Co-Optimization of Memory BIST Grouping, Test Scheduling, and Logic Placement

Andrew B. Kahng[†‡] and Ilgweon Kang[‡]
UC San Diego ECE[†] and CSE[‡] Departments, La Jolla, CA 92093
{abk, igkang}@ucsd.edu

Abstract—**Built-in self-test (BIST) is a well-known design technique in which part of a circuit is used to test the circuit itself. BIST plays an important role for embedded memories, which do not have pins or pads exposed toward the periphery of the chip for testing with automatic test equipment. With the rapidly increasing number of embedded memories in modern SOCs (up to hundreds of memories in each hard macro of the SOC), product designers incur substantial costs of test time (subject to possible power constraints) and BIST logic physical resources (area, routing, power). However, only limited previous work addresses the physical design optimization of BIST logic; notably, Chien et al. [7] optimize BIST design with respect to test time, routing length, and area. In our work, we propose a new three-step heuristic approach to minimize test time as well as test physical layout resources, subject to given upper bounds on power consumption. A key contribution is an integer linear programming ILP framework that determines optimal test time for a given cluster of memories using either one or two BIST controllers, subject to test power limits and with full comprehension of available serialization and parallelization. Our heuristic approach integrates (i) generation of a hypergraph over the memories, with test time-aware weighting of hyperedges, along with top-down, FM-style min-cut partitioning; (ii) solution of an ILP that comprehends parallel and serial testing to optimize test scheduling per BIST controller; and (iii) placement of BIST logic to minimize routing and buffering costs. When evaluated on hard macros from a recent industrial 28nm networking SOC, our heuristic solutions reduce test time estimates by up to 11.57% with strictly fewer BIST controllers per hard macro, compared to the industrial solutions.**

I. Introduction

In modern SOCs, embedded memories (normally, SRAM blocks) can account for more than 50% of die area [25]. Since a defect in embedded memory can make the entire chip fail, design for test (DFT) techniques for embedded memory are essential. *Built-in self-test* (BIST) is an increasingly effective and necessary DFT technique in which part of a circuit is used to test the circuit itself [1]. In particular, BIST is now ubiquitous for embedded memories, which do not have pins or pads exposed for testing with automated test equipment (ATE).

Memory BIST affects design quality and chip cost in several basic ways.

- BIST controller logic occupies silicon real estate, and contributes to die area, leakage power, and routing congestion. All else being equal, the fewer BIST controller blocks, the better.
- The widths and depths of embedded memories assigned to a given BIST controller must be "packed" into a feasible test schedule that minimizes test time subject to maximum power constraints. The test time directly impacts product cost and is a first-class design consideration, especially in a design with many memories.
- The physical placement of a BIST controller logic block relative to its associated memory blocks affects not only routability, but also the signal delay between the controller and the memories. Larger distances force the use of more buffering and lower-V_T devices to meet timing and electrical constraints; this costs more power.

From these considerations, it is apparent that the co-optimization of physical design resources, test power, leakage power, and test time falls between front-end DFT groups and back-end physical design groups. On the one hand, a floorplan-oblivious partitioning of memories to BIST controllers might force use of low-V_T (LVT) cells to meet timing requirements. On the other hand, the physical design (PD) engineer's suggested partitioning may lead to less congestion, routing cost, and signal delay between memories and BIST logic, but with dramatically increased test time.

In this paper, we describe a heuristic optimization that smooths the interactions between front-end DFT and back-end PD, reducing iterations and schedule costs. Our heuristic minimizes test time as well as test physical layout resources, subject to given upper bounds on power consumption. A new integer linear program (ILP) formulation finds the optimal test time for a given cluster of memories using either one or two BIST controllers, taking full advantage of any available serialization and parallelization of the memory self-test. When evaluated on hard macros from a recent industrial 28nm networking SOC, our heuristic solutions reduce test time estimates by up to 11.57% with strictly fewer BIST controllers per hard macro, compared to the industrial solutions.

Our main contributions can be summarized as follows.

- We propose a weighted hypergraph construction that allows use of top-down min-cut partitioning of memories into clusters that have good physical design and test scheduling attributes.
- We propose an ILP that comprehends parallel and serial testing of a given group of memories as it finds a minimum-test time solution with one or two BIST controllers.
- We use the above two elements, along with bottleneck matching to find BIST logic placement locations, in a heuristic that simultaneously reduces both BIST logic and test time costs in hard macros from a recent 28nm networking SOC.

In the remainder of this paper, Section II briefly reviews related works in the areas of test scheduling and memory BIST. Section III describes our ILP formulation to minimize test time taking advantage of available serialization and parallelization. Section IV presents our heuristic approach, and Section V gives experimental results with industrial testcases. Section VI describes directions of ongoing work and concludes the paper.

II. Related Works

In this section, we broadly classify related literature as dealing with (1) test scheduling and (2) BIST controller optimizations.

A. Test Scheduling

Test time reduction has long been a basic goal of DFT research, since test time is directly related to test cost. Parallel (simultaneous) testing reduces test time but is constrained by power and bandwidth (pin count) limits. Works such as that of Yao et al. [20], formulate and solve the test scheduling problem to minimize total test time while satisfying such constraints. Iyengar et al. [12] [13] adapt a rectangle packing problem formulation to test scheduling; they co-optimize test access mechanism (TAM) architecture and test wrapper, while designating a group of tests. Zou et al. [21] formulate SOC test scheduling as two-dimensional bin packing under given pin constraints, and simulated annealing is used to search for a heuristic optimum test schedule by perturbations to an initial solution.

Other researchers have applied integer linear programming (ILP) to find optimal test schedules under constraints [5] [6] [8] [15]. Chakrabarty [5] proposes test access architectures that incorporate place-and-route constraints arising from interconnections. [6] uses mixed integer-linear programming (MILP) to optimize test schedules for core-based systems; a heuristic algorithm efficiently solves larger problem instances for which the MILP approach has excessive runtime. Liu et al. [15] apply ILP formulation for NOC instances. Chin and Nourani [8] propose a flexible ILP-based test scheduling environment with many user options.

