Area Efficient Pipelined Pseudo-Exhaustive Testing with Retiming

Huoy-Yu Liou and Ting-Ting Y. Lin*
Department of Electrical and Computer Engineering
University of California, San Diego
Chung-Kuan Cheng
Department of Computer Science Engineering
University of California, San Diego

Abstract

Pseudo-exhaustive testing (PET) offers a simple solution to testing complex circuits and systems [12]. However, PET suffers long testing time for test generation and high area overhead of test hardware. The pipelined pseudo-exhaustive testing (PPET) achieves fast testing time with high fault coverage by pipelining test vectors and test responses among partitioned circuit segments [15]. To reduce hardware overhead in PPET, a novel approach for implementing area-efficient PPET is presented. Circuit partitioning with retiming is used to convert designs for PPET. Experimental results show that this approach exhibits an average of 20% area reduction over non-retimed testable circuits. Our algorithm offers high utilization of existing flip-flops (FFs) and provides a framework for further performance optimization.

1 Introduction

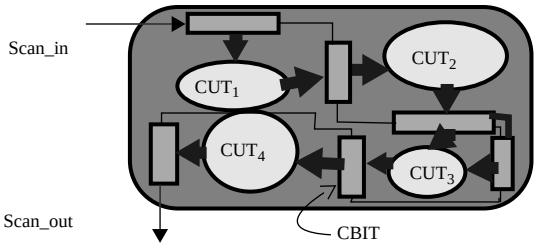
Pipelined pseudo-exhaustive testing (PPET) [8] was proposed as an efficient self-testing scheme for high fault coverage on stuck faults. In this testing methodology, test registers are grouped into cascadable multiple-input shift registers (MISRs), namely the Cascadable Built-in Testers (CBITs), to perform dual-mode pseudo-exhaustive test pattern generation (TPG) and parallel signature analysis (PSA). A scan chain links all the test registers for initialization and signatures read-out.

During self-testing mode, non-overlapping segments of logic are tested concurrently by pairs of distinct CBITs. In each pair, the proceeding CBIT generates test patterns and the succeeding CBIT reads outputs of circuit-under-test (CUT) for deriving test signature. The dual-mode capability enables the second CBIT (performing PSA) to be the first CBIT (performing TPG) of *other* pairs in PPET.

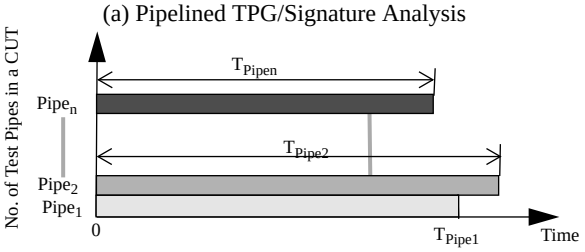
Figure 1(a) shows circuit configuration with CBITs in self-testing mode. Circuit segments, CUT_1,..., CUT_4, are results of circuit partitioning and surrounded by CBITs. Test patterns and test responses propagate to CBITs through existing data paths. Since all CUTs are tested concurrently after global initialization of CBITs, the total testing time is dominated by the CUT with the largest set of test vectors. Because number of test vectors grows exponentially with CBIT width, the maximum width of CBITs determines the overall testing time (Figure 1(b)).

The validity of PPET in realistic circuit is determined by two criteria: the saving in testing time needs to be significant and the additional test hardware is minimal. Testing time can be improved when circuit partitioning techniques are used to reduce the width of CBITs with a given number of segments. A dual formulation of the partitioning is to set the upper bound on CBIT width while minimizing the number of segments. However, both problems have been shown to be *NP* complete [4]. Heuristics have been proposed to find the smallest set of cut nets to minimize additional test hardware in complex circuits [4][13].

Test hardware can be warranted through *legal retiming* [1] by repositioning, adding, or removing flip-flops (FFs) while preserving functionality of the new circuit in normal operation. Functional FFs are moved to desired CBIT locations to construct test paths for self testing. If the number of the suggested test FFs exceeds that provided by retiming, extra multiplexing circuitry is needed between existing data paths in normal operation and the additional test FFs for self-testing mode.



(a) Pipelined TPG/Signature Analysis



(b) Testing time for different test pipes in MISR mode where T_{CBIT} is the testing time for each test pipe dominated by the widest CBIT in each pipe

Figure 1: The Pipelined Pseudo-exhaustive Testing Scheme

Previous works on retiming for improved testability were proposed for partial scan [2][3]. Partial scannable FF configuration is identified as the minimum feedback vertex set (MFVS) of the FFs in a circuit under test. Retiming is used to transform the existing FF assignment to timing/area optimal partial scannable design.

In this work, a scheme that combines circuit partitioning algorithm with retiming is presented for test hardware savings for implementing PPET. A set of minimal cut nets which dissect the circuit into disjoint clusters is obtained by the multicommodity flow partitioning algorithm [6]. Legal retiming is then used to add, delete, or move existing functional registers to CBIT locations. Additional test registers not allocated by retiming are recognized. CBIT assignment is performed through a greedy approach to find the best area savings on testing smaller clusters. Experimental results show flow-based circuit segmentation with retiming for PPET achieves an average of 20% area reduction over that without retiming technique.

