Improving the Accuracy of Support-Set Finding Method for Power Estimation of Combinational Circuits

Hoon Choi and Seung Ho Hwang

Department of Electrical Engineering
Korea Advanced Institute of Science and Technology, Korea
hchoi@allegro.kaist.ac.kr, shwang@ee.kaist.ac.kr

Abstract

We address a way to improve the accuracy of support-set finding method for a probability-based power estimation of combinational circuits. Support-set finding methods to build local BDDs have been proposed to handle large circuits. However, because they consider only the shallow reconvergence, they are not accurate enough to be used in the power optimization. To solve this problem, we propose a new algorithm, Feather algorithm, which can efficiently detect minimal support-set with 100% reconvergent node detection rate. The experimental results show that the average error of our proposed method is 0.1% for the total power and 1.6% for the node-specific power.

1. Introduction

The increasing needs for portable applications such as laptop and notebook computers have made the power consumption a critical concern, and power analysis and optimization has become one of the most important tasks to be solved by computer-aided design tools. There have been several methods to estimate power consumption. Ghosh [1] used symbolic simulation to produce a set of boolean functions which represent the conditions for switching at each gate in a circuit at a specific time instance. Though very accurate, it suffered from low speed and large memory usage because it used a global BDD to calculate signal probability. As a result, it could not handle large circuits. To overcome this problem, Kapoor [3] proposed a method that partitions a circuit and builds a local BDD. It tries to maximize the number of correlated nodes within each partition to obtain more accurate results. The circuit is partitioned into blocks, each of which has only independent inputs, and each block is represented by a BDD to calculate the exact signal activity. Though being accurate, it makes some blocks unnecessarily large and as a result requires more memory than really needed. Mehta [4] also showed an algorithm to build a local BDD. However, it only considers a shallow reconvergence, hence it often misses some correlation information among input signals of a gate. Therefore, though it can handle large circuits, the results are often erroneous. These errors may not be a serious problem for the power estimation, but for the power optimization the accurate estimation of node-specific power is very crucial because it steers the direction of the optimizer [3].

In this paper, we revisit the support-set finding methods. First, we show the problems of previous methods. Then we propose a new method, called *Feather algorithm*, which overcomes the problems and limitations of previous methods. The validity of the proposed method is demonstrated by experimental results.

The paper is organized as follows: Section 2 describes the problems of previous methods. In section 3 a new support-set finding method is proposed. We show the experimental results in section 4 and section 5 concludes the paper.

2. The Problems of Previous Methods

Most of the power in CMOS circuits is consumed during charging and discharging of the load capacitance. To estimate the power consumption, one has to calculate the switching activity factors of the internal nodes of the circuit. Methods of estimating the activity factor $E_n(sw)$ at a circuit node n involve estimation of signal probability prob(n), which is the probability that the signal value at the node is one. Under the assumption that the values applied to each circuit input are temporally independent, we can estimate $E_n(sw)$ by using

$$E_n(sw) = 2\,prob(n)(1 - prob(n)).$$

The BDD can be used to calculate the signal probability of a gate if all nodes of a BDD are independent each other [2]. We call such a node set *support-set* of a gate. The size of a BDD heavily depends on the number of nodes and its order. Hence it is very important to extract

ED&TC '97 on CD-ROM

the minimal support-set of a gate to reduce the size of a BDD.

Kapoor [3] proposed a method to partition a circuit with the goal of packing as many correlated nodes into a same partition as possible. Then, the input signals of a cluster become the support-set of all gates in the cluster. But this method has a drawback as demonstrated in Fig. 1 where each gray circle represents a partition which has only independent signals as inputs.

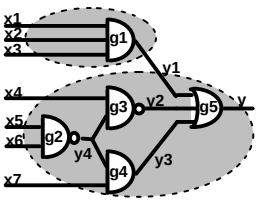

Fig. 1: Drawback of the method in [3]

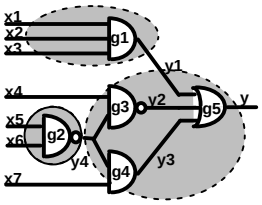

Fig. 2: Improved partition of Fig. 1

The bottom circle encloses four gates and five input signals: x4, x5, x6, x7 and y1. However, the NAND gate g2 can be excluded from the partition as shown in Fig. 2. The latter one is better than the former one because the size of the associated BDD is smaller. This problem comes from the fact that the method does not use the circuit topology information.

