In Search of a Better Heuristic Algorithm for Simplification of Switching Function – A Challenge to ESPRESSO

Subhajit Guha
Dept. of Computer Science
B.R.S College
Barrackpore, India

Uma Mitra
Dept. of Computer Science
B.R.S College
Barrackpore, India

Pinki Dey
Dept. of Computer Science
B.R.S College
Barrackpore, India

Samar Sen Sarma
Dept. of Computer Science and Engg. , CU
Kolkata, India

ABSTRACT

Finding minimum cost switching function is an intractable problem. The number of prime implicant of a typical switching function is the order of 3^n [5][13]. When fitness function with respect to cost is the only goal, the existing algorithms require faces combinatorial explosion [20]. The allurement of approximation and heuristic algorithms in this area has already generated ESPRESSO algorithm. We find that our algorithm presented here is at least equal or more cost effective than the ESPRESSO algorithm. The paper shows a new approach for attainment of our claim. We hope the algorithm presented here is a new attachment to the existing state of the art technology.

Keywords

Minimization, Two-level logic, Prime implicant, greedy-heuristic, Arrange

1. INTRODUCTION

The problems of simplification of switching functions occur in many areas of the logic design. Boolean representations of switching functions are obtained from a function description. The optimization of logic functions, and in particular two-level logic minimization is performed on the Boolean representation. The result of simplification depends on the representation of literals. As an example, the complexity of the combinatorial component of a FSM depends on the Boolean literals to the internal states. The design of switching functions is restricted to combinatorial two-level SOP representations, as it appears in the logic synthesis process. The need for simplification of the number of terms of the SOP form is apparent. Starting with the classical approach of minimization techniques many different simplification techniques have been developed. In 1953 Karnaugh [4] proposed a technique for simplifying Boolean expressions using an elegant visual technique, Quine [2] and McCluskey [3] proposed a tabular technique that uses a two-step process which first generates all prime implicants and then obtains a minimal covering for simplifying switching functions. The heuristic based approach differs from the classical one in two aspects. At first, the cost function is simplified by assigning an equal weight to every implicant, and then the final solution is obtained from an initial solution by iterative improvement rather than by generating and covering prime implicants. Because of the required storage and computations in many applications, exact solutions are not necessary but near minimum solution solutions are sufficient. Heuristic usually produce solution that are near to the optimum in a relatively short time. MINI[6], ESPRESSO[7] are the heuristic based simplification procedures, but lacks in quality and runtime for functions with large number of literals. BOOM[9] handles it well but it needs to have the function's off-set specified explicitly, which limits its usability in cases of functions specified by their on-sets only. We have proposed a SOP representation based on greedy-heuristic strategy that suggests that one can devise an algorithm that work in stages, considering one input at a time. At each stage, a direction is regarding whether a particular input is in an optimal solution and reduces time-space complexity.

2. PRELIMINERIES

Switching function [Definition]: Let $T(x_1, x2. \ldots x_n)$ be a switching expression. Since each of the variables $x_1, x_2. \ldots x_n$ can independently assume either of the two values 0 or 1, there are 2^n combinations of values to be considered in determining the values of T . In order to determine the value of an expression for a given combination, it is only necessary to substitute the values for the variables in the expression. In other words, a switching function $f(x_1,x_2, \ldots, x_n)$is a correspondence that associates an element of the Boolean algebra with each of the 2^n combinations of variables $x_1,x_2, \ldots, x_n$. This correspondence is best specified by means of a truth table. Note that each truth table defines only one switching function, although this function may be expressed in a number of ways. For example, if $T(x, y, z) = x'z + xz'+ xy$, then for the combination $x = 0$, $y = 0$, $z = 1$, the value of the expression is 1 because $T(0, 0, 1) = 0'1 + 01' +0'0'= 1$. In a similar manner, the value of T may be computed for every combination, as shown in the right-hand column of Table 1.If we now repeat the above procedure and construct the truth table for the expression $x'z + xz' + y'z'$, we find that it is identical to that of Table 1.

Minimization [Definition][19]: In simplifying a switching function $f(x_1, x_2. \ldots x_n)$ is to find an expression $g(x_1, x_2, \ldots, x_n)$ which is equivalent to f and which minimizes some cost criteria. The most common are:

1. The minimum number of appearances of literals;
2. The minimum number of literals in a SOP expression;
3. The minimum number of terms in a SOP expression, provided that there is no other such expression with the same number of terms and fewer literals.

