Mixed Synchronous/Asynchronous State Memory for Low Power FSM Design

Cao Cao and Bengt Oelmann

Department of Information Technology and Media, Mid-Sweden University
S-851 70 Sundsvall, Sweden
{cao.cao@mh.se}

Abstract

Finite state machine (FSM) partitioning proves effective for power optimization. In this paper we propose a design model based on mixed synchronous/asynchronous state memory that results in implementations with low power dissipation and low area overhead for partitioned FSMs. The state memory here is composed of the synchronous local state memory and asynchronous global state memory, where the former is used to distinguish the states inside a sub-FSM, and the latter is responsible for controlling sub-FSM communication. The input and output behaviour of the decomposed FSM is cycle by cycle equivalent to the undecomposed synchronous FSM. Together with clock gating technique, substantial power reduction can be demonstrated.

1. Introduction

The majority of low power optimization techniques on architectural level focus on shutting down parts of the circuits that are idle, techniques that go under the name dynamic power management [2]. For the contemporary CMOS technology where the dynamic power dissipation dominates over the static in digital circuits [1], minimizing the switching capacitance is the objective of power minimization. Here, shutting down means preventing idle circuits and nets from switching. Normally, systems are designed to meet a certain peak performance that is only required for a small portion of its entire operational time; therefore, parts of the circuit are often temporarily idle. There are also situations where operations, known in advance, will never be executed at the same time, which always lead to having idle units consequently. In these situations, dynamic power management may be successfully used.

Dynamic power management techniques disable the clock signal or prevent inputs from switching to the parts not in use. In order to do so, mechanism for detecting idle states of different units is needed, also methods for "shutting down" the idle units must be added to the design. Circuits responsible for handling this will constitute a functional overhead and will consequently contribute to increased circuit area, additional power consumption, and possibly reduced speed performance. Careful analysis must be undertaken so that the introduction of circuits for power management will lead to as large power reduction as possible. An optimization procedure for dynamic power management seeks the partitioned system that has the lowest power consumption. The procedure partitions the design after identifying the most beneficial idle conditions taking the overhead of detecting and shutting down circuits into account.

For low power FSM design, the most efficient way is to divide the FSM into two or more sub-FSMs where only one of them is active at a time [3]. The partitioned FSM is constructed in such a way that each of the sub-FSMs will constitute a smaller effective capacitance than the original FSM and consequently power can be saved. Gating the clock signal to shut down the FSM not active is an efficient way and it has been practised in several works, e.g. in [4, 5]. There are two drawbacks in these approaches. First, in minimum length state encoding the area overhead from the increased number of bits in the state memory is substantial for a partitioned FSM. Second, the power consumption for activating and deactivating a sub-FSM is relatively high. These problems have been addressed separately before in e.g. [6] and [7]. In contrast to previous work, we propose a design model that is able to handle both issues in an efficient way.

In the design model for partitioned FSMs we are proposing in this paper, both synchronous and asynchronous state memories are used to implement FSMs with synchronous input/output behaviour. This means that externally the FSM will work as a synchronous FSM but internally there is a mechanism operating asynchronously. This model is the result of our search for finding ways to utilize asynchronous logic in synchronous designs. The general idea is to only use synchronous state memory for state bits that have high probability of changing and asynchronous state memory for those bits with low probability of chang-

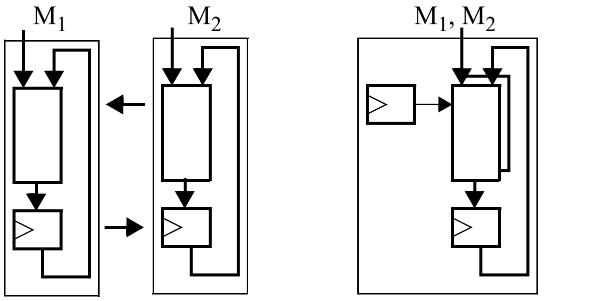
Proceedings of the EUROMICRO Systems on Digital System Design (DSD'04)
0-7695-2203-3/04 $ 20.00 IEEE

ing.

The outline of rest of the paper is as follows. First a presentation is given on approaches to low power FSM design based on FSM partitioning and how the proposed design model is related to them. After that the proposed model is described, first through an example and then by a formal description. This is followed by a description of how to transform a finite state machine specification, in the form of a state transition graph, to the form suitable for implementing it as a partitioned FSM with mixed synchronous/asynchronous state memory. An implementation architecture is then proposed and the effectiveness is illustrated by optimizations through two-way partitioning of a subset of the MCNC FSM benchmarks [8].

