Performance Optimization using Exact Sensitization

Alexander Saldanha* Heather Harkness† Patrick C. McGeer‡

Robert K. Brayton Alberto L. Sangiovanni-Vincentelli

University of California - Berkeley CA

Abstract

A common approach to performance optimization of circuits focuses on re-synthesis to reduce the length of *all paths* greater than the desired delay τ. We describe a new delay optimization procedure that optimizes *only sensitizable paths* greater than τ. Unlike previous methods that use topological analysis only, this method accounts for both functional and topological interactions in the circuit. Comprehensive experimental results comparing the proposed technique to a state-of-the-art performance optimization procedure are presented for combinational and sequential logic circuits.

1 Introduction

A common technique for performance optimization of logic circuits is re-synthesis to reduce the length of statically long paths. Optimization methods which apply this technique assume that the delay of the circuit is equivalent to the delay of the topologically longest path, and use path lengths to identify critical paths [8], or as a rough measure of delay optimization [11].These techniques are topology dependent and function independent, since no consideration is made for false paths. However, in many circuits the topologically long paths are false paths, which do not contribute to the actual delay. For such circuits, the methods which rely on topological path length may waste unnecessary resources optimizing false paths, or may fail to achieve the maximum performance optimization.

More recent performance optimization work utilizes the fact that the actual delay of the circuit is not equivalent to the longest topological delay when the longest paths in a circuit are *false*. The simplest methods [6, 2] just remove long false paths so that the longest topological path in the resulting circuit is equal to the actual delay of the circuit.

A more aggressive method, the generalized bypass transform [4], accelerates circuits by introducing redundancy which makes long paths false. This method relies on a generalization of the carry-bypass adder optimization technique to perform local optimizations. It bypasses long paths by adding local redundancy to the network. The algorithm of [6] is then used to remove the long false paths. While this method relies on local sensitization to optimize paths, it uses topological information to select paths for optimization and doesn't consider global sensitization information.

The method of [3] considers path sensitization as one criterion in the selection of paths for delay optimization. This method considers only gate resizing and buffer insertion as delay optimization techniques, leaving the topology of the circuit unchanged.

In this paper we propose a new delay optimization technique that relies on recent advances in timing analysis and optimization to speed up only sensitizable paths. A step beyond path-based procedures, it is a functional procedure, based on recent sensitization work[5]. In this work, delays are associated not with paths but with specific input vectors or vector sequences. Thus, with each range of circuit delays, there is an associated boolean function representing the sensitizable paths which terminate within that delay range. This function is extracted for the range of delays $[\tau, \infty]$ where τ is the desired circuit delay, and optimized using any delay reduction technique. The result is then combined with the original circuit and the whole circuit passed through the false path elimination procedure [6]. The approach may be extended to consider multi-cycle false paths in sequential circuits, and it combines multi-cycle false path elimination and retiming to exploit the advantages demonstrated by the approach of [1].

This functional timing optimization technique provides significant advantages over the existing false path based optimization methods. It is the first approach to consider exact sensitization conditions in selecting paths for performance optimization. This global circuit information is used to derive a more topology-independent function for optimization than that produced by the existing methods. This produces logic which is potentially more optimal in terms of both area and performance. Also, rather than operating on a path-by-path basis, which might be impractical, the functional method considers the entire set of long paths simultaneously.

This paper first demonstrates how the proposed optimization technique is applied to combinational logic circuits and how don't cares can be exploited in performance optimization of the sensitization functions. Next, the technique is extended to sequential logic circuits where unreachable states are used as don't cares. We also demonstrate how information related to multi-cycle false paths may be utilized in performance optimization. Comprehensive experimental results comparing the proposed technique to a state-of-the-art performance optimization procedure are presented.

