

Profiling-based State Assignment for Low Power Dissipation

Robbert Eggermont Sorin Cotofana Casper Lageweg
Electrical Engineering Department,
Delft University of Technology,
Mekelweg 4, 2628 CD,
Delft, The Netherlands
Email: {robbert,sorin,casper}@ce.et.tudelft.nl

Abstract—In this paper we address the problem of state assignment for Finite State Machines (FSMs). We target the reduction of power dissipation in FSM circuits by minimizing the switching activity in the state register. We introduce a novel state assignment method that utilizes dynamic loop information extracted from FSM profiling data. We propose three different loop-based state assignment algorithms: depth-first search (DFS), loop-based DFS and per-state encoding. The algorithms are implemented and evaluated on the standard FSM benchmark suite MCNC/LGSynth '89. Simulation results indicate that when compared with state-of-the-art state assignment algorithms for low power dissipation, our methods produce up to 14% average reduction of the switching activity in the state register.

Keywords—FSM, State Assignment, Low Power Design, Logic Design

I. INTRODUCTION

Finite state machine (FSM) state assignment for low power dissipation is usually based on the static FSM description [6] (POW3 [1], Nöth et.al. [5], JEDI [4], NOVA [7]). This however does not take into consideration an important aspect of the behavior of an FSM, i.e., the interaction between the FSM and the outside world.

In this paper we propose an FSM state assignment approach based on dynamic FSM state profiling, where the profiling data is obtained through the analysis of actual execution traces. The profiling data allows us to identify the most frequently executed states or sequences of states within the FSM. More specifically, our approach targets frequently executed cycles of states, i.e., loops. We propose the following three state assignment algorithms that utilize this profiling data to minimize the power dissipation of the FSMs: the depth-first search (DFS) algorithm, the loop-based DFS algorithm, and the per-state encoding algorithm. The DFS algorithm performs an exhaustive search of the FSM encoding space using the loop information for intermediate cost estimates of an encoding. The loop-based DFS algorithm performs a similar search on a loop-by-loop basis, where the loops are ordered in the de-

scending order of weight. The per-state algorithm encodes the states individually, on the same loop-by-loop basis.

The proposed algorithms have been implemented and evaluated on the standard FSM benchmark suite MCNC/LGSynth '89 [3]. Simulation results indicate that when compared with POW3 [1], a state-of-the-art state assignment algorithm for low power dissipation, our method produces 14.0%, 5.6%, and 6.4% average reduction of the switching activity in the state register for the respective algorithms. We note here that even though the DFS algorithm produces state encodings that lead to the highest average reduction this happens at the expense of an exponential computational time. Thus the DFS algorithm cannot be utilized for very large circuits. The last algorithm however requires a linear execution time and the quality of the generated encoding is comparable with the quality produced by the DFS based algorithms.

The remainder of the paper is organized as follows: Section II describes the general approach to FSM state assignment and explains the utilized terminology. Section III discusses the concept and implementation of FSM state profiling and loop detection. Section IV proposes three loop-based FSM state assignment algorithm for low power dissipation. Section V discusses the obtained simulation results. Section VI concludes the paper.

II. FSM STATE ASSIGNMENT

The objective of a state assignment algorithm is to assign a unique code (state register bit vector) to every state in the FSM. A state assignment algorithm has a significant influence on the power dissipation of the resulting FSM circuit. Through the state codes, the encoding determines not only the switching activity in the state register, but also the structure of the combinatorial logic circuit of the FSM and its switching activity. The state register switching activity can be determined by evaluating the (known) FSM's state transitions, but combinatorial logic synthesis is a complex, and time consuming, process, which makes

it unfeasible to evaluate the cost of each choice for the circuit switching activity. Therefore, low power state assignment algorithms consider only the state register switching activity.

In this paper it is assumed that an optimal solution of the state assignment problem is a solution which results in the minimal amount of switching activity in the state register. We present a general approach to FSM state assignment based on dynamic FSM state profiling data. This approach is based on the idea that the operation of many FSMs consists of recurring cycles (loops) of the same states, and that the FSMs spend most of their time walking through these loops. Therefore, the state assignments of the states in these loops have the largest impact on the overall switching activity in the FSM state register. By targeting these loops with our state assignment algorithm, the largest reduction in switching activity can potentially be realized.

