 ;
11.               Max_gain = G(P1, P2) ;
12.           }
13.       }
14.   }
15. }
END
```

Figure 5: An exact algorithm to solve **P1**

7 Some Polynomial Time Sub-Classes

In this section, we focus on some sub-classes of **P1** for which we can present polynomial time algorithms. The polynomial time algorithm presented for each of these sub-classes is obtained by slight modifications of the algorithm PARTITION_EXACT.

7.1 Single Interval NISes

This section addresses the following sub-class of **P1**, denoted as **P2**:

Instance: Same as **P1**, with each NIS containing a single interval.

Objective: Same as **P1**.

We can use the basic algorithm PARTITION_EXACT to solve **P2**, however, the following observation allows us to achieve a much faster algorithm.

[3] See [4] for the proof of this lemma

Observation 2. Let $P = \{N_1, N_2, ..., N_k\}$ be a set of NISes, each containing a single interval. Then the internal-intersection of P is a NIS that consists of either a single or no intervals.

This observation tells us that no matter how we partition the set of NISes S of **P2** instance into S_1 and S_2, the internal-intersection of either of the partitions P_1 , P_2 consists of only a single interval. That is, we do not need to spend time on multiple interval NISes for N and M at all, since such NISes cannot possibly be the internal-intersection of partitions for a bi-partitioning (S_1, S_2) of S. Therefore to solve **P2** we can use algorithm PARTITION_EXACT with the for loops modified such that only single interval NISes are picked for N and M. This leads to a time complexity of $O(sp^4)$ where s is the number of intervals in the problem instance, that is, $s = \sum_{i=1}^{r}(|N_i|)$, and p is the cardinality of the endpoint set of S, and hence we have the following theorem:

Theorem 2. **P2** can be solved in polynomial time.

Bounding the Search Space: As here we are only dealing with single interval NISes, the task of enumerating pairs of NISes would become equivalent to enumerating single intervals. The following observation reduces the solution space to be searched during the execution of the algorithm. However, it does not improve the asymptotic time complexity of the algorithm.

Observation 3. Let I_{med} and I_{min} represent the intervals in the **P2** instance with the median and minimum lengths, respectively. Then it is easy to show that for one of the partitions we only need to enumerate intervals of lengths no more than $L(I_{med})$ and for the other partition we only need to enumerate intervals of lengths no more than $L(I_{min})$.

7.2 Bounded Number of Switchings

In practice, having to switch the partitions in and out of sleep mode too frequently may not be practical. Because swiching a partition into or out of sleep mode is also a power consuming activity which is desired to be minimized. Moreover, as the number of such switchings is increased, the complexity of the extra control logic needed to supervise the sleep mode is also increased. This can lead to higher percentage for the penalty caused by each time a partition is switched into or out of sleep mode, on the savings in power consumption. Therefore we introduce the following restricted version of **P1**, called **P3**, in which the allowable number of switchings in and out of sleep mode are forced by an input parameter d to be limited:

Instance: Same as **P1**, plus integer, d.

Objective: Same as **P1**, with additional constraint $sw_1 + sw_2 \leq d$.

Note that by increasing parameter a in a **P1** instance we can control $sw_1 + sw_2$ in the final solution. However, this does not affect the time complexity of the algorithm PARTITION_EXACT. On the other hand, by upper-bounding $sw_1 + sw_2$ in problem **P1** we can achieve an $O(sp^{2d})$ algorithm to solve the optimization version of **P3** optimally, which is a pseudopolynomial algorithm [4]. To do this we restrict the algorithm PARTITION_EXACT to only testing those combinations of N, M that satisfy $|N| + |M| \leq d$. Note that $|N| = sw_1$ and $|M| = sw_2$. The following theorem is an immediate result:

Theorem 3. **P3** can be solved in (pseudo)polynomial time.

8 Generalization

In this section we briefly mention a couple of the generalizations of **P1** (and its counterparts **P2**, **P3**. This is intended to suggest that the basic formulation is easily adaptable to cover broader range of optimization problems. We discuss two generalizations: the multi-way partitioning, and the partitioning in a weighted environment. Note that we could also have multi-way partitioning and weighted combined.

Multi-way Partitioning: This is a straightforward generalization. To solve this problem we can either perform a recursive application of the algorithms presented for the corresponding bi-partitioning problem, or enumerating the potential internal-intersection for each of the partitions using m nested loops that would replace the two nested loops in steps 4 and 5 of algorithm PARTITION_EXACT. The trade off is between the quality of results and the running time of the algorithm. This problem is especially useful in partitioning for activity-driven clock tree construction (see Figure 1 c). An interesting idea would be have the algorithm compute the optimal number of partitions as well as the contents of each partition.

Weigted Partitioning: This version of the problem is applicable in circumstances where only statistical analysis is possible to obtain the idle times for each element. In such cases there is a weight w_i associated with each interval I_i. The weight of an interval can represent the probability that the corresponding element is idle during that interval. The weighted version is also useful to model a circuit where different sub-circuits have different power attributes due to the fact that they result in more savings in power consumption even if they are switched into sleep mode for the same period of time (unlike the case for memory which the power attributes are the same). In that case, each NIS N_i has a weight w_i associated with it.

9 Experimental Results

This section presents our preliminary results for this problem. The algorithm PARTITION_EXACT and its modifications to optimally solve **P2** and **P3** are implemented in C and tested. Because of unavailability of test data due to novelty of the problem and its formulation, a set of randomly generated data with controlled parameters were used as test cases to show the effectiveness of the algorithm and applicability of the approach. The results of experiments are shown in Table 1. To simplify the comparison, the following settings are made for all the test cases:

- $|S| = 100$ (S is set of memory elements).