2 Background

2.1 Delay model

Every optimization technique makes use of timing information contained in some delay model. If the circuit is built from library gates, and if layout information is available for the extraction of routing delays, then exact delay information is a reasonable model. When layout and gate delay information are not known precisely, weaker models are preferred. The general model is the unit-fanout model, where each gate is thought of as having a unit delay, and where each fanout is thought of as contributing a smaller delay. This model is intended to capture wiring effects.

Touati *et al.* [11] argue persuasively that layout and mapping factors such as transistor sizing, buffering, and routing tend to dwarf pure fanout effects, and that as a result technology-independent optimization should concentrate on reduction of circuit levels; this is the *unit* delay model where each gate is

* Cadence Berkeley Labs - Berkeley CA

† Digital Equipment Corporation - Hudson MA

‡ Cadence Berkeley Labs - Berkeley CA

thought of as having a unit delay and there are no fanout effects. The effectiveness of the unit delay model over two-input gates has been experimentally confirmed by Singh [8], who shows that the final delay of the mapped circuit is reasonably predicted by the unit delay of the two-input NAND network.

Although our experimental results are restricted to the simple unit delay model, the proposed method applies to circuits with more realistic delay models. We are limited only by the resynthesis procedures used for performance optimization, which are less predictable and significantly slower when technology mapped delays are used [8].

2.2 Exact sensitization

Recently a ternary waveform algebra framework was provided in which the exact sensitization conditions under several common delay models are easily described [5]. Here we adapt the exact sensitization conditions of [5] to "floating delay" mode computations only.

Definition 2.1 *A* **characteristic function** *is a mapping:*

$$\chi : B^n \mapsto \{0, 1\}.$$

χ is associated with some set $S \subseteq B^n$: $\chi(w) = 1$ iff $w \in S$

Characteristic functions are a feature of the timing verification algorithms developed in [5]. They are used to represent functions of the form:

$$\chi^{g(t)=v} \equiv \{w |\ g \text{ settles to stable value } v \text{ before time } t \text{ under } w\}$$

Here g is a gate, $v \in \{0, 1\}$ and $\chi^{g(t)=v}$ is the set of all input vectors such that the output of g is constant at v on the interval (t, ∞). Since $\chi^{g(t)=1}$ contains all the vectors under which g settles to a stable value of 1 before t, the set $g.\overline{\chi^{g(t)=1}}$ contains all the vectors that cause g to settle to a final value of 1 after t. This set is denoted as $\mathcal{R}^{g(t)}$. Similarly, the set $\overline{g}.\overline{\chi^{g(t)=0}}$ contains all the vectors that cause g to settle to a final value of 0 after t, and is denoted as $\mathcal{F}^{g(t)}$. Thus, we have the complementary characteristic functions:

$$\mathcal{R}^{g(t)} \equiv \{w |\ g \text{ settles to stable value 1 after time } t \text{ under } w\}$$

and

$$\mathcal{F}^{g(t)} \equiv \{w |\ g \text{ settles to stable value 0 after time } t \text{ under } w\}$$

$\mathcal{R}^{g(t)}$ ($\mathcal{F}^{g(t)}$) is the set of all input vectors for which a rising (falling) sensitizable path of length $> t$ exists up to the output of g. It follows from their definitions that $\mathcal{R}^{g(t)} \subseteq g$ and $\mathcal{F}^{g(t)} \subseteq \overline{g}$ for any gate g and time t.

In [5], an efficient recursive form of the $\mathcal{R}^{g(t)}$ or $\mathcal{F}^{g(t)}$ sensitization function is introduced. The expression for $\chi^{g(t)=v}$ is written so that it depends only on the function g and the sensitization functions of the fanin of g.