Table 1

x	y	z	f	g	f'	f + g	f.g
0	0	0	1	0	0	1	0
0	0	1	0	1	1	1	0
0	1	0	1	0	0	1	0
0	1	1	1	1	0	0	1
1	0	0	0	0	1	0	0
1	0	1	0	1	1	1	0
1	1	0	1	0	0	1	0
1	1	1	1	1	0	0	1

Two-Level Logic Minimization: A disjunctive normal form (DNF), also called SOP, is a disjunction of products. A product is in term of literals. For instance, x and y' are literals, xy'z is a product, and xy'z + xyz + y'z' is a sum-of-products (SOP). The problem is to find, given a Boolean function *f*, the minimal SOP that represents *f*. This problem is known as two-level logic minimization.

Heuristic [Definition]: It refers to experience-based techniques for problem solving, learning, and discovery. By using a rule of thumb, an educated guess, an intuitive judgment, or common senses; heuristic methods are used to speed up the process of finding a near optimal solution.

Greedy Heuristic [Definition]: It follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum. In many problems, a greedy strategy does not in general produce an optimal solution, but nonetheless a greedy heuristic may yield locally optimal solutions that approximate a global optimal solution in a reasonable time. In switching function simplification problem, the cost function is minimized by distributing an equal weight to every implicant. The ultimate solution is obtained from an initial solution by iterative improvement, rather than by covering prime implicants. Bounding the cost function to the number of implicants in the solution has the improvement of eliminating many of the problems associated with local minima. Since only the number of prime implicants is important, their figures can be changed as long as the coverage of the minterms remains proper.

3. SIMPLIFICATION OF SWITCHING FUNCTION

A switching function can usually be represented by a number of expressions. Our aim was to develop a procedure for obtaining a minimal expression for such a function. All algorithms based on the classical approach for the simplification of Boolean logic, start with the computation of all prime implicants. Afterwards prime implicant is used to construct a minimal logic. It is not known whether one may compute efficiently minimize the function without computing implicitly prime implicants. However, the number of prime implicants of one class with n literal is proportional to $3^n/n$ [5] [13]. Thus, for many functions, the number of prime implicants can be very large. Consequently, the covering step leads to a greater problem because of its well known computational complexity. Because of the required storage-space and computations, machine processing to obtain the minimum solution by the classical approach becomes impractical. On the other hand, heuristic seeks a minimal implicant solution, without generating all prime implicants, which can be converted to prime implicants if required. The ESPRESSO is a standard heuristic process that uses approximate methods based on previous experience to obtain a near optimal solution. As a successful minimization algorithm, the ESPRESSO plays the important role of our proposed work.

3.1 ESPRESSO Algorithm

```
ESPRESSO (Fan, Fdc)
    Foff = Complement (Fan, Fdc);
    F = Expand (Fan, Foff);
    F = Irredundant (F, Fdc);
    E = Essentials (F, Fdc);
    F = F - E;
    Fdc = Fdc - E;
    do {
        F= Reduce (F, Fdc);
        F= Expand (F, Foff);
        F= Irredundant (F, Fdc);
    } while (Too_High (Cost (F)));
  return (F U E);
```

Basic principle is laid on three steps; Expand, Reduce and Irredundant.

Expand: Expands implicants into prime implicants. Any implicants covered by the expanded prime implicant omitted from further consideration.

Reduce: Transforms prime implicants into implicants of least possible size such that all minterms of the function are still covered. This may lead to better solutions later.

Irredundant: Chooses a minimal subset of prime implicants obtained so far such that the subset covers all minterms of the function. Similar to prime implicant chart covering, however, less time-consuming due to fewer prime implicants. In Figure 1 to Figure4 applying $f(x, y, z) = \sum(1, 2, 3, 4, 5, 6)$ for simplification according to Expand, Reduce and Irredundant to the initial set of minterms. After the set of transformations, a superior SOP is obtained.

