Partial Scan with Retiming*

Dimitrios Kagaris
Computer Science Program
Dartmouth College
Hanover, NH 03755

Spyros Tragoudas
Computer Science Department
Southern Illinois University
Carbondale, IL 62901

Abstract— A generally effective approach to the partial scan problem is to select flip–flops that break the cyclic structure of the circuit. A large number of techniques are based on this framework, but they are all static, in the sense that the flip-flops remain fixed on their original positions. In this paper, we present a method that rearranges the D flip–flops of a synchronous sequential circuit by retiming, so that the overhead of partial scan is minimized. Experiments on ISCAS'89 circuits show that retiming improves significantly both non–timing-driven and timing-driven partial scan.

I. Introduction

The sequential circuits are notoriously difficult to test. *Full scan* and *partial scan* design techniques [1] have been proposed to alleviate this problem. All flip-flops in the circuit or an appropriate subset of them are linked into a shift register so that test values can be serially loaded into the flip–flops and also their stored values can be observed. A rather effective criterion for the inclusion of flip–flops in a partial scan chain is to select flip–flops that break the cyclic structure of the circuit. This criterion forms the basis of several works [2, 12, 8, 4, 3, 11].

Almost all approaches consider the S–graph of the sequential circuit under test, that is, a directed graph $G = (V, E)$ that has a node for each flip–flop and an edge (u, v) if there is a combinational path between the output of flip–flop u to the input of flip–flop v. In terms of the S–graph, the problem is to find a set of nodes $F_G \subseteq V$, so that the graph $G' = (V', E')$ with $V' = V - F_G$ is acyclic. Obviously, in order to minimize the hardware (area) overhead, the number of selected flip–flops must be small. The above problem is equivalent to the Minimum Feedback Vertex Set (MFVS) problem which is NP-hard [7]. As proposed in [2], a solution to MFVS is allowed to leave all self–loops on the S–graph unbroken. The MFVS problem has been extented in various works to account for additional requirements on top of the acyclic property. Cheng and Agrawal [2] gave a heuristic for the MFVS problem which guarantees also that the sequential depth is no more than a given integer d. Lee and Reddy [12] have built upon this approach with additional heuristics that yield scan registers with relatively few flip–flops and high fault coverage. The same problem was addressed by Chickermane and Patel in [3, 4], where the S–graph was extended to have weights on its arcs and profits on its vertices. In another approach, Gupta et al. [8] select flip–flops that break the cyclic structure of the circuit and, in addition, make the resulting graph "balanced".

Recently, Jou and Cheng [9] considered *timing-driven* partial scan. Now the selection of scan flip–flops is guided to meet a given performance requirement T. The key observation in [9] is that the area overhead that a performance optimizer (e.g, [6]) associates with each flip–flop, depends on its time slack and can be precomputed approximately. The additional area for a flip–flop with negative slack s_f can be expressed as $\sigma \cdot (1/T - 1/(T - s_f))$, where σ is a constant. Jou and Cheng take into consideration the extra delay d_m associated with a scan flip–flop, and compute an area overhead function $AOF(F_G)$ for scanning the flip–flops in a selected set F_G as follows:

$$AOF(F_G) = \alpha \cdot |F_G| + \sum_{f_i \in F_G} u(s_{f_i}), \qquad (1)$$

where α is the area overhead due to additional circuitry associated with each scan flip–flop and the function $u(s_{f_i})$ is evaluated as follows: If $s_{f_i} \geq d_m$ then $u(s_{f_i}) = 0$, otherwise, $u(s_{f_i}) = \sigma(\frac{1}{T - s_{f_i}} - \frac{1}{T - s_{f_i} - d_m})$. Then they modify the heuristic by Lee and Reddy [12] to take into account the slacks of the flip–flops and minimize the above function.

In all of the above schemes, the positions of the flip-flops remain fixed. However, the D-type flip–flops can be repositioned so that various objective functions are optimized and the functionality of the circuit is preserved. This generic transformation on synchronous sequential circuits is known as *retiming* [13].

