Synthesis tool for low-power finite-state machines with mixed synchronous/asynchronous state memory

C. Cao, M. O'Nils and B. Oelmann

Abstract: An efficient way to obtain finite-state machines (FSMs) with low-power consumption is to partition the machine into two or more sub-FSMs and then use dynamic power management where all sub-FSMs not active are shut down, with the effect of reducing dynamic power dissipation. Thus, FSM partitioning algorithms and register-transfer-level power estimation functions are the main focus of the paper as these are key issues in the design of a computer-aided design tool for synthesis of low-power partitioned FSMs. An implementation architecture is targeted, which is based on both synchronous and asynchronous state memory elements that enable larger power reductions than fully synchronous architectures do. Power reductions of up to 77% have been achieved at a cost of an 18% increase in area.

1 Introduction

Power optimisations at the architectural level often involve some kind of dynamic power management (DPM) scheme that reduces the dynamic power consumption [1]. Whenever DPM is applied, the original design has to be partitioned into two or more units in such a way that they can be dynamically shut down when idle. An automated optimisation procedure takes the original design description, along with the statistics for the primary input signals, to a partitioning algorithm with cost-functions that seeks the best partition. The number of possible partitions (candidates) for non-trivial problems was too large to explore. Therefore, an algorithm for selecting only the most promising candidates was necessary. Among these candidates, one should be ranked the best. Here, it was crucial to have accurate cost functions despite a lack of detailed information of the final implementation.

2 Background

The initial design description for most approaches to the partitioned finite-state machine (FSM) design is the synchronous state transition graph (STG). Partitioning, cost estimations and transformations are done on the STG. The first step is typically to identify clusters of states with high mutual state transition probabilities. These states are said to be strongly connected. The objective is to find small clusters with strongly connected states because they will result in small sub-FSMs that are active most of the time and lead to a low average power consumption. Each sub-FSM requires circuitry for an idle condition detection and a shut-down mechanism, both constituting a functional overhead. The partitioner must seek the most beneficial idle conditions, taking this overhead into account. In the early work by Benini *et al.* [2], the so-called self-loops with high transition probabilities were implemented as separate sub-FSMs. This work was generalised to involve clusters of many states [3]. The major power-overhead introduced in a partitioned FSM comes from the fact that at the event of a crossing transition (a state transition from a state residing in another sub-FSM other than the destination state), two sub-FSMs have to be clocked in that cycle to complete the transition making it very costly [4].

Oelmann *et al.* [5] have shown that by introducing multiple exit states in each sub-FSM, the sub-FSM does not need to return to a single 'reset state' [3], which makes it possible to remove the double-clocking requirement. By allowing asynchronous state changes and having multiple exit states, two state changes can be made in the same clock cycle. Another advantage of using asynchronous control is that the capacitive load on the free-running global clock decreases and power can subsequently be reduced. This method can be viewed as the exploitation of the mixed synchronous/asynchronous method at register-transfer-level (RT-level) for FSM low power design. Unlike in the work of Chapiro [6], where the environment of the globally asynchronous locally synchronous system is asynchronous, the mixed-mode circuit is externally synchronous while internally combining synchronous and asynchronous methods.

The straightforward way to implement a partitioned FSM is to have separate state memory for each of the sub-FSMs [3] (Fig. 1*a*). Alternatively, the state memories can be shared by all the sub-FSMs since only one is active at a time [7]. The main advantage here is a reduced area for flip-flops. In this case there is, however, a need for a global state determining that one of the sub-FSMs is active (Fig. 1*b*). The global state memory (GSM) needs to be clocked by the global clock and this adds substantially to the power consumption.

