applied sciences

MDPI

Review

Basic Approaches for Reducing Power Consumption in Finite State Machine Circuits—A Review

Alexander Barkalov [1,*], Larysa Titarenko [1,2], Jacek Bieganowski [1,*] and Kazimierz Krzywicki [3]

[1] Institute of Metrology, Electronics and Computer Science, University of Zielona Gora, ul. Licealna 9, 65-417 Zielona Gora, Poland; l.titarenko@imei.uz.zgora.pl
[2] Department of Infocommunications, Kharkov National University of Radio Electronics, 61000 Kharkov, Ukraine
[3] Department of Technology, The Jacob of Paradies University, ul. Teatralna 25, 66-400 Gorzow Wielkopolski, Poland; kkrzywicki@ajp.edu.pl
* Correspondence: a.barkalov@imei.uz.zgora.pl (A.B.); j.bieganowski@imei.uz.zgora.pl (J.B.)

Abstract: Methods for reducing power consumption in circuits of finite state machines (FSMs) are discussed in this review. The review outlines the main approaches to solving this problem that have been developed over the last 40 years. The main sources of power dissipation in CMOS circuits are shown; the static and dynamic components of this phenomenon are analyzed. The power consumption saving can be achieved by using coarse-grained methods common to all digital systems. These methods are based on voltage or/and clock frequency scaling. The review shows the main structural diagrams generated by the use of these methods when optimizing the power characteristics of FSM circuits. Also, there are various known fine-grained methods taking into account the specifics of both FSMs and logic elements used. Three groups of the fine-grained methods targeting FPGA-based FSM circuits are analyzed. These groups include clock gating, state assignment, and replacing look-up table (LUT) elements by embedded memory blocks (EMBs). The clock gating involves a separate or joint use of such approaches as the (1) decomposition of FSM inputs and (2) disabling FSM inputs. The aim of the power-saving state assignment is to reduce the switching activity of a resulting FSM circuit. The replacement of LUTs by EMBs allows a reduction in the power consumption due to a decrease in the number of FSM circuit elements and their interconnections. We hope that the review will help experts to use known methods and develop new ones for reducing power consumption. We think that a good knowledge and understanding of existing methods of reducing power consumption is a prerequisite for the development of new, more effective methods to solve this very important problem. Although the methods considered are mainly aimed at FPGA-based FSMs, they can be modified, if necessary, and used for the power consumption optimization of FSM circuits implemented with other logic elements.

Keywords: FPGAs; LUTs; EMBs; clock gating; decomposition; state assignment; coarse-grained methods; fine-grained methods

Citation: Barkalov, A.; Titarenko, L.; Bieganowski, J.; Krzywicki, K. Basic Approaches for Reducing Power Consumption in Finite State Machine Circuits—A Review. *Appl. Sci.* **2024**, *14*, 2693. https://doi.org/10.3390/app14072693

Academic Editors: Chia-Hung Lin, Neng-Sheng Pai, Chao-Lin Kuo and Chang-Hua Lien

Received: 26 February 2024
Revised: 15 March 2024
Accepted: 19 March 2024
Published: 22 March 2024

1. Introduction

Currently, humanity is literally immersed in a sea of various VLSI-based digital systems. As can be seen, for example, from the rapid development of the Internet of things, robotics, and mobile technologies, this sea will deepen and expand. Obviously, the number of digital systems around us will only increase. On the other hand, the modern world is characterized by the need for a reasonable use of electrical energy. This characteristic feature is also evident in the field of information technologies, which has led to the concept of "green computing" [1].

So modern digital systems should be power-efficient. They should consume as little power as possible [2]. It means that power consumption has become a primary concern in the design of integrated circuits [3]. Two main issues are connected with this demand. The

first of them can be formulated as follows: the less the power consumption, the longer the life of various mobile and autonomous devices. The second issue is connected with the increase in heat dissipation.

The vast majority of digital systems consist of both combinational and sequential blocks [4,5]. In this paper, we consider the problem of reducing power consumption by sequential blocks of digital systems. The behavior of a sequential block can be represented using a finite state machine (FSM) model [6,7]. These models are widely used for representing, for example, such sequential blocks as (1) the control units of computers and microcontrollers [8–13]; (2) the hardware–software interfaces and communication protocols of embedded systems [14,15]; (3) various cryptoprocessors [16]; (4) hypertangent and exponential functions [17]; (5) various integral stochastic computing blocks [18]; (6) the activation functions of deep neural networks [19,20]; (7) different stages of cascaded digital processing systems [21,22].

To optimize the power consumption of FSM circuits, it is necessary to take into account the main technological specifics and other main features of logic elements implementing FSM circuits [8,9]. Since the late eighties of the last century, field programmable gate arrays (FPGAs) [23–25] have been used more and more to implement electrical circuits of various systems [26,27]. In this survey, we mostly analyze various approaches used for improving the power consumption of FPGA-based FSM circuits.

Currently, the vast majority of VLSI chips (including FPGAs) are manufactured using the complementary metal–oxide–semiconductor (CMOS) technology [1,28]. In this regard, we are considering methods of reducing the energy consumption aimed at this technology.

The main purpose of the article is a non-analytical review of possible solutions to the problem of reducing power consumption in FSM circuits. The review also shows the results of studies on the effectiveness of these methods. The review considers methods that have appeared in the last 40 years. We did not perform a comparative analysis of these methods and did not conduct additional studies of their efficiency (hence, the review is non-analytical). All research results are owned by the authors of articles and monographs listed in the Section "References".

The rest of the paper is organized as follows: Section 2 briefly shows the theoretical background of FPGA-based FSM synthesis. Section 3 contains the analysis of power dissipation sources in CMOS integrated circuits and gives a classification method for reducing power consumption. Section 4 presents methods for saving power consumption based on clock gating and a decomposition of the initial FSM. Section 5 includes methods for saving power consumption based on various outcomes of state assignment. The methods based on replacing LUTs by EMBs are discussed in Section 6. A brief conclusion ends the paper.

2. FSMs and FPGAs: Background Information

An FSM can be defined as a six-tuple $S = \langle A, X, Y, \delta, \lambda, a_1 \rangle$ [29], where $A = \{a_1, \ldots, a_M\}$ is a set of internal states, $X = \{x_1, \ldots, x_L\}$ is a set of inputs, $Y = \{y_1, \ldots, y_N\}$ is a set of outputs, δ is a transition function, λ is a function of the output, and $a_1 \in A$ is an initial state. An FSM can be represented using various tools, such as state transition graphs (STGs) [4], state transition tables (STTs) [29], algorithmic state machines [30], binary decision diagrams [10,31], and-inverter graphs [32], and graph-schemes of algorithms [29]. In this survey, we used either STGs or STT for the specification of FSMs.

The FSM states are represented by the nodes of an STG. The arcs connecting the nodes define the interstate transitions determined by the input signals which are the conjunctions of inputs $x_l \in X$ (or their complements). These conjunctions are written above the arcs together with the outputs generated during the transitions. To design an FSM circuit, an STG should be transformed into the corresponding STT. An STT includes the following columns [29]: a_m is a current state; a_S is a state of transition; X_h is an input signal determining a transition from a_m to a_s; Y_h is a subset of the set Y generated during

the particular transition. We name this subset a collection of outputs. The numbers of transitions ($h \in \{1, \ldots, H\}$) are shown in the last column of the STT.

There are two main types of FSM, namely, Mealy [33] and Moore FSMs [34]. The first of them was proposed in 1955 by G. Mealy; the second was proposed in 1956 by E. Moore. In both cases, the function δ determines the states of transition as functions depending on the current states and inputs. Thus, it is the following function:

$$\delta(A, X) = A. \quad (1)$$

For Mealy FSMs, the function λ determines the outputs as functions depending on the current states and inputs. It gives the following function:

$$\lambda(A, X) = Y. \quad (2)$$

For Moore FSMs, the function λ determines the outputs with the following function:

$$\lambda(A) = Y. \quad (3)$$

In this article, we mostly analyze the reduced power consumption (RPC) methods for Mealy FSMs. Our choice is explained by the fact that these methods are widely represented in the open scientific and technical literature.

In 1965, Viktor Glushkov proved a theorem on the structural completeness of FSMs [35]. According to this theorem, an FSM circuit is represented as a composition of the combinational part and the memory. The memory is necessary to keep the history of the FSM operation. The history is represented by FSM internal states. This fundamental approach is still widely used for the synthesis of FSM circuits.

An FSM logic circuit is represented by some systems of Boolean functions (SBFs) [29]. To find these SBFs for Mealy FSMs, it is necessary to [29]: (1) encode states $a_m \in A$ by binary codes $K(a_m)$; (2) construct sets of state variables $T = \{T_1, \ldots, T_R\}$ and input memory functions (IMFs) D $= \{D_1, \ldots, D_R\}$; and (3) transform an initial STT into a direct structure table (DST). States $a_m \in A$ are encoded during the step of state assignment [4].