Wang et al. [19] develop a test scheduling algorithm based on elements of the March algorithm for memory BIST; the objective is to minimize overall testing time under a power constraint.

Unlike previous works, we study the minimization of total test time in the context of a *mixture of serial and parallel testing, with multiple memory BIST controllers, by considering physical information*

978-3-9815370-2-4/DATE14/©2014 EDAA

of memories.[1] We note that most of the previous literature on test scheduling addresses scheduling for logic cores, where the testing is mainly performed by scan chain techniques. By contrast, we address embedded memory testing using multiple memory BIST controllers, where the memories have different sizes, test times, and test power values.

B. *Design Optimizations for Memory BIST Controllers*

Most works in the memory BIST literature focus on architectural and testing aspects, even though design optimization of memory BIST can provide substantial benefits to the entire chip design and to test quality. To our knowledge, relatively few works exist in the realm of (physically-aware) design optimization of memory BIST.[2]

A memory grouping method for sharing memory BIST logic is proposed by Miyazaki et al. in [17]. Area overhead reductions are achieved by the grouping of memories for parallel and serial testing. Devanathan et al. [9] propose a physically-aware memory BIST datapath synthesis framework, wherein a hierarchical synthesis approach achieves correct-by-construction, area-efficient memory BIST solutions. Devanathan et al. demonstrate the benefits from strategic approaches to physically-aware BIST in [11] and built-in self-repair (BISR) design optimization methods in [10]: such techniques mitigate the difficulties of physical design closure such as congestion and timing closure, even as the numbers of memory instances and BIST controllers in complex SOCs continue to increase. The authors of [10] [11] also note that their methods enable designers to apply more effective tests and reduce verification cycle times.

Chien et al. [7] propose a memory BIST design optimization method to minimize test time, wire length and total area while considering several practical design constraints. To our knowledge, [7] is the first published work considering aspects of physical design for memory BIST controllers. The authors adopt an integer linear programming (ILP) formulation for the assignment of memories to controllers. They then apply legalization and refinement steps to meet user-specified constraints and to further improve the quality of their solution. Although [7] is the previous work that is closest to ours, we observe that it makes a number of simplifications that we avoid, e.g., (i) all memory instances in a BIST cluster are tested in parallel (leading to an unrealistic test time estimate); and (ii) only one cluster is tested at a time (preventing exploitation of parallel testing with multiple BIST controllers).[3]

III. ILP Formulation

We develop an integer linear program (ILP) to solve the memory test scheduling problem when using multiple BIST controllers. Note that our ILP formulation is very different from those of [5] [6] [8] [15] since we use logical constraints to define parallel and serial testing. Table I defines notations used in our discussion. The objective is to minimize total test time, i.e.,

$$\text{minimize} \quad \max_{\forall m_i} T_{E_i} \tag{1}$$

where $T_{E_i} = T_{S_i} + T_{D_i}$, and $T_{S_i} \geq 0$. We assume that a memory has test time proportional to its depth [17] and test power proportional to the square root of its size. Based on our studies, we see that allowing both serial and parallel testing of memories can reduce test time as illustrated in Figure 1.

[1]The mixture of serial and parallel testing induces what may be thought of as a *partial level-oriented strip packing problem*. (In the two-dimensional strip packing problem, (rectangular) items are packed into an "open-ended" rectangle of given height and infinite width, and the objective is to minimize width while packing all of the items into the rectangle [2].) To our knowledge, the DFT literature has not yet considered this partial level-oriented strip packing formulation.

[2]This being said, the test cost and the area/power overheads of memory BIST are rapidly drawing more attention to this topic.

[3]In [7], the estimation of test time without consideration of test scheduling leads to unnecessary expense of test time when there is power slack below the power constraint. Further, the placement of BIST logic at median x- and y-coordinates of all memory instances in [7] is oblivious to the underlying min-weight maximum-matching problem when path timing is considered.

TABLE I: Notations.

Term	Meaning
M	Set of memory instances
m_i	i^{th} memory instance, where $1 \leq i \leq \lvert M \rvert$
x_i	Size of word in m_i
y_i	Number of words in m_i
B	Set of memory BIST controllers
b_k	k^{th} memory BIST controller, where $1 \leq k \leq \lvert B \rvert$
P^k	Set of partitions, (k-way partitioning)
p_i	i^{th} partition
$D(p_i)$	Diameter of the i^{th} partition p_i
T_{S_i}	Test start time of m_i
T_{E_i}	Test end time of m_i
T_{D_i}	Test duration of m_i
t_q	Instantaneous time
$E(m_i)$	Test power of m_i
$E(t_q)$	Total test power at time t_q
E_{MAX}	Upper bound on total test power
$U_i(t_q)$	Indicator whether m_i is under testing at time t_q
$V_i(t_q)$	Indicator whether $t_q \geq T_{S_i}$
$W_i(t_q)$	Indicator whether $t_q \leq T_{E_i}$
$B_{k,i}$	Indicator whether m_i is tested with BIST controller b_k
$I_{k,i,j}$	Indicator whether m_i and m_j belong to the same BIST controller b_k
$L_{k,i,j}$	Indicator whether m_i and m_j are tested in parallel with the same BIST controller b_k
$F_{k,i,j}$	Indicator whether m_i is tested before starting test of m_j with the same BIST controller b_k
$Q_{k,i,j}$	Indicator whether $T_{S_i} \leq T_{S_j}$
N_L	Large integer
N_{LL}	Large integer $N_{LL} \gg N_L$
ε	Positive and very small real number, $0 < \varepsilon \ll 1$

The ILP constraints are as follows.

