Exploiting Sleep Mode for Memory Partitioning and Other Applications*

AMIR H. FARRAH1, GUSTAVO E. TÉLLEZ and MAJID SARRAFZADEH**

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

Sleep mode operation and exploiting it to minimize the average power consumption are of great importance in modern VLSI circuits. In general, sleep mode refers to the mode in which part(s) of the system are idle. In this paper, we study the problem of partitioning a circuit according to the activity patterns of its elements such that circuit elements with similar activity patterns are packed into the same partition. Then a partition can be placed in sleep mode during the time intervals all elements contained in that partition are idle. We formulate the partitioning problem to exploit sleep mode operation and show that the problem is NP-complete. We present polynomial time algorithms for practical classes of the problem. Applications of the problem to memory and module partitioning and clock gating are discussed. 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.

Keywords: Power minimization, sleep mode, partitioning, NP-Completeness

1. INTRODUCTION

Advances in VLSI and packaging technologies have increased the average transistor count in a chip by about one-hundredfold every decade [2], allowing much more complex functionality. Moreover, the advent of portable and mobile communication and computing services has stirred a great deal of interest in both the commercial and research areas. The dissipation of the heat generated by highly integrated circuits is a crucial factor because virtually all failure mechanisms are boosted at higher temperatures [2]. The minimization of power consumption in modern circuits is therefore of great importance. Due to this importance, there has been considerable shift of attention in the logic and layout synthesis areas [14, 17, 20, 22, 23] and more recently in high-level synthesis [4, 5, 15] from the delay and area minimization issues towards low power design.


**Corresponding author.

1Present address: IBM T. J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598.
Previous research for low power synthesis of digital circuits has focused on issues such as activity-driven technology decomposition and mapping [17, 20, 22], low-power state assignment [12, 21], architectural transformation and reduction of power supply voltage [4], wire and driver sizing [6, 18], and reversible and adiabatic computing [7, 26]. For a survey of these techniques see [8].

Transition density or average switching rate at different sites in a circuit is introduced in [16] as a quantity to measure the circuit activity, which can be used to estimate the average power consumption in a digital circuit. 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 levels, e.g., behavioral level, register-transfer level (RTL), logic level and transistor level.

In this paper we study the partitioning problem to exploit sleep mode 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 the problem and study its complexity. The problem finds many applications in low power design, e.g., the following (see Fig. 1):

- memory segmentation.
- partitioning to power-down portions of the design.
- clock tree construction.

We assume we have synthesis (simulation-based) or statistical data on the idle times of the data items (in case of memory segmentation) or the idle times of the modules (for the two other cases). We present a general formulation for this problem, propose polynomial time algorithms to solve special classes of the problem optimally, and show that the general problem is NP-complete. This rest of this paper is organized as follows: Section 2 presents the necessary background. Section 3 briefly describes how to obtain the idle times for a set of memory or clocked elements in a design. Section 4 presents the problem formulation. The complexity of the problem is discussed in Section 5. Exact algorithms to solve the general problem are presented in Section 6. Some special classes of the problem are discussed in Section 7 and polynomial time algorithms are presented to solve these classes. Section 8 focuses on some generalization of the problem. Experimental results for memory segmentation 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 transitions 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 [24], 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:

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

where $D(g)$ and $V_{dd}$ represent the transition density 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. 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 is an attempt to do so. 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 power consumption. A similar partitioning approach can be applied for clock-tree construction when the activity patterns of the clocked elements are known. The clock signal destinations with close activity patterns should be partitioned into the same subtree to allow maximum savings in power consumption via clock gating (see Fig. 1). 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.

To have a general formulation, we talk about elements. Depending on the application, an element may refer to a memory element, clocked element, or a module in the circuit. Given the activity patterns of a set of elements, the question is how to partition this set to maximize the savings in power consumption achievable through sleep mode, 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.

3. OBTAINING IDLE SETS

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

3.1. Idle Sets for Memory Elements

Let $M = \{m_1, m_2, \ldots, m_r\}$ represent the set of dynamic memory elements (MEs) in an application. Assume that the access sequence for each

\footnote{Average number of transitions per unit time.}
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 Fig. 2). Given the access sequence for all the MEs, we can use the following rules to generate the set of intervals for each ME $m_i$, during which $m_i$ need not be refreshed (see Fig. 2) and thus obtain the idle set for each ME. We say ME $m_i$ is idle during interval $I$ if it need not be refreshed during $I$. Therefore ME $m_i$ is idle:

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

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.

