Performance Driven Resynthesis by Exploiting Retiming-Induced State Register Equivalence

Priyank Kalla and Maciej J. Ciesielski
Department of Electrical and Computer Engineering, University of Massachusetts at Amherst
Amherst, MA-01003, USA
{pkalla, ciesiel}@ecs.umass.edu

Abstract: *This paper presents a retiming and resynthesis technique for cycle-time minimization of sequential circuits circuits with feedbacks (finite state machines). Operating on the delay critical paths of the circuit, we perform a set of controlled local retimings of registers across fanout stems and logic gates, followed by local node simplifications. We guide the retiming of registers across fanout stems to induce equivalence relations among them, which are exploited for subsequent logic simplification. Our technique is able to analyze correlation of logic across register boundaries during simplification. We strive to minimize the increase in number of registers without sacrificing the cycle-time performance. The results demonstrate a favourable performance/area trade-off when compared with optimally retimed circuits.*

I. Introduction

Conventional sequential synthesis techniques apply a variety of heuristic transformations that target the optimization of the combinational logic components separated by register boundaries. Sequential optimization, such as retiming [1], is generally applied on gate-level networks as a post-processing step. Application of such combinational logic optimization techniques on separate blocks is restrictive inasmuch as it does not allow the interactions between gates separated by register boundaries for examination in the optimization process.

Retiming, on the other hand, is the process of relocating registers across logic gates, without affecting the underlying combinational logic structure. In effect, it borrows logic from one time frame and lends it to another, while maintaining the design behaviour. It can be used for cycle-time minimization or for register minimization under cycle-time constraints [2]. Recent research has significantly improved the efficiency and modeling accuracy of gate-level retiming [3], [4], [5]. In addition to timing optimization, it has also been used for low power design [6], [7], [8]. These and other works have sparked further interest in exploring retiming as a general optimization technique during architectural [9] and logic synthesis. Despite all these advances, the potential for gate-level retiming to achieve significant circuit optimization remains limited.

Retiming is guided by the minimization of cycle-time, which is based on a precomputed function of the location of registers in the network, and not by a prospective logic simplification. As it exploits only one degree of freedom in circuit optimization, namely, the relocation of registers, the potential for optimization by subsequent resynthesis is limited. Also, retiming techniques for cycle time minimization have no control over the increase in the number of registers. A significant increase in the number of registers not only affects the area of the resulting circuit, but also complicates testing and verification. For such reasons there have been many attempts to combine retiming with network transformations in order to optimize logic across register boundaries [10] [11] [12] [13] [14]. However, no satisfactory solutions have emerged - either the cycle time improvement comes at prohibitive area overheads or it has been marginal.

This work has been supported in part by a grant from NSF under contract No. MIP-9613864

In this paper, we present a novel retiming and resynthesis technique for cycle-time minimization of sequential circuits. Our technique operates on, though is not restricted to, the critical paths of a gate-level netlist. At the heart of our procedure is a set of *controlled* local retimings across fanout-stems and logic gates and local node simplifications. We guide the retiming of state registers across fanout stems in order to induce *equivalence relations* among them. These equivalence relations are exploited, subsequently, for logic simplification.

Forward retiming of a register across its fanout stem creates equivalent registers - this fact appears trivial in concept. However, exploitation of these equivalence relations, which are viewed as a special class of don't cares, offers a freedom to analyze the correlation of logic across register boundaries during simplification. We show that these equivalence relations (don't care conditions) translate into EX-NOR (EX-OR) forms of retimed registers and can *directly* be used for logic simplification. Guiding the retiming and using these equivalence relations in subsequent logic simplification leads to a performance enhancement beyond what is achievable by conventional retiming and resynthesis procedures. We strive to minimize the increase in the number of registers without sacrificing the cycle-time performance of the resynthesized circuit. Experimental results demonstrate that, as compared to retiming, our technique results in circuits not only with fewer registers but also with better cycle-time performance. We provide a simple method to compute the initial state of the modified circuit consistent with the original network specifications.

In the sequel, we consider synchronous implementations of sequential circuits using edge-triggered flip-flops. We assume that the circuits have a known initial state, and can always be driven to that initial state either by explicit reset circuitry, or by application of a synchronizing sequence. In the following section, we address some important preliminary concepts regarding retiming of registers, their initial state computation and don't cares. Section III explains our retiming and resynthesis technique by means of a simple example. Section IV presents a practical algorithm and discusses related important issues. The results are presented and analyzed in Section V and the paper is concluded.