To further explore the potential of asynchronous circuits, in contrast to Oelmann *et al.* [5] and Chapiro [6] where the asynchronous part is only used as the interface, the CAD-tool discussed in this paper targets the mixed

IEE Proceedings online no. 20050048

doi:10.1049/ip-cdt:20050048

Paper first received 13th March and in revised form 15th September 2005

The authors are with the Department of Information Technology and Media, Mid Sweden University, SE-85170 Sundsvall, Sweden

E-mail: cao.cao@miun.se

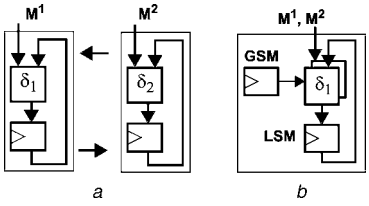

Fig. 1 *Structural decomposition of FSM*

a Separate state memory
b Shared state memory

synchronous/asynchronous architecture, developed by Cao and Oelmann [8], that has a shared synchronous local state memory (LSM) together with a global asynchronous state memory. The basic idea was to have a synchronous memory in the always clocked part, that is, LSM, and asynchronous memory for the GSM, which has a low probability of being updated. In this way, the GSM only adds a very low power-overhead. The shut-down mechanisms used are input-gating to reduce power dissipation in the idle combinational logic and gated-clocks to shut down flip-flops that are temporarily not needed in the LSM. Despite the introduction of asynchronous circuits, the input/output behaviour of the partitioned FSM is still cycle-by-cycle equal to the original unpartitioned synchronous one.

3 Introduction to the design model

In our design model, partitioned sub-FSMs share the same synchronous LSM and control of which one of the sub-FSMs should be active is governed by the asynchronous GSM.

3.1 STG transformation

In order to handle crossing transitions, the original STG (representing the behaviour of the monolithic FSM), was transformed to support an interaction scheme for asynchronously activating and deactivating the sub-FSMs. The example in Fig. 2 serves as an illustration of the STG transformation.

After FSM partitioning, the STG in Fig. 2 was divided into two sub-FSMs (M^1 and M^2) with state subsets $S^1 = \{s_1, s_4, s_6\}$ in M^1 and $S^2 = \{s_2, s_3, s_5, s_7\}$ in M^2. There are three crossing transitions between M^1 and M^2. In order to detect a crossing transition, an extra g-state was introduced. A g-state is inside the sub-FSM that contains the source state of a crossing transition and it has the same index as that of the destination state. After the STG transformation, two new state subsets were formed as $U^1 = \{s_1, s_4, s_6, g_2\}$ in M^1 and $U^2 = \{s_2, s_3, s_5, s_7, g_1, g_6\}$ in M^2. The transformed STG is shown in Fig. 3 where input/output is ignored for clarity.

The behaviour of a crossing transition changes after the introduction of a g-state. Take the crossing transition $s_6 \rightarrow s_2$ as an example (Fig. 3). After g_2 was introduced in M^1, the first event was the transition $s_6 \rightarrow g_2$, inside M^1. Then, at the second event, the detection of g_2 made the asynchronous state memory update its state from r_1 to r_2 (labelled as r_1^-, r_2^+ on the edge). The global states r_1 and r_2 indicate the active sub-FSMs M^1 and M^2, respectively. After the completion of the asynchronous transition, M^1 was deactivated and M^2 activated.

The entire crossing transition was completed within one clock cycle. The first event was synchronous because the LSM was updated to the g-state at the active edge of the clock signal. States s_6 and g_2 should be distinguished

