Logic Clause Analysis for Delay Optimization

Bernhard Rohfleisch Bernd Wurth* Kurt Antreich
Institute of Electronic Design Automation
Technical University of Munich
80290 Munich, Germany

Abstract – *In this paper, we present a novel method for topological delay optimization of combinational circuits. Unlike most previous techniques, optimization is performed after technology mapping. Therefore, exact gate delay information is known during optimization.*

Our method performs incremental network transformations, specifically substitutions of gate input or output signals by new gates. We present new theory which relates incremental network transformations to combinations of global clauses, and show how to detect such valid clause combinations. Employing techniques which originated in the test area, our method is capable to globally optimize large circuits.

Comprehensive experimental results show that our method reduces the delay of large standard cell netlists by 23% on average. In contrast to most other delay optimization techniques, area reductions are achieved concurrently.

1 Introduction

We address the problem of delay optimization of combinational circuits. There are two classes of approaches to performance optimization. The first class consists of approaches which determine and remove long false paths [1,2]. After removing all long false paths, the circuit delay is equal to the delay of the longest topological path. Techniques which reduce the delay of the topological critical path form the second class [3,4]. Our work belongs to the second class. Recently, Saldanha et al. [5] presented an approach which takes into account both functional and topological aspects. Sensitization functions are added to a circuit to improve performance functionally. Effective topological delay optimization techniques are required to speed up the sensitization function circuitry.

As we consider methods of the second class, the *critical* path will always refer to the longest topological path. Most delay optimization methods of the second class are applied before technology mapping. The idea is to shorten the critical path of the logic network such that a possibly delay-oriented technology mapping step [6] produces a circuit with reduced delay. However, during optimization the delay must be evaluated on the unmapped network. Thus, simplified delay models such as the unit delay model are used. Singh et al. [3] identify regions of logic near the critical path that are collapsed and resynthesized. As the identification of the critical path is based on a simple delay model, this method may miss the real critical path and thus yield unpredictable results. This problem was pointed out by Touati et al. [4], who therefore attempt to shorten all paths of the unmapped network. Since also uncritical network regions are optimized for delay, the associated area penalty may be unnecessarily large. Touati's as well as Singh's methods are based on a partial collapsing and resynthesis of network regions.

Chen et al. propose a delay optimization method which has the advantage that it works on mapped networks [7]. This method directly reduces the levels of gates on the critical path by removing critical inputs of gates. Allowed removals of critical inputs are identified by permissible functions [8] which express global information about the network. However, permissible functions must be computed in terms of primary inputs. Thus, this method is not applicable to large circuits.

Our new approach overcomes the limitations of these approaches. It optimizes a netlist of gates after technology mapping such that exact delay information of gates is known. Only gates on the critical paths are optimized. Consequently, in most cases no area penalty is associated with delay optimization. On the contrary, area can mostly be reduced concurrently with delay. Moreover, our method is able to globally optimize large circuits. As an example, we succeed in reducing the delay of benchmark circuit `C6288` by 22% after technology mapping. Our approach is based on detecting global dependencies between signals. To detect such global dependencies, we generalize techniques which originated in the test area.

A further contribution of our work is a comprehensive theory on incremental network transformations based on clauses. We develop basic correspondences between allowed network transformations and valid clause combinations. Our method incorporates a rich set of incremental network transformations. We introduce transformations where the output of a new AND-, OR-, or XOR-gate substitutes existing gate outputs or inputs. Combined with substitutions of gate outputs and inputs by single signals, these new transformations are very powerful in restructuring the network.

The remainder of the paper is organized as follows. In Section 2 we demonstrate how a circuit can be described by valid clauses. In Section 3, we introduce netlist transformations such as output and input substitutions involving two or three signals, and show their relation to global clauses. Our approach to compute valid clauses and the application of clause analysis to delay optimization are discussed in Section 4 and 5. We present experimental results in Section 6.

2 Circuit Description by Clauses

We consider netlists of combinational logic gates. If the signal at the output of a gate branches out to several fanout gates, we distinguish between the root of the branching signal, called *stem signal*, and the individual branches, called *branch signals*. The output signal of a gate with a single fanout is regarded as a stem signal.

Our view of a circuit is similar as introduced by Larrabee [9]. Each gate of the netlist has associated a logic formula, which contains the variables corresponding to the gate's input

*Bernd Wurth was supported by an Ernst von Siemens-grant given by Siemens AG.

32nd ACM/IEEE Design Automation Conference ®

and output terminals. The formula is a characteristic function that is true iff the values assigned to the variables are consistent with the gate's truth table. The formula is represented in conjunctive normal form, i.e., a product of sums form. A characteristic formula for the whole circuit is obtained by taking the conjunction of the formulas of all gates. Thus, the function for the whole circuit is true iff the values assigned to input, output, and internal variables are consistent with the truth tables of all gates.

