FINITE-STATE MACHINE PARTITIONING FOR LOW POWER

Luca Benini, Giovanni De Micheli

Stanford University
Computer Systems Laboratory
Stanford, CA 94305

Frederik Vermeulen

IMEC
Kapeldreef 75, B-3001
Leuven, Belgium

ABSTRACT

We describe an algorithm for the automatic synthesis of a network of interacting FSMs starting from a single state-table specification. The sub-machines in the decomposed FSM communicate through a set of additional interface signals. The decomposed implementation has low power dissipation because *one single sub-machine is clocked* at any given time and it controls the outputs values while all other sub-machines are idle. There is full cycle-by-cycle equivalence between the input-output behavior of the decomposed and undecomposed implementation.

1. INTRODUCTION

The ever increasing integration density of CMOS technology has lessened the concern for area usage for VLSI circuits, giving more importance to timing and power dissipation constraints. Controllers are often critical for power, because they are continuously running (while parts of the data path may be shut down), and for timing because the delay through the controller may constrain the delay through the data path.

In this work we propose a procedure for the automatic synthesis of a network of interacting FSMs starting from a single state-table specification. The straightforward single-machine implementation is called *undecomposed FSM*. We call *decomposed FSM* the interacting FSM implementation. The sub-machines in the decomposed FSM communicate through a set of additional interface signals. The decomposed implementation has low power dissipation because *one single sub-machine is clocked* at any given time and it controls the outputs values while all other sub-machines are idle: they do not receive the clock signal and dissipate little power. When a sub-machine terminates its execution, it sends an *activation signal* to another sub-machine which takes control of the computation, then it de-activates itself. This transition is characterized by a single cycle for which both sub-machines are clocked. There is full cycle-by-cycle equivalence between the input-output behavior of the decomposed and undecomposed implementation.

This research is partially supported by NSF under contract MIP-9421129 and by ARPA under contract DABT-63-95-C-0049.

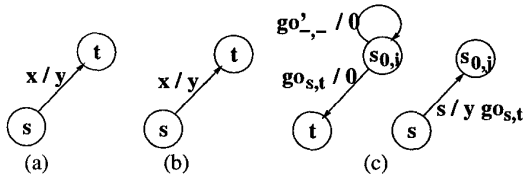

Figure 1: Definitions of δ_i and λ_i

Decomposition for low power has been studied in [2]. In this work the authors exploit the principle of mutual exclusion between states and formulate decomposition as a state assignment problem. Our decomposition strategy operates at a higher level of abstraction. No constraints are enforced on the state codes of the FSMs since our algorithm is applied *before* state assignment. Moreover, our gated clock architecture is novel and allows us to completely shut down not only the combinational logic, but also the flip-flops. The practical effect of these differences that our technique may have higher area overhead but it can achieve larger power savings.

2. LOW-POWER INTERACTING FINITE-STATE MACHINES

Given the state stable or the state transition graph (STG) of a monolithic FSM we first compute a partition $\Pi(S)$ of its state set S. The elements of $P(S)$ are sets of states, called *blocks*. The blocks are mutually disjoint and cover S. The algorithm for the computation of $\Pi(S)$ is described in the next section. Here we focus on how to decompose in a power-efficient fashion a monolithic FSM given a partition $\Pi(S)$.

Refer to Figure 1. Part (a) shows a transition in the undecomposed FSM. Part (b) shows the transition in the decomposed FSM when its source and destination state both belong to the same partition block P_i (the transition is unchanged). Part (c) shows the case when the source and des-

