Integrated Memory Grouping and Power-Aware MBIST Scheduling for MPSoCs

Koki Asahina
Nara Institute of Science and Technology (NAIST), Ikoma, Japan

Yasuhiko Nakashima
Nara Institute of Science and Technology (NAIST), Ikoma, Japan

Abstract

Memory Built-In Self-Test (MBIST) is a widely adopted technique for testing memory. In modern large-scale SoCs, hundreds to thousands of embedded memories are integrated, and to test them efficiently, methods that group memories and test them in parallel within each group are employed. However, many existing approaches either do not account for test scheduling or rely on evolutionary methods, such as genetic algorithms (GAs), for grouping, which incur high computational costs. In this work, we propose a framework that covers the flow from memory grouping to test scheduling. Taking the specifications and layout information of multiple SRAMs into account, the framework comprises a flexible, fast memory grouping method and a scheduling method that minimizes the total test time under a power-constrained constraint. In the proposed approach, DBSCAN and rectangular partitioning are used to perform fast grouping while suppressing long routing connections, and an LPT-based greedy heuristic is employed to shorten the total test time under constraints on the power limit and the number of simultaneously active BIST controllers. Experimental evaluation using SRAM placement data based on the ASAP7 PDK shows that, compared with existing K-means, Greedy, and GA-based methods, the proposed method reduces the number of groups by up to 48% while achieving approximately 87× speedup in clustering runtime. Furthermore, compared with a commercial Industrial Solution, it reduces the test time by 53%. These results demonstrate that the proposed method provides high scalability and practical effectiveness for MBIST design, even in large-scale MPSoCs with a large number and variety of embedded memories.

2012 ACM Subject Classification Hardware → Testing with distributed and parallel systems

Keywords and phrases MBIST, DfT, Memory Grouping, Power-Aware Scheduling

Digital Object Identifier 10.4230/OASIcs.NG-RES.2026.3

Funding *Koki Asahina*: JST-ALCA-Next Program (Grant Number JPMJAN23F4)
Yasuhiko Nakashima: JST-ALCA-Next Program (Grant Number JPMJAN23F4)

1 Introduction

Modern multi-processor systems-on-chip (MPSoCs) commonly integrate hundreds to thousands of embedded memory blocks [30] alongside heterogeneous processing elements such as CPUs, GPUs, and NPUs. To ensure the correct functionality of these memories after manufacturing, implementing Memory Built-In Self-Test (MBIST) is indispensable. However, as the number and variety of memories increase, the circuitry required for test access becomes increasingly complex, leading to significant growth in chip area, routing overhead, and power consumption [25, 2]. To mitigate these issues, memory grouping – in which multiple memories share a single MBIST controller is widely employed [22].

For efficient MBIST design, it is crucial to group physically proximate memories while respecting their physical placement and clock domains, enabling parallel test execution. However, the number of groups presents a fundamental trade-off between circuit area and test application time [28, 11]: creating too many groups increases the number of MBIST

controllers, thereby inflating area overhead. Consequently, optimizing memory grouping can reduce test wirelength, latency, and power consumption while achieving a balanced trade-off between silicon cost and test efficiency.

Existing grouping methods, however, often neglect essential physical or architectural constraints, resulting in poor routability [22, 12, 15, 21, 4, 27, 20, 19, 16], or conversely, incorporate unnecessary constraints, significantly increasing tool runtime. Such limitations hinder the adoption of these methods in industrial-scale MPSoC design flows [13].

Furthermore, in large-scale MPSoCs, the importance of *test scheduling* within MBIST has progressively increased. Each memory exhibits distinct test-time and test-power characteristics depending on its capacity and internal architecture. When multiple memories are tested concurrently, the total power consumption may exceed the power delivery network limits, or the overall test time may unintentionally increase due to imbalanced test durations [8, 3, 23, 27, 14]. Nevertheless, in current industrial design flows, grouping and scheduling are typically performed independently. As a result, physically reasonable groups may violate power constraints at runtime, or a significant timing imbalance among grouped memories may yield suboptimal or even infeasible schedules [12, 17, 10, 29].

To address these challenges, this work proposes an integrated optimization framework that combines physical-aware memory grouping with power- and concurrency-constrained MBIST test scheduling. The proposed grouping method leverages layout information and memory attributes to automatically generate physically coherent MBIST groups through a non-parametric clustering approach. The scheduling method models each memory's test power and test duration. It formulates the resulting scheduling task as a Resource-Constrained Project Scheduling Problem (RCPSP), enabling the derivation of a power-feasible schedule that minimizes overall test time.

Experimental evaluations on several MPSoC blocks demonstrate that the proposed grouping approach achieves the minimum number of groups under specified physical constraints for all cases, outperforming K-means [18], greedy heuristics, and genetic algorithms. Moreover, the proposed scheduling algorithm consistently reduces the end-to-end MBIST test time compared to a commercial industrial MBIST solution, achieving up to 53% reduction in total test time. By jointly optimizing *grouping* (spatial and physical coherence) and *scheduling* (runtime feasibility under power constraints), the proposed method maximizes MBIST parallelism while respecting strict power limits, thus simultaneously improving multiple design objectives – including group count, power integrity, test application time, and EDA tool runtime – within a unified and practical MBIST optimization framework.