In this paper, we present a novel approach to the partial scan problem that transforms the circuit by retiming, so that the resulting S–graph becomes acyclic and has bounded sequential depth by selecting only a very small set of flip–flops. We show that rearranging the flip–flops so their number is minimized, guarantees a new configuration where the average number of cycles in which each flip–flop participates is maximized. This relates the retiming problem to the non–timing–driven partial scan problem, because the positions of the flip-flops after the retiming are good candidates for breaking cycles. We also present an algorithm based on retiming

*This work was partially supported by a SIGDA Design Automation Scholarship and a grant from Nissan Corporation.

that minimizes the sequential depth of the circuit after all cycles (with possible exclusion of self loops) have been broken. Finally, for timing–driven partial scan, we present a technique based on retiming and Jou–Cheng's approach [9], so that the performance requirement is met and the area overhead is minimized.

The importance of our approach is that it can be combined with the existing techniques [2, 12, 9] to further optimize the area overhead or to lead to feasible performance (timing) requirements that cannot be obtained otherwise. We have obtained significant improvements by experimenting on the ISCAS'89 benchmark circuits.

The rest of the paper is organized as follows. In Section II, we give a short overview of retiming. In Section III, we present our approach for rearranging the flip-flops for non–timing–driven partial scan. We present an algorithm for obtaining flip–flop rearrangements that become acyclic with small area overhead and also an algorithm for bounding the sequential depth. In Section IV, we focus on timing–driven partial scan. In Section V, we give the experimental results and conclude in Section VI.

II. Retiming Principles

Retiming [13] is a transformation on synchronous sequential circuits that rearranges, adds or removes the D flip–flops of the circuit without changing its functionality. In this way, various objective functions can be optimized. Leiserson and Saxe [13] presented algorithms for clock period minimization, state minimization (i.e., minimization of the number of flip–flops in the circuit), as well as state minimization under an imposed bound on the clock period. Below, we describe briefly the method in [13] for (unconstrained) state minimization.

Consider a directed graph $G = (V, E)$ for a sequential circuit, where the nodes represent the combinational elements of the circuit and the edges represent their interconnections. This graph is called the *circuit graph* of the given circuit. Each edge $e \in E$ has an associated weight $w(e)$, that equals the number of flip–flops along the corresponding connection. (For the circuits that we consider here, $w(e) = 0$ or $w(e) = 1$.)

The retiming of a circuit graph $G = (V, E)$ is defined as an assignment $r()$ of integers (positive, negative or zero) to the nodes of G so that for each edge $e = (u, v) \in E$, $r(u) - r(v) \leq w(e)$. The positions of the flip–flops in a circuit retimed by $r()$ are determined by the equation

$$w^{(r)}(e) = w(e) + r(v) - r(u),$$

where $w^{(r)}(e)$ denotes the weight of edge $e = (u, v) \in E$ under retiming r. According to whether $w^{(r)}(e)$ is zero or positive, a flip–flop should be removed from e, if it exists on e, or $w^{(r)}(e)$ flip–flops should be placed on e, if not already there. (We note that even if each edge before retiming has $w(e) \leq 1$, after retiming there may exist some edges with $w^{(r)}(e) > 1$.) All legal retimings have the property (*retiming invariant*) that they do not change the number of flip–flops in any of the cycles in the original circuit.

The *state* $S(G)$ of a circuit $G = (V, E)$ is defined as $S(G) = \sum_{e \in E} w(e)$. A retiming r with minimal state $S_r(G)$ should minimize the quantity $Q = \sum_{v \in V} r(v)(\text{ ind}(v) - \text{ outd}(v))$, where $\text{ind}(v)$ and $\text{outd}(v)$ denote the in–degree and out–degree of node v. The minimization of the quantity Q is subject to the constraint R1: $r(u) - r(v) \leq w(e)$, for each edge $e = (u, v)$. This linear programming problem has been expressed in [13] as a minimum–cost flow problem [5]. Given a minimum cost flow $f()$, the values $r()$ assigned to the nodes must be such so that R2: $r(u) - r(v) = w(u, v)$, for each edge (u, v) with $f(u, v) > 0$. The constraints R1 and R2 on the values that a retiming $r()$ must have constitute a special case of linear programming, that can be efficiently solved or determined to be inconsistent [13].

In addition, the objective function Q was generalized in [13] to take into account the potential sharing of flip–flops on the outgoing edges of a node with fanout greater than one. The state minimization in this case involves a slight modification of the layout of the circuit [13].

