Timing Optimization by an Improved Redundancy Addition and Removal Technique

Luis A. Entrena*, José A. Espejo*, Emilio Olías*, Javier Uceda**

* Area de Tecnología Electrónica
Universidad Carlos III de Madrid
c/Butarque, 15 28911 Leganés, Madrid, SPAIN
{entrena, ppespejo, olias}@ing.uc3m.es

** División de Ingeniería Electrónica (DIE)
Universidad Politécnica de Madrid
c/ José Gutiérrez Abascal, 2 28006 Madrid, SPAIN
uceda@upmdie.upm.es

Abstract

Redundancy Addition and Removal (RAR) uses Automatic Test Pattern Generation (ATPG) techniques to identify logic optimization transforms. It has been applied successfully to combinational and sequential logic optimization and to layout-driven logic synthesis for FPGAs. In this paper we present an improved RAR technique that allows to identify new types of optimization transforms and it is more efficient because it reduces the number of ATPG runs required. Also, we apply the RAR method to timing optimization. The experimental results show that this improved RAR technique produces significant timing optimization with very little area cost.

1. Introduction

Redundancy Addition and Removal (RAR) has been shown to be a powerful logic optimization method by several authors [1-7]. With this method, a logic network is optimized by iteratively adding and removing redundancies that are identified using Automatic Test Pattern Generation (ATPG) techniques. If the addition of *k* redundant wires/gates creates more than *k* redundant wires/gates elsewhere in the network, the removal of the created redundancies will result in a smaller area. Also, if the addition of redundant wires/gates creates redundancies in the critical path of the circuit, the removal of the created redundancies will result in a smaller circuit delay.

The RAR approach is illustrated with the example in Fig. 1 (taken from [1]). This is an irredundant circuit. In this circuit a connection can be added from the output of *g5* as a new input to *g9* without changing the logic functionality of the network. In other words, the added connection is redundant. By adding this connection, two connections, *g1-g4* and *g6-g7* become redundant and can be removed. The resulting network contains less gates and a shorter critical path.

The identification and selection of optimization transforms is a complex problem because of the large number of possible transforms that must be checked. One approach to check for valid transforms uses a combination of fault simulation and test generation [1], [12]. In another approach [2-7], that we follow in this work, the search for valid transforms is guided by the mandatory assignments obtained during test generation.

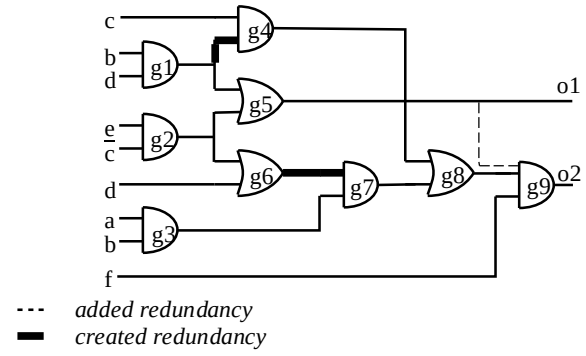

Figure 1. Example of redundancy addition and removal

The original RAR method as it was proposed in [1][2] considered only the addition of single connections to create new redundancies. Multiple wire addition and gate function substitution techniques were proposed in [3][4]. New extensions for sequential logic optimization that consider the addition of redundancies across time frames were also proposed in [7]. The basic idea underlying these approaches can be summarized as follows. A wire is selected and tested for stuck-at fault. If no test is possible, then the wire is redundant and can be removed. Otherwise, the mandatory assignments (those assignments that are required for a test to exist) obtained during test generation suggest the additions that will force the tested wire to become redundant. However, it is not known whether these additions can be performed without changing the network functionality. This must be further verified by performing additional tests. In this paper we propose an efficient technique that allows to identify which

EURO-DAC '96 with EURO-VHDL '96
0-89791-848-7/96 $4.00

connections/gates can be certainly added to the input of an existing gate in the network with a single test run. This technique is also extended to multiple wire addition, allowing to identify a bigger set of logic optimization transforms than with previous approaches.