2 Related Work

Existing MBIST grouping and scheduling optimization methods can be broadly classified into four directions. The first is the *placement- and scheduling-integrated* approach, in which grouping, test scheduling, and logical placement are optimized simultaneously. Kang and Kahng [12] proposed a framework that cooperatively optimizes these aspects and pioneered the idea of considering testability already at the placement stage. However, the approach suffers from combinatorial explosion, which limits its applicability to large memory instances and makes it difficult to flexibly handle dynamic constraints such as power and distance.

The second direction consists of *constraint-specific and architecture-oriented* approaches. Rodrigo et al. [24] proposed a reconfigurable MBIST architecture that allows heterogeneous memories to share a common BIST controller and thereby reduces hardware resources, but the grouping algorithm itself is only coarsely optimized. Shoukourian et al. [20] introduced a grouping method that explicitly considers a test-power upper bound, yet it lacks the flexibility to simultaneously handle other physical or attribute-based factors.

The third direction is *scheduling-oriented* optimization, which mainly focuses on the execution order of tests. Padmini and Kriti [6] shortened test time by tailoring the schedule to memory characteristics, but did not consider grouping or power constraints. ShihHsu et al. [27] optimized MBIST grouping and controller placement while considering physical distances and clock domains, achieving test-time reduction, but their method relies on local search and thus has limitations in terms of global optimality and computational efficiency.

The fourth direction is *grouping-centric* optimization, with EMGA [16] as a representative example. EMGA separates hard and soft constraints, combines a greedy heuristic for initial solution generation with a genetic algorithm for refinement, and thereby achieves high-quality groupings and significant reductions in runtime. However, EMGA strictly focuses on the grouping problem itself and does not address test scheduling or runtime power constraints.

In summary, prior work has achieved improvements by focusing on individual aspects such as placement, power, distance, parallelism, and grouping quality. Nevertheless, a method that *jointly* optimizes grouping and scheduling, remains applicable to designs with hundreds to thousands of memories, and is computationally efficient in terms of tool runtime, has not yet been established. This work targets this gap and proposes a new MBIST framework that integrates the optimization of both grouping and scheduling.

3 Proposed Method

In this work, we construct a three-stage heuristic method based on an approach that exploits physical layout information and test constraints in a stepwise manner.

1. First, DBSCAN [7] is applied to the placement coordinates on the layout, and memories are clustered according to local density, thereby extracting global candidate regions.
2. Next, within each cluster, rectangular partitioning is repeatedly applied to generate rectangular groups that satisfy constraints on wiring distance and group size. These groups are then fixed as MBIST groups (i.e., units controlled by a single BIST controller).
3. Finally, based on the test time and test power of each memory estimated from the SRAM libraries, RCPSP-type test scheduling is performed to minimize the total test time under a global power limit and a constraint on the degree of MBIST parallelism.

3.1 Memory Grouping

In this work, we propose a two-stage clustering-based memory grouping method that simultaneously accounts for memory physical characteristics and placement constraints. Figure 1 illustrates the overall flow of the proposed memory grouping procedure. The method consists of the following three steps:

- Step 1: extraction of memory features,
- Step 2: non-parametric clustering (DBSCAN) that integrates geometric and shape information, and
- Step 3: a Rectangular Partitioning algorithm that subdivides each cluster into rectangular groups satisfying design constraints such as power and spatial proximity.

In this subsection, we describe each of these steps and the corresponding design rationale in detail.

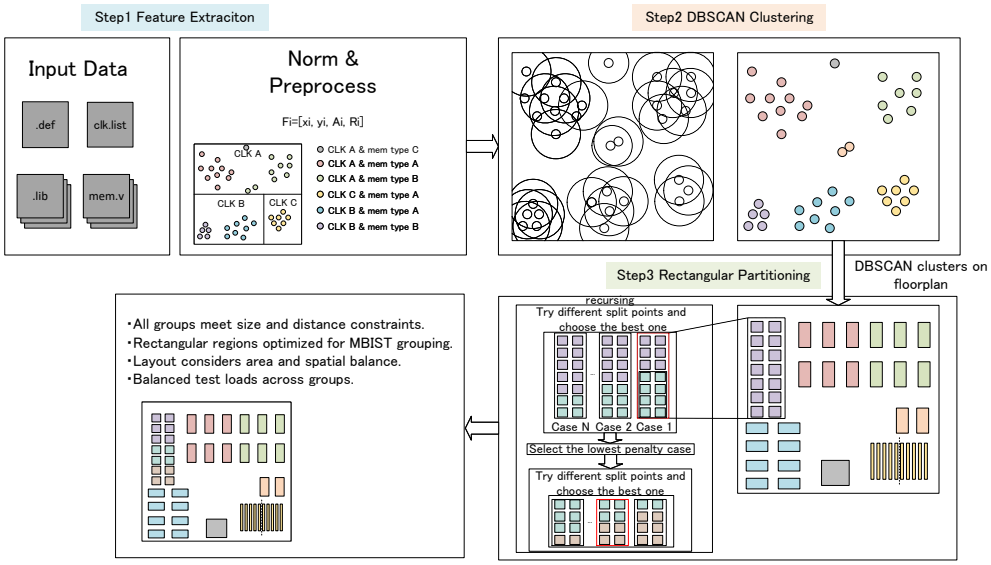

Figure 1 Example of memory grouping where multiple SRAMs share a single MBIST controller.