2. Background

From the point of view of structural decomposition, there are basically two approaches to partition FSMs. The first one is based on separate state memory for each sub-FSM and the second one has shared state memory for all sub-FSMs. The two alternative structures are shown in the figure below. In this section we first introduce the key issues in the implementation of partitioned FSMs, and from that motivate our approach based on mixed synchronous/asynchronous technique.



a) Separate state memory

b) Shared state memory

Figure 1. Structural decomposition of FSMs

2.1. FSM decomposition with separate state memory

As depicted in Figure 1a) above, each sub-FSM has its own state memory. These state registers are local to the sub-FSM and are referred to as *local state memory.* A state transition with a destination state not residing in the same sub-FSM as the source state we refer to as a *crossing transition*. No global state is needed and the interaction between different sub-FSMs is handled by adding reset states, one in each sub-FSM, to the local states and an additional signal interface for activating and deactivating different sub-FSMs. Assume a *crossing transition* from sub-FSM M_1 to sub-FSM M_2, when exiting M_1 it turns to its reset state and causes the activation of M_2 that goes from its reset state to the correct destination state of the *crossing transition*. M_1 will reside in its reset state and shut itself down through gating the clock and input signals.

Power reductions can be achieved through clock gating and disabling the primary inputs to the sub-FSMs not active.

Suppose the original, monolithic machine is partitioned into n sub-FSMs with the state subsets S_1, S_2, ..., S_n respectively, the total number of bits for the local state will be:

$$\sum_{i=1}^{n}\left\lceil \log_2|S_i| \right\rceil$$

in the case minimum encoding is used. It will always be more bits than what is required in the monolithic implementation. The disadvantage here is the area overhead. The additional flip-flops often constitute a large portion of a state machine. This approach has for example been used in fully synchronous partitioned FSM by Benini et al. [2, 3]. In the events of *crossing transitions* between sub-FSMs there are actually two state transitions taking place (from the source state to the reset state in M_1 and from the reset state to the destination state in M_2). This makes *crossing transitions* more power consuming than local transitions. The work by Oelmann et al. [10] introduces a mechanism that makes the *crossing transition* asynchronously and thereby removes the double-clocking requirement, which leads to lower power consumption. This approach leads however to large area overhead mainly due to complex asynchronous logic and large overhead in the output logic.

2.2. FSM decomposition with shared state memory

To overcome the problem of the large area overhead, the *local state memory* is shared by all the sub-FSMs [7] as depicted in Figure 1b). Considering the previously described approach, it can be realized that only the state memory in the active sub-FSM is of importance when computing the next state and the outputs, the rest of the state memory is in that sense of no importance. By dividing the states into two parts, global states and local states, the bits for the local states can be shared by all sub-FSMs. The global states decide which one of the sub-FSMs is active. In this way identical state codes can be used for states residing in different sub-FSMs and being distinguished by the global state.

A monolithic FSM is partitioned into n partitions with state subsets S_1, S_2, ..., S_n respectively. The global state needs $\lceil \log_2 n \rceil$ bits to distinguish between n sub-FSMs and the local state needs $\lceil max(\log_2|S_1|, ..., \log_2|S_n|) \rceil$ bits to represent the sub-FSM with the largest number of states. The total number of bits in the state memory will be lower compared to the separate state memory approach. However, from the power consumption point of view, the disadvantage is that the extra flip-flops for the *global state memory* and the identical number of flip-flops required for each current active sub-FSM. The increased capacitive load on the clock signal will be the major reason for increased power dissipation.

In the design model for partitioned FSMs introduced in this paper, a shared state memory approach is used where the *global state memory* is asynchronous. The basic idea of having asynchronous *global state memory* comes from the fact that the *crossing transitions*, which lead to changes in the global state, are of low probability and are therefore idle most of the time. By not having the global state continuously clocked, power reduction is achieved. The *local state memory* is kept synchronous and is conditionally clocked based on the number of bits required for the sub-FSM currently active.

3. FSM decomposition model

The main objective of this work is to propose a new FSM decomposition model based on mixed synchronous/asynchronous state memory to achieve low power consumption and low circuit overhead. At the same time, the input/output behaviour of the decomposed FSM is identical to the original fully synchronous one.

3.1. Design model overview

In our model, the partitioned sub-FSMs share the same synchronous *local state memory* while asynchronous *global state memory* controls which one of the sub-FSMs should be active. In order to handle *crossing transitions*, the STG is transformed to support an interaction scheme for asynchronously activating and deactivating the sub-FSMs.