Maximum power constraint. We use E_{MAX} to denote an upper bound on maximum available test power. The instantaneous testing power $E(t_q)$ cannot exceed E_{MAX}, as indicated by constraint (2). $E(t_q)$ is the sum of test power consumption for all memory instances m_i being tested at time t_q, as shown in Equation (3), where $U_i(t_q)$ indicates whether m_i is being tested at time t_q, and $E(m_i)$ is the test power of m_i. The constraint (4) ensures that all memories must be tested to obtain a valid solution.

$$E(t_q) \leq E_{MAX} \tag{2}$$

$$E(t_q) = \sum_{m_i \in M} U_i(t_q) \cdot E(m_i)$$

$$\text{where } U_i(t_q) = \begin{cases} 1, & T_{S_i} \leq t_q < T_{E_i} \\ 0, & otherwise \end{cases} \tag{3}$$

$$\forall m_i, \ \sum_{\forall t_q} U_i(t_q) \geq 1 \tag{4}$$

BIST assignment constraint. We use the constraint (5) to ensure that each memory is uniquely assigned to a BIST controller for testing.

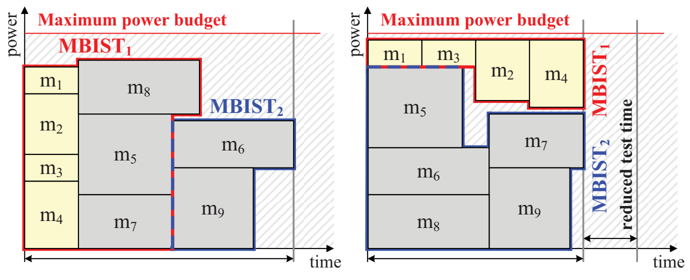


Fig. 1: Example with nine memories and two BIST controllers showing test time reduction when both serial and parallel testing are allowed. The left figure shows test time when only parallel testing is allowed. The right figure shows the reduced test time by allowing both serial and parallel testing.

$B_{k,i}$ indicates whether m_i is assigned to BIST controller b_k for testing.

$$\sum_{\forall b_k} B_{k,i} = 1, \quad m_i \in M$$
$$\text{where } B_{k,i} = \begin{cases} 1, & \text{if } m_i \text{ assigned to } b_k \\ 0, & \textit{otherwise} \end{cases} \tag{5}$$

Scheduling constraint. We define three indicator variables $I_{k,i,j}$, $F_{k,i,j}$, and $L_{k,i,j}$ to constrain the order of testing between two memories m_i and m_j that are assigned to the same controller b_k. These ensure that m_i and m_j are tested either in series or in parallel.

$I_{k,i,j}$ indicates whether m_i and m_j share the same BIST controller b_k, and has a value of zero when this is true, as shown in Equation (6). If $I_{k,i,j} = 0$ (i.e., m_i and m_j share the same BIST controller),

- $F_{k,i,j}$ indicates whether m_i is tested before m_j when tested serially; or
- $L_{k,i,j}$ indicates whether m_i and m_j are tested in parallel,

as shown in Equations (6)–(8). When $I_{k,i,j} = 1$, there is no scheduling relationship between m_i and m_j.

$$\mathrm{I}_{k,i,j} = \begin{cases} 0, & (B_{k,i} = 1) \wedge (B_{k,j} = 1) \\ 1, & \textit{otherwise} \end{cases} \tag{6}$$

$$\mathrm{F}_{k,i,j} = \begin{cases} 1, & (B_{k,i} = B_{k,j} = 1) \wedge (T_{E_i} \leq T_{S_j}) \\ 0, & \textit{otherwise} \end{cases} \tag{7}$$

$$(L_{k,i,j} + F_{k,i,j} + F_{k,j,i}) \cdot (1 - I_{k,i,j}) = 1 \text{ ,where } i \neq j \tag{8}$$

IV. Co-optimization of Test Scheduling and Memory BIST Logic Placement

We now describe our heuristic methodology for the co-optimization of test scheduling and memory BIST logic placement. Modern semiconductor chips contain hundreds of embedded memories scattered across the entire die. These memories can have various widths and depths, and can belong to different clock and logic hierarchies. Both the number and complexity of memory instances make the test scheduling problem extremely hard. We utilize a "divide-and-conquer" approach to develop a three-step heuristic method that (1) initially partitions all memories based on physical information using *MLPart* [4] [24]; (2) solves the test scheduling problem using an ILP formulation, followed by additional partitioning for better test time optimization; and (3) places memory BIST logic for each partition to minimize wirelength between memory BIST logic and memories. The goals of our heuristic approach are (1) minimization of test time, (2) reduction of number of partitions (i.e., number of BIST controllers), and (3) minimization of wirelength between each BIST and memories. We have developed a solver that uses command-line options as shown in Table II. Algorithms 1–3 outline our heuristic modeling approach.

TABLE II: MBIST solver command-line options.

-numMaxP	Maximum possible number of partitions for the design
-numMinP	Minimum required number of partitions for the design
-maxMemP	Upper bound of number of memories in a partition
-minMemP	Lower bound of number of memories in a partition
-maxD	Maximum allowed diameter for a partition (μm)
-conP	Power constraint
-gridS	Size of grid cell (μm) for BIST logic placement
-t	Tolerance (%) for balanced partitioning
-longD	Longer diameter criterion for edge weight K_4
-shortD	Shorter diameter criterion for edge weight K_6

A. Memory Partitioning

Memory partitioning is the "divide" step in our heuristic approach. We divide memory instances into k partitions using *MLPart* [4] [24], a min-cut hypergraph partitioner based on the multilevel Fiduccia-Mattheyses hypergraph partitioning [24] algorithm. The input to *MLPart* is a hypergraph G, where each node in G corresponds to a memory in the design (Algorithm 2). We define edge weights based on parameters such as memory shape, depth, power, location, etc. We expect that partitioning memories that have the same shape or depth into one group leads to higher opportunity to minimize test time. This is because memories with the same depth can be tested in parallel and memories with the same power can be tested in serial, which minimizes idle space in test time and power. In addition, we assign larger weights to edges when memories are closer.