A sum of variables is called a *clause*. Valid clauses describe dependencies among the variables they contain.

Definition 1 *A clause is* valid *iff it evaluates to one for every assignment of signal values produced by primary input vectors.*

To denote that a clause, say $(\mathbf{a}+\mathbf{b})$, is valid we use the notation $((\mathbf{a}+\mathbf{b}) \equiv 1)$.

As an example, consider the circuit in Figure 1, which is taken from [9]. The formula for the AND-gate is $(\overline{\mathbf{d}}+\mathbf{a}) \cdot (\overline{\mathbf{d}}+\mathbf{b}) \cdot (\mathbf{d}+\overline{\mathbf{a}}+\overline{\mathbf{b}})$, the formula for the inverter is $(\mathbf{c}+\mathbf{e}) \cdot (\overline{\mathbf{c}}+\overline{\mathbf{e}})$, and the formula for the OR-gate is $(\mathbf{f}+\overline{\mathbf{d}}) \cdot (\mathbf{f}+\overline{\mathbf{e}}) \cdot (\overline{\mathbf{f}}+\mathbf{d}+\mathbf{e})$. The individual clause $(\overline{\mathbf{d}}+\mathbf{a})$ denotes that any consistent value assignment requires either $(\mathbf{d}=0)$ or $(\mathbf{a}=1)$.

In the context of test pattern generation, *global implications* have been introduced by Schulz et al. [10]. Global implications correspond to valid global clauses which cannot be derived from a single gate's formula. Global clauses describe global signal dependencies and thereby indicate optimization potential.

The local and global clauses considered so far are sums of signal variables. Additionally, an *observability variable* can be associated with a signal to express whether a fault on this signal is observable at an output of the circuit. We will denote the observability of variable $\mathbf{a}$ by $\mathbf{O_a}$. Clauses of signal and observability variables express the relation between signal observabilities and signal values. Clauses derivable from the circuit structure can be added to the circuit's characteristic formula.

From the circuit of Figure 1, the following clauses can be derived. If input $\mathbf{a}$ of the AND-gate is observable, output $\mathbf{d}$ must be observable. This implication between observability variables is expressed by the valid clause $(\overline{\mathbf{O_a}}+\mathbf{O_d})$. Similarly, the clause $(\overline{\mathbf{O_b}}+\mathbf{O_d})$ is valid. Since input $\mathbf{a}$ of the AND-gate is observable at signal $\mathbf{d}$ only if the other input $\mathbf{b}$ equals 1, the clause $(\overline{\mathbf{O_a}}+\mathbf{b})$ is valid, also $(\overline{\mathbf{O_b}}+\mathbf{a})$ is valid.

Similarly to the global clauses of signal variables, there exist global clauses of signal variables and observability variables. As an example, assume a circuit with a stuck-at-1 redundant fault at signal $\mathbf{a}$. For any primary input vector, the signal is either unobservable, $(\overline{\mathbf{O_a}}=1)$, or its value equals one, $(\mathbf{a}=1)$. Hence, $(\overline{\mathbf{O_a}}+\mathbf{a})$ is a valid global binary clause.

In this paper, we are dealing with a class of global clauses which contain the negated observability variable $\overline{\mathbf{O_a}}$ and a number of signal variables including $\mathbf{a}$. We classify clauses depending on how many signal variables they contain. The signals $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ are stem or branch signals.

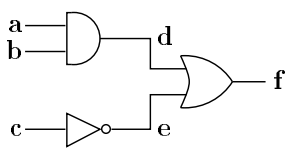

Figure 1: Example circuit.

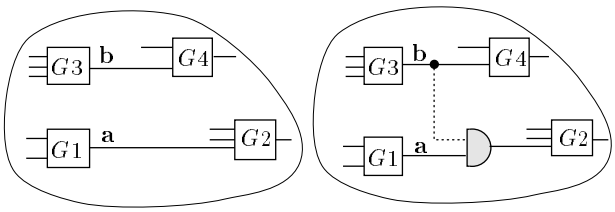

Figure 2: Additional AND-gate associated with the valid clause $(\overline{\mathbf{O_a}}+\overline{\mathbf{a}}+\mathbf{b})$.

