A Don't-Care Based Image Circuit for Function Verification

J. C. Rau, Y. M. Chen*, and S. C. Chang*
Department of Electrical Engineering
Tamkang University, Taipei, Taiwan, R. O. C.
*Department of Computer Science and Information Engineering
National Chung-Cheng University, Chiayi, Taiwan, R. O. C.

Abstract

In this paper, we propose a novel way to build a "DC_image" circuit for the don't cares. The DC_image circuits are concatenated with the inputs of the two circuits under verification. By adding the image circuits, no matter how don't cares are in on-/off- sets, we can directly verify the two circuits with DC_image circuits and claim whether there exists an inconsistency between the original and optimized circuits. Our experimental results show that by the DC_image circuits, the verification process can be sped up tremendously.

1. Introduction

Traditionally, equivalence checking of circuits can be achieved by the method of Automatic Test Pattern Generation (ATPG) [1][2][4][6] or the method of Binary Decision Diagrams (BDD) [3][7]. However, for large circuits, these two methods may suffer from huge CPU run time or memory exploration because the problem of equivalence checking is an NP-complete problem. On the other hand, if two circuits under functional verification are structurally similar, a modern equivalence checker such as AQUILA [5] can verify two large circuits in few seconds. This type of function verification, also called *incremental verification*, attempts to identify equivalent pairs for verification. An observation shows that a high percentage of equivalent pairs can be identified by only considering a small sub-circuit surrounding a candidate signal pair. Therefore, instead of checking two entire circuits, one can verify the equivalence by gradually finding equivalent pairs between two circuits. In this way, the verification process can be sped up.

In this work, we propose a novel way to build a "DC_image" circuit of don't cares to resolve the problem of function verification with don't cares. In our method, the outputs of DC_image circuits are concatenated with the inputs of circuits under verification where the number of outputs for a DC_image circuit is the same as the number of inputs for the verified circuits. Note that for a multiple output circuit, it is possible that some output combinations cannot be generated by all possible input combinations. A DC_image circuit of don't cares is built in the way that any don't care vector cannot be generated by all possible input combinations.

The rest of this paper is organized as follows. In Section 2, we describe the basic concept of DC_image circuits for don't cares. In Section 3, we describe our algorithms to construct the DC_image circuits. We present the experimental results in Section 4 and give the conclusions in Section 5.

2. Basic Concepts of DC_image Circuits

Suppose a circuit C_0 has the don't care set DC and let the optimized circuit be C_1. We say that two circuits C_0 and C_1 are *consistent* if we evaluate each min-term in the care set, the outputs of C_0 and C_1 are the same; otherwise, we say there is an *inconsistency* between C_0 and C_1. In our technique, a

0-7803-7448-7/02/$17.00 ©2002 IEEE

DC_image circuit is added at the inputs of C_0 and C_1 as in Fig. 1. In our definition, a DC_image circuit of a don't care set *DC* must obey the following two rules.

1. No min-terms in the don't care set *DC* can be generated from a DC_image circuit. In other words, for all possible input combinations, there is no such an output y $\in$ *DC* in the DC_image circuit.
2. All other min-terms ***not*** in the don't care set *DC* must be generated by the DC_image circuit.

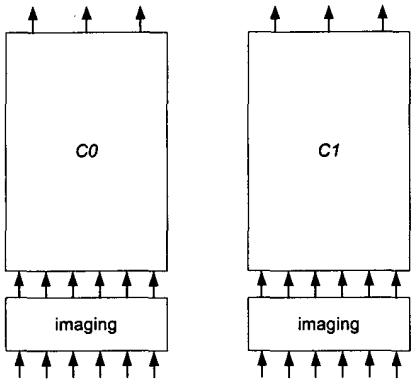


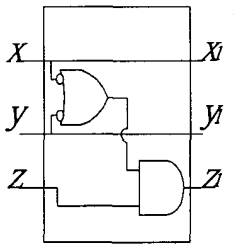
Fig. 1: The construction of a DC_image Circuit

3. Serial Cube Chain DC_image Circuits

In this section, we discuss a novel method that can have a DC_image circuit of small size if the DC set is small. We assume that the DC set is given in the form of DC cubes (SOP). In most of cases, the size of a DC_image circuit constructed in this method is the same as the number of literals in the corresponding DC set. The method basically constructs a circuit module for each DC cube and then concatenates all circuit modules to form a DC_image circuit. Each circuit module "filters" out the corresponding DC cube and when all of them concatenate together, it becomes a DC_image circuit whose output cannot produce all DC cubes in the DC set. We say that a circuit module is a *cube_filter* module of a DC cube if