After decomposition, the original state set is partitioned into several subsets. State transitions having the source and destination states belonging to the same state subset will be copied without transformation. For every *crossing transition*, an extra *g state* is introduced.

A *crossing transition* is completed by the following sequence of events:

1. A synchronous state transition from the source state of the *crossing transition* to the *g state*, which has the same index as the original destination state.
2. An asynchronous state transition from the *g state* to the original destination state, both of which have the same index.

The first event is called synchronous because the *local state memory* is updated to the *g state* at the active edge of the clock signal. The second event is called asynchronous because it takes place in the *global state memory* upon detection of transitions in the *g states*. The global state is then used to deactivate the currently active sub-FSM, activate the sub-FSM in which the destination state of the crossing transition is. Thanks to the asynchronous global state transition the entire *crossing transition* is completed within one clock cycle.

Consider the STG in Figure 2 and assume a partition of 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.

.

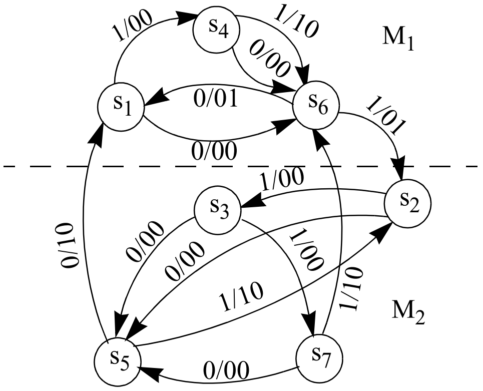


Figure 2. FSM example dk27

Figure 3 shows the transformed STG after decomposition (Input/output is ignored here for clarity). After introducing g states, two new state subsets are formed as U_1= {s_1,s_4,s_6,g_2} in M_1, U_2={s_2,s_3,s_5,s_7,g_1,g_6} in M_2.

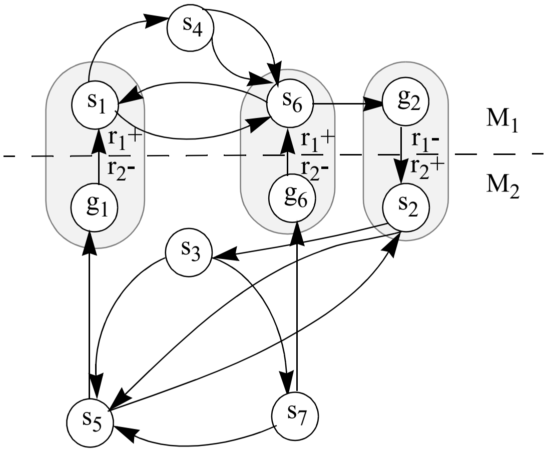


Figure 3. Transformed STG in decomposed FSM

Take the *crossing transition* $s_6 \to s_2$ as an example. After g_2 is introduced in M_1, the first event is the transition $s_6 \to g_2$, inside M_1. Then at the second event, the detection of g_2 makes the asynchronous state memory update its state from r_1 to r_2 (labelled as r_1-,r_2+ on edge $g_2 \to s_2$). The global states r_1, r_2 indicates the active sub-FSMs M_1, M_2 respectively. After the completion of the asynchronous transition, M_1 is deactivated and M_2 is activated.

The asynchronous transition $g_2 \to s_2$ will not influence the *local state memory* which only can be triggered by clock signal; therefore, the source state g_2 and the destination state s_2 will have the same 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* in this paper. Specially, the *state bundle* including *g state* is called a *g state bundle*. In Figure 3, there are three *g state bundles* (g_1,s_1),(g_2,s_2),(g_6,s_6) indicated with circles shaded gray.

3.2. Definitions

To study state transitions separately, a state machine is defined as a triplet: $M = (S, I, \delta)$, where S is the set of states, I is the set of binary inputs, $\delta: S \times I \rightarrow S$ is the transition function.

Let there be a partition on the set S: $\pi = \{S_1, \ldots, S_n\}$ where π is defined as a collection of n subsets, called *block*s also, such that

$$\bigcup_{i=1}^{n} S_i = S$$

and $S_i \cap S_j = \varnothing$ for $i \neq j$ where $1 \leq i, j \leq n$.

The monolithic FSM associated with S is then partitioned into sub-FSMs $M_1, M_2, \ldots, M_n$.