C1-clauses:	C2-clauses:	C3-clauses:
$(\overline{\mathbf{O_a}}+\overline{\mathbf{a}})$	$(\overline{\mathbf{O_a}}+\overline{\mathbf{a}}+\overline{\mathbf{b}})$	$(\overline{\mathbf{O_a}}+\overline{\mathbf{a}}+\overline{\mathbf{b}}+\overline{\mathbf{c}})$
$(\overline{\mathbf{O_a}}+\mathbf{a})$	$(\overline{\mathbf{O_a}}+\overline{\mathbf{a}}+\mathbf{b})$	$(\overline{\mathbf{O_a}}+\overline{\mathbf{a}}+\overline{\mathbf{b}}+\mathbf{c})$
	$(\overline{\mathbf{O_a}}+\mathbf{a}+\overline{\mathbf{b}})$	$(\overline{\mathbf{O_a}}+\overline{\mathbf{a}}+\mathbf{b}+\overline{\mathbf{c}})$
	$(\overline{\mathbf{O_a}}+\mathbf{a}+\mathbf{b})$	$(\overline{\mathbf{O_a}}+\overline{\mathbf{a}}+\mathbf{b}+\mathbf{c})$
		$(\overline{\mathbf{O_a}}+\mathbf{a}+\overline{\mathbf{b}}+\overline{\mathbf{c}})$
		$(\overline{\mathbf{O_a}}+\mathbf{a}+\overline{\mathbf{b}}+\mathbf{c})$
		$(\overline{\mathbf{O_a}}+\mathbf{a}+\mathbf{b}+\overline{\mathbf{c}})$
		$(\overline{\mathbf{O_a}}+\mathbf{a}+\mathbf{b}+\mathbf{c})$

We will now show the usefulness of these global clauses for incremental circuit transformations.

3 Circuit Transformations and Valid Clauses

We have already seen that a valid C1-clause describes a stuck-at redundant fault. The permissible circuit transformation associated with a valid C1-clause is redundancy removal [11].

Definition 2 *A circuit transformation is called* permissible *if it preserves the input/output behavior of the circuit.*

Circuit Transformations related to C2-Clauses. A single C2-clause is associated with inserting a 2-input gate into the circuit, as shown in Figure 2. The connection between gates $G1$ and $G2$ is cut and an AND-gate driven by $\mathbf{a}$ and $\mathbf{b}$ is inserted. This transformation is permissible iff all primary input vectors for which $\mathbf{a}$ is observable, i.e. $(\mathbf{O_a}=1)$, assign values to $\mathbf{a}$ and $\mathbf{b}$ such that $(\mathbf{a}=\mathbf{a}\cdot\mathbf{b})$ holds. Formally, this condition can be written as an implication: $(\mathbf{O_a}=1) \Longrightarrow (\mathbf{a}=\mathbf{a}\cdot\mathbf{b})$. The term $(\mathbf{a}=\mathbf{a}\cdot\mathbf{b})$ is true whenever $(\overline{\mathbf{a}}+\mathbf{b}=1)$ is true, so the condition can be rewritten as $(\mathbf{O_a}=1) \Longrightarrow (\overline{\mathbf{a}}+\mathbf{b}=1)$, or $((\overline{\mathbf{O_a}}+\overline{\mathbf{a}}+\mathbf{b}) \equiv 1)$. Hence, the C2-clause $(\overline{\mathbf{O_a}}+\overline{\mathbf{a}}+\mathbf{b})$ is valid.

Several circuit optimization approaches have been presented based on this modification. In [12], the new 2-input gate is termed a permissible basis bridge. Inserting 2-input gates is described by D-implications [13] and connection faults [14,15]. Adding a new gate perturbs the network and can make other signals stuck-at redundant such that after removal of these redundancies an optimization gain is achieved. This concept is exploited in [13,14].

Combining several C2-clauses may directly yield a gain in circuit area [12] or delay as shown in this paper.

Definition 3 *An* output substitution OS2$(\mathbf{a},\mathbf{b})$ *substitutes the stem signal* $\mathbf{a}$ *by signal* $\mathbf{b}$. *An* input substitution IS2$(\mathbf{a},\mathbf{b})$ *substitutes the branch signal* $\mathbf{a}$ *by signal* $\mathbf{b}$. *Output and input substitutions with inverted signal* $\mathbf{b}$ *are defined analogously.*

OS2 and IS2 substitutions are illustrated in Figure 3.

Theorem 1 *An output substitution OS2*$(\mathbf{a},\mathbf{b})$ *is permissible iff the conjunction of the two C2-clauses* $(\overline{\mathbf{O_a}}+\mathbf{a}+\overline{\mathbf{b}}) \cdot (\overline{\mathbf{O_a}}+\overline{\mathbf{a}}+\mathbf{b})$ *is valid.*

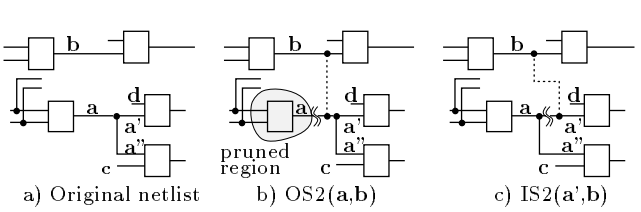

Figure 3: Output substitution OS2(**a**,**b**) and input substitution IS2(**a**',**b**).