0-7803-4455-3/98/$10.00 © 1998 IEEE

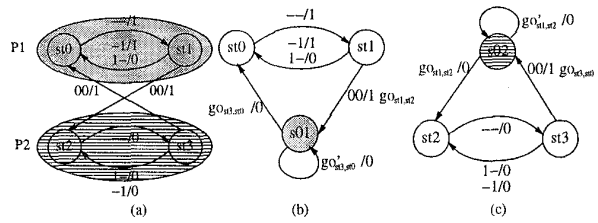

Figure 2: Decomposition of the monolithic FSM

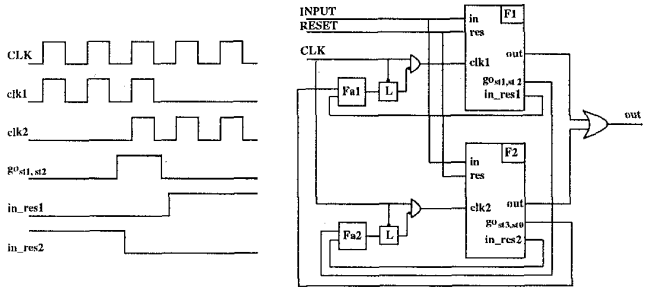

Figure 3: Gated-clock implementation of the interacting FSMs

tination state belong to different partition blocks. For each transition leaving a state sub-set P_i in the undecomposed FSM, the sub-FSM F_i associated with P_i performs a transition to its reset state. On the other hand, a transition entering a sub-set P_i from $P_i \neq P_j$ corresponds to a transition exiting the reset state in the sub-FSM F_i. A sub-machine can exit the reset state only upon assertion of a *go* signal by another submachine. At any given clock cycle only two situations are possible: i) one sub-machine is performing state transitions and all other sub-machines are in reset state, ii) one sub-machine is transitioning toward its reset state, while another one is leaving it.

All inputs and outputs of the undecomposed FSM are inputs and outputs of the sub-machines. The *go* signals are new, additional inputs and outputs. If an edge $s \rightarrow t$ of the original machine has head and tail state included in sub-machine F_i, the edge is replicated in F_i, with the same input and output fields. Edges in the global FSM connecting states which belong to different partitions are associated with edges representing transitions to and from the reset states of the corresponding sub-FSMs. These transitions are labeled as follows: i) edges toward reset have the same input field as the original edge, assert an additional output $go = 1$ and have the same output field as in the original transition edge of the undecomposed FSM. ii) Transitions leaving reset have only one specified input *go* and all outputs set to zero. The outputs of a sub-machine blocked in reset state are zero.

Example 1 *Consider the FSM in Figure 2 (a). We assume that the state partition is* $\Pi(S) = \{P_1, P_2\} = \{\{st0, st1\}, \{st2, st3\}\}$. *The two sub-machines created by the decomposition procedure are shown in Figure 2 (b) and (c). The additional reset states are shaded.* P_1 *originates sub machine (b) and* P_2 *originates sub machine (c). Notice that the "go" signals are shown only on the transitions from and to the reset states. A sub-machine asserts a "go" signal only when transitioning to the reset state, in all other cases the signal has value zero. Similarly, a submachine is sensitive to input "go" signals only when it is in reset state. The "go" inputs are not observed for all other transitions.*

In the interacting FSM system, most of the machines F_i remain in state $s_{0,i}$ during a significant number of cycles. If we stop their clock while they stay in reset state, we save power because only a part of the system is active and has significant switching activity. To be able to stop the clock, we need to observe the following conditions.

- The condition under which F_i is idle. It is true when F_i reaches the state $s_{0,i}$. We use the Boolean function $is\_in\_reset_i$ that is 1 if F_i is in state $s_{0,i}$, 0 otherwise.
- The condition under which we need to resume clocking, even if the sub-FSM is in reset state. This happens when the sub-FSM machine receives a *go* signal and must perform a transition from s_{0i} to any other state.