The minimum possible number of state variables R_S is determined by

$$R_S = \lceil log_2 M \rceil. \quad (4)$$

The approach based on (4) defines so-called maximum binary codes [4]. This method is used, for example, in the well-known academic system SIS [36]. But the number of state variables can be different from (4). For example, the one-hot state codes with $R = M$ are used in the academic system ABC [32,37] of Berkeley. The maximum binary codes and one-hot codes define extreme points of the encoding space. There are other approaches for state assignment where the following relation holds: $\lceil log_2 M \rceil \leq R \leq M$.

A state register (RG) keeps the state codes. The register includes R memory elements (flip-flops) having shared inputs of synchronization (*Clock*) and reset (*Start*). Very often, master–slave D flip-flops are used to organize state registers [38,39]. The pulse *Clock* allows the functions $D_r \in$ D to change the RG content.

After the execution of the state assignment, we should create a direct structure table. A DST includes all columns of an STT and three additional columns. These columns include the current state codes $K(a_m)$ and the codes $K(a_s)$ of the states of transitions. At last, a column Φ_h includes the symbols $D_r \in \Phi$ corresponding to ones in the code $K(a_s)$ from the row h of a DST ($h \in \{1, \ldots, H\}$). A DST is a base to construct the following SBFs:

$$\Phi = \Phi(T, X); \quad (5)$$

$$Y = Y(T, X). \quad (6)$$

SBF (5) corresponds to function (1), SBF (6) to function (2). Systems (5)–(6) determine a structural diagram of a so-called P Mealy FSM (Figure 1) [39].

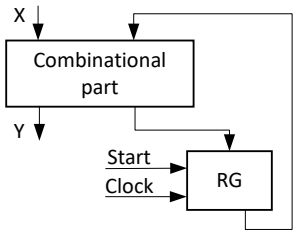

Figure 1. Structural diagram of a P Mealy FSM [39].

The combinational part consists of two blocks. The block of input memory functions generates functions (5). The block of outputs generates system (6). The pulse *Start* writes the code of the initial state to RG. The pulse of the synchronization *Clock* allows information to be written to the register.

In this survey, we mostly discuss the RPC methods for FPGA-based Mealy FSMs. Let us shortly describe the peculiarities of FPGAs.

As a rule, modern FPGAs have an "island-style" architecture [40]. They include different configurable logic blocks (CLBs) and a matrix of programmable interconnections [23–25]. To implement an FSM circuit, we can use either CLBs consisting of look-up table (LUT) elements or embedded memory blocks (EMBs). The output of a LUT can be connected with a flip-flop through a dedicated multiplexor. The flip-flops are necessary for implementing register circuits of sequential blocks [6]. This register is distributed among the LUTs implementing IMFs. The EMBs are synchronous blocks; thus, there is no need for an additional register to keep FSM state codes.

A LUT consists of SRAM cells and can keep a truth table of an arbitrary Boolean function having up to S_L arguments [40,41]. The main feature of a LUT is an extremely small number of inputs, S_L. In modern FPGAs, the number of LUT inputs does not exceed six [23–25]. If some Boolean function depends on more than S_L arguments, it should be transformed using some methods of functional decomposition [41]. It results in multi-level FSM circuits with irregular systems of interconnections. Such circuits resemble programs based on an intensive use of "go-to" operators [42]. Using terminology from programming, we can say that the functional decomposition produces LUT-based circuits with "spaghetti-type" interconnections.

A chip area occupied by a LUT-based FSM circuit is determined mostly by the number of LUTs and the system of their interconnections. Obviously, to reduce the occupied area, it is necessary to reduce the number of LUTs in a circuit. The number of LUTs also influences the power consumption. As noted in [43], "process technology has scaled considerably…with current design activity at 14 and 7 nm". Hence, interconnection delay now dominates logic delay [43]. Also, it is known that interconnections are responsible for consuming up to 70% of the energy [40,44]. Thus, to reduce the consumed energy, it is necessary to reduce the number of interconnections. This improves both the operating frequency and power consumption.

Modern FPGAs include a lot of configurable embedded memory blocks [25]. The EMBs allow the implementation of systems of regular functions [45]. The replacement of LUTs by EMBs allows one to significantly improve the characteristics of resulting FSM circuits [46]. Because of it, there are a lot of design methods targeting EMB-based FSMs [22,46–56]. The survey of different methods of EMB-based design can be found in [45]. Unfortunately, these methods can be used only if there are "free" EMBs, which are not used to implement other parts of a digital system.

An EMB can be characterized by a pair $\langle S_A, t_F \rangle$, where S_A is a number of address inputs, and t_F is a number of memory cell outputs. A single EMB can keep a truth table of an SBF including up to t_F Boolean functions depended on up to S_A arguments [57]. A pair $\langle S_A, t_F \rangle$ defines a configuration of an EMB with a constant total number of bits (size of EMB):

$$V_0 = 2^{S_A} \times t_F. \quad (7)$$

The parameters S_A and t_F could be defined by a designer [58]. It means that EMBs are configurable memory blocks [59]. The following configurations exist for modern EMBs [25]: $\langle 15,\ 1\rangle, \langle 14,\ 2\rangle, \ldots, \langle 9,\ 64\rangle$. Therefore, modern EMBs are very flexible and can be tuned to meet the characteristics of a particular FSM. Because of it, there are a lot of design methods for EMB-based FSMs [22,46–56].

If the condition

$$2^{R+L}(R+N) \leq V_0 \tag{8}$$

holds, then an FSM circuit is implemented as a single EMB [45]. If (8) is violated, then an FSM circuit could be implemented as (1) a network of EMBs or (2) a network of LUTs and EMBs [46,55].

3. Methods of Reducing Power Consumption in CMOS Integrated Circuits

The CMOS technology uses metal–oxide–semiconductor (MOS) field-effect transistors to create gates, flip-plops, and memory blocks such as RAM and ROM, and so on [60]. Each gate uses complementary and symmetrical pairs of p-type and n-type transistors. For example, it requires two MOS transistors to implement the circuit of a NOT gate (Figure 2a).

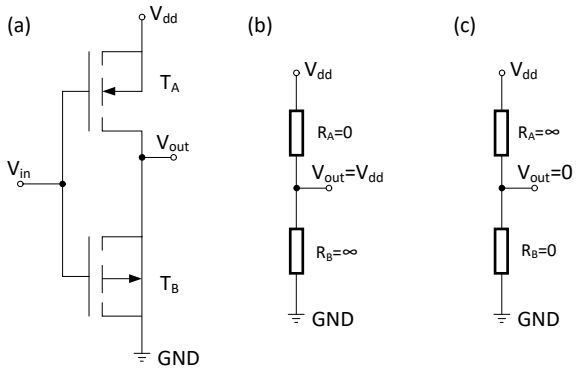

Figure 2. CMOS-based NOT gate (**a**) and its stable states (**b**,**c**).

The NOT gate operates in the following manner. If the voltage V_{in} = "0", then the equivalent electrical circuit is shown in Figure 2b. The transistor T_A is open; its resistance $R_A = 0$. At the same time, the transistor T_B is closed, and its resistance R_B is close to infinity. It means that the following relation takes place: $V_{out} = V_{dd}$ = "1". If V_{in} = "1", then T_A is closed (its resistance is close to infinity) and T_B is open (its resistance is close to zero). It gives $V_{out} = GND = 0$ (Figure 2c). This situation is common: if one of transistors is open, then the second transistor of the pair is closed.

Of course, there is an ideal mode of operation shown in Figure 2. In this ideal case, there is no current between V_{dd} and GND. Thus, the so-called leakage current I_{leak} is absent. But in reality, the resistance of a closed transistor is far from infinity, and the resistance of an open transistor is greater than zero. This means that a small leakage current still exists. This current is responsible for the static power consumption of a CMOS gate in its stable state.

There is parasitic load capacitance between the wires V_{out} and GND. It is responsible for a dynamic power consumption of a gate. Obviously, it takes some time to charge (from "0" to "1") or discharge (from "1" to "0") the parasitic capacitor C_{par}. Until the final stable voltage (either "0" or "1") is established at the gate output, some power is consumed.

When a gate is switched, there is a very small instant of time when both transistors are open. It means that during that time, there is a short circuit current I_{sc} between the voltage source V_{dd} and the ground, GND.

Therefore, there are two categories of power consumption in CMOS gates: static and dynamic. The static power P_{st} is connected with the existence of the leakage current I_{leak}. The static power is determined as follows [61,62]:

$$P_{st} = I_{leak} V_{dd}. \tag{9}$$

The dynamic power P_{dyn} is connected mostly with the charging and discharging of the capacitor C_{par}. It is determined by the following expression [63]:

$$P_{dyn} = \alpha C_{par} V_{dd}^2 f_{op}. \quad (10)$$

In (10), the symbol α stands for a switching activity, f_{op} is an operating frequency.