Proof: The substitution of signal **a** by signal **b** is permissible iff all input vectors for which **a** is observable, assign values to **a** and **b** such that $(\mathbf{a} = \mathbf{b})$. Formally, $(\mathbf{O_a} = 1) \Longrightarrow (\mathbf{a} = \mathbf{b})$ holds. The term $(\mathbf{a} = \mathbf{b})$ is true whenever $((\mathbf{a} + \overline{\mathbf{b}}) \cdot (\overline{\mathbf{a}} + \mathbf{b}) = 1)$ is true, so the implication can be rewritten as $((\overline{\mathbf{O_a}} + \mathbf{a} + \overline{\mathbf{b}}) \cdot (\overline{\mathbf{O_a}} + \overline{\mathbf{a}} + \mathbf{b}) \equiv 1)$. Hence, the conjunction of the two C2-clauses $(\overline{\mathbf{O_a}} + \mathbf{a} + \overline{\mathbf{b}}) \cdot (\overline{\mathbf{O_a}} + \overline{\mathbf{a}} + \mathbf{b})$ is valid. □

Theorem 1 can easily be extended to substitutions with inverted signal **b**.

If an output substitution for the stem signal **a** is performed, all gates driven by **a** are connected to **b** instead and all gates exclusively necessary to compute **a** are pruned, as shown in Figure 3 b). An output substitution yields a gain in area and possibly in delay. Input substitutions are potentially useful for delay optimization. For example, if the input substitution IS2($\mathbf{a}', \mathbf{b}$) for the branch signal $\mathbf{a}'$ is permissible, and if $\mathbf{a}'$ is on a critical path, its substitution by signal **b** reduces the path delay if the arrival time at **b** is smaller than the arrival time at $\mathbf{a}'$.

Circuit transformations related to C3-clauses. Now we introduce further netlist modifications, which generalize the concept of OS2 and IS2 substitutions. Substitutions associated with C3-clauses are the *output substitution OS3* and the *input substitution IS3*.

Definition 4 *An* output substitution OS3($\mathbf{a}, \mathbf{b}, \mathbf{c}$) *substitutes the stem signal* **a** *by the output of a new gate driven by the signals* **b** *and* **c**. *Similarly, an* input substitution IS3($\mathbf{a}, \mathbf{b}, \mathbf{c}$) *substitutes the branch signal* **a** *by the output of a new gate driven by the signals* **b** *and* **c**.

For an OS3 or an IS3 substitution, an AND-, OR-, or XOR-gate with a certain phase assignment to the driving signals **b** and **c** is inserted. As in the case of the substitutions OS2 and IS2, there can be a gain in area and delay due to the substitutions OS3 and IS3. The circuit area is reduced by an output substitution OS3, if the area required by the new 2-input gate is smaller than the area of the pruned region. Delay reduction by OS3 and IS3 is possible, if the substituted signal **a** belongs to a critical path and if the arrival time of the newly inserted gate is smaller than the arrival time of signal **a**.

Theorem 2 *The substitution of signal* **a** *by the output signal of a new AND-gate with inputs* **b** *and* **c** *is permissible iff the conjunction of the clauses* $(\overline{\mathbf{O_a}} + \overline{\mathbf{a}} + \mathbf{b}) \cdot (\overline{\mathbf{O_a}} + \overline{\mathbf{a}} + \mathbf{c}) \cdot (\overline{\mathbf{O_a}} + \mathbf{a} + \overline{\mathbf{b}} + \overline{\mathbf{c}})$ *is valid.*

The substitution is illustrated in Figure 4.

Proof: Whenever signal **a** is unobservable, i.e. $(\mathbf{O_a} = 0)$, the value of signal **a** may be changed. However, if **a** is observable, the value produced at the output of the AND-gate must be equal to the value of **a**. We can write this condition as an implication: $(\mathbf{O_a} = 1) \Longrightarrow (\mathbf{a} = \mathbf{b} \cdot \mathbf{c})$. The term $(\mathbf{a} = \mathbf{b} \cdot \mathbf{c})$ can be expressed in sum of clauses form as $((\overline{\mathbf{a}} + \mathbf{b}) \cdot (\overline{\mathbf{a}} + \mathbf{c}) \cdot (\mathbf{a} + \overline{\mathbf{b}} + \overline{\mathbf{c}}) \equiv 1)$. Therefore, the condition for the substitution to be permissible is $((\overline{\mathbf{O_a}} + \overline{\mathbf{a}} + \mathbf{b}) \cdot (\overline{\mathbf{O_a}} + \overline{\mathbf{a}} + \mathbf{c}) \cdot (\overline{\mathbf{O_a}} + \mathbf{a} + \overline{\mathbf{b}} + \overline{\mathbf{c}}) \equiv 1)$. □

