SWITCHING ACTIVITY ESTIMATION OF FINITE STATE MACHINES FOR LOW POWER SYNTHESIS

Mikael Kerttu   Per Lindgren
Luleå University of Technology
Luleå, Sweden
{kerttu,pln}@sm.luth.se

Mitch Thornton
Mississippi State University
Mississippi State, MS, USA
mitch@ece.msstate.edu

Rolf Drechsler
University of Bremen
28359 Bremen, Germany
drechsle@informatik.uni-bremen.de

ABSTRACT

A technique for computing the switching activity of synchronous Finite State Machine *(FSM) implementations including the influence of temporal correlation among the next state signals is described. The approach is based upon the computation that a FSM is in a given state which, in turn, is used to compute the conditional probability that a next state bit changes given its present state value. All computations are performed using* Decision Diagram *(DD) data structures. As an application of this method, the next state activity information is utilized for low power optimization in the synthesis of* Binary Decision Diagram *(BDD) mapped circuits. Experimental results are presented based on a set of the ISCAS89 sequential benchmarks showing an average power reduction of 40 percent and up to 90 percent reduction for individual benchmarks on the estimated power dissipation.*

1. INTRODUCTION

In CMOS based digital circuits switching activity is one of the main contributors to overall power dissipation [7]. In order to optimize a circuit for low power it is of crucial importance to account for switching activities of internal signals. Unfortunately, it is difficult to account for internal net switching activity in general since these quantities are dependent upon spatial and temporal correlation of input signals. In [5] a method for low power optimization that utilizes temporal correlation is presented. Assuming that input signals switch randomly can lead to grossly inaccurate switching activity estimates. This provides motivation for investigating methods for computing switching activities taking correlations into consideration. The focus here is to exploit the relationship specified in the transition function portion of an FSM net-list to extract temporal correlation information about the state vector bits.

CMOS circuit power dissipation depends heavily on the input signal switching characteristics. If the signals change the circuit will dissipate power and if they are constant the power dissipation is predominately due to leakage. This observation leads to the conclusion that minimizing signal changes in the circuit will lead to a smaller amount of energy consumption. One problem with optimizing digital circuits is that information about input signals to each internal sub-circuit is not usually known. It is necessary to know how these signals behave to get an accurate result. One way to gather this information is to use simulation of the circuit; however, it far too complex to get full coverage of even moderately sized circuits in this manner. Alternatively, analysis of the circuit structure may be used to obtain the needed information. The technique described here uses analysis of a net-list description of an FSM to acquire information that can be used in determining internal switching activities within the circuit.

The approach is to first extract the state transition relation from a given net-list and then to compute limiting distribution giving the probability that the FSM is in some state. The limiting distribution computation requires that input signal probabilities are provided and a state space traversal for extraction of the transition graph. All computations are performed through the manipulation of Algebraic Decision Diagrams (ADDs) [2][3]. Also, BDDs are used for the state space traversal which is based on a depth-first traversal method. During the traversal the transition probabilities are calculated. To span the state space of FSMs, BDD techniques have shown great efficiency as in [1].

If a priori information on input probabilities is known, it is possible to derive the probability and activity for next state Using an iterative method, the steady state probabilities are found which are used to compute the switching activities of the next state and output signals. These quantities are obtained efficiently and require only a single traversal of the ADD.

The paper is organized as follows. Section 2 describes FSM analysis and previous work. Section 3, presents how we extract the signal statistics from the steady state probabilities and the transition probability matrix. Then in section 4 we describe how we use the signal statistics in low power synthesis. The experimental results are shown in section 5. Finally the paper is concluded in section 6.

2. FSM ANALYSIS

There are several approaches for efficient FSM spanning [1]. We have chosen to implement the spanning function in a strait forward way by a depth-first recursive algorithm, which also calculates the transition probability matrix represented by an ADD in the same pass. In [2] and [3] Algebraic Decision Diagrams (ADD) were used as the transition probability matrix and the steady state probabilities were calculated in an efficient way. We have done our calculations on the ADD in an iterative way, which is sufficient for our purpose of signal activity extraction.