II. Retiming Across Fanout Stems: Register Equivalence

Forward retiming across a single output node is the operation of shifting the registers from the input edges to the output edge of a node in a Boolean network; *backward retiming* is the reverse operation. The node can represent an arbitrary Boolean function. Forward and backward retiming transformations across a single output node are illustrated in Fig. 1. It has been shown that such a transformation preserves the behaviour of the circuit [1] [15] [16]. The circuit resulting from a series of forward and/or backward retiming moves across *single output nodes* is *space-equivalent* to the original one[16]; for every state in the retimed circuit, there is an equivalent state in the original one and vice-versa.

Computation of initial states of registers after their forward or backward retiming across *single output* logic blocks (nodes) is always possible as addressed in [17] [15] [16]. Let a and b be the initial states of

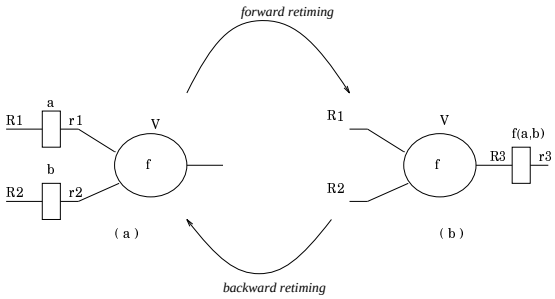

Fig. 1. Retiming across a logic node.

registers R_1 and R_2 as shown in Fig. 1. Let f be a Boolean function implemented by node V. Forward retiming of R_1 and R_2 across the node V results in register R_3. The initial state of R_3 can be computed as $f(a, b)$, and vice-versa for backward retiming.

Retiming across a multiple-fanout stem junction is the operation of retiming of a "multiple-fanout" register across its fanout stem junction as illustrated in Fig. 2. Forward retiming of register R_1 across the fanout stem V_x, results in two registers R_{11} and R_{12}. Similarly, backward retiming of registers R_{11} and R_{12} across the fanout stem V_x would result in R_1. However, it may not always be possible to backward retime registers R_{11} and R_{12} because of constraints on the initial states. If R_{11} and R_{12} have different initial values, then there exists no equivalent initial value assignment for the register resulting from backward retiming.

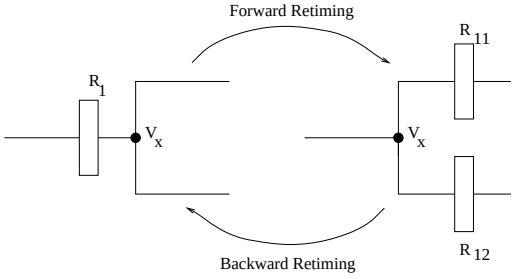

Fig. 2. Retiming across a fanout stem.

The forward retiming of registers across their fanout stems is of interest to us. In general, it is not guaranteed that space equivalence relation holds under forward retiming across multiple-fanout stem junctions [1] [15] [16]. This is because not every value on the outputs of a multiple fanout stem has an input value that produces the desired outputs. Consider, again, the forward retiming across a fanout stem as depicted in Fig. 2. As a result of this retiming, the STG of the original hypothetical circuit is transformed from that in Fig. 3(a) to that in Fig. 3(b). Not every state in the retimed circuit has an equivalent state in the original circuit (state $\langle 01 \rangle$ and $\langle 10 \rangle$ have no equivalent states in the original circuit). This implies that the two circuits may not be space equivalent as their behaviour might be different. However, this problem is restricted to the initial state operation of the retimed circuit. If we can guarantee *preservation* of the initial state, the new states created by forward retiming across fanout stem can be declared invalid. These invalid states in the retimed circuit, that do not have an equivalent counterpart in the original circuit, can be considered a managable performance optimization resource rather than a liability.