3.1.1 Feature Extraction

At this stage, we extract the physical parameters of each memory instance m_i from layout information, such as DEF files, and construct a feature vector for clustering. Here, m_i denotes the i-th memory instance, C_i its clock domain, T_i its memory type (SRAM macro type), (x_i, y_i) the coordinates of the instance center in the layout, A_i the cell area, R_i the aspect ratio (e.g., $R_i = \log(w_i/h_i)$), and P_i the power of the memory instance. From these physical continuous-valued quantities, we define the feature vector of memory m_i as

$$\mathbf{F}_i = \big(\underbrace{C_i, T_i}_{\text{discrete attrs}} \; ; \underbrace{x_i, y_i, A_i, R_i, P_i}_{\text{continuous features}} \big).$$

The coordinates (x_i, y_i), the area A_i, and the aspect ratio R_i are standardized along each dimension to have zero mean and unit variance, so that none of these features dominates the distance computation excessively, and the relative weighting among position, area, and shape can be indirectly adjusted.

In addition, to preserve consistency in terms of clock domains and macro types, we first partition the set of memories into subsets for each combination (C_i, T_i), and then independently apply DBSCAN and rectangular partitioning to each "clock domain × memory type" subset. As a result, the clustering process focuses on continuous position/area/shape information represented by (x_i, y_i, A_i, R_i) rather than discrete labels such as clock domains and types, which leads to a region partitioning that more faithfully reflects the physical placement of memories.

3.1.2 Density-Based Clustering and Rectangular Partitioning

Next, we describe our memory grouping method based on DBSCAN and rectangular partitioning.

We first summarize in Eq. (1) the local constraints that any pair of memories (m_i, m_j) belonging to the same group $\mathcal{G}_k$ must satisfy.

Here, m_i denotes the i-th memory instance. The attributes m_i.clk and m_i.type represent the clock domain C_i and the memory type T_i, respectively. The coordinates (x_i, y_i) denote the center of the instance in the layout, a_i denotes the cell area, and r_i denotes the aspect ratio. P_m is the (relative) test power of memory m, and $D_{\max}, N_{\max}, A^{\max}_{\text{area}}, R^{\max}_{\text{aspect}}$, and $P^{\max}_{\text{group}}$ are the thresholds for distance, maximum group size, area similarity, aspect-ratio similarity, and total power within a group, respectively.

$$\begin{cases} m_i.\text{clk} = m_j.\text{clk}, & \text{(clock domain consistency)} \\ m_i.\text{type} = m_j.\text{type}, & \text{(memory type consistency)} \\ \|(x_i, y_i) - (x_j, y_j)\| \le D_{\max}, & \text{(proximity constraint)} \\ |\mathcal{G}_k| \le N_{\max}, & \text{(group capacity limit)} \\ \dfrac{\max(a_i, a_j)}{\min(a_i, a_j)} \le A^{\max}_{\text{area}}, & \text{(area similarity)} \\ \left|R_i - R_j\right| \le R^{\max}_{\text{aspect}}, & \text{(aspect-ratio similarity)} \\ \displaystyle\sum_{m \in \mathcal{G}_k} P_m \le P^{\max}_{\text{group}}, & \text{(group power constraint)} \end{cases} \quad \forall\, \mathcal{G}_k. \tag{1}$$

We relate the grouping-stage power bound $P^{\max}_{\text{group}}$ to the chip-level instantaneous power limit $P_{\max}$ used in scheduling as follows. Since our scheduling targets activating up to $B_{\max}$ MBIST controllers in parallel, we set $P^{\max}_{\text{group}}$ so that even the worst-case concurrent execution of $B_{\max}$ groups stays within the chip-level budget:

$$B_{\max}\, P^{\max}_{\text{group}} \le P_{\max}. \tag{2}$$

In our experiments, we used $P_{\max} = 4.0$ and $B_{\max} = 2$, and thus set $P^{\max}_{\text{group}} = 2.0$. This setting directly affects achievable parallelism and makespan: an excessively large $P^{\max}_{\text{group}}$ may allow high-power memories to concentrate in a few groups, creating bottlenecks that reduce effective utilization of the $B_{\max}$ parallel resources, whereas an excessively small $P^{\max}_{\text{group}}$ may over-fragment groups, increasing the number of tasks and potentially inflating scheduling overhead. Therefore, we use $B_{\max} P^{\max}_{\text{group}} \le P_{\max}$ as a practical guideline to balance per-group power while facilitating parallel execution during scheduling.

Conventional distance-based clustering methods rely solely on limited information, such as memory coordinates and types. They therefore cannot adequately reflect placement information, such as differences in size and shape. To address this issue, in this work, we construct a multidimensional feature space based on the continuous-valued features (x_i, y_i, A_i, R_i) defined in the previous subsection and perform density-based clustering using DBSCAN in this space. This allows us to obtain initial clusters that simultaneously account for geometric proximity and similarity in area and aspect ratio.

