Thesis for the degree of Licentiate
Sundsvall 2005

Automatic Synthesis of Partitioned FSMs Based on Mixed Synchronous/Asynchronous State Memory

Cao Cao

Supervisors: Associate Professor Bengt Oelmann
Associate Professor Mattias O'Nils

Electronics Design Division, in the
Department of Information Technology and Media
Mid Sweden University, SE-851 70 Sundsvall, Sweden

ISSN 1652-1064
Mid Sweden University Licentiate Thesis 4

ISBN 91-87908-84-0

Akademisk avhandling som med tillstånd av Mitthögskolan i Sundsvall framläggs till offentlig granskning för avläggande av licentiatexamen i elektronik torsdagen den 24 May 2005, klockan 13.15 i sal O102, Mitthögskolan Sundsvall. Seminariet kommer att hållas på engelska

Automatic Synthesis of Partitioned FSMs Based on Mixed Synchronous/Asynchronous State Memory

Cao Cao

Electronics Design Division, in the
Department of Information Technology and Media
Mid Sweden University, SE-851 70 Sundsvall
Sweden

Telephone: +46 (0)60 148925

Printed by Kopieringen Mitthögskolan and Kaltes Grafiska AB, Sundsvall, Sweden, 2005

To my parents

ABSTRACT

The rapid development of digital circuits with high density and frequency motivates power, in addition to area and speed, to become an important parameter in design constraints. Nowadays, the electronics design industry is confronted by increasingly costly package and cooling systems due to power dissipation. Battery-powered portable devices, such as laptops, mobile phones etc., which provide higher computational capacity and support multi-media information transformation, greatly increase the previously rather small power budget. As synchronous digital design has, over the past few decades, become the industry standard, this new challenge means that asynchronous design techniques must now be reconsidered, as they possess the potential for a reduction in power dissipation.

Finite state machine (FSM) partitioning proves effective for power optimization. In this thesis, a mixed synchronous/asynchronous state memory structure in the decomposed FSM is proposed, which results in implementations with low power dissipation and low area overhead. The state memory 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. Although asynchronous communication mechanism is introduced between sub-FSMs, the input/output behaviour of the decomposed FSM is still, cycle by cycle, equal to a complete synchronous one. Power consumption can be further reduced by using a clock gating technique and low power state assignment.

Based on this mixed synchronous/asynchronous structure an automatic synthesis tool was developed, which accepted state transition graph (STG) as input and outputted synthesizable VHDL code that can be directly used for logic synthesis. An FSM partitioning algorithm, power estimation functions and state encoding optimization aimed at this specific structure are also integrated into the tool to find low power partitioning within a reasonable run time. The effectiveness of the whole procedure is verified through optimization of standard benchmarks where a power reduction of up to 70% has been demonstrated.

ACKNOWLEDGEMENTS

First and foremost, I am deeply grateful to my supervisor Bengt Oelmann whose perspective and insight contributed both to the initial concept of my research work and at every stage of its development. Thanks for all the guidance and discussions that helped me stay on the right track and inspired me to put thoughts into action. I would also like to thank my current assistant supervisor Dr. Mattias O'Nils, who proposed the "candidate generation" algorithm incorporated in the automatic synthesis tool developed, and my former supervisor Professor Hans-Erik Nilsson.

Special thanks go to Shang Xue. I am so lucky to have you as my roommate. I really appreciate all the help you gave me when I first arrived in Sweden, knowing nothing about the place and the conversations with you will always be my precious memory.

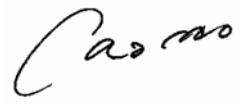
All exceptional people in Electronic design division have my gratitude for the kindness and friendship that makes it such a pleasant place to work in. Namely thanks to Lixin Ning, Krister Alden, Mats Hjelm, Fanny Burman, Jon Alfredsson, Henrik Andersson, Jan Lundgren, Suliman Abdalla, Johan Siden, Claes Mattsson, Hakan Norell, Torbjorn Olsson, Borje Norlin, Benny Thornberg, GoranThungstrom. I would also like to mention Tao Feng and Xiaosong Ding for their friendship, also other PhD students not mentioned but who have had wonderful conversations with me.

Financial support from the Mid Sweden University and the Foundation for Knowledge and Competence Development (KK-stiftelsen) is also gratefully acknowledged.

Finally, I want to express my love to my family. My grandmother, parents and elder brother, you are always the most important part of my life.

And, to Lebing Gong, thank you for always being there.

Sundsvall, March 2005



Cao Cao

TABLE OF CONTENTS

ABBREVIATIONS AND ACRONYMS

GENERAL

ALU……..	Arithmetic Logic Unit
CAD……..	Computer Aided Design
CMOS……...	Complementary Metal Oxide Silicon
DSP……...	Digital Signal Processor
EMI……..	Electro Magnetic Interference
FSM……..	Finite State Machine
FSMD.....	FSMs with Datapath
GALS.....	Globally asynchronous Locally synchronous
GDL……..	G State Bundle Detection Logic
GSM ……	Global State Memory
IC…….....	Integrated Circuit
K-L……...	Kernighan-Lin
LSM…….....	Local State Memory
RT…….	Register Transfer
RTL……..	Register Transfer Level
STG……..	State Transition Graph
Syn/Asyn……..	Synchronous/Asynchronous
VHDL……...	Very High Speed Hardware Description Language
VLSI…….	Very Large Scale Integration

LIST OF FIGURES

LIST OF PAPERS

This thesis is mainly based on the following 3 papers, herein referred to by their Roman numerals:

Paper I **Mixed Synchronous/Asynchronous State Memory for Low Power FSM Design**
Cao Cao and Bengt Oelmann,
Proceedings of EUROMICRO Symposium on Digital System Design, pp. 363-370, France, 2004.

Paper II **A Tool for Low-Power Synthesis of FSMs with Mixed Synchronous/Asynchronous State Memory**
Cao Cao, Mattias O'Nils, Bengt Oelmann,
IEEE Norchip Conference, Oslo, Norway, 2004
(Selected for publication in the Proceedings of the IEE Computer & Digital Techniques)

Paper III **State-Encoding for Partitioned FSMs with Mixed Synchronous/Asynchronous State Memory**
Cao Cao and Bengt Oelmann,
Submitted to the Proceedings of the IEE Computer & Digital Techniques, 2005.

1 INTRODUCTION

1.1 MOTIVATION FOR LOW POWER

Historically, digital integrated circuit design focused on the optimization of area and speed. Power consumption was often of secondary concern. In recent years, however, there has been a rapidly growing interest in low power design. Among the factors contributing to this trend, one most remarkable driving force stems from the portable consumer electronics applications.

The portable consumer electronics market continues to develop at a rapid rate. Laptop computers, cellular phones, digital video cameras etc., all of these portable devices require powerful systems that run on lightweight battery packs. Reducing power consumption is obviously a primary concern here for prolonging the operational life of a particular battery technology.

Besides portability, the more generic motivation for low power originates from the heat dissipation problem. Nowadays, high-end products, such as microprocessors, are designed with increasing circuit integration and faster clock frequencies. Subsequently, the magnitude of power per unit area is growing and a considerable amount of heat is generated. High temperature can affect the reliability and shorten the lifetime of such systems. To address this problem, either costly packaging technology or cooling devices should be introduced, or, the chip has to be divided into several chips, which thus directly limits the circuit integration capability. In [4], it was concluded that the constraint facing microprocessors with reference to the die size is introduced by the power dissipation and not the fabrication ability.

As a result, present day circuit designers must explore area, speed and power to find suitable solutions. The available choices are expanded and in the meantime the required complexity is also increased.

1.2 SOURCES OF POWER DISSIPATION

CMOS circuits (which combine PMOS and NMOS transistors) are the dominant technology for modern high-performance digital electronics. The average power consumption of a CMOS circuit can be modeled by the following equation:

$$P_{avg} = P_{switcing} + P_{short\_circuit} + P_{leakage} \quad (1)$$

The first term represents the switching power component. In a circuit, it can be expressed as:

$$P_{swithing} = \frac{1}{2} V_{dd}{}^2 f_{clk} \sum_{i=1}^{N} \alpha_i C_i \quad (2)$$

where V_{dd} is the supply voltage, f_{clk} is the clock frequency, α_i is the average number of logic transitions of node *i* per clock cycle, and C_i is the loading capacitance at node *i*. When V_{dd} and f_{clk} are settled, the power reduction stems from the reduction of $\sum_{i=1}^{N} \alpha_i C_i$, denoted as the effective capacitance in the rest of the thesis.

The second term is due to the direct-path arising when both the NMOS and PMOS transistors, in a static CMOS gate, are simultaneously conducting and a short-circuit current is going directly from the supply to the ground.

The final term originates from various leakage currents that exist for idle CMOS gates. It should be noted that leakage power has become an important component for the whole power dissipation and will be comparable to the switching power as the feature size continues to decrease [1].

Because $P_{switching}$ is still the dominant term in static CMOS gate circuits [2], in this thesis, only the switching power (or dynamic power) is considered. In the rest of the thesis, the word "power" means switching power if not specified.

1.3 LOW POWER DESIGN METHODOLOGY

Low power design can be performed at all levels of abstractions. Typical abstraction levels, in descending order, are shown in Figure 1. They are system, architecture (or algorithm), register transfer (RT), gate, circuit and technology levels. The most commonly used power optimization techniques at each level are also shown.

At the system level, since the system can be viewed as a hardware platform executing software program, a partitioning strategy, which decides whether a task should be implemented in the hardware or the software, can be exploited to minimize power dissipation [3]. Power management schemes can also be used to shut down the idle system (or the system's various components) to reduce power [5]. In [9], power management is applied to a digital signal processor (DSP) design. As a result, the power consumption of the DSP in idle-mode was less than 1/10 of the original un-optimized one.

It is apparent from Equation (2) that reducing the power supply voltage can decrease the power quadratically. However, when the supply voltage is reduced, the power-delay product of CMOS circuits also decreases and the delays increase monotonically. To compensate for the speed penalty introduced by voltage scaling, at the architecture level, transformations, such as pipelining and parallelism [12], are employed to increase the level of concurrency.

At the RT level, a circuit can be considered to be the sequential logic, composed of the memory elements (registers) and functions responsible for determining not only the state but also the data computation. Power optimization at this level can be roughly categorized into two classes. One class is state assignment and the other is an extension of the dynamic power management from the system

level to the RT level [5]. More details with regard to the dynamic power management at the RT level will be given in the following chapters.

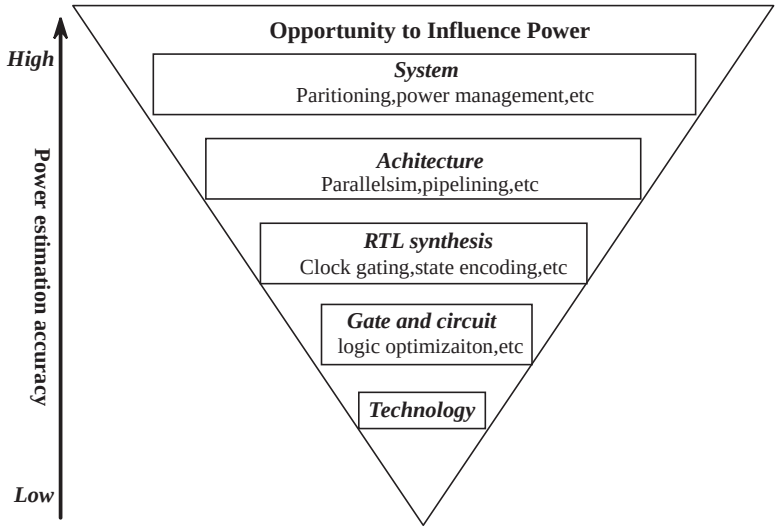


Figure 1. Design abstraction level

At the gate and circuit levels, logic optimization methods, such as transistor reordering, can be used to reduce switching activity and subsequently reduce power dissipation [10]. Design styles of global signals, such as bus architecture configuration, can result in low power implementations by reducing the physical capacitance [59].

At the technology level, methods such as reducing both the threshold voltage and power supply voltage and scaling transistor sizes [11] can be used for low power design.

1.4 POWER-CONSCIOUS SYNTHESIS TOOL

Computer aided design (CAD) plays an important role in the development of integrated circuits. When transistors can be counted in millions in contemporary circuits, it is impossible to synthesize manually without the assistance of CAD tools.

A complete synthesis flow from the behavioural specification to the final fabrication is shown in Figure 2. Each synthesis step translates a description of the circuit to an optimized description at a lower level. At each level, estimation for area, timing (speed) and power can be incorporated into the synthesis process to verify whether or not the solution satisfies the design's constraints.

Because area and speed have, for a long time, been the major design concerns, a number of industrial standard synthesis tools are associated with these

areas. In contrast, power-conscious synthesis tools are a relatively new area and the focus has been primarily at lower levels.

In general, when optimizations are introduced at the higher abstraction levels, larger power reductions can be expected [2] since the design space to be explored is larger. However, the accuracy of the power estimation is in inverse proportion to the design space. The lower the level, the more information is available regarding the implementation of the design (see Figure 1). Therefore, when the possibility of employing a global strategy to achieve significant power reduction at higher levels exists, the lack of detailed implementation information makes it difficult to evaluate the quality of the strategy. Based on the above, power-conscious tools at higher levels are more significant, but also more difficult.

For power analysis (or estimation), mature commercial tools such as SPICE and PowerMill are available at the circuit and gate level and they provide accurate power values. However, the solutions at higher levels come mainly from academia [62].

As to power optimization, although considerable methodologies have been proposed [12], an industry standard framework for synthesizing low power circuits has not yet been developed. Synopsys can be used for synthesizing low power circuits at the gate level. However, the framework is designed to fulfil area and speed constraints, so necessary critical information for power estimation and optimization is not considered in the power-conscious procedure.

As an effort to provide a comprehensive environment for low power design, in this thesis, an automatic synthesis tool at the RT level is presented incorporating power analysis and optimization.

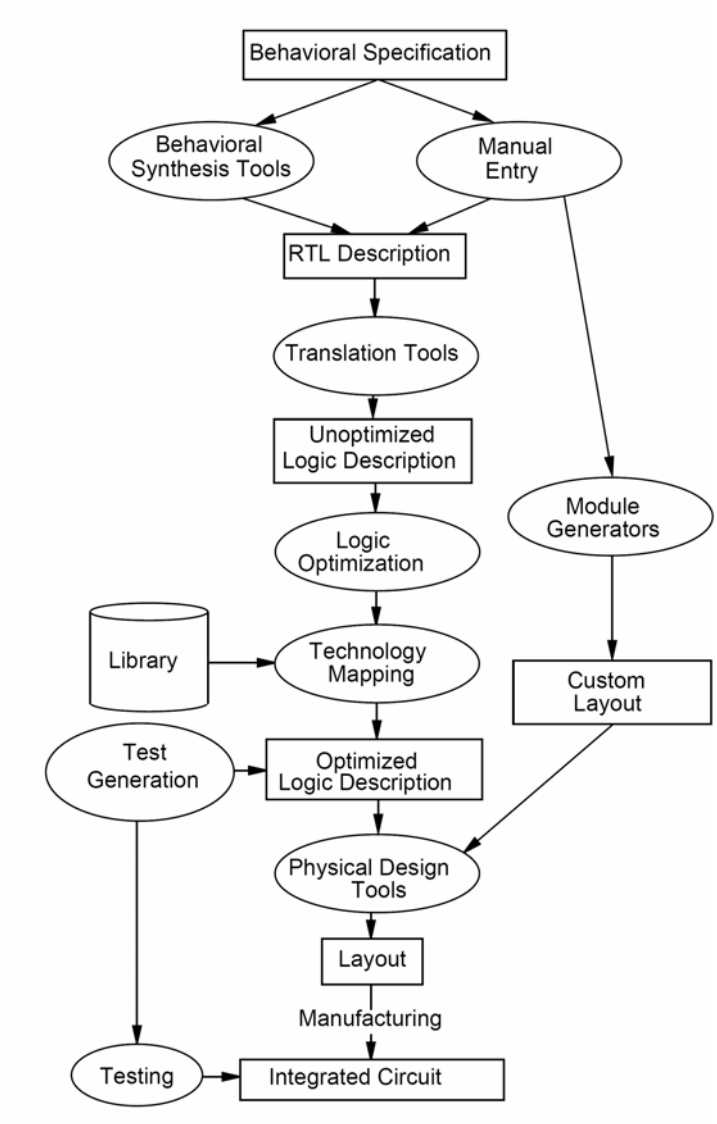


Figure 2. Synthesis design flow from [57]

2 FSM LOW POWER DESIGN

At the RT level, a design synthesized from a higher level can be viewed upon as an interacting system composed of two parts: controller and datapath. Given that the controller is always running, it may consume a great deal of power (about 40% of the total power is consumed in the controller [40]). Since the controller is often implemented as finite state machines (FSM), the power reduction problem reformulates to FSM power minimization. In this chapter, a background concerning FSM is presented (section 2.1), followed by the two most important design aspects targeting FSM power optimization, that is, the application of dynamic power management at the RT level (section 2.2) and state assignment optimization (section 2.3).

2.1 FSM FUNDAMENTALS

The general structure of a design at RT level is shown in Figure 3. It consists of a datapath that is a network of ALUs (arithmetic logic units), multiplexers, registers and busses, responsible for data storage and manipulation. The controller is represented as the FSM that controls data transfers in the datapath.

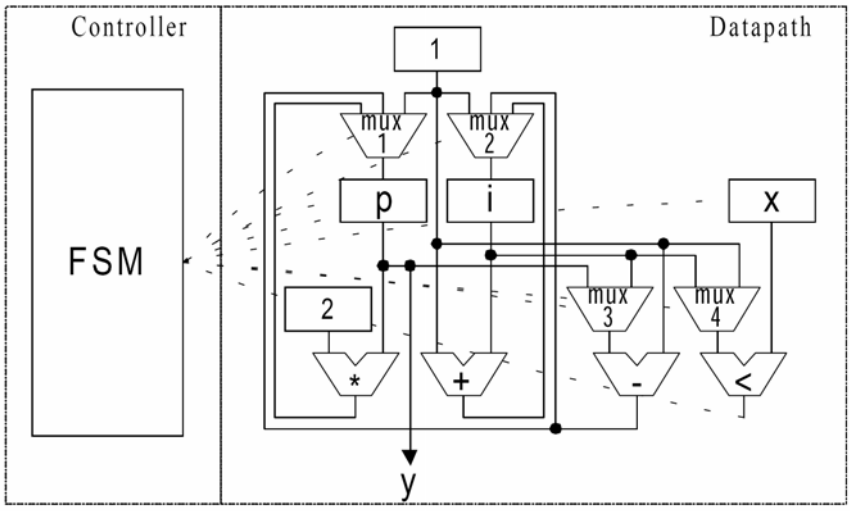


Figure 3. RT level design structure from [58]

The name of finite state machine (FSM) comes from the fact that it consists of a finite number of states and its formal definition can be found in [18]. As shown in Figure 4a), state transition graph (STG) is widely used to describe the behaviour of an FSM, where every state is labeled as a node with a unique

symbolic name and the state transitions among them are represented as edges with input and output values.