RAR has been applied in the past to area optimization of combinational and sequential circuits. It is also well suited for optimization at the technology-dependent level [3]. In this work, we apply this technique to timing optimization of combinational logic networks. Logic restructuring techniques for timing optimization have been proposed based on other optimization methods [10], [11]. These techniques are commonly used in combination with other timing synthesis techniques [9]. We show that the improved Redundancy Addition and Removal technique proposed produces significant timing optimization with very little area cost.

This paper is organized as follows. Section 2 gives some definitions for the rest of the paper. Section 3 describes the improved transform identification technique. Section 4 describes how this technique is extended with the addition of redundant gates. Section 5 describes the timing optimization algorithm based on this approach. Section 6 presents the experimental results. Finally, section 7 presents the conclusions of this work.

2. Notation and definitions

We note a connection as a triple (S, D, P), where S is the source node, D is the destination node and P is the polarity (1 for inverted and 0 for non-inverted). An input to a gate G has a *controlling value Cont(G)* if this value determines the output of the gate G regardless of the other inputs. The controlling value of an AND(OR) gate is 0(1). The inverse of the controlling value is called the *sensitizing value Sens(G)*. The sensitizing value of an AND(OR) gate is 1(0).

Mandatory assignments are unique values that are required at certain nodes for a test to exist. If the mandatory assignments for a stuck-at fault on a connection cannot be consistently justified, the stuck-at fault is untestable and the connection is redundant. When using recursive learning to compute the set of mandatory assignments (SMA), we call *Extended Set of Mandatory Assignments (ESMA)* a set of mandatory assignments that is not restricted to recursion depth 0. A set of branches $p_1, ..., p_n$ of the recursion tree is called *supplementary* if the union of them gives the entire ESMA.

3. Improved Transform Identification

The techniques developed so far to identify logic optimization transforms by redundancy addition and removal [1-7] can be called "one-way", because the redundancy test of a fault allows to identify candidate connections for addition, but it is not known whether they

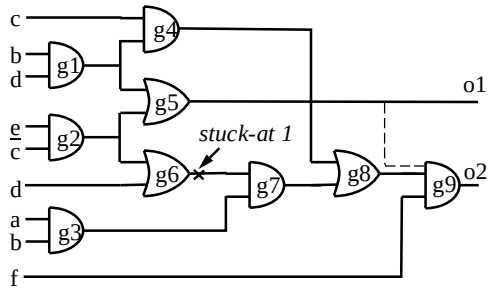

Figure 2. An irredundant circuit

can be certainly added without changing the circuit functionality. Consider for example the circuit shown in Fig. 2 and the fault *g6* stuck-at 1. When this fault is tested, a mandatory assignment *g5* = 0 is obtained; in other words, all input vectors that are able to test this fault put a logic value 0 at the output of *g5*. This result suggests the addition of a new connection from *g5* as a new input of *g9* (dashed line in Fig. 2), because when this wire is added, it blocks the propagation of the fault *g6* stuck-at 1, which thus becomes redundant. However, we do not know if this new connection can be added without changing the circuit functionality, i.e., if it is a redundant connection. In order to check this, an additional redundancy test is required.

In this section, we will describe how the connections that can be certainly added to a destination node can be identified with a single redundancy test. The optimization transforms are obtained by comparing the results of this redundancy test with the SMA of the removal candidate. We call this technique "two-way" transform identification.

Note that all connection faults that have the same destination node have the same mandatory observation assignments and only differ in the mandatory control assignment. Let D be a node and c be the controlling value of D. Suppose that we perform the implication of the mandatory observation assignments that are common to all connection faults that have the same destination node D. If a mandatory assignment *v* is obtained this way in a node N, such that *v* = *c*', then the connection C = (N, D, 0) is redundant and therefore it can be added without changing the circuit's functionality. The demonstration of this statement is simple: the fault associated to the added connection is C stuck-at *c*' and the mandatory control assignment for this fault is N = *c*, which is incompatible with the previous assignment N = *v*. Analogously, if a mandatory assignment *v* is obtained in a node N, such that *v* = *c*, then the connection C = (N, D, 1) is redundant and therefore it can be added without changing the circuit's functionality.