In state transitions, to reflect the property of states entering or exiting a certain partition *block* S_i, let us define

$$V(S_i) = \{s_j | \delta(s_j, I) = s_k, s_j \notin S_i, s_k \in S_i\}$$

$$T(S_i) = \{s_j | \delta(s_k, I) = s_j, s_j \notin S_i, s_k \in S_i\}$$

Both $V(S_i)$ and $T(S_i)$ are set of states outside block S_i, the former has state transitions to S_i; the latter has state transitions originating from S_i.

Inside S_i, let us define:

$$Q(S_i) = \{s_j | \delta(s_k, I) = s_j, s_j \in S_i, s_k \notin S_i\}$$

$$W(S_i) = \{s_j | \delta(s_j, I) = s_k, s_j \in S_i, s_k \notin S_i\}$$

Both $Q(S_i)$ and $W(S_i)$ are state subsets inside *block* S_i, the former has state transitions originating from another partition *block*; the latter has state transitions to another partition *block*.

These four state sets defined above are depicted in Figure 4.

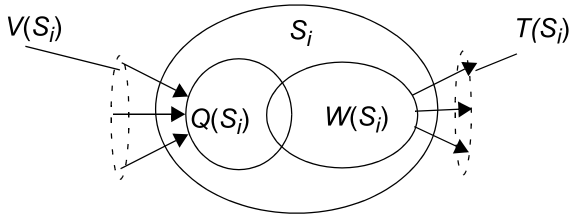


Figure 4. State sets associated with S_i

They will be denoted as V_i, T_i, Q_i and W_i in short in the rest of this paper.

3.3. Network transformation

According to the definition in section 3.2, the STG transformation is made in the following steps:

3.3.1 Introduce *g states*

For a certain *block* S_i, G_i is a collection of *g states,* which are introduced based on the destination states of *crossing transitions* exiting S_i.

$$G_i = \{g_k | s_k \in T_i\}$$

The state subset associated with sub-FSM M_i is then modified from S_i to U_i, where

$$U_i = S_i \cup G_i$$

In the transformed network, let us define

$\bigcup_{i=1}^{n} G_i = G$ as the collection of all *g states* and

$\bigcup_{i=1}^{n} U_i = U$ as the modified collection of all states. The elements in U can be generally designated as u_k, where k is a subscript variable.

3.3.2 Transition function transformation

The original transition function δ is transformed into δ_L and δ_G, representing the state transition inside the *local state memory* and *global state memory*, separately.

1. Form the local transition function δ_L.

Let us define $\delta_L : S \times I \rightarrow U$ as

$$\delta_L(s_i, I) = \begin{cases} \delta(s_i, I) & \text{if} \quad \delta(s_i, I) = s_k \notin T \\ g_k & \text{if} \quad \delta(s_i, I) = s_k \in T \end{cases}$$

Transitions from a certain set W_i to T_i are replaced with transitions from W_i to the additional introduced set $G_{i.}$

2. Form the global transition function δ_G.

The global state set is defined as $R = \{r_1, r_2, \ldots r_n\}$

There are as many states in R as the number of sub-FSMs in the partitioned FSM. The global state identical to r_i indicates sub-FSM M_i as the active sub-FSM.

Let us define $\delta_G : R \times U \rightarrow R$ as

$$\delta_G(r_i, u_k) = \begin{cases} r_i\text{-}, r_m+ & \text{if} \quad u_k \in G_i \\ r_i & \text{otherwise} \end{cases}$$

Where r_i-,r_m+ representing the asynchronous state transition. Since $u_k \in G_i$, we assume it represents the *g state* g_k. A *crossing transition* is thus implied and its destination state is s_k. Thereby, r_i-,r_m+ indicates sub-FSM M_1 is deactivated and M_m satisfying $s_k \in S_m$ is activated.

3.4. State bundling

In section 3.1, we 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 handover mechanism. 3) The *g state bundle* enables an efficient clock gating implementation.

After the network transformation, a *bundled state table* is built. Every column of the table represents a s*tate bundle.* A *state bundle* is a set of states with same local state code but different global state code. Every row of the table represents the states in a sub-FSM which have the same global state code. The number of rows is the same as the number of sub-FSMs.

It is known that the *g state* in *G* and its corresponding state in *S* with the same index must be put into the same g *state bundle*, so we build the table beginning with *g state bundles*.

To be specific, let us examine the example in Figure 3 again. Its *bundled state table* is 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 state* is also included). Firstly, three *g state bundles* are put into the table cells shaded gray.