III. Rearranging the Flip–Flops

In this section, we present an algorithm that rearranges the flip–flops so that we obtain a small set of flip–flops that break all resulting cycles and reduce the sequential depth to no more than a given integer d. Our algorithm consists of two phases. In the first phase, we ignore the restriction on the sequential depth of the circuit (this is handled in the second phase) and concentrate on the issue of finding a minimum set of nodes that break all cycles in the S–graph.

A. Good Configurations for Breaking Cycles

Our intention is to add and remove D-type flip–flops from the given circuit, preserving the functionality of the circuit by means of the retiming invariant, in order to obtain an S–graph that is more likely to have a small MFVS solution. An illustration is given in Fig. 1.

Let C be any configuration of flip–flops obtained from the initial configuration by retiming. A solution (not necessarily optimal) of the MFVS problem will be referred to as an FV set. The FV set for the S–graph corresponding to a configuration C will be denoted by F_C. We first show that one can infer all relevant information from the circuit graph (referred to as the C–graph), without recourse to the S–graph. By the definition of the S–graph, for each cycle in the C–graph that contains flip–flops $f_1, f_2, ..., f_k$, $k \geq 1$, there corresponds a cycle in the S–graph with vertices $f_1, , f_2, ..., f_k$. Such a cycle will be called *basic*. The S–graph may contain additional cycles that have no corresponding cycle in the C–graph. However, each vertex of the S–graph belongs to some basic cycle. Therefore, a minimal FV set on the S–graph, is equal to a minimal set of flip–flops that must be removed in order to break all cycles in the C–graph. More formally, we can consider all edges with flip–flops in the C–graph labeled "black" and all other edges labeled "white". We want to remove the smallest number of black edges, so that the graph becomes acyclic. We will refer to such a set of black edges as black feedback Arc (FA) set. The corresponding problem in a graph whose all edges are black, is known as the Minimal Feedback Arc Set (MFAS) problem and is also NP–hard [7]. By means of this analogy, we have

Lemma 1: A minimal FV set on the S–graph amounts to a minimal black FA set on the C–graph.

Given the above lemma, a good rearrangement of flip–flops in the circuit for a minimal FV set in the S–graph would seem to be the one obtained by applying the state minimization algorithm of [13] on the C–graph. Because of the retiming invariant, all cycles in the C–graph must have the same number of flip–flops (black edges), both in the original and the optimal configurations. Consequently, the number of flip–flops will decrease only if some of them are shared among several cycles, which amounts to the flip–flops being placed on edges common to many cycles (many cycles having a common black edge).

Let $\mathcal{F} = \min_C\{F_C\}$ denote an optimal FV set that has size less or equal to the size of any other FV set F_C obtained under a configuration C. The problem of finding a configuration C' with $F_{C'} = \mathcal{F}$ is, obviously, NP–hard. Nevertheless, a minimal state configuration C_m is one of the strongest candidates among all possible flip–flop configurations for yielding an FV set $F_{C'} = \mathcal{F}$. This is justified in the following sense:

Lemma 2: In a minimal state configuration, the average number of cycles that each flip–flop participates is maximum.

However, the above lemma does not guarantee optimality. (An example is shown in Fig. 1c). The reason is that two or more flip–flops may participate in exactly the same cycles. In contrast, we would prefer a small number of flip–flops to participate in many non–overlapping groups of cycles. In order to obtain a configuration that yields an FV set close to $\mathcal{F}$, we take the following measures.

First, we use the algorithm of [13] for state minimization that takes into account fan–out sharing (see Section II). If two or more flip–flops have been placed on the outgoing edges of the same node in the circuit graph, all these flip–flops can be replaced by a single one, with a circuit modification as in [13]. In view of this transformation, a configuration that contains many groups of flip–flops, so that the flip–flops in each group are placed on the outgoing edges of the same node, greatly increases the average number of cycles per flip–flop in the modified graph.

Lemma 3: Let C_1 be a configuration such that there are at least two flip–flops on the outgoing edges of the same node. Then a configuration C_2 in which these two flip–flops have been united by a fan–out absorption, has an FV set F_{C_2} with $|F_{C_2}| < |F_{C_1}|$.