Example. Consider again the example in Fig. 2. The mandatory observation assignments for all connection faults whose destination node is *g9* are: *g8* = 1, *f* = 1. By implication, using recursive learning [8], we get the

mandatory assignments $b = 1$, $g6 = 1$ and $g5 = 1$. Since Cont($g9$) = 0, we obtain that the connections (b, $g9$, 0), ($g6$, $g9$, 0) and ($g5$, $g9$, 0), can be added without changing the circuit's functionality. It can be easily verified that their associated stuck-at faults are not testable because the mandatory control assignment is inconsistent with the mandatory observation assignments obtained previously for these connections.

The subset of mandatory assignments that are common to all candidate connection faults that have the same destination node D coincide with the SMA for the fault D stuck-at Cont(D). We will call the test for the fault D stuck-at Cont(D) the *redundancy test of the destination node*. The optimization transforms are easily identified by comparing the mandatory assignments that are obtained in each node of the network by the redundancy test of the destination node and the redundancy test for each fault dominated by the destination node. If a node N has mandatory assignments of different value for each of these tests, then it is possible to add one connection from N to the destination node to eliminate at least one connection. Thus, for the previous example, we have the following mandatory assignments: (i) $g5 = 1$ obtained from the redundancy test of the destination node (fault $g9$ stuck-at 0); (ii) $g5 = 0$ obtained from the redundancy test of the target wire (fault $g6$ stuck-at 1). Then, the addition of connection ($g5$, $g9$, 0) allows to eliminate connection ($g6$, $g7$, 0).

The two-way algorithm is able to identify all the transforms that can be identified with the one-way algorithm as long as all mandatory assignments can be found. However, this is only possible if recursive learning is used. Generally, it is not possible to know beforehand what is the maximum recursion depth to obtain equivalent results to one-way identification. For instance, the transform of the previous example can be identified using a one-way algorithm with a maximum recursion depth equal to 0, while a two-way algorithm requires at least a recursion depth equal to 2.

4. Adding Gates

In this section, we explain how the two-way identification technique can be generalized to the addition of gates. Note that a node may have a mandatory assignment even though its predecessors and successors do not. This is because the higher level mandatory assignments do only propagate to the next inferior level when they do all coincide. The key to identify more general transforms is the information contained in mandatory assignments of a level higher than 0. The following example illustrates this concept.

Example. Consider the example in Fig. 3. This circuit is the same of Fig. 1, except for the absence of node $g5$.

The redundancy test of the destination node stuck-at 0

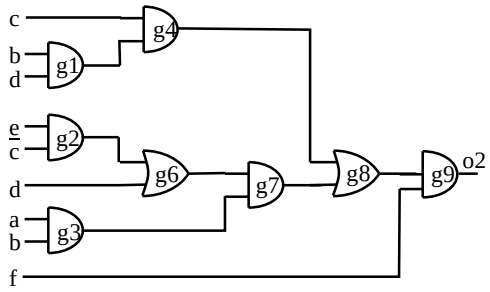

Figure 3. Addition of a gate

is shown below. The recursivity level is indicated by the indentation.

g9 = 1 => g8 = 1, f = 1
g8 = 1 => *Recursivity level = 1*
Justification of g4 = 1
g4 = 1 => c = 1, g1 = 1
g1 = 1 *=> b = 1, d = 1*
c = 1 => g2 = 0
d = 1 => g6 = 1
Justification of g7 =1
g7 = 1 => g6 = 1, g3 = 1
g3 = 1 => a = 1, b = 1
g6 = 1 => *Recursivity level = 2*
Justification of g2 = 1
g2 = 1 *=> e = 1, c = 0*
c = 0 => g4 = 0
Justification of d = 1
d = 1, b = 1 => ***g1 = 1***
b = 1 <=
g6 = 1 <=

As it can be observed, there is not a level-0 mandatory assignment neither in $g1$ nor in $g2$. However, all branches in the recursivity tree have a mandatory assignment $g1 = 1$ or $g2 = 1$. Hence, consider a gate such that its output is 1 when $g1 = 1$ or $g2 = 1$. Such a gate is an OR gate whose inputs are $g1$ and $g2$. This gate is redundant and can be added to the circuit without changing its functionality, since it shows a level-0 mandatory assignment in the redundancy test of the destination node. The following theorem characterizes the addition of a two-input gate.