Another method to reduce the size of a BDD is proposed in [4]. It will be denoted as X method in this paper. The support-set finding algorithm is as follow: for all input pairs of a gate, (i, j), check whether the support-set of signal i and j, i_{sup} and j_{sup}, have common items. If they have common items, the support set of the gate output is the union of i_{sup} and j_{sup}. If not, the union of i and j is the support set of the gate output. It can resolve the problem of the method in [3], but this approach has the following problem. Let's consider the example of Fig. 3 where each circle is a gate and the support-set is shown in a gray box near each gate. When we process node D which has input signals 6 and 7, the support sets {6} and {3, 4} have no common elements. Hence the support set of signal 10 becomes {6, 7}. Same argument is applied to node E and the resulting support set is {8, 9}. Then, signal 10 and 11 are considered as independent when calculating the support-set of node F because the support-sets of the two signals, {6,7} and {8,9} respectively, are independent. However, in fact they are not independent because signal 4 reconverges at node F, and as a result an incorrect support-set and its BDD will be constructed for node F. The correct support set of each gate including node F is shown in Fig. 4. This problem is due to the fact that the X method only considers the shallow reconvergence. Hence it misses some correlation information among the input signals of a gate.

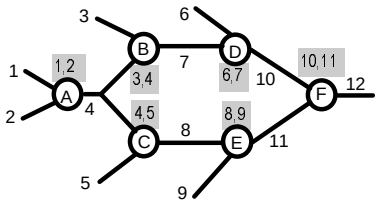

Fig. 3: Drawback of the method in [4]

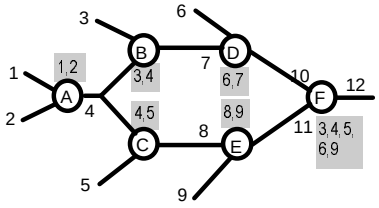

Fig. 4: Correct support-set of Fig. 3

3. New Support-Set Finding Algorithm

In order to detect all the reconvergence including the deep one, our new algorithm uses the primary input signals as a criteria to determine the dependence among input signals of a gate. Then, it finds the minimal support-set of a gate composed of internal signals to reduce the size of a BDD. The pseudo code of our algorithm is shown in Fig. 5. First, we visit all nodes and write transitive fanins of a node using only primary inputs, called PISet. Then we revisit every node and check whether the PISet's of input signals are independent. If they are independent, the union of input signals themselves become the support-set. However, if they are dependent, we have to find minimal support-set by examining PISet's of input signals: We list all the input signals, which we call *dependent node list* and check if any of them is independent from other signals by using the PISet. If any of them is, it is deleted from the list and added to the support-set of the node. The other signals, each of which is dependent with at least one of the remaining signals, are examined to select the signal whose support-set cardinality is maximum. The selected signal is deleted and its input signals are added to the tail of the list. The above procedures are repeated until the list becomes empty. To prevent the resulting support-set from growing too large, we employ a limit. If the sum of the support-set cardinality and the *dependent node list* cardinality under process is more than the prescribed limit, we exit from the loop and the union of those two is returned as the support-set.

We call this algorithm as *Feather algorithm* because it looks like a falling feather which glides back and forth continuously.

```
Algorithm Feather
/* FIs(i): immediate fanins of node i */
/* SSi: support-set of node i */
/* ID(S): true iff ∀i, j∈S ∧ i≠j, PISeti∩PISetj=∅ */
/* DNi: dependentNodeList of node i */
/* IDs(S): {i | i∈S, ∀j∈S ∧ i≠j, PISeti∩PISetj=∅} */
for each node i from PI to PO{
   PISeti = ∪{PISetj | j ∈ FIs(i)}
}
for each node i from PI to PO{
   if ( |FIs(i)| == 1 || ID(FIs(i)) == true ){
      SSi = ∪ FIs(i)
   }else {
      DNi = ∪ FIs(i)
      while DNi ≠ ∅ {
         if ( |SSi| + |DNi| > limit) {
            SSi = SSi ∪ DNi
            break while loop
         }
         if (IDs(DNi) ≠ ∅){
            SSi = SSi ∪ IDs(DNi)
            DNi = DNi \ IDs(DNi)
         }
         MCS = {k | |SSk| == MAX∀j∈DNi(|SSj|)}
         DNi = DNi \ MCS
         DNi = DNi ∪ FIs(MCS)
      }
   }
}
```

Fig. 5: Pseudo-code of Feather algorithm

To illustrate the operations of our algorithm, we show the steps involved in finding the support-set of node F of Fig. 3 in Fig. 6. In the first row, signal 10 and 11 has common PISet element, 1 and 2. Hence signal 10 is replaced with 6 and 7. In the second row, signal 11 and 7 has common PISet element 1 and 2, but 6 is an independent signal. So 6 is deleted from the list and added to the support-set. After similar steps, the support-set is obtained as in the last row.

