An Experimental Evaluation of Algorithms for Computation of Internal Don't-Cares in Boolean Networks

Alan Mishchenko
Department of Electrical and Computer Engineering
Portland State University
September 28, 2001

Abstract

The goal of this study is to evaluate the efficiency of don't-care computation in boolean networks using several algorithms. The algorithms are compared using the following three parameters: (1) the amount of don't-cares generated for the nodes of the network; (2) the sum total of literal counts in the factored forms of all nodes after the generated don't-cares have been used to simplify each node in the boolean network; and (3) the runtime requirements.

Methods Considered

RR – redundancy removal consists in removing internal signals that are functionally equivalent (up to complementation). As a result, some of the remaining nodes may have the fanout count equal to zero. These nodes are also removed in the final sweep over the network. Redundancy removal is based on the completely-specified global functions of each node in the network (that is functions expressed in terms of the primary inputs). This algorithm does not use don't-cares. In this survey, it is included for completeness.

SDC – satisfiability don't-cares arise because, due to the structure of the boolean network, some of the combinations of local inputs of nodes never occur under any combinations of primary input variables.

CODC – compatible observability don't-cares computed using the Savoj formula [1] in terms of primary inputs and then imaged into the local space of a node. The local don't-cares derived using this method contain both CODCs and SDCs.

CODCm – a simplified CODC computation method, which consists in computing only universal quantifiers in the Savoj formula, without adding the boolean difference w.r.t. each edge variable preceding the given variable in the order.

Comp – a complete observability don't-care computation algorithm computed by permuter insertion. The permuter is an additional boolean variable replacing the output of the node, for which the don't-cares are being computed. After inserting the permuter, global BDDs are derived for all the primary outputs in terms of the primary inputs and the additional variable. The complete observability don't-cares for the node is equal to

the product of unobservability conditions of the primary outputs (the unobservability conditions are the complemented boolean differences of primary output functions w.r.t. the permuter variable). This method have been considered infeasible for relatively large networks, because it requires iterative recomputation of global functions after moving the permute into a new location. However, due to the recent advances in the BDD computation (in particular, the efficient subsetting/supersetting algorithms implemented in CUDD 2.3.1), it was possible to run this algorithms in reasonable time for all considered benchmarks.

External don't-cares have not been provided for the given benchmark circuits.

There was an attempt to implement the improved procedure for CODC computation using the formula proposed by Brayton in [2]. One of the parameters in this formula is the previously computed sets of CODCs expressed in terms of local inputs of the fanout node. However, in the current implementation of CODC computation, all CODCs are stored in terms of the primary inputs. The problem is, how to remap the CODCs from the primary input space into the local space of a node? The image computation cannot be used, because by computing the image, we derive the set of CODCs+SDCs, while we need only CODCs. An attempt was made to compute SDCs independently and then subtract them from CODCs+SDCs resulting from the image computation. However, in the process a large part of CODCs is lost, in particular that part, which overlaps with SDCs. A better method of mapping CODCs into the local space of a node is needed in order to use the approach presented in [2].

Unfortunately, it is not possible to compare the results of network simplification with the results using the state-of-the-art computation proposed in [3], because [3] lists the results for benchmarks after a complete logic synthesis flow. Therefore, the comparison so far is done only with SIS command **full_simplify** applied to the benchmark circuits without preprocessing and with redundancy removal.