We can derive F_{ai}, called *activation function* (in negative logic). The clock to F_i is halted when $F_{ai} = 1$. Namely:

$$F_{ai} = is\_in\_reset_i \wedge \overline{\left(\bigvee_{q \in F_i,\, p \in F_j \neq F_i} go_{p,q} \right)} \qquad (1)$$

The first term $is\_in\_reset_i(s)$ stops the clock when the machine reaches $s_{0,i}$. The second term ensures that clock is not halted when one of the $go_{p,q}$ is asserted and the sub-FSM must exit the reset state. This activation function allows the newly activated sub-FSM to have its first active cycle during the last cycle of the previously active FSM. The two sub-FSMs make a transition in the same clock cycle: one is transitioning to its idle state, and the other from its idle state. The local clocks of F_i and F_j are both active. We call *transitions of control* the cycles when a sub-FSMs shuts down and another activates.

Example 2 *The gated-clock implementation of the interacting FSMs of Figure 2 is shown in Figure 3. Notice how the external output is obtained by OR-ing the outputs of the sub-FSMs. Figure 3 also shows the clock waveforms, the "in_reset" signals and the "go" signals. Notice that there is a clock cycle for which both local clocks are enabled. The waveforms show how sub-FSM 1 is deactivated and*

sub-FSM 2 activates thanks to the assertion of the $go_{st1,st2}$ signal.

3. PARTITIONING ALGORITHM

The ideal mode of operation for the interactive FSM circuit is one of minimum transition of control between different sub-FSMs. When a sub-FSMs disables itself and another one takes control, both machines are clocked for one cycle, the *go* signals involved in the control transfer change value, and the clock control circuitry switches as well. As a result, transitions of control are power consuming and should be avoided as much as possible.

Minimizing the number of *go* signals is another important objective. The generation of such signals requires additional hardware, that increases power dissipation. Moreover, the *go* signals increase the coupling between sub-machines, complicating the placement and routing of the circuit. On the other hand, if we reduce the number of *go* signals to zero, i.e. we do not decompose the FSM, no power savings are achieved.

In summary, we should look for a partition $\Pi(S)$ which maximizes the locality of the computation and minimizes the hardware overhead for communications between sub-FSMs.

We implemented a heuristic partitioning procedure based on a *genetic algorithm* [3] (GA) Some properties of the problem made it well suited for evolutionary optimization. First, the solution space is easily representable as a set of bit-strings: a chromosome is encoded as a set of $|S|$ blocks of $\lceil log_2 n_{max} \rceil$ bits. Each block is associated to a state and represent the number identifying the partition block to which the state belongs. The length of the chromosome in bytes is $|S| \cdot \lceil log_2 n_{max} \rceil / 8$. This is a very compact encoding and if n_{max} is chosen as a power of two, every chromosome represents a valid solution.

Third, and more importantly, the cost function can be efficiently evaluated. We define the *cost of a partition* Π as:

$$C(\Pi) = \sum_{b=1}^{N_{block}} K_b \times Prob(b) \qquad (2)$$

where N_{block} is the number of partition blocks, K_b is an approximate measure of the hardware cost of the implementation of the sub-machine corresponding to block b (which is proportional to the number of states in b and on the number of *go* signals in its interface) and $Prob(b)$ is the probability of the sub-machine to be active. We do not describe the compotation of $C(\Pi)$ in detail because of space limitation. However, it is important to notice that the computation is $O(|E|)$ where $|E|$ is the number of edges in the STG of the original machine.

The compact encoding (with no invalid solutions) and the inexpensive computation of the cost function allow us to take very large populations (in the order of 10^6 individuals) in our GA solver.

4. EXPERIMENTAL RESULTS

Our decompostion tool consists of two modules: the *partitioner* and the *netlister*. The partitioner reads in the STG of the undecomposed FSM and finds an optimal partition $\Pi(S)$. The frame for the genetic algorithm implemented in the partitioner is provided by the Genesis package [4]. The netlister reads in the partition $\Pi(S)$, the STG of the specification and produces the decomposed FSM.