The paper is organized as follows: Section 2 shows the theoretical modeling of the area-efficient PPET problem with cost function and constraints imposed by legal retiming. Section 3 explains our heuristics in detail for solving this optimization problem. Experimental results on ISCAS89 benchmark circuits [5] are discussed in Section 4. Trade-offs in test hardware overhead among various testing time requirements are demonstrated. Concluding remarks are briefed in Section 5.

2 Problem Statement

In the following sections, a formal representation of the area-optimal partitioning for PPET with retiming is derived.

* This author is currently with Tandem Computers Inc., Cupertino, CA.

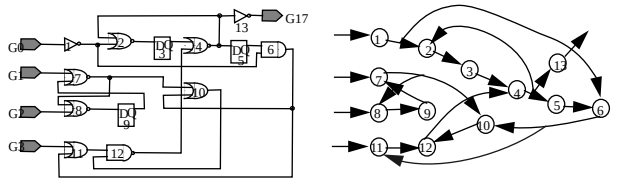
33rd Design Automation Conference ®

DAC 96 - 06/96 Las Vegas, NV, USA

2.1 Graph Representation

A synchronous circuit can be represented by a directed graph, $G(V = R \cup C, E)$, where V is the set of nodes including registers, R, and combinational components, C, and E is the set of directed edges describing signal flows from source to sink(s) among nodes.

Figure 2 shows an example of constructing $G(V, E)$ from a given circuit netlist. The schematic representation of a synchronous circuit, s27 from ISCAS89 benchmark [5], is depicted in Figure 2(a). Directed graph representation using the multi-pin model [6] is illustrated in Figure 2(b). Under the multi-pin model, each net is represented by a directed edge with branches from the source module indicating fan-outs to different nodes. For the example in Figure 2(a), this is shown as the out-going edges from node 1, 4, 7, and 10 with multiple fan-outs in Figure 2(b).



(a) Schematic of s27 (b) Digraph of s27

Figure 2: Graph Representation of A Circuit

2.2 Retiming Principles

Retiming is an operation on synchronous circuits which deletes or adds registers to a circuit to meet certain performance constraint(s) [1]. Let $\rho: C \to Z$ be a retiming which transforms the original circuit, $G(V = R \cup C, E)$, to a new circuit, $G_\rho(V_\rho = R_\rho \cup C, E_\rho)$, by adding or removing registers, R, of V through an integer-valued vertex-labeling procedure on C. Furthermore, let $f: V \to I = \{0, 1, \ldots, k\}$ denote the number of registers for a given path in G. We will use the following retiming principles from [1] in the application to area efficient PPET:

Lemma 1. Let $\rho: C \to Z$ be a retiming on a synchronous circuit, $G(V = R \cup C, E)$. If $p = v_0 \xrightarrow{e_1} v_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} v_n, \forall v \in V$, is a path in G, let $f(p)$ represent the number of registers in p. Thus we have for each path $p \in G$,

$$f_\rho(p) = f(p) + \rho(v_n) - \rho(v_0). \quad (1)$$

Corollary 2. For any directed cycle p in G, if ρ is a retiming on G, then

$$f_\rho(p) = f(p) \quad (2)$$

Corollary 3. A *legal retiming* on $G(V = R \cup C, E)$ means $f_\rho(p) = f(p) + \rho(v_n) - \rho(v_0) \geq 0$, for all paths in the retimed circuit, G_ρ. (3)

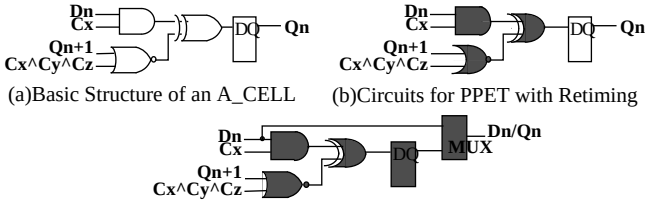
2.3 Input Constraint Partitioning for PPET on Sequential Circuits

The implementation of PPET according to a specific testing time on circuits with high fan-ins can be modeled as the classical m-way partitioning problem. When the given testing time is $O(2^N)$ clock cycles, the largest input size of a CUT can not exceed N in order to warrant the user-specified testing time. This is the partition with input constraint (PIC) problem [4], which has been proven to be *NP*-complete.

Let $\iota: G \to N$ be a function which represents the number of inputs to a circuit described by a directed graph $G(V, E)$. The input-constraint m-way partition, $\Pi_m: V \to \{1, 2, 3, \ldots, m\}$ is an operation on graph $G(V, E)$ with a given positive integer, κ, such that for each $\pi_i = \{v \in V | \Pi_m(v) = i, 1 \leq i \leq m\}$, $1 \leq \iota(\pi_i) \leq \kappa$, and $V = \bigcup_{i=1}^{m} \pi_i$. No overlapping/gate-sharing among π_i's [7] is allowed in PPET since each CBIT is dedicated to pseudo-exhaustive test pattern generation (TPG) for one CUT. Overlapping CUTs cannot guarantee high fault coverage in $O(2^N)$ clock cycles under PPET.