As an additional measure, we do not allow more than one flip–flops per edge on any retimed configuration. (We call such a configuration *binary*). Obviously, more than one flip–flops on the same edge are redundant, as far as the derivation of an FV set is concerned. The initial configuration has at most one flip–flop per edge, but as mentioned before, the original state minimization algorithm of [13] may end up placing more than one flip–flops on the same edge. The presence of a flip–flop on an edge $e = (u, v) \in E$ under a retiming r is determined by the equation $w^{(r)}(e) = w(e) + r(v) - r(u)$, where the $r()$ and $w()$ functions are as in Section II. In order to have $w^{(r)}(e) \leq 1$, we must have $r(v) - r(u) \leq 1 - w(e)$. Since the R1 constraint must still hold, we require also $r(u) - r(v) \leq w(e)$. In [10], we have shown that the corresponding linear programming problem can be formulated as the dual of a minimum–cost flow problem by expanding the edge set E with the set E' of reversed edges. A solution to this minimum cost flow problem yields a binary minimal state configuration as follows. (Proof is given in [10]).

Theorem 1: A binary minimal state configuration is obtained by assigning integer values $r()$ to the vertices of the C–graph so that the following system is satisfied:
(a) $r(u) - r(v) = w(u, v)$, for each edge $(u, v) \in E$ with positive flow.
(b) $r(u) - r(v) = 1 - w(v, u)$, for each edge $(u, v) \in E'$ with positive flow.
(c) $r(u) - r(v) \leq w(u, v)$, for each edge $(u, v) \in E$ with zero flow.
(d) $r(u) - r(v) \leq 1 - w(v, u)$, for each edge $(u, v) \in E'$ with zero flow.

Below we define two more retiming restrictions that are useful in our scheme. An edge is called *prohibitive*, if it is not allowed to host any flip–flop. We make an edge $e = (u, v)$ prohibitive by requiring that $w^{(r)}(e) = 0 \Leftrightarrow r(u) - r(v) = w(e)$. Also we call a flip–flop *fixed*, if it must stay on its edge. If edge $e = (u, v)$ has a flip–flop on it, i.e., $w(e) = 1$, we can fix it by requiring that $w^{(r)}(e) = 1 \Leftrightarrow r(u) - r(v) = 0$.

A binary state configuration can still be enhanced in order to reduce the number of flip–flops that end up participating in exactly the same cycles. A cause of redundancy is the presence of *chain paths*. A chain path is a path $(u_0, u_1, ..., u_k)$ such that nodes u_i, $1 \leq i \leq k$ have in–degree 1 and out–degree 1. More than one one flip–flops on the same chain path cause the same problem as more than one flip–flops on the same edge. We cannot contract the path to a single edge and require that it receives at most one flip–flop, because this may cause infeasibility. However, we can safely contract any sub–paths of the chain path that have no flip–flops on them in the initial configuration. In terms of a retiming restriction, we can make all such edges prohibitive.

Our technique, we call it REARRANGE , starts by identifying all chain paths in the circuit graph . This is a structural property of the graph that does not change by retiming. We also apply a routine CYCLEDGE that assigns to each edge e of G an estimated count $cc(e)$ of how many cycles contain that edge. (Routine CYCLEDGE is described in [10].) Then we select a number c of flip–flops that belong to a (different) chain path and have largest $cc()$ counts. We fix all these flip–flops on their respective edges, and make prohibitive all edges that belong to the selected chain paths and do not have a flip–flop on them. Then we apply the state minimization algorithm of [13] with fan–out sharing for binary states.

After procedure REARRANGE the flip–flops have been moved to new positions. Let $G_c^S = (V_c^S, E_c^S)$ be the corresponding S–graph. We apply a heuristic for the MFVS problem on G_c^S—we call it Approximate Vertex Set (AVS)— to find a small set of flip–flops F so that the graph G_a with nodes in $V_c^S - F$ is acyclic. The AVS heuristic uses CYCLEDGE and is fully described in [10].

B. Bounding the Sequential Depth

In the second phase of our approach, we concentrate on selecting additional flip–flops to guarantee that the sequential depth of the circuit graph G_c is no more than a given integer d. The sequential depth is defined as the maximum distance of any two vertices in the resulting acyclic S–graph. Lee and Reddy [12] gave a heuristic for the above problem that obtains a solution by finding a small FV set on a graph $G_x = (V_x, E_x)$ with $V_x = V_s$ and $E_x = E_s \cup \{(u,v) |$ length of any path from u to v is $d+1\}$.