Theorem. Let A, B, D nodes in a network. Let T be a transform that involves making a fault f untestable by the addition of an elementary two-input gate G with one output. The inputs of G are connected to nodes A and B, and the output of G is connected to the destination node D. Let $ESMA_f$ and $ESMA_D$ be the extended sets of mandatory assignments corresponding to fault f and to the redundancy test of the destination node, respectively. Let v(A, 0) and v(B, 0) the level-0 mandatory assignments at A and B in one of these extended sets, and v'(A, p_1), ..., v'(A, p_n), v'(B, p_1), ..., v'(B, p_n) the mandatory assignments at A

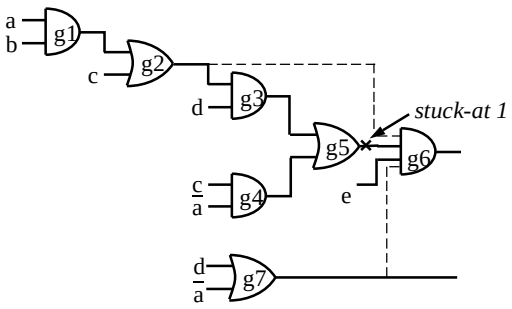

Figure 4. Example circuit

and B for a set of supplementary branches $p_1, ..., p_n$ of the other set, i.e., either

$v(A, 0), v(B, 0) \in SMA_f$
$v'(A, p_i) \in ESMA_D(p_i)$ $\quad i = 1, ..., n$
$v'(B, p_i) \in ESMA_D(p_i)$ $\quad i = 1, ..., n$
$SMA_D = ESMA_D(p_1) \cap ... \cap ESMA_D(p_n)$

or

$v'(A, p_i) \in ESMA_f(p_i)$ $\quad i = 1, ..., n$
$v'(B, p_i) \in ESMA_f(p_i)$ $\quad i = 1, ..., n$
$SMA_f = ESMA_f(p_1) \cap ... \cap ESMA_f(p_n)$
$v(A, 0), v(B, 0) \in SMA_D$

A sufficient condition to guarantee that this transform does not change the circuit's functionality is

$\forall p_i, i = 1, ..., n \quad v(A, 0) \neq v'(A, p_i)$ or $v(B, 0) \neq v'(B, p_i)$

Proof. The theorem will be demonstrated for the first set of premises. The demonstration is analogous for the second set. The demonstration is constructive, i.e., it determines exactly the types of the gate and the connections of the transform T. Without loss of generality, we will take G ∈ {AND, OR}. The connections between A, B and G are of the following types

(A, G, 0) if $v(A, 0) = Sens(G)$
(A, G, 1) if $v(A, 0) = Cont(G)$
(B, G, 0) if $v(B, 0) = Sens(G)$
(B, G, 1) if $v(B, 0) = Cont(G)$

With this connections, we have $v(G, 0) = Sens(G)$. From the sufficient condition of the theorem, at least one of the G inputs has a controlling value for each p_i, $i = 1, ..., n$. Therefore

$v'(G, p_i) = Cont(G)$ $\quad \forall p_i, i = 1, ..., n$

Since the value of G is the same at all supplementary branches of the recursivity tree, we have

$v'(G, 0) = v'(G, p_i) \cap ... \cap v'(G, p_n) = Cont(G) \neq Sens(G)$

This demonstrates that the value of G is different for each of the sets of mandatory assignments SMA_f and SMA_D. Therefore T is a valid transformation. The polarity of the connection from G to the destination node D depends on the controlling value of D. The polarity is not inverted if $v_f(G) = Cont(D)$, and inverted otherwise.