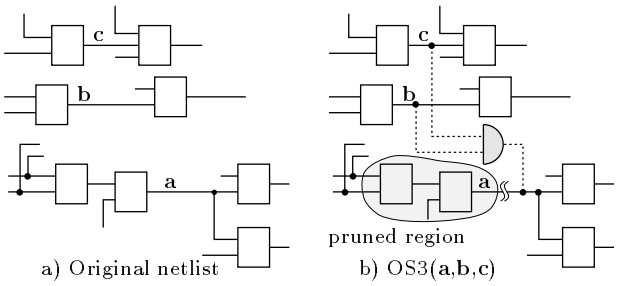

Figure 4: OS3(**a**,**b**,**c**) substitution with AND-gate.

Extension of Theorem 2 to other types of 2-input gates is straightforward. It shows that any output substitution OS3($\mathbf{a}, \mathbf{b}, \mathbf{c}$) by an AND- or an OR-gate with arbitrary phase-assignment to **b** and **c** involves two valid C2-clauses and one valid C3-clause. OS3-substitutions involving an XOR- or XNOR-gate are associated with the combination of four valid C3-clauses each. For the XOR-gate, we have $((\overline{\mathbf{O_a}} + \overline{\mathbf{a}} + \mathbf{b} + \mathbf{c}) \cdot (\overline{\mathbf{O_a}} + \overline{\mathbf{a}} + \overline{\mathbf{b}} + \overline{\mathbf{c}}) \cdot (\overline{\mathbf{O_a}} + \mathbf{a} + \mathbf{b} + \overline{\mathbf{c}}) \cdot (\overline{\mathbf{O_a}} + \mathbf{a} + \overline{\mathbf{b}} + \mathbf{c}) \equiv 1)$, and for the XNOR-gate, we have $((\overline{\mathbf{O_a}} + \overline{\mathbf{a}} + \mathbf{b} + \overline{\mathbf{c}}) \cdot (\overline{\mathbf{O_a}} + \overline{\mathbf{a}} + \overline{\mathbf{b}} + \mathbf{c}) \cdot (\overline{\mathbf{O_a}} + \mathbf{a} + \mathbf{b} + \mathbf{c}) \cdot (\overline{\mathbf{O_a}} + \mathbf{a} + \overline{\mathbf{b}} + \overline{\mathbf{c}}) \equiv 1)$. Note, that the same conditions hold for input subsitutions, where **a** refers to a branch signal.

4 Computation of Valid Clauses

There exists a variety of methods to determine valid global clauses. Bit-parallel fault simulation (BPFS)[16] has been adapted and combined with automatic test pattern generation (ATPG) for this purpose [12]. Another method is to compute global implications using the circuit structure [10,13], or an implication graph [9,17]. So far, however, all of these methods have only been used to compute C2-clauses. To calculate valid C3-clauses, we generalized the concept based on BPFS.

According to Definition 1, a clause is valid iff it evaluates to one for all signal assignments produced by primary input vectors. Therefore, a clause is *invalid* iff there exists a primary input vector which causes an assignment of values to the signals such that the clause evaluates to zero.

We start BPFS assuming that each clause is valid. For each simulated input vector, each clause is checked and immediately discarded if proven invalid by the current signal assignment. Using BPFS, a set of l input vectors is simulated parallely, where l denotes the machine word length. Thus, BPFS can be used to efficiently identify invalid clauses.

If simulation is exhaustively, i.e., the complete set of input vectors is simulated, then the clauses not discarded during BPFS are valid. In general, of course, exhaustive simulation can not be performed since the number of input vectors grows exponentially with the number of circuit inputs. Instead, a set of random input vectors is simulated to discard the vast majority of invalid clauses. The clauses not discarded during BPFS are either valid or invalid. We still have to prove the validity or invalidity of these *potentially valid* clauses.

As we intend to compute valid clause combinations associated with output or input substitutions, we do not prove the validity of a single clause, but of clause combinations. Potentially valid clauses are combined to the clause combinations discussed in Section 3. A set of *potentially valid clause combinations* (PVCCs) is thus obtained. Then, validity of the individual PVCCs can be checked via ATPG [10]. Alternatively, the validity of a PVCC can be checked by carrying out the circuit modification associated with the PVCC, and performing a

BDD-based verification of the original circuit versus the modified circuit. For small and medium sized circuits, this method turned out to consume less CPU time. ATPG, however, enables the optimization of circuits for which BDD representations become too large.