In our method, memories labeled as noise in the standard DBSCAN output are relabeled as individual clusters, so that extremely isolated memories are retained as singleton groups. This post-processing yields a consistent initial clustering for both dense regions and isolated regions.

For neighborhood queries, we employ a spatial index structure (e.g., a k-d tree), which reduces the overall computational complexity of the DBSCAN phase to approximately $O(N \log N)$ [1] for N memories. As a result, for each subset of memories with the same clock domain and macro type, those that are similar in area and shape are automatically grouped into the same cluster. In contrast, special memories placed in isolated regions are

classified as independent clusters. In this way, the DBSCAN-based clustering can generate initial groupings that reflect physical placement and design characteristics, while remaining scalable even for large numbers of memories.

The initial clusters obtained by DBSCAN do not necessarily satisfy all test-related constraints summarized in Eq. (1). Therefore, for each cluster G_j, we recursively apply rectangular partitioning so that the following constraints in Eq. (1) become satisfied:

$$|G_j| \le N_{\max}, \qquad \max_{i,k \in G_j} d(i,k) \le D_{\max}, \qquad \sum_{m \in G_j} P_m \le P_{\text{group}}^{\max},$$

where $d(i,k)$ denotes the Euclidean distance between the centers of memories i and k.

Algorithm 1 shows the pseudocode of the rectangular partitioning procedure. The function RectPartition first checks whether the current group G already satisfies the above cardinality, distance, and power constraints; if so, it is returned as a single valid group. Otherwise, the bounding box of G is computed and the region is split along its longer side. For each split ratio $r \in \{0.4, 0.5, 0.6\}$, the group is divided into two subgroups (G_1, G_2) using Split, and RectPartition is applied recursively to each subgroup. For the resulting partition P, we consider it valid only if all groups satisfy the intra-group power constraint $\sum_{m \in G_j} P_m \le P_{\text{group}}^{\max}$. Among all valid candidates, we evaluate the cost

$$C = \alpha N_g + \beta \sigma_{\text{size}},$$

where N_g is the number of generated groups and σ_{size} is the relative standard deviation of group sizes, and select the partition with the minimum cost. By recursively repeating this process, we obtain rectangular memory groups that satisfy all constraints on group size, spatial extent, and total test power.

3.2 Test Scheduling

In this section, we describe a test scheduling method that, for the final memory groups obtained in the previous section, minimizes the total test time while satisfying both power constraints and the MBIST parallelism constraint. In our approach, each memory instance is modeled as a task characterized by its *test power* and *test time*, and the scheduling problem is formulated as a Resource-Constrained Project Scheduling Problem (RCPSP) over these tasks. In addition to this formulation, we employ a lightweight greedy algorithm based on the Longest Processing Time first (LPT) rule to generate high-quality test schedules efficiently.

3.2.1 Test-time and power model

First, we estimate the test power and test time for each memory instance from the Liberty file of the corresponding SRAM macro. We take the average value of the entries in the internal power table as a relative test-power index for that macro, and assign this value to all instances of the same macro. In this way, we define the test power P_i for each memory instance. The goal is not to obtain an accurate estimate of absolute power consumption, but rather to provide a relative measure that can be compared with the power limit introduced later.

For the test time, let depth_i denote the number of words (depth) of memory m_i, k_{march} the average number of accesses per word in the applied March test [26], and f_{test} the MBIST test clock frequency in MHz. Then, denoting the required number of test cycles by $N_{\text{cyc},i}$ and the test clock period by T_{clk}, we approximate the test time T_i as

$$T_i = N_{\text{cyc},i} \cdot T_{\text{clk}} = k_{\text{march}} \cdot \text{depth}_i \cdot T_{\text{clk}}.$$

Algorithm 1 Rectangular partitioning of a memory group.

```
function RectPartition(G)
    if G = ∅ then
        return ∅
    end if
    if SatisfiesConstraints(G) then
        return {G}                                   ▷ |G|, d(i,k), Σ P_m within bounds
    end if
    (bbox, dir) ← BoundingBoxAndDir(G)
    best ← {G}, bestCost ← ∞
    for r ∈ {0.4, 0.5, 0.6} do
        (G1, G2) ← Split(G, dir, r)
        if G1 = ∅ or G2 = ∅ then
            continue
        end if
        P ← RectPartition(G1) ∪ RectPartition(G2)
        if ViolatesPower(P) then
            continue
        end if
        c ← Cost(P)                                  ▷ αN_g + βσ_size
        if c < bestCost then
            bestCost ← c, best ← P
        end if
    end for
    return best
end function
```

By assigning the pair (P_i, T_i) to each memory instance m_i, we obtain a task representation that is convenient for the subsequent scheduling algorithm.

3.2.2 Formulation as an RCPSP

For each memory group obtained by the rectangular partitioning, we assign one MBIST controller and denote the group IDs by $g = 1, 2, \ldots, G$. Let $b(i)$ denote the group (BIST ID) to which memory m_i belongs, and let $\mathcal{T}$ be the set of tasks. Then each task $i \in \mathcal{T}$ is associated with the attributes $(P_i, T_i, b(i))$.