Algorithm 1 Memory Partitioning

```
Procedure Partitioning(M)
Input: M, numMaxP, numMinP, maxMemP, maxD, $K_{\{1-6\}}$
Output: $P^{out}$
for n = numMaxP to numMinP do
    $P^1 \leftarrow$ single partition of M;
    for k = 1 to n − 1 do
        $G \leftarrow null$; $p_j \leftarrow \emptyset$;
        if $\max_{p_i \in P^k}\{|p_i|\} > maxMemP$ then
            $p_j \leftarrow \arg\max_{p_i \in P^k}\{|p_i|\}$;
        else if $\max_{p_i \in P^k}\{D(p_i)\} > maxD$ then
            $p_j \leftarrow \arg\max_{p_i \in P^k}\{D(p_i)\}$;
        else
            $p_{j_1} \leftarrow \arg\max_{p_i \in P^k}\{|p_i|\}$; $p_{j_2} \leftarrow \arg\max_{p_i \in P^k}\{D(p_i)\}$;
            $p_j \leftarrow \arg\min_{p_{j_1}, p_{j_2}}(p_{j_1}.\text{cut}, p_{j_2}.\text{cut})$;
        end if   // partition $p_j$ is the input for the next bipartitioning
        $P^{k+1} \leftarrow P^k \setminus \{p_i\}$;
        for criterion index r = 1 to 6 do
            $G \leftarrow GenerateHypergraph(p_j, r, G)$;
        end for
        $\{p_j, p_k\} \leftarrow MLPart(G)$;
        $P^{k+1} \leftarrow P^{k+1} \cup \{p_j\} \cup \{p_k\}$;
    end for
    $D_{max} \leftarrow \max_{p_j \in P^n}\{D(p_j)\}$;  // $P^{k+1} = P^n$
    if $D_{max} > maxD$ then
        if n == numMaxP then
            return $P^{numMaxP}$;
        else
            return $P^{n+1}$;
        end if
    end if
end for
return $P^{numMinP}$;
```

Algorithm 2 Construct Weighted Hypergraph (for r^{th} criterion)

```
Procedure GenerateHypergraph(p, r, $G_{in}$)
Input: p, criterion index r, (hyper)edge weight $K_r$, hypergraph $G_{in}$
Output: G
G ← $G_{in}$;
for all $m_i \in p$, i = 0 to $|p| - 1$ do
    $v_i \leftarrow mapping(m_i)$;  // node $v_i$ corresponds to $m_i$ in partition p
    add node $v_i$ to G;
    $visited(v_i) \leftarrow$ false;
end for
for i = 0 to $|p| - 1$ do
    $V_{conn} \leftarrow \emptyset$; $e \leftarrow null$;
    $V_{conn} \leftarrow \{v_i\}$; // $v_i$ is reference node
    $visited(v_i) \leftarrow$ true;
    for j = 0 to $|p| - 1$ do
        if $i \neq j$ then
            if ($visited(v_j)$ == false) || ($r \geq 4$) then
                if $v_i$ and $v_j$ satisfy criterion $crit_r$ then
                    $V_{conn} \leftarrow V_{conn} \cup \{v_j\}$;
                    // $V_{conn}$ is set of nodes that satisfy $crit_r$ w.r.t. $v_i$
                    $visited(v_j) \leftarrow$ true;
                end if
            end if
        end if
    end for
    if $|V_{conn}| \geq 2$ then
        e ← connect all $v \in V_{conn}$ as (hyper)edge;
        $weight(e) \leftarrow K_r$;
        add (hyper)edge e to G;
    end if
end for
return G;
```

Algorithm 3 Test Scheduling

```
    Procedure Scheduling($M$, $P^k$)
    Input: $M$, $P^k$, $numMaxP$, $GroupSizes$
    Output: $P^{out}$, test scheduling for each partition
 1: $numAddBIST \leftarrow numMaxP - |P^k|$;
 2: while $numAddBIST > 0$ do
 3:     for all $p_i \in P^k$ do
 4:         for all $s_j \in GroupSizes$ do
 5:             $GroupMemories(s_j)$; // grouping memories with size $s_j$
 6:             $Sol_{p_i,s_j} \leftarrow SolveMBISTILP(1)$;
 7:             $\{Sol_{p_{i_1},s_j},\ Sol_{p_{i_2},s_j}\} \leftarrow SolveMBISTILP(2)$;
 8:         end for
 9:         $Sol_{p_i} \leftarrow Sol_{p_i,s_1,best}$;
10:         $Sol_{p_{i_1}} \leftarrow Sol_{p_{i_1},s_2,best}$; $Sol_{p_{i_2}} \leftarrow Sol_{p_{i_2},s_2,best}$;
11:         $Gain_{p_i} \leftarrow Sol_{p_i}.cost - \max(Sol_{p_{i_1}}.cost,\ Sol_{p_{i_2}}.cost)$;
12:     end for
13:     choose $p_i \in P^k$ that has the largest $Gain_{p_i}$;
14:     if the largest $Gain_{p_i} == 0$ then
15:         break;
16:     end if
17:     $\{p_a,\ p_b\} \leftarrow$ result of partition $p_i \in P^k$ that has the largest $Gain_{p_i}$;
18:     $P^k \leftarrow (P^k \setminus p_i) \cup p_a \cup p_b$;
19:     $numAddBIST \leftarrow numAddBIST - 1$;
20: end while
21: $P^{out} \leftarrow P^k$;
```