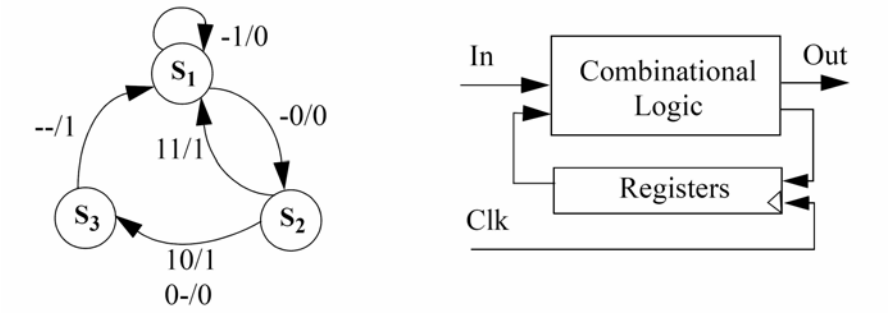


Figure 4. FSM representation

From a circuit point of view, it is shown in Figure 4b) that FSM is normally implemented as a synchronous model composed of combinational logic and registers. In every clock cycle, the combinational logic is responsible for calculating the next state and output value while the registers store the updated state information.

2.2 DYNAMIC POWER MANAGEMENT

2.2.1 Introduction

Benini et al. proposed the concept of dynamic power management [5] which is based on idleness exploitation. 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 thus always leads idle units being available. In these situations, dynamic power management may be successfully used. Firstly, it highly accurately detects idleness; secondly it rapidly shuts down the idle resources and forces it to a state where power dissipation is as low as possible. Since a power management scheme is able to eliminate a fraction of the useless switching activity that consumes power without producing useful results, it proves to be effective at various levels of abstractions. Its exploitation at the RT level is the main focus of the rest of this section.

2.2.2 FSM idleness exploitation

Many FSM low power methods can be collectively viewed upon as the exploitation of idleness, *internal* or *external*. When outputs of an FSM are observable to primary outputs but remain unchanged, *internal* idleness can be exploited. In [6], under the condition of self-loops where both state and primary

output values remain constant, the whole FSM can be shut down after adding state-holding mechanism.

When an FSM is decomposed into sub-systems, the output of a sub-network may change but not influence the primary output. In this case, *external* idleness can be exploited. As opposed to *internal* idleness, *external* idleness is induced by the environment, and depends on the entire output behaviour of the system. For example, in [19], after introducing pre-computation methodology, the original synchronous network is decomposed into two sub-networks. One of them is unconditionally clocked while the other can be conditionally shut down if the calculation performed is irrelevant to the network output, that is, *externally* idle.

A more aggressive method of exploiting the *external* idleness of FSM is FSM decomposition. The original FSM is partitioned into two or more sub-FSMs where only one of them is active at a time and others can be deactivated without consuming power since their outputs are unobservable (or irrelevant) to the primary outputs [20]. The partitioned FSM is constructed in such a way that each of the sub-FSMs constitutes a smaller effective capacitance than the original FSM and consequently power can be saved.

2.2.3 Shut-down circuitry

To prevent idle components from consuming switching power, dynamic power management techniques disable the clock signal or, make input values to the parts not in use remain constant. Mechanisms for detecting when the unit is idle then shutting it down must therefore be added to the design. Circuits responsible for handling this mechanism will constitute a functional overhead and will consequently contribute to the increased circuit area, additional power consumption, and possibly reduced performance. Careful analysis must be undertaken so that the introduction of circuits for power management will contribute to as little power consumption as possible.

2.2.3.1 Clock gating

As shown in Figure 5, the clock gating logic (CL) accepts the clock signal Clk and the control signal CNTRL as its inputs and generates the gated clock signal (Gclk) as its output to control the update of registers. When the gated clock is stopped by CNTRL, power consumption can be minimized in combinational logic because the flip-flops are not triggered on any rising clock edge, hence their outputs remain unchanged. The disadvantage of this method is that the presence of a gate in the clock line usually increases clock skew, which may cause problems in high performance design [6].

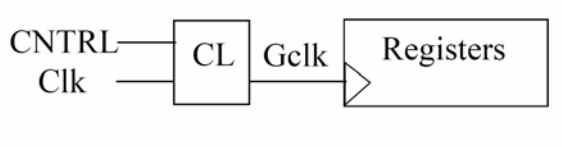


Figure 5. Gated clock for shutting down

2.2.3.2 Input disabling

In Figure 6, combinational logic can be selectively turned off by the input disabling logic (IL), which consists of transparent latches with an enable signal EN. When units are executing useful calculation, EN makes the latches transparent, thus permitting normal operations. If this does not occur, the latches retain their previous state and no transitions propagate through the inactive units. This method is called the guarded evaluation in [23] where both a theoretical framework and algorithms, which automatically decide when the logic units performing useless calculations should be shut down, are provided.

Compared with the clock gating technique, this method is less power effective because the power in the clock line is not saved. However, in the case where two functions share the same register but never work simultaneously, the register should remain active and the clock gating methodology cannot be exploited. By disabling the input to each function, it is still possible to reduce the power. Also, an input disabling strategy is safer than clock gating when considering timing issues. Note that in either method it is impossible to avoid leakage power as it does not depend on signal transitions.

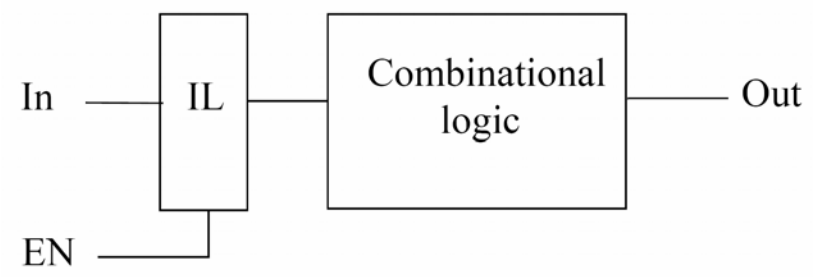


Figure 6. Disabled input for shutting down

2.3 STATE ENCODING

State encoding，which strongly influences the final realization of an FSM, has been an active research area for decades. Until the early 1990's, its main objective was towards area optimization for two-level or multilevel logic [24]. The requirement for low power, high computing portable systems determined the current focus on state assignment optimization for power. Generally, the search

area for state encoding is too large to explore, therefore, approximate methods, depending on pre-logic cost functions, are used to obtain the optimal solution.

From Equation (2) it can be seen that dynamic power is related to both area (the total number of nodes) and switching activity, therefore, state encoding for low power is, to some extent, more difficult than for area minimization. To simplify this problem, in [26], the cost function assumes that power consumption is proportional to the switching activity of state bit lines. The problem concerning power reduction is reformulated to reduce the Hamming distance of state transitions that have a high probability. Both minimum length [8] and non-minimum encoding are subsequently developed [27]. In [29], two code lengths are used in the same state machine. After the introduction of the Huffman coding algorithm, states that are highly probable of being active are coded with less than $\lceil \log|S| \rceil$ state bits, where |S| is the number of states. Other states, which have less likelihood of being active, are assigned state bits greater than $\lceil \log|S| \rceil$.

Since reducing switching activity in state lines does not always lead to reduced power in the combinational logic, efforts are also being made to take area into account. Among them, Benini et al [8] adds the area constraint to the cost criteria and explores the trade-off between computation complexity and the quality by using different algorithms. Olson et al [31] use the linear combination of the switching activity and the number of literals as the cost function. Tsui et al [30] propose the power model, considering switching activity and capacitive loading simultaneously. All the above state encoding methods aim at monolithic FSM optimization. Low power state assignment in decomposed FSM will be further discussed in chapter 4.

3 MIXED SYNCHRONOUS/ASYNCHRONOUS STRUCTURE

In terms of operation mode, digital circuits can be classified into two categories: synchronous and asynchronous. In synchronous circuits, information storage or process is orchestrated by one global signal, called the clock signal. Conversely, asynchronous circuits remove the clock signal and locally generated timing signals are used to ensure proper control of the sequence of events. Nowadays, even though synchronous systems dominate the circuit design field due to their simple rules, asynchronous systems are being looked as an increasingly viable alternative to purely synchronous systems. In this chapter, the advantages and disadvantages of both classes are discussed from various design perspectives (section 3.1), then the concept of mixed synchronous/asynchronous design as well as its implementation is presented (section 3.2).

3.1 SYNCHRONOUS AND ASYNCHRONOUS DESIGN COMPARISON

With the rapid development of digital circuits, the limitations facing purely synchronous designs offer asynchronous designs the possibility to realize their potential. The understanding of the properties of both operational modes from various design aspects enables the design space to be explored more freely and reveals the reason behind mixed synchronous/asynchronous design.

- **Design efficiency**

In a synchronous system, a designer can simply define the combinational logic necessary to compute the given functions, and surround it with latches (or registers). By setting the clock rate to a long enough period, all worries about hazards (undesired signal transitions) and the dynamic states of the circuit are removed. However with asynchronous systems, a great deal of attention must be paid to the dynamic state of the circuit. Hazards must also be explicitly removed from the circuit or, not introduced in the first place, to avoid incorrect results [32]. The ordering of operations, which is fixed by the placement of latches in a synchronous system, requires careful execution through the asynchronous control logic. As reducing the design cycle is a necessity in the present intense industrial competition, the overwhelming design efficiency of the synchronous circuit means that it constitutes the bulk of commercial practices as well as CAD tools.

- **Clock skew problem**

Clock skew is the difference in arrival times of the clock signal in different parts of the circuit and it restricts the maximal frequency achievable by the clock. In current high speed, highly complex circuits, it is very costly to limit the clock skew to an acceptable range and sometimes systems have to be slowed down to

accommodate the skew. This problem has already been noted in [33]. In the design of DEC Alpha CPU, keeping the clock skew within 300 picoseconds results in a clock driver circuit that occupies 10% of the circuit area and consumes over 40% of the power. For asynchronous circuits, which by definition have no globally distributed clock, this problem does not exist. As feature sizes decrease, the clock skew problem, which is inherent in the synchronous design, will become more serious in the future.

- **Area**

To provide glitch or hazard free outputs in the timing constraints, asynchronous design must introduce extra logic. Also, the control signals necessary for initializing an action or denoting the completion of the action [34] make the asynchronous system generally larger than its functionally equivalent synchronous counterpart. The generation of area overhead may cause performance degradation or, consumes considerable power.

- **power**

Standard synchronous circuits have to toggle clock lines, and possibly precharge and discharge signals, in portions of a circuit that remain idle in the current computation. Although power management can partially remove the wasteful power dissipation, it only works at a course granunarity and introduces area overhead. Asynchronous circuits, by their nature, only activate the units currently involved in useful calculation and therefore result in lower power solutions [35].

- **Performance**

Synchronous circuits must wait until all possible computations have been completed before latching the results, so the chosen fixed clock period must accommodate the worst-case timing condition. Average-case or best-case performance can not be explored. Many asynchronous systems, on the other hand, sense immediately when a computation is complete. This inherent adaptivity allows them to exhibit average-case performance. For circuits where the worst-case delay is significantly worse than the average-case delay, an asynchronous implementation can result in a better performance [36]. But it should also be noted that asynchronous circuits generally require extra time due to their signaling policies, hence cause an increase in the average-case delay. Whether this cost is greater or less than the benefit differs from case to case.

- **Technology migration potential**

During their lifetime, integrated circuits are often implemented in several different technologies. Early versions of systems may be implemented using gate arrays, while later products may migrate to semi-custom or custom ICs. Greater performance for synchronous systems can often only be achieved by migrating all system components to a new technology, since again the overall system

performance is decided by the longest path. In contrast, many asynchronous systems are able to migrate only the more critical system components in order to achieve higher performance, since performance is based on the currently active path. Furthermore, the adaptivity of asynchronous systems makes it possible for components with different delays to be combined into a larger asynchronous system without any special structural alteration, whereas careful analysis is required for synchrnous circuit. The modularity in asynchronous circuits is demonstrated in [37].

- **EMI and noise**

Without the clock, noise and electro magnetic interference (EMI) spectrums are significantly flatter across the entire frequency domain. According to McCardle et al. [38], there can be a 10-dB drop in noise in an asynchronous processor. Until recently, EMI and noise metrics were ignored when area, speed or power were being considered. But EMI and noise metrics are now attracting more attention due to two emerging applications: mixed-signal design and smart cards. In the former, analog functions are particularly sensitive to clock-correlated, digital switching noise. Reducing noise and EMI will significantly boost both precision and performance. In the latter, EMI has a significant impact on security. Non-invasive security attacks depend on monitoring a smart card's power usage, or EMI signature, to extract key information on the card. Even distribution of circuit-switching activities in the asynchronous system obviously improves security [39].

Even though asynchronous design is not the mainstay of commercial practice, its beneficial properties with regards to low power, low noise etc., suggests that instead of having completely synchronous systems the introduction of asynchronous methodology offers great potential for the future. This confidence has also acted as the inspiration for the research on mixed synchronous/asynchronous design, dealt with in greater detail in the next section.

3.2 MIXED SYN/ASYN APPLICATION FOR LOW POWER

Industrial standard asynchronous CAD tools are far from mature and the temporal trends in mixed synchronous/asynchronous design thus involve the exploitation of some proven benefits of the asynchronous circuit in a largely synchronous environment. In this case, the widely accepted synchronous system design methodology can be utilized and the asynchronous design can be taken advantage of simultaneously. In this section, the mixed design concept at the system level is introduced. After the comparison between two different implementation models of the state memory is given, an RT level mixed synchronous/asynchronous design method is proposed.

3.2.1 System level mixed synchronous/asynchronous design

In a synchronous circuit, the clock signal connects every part, registers, latches and also the pre-charge and evaluation transistors of dynamic gates. These elements constitute a huge capacitance load on the clock line which is further added to by the capacitance of clock wire itself. The total capacitance in the clock line makes the clock net power dissipation in a high frequency circuit unacceptable. It has been demonstrated in an Alpha 200MHZ processor that 40% of the whole power originates from clock [33]. To tackle this problem, at the system level, asynchronous logic can be introduced as the interfacing circuit to synchronous modules and the requirement of a global clock is thus removed.

The concept of globally asynchronous, locally synchronous (GALS) was founded by D. M. Chapiro [63] to avoid the costly global synchrony in large scale VLSI circuit. Its basic model is shown in Figure 7 where the main modules are synchronous but the data exchange between any two modules is handled by an asynchronous handshake protocol. A prototype GALS system is built in [41] by using pausible clocking control to prevent synchronization failures. The effects of GALS approach is verified by Hemani et al. [42] with a power reduction of up to 70% in the clock net and a 20% reduction in the overall dissipation compared to a conventional globally synchronous design.

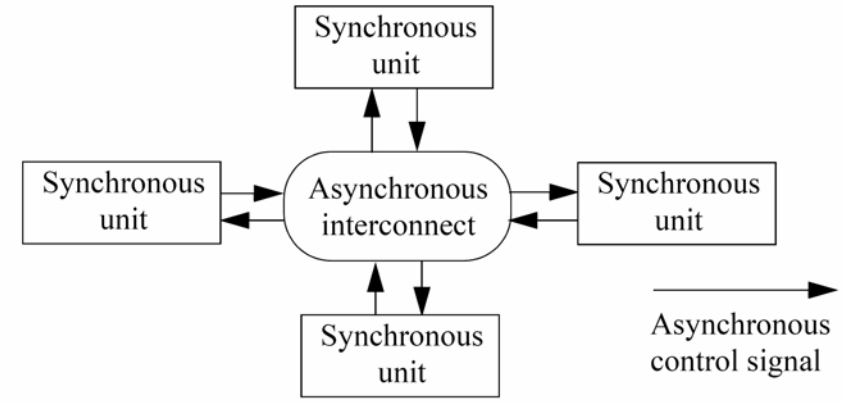


Figure 7. GALS basic model

3.2.2 RT level mixed synchronous/asynchronous design

Generally at the RT level, the finite state machine is implemented completely synchronously. Efforts made towards mixed synchronous/asynchronous design involve the introduction of asynchronous communication into the sub-FSM network after FSM decomposition. Meanwhile, the input and output behaviour is still cycle by cycle equivalent to a complete synchronous one.

In the decomposed FSM design, there are two ways of implementing state memory as shown in Figure 8.

The first method is rather straightforward. After FSM partitioning, each of the sub-FSMs has its own state memory, see Figure 8a). These state memories are local to the sub-FSMs and named after *local state memory*. Global state is not required while reset states, one in each sub-FSM, are added to the local state subsets. An additional signal interface is introduced between sub-FSMs to activate or deactivate them. This approach has, for example, been used in a fully synchronous partitioned FSM by Benini et al. [7]. Its disadvantage is the area overhead introduced by the additional flip-flops. In some sense, these local state memories are redundant because only the one in the current active sub-FSM is of importance for storing the state information. In the meantime those remaining in the deactivated sub-FSMs are not useful.

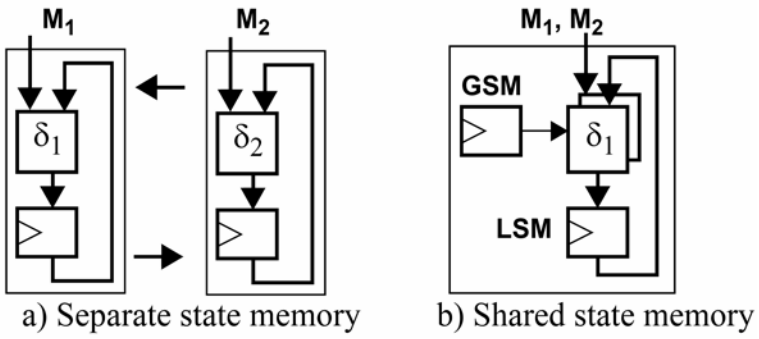


Figure 8. State memory structure in decomposed FSM

In contrast, Chow et al. [21] propose a structure where the *local state memory* (LSM) is shared by all the sub-FSMs, as depicted in Figure 8b). By dividing the states into two parts, global states and local states, the local state bits can be shared among the sub-FSMs whereas the global states are used to determine the active sub-FSM. States residing in different sub-FSMs can therefore use identical local state codes and be distinguished by different global states. The total number of flip-flops required in the state memory will be lower in comparison to that for separate state memory implementation. However, from the power consumption point of view, the disadvantage concerns the flip-flops introduced for *global state memory* (GSM, the memory of global states). These flip-flops are always clocked and will add substantially to the power consumption.