Let R_1^0 be the initial state of register R_1 in the original network. Let the initial states of registers R_{11} and R_{12} in the retimed network be R_{11}^0 and R_{12}^0 respectively. To guarantee preservation of initial states [15], both R_{11}^0 and R_{12}^0 have to be equal to R_1^0. This follows directly from the definition of retiming across fanout stems [18] in order to maintain the design behaviour. If explicit reset circuitry is available, the

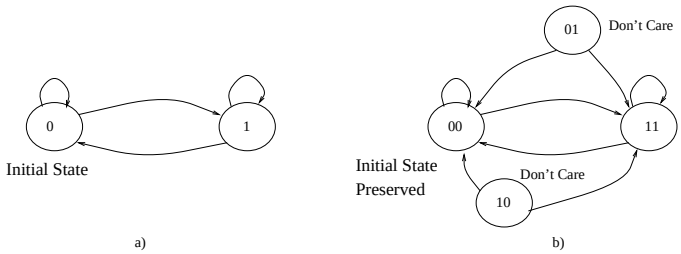

Fig. 3. STG transformation after forward retiming across fanout stem.

initial state equivalence can be maintained easily [17][19]. If, however, explicit reset circuitry is not available, then we need to compute a synchronizing sequence to drive the retimed circuit to its corresponding equivalent initial state. Notice that the original synchronizing sequence cannot always be used for this purpose as it may not be preserved under such a retiming. Though structural synchronizing sequences (based on conservative 3-valued simulation) are preserved, functional synchronizing sequences (those derived by the STG of the circuit) need to be prepended by a prefix sequence of a predetermined number, N, of arbitrary input vectors. Here N is equal to the number of atomic forward retiming moves across fanout stems [16]. Singhal *et al.* [15] refer to this as *delayed replacement* as opposed to *safe replacement.* Thus, if we clock our *delayed design* of Fig. 3(b) for $N = 1$ clock cycles after power-up, we can never reach the states $\langle 01 \rangle$ or $\langle 10 \rangle$. The delayed design consists only of states $\langle 00 \rangle$ and $\langle 11 \rangle$ and is equivalent to our original design.

The registers produced from this type of retiming, with preservation of initial states under safe or delayed replacement, have the constraint that their outputs be equal at all times of valid circuit operation. This imposes an *equivalence relation* or equivalently, *don't care conditions* on the retimed registers, and the registers are said to be equivalent. Mathematically, if R_{11} and R_{12} are equivalent registers, then the equivalence relation can be represented as: $R_{11} \overline{\bigoplus} R_{12}$, as both registers have to have the same values. We define such an equivalence relation as *retiming-induced state variable equivalence.* Conversely, since these registers will have the same values at all times, the possibility of them having different values can be considered as a *don't care* condition. This can be mathematically expressed as: $R_{11} \bigoplus R_{12}$, and is defined as *retiming-induced don't care* condition and represented as DC_{ret} throughout the rest of the paper.

Algorithms for don't care set computation [20] [21] [22] that work at the network level without any state transition information, will not identify R_{11} and R_{12} as equivalent registers, as they may have different initial values. Since we use retiming across fanout stems as a resynthesis operation, we can explicitly maintain the equivalence relation without any loss of generality. Indeed, implicit state enumeration methods [23][24] can be used to compute reachable and unreachable states of a machine. These illegal (or invalid) states can then be used as external don't cares in subsequent logic optimization [25][26]. However, implicit state enumeration methods using BDDs are computationally intensive and prohibitive in memory requirements for large circuits. In contrast, we do not have to perform any computation to evaluate these retiming induced don't care conditions.

III. Resynthesis using Retiming-Induced *Don't Cares*

In this section, we present the basic concepts behind our resynthesis technique and show, by means of a simple example, how we can guide the retiming across fanout stems and logic gates, and use the retiming-induced don't care conditions for subsequent logic simplification. An algorithm to perform the retimings and node simplifications is formally presented in the next section.

