Path Reduction with Register Repositioning *

Spyros Tragoudas
Electrical Engineering Dept.
Southern Illinois University
Carbondale, IL 62901

Nathan Denny
Electrical and Computer Engr. Dept.
University of Arizona
Tucson, AZ 85721

Abstract Circuits with large numbers of paths present difficulties for path delay fault testing and timing analysis. By repositioning registers we can in many cases substantially reduce the number of paths in the circuit. The proposed method applies on an acyclic representation of the circuit after some of its edges are removed. We present an integer linear program for repositioning registers such that they minimize the number of paths in acyclic circuits with at most one register on each I/O path. In the more general case of sequentially deep circuits, we first decompose the circuit into sequentially shallow acyclic subcircuits, then apply our method on the subcircuits. Our results show that in many cases local optimization for path reduction often results in a global reduction as well.

1 Introduction

The number of paths in a circuit are of particular interest to the VLSI design community. In design for testability, the number of potential delay faults in the circuit is directly proportional to the number of I/O paths through the circuit. Controlling the number of subpaths in a circuit is thus of primary concern for testability. Timing analysis, which often relies on enumerative path techniques, can also benefit from a reduction in the number of circuit paths [1].

Recently, [2, 3, 4] have proposed methods for reducing the number of paths in an acyclic circuit. [2] uses logic synthesis techniques and [3, 4] propose to insert test points. All these approaches consider that the input is a combinational circuit.

This paper proposes a method for reducing the number of paths in sequential circuits. We assume, however, that at a preprocessing step the circuit is transformed to an acyclic one. The cycles are eliminated by removing edges.[1] We present an integer linear program (ILP) for minimizing the number of paths in a circuit by repositioning registers. Our method may be applied in conjunction with the abovementioned path reduction techniques for combinational circuits. For example, these approaches can apply as a preprocessing or postprocessing step of our method.

Our ILP formulation is a special case that is solvable in polynomial time by a reduction to the minimum cost network flow problem. We will demonstrate in the next section that our ILP results are provably optimal for circuits with unit sequential depth, i.e. where each I/O path has at most one register. Furthermore, we will give a heuristic application of our ILP to the more general class of circuits with unlimited sequential depth.

The paper is organized as follows. Section 2 describes the algoriths that form the basis of the proposed method. Section 3 gives experimental results and Section 4 concludes.

2 Algorithms

Leiserson and Saxe [5] introduced the use of integer linear programming for minimizing the total cost of register placement in a circuit through the operation of retiming. In their formulation, each possible edge had an associated placement cost. We have used this as the basis for our formulation of an integer linear program for minimizing the number of paths in circuits. In general, ILP is a difficult problem. However, in the specific case presented here and in [5] the ILP problem reduces to a minimum cost network flow problem. Several polynomial time algorithms are known for solving minimum cost flow networks in approximately $O(n^3)$.

*Partially supported by NSF grant CCR-9815229

[1] This is a virtual cycle breaking process, i.e., the removed edges will be reinserted at the end of the approach.

2.1 Shallow acyclic circuits

We proceed with our ILP formulation by first modelling the acyclic circuit with unit sequential depth as the directed acyclic graph $G = (V, E)$ where V is the set of logic blocks or gates and E is the set of interconnecting edges. Registers are not modelled as gates, but as an attribute associated to each edge. Let $r(e)$ be the number of registers on the edge, $e \in E$. The circuit graph is then translated into a set of inequalities that represent the initial circuit conditions and constrain the ILP solution to the space of legal retimings.

The ILP will assign to each node a shift value, $s(v) \in \{-1, 0, 1\}$, $v \in V$. The shift of a node reflects the movement of registers relative to the initial conditions of the circuit. For some node, v, $s(v) = -1$ if all registers on the inputs of v have been shifted to the outputs of v, $s(v) = 0$ if v has not participated in any shift operations, and $s(v) = 1$ if all registers on the outputs of v have been shifted to the inputs of v. To prevent the ILP from shifting registers before the primary inputs or after the primary outputs, we assign the static value $s(v) = 0$ for $v \in \{primary\ inputs\} \bigcup \{primary\ outputs\}$.

For all edges, we can use the initial register distribution, and the shift values of the nodes to compute the final register distribution after retiming.

$$r'(e) = r(e) + s(v) - s(u)$$

where e is the edge (u, v), $r'(e)$ is the number of registers on e after retiming, $r(e)$ is the number of registers on e before retiming, and $s(v)$ and $s(u)$ are the shift values of the nodes. For the retiming to be physically realizable, the ILP must not assign shift values such that some $r'(e)$ is negative. Thus, for each edge, $e = (u, v)$, we add the constraint

$$s(u) - s(v) \leq r(e) \qquad (1)$$

Having defined the set of constraints we now state a theorem.