Example. Let's take the previous example shown in Fig. 3. We have the mandatory assignments *g1* = 0 and *g2* = 0 for the fault *g6* stuck-at 1. Selecting G as and OR gate, we have

$v(g1, 0) = Sens(G) = 0$
$v(g2, 0) = Sens(G) = 0$

and the connections from *g1* to G and *g2* to G are non-inverted. If we perform such connections, we have $v(G) = 0$.

The redundancy test of the destination node was shown in a previous example. It was demonstrated that there is a mandatory assignment *g1* = 1 or *g2* = 1 in all branches of the recursivity tree for this test. Therefore, $v'(G) = 1$. Since $v(G) \neq v'(G)$, the addition of G allows to make fault *f* redundant. To complete the transform it is necessary to determine the polarity of connection G to *g9*. This connection is non-inverted, since $v_f(G) = Cont(g9)$.

In some particular cases it is possible to identify multiple-wire addition transforms without considering recursive learning [4]. However, this cannot be generalized. For instance, consider the interesting example shown in Fig. 4.The redundancy test of the target fault *g5* stuck-at 1 does not suggest any interesting candidate connection. Hovewer, by performing the redundancy test of the destination node *g6* stuck-at 0, we find that the connections (*g2*, *g6*, 0) and (*g7*, *g6*, 0) can be added without changing the network functionality.

If we compare the SMA of the destination node test with the SMA of the target fault, the two-way transform condition is not met because there is not a level-0 mandatory assignment neither in *g2* nor in *g7*. However, all branches in the recursivity tree have a mandatory assignment *g2* = 0 or *g7* = 0 for the fault shown. Hence, consider a gate such that its output is 0 when *g2* = 0 or *g7* = 0. Such a gate is an AND gate whose inputs are *g2* and *g7*. In other words, the addition of (*g2*, *g6*, 0) or (*g7*, *g6*, 0) separately does not cause the target fault to be redundant, but the addition of both of them at the same time does. Note that this transform cannot be identified with previous approaches.

5. Timing Optimization

Redundancy addition and removal techniques can be applied to timing optimization [1]. If the addition of redundant wires/gates to a circuit causes some connection(s) in the critical path to be redundant, then the transform will result in a faster circuit.

For timing optimization, the selection of optimization transforms is performed according to the following criteria (see Fig. 5):

- The destination node of the connection/gate added belongs to the critical path. Although this is not strictly required to create redundancies in the critical path, it is very unlikely that redundancies may be created in the critical path otherwise and the search would require a

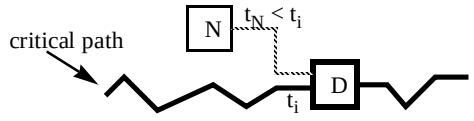

Figure 5. Timing optimization approach

much larger computational effort.

- The arrival time of the output of the connection/gate added (t_N) must not be greater than the arrival times of the inputs of the destination node (t_i) plus the delay increase produced by adding an extra input to the destination node. In this way, the timing of the network is not degraded by this addition.

The selection of a transform among the possible transform candidates is based in a cost function that estimates the delay reduction. The cost of a connection/ gate is defined as the circuit delay reduction that can be obtained by removing/adding the connection/gate. From this definition, the cost of a critical connection can be obtained as the delay difference between the fault-free and faulty circuits associated to the connection. The cost of a non-critical connection is 0. Also the cost of adding a connection following the above mentioned criteria is 0.

Note that when a connection is added to remove at least another connection in the critical path, the area of the circuit does not increase. Therefore, with this type of transforms the length of the critical path can be reduced without augmenting the circuit area. If a gate is added then the area will be augmented.

The timing optimization algorithm follows a greedy approach based on the cost function. We focus on critical path segments in order to apply the transform identification techniques presented in the previous sections. We define a critical path segment (CPS) as the portion of the critical path between two multiple fanout nodes. The timing optimization algorithm iterates over the critical path segments in decreasing order of its cost function. The cost associated to each segment and each connection is computed at the time the critical path is identified.