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 $M$ is not used for a consecutive set of cycles, then we can gate the clock signal to the registers feeding this FU during this idle time, which will reduce the power consumption due to the clock tree. Furthermore, it guarantees that there would be no dynamic power consuming activity during this time in $M$. From the scheduled and allocated design we can say that if FU $M$ is assigned to a control step $c$, then it is active during $c$. Otherwise, it is idle during this time. This allows us to generate 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 $y'' + 3y' + 3y = 0$ after scheduling and allocation have taken place. The design contains the following functional units: three multipliers $M1, M2, M3$, 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 the names of their corresponding FUs. Note that the multiplier units take 2 control steps to execute.

However, they are clocked only during the first of the two control steps. This is assuming that the multiplier units are purely combinational, and require no clock signal during their operation. In other contexts, the multipliers (or other multi-step FUs) may need to be clocked during their whole execution cycle. The idle times should be computed according to these requirements.

4. FORMULATION OF THE PROBLEM

Consider a set \( M = \{m_1, m_2, \ldots, m_r\} \) of elements. We say that element \( m \) is idle during time interval \( I = (l, r) \), \( l < r \), if \( m \) can be switched into sleep mode during \( I \). We say that interval \( I = (l, r) \) contains point \( p \), or \( p \) is contained in \( I \) if \( l \leq p \leq r \). 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 NISs (Non-overlapping Interval Sets) during all of which \( m \) is idle. We assume that the idle sets of elements in \( M \) are given as a set \( S = \{N_1, N_2, \ldots, N_r\} \), where \( N_i = \{I_{i1}, I_{i2}, \ldots, I_{in}\} \) is the idle set of \( m_i \) (see Fig. 4).

The notation \((\cdot)\) 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 \( l_1 \leq l_2 \) and \( r_2 \leq r_1 \), 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 \( I_1 \land I_2 \), 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. It is easy to see that the intersection of more than two intervals is commutative, hence no parentheses are needed. The intersection of two NISs \( N_1 = \{I_{11}, I_{12}, \ldots, I_{1n}\} \) and \( N_2 = \{I_{21}, I_{22}, \ldots, I_{2m}\} \), denoted as \( N_1 \land N_2 \), 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:

\[
N_1 \land N_2 = \{I = I_1 \land I_2 | I_1 \in N_1, I_2 \in N_2, \\
I_1 \land I_2 \neq ()\}
\]

(2)

The intersection of more than two NISs is defined similarly. As for the intervals, the intersection of multiple NISs is a commutative operation and parentheses can be omitted without causing ambiguity. Given NISs \( N_1, N_2 \), we say \( N_1 \) covers \( N_2 \) if \( N_1 \land N_2 = N_2 \). The endpoint set \( E_N \) of a NIS \( N \) is defined as the set of endpoints of the intervals in \( N \), that is: \( E_N = \{p|\exists p: (p, q) \in N \text{ or } (q, p) \in N\} \). The duration \( D(N) \) of a NIS \( N = \{I_1, I_2, \ldots, I_k\} \) 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 \( S = \{N_1, N_2, \ldots, N_k\} \) of NISs, the internal-intersection \( A(S) \) of \( S \) is defined as the intersection of all the NISs in \( S \), that is:

\[
A(S) = \land_{N_i \in S} N_i
\]

(3)

For example, Figure 4 shows 6 memory elements \( m_1, \ldots, m_6 \). A partitioning of the memory is shown which dictates a corresponding partitioning \((S_1, S_2)\) of their idle sets, where \( S_1 = \{N_1, N_2, N_4\} \) and \( S_2 = \{N_3, N_5, N_6\} \). The internal intersection of each partition is shown as a set of shaded regions for each partition. The endpoint set \( E_S \) of \( S \) is defined as the union of the endpoint sets of the NISs in \( S \), that is \( E_S = \{p|\exists N \in S \ni p \in E_N\} \).

Given a set \( S \), \((S_1, S_2)\) is a bi-partitioning for \( S \), if: \( S_1, S_2 \subset S \), \( S_1 \cap S_2 = \emptyset \), and \( S_1 \cup S_2 = S \). Each of \( S_1 \) and \( S_2 \) is called a partition of \( S \). The density of a partition at a given point \( p \), is the number of NISs in that partition that contain some interval containing \( p \). The bi-partitioning \((S_1, S_2)\) is \( b \)-balanced if \( |S_1| \geq b \) and \( |S_2| \geq b \), where the notation \( |S| \) denotes the cardinality of set \( S \). The gain \( G_a(S_1, S_2) \) of a \( b \)-balanced bi-partitioning \((S_1, S_2)\) is defined as:

\[
G_a(S_1, S_2) = t_1 + t_2 - a \times (sw_1 + sw_2)
\]
In (4), 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 resulting from the extra control circuitry needed to supervise sleep mode operation. 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.