Table 1. Bundled state table

B	b_1	b_2	b_3	b_4	b_5	b_6
M_1	s_1	s_6	g_2	s_4		
M_2	g_1	g_6	s_2	s_3	s_5	s_7

Other states in each sub-FSM are then put into the table ordinally from the leftmost empty cell. We finally get six bundles and every sub-FSM has the same number of bundles as the number of states inside it. After building the *bundled state table*, the state transition inside a sub-FSM can be viewed upon as the *state bundle* transition.

Let us observe the *crossing transition* from s_6 to s_2 again. From Table 1, this transition can be explained in the following sequence: 1) local state transition from *state bundle* b_2 to b_3 inside M_1. 2) global state transition from M_1 to M_2, when *local state memory* still resides in b_3.

3.5. State encoding

In the *global state memory*, one hot encoding is used for state encoding. Every global state r_i is encoded with only one bit to be one and all other bits to be zero. The rest of this section explains how to encode the states in the *local state memory* and the influence of the state assignment to the final gated clock implementation.

State encoding in the *local state memory* has the same meaning as *state bundle* encoding. The requirement on the state bundle encoding is that minimum number of bits in the state code are changeable for a certain sub-FSM. This will enable efficient clock gating and minimize the size of the combinational logic and often the switching activity of this logic. Binary encoding, which satisfies the requirement, will be used in the rest of the paper. It gives the binary code of zero to the leftmost column of the *bundled state table*. Codes are then increased by one for the columns from left to right.

As mentioned in section 3.4, the number of local state bits is decided by the sub-FSM with the largest number of *state bundles*, that is, $\lceil max(\log_2|U_1|, \ldots, \log_2|U_n|)\rceil$.

Due to the property of binary encoding, for state transitions inside a sub-FSM M_i, only $\lceil \log_2|U_i| \rceil$ bits can be changed. These bits are called the *changeable bit field* of M_i. Other bits which are always zero can be called *don't care bits* of M_i. Thereby, when M_i is active, only the *changeable bit field* needs to be triggered by the clock signal and taken as inputs to the combinational logic of M_i. One thing that needs to be pointed out is each *changeable bit field* related with a certain sub-FSM is decided by the global state; therefore, it only changes after the global state asynchronous transition, that is, the next clock cycle after the *crossing transition*. The problem left is how we can get the correct code in *local state memory* when there is a *crossing transition* between two sub-FSMs with different *changeable bit fields*. This problem is solved by the introduction of *g state bundles* which give extra restrictions to the state encoding. The *g state* which is in the same sub-FSM as the source state of the *crossing transition,* working as a transition state, makes the source and destination state of a *crossing transition* have their local state codes within the same *changeable bit field* of the current active sub-FSM. Accordingly, the current sub-FSM*'s don't care bits* which keep zero after the completion of the *crossing transition* will not influence the correct code of the *crossing transition* destination state.

To be specific, we examine the example in Figure 3 again and binary encoding is assigned in the *bundled state table*.

From Table 2, we can see the number of local state bits is three. In M_1, only bit0 and bit1 are changeable and belong to the *changeable bit field*. The bit2 which is always zero is regarded as *don't care bit* of M_1. In M_2, all three state bits are in its *changeable bit field*.

Table 2. State encoding for bundled state table

B	b_1 000	b_2 001	b_3 010	b_4 011	b_5 100	b_6 101	$\lceil \log_2\|U_i\| \rceil$
M_1	s_1	s_6	g_2	s_4			2
M_2	g_1	g_6	s_2	s_3	s_5	s_7	3

Suppose there is a *crossing transition* from s_5 in M_2 to s_1 in M_1. After the synchronous transition from b_5 to b_1, the *local state memory* is changed to “000”. Bit2 becomes zero and will be disabled in the next clock cycle after the asynchronous transition from M_2 to M_1. If there is a *crossing transition* from s_6 in M_1 to s_2 in M_2 reversely, after the synchronous transition from b_2 to b_3, the local state memory will be changed to "010". The *g state bundle* b_3 makes the highest bit of s_2 zero only, which is restricted by g_2. Without this encoding restriction, a *crossing transition* from M_1 to M_2 may require the local code to change from "001" to "110", for example, then the disabled bit2 is still zero and the result will be "010" instead. In other words, *g state bundles* ensure a correct state code in the *local state memory* after the completion of the *crossing transition*.

4. Implementation structure

In this section we first propose a general structure for our decomposed FSM model. Then we give a detailed description of the implementation. For clarity we limit ourselves to describing the two-way partitioned FSM.