Consider a sequential circuit specified by the following optimized

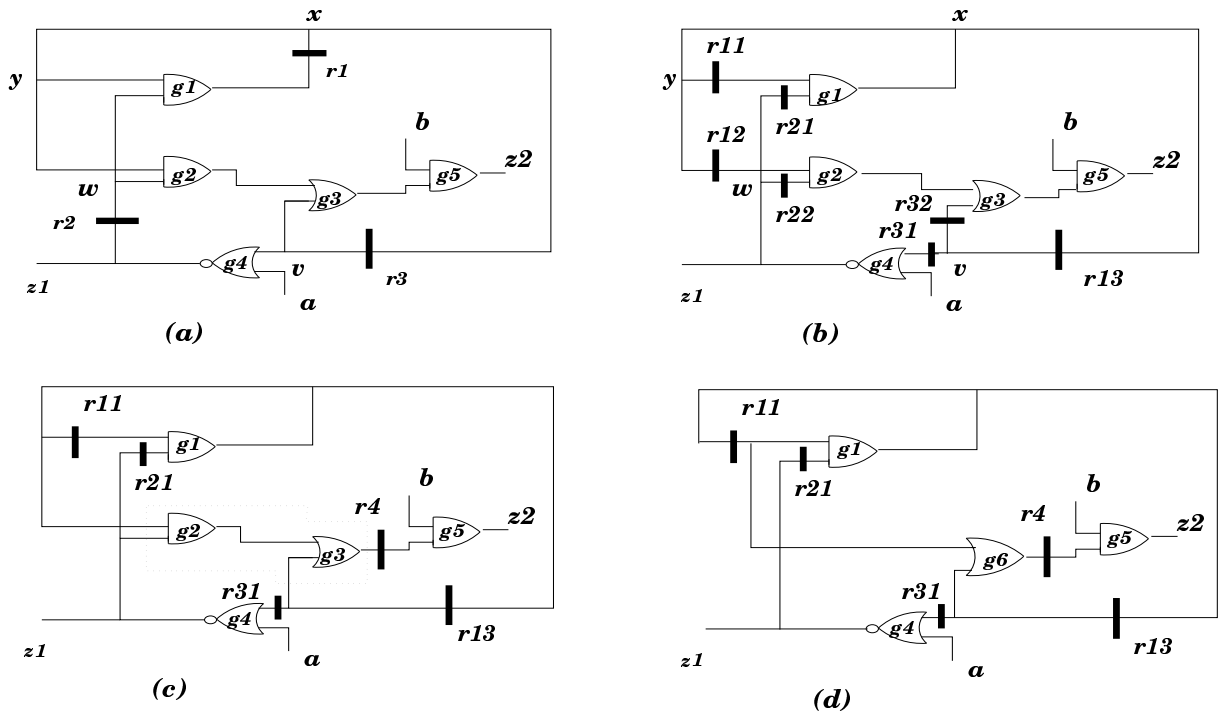

Fig. 5. Interpretation of proposed retiming and resynthesis: a) original circuit with gate g_1 duplicated, b) circuit after forward retiming of r_1, r_2, r_3 across their respective fanout stems, c) circuit after forward retiming across gates g_2, g_3, d) simplification using DC_{ret}.

functional equations: $R_1 = r_1 r_2; R_2 = \overline{a + r_3}; R_3 = r_1; z_1 = \overline{a + r_3}; z_2 = b(r_1 r_2 + r_3)$. **Notation:** a, b are the inputs, z_1, z_2 are the outputs, r_i the present states and R_i the next state variables. Our objective is to find an implementation of the circuit with minimum cycle time. Assume, for simplicity, the unit delay model. The network, when mapped directly onto basic 2-input logic gates, results in the circuit shown in Fig.4(a). The longest delay in the combinational logic (cycle-time) is equal to 3 gate delays (g_1, g_3, g_5). The circuit after retiming, shown in Fig. 4(b), has a delay of 2 gates. Analysis of this solution reveals that it can be obtained by forward retiming across gate g_1 (this result was verified by SIS). Conventional retiming cannot reduce the delay of the circuit any further.

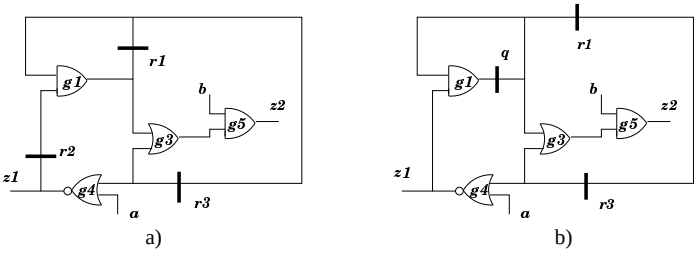

Fig. 4. Retiming of an optimized circuit; a) original circuit, b) retimed circuit.