Up to this point, we have analyzed 64 articles and monographs. Summarizing the analysis of these sources, we can list some reasons showing the importance of reducing power consumption. They are the following:

1. A lot of devices are mobile and/or autonomous. They receive energy from batteries. To prolong the lifetime of these devices, it is necessary to consume as little energy as possible. If we diminish the power consumption, then we reduce the degree of heating of a chip. In turn, we are able to use smaller power supplies and reduce heat-dissipation overhead. Most importantly, it reduces the cost, weight, and size of devices. This is especially important when implementing embedded systems [64].
2. The lower the operating temperature, the higher the reliability and the longer the lifetime of the device. As shown in [1], the device failure rates are increased by up to a factor of two, if there is a 15 degree Celsius rise in temperature. Thus, the heat dissipation should be reduced to make CMOS-based systems more reliable.
3. The improvement of CMOS technology results in a growth of the on-chip transistor densities and in diminishing the delay. Unfortunately, it results in a technology-imposed utilization wall: only a fraction of an FPGA chip can be used at full speed within a power budget.
4. It is known that information and telecommunications technology contribute around 3% to the overall carbon footprint [65]. Thus, to contribute to the green computing, it is necessary to diminish the power consumption of digital systems.

All these factors should be taken into account in the process of FPGA-based FSMs' design. To achieve this, it is necessary to have efficient methods for reducing the power consumption represented by (9)–(10). How can it be done?

As follows from (9), the static power is determined by technology. It is shown in [66] that the value of P_{st} increases drastically with CMOS scaling. The higher the FPGA chip density is, the higher the value of P_{st} is. Obviously, within a certain technology, the static power consumption of VLSI-based FSM circuit could be decreased by reducing the chip area occupied by an FSM circuit. Thus, it is necessary to reduce the quantity of internal occupied resources (IORs) used by an FSM circuit. It means that it is necessary to improve methods of IOR optimization used in VLSI-based FSM design.

The analysis of (10) shows that the value of P_{dyn} can be reduced by reducing the value of C_{par}. This can be achieved simply by improving the semiconductor technology. Next, the reducing supply voltage V_{dd} significantly diminishes the value of P_{dyn}, but it diminishes the possible operating frequency of an FSM circuit. Reducing the value of f_{op} also leads to a decrease in P_{dyn}. But very often, there is a deadline for producing FSM outputs $y_n \in Y$. For example, it is very important for real-time embedded systems [14,67,68]. An FSM is a part of some digital system including various operational blocks. The lower the operating frequency, the more time is required by a system to fulfill a specific task. The system's consumed energy depends on the time of system operation. It means that reducing the f_{op} of an FSM can increase the overall power consumption of a digital system.

To control the values of V_{dd} and f_{op}, various methods of dynamic voltage and frequency scaling (DVFS) can be used [3]. Also, it can be done using low-power modes. These methods belong to a group of power mode management (PMM) sometimes named dynamic power management (DPM) [69].

Thus, only a parameter whose value can be changed due to the synthesis strategy represents a switching activity. To minimize the value of α, various methods of state assignment can be used [70]. We discuss them a bit later.

The analysis of the literature allows us to classify the known RPC methods. This classification is shown in Figure 3.

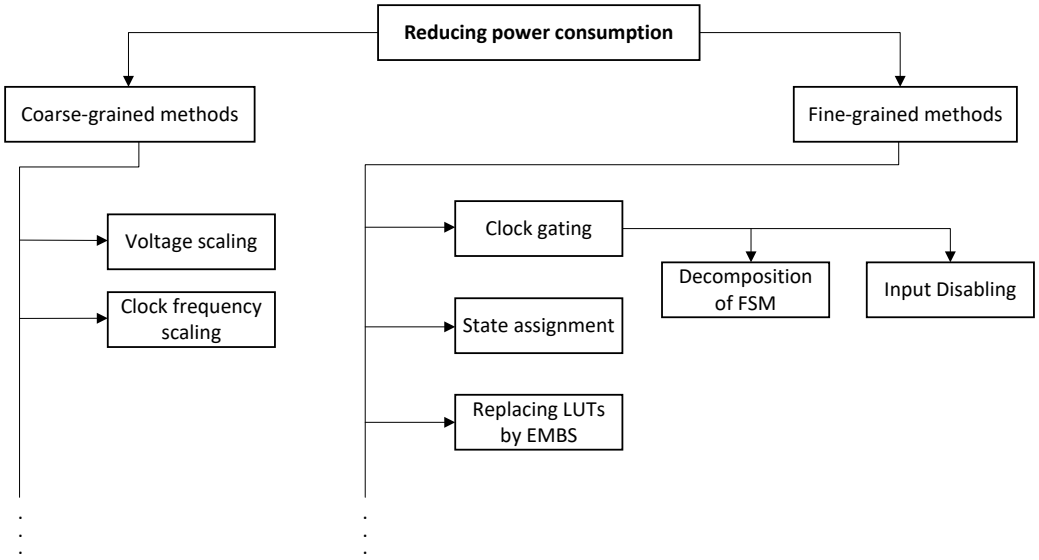

Figure 3. Classification of reduced power consumption methods.

We have divided RPC methods into two groups. The coarse-grained methods (CGMs) are the same for any block of a digital system. The fine-grained methods (FGMs) take into account specifics of a particular block. As a rule, all these methods assume the presence of some additional block providing the RPC (Figure 4).

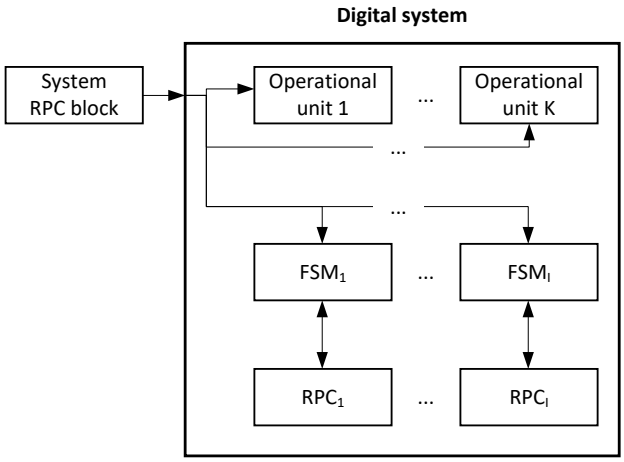

Figure 4. Providing RPC for a digital system.

The system's RPC block executes rules of DVFS accepted in a particular digital system. It could be either the voltage scaling or clock frequency scaling or both. For example, the value of V_{dd} can be reduced for any operational or sequential block such as FSM. Also, either V_{dd} or *Clock* could be cut off a particular block. The system's RPC can replace the GND voltage by some other voltage to reduce the values of leakage currents.

Obviously, if either $V_{dd} = 0$ or $f_{op} = 0$, then $P_{dyn} = 0$. This follows from (10). From (9), diminishing the value of I_{leak} leads to reducing the value of P_{st}. This is a positive effect of DVFS. But this approach also has two negative effects [69]. Firstly, to implement the system's RPC block, it is necessary to use some IORs. Thus, this block requires an additional chip area. Also, the block consumes some power and adds to the system's latency time. If $f_{op} = 0$, then an FSM is in the idle mode (it is "sleeping"). To "wake up" an FSM, it is

necessary to start the clock generator. In turn, the generator takes some time to stabilize the operating frequency. An increase in the latency time is the second negative effect of DVFS.

Therefore, DVFS is connected with a so-called power overhead [70]. The power overhead includes the three following components: the extra chip area, additional power consumption, and increased latency time. It is necessary to find a reasonable trade-off between the inevitable overhead and the required characteristics of a digital system. We do not discuss these methods in this paper.

As follows from Figure 3, there are three groups of fine-grained methods of RPC. The clock-gating (CG) approach is connected with interrupting connections between the clock generator and synchronization inputs of flip-flops. There are two approaches based on CG: (1) the decomposition of an FSM and (2) the input disability. This approach is connected with using additional blocks $RPC1 - RPCI$ to control the timing of automata $FSM1 - FSMI$ (Figure 4). Thus, this approach is connected with an RPC overhead.

The second group of FGMs consists of special methods of state assignment. The states $a_m \in A$ are encoded in a way that reduces the value of switching activity α. From (10), this reduces the value of P_{dyn} (if C_{par}, V_{dd}, and f_{op} have constant values). Sometimes, this leads to increasing the value of the bit depth of state codes, R, compared to its minimum value determined by (4). This growth is the RPC overhead for this group of FGM.

The third group is based on the replacement of LUTs by EMBs. In fact, we are moving from fine-grained LUTs to coarse-grained EMBs. As follows from Figure 5, some group consisting of four LUTs and their interconnections is replaced by a single EMB.