It has been proposed in [43] that an asynchronous communication protocol is more power efficient than its synchronous counterpart in the decomposed FSM. This idea of mixed synchronous/asynchronous design in FSM partitioning is implemented in an automatic synthesis tool in [56]. It uses separate synchronous *local state memories* for sub-FSMs but the disadvantage is the substantial area overhead.

Targeting an implementation with low power and low area overhead, the idea is now suggested that a shared synchronous *local state memory* should be in the part always clocked and an asynchronous *global state memory* should be used to decide which sub-FSM is active. *Global state memory* has a low probability of being updated. It is idle most of the time and therefore adds very low power overhead. By using clock gating technique in the *local state memory*, power dissipation can be further reduced. The mixed synchronous/asynchronous state memory structure is shown in Figure 9, where the input/output behaviour is cycle by cycle equivalent to that of a non-decomposed synchronous one.

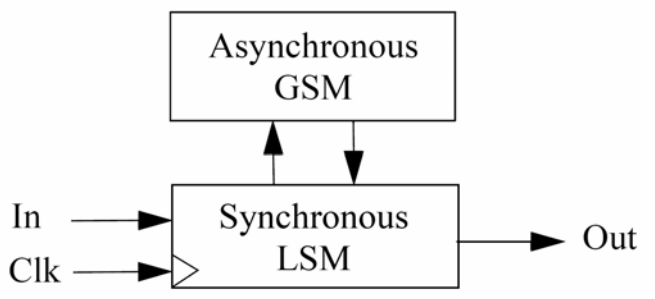


Figure 9. Mixed synchrnous/asynchronous structure

Based on this structure, an automatic synthesis tool for low power decomposed FSM implementation is also developed, which will be described in the next chapter.

4 AUTOMATIC SYNTHESIS TOOL

With the increasing design complexity, designers have to resort to automatic tools to speed up the design process. At present there are many mature CAD synthesis tools which target area and performance optimization. However, low power design, particularly for higher levels, is still far more of an art than a standard industrial practice. In an effort to address this problem and normalize the design process for power optimization, an automatic synthesis tool at the RT level, which is based on mixed synchronous/asynchronous state memory, has been developed. In this chapter, an overview of the whole design flow of this tool (section 4.1) is followed by a detailed description of each step in the flow. Firstly, effective ways of collecting information from the input of the tool (section 4.2) are discussed. FSM partitioning algorithm is then considered (section 4.3). The required transformation steps for this mixed synchronous/asynchronous state memory implementation as well as the associated state assignment problem (section 4.4) are then presented. Following this, the power estimation model at the RT level is built (section 4.5). Finally, the format of the tool output and the related technology information are described (section 4.6).

4.1 DESIGN FLOW DESCRIPTION OF THE TOOL

Starting from a single state transition graph (STG) description, a procedure is proposed for automatically synthesizing a monolithic FSM into a network of interacting sub-FSMs. A standard-cell based design flow (see Figure 10) is assumed, which means that there are no special library requirements beyond that normally provided. However, the tool does require some cell library dependent information to perform accurate power estimations and to define the gate level implementation of the asynchronous elements.

In Figure 10, in addition to STG specification, the signal probabilities of the primary inputs are also given in order to generate a long series of inputs to the STG simulator. The outputs of the simulator are probabilities related to the states and primary outputs of the FSM. According to the mutual state transition probabilities derived from the STG simulator, states are firstly clustered into a hierarchical tree. A novel algorithm is then adopted to group the clusters at each level and form a limited number of partitioning candidates. Each candidate is subsequently synthesized to an RT level description and its power dissipation is measured by the cost function. The candidate with the lowest power is considered to be the best, and its RT level VHDL description and synthesis scripts are finally generated. The VHDL file and the scripts can be used directly as the inputs to a standard synchronous tool for the optimization of the decomposed FSM at the gate level.

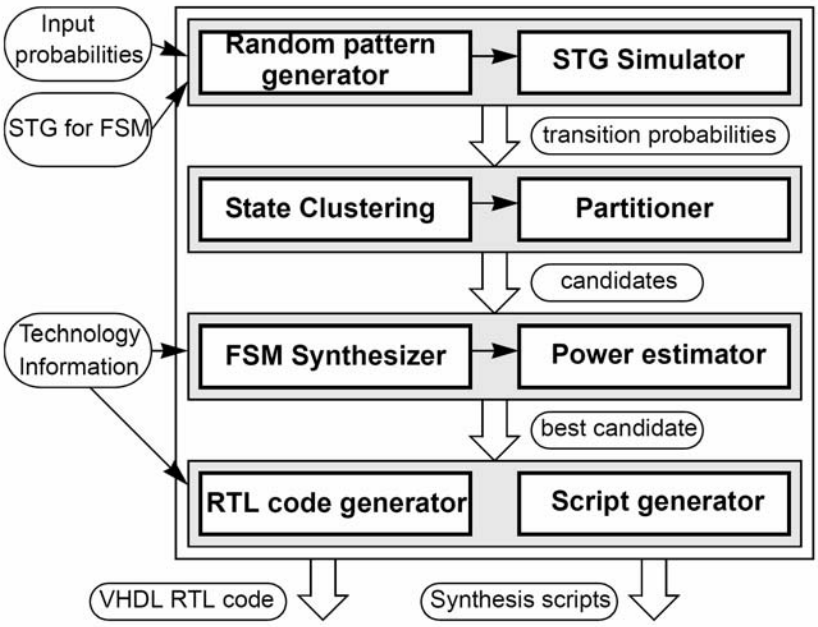


Figure 10. Tool design flow

4.2 STATISTICS COLLECTION

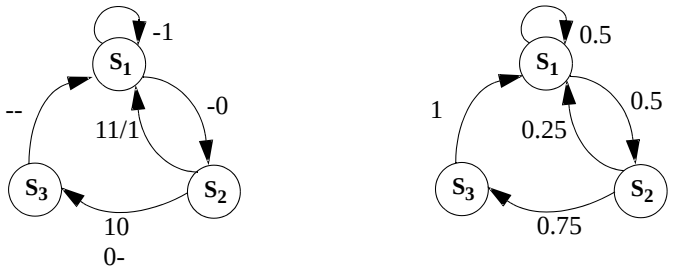
Power dissipation is strongly dependent on the switching activity of the circuit, which in turn is related to the input pattern. For effective FSM partitioning and power estimation, the first step is to specify information about the primary inputs of the FSM. Other FSM statistics, such as state and primary output related probabilities, can be obtained subsequently. To obtain the above information, there are basically two ways depending on the knowledge available about primary inputs.

4.2.1 FSM probabilistic model

If the information of the inputs is provided by input probabilities, i.e., the probability of the value of the input to be one, the STG behaviour of an FSM can be modelled as a Markov chain [8]. A Markov chain represents a finite state Markov process, where the probability distribution at any time is decided only by the current state, regardless of how the process reaches that state. The Markov chain model for the STG can be described as a directed graph isomorphic to the STG with weighted edges.

In Figure 11a), input configurations for state transitions are labelled on the edges of the STG. It is assumed that all input probabilities are 0.5, that is, $Prob(i_1)=Prob(i_2)$ = 0.5. The corresponding Markov chain model of the STG is shown in Figure 11b). In the Markov chain model, edges are weighted using the *conditional transition probability*, that is, weight $p_{i,j}$ on the corresponding edge

represents the probability of a transition to state s_j given that the machine is in state s_i. For instance, the transition from s_2 to s_1 occurs when the input is "11" and the corresponding *conditional transition probability* is $p_{2,1} = Prob(i_2) \times Prob(i_1) = 0.25$, as shown in Figure 11b). Primary inputs are assumed to be independent of each other in this case.



a) a FSM b) Its Markov chain model

Figure 11. A FSM example

Conditional transition probability itself is not sufficient to represent the probabilistic property of an FSM. For example, if the *conditional transition probability* from s_i to s_j is high but the FSM will never reside in s_i, the actual transition probability between the two states is still zero. Hence, the *total transition probability* is introduced, independently to the present state of the FSM. *Total transition probability* is the product of the *conditional transition probability* and the *static state probability*. The *static state probability* represents the probability of a state that the FSM will reside in when time increases to infinity. The calculation of the *total transition probability* $P_{i,j}$ can be expressed as:

$$P_{i,j} = p_{i,j}P_i \quad i, j = 1, 2, \ldots, |S| \tag{3}$$

where $p_{i,j}$ is the *conditional transition probability* from s_i to s_j, P_i is the *static state probability* of state s_i and $|S|$ is the number of states. Under the assumption that input variables are mutually independent, $p_{i,j}$ can be calculated directly from the STG by multiplying the input probabilities. The remaining problem is to compute P_i.

Given a STG with $|S|$ states, let **P** represent the *conditional transition probability* matrix of size of $|S| \times |S|$. The *static state probability* P_i of each state can be obtained by solving the following equations:

$$\mathbf{q}^T \mathrm{P} = \mathbf{q}^T \tag{4}$$

$$\sum_{i=1}^{|S|} P_i = 1 \tag{5}$$

where **q** is the *static state probability* vector whose components are the *static state probability* P_i of the state s_i (i.e., $\mathbf{q} = \left[P_1, P_2, \ldots, P_{|S|}\right]^T$). The sufficient condition that the *static state probability* vector of states exists is that an STG has a reset state. For those STGs without a reset state, cases also exist for which the *total transition probability* [8] can be obtained.

It should be noted that the value of the *total transition probability* and the *conditional transition probability* between two states is generally different. More information about Markov analysis of FSM can be found in [60].

4.2.2 Monte-Carlo-based simulation

A long specified input stream provides the most complete information about the inputs. In this case, collecting other FSM statistics becomes straightforward by simulating the state machine for a sufficient length of time. For example, it is possible to calculate the *total transition probability* as the number of state transitions during the whole simulation time divided by the number of time units (often the clock cycle). Formally, this method is described as Monte-Carlo-based simulation [45]. Stopping criteria (or convergence criteria), based on statistical techniques, are used to decide when the simulation should stop. For sequential circuits, which have feedback of state bits, the stopping criteria can be obtained by determining whether the probabilities of the state bits are stable or not. The Monte-Carlo-Based simulation flow chart for FSM is shown in Figure 12.

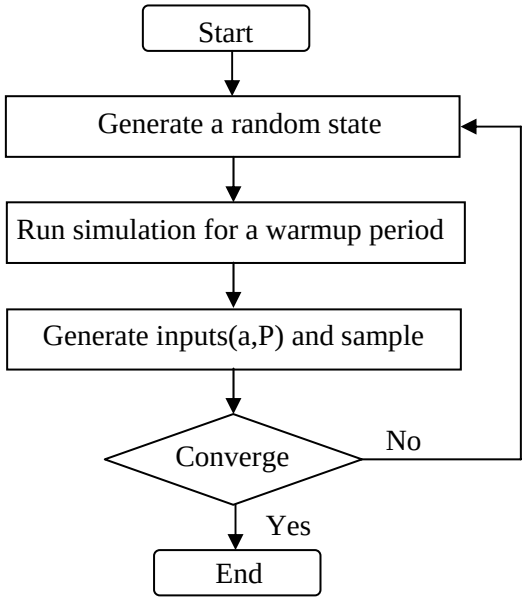


Figure 12. Monte-Carlo-based simulation flow chart for FSM

In this tool, a randomly generated input stream is used in the simulator. The probabilistic distribution of inputs can be specifed by the user and the default value is set to 0.5. It is assumed that the *static state probability* of each state becomes a constant as time increases to infinity, and after a warm-up period of clock cycles, the state probability of every state is sampled. A simplified convergence criterion is used, i.e., only when the maximum difference value between the probabilities of each state sampled in two consecutive time units (or clock cycles) is less than ε, a user specified constant, does the simulation stop. The default value of ε is set to 10^{-6}. For all the standard benchmarks [46] tested, their simulations converged in a reasonable time.

From the simulator, FSM information such as the *static state probability*, *total transition probability*, signal probabilities of primary outputs etc. are collected for further use.

4.3 FSM PARTITIONING

In VLSI design, the initial interest in partitioning arises from min-cut placement [47]. As the complexity of circuits increases and the desired number of transistors is above that which a chip or module can accommodate, the circuit must then be divided into components. Because the load of driving an external net in another component is significantly bigger than that of driving an internal net, partitioning techniques are needed to reduce the interconnection between components. By dividing a complex system into smaller, more manageable components, partitioning proves effective in reducing the design complexity and emerges in many phases of circuit design. Although here the focus is only on FSM partitioning, the proposed partitioning algorithm may also be useful for addressing the general partitioning problem.

As mentioned in section 2.2.2, at the RT level, FSM partitioning is an important technique for dynamic power management. After partitioning, the original FSM is decomposed into several smaller sub-FSMs. Apart from the case involving a state transition between two sub-FSMs, only one sub-FSM is active and thus all others are idle and can be deactivated without consuming power. Because each sub-FSM is smaller than the original one, sub-FSMs as a whole contribute to a lower average power. Depending on the quality of partitioning algorithms, this FSM power reduction can be significantly different.

An efficient FSM partitioning algorithm can select a “good” partition within a reasonable running time. The measure of “good” is performed by the cost function and states having high *total transition probability* between them are placed in the same sub-FSM in a “good” partition. Because the number of possible partitioning solutions is generally too large to explore, heuristic partitioning algorithms are used for reducing the complexity. Two main categories of partitioning algorithms are discussed here, namely, the iterative-based algorithm and the clustering algorithm. The partitioning algorithm proposed in this thesis is then discussed.

From an initial feasible solution, iterative-based algorithms iteratively move to a better solution according to the cost metric. Among these algorithms, one of the best known is the Kernighan-Lin algorithm (K-L) [48]. Its partitioning process is illustrated in Figure 13.

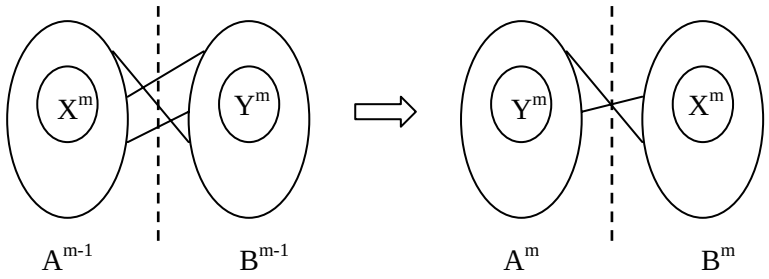


Figure 13. Interchange of subsets in KL algorithm

It is assumed that there are *2n* nodes, and two initial equal partitions, each with n nodes, are formed. These are referred to as A^0, B^0. Then in each iterative step, pairs of nodes are chosen to swap between the two partitions to reduce the interconnection. For instance, in the iterative step *m*, subsets X^m from partition A^{m-1} and Y^m from partition B^{m-1} will swap their positions to achieve a minimal cut cost.

This algorithm can produce good results for small amounts of CPU time. It can also be used as the basis for solving general n-way partitioning problems. Its employment in the FSM partitioning for low power can refer to [20], where two-way unbalanced K-L partitioning is used to minimize the *total transition probabilities* between two state sub-sets.

Another widely applied iterative-based partitioning algorithm is the genetic algorithm [49]. The motivation behind its use is Darwin's theory of natural selection in evolution where "superior" groups of a species produce more offspring in successive generation than "inferior" members. Its successful utilization in FSM low power partitioning can be found in [7]. However, the algorithm sometimes faces the problem of long running time.

Hierarchical clustering algorithms [50] consider sets of objects and they group them according to given measures of closeness. For a specific problem, closeness is defined by the corresponding cost function, representing the possibility of clustering objects. For example, in the FSM partitioning problem for low power, the *total transition probability* between states is used as the closeness criterion. Two states having mutually high transition probability are called "close" and they are more likely to belong to the same sub-FSM. Algorithms for hierarchical clustering can be further divided into two classes, both of which are shown in the clustering tree of Figure 14, using arrows to represent different directions.

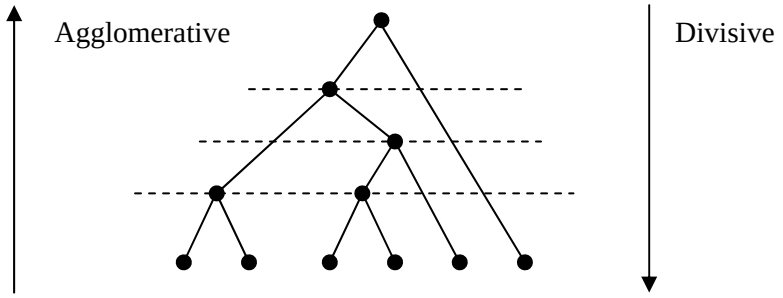


Figure 14. Hierachical clustering tree

The first class of clustering is in an agglomerative way (bottom-up). In the initial clustering solution, each object is itself taken as a single cluster. The algorithm continues by grouping two objects (single object or the object merged from single objects) and stops when all the objects are included in a single cluster. The second class of clustering is in a divisive way (top-down). It can be thought of as the reverse process of the agglomerative way. All objects are in a single cluster at the beginning and objects with the worst closeness are split in the subsequent steps.

Actually, clustering itself is rarely the goal. However, hierarchical trees provide a means of organizing objects at different levels of granularity. If the tree is cut at a particular level, clusters with corresponding granularity can be extracted. A cut-line closer to the leaves of the tree generates more clusters and the states in each cluster are closer. A cut-line closer to the root generates fewer clusters and the states in each cluster are more distant. Iterative-based algorithms, meanwhile, are more effective for a smaller solution space with greater density [51]. Hence, if the clustering algorithm is used initially followed by the iterative-based algorithm on the clusters obtained, better partitioning solutions can be expected, compared to those where only iterative-based algorithms are used. On this basis, in the FSM partitioning algorithm proposed here, a hierarchical tree integrating an iterative-based algorithm is built firstly for further algorithm optimization.