We shall now show that it is possible to obtain a circuit, from the original specification, with a delay of just one gate. Consider again the original circuit shown in Fig. 4(a). Let us make the critical path (g_1, g_3, g_5) fanout free (the reason for which will be apparent later). This process may involve duplication of certain gates in the network. Starting from the final connection of the longest path (z_2), we perform a backward traversal towards the fanout at the output of the registers (r_1 and/or r_2). The first point at which a fanout is encountered is the output of gate g_1. Hence, we duplicate gate g_1. The resulting circuit is shown in Fig. 5(a). Let us now perform a series of forward retimings across fanout stems: 1) forward retiming of register r_1 across fanout stems x and y, creating registers r_{11}, r_{12} and r_{13}; 2) forward retiming of register r_2 across fanout stem w, giving rise to registers r_{21}, r_{22}; and 3) forward retiming of register r_3 across fanout stem v, creating registers r_{31}, r_{32}. These retimings are depicted in Fig. 5(b). To maintain the *initial state* of the retimed circuit we need to impose the following equivalence relations on the register variables: $r_{11} \overline{\oplus} r_{12},\ r_{12} \overline{\oplus} r_{13},\ r_{11} \overline{\oplus} r_{13},\ r_{21} \overline{\oplus} r_{22},\ r_{31} \overline{\oplus} r_{32}$. The *retiming-induced don't care* conditions (DC_{ret}), counterpart of these equivalence relations, resulting from such a series of retimings across fanout stems are the following: $r_{11} \oplus r_{12},\ r_{12} \oplus r_{13},\ r_{11} \oplus r_{13},\ r_{21} \oplus r_{22},\ r_{31} \oplus r_{32}$.

Now we can perform forward retiming across the logic block composed of g_2, g_3 by moving registers r_{12}, r_{22}, r_{32} from their inputs to the output of gate g_3. Fig. 5(c) shows the result of such a retiming, with new register r_4 placed at the output of gate g_3. Now the expression for R_4 can be simplified using DC_{ret} as follows:

$$R_4 = (r_{11} r_{21}) z_1 + r_{13} + DC_{ret} \quad (1)$$

$$R_4 = (r_{11} r_{21}) z_1 + r_{13} + r_{11} \oplus r_{13} \quad (2)$$

$$R_4 = r_{11} + r_{13} \quad (3)$$

The resynthesized circuit, depicted in Fig. 5(d), has a delay of just 1 gate, which is an optimum solution. A similar result can also be obtained by using the retiming-induced register equivalence relations. Note that without the don't care set, DC_{ret}, no simplification could have been achieved at all. The initial state computation for this register R_4 is straight-forward and is performed as explained in Section II.

The resulting circuit has 5 registers. We can now apply the constrained min-area retiming to minimize the number of registers in the circuit subject to the constraint that the delay of this circuit does not increase. Notice that registers r_{11} and r_{13} can be backward retimed (as they have the same initial state) without increasing the delay of the circuit. The resulting circuit is shown in Fig. 6, with registers r_{11} and r_{13} retimed backwards resulting into register r_1 with the same initial state.

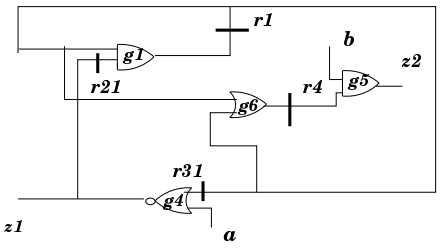

Fig. 6. Final circuit: Retiming to minimize registers under the same delay constraint.

IV. The Resynthesis Algorithm

The resynthesis algorithm is presented in Algorithm 1. Let us briefly discuss the transformations used in the proposed resynthesis procedure and justify the intuition behind them.

After the critical path is made fanout-free, we identify all the state registers that fan-out to the nodes on the critical path. These registers are retimed across their respective fanout stems. Those registers that do not fan-out to the nodes on the critical path are untouched. The retiming-induced don't care set DC_{ret} is computed (straightforwardly) and stored. The next step is retiming across logic gates (nodes) on the critical path. All the registers feeding the critical path are forward retimed as much as possible. For each node on the critical path, we identify whether or not it is forward-retimable. A node is forward-retimable if it contains only registers as its fanins. If the node is forward-retimable, forward retiming is performed across it. The *retiming engine* iteratively peforms forward retimings across the nodes on the critical path untill they cannot be retimed any further (note that unlike in [10], we do not borrow any registers from the environment). The worst case complexity of the retiming engine is $O(n^2)$, where n is the total number of nodes on the fanout-free critical path. The initial states of the forward retimed registers is computed as explained in Section II. After the retiming of registers across fanout stems and gates, the restructured (critical) path logic is optimized using the retiming-induced don't care set. Each node on the critical path is simplified using DC_{ret}. Local node re-mapping can then be performed. As a post-processing step, constrained min-area retiming is performed in order to minimize the number of registers subject to the constraint that the delay of the circuit does not increase. Other latch count minimization techniques, such as [27], can also be used.