The scheduling problem considered here can be formulated as a Resource-Constrained Project Scheduling Problem (RCPSP) [9], where tasks are executed under two types of resource constraints, namely the test-power constraint and the MBIST-parallelism constraint, and the objective is to minimize the time until all tasks are completed. Let $\mathcal{A}(t)$ denote the set of tasks under test at time t. The power constraint can then be written as

$$\sum_{i \in \mathcal{A}(t)} P_i \;\leq\; P_{\max},$$

where $P_{\max}$ is the maximum allowable test power, given as a design parameter reflecting, for example, the capacity of the power-distribution network.

We also assume that there exists an upper bound $B_{\max}$ on the number of distinct MBIST controllers (i.e., distinct BIST IDs) that can operate simultaneously, and impose the following parallel-BIST constraint:

$$|\{\, b(i) \mid i \in \mathcal{A}(t)\}| \;\le\; B_{\max}.$$

Here, $B_{\max}$ denotes the maximum number of MBIST controllers that can be active simultaneously; in the experimental evaluation of this work, we set $B_{\max} = 2$.

Each task is assumed to be non-preemptive and is executed continuously between its start time S_i and completion time C_i. The objective function is to minimize the makespan, i.e., the maximum completion time over all tasks,

$$C_{\max} = \max_{i \in \mathcal{T}} C_i.$$

Since RCPSP is generally NP-hard, we do not pursue exact optimization in this work and instead apply a greedy heuristic described in the following subsection.

3.2.3 LPT-based greedy scheduling

The proposed scheduler is implemented as an event-driven algorithm that alternates between two phases: (i) a phase in which all tasks that can be started at the current time t are launched, and (ii) a phase in which, if no further task can be started, the time is advanced to the next task-completion event. Throughout the procedure, the algorithm maintains the sets of unscheduled and running tasks.

At each time t, the scheduler first computes the total test power consumed by the running tasks and the set of active BIST controllers, and derives the residual power from the remaining budget. It then extracts from the unscheduled tasks a *candidate set* consisting only of tasks that satisfy the following two conditions: (i) their test power fits within the residual power budget, and (ii) they either belong to a BIST controller that is already active, or activating a new BIST controller for them does not violate the parallel-BIST limit. The tasks in this candidate set are sorted in descending order of test time T_i (and in descending order of power P_i in case of ties), and are launched one by one following the LPT rule, while checking the power and BIST constraints.

If the candidate set becomes empty and no additional task can be started, the current time is advanced to the completion time of the earliest finishing task among the running tasks, and that task is removed from the set of running tasks. By repeating these "task-start" and "time-jump" steps until both the unscheduled and running task sets become empty, we obtain the start and completion times (S_i, C_i) for all tasks as well as the final makespan $C_{\max}$. Considering the sorting of candidate sets and the number of events, the computational complexity is estimated to be on the order of $O(N^2 \log N)$ with respect to the number of tasks N, which allows practical scheduling times even for SoCs with several hundred to several thousand memory instances.

The pseudocode of the proposed scheduler is shown in Algorithm 2.

Figure 2 shows an example of a schedule generated by the proposed scheduler for the first four MBIST controllers in a test case with 1,523 SRAM instances. The horizontal axis represents test time, and the vertical axis represents the total test power of the active memories at each time. Since power values express only relative relationships among memories, they have no physical units. Each color corresponds to a single MBIST controller. Under the maximum power budget of $P_{\max} = 4.0$, the schedule satisfies the constraint that the instantaneous total test power never exceeds the upper limit. At the same time, the number of concurrently active MBIST controllers is kept no greater than $B_{\max} = 2$ at any time.

Algorithm 2 LPT-based MBIST test scheduler.

function LPT_SCHEDULE($\mathcal{T}, P_{\max}, B_{\max}$)
  $t \leftarrow 0$
  $U \leftarrow \mathcal{T}$ ▷ unscheduled tasks
  $A \leftarrow \emptyset$ ▷ active tasks
  while $U \neq \emptyset$ **or** $A \neq \emptyset$ **do**
    $B_{\text{act}} \leftarrow \{\, b(i) \mid i \in A \,\}$
    $P_{\text{use}} \leftarrow \sum_{i \in A} P_i$
    $P_{\text{rem}} \leftarrow P_{\max} - P_{\text{use}}$
    $C \leftarrow \{\, i \in U \mid P_i \le P_{\text{rem}} \wedge (\, b(i) \in B_{\text{act}} \vee |B_{\text{act}}| < B_{\max} \,) \,\}$
    if $C \neq \emptyset$ **then**
      sort C in non-increasing order of (T_i, P_i) ▷ LPT with power tie-break
      for all $i \in C$ **do**
        $B_{\text{act}} \leftarrow \{\, b(j) \mid j \in A \,\}$
        $P_{\text{use}} \leftarrow \sum_{j \in A} P_j$
        $P_{\text{rem}} \leftarrow P_{\max} - P_{\text{use}}$
        if $P_i \le P_{\text{rem}}$ **and** $(\, b(i) \in B_{\text{act}} \vee |B_{\text{act}}| < B_{\max} \,)$ **then**
          $S_i \leftarrow t$; $C_i \leftarrow t + T_i$
          $A \leftarrow A \cup \{i\}$; $U \leftarrow U \setminus \{i\}$
        end if
      end for
    else
      $t \leftarrow \min\{\, C_i \mid i \in A \,\}$ ▷ jump to next finish
      $A \leftarrow \{\, i \in A \mid C_i > t \,\}$ ▷ remove finished tasks
    end if
  end while
  $C_{\max} \leftarrow \max\{\, C_i \mid i \in \mathcal{T} \,\}$
  return $\{\, (S_i, C_i) \mid i \in \mathcal{T} \,\}, C_{\max}$
end function

4 Assessment

We evaluated the proposed method using four MPSoC blocks synthesized with the ASAP7 PDK [5] (case0–case3), containing 83, 105, 701, and 1523 SRAM macros, respectively. All experiments were conducted on an Intel Core i9-10940X machine (32 GB RAM, Ubuntu 22.04) using a C-based implementation.