The partitioning criterion in this case is to obtain a small cluster of states which are active most of the time. Meanwhile, the probability of the state transitions within a cluster should be high and the probability of the state transitions between two clusters (two sub-FSMs) should be low. A two-phase partitioning algorithm is employed. In the first phase, by recursively applying the K-L two-way partitioning, a hierarchical binary tree is built as shown in Figure 15. Depending on their state transition probabilities, states are divided into groups in order to minimize the inter-transitions between two groups. The complexity of this algorithm is $O(n^2 \log n)$. For the benefit of the second phase, the tree is built in such a way that the states in the left hand cluster are more likely to be active. The left-most cluster for each level therefore has the highest probability of being active. In the second phase, an efficient algorithm is proposed that groups the clusters on

every level of the binary tree and generates a limited number of partitioning candidates. For *n* states, this algorithm finds the candidates ranging from 1-way to n-way partitioning with a complexity of only $O(n\log^3 n)$. Further explanation about the algorithm can be found in [52].

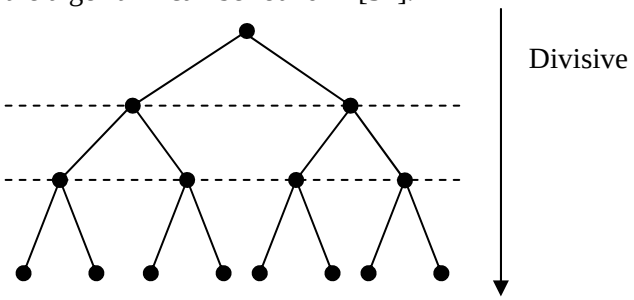


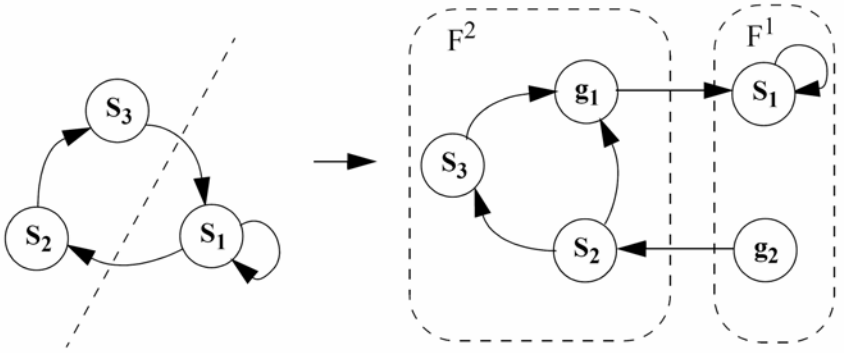
Figure 15. Bi-Partitiong hirarchical tree

4.4 FSM SYNTHESIZER

In this stage, every partitioning candidate obtained from the partitioning algorithm is synthesized into a network of sub-FSMs. In the first instance, the original STG is partitioned and transformed to support the interaction between sub-FSMs. Then state codes for low power are assigned to each state. Finally, the structure of the sub-FSM network is determined. The gate level implementation of the combinational logic for each sub-FSM is still unknown. But, for the asynchronous logic, its gate level implementation is decided in the synthesizer to prevent glitches from the synchronous part of the decomposed FSM resulting in hazards to the asynchronous part.

4.4.1 STG Transformation

To illustrate the procedure of STG transformation, the FSM in Figure 16a) is divided into two sub-FSMs F^1 and F^2, with state subsets $S^1 = \{s_1\}$ in F^1 and $S^2 = \{s_2, s_3\}$ in F^2. There are two *crossing transitions* between F^1 and F^2. A *crossing transition* is the state transition whose source state and destination state reside in different sub-FSMs. In order to be able to detect a *crossing transition*, an extra *g-state* is introduced. A *g-state* is inside the sub-FSM which contains the source state of a *crossing transition*, but it has the same index as that of the destination state. After the STG transformation, two new state subsets are formed which are $U^1 = \{s_1, g_2\}$ in F^1 and $U^2 = \{s_2, s_3, g_1\}$ in F^2. The transformed STG is shown in Figure 16b).



a) STG before transformation b) STG after transformation

Figure 16. STG before and after transformation

The behaviour of a *crossing transition* changes after the introduction of a *g-state*. The *crossing transition* s_3 in F^2 to s_1 in F^1 can be taken as an example. After introducing g_1 in F^2, the original transition is transformed via the following sequence of events:

1) A synchronous state transition in the *local state memory*, from the source state of the *crossing transition* to the *g-state*, denoted as $s_3 \rightarrow g_1$.

2) An asynchronous state transition in the *global state memory*, from the *g-state* to the original destination state, denoted as $g_1 \rightarrow s_1$. Both these transition states have the same index.

The entire *crossing transition* is completed within one clock cycle. The first event is synchronous because the *local state memory* is updated to the *g-state* at the active edge of the clock signal. s_3 and g_1 should be distinguished from the local state code when sharing the same global state.

The second event is asynchronous because the *global state memory* is updated immediately upon detection of the transition in the *g-state*. The *local state memory* is only triggered by the clock signal and therefore remains unchanged. In this example, g_1 and s_1 share the same local state code whereas their global states are different. The global state is then used to deactivate the currently active sub-FSM F^2 and activate the sub-FSM F^1 as the destination state of the crossing transition s_1, is to be found here.

Coupled states are used to indicate the g-state and its corresponding state which both share the same local state code. In Figure 16b), two *coupled states* (s_1, g_1) and (s_2, g_2) are then obtained. A formalized description concerning STG transformation can be found in [53].

4.4.2 State assignment for decomposed FSM

When synthesizing a network of sub-FSMs, state encoding is strongly related to the structure in which the sub-FSMs are implemented. In other words, whether or not the sub-FSMs share the same state memory will greatly influence the state assignment strategy.

In [7], after partitioning, each sub-FSM has its own *local state memory*. Because there is no direct *crossing transition* connecting the states in two different sub-FSMs in the sub-FSM network, it is possible to synthesize each sub-FSM separately. This state assignment problem can be considered to be the same as for the monolithic FSM.

In contrast to the separate state memory method, Chow et al. [21] propose a decomposition model with shared *local state memory*. Additional state bits are added to decide which one of the sub-FSMs is active. For state encoding, they present a method that considers *crossing transitions* by introducing pseudo-outputs. A pseudo-output bit represents a fanout-oriented relation imposed by the *crossing transitions.* For example, if there are *crossing transitions* that have the source states in the same sub-FSM and toward the same destination state, these source states should be assigned "close" state codes (in terms of Hamming distance). Transitions (rows in the state transition table) whose current states are these source states should have a pseudo-output of "1" added when all other transitions are given a pseudo code of "0". Subsequently, all *crossing transitions* are deleted and Jedi [54] is used to perform low power state assignment for each individual sub-FSM.

Both approaches described above assume fully synchronous implementations. Based on the decomposed FSM structure with mixed synchronous/asynchronous state memory, a state assignment procedure called state-bundling is proposed to address the low power state encoding problem.

As mentioned in section 4.4.1, after the STG transformation, a group of *coupled states* is formed. The proposed state assignment begins from these *coupled states* because they are related to c*rossing transitions* and should be assigned the same local state code. The whole procedure can be described in the following steps through the example in Figure 17.

1) A *state bundle* table is built (see Table 1). Each row represents the states inside the same sub-FSM. Each column, named after a *state bundle,* includes states residing in different sub-FSMs with the same local state code. Binary encoding in incremental order is assigned to columns from left to right. The *state bundle* including g-states can be further referred to as the *g state bundle*.

2) The *coupled states* are put into the table, as shown in Table 1.

Duty time: $T_1>T_4>T_3>T_2>T_5$

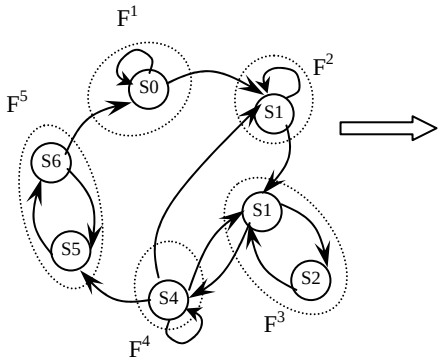


Table 1. Initial state bundle table

B:	**b0 000**	**b1 001**	**b2 010**	**b3 011**	**b4 100**
F^1	s_0	g_1	-	-	-
F^2	-	s_1	g_3	-	-
F^3	-	-	s_3	g_4	-
F^4	-	g_1	g_3	s_4	g_5
F^5	g_0	-	-	-	s_5

Figure 17. STG example

3) Rows in the table are sorted according to the duty time of their corresponding sub-FSMs (see Table 2). The duty time T_i represents the probability of the sub-FSM F^i being active. Its value is decided by the sum of the *static state probability* of states in the sub-FSM, that is, $T_i = \sum prob(s_i), s_i \in S^i$ where S^i is the state subset of F^i. The duty time of the sub-FSMs top down is in descending order after rearrangement.

Table 2. Sorted state bundle table

B:	**b0 000**	**b1 001**	**b2 010**	**b3 011**	**b4 100**
F^1	s_0	g_1	-	-	-
F^4	-	g_1	g_3	s_4	g_5
F^3	-	-	s_3	g_4	-
F^2	-	s_1	g_3	-	-
F^5	g_0	-		-	s_5

Table 3. After merging coupled states

B:	**b0 000**	**b1 001**	**b2 010**	**b3 011**	**b4 100**
F^1	s_0	g_1	-	-	
F^4	s_4	g_1	g_3	g_5	
F^3	g_4	-	s_3	-	
F^2	-	s_1	g_3	-	
F^5	g_0	-	-	s_5	

4) The *coupled states* are merged and two or more of them may possibly reside in the same column (see Table 3). The reason for merging is to reduce local state bits, which often results in lower power in the final implementation. The proposed algorithm ensures that the sum of the state probabilities of states in the first column is a maximum.

Table 4. g state bundle optimization

B:	**b0 000**	**b1 001**	**b3 010**	**b2 011**
F^1	s_0	g_1	-	-
F^4	s_4	g_1	g_5	g_3
F^3	g_4	-	-	s_3
F^2	-	s_1	-	g_3
F^5	g_0	-	s_5	-

Table 5. Final state bundle table

B:	**b0 000**	**b1 001**	**b3 010**	**b2 011**
F^1	s_0	g_1	-	-
F^4	s_4	g_1	g_5	g_3
F^3	g_4	s_2	-	s_3
F^2	-	s_1	-	g_3
F^5	g_0	s_6	s_5	-

5) Each *coupled states* is taken as a whole and its position is optimized by a greedy algorithm. If the states in two different *coupled states* have a high mutual state transition possibility, they are assigned the state codes for the least Hamming distance (see Table 4).

6) Other states, not included in the *coupled states*, are finally put into the table in a greedy way. The Hamming distance of states with high transition probability is minimized (see Table 5).

More details about the computational efficient state-encoding algorithm for low power can be found in Paper III.

4.4.3 FSM decomposition structure

Suppose the monolithic FSM has *I* as its input, *O* as its output and is partitioned into sub-FSMs F^1, F^2, ..., F^n. The original state subsets S^1, S^2, ..., S^n, in combination with the introduced *g-states*, form the new state subsets U^1, U^2, ...,U^n in F^1, F^2, ..., F^n, respectively. All sub-FSMs share the same *local state memory* but have their own combinational logic. The general structure of the proposed decomposed FSM model is shown in Figure 18.

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).

The GSM decides which is the current active sub-FSM. It is implemented as an asynchronous finite state machine. A Muller-C element [61] is used as the basic asynchronous element. A state transition in the GSM only takes place at the event of a *crossing transition*, in which case, a *g-state* will be detected. In a "well-partitioned" FSM, the probability of *crossing transitions* is very low. Therefore, the GSM will be idle for most of the time and dissipate no dynamic power due to the inherent property of an asynchronous circuit. The state information in the GSM is directly used as the control signals to both the LSM and the combinational part (implementing the next state and primary output function) of the sub-FSMs (labeled F^1, F^2, ..., F^n in Figure 18). The state bits in the LSM can be selectively turned off via the clock gating logic controlled by the GSM.

At any given time, apart from *crossing transition events*, only one sub-FSM is active. The active sub-FSM is responsible for determining the primary output and the next local state. When deactivated, the inputs to the sub-FSM are disabled by AND gates and no dynamic power will be dissipated. The outputs of a deactivated sub-FSM are all blocked to zero. By using OR gates, the correct primary outputs and next state outputs can be obtained by collecting output information from all sub-FSMs.

The structural information will be used in the power estimation function in the next section.

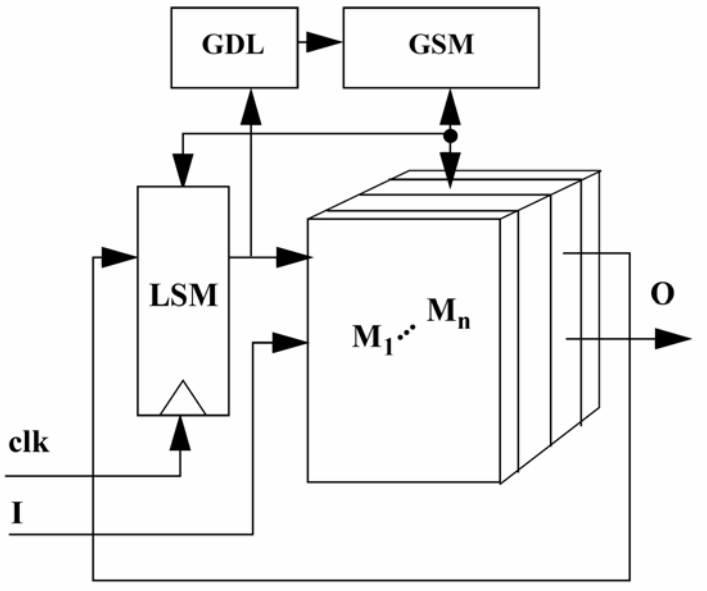


Figure 18. Decomposed FSM structure with mixed synchronous/asynchronous state memory

4.5 POWER ESTIMATION

Power estimation is an indispensable component in the design process. In the early design steps, power estimation can help to avoid power violation of the design constraints. In synthesis practices for each level, power estimation can be combined into cost metrics to explore the design space between power, area and speed and assist in choosing the design methodology most suitable for a given circuit.

The tradeoff in power estimation is between accuracy and running-time. At lower levels, more information can be obtained and more accurate power estimation can be made. However their corresponding computational costs are higher. SPICE, the circuit level simulator for power estimation, provides accurate power information. However, it cannot be used for a circuit that includes more than

thousands of transistors due to the computational expense. Therefore, the utilization of SPICE is limited to power analysis of the basic cell module.

At the gate level, when the design layout has been determined, power analysis is based on the signal model to calculate the switching activity of internal nodes of the circuit. Both probabilistic and statistical techniques can be exploited [12].

At higher levels, the final hardware implementation is uncertain and precise information is absent, so it is more difficult to estimate the power. Only the RT level power estimation is focussed on in the rest of this section.

At the RT level, [55] used the statistical modelling method for DSP circuits, where the basic models can be built as adders, comparators, registers etc. Power estimation is performed by combining the power coefficients of each model with the statistics of the circuit activity. The former are stored in the library database and the latter are derived from the simulation of specified input patterns.

Another more general power estimation method is based on information theory, using entropy as a measure of circuit average switching activity [13]. It is known that the average power dissipation is proportional to the effective capacitance (see equation (2)), which can further be expressed as the average switching activity multiplied by the whole circuit capacitance. Nemani et al. [13], after reasonable approximation, concluded that:

$$P_{avg} \propto A \times H \tag{6}$$

where A is an estimate of the circuit area, representative of the whole capacitance of the circuit. H is the average value of entropy $H(i)$ over all nodes i in the circuit and represents the node average switching activity. After a series of deduction, H can be finally expressed as:

$$H \approx \frac{2/3}{n+m}(H_i + 2H_o) \tag{7}$$

where H_i is the sum of node entropy of the inputs to the combinational logic. H_o is the sum of node entropy of the outputs to the combinational logic. n is the total number of inputs and m is the number of outputs.

To calculate the value of A, an area estimation model is proposed in [14]. Firstly a multi-output Boolean function is transformed to an equivalent single output function and then the associated complexity measure is computed.

In the proposed tool, the power estimation function including structural information (see power estimator in Figure 10) is employed to every synthesized partitioning candidate. The best candidate with the lowest power will be chosen as the input for the code generator. The purpose is to find a candidate with the actual lowest power and which is also the candidate with the lowest estimated power. Therefore, the absolute difference between the actual power and the estimated power is not important.

In this decomposed FSM model, the power of the combinational logic of each sub-FSM is estimated using this entropy-based method. The area A associated with each sub-FSM is computed directly from the state transition table for the sake of simplicity. As for the other parts, their power estimations are straightforward since their gate level implementations are known.

Power estimation for combinational logic:

As mentioned, the power estimation of the combinational logic is made via the state transition table, together with entropy. It can be represented as:

$$P_{comb} = \sum_{i=1}^{n} H_i \times Row_i \times k_{tech} \times T_i \qquad (8)$$

where H_i is the entropy of the combinational logic implementing sub-FSM F^i. Row_i is the number of rows in the state transition table with source states in F^i. k_{tech} is an empirically determined constant to adjust to the cell library used. T_i is the duty period of F^i .

Power for global state memory:

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

$$P_{GSM} = (k_B \times p_{LSM\_B} + k_G \times p_G + k_g \times |g|) \qquad \text{a)}$$

$$+ \sum_{i=1}^{n} P_C \times T_i \qquad \text{b)} \qquad (9)$$

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 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 b) represents the contribution from the global state memory devices, implemented as muller-C elements where T_i is the probability of global state transition, i.e., the probability of a *crossing transition* between different sub-FSMs. The number of sub-FSMs is denoted by *n*. The constants are determined empirically based on a single FSM partitioning run.

Power for D type flip flop:

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

$$P_{LSM} = \sum_{i=1}^{m} P_{Dff_i} \times T_i \quad (10)$$

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:

$$P_{clock} = |FF| \times C_{clkin} \times f \times V_{dd}^{\ 2} \times k_{buffer} \times k_{wire} \quad (11)$$

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:

$$P_{overhead} = P_{gatedCom} + P_{gatedDff} \quad (12)$$

where $P_{gatedCom}$ includes the power of AND gates to activate and deactivate the combinational logic, as well as the power of OR gates for merging the outputs. $P_{gatedDff}$ is the power to activate and deactivate 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:

$$P_{whole} = P_{comb} + P_{GSM} + P_{LSM} + P_{clock} + P_{overhead} \quad (13)$$

The verification of the cost function can be found in [52].