This procedure can be adapted to detect valid clauses of any order. While very efficient for the identification of C2-clauses, the initial number of clauses is a serious problem when C3-clauses are to be found. In a circuit with n signals, an upper limit N_{C2} on the number of C2-clauses is given by the number of **a**-signals times the number of signals $\mathbf{b} \neq \mathbf{a}$: $N_{C2} = n \cdot (n-1)$. To avoid cycles, signal **b** may not be situated in the transitive fanout of signal **a**. Thus, the upper limit on the number of C2-clauses is somewhat smaller than $n \cdot (n-1)$. The upper limit N_{C3} on the number of C3-clauses is given by $n \cdot \binom{n-1}{2}$. For $n = 1000$, we have $N_{C3} = 5 \cdot 10^8$. Simply by the number of potential C3-clauses, the computation of valid C3-clauses is a difficult problem due to memory consumption and CPU times.

Therefore, methods are needed to reduce the set of considered clauses before BPFS. A list of clauses is associated with each **a**-signal. Reducing the number of **a**-signals decreases the number of clause lists, while a reduction of the number of **b**/**c**-signals for a given **a**-signal decreases the lengths of individual clause lists.

In the sequel, we will shortly discuss three methods to reduce the set of C3-clauses considered before BPFS:

Reduction without loss of optimization quality: Trivially, branch signals need not be considered as **b**/**c**-signals. Furthermore, a clause need not be considered if it can only be used in clause combinations such that the associated circuit modifications yield no gain. In the case of delay optimization, a clause need not be considered if the arrival time of a **b**/**c**-signal plus the delay of the inserted gate is larger than the arrival time of the **a**-signal.

Reduction with loss of optimization quality by exploitation of C2-clauses: According to Theorem (2), an output substitution OS3 involving an AND- or OR- gate is permissible iff two C2-clauses are valid. Therefore, we exploit the results of a simulation for C2-clauses, which precedes simulation for C3-clauses. This reduces the number of clause lists as well as the lengths of clause lists. Experiments have shown that the number of considered clauses is thus reduced to some percent. However, some OS3 substitutions involving XOR-gates cannot be detected any more.

Reduction with loss of optimization quality by exploitation of structural dependencies: The **b**/**c**-signals must be structurally related to the **a**-signal. Taking into account path lengths from the primary inputs to **a**-signal and **b**/**c**-signals, and path lengths between **a**-signal and **b**/**c**-signals, we reduce the number of considered clauses by 90 % at a loss of valid clause combinations of about 10 %.

Using these methods, the number of considered clauses is reduced to less than 10^7 in all the benchmark circuits we examined.

5 Exploitation of Valid Clauses for Delay Optimization

We achieve an optimization of the netlist by incremental transformations. As we work on mapped netlists, we can use the gate delay times as specified in the library. Any of the applied modifications preserves the mapping. If XOR-gates are not contained in the library, they can be excluded from consideration by not building the corresponding PVCCs from the clauses after simulation.

Gates on a critical path, called *critical gates*, are identified by slack computation. We directly optimize the critical path by substituting outputs and inputs of critical gates. Only critical gates are considered as **a**-signals during BPFS.

After simulation, we sort the PVCCs according to two criteria. The first criterion is the number of critical paths (NCPs) leading through the **a**-signal of a PVCC. Selecting a modification with a maximum NCP value shortens as many critical paths as possible in order to obtain a reduction of the overall circuit delay.

Modifications with an identical NCP are sorted according to their local delay save (LDS). The LDS for the OS2 substitution shown in Figure 3 b) is given by the difference of the arrival time of signal **a** and the maximum arrival time of signals **b**, **c**, and **d**. The extension of the LDS-computation to other modifications (IS2, OS3 and IS3) is straightforward. The valid modification with the maximum number of critical paths and the maximum local delay save is chosen and executed in the circuit. Note that the LDS of a modification is just an upper bound for the reduction in circuit delay by this modification. Other paths may be critical in the modified circuit. Therefore, after a modification we update the slack and determine the current set of critical gates.

Our main procedure consists of two phases. In the first phase, called *delay reduction phase*, we apply substitutions of critical gate outputs and inputs. This phase incrementally reduces the delay of the circuit. As simulation of C2-clauses costs less CPU time, substitutions OS2 and IS2 are carried out first. Note that several modifications per simulation are performed. If no delay improvement by C2-related modifications is possible any more, we start simulation for C3-clauses. As discussed in the previous section, the result of the previous simulation for C2-clauses can be used to reduce the number of considered C3-clauses. If no delay reducing substitutions are found anymore, we enter the second phase.