4.1 Evaluation results

Table 1 summarizes the number of groups and runtime for the four memory-grouping methods over the four test cases.

For the grouping baselines, Greedy forms a group by selecting one unassigned memory as a seed and then scanning candidate memories in ascending order of physical proximity (layout distance). A candidate is added to the current group only if it satisfies the clock-domain and memory-type consistency and the thresholds defined in Eq. (1) (distance $D_{\max}$, maximum group size $N_{\max}$, area similarity $A^{\max}_{\text{area}}$, aspect-ratio similarity $R^{\max}_{\text{aspect}}$, and group power limit $P^{\max}_{\text{group}}$). In GA, each individual is represented as an assignment vector (a sequence of group IDs for all memories). The assignment is updated via crossover and mutation,

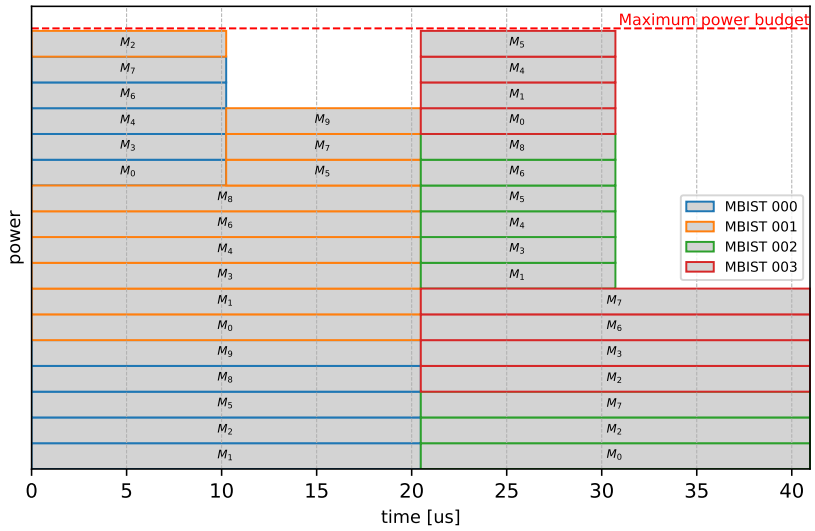

Figure 2 Example test schedule generated by the proposed LPT-based MBIST scheduler under $P_{\max} = 4.0$ and $B_{\max} = 2$.

Table 1 Comparison of the number of groups and runtime for different memory-grouping methods. The numbers of SRAM macros (#SRAM) for case0–case3 are 83, 105, 701, and 1523, respectively.

	case0		case1		case2		case3	
Method	#groups	run time [s]	#groups	run time [s]	#groups	run time [s]	#groups	run time [s]
K-means	60	0.035	39	0.036	106	1.705	321	10.5
Greedy	60	0.013	39	0.008	107	0.174	324	5.3
GA	65	1.902	51	3.349	181	26.702	125	63.1
Ours	23	0.011	20	0.007	94	0.308	121	10.0

and grouping is obtained by optimizing an objective function that primarily encourages fewer groups and higher intra-group proximity, while penalizing constraint violations (the above thresholds). If necessary, a simple repair step (e.g., splitting an infeasible group) is applied to obtain a feasible solution. The constraint thresholds such as $D_{\max}$ are set based on design requirements (e.g., wiring length, allowable parallelism, power budget, and tolerated macro-shape deviation), and the GA search parameters (population size, number of generations, crossover rate, mutation rate, and penalty weights) are chosen via preliminary experiments to balance convergence and runtime.

We first present the memory grouping results. The proposed method (Ours) achieved significantly fewer clusters than the existing approaches in almost all test cases, while also completing the computation with shorter runtime. In particular, for case2 (701 memories), our method was approximately 87× faster than the GA-based approach (26.7 s → 0.31 s), and reduced the number of clusters from 181 to 94 (48% reduction). These results demonstrate that the proposed method can achieve high-quality and efficient grouping without relying on evolutionary optimization.