4.1. N-way partitioning structure

Suppose the monolithic machine has *I* as input, *O* as output and is partitioned into sub-FSMs $M_1, M_2, ..., M_n$. The original state subsets $S_1, S_2, ..., S_n$, combining the introduced *g states*, form the new state subsets $U_1, U_2, ..., U_n$ for $M_1, M_2, ..., M_n$, respectively. All sub-FSMs share the same *local state memory* but have their own combinational logic. Our decomposed FSM structural model is shown in Figure 5.

The *G state bundle* Detection Logic (referred to as GDL) decodes the state bits in the *Local State Memory* (referred to as LSM). If a *g state bundle* is detected, a signal is sent to the *Global State Memory* (referred to as 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 and primary output function) of the sub-FSMs (labeled $M_1 ..., M_n$ in Figure 5).

As pointed out in section 3.5, the number of local state bits to the combinational part of M_i is $\lceil \log_2 |U_i| \rceil$. For an active M_i, only the *changeable bit field* of the LSM is clocked when the other bits 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 less number of state bits needed for sub-FSMs. Reduction in both area and power can thus be achieved. Large power reductions is obtained when a good partitioning is found where a small sub-FSM active most of the time.

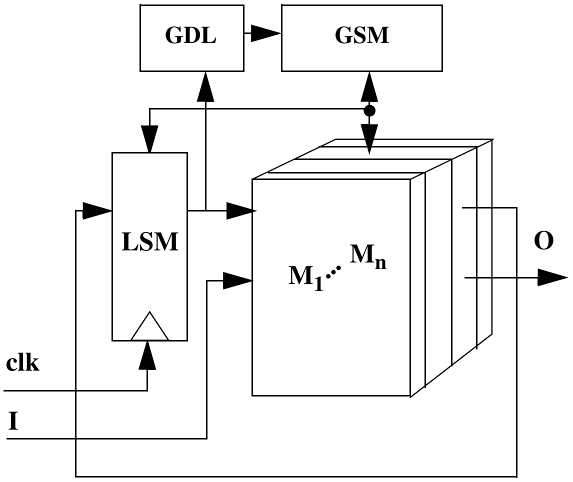


Figure 5. Structural model based on mixed synchronous/asynchronous state memory

4.2. Two-way partitioning implementation

For the sake of clarity, we limit ourselves to present the detailed implementation architecture for two-way partitioning, but it can easily be extended to FSMs with more partitions. In addition, according to our experiments, two-way partitioning can result in large power savings.

To be specific, we examine the example in Figure 2 again. The original STG is transformed in Figure 3 and *bundled state table* is set up in Table 1. Local state codes are given in Table 2. The global state set is defined as R={r_1,r_2} and the state codes of r_1 or r_2 are indicated as (n_1,n_0), where (n_1,n_0)=01 represents that sub-FSM M_1 is active, (n_1,n_0)=10 represents that sub-FSM M_2 is active. By one-hot encoding of the global state, it is possible to decode the active sub-FSM directly from the state bits.

Figure 6 shows the block diagram for the overall realization. The *G state bundle* Detection Logic (GDL) detects the local states. The *g state bundle* b_1, b_2, and b_3 (in Table 1) corresponds to the output signal *a* (a_0-a_2), which are sent to the *Global State Memory* (GSM).

The clock gating logic for glitch-free operation is composed of a NAND gate and an inverter here. Three bits are needed for the local state since M_2 has six states, but only two bits are needed for M_1. The bundled state encoding restriction results in that the lower two bits FF1, FF0 in the *Local State Memory* (LSM) are always active and are therefore directly controlled by the global clock. State bit FF2 is not used in M_1 and is therefore conditionally clocked. The global state bit n_1 controls the clock gating of FF2. The highest bit FF2 is always zero when M_1 is active, in which case it is disabled. When M_2 is active the global state bit n_1 equals one and enable the clock signal of FF2.

.

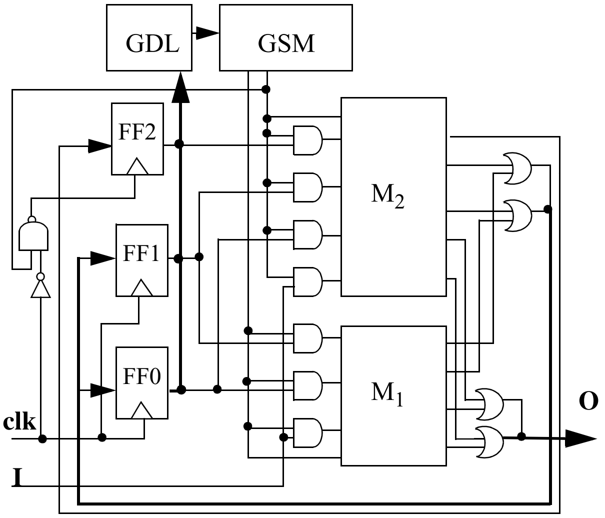


Figure 6. Circuit of a decomposed FSM(dk27)

Besides clock gating, disabling of the inputs to the combinational logic is used to reduce the power dissipation. In our example, the input disabling logic is implemented by three AND gates in front of M_1 and four AND gates in front of M_2. Depending on the global bits, these AND gates can block the state bits and primary input signals from propagating through M_1 or M_2.