The second phase, called *area optimization phase*, reduces circuit area without increasing the delay. In this phase, substitutions of noncritical gates are carried out unless they create new critical paths. Again, C3-simulations follow C2-simulations. Our experiments showed that delay reducing substitutions may be permissible again after some substitutions in the area optimization phase have been performed. Therefore, the algorithm goes back to the delay reduction phase whenever a certain number of substitutions has been executed in the area optimization phase. The algorithm terminates if neither delay reducing nor area reducing substitutions are found anymore.

6 Experimental Results

To evaluate our delay optimization method, we used ISCAS-85 and ISCAS-89 benchmark circuits [18]. We implemented our delay optimization technique in program GDO (Global Delay Optimization), which is embedded into the synthesis tool TOS.

We present two sets of experimental results. For both sets, we first optimized each circuit using the standard script (*script.rugged*) in SIS [19].

In the first set, we mapped the circuits to the MCNC library *mcnc.genlib* with the SIS command *map -n 1*. Mapping was done without fanout optimization since at this point we do not consider fanout dependencies in our implementation. Table 1 shows the number of *gates*, *literals*, and the circuit *delay* before and after GDO was run. We give the number of OS2/IS2-modifications, the number of OS3/IS3-modifications, and the CPU time in seconds on a DEC 3000/600.

Our global delay optimization technique yields an average

Table 1: Results of GDO on a set of benchmark circuits.

circuit	♯gates		♯literals		delay		♯mod.		CPU
	before	after	before	after	before	after	OS/IS2	OS/IS3	[sec]
Z5xp1	106	77	212	152	32.7	10.6	42	0	52
term1	152	125	301	260	13.3	10.7	21	12	197
9sym	193	170	403	359	14.1	12.6	24	6	84
C432	150	140	318	302	29.9	26.4	9	4	238
C499	370	352	920	772	23.4	19.0	88	74	2416
C1355	370	352	920	772	23.4	19.0	88	74	2400
C880	337	289	722	650	50.6	41.0	19	24	658
C1908	488	402	933	803	41.2	33.9	51	53	1364
vda	700	633	1165	1086	25.0	16.6	27	37	4608
rot	637	581	1178	1089	26.6	25.2	15	24	1431
alu4	680	528	1260	1026	40.2	28.6	82	69	7252
x3	654	631	1354	1286	17.2	14.5	43	10	1480
apex6	686	668	1372	1306	20.1	18.6	27	11	1875
frg2	825	777	1489	1416	26.3	15.2	47	27	6581
pair	1447	1371	2893	2695	34.9	27.0	79	49	18654
C5315	1576	1374	3249	2790	37.3	31.0	138	86	16288
C6288	3148	3009	5357	5923	117.7	92.3	310	327	60083
$\sum$:	12519	11479	24046	22687	573.9	442.2	-	-	-
red.:		8.3%		5.7%		22.9%	-	-	-

Table 2: Results of GDO on circuits synthesized by SIS with the script *script.delay*.

circuit	♯gates		♯literals		delay		♯mod.		CPU
	before	after	before	after	before	after	OS/IS2	OS/IS3	[sec]
Z5xp1	98	82	191	168	21.4	11.9	23	9	34
term1	128	109	281	237	11.0	11.0	13	6	110
9sym	168	146	378	331	13.6	12.3	32	10	117
C432	236	173	458	351	27.0	26.5	28	8	564
C499	344	300	810	729	15.5	15.2	29	2	392
C1355	351	301	833	730	15.5	15.2	35	2	437
C880	370	320	865	721	26.6	25.6	61	16	799
C1908	683	441	1337	964	29.2	25.6	140	73	4891
apex6	743	697	1641	1464	13.5	13.5	56	10	1863
rot	760	618	1501	1226	18.9	16.0	86	29	4943
frg2	819	708	1667	1415	15.4	12.8	132	43	8428
$\sum$:	4700	3895	9962	8336	207.6	185.6	-	-	-
red.:		17.1%		16.3%		10.6%	-	-	-

delay reduction of 22.9%. Delay reductions were achieved for each circuit. The individual delay reduction ranges from 5.3% for circuit `rot` to 68% for the small circuit `Z5xp1`. Remarkably, we achieved area reductions of 5.7%. Note that the number of literals increased only for circuit `C6288`. The average area reduction of all circuits except `C6288` is above 10%. CPU times are strongly correlated with the number of performed modifications and thus the optimization quality.