Benchmark Statistics

Benchmark name	Ins	Outs	Nodes	Fanin		Fanout		Shared BDD	
				Ave	Max	Ave	Max	POs	All
alu2	10	6	59	5.2	33	3.3	9	231	441
alu4	14	8	112	5.3	36	3.6	11	503	2136
dalu	75	16	1131	2.7	4	1.9	112	863	3282
des	256	245	681	7.1	34	5.7	184	3036	4758
frg2	143	139	522	3.6	7	1.8	87	1095	2070
i10	257	224	2497	2.2	16	1.9	39	44090	65806
k2	45	45	225	12.6	188	5.5	13	1273	2239
pair	173	137	824	2.4	7	1.5	22	3678	6073
t481	16	1	2072	3.3	4	1.8	151	21	3204
C1355	41	32	514	2.0	5	1.7	12	29562	109839
C1908	33	25	880	1.7	8	1.6	16	6253	20869
C2670	233	140	1153	1.8	5	1.6	11	3269	7338
C432	36	7	160	2.1	9	1.6	9	1210	4696
C499	41	32	202	2.0	5	1.6	12	27224	38823
C5315	178	123	2290	1.9	9	1.7	15	2184	12671
C7552	207	108	3463	1.8	5	1.7	15	9302	40632
C880	60	26	357	2.0	4	1.4	8	8669	11677

Notation:

Ins, Outs – are the number of primary inputs and outputs.
Nodes – the number of internal nodes in the boolean network.
Fanin – refers to the local inputs of the node.
Fanout – refers to the fanout count (the number of times the node's output in used as an input to other nodes).
Ave – the average value over all internal nodes of the network (excluding PIs and POs).
Max – the maximum value over all the internal nodes.
Shared BDD – the size of the shared ROBDD after reordering using symmetric sifting.
POs is the size of the shared ROBDD for the primary outputs after reordering.
All is the size of the shared ROBDD for all "global functions" in the network, that is, for all intermediate signals expressed in terms of the primary inputs.

Intenal Don't-Cares

Name	ODCs in the PI space, %				ODCs+SDCs in the local space, %			
	SDC	CODC	CODCm	Comp	SDC	CODC	CODCm	Comp
alu2	0.00	21.33	42.23	68.32	21.32	27.63	28.09	31.27
alu4	0.00	19.04	44.21	71.44	20.19	22.03	23.62	25.20
dalu	0.00	83.38	83.40	84.04	19.68	30.90	30.82	30.98
des	0.00	1.16	4.68	11.79	13.95	14.89	18.42	25.67
frg2	0.00	44.98	42.25	53.78	10.99	18.75	18.60	22.95
i10	0.00	21.13	18.89	44.46	16.11	19.42	17.93	21.54
k2	0.00	0.47	2.20	3.48	13.88	14.33	15.10	15.42
pair	0.00	41.99	35.62	57.26	3.22	4.35	4.05	4.58
t481	0.00	61.15	19.72	70.42	7.66	8.61	8.55	9.66
C1355	0.00	11.60	8.02	58.10	16.75	20.54	20.54	22.44
C1908	0.00	21.98	25.33	30.77	6.79	14.82	10.80	17.99
C2670	0.00	36.82	30.17	55.15	7.83	15.95	12.21	16.94
C432	0.00	32.42	37.95	72.22	3.81	8.16	10.28	10.99
C499	0.00	0.00	0.00	42.77	0.85	0.85	0.85	2.08
C5315	0.00	38.78	34.44	43.19	9.81	16.26	13.41	16.66
C7552	0.00	30.52	26.64	57.01	9.36	18.64	14.62	20.75
C880	0.00	16.54	15.59	42.26	3.72	7.43	5.74	8.95
Average	0.00	26.85	26.19	48.14	10.33	14.64	14.09	16.89
Ratio,%	0.0	55.8	54.4	100.0	61.2	86.7	83.4	100.0

Comments:

Two types of don't-cares have been measured: (1) the observability don't-cares only (left), and (2) the complete don't-cares (ODCs and SDCs) (right).

The relative amount of don't-cares depends on the algorithm applied. In the left side of the table, algorithm **SDC** does not compute observability don't-cares (therefore there are zeros in second column). **CODC** computes compatible don't-cares using the Savoj formula. **CODCm** computes a subset of these don't-cares using a simplified Savoj formula. **Comp** computes the complete ODCs.

The right side of the table shows the result of imaging the ODCs computed by each algorithm into the local space of the node. This measurement contains both the ODCs from the left part of the table and the satisfiability don't-cares (SDCs) added during the image computation.