Note that many problems can be formulated as a decision or an optimization problem and that if the decision version of a problem \( P \) is NP-complete then its optimization version is also NP-complete, and if its optimization version is polynomially solvable then its decision version can also be solved in polynomial time. We now formulate our problem as a decision problem:

**P1:**

- **Instance:** Ordered quadruple \((a, b, c, S)\), where \(a\) is a positive (non-negative) number, \(b, c\) are positive integers, and \(S = \{N_1, N_2, \ldots, N_r\}\) is a set of NISs.
- **Objective:** Determine whether there exists a \(b\)-balanced bi-partitioning \((S_1, S_2)\) of \(S\) such that:

\[
G_a(S_1, S_2) \geq c^5
\]

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** problem [11], that we will denote as \( MCP \) (Min-Cut Problem). This problem can be stated as follows:

**MCP:**

- **Instance:** Graph \( G = (V, E)\), positive integer \(B \leq |V|\), positive integer \(K\).

---

4 An interval is **maximal** with respect to a property \( \mathcal{P} \), if it has the property \( \mathcal{P} \), but no interval containing it as a proper sub-interval has property \( \mathcal{P} \). Here, the property \( \mathcal{P} \) is that the density of the partition during this interval is equal to the size of the partition.

5 In the optimization formulation of \( P1 \), \( G_a(S_1, S_2) \) should be maximized.

6 Note that we are using a special formulation of **MIN-CUT INTO BOUNDED SETS** problem which is still NP-complete. This special formulation is used to simplify our NP-completeness proofs for problem \( P1 \).
• **Objective:** Determine whether there exists a balanced bi-partitioning \((V_1, V_2)\) of \(V\) such that:

\[
\sum_{u \in V_1, v \in V_2} w(u, v) \leq K
\]

where,

\[
w(u, v) = \begin{cases} 
0 & \text{if } (u, v) \notin E \\
1 & \text{if } (u, v) \in E
\end{cases}
\]

That is, the size of each partition is lower bounded by \(B\), and 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 will refer to the number of such edges as the *cost* of the bipartitioning and denote it as \(C(V_1, V_2)\).

Given a partitioning \((V_1, V_2)\) of \(V\), we define an attribute \(c_i\) for each edge \(e_i = (v_i_1, v_i_2)\) in \(E\), referred to as the *cost* of that edge under partitioning \((V_1, V_2)\), as follows:

\[
c_i = \begin{cases} 
0 & \text{if } v_i_1, v_i_2 \text{ belong to the same partition} \\
1 & \text{otherwise}
\end{cases}
\]  

If \(c_i = 1\) we say that edge \(e_i\) is *cut* by the partitioning, otherwise \(e_i\) is not cut or *uncut*. It is straightforward to show that:

\[
C(V_1, V_2) = \sum_{i=1}^{\left|E\right|} c_i
\]  

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

**Proof** Given an instance \((G(V, E), B, K)\) of *MCP*, with \(V = \{v_1, v_2, \ldots, v_{\left|V\right|}\}\) and \(E = \{e_1, e_2, \ldots, e_{\left|E\right|}\}\), we construct the corresponding instance \((a, b, c, S)\) of *P1* as follows:

- The set \(S = \{N_1, N_2, \ldots, N_{\left|V\right|}\}\) consists of a set of NISs, where each NIS \(N_i \in S\) corresponds to a vertex \(v_i \in V\). Let \(\{e_{i_1}, e_{i_2}, e_{i_3}, \ldots, e_{i_{n_i}}\}\) represent the set of edges in \(E\) incident to vertex \(v_i\). Let \(X_i = \{i_1, i_2, \ldots, i_{n_i}\}\) represent the set of indices of the edges incident to \(v_i\), and let \(X = \{1, 2, \ldots, \left|E\right|\}\). Then the NIS \(N_i\) of the *P1* instance corresponding to vertex \(v_i\) of the *MCP* instance will be defined as follows:

\[
N_i = \{(3x, 3x + 1)|x \in X_i\} \
\cup \{(3x, 3x + 2)|x \in (X - X_i)\}
\]  

That is, each NIS \(N_i\) corresponding to a vertex \(v_i\) consists of \(\left|E\right|\) intervals, one corresponding to each edge \(e \in E\).

- \(a = 0\)
- \(b = B\)
- \(c = 3\left|E\right| + K\)

Figure 5 shows a given instance of *MCP* and the constructed instance of *P1*. Notice that each NIS \(N_i\) in the constructed instance of *P1* consists of exactly \(\left|E\right|\) intervals, \(|X_i|\) of which of length 1 and the rest of length 2.