Step	dependentNodeList	Support-set	DividedSignal
1	10, 11		10
2	11, 6, 7	6	11
3	7, 8, 9	6, 9	7
4	8, 3, 4	6, 9, 3	8
5	4, 5	6, 9, 3, 4, 5	

Fig. 6: Steps involved in finding the support-set of node F of Fig. 3 by Feather algorithm

In implementation, we have employed the following heuristics to reduce the CPU time and memory requirement. First, in detecting independent signals from the *dependent node list*, only those signals are checked that were added in the very last iteration because only they can be the candidates for independent signals. Second, when the cardinality of PISet is smaller than the predefined value, we use the PISet itself as a support-set without finding the minimal one. This reduces the number of nodes that has to be processed with a little memory penalty because it does not use the signal probability of the internal nodes but only uses that of the primary input that is given at the start of the estimation. Third, to reduce the memory requirement we free the memory space that is used by PISet as soon as the node's support-set is calculated. We also free the support-set information as soon as it is used to build a BDD. Finally, the memory used for BDD is freed right after the signal probability of the node is calculated. Because we process the nodes from the primary input side to the primary output side, as we approach to primary output side where more memory is needed due to the larger BDD than that at primary input side generally, the more memory has been already freed and available for a BDD. These heuristics enable us to reduce the CPU time and memory requirement.

4. Experimental Results

We have implemented the previous support-set finding methods and our algorithm in SIS environment[5] and used ISCAS85 benchmark circuits mapped onto 2-input gates of mcnc.genlib library as our input circuits[6]. All power estimations are measured in μW with 20MHz clock frequency and compared to those obtained using the method presented in [1] for circuits from C17 to C1908 and to those obtained using simulation for the other circuits. All the methods employed in our experiments are represented by the pair of (M, C). M represents the support-set finding methods: X method and FE(*Feather algorithm*: our proposed algorithm). C represents the cardinality limit of a support-set. It can be either U (unlimited) or an integer number.

First, we compared the reconvergent node detection rate. The result is shown in Table 1. The X method detects only 18.7% reconvergent nodes on the average because it can not detect the deep reconvergence. In contrast, our method detects all the reconvergent nodes because we use the primary input signals as a criteria to determine the dependence among input signals of a gate.

Second, to see the effect of different reconvergent node detection rate to the accuracy of power estimation, we compared the total power and node-specific power as shown in Table 2 and Table 3, respectively. For our algorithm, we tested it with varying the limits of support-set cardinality. Table 2 shows that our proposed method es-

timates the total power within 0.1-0.5% error on the average, which is almost one order of magnitude improvement compared to 2.2% error of X method. In addition, Table 3 describes that our proposed method estimates the node-specific power within 1.6-7.0% error on the average, which is much better than that of the X method, 13.8% on the average.

Third, we compared the CPU time of global BDD method in [1], X method and our proposed method measured on SPARC20. We used ordering of BDD variables in [7] for all three methods. The result is listed in Table 4. The global BDD method not only takes lots of CPU time but also can handle only the circuits from C17 to C1908 due to the excessive memory requirements for other circuits. The X method and our proposed method solve all the circuits from C17 to C7552 due to the minimal support-set finding capability. Our proposed method is slower than the X method because of the additional routines to find all the reconvergent nodes. Nevertheless, the CPU time of our method is still reasonable even for large circuits.

	# total nodes	# total reconv nodes	# detected reconv nodes			
			X	rate	FE	rate
C17	6	2	1	50.0%	2	100.0%
C432	256	177	8	4.5%	177	100.0%
C499	240	134	4	3.0%	134	100.0%
C880	349	111	5	4.5%	111	100.0%
C1355	564	438	200	45.7%	438	100.0%
C1908	497	311	27	8.7%	311	100.0%
C2670	901	405	131	32.3%	405	100.0%
C3540	1374	796	81	10.2%	796	100.0%
C5315	2252	1043	146	14.0%	1043	100.0%
C6288	2370	2081	464	22.3%	2081	100.0%
C7552	2897	1994	215	10.8%	1994	100.0%
avr				18.7%		100.0%