Our technique referred to as R_SD has two steps. In the first step, we use a retiming based algorithm in order to reduce the sequential depth of the circuit by flip–flop rearrangements. Let d_0 be the initial sequential depth in the resulting acyclic S–graph and d the prescribed bound. We present a polynomial–time algorithm (referred to as SD) that rearranges the unscanned flip–flops in the S–graph so that the resulting sequential depth d_m is the *minimum* among all configurations that are legal under retiming. If $d_m > d$ then we use, in a second step, Lee–Reddy's algorithm for removing some flip–flops and guaranteeing the depth bound. We observed that after the preprocessing by algorithm SD, the number of flip–flops needed to reduce the depth from d_m to d is much smaller than the number to reduce the depth from d_0 to d. An example is shown in Fig. 2.

Below we describe algorithm SD. We consider the circuit graph $G_a = (V_a, E_a)$ with a flip–flop configuration C determined by routine REARRANGE of the previous Section. Let F_C be the set of flip–flops that were selected for breaking all cycles in G_a under configuration C. We transform G_a to a slightly different graph $G_d = (V_d, E_d)$ as follows: For each flip–flop $f \in F_C$ on edge (u,v), we introduce two unique nodes f_i and f_o so that u points to f_i, f_i points to f_o and f_o points to v. The new edges are assigned weights $w(u, f_i) = 0$, $w(f_i, f_o) = 1$ and $w(f_o, v) = 0$, with all other edges having the same weights as before. Let PI be the set of the primary input nodes of G_d and SI the set $SI = \{f_o : f \in F_C\}$. Similarly, let PO be the set of the primary output nodes of G_d and SO the set $SO = \{f_i : f \in F_C\}$. A *sequential path* $p = (u_0, u_2, ..., u_k)$ in G_d is a path such that $u_0 \in PI \cup SI$, $u_k \in PO \cup SO$ and $u_i \notin PI \cup SI \cup PO \cup SO$, for any $u_i, 1 \leq i \leq k-1$. The *length* of a sequential path $p = (u_0, u_2, ..., u_k)$ is defined as $w(p) = \sum_{i=0}^{k-1} w(u_i, u_{i+1})$. We denote by $w_{\max}(u,v)$ the maximum length of any sequential path. The sequential depth D_C in the acyclic S–graph of G_a under configuration C is defined as the maximum length among all sequential paths in the corresponding graph G_d.

We want to rearrange the flip–flops of configuration C in G_d by a retiming $r()$ so that we obtain a new configuration C_r with the sequential depth D_{C_r} being minimum. We have the following theorem (the proof is given in [10]):

Theorem 2: Let k be an arbitrary positive integer and C_r a configuration obtained by a retiming $r()$ on G_d. Then $D_{C_r} \leq k$ if and only if:
(a) $r(u) - r(v) \leq w(u,v)$, for every edge $(u,v) \in E_d$.
(b1) $r(u) - r(v) = 0$, for every edge $(u,v) = (f_i, f_o)$ for some $f \in F_C$.
(b2) $r(u) - r(v) = 0$, for every edge $(u,v) = (u, f_i)$ or $(u,v) = (f_o, v)$ for some $f \in F_C$.
(c) $r(v) - r(u) \leq k - w_{\max}(u,v)$, for every pair of nodes $u \in PI \cup SI, v \in PO \cup SO$.

The minimum sequential depth can now be derived as follows (details are given in [10]):

```
Algorithm SD (sequential depth minimization)
1. Obtain graph $G_d$ from $G_a$.
2. Compute $w_{\max}(u,v)$ for every pair $u \in PI \cup SI, v \in PO \cup SO$.
3. Assign values to $k$ by binary search in the interval $[1..f]$, where $f$ is the number of unscanned flip–flops in the circuit. For each $k$, check whether the conditions in Theorem 2 can be satisfied by applying the algorithm in [5] for solving *difference constraints*.
4. The retiming $r()$ for the minimum achievable value of $k$ yields the configuration $C_r$ with minimum sequential depth $D_{C_r}$.
```

We have also modified SD to control the total number of flip–flops in the circuit [10].

IV. Timing–Driven Partial Scan

In this section we focus on timing–driven partial scan where the clock period after the inclusion of the scan logic must meet a time bound T. The goal here is to obtain (a) an efficient algorithm for rearranging the flip–flops and (b) an efficient algorithm for selecting flip–flops to break cycles, so that after performance optimization (if needed), the area is minimized. As target delay T, we use approximately half the original clock period of each circuit, as in [9]. We also consider the delay d_m, added to each flip–flop selected for partial scan [9]. We experimented with alternative schemes, which we briefly describe below.