The approach can be divided into the following three steps:

1. **FSM state profiling** collects information about the dynamic behavior of the FSM. A (simulated) FSM run under a relevant input data set generates an FSM state register trace, and from this trace, state and transition statistics are collected.
2. **Loop detection** searches for loops in the state trace. Loops are identified by the repeated occurrence of the same state in the trace, and each discovered loop is stored and counted to obtain the frequency of the loops.
3. **FSM state assignment** assigns each state of the FSM a unique code (bit vector) to represent it in the state register. The data gathered in the first two steps are utilized to minimize the switching activity in the FSM state register.

The FSM state assignment problem has a computational complexity that is exponential in the number of states S of the FSM: the best solution can only be determined by trying every combination of possible codes for the states of the FSM. The number of possible codes C is determined by the width, or number of bits B , of the state register: $C = 2^B$. To minimize the complexity of the assignment, the number of possible codes should be chosen as the smallest power of two needed to assign each state a unique code: $C = 2^{\lceil \log_2 S \rceil}$. The number of possible FSM encodings E then becomes:

$$E = \frac{C!}{(C - S)!}. \quad (1)$$

For FSMs with a large number of states, the exponential complexity makes it unfeasible to try every possible encoding even with a minimal number of possible encodings. Therefore, we present several algorithms that evolve from an exhaustive search algorithm with exponential computa-

tional complexity to a linear complexity state assignment heuristic featuring a “best guess” approach. Although the heuristic cannot guarantee an optimal solution, the computational complexity of the heuristic allows large size FSMs to be assigned. Even more, the computational complexity allows several runs of the heuristics, for example to compare different loop weighting strategies, and choose the best solution.

Given that our approach is based on profiling data which is obtained through the analysis of actual execution traces, the next section describes how FSM state profiling data is obtained and loops are detected.

III. FSM STATE PROFILING AND LOOP DETECTION

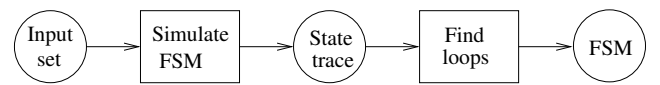


Fig. 1

FSM STATE PROFILING AND LOOP DETECTION.

For programs, code profiling means determining how often certain pieces of code are executed. We define FSM state profiling in the same way: state profiling determines how often a certain state is entered during an FSM run. FSM state profiling (Figure 1) records the state of the FSM during a (simulated) run of the FSM with a relevant input data set. From the resulting state register trace the loop frequencies, which determine the importance of a loop for the switching activity, can be derived.

FSMs consist of a finite number of states, between which the FSM switches during its operation. Unless the FSM enters a state from which no state transitions to other states are possible, it is very likely that the FSM at some time will enter a certain state for the second time. The FSM has no memory of its previous states, thus for the FSM there is no difference between the first and the second time the state was entered, and the FSM has in effect looped back to this state. We call the cycle, formed by the sequence of states from the first occurrence of a state (up) to the second occurrence of that same state, an FSM state loop.

Our FSM state assignment approach attempts to lower the FSM’s power dissipation by reducing the state register switching activity. Conflicting state sequences between loops inhibit an optimal encoding of all loops, therefore the best results are obtained by assuring that at least the loops that contribute the most to the overall state register switching activity are encoded optimally. This contribution, or weight, is a function of the loop’s frequency. Thus, our approach is most successful for FSMs that feature a small

number of loops with a significantly higher frequency than the other loops.

We propose a loop detection algorithm that takes a linear search approach, i.e., it performs a single analysis of the state trace, in one direction. The algorithm utilizes its own internal memory, called the minimal trace, to store a list of the states it encounters in the state trace. Before a state is added to the list, the algorithm performs a simple check for the presence of the encountered state in the minimal trace, which indicates the presence of a loop. The order of the states in the minimal trace matches the order of the states in the state trace, therefore the algorithm can determine the states in the loop, and the order of those states, from the minimal trace. When a loop is detected, the states in the loop are removed from the minimal trace, and the loop is added to the set of detected loops. If there is already a loop present that matches the states and the order of the states in the detect loop, the frequency count for that loop is simply incremented instead.

The removal of the detected loop serves an important purpose: removing the detected loop in effect removes the innermost loop from a set of nested loops within the state trace. When the loop is removed, the last state in memory is the last state of the outer loop before entering the innermost loop. The algorithm continues to add states to the memory until the outer loop is detected. This loop does not contain an inner loop and is counted as a separate, simple loop (hence the name *minimal trace*). This process continues until all loops of the nested set are detected.