Throughout this section we will provide a description of the methodology of how to do this analysis and also demonstrate it with an example. We have used a set of ISCAS89 benchmarks to test our method.

2.1. Building a BDD

The building is done in two steps. The first part is non-recursive and it builds up small BDD fragments representing the function of a row. The second part is based on a recursive algorithm that starts from the outputs and composes the fragments into a single BDD.

The first part of building the BDD is done by translating each line of the circuit description to a BDD with pseudo variables (wire names). The pseudo variables are referring to some output from an other BDD or an input variable. In figure 1 you can see the BDD fragment generated from the last line of the example.

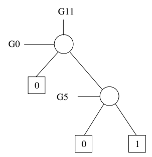

Fig. 1. Fragment of the function {G11 = AND(G0,G5)}

When we have a generated BDDs of all the lines we have to compose a single BDD for each output. This is done recursively starting from each output BDD fragment (primary and next state outputs). The recursion terminates when reaching primary inputs or register outputs. When the recursion unfolds the actual function substitutes the pseudo variables and the composed BDD is returned.

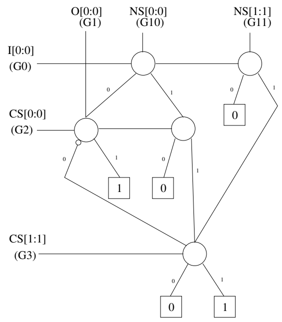

Fig. 2. The complete function

2.2. Span FSM states

The completed BDD is used to span the FSM. Starting from the reset state each possible new state is recursively visited (depth first) until reaching an already visited state. During the recursion a transition probability matrix is constructed. The usually sparse matrix is efficiently represented by an ADD (Algebraic Decision Diagram) [2][3]. This matrix is addressed with the current state as the columns and next state as the rows. The value in each entry in the matrix (ADD leaf) represents the probability to go from the current state to the next state.

Example 1 *When we calculate the transition probabilities the matrix starts empty and a new entries are added during the recursion. We assume that the probability of input I is equal to one is 1/4 (P(I)=1/4).*

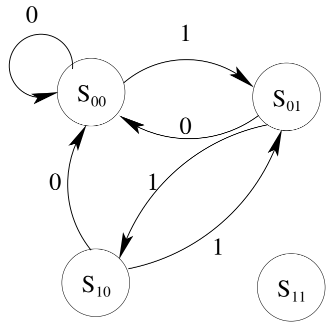

Fig. 3. FSM states

$$\begin{array}{c} \\ NS_{00} \\ NS_{01} \\ NS_{10} \\ NS_{11} \end{array} \begin{array}{c} \begin{array}{cccc} CS_{00} & CS_{01} & CS_{10} & CS_{11} \end{array} \\ \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right] \end{array}$$

Here we go from state 00 to state 01 and add the probability of P(I) to row 01 and column 00.

$$\begin{array}{c} \\ NS_{00} \\ NS_{01} \\ NS_{10} \\ NS_{11} \end{array} \begin{array}{c} \begin{array}{cccc} CS_{00} & CS_{01} & CS_{10} & CS_{11} \end{array} \\ \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 1/4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right] \end{array}$$

Here we go from state 01 to state 10 and add the probability of P(I) to row 10 and column 01.

$$\begin{array}{c} \\ NS_{00} \\ NS_{01} \\ NS_{10} \\ NS_{11} \end{array} \begin{array}{c} \begin{array}{cccc} CS_{00} & CS_{01} & CS_{10} & CS_{11} \end{array} \\ \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 1/4 & 0 & 0 & 0 \\ 0 & 1/4 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right] \end{array}$$

•

•

Finally after we have spanned all reachable states we got the complete matrix.

$$\begin{array}{c} \\ NS_{00} \\ NS_{01} \\ NS_{10} \\ NS_{11} \end{array} \begin{array}{c} \begin{array}{cccc} CS_{00} & CS_{01} & CS_{10} & CS_{11} \end{array} \\ \left[\begin{array}{cccc} 3/4 & 3/4 & 3/4 & 0 \\ 1/4 & 0 & 1/4 & 0 \\ 0 & 1/4 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right] \end{array}$$

The ADD can describing this matrix is shown in figure 4 .