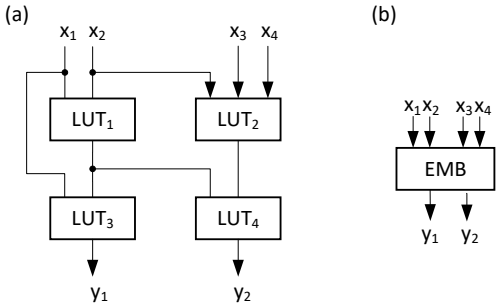

Figure 5. Replacing four LUTs by a single EMB.

The circuit (Figure 5a) consists of 4 LUTs and 11 interconnections. It is replaced by the circuit having a single EMB and six interconnections (Figure 5b). These interconnections correspond to inputs and outputs of this circuit. There are no additional interconnections which can be found in the LUT-based circuit (Figure 5a). Obviously, the EMB-based circuit has better area, time, and power characteristics than the equivalent LUT-based circuit. It does not require any power overhead. But this approach has two limitations. First, it can be used if there are "free" EMBs (very often, EMBs are used for implementing operational blocks of a system). Second, an EMB can be used if the number of arguments of an SBF does not exceed the number of address inputs, S_A.

Now, we discuss the most known fine-grained methods of reducing power consumption in the next three Sections of this survey. These methods are the clock-gating, FSM decomposition, state assignment restricting the switching activity, and replacing LUTs by EMBs.

4. Saving Power by Clock-Gating and FSM Decomposition

In Mealy FSMs, outputs $y_n \in Y$ are unstable [39]. Because outputs $y_n \in Y$ depend on inputs $x_l \in X$, then changing inputs during the clock cycle may cause short-term changes in outputs (glitches). This may cause a malfunction of a digital system. To stabilize the outputs, it is sufficient to stabilize the FSM inputs. This can be achieved by entering a special register RGX as shown in Figure 6. This figure also depicts the interaction of an FSM with other digital system blocks.

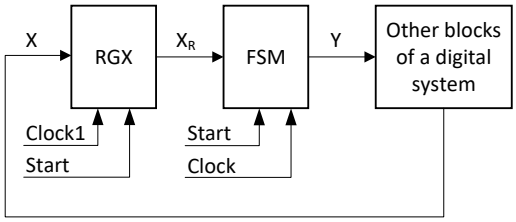

Figure 6. Interaction of an FSM with other blocks of a digital system.

To generate correct output values, an FSM should analyze the outputs of other blocks. They form the set $X = \{x_1, \ldots, x_L\}$. When values of the inputs are correct, the pulse *Clock1* is generated. The values of FSM inputs are loaded into *RGX*. Now, they correspond to registered inputs from a set X_R. The elements of X_R are stable during the cycle of *Clock*. Thus, an FSM generates the following SBF:

$$Y = Y(T,\ X_R). \tag{11}$$

Now, the outputs are stable after the completion of various transients in the FSM circuit.

Let us point out that there is no need of RGX if the model of a Moore FSM is used. This is connected with the nature of outputs (3). From (3), there is no direct dependence between the inputs and outputs of a Moore FSM. The outputs depend only on states. Thus, the outputs are registered. The state register outputs (state variables) are stable during each cycle of operation. Therefore, if some input is changed between two pulses of synchronization, the outputs are unchangeable.

Thus, in reality, there are two registers in the circuits of Mealy FSMs. The register *RG* includes *R* flip-flops, the register *RGX* consists of *L* flip-flops. These registers are synchronized by different pulses (Figure 7).

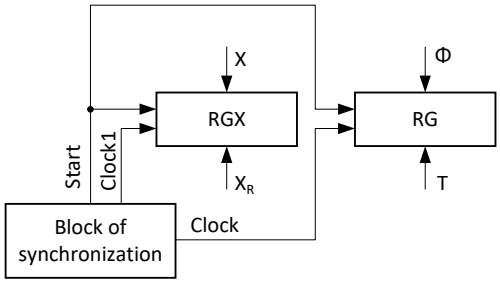

Figure 7. Registers of a Mealy FSM.

The pulses *Clock1* and *Clock* are generated by the special block of synchronization. This block contains a quartz generator, delay circuit, and a single vibrator generating the pulse *Start*. It is known that clock trees usually consumes up to 50% of the dynamic power [71]. The internal switching power of flip-flops is responsible for 45–50% of the clock tree's power consumption [72]. As a result, it is very important to deliver synchronization pulses only to flip-flops whose states will be changed in a particular cycle of FSM operation. This can be achieved by using the clock-gating approach.

CG assumes using an additional clock logic (CL) block [3]. This logic is based on the precomputation of inputs being disabled [73,74]. In this case, some precomputation logic is added to the CL. It analyzes inputs and state codes to disable the loading of all or a subset of flip-flops of RGX (Figure 8).

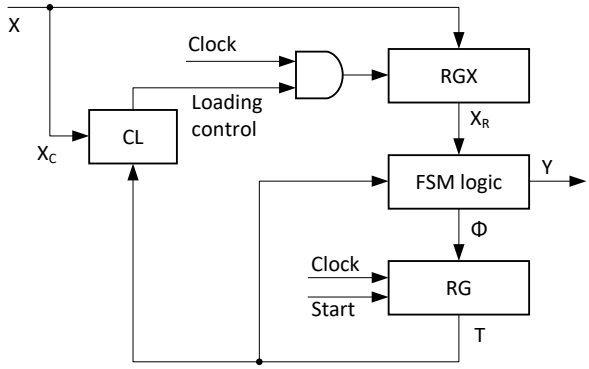

Figure 8. Organization of clock gating.

The CL generates loading control signals as functions of $X_C \subseteq X$ and state variables. These signals either allow or prevent the passage of *Clock1* to inputs of synchronization of flip-flops creating RGX.

It is very important to choose the subset of X which enters the CL. The smaller the difference $|X| - |X_C|$, the higher the probability that the CL is active. It leads to reducing the power consumption of both RGX and FSM logic block. Of course, this is connected with a CL-based overhead: this block requires some chip area, consumes additional power, and increases the FSM cycle duration. Thus, it is very important to find a set $X_C \subseteq X$ that reduces the negative influence of the CL and provides the minimum power consumption of an FSM circuit.

As noted in the monograph [75], 20% of program operators are responsible for 80% of the program execution time. The same may be true for FSM states. If a state $a_m \in A$ is a waiting state, then an FSM may remain in that state for a long time. If a state register consists of D flip-flops, then the code $K(a_m)$ should be reloaded during a lot of clock cycles. Based on a similar analysis, the model of a gated-clock FSM was proposed [76].

In [76], the waiting state is named a self-loop. If an FSM enters a self-loop, then a special logic makes the pulse Clock off. Therefore, in that case, the CL controls the state register RG (Figure 9).

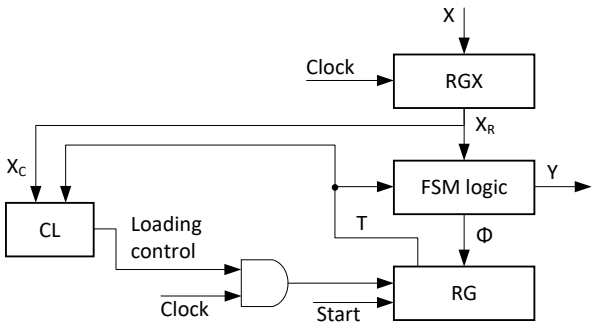

Figure 9. Organization of a clock-gating FSM [76].

A comparison of Figures 8 and 9 shows that these approaches are very similar. They have the same positive and negative features. These methods can be used simultaneously. Mostly, these two methods are used together with FSM decomposition [77].

The first work devoted to FSM decomposition appeared in 1960 [78]. There are three known basic approaches of decomposition: parallel, cascade, and general [79]. These approaches are shown in Figure 10.

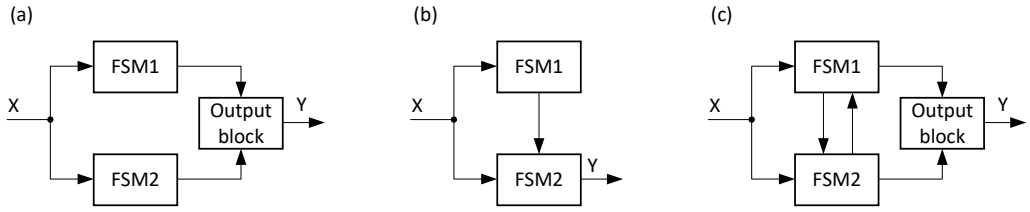

Figure 10. Three approaches for FSM decomposition [79].

Both methods of parallel (Figure 10a) and cascade (Figure 10b) decomposition have rather theoretical value [3]. But the general decomposition (Figure 10c) can be used for any FSM. This approach was used for implementing PLA-based FSMs [80,81].