Table III summarizes the edge weights used in G, each corresponding to one criterion. $crit_1$, $crit_2$, and $crit_3$ are the criteria of hyperedges between memories that have the same shape ($crit_1$), depth ($crit_2$) and test power ($crit_3$), respectively. In addition, $crit_4$, $crit_5$, and $crit_6$ specify the criteria of edges between pairs of memories with distances $\leq longD$ ($crit_4$), $\leq (longD + shortD)/2$ ($crit_5$), and $\leq shortD$ ($crit_6$), respectively. In our implementations, we set $longD$ and $shortD$ to $1000\mu m$ and $250\mu m$[4], respectively. The weights of hyperedges (respectively, edges) are additive, e.g., memories that have the same shape are connected by hyperedges with weight $K_1 + K_2 + K_3$ since memories having the same shape also have the same depth and power.[5]

In Algorithm 1, we loop through a number of partitions ranging from *numMaxP* down to *numMinP*, in order to obtain a k-way partitioning result that satisfies the given constraints of *maxMemP* and *maxD*. Since *MLPart* only returns bipartitions, we execute *MLPart* $k-1$ times to obtain a k-way partitioning (Line 3 in Algorithm 1). At each iteration, we choose one next partition as the input to *MLPart* (Lines 5-12 in Algorithm 1) based on the following criteria, in order of priority.

1) The partition that violates the *maxMemP* constraint, which is defined as twice the current average number of memories per partition.
2) The partition that violates the *maxD* constraint, which is defined as the half-perimeter of bounding box of the memory blocks in the corresponding partition.
3) One each of the partitions with the maximum number of nodes and with the largest diameter, respectively. Both are partitioned using *MLPart*, and the one with the smaller cut is selected.

We define *size* of a partition as the number of memories in the partition. The above criteria result in partitions that have similar sizes. We also specify a *tolerance* in *MLPart* to further promote balanced partition sizes.

Fewer partitions result in a larger solution space for scheduling, which in turn leads to less test time. Therefore we minimize the number of partitions with respect to the maximum diameter and maximum size constraints. In Algorithm 1, we keep reducing the number of partitions as long as the diameter of all partitions is $\leq maxD$

TABLE III: Summary of edge weights used in hypergraph generation.

Hyperedge selection parameter	$crit_r$	**Hyperedge weight**
Memories with the same shape	$crit_1$	$K_1 + K_2 + K_3$
Memories with the same depth	$crit_2$	K_2
Memories with the same power	$crit_3$	K_3
Edge distance between two memories		**Edge weight**
Distance $> longD$	-	0
Distance $\leq longD$	$crit_4$	K_4
Distance $\leq (longD + shortD)/2$	$crit_5$	$K_4 + K_5$
Distance $\leq shortD$	$crit_6$	$K_4 + K_5 + K_6$

(Lines 20-27 in Algorithm 1). If one of n partitions violates the *maxD* constraint, we end Algorithm 1 and return an (n+1)- or n-way partitioning result by comparing n to *numMaxP*.

In Algorithm 2, we construct weighted hypergraph. After mapping each m_i in p into v_i (Lines 2-6 in Algorithm 2), we collect the set $\{V_{conn}\}$ of all nodes that satisfy $crit_r$ with respect to the reference node v_i. If $|V_{conn}| \geq 2$, we connect all nodes in V_{conn} as hyperedge e and add e to hypergraph G (Lines 7-26 in Algorithm 2).

B. Test Scheduling

After partitioning, we solve the ILP described in Section III to obtain a test schedule. The number of extra BIST controllers (*numAddBIST*) is calculated as the difference between the current number of partitions (k) and *numMaxP*. Utilizing extra memory BIST controller resources can reduce the overall test time. Figure 2 illustrates an example showing how the test time can be reduced by utilizing additional memory BIST controllers.

For further test time reduction with extra BIST controllers, we try splitting each partition. $SolveMBISTILP(n)$ returns the solution (Sol) of $SolveMBISTILP(n)$ for a given number n of memory BIST controllers. The solution (Sol) contains test cost ($Sol.cost$) for the corresponding partition. Our heuristic allows for at most two memory BIST controllers (i.e., $n = 2$) for each partition.[6]

When *numAddBIST* is larger than zero, we run $SolveMBISTILP(n)$ with both one and two memory BIST controllers to calculate the benefit ($Gain_{p_i}$) of splitting the partition p_i. Since the two partitions are generated by $SolveMBISTILP(2)$ such that the given power constraint is satisfied, both of the split partitions can be tested simultaneously, enabling us to achieve a test time reduction.

We observe that most memories that have the same shape are scheduled in parallel with the same memory BIST controller; we can therefore pre-group those memories to reduce the runtime of ILP solver without significantly affecting solution quality. We group memories that have the same shape (width×depth) in each partition and consider the group as a single large memory to improve runtime by reducing the number of ILP constraints ($GroupMemories(s_j)$, Line 5 in Algorithm 3).[7] The number of memories in a group (s_j) can be decided by the max power constraints. After solving ILP (Lines 6-7 in Algorithm 3), these grouped memories are tested in parallel with the same memory BIST controller. Since s_j affects the solution of ILP, we try different s_j in *GroupSizes* to get the best solutions. In Algorithm 3, $Sol_{p_i,s_1,best}$ gives the minimum test cost with one BIST logic by s_1, and $Sol_{p_{i_1},s_2,best}$ and $Sol_{p_{i_2},s_2,best}$ give the minimum test cost with two BIST logics by s_2 (Lines 9-10).

At the last stage in this procedure (Lines 13-19 in Algorithm 3), we identify the partition that has the largest $Gain_{p_i}$ in the overall test scheduling from splitting with additional memory BIST controllers. The selected partition is divided into two partitions (p_a and p_b) to be mapped to the additional memory BIST controller. When all the available memory BIST controllers are consumed, the procedure ends and returns the partition and test scheduling result. We exit the **while** loop when the largest $Gain_{p_i}$ is zero (Lines 14-16 in Algorithm 3).

[4]We empirically find that these values give the best test time results.

[5]Latin Hypercube Sampling [14] was used to create a small design of experiments on the K_r values, but no combination was found that gave better solutions than the values $\{K_1, K_2, K_3, K_4, K_6\}$ = {8,5,1,1,2}, which we use to generate all reported results.