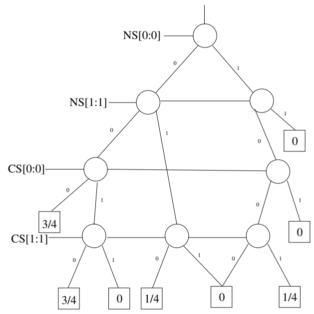

Fig. 4. ADD describing a Matrix

2.3. Calculate the state probabilities

The ADD obtained by spanning the FSM is used to calculate the steady state probabilities for each state. The FSM can be seen as a Markov chain [2][3] and this is used in the calculation of the state probabilities. The ADD is multiplied with an initial state probability vector, this represents a matrix multiplication. The initial state vector should have the sum of the entries equal to one and each column should have the sum equal to one..

$$A\bar{x} = \bar{x}' \tag{1}$$

,where A is the matrix represented by the ADD, $\bar{x}$ and $\bar{x}'$ are the steady state probability vectors after the iterations. The iteration terminates when $\bar{x}$ and $\bar{x}'$ are within the specified tolerance from each other. The resulting $\bar{x}'$ contains the resulting steady state probability vector.

Example 2 *The state probability vector is initialized such that each state entry takes on the value $1/nr$, where nr is the number of reachable states, except for the unreachable state entries, which takes on the value 0. The total probability in each column of Matrix A is one.*

$$\begin{bmatrix} 3/4 & 3/4 & 3/4 & 0 \\ 1/4 & 0 & 1/4 & 0 \\ 0 & 1/4 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1/nr \\ 1/nr \\ 1/nr \\ 0 \end{bmatrix} = \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \\ 0 \end{bmatrix} \tag{2}$$

$$\begin{bmatrix} 3/4 & 3/4 & 3/4 & 0 \\ 1/4 & 0 & 1/4 & 0 \\ 0 & 1/4 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \\ 0 \end{bmatrix} = \begin{bmatrix} 3/4 \\ 1/6 \\ 1/12 \\ 0 \end{bmatrix} \tag{3}$$

•
•

$$\begin{bmatrix} 3/4 & 3/4 & 3/4 & 0 \\ 1/4 & 0 & 1/4 & 0 \\ 0 & 1/4 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 3/4 \\ 0.1999 \\ 0.0501 \\ 0 \end{bmatrix} = \begin{bmatrix} 3/4 \\ 1/5 \\ 1/20 \\ 0 \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} 3/4 & 3/4 & 3/4 & 0 \\ 1/4 & 0 & 1/4 & 0 \\ 0 & 1/4 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 3/4 \\ 1/5 \\ 1/20 \\ 0 \end{bmatrix} = \begin{bmatrix} 3/4 \\ 1/5 \\ 1/20 \\ 0 \end{bmatrix} \tag{5}$$

The steady state probabilities(P_{SS}) are shown in equation 6.

$$\begin{array}{rcccl} P_{SS}(S[1:0]=00) & = & P_{SS}(00) & = & 3/4 \\ P_{SS}(S[1:0]=01) & = & P_{SS}(01) & = & 1/5 \\ P_{SS}(S[1:0]=10) & = & P_{SS}(10) & = & 1/20 \\ P_{SS}(S[1:0]=11) & = & P_{SS}(11) & = & 0 \end{array} \tag{6}$$

3. EXTRACTING SIGNAL STATISTICS

The Transition Probability Matrix and the Steady State probability vector can be used to calculate the bit probability and the switching activity of the next state bits.