For each cut net, an A_CELL [8] is placed at the cut location. Figure 3(a) shows the basic design of an A_CELL. For CMOS technology [14], the area of an A_CELL is $(3 + 2 + 4 + 10)/10 = 1.9$ times that of a DFF. There are one 2-input AND (3 area units), one 2-input NOR (2 area units), and one 2-input XOR (4 area units) gates added to the input of a DFF (7area units). If an A_CELL is converted by a functional register through retiming, only the three logic gates are added. Therefore, the area overhead is given by the three gates which is 0.9 times of a DFF as the shaded gates shown in Figure 3(b).

A 2-to-1 MUX (3 area units) should be added when an A_CELL replaces a cut net without utilizing existing DFFs as prohibited by Eq. (2). Figure 3(c) shows the multiplexed data paths both for normal operation: $D_n \to MUX \to D_n/Q_n$, and for self testing mode: $D_n \to AND \to XOR \to DFF \to MUX \to D_n/Q_n$. The total area of an A_CELL and a MUX is 2.3 times of a DFF without considering additional routing.



(a)Basic Structure of an A_CELL (b)Circuits for PPET with Retiming

(c) Circuits for PPET w/o Retiming

Figure 3: Schematic of the A_CELL

Generally, a modified register (e.g., the A_CELL in the CBIT design) is placed on a cut net and grouped later into a CBIT. As discussed in Section 2.2, additional registers can be added arbitrarily for PPET based on Eq. (1) of the retiming principles. However, the number of registers in loops should follow Eq. (2) to guarantee *legal retiming*. This restricts the number of cut nets (i.e., the suggested location of registers for PPET) in loops (or *strongly connected components* [9]). Consequently, additional multiplexing circuitry between normal operation and self-testing mode is needed when there are more registers required by PPET than existing number of FFs in loops.

Let $\chi: E \to I$ represent the number of cuts on the edge set, E. $\chi(e) = 1$ if edge e is cut, otherwise $\chi(e) = 0$. The number of cut nets on a cyclic path p in G is then $\chi(p)$. An m-way PIC for PPET can be legally retimed when Eq. (3) is satisfied. For a cycle p, there are $\chi(p) - f(p)$ multiplexed registers added for PPET if $\chi(p) > f(p)$. No multiplexing circuitry is needed when $\chi(p) \leq f(p)$.

2.4 The Object Function

The main objective of this implementation of PPET on sequential circuits is to minimize total CBIT area according to user-specified testing time. Table 1 lists relative area cost comparing to that of a non-self-test D-type flip-flop (DFF) for various sizes of CBITs. Column 3 specifies the estimated area, p_k, of each CBIT type, d_k, with respect to that of a simple DFF when each register is replaced by an A_CELL and the feedback polynomial is primitive. Column 4 shows less bit-wise cost, σ_k, for bigger l_k. The reason is that simple primitive feedback polynomial which gives smaller area cost for a larger l_k may exist. However, testing time grows exponentially with increasing l_k as shown in Figure 4. Therefore, d_4 and d_5 are candidates for maximal CUT sizes which give both feasible testing time and better area savings in PPET.

Table 1: Area Cost for Various CBIT Sizes

CBIT Type (d_k)	CBIT Length (l_k)	Area/DFF (p_k)	p_k/Bit (σ_k)
d1	4	8.14	2.04
d2	8	16.68	2.09
d3	12	24.48	2.04

Table 1: Area Cost for Various CBIT Sizes

CBIT Type (d_k)	CBIT Length (l_k)	Area/DFF (p_k)	p_k/Bit (σ_k)
d4	16	32.21	2.01
d5	24	47.66	1.99
d6	32	63.12	1.97

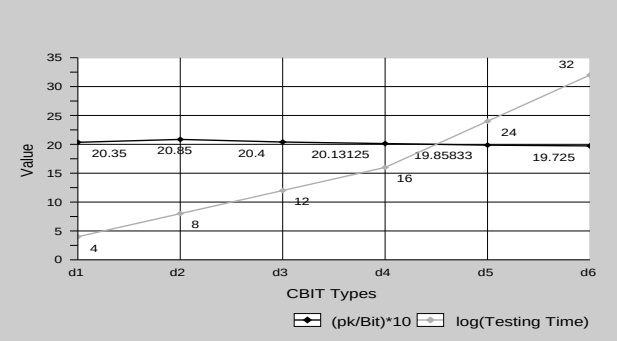


Figure 4: Bit-wise Area vs. Testing Time for Various CBIT Types

To formulate the area-minimal constraint for PIC with retiming, let n_k denote the number of type d_k CBITs, Σ be the total cost of combinations of CBITs, and q be the total number of CBITs. The goal is to minimize

$$\Sigma \equiv \sum_{k=1}^{q} p_k n_k, \text{ where } 0 \le total\_cut\_nets \le min\left\{\sum_{k=1}^{q} l_k n_k\right\} \quad (4)$$

with constraints (including primary inputs)

$$\pi_i \subset V, 1 \le i \le m \ni 1 \le \iota(\pi_i) \le l_k, k = 4, 5, \text{ for each partition} \quad (5)$$

$$\text{and for each loop } \lambda, \chi(\lambda) \le \beta \times f(\lambda), \quad (6)$$

where $\beta \ge 1$ is an integer-valued multiplication factor manipulating number of cuts on loop λ.

