A Tool for Low-Power Synthesis of FSMs with Mixed Synchronous/Asynchronous State Memory

Cao Cao, Mattias O’Nils, and Bengt Oelmann
Department of Information Technology and Media, Mid Sweden University
S-851 70 Sundsvall, Sweden; {cao.cao, mattias.onils, bengt.oelmann}@mh.se

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 use dynamic power management, where all sub-FSMs not active are shut down, to reduce dynamic power dissipation. In this paper we focus on FSM partitioning algorithms and RT-level power estimation functions that are the key issues in the design of a CAD tool for synthesis of low-power partitioned FSMs. We target an implementation architecture that is based on both synchronous and asynchronous state memory elements that enables larger power reductions than fully synchronous architectures do. Power reductions of up to 77% have been achieved at a cost of an increase in area of 18%.

1. Introduction

Power optimizations at the architectural level often involves some *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 dynamically can be “shut down” when idle. An automated optimization procedure will take the original design description along with statistics for the primary input signals to a partitioning algorithm with cost-functions that seeks for the best partition. The number of possible partitions (candidates) are, for non-trivial problems, too large to explore. Therefore, an algorithm for selecting only the most promising candidates is required. Among these candidates one should be ranked to be the best. Here, it is crucial to have accurate cost-functions despite lack of detailed information of the final implementation.

This paper focuses on the partitioning and candidate-selection procedures and the RT-level power estimation functions which are implemented in a tool for low-power FSM synthesis. The outline for the paper is as follows. In section 2 a background on partitioned FSM design for low-power is given. This is followed by an overview of the tool we have developed. Section 4 goes into the details on partitioning algorithm and power estimation functions. In section 5 synthesis results are given and after that the paper is concluded.

2. Background

The initial design description for most approaches of partitioned 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 which leads to low average power consumption. Each sub-FSM will require circuitry for idle condition detection and for the shut-down mechanism, which both constitutes a functional overhead. The partitioner seeks the most beneficial idle conditions, taking this overhead into account. In the early work by Benini et al. [2], so called self-loops with high transition probabilities were implemented as separate sub-FSMs. This work was generalized to involve clusters of many states [1]. The major power-overhead introduced in a partitioned FSM comes from the fact that at the event of a crossing transition (a state transition has source state and destination state residing in different sub-FSMs). Two sub-FSMs have to be clocked in that cycle to complete the transition [3] which makes it very costly. To overcome this double-clocking requirement, an asynchronous mechanism has been proposed [4]. By allowing asynchronous state changes, 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 is reduced [4].

The straight-forward way to implement a partitioned FSM is to have separate state memory for each of the sub-FSMs, see Figure 1a. On the other hand, the state memories can be shared by all the sub-FSMs since only one is active at a time. One main advantage here is reduced area for flip-flops. In this case there is, however, a need for a global state determining which one of the sub-FSMs is active, see Figure 1b. The global state memory needs to be clocked by the global clock and will add substantial power consumption.

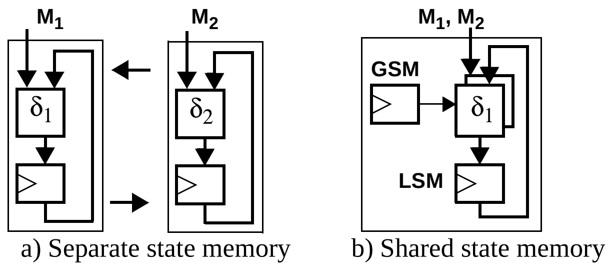

Figure 1. Structural decomposition of FSM

The CAD-tool discussed in this paper targets the mixed synchronous/asynchronous architecture developed in [5] that has a shared synchronous local state memory (LSM) together with a global asynchronous state memory (GSM), see Figure 2. The basic idea is to have synchronous memory in the part always clocked, i.e. the local state memory, and asynchronous memory for the global state memory, that have a low probability of being updated. In this way the global state memory adds very low power-overhead. The shut-down mechanisms used are input-gating to reduce power dissipation in idle combinational logic, and gatedclocks to shut down flip-flops temporarily not needed in the local state memory.