Lemma 2.1 *Let g be a gate with inputs $f_1, ..., f_r$. Let $p_1, ..., p_n$ be all the primes of g, and $q_1, ..., q_m$ all the primes of $\overline{g}$. Let $F_k(p)$ denote the value of input f_k in a prime p. Then:*

$$\chi^{g(t)=1} = \sum_{i=1}^{n} \prod_{k=1}^{r} \left[\begin{array}{l} \{(F_k(p_i) = 1) \Rightarrow \chi^{f_k(t-D^k)=1}\} \\ \{(F_k(p_i) = 0) \Rightarrow \chi^{f_k(t-D^k)=0}\} \end{array} \right]$$

$$\chi^{g(t)=0} = \sum_{j=1}^{m} \prod_{k=1}^{r} \left[\begin{array}{l} \{(F_k(q_j) = 1) \Rightarrow \chi^{f_k(t-D^k)=1}\} \\ \{(F_k(q_j) = 0) \Rightarrow \chi^{f_k(t-D^k)=0}\} \end{array} \right]$$

Since we are working with the technology independent delay model of two-input NAND gates, an attractive side-effect of the use of these recursive functions is that they retain the structure of the original network as closely as possible. In fact, for our results using the exact sensitization formulation, the depth of the recursive function at a node is exactly equal to the depth of that node in the original network. This is easy to observe in the recursive formula of Lemma 2.1. For computing the rising delay, since a NAND gate has two primes, a single two-input NAND gate captures the outer summation term. Since the primes each have a single literal, a gate is not needed to represent the inner product term. Analogous reasoning holds for the falling delay sensitization function. Not surprisingly, the delay of the recursive functions $\chi^{g(t)=1}$ and $\chi^{g(t)=0}$ in Lemma 2.1 is at most the length of the longest sensitizable path up to g in the original network. The function $\mathcal{R}^{g(t)}$ is formed as the product of $\overline{\chi^{g(t)=1}}$ and the original network, so the delay of $\mathcal{R}^{g(t)}$ is most one greater than the longest sensitizable path delay of function g. Similar reasoning indicates that the delay of $\mathcal{F}^{g(t)}$ is at most one greater than the delay of g. Even in the case of technology dependent delays, the delay of the recursive functions can be expected to closely match those of the original network.

3 Combinational performance optimization

Given a combinational logic circuit and a target time of $\tau + 1$, the exact sensitization optimization procedure, *sense_opt*, operates as follows. The rising and falling sensitization functions, $\mathcal{R}^{g(\tau)}$ and $\mathcal{F}^{g(\tau)}$, for each primary output g are created using the recursive formulation of Lemma 2.1. If the delay of these two functions can be optimized $\leq \tau$, then by composing the circuits as shown in Figure 1, the longest sensitizable path in the final network, denoted *g_fast*, is guaranteed to be no more than τ plus the delay the two 2-input gates used to combine the sensitization functions with the original function.

The proof that the delay of *g_fast* $\leq \tau + 2$ follows from sensitization theory. We provide only a brief sketch of the proof here. Consider the input vectors under which primary output g settles to a final value of 1. These are the only vectors that impact the rising delay at g. There are two cases:

1. If g settles to 1 before time τ under input vector v in the original circuit, then the OR gate in *g_fast* of Figure 1 settles to 1 before time $\tau + 1$. Since the fall sensitization network is zero under this vector, the AND gate settles to 1 by time $\tau + 2$.

2. If g settles to 1 after time τ under vector v in the original circuit, then v is included in $\mathcal{R}^{g(\tau)}$; *i.e.* the output of the network $\mathcal{R}^{g(\tau)}$ is 1 under v. If the delay of $\mathcal{R}^{g(\tau)}$ is less than τ, then the OR gate in *g_fast* of Figure 1 settles to 1 before time $\tau + 1$. Hence, all paths of rising delay $> \tau$ are false under v in the original network for g of Figure 1.

An analogous argument is valid for the falling delay. Since the delay of *g_fast* is $\leq \tau + 2$, all paths of length $> \tau$ in the original circuit can be removed using the false path elimination algorithm described in [6].