3 The Heuristics

The first version of our PPET implementation on structural circuit netlists is called Merced. Table 2 demonstrates the top-level design of the software. STEP 1 reads in the circuit netlist and converts it to an internal data structure. A list of strongly-connected components (SCC) is built [9] for observing legal retiming which is tunable by user as in Eq. (6). At STEP 3, an input-constraint partitioning procedure, *Assign_CBIT*, is invoked to find the minimal value of Eq. (4), satisfying both Eq. (5) and Eq. (6). At the end of the procedure, a report on the partitioned circuits and its cost is generated in STEP 4.

STEP 1 Construct the graph representation of input design, *G(V,E)*.
STEP 2 Identify strongly connected components in *G*, *SCC(G)*.
STEP 3 *Assign_CBIT (G, Δ, α, l_k)* with Eq. (6)
STEP 4 Return solution *P* and *cost*.

Table 2: Merced--The BIST Compiler

3.1 Partitioning with Retiming by *M*-way Clustering Algorithm

Probabilistic multicommodity-flow approach [10] is adopted to identify a minimal cut set of all nets, E, in the circuit/network G. Edges are removed from G according to their congestion until all clusters, $\{\varpi\}$, are disjoint and have input number smaller than $\kappa = l_k$, i.e., $\iota(\varpi) \le l_k, \forall \varpi \subseteq V$. A distance function is defined as $d:flow(E) \to \Re$ which indexes the congestion of each net according to an exponential function of the flow generated by the stochastic process [10]. Since our objective is not only to have a set of disjoint clusters, but also to satisfy Eq. (5), the network saturation procedure in [6] needs to be modified. To guarantee a *fair* sampling on the random-flow injection process, an index on the nodes, V, in G, $visit:V \to I$, is introduced to monitor the source selection by Dijkstra's algorithm. Distance function will truly reflect the congestion of the network when values of $visit(V)$ is large enough to reflect a random process. Table 3 outlines the modified *Saturate_Network* procedure:

STEP 1 For each net, $e \in E$ do
1.1 $d(e) = 1$, $flow(e) = 0$, $cap(e) = b$.
STEP 2 For each node, $v \in V$ do
2.1 $visit(v) = 0$.
STEP 3 While ($\exists v \in V \ni visit(v) \le min\_visit$) do
3.1 Randomly pick a node, *v, visit(v)=visit(v)+1*.
3.2 Find shortest path(s) from *v* to all sinks, *Tv = Dijkstra(G, d(E), v)*.
3.3 For each net in the shortest path tree, $e \in T_v$ do
3.3.1 $flow(e) = flow(e) + \Delta$
3.3.2 $d(e) = exp(\alpha \times flow(e) / cap(e))$.
STEP 4 Return *G* with $d(E)$ information.

Table 3: Modified Saturate_Network (G, Δ, α)

Result from *Saturate_Network* is demonstrated in Figure 5 of the example of Figure 2(b). Wider arrows represent more flows injected and are more congested. Nets in big SCC's are more congested because the equi-probable selection of source nodes in STEP 3.1 injects flows to those "strongly-connected" nets more often.

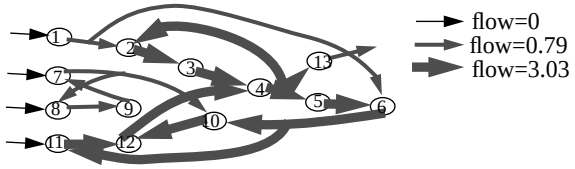


Figure 5: Figure 2 (b) after *Saturate_Network*

Clustering procedure begins with the distance function, $d(E)$, returned by the modified *Saturate_Network*. A sorted stack of all different values of $d(E)$'s is built from max to min for constructing a set of clusters, $\{\varpi | \iota(\varpi) \le l_k, \forall \varpi \subseteq V\}$. This $d(E)$ stack represents the congestion degree of nets in the network. A net-removing process according to net congestivity minimizes the set of cut nets. Table 4 summarizes the clustering process which selects cut nets according to $d(E)$ and assures Eq. (5) and Eq. (6). For Eq. (5), the solution to the While loop in STEP 5 is guaranteed as long as l_k is bigger than the maximal fan-in number of each primitive cell in the circuit. Since reducing fan-in numbers by resynthesizing the circuit is not of concern in implementation current version of Merced, each primitive cell is considered non-alternative. The constraint of Eq. (6) is checked at each call of *Make_Set* in Table 5.

STEP 1 Initialize $S = \varnothing$.
STEP 2 *Saturate_Network* (*G*, Δ, α)
STEP 3 Construct a sorted list, *D*, of $d(e)$'s from max to min.
STEP 4 *S=Make_Set* (*G*, *d(E), Extract_Max(D)*).
STEP 5 While (($\exists g \in S$) $\ni in(g) > l_k$) do
5.1 *boundary=Extract_Max(D), D=D-{boundary}*.
5.2 *S=S-{g}+Make_Set* (*g, boundary*).
STEP 6 Sort *S* from max *in(g)* to min *in(g)*.
STEP 7 Return *S*.

Table 4: Make_Group (G, Δ, α, l_k)

Make_Set is based on modified depth-first-search (DFS) [11] according to a given *unassigned* cell (the *seed*) and the current value of the searching *boundary* given by current value of $d(E)$. Table 5 is the *Make_Set* procedure. At STEP 2 of Table 5, an option is offered for user to *lock* cells in the current design (i.e., a

portion of the circuit) that Merced is not expected to work on.