![Figure 5](image-url)
Main Idea: In the construction of the PI instance, the following issues have been taken into account:

- A partitioning \((V_1, V_2)\) of vertices in the MCP instance should correspond to the partitioning \((S_1, S_2)\) of the corresponding NISs in the PI instance and vice versa, e.g., the partitioning \(\{(v_1, v_3), (v_2, v_4)\}\) of \(V\) corresponds to the partitioning \(\{(N_1, N_3), (N_2, N_4)\}\) of \(S\).

- The gain \(G_a\) of the partitioning \((S_1, S_2)\) of \(S\) in the PI instance should be a decreasing function of the cost \(C\) of the corresponding partitioning \((V_1, V_2)\) of \(V\) in the MCP instance.

Having established such relationship between the MCP and PI instances, it is easy to see that the cost of a partition in the MCP instance is minimized if and only if the gain of the corresponding partitioning of the PI instance is maximized. Then by selecting the parameters \(a, b, c\) properly, we can show that the answer to the MCP instance is YES if and only if the answer to the constructed PI instance in YES. The following elaborates further on these arguments:

General Properties: As it is shown in Figure 5, the PI instance is constructed such that corresponding to each edge \(e_i = (v_j, v_k)\) in the MCP instance there are \(|V|\) intervals, one in each of the NISs, and they are all overlapping. Let \(I_i = \{I_{i_1}, I_{i_2}, \ldots, I_{i_{|V|}}\}\) represent the set of these intervals, where \(I_{i_j}\) is the interval corresponding to \(e_i = (v_j, v_k)\) in \(N_{p}\). Among these intervals, \(I_{i_1}\) and \(I_{i_2}\) are the two in \(N_{p}\) \(N_{b}\) (NISs corresponding to vertices \(v_j, v_k\), the two ends of edge \(e_i\)), extend from \(3i\) to \(3i+1\), and the rest extend from \(3i\) to \(3i+2\). Consider a bipartitioning \((S_1, S_2)\) of \(S\). Note that this partitioning induces a corresponding partitioning \((I_{i_1}, I_{i_2})\) on each of the \(I_i\)’s. We can make the following observations from the construction of PI instance:

Observation 2. Two intervals in the constructed PI instance corresponding to two distinct edges \(e_i, e_j\) of MCP instance do not overlap each other.

Observation 3. Since \(a=0\), we have: \(G_a(S_1, S_2) = t_1 + t_2\).

From Observation 2 and the definition of \(t_1\) and \(t_2\), it becomes clear that \(t_1\) and \(t_2\) can each be computed as the summation of \(|E|\) terms. Each of these \(|E|\) terms corresponds to the contribution of the intervals corresponding to one of the edges in the MCP instance. Therefore we can write:

\[
t_1 = \sum_{i=1}^{|E|} t_{1_i} \\
t_2 = \sum_{i=1}^{|E|} t_{2_i}
\]

\[
G_a(S_1, S_2) = \sum_{i=1}^{|E|} (t_{1_i} + t_{2_i})
\]

where:

\[
t_{1_i} = L(N_{b_i} \in I_{i_1} I_{i_2}); \quad \forall i \in \{1, 2, \ldots, |E|\}
\]

\[
t_{2_i} = L(N_{b_i} \in I_{i_2} I_{i_3}); \quad \forall i \in \{1, 2, \ldots, |E|\}
\]

An example is shown in Figure 6. The partitioning \((V_1, V_2)\) of the MCP instance and the corresponding partitioning \((S_1, S_2)\) of the constructed PI instance are shown and the values \(C(V_1, V_2), G_a(S_1, S_2)\) are computed.

As it is shown in Figure 7 we can categorize the edges \(e_i\) in \(E\) as cut and uncut edges, and hence categorize the sets \(I_i\) corresponding to them and compute the contribution \(t_{1_i}\) and \(t_{2_i}\) for each of the edges to \(C(V_1, V_2)\) and \(G_a(S_1, S_2)\) as follows:

<table>
<thead>
<tr>
<th>Category</th>
<th>Contribution (c_i) to (C(V_1, V_2))</th>
<th>Contributions (t_{1_i}, t_{2_i}) to (G_a(S_1, S_2))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut edges (e_i)</td>
<td>(c_i = 1)</td>
<td>(t_{1_i} = 1, t_{2_i} = 1)</td>
</tr>
<tr>
<td>Uncut edges (e_i)</td>
<td>(c_i = 0)</td>
<td>((t_{1_i} = 1, t_{2_i} = 2)) or ((t_{1_i} = 2, t_{2_i} = 1))</td>
</tr>
</tbody>
</table>
MEMORY PARTITIONING