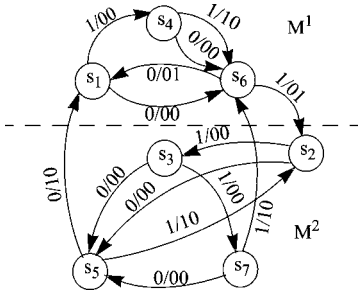

Fig. 2 *FSM example dk27*

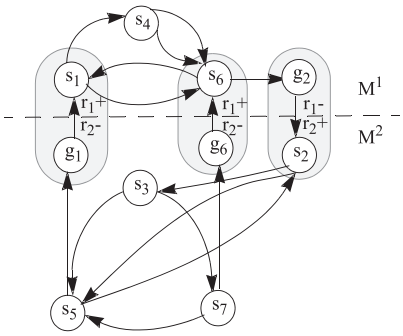

Fig. 3 *Transformed STG in decomposed FSM*

from the local state code when sharing the same global state r_1. The second event was asynchronous because the GSM was updated to r_2 immediately upon detection of the transition in the g-state. The LSM was only triggered by the clock signal and therefore remained unchanged. In this example, g_2 and s_2 should share the same local state code whereas their global states are different.

A group of states with identical local state codes and different global states is called a state-bundle. A state-bundle including a g-state is called a g-state-bundle. In Fig. 3, there are three g-state-bundles (g_1, s_1), (g_2, s_2) and (g_6, s_6), indicated with circles in a shaded grey area.

3.2 State bundling and encoding

In Section 3.1, we have proposed the g-state-bundle and state-bundle concept through an example. The reasons for state bundling are: (1) it enables states to share the same local state code; (2) it enables an efficient asynchronous hand-over mechanism; and (3) the g-state-bundle enables an efficient clock gating implementation.

After the STG transformation, a state-bundle table was built. Every column of the table represents a state-bundle, including a set of states with the same local state code but with different global state code. Every row of the table represents the states in a sub-FSM that have the same global state code. The number of rows is the same as the number of sub-FSMs. As the states in a g-state-bundle should be put in the same column, we built the table beginning with g-state-bundles.

To be specific, let us examine the example in Fig. 3 again. Its state-bundle table was built with two rows, representing M^1 and M^2, and $\max(|U^1|, |U^2|) = 6$ columns, representing the larger number of states in a single sub-FSM (g-states are also included). Firstly, three g-state-bundles were put into the table cells (shaded grey). Other states in each sub-FSM were then put into the table ordinally from the leftmost empty cell. After building the table, six

Table 1: State-bundle table

B	b_0	b_1	b_2	b_3	b_4	b_5
M^1	s_1	s_6	g_2	s_4		
M^2	g_1	g_6	s_2	s_3	s_5	s_7

state-bundles were obtained and the state transition inside a sub-FSM could be viewed as the state-bundle transition.

Let us observe the crossing transition $s_6 \rightarrow s_2$ again. From Table 1, this transition can be explained in the following sequence: (1) a horizontal state transition from state-bundle b_1 to b_2 inside the LSM and (2) a vertical state transition from M^1 to M^2 in the GSM. Note that the state-bundle remains b_2.

Binary codes were assigned to the columns from left to right in incremental order. The environment makes sure that the asynchronous GSM operates in a fundamental mode [9]. By using the specified one-hot encoding to the global states, a hazard-free implementation was ensured.

3.3 Implementation structure

Suppose the monolithic machine has I as input, O as output and is partitioned into sub-FSMs $M^1, M^2, \ldots, M^n$. The original state subsets $S^1, S^2, \ldots, S^n$, combining the introduced g-states, form the new state subsets $U^1, U^2, \ldots, U^n$ for $M^1, M^2, \ldots, M^n$, respectively. All sub-FSMs share the same LSM but have their own combinational logic. Our decomposed FSM structural model is shown in Fig. 4.

The g-state-bundle detection logic (GDL) decodes the state bits in the LSM. If a g-state-bundle is detected, a signal is sent to the GSM.

GSM decides the current active sub-FSM. It is implemented as an asynchronous finite state machine. A state transition in the GSM only takes place at the event of a crossing transition, that is, when a g-state has been detected. In a 'well-partitioned' FSM, where the probability of a crossing transition is low, the GSM will be idle most of the time and will therefore dissipate no dynamic power. The state information in the GSM is directly used as control signals to both the LSM and the combinational part (implementing the next state function) of the sub-FSMs (labelled $M^1, \ldots, M^n$ in Fig. 4).