Both the primary outputs and the next state values are computed by both sub-FSMs but separated in time. The signals from M_1 and M_2 have to be merged. There are four OR gates. Two of them are used to decide the correct primary output; the other two are used for FF0 and FF1. Note that FF2 is *don't care bit* to the combinational part of M_1 and it is only updated by the next state signal from the combinational part of M_2.

For two-way partitioning, it is shown by Figure 7 that GSM is composed of two asynchronous memory elements AS0 and AS1 with output n_1, n_0 respectively. AS0 is reset by AS1 and set by the signal which is a collection of *g state* in sub-FSM M_2 (see g_1 and g_6 in Table 1). AS1 is reset by AS0 and set by a collection of *g state* in sub-FSM M_1 (see g_2 in Table 1).

Suppose there is a *crossing transition* from s_6 in M_1 to s_2 in M_2. At the beginning, global state bits (n_1,n_0)=01. In the first step, the *local state memory* is updated by the *g state bundle* b_3. In the second step, after detecting b_3, GDL will set the output a_2 to be one and send this signal to GSM. In GSM, together with its own feedback signal n_0=1, g_2 is detected, which set AS1 immediately. AS1 will then reset AS0. Now (n_1,n_0)=10 and the *crossing transition* from M_1 to M_2 is completed. The completion of g_2 signal can be depicted by the signal sequence: g_2+, n_1+, n_0-, g_2-, where "+" represents a monotonical change from 0 to 1, "-" represents a monotonical change from 1 to 0.

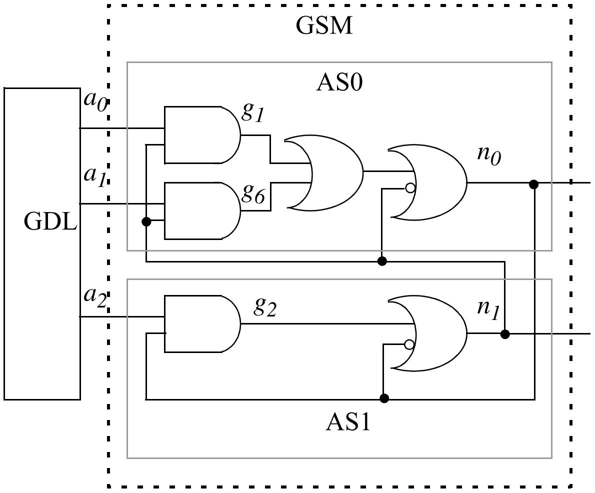


Figure 7. Global state memory structure in dk27

Through this example, the whole procedure for two-way FSM decomposition is explained, also the potential is shown that a good partition with unbalanced size of sub-FSMs can efficiently reduce the area size in the combinational logic. The structure inside asynchronous *global state memory* (in Figure 7) is similar for all two-way partitioning and used in the experiments of the next section.

5. Experimental results

By two-way decomposition, our solution of mixed synchronous/asynchronous state memory was applied on circuits from the standard benchmark set. The number of states in the benchmarks range from 19 to 121 states.

For state partitioning, we use Kernighan-Lin algorithm to find a small cluster of states composing the first sub-FSM and all other states composing the second one [9]. The cost function is based on transition probability and the smaller sub-FSM should has high probability of state transitions inside itself, and low probability of *crossing transitions* to the other sub-FSM.

The power dissipation was obtained from gate level power estimation by Power Compiler (Synopsys), assuming a supply voltage of 1.8V, a clock frequency of 20MHz.

The area estimation was based on the cell area and the target technology is a 0.25μm CMOS standard cell technology.

The primary input probability was set to 0.5 and its switching activity was set to 0.5 also. The stationary state probabilities are computed based on random-walk simulations.

In Table 3, characteristics of the original finite state machine are shown. The circuit name, input, output and number of states are given in the first four columns. The area and power statistics is given in the last two columns.