Let us discuss the FSM architecture based on the general decomposition. The FSM circuit includes three combinational blocks and two registers keeping the state codes of different FSMs (Figure 11).

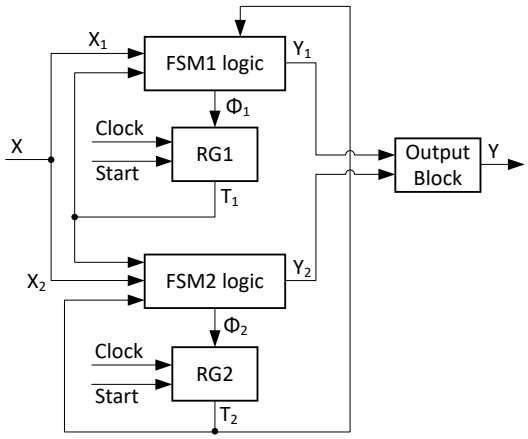

Figure 11. Structural diagram of an FSM based on general decomposition.

The set A is decomposed by two disjoint sets, A^1 and A^2. The states $a_m \in A^1$ are encoded using $R1$ state variables, which form a set $T1$. The value of $R1$ is determined by $R1 = \lceil log_2 |A^1| \rceil$. The states $a_m \in A^2$ are encoded using $R2$ state variables, which form a set $T2$. The value of $R2$ is determined by $R2 = \lceil log_2 |A^2| \rceil$. There are $R1$ elements in the set of IMFs Φ_1; there are $R2$ elements in the set of IMFs Φ_2. Both registers have the same pulses *Start* and *Clock*. The set of FSM inputs is represented as $X = X1 \bigcup X2$. It is quite possible to have identical elements in these sets.

As follows from Figure 11, the following SBFs should be implemented:

$$\begin{aligned} \Phi_1 &= \Phi_1(T1,\ T2,\ X1); \\ \Phi_2 &= \Phi_2(T1,\ T2,\ X2); \\ Y_1 &= Y1(T1,\ T2,\ X1); \\ Y_2 &= Y2(T1,\ T2,\ X2); \\ Y &= Y(Y1,\ Y2). \end{aligned} \tag{12}$$

For FSMs based on (12), the following design method is proposed in [3]:

1. Select disjoint subsets A^1 and A^2.
2. Generate STGs for each sub-FSM. Add additional RESET states into each STG.
3. Copy all transitions from the initial STG in unmodified form into new STGs.
4. Replace the transitions $\langle a_m,\ a_s \rangle$ where $a_m \in A^1$ and $a_s \in A^2$ by the two following transitions: $\langle a_m,\ RESET2 \rangle$ and $\langle RESET2,\ a_s \rangle$.

5. Replacing the transitions $\langle a_m, a_s \rangle$ where $a_m \in A^2$ and $a_s \in A^1$ by the two following transitions: $\langle a_m, RESET1 \rangle$ and $\langle RESET1, a_s \rangle$.

In [3], this approach is combined with clock gating for both inputs and state registers. This combines approaches from [76,77] with some new approach. The initial FSM is divided into two FSMs: FSM1 and FSM2.

FSM1 is small; it includes states $a_m \in A^1$ with very high probabilities of transitions $\langle a_m, a_s \rangle$ where $a_s \in A^1$. This FSM corresponds to the famous 20% of operators determined by [75]. All other initial FSM states belong to the set A^2. FSM1 is mostly active, and FSM2 is mostly idle. As a result, it is possible to disable the flip-flops of RG2 and RGX for FSM2. If FSM1 is idle, its state register RG1 can be disabled too. This idea leads to the structural diagram shown in Figure 12.

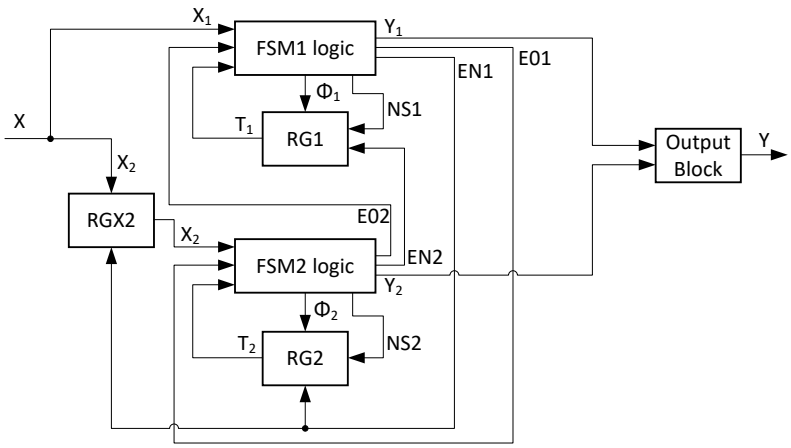

Figure 12. Structural diagram of a decomposed FSM [75].

The following sets can be obtained from Figure 12: sets of inputs $X1$ and $X2$ which can have common elements; sets of outputs $Y1$ and $Y2$ which can be disjoint; disjoint sets of IMFs $\Phi 1$ and $\Phi 2$; disjoint sets of state variables $T1$ and $T2$; sets of internal control signals (ICSs) $SC1$ and $SC2$. These last sets are the following: $CS1 = \{EO1, EN1, NS1\}$ and $CS2 = \{EO2, EN2, NS2\}$. Using the ICS EN, FSM1 may disable both RGX2 and RG2. The signal $EO1$ determines the required state of FSM2. The same function is executed by FSM1 using outputs $EO2$. The signals EN1 and EN2 disable the registers of FSM2 and FSM1, respectively. Also, the signal EN1 disables the loading of inputs $x_l \in X2$ into $RGX2$.

In [3], the authors show results of experiments conducted using the CAD system SIS [36] and library [82]. The results show that the "impressive power savings correspond to larger FSMs (for example, 79.5% for the benchmark `planet`)". There is no gain for small FSMs. This can be explained by adding some circuitry and two extra states. For example, around 30% of the area is added to the FSM circuit implementing the benchmark `planet`.

Each FSM of a decomposed circuit can be treated as a superstate (SS). For example, the structural diagram from Figure 12 corresponds to the STG shown in Figure 13.

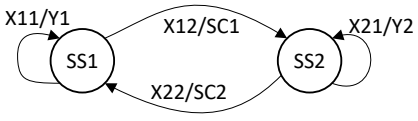

Figure 13. State transition graph with two superstates.

We divided the sets $X1, X2 \subseteq X$ into two subsets each. For example, the set $X11 \subseteq X1$ causes transitions inside FSM1 with the generation of outputs $y_n \in Y1$. The set $X12 \subseteq X1$ causes transitions into the RESET state of FSM2. These transitions are accompanied with

the generation of ICSs from set $SC1$. Because the transitions are determined by FSM states and inputs, then it makes sense to use clock gating for both states and inputs.

As shown in [70], the approach similar to [71] has two drawbacks:

1. The decomposed network always includes only two FSMs.
2. The blocks of clock logic are synchronized, and *Clock* pulses enter these blocks. As a result, the CL blocks consume a lot of power.

In [70], an approach is proposed that has the following advantages:

1. The decomposed FSM is represented by K submachines FSM1, ..., FSMK, where $K \geq 2$.
2. The blocks of clock logic are asynchronous.

Summarizing the results [70], it is possible to represent an FSM as a network including K interrelated partial FSMs. Each of them includes its own synchronization circuit of CL. The circuits $CL_1 - CL_K$ are interrelated (Figure 14).

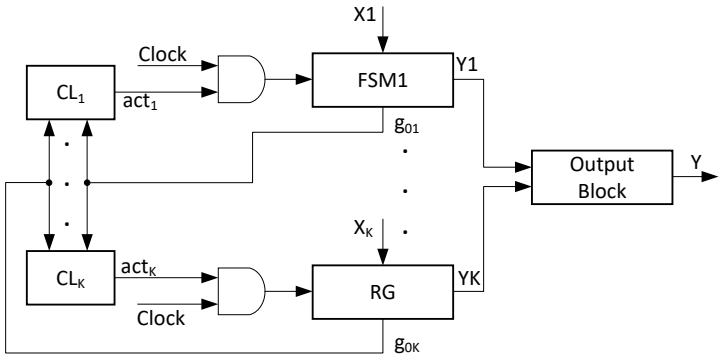

Figure 14. Decomposition of an FSM based on [70].

Special signals $g_{0k}(k \in \{1, \ldots, K\})$ point to the machine FSMk that should be active in the next cycle of *Clock*. Using this signal, blocks CL_k generate signals $act_k(k \in \{1, \ldots, K\})$. Only one of these signals are equal to one. This determines the particular active sub-FSM. The ABD gates help to implement the synchronization for sub-FSMs:

$$C_k = act_k \& Clock \ (k \in \{1, \ldots, K\}). \tag{13}$$

In [71], the clock signal enters the circuits of clock logic. This is the signal with the highest switching activity among all other signals such as FSM inputs, outputs, and state variables. If the pulse *Clock* enters the circuits of CL, then the power consumption is increased as compared with the case discussed in [70].