The last step of each iteration adds the encountered state to the minimal trace, even when the state is part of a detected loop, because the state that joins an inner and an outer loop is an indispensable part of both loops, and the previous occurrence of the state has been removed from the memory.

In order to clarify the above we present the following trace-based loop detection example. Assume that the FSM depicted in Figure 2 has produced the following state trace: **B** → **a1** → **b1** → **b2** → **b3** → **b4** → **b1** → **b2** → **b3** → **b4** → **b1** → **b2** → **a2** → **a3** → **a4** → **a1** → **b1** → **b2** → **a2** → **E**. The trace contains two nested loops: the inner loop **b1** → **b2** → **b3** → **b4** (twice) and the outer loop **a1** → **b1** → **b2** → **a2** → **a3** → **a4**.

Table I demonstrates the iterations of the loop detection algorithm for this state trace. For each step it shows the state being checked, the minimal trace after the step, and if present, the detected loop. A loop is detected when the current state is found present in the minimal trace (indicated in **bold**). The loop is removed from the minimal trace, and added to the set of loops. After the check, the current state is added to the minimal trace.

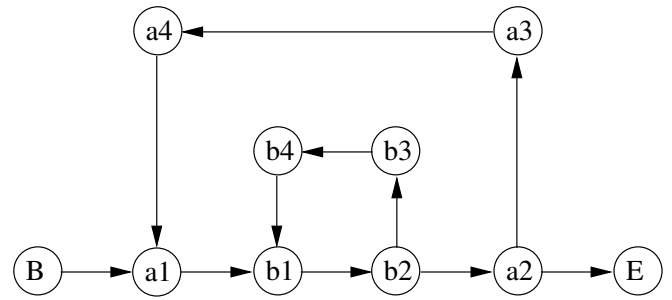


Fig. 2
FSM WITH NESTED LOOPS.

IV. LOOP-BASED FSM STATE ASSIGNMENT ALGORITHMS

In this section we propose the following three state assignment algorithms that utilize profiling data to minimize the power dissipation of the FSMs: the depth-first search (DFS) algorithm, the loop-based DFS algorithm, and the per-state encoding algorithm. The depth-first search (DFS) algorithm performs an exhaustive search of the FSM encoding space using the loop information for intermediate cost estimates of an encoding. The loop-based DFS algorithm performs a similar search on a loop-by-loop basis, where the loops are ordered in the descending order of weight. The per-state algorithm encodes the states individually, on the same loop-by-loop basis.

A. Basic DFS state assignment algorithm

The basic DFS state assignment algorithm finds the FSM state assignment solution with the lowest switching activity by trying every possible combination of state codes for the states. The algorithm performs a depth-first search (DFS) in a search tree, where each state is represented by a level, each node of a level corresponds with an unassigned code, and each incoming branch to a node symbolizes a state assignment. The top level (zero) represents the unassigned FSM. When the search reaches the bottom level, the resulting search path corresponds with a valid encoding for the FSM.

To find the best state encoding for low power consumption, the algorithm uses the state register switching activity as its cost metric, and minimizes this. The cost of an encoding is an estimate of the state register switching activity defined as:

$$cost = \sum_{l \in \text{loops}} (\text{frequency}(l) \times \sum_{t \in T_l} H(t)), \quad (2)$$

where T_l is the set of transitions in loop l , and $H(t)$ is the Hamming distance between the codes of the begin and end state of transition t . Therefore, the complexity of the

State	Minimal trace	Detected loop
B	B	
a1	B → a1	
b1	B → a1 → b1	
b2	B → a1 → b1 → b2	
b3	B → a1 → b1 → b2 → b3	
b4	B → a1 → b1 → b2 → b3 → b4	
b1	B → a1 → b1	b1 → b2 → b3 → b4
b2	B → a1 → b1 → b2	
b3	B → a1 → b1 → b2 → b3	
b4	B → a1 → b1 → b2 → b3 → b4	
b1	B → a1 → b1	b1 → b2 → b3 → b4
b2	B → a1 → b1 → b2	
a2	B → a1 → b1 → b2 → a2	
a3	B → a1 → b1 → b2 → a2 → a3	
a4	B → a1 → b1 → b2 → a2 → a3 → a4	
a1	B → a1	a1 → b1 → b2 → a2 → a3 → a4
b1	B → a1 → b1	
b2	B → a1 → b1 → b2	
a2	B → a1 → b1 → b2 → a2	
E	B → a1 → b1 → b2 → a2 → E	

TABLE I
ITERATIONS OF THE LOOP DETECTION ALGORITHM

cost estimate is linearly related to the combined number of states for all loops. After the evaluation of all possible search paths, the encoding with the lowest cost is assigned to the FSM.