From the above discussion, it becomes apparent that if the path in question does not have "retimable-gates," our technique would fail. Also, to exploit the retiming-induced don't care conditions, the circuit should have feedback loop(s). Because of these feedback loop(s), we are able to analyze the correlation of logic across register boundaries for simplification. This leads us to the conclusion that fully combinational I/O paths and pipelined circuits would not benefit from our technique.

Inputs: Sequential circuit, technology library.
Outputs: Resynthesized circuit.
Find the critical path of the circuit;
Make the critical path fanout-free by node duplication;
/* List all nodes on the critical path ordered from input to output */
critical_path_nodelist = list of nodes on the fanout-free critical path;
critical_path_fanout_registers = list of registers that fanout to the nodes on the *'critical_path_nodelist'*;
DC_ret_flag = FALSE;
for (each register in *critical_path_fanout_registers*) **do**
  if (register output == multiple fanout stem junction) **then**
    Retiming across fanout stems;
    Maintain same initial states;
    Evaluate and store DC_{ret};
    DC_ret_flag = TRUE;
  end if
end for
/* If no retimings across fanout stems, no DC_{ret} created */
/* So the ckt. cannot be resynthesized by our technique */
if (DC_ret_flag == FALSE) **then**
  return original network; /* resynthesis not possible */
end if
/* The Retiming engine for retiming across single output nodes */
forward_retiming_flag = TRUE;
while (forward_retiming_flag == TRUE) **do**
  forward_retiming_flag = FALSE;
  for (each node in *critical_path_nodelist*) **do**
    if (node is retimable) **then**
      Forward retime registers across this node;
      Compute initial value for the retimed register;
      Update the changes in fanins and fanouts of all the nodes in *critical_path_nodelist* due to this retiming;
      forward_retiming_flag = TRUE;
    end if
  end for
end while /* Not possible to retime any further */
Simplify the next state logic of the retimed register using DC_{ret};
Perform local node re-mapping;
Find the delay of the circuit;
/* constrained min-area retiming */
Retime to minimize registers under the same delay constraints;
Return the resynthesized circuit;

Algorithm 1: Resynthesis using retiming-induced don't cares.

V. Implementing the Algorithm: Results & Conclusions

The above mentioned resynthesis algorithm was programmed within the SIS [28] synthesis framework and its effectiveness was analyzed by applying it to the MCNC and ISCAS'91 benchmark set. Table I reports the overall results obtained by applying the algorithm on the benchmark circuits. The gate level netlists were optimized for performance using *script.delay*. The delay optimized netlists were mapped to produce minimum delay circuits, using the $lib2.genlib$ technology library. The performance/area statistics of the mapped circuits are provided (script.delay column). These mapped networks were then retimed to further minimize the cycle time (min-delay retiming). The external don't cares induced by retiming were extracted using implicit state enumeration, and were used to further simplify the logic. The network was remapped to produce a minimum delay circuit. These results are presented under the 'script.delay+ret.+comb .opt.' column.

On the other hand, the circuits obtained after delay optimization (script.delay+tech. mapping) were *resynthesized* using the proposed technique. The resynthesized circuits compare favourably not only with the ones obtained by just performance optimization, but also with the ones obtained by subsequent retiming. The circuits resynthesized by our technique are not only smaller, but also faster than the ones obtained by conventional retiming and resynthesis procedure.

Notice that after performance optimization (using script.delay and mapping), not all synthesized circuits could be retimed by SIS. For the benchmarks *ex6, bbtas, s420, s344, s382, s386, s400, s510, s1196, s1238*, and *s5378*, we found that retiming was either unable to minimize the cycle time, or was unable to preserve/compute the initial states. In contrast, our approach was able to resynthesize (retime across fanout stems and logic gates and simplify) the circuit and compute the initial values for the registers. In most cases our approach resulted in better circuit performance as compared to conventional retiming. The last few circuits, $s713$, $s953$ and $s386$, could not be resynthesized using our

TABLE I

EXPERIMENTAL RESULTS: APPLYING THE RESYNTHESIZE ALGORITHM.