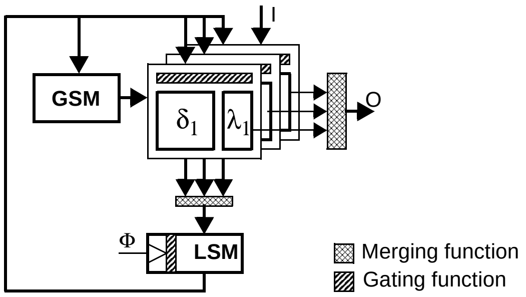

Figure 2. Mixed Synch/Asynch. FSM Architecture

3. Design Flow and Tool

As shown in Figure 3, the tool accepts FSMs described as 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 get the state-transition probabilities, input and output statistics for the state-memory, the transition and output functions. Based on the state-transition probabilities, the states are clustered according to their mutual state-transition probabilities. We then use an algorithm that selects those 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 synthesized to a RT-level description and power consumption is estimated. From these results the best candidate is selected and RT-level VHDL code is generated along with synthesis scripts for logic synthesis in a standard tool.

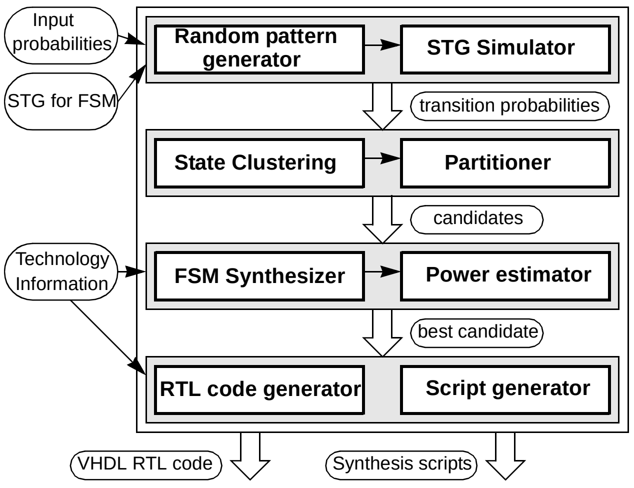

Figure 3. Overview of the tool

4. Automatic Synthesis of Partitioned FSMs

4.1. 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, states are clustered depending on their state transition probability for minimizing the crossing transitions between two sets. Redundant states, that in later stage are discarded, are introduced to *V* firstly, forming *V'*, to make sure |*V'*| (number of vertices) is the power of 2. Assumed |*V'*| equals *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 has then the highest static probability. Take benchmark *dk27* [7] which has 7 states as an example. After introducing one redundant state (8), a full binary tree is built as shown in Figure 4.

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

Figure 4. Full binary tree for dk27

4.2. Candidate Generation

We propose an efficient algorithm that combines clusters in 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 Figure 6, on the binary tree shown in Figure 4, candidates are generated as shown in Figure 5.

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}

Figure 5. Generated candidates for dk27

4.3. Power Estimation

The power estimation functions are used on the partitioning candidates obtained in Section 4.2. 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 which requires different power estimation techniques. From the STG simulator input and output statistics are obtained and used for the power estimation.

```
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 ← $P_{base} \cup P_{restBase}$;
            candidates ← candidates ∪ TMP;
            int restBase ← N - base;
            int place ← N;
            int r ← 0;
            if (restBase > 2) {
                r←restBase;
                while (r > 0) {
                    int i ← $\lfloor \log_2 r \rfloor$;
                    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 ← $P_{base} \cup P_{restBase}$;
                        candidates ← candidates ∪ TMP;
                    }
                }
            }
        }
    }
    return candidates;
```

Figure 6. Algorithm for selecting candidates

Power estimation for combinational logic:

An entropy-based power estimation approach proposed in [6] is 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, are used:

$P_{comb} = \sum_{i=1}^{n} H_i \times Row_i \times k_{tech} \times T_i$. Where H_i is the entropy of the logic, Row_i is the number of rows in the state transition table originating from the sub-FSM i, k_{tech} is an empirically determined constant to adjust to the cell library used, and T_i is the duty period of the sub-FSM i.