[6]We observe that $SolveMBISTILP(3)$ does not provide significant test time reduction, but ILP runtime increases dramatically. We therefore use only $SolveMBISTILP(2)$.

[7]The ILP for each partition typically has $O(10^4)$ variables and $O(10^5)$ constraints.

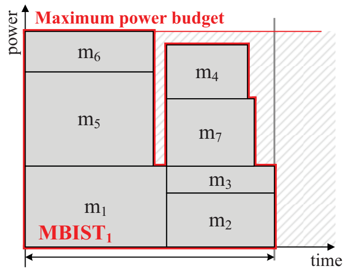


Fig. 2: Example test schedule (left) can be improved by adding an extra BIST controller to reduce test time (right).

C. Memory BIST Logic Placement

In the memory BIST logic placement step, we first define grids that cover the entire design. Any grid square that does not intersect memories is a possible location for BIST logic placement. We calculate the diameter from a grid square to all memories in a partition, and use this as a cost parameter. By calculating this cost parameter for all grid squares and all partitions, we generate a two-dimensional cost matrix for each grid square and memory partition. We then use this cost matrix to formulate and solve a min-weight maximum-matching problem in a bipartite graph, which is efficiently solvable using the Hungarian algorithm [23]. The resulting matching heuristically addresses timing criticality in paths between BIST logic and memories.

V. Validation and Experimental Results

Our heuristic implementation is developed in C++ and compiled with g++ 4.8.0. All experiments are run on a 2.5GHz Intel Xeon E5-2640 Linux workstation with 128GB memory and 12 hyperthreaded CPU cores. In the partitioning step, we apply *MLPart* [24] on hypergraphs generated using Algorithm 2 above. In the scheduling step, we use *CPLEX 12.5.1* [22] as our ILP solver to schedule testing of memories in each partition. Last, we solve the min-weight maximum matching problem in a bipartite graph [23] to assign BIST logic placement locations to partitions. (To our understanding, the turnaround time of our heuristic is not critical, and resynthesis of memory BIST logic after memory grouping takes only a few hours [26].) Table II presents command-line options in our implementation. In all of our experiments, we set 200 as the power constraint since the maximum $E(m_i)$ in testcases is $150 < E(m_i) < 200$.

To validate our heuristic methodology, we use six industrial testcases, each derived from a separate hard macro in a recent 28nm networking SOC product. Parameters of these testcases are given in Table IV. The number of memories in each testcase ranges from 124 to 160 and the number of partitions ranges from 7 to 13. Maximum and minimum number of memories, and maximum diameters without BIST logic, are also presented in Table IV.

TABLE IV: Description of testcases. (TC = testcase, M = memory, P = partition, D = maximum diameter without BIST logic, MAX = $\max_{1 \le i \le k} |p_i|$, and MIN = $\min_{1 \le i \le k} |p_i|$.)

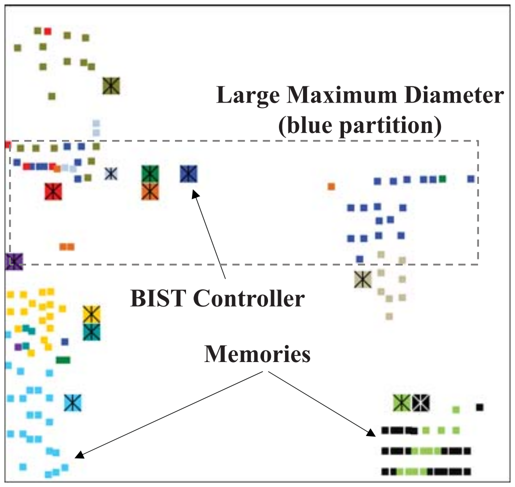
TCs	$\|M\|$	$\|P^k\|$	MAX	MIN	D (μm)
TC1	143	13	26	1	3900
TC2	150	11	28	2	4500
TC3	124	8	22	8	2200
TC4	160	13	30	1	3400
TC5	137	7	26	11	3200
TC6	148	12	25	1	4100

Table V compares industrial results and our results. We achieve up to 11.57% improvement in estimated test time, strictly smaller number of partitions (i.e., number of memory BIST controllers), and reduced maximum diameter with respect to BIST logic placement location, compared to the industrial results. Considering that test time is directly related to test cost and that fewer number of memory BIST logic leads to smaller die area, we believe this is a significant improvement. Furthermore, smaller maximum diameter of each memory partition (as shown in Figure 3) indicates better timing, which allows at-speed testing with smaller gate sizes and higher-V_T cell instances.

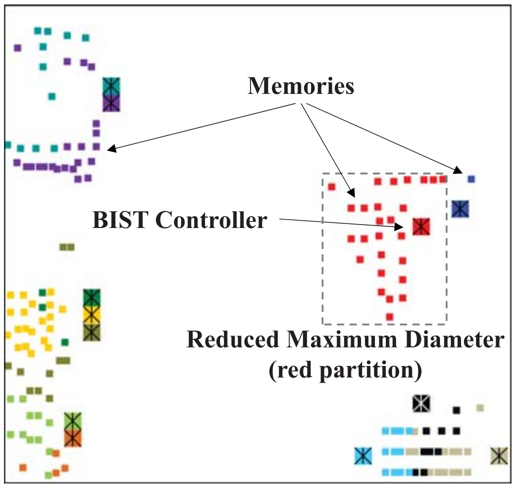
VI. Conclusions

In this work, we propose a heuristic methodology to co-optimize partitioning, test scheduling and memory BIST logic placement to minimize test time. Our heuristic approach generates hypergraphs over memories with test time-aware weighting of hyperedges, along with top-down, FM-style min-cut partitioning. Our ILP formulation comprehends parallel and serial testing for test time optimization with respect to power constraints. Further, we place the BIST logic to minimize the maximum diameter for each BIST group, which minimizes routing and buffering costs and improves timing. On hard macros from a recent industrial 28nm networking SOC, our results achieve up to 11.57% reduction in test time compared to the industrial solutions, using strictly fewer BIST controllers.