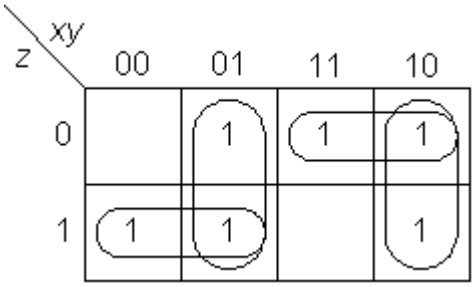

Fig 1: Initial covering of *f*

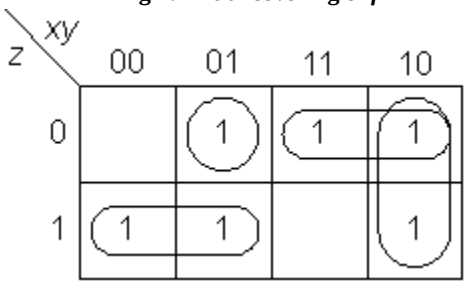

Fig 2: After the Reduce Step

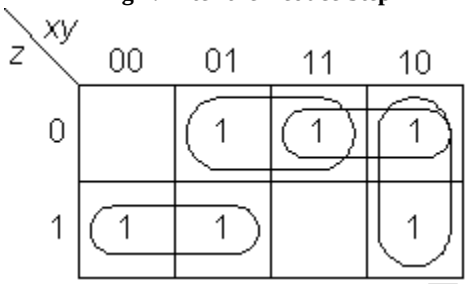

Fig 3: After the Expand Step

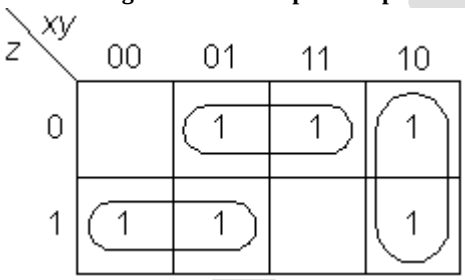

Fig 4: After the Irredundant Step

These three procedures are iterated with different starting points until there is no further improvement in the optimality of the reduction.

3.2 Proposed Algorithm for Simplifying Switching Functions

We have developed an alternative to ESPRESSO algorithm that reduces average computational complexity in time and space domain. The Reduce step used in basic ESPRESSO has been removed, as Expand and Irredundant form are the sufficient criteria to lead a near optimal solution. Here we introduced a greedy-heuristic approach, the greedy method suggests that one can devise an algorithm that work in stages under the consideration of one input at a time. At each stage a direction is regarding whether a particular input is in an optimal solution. This is done by considering the input literals in an order into a list L, determined by an efficient arrangement procedure Arrange (L). The first element (min-term) belong to the list will be compared with remaining one.

- If the difference between two of this literals is power of two or only one bit change in binary representation, the literals will generate a pair, this process in known as Expand.
- If any literals in the list are generating a pair before, it will not compare any other literals in this iteration. Therefore, an irredundant form of output will be generated. This irredundant output set is the input set of the next iteration, until no further pair generates.

Hence, all possible pairs of input are not required to select proper pairs to achieve the desire goal in irreducible form. Therefore, the space-time complexity improves in all sense.

Algorithm Simplification {
F =**Arrange** (L)
F =**EXPAND** (F)
F =**IRREDUNDANT** (F)
If F is a goal state {
return 0;
}
else {
continue with initial state as the current state;
}
Repeat
Pick the **IRREDUNDANT** solution;
EXPAND it in a specific direction using Boolean simplification law;
for each COVER **do {**
If it is not generate before {
Evaluate its **IRREDANDENT** form and store it with its parent.
}
If **IRREDANDENT** form is a goal solution {
return 0;
}
If it is better than present form {
It will be **IRREDANDENT** form
}
If it is not better solution {
Retrieve the parent as a **IRREDANDENT** form in storage;
}
} **Until** a solution found
}

The simplification algorithm is applied on the following function *f* (w, x, y, z) = ∑(0 1 3 4 5 7 8 9 11 15). In Figure5 (0-1) makes a cover as only one bit change in binary representation among literal 0 and 1, similarly (4-5), (3-7),(8-9) and (11,15) make covers as the difference between two of this literals is power of two (see Figure6). Furthermore if any literals generate a pair before, would not be compared to other literals. Therefore, an irredundant form of output will be produced, that leads to the input for next iteration. After further expansion among irredundant literals (0-1-4-5) and (3-7-11-15) produce covers and (8-9) remains alone. This will be the simplified function.