Table 3. Finite state machine statistics

Circuit	#PI	#PO	#states	area	power
s1488	8	19	48	924.7	155.9
s820	18	19	25	443.9	71.1
s1494	8	19	48	899.5	136.7
styr	9	10	30	427.9	54.3
keyb	7	2	19	271.2	68.0
s832	18	19	25	466.5	75.9
scf	27	56	121	786.1	76.3

* power: *uW*   area: #gate eq

Table 4. Results after decomposition

Circuit	$\lvert S_1\rvert/\lvert S_2\rvert$	$\lvert U_1\rvert/\lvert U_2\rvert$	area	power	%A	%P
s1488	4/44	6/48	821.7	51.4	-11.1%	67.0%
s820	5/20	7/23	505.5	40.2	+13.9%	43.5%
s1494	4/44	6/48	841.0	50.5	-6.5%	63.1%
styr	4/26	6/29	534.8	43.0	+25.0%	20.8%
keyb	4/15	7/16	330.9	39.9	+22.0%	41.3%
s832	3/22	4/24	506.5	39.7	+8.6%	47.7%
scf	6/112	8/114	963.7	46.8	+14.3%	38.7%

* power: *uW*   area: #gate eq

In Table 4, The column labeled “$|S_1|/|S_2|$” shows the state subsets for respective partition in the decomposed FSM. The column labeled “$|U_1/|U_2|$” shows the modified state subsets after introducing *g states*. The following two columns show the area, power of the decomposed FSM. The percentage area increase, power reductions of the decomposed FSMs are shown in the last two columns. An average power reduction of 46.0% is achieved with an area increase of 9.5%. For benchmarks such as *s1488*, power reduction can be up to 70%.

6. Conclusions

In this paper we propose a novel design model for partitioned FSMs that is based on mixed synchronous/asynchronous state memory. In spite of the internal asynchronous operation, the input/output behaviour of the decomposed FSM is equivalent to the synchronous one. By applying this model to a number of standard FSM benchmark circuits using two-way partitioning, we have demonstrated that large power reductions (up to 70%) can be achieved with low or no area overhead.

The partitioning and STG transformations are made automatically in our prototype tool, which takes an STG as input, generates synthesizable RT-level VHDL code that is fed to a standard logic synthesis tool. A standard CMOS cell-library can be used without the need of any special cells.

In this work we have not paid any special attention to the optimization of state clustering and state encoding. We believe that there is room for further power reductions when these issues are addressed.

We also believe the mixed synchronous/asynchronous state memory concept deserves further investigation. By applying it to n-way partitioning, more power reductions can be expected, especially for large FSMs.

7. REFERENCE

[1] 1999 ITRS Roadmap.

[2] L. Benini and G. De Micheli, “Dynamic Power Management - Design Techniques and CAD Tools,” *Kluwer Academic Publisher*, 1998.

[3] L. Benini and G. De Micheli, “*Automatic Synthesis of Low-Power Gated Clock Finite-State Machines*,” IEEE Transactions on Computer-Aided Design for Integrated Circuits and Systems, 1996, vol. 15, no. 6, pp. 630-643.

[4] L. Benini, P. Siegel, and G. De Micheli, “*Saving Power by Synthesizing Gated Clocks for Sequential Circuits*,” IEEE Deisgn and Test of Computers, 1994, vol. 11, pp. 32-41.

[5] E. Hwang, F. Vahid, and Y-C. Hsu, “FSMD Functional Partitioning for Low Power,” in *Proceedings of Design and Test in Europe*, March, 1999, pp. 22-28.

[6] B. Oelmann and M. O’Nils, “Asynchronous Control of Low-Power Gated Clock Finite-State Machines,” in *Proceedings of the IEEE International Conference on Electronics, Circuits, and Systems,* 1999, pp. 915-918.

[7] S-H. Chow, Y-C. Ho, anf T.Hwang, “Low-Power Realization of Finite-State Machines - A Decomposition Approach,” A*CM Transactions on Design Automation of Electronics Systems*, 1996, vol. 1, no. 3, pp. 315-340.

[8] Yang. S, (1991) Logic Synthesis and Optimization Benchmarks *User Guide, version 3.0, MCNC Technical Report.*

[9] J.Monteiro, A.Oliveira “Finite State Machine Decomposition for Low Power,” in 35th Design Automation Conference, June, 1998, pp. 758-763.

[10] B. Oelmann, M. K. Tammemäe, M. Kruus, and M. O’Nils, “Automatic FSM Synthesis for Low-Power Mixed Synchronous/Asynchronous Implementation,” *Journal of VLSI Design 2001, Special Issue on Low-Power Design*, vol. 12, no. 2, pp. 167-186.