3.1. Calculate bit probabilities

The state probabilities are used to calculate the bit probabilities for each register. The bit probabilities is calculated by traversing the ADD and utilizing equation 7.

$$(\forall i)P(NS[i:i]) = \sum_{\forall S[N-1:0]\epsilon S[i:i]=1} P_{SS}(S[N-1:0]) \tag{7}$$

Example 3 *For next state bit zero we add all Steady State probabilities, which has one on bit zero. In a similar way we do this for bit one. The computations are shown in equation 8.*

$$\begin{array}{rcccl} P(NS[0:0]) & = & P_{SS}(01)+P_{SS}(11) & = & 1/5 \\ P(NS[1:1]) & = & P_{SS}(10)+P_{SS}(11) & = & 1/20 \end{array} \tag{8}$$

3.2. Calculate Bit activities

To calculate the activity for each bit we use the ADD with the state transition probabilities and the steady state probabilities calculated earlier. $P_{SS}(n)$ denotes the steady state probability for state n, n[i:i] is the i:th bit of the vector n, A is the Matrix containing the state transition probabilities $(A[NS_k][CS_n] = P(NS_k|CS_n))$, a(NS[i:i]) is the activity for the next state bit i and is derived by the following formula and algorithm.

$$(\forall i)a(NS[i:i]) = \sum_{\forall n} P_{SS}(n) \times \sum_{\forall k\epsilon(k[i:i]\neq n[i:i])} P(NS_k|CS_n) \tag{9}$$

```
BitActivity(state[],Cur,lev,NrStBits) {
 if(IsConst(Cur)&&(lev==2*NrStBits-1)){
  for(i=0;i<NrStBits;i++)
   if(state[i]!=state[i+NrStBits])//Toggle
  Activity[i]+=StateP[GetCS(state[])]*Cur.P;
 } else if(Cur.lev != lev)||IsConst(Cur)){
 // Expand tree
  state[lev]=0
  BitActivity(state[],Cur,lev+1,NrStBits)
  state[lev]=1
  BitActivity(state[],Cur,lev+1,NrStBits)
 } else { // Traverse Level
  state[lev]=0
  BitActivity(state[],Cur.l,lev+1,NrStBits)
  state[lev]=1
  BitActivity(state[],Cur.r,lev+1,NrStBits)
 }
}
```

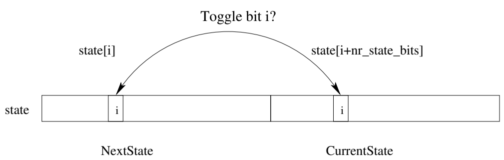

Fig. 5. Combined state vector

Example 4 *The equation 9 is used to calculate the next state activities in the following example.*

$$\begin{aligned} a(NS[0:0]) &= P_{SS}(00) \times P(NS_{01}|CS_{00}) + P_{SS}(01) \\ &\times \ (P(NS_{00}|CS_{01}) + P(NS_{10}|CS_{01})) \\ &+ \ P_{SS}(10) \times P(NS_{01}|CS_{10}) = 3/4 \times 1/4 \\ &+ \ 1/5 \times (3/4 + 1/4) + 1/20 \times 1/4 = 2/5 \\ a(NS[1:1]) &= P_{SS}(00) \times 0 + P_{SS}(01) \times P(NS_{10}|CS_{01}) \\ &+ \ P_{SS}(10) \times (P(NS_{01}|CS_{10}) \\ &+ \ P(NS_{00}|CS_{10})) = 0 + 1/5 \times 1/4 \\ &+ \ 1/20 \times (1/4 + 3/4) = 1/10 \end{aligned} \tag{10}$$

4. LOW POWER SYNTHESIS

We demonstrate the use of statistical information (signal probability and activity) for low power synthesis of BDD mapped circuits [5]. The next state and output functions represented by BDDs, are fed to the low power optimization tool, which seeks to minimize the estimated power dissipation by iterative variable reordering of the BDDs. The resulting BDDs can be directly mapped to a pass transistor logic circuit [6]. The power dissipation is based on such circuits.