The measurements in the table are the average (over all internal nodes of the network) of the percentage of the primary input space (the left side) and the local input space (the right side) occupied by the respective don't-care set.

Because the boolean spaces used in the measurements of the left and the right side of the table are different, the numbers in the left side may be less, even though they stand for a larger set of don't-cares.

Comparison of Literal Count in Factored Form after Simplification

Name	Original	RR	SDC	CODC	CODCm	CompOI	CompIO	SIS	RR + SIS
alu2	453	451	385	364	360	345	323	374	374
alu4	855	855	800	791	781	754	725	timeout	743
Dalu	3067	2227	2105	1817	1821	1803	1826	2331	1876
Des	6101	6101	6071	6060	6036	5789	5977	5677	5734
frg2	2010	1908	1793	1600	1601	1519	1756	1522	1473
i10	5376	4035	3791	3748	3757	3677	3668	spaceout	spaceout
k2	2928	2928	2891	2891	2859	2771	2696	2889	2889
Pair	2420	2388	2273	2249	2253	2237	2242	2203	2188
t481	6823	3005	3005	3005	3005	3005	3005	timeout	2424
C1355	1032	992	992	992	992	984	984	1024	984
C1908	1497	759	759	759	759	754	750	1406	751
C2670	2035	1239	1238	1202	1214	1162	1164	spaceout	spaceout
C432	372	335	299	289	289	288	288	335	289
C499	616	576	576	576	576	568	568	608	568
C5315	4369	3062	3062	2997	3016	2996	2992	4187	2951
C7552	6095	3777	3777	3677	3688	3532	3531	spaceout	spaceout
C880	703	633	633	633	633	633	633	687	625
Total	46752	35271	34450	33650	33640	32817	33128	42501*	37854*
Ratio,%	100.0	75.4	73.6	72.0	71.9	70.1	70.9	90.9*	81.0*

Comments:

Even though **CODCm** computes a subset of don't-cared computed by **CODC**, the resulting simplification of **CODCm** is more substantial for some benchmarks. The reason is that, in the current implementation, to ensure fast processing of BDDs of don't-care sets, subsetting is used to simplify the BDD in each iteration. (If the subsetting techniques are not applied, the computation does not complete in reasonable time for benchmarks, which contain nodes with large fanin/fanout count, such as "des.blif", "k2.blif", etc.) Because **CODC** uses the complete Savoj formula, it requires as many iterations as there are fanins in the fanout node. In each iteration, the don't-cares are filtered by subsetting. Meanwhile, **CODCm** does not iterate over the fanins. It computes the universal quantifier followed by a single call to a subsetting routine. As a result, the don't-care set derived by **CODCm** may, in fact, contain more don't-cares than that derived by **CODC**.

Two measurements for complete ODC computation demonstrate that the degree of simplification depends on the order, in which the nodes are considered; **CompOI** stands for topological order of the nodes from outputs to inputs, and **CompIO** vice versa.

The implementation of **full_simplify** in SIS is based on a compatible ODC computation algorithm, yet in some cases it get a better result than the complete don't-care computation. There are two possible reasons: (a) having a good order and a partial set of don't-cares may be better than having a bad order and a complete set; (b) the implementation of **full_simplify** in SIS, in addition to simplifying each node using don't-cares, also performs boolean resubstitution. (This is obviously true from the results for benchmarks "t481.blif" and "c880.blif".) (Symbol "*" stands for a normalized result.)