In our first attempt, we extended the heuristic for the non–timing driven partial scan as follows: After the application of REARRANGE (see Section III), there are some flip–flops that can still be rearranged. We apply the clock period minimization algorithm of Leiserson and Saxe [13] (referred to as CP) in order to obtain a minimum clock period T_c. Since CP may introduce additional flip–flops, we did not allow the tolal number to increase by more than 5%. If $T_c > T - d_m$, then we used performance optimization (as in [6]) to achieve the time bound, if possible. This approach did not give satisfactory results for time bounds T with $T < T_{c'} + d_m$, where $T_{c'}$ is the clock period of the circuit after the application of REARRANGE. We observed that retiming does not help that much at this late stage. Very often the clock period has been determined between a pair of flip–flops on a cycle that has fixed flip–flops only.

In our second attempt, we used again algorithm CP for rearranging the flip–flops and obtaining a minimum clock period T_c. If $T_c \leq T - d_m$ we called the AVS heuristic for selecting the scan flip–flops. If $T_c > T - d_m$, we applied the heuristic by Jou–Cheng [9]. The results were not satisfactory even when $T_c \leq T - d_m$. The reason is that algorithm CP often increased the number of flip–flops and the final rearrangement was not good for the AVS heuristic. Subsequently, we modified algorithm CP so that it selects rearrangements that do not increase the number of flip–flops. Again, the flip-flop rearrangement was not good for AVS (in comparison with our final scheme), even for the case where $T_c \leq T - d_m$.

Finally, we chose the following scheme: We first use the algorithm in [13] (referred to as SMCP), that obtains state minimization under a clock period bound. (The algorithm is modified so that the allowable flip-flop configurations are binary.) We set $T_c = T - d_m$. If the number of flip–flops is not increased with respect to the initial configuration, then we apply procedure REARRANGE. If there is no such configuration or if the number of flip–flops is increased, with respect to the initial configuration, then we apply SMCP for all clock periods T_c in the range $(T - d_m, T]$. Let $\mathcal{S}$ be the set of the resulting configurations. ($\mathcal{S}$ contains at least the initial configuration). For each configuration $i \in \mathcal{S}$, we run the heuristic in [9] to select a set of flip–flops F_i for partial scan. For each configuration i, we compute the area overhead $AO(i) = AOF(F_i) + FFA(F_i)$, where $AOF(F_i)$ is the area overhead function given in (1) and $FFA(F_i)$ is a function that computes the total flip–flop area for configuration i, without considering the extra circuitry associated with each scan flip–flop. We select the configuration with minimum $AO(i)$ as the best solution.

V. Experimental Results

We implemented our algorithms in C and run on a Sun Sparc System 2. We experimented on several ISCAS'89 benchmarks. All flip–flops here are D flip–flops. Our first set of experiments compares our non–timing–driven partial scan algorithm (denoted by R_ntd) over the Lee-Reddy algorithm (denoted by LR). Table 1 shows the results for making the S–graph acyclic. We omit cases where the absolute difference in the flip–flop numbers was below 9. Entries with an asterisk show the corresponding requirements, when self–loops are allowed to remain in the S–graph, as proposed in [2]. Although the overhead is much smaller when self–loops are allowed to remain in the S–graph, this may occasionally cause difficulties during test pattern generation, and in some applications (like pseudoexhaustive testing, [16]), *all* cycles must be broken.

In Table 2, we report results when the sequential depth was set to be 3 and 7. In this Table, the self–loops were allowed to remain in the S–graph. Both tables indicate that our approach helps reduce the hardware overhead of partial scan significantly. (Currently we are examining the effects on fault coverage and fault efficiency by using the HITEC [14] sequential test pattern generator. However, as observed in [9, 2], the fault coverage of partial–scan schemes based on cycle–breaking are very high and our initial observations testify to this fact.)