STEP 1 Initialize $Q = \varnothing$
STEP 2 For each $v \in list$ do
  2.1 If (v is *not locked*) Mark v *not assigned*.
STEP 3 While ($\exists v \in list$ is *not assigned*) do
  3.1 g =DFS (*list, v, boundary*).
  3.2 $Q = Q + g$.
STEP 4 Return Q.

Table 5: Make_Set (list, boundary)

Table 6 and Table 7 briefs the modified DFS with consideration of Eq. (6). At STEP 2.1 of Table 7, the value of $d(e \in SCC)$ is modified to be insignificant if the number of nets removed from the SCC, $c(SCC)$, is greater than $\beta \times f(SCC)$, which is specified by the designer.

STEP 1 $group = v$.
STEP 2 For each $u \in list \neq \varnothing$ do $color(u) = white$.
STEP 3 *DFS_Visit(list, v, boundary)*.
STEP 4 Return *group*.

Table 6: DFS (list, v, boundary)

STEP 1 $color(u) = gray$.
STEP 2 For each $e \in Adj(u)$ do
  2.1 If ($d(e) \geq boundary$) then
    2.1.1 If ($c(SCC) < \beta \times f(SCC)$) then $c(SCC) = c(SCC) + 1$
    2.1.2 else
      2.1.2.1 For each $e \in SCC$: If ($d(e) < boundary$), $d(e) = 0$.
  2.2 else
    2.2.1 For each $w \in Adj(e)$ do
      2.2.1.1 If ($color(w) = white$) then
        $set\_next(u) = w$,
        DFS_Visit(list, w, boundary).
STEP 3 $color(u) = black$.

Table 7: DFS_Visit (list, u, boundary)

For the example of Figure 5 with l_k=3, *Make_Group* produces result shown in Figure 6. The removal of the nets with $flow(e)$=3.03 gives clusters with input number less than l_k=3.

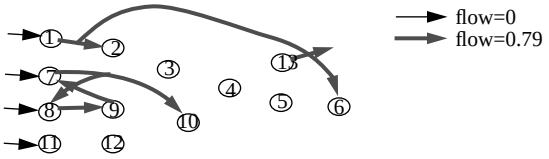


Figure 6: Figure 5 after *Make_Group* (l_k=3)

3.2 Near-Optimal Solution for CBIT Assignment

Circuit clusters identified by *Make_Group* may contain a lot of small clusters with input sizes less than $0.5l_k$ and shared input nets. As shown in Figure 4, the bit-wise area cost reduces as CBIT length increases, it is cost-effective to merge small clusters to fit in one CBIT with length l_k as much as possible rather than assigning individual small CBITs to small input clusters. The final pass for assigning CBITs will perform cluster merging in greedy approach.

Table 8 lists the final pass for merging small clusters, *Assign_CBIT*. At STEP 3, the input number of unassigned/remaining circuit, *S*, is calculated and compared with l_k. During each pass of the While loop, a cluster with the largest input number, *O*, is removed from the *S* list. The inner loop of STEP 3.2 calculates merging gain of the clusters remaining in the *S* list with respect to *O* and finds the best solution, cluster *g*, from *S*. A new cluster is formed as $O = O + g$. This process continues until $\iota(O) = l_k$ or no feasible solution in *S* can be found. Then *O* is added to the partitioned list, *P*, and total cost, Σ, is updated accordingly.

The gain function, $\gamma: \varpi \rightarrow Z$, of merging two clusters, ϖ_1 and ϖ_2, is defined as the difference between the input size of the merged cluster, $\varpi_1 + \varpi_2$, and l_k:

$$\gamma(\varpi_1 + \varpi_2) \equiv l_k - \iota(\varpi_1 + \varpi_2) \quad (7)$$

Paired clusters with the same merged input number are discerned at STEP 3.2.1 according to number of cut nets that current merge removes. Zero or positive number of γ indicates current merging is possible in proportion to the gain, otherwise the merging is not feasible according to Eq. (5).

STEP 1 k=0, $cost$=0, $P = \varnothing$.
STEP 2 S=*Make_Group* (G, Δ, α, l_k).
STEP 3 While ($in(S) > l_k$) do
  3.1 O=*Extract_Max*(S), $S = S - \{O\}$.
  3.2 While ($in(O) < l_k$ **and** $\exists g \in S$ *not visited*) do
    3.2.1 Search for the best and feasible g from S.
    3.2.2 If ($in(O + g) \leq l_k$) then
      $O = O + g$, $S = S - g$.
  3.3 $P = P + O$, $cost = cost + p_O$, $k = k + 1$.
STEP 4 $P = P + S$, $cost = cost + p_S$, $k = k + 1$.
STEP 5 Return P, $cost$ and k.

Table 8: Assign_CBIT (G, Δ, α, l_k)

When the outer loop of STEP 3 stops, there might be a remaining part of *G*, *i.e.*, *S*, left which cannot be merged to any existing partition given by this greedy approach. Therefore, a suitable size of CBIT is assigned to the residual *S* and the near-optimal solution is found by *Assign_CBIT*. Figure 7 shows final merged result of Figure 6 and four partitions are found: $V = \{1, 2, 3, 6\} \cup \{4, 5, 13\} \cup \{7, 8, 9, 10\} \cup \{11, 12\}$.