For an active sub-FSM M^i, as the number of local state bits needed can be smaller than the total number of local state bits, the flip-flops that are currently not in use are disabled by clock gating. The global state controls the clock gating.

At any given time, except for the events of crossing transitions, only one sub-FSM is active. The active sub-FSM is responsible for determining the primary output and the next local state. When inactive, all its inputs are disabled by AND gates and no dynamic power will be dissipated. All outputs of an inactive sub-FSM are set to zero. By using OR gates, the correct primary outputs and next state outputs can be obtained by collecting corresponding outputs from all sub-FSMs.

It is known that the number of state bits into the combinational logic of a sub-FSM is important to its implementation size and is also related to the power dissipation. This partitioning of a FSM results in a lower number of bits in the state code and input bits for the sub-FSMs. Reduction in both area and power can thus be achieved. Large power reductions were obtained when a good partitioning was found where a small sub-FSM was active most of the time.

For more details on the implementation of the partitioned FSM on the basis of synchronous/asynchronous state memory, we refer the reader to the work of Cao and Oelmann [8].

4 Design flow and tool

As shown in Fig. 5, the tool accepts standard benchmarks in the kiss2 format [10] whose behaviour can be described as unoptimised synchronous STGs. For each input, the switching activity and signal probability are given. A standard-cell-based design flow is assumed, which means that there are no special requirements on the library that goes beyond what is normally provided. However, the tool requires some cell library dependent information in order to make accurate power estimations and gate-level synthesis of the asynchronous elements.

In order to enable power estimation, the first step is to generate necessary statistics for the FSM. From the behavioural FSM description (STG) and the primary input probabilities, we obtain the state transition probabilities, state probabilities and output statistics, the transition and output functions. On the basis of their mutual state transition probabilities the states are clustered. Then an algorithm is used for selecting the candidates most likely to give the best partition. With a limited number of candidates, more accurate RT-level power estimation is made. Each candidate is synthesised to a RT-level description and power consumption is estimated. From these results, the best candidate is selected and a RT-level very high-speed integrated circuit hardware description language (VHDL) code is generated along with synthesis scripts for logic synthesis in a standard tool.

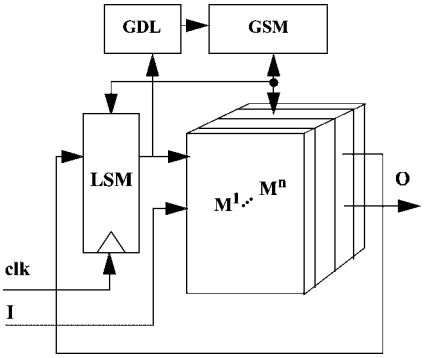

Fig. 4 *Mixed synchronous/asynchronous FSM structure*

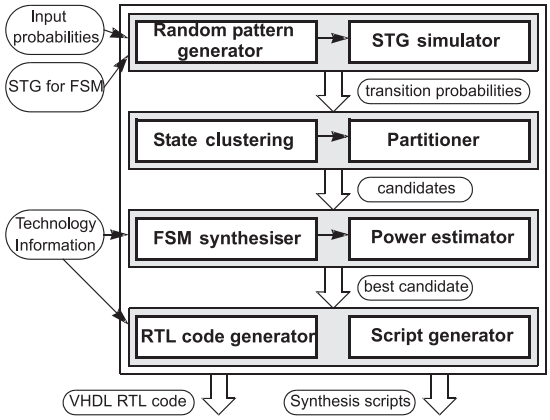

Fig. 5 *Overview of the tool*

5 Automatic synthesis of partitioned FSMs

5.1 State clustering

The original state transition graph can be looked upon as an edge-weighted undirected graph $G(V, E)$. A binary tree is built by recursively applying the Kernighan–Lin two-way partitioning, where states are clustered depending on their state transition probability for minimising the crossing transitions between two sets. Redundant states, that in a later stage are discarded, are introduced to V initially, forming V', to make sure $|V'|$ (number of vertices) is the power of 2. Assuming $|V'| = 2n$, the complexity of this algorithm is $O(n^2 \log n)$. For the benefit of the second phase, the tree is built with the left-hand cluster having higher static probability. The left-most cluster at each level then has the highest static probability. Take benchmark dk27 [10], which has seven states, as an example. After introducing one redundant state (8), a full binary tree is built as shown in Fig. 6.