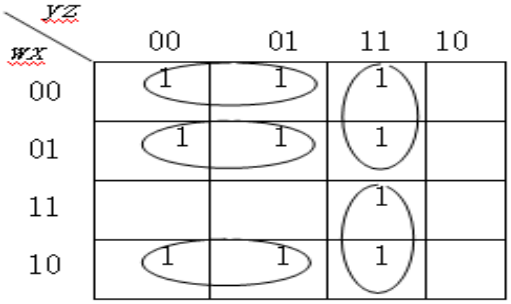

Figure5: Initial covering of *f* based on ordered covering rule

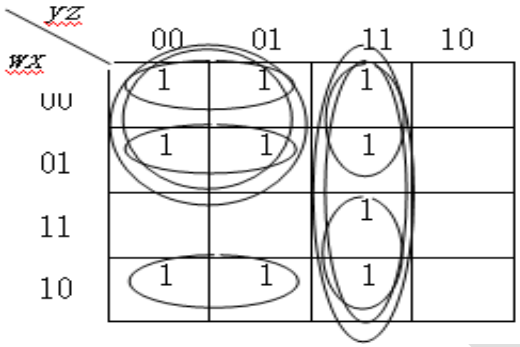

Figure6: After Expansion of covers using greedy Approach

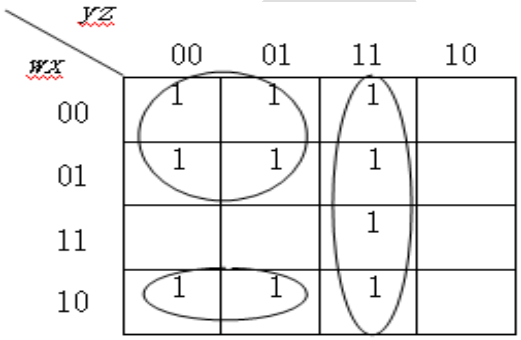

Figure7: Simplified form of function *f* (w,x,y,z)

4. EXPERIMENTAL RESULTS

The proposed algorithm for simplification of switching function has been implemented in C language on UNIX platform. To demonstrate the efficiency of our algorithm, we have taken ESPRESSO for comparison purpose. TABLE II shows the experimental results. The columns with labels 'Number of Literals' and 'Delivered switching functions randomly in SOP' show the number of inputs respectively. The next columns indicate the CPU time needed in millisecond unit for ESPRESSO and the proposed algorithm. The average time column gives the information of average time needed for the ESPRESSO and the proposed simplification algorithm based on random literals. Our algorithm shows approximately 9.5% improvement.

Table 2. Benchmarking Results for Simplification of Switching Function

No. of Literals	No. of terms in SOP	ESPRESSO (ms)	Avg. Time (ms)	Our (ms)	Avg. Time (ms)
4	4	23.51	**33.84**	17.03	**29.07**
	5	17.59		27.36	
	6	21.20		14.28	
	7	22.12		14.28	
	8	27.47		33.51	
	11	26.92		17.03	
	12	32.47		23.73	
	13	29.67		23.07	
	14	39.34		26.37	
	15	65.52		63.18	
	15	66.51		60.03	
5	5	28.04	**45.76**	15.93	**40.86**
	7	33.11		17.05	
	8	34.08		22.52	
	10	40.10		26.37	
	12	44.90		31.56	
	16	31.80		34.61	
	19	30.76		50.13	
	23	45.05		40.65	
	28	51.90		49.80	
	30	81.85		81.61	
	30	81.86		79.23	
6	6	33.51	**51.03**	17.58	**44.01**
	10	43.54		29.67	
	22	40.65		34.56	
	25	48.90		40.55	
	27	40.56		40.51	
	30	50.54		43.57	
	38	50.54		34.87	
	45	45.87		46.80	
	50	67.03		60.57	
	55	69.01		65.89	
	60	71.23		69.59	
7	7	23.51	**100.72**	18.68	**92.76**
	10	27.47		21.22	
	14	65.08		35.00	
	16	91.28		87.42	
	21	91.56		80.01	
	24	108.54		100.55	
	28	121.53		113.8	
	30	121.90		120.5	
	35	124.55		109.54	
	40	127.55		122.83	
	45	129.85		122.55	
	51	131.39		121.23	
	54	110.54		100.05	
	85	135.45		145.29	