Under the assumption that the rise and fall delay of a connection are identical, only one of the sensitization functions need be used. For example, if $\mathcal{R}^{g(t)}$ has delay $\leq \tau$, then all paths of length $> \tau$ are false in the original circuit, after the composition shown in Figure 2. Removal of these long false paths also eliminates all paths with falling delay $> \tau$, so the final composed circuit has delay $\leq \tau + 1$. Since we have employed the unit delay model, the requirement of equal rise and

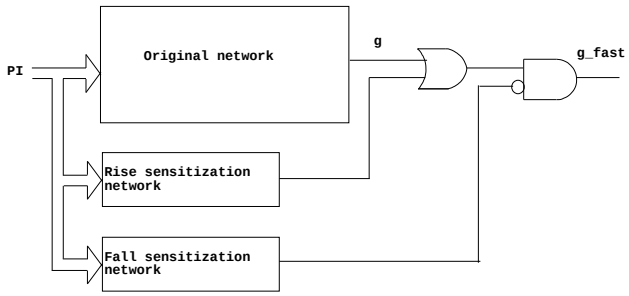

Figure 1: Speed up of circuit using sensitization functions

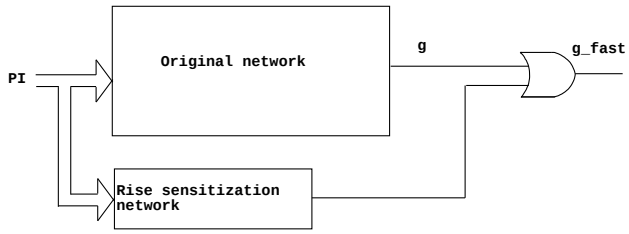

Figure 2: Speed up of circuit using only rise sensitization function

fall times is trivially satisfied. In the case of different rise and fall delays on a connection (*e.g.* technology mapped circuits), the maximum of these two quantities may be used to avoid the creation of both sensitization functions.

Don't care conditions which arise from the relationship of the sensitization functions to the original functions may be exploited in the simplification of the rise and fall sensitization functions. One can see that $g.\overline{\mathcal{R}^{g(\tau)}}$ is the observability don't care set for the function $\mathcal{R}^{g(\tau)}$. Similarly, $\overline{g}.\overline{\mathcal{F}^{g(\tau)}}$ is the don't care set for $\mathcal{F}^{g(\tau)}$. The sensitization functions may be optimized using an algorithm such as the *full_simplify* command available in SIS [7], which uses satisfiability, observability and external don't cares in logic minimization. For our experiments, however, simplification is performed using a redundancy removal algorithm that considers external don't cares.

Delay optimization of each sensitization function may be achieved using any combinational delay reduction technique. For this purpose, we use the latest *speed_up* procedure described in [8], which is shown to include or subsume most of the well known topology based performance optimization methods. This improved *speed_up* is an iterative delay optimization method, which applies a number of different local delay optimization techniques to produce a global decrease in delay. Among the local delay optimizations considered are timing driven simplification, decomposition, cofactoring, the generalized bypass, and the function complement. The method divides a network into small sub-networks of fixed size, and evaluates various delay optimization operations on each to determine maximum performance improvement, and associated area increase. This data is used to determine a lower bound on the attainable delay reduction. The possible delay reduction then determines which regions of the circuit should be optimized and which local optimization methods should be performed to meet the delay requirement at each output with minimal area cost. Note that the local transformations performed by *speed_up* rely on static delay analysis. Therefore, *sense_opt* is invoked recursively on the sensitization functions to ensure that false paths in these function representations are not considered in performance optimization.

```
/* τ+1 is the target delay */
sense_opt {
    Create R^{g(τ)}, F^{g(τ)}
    DC_rise = g.¬R^{g(τ)}
    DC_fall = ḡ.¬F^{g(τ)}
    RISE_min = R^{g(τ)} simplified wrt DC_rise
    FALL_min = F^{g(τ)} simplified wrt DC_fall
    RISE_opt = speed_up(RISE_min, τ)
    FALL_opt = speed_up(FALL_min, τ)
    g_fast = Either g+RISE_opt or g.¬FALL_opt
    g_fast = KMS(g_fast, > τ+1)
    Recover area by redundancy removal and
       resubstitution
}
Longest path in g_fast has delay ≤ τ+1
```