- Balance factor $b = 40$ (each partition should contain at least 40 memory elements).
- A single interval per NIS (complying with **P2** instance).
- Factor a is set to 0 (a is the penalty factor for the total

[4] It is polynomial in s, p, and exponential in $log\ d$, the minimum size needed to express d in the problem instance

min-len	careful partitioning			random partitioning		
	$\frac{t_1+t_2}{T}$ (%)			$\frac{t_1+t_2}{T}$ (%)		
	min	max	avg	min	max	avg
5	6.0	8.0	**7.2**	0.0	0.0	**0.0**
10	16.0	18.0	**17.2**	0.0	0.0	**0.0**
15	26	30	**29**	0.0	0.0	**0.0**
20	38	42	**40**	0.0	0.0	**0.0**
25	60	84	**69**	12	60	**31**
30	88	104	**98**	52	80	**68**
35	116	130	**125**	84	112	**99**
total			**531.4**			**225**
Comparison			+%91.7			**1**

Table 1: Careful vs. random partitioning

number of switchings.)

- T = width of the time window = 50 (see Figure 4).

The parameter *min-len* shows the length of the shortest interval in each problem instance. For each value of *min-len*, 10 random inputs are generated and tested with the algorithm. The minimum, maximum and average values for the ratio $\frac{t_1+t_2}{T}$ resulted from our partitioning algorithm and from a random partitioning algorithm are shown, where t_1 and t_2 are the exploitable sleep time of the partitions in the resulting bi-partitioning. The higher this ratio is, the more savings in power consumption would be if we place the corresponding partitions in sleep mode. Note that if we do not consider the idle times in a partitioning scheme (as it has been done so far) the result is essentially equivalent to a random partitioning. We observe that as the length of the minimum idle times (*min-len*) is increased to cover the whole time window, the results get closer. Note that since the computation window has width $T = 50$, practical range for *min-len* is 5 to 25. These cases are shown in **bold face** in the first column in Table 1. In such cases, our algorithm produces superior results compared with random partitioning.

10 Conclusion

In this paper we studied the memory segmentation problem to exploit sleep mode operation for minimization of the average 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 memory unit. Since it is impractical to have a separate set of control signals for each memory element, it is advisable to partition the memory elements with close activity patterns together such that each partition of the memory can be switched into sleep mode during the time intervals all of its elements are idle. We formulated the problem and showed that it is NP-complete. We also showed some special classes of the problem which are solvable in polynomial time. Experiments were conducted to show the effectiveness and applicability of the presented algorithms. The results of experiments show possibility of significant savings in power consumption if the sleep mode is exploited properly. Our future research in this area would be at several fronts: 1) improve the time complexity of the presented algorithms as well as designing heuristic algorithms that exploit the geometric features of the problem to achieve good solutions, 2) study whether or not **P1** can be formulated as a graph-theoretic problem, and apply efficient existing heuristics for the graph-theoretic formulation (if one exists) to solve **P1** in case of positive answer, 3) apply the existing heuristics for **MCP**, e.g. Kernighan-Lin [7], Fiduccia-Matheyes [5], Ratio-Cut [15], etc., to see how fast the can be implemented for solving **P1** and how well they perform, 4) classify the designs for which the activity patterns could be generated efficiently, and in cases where such patterns may not be generated as a set of exact idle sets, study statistical approaches that will generate some weighted version of the idle sets in which the weights could represent the probabilities of being idle during different periods, and 5) ultimately, 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. In such a setting, the algorithm should take into account both the number of partitions and the number of switchings for the amount of overhead in power consumption, caused by the extra control circuitry and switching the partitions into and out of sleep mode, when computing the gain function, which translates to the amount of savings in power consumption.

11 Acknowledgments

This work has been supported in part by the National Science Foundation under grant MIP 9207267, and an IBM PhD resident study program.

References

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

[2] A. Chandrakasan et. al. "Optimizing Power Using Transformations". IEEE Transactions on Computer Aided Design. Submitted, Dec. 1993.

[3] A. Chandrakasan et. al. "HYPER-LP: A System for Power Minimization Using Architectural Transformations". In International Conference on Computer-Aided Design. IEEE/ACM, 1992.

[4] A. H. Farrahi, G. Téllez, and M. Sarrafzadeh. "Exploiting Sleep Mode Through Activity-Driven Partitioning". Technical report, Northwestern University, EECS Department, Evanston, IL, November 1994.

[5] C. M. Fiduccia and R. M. Mattheyes. "A Linear Time Heuristic for Improving Network Partitions". In Design Automation Conference, pages 175–181, 1982.

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

[7] B.W. Kernighan and S. Lin. "An Efficient Heuristic Procedure for Partitioning Graphs". Bell System Technical Journal, 49:291–307, Feburary 1970.

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

[9] K. Roy and S. Prasad. "Circuit Activity Based Logic Synthesis for Low Power Reliable Operations". IEEE Transactions on VLSI Systems, 1(4):503–513, 1993.

[10] G. Téllez, A. H. Farrahi, and M. Sarrafzadeh. "Activity-Driven Clock Design for Low Power Circuits". Technical report, Northwestern University, EECS Department, Evanston, IL, November 1994.

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

[12] 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.

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

[14] H. J. M. Veendrick. "Short-circuit Dissipation of Static CMOS Circuitry and its Impact on the Design of Buffer Circuits". Journal of Solid State Circuits, pages 468–473, August 1984.

[15] Y. C. Wei and C. K. Cheng. "Ratio-Cut Partitioning for Hierachical Designs". IEEE Transactions on Computer Aided Design, 40(7):911–921, July 1991.