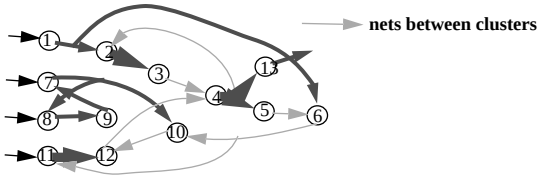


Figure 7: Figure 6 after *Assign_CBIT* (l_k=3)

3.3 Complexity Analysis

Finding the strongly connected components by DFS takes $O(|V| + |E|)$ times on a given graph $G(V, E)$. Since *Make_Group* is based on DFS, the complexity is bounded by $O(|d(E)| \times (|V| + |E|))$. In the worst case, $O(|d(E)|) \approx O(|E|)$, so the worst-case complexity is $O(|E| \times (|V| + |E|))$. However, in realistic practice, $|d(E)| \lhd |E|$, and the clustering process at STEP 5 of Table 4 finishes with number of iterations much less than $|E|$. The average complexity can be considered as $O(\Gamma \times (|V| + |E|))$, where $\Gamma \ll |E|$ is a constant.

For the modified *Saturate_Network* procedure, STEP 3 of Table 3 takes $[visit] \times |V| \log(|V|)$, where $[visit]$ is the average/expected number of visits to each node in a graph, $G(V, E)$, by the random selection process in STEP 3.1. Thus *Saturate_Network* takes at most $O((\,[visit] + Var[visit]\,) \times |V| \log(|V|))$ iterations, where $Var[visit]$ is the variance of the random visiting process.

The greedy approach of *Assign_CBIT* takes $O(|\{\varpi\}| \log(|\{\varpi\}|))$ for finding the maximum $\iota(\varpi)$ at STEP 3.1. The inner loop of STEP 3.2 takes at most $O(|\{\varpi\}|)$ iterations for merging small clusters. Therefore, the complexity of *Assign_CBIT* is $O(|\{\varpi\}| \log(|\{\varpi\}|))$.

Since the number of clusters, $O(|\{\varpi\}|)$, is smaller than the number of nodes, $|V|$, and $O(\Gamma \times (|V| + |E|))$ from *Make_Group* grows with $O(|V| + |E|)$ for a constant Γ, the overall complexity is thus dominated by $O((\,[visit] + Var[visit]\,) \times |V| \log(|V|))$

from *Saturate_Network*.

4 Experiments

The sequential benchmark circuits, ISCAS89 [5] from Microelectronics Center of North Carolina (MCNC), is used for testing Merced. Table 9 lists the netlist information of the seventeen test cases from small to large sizes:

Table 9: Circuit Information of Selected ISCAS89 Benchmark Circuits

Circuit Name	No. of PIs	No. of DFFs	No. of Gates	No. of INVs	Estimated Area
S510	19	6	179	32	547
S420.1	18	16	140	78	620
S641	35	19	107	272	832
S713	35	19	139	254	892
S820	18	5	256	33	943
S832	18	5	262	25	961
S838.1	34	32	288	158	1268
S1423	17	74	490	167	2238
S5378	35	179	1004	1775	6241
S9234.1	36	211	2027	3570	11467
S9234	19	228	2027	3570	11637
S13207.1	62	638	2573	5378	19171
S13207	31	669	2573	5378	19476
S15850.1	77	534	3448	6324	21305
S35932	35	1728	12204	3861	50625
S38417	28	1636	8709	13470	52768
S38584.1	38	1426	11448	7805	55147

The last column shows the estimated area of the circuits by counting 1 unit area per inverter, 3 units for 2-input AND gates, 2 units per 2-input NAND gate, 3 units per 2-input OR gate, 2 units for 2-input NOR gates, and 10 units for each DFF. Gates with higher fan-ins are scaled up with 1 unit area per additional input by the CMOS technology [14].

4.1 Setting the Parameters

For the modified *Saturate_Network* procedure, a few experiments are conducted to obtain a set of parameters for a properly distributed distance function, $d(E)$, which can be used to differentiate the net congestivity for a moderate sampling size, i.e., the average value of *visit* in STEP 3 of Table 3. A small sampling size (i.e., a smaller value of *min_visit*) will identify more locally congested nets and a first pass of the *Make_Set* in STEP 4 of Table 4 gives a lot of small segments which slows down STEP 3.2.2 of Table 8. The values of Δ should be chosen so that the averaged flow value, *f()*, is not larger than the capacity of the net, i.e., $min\_visit \times \Delta \le b$. The value of α should be adjusted to magnify the flow difference into the distance function appropriately for the same reason as that for setting *min_visit*. From our observations, we set $b = 1$, $min\_visit = 20$, $\alpha = 4$, and $\Delta = 0.01$.

The retiming constraint for loops in Eq. (6) is relaxed by setting $\beta = 50$ in STEP 2.1.1 of Table 7 for all circuits. The purpose is to get the unrestricted results from *Assign_CBIT* for the best testing time specified by l_k, i.e., $O\left(2^{l_k}\right)$ clock cycles. A designer can give a smaller value of β to restrict the number of cut nets in the strongly connected components by trading off with longer testing time. Thus β is a design-dependent parameter set by the user.