In Table 3, we examine the effects of our timing–driven algorithm (denoted by R_td). We show approximate area overhead reductions that were achieved over the non–timing driven version R_ntd and Jou–Cheng's heuristic [9] (denoted by JC). We computed first the clock period (in nsecs) of each circuit, given in column 2 of Table 3. We assumed 10 ns for NOT, AND, NAND and NOR and 14 ns for an OR gate [15]. Also we assumed a delay $d_m = 20$ ns for a 2–to–1 multiplexer. We then set the target delay for each circuit to half its original clock period minus d_m as was also done in [9]. Let $A_{\mathrm{R\_td}}, A_{\mathrm{R\_ntd}}, A_{\mathrm{JC}}$ be the total areas required in order to achieve the target delay after the scan flip–flops have been selected by R_td, R_ntd and JC, respectively. In columns 3 and 4 of Table 3, we report the relative area overhead savings (RAOS) of R_td over R_ntd and JC in the form $\frac{A_x - A_{\mathrm{R\_td}}}{A_x}$, where x is R_ntd or JC respectively.

Since we were not able to obtain the performance optimizer used in [9], we based our comparisons on an estimated area overhead given by (1). This requires the determination of the flip–flop slacks and the constant σ representing the area–delay product [9]. For the comparative purposes here, we computed the slacks (in a breadth–first search manner) and set the σ constant for each circuit to be the product of its clock period (given in column 1) and a quantity determined by the number of gates in the circuit as well as the fan–in and fan–out of each gate.

In what follows, we explain some of the reasons for the superiority of R_td over JC and R_ntd. R_td always guarantees no more area overhead than JC, if there is no retiming with clock period $T - d_m$ and no more flip–flops than the initial configuration. This happens because the set $\mathcal{S}$ contains at least the initial configuration. Consider now the case, where we can obtain a clock period $T - d_m$ without increasing the number of flip–flops. Our algorithm uses the AVS algorithm in this case. One can argue, however, that we could use Saxe's algorithm to obtain state minimization for a clock period close to $2 \cdot T$ and then run the performance optimizer [6] which drops that critical path approximately by 50%. By simulating the area overhead that would result in this case (using (1)), we found that for these benchmarks such a scenario never gave better results.

Also R_td is always no worse than R_ntd in terms of area overhead. Note that if the R_td algorithm can achieve a clock period $T - d_m$ by calling algorithm SMCP, then the number of flip–flops will be no more than the number of flip–flops in the non–timing–driven algorithm, provided that the non–timing–driven algorithm gives clock period no more than $T - d_m$. One can argue that the non–timing–driven algorithm could give a minimum number of flip–flops with clock period in the range $(T - d_m, T]$ (without considering the extra delays for the scanned flip–flops) and that, subsequently, the scanned flip–flops by the AVS algorithm were not on critical paths. This is the single case where the non–timing–driven algorithm can be superior to our timing–driven algorithm because now the number of flip–flops in the latter scheme is more. Experimentation showed that this did not happen. In almost all cases that we tried, the non–timing–driven scheme fails to meet the time bound.

VI. Conclusions

We presented algorithms that use retiming to minimize the hardware overhead for partial scan. The importance of our work is that it can be built over existing techniques for partial scan (timing–driven, non–timing–driven). Experimentation on the ISCAS'89 benchmarks showed that retiming always helps partial scan.

TABLE 1
Comparison of R_ntd with Lee–Reddy [12] for acyclic S–graph.

Circuit	R_ntd	LR
s9234	136	152
s9234*	44	53
s15850	410	441
s15850*	64	88
s13207	295	310
s38417	1041	1080
s38417*	363	374

TABLE 2
Comparison of R_ntd with Lee–Reddy [12]
for sequential depth 3 and 7

Circuit	$d = 3$		$d = 7$		No bound	
	R_ntd	LR	R_ntd	LR	R_ntd	LR
s9234	102	109	74	88	44	53
s13207	134	153	112	120	58	59
s15850	229	252	186	221	64	90
s38417	728	766	511	539	363	374

TABLE 3
Relative Area overhead savings (RAOS) of R_td over R_ntd and JC.

Circuit	CP	RAOS R_ntd	RAOS JC
s9234	596	24%	23%
s13207	598	10%	16%
s15850	828	18%	22%
s38417	506	12%	15%

References

[1] M. Abramovici, M. A. Breuer, A. D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, New York, 1990.

[2] K.-T. Cheng, V. D. Agrawal, "A Partial Scan Method for Sequential Circuits with Feedback", *IEEE Trans. Computers*, Vol. 39, No. 4, pp. 544–548, April 1990.