$G(V, E) = \{v_1, v_2\}$

$G(V, E) = \{v_3, v_4\}$

$C(V_1, V_2) = 0 + 1 + 1 + 0 = 2$

$C(V_1, V_2) = 0 + 1 + 1 + 0 = 2$

$|E| = 4$

$3|E| - C(V_1, V_2) = 3(4) - 2 = 10$

$\text{(If)}$ Suppose that the answer to the constructed P1 instance $(a, b, c, S)$ is YES. Then there exists a bi-partitioning $(S_1, S_2)$ of $S$ such that: $|S_1| \geq b = B$, $|S_2| \geq b = B$. On the other hand from (16) and (17) get: $C(V_1, V_2) \leq K$. This means that the answer to the given MCP instance is YES.

$\text{(Only if)}$ Suppose that the answer to the given instance of MCP is YES. Then there exists a bi-partitioning $(V_1, V_2)$ of $V$ such that: $|V_1| \geq B$, $|V_2| \geq B$, and that:

$C(V_1, V_2) \leq K = 3|E| - c$  \hspace{1cm} (18)

Consider the corresponding bi-partitioning $(S_1, S_2)$ of $S$ in the constructed P1 instance. We have $|S_1| = |V_1| \geq b = B$, $|S_2| = |V_2| \geq B = b$. On the other hand from (16) and (18) we get: $G_d(S_1, S_2) \geq 3|E| - K = c$. Therefore the answer to the constructed P1 instance is YES.

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 $G_d(S_1, S_2)$ of this bi-partitioning satisfies $G_d(S_1, S_2) \geq c$. Hence, we have the following result:

**Theorem 1** The problem P1 is NP-complete.
6. EXACT ALGORITHMS

The fact that $P_1$ is NP-complete, rules out the possibility of existence of a polynomial time algorithm for $P_1$ unless $P = NP$ [11]. 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. Pinning out such sub-classes, of course, is not always an easy task. 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, despite its exponential running time, provides valuable insights on how to design practical heuristic algorithms for the problem. Such exact solutions may also help understanding some special sub-classes of the general problem that are optimally solvable in polynomial time.

In this section we address two algorithms $\text{PARTITION\_EXACT1}$ and $\text{PARTITION\_EXACT2}$ to solve $P_1$. The outline of these algorithms are shown in Figure 8.

- $\text{PARTITION\_EXACT1}$: The first algorithm (Figure 8a) which is also the trivial one would be to try all possible ways to $b$-balance bi-partition $S$, compute the gain for each, and report the partitioning with maximum gain value $G_a(S_1, S_2)$. The running time of this algorithm would be

$$O\left( \binom{r}{b} + \binom{r}{b+1} + \cdots + \binom{r}{r-b} \right) f_1(r, b),$$

where $r = |S|$ is the number of NISs in $S$, $\binom{r}{k}$ is the number of ways to pick $k$ elements out of a set with $r$ elements, and $f_1(r, b)$ is the time required to compute the gain of a $b$-balanced partitioning $(S_1, S_2)$ of $S$. Since in practice $b$ is not a constant, this approach results in an exponential running time.

- $\text{PARTITION\_EXACT2}$: The second algorithm (Fig. 8b) which forms the foundation of polynomial time algorithms that we will present in later sections to solve special classes of $P_1$, works as follows: it tries all possible ways to select a pair of NISs $N, M$ 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 $M, N$, which results in 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 $2p^2$ ways to choose either of $N$ and $M$. Thus the algorithm $\text{PARTITION\_EXACT 2}$ would have the time complexity $O(2^{2p^2}f_2)$, where $f_2$ is the time needed to perform steps 6 and 7 of the algorithm. Figure 9 presents an implementation of steps 6 and 7 of algorithm...
MEMORY PARTITIONING

P1 = (N \subseteq S | N \cap N_1 = N_1);
P_1 = S - P_1;
P_2 = (N \subseteq P_1 | M \cap N_2 = M_2);
1. If |P_1| < 0 or |(P_1)| + |P_2| < 2b
   Goto next iteration of loop at step 5;
2. Let P = a subset of P with size p, where (b \leq |P| \leq |P_1|); 
P = P - P_1;
P = P \cup \Delta P;

FIGURE 9 Implementation of steps 6, 7 of Algorithm_Ex-
act2.