For test scheduling, we compared our method with the Industrial Solution of a commercial MBIST tool using the same design data; to focus on the difference in scheduling strategies, we do not use a commercial/industrial tool for memory grouping and instead provide the same rectangular-partition-based MBIST grouping produced by our method as a common

Table 2 Scheduling results for the four MPSoC blocks (case0–case3). $C_{\max}$ denotes total test time (makespan).

	case0		case1		case2		case3	
Method	Test time	run time [s]	Test time	run time [s]	Test time	run time [s]	Test time	run time [s]
Industrial Solution	0.23	0.67	0.082	0.41	0.78	2.51	2.394	3.5
Ours	0.21	0.13	0.092	0.10	0.63	0.80	1.137	2.3

input to both approaches. Both schedules are generated under identical constraints (a power limit of $P_{\max} = 4.0$ and a maximum parallel BIST count of $B_{\max} = 2$). In the Industrial Solution, test scheduling is performed by the commercial MBIST tool, which estimates the allowable parallelism (i.e., the number of concurrently active MBIST controllers) under the power budget and adjusts the execution order and the set of tests executed in parallel (test plan) so as to avoid instantaneous power spikes. For both the proposed method and the Industrial Solution, the test time is estimated by gate-level simulation on the post-layout netlist, and we report the total test time $C_{\max}$ and the scheduling runtime (CPU time) for each case as evaluation metrics.

The results are summarized in Table 2. The proposed method achieved shorter scheduling runtime than the Industrial Solution in all cases, with a 4–5× speedup for case0 and case1. In terms of test time, our method reduced $C_{\max}$ by approximately 9%, 19%, and 53% for case0, case2, and case3, respectively, showing larger improvements for larger designs. Although case1 exhibits a slight increase in test time, the scheduling runtime is significantly reduced, enabling comparable turnaround time with much lower overhead even for small blocks.

4.2 Integrated optimization with grouping

The proposed scheduling method is designed to work with the memory grouping obtained through rectangular partitioning. Each group produced by the rectangular partitioning naturally satisfies key static constraints – including physical proximity, limited group size, bounded maximum distance, and total in-group test power – while also reducing wiring length. This allows the scheduler to treat each group as a single BIST resource and focus solely on optimizing inter-group power allocation and parallel execution.

During scheduling, task start times are determined under the power limit $P_{\max}$ and the maximum parallel BIST constraint $B_{\max}$. Our scheduler improves $C_{\max}$ mainly by packing executions more densely within these limits: it greedily launches feasible tasks without exceeding $P_{\max}$ and prioritizes longer tasks (LPT) to shorten the end-stage tail, thereby reducing both power idling and resource idling of the $B_{\max}$ BIST slots. In contrast, a commercial scheduler may leave additional headroom below $P_{\max}$ to avoid power spikes, increasing idle intervals and lengthening the makespan. The in-group power bound $P_{\text{group}}^{\max}$ introduced during rectangular partitioning further suppresses load imbalance among BIST groups, helping maintain high utilization of up to $B_{\max}$ parallel BIST controllers.

Across four real MPSoC blocks, the proposed scheduler achieved shorter scheduling runtime than the Industrial Solution and significantly reduced total test time in medium- and large-scale designs. Even for small designs, it achieved comparable test time with much lower scheduling overhead. These results demonstrate that combining static constraint organization during grouping with dynamic constraint optimization during scheduling provides an effective and practical MBIST optimization framework that outperforms existing methods.

5 Conclusion

This paper presented an integrated MBIST optimization framework that combines a rectangular-partition-based memory grouping method with an LPT-based RCPSP scheduler. The proposed grouping technique generates physically coherent memory clusters that satisfy static constraints such as distance, area, and in-group power, while the scheduler focuses on optimizing dynamic constraints, including power limits and BIST parallelism to minimize total test time.

Evaluation using four real MPSoC blocks demonstrated that the proposed grouping consistently outperforms existing approaches (K-means, Greedy, GA) by producing fewer groups with shorter runtime. In addition, the proposed scheduler achieved significantly shorter scheduling runtime than a commercial MBIST tool (Industrial Solution) and reduced total test time by up to 53% for medium- and large-scale designs.

Overall, the results show that the proposed method is an effective and practical solution for jointly optimizing memory grouping and test scheduling, offering high scalability and applicability for future large-scale MPSoC MBIST designs.

References

1 Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. *ACM Sigmod record*, 28(2):49–60, 1999. doi:10.1145/304182.304187.

2 Krishnendu Chakrabarty. Design of system-on-a-chip test access architectures under place-and-route and power constraints. In *Proceedings of the 37th Annual Design Automation Conference*, pages 432–437, 2000. doi:10.1145/337292.337531.

3 Krishnendu Chakrabarty. Test scheduling for core-based systems using mixed-integer linear programming. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 19(10):1163–1174, 2002.

4 Tzuo-Fan Chien, Wen-Chi Chao, Chien-Mo Li, Yao-Wen Chang, Kuan-Yu Liao, Ming-Tung Chang, Min-Hsiu Tsai, and Chih-Mou Tseng. Bist design optimization for large-scale embedded memory cores. In *Proceedings of the 2009 International Conference on Computer-Aided Design*, pages 197–200, 2009.

5 Lawrence T Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandarasekaran Ramamurthy, and Greg Yeric. Asap7: A 7-nm finfet predictive process design kit. *Microelectronics Journal*, 53:105–115, 2016. doi:10.1016/j.mejo.2016.04.006.

6 Kriti Sundar Das and Padmini Prakash. Automatic mbist scheduling engine. In *2019 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, pages 1–6. IEEE, 2019.

7 Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96(34), pages 226–231, 1996.