Theorem 1: The solution of the integer linear program applied to circuits with unit sequential depth, yields an optimal register placement for the minimal number of paths in the circuit.

Proof: The ILP constraints limit the solution space to legal retimings. Thus to prove our theorem, we need only to concentrate on the condition of the optimality of the solution. Here we rely on the independent nature of each register in the circuit. Since the sequential depth of the circuit is limited to unity, no I/O path can contain more than one register. Furthermore, through the operation of retiming, the number of registers on an I/O path is constant. This allows us to assign to each edge a benefit function that is independent of the positioning of the registers. The benefit of each edge is the number of paths that the edge divides. An efficient method for counting paths using a topological sort was given in [3]. Let $P_{in}(e)$ be the number of paths from the primary inputs to the edge, e. Similarly, let $P_{out}(e)$ be the number of paths from the e to the primary outputs. Our benefit function, $b(e)$, for the edge, e, is

$$b(e) = P_{in}(e) \cdot P_{out}(e) - P_{in}(e) - P_{out}(e)$$

For each node, $v \in V$, we can compute the benefit of v participating in a shift operation by

$$B(v) = \sum_{e \in inedges} b(e) - \sum_{e \in outedges} b(e)$$

Due to the independent properties of the register placement, the total reduction in the number of paths in the circuit is a linear combination of the effects of each register in isolation. Thus our objective function is

$$\max \sum_{v} B(v) \cdot s(v) \qquad (2)$$

The ILP is defined by Equations (1) and (2). •

This ILP can be solved using minimum cost flow techniques, following arguments as in [5].

2.2 Generalization

When operating on circuits that are sequentially deep, we no longer have the luxury of independent registers. Moving a register in a sequentially deep circuit has an immediate effect on all other registers in the fanin and fanout cones of the register. Our ILP is therefore limited to optimality only on sequentially shallow circuits.

We define the timing of a node, $timing(v)$, as the maximum number of registers on any path from a primary input to v. We define the n^{th} frame, f_n of the circuit graph, G, as the subcircuit induced by $v \in V$ such that $timing(v) \in \{n, n+1\}$. In implementation, we also add another class of nodes to model the effects at the boundary of the frame. For $timing(v) = n$ we add a border node, v', and an edge, (v', v) if all inbound arcs to v do not originate from within f_n. Similarly, for $timing(v) = n+1$ we add a border node, v', and an edge, (v, v'), if all outbound arcs of v do not terminate within f_n.

To apply the ILP to sequentially deep circuits, we decompose the circuit into frames and apply our ILP to the set of frames in some sequential ordering. After retiming each frame, we measure its benefit when the frame is merged back into the circuit. If the retimed frame leads to a decrease in the number of paths in the circuit, then we keep the results and retime the next frame in the sequence. If the number of paths in the circuit does not decrease, then we ignore the retimed frame and continue to the next frame.

3 Experimental results

We tested our method on the standard ISCAS89 benchmark set. For each circuit, we generated an acyclic graph of the circuit by removing feedback edges discovered by a simple depth first search. A more intelligent method of cycle cutting such as the methods proposed in [6] and [7] could lead to better performance. As a rule for generating the frame sequence, we counted the number of paths internal to each frame. The frame sequence begins with the frame with the least internal paths and proceeds in an increasing order to the frame with the most internal paths.

For the moderately sized circuits in the ISCAS89 benchmark set, reducing our ILP to a minimum cost network flow was not necessary. The freely available LP_SOLVE package was sufficient to find solutions for all benchmarks within reasonable time. [2]

For most cases of the ISCAS89 benchmark, our heuristic retiming algorithm resulted in some reduction in the number of subpaths. We observed no reduction in s208.1, s298, s344, s420, s510, s526, s526n, s820, s832, s1423, s1488, s1494, and s35932. We present our results in Tables 1 and 2.

[2] The LP_SOLVE package is available via FTP at ftp.es.ele.tue.nl in directory /pub/lp_solve

Table 1. Before and after

	Before		After	
Circuit	Subpaths	State	Subpaths	State
s349	365	15	347	25
s382	400	21	355	24
s444	535	21	507	26
s641	1,744	19	355	19
s713	21,812	19	1,838	19
s953	1,156	29	1,036	59
s1196	3,098	3	3,096	3
s1238	3,559	3	3,557	3
s5378	13,523	164	11,708	218
s9234	244,854	228	200,215	298
s9234.1	244,854	211	163,207	257
s13207	1,345,369	669	137,559	747
s13207.1	1,345,369	638	259,015	761
s15850	164,738,046	597	79,855,156	648
s15850.1	164,738,046	534	2,205,768	601
s38417	1,391,579	1,636	878,399	1,829
s38584	1,080,723	1,452	373,778	1,533
s38584.1	1,080,723	1,426	589,215	1,448

Table 1 reports the absolute number of subpaths and flip-flops before and after the retiming.