PARTITION_EXACT2. Noting that the intersection of two NISs N_i and N_j can be computed using a linear scan of the endpoints in O(|N_i| + |N_j|) time, we get the following running time for the implementation shown in Figure 9:

\[ f_2 = O\left( \sum_{N_i \in S} |N_i| + |S|(|M| + |M'|) \right) = O(s + rp) \]

where \( s = |S| \), \( p \) represents the size of the endpoint set of \( S \), and \( r = \sum_{N_i \in S} |N_i| \) represents the number of intervals in the given P1 instance, respectively.

Replacing this, we achieve time complexity \( O(2^{2p} (s + rp)) \) for PARTITION_EXACT2 algorithm. The following observation can be used to bound the search space in PARTITION_EXACT2 algorithm:

Observation 4. Let \( I_{\text{max}} \) 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 possibly be longer than \( I_{\text{max}} \).

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_EXACT2.

7.1. Single Interval NISs

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

P2:
- **Instance**: Ordered quadruple \((a, b, c, S)\). Where \( a \) is a positive number, \( b \) and \( c \) are integers, and \( S = \{N_1, N_2, \ldots, N_k\} \) is a set of NISs of the form: \( N_i = \{I_i\} \), that is each NIS consists of a single interval.

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

We can use the basic algorithm PARTITION_EXACT2 to solve P2, however, the following observation allows us to achieve a much faster algorithm.

Observation 5. Let \( P = \{N_1, N_2, \ldots, N_k\} \) be a set of NISs, each containing a single interval. Then the internal-intersection of \( P \) is a NIS that consists of either a single or no interval.

We can use the basic algorithm PARTITION_EXACT2 to solve P2, however, the following observation allows us to achieve a much faster algorithm.

**Observation 5.** Let \( P = \{N_1, N_2, \ldots, N_k\} \) be a set of NISs, each containing a single interval. Then the internal-intersection of \( P \) is a NIS that consists of either a single or no interval.

This observation tells us that no matter how we partition the set of NISs \( S \) of P2 instance into \( S_1 \) and \( S_2 \), the internal-intersection of either of the partitions \( S_1, S_2 \) consists of only a single interval. That is, we do not need to spend time on multiple interval NISs for \( N \) and \( M \), since such NISs 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_EXACT2 with the for loops modified such that only single interval NISs are picked for \( N \) and \( M \). This leads to \( f_2 = O(s + r) = O(s) \) and the time complexity of \( O(sp^2) \) where \( s \) is the number of intervals in the problem instance, and \( p \) is the cardinality of the endpoint set of \( S \), and hence we have the following theorem:

**Theorem 2.** The problem P2 can be solved in polynomial time.
**Observation 6.** Let $I_{\text{min}}$ and $I_{\text{med}}$ represent the intervals in the P2 instance with the smallest and $b$-th largest 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 $I_{\text{med}}$ and for the other partition we only need to enumerate intervals of lengths no more than $I_{\text{min}}$.

This observation allows limiting the solution space to be searched during the execution of the algorithm. However, it does not improve the asymptotic time complexity of the algorithm.

It should be noted that the solution to problem P2 suggests a heuristic algorithm for the general problem P1. The idea would be to devise a function $\mathcal{F}$ mapping the set $S$ of multi-interval NISs in the given instance of P1 onto a set $S'$ of single interval NISs and thus generate a P2 instance. Then the optimal partitioning solution for $S'$ yields a heuristic partitioning solution for $S$. The choice of $\mathcal{F}$ could affect the quality of the heuristic solution, and is to be studied. A reasonable candidate for $\mathcal{F}$ maps an NIS $N$ to NIS $N'$, where $N' = \{I\}$ and $I$ is the longest interval in $N$.

Figure 10 demonstrates the idea.

**7.2. Bounded Number of Switchings**

In practice, switching the partitions in and out of sleep mode is itself a power consuming activity which should 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. As a solution to this problem, the following (restricted) version of P1, called P3 is introduced, in which the summation of the switchings of the partitions is upper bounded by an input parameter $d$.

**P3:**

- **Instance:** Ordered quintuple $(a, b, c, d, S)$,
  - Where $a$ is a positive number, $b, c, d$ are integers, and $S = \{N_1, N_2, \ldots, N_r\}$ is a set of NISs.