The algorithm consists of two parts: the initialization routine and the DFS function. The initialization routine is the top level of the search. This routine sets the parameters for the search, initializes the variables and starts the search. The actual search is performed by the DFS function. When the search is completed, the encoding with the minimum cost is assigned to the FSM, and the algorithm finishes.

The DFS function recursively traverses the search tree of possible FSM encodings in a depth-first manner. Each level in the search tree corresponds with a state. A node on level L indicates a partial FSM encoding for the first L states. Each branch in the search tree corresponds with a code assignment to a state. Every time a leaf node (at the bottom) of the tree is reached, the search path corresponds with a possible FSM encoding, and the algorithm estimates the cost of that FSM encoding. If the cost of the encoding is lower than the minimum cost so far, the minimum cost is updated and the encoding is stored. When the entire tree has been traversed, the algorithm ends.

The search algorithm arbitrarily selects an unassigned state for the next level of the search tree. The available (unassigned) codes correspond with the nodes of that level. The assignment of a code to the state symbolizes the

traversal of a branch from the current level to a node on the next level. Then, this process is repeated for the next level(s). When all states are assigned, the cost of the encoding is calculated by using Equation 2. If the cost is less than the minimum cost found so far, the minimum cost is updated and the codes in the search path are stored as the minimum cost encoding. The algorithm returns to the previous level by releasing the code, i.e., going back up the branch.

As stated in section II, the basic DFS state assignment algorithm has an exponential computational complexity, which makes it impossible to find the best solution for an FSM with a larger number of states. To find solutions for larger FSMs, an heuristic based approach is required as explained in the following.

B. Loop-based DFS state assignment algorithm

The basic state assignment algorithm has an exponential computational complexity to the number of states, which makes it unsuitable for FSMs with many states. A well-known solution is to divide the FSMs in several smaller groups, and assign each group of states separately. This way the computational complexity is only exponential to the number of unassigned states in the largest unassigned group. However, the algorithm is only able to find the best solution if all groups are completely unrelated. However, in FSMs all states are connected, because all states can be

reached from the starting state of the FSM. Therefore the solution cannot be guaranteed to be optimal.

The best way to partition an FSM is to group together strongly connected states, so that at least the assignments within each group are optimal. The FSM is already partitioned into strongly connected groups in the form of the loops found during the FSM profiling, because the higher the frequency of a loop, the stronger the connection between its states. We propose a loop-based state assignment algorithm that assigns the states in a loop by loop manner in descending order of loop frequency. Thus the stronger connected groups are assigned first. Because several loops can contain the same state, loops do not divide the FSM in disjunct partitions, and the algorithm has to take into account the states that were encoded previously. Therefore, only the first loop is guaranteed to be assigned optimally, and other loops might not be assigned optimally.

The loop-based state assignment algorithm starts to encode the loops with the highest weight. The encoding of these loops has the largest impact on the overall cost, and because the pool of free codes is still quite full, it is possible to reduce the cost of a state assignment to a minimum. By optimally assigning the highest weight loops that contribute most to the state register switching activity, the overall switching activity should be reduced.

The loop-based state assignment algorithm consists of two parts: the setup part and the recursive DFS function. The setup sorts the loops in descending order according to the loop frequencies, and calls the DFS function for every loop. The DFS function optimizes on the loop level, therefore the assignment cost is only relevant within each loop. When all solutions for a loops have been tried, the best solution is assigned and the setup continues with the next loop.

Summarizing, loop-based DFS state assignment algorithm somewhat resembles the basic DFS state assignment algorithm as it performs optimal state assignment on individual loops within the FSM. The cost estimation function for the loop-based DFS state assignment algorithm can thus be deduced from Equation (2) as in this case the frequency is one, resulting in

$$cost = \sum_{t \in T_l} H(t), \quad (3)$$

where T_l is the set of transitions in the loop, and H is the Hamming distance between the codes of the from and to states of transition t . The DFS only processes the unassigned states of a loop, but the cost estimate takes into account all states, including the states that were assigned previously.