	Script.delay			Script.delay+Retiming+Comb. Opt.			Script.delay + Resynthesis		
Circuit	Reg.	Clk.	Area	Reg.	Clk.	Area	Reg.	Clk.	Area
ex3	4	9.62	87696	18	8.45	152656	5	9.55	106720
ex6	3	11.76	182352	-	-	-	4	9.42	197664
bbtas	3	5.26	35728	-	-	-	4	4.41	46864
bbara	4	8.49	73312	11	7.14	105792	5	6.38	81164
planet	6	14.46	705744	16	14.28	752144	7	13.0	692288
s420	5	9.39	135488	-	-	-	6	10.85	155440
s510	6	11.30	342896	-	-	-	7	12.07	363776
s298	14	6.48	166576	24	6.34	212976	15	6.28	177120
s344	15	13.81	236640	-	-	-	16	12.35	264480
s349	15	14.46	232464	15	11.95	232464	16	11.58	257984
s1196	18	27.67	679760	-	-	-	19	26.00	753072
s1238	18	26.25	637072	-	-	-	19	20.46	874604
s382	21	9.73	289072	-	-	-	24	7.89	300672
s400	21	10.25	277008	-	-	-	24	7.21	277936
s526	21	9.95	291856	47	8.35	412496	31	7.74	342432
s9234	135	31.05	1757632	303	19.75	2537152	194	21.98	2322768
s5378	162	29.15	1911680	-	-	-	165	25.94	2155744
s386	21	10.2	142912	-	-	-	-	-	-
s713	14	14.35	263008	16	13.98	272368	-	-	-
s953	29	19.10	437552	42	17.68	446832	-	-	-

approach, as the critical paths did not contain any multiple-fanout registers that could be retimed across their fanout stems. The area increase in our approach results from duplication of certain nodes on the critical path. However, subsequent simplification of the underlying logic reduces the area overheads. In some cases, logic simplification using DC_{ret} may not be possible. This may result in an increase in the cycle-time of the circuit. For example, circuits $s420$ and $s510$, could not be simplified using DC_{ret}. Forward retiming increased the delay of the circuit, and lack of simplification resulted in a slightly larger cycle-time. We are investigating about how far should forward retiming be performed such that our technique can be stopped from doing any harm.

To conclude, this paper has presented a controlled retiming and resynthesis procedure that can be applied to improve upon the cycle-time performance of sequential circuits. Our procedure guides retiming across fanout stems and gates and exploits the register equivalence relations as don't cares in optimization. We do not have to perform intensive computations to evaluate these don't care conditions. The proposed approach strives to minimize the increase in the number of registers without sacrificing the cycle-time performance. In many cases, conventional retiming techniques are unable to minimize the cycle-time of the circuit any further. However, our approach restructures the circuit and then guides retiming to achieve a cycle-time reduction. Experimental results demonstrate a favourable area/performance trade-off as compared to conventional retiming and resynthesis techniques.

REFERENCES

[1] C.E. Leiserson, F.M. Rose, and J.B. Saxe, "Optimizing Synchronous Circuitry by Retiming", *in Third Caltech Conference on VLSI*, pp. 87–116, 1983.

[2] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, Inc., 1994.

[3] N. Shenoy and R. Rudell, "Efficient Implementation of Retiming", *in Proc. ICCAD*, 1994.

[4] K.N. Lalgudi and M.C. Papaefthymiou, "DelaY: An Efficient Tool for Retiming with Realistic Delay Modeling", *in Proc. DAC*, pp. 304–309, June 1995.

[5] N. Maheshwari and S. Sapatnekar, "Efficient Retiming of Large Circuits", *IEEE Trans. on VLSI Systems*, vol. 6, pp. 74–83, March 1998.

[6] A.P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabey, and R.W. Broderson, "Optimizing Power Using Transformations", *IEEE Trans. on CAD*, vol. 14, Jan. 1995.

[7] J. Monteiro, S. Devadas, and A. Ghosh, "Retiming Sequential Circuits for Low Power", *in ICCAD*, pp. 398–402, 1993.