4.2 Results and Comparison

Two sets of experiments are conducted for $l_k = 16$ and $l_k = 24$ on a SUN/Sparc10 station. Table 10 and Table 11 list the result for $l_k = 16$ and $l_k = 24$ respectively. As shown on the 4-th columns of the tables, most cut nets returned by *Assign_CBIT* are on SCC's where retiming can fully utilize the existing DFFs on SCC's. There is no significant trend on the number of cut nets of SCC's growing with circuit sizes. It varies on individual design as design style differs.

Table 10: Partition Results for $l_k = 16$

Circuit Name	No. of DFFs	DFFs on SCC	cut nets on SCC	nets cut	CPU time (sec.)
S510	6	6	77	92	0.1
S420.1	16	16	0	8	<0.05
S641	19	15	19	28	<0.05
S713	19	15	24	34	<0.05
S820	5	5	68	88	<0.05
S832	5	5	77	96	<0.05
S838.1	32	32	0	23	<0.05
S1423	74	71	53	65	<0.05
S5378	179	124	283	420	0.6
S9234.1	211	172	497	700	1.2
S9234	228	173	471	649	4.9
S13207.1	638	462	794	975	3.3
S13207	669	463	817	978	2.9
S15850.1	534	487	720	1014	2.0
S35932	1728	1728	2881	2926	191.6
S38417	1636	1166	1703	2506	66.9
S38584.1	1426	1424	3110	3322	97.9

Table 11: Partition Results for $l_k = 24$

Circuit Name	No. of DFFs	DFFs on SCC	cut nets on SCC	nets cut	CPU time (sec.)
S641	19	15	12	17	<0.05
S713	19	15	32	38	<0.05
S5378	179	124	254	392	0.4
S9234.1	211	172	379	531	1.0
S13207.1	638	462	749	931	10.7
S13207	669	463	689	845	4.8
S15850.1	534	487	602	872	18.1
S35932	1728	1728	2639	2667	85.4
S38417	1636	1166	1555	2279	60.4
S38584.1	1426	1424	2593	2764	95.0

The number of cut nets increases as the circuit size grows up as shown in the last column of Table 10 and Table 11. This manifests the characteristics of the benchmark circuits which are not locally clustered according to the input size constraint, $l_k = 16$ and $l_k = 24$. Therefore, a bigger size of CBIT will accommodate more nets than a smaller CBIT and reduce the number of cut nets as comparing the last column of Table 11 with that of Table 10.

Table 12 shows retiming achieves 2% to 32% area savings over non-retimed circuits in terms of the percentage of CBIT area vs. total circuit area implementing PPET for $l_k = 16$ and $l_k = 24$. The CBIT area with retiming is calculated as the number of retimable cut nets multiplied by 0.9 of the area of a DFF (10 units) for adding three gates shown in Figure 3(b). And the excess cut nets on SCC's which need multiplexing circuitry which is 2.3 times the area of a DFF as depicted in Figure 3(c). For CBIT area without retiming, all internal cut nets are added with an A_CELL with a MUX by the fact that the DFFs in the original circuit is not moved. Zero entries represent no internal cuts for circuits with input number less than $l_k = 24$. The area savings offered by retiming grows more significant for large circuits as depicted in Figure 8.

Table 12: CBIT Area Comparison for $l_k = 16$ and $l_k = 24$

Circuit Name	ACBIT/ATotal (%)			
	l_k=16		l_k=24	
	w/ Retiming	w/o Retiming	w/ Retiming	w/o Retiming
S510	78.8	80.6	0	0
S420.1	19.7	24.2	0	0
S641	18.9	45.4	13.2	33.5
S713	27.4	48.5	33.9	51.3
S820	67.2	69.7	0	0

Table 12: CBIT Area Comparison for l_k = 16 and l_k = 24

Circuit Name	ACBIT/ATotal (%)			
	l_k=16		l_k=24	
	w/ Retiming	w/o Retiming	w/ Retiming	w/o Retiming
S832	69.0	71.2	0	0
S838.1	25.6	30.9	0	0
S1423	22.5	41.8	0	0
S5378	46.8	62.4	43.4	60.8
S9234.1	49.3	60.1	38.8	53.4
S9234	45.5	57.9	0	0
S13207.1	30.2	55.7	27.3	54.5
S13207	34.4	55.4	26.4	51.7
S15850.1	32.9	54.0	24.9	50.3
S35932	36.7	58.8	31.3	56.5
S38417	27.1	54.0	21.5	51.6
S38584.1	45.3	59.8	36.8	55.3

5 Conclusion

Our experiments show retiming with flow-based partitioning offers an excellent solution for reducing CBIT area overhead in implementing PPET. Although the probabilistic multicommodity flow algorithm in Table 3 needs several iterations (as discussed in Section 3.3) to explore the network structure, its result gives very close to an minimal cut-set solution. Therefore, PPET can be area-efficient with retiming and achieve fast testing time and high fault coverage. In addition, functional equivalence of the retimed circuit to the original one can be assured through recalculation of the initial state of retimed circuit [16].