1. The number of inputs is the same as the number of outputs
2. The module outputs the same vector for each input vector except it is in the DC set

For example, consider a circuit module in Fig. 2(a) where the truth table of the module is also shown in Fig. 2(b). The circuit module has three inputs (x, y, z) and three outputs (x_1, y_1, z_1). From the truth table, one can find that the output vectors are the same as the corresponding input vectors except the input vector (x, y, z) = (*1,1,1*), whose output vector is (x_1, y_1, z_1) = (*1, 1, 0*). The circuit module in this example cannot generate the output of (*1, 1, 1*) but can generate all other 3-variable vectors {(*0,0,0*), (*0,0,1*), (*0,1,0*), (*0,1,1*), (*1,0,0*), (*1,0,1*), (*1,1,0*)}. In addition, except the input vector (1, 1, 1), the circuit module outputs the same vector as the input vector. In this case, we say that the circuit module "filters" out the vector (*1, 1, 1*). The module is called a cube_filter module of cube (*1,1,1*). We also say that the inputs of a module are directly mapped to the outputs except the input vector (*1,1,1*).



(a)

x	y	z	x_1	y_1	z_1
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	0
1	1	1	1	1	0

(b)

Fig. 2: (a) Cube_filter module construction for DC cube *xyz*; (b) The truth table of (a)

Note that the condition of a cube_filter module is stricter than the condition of a DC_image circuit because in a DC_image circuit, we only require to output vectors in the care set and not to output vectors in the don't care set. On the other hand, a cube_filter module of a cube requires the module outputs the same vector as the input vector except the DC cube vectors

There are many different ways to build a cube_filter module of a cube. Our proposed method of constructing a cube_filter module is as follows. Basically, we construct a cube_filter module of a cube c by mapping cube c to another cube c'. Without losing generality, we illustrate the concept by the example in Fig. 2. Let the cube under consideration be $c=xyz$. In the first step, we select a new cube c' to which cube $c=xyz$ is mapped. (The selection process is discussed later.) Let us assume that cube $c'=x_1y_1z_1'$ is chosen. We then construct a circuit as in the Fig. 2(a). In this circuit, when $x=1$ and $y=1$, the output of z_1 is forced to 0 such that the output vector of $(1,1,1)$ cannot be produced, shown in Fig. 2(a). Therefore, the cube vector $c=xyz$ is mapped to $c'= x_1y_1z_1'$ while all other vectors are mapped to the same values.

Finally, a DC_image circuit can be constructed by concatenating all cube_filter modules of all DC cubes together as in Fig.3.

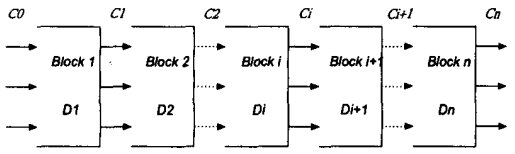


Fig. 3: The structural drawing of serial cube chain DC_image circuit

To avoid the re-generation of filtered cubes, we enforce the following rules. When the k^{th} cube_filter module of cube c_k is constructed, the mapped cube c_k' must not be contained in the sum of cubes in the previous modules; i.e., $c_k' \notin \Sigma c_i$ (where $i=1..k-1$).

The heuristic method of selecting mapped c' for a cube_filter module is described as follows.

1. Simplify and collapse DC to get the minimum SOP.
2. Order the cubes of the minimum SOP by the increasing number of literal count of the cubes.
3. Get a cube c_i, compute the differences between c_i and the cubes that are after c_i.
4. According to the differences, we can build the new cube d_i, which can guarantee the minterms of c_i will not appear on DC_image circuit.
5. Repeat (2) to (4) until all cubes of DC are chosen.
6. From all d_i, we can construct DC_image circuit.

4. Experimental Results

In this section, we present our experimental results in Table 1. All experiments are conducted on ULTRA 2 machines. To verify the correctness of circuits optimized with don't cares, we use the verification package of SIS [8], AQUILA [5], and the concept of *set containment* as a way of comparison. The set containment method is described as follows.