As shown in [70], there are three operating modes for clock logic blocks. During a transition between different sub-FSMs (hand-over mode), all CL blocks are active. In this mode, the maximum amount of power is consumed by these blocks. If $act_k = 0$, then the block CL_k is in the disable mode. It means that the power is consumed only by AND gate. The third mode is connected with enabling the block CL_k ($act_k = 1$). The circuits of clock logic are passive; no power is consumed. Of course, switching AND gates requires some power.

The asynchronous approach allows a significant saving in power consumption compared with the synchronous approach. As shown in [70], the power consumption is 1.36 times less for the hand-over mode, 4.13 times less for the enable mode, and 5.9 times less for the disable mode. Also, the difference in power consumption is greater (for different modes).

In [70], experimental results are shown based on the use of the proposed approach for benchmarks `bbara`, `dk512`, `ex1`, `keyb`, `styr`, `donfile`, `tma`, and `scf`. They show that for the rather simple benchmark `dk512`, the value of $K = 3$ provides the best solution. At the same time, the best result for the not too complex benchmark `ex1` is connected with $K = 4$. The

most complex benchmark is scf ($M = 121$, $L = 27$, $N = 56$). But the best solution for this benchmark is provided by splitting it into only two interrelated FSMs ($K = 2$). The same occurs for the simplest FSM represented by the benchmark bbara, having the following characteristics: $M = 10$, $L = 4$, and $N = 2$. Thus, the optimal value of K does not depend on the number of states, M, or inputs, L, or outputs, N. The results in [70] show that the optimal value of partial FSMs, K, depends on the probabilities of interstate transitions.

Also, the results in [70] indicate that saving power is connected with the overhead. The conclusion is the same as for other discussed methods: the more complex an original FSM is, the smaller the relative overhead area added. The same is true for the decomposed FSM performance: the more complex the original FSM is, the smaller the impact of additional circuitry on the performance is.

5. Saving Power by State Assignment

A huge number of state assignment methods are known. Some of them are aimed solely at power consumption reduction. But if some method minimizes the chip area occupied by an FSM circuit, then this method minimizes the static power consumption too. Due to this fact, we do not separate these two groups of methods. To prepare this part of our survey, we used the following sources: [83–111]. Of course, this is only the tip of the iceberg, but the generalization of these methods gives a general idea of executing RPC through the state assignment.

The power consumption depends significantly on the chip area occupied by an FSM circuit. This was proven, for example, in [109]. In [109], four different state assignment approaches are investigated: binary (with $R = \lceil log_2 M \rceil$), one-hot (with $R = M$), two-hot, and JEDI (the output-dominated version).

In the case of two-hot state assignment, no more than two code bits can be equal to one, simultaneously. This allows the use of less than M bits for the state codes. If $M = 6$, for example, then three bits are required to encode the states. The following codes are used: 001, 010, 100, 101, 011, and 110. This gives the same value of R as it is for the binary state assignment. But if $M = 7$, then four bits are necessary to create two-hot state codes. This is less than for the one-hot approach ($R = 7$), but more than for the binary approach ($R = 3$).

In [109], the benchmarks from [82] were used. The benchmarks were represented in the KISS format. The FPGA Express by Synopsis and Xilinx Foundation Tools F3 were used to obtain FSM circuits. The KISS files were transformed into a VHDL-based representation. To obtain the circuits' characteristics, the authors used the following FPGA sample: XC401EPC84-1. The characteristics were measured using the following operating frequencies: 100 Hz, 2 MHz, and 8 MHz. $S_L = 4$ was used for the FPGAs of XC401E/XL [112].

The occupied area was measured as a number of CLBs. This approach is still used nowadays [113]. Sometimes, the number of used flip-flops is added to the number of CLBs. Some results of the investigations in [82] are shown in Table 1.

We selected the results of experiments for five benchmarks having a wide range of characteristics. The last row of Table 1 includes the numbers of inputs, L, outputs, N, and states, M, of particular FSMs. The number of state variables, R, is the same as the number of flip-flops. It was taken from the reports generated by the CAD tools.

As follows from Table 1, the number of inputs influences significantly the area characteristics. For example, practically the same value of R is obtained for benchmarks ex4 and kirkman. But there are two times more CLBs in the circuit for kirkman. This is connected with the significant difference in the values of L for these two benchmarks.

The following conclusion is made in [109]: "For FSMs with up to 8 states, the binary encoding must be used. For FSMs with more than 16 states, the one-hot is always the best choice". We think this is true if FSMs have the same number of inputs $x_l \in X$. Otherwise, a lot depends on the value of $L + R$.

Table 1. Results of experiments from [82].

Characteristics	Method	bbara	dk512	ex4	kirkman	planet
Area (CLB + FF)	Binary	11 + 3	14 + 4	22 + 4	45 + 4	113 + 6
	One-hot	8 + 7	10 + 14	15 + 14	43 + 16	65 + 48
	JEDI	10 + 3	9 + 4	19 + 4	45 + 4	106 + 6
	Two-hot	15 + 4	16 + 5	18 + 5	57 + 5	99 + 10
Delay (ns)	Binary	30.0	20.8	31.2	38.3	60.6
	One-hot	25.6	20.4	29.4	36.2	41.3
	JEDI	29.4	26.0	27.0	38.9	54.3
	Two-hot	31.2	23.9	27.2	36.6	61.1
Power (mW/MHz)	Binary	1.39	2.46	2.51	4.14	14.4
	One-hot	1.38	1.54	1.66	4.00	6.23
	JEDI	1.87	1.85	2.10	3.73	13.2
	Two-hot	1.46	2.48	2.11	5.21	11.7
L + N + M		4 + 2 + 7	1 + 3 + 15	6 + 9 + 14	12 + 6 + 16	7 + 19 + 48

It is very interesting that "for any state encoding, the power is linearly correlated with the number of states. The coefficient of correlation is over 0.85" [109]. The same is true for the relationship "number of states-area".

Also, there is a very important conclusion made in [109]: "between area and power, there is the coefficient of correlation 0.91". It is shown in [109] that "the 77% of smaller circuits consume lower power". The results of [109] show that "area, time and power consumption correlation with other FSM parameters (inputs, outputs and states) and combinations of these parameters neither produce significant results".

As shown in [109], the proper state assignment can give up to 57% of power saving. Of course, this is true only for the investigated benchmarks and that particular FPGA chip. The saving amount can be different for a different suite. It is interesting that the discussed methods do not use probabilities of interstate transitions. If we take them into account, we can reduce the switching activity, α. To do it, some special state assignment methods are used.

One of the first algorithms decreasing the switching activity was proposed in [85]. It targets a state assignment that minimizes the switching activity and takes into account the issue of area. Due to this integral approach, both types of power consumption, static and dynamic, are optimized. The method is based on a probabilistic description of FSMs.

The method uses an average switching activity to find the switching (transition) probability. This allows the obtainment of the probabilities of FSM interstate transitions. This information is used for executing the state assignment. But to do it, it is necessary to have the input switching probabilities. In [85], an STG is modeled as a Markov chain [114]. The Markov chain model describes an STG as a directed graph with weighted edges and a structure isomorphic to the initial STG. The STG is transformed into a weighted undirected graph. The weight of each edge is proportional to the total probability of a transition between FSM states $a_m, a_s \in A$ connected by this edge. This final STG is used as the initial information for the state assignment step.

The main idea of [85] is "to find a state assignment that minimizes the number of state variables that change their values when the FSM moves between two adjacent states". In the best case, only a single state variable is changed, as it is for Gray codes. But $M/2$ state variables are necessary for the Gray state assignment. The aim of [85] was to find the value of state code bits close to the minimum value defined by (4).

A state encoding is represented by a Boolean matrix. Its rows correspond to state codes and columns to state variables. The required state assignment can be found by the solution of the integer linear programming problem formulated in [85].

For small FSMs, it is possible to find the exact solution. For complex FSMs, only a suboptimal solution can be found because the problem is NP-complete. Thus, there are a lot of heuristic algorithms for its solution [90,91,115]. In [85], the column-based approach

is used. In this case, each state variable corresponds to a column. The method includes R iterations, when each state variable $T_r \in T$ receives either zero or one. The assignment is done in a way that minimizes the switching activity. The algorithm tries to minimize the number of different values of state variables for states with the highest switching probabilities. The algorithm produces a semi-exact solution.

To minimize the chip area, some additional constraints are used. Additional metrics are used, similar to the ones proposed in [90]. One of them is a fan-out-oriented metric. It can be used for FSMs with a small number of inputs and a large number of outputs. The second metric is a fan-in-oriented metric. It can be used for FSMs with a large number of inputs and a small number of outputs. These area constraints are added as weights for an STG.