8 Kunal Ganeshpure and Sandip Kundu. A dft methodology for repairing embedded memories of large mpsocs. In *2012 IEEE Computer Society Annual Symposium on VLSI*, pages 108–113. IEEE, 2012. doi:10.1109/ISVLSI.2012.17.

9 Sönke Hartmann and Dirk Briskorn. An updated survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of operational research*, 297(1):1–14, 2022. doi:10.1016/j.ejor.2021.05.004.

10 Shou-Yi Huang and Shih-Hsu Huang. Memory grouping for the built-in self-test of three-dimensional integrated circuits. *Electronics*, 13(18):3759, 2024.

11 Preethy K John et al. Bist architecture for multiple rams in soc. *Procedia computer science*, 115:159–165, 2017.

12 Andrew B Kahng and Ilgweon Kang. Co-optimization of memory bist grouping, test scheduling, and logic placement. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2014. doi:10.7873/DATE.2014.209.

13 Josef Kinseher. *New Methods for Improving Embedded Memory Manufacturing Tests*. PhD thesis, Universität Passau, 2018.

14 Yen-Chun Ko and Shih-Hsu Huang. 3d ic memory bist controller allocation for test time minimization under power constraints. In *2017 IEEE 26th Asian Test Symposium (ATS)*, pages 260–265. Ieee, 2017. doi:10.1109/ATS.2017.56.

15 Aman Kokrady, CP Ravikumar, and Nitin Chandrachoodan. Layout-aware and programmable memory bist synthesis for nanoscale system-on-chip designs. In *2008 17th Asian Test Symposium*, pages 351–356. IEEE, 2008.

16 Yang Li, Yongqiang Duan, Hao Zhang, Dan Niu, Xiao Wu, and Zhou Jin. Emga: An evolutionary memory grouping algorithm for mbist. In *2024 2nd International Symposium of Electronics Design Automation (ISEDA)*, pages 492–497. IEEE, 2024.

17 Zhixing Liu, Jielin Xu, Jing Tang, Song Yang, and Hailong You. An efficient grouping algorithm with build-in-self-test for multiple memories. In *2024 2nd International Symposium of Electronics Design Automation (ISEDA)*, pages 444–449. IEEE, 2024.

18 Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982. doi:10.1109/TIT.1982.1056489.

19 Narek Mamikonyan et al. Multi-memory grouping wrapper with top level bist algorithm. *Open Access Library Journal*, 7(05):1, 2020.

20 L Martirosyan, Gurgen Harutyunyan, S Shoukourian, and Yervant Zorian. A power based memory bist grouping methodology. In *2015 IEEE East-West Design & Test Symposium (EWDTS)*, pages 1–4. IEEE, 2015.

21 Masahide Miyazaki, Tomokazu Yoneda, and Hideo Fujiwara. A memory grouping method for reducing memory bist logic of system-on-chips. *IEICE transactions on information and systems*, 89(4):1490–1497, 2006. doi:10.1093/ietisy/e89-d.4.1490.

22 Masahide Miyazaki, Tomokazu Yoneda, and Hideo Fujiwara. A memory grouping method for sharing memory bist logic. In *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, pages 671–676, 2006. doi:10.1109/ASPDAC.2006.1594763.

23 Chia Yee Ooi, Jia Pao Sua, and Siaw Chen Lee. Power-aware system-on-chip test scheduling using enhanced rectangle packing algorithm. *Computers & Electrical Engineering*, 38(6):1444–1455, 2012. doi:10.1016/j.compeleceng.2012.04.010.

24 Reinaldo Silveira, Qadeer Qureshi, and Rodrigo Zeli. Flexible architecture of memory bists. In *2018 IEEE 19th Latin-American Test Symposium (LATS)*, pages 1–6. IEEE, 2018. doi:10.1109/LATW.2018.8349666.

25 Balwinder Singh, Arun Khosla, and Sukhleen Bindra Narang. Area overhead and power analysis of march algorithms for memory bist. *Procedia Engineering*, 30:930–936, 2012.

26 Ad J Van De Goor. Using march tests to test srams. *IEEE Design & Test of Computers*, 10(1):8–14, 2002.

27 Chang-Han Yeh, Chun-Hua Cheng, and Shih-Hsu Huang. Grouping and placement of memory bist controllers for test application time minimization. In *2016 5th International Symposium on Next-Generation Electronics (ISNE)*, pages 1–2. Ieee, 2016.

28 Lilia Zaourar, Jihane Alami Chentoufi, Yann Kieffer, Arnaud Wenzel, and Frederic Grandvaux. A shared bist optimization methodology for memory test. In *2010 15th IEEE European Test Symposium (ETS)*, pages 255–255, 05 2010. doi:10.1109/ETSYM.2010.5512736.

29 Rodrigo Zeli, Reinaldo Silveira, and Qadeer Qureshi. Soc memory test optimization using nxp mtr solutions. In *2019 IEEE Latin American Test Symposium (LATS)*, pages 1–5. IEEE, 2019. doi:10.1109/LATW.2019.8704566.

30 Yervant Zorian. Test requirements for embedded core-based systems and ieee p1500. In *Proceedings International Test Conference 1997*, pages 191–199. IEEE, 1997. doi:10.1109/TEST.1997.639613.