Table 1 shows the results on a number of benchmarks, the first three examples are controllers of data-path small full-custom chips implemented in a class project. The remaining FSMs are standard MCNC benchmarks [5], with the exception of the last one which is a modified version of MCNC benchmark `s298` (we reduced the number of states because the commercial tool we used for FSM synthesis could not optimize the undecomposed implementation with the memory resources available on our machines).

The differences in area, power and speed between the partitioned machine and the original unpartitioned design is given in Table 1 between parentheses in the last three columns. The average power reduction is 31%. There is also an increase in speed of 12%. The number of standard cells increases on average by 48%. The time spent in decomposition is upper bounded by 9 hours. However we observed that the GA converges quite rapidly (i.e. 90% quality is reached in 1 to 2 hours for all benchmarks).

Notice also that the increase in area is marked on all the examples. The main reason for this phenomenon is the overhead due to additional flip-flops. We specified minimum-length state encoding in all our experiments. This encoding style implies that the number of flip-flops in the undecomposed machine increases only logarithmically with the number of states. When the machine is decomposed, the number of states in each sub-machine is decreased by a factor of two if the partition is balanced. In this case each sub-machine has just one flip-flop less than the original machine and the total number of flip-flops is almost doubled. If the partition is unbalanced, the number of flip-flop is generally increased by the flip-flops required in the smaller machine. Obviously the sequential overhead is larger for N-way partitions, with $N > 2$. We could have performed our tests specifying *one-hot* encoding. In this case, the sequential overhead would have been null. However, we feel that the comparison with minimum-length encoding is fairer towards the undecomposed implementation.

Name	# of states	# of partitions	Original # of std cells	power	crit. path	Partitioned # of std cells	power	crit. path
test1	24	4	67	804	3.81	118 (+76%)	679 (–16%)	3.01 (–21%)
test2	18	3	58	930	2.83	89 (+53%)	642 (–31%)	2.98 (–5%)
test6	80	4	252	2115	7.09	336 (+33%)	1209 (–43%)	5.92 (–17%)
bbsse	13	4	112	1146	4.21	145 (+29%)	847 (–26%)	3.60 (–14%)
dk512	14	2	61	1138	3.29	88 (+44%)	853 (–25%)	2.79 (–15%)
keyb	18	3	157	1688	4.15	262 (+67%)	1387 (–18%)	4.69 (+13%)
planet	48	4	360	4967	6.76	503 (+40%)	3241 (–35%)	5.85 (–13%)
s1488	48	4	433	2743	6.68	642 (+48%)	1717 (–37%)	5.82 (–13%)
s820	25	3	191	1717	5.01	238 (+25%)	1171 (–32%)	3.83 (–24%)
s832	25	3	211	1889	4.61	274 (+30%)	1244 (–34%)	3.72 (–19%)
sand	32	4	429	3395	7.86	471 (+10%)	2554 (–25%)	6.78 (–14%)
scf	112	8	672	3719	7.07	988 (+47%)	2280 (–39%)	5.36 (–24%)
test13	166	8	681	7006	7.38	1610 (+137%)	4124 (–41%)	7.57 (+3%)

Table 1: Power, area and speed of the decomposed implementation versus the undecomposed one

5. REFERENCES

[1] M. Kuo, L. Liu and C Cheng, "Finite-State Machine decomposition for I/O minimization," *IEEE International Symposium on Circuits and Systems*, pp. 1061–1064, April 1995.

[2] S-H. Chow, Y-Chen Ho et al., "Low-power realization of Finite-State Machines - A decomposition approach," *ACM TODAES* Vol. 1, No. 3, pp. 315–340, July 1996.

[3] D. Goldberg, *Genetic Algorithms in search, optimization and machine learning*, Addison-Wesley, 1989.

[4] J. J. Grefenstette, *A user's guide to GENESIS*, 1990.

[5] S. Yang, "Logic synthesis and optimization benchmarks user guide. Version 3.0," *MCNC Technical Report*, 1991.