To reach some trade-off for area–power, the parameter $\alpha \leq 1$ is introduced in [85]. It shows what is more important for a given task. The weight $\omega_{m,s}$ of an edge connecting states a_m and a_s is determined by

$$\omega_{m,s} = (1 - \alpha)\omega_{m,s}^{area} + \alpha\omega_{m,s}^{power}. \quad (14)$$

The weights for area ($\omega_{m,s}^{area}$) are determined using the MUSTANG approach [90]. The weights for power ($\omega_{m,s}^{power}$) are determined by the heuristic algorithm from [85].

The results of experiments with the benchmarks from [82] show that the saving power increases with the growth in FSM complexity. If $R = 4$, then the maximum saving is up to 8%; if $R = 5$, the maximum saving is up to 16%; if $R = 6$, then the maximum saving is up to 25%. Thus, adding one to R improves the power consumption by approximately 8%. Also, the growth in the number of state variables, R, leads to reducing the area overhead. It means that applying similar approaches makes sense for rather complex FSMs.

Using [85] allows the creation of the block diagram shown in Figure 15. In this algorithm, we assume that pairs $P1, ..., PI$ are created for FSM states. These pairs include states $a_m, a_s \in A$ such that there is at least a single transition between these states (either $\langle a_m, a_s \rangle$ or $\langle a_s, a_m \rangle$). Each pair has a weight $W(Pi)$. The block diagram is shown in Figure 15.

In the beginning, it is necessary to organize a queue γ of pairs $P1, ..., PI$. The pairs are placed in the queue in the descending order of weights $W(Pi)$. The algorithm has no more than I steps.

Every step is connected with the following operations. The ith pair is selected from the queue γ (block 3). The pair includes some states a_m and a_s. If state a_m has no code (the output "No" from block 4), then we check whether state a_s has a code (block 5). If there is a preliminary selected code $K(a_s)$ (the output "Yes" from block 5), then we should select the best possible code for state a_m (block 7). The best possible code is selected from still "free" state assignments. The best code should have a minimum possible Hamming distance (HD) with the code $K(a_s)$. Next, the value of i is increased by one (block 9). If the queue is not empty, then the selection process is repeated (the transition to block 3). Otherwise, the process is terminated. If there is still no code for state a_s (the output "No" from block 5), then the selection of the best codes for both states is executed.

If $K(a_m)$ already exists (the output "Yes" from block 4), then we check for code $K(a_s)$ (block 6). If there is no code $K(a_s)$ (the output "No" from block 6), then a possible best code $K(a_s)$ is selected (block 8). Otherwise, no codes should be selected, and the process is repeated (go to block 9).

Consider the following example. Consider the set $A = \{a_1, \ldots, a_5\}$ for some FSM S_1. These states form the following I = 9 pairs: $P1 = \langle a_3, a_4 \rangle$, $P2 = \langle a_2, a_4 \rangle$, $P3 = \langle a_3, a_2 \rangle$, $P4 = \langle a_1, a_5 \rangle$, $P5 = \langle a_1, a_2 \rangle$, $P6 = \langle a_4, a_1 \rangle$, $P7 = \langle a_2, a_5 \rangle$, $P8 = \langle a_3, a_3 \rangle$, and $P9 = \langle a_5, a_4 \rangle$. R = 3 and $K(a_1) = 000$. Now, we should find the best state codes using the algorithm shown in Figure 15. The process of state assignment is shown in Figure 16.

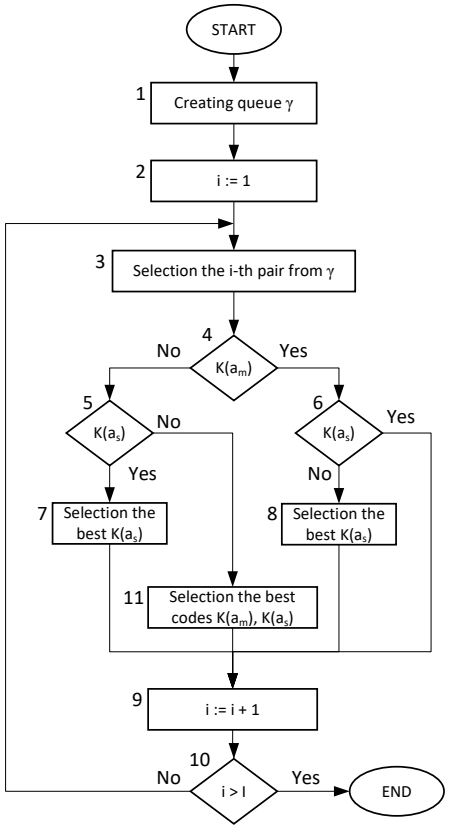

Figure 15. Block diagram of algorithm [85].

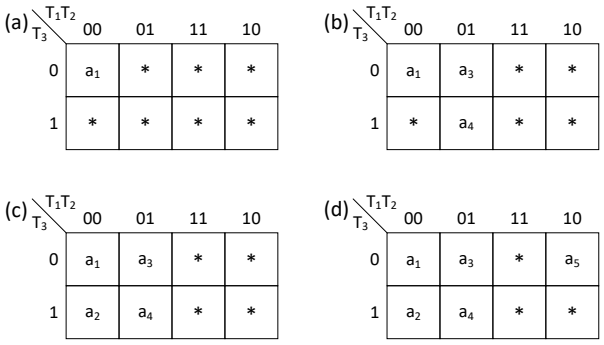

Figure 16. Process of state assignment for saving power.

The start point of the state assignment process is shown in Figure 16a. The code 000 is assigned to the initial state $a_1 \in A$. All other cells of the Karnaugh map contain the asterisk signs.

Step 1. The first pair of the queue is selected. This the pair $P1 = \langle a_3, a_4 \rangle$. To encode the states $a_3, a_4 \in P1$, we should select codes with a minimum number of ones, and $HD = 1$. This means, the actions from block 11 are executed. The codes of $a_3, a_4 \in P1$ are shown in Figure 16b.

Step 2. Now, the pair $P2 = \langle a_2, a_4 \rangle$ is selected. Because the transition state $a_4 \in P2$ is already encoded, the action from block 7 is executed. The best possible solution is shown in Figure 16c.

Step 3. The pair $P3 = \langle a_3, a_2 \rangle$ is selected. Both states from this pair are encoded. Thus, no new codes are assigned during this step.

Step 4. The pair $P4 = \langle a_1, a_5 \rangle$ is selected. Now, the code for $a_5 \in A$ should be selected. This corresponds to block 8. The final solution is shown in Figure 16d.

This algorithm belongs to the group of "greedy" algorithms. It makes the optimal choice at each step. The algorithm does not change already selected codes. As a result, such a solution is not optimal. It can be improved.

Consider what causes the overhead for this algorithm. For example, an STG includes the subgraph shown in Figure 17a.

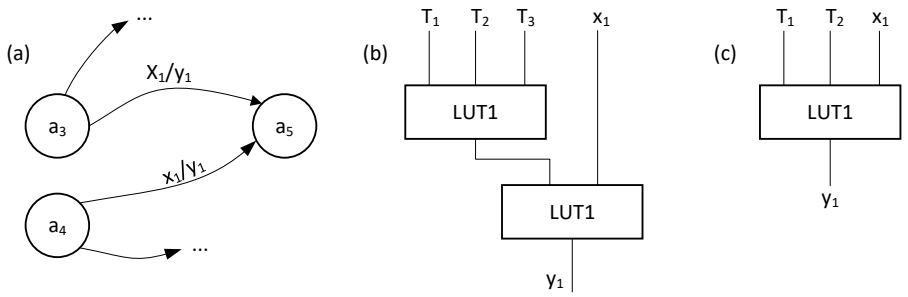

Figure 17. Explanation of power overhead for algorithm [85].

If $R = 3$, then the algorithm [85] will select the codes $K(a_3) = 001$ and $K(a_5) = 011$. Because the code $K(a_5)$ is now fixed, there is a limited choice of possible codes for a_4. To provide HD = 1, the algorithm can assign the code 010 to a_4. These codes provide the best solution from the greedy algorithm's point of view.

But this approach does not take into account LUT counts for the output logic. As follows from Figure 17a, $y_1 = A_3x_1 \bigvee A_4x_1 = \overline{T_1}\overline{T_2}T_3x_1 \vee \overline{T_1}T_2\overline{T_3}x_1$. If an FSM circuit is implemented with LUTs having three inputs, then two LUTs are necessary to implement the circuit for y_1 (Figure 17b). This circuit has two levels of LUTs and six interconnections.