In the second set of experiments, we applied GDO in combination with the depth reduction technique implemented in SIS [4]. Each circuit was optimized and mapped onto the *mcnc.genlib* library with the script *script.delay*. We then applied global delay optimization with GDO. GDO achieved additional delay reductions of 10% on average, although for some circuits, e.g, `term1, apex6`, no delay reductions were possible. Our approach concurrently reduces circuit area by 16.3%. We conjecture that GDO recovers area penalties which are due to the depth reduction technique in SIS.

7 Conclusion

We introduced logic clause analysis of combinational circuits with application to delay optimization. For the first time, basic relations between clauses and incremental netlist transformations were developed. In this context, we introduced OS3 and IS3 substitutions which contributed to delay and area reductions in mapped netlists. Our approach, which has been implemented in the delay optimizer GDO, exploits global optimization potential detected by valid clauses. Experimental results show that global delay optimization achieves significant reductions in circuit delay and area.

Acknowledgment

Special thanks are to Manfred Henftling and Hannes Wittmann who provided helpful advice in understanding and implementing clause analysis.

References

[1] K. Keutzer, S. Malik, and A. Saldanha, "Is redundancy necessary to reduce delay?," *IEEE Transactions on Computer-Aided Design*, vol. 10, no. 4, pp. 427–435, 1991.

[2] A. Saldanha, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Circuit structure relations to redundancy and delay," *IEEE Transactions on Computer-Aided Design*, vol. 13, no. 7, pp. 875–883, 1994.

[3] K. J. Singh, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Timing optimization of combinational logic," *IEEE/ACM International Conference on Computer-Aided Design, ICCAD*, pp. 282–285, 1988.

[4] H. Touati, H. Savoj, and R. K. Brayton, "Delay optimization of combinational logic circuits by clustering and partial collapsing," *IEEE/ACM International Conference on Computer-Aided Design, ICCAD*, pp. 188–191, 1991.

[5] A. Saldanha, H. Harkness, P. C. McGeer, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Performance optimization using exact sensitization," *31th ACM/IEEE Design Automation Conference, DAC*, pp. 425–429, 1994.

[6] R. Rudell, "Logic synthesis for vlsi design," *Ph. D. thesis, U.C. Berkeley Memorandum UCB/ERL M89/49*, 1989.

[7] K.-C. Chen and S. Muroga, "Timing optimization for multi-level combinational networks," *27th ACM/IEEE Design Automation Conference, DAC*, pp. 339–344, 1990.

[8] S. Muroga, Y. Kambayashi, H. C. Lai, and J. N. Culliney, "The transduction method - design of logic networks based on permissible functions," *IEEE Transactions on Computers*, vol. 38, no. 10, pp. 1404–1424, 1989.

[9] T. Larrabee, "Test pattern generation using boolean satisfiability," *IEEE Transactions on Computer-Aided Design*, vol. 11, no. 1, pp. 4–15, 1992.

[10] M. H. Schulz and E. Auth, "Improved deterministic test pattern generation with applications to redundancy identification," *IEEE Transactions on Computer-Aided Design*, vol. 8, no. 7, pp. 811–816, 1989.

[11] D. Bryan, F. Brglez, and R. Lisanke, "Redundancy identification and removal," *International Workshop on Logic Synthesis*, 1989.

[12] B. Rohfleisch and F. Brglez, "Introduction of permissible bridges with application to logic optimization after technology mapping," *The European Design and Test Conference, ED&TC*, pp. 87–93, 1994.

[13] W. Kunz and P. Menon, "Multi-level logic optimization by implication analysis," *IEEE/ACM International Conference on Computer-Aided Design, ICCAD*, pp. 6–13, 1994.

[14] K.-T. Cheng and L. A. Entrena, "Multi-level logic optimization by redundancy addition and removal," *The European Design and Test Conference, ED&TC*, pp. 373–377, 1993.

[15] S.-C. Chang, K.-T. Cheng, N.-S. Woo, and M. Marek-Sadowska, "Layout driven logic synthesis for FPGAs," *31th ACM/IEEE Design Automation Conference, DAC*, pp. 308–313, 1994.

[16] J. A. Waicukauski, E. B. Eichelberger, D. O. Forlenza, E. . Lindbloom, and T. McCarthy, "Fault simulation for structured VLSI," *VLSI Systems Design*, pp. 20–32, 1985.

[17] S. T. Chakradhar, V. D. Agrawal, and S. G. Rothweiler, "A transitive closure algorithm for test generation," *IEEE Transactions on Computer-Aided Design*, vol. 12, no. 7, pp. 1015–1028, 1993.

[18] S. Yang, "Logic synthesis and optimization benchmarks user guide, version 3.0," *MCNC, Research Triangle Park, N.C. 27709*, 1991.

[19] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Sequential circuit design using synthesis and optimization," *IEEE/ACM International Conference on Computer-Aided Design, ICCAD*, pp. 328–333, 1992.