Figure 3: Overview of proposed delay optimization procedure

After successful delay optimization of the sensitization functions, and recombination with the original circuit, a single pass implementation of the KMS algorithm [6] is performed on the resulting circuit to eliminate the false paths without increasing delay. Once the false paths have been removed, the static delay reflects the actual delay of the circuit. A final step of area recovery by redundancy removal and resubstitution is then performed. The basic steps of the algorithm are shown in Figure 3.

4 Sequential performance optimization

The first extension of this method to sequential circuits exploits unreachable states for simplification of the sensitization functions. The set of reachable states is computed using an implicit breadth-first algorithm. The complement of this set is added to the don't care set used for minimization of the sensitization functions.

An **n-cycle path** $\mathcal{Q} = P_1P_2...P_n$ is composed of paths P_i in the combinational logic such that the output of path P_i is the input of path P_{i+1} for $1 \leq i < n$. Each P_i is termed a component path. Timing analysis of multi-cycle paths is described in [1]. An algorithm for removal of paths in the combinational logic which are part of long multi-cycle false paths is also provided in [1].

A second extension of *sense_opt* to sequential circuits exploits retiming to reduce the clock period. Assume that the combinational delay cannot be reduced below τ. However, if every n-cycle path is of length $\leq n\tau$, then retiming may be attempted to reduce the clock period. (Note that retiming cannot guarantee a final clock period less than τ since the initial state may not be preserved on retiming [10].) The main point to note is that each combinational path does not need to have delay $\leq \tau$ to ensure a n-cycle path delay $\leq n\tau$. The problem now is to distribute appropriate required times on the outputs of the combinational logic such that satisfaction of these constraints results in multi-cycle paths of at most the target delay. Several heuristics to perform the distribution of delay reduction to each of the combinational paths over multi-cycles are described in [8]. We have not incorporated these heuristics into *sense_opt*; instead we simply reduce the delay of the sensitization functions maximally. This is in contrast to satisfying the maximum required time of τ in the combinational case.

A third extension exploits false paths over multiple cycles. It has been demonstrated that a path P that is sensitizable over a single time frame (*i.e.* in the combinational logic) may be a component path of an n-cycle path $\mathcal{Q}$ which is false [1]. This

information can be utilized by *sense_opt* in further improving the performance optimization procedure. Assume that we are interested in reducing the length of n-cycle paths to $< n\tau$ to enable re-timing to reduce the clock period. Consider a path P in the combinational logic. If every n-cycle path $\mathcal{Q}$ of length $> n\tau$, of which P is a component path, is false, then P need not be considered in performance optimization[1]. This condition is easily captured as a don't care of the sensitization function. Given a target n-cycle delay of $n\tau$, we perform timing analysis over n-cycle paths to determine the set of vectors in the first time frame which can sensitize paths $> n\tau$. The vectors in $\mathcal{R}^{g(\tau)}$ and $\mathcal{F}^{g(\tau)}$ that are not included in this set are don't cares, since they do not sensitize n-cycle paths of length $> n\tau$. Following the performance optimization, elimination of all n-cycle paths of length $> n\tau$ ensures that the longest path in the resulting circuit over n cycles is $\leq n\tau$.

This procedure includes the technique of [1]. Like that approach, if all n-cycle paths of length $> n\tau$ are false, *sense_opt* returns a circuit with n-cycle delay $\leq n\tau$. However, unlike the technique of [1], which cannot reduce the n-cycle delay less than $n\tau$ if at least one path of delay $> n\tau$ exists, our procedure can utilize don't care conditions for only those n-cycle paths which are false in the simplification of the sensitization functions for each path. This potentially aids in reducing all n-cycle paths to being either false or of delay $\leq n\tau$.