4.6 RT LEVEL CODE GENERATOR

For the best partitioning candidate with the estimated lowest power, the proposed automatic synthesis tool outputs RT level VHDL code and synthesis scripts, both of which can be used directly as inputs for a standard logic synthesis tool for gate level optimization .

The automatically generated VHDL file includes the detailed implementation information of the decomposed FSM. For the state memory and overhead circuit, the gate level implementation is defined in the file based on cell library dependent information. In the main it is the combinational logic of sub-FSMs which can be further optimised at gate level. The synthesis scripts provide the constraints and instruction for the logic optimization.

Both the gate level area and power estimation are performed using *Power Compiler* (Synopsys), assuming a supply voltage of 1.8V and a clock frequency of 20MHz. The area estimation is based on the cell area and the target technoglogy is 0.18μm CMOS circuit. The overall power dissipation at gate level is obtained by

dividing the decomposed FSM into several parts, synthesizing each part separately and then adding their power dissipations together. The combinational logic of each sub-FSM, for example, is synthesized separately and has its own power dissipation report. Because the logic syntheis tool only supports the synchronous design, for the asynchronous element (muller-C in this case), its gate level power estimation is assumed to be the same as the value computed by the cost function b) in equation (9).

The effectiveness of this tool is verified via a series of benchmarks. For all of them, the power consumption in the decomposed FSM is significantly reduced by an average of 56% compared to the original unpartitioned FSM.

5 SUMMARY OF PUBLICATIONS

The three papers included in the thesis can be categorized in two groups:
1. Initial concept and mathematical formulation.
2. Developed automatic synthesis tool refinement.

Section 5.1 and 5.2 outlines the content of every paper and Section 5.3 presents the contribution of each author to the papers.

5.1 INITIAL CONCEPT AND MATHEMATICAL FORMULATION

5.1.1 Paper I

In this paper a design model based on mixed synchronous/asynchronous state memory is proposed 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.

5.2 DEVELOPED AUTOMATIC SYNTHESIS TOOL REFINEMENT

Two papers cover issues related to procedural refinement inside the developed CAD tool for synthesis of low-power partitioned FSMs. They focus on FSM partitioning method and state encoding optimization individually.

5.2.1 Paper II

This paper presents FSM partitioning algorithms and RT-level power estimation functions that are the key issues in the tool. The proposed n-way partitioning algorithm with low complexity may also be used for general synchronous partitioning method. The accuracy of the power estimation functions is verified by standard benchmarks.

5.2.2 Paper III

This paper presents state encoding techniques for a partitioned FSM structure based on mixed synchronous/asynchronous state memory. The state memory is composed of synchronous local state memory and asynchronous global state memory. One hot encoding is used inside asynchronous global state memory for low complexity and low power. For the local state assignment, a procedure named as state-bundling is presented to enable states residing in different sub-FSMs to share the same state codes. Two state-encoding techniques, one based on binary encoding and one optimized for low-power consumption, are compared.

5.3 AUTHOR'S CONTRIBUTIONS

The contribution of the author is essential to all the papers presented in the thesis. The exact contribution of each author is specified in Table 6.

Table 6. Author's contribution (M = main contributor, C = co-author).

Paper #	CC[1]	MO[2]	BO[3]	Contributions
I	M		C	CC : Implemented the automatic synthesis tool of FSM decomposition model BO: Outlined the concept of mixed synchronous/asynchronous state memory and Supervisor
II	M	C	C	CC: Developed the tool and specified the power estimation function MO: proposed the "candidate generation" algorithm BO: Supervisor
III	M		C	CC: Proposed the state encoding optimization method BO: Supervisor
1. Cao Cao 2. Mattias O'Nils 3. Bengt Oelmann				

6 THESIS SUMMARY

This thesis proposed the concept and implementation structure of FSM decomposition based on mixed synchronous/asynchronous state memory. Key issues in the design of the CAD tools for the synthesis of low power decomposed-FSMs are also discussed.

A general introduction to the research area, the motivation, and the specific problem description of the thesis, are given in Section 1. Section 2 gives an introduction to the related research work. Section 3 introduced the special mixed synchronous/asynchronous design architecture. Section 4 presented the tool developed for automatic synthesis. Section 5 summarizes the three papers covered by the thesis and identifies the original contribution for each paper.

In this chapter, section 6.1 summarizes the conclusions reached during the research work on this thesis. Suggestions for future work are presented at Section 6.2.

6.1 CONCLUSIONS

6.1.1 Design model of mixed synchronous/asynchronous state memory

A novel design model for partitioned FSMs based on mixed synchronous/asynchronous state memory is proposed. 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, which has a low probability of being updated. In this way, the global state memory adds very low power-overhead. In spite of the internal asynchronous operation, the input/output behaviour of the decomposed FSM is equivalent to the original synchronous one.

6.1.2 Design flow of the automatic synthesis tool

The developed CAD tool fits into a standard-cell based design flow. It takes an STG as input, transforms it and generates synthesizable RT-level VHDL code that is fed to a standard logic synthesis tool. The effectiveness of the tool was demonstrated by benchmarks [Paper II] with an average power reduction of 56%. The best result was a power reduction in excess of 70%.

6.1.3 FSM partitioning algorithm and RT level power estimation function

A novel multi-way partitioning algorithm for partitioned FSM synthesis is proposed in the developed CAD tool. It was applied to a mixed synchronous/asynchronous architecture but can also be used for fully synchronous implementations. The proposed algorithms are of low complexity which is important when it comes to the practical usage of the tool. Among the partitioning candidates obtained from the algorithm, RT-level power estimation functions are

proposed with efficient accuracy for selecting the candidate with the lowest power consumption.

6.1.4 State encoding optimization

A state encoding algorithm for the partitioned FSM composed of inter-connected sub-FSMs with shared state memory was proposed. The algorithm takes the properties of partitioned FSMs and the constraints imposed by the implementation architecture into account. Further power reductions can be achieved for certain benchmarks after state assignment optimization. However, it is not possible to benefit much from the state encoding optimization compared to the method of FSM partitioning. Asynchronous state memory, idle condition detection logic, and the shut-down logic have already been established before state encoding and their power can not be reduced. The sub-FSM with the high possibility of being active often has few state bits, which is also unlikely to be optimized.

6.2 FUTURE WORK

To further explore the concept of finite-state machines based on mixed synchronous/asynchronous state memory, more detailed studies on the techniques presented in thesis have to be conducted as well as expanding the studies to issues not covered in this thesis. The following issues are to be addressed in future work:

- Formalized description of the asynchronous state memory
  In [Paper I] the asynchronous state memory bit for two-way partitioned FSMs is proposed and is actually an SR-latch. In [Paper II] the asynchronous state memory bit for an N-way partitioned FSM is a Muller-C element. Even though it has been demonstrated that these asynchronous state memory bits work very well in simulations, it has proven necessary to provide a formal description of their behaviour in order to allow synthesis for applications other than the type of FSMs discussed in this thesis. One such application could be locally clocked FSMs with datapath (FSMDs) using asynchronous interaction.

- Timing analysis of the asynchronous memory
  The delay penalty for the partitioned FSMs has not been addressed in this thesis. A partitioned FSM composed of two or more sub-FSMs has a critical timing path that in most cases is smaller in comparison to the monolithic FSM. However, the delay in the asynchronous state-memory is added to the delay in the next-state function of the sub-FSMs for the crossing transitions. How the partitioning affects the critical timing path is not known at present and requires further investigation.

- Power-area tradeoffs
  For all synthesis results reported in this thesis, the optimizations are not constrained by area. It might prove possible to achieve power reductions close to those reported but with much lower area-overhead. Due to the

large difference in the complexity of the asynchronous state memory between a two-way partitioned FSM and an FSM with more than two partitions, large savings in area and possibly faster circuits can be obtained by two-way partitioning.

- Benchmarking of power in FSMs
  Evaluating RT-level synthesis tools is quite difficult since the optimization results are highly dependent on the benchmark circuits used. For the evaluation of the power optimizations, the input data probabilities are of great importance. Even though standard MCNC benchmark circuits were used for all the developing tools for FSM optimizations, sufficient results to ensure fair comparisons to others reported in the literature were not found. For these benchmarks, no "typical" input patterns or input probabilities are specified which makes it almost impossible to compare results from two different tools. It is also difficult to make any general conclusion for an optimization tool. For example, it is not possible to say that this given tool gives good results for FSMs with certain characteristics but cannot handle FSMs with other characteristics. This is a common problem for all developing tools for power optimizations of FSMs and is not specific to this work. A first attempt to address this problem was presented in [64] by having an FSM benchmark generator tool. This tool could generate synthetic FSMs with characteristics specified by the user. Unfortunately this tool does not allow specification of state transition probabilities, which makes it of no interest for researchers developing power optimization tools. The belief is that a similar benchmark generator including specification of the characteristics for the state transition probabilities would be of great interest for those developing FSM power optimization tools.

- Formal verification of FSM transformations
  The functional equivalence of a transformed FSM and the original FSM description has been verified through extensive simulations. A formal verification would provide a more rigorous proof of functional equivalence.

7 REFERENCES

[1] M. C. Johnson, D. Somasekhar, and K. Roy, "Models and algorithmsfor bounds on leakage in CMOS circuits," *IEEE Transactions on Computer-Aided Design*, vol. 18, pp. 714–725, June 1999.

[2] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-Power CMOS Digital Design," *IEEE J. Solid-State Circuits*， vol. **27**, No. 4, pp. 473–484, April 1992.

[3] J. Henkel, "A Low-Power Hardware/Software Partitioning Approach for Core-based Embedded Systems," *Design Automation Conference*, pp. 122-127, June 1999.

[4] F. Pollack, "New microarchitecture challenges in the coming generations of CMOS process technologies", *32nd Annual International Symposium on Microarchitecture*, Nov.1999.

[5] L. Benini and G. De Micheli, "Dynamic Power Management design techniques and CAD tools", *Kluwer*, 1998.

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

[7] L. Benini, F. Vermeulen, G. De Micheli, "Finite-State Machine Partitioning for Low-Power," *Proceedings of the IEEE International Symposium on Circuits and Systems* 1998, vol. 2, pp. 5-8.

[8] L. Benini and G. De Micheli, "State Assignment for Low Power Dissipation," *IEEE Journal for Solid-State Circuits*, vol.30, No.3, March 95, pp.32-40.

[9] S. Mutch, S. Shigematsu, Y. Matsuya, H. Fukuda, and J. Yamada, "A 1V Multi-Threshold Voltage CMOS DSP with an Efficient Power Management Technique for Mobile Phone Applications," *Proceedings of the International Solid-State Circuits Conference*, 1996, pp. 168–169.

[10] R. Hossain, M. Zheng, and A. Albicki, "Reducing power dissipation in CMOS circuits by signal probability based transistor reordering," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, No. 3, Mar, 1996.

[11] W.Nebel and J. Mermets (Eds.), "Low power design in deep submicron electronics," *Kluwer*, 1997.

[12] K. ROY, S. Prasad, "Low Power Digital CMOS Design," *Kluwer*, 1995.

[13] M. Nemani, F. Najm, "Towards a high-level power estimation capability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, No. 6, pp. 588-598, 1996.

[14] M. Nemani, F. Najm, "High-level area and power estimation for VLSI circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18 , No. 6 , pp. 697 – 713, June 1999.

[15] N. Kumar, S. Katkoori, L. Rader and R. Vemuri, "Profile-Driven Behavioral Synthesis for Low Power VLSI Systems," *IEEE Design & Test of Computers, Fall Issue*, 1995, pp.70-84.

[16] V. Tiwari, S. Malik, A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2 , No. 4 , pp. 437 – 445, Dec. 1994.

[17] D. Lidsky, J. Rabaey, "Early Power Exploration - A World Wide Web Application," *Proceedings of Design Automation Conference*, pp. 27 – 32, June 1996.

[18] J. Hartmanis, R. E. Stearns, "Algebraic structure theory of sequential machines," *Prentice Hall*, 1966

[19] M. Alidina, J. Monteiro, S. Devadas, A, Ghosh, M. Papaefthymiou, "Precomputation-based sequential logic optimization for low power," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, No. 4, pp. 426 – 436, Dec. 1994.

[20] J. Monteiro, A. Oliveira, "Finite State Machine Decomposition for Low Power," *Proceedings of the 35th Design Automation Conference*, pp. 758-763, June, 1998.

[21] S-H. Chow, Y-C. Ho, T. Hwang, "Low-Power Realization of Finite-State Machines - A Decomposition Approach," *ACM Transactions on Design Automation of Electronics Systems*, 1996, vol. 1, No. 3, pp. 315-340.

[22] 1999 ITRS Roadmap.

[23] V.Tiwari, S. Malik, P. Ashar, "Guarded evaluation: pushing power management to logic synthesis/design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, No. 10, pp. 1051 – 1060, Oct.1998.

[24] G. De Micheli, "Symbolic design of combinational and sequential logic circuits implemented by two-level logic macros," *IEEE Transactions on Computed-Aided design*, vol.CAD-5, pp.597-616, Oct.1986.

[25] S. Devadas, H. Ma, AR Newton and A. Sangiovanni-Vincentelli, "MUSTANG: state assignment of finite state machines targeting multilevel logic implementations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, No. 12, pp. 1290 – 1300, Dec. 1988.

[26] K. Roy and S. Prasad, "SYCLOP: Synthesis of CMOS Logic for Low Power Applications," *Proceedings of International Conference on Computer Design (ICCD)*, 1992, pp. 464–467.

[27] W. Nöth and R. Kolla, "Spanning Trees Based State Encoding for Low Power Dissipation." *Proceedings of the conference on Design Automation and Test in Europe*, 1999, pp. 168-174.

[28] I. Lemberski, M. Koegst, S. Cotofana, B. Juurlink, "FSM non-minimal state encoding for low power," *Proceedings of the 23rd International Conference on Microelectronics*, vol. 2, pp. 605 – 608, May 2002.

[29] P. Surti, L. F. Chao, A.Tyagi, "Low power FSM Design Using Huffman-style encoding," *Proceedings of European Design and Test Conference*, pp.521-525, Mar.1997.

[30] CY. Tsui, M. Pedram, AM. Despain, "Low power state assignment targeting two and multilevel logic implementations," *IEEE Transactions on Computer Aided Design*, vol.17, No. 12, pp. 1281-1291, Dec. 1998.

[31] E. OLSON, and S. KANG, "Low-power state assignment for finite state machines search," *International Workshop on Low power design*, 1994, pp. 63–68.

[32] A. Davis and S.M. Nowick, "Asynchronous circuit design: Motivation, background and methods," in book Asynchronous Digital Circuit Design, pp. 1-49, *Springer*, 1995

[33] D. Dobberpuhl et al., "A 200-MHz 64-b dual-issue CMOS microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 27, No. 11, pp. 1555-1567, Nov. 1992.

[34] K. van Berkel and M. Rem, "*VLSI programming of asynchronous circuits for low-power*", In book Asynchronous Digital Circuit Design, pp. 152-210, *Springer*, 1995

[35] S. B. Furber, D. A. Edwards, and J. D. Garside. "AMULET3: a 100 MIPS asynchronous embedded processor," *Proceedings of International Conference on Computer Design (ICCD)*, pp. 329-334, September 2000. (power)

[36] Ivan Sutherland and Jon Lexau. "Designing fast asynchronous circuits". In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 184-193. IEEE Computer Society Press, March 2001.

[37] T. Meng, R. Bordersen, D. Messerschmitt, "Asynchronous Design for Programmable Digital Signal Processors," *IEEE Transactions on Signal Processing*, pp. 939-952, April 1991.

[38] J. McCardle and D. Chester, "Measuring an Asynchronous Processor's Power and Noise," *Proceedings of Synopsys Users Group Conference* (SNUG 01), Synopsys, Mountain View, Calif., 2001, pp. 66-70.

[39] S. Moore, R. Anderson, R. Mullins, G. Taylor, J. Fournier, "Balanced Self-Checking Asynchronous Logic for Smart Card Applications" *Journal of the Microprocessors and Microsystems, Special Issue on Asynchronous System Design*, pp. 421-430, Oct. 2003

[40] K. Borum, T. Gleerup, J. Madsen, S. Pedersen, "Power-over-time estimation for processor design," *Proceedings of NORCHIP'01*, pp. 87-92, Nov. 2001.

[41] K. Y. Yun and R. P. Donohue, "Pausible clocking: a first step toward heterogeneous systems," *Proceedings of International Conference on Computer Design(ICCD)*, pp. 118 - 123, Oct. 1996.

[42] A. Hemani et al., “Lowering power consumption in clock by using globally asynchronous locally synchronous design style**,”** *Proceedings of the 36th Design Automation Conference*, pp. 873 – 878, June 1999.

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

[44] K. Trivedi, “Probability and Statistics with Reliability, Queueing, and Computer Science Applications,” *Prentice-Hall*, 1982

[45] R. Burch, F. N. Najm, P. Yang, T. N. Trick, “A Monte Carlo Approach for Power Estimation,” *IEEE Transactions on VLSI Systems*, vol. 1, No. 1, pp. 63-71, March 1993.

[46] S. Yang, “Logic synthesis and optimization benchmarks user guide,” *Technical report*, Stanford University, 1991.

[47] M. Breuer. “A class of min-cut placement algorithms,” *Proceedings of the 14th Design Automation Conference*, pp. 284-290, June 1977.

[48] B. W. Kernighan and S. Lin, “An efficient heuristic for partitioning graphs,” *Bell System Technical Journal*, vol. 49, No. 1, pp. 291 - 307, 1970.

[49] H. Inayoshi and B. Manderick, “The Weighted Graph Bi-Partitioning Problem: A Look at GA Performance,” In Parallel Problem Solving from Nature**,** pp. 617 - 625, *Springer-Verlag*,1992.

[50] S. C. Johnson, “Hierarchical Clustering Schemes” *Psychometrika 2*, pp. 241-254, 1967.

[51] C. J. Alpert and A. B. Kahng, “Recent Directions in Netlist Partitioning: A Survey,” *Integration: the VLSI Journal*, 19(1-2), 1995, pp. 1 - 81.