Runtime

Name	Read	RR	Image	SOP	SDC	CODC	CODm	ComOI	ComIO	SIS	RR+SIS
alu2	0.01	0.00	0.21	2.71	2.92	4.53	3.28	2.86	2.39	1.89	2.02
alu4	0.53	0.00	0.49	6.08	6.62	14.86	7.51	7.96	9.05	time	2.86
dalu	1.04	0.00	0.07	0.31	0.47	0.99	0.67	6.13	5.12	145.85	72.17
des	1.93	0.00	0.49	6.13	6.90	28.90	9.18	12.21	16.63	16.19	16.29
frg2	0.55	0.00	0.02	0.46	0.55	1.07	0.70	4.63	4.87	9.51	8.22
i10	20.79	0.02	0.46	2.31	3.14	12.92	6.79	328.26	353.15	space	space
k2	2.01	0.00	2.63	74.44	77.19	105.43	79.51	80.14	125.04	14.22	14.22
pair	2.59	0.00	0.05	0.71	0.83	1.66	1.10	3.36	3.58	6.55	6.34
t481	0.20	0.02	0.06	0.17	0.26	2.80	1.58	0.96	1.07	time	122.19
C1355	28.81	0.09	0.25	0.12	0.43	1.80	0.92	96.86	95.74	56.52	44.15
C1908	1.39	0.01	0.10	0.06	0.18	1.23	0.45	15.01	15.07	68.13	14.99
C2670	4.13	0.01	0.04	0.66	0.83	2.04	1.46	2.89	3.01	space	space
C432	0.57	0.01	0.05	0.04	0.09	0.57	0.32	1.52	1.69	4.60	3.76
C499	3.76	0.00	0.06	0.08	0.16	0.70	0.25	15.74	16.00	2.67	2.10
C5315	1.48	0.02	0.12	1.10	1.43	4.48	2.69	16.38	15.27	69.75	55.23
C7552	44.09	0.03	0.31	1.88	2.40	7.61	5.29	37.76	39.54	space	space
C880	0.94	0.00	0.02	0.10	0.13	0.70	0.40	17.69	16.64	2.34	1.91
Total	114.82	0.21	5.43	97.36	104.53	192.29	122.1	650.36	723.86		
Rat.%					14.4	26.6	16.9	89.8	100.0		

Comments:

The runtime of all algorithms is in seconds on a 933MHz Pentium III with 512Mb RAM under MS Windows 2000. For all but the largest benchmarks, the programs used only a small fraction of memory. Time limit has been set to 10 minutes. SIS reported the space out in the following manner: “The BDD's for this circuit have more than 480,000 nodes. **full_simplify** is not performed”.

The column **Read** shows the time needed to read the benchmark from file and reorder variables using symmetric sifting. The reading time is not included in the runtime of the algorithm (columns **RR**, **SDC**, **CODC**, **CODCm**, **ComOI**, **ComIO**).

In addition to the redundancy removal step, which was used as a preprocessing step for all the algorithms including SIS (column **RR+SIS**), the table also shows the runtime of two important steps in the network simplification: image computation (column **Image**) and two-level SOP minimization (column **SOP**). These steps are the same for all the described algorithms (**SDC**, **CODC**, **CODCm**, **ComOI**, **ComIO**).

For the image computation, the modified algorithm [4] has been used. This algorithm employs the partitioned transition relation with quantification scheduling.

The two-level minimization is performed by a call to the ESPRESSO, through a BDD/ZDD interface available in EXTRA library [5]. CUDD 2.3.1. package [6] has been used in all the experiments.

References

[1] H. Savoj. R. K. Brayton. “The use of observability and external don’t-cares for the simplification of multi-level networks”. Proc. of DAC’ 90. pp. 297-301.

[2] R. K. Brayton. “Compatible Observability Don’t-Cares Revisited”. Proc. of IWLS’01. pp. 121-126.

[3] H. Savoj. “Improvements in Technology Independent Optimization of Logic Circuits”. Proc. of IWLS’97.

[4] D. Geist, I. Beer. “Efficient Mode Checking by Automated Ordering of Transition Relation Partitions”. Proc. of CAV, LNCS-818, pp. 52-71. Springer-Verlag, 1994.

[5] http://www.ece.pdx.edu/~alanmi/research/extra.htm

[6] ftp://vlsi.colorado.edu/pub/cudd-2.3.1.tar.gz