Let the Boolean function under verification be F_0, the don't care set of F_0 be DC and the Boolean function after optimizing F_0 with DC be F_1. We have the following relation.

$$F_0 - DC \subset F_1 \subset F_0 + DC$$

To verify whether there is an inconsistency between F_0 and F_1, we check whether the above containment operations are satisfied. The set containment operations can be achieved by using BDD operations.

In Table 1, the first column shows the names of benchmark circuits. The external don't cares of circuits in Table 1 are randomly generated. All the circuits are then simplified with the don't cares by SIS command "*full_simplify -d*." We use cube_filter modules to construct DC_image circuits. Column 2 shows the time to construct DC_image circuits and to verify using SIS. Column 3 shows the time to construct DC_image circuits and to verify using AQUILA. The last column shows the verification time using set containment operations. For example, when using the

cube_filter DC_image technique, circuit *cse* takes *0.17* seconds by SIS *0.44* seconds by AQUILA., and *0.30* seconds by the set containment technique.

In general, by constructing the DC_image circuit, we have obtained reasonable improvement for the experiments on sequential circuits. However, we are unable to extract the external don't cares for large sequential circuits due to the limitation of SIS command "*xdc*." As a result, the circuits in our experiments are small and cannot demonstrate the full advantages of our approaches. We, on the other hand, have shown significant improvement using the DC_image techniques. Many circuits cannot finish the function verification by the set containment operations while can be finished by the DC_image circuit approaches. Hence, the serially chained cube_filter is better.

	Serially Chained Cube Imaging Circuits		
circuit names	SIS	AQUILA	Sets
cse	0.17	0.44	0.30
dk15	0.08	0.16	0.08
donfile	0.17	0.37	0.19
keyb	0.30	1.30	1.72
kirkman	0.20	0.62	0.30
mc	0.04	0.09	0.05
planet	2.37	3.62	3.70
planet1	2.34	3.59	3.59
s1a	0.43	1.14	4.53
sand	0.41	3.01	7.12
styr	0.54	1.23	2.09
train11	0.04	0.12	0.05

Table1: The verification results of sequential circuits

5. Conclusions

In this work, we discusses the verification problem resulted form using DC condition to simplify circuits. For efficiently solving the problem, we present a novel strategies. First, we design a cube_filter module for a DC cube, the function of the cube_filter module is to filter the min-terms in the DC cube. After constructing the cube_filter modules for all DC cubes of a DC condition, we connect them into a DC_image circuit. Such DC_image circuit can guarantee to produce no min-terms in the DC condition and can be employed to efficiently verify the consistence of a circuit and its simplified version with the DC condition. For avoid re-producing the min-terms filtered in former blocks, we also propose a heuristic to determine the cube order by which we construct the cube_filter modules. In the experiments, we use three methods to perform verification, and the results show that with the DC_image circuits, the verification can be sped up and are very encouraging.

References

[1] M. Abramovivi, M. A. Breuer, and A. D. Friedman, "Digital Systems Testing and Testable Design," *IEEE Press*, 1990.

[2] D. Brand, "Verify of Large Synthesized Designs," *Proc. of Int Conf. on Computer-Aided Design*, pp. 534-537, Nov. 1993.

[3] R. E. Bryant, "Graph-based Algorithms for Boolean Function Manipulation," *IEEE Trans. on Computes*, vol. 35, No. 8, pp. 677-691, Aug. 1986.

[4] A. Ghosh, S. Devadas, and A. R. Newton, "Test Generation and Verification for Highly Sequential Circuits," *IEEE Trans. on Computer-Aided Design*, pp. 652-667, May 1991.

[5] S. Y. Huang, K. T. Cheng, and K. C. Chen, "AQUILA: An Equivalence Verifier for Large Sequential Circuits," *Proc. of Asia and South Pacific Design Automation Conf.*, pp. 455-460, Jan. 1997.

[6] W. Kunz, "CANNIBAL: An Efficient Tool for Logic Verification Based on Recursive Learning," *Proc. of Int Conf. on Computer-Aided Design*, pp. 538-543, Nov. 1993.

[7] R. Rudell, "Dynamic Variable Ordering for Ordered Binary Decision Diagram," *Proc. of Int Conf. on Computer-Aided Design*, pp. 42-47, Nov. 1993.

[8] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Muragi, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A System for Sequential Curcuit Synthesis," University of California, Berkeley, *Technical Report* UCB/ERL M92/41, May 1992.