[52] C. Cao, M. O'Nils, B. Oelmann, “A Tool for Low-Power Synthesis of FSMs with Mixed Synchronous/Asynchronous State Memory,” *Proceedings of Norchip*, pp. 199-202, Nov. 2004.

[53] C. Cao and B. Oelmann, “Mixed Synchronous/Asynchronous State Memory for Low Power FSM Design,” *Proceedings of EUROMICRO Symposium on Digital System Design*, France, 2004, pp. 363-370.

[54] Bill Lin, A. Richard Newton, “Synthesis of Multiple Level Logic from Symbolic High-Level Description Languages,” *VLSI 89*, Munich, 1989, pp. 187- 196.

[55] P. Landman and J. Rabaey, “Black-Box Capacitance Models for Architectural Analysis,” *Proceedings of the International Workshop on Low Power Design*, pp. 165–170, April 1994.

[56] B. Oelmann, K. Tammemäe, M. Kruus, 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, 2001, pp. 167-186.

[57] J. Monteiro, “A Computer-Aided Design Methodology for Low Power Sequential Logic Circuits,” *PhD Thesis*, Massachusetts Institute of Technology, May 1996.

[58] E. Huwang, F. Vahid, Y.-C. Hsu, “FSMD functional partitioning for low power,” *Proceedings of Design, Automation and Test in Europe*, 1999, pp. 22–28.

[59] C. T. Hsieh and M. Pedram, “Architectural energy optimization by bus splitting,” *IEEE Transactions on Computer-Aided Design*, vol. 21, No. 4, pp. 408–414, April 2002.

[60] G. Hachtel, E. Macii, A. Pardo, and E Somenzi, “Markovian Analysis of Large Finite State Machines,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, No. 12, pp. 1479 - 1493, Dec 1996.

[61] David E. Muller and W. S. Bartky, “A theory of asynchronous circuits,” *Proceedings of an International Symposium on the Theory of Switching*, pp. 204 - 243, April 1959.

[62] D. Singh, J. Rabaey, M. Pedram, F. Catthoor, S. Rajgopal, N. Sehgal, T. Mozdzen, “Power-conscious CAD tools and methodologies: a perspective,” *Proceedings of the IEEE*, pp. 570 - 594, April 1995.

[63] D. M. Chapiro, “Globally-Asynchronous Locally-Synchronous Systems,” *PhD thesis*, Stanford University, Oct. 1984.

[64] L. Józwiak, D. Gawlowski, A. Slusarczyk, “An effective solution of benchmarking problem: FSM benchmark generator and its application to analysis of state assignment methods,” *Proceedings of EUROMICRO Symposium on Digital System Design*, France, 2004, pp. 160-167.

PAPER I

Mixed synchronous/asynchronous state memory for low power FSM design

P a p e r I

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.

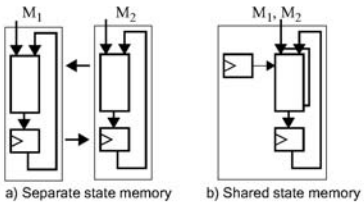


Figure 1. Structural decomposition of FSMs

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:

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

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.

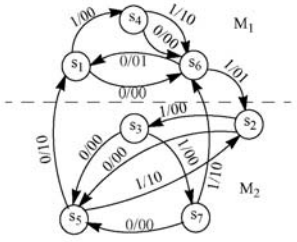


Figure 2. FSM example dk27

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.

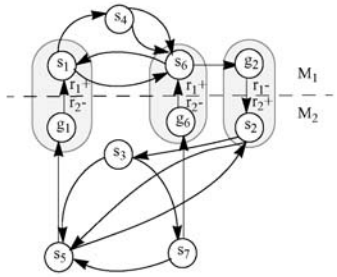


Figure 3. Transformed STG in decomposed FSM

Take the *crossing transition* $s_6 \rightarrow s_2$ as an example. After g_2 is introduced in M_1, the first event is the transition $s_6 \rightarrow g_2$, 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 $g_2 \rightarrow s_2$). 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 $g_2 \rightarrow s_2$ 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.

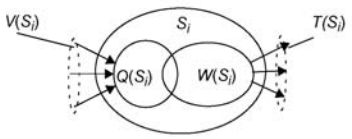


Figure 4. State sets associated with S_i

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

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

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

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

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_i 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 *state 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	$\lceil \log_2\lvert U_i\rvert\rceil$
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.

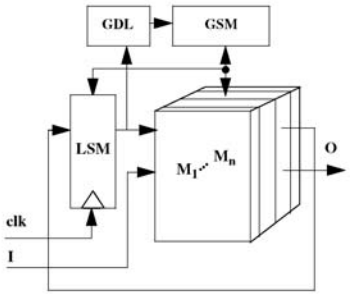


Figure 5. Structural model based on mixed synchronous/asynchronous state memory

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 com-

posed 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.

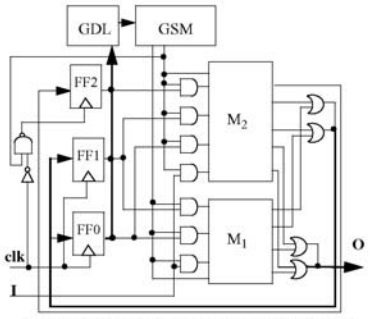


Figure 6. Circuit of a decomposed FSM(dk27)

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.

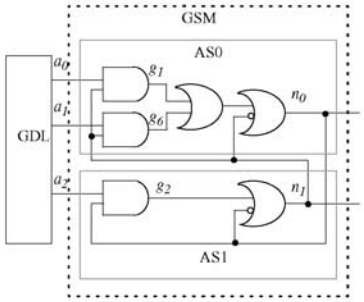


Figure 7. Global state memory structure in dk27

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	$\|S_1\|/\|S_2\|$	$\|U_1\|/\|U_2\|$	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," *ACM 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.

PAPER II

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

Paper II

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

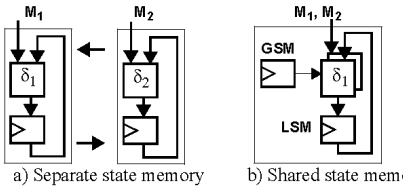
Power optimizations at the architectural level often involves some *Dynamic Power Management* (DPM) scheme that reduces the dynamic power consumption [1]. Whenever DPM is applied, the original design has to be partitioned into two or more units in such a way that they dynamically can be "shut down" when 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.



a) Separate state memory b) Shared state memory

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 gated-

clocks to shut down flip-flops temporarily not needed in the local state memory.

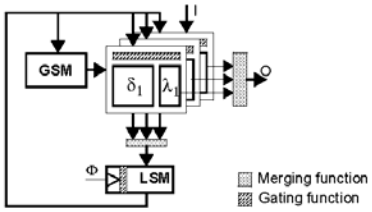


Figure 2. Mixed Synch/Asynch. FSM Architecture

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.