In the next section we describe a second heuristic based state assignment algorithm that also operates on a per-loop

basis.

C. Loop-based per-state heuristic state assignment algorithm

The previous state assignment algorithms both have an exponential computational complexity to the number of (unassigned) states in the FSM or the loop. In practice, when the number of states, and thus the computational complexity, is large, these methods cannot be used. We propose a loop-based heuristic state assignment algorithm that uses a “best guess” approach to assign a free code to an unassigned state. The heuristic has a linear computational complexity to the number of states in the FSM, which is very well suited to large FSMs. Because the cost of the generated solution depends heavily on the quality of the guess, the heuristic chooses a code based upon a minimal Hamming distance to its previous and next states.

The first step of the algorithm sets the number of latches (and thus the number of possible codes) of the FSM to the minimum required for the number of states in the FSM ($\lceil \log_2(\#states) \rceil$).

The second step of the algorithm assigns a weight to each loop according to a specified function. Our approach assumes a difference in the frequency of occurrence of loops, therefore the loop frequency obtained by the profiling algorithm is the primary variable of the weight function. However, the number of states in a loop can also be a factor, because the chance of finding an optimal encoding for large loops is higher when less states of the loops are assigned previously and more free codes are still available. Therefore, the following three weighing functions are considered:

- $weight(l) = frequency(l)$, where $frequency(l)$ is the occurrence frequency of loop l ,
- $weight(l) = frequency(l) \times states(l)$, where $states(l)$ is the number of states in loop l ,
- $weight(l) = frequency(l) \times bits(l)$, where $bits(l) = \lceil \log_2 states(l) \rceil$.

The first function only considers the frequency in which loops occur. The second function also considers the number of states within a loop. The third function is similar to the second but considers the actual number of required state storage bits. The main loop of the algorithm targets each loop in weight order, assigning codes to each unassigned state of the loop while minimizing the cost of the assignment. The algorithm needs to be aware of the states that were assigned earlier, as the codes of these states limit the freedom of choice for the codes of the unassigned states. The algorithm features three successive methods to address this problem, falling back to the next method if a method fails.

The first and most advanced assignment method uses both backward and forward dependencies on the codes of assigned states. This method requires the previous state to be assigned. To this end, for each new loop the algorithm starts by searching for a state that was already assigned. If such a state is present, the loop is rotated so that state becomes the first state in the loop. If no assigned state was present, the algorithm is free to assign an arbitrary code to the first state without cost penalties for the assignments in this loop. Subsequently, the algorithm searches forward in the loop to find the next state that is assigned (folding back to the first state in the loop if required). If the previous and next assigned state differ, the first method determines the bits differing between the previous state's code and the next assigned state's code. Each differing bit needs to be changed in some state assigned between the previous state and the next assigned state. Therefore, a code that only differs from the previous state's code by one of the differing bits does not increase the cost for the assignments of this loop. If such a code is found, it is assigned to the state and the algorithm continues with the next state.

The second method, which is utilized when the first method fails, determines the code based solely on the previous state's code. This method searches for a free code with a Hamming distance of 1 from the previous state's code. While this code is an optimal assignment for this state, it cannot be guaranteed that this code leads to an optimal assignment of the whole loop because this assignment could prevent an optimal assignment for some of the subsequent states.

The third method is the most expensive method. This method searches for a free code with the smallest cost. The cost of an assignment is based on the Hamming distance to both the previous state's code and the next assigned state's code, taking into account the distance (in states) to the next assigned state. The minimum cost is $1 + distance$, with a Hamming distance of 1 between the previous state and the current state, and a cost of $distance$ to reach the next assigned state. This is equal to the distance from the current state to the next assigned state. For example, if the next assigned state is separated from the current state by one state, or two states away, the minimal cost of the assignments up to the next assigned state is two: 1 for the minimal Hamming distance between the current state and the next state, and 1 for the minimal Hamming distance between the next state and the assigned state. When all free codes have been evaluated, or a code with the (minimal) cost of $1 + distance$ is found, the code with the minimal cost is assigned to the state.

When the algorithm has assigned all states in all loops, it assigns arbitrary codes to any unassigned states to com-

plete the FSM state assignment. As the unassigned states were not present in any loops, their assignments have little to no impact on the efficiency of the resulting state assignment.