Partitioning with retiming offers a new paradigm of implementing pseudo-exhaustive testing techniques for considering effective testing of complex circuits and systems. Even with conventional pseudo-exhaustive testing (PET) approaches [7], partitioning with retiming helps to make best use of the functional registers while reducing test hardware overhead as shown through our examples.

6 Reference

[1] C. E. Leiserson and J. B. Saxe, “Retiming Synchronous Circuitry”, Algorithmica, vol. 6, no. 1, 1991, pp.5-35.

[2] D. Kagaris and S. Tragoudas, “Partial Scan with Retiming”, Proc. 30th ACM/IEEE DAC, June 1993, pp. 249-254.

[3] S. T. Chakradhar and Suji Dey, “Resynthesis and Retiming for Optimum Partial Scan”, Proc. 31st ACM/IEEE DAC, June 1994, pp. 87-93.

[4] H. Y. Liou, T. Y. Lin, C. K. Cheng and L. T. Liu, “Circuit Partitioning for Pipelined Pseudo-Exhaustive Testing Using Simulated Annealing”, Proc. IEEE Custom Integrated Circuits Conference, May 1994, pp. 417-20.

[5] F. Brglez, D. Bryan, and K. Kozminski, “Combinational Profiles of Sequential Benchmark CIrcuits”, Proc. IEEE ISCAS, 1989, pp. 1929-34.

[6] C. W. Yeh, C. K. Cheng, and T. Y. Lin, “A General Purpose Multiple-Way Partitioning Algorithm”, Proc. 28th ACM/ IEEE DAC, June 1991, pp. 421-425.

[7] E. Wu, “A Tool for Implementing Pseudo-exhaustive Self-test”, AT&T Tech. Journal, Jan. 1991, pp. 87-100.

[8] T. Y. Lin and H. Y. Liou, “A New Framework for Designing BIT Multichip Modules with Pipelined Test Strategy”, IEEE D&T, vol. 10, no. 4, Dec. 1993, pp. 38-51.

[9] R. Tarjan, “Depth-first Search and Linear Graph Algorithms”, SIAM J. Computing, June 1972, vol.1, no.2, pp. 146-60.

[10] C. W. Yeh, C. K. Cheng, and T. Y. Lin, “A Probabilistic Multicommodity-flow Solution to Circuit Clustering Problems”, Proc. ICCAD, Nov. 1992, pp. 428-31.

[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, “Introduction to Algorithms”, MIT Press, Cambridge, Ma., 1990.

[12] S. Sastry, and A. Majumdar, “Test Efficiency Analysis of Random Self-Test of Sequential Circuits”, IEEE Trans. on CAD, vol. 10, no. 3, Mar. 1991, pp. 390-398.

[13] R. Srinivasan, S. K. Gupta and M. A. Breuer, “An Efficient Partitioning Strategy for Pseudo-Exhaustive Testing”, Proc. 30th IEEE Design Automation Conference, June, 1993, pp. 525- 530.

[14] R. L. Geiger, P. E. Allen, and N. R. Strader, “VLSI Design Techniques for Analog and Digital Circuits”, McGraw-Hill, 1990.

[15] H. Y. Liou, T. Y. Lin, and C. K. Cheng, “A Study of Pipelined Pseudo-Exhaustive Testing on VLSI Circuits with Feedback”, Proc. of ASIC’94, Sep. 1994, pp. 421-425.

[16] H. J. Trouati and R. K. Brayton, “Computing the Initial States of Retimed Circuits”, IEEE Trans. on CAD, vol. 12, no. 1, Jan. 1993, pp. 157-162.

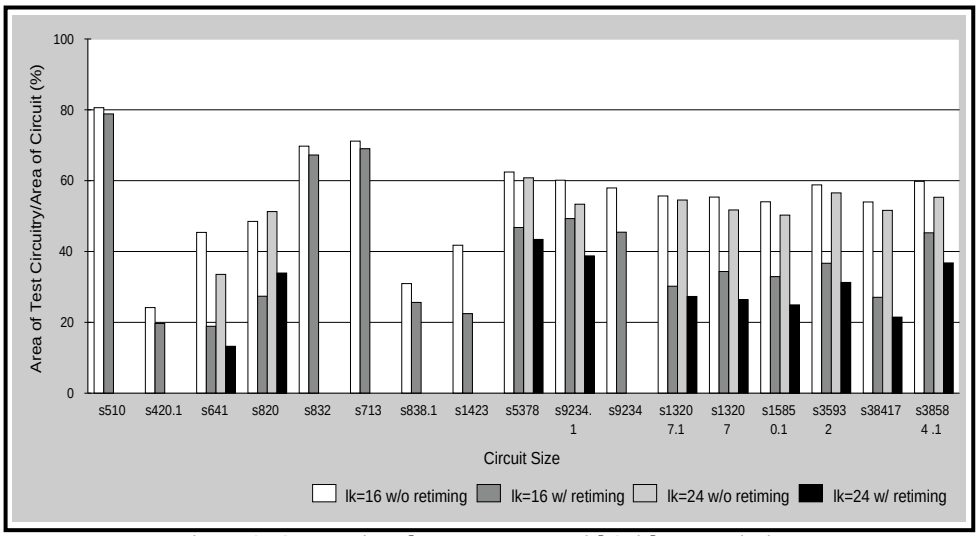


Figure 8: Comparison between PPET with/without Retiming