5.2 Candidate generation

We have proposed an efficient algorithm that combines clusters at each level of the binary tree for generating the partitioning candidates. For n states, this algorithm finds candidates ranging from 1-way to n-way partitioning with a complexity of only $O(n \log^3 n)$. Within a limited number of candidates, a good partition with low power can be found. Applying the algorithm, given in Fig. 7, on the binary tree shown in Fig. 6, candidates are generated as shown in Fig. 8.

Level 1: {1,2,3,4,5,6,7,8} 1 Cluster
Level 2: {2,3,5,7},{1,4,6,8} 2 Clusters
Level 3: {2,5},{3,7},{1,6},{4,8} 4 Clusters
Level 4: {2},{5},{3},{7},{6},{1},{4},{8} 8 Clusters

Fig. 6 *Full binary tree for dk27*

```
Candidate_Select(set of Clusters ClusterTree) {
    for (level ← 1; level< clusterTree.depth(); level ← level+1) {
      Clusters C ← cutlevel(clusterTree, level);
      int N ← C.size();
      for (base ← 1; base< N; base ← base+1) {
        Clusters P_base ← {c_1}, ... ,{c_base};
        Clusters P_restbase ← {c_base+1}, ... ,{c_N};
        Clusters TMP ← P_base ∪ P_restBase;
        candidates ← candidates ∪ TMP;
        int restBase ← N - base;
        int place ← N;
        int r ← 0;
        if (restBase > 2) {
          r←restBase;
          while (r > 0) {
            int i ← ⌊log_2 r⌋;
            int d ← 2^i;
            int q ← (r - mod(r,d))/d;     // the quotient of r/d
              r ←mod(r,d);                // the residue of r/d
            for (j ← q; j > 0; j ← j-1) {
              P_restBase ← {c_place-d+1}, ... ,{c_place};
              place ← place - d;
              TMP ← P_base ∪ P_restBase;
              candidates ← candidates ∪ TMP;
            }
          }
        }
      }
    }
    return candidates;
}
```

Fig. 7 *Algorithm for selecting candidates*

Level 1: {1,2,3,4,5,6,7,8}
Level 2: {2,3,5,7},{1,4,6,8}
Level 3: {2,5},{3,7,6,1,4,8};{2,5},{3,7},{6,1,4,8}; {2,5},{3,7},{6,1,4,8}; {2,5},{3,7},{6,1},{4,8}

Level 4:
{2},{5,3,7,6,1,4,8};{2},{5},{3,7},{6,1,4,8};{2},{5},{3,7},{6,1},{4,8};
{2},{5},{3,7,6,1,4,8};{2},{5},{3,7},{1,6,4,8};{2},{5},{3,7},{6,1},{4,8};
{2},{5},{3},{7,6,1,4,8};{2},{5},{3},{7},{1,6,4,8};{2},{5},{3},{7},{,6,1},{4,8};
{2},{5},{3},{7},{6,1,4,8};{2},{5},{3},{7},{6,1,4,8};{2},{5},{3},{7},{6,1},{4,8};
{2},{5},{3},{7},{6},{1,4,8};{2},{5},{3},{7},{6},{1},{4,8};
{2},{5},{3},{7},{6},{1},{4,8};
{2},{5},{3},{7},{6},{1},{4},{8}

Fig. 8 *Generated candidates for dk27*

5.3 Power estimation

The power estimation functions are used on the partitioning candidates obtained in Section 5.2 to find the best one with lowest power. For both the asynchronous global and synchronous local state memories the gate-level implementations are known. It is not the same as the combinational logic as that requires different power estimation techniques. From the STG simulator, input and output statistics are obtained and used for the power estimation.