Our ongoing work pursues three main directions. (1) First, recall that we construct the weighted hypergraph instance for top-down partitioning independently of any map of placement density or routing congestion. We currently do not evaluate our memory partitioning and BIST logic placement solutions after placement and routing, and signoff timing analysis. To bridge this gap, we seek to integrate our partitioning and BIST logic placement optimizations into a (production) physical implementation flow. (2) Second, our need to



(a) Industrial solution (TC1). k = 13; $|B|$ = 13; maximum diameter = 3900; estimated test time = 1067. Blue and green partitions have large diameters.



(b) Our best solution (TC1). k = 12; $|B|$ = 12; maximum diameter = 2100; estimated test time = 969.

Fig. 3: Example showing superior outcome of our heuristic method. Different colors indicate different partitions, and rectangles (checked inside with '$*$') indicate BIST logic locations. Small squares indicate center locations of memories (to obfuscate the industry design floorplan).

TABLE V: Comparison between industrial solution and our solution. (P = partition, B = BIST controller, MAX = $\max_{1\leq i\leq k}|p_i|$, MIN = $\min_{1\leq i\leq k}|p_i|$, D = maximum diameter with BIST logic, TT = test time, TTR = test time reduction, and Power = peak power.)

	Industrial Solution				**Our Solution**							
	$\lvert P^k\rvert = \lvert B\rvert$	**D (μm)**	**est. TT**	**Power**	$\lvert P^k\rvert = \lvert B\rvert$	**MAX**	**MIN**	**D (μm)**	**est. TT**	**TTR (%)**	**Power**	**runtime (m)**
TC1	13	3900	1067	196.469	12	26	1	2100	969	9.18	188.277	37
TC2	11	4500	1138	195.504	8	37	9	2200	1095	3.78	195.389	22
TC3	8	2200	1633	194.922	7	42	9	2600	1444	11.57	192.236	114
TC4	13	3400	2103	197.839	12	22	2	2300	1956	6.99	199.588	189
TC5	7	3200	545	185.706	5	41	6	2300	556	–2.02	197.053	96
TC6	12	4100	6622	181.989	7	25	1	2800	6558	0.97	197.526	263

apply $SolveMBISTILP(2)$ to optimally schedule the testing of a large cluster of memories using two BIST controllers means that the hypergraph construction at some point leads min-cut partitioning "away" from good memory clusters. Thus, we seek improved hypergraph construction and weighting such that top-down mincut partitioning more directly produces a multi-way clustering that achieves minimum test time with k BIST controllers. (3) Third, recall that an initial motivation for this work is the disconnect between front-end DFT teams and back-end PD teams. We plan to enable the use of our tool by a PD team in a production SOC design environment, to validate the accuracy and schedule impact of (i) early feedback on timing and need for LVT devices in the BIST logic, (ii) understanding of feasible memory groupings in light of test schedule and power constraints.

Acknowledgments

We thank Dr. Jae-Gon Lee and Ms. Nancy MacDonald for valuable feedback and discussions of problem formulations.

References

[1] M. Abramovici, M. A. Breuer and A. D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, 1990.
[2] B. S. Baker, E. G. Coffman Jr. and R. L. Rivest, "Orthogonal Packings in Two Dimensions", *SIAM J. Computing* 9(4) (1980), pp. 846-855.
[3] S. P. Bradley, A. C. Hax and T. L. Magnanti, *Applied Mathematical Programming*, Addison-Wesley, 1977.
[4] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Improved Algorithms for Hypergraph Bipartitioning", *Proc. ASPDAC*, 2000, pp. 661-666.
[5] K. Chakrabarty, "Design of System-on-a-Chip Test Access Architecture under Place-and-Route and Power Constraints", *Proc. DAC*, 2000, pp. 432-437.
[6] K. Chakrabarty, "Test Scheduling for Core-Based Systems Using Mixed-Integer Linear Programming", *IEEE Trans. on CAD* 19(10) (2000), pp. 1163-1174.
[7] T.-F. Chien, W.-C. Chao, C.-M. Li, Y.-W. Chang, K.-Y. Liao, M.-T. Chang, M.-H. Tsai and C.-M. Tseng, "BIST Design Optimization for Large-Scale Embedded Memory Cores", *Proc. ICCAD*, 2009, pp. 197-200.
[8] J. Chin and M. Nourani, "FITS: An Integrated ILP-Based Test Scheduling Environment", *IEEE Trans. on Computers* 54(12) (2005), pp. 1598-1613.
[9] V. R. Devanathan, S. Bhavsar and R. Mehrotra, "Physical-Aware Memory BIST Datapath Synthesis: Architecture and Case-Studies on Complex SoCs", *Proc. ATS*, 2011, pp. 457-458.
[10] V. R. Devanathan, H. Ellur, M. Imran and S. Bathla, "Hierarchical, Physical-Aware, Built-in Self-Repair of Embedded Memories", *Proc. DAC*, 2013, Designer Track Session 11.
[11] V. R. Devanathan, R. Mehrotra, P. S. V. Kumar, V. Sarkar, I. M. Beg and S. Bathla, "Novel Approaches for Effective and Optimized Memory Test Flow in Nano-Scale Technologies", *Proc. VTS*, 2012.
[12] V. Iyengar, K. Chakrabarty and E. J. Marinissen, "Wrapper/TAM Co-Optimization, Constraint-Driven Test Scheduling, and Test Data Volume Reduction for SoCs", *Proc. DAC*, 2002, pp. 685-690.
[13] V. Iyengar, K. Chakrabarty and E. J. Marinissen, "Test Access Mechanism Optimization, Test Scheduling, and Tester Data Volume Reduction for System-on-Chip", *IEEE Trans. on Computers* 52(12) (2003), pp. 1619-1632.
[14] R. Jin, W. Chen and T. W. Simpson, "Comparative Studies of Metamodeling Techniques Under Multiple Modeling Criteria", *Struct. Multidiscip. Optim.* 23 (2001), pp. 1-13.
[15] C. Liu, E. Cota, H. Sharif and D. K. Pradhan, "Test Scheduling for Network-on-Chip with BIST and Precedence Constraints", *Proc. ITC*, 2004, pp. 1369-1378.
[16] S. J. Miller, "An Introduction to Linear Programming", *lecture notes*, Brown University Mathematics Dept., 2007.
[17] M. Miyazaki, T. Yoneda and H. Fujiwara, "A Memory Grouping Method for Sharing Memory BIST Logic", *Proc. ASPDAC*, 2006, pp. 671-676.
[18] R. Raman and I. E. Grossmann, "Modelling and Computational Techniques for Logic Based Integer Programming", *Computers and Chemical Engineering* 18(7) (1994), pp. 563-578.
[19] C.-W. Wang, J.-R. Huang, Y.-F. Lin, K.-L. Cheng, C.-T. Huang, C.-W. Wu and Y.-L. Lin, "Test Scheduling of BISTed Memory Cores for SoC", *Proc. ATS*, 2002, pp. 356-361.
[20] C. Yao, K. K. Saluja and P. Ramanathan, "Test Scheduling for Circuits in Micron to Deep Submicron Technologies", *Proc. VLSID*, 2011, Embedded Tutorial.
[21] W. Zou, S. M. Reddy, I. Pomeranz and Y. Huang, "SoC Test Scheduling Using Simulated Annealing", *Proc. VTS*, 2003, pp. 325-330.
[22] *IBM ILOG CPLEX Optimizer*. http://www.ilog.com/products/cplex/
[23] *Hungarian algorithm source code*. http://www.informatik.uni-freiburg.de/~stachnis/index.html
[24] *MLPart*. http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/Partitioning/MLPart
[25] *SEMICO Research Corporation*. http://www.semico.com
[26] N. MacDonald, Broadcom Corporation, *personal communication*, September 2013.