5 Results

In this section we detail the experimental procedure and results obtained for combinational and sequential logic circuits. In both cases we have compared *sense_opt* to the latest version of the performance optimization procedure *speed_up* described in [8]. Comprehensive experimental results in [8] have shown it to be the state-of-the-art optimization procedure. The results obtained using the latest version of *speed_up* are significantly better than those originally presented in [9]. The results are also better than those reported in [4, 11].

5.1 Combinational logic circuits

Table 1 shows results produced by *sense_opt* and *speed_up* on benchmark combinational logic circuits. Each circuit is first optimized for area using the standard script (*script.rugged*) in SIS [7]. The circuit is next decomposed into 2-input NAND gates using an area and delay driven decomposition (*good_decomp; eliminate -1; speed_up -i*). We do not use the area driven decomposition (*good_decomp; tech_decomp -a 2*) since this provides a dis-advantageous starting point for *speed_up* [8]. A final single stuck fault redundancy removal is performed to ensure that the initial circuit is minimal.

The table shows the area and delay results produced by each optimization method. Here, area is the number of literals and delay is the longest sensitizable path delay under the unit delay model. For initial circuits, the longest topological path delay is also shown in parentheses where this delay differs from the longest sensitizable delay. For the *sense_opt* and *speed_up* optimized circuits, the static delay of the final circuit is equal to the longest sensitizable path under the unit delay model. Data for the minimum delay circuits obtained by each optimization method are shown. *Speed_up* generates a minimum delay circuit by repeatedly performing local delay optimizations until no further improvement in delay is achieved. To obtain a minimum delay circuit using *sense_opt*, the *sense_opt* algorithm is performed on the original network for various target delays to find the minimum target delay for which the sensitization functions meet the required timing constraints. The *sense_opt* algorithm is also iterated on the resulting network until there is no further delay improvement. Although not shown, a tradeoff between the delay and area using *sense_opt* can be realized similar to that attainable with *speed_up* [8].

[1] If P is also part of some path of length $\leq n\tau$, then the false path elimination algorithm performs duplication to ensure that P is retained for these shorter paths [6, 1].

Circuit	Initial		Speed_up		Sense_opt	
	Area	Delay	Area	Delay	Area	Delay
bw	288	23	378	19	375	8
k2	1699	17	1681	16	1813	13
ampbpreg	1395	18	1529	14	1793	11
duke2	666	14	688	12	829	10
amppint2	919	(20) 16	965	13	1035	11
misex3c	820	(25) 22	909	19	1327	17
cbp.16.4	353	43	614	16	532	15
t481	1187	17	529	16	636	15
sbiucb1	390	16	395	13	573	12
cps	1884	16	1904	12	2123	11
rd84	209	11	209	11	244	10
pdc	601	14	600	11	658	10
ampxhdl	533	16	549	11	674	10
ampbsm	1120	(15) 14	1112	10	1220	9
b9	194	9	204	7	229	6
des	6051	18	6115	17	6401	16
ex1010	3936	14	3920	14	4006	13
ex4	918	11	936	10	1007	9
dflgrcb1	495	12	502	9	576	8
fconrcb1	360	12	382	9	438	8
cordic	130	11	170	9	180	8
tfaultcb1	278	9	280	8	296	7
b12	148	6	142	6	142	6
misex2	165	7	182	6	172	6
kcctlcb3	349	9	399	7	414	7
rot	1086	(21) 20	1164	14	1469	14
spla	966	(16) 14	1088	10	1178	10
dalu	1594	19	1515	13	2004	13
C1908	800	(30) 28	988	22	1899	22
C1355	820	19	1140	15	1302	15
C2670	1237	24	1298	16	1228	18