[8] G.D. Hachtel, M. Hermida, A. Pardo, M. Poncino, and F. Somenzi, "Re-encoding Sequential Circuits to Reduce Power Dissipation", *in Proc. ICCAD*, pp. 70–73, 1994.

[9] S. Hassoun and C. Ebeling, "Architectural Retiming: Pipelining Latency Constrained Circuits", *in Proc. DAC*, June 1996.

[10] S. Malik, E. Sentovich, R. Brayton, and A. Sangiovanni-Vincentelli, "Retiming and Resynthesis: Optimizing Sequential Networks with Combinational Techniques", *IEEE Trans. on CAD*, vol. 10, pp. 74–84, Jan. 1991.

[11] G. De Micheli, "Synchronous Logic Synthesis: Algorithms for Cycle-Time Optimization", *IEEE Trans. on CAD*, vol. 10, pp. 63–73, Jan. 1991.

[12] B. Lin, "Restructuring of Synchronous Logic Circuits", *in Proc. EDAC*, pp. 205–209, 1993.

[13] M. Potkonjak, S. Dey, Z. Iqbal, and A. Parker, "High Performance Embedded System Optimization using Algebraic and Generalized Retiming Techniques", *in Proc. ICCD*, pp. 498–504, 1993.

[14] S. Bommu, M. Ciesielski, N. O'Neill, and P. Kalla, "Sequential Logic Optimization with Implicit Retiming and Resynthesis", *in Proc. Intl. Conf. VLSI'97*, pp. 327–338, 1997.

[15] V. Singhal, C. Pixley, R. L. Rudell, and R.K. Brayton, "The Validity of Retiming Sequential Circuits", *in Proc. Design Automation Conference*, 1995.

[16] A. El-Maleh, T. Marchock, J. Rajski, and W. Maly, "Behavior and Testability Preservation under Retiming Transformation", *IEEE Trans. on CAD*, vol. 16, pp. 528–543, May 1997.

[17] H.J. Touati and R.K. Brayton, "Computing the Initial States of Retimed Circuits", *IEEE Tr. on CAD*, vol. 12, pp. 157–162, Jan. 1993.

[18] C.E. Leiserson and J.B. Saxe, "Retiming Synchronous Circuitry", *Algorithmica*, vol. 6, pp. 5–35, Jan. 1991.

[19] V. Singhal, S. Malik, and R.K. Brayton, "The Case for Retiming with Explicit Reset Circuitry", *in Proc. ICCAD*, pp. 618–625, 1996.

[20] K. A. Barlett *et al.*, "Multilevel Logic Minimization using Implicit Don't Cares", *in IEEE Trans. on CAD*, vol. 7, pp. 723–729, June 1988.

[21] M. Damiani and G. De Micheli, "Don't Care Set Specification in Combinational and Synchronous Circuits", *IEEE Trans. on CAD*, vol. 12, pp. 365–388, March 1993.

[22] M. Damiani and G. De Micheli, "Recurrence Equations and the Optimization of Synchronous Circuits", *Proc. Design Automation Conference*, pp. 556–561, 1992.

[23] O. Coudert and J.C. Madre, "A Unified Framework for the Formal Verification of Sequential Circuits", *in Proc. ICCAD*, pp. 126–129, 1990.

[24] H.J. Touati, H. Savoj, B. Lin, R.K. Brayton, and A. Sangiovanni-Vincentelli, "Implicit State Enumeration of Finite State Machines using BDDs", *in Proc. ICCAD*, pp. 130–133, 1990.

[25] B. Lin, H. Touati, and A. R. Newton, "Don't Care Minimization of Multilevel Sequential Logic Networks", *in Proc. ICCAD*, pp. 414–417, 1990.

[26] H. Savoj and R.K. Brayton, "The use of Observability and External Don't Cares for the Simplification of Multiple-Level Networks", *in Proc. DAC'90*, 1990.

[27] E.M. Sentovich, H. Toma, and G. Berry, "Latch Optimization in Circuits Generated from High-level Descriptions", *in Proc. ICCAD*, pp. 428–435, Nov. 1996.

[28] E. Sentovich *et al.*, "SIS: A System for Sequential Circuit Synthesis", Technical Report UCB/ERL M92/41, ERL, Dept. of EECS, Univ. of California, Berkeley., 1992.