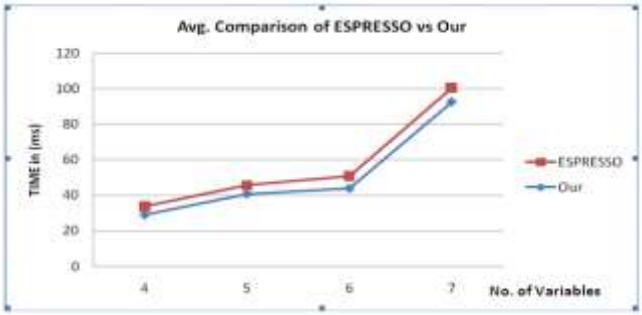

Fig 8: Comparison chart of ESPRESSO vs. proposed algorithm

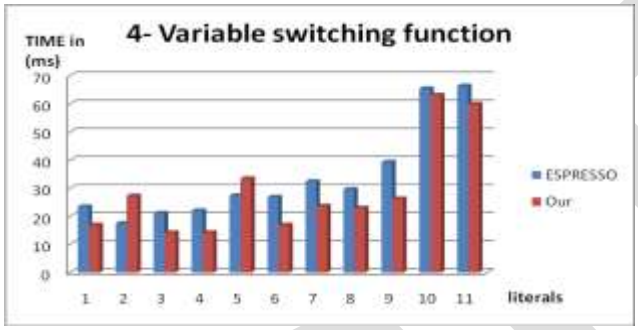

Fig 9: CPU time comaparison for 4 variable random literals

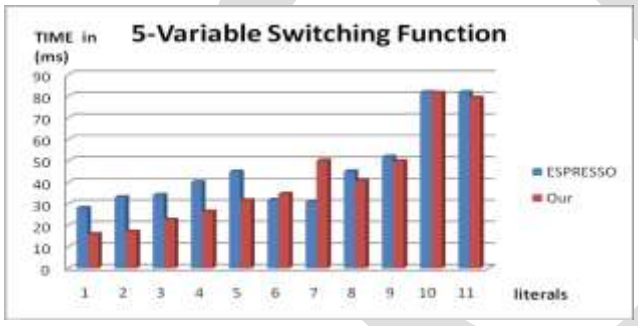

Fig 10: CPU time comaparison for 5 variable random literals

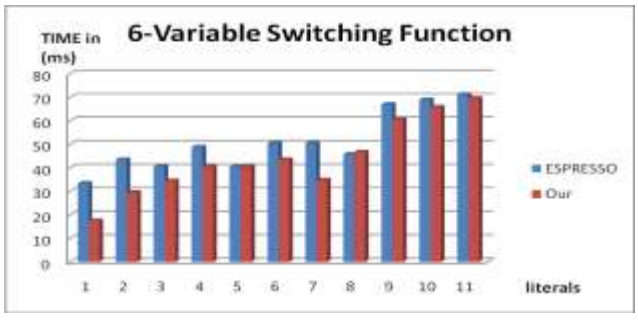

Fig 11: CPU time comaparison for 6 variable random literals

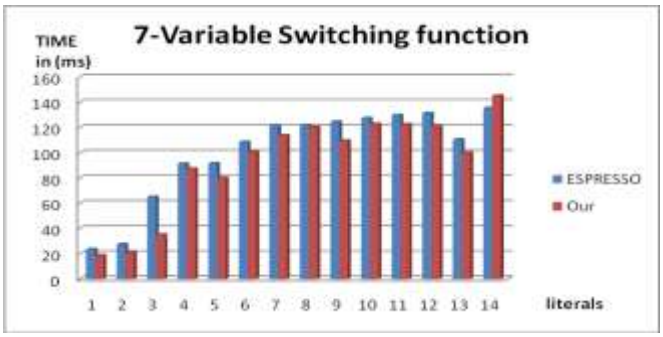

Fig 12: CPU time comaparison for 7 variable random literals

5. CONCLUSION

Sum-of-Product (SOP) of a switching function is equivalent to prime implicant coverage of true output with minimum cost. It amounts to searching all possible combinations of prime implicant coverage of the function. This is an unsolvable problem. Using the greedy heuristic procedure we have developed an alternative to ESPRESSO algorithm that reduces average complexity in time and space domain. The attempt is worth studying. Better approach like BDD approach is our own envy. We hope on next approach will be graphical mapping of the problem.