Appendix
Logical constraint handling in ILP

We describe the handling of *logical constraints* using indicator variables and very large numbers [3] [16] [18]. This method is extended for all indicators used in our formulation. To describe $U_i(t_q)$ in Equation (3), we define two more indicators, $V_i(t_q)$ and $W_i(t_q)$ as shown in Equations (9)–(11). A pair of inequalities in Equation (9) shows that $V_i(t_q) = 1$ when $T_{S_i} \leq t_q$. Likewise, Equation (10) shows that $W_i(t_q) = 1$ when $T_{S_i} > t_q$. $U_i(t_q) = 1$ if $t_q \geq T_{S_i}$ and $t_q < T_{E_i}$. In other words, when $V_i(t_q) = 1$ and $W_i(t_q) = 1$, we have $U_i(t_q) = 1$. Equation (11) shows the relation between $V_i(t_q)$, $W_i(t_q)$ and $U_i(t_q)$. Note that $V_i(t_q)$ and $W_i(t_q)$ can never be zero at the same time.

$$\begin{gathered} t_q - T_{S_i} + \varepsilon \leq N_L \cdot V_i(t_q) \\ T_{S_i} - t_q \leq N_L \cdot \{1 - V_i(t_q)\} \\ V_i(t_q) = \begin{cases} 1, & T_{S_i} \leq t_q \\ 0, & otherwise \end{cases} \end{gathered} \tag{9}$$

$$\begin{gathered} t_q - T_{E_i} + \varepsilon \leq N_L \cdot \{1 - W_i(t_q)\} \\ T_{E_i} - t_q \leq N_L \cdot W_i(t_q) \\ W_i(t_q) = \begin{cases} 1, & t_q < T_{E_i} \\ 0, & otherwise \end{cases} \end{gathered} \tag{10}$$

$$V_i(t_q) + W_i(t_q) - 1 = U_i(t_q) \tag{11}$$

Equations (12)–(14) show how we define $F_{k,i,j}$ and $L_{k,i,j}$ with $I_{k,i,j}$, $Q_{k,i,j}$, and N_{LL}. Whenever $I_{k,i,j} = 1$, inequalities are always true by virtue of N_{LL}, which means that m_i and m_j are irrelevant.

$$\begin{gathered} T_{E_i} - T_{S_j} - N_{LL} \cdot I_{k,i,j} \leq N_L \cdot (1 - F_{k,i,j}) \\ T_{S_j} - T_{E_i} + \varepsilon - N_{LL} \cdot I_{k,i,j} \leq N_L \cdot F_{k,i,j} \\ \mathrm{F}_{k,i,j} = \begin{cases} 1, & (B_{k,i} = B_{k,j} = 1) \wedge (T_{E_i} \leq T_{S_j}) \\ 0, & otherwise \end{cases} \end{gathered} \tag{12}$$

$$\begin{gathered} T_{S_j} - T_{S_i} + \varepsilon - N_{LL} \cdot I_{k,i,j} \leq N_L \cdot Q_{k,i,j} \\ T_{S_i} - T_{S_j} - N_{LL} \cdot I_{k,i,j} \leq N_L \cdot (1 - Q_{k,i,j}) \\ \mathrm{Q}_{k,i,j} = \begin{cases} 1, & (B_{k,i} = B_{k,j} = 1) \wedge (T_{S_i} \leq T_{S_j}) \\ 0, & (B_{k,i} = B_{k,j} = 1) \wedge (T_{S_i} > T_{S_j}) \end{cases} \end{gathered} \tag{13}$$

$$Q_{k,i,j} + Q_{k,j,i} - 1 = L_{k,i,j} \text{ , where } i \neq j \tag{14}$$