Table 1: Delay optimization on combinational circuits

There are three sets of results. The first set of examples are those for which *sense_opt* outperforms *speed_up*. Many of these circuits have large numbers of false paths. For these circuits, *sense_opt* yields substantial gains over *speed_up* corroborating the thesis of this paper. The second set of examples are those for which both techniques yield the same delay. In most of these cases, *speed_up* produces a smaller circuit. Notice that the area difference is small in most cases. However, for examples *dalu* and *C1908*, the area resulting from *sense_opt* is substantially larger than that produced by *speed_up*. The greater area overhead for these *sense_opt* examples results from duplication during the network recombination step, and fanout path duplication during the KMS procedure. In the third set, is the one benchmark example for which *sense_opt* fails to achieve the delay produced by *speed_up*. For this example, the sensitization functions cannot be optimized to be at most $\tau - 1$ although the original circuit can be optimized to have delay τ. As mentioned previously, the delay of the sensitization function is initially at most one more than the delay of the original function. For this example, that difference in initial delay apparently prevents achievement of the same delay as *speed_up*.

5.2 Sequential logic circuits

Table 2 compares the results of *speed_up* against *sense_opt* on optimized sequential logic circuits. The unreachable states of the circuit are first extracted and used as external don't cares in area optimization using the standard script (*script.rugged*) in SIS [7]. The circuit is next decomposed into 2-input NAND gates using a delay driven decomposition (*good_decomp; eliminate -1; speed_up -i*). Finally, single stuck-fault sequential redundancy removal is performed to ensure the circuit is minimal. These steps produce the initial circuit used for comparing *speed_up* and *sense_opt*.

Delay optimization is performed as follows. *Speed_up* or

Circuit	Initial			Speed_up		Sense_opt		
	Area	Delay	L	Area	Delay	Area	Delay	L
bbara	92	10	4	96	8	84	5	9
s510	429	16	6	461	13	548	10	6
s349	207	16	15	248	11	263	8	15
dk512	102	12	4	110	8	129	6	8
s386	169	12	6	211	9	233	7	6
s344	206	16	15	251	11	231	9	15
dk27	44	8	3	46	5	39	4	3
mc	45	7	2	42	5	44	4	2
modulo12	44	5	4	44	5	48	4	2
s298	154	7	14	146	6	154	5	14
train11	81	7	4	81	7	71	6	4
ex7	114	7	4	114	7	148	6	4
bbsse	210	9	4	214	8	227	7	7
dk14	182	8	3	182	8	173	7	3
s208	139	12	8	157	9	149	8	8
kirkman	434	10	4	434	10	431	9	4
s641	235	16	14	295	11	403	10	14
s820	503	12	5	511	11	551	10	5
s832	472	12	5	472	11	514	10	5
s1238	960	15	18	990	14	1131	13	18
bbtas	41	5	3	37	4	36	4	3
dk15	129	7	2	131	6	131	6	2
dk16	516	9	5	502	9	500	9	5
dk17	126	7	3	126	7	126	7	3
ex3	132	7	4	132	7	130	7	4
ex4	131	7	4	131	7	130	7	4
planet	881	13	6	991	11	955	11	6
scf	1342	12	7	1342	12	1314	12	7
beecount	79	7	3	78	6	79	6	3
cse	313	11	4	309	10	331	10	4
ex1	410	12	5	395	10	409	10	5
ex2	261	10	5	271	9	274	9	5
mark1	126	7	4	120	6	130	6	4
opus	143	9	4	137	8	165	8	4
sand	876	13	5	936	11	997	11	5
s1196	976	16	18	1020	13	1172	13	18
s713	237	16	14	303	11	318	11	15
keyb	440	14	5	444	11	435	12	5

Table 2: Delay optimization on sequential circuits

sense_opt is used to reduce the combinational delay of the circuit. Appropriate don't cares arising from two-cycle false paths are generated and used in simplification of the sensitization functions. Next, two-cycle false paths are identified and eliminated; this is followed by sequential redundancy removal and retiming to reduce the clock period [1]. As a safety check, we verify that the final circuit is equivalent to the initial circuit.