Table 2. Summary of changes

Circuit	Subpaths (%)	State (%)	CPU (secs)
s349	- 4.93	+ 66.67	3.03
s382	-11.25	+ 14.29	3.14
s444	- 5.23	+ 23.81	3.44
s641	-79.64	+ 0.00	3.81
s713	-91.57	+ 0.00	4.23
s953	-10.38	+103.44	5.44
s1196	- 0.06	+ 0.00	4.97
s1238	- 0.06	+ 0.00	4.73
s5378	-13.42	+ 32.93	35.05
s9234	-18.23	+ 30.70	64.89
s9234.1	-33.35	+ 21.80	65.65
s13207	-89.78	+ 11.66	95.12
s13207.1	-80.75	+ 19.27	93.98
s15850	-51.53	+ 8.54	118.21
s15850.1	-98.66	+ 12.55	113.16
s38417	-36.88	+ 11.80	342.74
s38584	-65.41	+ 5.58	307.54
s38584.1	-45.48	+ 1.54	366.45

In Table 2 we display the relative decrease in the number of subpaths and the relative increase in the number of flip-flops. In most cases, when a reduction is observed, the reduction is quite signifigant. Furthermore only the relatively small circuits of s349 and s953 exhibit a state increase of more than 33%. The computation time reported for each circuit in the CPU column is the total time required by the ILP solver to find solutions for all frames in the circuit. All computations were executed on a Sun ULTRA 5 workstation with 128MB of physical RAM.

Table 3. Estimated timing impact

Circuit	Before	After
s349	30	30
s382	16	16
s444	19	19
s641	94	46
s713	102	50
s953	28	27
s1196	36	36
s1238	34	34
s5378	34	34
s9234	70	62
s9234.1	70	76
s13207	76	82
s13207.1	76	86
s15850	108	91
s15850.1	108	95
s38417	64	60
s38584	75	85
s38584.1	75	85

When repositioning flip-flops within in a circuit, the timing characteristics of the circuit can change signifigantly. For completeness we have included an estimate of the impact on the timing of each circuit that was modified by our method. The longest path in the original model and the retimed model was used as the metric for comparison. In many cases, the timing of the circuit is actually improved and in those cases where the timing is increased, the performance penalty is modest.

We note there that the set of inequalities in (1) can be enhanced appropriately in order to retime the circuit so that we maintain an upper bound on the clock period. (See [5] for details.) Such modifications may be appropriate when our tool increases the clock period significantly.

4 Conclusions

We have presented a provably optimal method for reducing the number of paths in an acyclic circuit with unit sequential depth by the operation of retiming. Our retiming method uses an ILP that can be formulated as a minimum cost flow network. We have used the ILP as a heuristic for reducing the number of subpaths in circuits that have unlimited sequential depth.

Our experimental results show that our method is computationally inexpensive and quite effective in reducing the number of subpaths in a circuit with only a modest increase in hardware. Furthermore, our method has a minimal if not beneficial impact on the timing performance of the circuit.

References

[1] W. Wolfe, Modern VLSI Design: Systerms on Silicon, 2nd Edition, Prentice Hall, 1998.

[2] A. Krstic, K. Cheng, "Resynthesis of Combinational Circuits for Path Count Reduction and for Path Delay Fault", Journal of Electronic Testing: Theory and Applications vol 11, pp. 43–54, 1997

[3] I. Pomeranz, S. Reddy, "Design for Testability for Path Delay Faults in Large Combinational Circuits Using Test Points", IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol. 17, no. 4, pp.333–343, April 1998.

[4] S. Tragoudas, N. Denny, "Testing for Path Delay Faults using Test Points", to appear in the Proceedings of the IEEE International Symposium on Defect and Fault Tolerant Computing, Nov. 1999.

[5] C. Leiserson, J. Saxe, "Retiming Synchronous Circuitry", Algorithmica, vol 6, pp. 5–35, 1991

[6] H. Levy, D.W. Low, "A Contraction Algorithm for Finding Small Cycle Cutsets", Journal of Algorithms, vol 9, pp. 470–493, 1988

[7] D. Kagaris, S. Tragoudas "Retiming Based on Partial Scan", IEEE Trans. on Computers, vol 45, no. 1, pp. 74–87, 1996