5. EXPERIMENTAL RESULTS

In this section we preset experimental results from the ISCAS sequential benchmarks. We used the CUDD 2.3.0 BDD-package [8] and our synthesis tool [5]. The Area opt column contains the results for the area optimized circuits. The optimization done on these circuits are a heuristic minimization on the number of nodes in the BDD. The NonFSM opt column describes the low power synthesis approach without the FSM analysis. Instead it uses default values on the state vector for signal probabilities and switching activities. Finally we have the FSM opt column with the extracted signal statistics applied on the state vector and using our low power synthesis [5] tool. Size stands for the number of BDD nodes and $\hat{PD}$ is our estimate of power.

We have assumed $P(I[X:X]) = 0.5$ for all primary inputs.

The results show an average power reduction of 43 percent by the new proposed method compared to the area optimized method. The improvements range from 0 percent to 95 percent reduction.

The results show that the power optimized circuit has on average an increase of 51 percent area over the area optimized circuit.

The low power algorithm that use the default values on State bits for the activities $(a(S[X:X]) = 0.5)$ and probabilities $(P(S[X:X]) = 0.5)$ has a limited success on reducing power. For some benchmarks produce circuits with higher power dissipation than the area optimized, however on average it still has better results than the area optimized circuit.

	Area opt		NonFSM opt		FSM opt		Change
name	*Size*	$\hat{PD}$	*Size*	$\hat{PD}$	*Size*	$\hat{PD}$	$\hat{PD}$
s208.1	40	25	40	25	64	19	-24%
s27	9	4.1	9	4.1	9	4.1	0%
s298	73	4.3	74	4.2	77	2.9	-33%
s344	103	12	108	19	148	3.4	-72%
s349	103	12	108	19	127	3.3	-73%
s382	120	2.1	122	2.1	120	2.1	0%
s386	113	44	114	41	114	39	-11%
s400	120	2.1	122	2.1	120	2.1	0%
s444	150	43	161	19	156	2.1	-95%
s510	163	118	168	81	153	61	-48%
s526	137	8.4	139	8.1	136	4.6	-55%
s641	398	81	384	77	1149	15	-81%
s713	398	81	384	77	1149	15	-81%
s820	219	172	261	149	280	108	-37%
s832	219	174	261	148	294	103	-41%

Table 1. ISCAS89 benchmarks

6. CONCLUSIONS

As the experiments show we obtain significant reduction of power by analyzing the FSM and using this information in our synthesis algorithm[5]. These results show that the need for knowing the signal properties and applying that knowledge in the synthesis process is crucial for low power applications. We expect our approach to be further improved by incorporating low power state encoding techniques, for example by restructuring the FSM as in [4]. Methods for multi level synthesis using BDD based functional decomposition may increase the usability of the proposed approach. These are topics set for further research.

7. REFERENCES

[1] G. Cabodi, P. Camurati, and S. Quer. Improving symbolic reachability analysis by means of activity profiles. *IEEE Trans. on Comp.*, 19(9):1065–1075, 2000.

[2] G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Symbolic algorithms to calculate steady-state probabilities of a finite state machine. In *EDAC 94*, pages 214–218, 1994.

[3] G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Markovian analysis of large finite state machines. *IEEE Trans. on Comp.*, 15(12):1479–1493, 1996.

[4] B. Kumthekar, I. Moon, and F. Somenzi. A symbolic algorithm for low power sequential synthesis. In *Int'l Symp. on Low Power Electronics and Design*, pages 56–61, 1997.

[5] P. Lindgren, M. Kerttu, R. Drechsler, and M. Thornton. Low power optimization techniques for bdd mapped circuits using temporal correlation. In *technical report*, 2001.

[6] P. Lindgren, M. Kerttu, M. Thornton, and R. Drechsler. Low power optimization technique for BDD mapped circuits. In *Asia and South Pacific Design Automation Conference 2001*, pages 615–621, 2001.

[7] K. Roy and S. Prasad. *Low-Power CMOS VLSI Circuit Design*. Wiley Interscience, 2000.

[8] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.3.0*. University of Colorado at Boulder, 1998.