For more details on the proposed algorithms we refer the reader to [2]. Some experimental results obtained by the algorithms are discussed in the next section.

V. EXPERIMENTAL RESULTS

In this section, we evaluate the proposed algorithms utilizing a finite state machine (FSM) benchmark suite, and compare the results with state-of-the-art state assignment algorithms. The algorithms are evaluated based upon the switching activity in the state register. We compare our new state assignment algorithms with two state-of-the-art low-power FSM state assignment algorithms, POW3 [1] and the algorithm by Nöth and Kolla [5], as well as with the base for all state assignment algorithm comparisons, the area-oriented JEDI [4] state assignment algorithm. The algorithms are evaluated by means of the industry-standard MCNC/LGSynth '89 FSM benchmark suite [3].

The experimental method consist of the following steps. The setup phase prepares the FSM descriptions and the input vector data sets. Subsequently, the FSM profiling simulates the FSM under an input vector data set to obtain the loops in the state trace. Note that this step is not required by the algorithms we compare with, as these algorithms all operate on the static FSM description. Each algorithm then performs a state assignment, either based on the profiling data or based on the static FSM description. During the simulation of the circuit, the switching activity of the FSM state register is measured in order to evaluate the overall power consumption.

We have applied our experimental method to all algorithms and for all FSM benchmarks and present a summary of the results in Table II. Each number represents the percentage of bit switches in the state register per state transition. The average result is calculated over the benchmarks for which all algorithms produced results. The simulation results indicate that when compared with POW3 [1], a state-of-the-art state assignment algorithm for low power dissipation, the basic DFS algorithm, the loop-based DFS algorithm and the per-state algorithm on average reduce the switching activity in the state register by 14.0%, 5.6%, and 6.4%, respectively. The state assignment algorithm by Nöth e.a. [5] utilizes wider state registers, which relieves one of the constraints of the other low power state assignment algorithms. Therefore, this does not allow of a fair comparison to the other algorithms.

We note here that, even though the DFS algorithm produces state encodings that lead to the highest average re-

Benchmark	DFS	Loop DFS	Heuristic	Nöth	POW3	JEDI
bbara	123.8%	129.0%	127.5%	126.0%	126.0%	142.5%
bbsse	115.1%	117.0%	116.0%	116.0%	116.1%	153.3%
bbtas	100.0%	100.0%	100.0%	100.0%	126.3%	135.1%
beecount	105.3%	107.4%	105.5%	105.5%	105.4%	107.7%
cse	104.6%	105.8%	105.8%	104.9%	108.0%	151.8%
dk14	134.1%	138.3%	142.3%	134.1%	144.8%	170.5%
dk15	119.8%	125.7%	124.6%	116.9%	119.8%	138.9%
dk17	122.8%	133.6%	125.9%	123.4%	123.5%	151.6%
dk27	118.8%	141.3%	132.1%	119.2%	135.6%	178.5%
dk512	118.4%	158.3%	146.9%	125.2%	153.2%	193.1%
donfile	-	193.7%	206.1%	174.8%	210.1%	244.3%
ex1	129.0%	143.3%	141.7%	134.8%	155.5%	216.4%
ex4	110.3%	114.4%	114.2%	-	-	205.5%
ex6	125.8%	148.4%	130.8%	-	-	157.9%
keyb	101.3%	102.0%	102.0%	101.3%	101.3%	118.5%
kirkman	100.0%	100.0%	100.0%	100.0%	-	162.5%
lion	100.0%	109.7%	100.0%	100.0%	100.0%	133.8%
lion9	100.0%	121.5%	139.5%	100.0%	161.8%	124.3%
mark1	130.2%	134.4%	137.3%	-	-	179.8%
mc	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
modulo12	100.0%	100.0%	100.0%	-	133.3%	100.0%
opus	127.3%	137.9%	132.4%	-	-	145.5%
planet	-	-	127.8%	117.2%	164.1%	338.9%
pma	121.4%	128.2%	130.6%	122.3%	-	203.1%
s1	-	176.6%	173.9%	148.9%	175.2%	182.3%
s1488	-	129.1%	115.0%	120.8%	-	210.7%
s1494	-	129.1%	115.0%	120.8%	-	210.7%
s208	108.6%	114.2%	108.6%	-	-	114.5%
s27	130.1%	146.2%	148.0%	132.2%	132.2%	133.7%
s298	-	-	155.2%	155.3%	-	277.0%
s386	115.2%	117.7%	116.2%	116.2%	-	134.2%
s420	108.6%	114.2%	108.6%	-	-	137.3%
s510	-	-	104.1%	104.1%	-	231.4%
s8	114.5%	117.4%	114.5%	114.5%	133.7%	114.5%
s820	101.8%	103.1%	103.0%	103.2%	-	296.1%
s832	101.8%	103.1%	103.0%	103.2%	-	295.0%
sand	-	-	135.9%	118.1%	156.4%	158.0%
scf	-	-	121.8%	-	-	366.3%
shiftreg	114.1%	146.0%	134.9%	114.1%	156.7%	171.2%
sse	115.1%	117.0%	116.0%	116.0%	-	153.3%
styr	108.0%	108.8%	112.2%	113.0%	112.2%	120.0%
tav	100.0%	100.0%	100.0%	100.0%	100.0%	150.0%
tbk	-	178.9%	182.7%	135.5%	190.6%	179.6%
tma	114.6%	118.6%	120.6%	128.9%	145.5%	228.9%
train11	122.4%	122.4%	123.4%	123.5%	150.7%	152.2%
train4	100.0%	100.0%	100.0%	100.0%	150.0%	100.0%
Average	112.9%	123.8%	122.8%	115.4%	131.2%	148.6%