For each CPS we try to add a connection/gate as a new input of a node in the CPS to eliminate at least a connection in the critical path. If a transform is found, the algorithm starts again by identifying the new critical path and the critical path segment with the highest cost.

We consider three types of transforms: (1) adding one connection to eliminate at least 1 connection/gate in the critical path (area cost 0); (2) adding 2 connections to eliminate at least 1 connection/gate in the critical path; and (3) adding a 2-input gate to eliminate at least 1 connection/gate in the critical path. These transforms are applied successively in the given order.

6. Experimental Results

In this section we will present experimental results of timing optimization with the improved Redundancy Addition and Removal technique. The optimization was made before mapping for simplicity. However, the redundancy addition and removal technique can be similarly applied after technology mapping [12], where more precise estimations of area and delay are available.

In the experiments carried out we used a unit delay model with a fanout factor of 0.2. The critical path was determined by static timing analysis.

Table 1 shows the results for the benchmark circuits. The initial circuits were obtained after strong area optimization with RAMBO [2] and SIS. The final results where obtained with the proposed algorithm using a maximum recursion depth of 3. For each example the number of connections (#C), the number of nodes (#N), the estimated delay before mapping (D) , the delay after mapping (Dm) and the CPU time consumption in a Sun Sparcstation 2 (T) are presented. The examples were mapped to the example library *example.genlib* with the default SIS command *map -n 1 -AFG*.

The timing improvement is significant in most of the examples. The greatest speed-ups are obtained for *frg2* (3/ 1.9 before/after mapping), *k2* (2.4/2.7) and *vda* (2.5/2.2). The discrepancies in the figures before mapping and after mapping are due to the difference between the simple delay model used and the delay of the library cells. The average speed-up factor was 1.4 before mapping and 1.38 after mapping.

Note that the area (estimated by the number of nodes and connections) is not degraded by applying one connection addition transforms. With more complex transforms, the area may be degraded as we add more connections/gates than we remove. However, except in one case (*vda*), the estimated area increase is less than 1.5%. With this same exception, the final area increase after technology mapping (not shown in the table) falls between a range of -3.5% to +4%, which can be considered as a negligeable variation introduced by the technology mapping process. This result shows that the proposed technique is able to keep area increase under control.

There is heavy CPU time consumption in some cases. This is because when a transform is found, the critical path must be recomputed and the search for new transforms must start all over again. The CPU time consumption will be greatly reduced in a future implementation by reusing previous computations and eliminating repeated runs over already optimized critical path segments.

7. Conclusions And Future Work

We have proposed an improved transform identification technique for Redundancy Addition and

Removal. This technique allows to efficiently identify the set of redundant connections that can be added to a logic network without changing its functionality by reducing dramatically the number of test generation runs required. This technique has also been extended for the addition of gates, obtaining new transformations that could not be identified with previous approaches.

Previous work in Redundancy Addition and Removal only considered area optimization. In this paper, we have extended this technique to timing optimization and obtained promising results. These results show that this technique allows to reduce circuit delay significantly with very little area increase and therefore is well suited for timing optimization when there is not much room to trade area for speed. In the future, we plan to experiment with more accurate delay models and to extend the improved RAR technique with the addition of several gates at a time, in order to obtain additional timing optimization, although at bigger area cost.

References

[1] K.-T. Cheng, L. A. Entrena. "Multi-Level Logic Optimization by Redundancy Addition and Removal". Proc. EDAC-93, p. 373-377. Feb., 1993.

[2] L. A. Entrena, K.-T. Cheng. "Sequential Logic Optimization by Redundancy Addition and Removal". Proc. ICCAD-93, p. 310-315. Nov., 1993.

[3] S. C. Chang, K.-T. Cheng, N.-S. Woo, M. Marek-Sadowska. "Layout Driven Logic Synthesis for FPGAs". Proc. 30th DAC, p. 308-313. June, 1994.