Power for global state memory:

For the global state memory we use an empirical model that is based on the structure of the memory. Even though the gate-level implementation is known, we found it more accurate to use the macro model shown below that consists of two parts representing the power of 1) the logic that detects and initiates the transition from one sub-FSM to another and 2) the asynchronous state memory element.

$$P_{GSM} = \quad (k_B \times p_{LSM-B} + k_G \times p_G + k_g \times |g|) \qquad 1)$$

$$+ \sum_{i=1}^{n} P_C \times T_i \qquad 2)$$

The expression inside the parenthesis estimates the power in the global state transition function that is a function of the local state and the global state. The first term represents the contribution from the local state memory where p_{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 sum 2) represents the contribution from the global state memory devices, implemented as muller-C elements where T_i is the probability of global state transition, which is the probability of a crossing-transition between different sub-FSMs. The number of sub-FSMs is denoted n. The constants k_B, k_G, and, k_g are determined empirically. These can be determined based on a single FSM partition run. For more details on the global state memory architecture we refer to [5].

Power for D type flip flop:

The local state memory consists of a set of D flip-flops and is estimated by:

$$P_{LSM} = \sum_{i=1}^{m} P_{Dff_i} \times T_i$$

Where T_i is the duty time of the flip-flop, m is the number of local state memory bits.

Power for clock net energy:

The power dissipation in the clock net is estimated by: $P_{clock} = |FF| \times C_{clkin} \times f \times V_{DD}^2 \times k_{buffer} \times k_{wire}$. Where $|FF|$ is the average number of flip flops clocked, C_{clkin} is the capacitance of the clock input, V_{DD} is the power supply voltage, f is the clock frequency, k_{buffer} is the clock buffer capacitance coefficient, and k_{wire} is the wire capacitance coefficient.

Power for overhead:

$P_{overhead} = P_{gatedCom} + P_{gatedDff}$, where $P_{gatedCom}$ includes the power of AND gates for activating and deactivating the combinational logic, also the OR gates for merging the output; $P_{gatedDff}$ is the power for activating and deactivating the local state bits and basically originates from NAND gates.

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

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

5. Results

The accuracy of the power estimation functions is verified by comparing the estimated power before and after logic synthesis. As reference we use *Power Compiler* (Synopsys) for gate-level power estimation. In Figure 7, the results from the estimation functions P_{whole}, P_{comb}, P_{GSM}, and P_{LSM} (labeled *Estimated*) and the results from Power Compiler (labeled *Actual*) can be compared. A 0.18μm technology is used with V_{DD} of 1.8V, clock frequency of 20MHz. 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, which 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 is the candidate with lowest *estimated* power. Therefore, the absolute difference of these two 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*.