5.3.1 Power estimation for combinational logic: An entropy-based power estimation approach proposed in Nemani and Najm [11] was used for the combinational logic. The transition table together with entropy for the combinational logic, based on the switching activity of the inputs and the outputs, were used

$$P_{\text{comb}} = \sum_{i=1}^{n} H_i \times \text{row}_i \times k_{\text{tech}} \times T_{\text{com}_i}$$

where n is the number of sub-FSMs, H_i the entropy of the logic, row_i the number of rows in the state transition table originating from the sub-FSM M^i, k_{tech} an empirically determined constant to adjust to the cell library used, and T_{com_i} the probability of M^i being active.

5.3.2 Power for GSM: For the GSM, we use an empirical model that is based on the structure of the memory. Although the gate-level implementation is known, we found it more accurate to use the macro model shown below, which consists of two parts: the expression in parentheses representing the power of the logic that detects and initiates the transition from one sub-FSM to another and the summation term representing the asynchronous state memory elements

$$P_{\text{GSM}} = (k_B \times p_{\text{LSM}-B} + k_G \times p_G + k_g \times |g|) + \sum_{i=1}^{n} P_C \times T_{C_i}$$

The expression inside the parenthesis estimates the power in the global state transition function, which is a function of the local state and the global state. The first term represents the contribution from the LSM where $p_{\text{LSM}-B}$ is the toggle probability of local state bits. The second term represents the contribution from the global memory where p_G is the sum of toggle probabilities of the g-states. A g-state is a local state that initiates a global state transition. The third term represents the complexity of global state transition logic where $|g|$ is the number of g-states. The summation term represents the contribution from the GSM devices, implemented as Muller C-elements where T_{C_i} is the

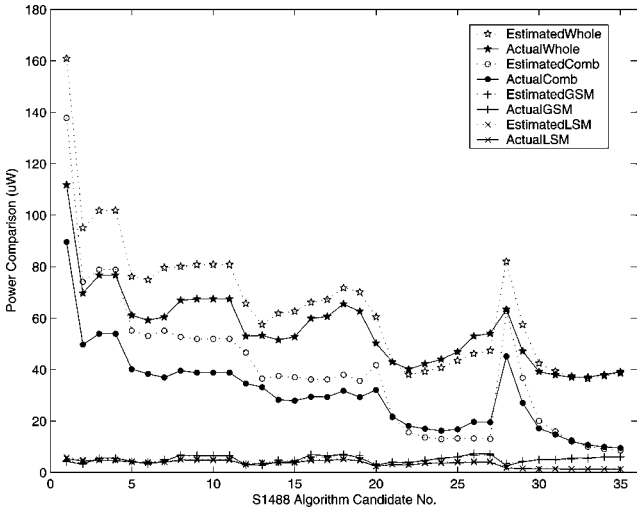

Fig. 9 *Cost-function verification*

probability of global state transition, which is the probability of crossing transitions related with the sub-FSM M^i. The number of C-elements is the same as the number of sub-FSMs and is denoted n. The constants k_B, k_G and k_g are determined empirically and are based on a single FSM partition run.

5.3.3 Power estimation for D-flip/flop: The LSM consists of a set of D-flip-flops and is estimated by

$$P_{\text{LSM}} = \sum_{i=1}^{m} P_{\text{DFF}_i} \times T_{D_i}$$

where T_{D_i} is the probability of the flip-flop i being active and m is the number of LSM bits.