The additional last column in the table indicates the number of latches in the final circuit. The number of latches changes only if multi-cycle false paths are exploited or retiming is successful in reducing the delay of the circuit. We have experimented only with multi-cycle paths. More extensive experiments with multi-cycle paths over more than two time frames remain to be done.

The first set of examples are those for which the delay obtained using *sense_opt* is better than that produced by *speed_up*. The delay reductions are substantial on several examples. Although the area decreases in some cases with *sense_opt*, the increased number of latches offsets some of this gain. As expected, the results obtained using *sense_opt* are also uniformly as good as those reported in [1], where delay reduction is achieved by removing multi-cycle false paths but without resynthesis of the combinational logic.

The second set of examples are those for which the delay achieved using *speed_up* and *sense_opt* are the same. This typically occurs in those circuits where multi-cycle false paths do not exist and retiming cannot reduce the clock period. In some of the examples, the set of unreachable states is either empty or very small, leaving little room for the exploitation of external don't cares in simplification of the sensitization functions. On the circuit *keyb*, *sense_opt* is unable to achieve the same delay reduction as *speed_up*.

6 Conclusions

Previous work in performance optimization falls into two categories: path-based topological approaches that do not consider functional interactions, and false path elimination based techniques which do not perform resynthesis of the nodes within the circuit. We have presented a procedure that considers both functional and topological interactions in performance optimization of circuits. Our procedure combines both the topological and functional interactions in combinational and sequential logic circuits and incorporates the advantages of both categories of performance optimization techniques. Comprehensive experimental results demonstrate that the gains over a state-of-the-art path-based approach [8] are significant for combinational and sequential logic circuits.

References

[1] P. Ashar, S. Dey, and S. Malik. Exploiting multi-cycle false paths in performance optimization. In *Proceedings of the International Conference on Computer-Aided Design*, pages 510–517, November 1992.

[2] H. Chen and D. Du. Circuit enhancement by eliminating long false paths. In *Proceedings of the Design Automation Conference*, pages 249–252, June 1992.

[3] H. Chen, D. Du, and L-R. Liu. Critical path selection for performance optimization. In *IEEE Transactions on Computer-Aided Design*, pages 165–195, February 1993.

[4] P. McGeer, R. Brayton, A. Sangiovanni-Vincentelli, and S. Sahni. Performance enhancement through the generalized bypass transform. In *Proceedings of the International Conference on Computer-Aided Design*, pages 184–187, November 1991.

[5] P. McGeer, A. Saldanha, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. Timing analysis and delay-fault test generation using path recursive functions. In *Proceedings of the International Conference on Computer-Aided Design*, pages 180–183, November 1991.

[6] A. Saldanha, R. Brayton, and A. Sangiovanni-Vincentelli. Circuit structure relations to redundancy and delay: The KMS algorithm revisited. In *Proceedings of the Design Automation Conference*, pages 245–248, June 1992.

[7] E. Sentovich, K. Singh, C. Moon, H. Savoj, R. Brayton, and A. Sangiovanni-Vincentelli. Sequential circuit design using synthesis and optimization. In *Proceedings of the International Conference on Computer Design*, pages 328–333, October 1992.

[8] K. Singh. Performance optimization of digital circuits. *Ph.D. Thesis, University of California - Berkeley*, November 1992.

[9] K. Singh, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli. Timing optimization of combinational logic. In *Proceedings of the International Conference on Computer-Aided Design*, pages 282–285, November 1988.

[10] H. Touati and R.K. Brayton. Computing the initial states of retimed circuits. *IEEE Transactions on Computer-Aided Design*, July 1992.

[11] H. Touati, H. Savoj, and R. Brayton. Delay optimization of combinational logic circuits through clustering and partial collapsing. In *Proceedings of the International Conference on Computer-Aided Design*, pages 188–191, November 1991.