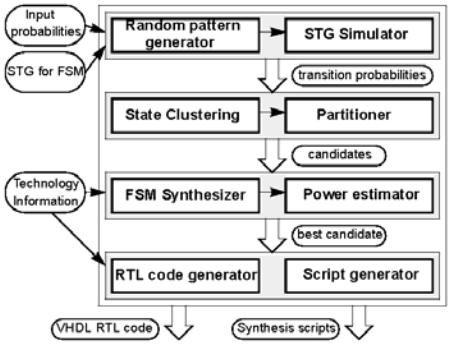


Figure 3. Overview of the tool

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 ∪ 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;
```

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.

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

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

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 verfication. 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*.

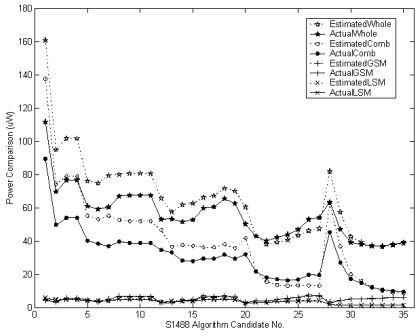


Figure 7. Cost function verification

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.

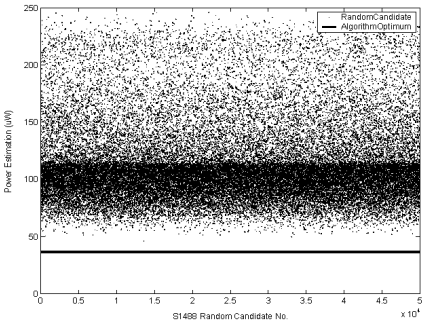


Figure 8. Algorithm verification

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.

PAPER III

State-Encoding for Partitioned FSMs with Mixed Synchronous/Asynchronous State Memory

Paper III

Low-Power State-Encoding for Partitioned FSMs with Mixed Synchronous/Asynchronous State Memory

Cao Cao and Bengt Oelmann

Abstract: Partitioned Finite-State Machine (FSM) architectures in general enable low-power implementations and it has been shown that for these architectures, state memory based on both synchronous and asynchronous storage elements gives lower power consumption compared to the fully synchronous ones. In this paper we present state-encoding techniques for a partitioned FSM architecture based on mixed synchronous/asynchronous state memory. The state memory here is composed of a synchronous local state memory and a global asynchronous state memory. The local state memory is shared by all sub-FSMs and uses synchronous storage elements. The global state memory is operating asynchronously and is responsible for handling the interaction between the different sub-FSMs. Even though the partitioned FSM contains asynchronous mechanisms, its input/output behavior is cycle by cycle equivalent to the original monolithic synchronous FSM. In this paper we study state encoding for partitioned FSMs that have been partitioned according to their state-transition probabilities. For the local state assignment we present a, what we call, state-bundling procedure to enable states residing in different sub-FSMs to share the same state codes. Two state-encoding techniques, one based on binary encoding and one optimized for low-power consumption, are compared.

1 Introduction

Dynamic Power Management (DPM) is a commonly used approach for low-power optimization on the Register-Transfer (RT) and architectural level [1]. The objective for a DPM scheme is to shut-down the parts of the design that are temporarily idle. By shutting down a part it is meant that the power dissipation is reduced for that part. For reduction of dynamic power dissipation, clock-gating and input-disabling is used. For reduction of leakage currents, such as subthrehold and diod leakage, various approaches have been proposed in the literature, e.g. [2]. Common for all these techniques is that mechanisms for detecting idle conditions of the different units are added to the design. Also means for shutting down the units are added. Implementation of these will result in additional circuits that will add to the circuit area and power dissipation. Before introducing shut-down circuits in the design, careful analysis must be made to achieve a solution with as low power consumption as possible. The objective for a power optimization procedure is to find the most beneficial idle conditions, taking the overhead into account. Complex designs, composed of several functional units, such as microprocessors that are composed of large functional units like floating-point unit and cache memory can be temporarily shut-down when not used. For a given architecture, this kind of coarse-grained DPM is possible to implement manually by the designer thanks to the small number of places shut-down circuits are introduced and to the fact that the different units are functionally well separated and therefore easy to identify. When applying DPM to a single functional unit, the unit has to be partitioned into two or more sub-units where each of them can be individually shut-down. The unit is decomposed in such a

way that the lowest possible power consumption is achieved. This type of fine-grained DPM requires an automated optimization procedure since the optimum decomposition is not necessarily made according to the functionality it is therefore not obvious to know how to make the decomposition. The most commonly used approaches for power optimization procedures, takes a behavioral description of the design and seeks the optimum, or near-optimum solution, for a pre-defined architecture. For data path units (combinational logic) the precomputation-based logic has been proposed [3]. The idea is to pre-compute a part of the function one clock cycle ahead in order to gate the clock signal to the register holding the inputs to the combinational logic and thereby reducing the average switching in the logic. Different architectures have been proposed that block either all inputs or a subset of the inputs [4]. This approach can also be used for synchronous FSMs. For low-power FSM design Benini et al. presented an approach called computational kernels [5]. From the State Transition Graph (STG) of the FSM, a sub-FSM is extracted that implements the function of the FSM for a subset of its states whose steady-state occupation probability is high. When the FSM is in one of these states a smaller and less power-consuming circuit is used (the kernel) and otherwise the original function is used. Chow et al [6] propose an implementation architecture that resembles of the one used for computational kernels. They propose a decomposition model for multiple coupled sub-FSMs. A shared state memory stores two sets of states, the original states and additional states that are used for determining which one of the different sub-FSMs that is active. For state-encoding they present a method that consider the crossing transitions (transitions where the source and destination state do not reside in the same sub-FSM) is used. In contrast to the shared state memory architecture, an

architecture with separate state memory, one for each sub-FSM, has been used by for example [7,8]. For state encoding and other optimizations, each sub-FSM can be separately optimized using standard methods. The disadvantage is that the circuit area for the state memory becomes larger compared to using shared state memory.

The approaches to low-power FSM design described above, all assume fully synchronous implementations. For both architectures, based on shared and separate state memory, fully synchronous implementations have disadvantages. For the cycle when a crossing transition occurs, the two sub-FSMs involved, both have to be clocked which is very power consuming. For partitioned FSMs with separate state memory an asynchronous hand-over mechanism has been proposed that removes the requirement of clocking two sub-FSMs at a crossing transition and thereby the power-overhead introduced for managing the interaction between the sub-FSMs can be reduced. In [9] it has been shown that asynchronous control for sub-FSM interaction is 5.8 times more power efficient when idle compared to synchronous control.

In [8] it was demonstrated that automated synthesis for low-power FSMs based on a mixed synchronous/asynchronous architecture with separate state memory achieved power reductions of 45% in average for a set of FSM benchmark circuits. For a recently presented decomposition model [10] for FSMs with shared state memory power, reductions of 56% in average. Here, state encoding optimizations for low power was not considered.

In this paper, a novel low-power state encoding algorithm for coupled FSMs is proposed and applied to partitioned FSMs based on mixed

synchronous/asynchronous state memory. The main contributions of this paper are the following:

- A state assignment procedure: State bundling enables crossing transitions in one single clock cycle (or in other words, only one sub-FSM has to be clocked).
- Power-optimized state encoding: A computational efficient state-encoding algorithm for coupled FSMs.
- Demonstration of efficiency: The algorithms presented have been implemented in a tool for low-power synthesis of partitioned FSMs and it is demonstrated that the state-encoding algorithm leads to power reductions of 6% in average for low-power partitioned FSMs originating from the MCNC benchmark circuits. The total average power reduction that is the result from both partitioning and state-encoding is 59%.

The outline for the rest of this paper is as follows: The next chapter introduces the partitioned FSM implementation architecture with a focus on the organization and operation of the mixed synchronous/asynchronous state memory. In chapter 3 the basic binary state encoding procedure and our procedure that we propose for power optimized state encoding are presented. In chapter 4 some experimental results from automatic synthesis of a set of FSM benchmark circuits show the possibility of reducing the power consumption in a partitioned FSM by using power-optimized state encoding after partitioning. In chapter 5 we conclude the paper by a discussion regarding the limitations of the two step approach with a partitioning step followed by a state encoding step.

2 Partitioned FSM with Mixed Synchronous/Asynchronous State Memory

2.1 Implementation Architecture

The straight-forward way to implement a partitioned FSM is to have separate state memory for each of the sub-FSMs, see Figure 1a. From state encoding point of view, state-encoding is made separately for each of them and well-established optimization algorithms can therefore be used. Since only one of the sub-FSM is active at a time, the state memory can be shared by all the sub-FSMs. The main advantage with shared state memory is the reduced area for the state memory, see Figure 1b. There is, however, a need for a global state memory determining which one of the sub-FSMs is for the moment active. For power-optimized state encoding, state-transition probabilities of the crossing transistions must be considered which is not the case for the separate state memory implementation. For a synchronous solution the global state memory needs to be clocked by the system clock signal that cannot be gated and will therefore increase the power consumption substantially, especially for a partitioned FSM composed of large number sub-FSMs. The architecture considered in this paper is a mixed synchronous/asynchronous architecture developed in [10] that has a shared local state memory (LSM) with a global asynchronous state memory (GSM). The basic idea is to have synchronous local state memory in the part always clocked and asynchronous memory for the global state memory. The partitioned FSM is made on the basis of the state transition probabilities which results in clustering of states with high probabilities that will be implemented in the same sub-FSM. The state-transition probabilities between the sub-FSMs will be of low probability and hence

the state-change probability is low for the global states which make an asynchronous implementation power efficient [12].

2.2 STG decomposition

In this section the decomposition of the STG of the monolithic FSM for the architecture described in the previous section is presented. To describe to basic ideas of the design model for STG decomposition, the example in Figure 2 will serve as an illustration.

The initial monolithic machine is decomposed into two separate sub-FSMs F^1 and F^2 as indicated in Figure 2a. We can see that there are two crossing transitions, one from s_2 in F^1 and one from s_5 in F^2. For each crossing transition, an additional g-state is introduced and the source state of the original crossing transition will have that as destination state. In Figure 2b the destination states of the crossing transitions from s_2 are changed from s_3 and s_4 to g_3 and g_4 respectively. A crossing transition is completed by the following sequence of events. When the machine enters a g-state this is detected and the global state, denoted *R*, of the partitioned FSM will change. The global state is pointing out which one of the sub-FSMs that is active. A change in the global state will deactivate the sub-FSM containing the source state and activate the sub-FSM containing the destination state. Consider the crossing transition from s_2 to s_3 in the example. The transition from s_2 will enter g_3. This will cause the global state *R* making a transition from r_1 to r_2. The global state will after completion of the crossing transition point out F^2 as the active sub-FSM and not F^1 as before. In a synchronous FSM the crossing transition, as all transitions, must be completed within one clock cycle. From the example above it can be seen that a crossing

transition requires two state transitions which will take two clock cycles to complete in a synchronous machine. To solve this, the transition from the g-state to the entry state of the destination sub-FSM (e.g. g_1 to s_1) is made asynchronously. By that it is meant that the transition is triggered by a signal transition rather than by the active edge of the clock signal. A control signal, decoded from the g-state, makes the global state to change. The states originating from the initial FSM and the additional g-states are stored in a state memory clocked by the common clock signal. We call this local state memory. A global, asynchronous, state transition does not permit a local state change which puts a restriction on the state encoding. The local states must be coded in such a way that the code for a g-state and its associated entry state must be identical. From the example in Figure 2b, the following pairs of states, that we call coupled-states, must have identical codes: (s_1,g_1), (s_3,g_3), and (s_4,g_4). The states s_2 and s_5 may share the same state code since they are located in different sub-FSMs and distinguished be the global state.

A, what we call, a coupled-state table describes the behaviour of the decomposed FSM including the sub-FSM interaction. To illustrate the construction of the coupled-state table the example from Figure 2 is used. Its coupled-state table is shown in Figure 3. Each row in the table holds all states for one sub-FSM and each column represents a bundle of states that will have the same local state code. A sequence of state transition $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_5$ will result in the following sequence in the partitioned FSM $s_1 \rightarrow s_2 \rightarrow g_3 \rightarrow s_3 \rightarrow s_5$ where the transition $g_3 \rightarrow s_3$ is asynchronous. In

the table, a local state transition is represented by a horizontal change and a global state transition is represented by a vertical change.

The coupled-states will of course impose restrictions on the state encoding of the local states because it contains information about how the different sub-FSMs are related.

3 State Encoding for Local States

After the FSM partitioning, the state encoding is performed in two steps. First the coupled-state table is build by locating the coupled-states together in bundles. After that the total number of bits in the local state memory is minimised by the "coupled-state merging" algorithm which also takes the state-transition probabilities into account in order to reduce the power. In the second step state-codes are assigned to each bundle. State encoding is made for one sub-FSM at a time, starting with the most active sub-FSM.

3.1 Basic Definitions

The monolithic Mealy-type FSM is defined as a sextuple: $F = (S, X, Y, \delta, \lambda, s_0)$ where *S* is the set of states, *X* is the set of binary inputs, *Y* is the set of binary outputs, δ is the transition function, λ is the output function and s_0 is the initial state.

Let there be a partition on the set S: $\Pi = \{S^1, S^2, \ldots, S^n\}$ where Π is defined as a collection of subsets such that $\bigcup_{m=1}^{n} S^m = S$ and $S^i \cap S^j = \emptyset$ for $i \neq j$ where $1 \leq i, j \leq n$.

The monolithic FSM is decomposed into a set of sub-FSMs where every subset $S^i \in \Pi$ defines a sub-FSM as: $F^m = (S^m, X^m, Y^m, \delta^m, \lambda^m, s_0^m)$. We call states S^m internal states of the sub-FSM. X^m is the set of input variables at all transitions from the states in S^m, and Y^m is the set of outputs variables on the sets S^m and X^m.

We define a set of states, $T(S^m)$, not included in F^m to which there are transitions from the states of F^m : $T(S^m) = \{s_j \mid \delta(s_k, X_h) = s_j, s_j \notin S^m, s_k \in S^m\}$.

$Q(S^m)$ is defined as the set of states in F^m where there are transitions from other sub-FSMs as: $Q(S^m) = \{s_j \mid \delta(s_k, X_h) = s_j, s_j \in S^m, s_k \notin S^m\}$.

For the above defined sets we will use the shorter notations T^m, Q^m.

The set of g-states G^m, that reflects the set of destinations states of the crossing transitions in F^m is defined as: $G^m = \{g_i \mid s_i \in T^m\}$.

Let the set of local states in the transformed network of F^m to be U^m: $U^m = S^m \cup G^m$.

3.2 State Bundling

There are two reasons for state-bundling: 1) it enables state in different sub-FSMs to share state codes and 2) it enables an efficient asynchronous global state transition. In the state encoding step the state bundles are considered as states. In this section the criteria and procedures for state bundling will be introduced. First a basic procedure is introduced which will give good results for most

partitioned FSMs. Then a procedure for merging the coupled-states into the same bundles is presented. This will for even exceptional cases give improved results.

3.2.1 Basic algorithm

We start with the following example of a partitioned FSM. Let there be a partition $\Pi = \{S^1, S^2, S^3, S^4\}$ which results in the following local sets of states: $U^1 = \{s_1, s_2, s_3, g_4\}$, $U^2 = \{s_4, s_5, s_6, g_7\}$, $U^3 = \{s_7, g_1\}$, and $U^4 = \{s_8, s_9, s_{10}, s_{11}, s_{12}, g_1, g_5\}$. The duty time of each partition U^m , or the probability of the corresponding sub-FSM to be active, is given by the sum of the static state probability of states inside the partition, that is $T(U^m) = \sum prob(s_i), s_i \in S^m$. In the rest of the paper, it is denoted as T^m.

State bundling starts from the coupled-states, the states that are the source and destination states of an asynchronous transition. From previous discussion we know that these have identical state codes. We construct a table of *n* rows for an n-way partitioned FSM where each column represents a bundle of states that after state encoding will have the same state code. The set of bundles *B* needed is defined as $B = \{b_1, b_2, \ldots, b_p\}$, where $p = \left| \bigcup_{m=1}^{n} Q^m \right|$. In other words, the number of bundles needed for the coupled-states are the sum of the entry-states of all sub-FSMs. Two probabilities are defined that reflects the property of state bundles. *State bundle probability* is defined as: $prob(b_m) = \sum prob(s_i), s_i \in b_m$. *Bundle transition probability* is defined as: $prob(b_m b_k) = \sum prob(s_i s_j), s_i \in b_m, s_j \in b_k$ describes the probability for a state transition between states in the bundles b_m

The monolithic FSM is decomposed into a set of sub-FSMs where every subset $S^i \in \Pi$ defines a sub-FSM as: $F^m = (S^m, X^m, Y^m, \delta^m, \lambda^m, s_0^m)$. We call states S^m internal states of the sub-FSM. X^m is the set of input variables at all transitions from the states in S^m, and Y^m is the set of outputs variables on the sets S^m and X^m.

We define a set of states, $T(S^m)$, not included in F^m to which there are transitions from the states of F^m : $T(S^m) = \{s_j \mid \delta(s_k, X_h) = s_j, s_j \notin S^m, s_k \in S^m\}$.

$Q(S^m)$ is defined as the set of states in F^m where there are transitions from other sub-FSMs as: $Q(S^m) = \{s_j \mid \delta(s_k, X_h) = s_j, s_j \in S^m, s_k \notin S^m\}$.

For the above defined sets we will use the shorter notations T^m, Q^m.

The set of g-states G^m, that reflects the set of destinations states of the crossing transitions in F^m is defined as: $G^m = \{g_i \mid s_i \in T^m\}$.

Let the set of local states in the transformed network of F^m to be U^m: $U^m = S^m \cup G^m$.

3.2 State Bundling

There are two reasons for state-bundling: 1) it enables state in different sub-FSMs to share state codes and 2) it enables an efficient asynchronous global state transition. In the state encoding step the state bundles are considered as states. In this section the criteria and procedures for state bundling will be introduced. First a basic procedure is introduced which will give good results for most

partitioned FSMs. Then a procedure for merging the coupled-states into the same bundles is presented. This will for even exceptional cases give improved results.

3.2.1 Basic algorithm

We start with the following example of a partitioned FSM. Let there be a partition $\Pi = \{S^1, S^2, S^3, S^4\}$ which results in the following local sets of states: $U^1 = \{s_1, s_2, s_3, g_4\}$, $U^2 = \{s_4, s_5, s_6, g_7\}$, $U^3 = \{s_7, g_1\}$, and $U^4 = \{s_8, s_9, s_{10}, s_{11}, s_{12}, g_1, g_5\}$. The duty time of each partition U^m , or the probability of the corresponding sub-FSM to be active, is given by the sum of the static state probability of states inside the partition, that is $T(U^m) = \sum prob(s_i), s_i \in S^m$. In the rest of the paper, it is denoted as T^m.

State bundling starts from the coupled-states, the states that are the source and destination states of an asynchronous transition. From previous discussion we know that these have identical state codes. We construct a table of *n* rows for an n-way partitioned FSM where each column represents a bundle of states that after state encoding will have the same state code. The set of bundles *B* needed is defined as $B = \{b_1, b_2, \ldots, b_p\}$, where $p = \left| \bigcup_{m=1}^{n} Q^m \right|$. In other words, the number of bundles needed for the coupled-states are the sum of the entry-states of all sub-FSMs. Two probabilities are defined that reflects the property of state bundles. *State bundle probability* is defined as: $prob(b_m) = \sum prob(s_i), s_i \in b_m$. *Bundle transition probability* is defined as: $prob(b_m b_k) = \sum prob(s_i s_j), s_i \in b_m, s_j \in b_k$ describes the probability for a state transition between states in the bundles b_m

and b_k. The states are aligned in columns in such a way that all states coupled to each other reside in the same column. In Figure 4, showing the state table for our example, the entries for the coupled-states are shaded grey. We can see that for example s_4 in F^2 is in the same column as g_4 in F^1, i.e. s_4 and g_4 are coupled-states. The state bundling procedure first adds the bundles containing coupled-states and thereafter states not coupled may be freely positioned in any bundle as long as all states residing in the same sub-FSM have unique state codes. The pseudo-code for the bundling algorithm is shown in Figure 5.

The efficiency of this procedure is dependent on the ratio of the number of coupled-states to the number of *free states* given by: $c = \frac{\left| \bigcup_{m=1}^{n} Q^m \right|}{\left| \bigcup_{m=1}^{n} S^m \setminus Q^m \right|}$.

For most partitioned FSMs, partitioned according the state transition probabilities, have small numbers of crossing transitions and will therefore have small *c*. For that reason this basic state bundling procedure works well in most cases.

3.2.2 Merged coupled-state algorithm

Using the basic bundling algorithm for FSM partitions with large *c* will result in large local state memory. However, the number of clocked state memory bits for each sub-FSM will not necessarily be all state bits. The objective of merging coupled-states is to reduce the total number of state bits. To illustrate the merged coupled-state algorithm we use the example in Figure 6 that have a $c = 5/2$.

The initial coupled-state table, before merging the coupled-states, is shown in Figure 7a, where the five g-states reside in five different bundles. Fixed state

codes for the state bundles are assumed were the bundle index indicate the binary value of the code (b_0 has the code "000", b_1 "001" and so on). The merging procedure is performed in the following steps.

1) The objective of the *Sort()* function is to introduce prioritization among the sub-FSMs. It sorts the sub-FSMs according to the descending order of their duty time T^m. Since the sub-FSMs with high duty time generally contribute more to the final power dissipation, they are given higher priority in the coupled-state merging step. The sorted coupled-state table is shown in Figure 7b. After that, the coupled-state with the highest state bundle probability is moved to the b_0 bundle that will always be assigned the state code "zero" after state encoding. The objective is to minimize the switching activity in the next-state bit-lines for crossing transitions. The reason for this is that a deactivated sub-FSM's next-state is always encoded to "zero" in order to enable efficient implementation of merging the next-state variables of the different sub-FSMs [10] by using OR gates.

2) In order to reduce the number of bits in the local state memory, the algorithm merges two or more coupled-states into the same bundles when possible. The algorithm first tries to merge the coupled-states in bundles to the right of the leftmost bundle (b_0) in the sorted coupled-state table. In the cases where only one of two or more coupled-states can be merged, the one in the bundle with highest state bundle probability is chosen. After a merging has been completed, the table is sorted again as described in step 1. When no more coupled-states can be merged into b_0 it is locked. Now the same procedure is done for b_1 and continues until the last column has been reached. In the example given in Figure

7, it is shown that both b_2 and b_3 can be merged into b_0. Because the state bundle probability of b_3 is 0.3 ($\Pr ob(b_3) = prob(s_4) = prob(T^4)$), higher than that of b_2 ($\Pr ob(b_2) = prob(s_3) \le prob(s_3) + prob(s_2) = prob(T^3) = 0.2$), b_3 is chosen to be merged into b_0. The updated coupled-state table after merging is shown in Figure 7c), where the total number of state bundles is reduced from 5 to 4.

3.3 Basic State Encoding Algorithm

The basic state encoding algorithm is a straight-forward technique that does not consider power optimizations at all. It takes the initial coupled-state table without merging and put free states (S^m) into the bundles starting from bundle b_0. The whole state bundling algorithm is given in Figure X. Each bundle is assigned the binary code that corresponds to its index. Binary-encoding makes sure the number of clocked local state bits in each sub-FSM is minimal.

3.4 Power Optimized State Encoding

Since we consider the state code assigned to the bundles to be fixed, the task of state encoding optimization is to move states to suitable bundles in order to reduce the switching activity in the state bit lines.

We first consider coupled-states in the table (Figure 7d). Since every bundle is given a unique state code and can be viewed upon as a state, the algorithm tries to reduce the switching activity in the transitions between these bundles. At the same time, the algorithm tries to keep the sub-FSMs with higher duty time to minimum-length encoding. The merging algorithm, described in the previous section, has sorted rows in descending order of the duty period. Therefore,

encoding starts from the top row. For each row, the position of state bundles will be optimized first and then locked, which will not be changed afterwards. A greedy algorithm is used to minimize the hamming distance for the bundle transition probability. The algorithm is shown in Figure X. We illustrate the procedure through an example. In Figure X, the initial coupled-state bundles have been built including b_0, b_1, b_2, b_3. As mentioned before, b_0 is the state bundle with highest state bundle probability and its position is locked initially. We start the state bundle optimization from b_1 because it is the only bundle besides b_0 that has a valid state in the top row representing sub-FSM F^1. To make coupled-state bundles in F^1 use the minimum length codes, b_1 can only be assigned the code "01" and thereby the coupled-state bundle in F^1 only use one state bit. Since the position of b_0 and b_1 is locked after the assignment of F^1, only b_2 and b_3 coupled-state bundle are left. In F^4 the number of minimum local state bits needed for the state bundles is 2 (obtained from minimumCodeLength() function in Figure 10). Since the codes "00" and "01" already has been assigned for b_2 and b_3, the only possible codes are "10" or "11". We compare the bundle transition probability of b_2 and b_3 with already assigned state bundles b_0 and b_1. If the transition probability between b_3 and b_0 is assumed to be the highest, we assign b_3 to the position "10" which has the hamming distance of 1 to b_0; b_2 is subsequently assigned the code "11". Since all state bundles have been assigned, state encoding for the coupled-state bundles is completet. The result of the coupled-state encoding optimization is shown in Figure 11a) where the position of b_3 and b_2 has been swapped.

The next step is to encode the *free states*, i.e. states not coupled to any other sub-FSM. Since these are internal states in a sub-FSM, each sub-FSM can be

separately optimized in an arbitrarily order. In each sub-FSM, one single free state is considered at a time. That is the one having the highest state transition probability to a certain state in this sub-FSM or to a state in another sub-FSM. Constrained by minimum-length encoding, the algorithm minimises the hamming distance for local state transitions with high state transition probabilities. For example, in sub-FSM F^3, s_2 is a free state. We first determine the minimum state code bits for F^3 that is 2. (obtained from minimumLengthCode() function in Figure 12). Since s_2 only has the state transition to s_3, we put s_2 in the state bundle of b_1, which has the smallest hamming distance to b_2, which is 1, (where state s_3 is in). It can be noticed that s_2 also can be put in b_3, which has the same hamming distance from b_2. In sub-FSM F^5, s_6 is a free state. It has the state transitions to s_5 and s_0, where the former transition occurs in sub-FSM F^5 and the latter transition is between sub-FSM F^5 and F^1. Assume that the state transition probability between s_6 and s_5 is higher than that between s_6 and s_0, we put s_6 in the bundle b_2 with code "11", which has only one bit hamming distance from b_3 with code "10". Bundle b_1 is not chosen for s_6 is because there are two bits hamming distance between b_1 and b_3.

The final state table including merging coupled-state and state encoding procedure is shown in Figure 11b) whereas the initial state bundle table without optimization shown in Figure 11c).

4 Experimental Results

In this section we present results showing how the state bundling and state encoding algorithms, given in section 3, influences the power consumption of partitioned FSMs. We have implemented the algorithms in an automatic

synthesis tool that is based on our previous work [12]. Seven of the MCNC [13] standard benchmarks were used in the experiments. The number of states in these benchmarks range from 19 to 118. To determine the state transition probabilities of the FSMs, average input probability and switching probability are inputs to the tool. In our experiments, both are set to 0.5. The power and area figures presented in graphs and tables come from gate-level estimations in Power Compiler and logic synthesis is done by Design Compiler, both these tools from Synopsys [14]. We use a 0.18μm CMOS standard cell library [15] and we assume power supply voltage V_{dd} of 1.8V and a clock frequency of 20 MHz.

The total average power of a monolithic FSM is $P_{tot,mono} = P_{clk} + P_{reg} + P_{ns} + P_{out}$. Where P_{clk} is the clock net power, P_{reg} is the power in the state registers, P_{ns} is the power in the next-state function, and P_{out} is the power in the output function. The total power of the partitioned FSM is $P_{tot,part} = P_{clk} + P_{reg} + P_{ns} + P_{out} + P_{oh}$. Where the P_{oh} is the power-overhead which is the sum of the power dissipated in the global state memory, circuits for idle condition detection, and shut-down circuits.

FSM partitioning is alone an efficient method for achieving power reductions. As shown in Figure 11, significant reductions have been obtained for the mixed synchronous/asynchronous architecture without optimized state encoding.

In the partitioned FSM a significant part of the power is dissipated in the global state memory and the circuits for idle condition detection and shut-down circuits (P_{oh}). This part is not affected by state encoding procedures presented in this paper. To look in detail on how the proposed procedures affect the power

consumption, we first consider only power dissipation in the sub-FSMs ($P_{tot,sub-FSM} = P_{tot,part} - P_{oh}$). In Figure 12, it is shown how coupled-state merging and optimized state encoding affects power dissipation in comparison to the basic procedures. Merging of coupled-states has very little to say for the power consumption and for three of the benchmarks (s832, s820, and scf) coupled-state merging has not affect at all. This is what could be expected since the objective here is to minimize the number of state bits and not the power. The state-encoding gives in average a reduction of 13%.

As shown in Figure 11, the sub-FSM power ($P_{tot,sub-FSM}$) is only a portion the total ($P_{tot,part}$) which is in average 40%. For the total power, the reductions are shown in Figure 12 with an average reduction of 6%.

It Table 1 can be seen that the partitioning algorithm result in small sub-FSMs with high duty probability T^i and the large sub-FSMs have low duty period. From In Figure 13 it can be seen that sub-FSMs with large number of bits in the local state memory, the power optimization procedure is efficient but for the ones with few bits only small reductions can be obtained.

5 Conclusions

In this paper we have presented a state encoding algorithm for partitioned FSM composed of coupled sub-FSM with shared state memory. The algorithm takes the properties of partitioned FSMs and the constraints imposed by the implementation architecture in to account. The relation between the coupled sub-FSMs is given by the state bundling. State encoding is carried out sequentially, one sub-FSM at a time where high priority is given to sub-FSMs with high duty

time. The power reductions we achieved for the sub-FSMs are promising. The reductions for the partitioned FSMs as a whole are obviously lower since state encoding cannot reduce the power in the asynchronous state memory, idle condition detection logic, and the shut-down logic that are already established before state encoding. This limitation comes from the fact that we first have the partitioning procedure followed by the state-encoding. An algorithm for simultaneous partitioning and state encoding, as the one presented in [16], removes this limitation. But the complexity of the problem increases dramatically and so do the run-times for the algorithms. The average power reduction achieved in [16] is very close to ours. It is however difficult to compare our results to theirs since there is no information on the statistics given of the input signals to the FSM benchmarks. A direction for future work is to develop an algorithm for simultaneous partitioning and state encoding for the mixed synchronous/asynchronous architecture in order to find out if the more complex algorithms will pay off in reduced power.

6 References

1. L. BENINI, G DE MICHELI: 'Dynamic power managment : Design techniques and CAD tools' (Kluwer Academic Publishers, 1998)

2. A. ABDOLLAHI, F. FALLAH, M. PEDRAM: ' Leakage Current Reduction in CMOS VLSI Circuits by Input Vector Control', IEEE Tans. On VLSI, 2004, **12**, pp. 140-154

3. M. ALADINA, J. MONTEIRO, S. DEVADAS, A. GOSH : 'Precomputational-based sequential logic optimization for low power', *IEEE Trans. on VLSI*, 1994, **2**, pp. 426-436

4. J. MONTEIRO, S. DEVADAS, A. GHOSH : 'Sequential logic optimization for low power using input-disabling precomputation architectures', *IEEE Trans. on CAD*, 1998, **17**, pp. 279-284

5. L. BENINI, G. DE MICHELI, A. LIOY, E. MACII, G. ODASSO, M. PONCINO : 'Synthesis of power-managed components based on computational kernal extraction', *IEEE Trans. On CAD*, 2001, **20**, pp. 1118-1131

6. S-H. CHOW, Y-C. HO, T. HWANG: 'Low-power realization of finite-state machines – a decomposition approach', *ACM Trans. on design automation of electronics systems*, 1996, **1**, pp. 315-340

7. L. BENINI, P. SIEGEL, G. DE MICHELI: 'Automatic synthesis of low-power gated-clock finite-state machines', 1996, **6**, *IEEE Trans. on CAD*, pp. 630-643

8. B. OELMANN, K. TAMMEMÄE, M. KRUUS, M. O'NILS: 'Automatic FSM synthesis for low-power mixed synchronous/asyncrhonous implementation', *Journal of VLSI Design - Special issue on low-power design*, 2001, 12, pp. 167-186

9. B. OELMANN, M. O'NILS: 'Asynchronous control of low-power gated-clock finite-state machines', *Proceedings of IEEE International conference on electronics, circuits, and systems*, 1999, pp. 915-918

10. C. CAO, B. OELMANN: 'Mixed synchronous/asynchronous state memory for low power FSM design', *Proceedings of the EUROMICRO symposium on digital system design*, 2004, pp. 363-370

11. C. CAO, M. O'NILS, B. OELMANN: 'A tool for low-power synthesis of FSMs with mixed synchronous/asynchronous state memory', 2004, *IEEE Proceedings of the Norchip Conference*, pp. 199-202

12. B. OELMANN, M. O'NILS: 'A low power hand-over mechanism for gated-clock FSMs', *Proceedings of the European conference on circuit theory and design*, 1999, pp. 118-121

13. S. YANG: 'Logic synthesis of optimization benchmarks – user guide version 3.0', *MCNC Technical report*

14. Synopsys inc.: 'http://www.synopsys.com', company homepage.

15. United Microelectronics Corp: 'http;//www.umc.com.tw', company homepage

16. G. VENKATARAMAN, S. M. REDDY, I. POMERANZ: 'GALLOP: Genetic Algorithm based Low Power FSM Synthesis by Simultaneous Partitioning and State' Assignment', The Sixteenth International Conference on VLSI Design, 2003, pp. 533-538

Figure captions:

Figures:

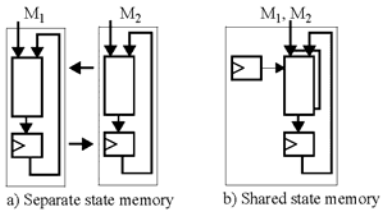


FIGURE 1. Structural decomposition of FSM

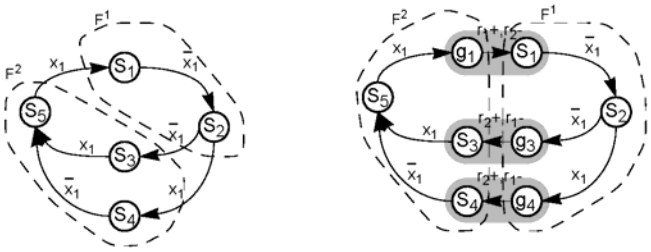


FIGURE 2. Example, a) Monolithic FSM with state partition indicated, b) Coupled states introduced

B:	b_1	b_2	b_3
F^1	g_1	s_3	s_4
F^2	s_1	g_3	g_4

FIGURE 3. Example, Coupled state table