Table 1: Reconvergent node detection rates of X and FE

	Sim	X,U	error	FE,10	error	FE,20	error	FE,30	error
C17	34.3	34.4	0.3%	34.3	0.0%	34.3	0.0%	34.3	0.0%
C432	1013.4	968.6	4.4%	992.6	2.1%	1018.4	0.5%	1012.4	0.1%
C499	1607.2	1606.4	0.0%	1607.7	0.0%	1607.2	0.0%	1607.2	0.0%
C880	1597.9	1598.4	0.0%	1596.8	0.1%	1595.6	0.1%	1597.1	0.1%
C1355	2286.0	2339.0	2.3%	2287.5	0.1%	2286.0	0.0%	2286.0	0.0%
C1908	2351.8	2339.1	0.5%	2351.8	0.0%	2350.6	0.1%	2352.9	0.0%
C2670	4153.6	4183.8	0.7%	4159.3	0.1%	4167.1	0.3%	4164.9	0.3%
C3540	5573.1	5527.0	0.8%	5530.2	0.8%	5548.6	0.4%	5561.3	0.2%
C5315	10715.1	10526.4	1.8%	10633.6	0.8%	10655.3	0.6%	10677.4	0.4%
C6288	12103.4	13557.0	12.0%	12312.0	1.7%	12225.5	1.0%	12166.1	0.5%
C7552	13295.6	13180.5	0.9%	13263.7	0.2%	13293.9	0.0%	13298.4	0.0%
avr.			2.2%		0.5%		0.3%		0.1%

Table 2: Total power comparison

	X,U	FE,10	FE,20	FE,30	FE,40	FE,50
C17	0.1%	0.0%	0.0%	0.0%	0.0%	0.0%
C432	17.4%	7.2%	7.7%	8.0%	2.9%	1.3%
C499	8.7%	9.8%	0.0%	0.0%	0.0%	0.0%
C880	1.6%	1.4%	0.5%	0.2%	0.1%	0.0%
C1355	23.0%	16.5%	3.9%	0.0%	0.0%	0.0%
C1908	7.0%	5.1%	5.2%	3.1%	2.4%	0.5%
C2670	15.3%	9.1%	4.3%	4.4%	3.3%	3.0%
C3540	14.8%	8.0%	6.9%	6.6%	5.5%	5.2%
C5315	7.9%	4.9%	3.5%	2.9%	2.5%	2.3%
C6288	43.7%	9.8%	5.2%	3.6%	3.0%	2.6%
C7552	11.8%	5.7%	3.5%	2.9%	3.0%	3.1%
avr.	13.8%	7.0%	3.7%	2.9%	2.1%	1.6%

Table 3: Node-specific power comparison

	CPU time in sec.						
	Global	X,U	FE,10	FE,20	FE,30	FE,40	FE,50
C17	0.1	0.1	0.1	0.1	0.1	0.1	0.1
C432	40.1	4.6	7.3	11.0	33.2	106.4	182.4
C499	22.4	4.2	6.6	15.0	19.4	31.1	43.2
C880	4.7	5.0	7.7	12.7	17.4	21.7	26.4
C1355	55.6	10.3	20.9	39.4	61.3	105.9	126.6
C1908	18.8	8.5	16.2	31.8	49.7	73.8	125.7
C2670	---	15.1	24.2	38.0	46.3	49.1	58.4
C3540	---	20.4	41.7	68.3	97.9	147.3	283.8
C5315	---	33.5	79.0	109.6	131.7	152.8	168.2
C6288	---	40.3	149.7	366.1	632.4	1167.8	1747.0
C7552	---	53.9	127.7	201.1	242.8	282.5	326.0

Table 4: CPU time comparison

5. Conclusion

In this paper we presented the problems of previous support-set finding methods and proposed a new algorithm called *Feather algorithm* which can detect all the reconvergent nodes and find the minimal support-set. The experimental results show that the average error of our proposed method is 0.1% for the total power and 1.6% for the node-specific power, which is almost one order of magnitude improvement compared to the previous methods. In addition, the CPU time of our method is still reasonable.

References

[1] A. Ghosh, S. Devadas, K. Keutzer, and J. White, "Estimation of Average Switching Activity in Combinational and Sequential Circuits," in *Proc. 29th Design Automation Conference*, 1992, pp. 253-259.

[2] F. N. Najm, "Transition Density: A New Measure of Activity in Digital Circuits," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 310-323, Feb. 1993.

[3] B. Kapoor, "Improving the Accuracy of Circuit Activity Measurement," in *Proc. 31st Design Automation Conference*, 1994, pp. 734-739.

[4] H. Mehta, M. Borah, R. M. Owens, and M. J. Irwin, "Accurate Estimation of Combinational Circuit Activity," in *Proc. 32nd Design Automation Conference*, 1995, pp. 618-622.

[5] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton and A. Sangiovanni-Vincentelli, "Sequential Circuit Design Using Synthesis and Optimization," in *Proc. IEEE Int'l Conf. Computer Design*, 1992, pp. 328-333.

[6] Saeyang Yang, *Logic Synthesis and Optimization Benchmarks User Guide: Version 3.0* Technical Report, Microelectronics Center of North Carolina, Jan. 1991.

[7] S. Malik, A. R. Wang, R. K. Brayton and A. Sangiovanni-Vincentelli, "Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment," in *Proc. IEEE Int'l Conf. Computer-Aided Design*, 1988, pp. 6-9.