  One may think 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_EXACT2. 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. To do this we restrict the algorithm PARTITION_EXACT to only testing those combinations of $N, M$ that satisfy $|N| + |M| \geq d$. Note

\[sw_1 + sw_2 \leq d\]

\[G_d(S_1, S_2) \geq c\]

where $sw_i = |A(S_i)|$ is the number of intervals in the internal-intersection of $S_i$.

\[It \text{ is polynomial in } s, p, \text{ and exponential in } \log d, \text{ the minimum size needed to express } d \text{ in the problem instance.}\]
that \(|N| = s_{w_1}\) and \(|M| = s_{w_2}\). The following theorem is an immediate result:

**Theorem 3** The problem \(P3\) 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 a broader range of optimization problems. We discuss two generalizations: the multi-way partitioning, and the weighted partitioning. 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_EXACT2. The trade off is between the quality of results and the running time of the algorithm. The recursive application of bi-partitioning approach is faster, however it generates results with inferior quality. Note that this problem is especially useful in partitioning for clock tree construction to maximize savings in power consumption by clock gating (see Fig. 1c).

- **Weighted 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 various savings in power consumption even if they are switched into sleep mode for the same period of time. In that case, each NIS \(N_i\) has a weight \(w_i\) associated with it.

### 9. EXPERIMENTAL RESULTS

The algorithm PARTITION_EXACT2 and its modifications to optimally solve \(P2\) 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. The results of experiments are shown in Table I. To simplify the comparison, the following settings are made for all the test cases:

- \(|S| = 100\) (\(S\) is the set of elements).
- Balance factor \(b = 40\) (each partition should contain at least 40 elements).
- A single interval per NIS (complying with \(P2\) instance).

<table>
<thead>
<tr>
<th>min-len</th>
<th>(%) (\frac{d_{\text{avg}}}{d_{\text{max}}}) (%)</th>
<th>Our Algorithm</th>
<th>Avg. Run Time (Sec.)</th>
<th>Random Partitioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6.0</td>
<td>8.0</td>
<td>7.2</td>
<td>70</td>
</tr>
<tr>
<td>10</td>
<td>16.0</td>
<td>18.0</td>
<td>17.2</td>
<td>252</td>
</tr>
<tr>
<td>15</td>
<td>26</td>
<td>30</td>
<td>29</td>
<td>408</td>
</tr>
<tr>
<td>20</td>
<td>38</td>
<td>42</td>
<td>40</td>
<td>522</td>
</tr>
<tr>
<td>25</td>
<td>60</td>
<td>84</td>
<td>69</td>
<td>288</td>
</tr>
<tr>
<td>30</td>
<td>88</td>
<td>104</td>
<td>98</td>
<td>156</td>
</tr>
<tr>
<td>35</td>
<td>116</td>
<td>130</td>
<td>125</td>
<td>273</td>
</tr>
<tr>
<td>40</td>
<td>142</td>
<td>150</td>
<td>146</td>
<td>121</td>
</tr>
</tbody>
</table>

**TABLE I** Comparison of our partitioning algorithm and random partitioning
Factor $a$ is set to 0 ($a$ is the penalty factor for the total number of switchings). This makes sense because the switching of either of the partitions is in the range $\{0,1\}$, hence the sleep mode control circuitry will cause negligible overhead on the area or power consumption.

- $T = \text{width of the time window} = 50$ (See Fig. 4).

To apply this algorithm for the general case, one can use a pre-processing step which takes as input the idle sets of the CEs, and generates as output a single idle interval for each CE. The generated single idle interval for a CE can simply be the longest interval in the idle set of that CE, or it can be obtained using a more complicated strategy. The parameter $\text{min-len}$ shows the length of the shortest interval in each problem instance. For each value of $\text{min-len}$, 10 random inputs are generated and tested with the algorithm. The minimum, maximum and average values for the ratio $(t_1 + t_2)/T$ 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 the savings in power consumption would be if we place the corresponding partitions in sleep mode. Note that if we don’t consider the idle times in a partitioning scheme (as it has been done so far) the result is essentially equivalent to a random partitioning. However, by partitioning the set of elements according to their idle times we can maximize ratio $(t_1 + t_2)/T$, and minimize the power consumption by exploiting sleep mode. It can be observed that as the length of the minimum idle times ($\text{min-len}$) 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 $\text{min-len}$ is 5 to 25. These cases are shown in bold face in the first column in Table I. In such cases, our algorithm produces superior results, with an average of 7 to 40% more sleep time, compared to random partitioning.