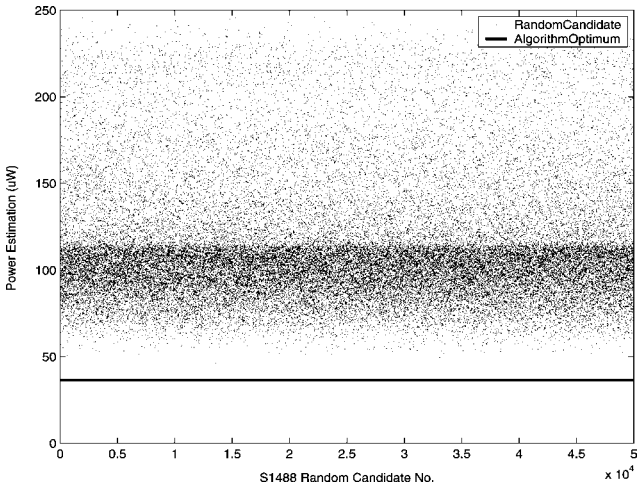

Fig. 10 *Algorithm verification*

5.3.4 Power estimation of clock net: The power dissipation in the clock net is estimated by

$$P_{\text{clock}} = |\text{FF}| \times C_{\text{clkin}} \times f \times V_{\text{DD}}^2 \times k_{\text{buffer}} \times k_{\text{wire}}$$

where |FF| is the average number of flip-flops clocked, C_{clkin} the capacitance of the clock input, V_{DD} the power supply voltage, f the clock frequency, k_{buffer} the clock buffer capacitance coefficient and k_{wire} is the wire capacitance coefficient.

5.3.5 Power estimation for partition overhead circuits:

$$P_{\text{overhead}} = P_{\text{GatedCom}} + P_{\text{GatedFF}}$$

where P_{GatedCom} includes the power of AND gates for activating and deactivating the combinational logic, also the OR gates for merging the output; P_{GatedFF} is the power for activating and deactivating the local state bits and basically are NAND gates.

The power of basic gates, such as the D-flip-flops, AND gate, originates from the standard-cell library.

The power dissipation for the whole partitioned FSM is simply a sum of the above

$$P_{\text{whole}} = P_{\text{comb}} + P_{\text{GSM}} + P_{\text{LSM}} + P_{\text{clock}} + P_{\text{overhead}}$$

6 Results

The accuracy of the power estimation functions is verified by comparing the estimated power before and after logic synthesis. As a reference, we use power compiler (Synopsys) for gate-level power estimation. In Fig. 9, for the benchmark s1488, the results from the estimation functions P_{whole}, P_{Comb}, P_{GSM} and P_{LSM} (labelled Estimated) and the results from power compiler (labelled Actual) can be compared. A 0.18 μm technology is used with V_{DD} of 1.8 V, clock frequency of 20 MHz. The primary input probability and switching activity are both set to 0.5. A series of candidates chosen from each level of the partitioning tree of three different FSM benchmarks (s820, keyb and s1488) were used in this verification. It can be seen that estimation functions match well with the results from the gate-level estimations. The correlation coefficient that measures the extent to which two sets of data match with each other, is used for verifying the cost-function. The reason for using the correlation coefficient is that we want to find a candidate with the actual lowest power (also the candidate with the lowest estimated power). Therefore the absolute difference of these two values of power is not important. The coefficient between estimated and actual power for the whole partitioned design P_{whole} is 0.77 for s820, 0.98 for s1488 and 0.88 for keyb.

It is crucial that the candidate generation algorithm finds the candidate with the lowest power consumption. In order to verify that, we randomly generated 50 000 partitions of the s1488 and compared them to the one selected

Table 2: Results for standard benchmarks [8]

FSM	A.O. gates	P.O. (μW)	n	A.D. gates	P.D. (μW)	A (%)	P (%)	cpu (s)
s1488	925	160	7	1090	37	18	77	2.7
s820	444	75	3	630	41	42	45	0.9
s1494	900	141	7	1092	38	21	73	3.3
s832	467	80	2	534	36	14	55	0.9
keyb	271	72	5	436	34	61	53	0.9
scf	786	80	3	1067	54	36	33	12.7

by the tool. From Fig. 10, it can be seen that none of the randomly generated partitions is better than the one selected by the tool.