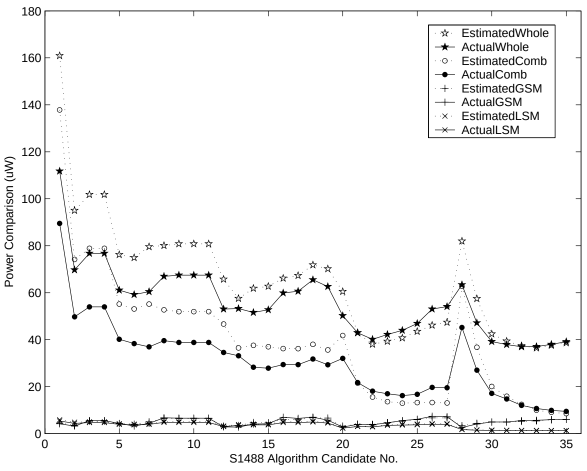

Figure 7. Cost function verification

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 compare them to the one selected by the tool. From Figure 8 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 1. The columns labeled “A.O” and "P.O” represent the area and power of the original monolithic FSM; the column labeled “n” represents the number of sub-FSMs after partitioning; “A.D” and "P.D” represents the area and power of the decomposed FSM; The following two columns represent the percentage of area increase and power reduction of the decomposed FSMs.

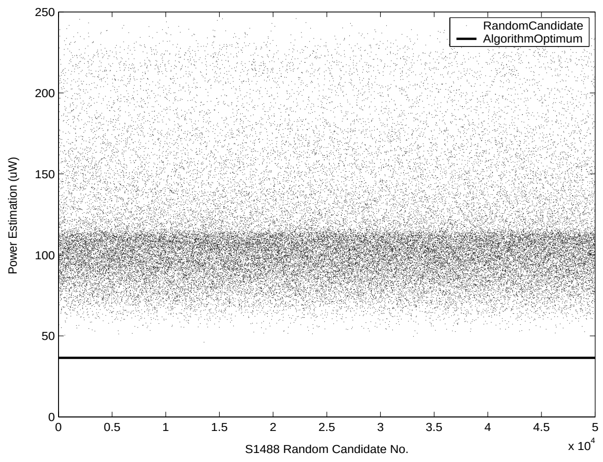

Figure 8. Algorithm verification

The CPU times in Table 1. are for the state clustering and candidate generation algorithms executed on a Pentium4, 1.6GHz processor, under Windows 2000. The total time for the largest benchmark (*scf* 121 states) is 5 minutes 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 early limits the number of candidates.

Table 1. Results for standard benchmarks [7]

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

6. Discussions and Conclusions

In this paper we present a novel multi-way partitioning algorithm for partitioned FSM synthesis. We have applied it to a mixed synchronous/asynchronous architecture but it can also be used for fully synchronous implementations. We also present RT-level power estimation functions that have sufficient accuracy for selecting the candidate with the lowest power consumption. The proposed algorithms are of low complexity which is important when it comes to practical usage of the tool. The tool, as shown in Figure 3, 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 reduces the power consumption significantly, in average 56% for the benchmarks, which is better than any previously reported results for partitioned FSMs.

7. References

[1] L. Benini, G. de Micheli, “Dynamic Power Management: Design Techniques and CAD Tools,” *Kluwer Academic Publishers, Norwell, MA*, 1998.

[2] L. Benini, G. De Micheli, Automatic Synthesis of Low-Power Gated Clock FSMs,” *IEEE Trans. on CAD of Int. Circuits and Systems*, vol. 15, no. 6, pp. 630-643,1996.

[3] B. Oelmann and M. O´Nils, “Locally Asynchronous control of low-power gated-clock finite-state machines,” *IEEE Int. Conference on Electronics, Circuits, and Systems*, pp. 915-918, 1999.

[4] B. Oelmann and M. O’Nils, “A Low-Power Hand-over Mechanism for Gated-Clock FSMs,” *Proc. of the European Conference on Circuit Theory and Design*, Stresa, Italy, 1999.

[5] C. Cao, B. Oelmann, “Mixed Synchronous/Asynchronous State Memory for Low Power FSM Design”, *Proceedings of EUROMICRO Symposium on Digital System Design*, 2004.

[6] M.Nemani, FN. Najm, “Towards a High-Level Power Estimation Capability,” *IEEE Trans. on CAD of Int. Circuits and Systems*, vol.15, no. 6, pp.588 -598,1996.

[7] S. Yang, “Logic Synthesis and Optimization Benchmarks User Guide version 3.0,” *MCNC Technical Report*, 1991.