10. DISCUSSION AND CONCLUSION

In this paper we studied the circuit partitioning 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 corresponding circuit elements. The idea is to partition the set of elements such that the elements with close activity patterns are grouped into the same partition so that each partition 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 discussed some special classes of the problem which are solvable in polynomial time. Experiments were conducted to show the effectiveness of the presented algorithms. The results of experiments show possibility of significant savings in power consumption if the sleep mode is exploited properly. Recently, a more realistic set of experiments have been reported in [9] for memory segmentation on a number of DSP and numerical applications, with considerable reduction on the estimated power consumption of the memory unit, using an iterative improvement partitioning technique. To obtain the idle sets for memory elements in the work reported in [9] the applications were run on an emulator with a profiling tool that kept track of different resource utilizations over time and space. The idle sets were then calculated using an idea similar to the one mentioned in Section 3.1 from the access sequence provided by the profiling tool. Further work on gated clock tree design have been reported in [3,19]. The following provides directions for further research in this area:

- Improving the time complexity of the algorithms. Although the algorithms presented for special cases $P2$ and $P3$ are polynomial time algorithms, the growth rate of the running time with problem size limits the applicability of this approach.
Having shown that sleep mode and its exploitation could lower the power consumption, gives rise to new problems in high level synthesis, that is, how to perform the scheduling and allocation tasks such that potential savings in power consumption achievable by exploiting sleep mode operation is maximized. It is noteworthy that the register allocation step in high-level synthesis tends to minimize the sleep time of the registers in order to reduce the required number of registers in the design. This brings up the trade off issue between area and power consumption in the high-level synthesis, which calls for further investigation.

We mentioned earlier that using a mapping function $F$ to obtain single interval NISs from multi-interval NISs in $S$, we can construct a $P_2$ instance from a given $P_1$ instance. The constructed $P_2$ instance can then be solved optimally to lead to a heuristic partitioning solution for our original $P_1$ instance. Further theoretical and experimental studies can be pursued to identify suitable choices for the mapping function $F$.

It would be worthwhile to devise heuristics based on which to perform the partitioning sub-optimally, but fast. This could be of use as a design aid for low power design to provide a quick feedback to the designer on how the design modifications or decisions made at higher levels would affect the sleep times of the partitions.

The geometric flavor of the problem demands for carefully designed algorithms that exploit the geometric features of the problem to achieve good solutions. Hence it is worthwhile to study this problem from a geometric viewpoint in search of fast approximation or heuristic algorithms for the general or special classes of the problem.

Another interesting problem is whether or not $P_1$ can be formulated as a (hyper)graph partitioning or in general any (hyper)graph problem at all. Our attempts indicate that such a formulation is unlikely to exist although we have no formal proof to present for it. Further research is in order to show whether or not such formulation is possible. In the case of positive answer, the existing algorithms for the (hyper)-graph formulation can be applied to solve $P_1$.

As $P_1$ is formulated as a set partitioning algorithm, it is nice to see how well the existing heuristics for MCP, e.g., Kernighan-Lin [13], Fiduccia-Mattheyses [10], Ratio-Cut [25], etc., can be modified to operate on $P_1$ instances, how fast they can be implemented, and how well they perform.

A crucial assumption in this paper was the availability of the activity patterns (idle times) as input to our problem. It is of particular interest to categorize the designs for which such patterns can be generated efficiently. Furthermore, in cases where such patterns may not be generated as a set of exact idle sets, statistical approaches could be employed to generate some weighted version of the idle sets in which the weights could represent the probabilities of being idle during different periods. It is therefore worthwhile to formulate and study the weighted version of the problem.

A generalization of the problem would be to allow multi-way partitioning, and perhaps to compute the optimal number of partitions as well as the contents of each partition.

It is also useful to investigate other areas in which problem ($P_1$) could find applications.

Acknowledgement

This work has been supported in part by the National Science Foundation under grant MIP 9207267.

References


1986 to 1990 he worked for IBM Corporation in their EDA Development facility in East Fishkill, NY. Presently, he is pursuing a Ph.D., in Computer Science at Northwestern University under the IBM Ph.D., Resident Study Program. His research interests include design and analysis of algorithms, computer architectures, timing and power driven VLSI design, and clock network design.

**Majid Sarrafzadeh** received his B.S., M.S., and Ph.D., in 1982, 1984, and 1987, respectively, all from the University of Illinois at Urbana-Champaign in Electrical and Computer Engineering Department. He joined Northwestern University as an Assistant Professor in 1987. Since 1991 he has been Associate Professor of Electrical Engineering and Computer Science at Northwestern University. His research interests lie in the area of design and analysis of algorithms and computational complexity, with emphasis in VLSI. Dr. Sarrafzadeh is a member of IEEE Computer Society. He received as NSF Engineering Initiation award in 1987, two distinguished paper awards in ICCAD-91, and the Best Paper Award for physical design in DAC-93. He has served on the technical program committee of various conferences, for example, ICCAD, EDAC and ISCAS. He is a co-editor of the book “Algorithmic Aspects of VLSI Layout”, co-author of a forthcoming book “An introduction to VLSI Physical design”, an Associate Editor of IEEE Transactions on Computer-Aided Design.