TABLE II

STATE REGISTER BIT SWITCHES PER STATE TRANSITION FOR DIFFERENT ALGORITHMS AND BENCHMARKS. THE AVERAGE RESULT IS CALCULATED OVER THE BENCHMARKS FOR WHICH ALL ALGORITHMS PRODUCED RESULTS.

duction, this happens at the expense of an exponential computational time. Thus the DFS algorithm cannot be utilized for very large circuits. The last algorithm however requires a linear execution time and the quality of the generated encoding is comparable with the one produced by the DFS based algorithms.

VI. CONCLUSIONS

In this paper we addressed the problem of state assignment for Finite State Machines (FSMs). We targeted the reduction of power dissipation in FSM circuits by minimizing the switching activity in the state register. We introduced a novel state assignment method that utilizes dynamic loop information extracted from FSM profiling data. Trading off solution quality for computational effort we proposed three different loop-based state assignment algorithms: depth-first search (DFS), loop-based DFS and per-state encoding. The algorithms were implemented and evaluated on the standard FSM benchmark suite MCNC/LGSynth '89. Simulation results indicated that when compared with state-of-the-art state assignment algorithms for low power dissipation, our methods produced up to 14.0% average reduction of the switching activity in the state register.

REFERENCES

- [1] L. Benini and G. Micheli. State assignment for low power dissipation. *IEEE Journal of Solid-State Circuits*, 30:258–268, March 1995. Available from: <http://citeseer.ist.psu.edu/benini95state.html>.
- [2] R. Eggermont. PROSA: Profiling-based State Assignment for Low Power Dissipation. Master's thesis, Electrical Engineering Department, Delft University of Technology, NL, 2003. CE-MS-2003-11. Available from: http://ce.et.tudelft.nl/publicationfiles/772_6_thesis.pdf.
- [3] LGSynth89 Benchmark Suite. Available from: http://www.cbl.ncsu.edu/CBL_Docs/lgs89.html.
- [4] B. Lin and A.R. Newton. Synthesis of Multiple Level Logic from Symbolic High-Level Description Languages. In *Proceedings of the International Conference on VLSI*, pages 187–196, Munich, August 16 - 18 1989.
- [5] Winfried Nöth and Reiner Kolla. Spanning Tree Based State Encoding for Low Power Dissipation. Technical report, Department of Computer Science, University of Würzburg, 1998. Available from: <http://citeseer.ist.psu.edu/25222.html>.
- [6] Chi-Ying Tsui, Massoud Pedram, and Alvin M. Despain. Exact and approximate methods for calculating signal and transition probabilities in FSMs. In *Design Automation Conference*, pages 18–23, 1994. Available from: citeseer.ist.psu.edu/tsui94exact.html.
- [7] Tiziano Villa and Alberto L. Sangiovanni-Vincentelli. NOVA: State Assignment of Finite State Machines for Optimal Two-level Logic Implementations. In *Proceedings of the 1989 26th ACM/IEEE conference on Design automation conference*, pages 327–332, 1989.