To illustrate the overall performance of our tool, a comparison between the original monolithic FSM and the multi-way partitioned FSM is shown in Table 2. The columns labelled 'A.O.' and 'P.O.' represent the area and power of the original monolithic FSM, respectively; the column labelled '*n*' represents the number of sub-FSMs after partitioning; 'A.D.' and 'P.D.' represents the area and power of the decomposed FSM, respectively; the following two columns represent the percentage of area increase and power reduction of the decomposed FSMs separately.

The CPU times in Table 2 are for the state clustering and candidate generation algorithms executed on a Pentium 4, 1.6 GHz processor running Windows 2000. The total time for the largest benchmark (scf 121 states) is 5 min, which shows that the most time consuming part is FSM synthesis to RT-level and power estimation. This supports our idea of the importance of having a candidate selection algorithm that limits the number of candidates early.

7 Discussions and conclusions

In this paper, we have presented a novel multi-way partitioning algorithm for partitioned FSM synthesis. Although we have applied it to a mixed synchronous/asynchronous architecture, it can also be used for fully synchronous implementations. We have also presented RT-level power estimation functions that have sufficient accuracy for selecting the candidate with the lowest power consumption. The proposed algorithms have low complexity, which is important when it comes to practical usage of the tool. The tool, as shown in Fig. 5, has been completely implemented in C. It fits into a standard-cell based design flow and is fully compatible with the Synopsys tool set. Our tool reduced power consumption significantly, on average 56%, for the benchmarks considered in this paper.

8 References

1 Benini, L., and de Micheli, G.: 'Dynamic power management: design techniques and CAD tools' (Kluwer Academic Publishers, 1998)

2 Benini, L., Siegel, P., and de Micheli, G.: 'Saving power by synthesizing gated clocks for sequential circuits', *IEEE Des. Test Comput.*, 1994, **11**, pp. 32–41

3 Benini, L., Vermeulen, F., and de Micheli, G.: 'Finite-state machine partitioning for low-power'. Proc. IEEE Int. Symp. on Circuits and Systems, 1998, vol. 2, pp. 5–8

4 Oelmann, B., and O'Nils, M.: 'Asynchronous control of low-power gated-clock finite-state machines'. IEEE Int. Conf. on Electronics, Circuits and Systems, 1999, pp. 915–918

5 Oelmann, B., Tammemäe, K., Kruus, M., and O'Nils, M.: 'Automatic FSM synthesis for low-power mixed synchronous/asynchronous implementation', *J. VLSI Design*, (special issue on low-power design), 2001, **12**, (2), pp. 167–186

6 Chapiro, D.M.: 'Globally-asynchronous locally-synchronous systems'. PhD Thesis, Stanford University, 1984

7 Tsui, C.Y., Pedram, M., and Despain, A.M.: 'Low power state assignment targeting two and multilevel logic implementations', *IEEE Trans. Comput. Aided Des.*, 1998, **17**, (12), pp. 1281–1291

8 Cao, C., and Oelmann, B.: 'Mixed synchronous/asynchronous state memory for low power FSM design'. Proc. EUROMICRO Symp. on Digital System Design, 2004, pp. 363–370

9 Unger, S.H.: 'Asynchronous sequential switching circuits' (Wiley-Interscience, 1969)

10 Yang, S.: 'Logic synthesis of optimization benchmarks – user's guide, version 3.0'. MCNC, Technical Report, 1991

11 Nemani, M., and Najm, F.: 'Towards a high-level power estimation capability', *IEEE Trans. Comput. Aided Des. Int. Circuits Syst.*, 1996, **15**, (6), pp. 588–598