```
struct subFSM {
    set of int S, G, Q;
}

set of struct subFSM F;
int sb[n, max(⌈ log2|U| ⌉) ] ← null;

assignCoupledStates(set of struct subFSM F, int sb)

    int i,j ← 1;
    for all f ∈ F {
        for all q ∈ f.Q {
            i ← indexOf(f);
            sb[i,j] ← q;
            for all ft ∈ F\f {
                for all g ∈ ft.G {                    //g states in other subFSMs
                    if (indexOf(g) = indexOf(q))
                        sb[indexOf(ft),j] ← g;
                }
            }
            j ← j +1;
        }
    }
}
assignFreeStates(set of struct subFSM F, int sb)
{
    for all f ∈ F {
        int j ← 1;
        i ← indexOf(f);
        for all s ∈ f.S \f.Q {
            while (sb[i,j] ≠ null)
                j ← j +1;
            sb[i,j] ← s;
        }
    }
}
```

FIGURE 4. Pseudo code for bundling of the coupled (assignCoupledStates) and free states (assignFreeStates)

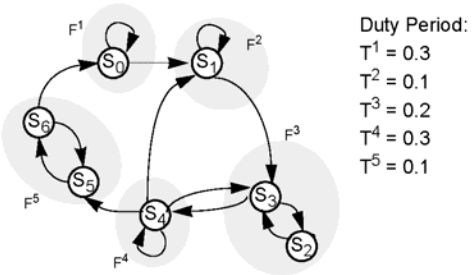


FIGURE 5. Example of a partitioned FSM with high *c*.

a) Initial coupled state table

B:	b_0	b_1	b_2	b_3	b_4
F^1	s_0	g_1	-	-	-
F^2	-	s_1	g_3	-	-
F^3	-	-	s_3	g_4	-
F^4	-	g_1	g_3	s_4	g_5
F^5	g_0	-	-	-	s_5

b) Sorted table

B:	b_0	b_1	b_2	b_3	b_4
F^1	s_0	g_1	-	-	-
F^4	-	g_1	g_3	s_4	g_5
F^3	-	-	s_3	g_4	-
F^2	-	s_1	g_3	-	-
F^5	g_0	-	-	-	s_5

c) After merging coupled-state

B:	b_0	b_1	b_2	b_3	b_4
F^1	s_0	g_1	-	-	-
F^4	s_4	g_1	g_3	g_5	-
F^3	g_4	-	s_3	-	-
F^2	-	s_1	g_3	-	-
F^5	g_0	-	-	s_5	-

d) Final coupled state table

B:	b_0	b_1	b_2	b_3
F^1	s_0	g_1	-	-
F^4	s_4	g_1	g_3	g_5
F^3	g_4	-	s_3	-
F^2	-	s_1	g_3	-
F^5	g_0	-	-	s_5

FIGURE 6. Optimized state table

```
struct subFSM {
    set of int S, G, Q;
}

set of struct subFSM F;
int sb[n, max(⌈log2|U#|⌉)];  //state bundle table
double probBundle[numberOf(F.G)]; //sum of static state probability of states in each state bundle
mergeCoupledStates(set of struct subFSM F, int sb, double probBundle)

    sort(sb);
    g_n ← numberOf(F.G);
    for (i ← 1; i < g_n; i ← i+1){
        max_gain ← 0;
        opt_b ← 0;
        for (j ← i+1; j ≤ g_n; j← j+1){
            row ← 1;
            while (sb[row, i]=null ||sb[row,j]=null)
                row ← row+1;
            if (row=n){                    //column i and j can be merged
                gain ← probBundle [i]+probBundle [j];
                if (gain > max_gain){
                    max_gain ← gain;
                    opt_b ← j;
                }
            }
        }
        if (opt_b > 0){                    //find column obt_b can be merged into column i
            for (k ← 1; k ≤  n; k ← k+1){
                if (sb[k, i] = null)
                    sb[k, i] ← sb[k, obt_b];
            }
            "remove column opt_b in sb";
            g_n ← g_n-1 ;
        }
    }
    sort(sb);
}
```

FIGURE 7. Pseudo code for g-state merging

B: C:	b_1 000	b_2 001	b_3 010	b_4 011	b_5 100	b_6 101	b_7 110	b_8 111	$\lceil \log_2 \lvert U^i \rvert \rceil$
F^1	s_1	g_4	s_2	s_3	-	-	-	-	2
F^2	s_6	s_4	s_5	g_7	-	-	-	-	2
F^3	g_1	-	-	s_7	-	-	-	-	2
F^4	g_1	s_8	g_5	s_9	s_{10}	s_{11}	s_{12}	s_{13}	3

FIGURE 8. State encoding in re-ordered state table

```
int old_sb[n, max(⌈ log2|U^i| ⌉) ];            //state bundle table before optimization
int new_sb[n, max(⌈ log2|U^i| ⌉) ]← null;   //state bundle table after optimization
double b_matrix[numberOf(mergedCoupledState),numberOf(mergedCoupledState)];
optimiseCoupledStates(int old_sb, double b_matrix, int new_sb)

    int b[numberOf(mergedCoupledState)];        //state bundles
    struct sub_b;   //subset of state bundles
    for (i ← 1; i ≤ numberOf(mergedCoupledState); i ← i+1)
        b[i] ← the ith column of old_sb;
    lock(b[1]) ;
    for (i ← 1; i ≤ n; i ← i+1)
        new_sb[i, 1] ← b[1];
    for (i ← 1; i ≤ n; i ← i+1){
        sub_b ← ∅;
        for (j ← 1; j ≤ numberOf(mergedCoupledState); j ← j+1){
            if (old_sb[i, j] ≠ null)
                sub_b ← sub_b ∪ b[j];
        }
        b_n ← least state bits needed for sub_b in new_sb;
        for unlocked state bundle b[x]∈ sub_b{
            for each locked state bundle  b[y] in b
                "find b_matrix[xiyi] with maximal state bundle transition probability";
        }
        for (j ← 1; j ≤ 2^b_n; j ← j+1)
            "find m is the column index of b[yi] in new_sb,such that
             Hammingdistance(binaryCode(m),binaryCode(j)) is minimal";
        for (k ← 1; k ≤ n; k ← k+1)
            new_sb[k, j] ← b[xi];
        lock(b[xi]);
    }
}
```

FIGURE 9. Pseudo code for optimized coupled state encoding

a) Final coupled state table after optimization

B: C:	b_0 00	b_1 01	b_3 10	b_2 11
F^1	s_0	g_1	-	-
F^4	s_4	g_1	g_5	g_3
F^3	g_4	-	-	s_3
F^2	-	s_1	-	g_3
F^5	g_0	-	s_5	-

b) Final state table after free state optimization

B: C:	b_0 00	b_1 01	b_3 10	b_2 11	bits
F^1	s_0	g_1	-	-	1
F^4	s_4	g_1	g_5	g_3	2
F^3	g_4	s_2	-	s_3	2
F^2	-	s_1	-	g_3	2
F^5	g_0		s_5	s_6	2

c) State table before state encoding optimization

B:	b_0 000	b_1 001	b_2 010	b_3 011	b_4 100	bits
F^1	s_0	g_1	-	-	-	1
F^2	-	s_1	g_3	-	-	2
F^3	s_2	-	s_3	g_4	-	2
F^4	-	g_1	g_3	s_4	g_5	3
F^5	g_0	s_6	-	-	s_5	3

FIGURE 10. Comparison of state bundle table before and after optimization

```
struct subFSM {
      set of int S, G, Q;
}

set of struct subFSM F;
int sb[n, max(⌈log2|U^fi|⌉) ];  //state bundle table before free states assignment
double s_matrix[numberOf(S),numberOf(S)];  //state transition probability matrix

optimizeFreeStates(set of struct subFSM F, int sb, double s_matrix)
{
      int b_n[n];     //minimun state code length in each subFSM
      int sb_backup[n, max(⌈log2|U^fi|⌉) ];
      sb_backup ← copy(sb);
      assignFreeStates(F,sb_backup);
      for (i ← 1; i ≤ n; i ← i+1)
            b_n[i] ← minimumLengthCode(sb_backup[i]);
      for all f ∈ F {
            i ← indexOf(f);
            A ← f.Q' ∪ f.G;                  //assigned states ,g states included
            D ← f.S \f.Q;                    //unassigned states
            do{
                  count ← numberOf(D);   //unassigned state number
                  if (count>0){
                        for all a ∈ A {
                              for all d ∈ D
                                    "find s_matrix[a_i,d_j] with highest state transition probability";
                        }
                        k ← 1;
                        while (sb[i,k] ≠ a_i)
                              k ←  k+1;
                        for (m ← 1; m ≤  2^b_n[i]; m ← m+1){
                              if(sb[i, m] ≠ null)
                                    "find position m_i with minimal Hammingdistance(binaryCode(m_i -1), binaryCode(k-1));"
                        }
                        sb[i, m_i]  ← d_j;
                        A← A' ∪ d_j;
                        D← D\d_j;
                        count ← count-1;
                  }
            }while (count>0)
      }
}
```

FIGURE 11. Pseudo code for free state encoding optimization

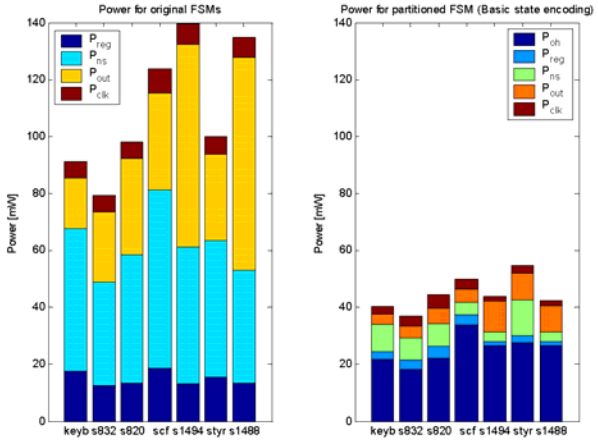


FIGURE 12. Power reductions for partitioned FSMs

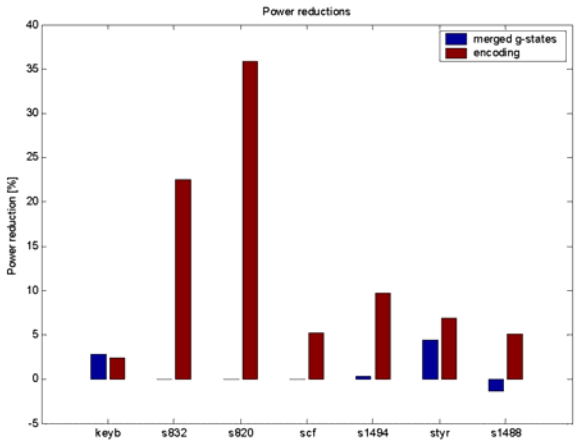


FIGURE 13. Power reductions in the sub-FSMs

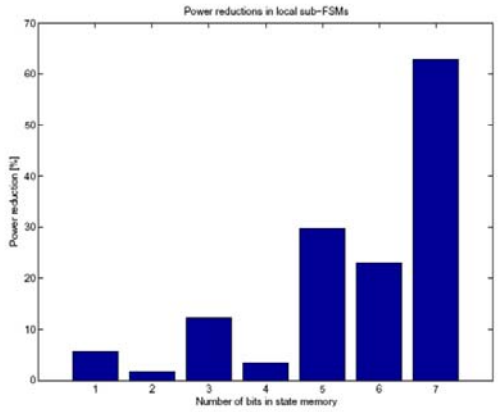


FIGURE 14. Power reductions versus number of bits in the state memory

TABLE 1. Structural information from the FSM decomposition

FSM	keyb	s832	s820	scf	s1494	styr	s1488
$\|S^1\|$	1	4	4	4	1	1	1
$\|U^1\|$	4	5	5	5	2	4	2
$\|PI^1\|$	3	6	6	1	3	5	3
$\|PO^1\|$	1	5	9	12	12	6	13
$\|T^1\|$	0.99	0.99	0.99	0.96	0.91	0.85	0.91
$\|S^2\|$	1	21	4	4	1	1	1
$\|U^2\|$	3	24	7	8	4	4	4
$\|PI^2\|$	6	18	9	3	3	5	3
$\|PO^2\|$	0	17	10	8	7	2	7
$\|T^2\|$	0.27	0.03	0.03	0.08	0.20	030	0.20
$\|S^3\|$	1		17	110	1	2	1
$\|U^3\|$	4		23	8	4	3	4
$\|PI^3\|$	7		17	3	6	6	6
$\|PO^3\|$	1		12	8	13	1	12
$\|T^3\|$	0.18		<0.01	0.02	0.08	0.20	0.08
$\|S^4\|$	1				1	4	1
$\|U^4\|$	4				2	8	3
$\|PI^4\|$	7				0	5	1
$\|PO^4\|$	1				5	5	4
$\|T^4\|$	0.09				0.02	0.08	0.02
$\|S^5\|$	15				1	8	1
$\|U^5\|$	16				3	16	2
$\|PI^5\|$	6				1	7	0
$\|PO^5\|$	2				4	10	3
$\|T^5\|$					0.03	0.03	0.03
$\|S^6\|$					1	14	42
$\|U^6\|$					3	21	46
$\|PI^6\|$					2	6	8
$\|PO^6\|$					7	10	19
$\|T^6\|$					0.02	<0.01	0.02
$\|S^7\|$					42		1
$\|U^7\|$					46		3
$\|PI^7\|$					8		2
$\|PO^7\|$					19		5
$\|T^7\|$					0.02		0.02