6. REFERENCES

[1] W.V.Quine, "The Problem of Simplifying Truth Functions," Am. Math. Monthly 59, pp. 521-531, 1952.

[2] W.V. Quine, " A Way to Simplify Truth functions, The American Mathematical Monthly, 62, pp. 627-631, Nov., 1955.

[3] E. J. McCluskey, Jr., "Minimization of Boolean Functions," Bell Syst. Tech. J. 35, pp. 1417-1444, Nov. 1956.

[4] M. Karnaugh, "The Map Method for Synthesis of Combinational Circuits, Trans. A.I.E.E., 72, Part I, pp. 593-598, 1953.

[5] R. E. Miller, Switching Theory, Vol. I: Combinatorial Circuits, John Wiley & Sons, Inc., New York, 1965.

[6] S.J. Hong, R.G. Cain and D.L. Ostapko, "MINI: A heuristic approach for logic minimization", IBM Journal of Res. & Dev., Sept. 1974, pp.443-458

[7] R.K. Brayton et al., "Logic minimization algorithms for VLSI synthesis", Boston, MA, Kluwer Academic Publishers, 1984, pp. 192.

[8] J. Hlavička and P. Fišer, „BOOM - a Heuristic Boolean Minimizer", Proc. ICCAD-2001, San Jose, Cal. (USA), 4.-8.11.2001, pp. 439-442.

[9] L. Jozwiak, A. Slusarczyk and M. Perkowski, "Term Trees in Application to an Effective and Efficient ATPG for AND/EXOR and AND-OR Circuits", VLSI Design, Vol. 14, No 1, January 2002, pp. 107-122.

[10] P. Fišer and H. Kubátová, "Flexible Two-Level Boolean Minimizer BOOM II and Its Applications", Proc. 9th Euro micro Conference on Digital Systems Design (DSD'06), Cavtat, (Croatia), 30.8. - 1.9.2006, pp. 369-376.

[11] P. Fišer, P. Rucký, and I. Váňová, "Fast Boolean Minimizer for Completely Specified Functions", Proc. 11th IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop 2008 (DDECS'08), Bratislava, SK, pp. 122-127.

[12] A.K.Chandra and G. Markowsky. "On the number of prime implicants", Discrete Mathematics, North-Holland Publishing Company©, Vol. 24, 7-11, 1978.

[13] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting - a fresh look at combinational logic synthesis". In Proceedings of the 43rd Annual Conference on Design Automation, San Francisco, CA, USA, July 24 - 28, 2006, pp. 532-535.

[14] N.N. Biswas, "Computer aided minimization procedure for Boolean functions", In Proceedings of the 21st Designed Automation Conference (DAC'84), pp. 699-702, IEEE Press Piscataway NJ, USA© 1984, ISBN: 0-8186-0542-1.

[15] St. Mihailov, A. Popov, Kr. Filipova, N. Kasev, "Comparative Analysis of Boolean Functions Minimization in Terms of Simplifying the Synthesis", First International Congress of Mechanical and Electrical Engineering and Technologies, ISBN 954-20-0215-7, MARIND 2002, 6-11 Oct., Varna, pp.273-276.

[16] A. Popov, Kr. Filipova, "Genetic Algorithms synthesis of finite state machines", 27th International Spring Seminar on Electronics Technology. IEEE Proc., Catalog N 04EX830, ISBN 0-7803-8422-9, pp.388-392, 13-16 may, 2004.

[17] A. Popov, "Genetic Algorithms for Optimization-Application in the regulator synthesis task", Bachelors thesis at Technical University.

[19] Z. Kohavi, Switching and Finite Automata Theory, Tata McGraw-Hill publishers, Second Edition.

[20] D.E. Knuth, "The Art of Computer Programming- Vol. 4, Introduction to Combinatorial Algorithms and Boolean Functions", Pearson Education, Inc., ISBN 0-321-53496-4.