[3] V. Chickermane, J. H. Patel, "An Optimization Based Approach to the Partial Scan Problem", *IEEE Int. Test Conf.*, pp. 377–386, 1990.

[4] V. Chickermane, J. H. Patel, "A Fault Oriented Partial Scan Approach", *IEEE ICCAD'91*, pp. 400-403, Nov. 1991.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge MA, 1990.

[6] J. P. Fishburn, "A Depth-Decreasing Heuristic for Combinational Logic", *27th ACM/IEEE Design Automation Conf.*, pp. 361–364, June 1990.

[7] M. R. Garey, D. S. Johnson, *Computers and Intractability*, W. H. Freeman and Co., New York, 1979.

[8] R. Gupta, R. Gupta, M. A. Breuer, "The BALLAST Methodology for Structured Partial Scan Design", *IEEE Trans. Computers*, Vol. 39, No. 4, pp. 538–544, April 1990.

[9] J.-Y. Jou, K.-T. Cheng, "Timing–Driven Partial Scan", *IEEE ICCAD'91*, pp. 404-407, Nov. 1991.

[10] D. Kagaris, S.Tragoudas, "Partial Scan with Retiming", Technical Report, CS Dept., Southern Illinois University, March 1993.

[11] A. Kunzmann, H.-J. Wunderlich, "An Analytical Approach to the Partial Scan Problem", *J. Electronic Testing: Theory and Applications*, Vol. 1, pp. 163–174, 1990.

[12] D. H. Lee, S. M. Reddy, "On Determining Scan Flip–Flops in Partial–Scan Designs" *IEEE ICCAD'90*, pp. 322–325, Nov. 1990.

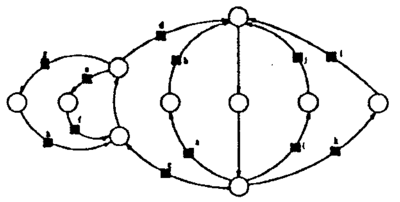
[13] C. E. Leiserson, J. B. Saxe, "Retiming Synchronous Circuitry", *Algorithmica*, Vol. 6, pp. 5–35, 1991.

[14] T. M. Niermann, J. H. Patel, "HITEC: A Test Generation Package for Sequential Circuits", *European Conf. on Design Automation*, pp. 241–218, 1991.

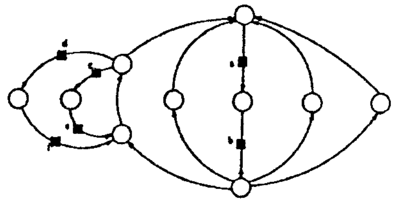
[15] Texas Instruments Eng. Staff, *The TTL Data Book for Design Engineers*, 2nd ed., Texas Instruments Inc., Dallas TX, 1976.

[16] H.-J. Wunderlich, S. Hellebrand, "The Pseudoexhaustive Test of Sequential Circuits," *IEEE Trans. on Computer–Aided Design*, vol. CAD-11, pp. 26–32, Jan. 1992.

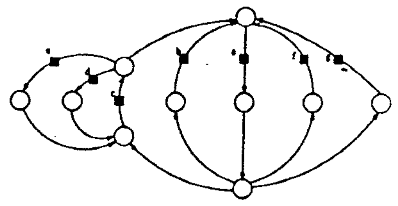
Figure 1: State minimization and minimum feedback vertex set.



(a) A graph with 12 flip–flops. MFVS $=\{a,c,e,g,i,k\}$.



(b) State minimization: 6 flip–flops. MFVS $=\{a,c,d\}$.



(c) A retiming with 7 flip–flops. MFVS $=\{a,c\}$.

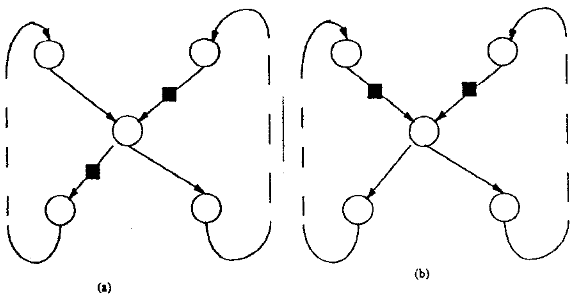


Fig. 2. Effect of retiming onsequential depth. (a) The sequential depth is 2. (b) After retiming, the sequential depth is 1.