[4] S. C. Chang, M. Marek-Sadowska. "Perturb and Simplify: Multi-level Boolean Network Optimizer". Proc. ICCAD-94, p. 2-5. Nov., 1994.

[5] W. Kunz, P. R. Menon. "Multi-level Logic Optimization by Implication Analysis". Proc. ICCAD-94, pp. 6-13. Nov., 1994.

[6] L. A. Entrena, K.-T. Cheng. "Combinational and Sequential Logic Optimization by Redundancy Addition and Removal". IEEE Transactions on CAD, vol.14, n. 7, p. 909-916. July, 1995.

[7] U. Gläser, K.-T. Cheng. "Logic Optimization by an Improved Sequential Redundancy Addition and Removal Technique". Proc. ASP-DAC. Sept., 1995

[8] W. Kunz, D. K. Pradhan. "Recursive Learning: an attractive alternative to the decision tree for test generation in digital circuits". Proc. ITC, p. 816-825. October 1992.

[9] J. P. Fishburn. "LATTIS: An Iterative Speedup Heuristic for Mapped Logic". Proc. 29th DAC, p. 488-491. June 1992.

[10] K. J. Singh, A. R. Wang, R. K. Brayton, A. Sangiovanni-Vincentelli. "Timing Optimization of Combinational Logic". Proc. ICCAD-88, p.282-285, Nov. 1988.

[11] K. C. Chen y S. Muroga. "Timing Optimization for Multi-Level Combinational Networks". Proc. 27th DAC, p. 339-344. June 1990.

[12] B. Rohfleisch, B. Wurth, K. Antreich. "Logic Clause Analysis for Delay Optimization". Proc. 32nd DAC, p. 668-672. June 1995.

Table 1: Experimental results

Name	Initial				Final				
	#N	#C	D	Dm	#N	#C	D	Dm	T
C1355	376	798	31.0	37.9	376	802	29.6	36.8	187.4
C1908	316	712	44.8	56.1	316	718	37.0	52.1	695.5
C2670	599	1316	46.2	50.4	596	1320	33.0	37.6	1476.3
C432	84	221	33.8	45.8	83	221	26.8	44.8	65.9
C499	376	798	31.0	37.1	376	802	29.6	36.0	182.8
C5315	1217	2671	62.6	66.5	1208	2673	44.6	53.2	2609.6
C6288	1871	3787	131.0	167.7	1871	3787	126.0	166.0	1009.4
C7552	1433	2992	217.0	207.7	1424	2975	159.4	119.3	2826.5
C880	246	603	49.4	66.4	274	613	39.8	41.7	261.8
alu4	372	910	61.0	72.1	370	906	50.4	62.0	1979.9
apex6	436	1096	29.8	37.5	437	1103	19.0	26.1	35.9
apex7	141	354	24.4	30.4	141	358	18.2	21.5	20.6
dalu	439	1073	45.4	53.4	443	1087	24.8	33.2	901.5
frg2	460	1145	62.0	44.8	458	1156	21.0	23.2	110.8
k2	381	1162	65.0	110.9	381	1161	26.6	40.5	4903.5
pair	936	2335	70.4	79.5	929	2335	39.4	50.3	656.5
rot	385	953	33.8	35.4	385	953	32.6	32.8	50.4
term1	85	206	16.0	22.0	83	208	13.0	20.0	27.6
ttt2	100	254	25.6	33.0	99	253	20.6	26.9	28.0
vda	186	645	52.2	87.0	187	684	21.2	39.0	4541.0
x1	147	414	11.6	15.2	150	424	9.2	13.1	6.9
x3	455	1122	20.0	23.9	456	1134	14.2	20.7	69.0
x4	221	571	28.0	22.0	220	570	12.2	15.8	6.7
9symml	110	282	20.0	24.7	109	280	14.2	19.6	4091.5
TOTAL	11372	26420	1212.0	1427.4	11372	26523	862.4	1032.2	
					1.00	1.00	1.40	1.38	