If a JEDI-based style of state assignment is used, then the states a_3 and a_4 will have adjacent codes. If, for example, there is $K(a_3) = 001$ and $K(a_4) = 101$, then it gives the Boolean equation $y_1 = \overline{T_2}T_3x_1$. The corresponding circuit has only a single LUT, a single level of logic, and four interconnections (Figure 17c).

Thus, for the discussed case, the JEDI-based circuit is faster and requires fewer LUTs. This is quite possible that the circuit from Figure 17c has better power characteristics than its equivalent shown in Figure 17b.

The following conclusion can be made from this example. To find a desirable trade-off among power consumption, area, and performance, it is necessary to take into account the output logic too. There are special resynthesis methods [116] that can improve the overall quality of an FSM circuit. They are out the scope of this survey.

Let us only point out that the resynthesis allows a reduction in the number of logic levels and simplifies the interconnection system. These two issues are very important in the LUT-based design [116].

As mentioned in [117], many systems of emerging computing and communications equipment are control-dominated. The controllers are mostly implemented as FSMs. Because many devices are mobile, then a RPC is a very important issue. In the case of controllers, it is very important to decrease the power consumption, because "controllers are always active. As a result, a good amount of system power is consumed by the controllers" [117]. This explains the necessity of RPC for FSMs.

The RPC can be achieved by adding states [95] or adding bits to state codes [108]. These additional states and bits can also be viewed as the power overhead.

We start from the state splitting approach [95]. The approach takes its roots in 1963 [118]. This approach was used for optimizing FSMs [119] and minimizing the number of FSM states [120]. Also, it can improve the power consumption.

Consider a part of an STG (Figure 18a) taken from [95]. The state codes are shown near the graph nodes.

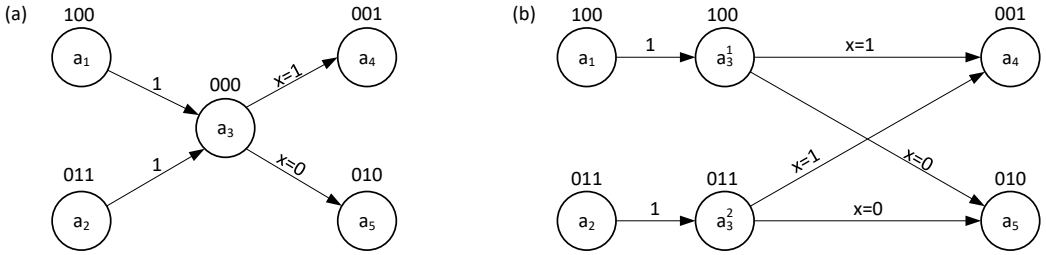

Figure 18. Initial STG (**a**) and its transformation (**b**).

To reduce the switching activity, α, it is necessary to diminish the value of the Hamming distance. In the best case, HD = 1 for all pairs $\langle a_m, a_s \rangle$ existing in a particular STG. Obviously, each state code can have only R adjacent state codes with HD = 1. The state a_3 (Figure 18a) is connected with R + 1 = 4 states. HD = 1 for the pairs $\langle a_1, a_3 \rangle$, $\langle a_3, a_4 \rangle$, and $\langle a_3, a_5 \rangle$. But HD = 2 for the pair $\langle a_2, a_3 \rangle$.

It is possible to "split" the state a_3 into two equivalent states a_3^1 and a_3^2. These states have the same transitions. Now, each of the new states (a_3^1 and a_3^2) has exactly R adjacent states. If the state codes are the ones shown in Figure 18b, then HD = 1 for all existing adjacent state pairs ($\langle a_1, a_3^1 \rangle$, $\langle a_3^1, a_4 \rangle$, $\langle a_3^1, a_5 \rangle$, $\langle a_1, a_3^2 \rangle$,$\langle a_3^2, a_4 \rangle$, and $\langle a_3^2, a_5 \rangle$.

Each state $a_m \in A$ can be characterized by two sets. The set $FI(a_m)$ includes states $a_s \in A$, such that there are transitions $\langle a_s, a_m \rangle$. The set $FO(a_m)$ includes states $a_s \in A$, such that there are transitions $\langle a_m, a_s \rangle$. As follows from Figure 18a, the following sets can be formed: $FI(a_3) = \{a_1, a_2\}$ and $FO(a_3) = \{a_4, a_5\}$. The splitting state a_3 makes sense because the following relations hold: $|FI(a_3)| = 2 \rangle 1$ and $|FI(a_3)| + |FO(a_3)| = 4 > R = 3$. This means that the splitting can be executed if

$$(|FI(a_m)| > 1) \wedge (|FI(a_m)| + |FO(a_m)| > R) = 1. \quad (15)$$

For states of transitions, state codes $K(a_s)$ depend on the codes of previous states. As a result, there are a lot of splitting options. It is necessary to choose the option leading to the maximum RPC.

In [95], two algorithms are proposed for the state splitting. In the first case, all possible splittings are investigated for states satisfying (15). This approach requires the extensive search of an optimal solution. In the second case, only two subsets are formed for $FI(a_m)$. One subset includes a state a_s having the maximum probability of transition into the state a_m to be split. The second subset includes states $a_i \in FI(a_m)/\{a_s\}$. This solution is a suboptimal one.

In [95], experimental results are provided. They were obtained using the library [82] and the software package ZUBR [121]. The results show that a RPC takes place for 27 benchmarks (57.4% of all benchmarks).

The results [95] show that the proposed approach reduces the power consumption by an average of 6.92%. At the same time, the maximum RPC is equal to 81.02% (for the benchmark *tma*). The simplified algorithm produces solutions very close to optimal. The average difference is around 0.08%.

Consider Figure 18a. $FI(a_3) = \{a_1, a_2\}$ and $FO(a_3) = \{a_4, a_5\}$; $R = 3$. As follows from (15), it is impossible to assign adjacent codes to all states included into the set $FI(a_3) \cup FO(a_3)$. If the number of state variables is greater than what is defined by (4), then the condition (15) is violated. In this case, there is an optimum solution shown in Figure 19.

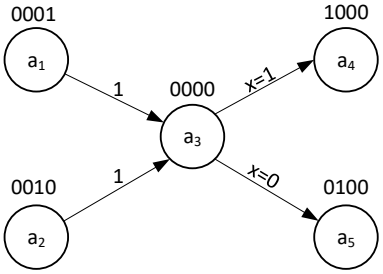

Figure 19. Optimal codes for the STG from Figure 18a.

The analysis of Figure 19 shows that the following relation holds: $HD(a_1, a_3) = HD(a_2, a_3) = HD(a_4, a_3) = HD(a_5, a_3) = 1$. The codes (Figure 19) provide the minimum power consumption for this part of the STG. But they add a power overhead since $R = 4$ instead of $R = 3$. It means that there is an additional flip-flop and additional loading for the clock tree.

A method [108] based on this idea was tested using the benchmarks from [82]. The outcomes of this approach were compared with results obtained for NOVA, JEDI, and a column algorithm [85]. The experiments were conducted for the following conditions: $V_{dd} = 5$ V, $f = 5$ MHz, $C_{par} = 3$ pF.

The approach from [108] allowed a reduction in the power consumption by a factor of 1.7 (NOVA), 1.36 (JEDI), and 1.12 (column algorithm). For example, in the case of benchmark *tbk*, adding one to the minimum value of R diminished the power consumption by 34%. Of course, it is necessary to take into account the influence of increasing the number of state variables on both area and time characteristics of a resulting FSM circuit.

All discussed methods of state assignment have the same specificity: a state code $K(a_m)$ is assigned to the state $a_m \in A$ as an R-bit string during some step of the state assignment process. In [122], a method is proposed where each state assignment step gives only a single bit of state codes.

The method [122] reduces the switching activity, α. But at the same time, it diminishes a chip area occupied by an FSM circuit. The decomposition strategy of the state assignment is proposed in [122]. The approach produces a binary tree whose leaves correspond to state codes.

As presented in [122], we explain this approach using the benchmark *dk*27 [82]. The benchmark's STG is shown in Figure 20.

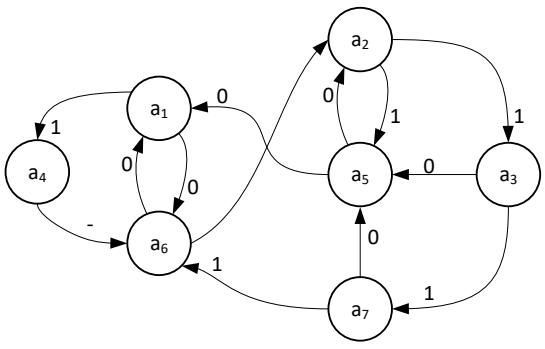

Figure 20. State transition graph of benchmark dk27 [122].

The state assignment [122] starts by calculating the probabilities $p_{m,s}$ of interstate transitions. To get the total probabilities of transitions $P_{m,s}$, the product of probabilities P_m and $p_{m,s}$ is calculated. The value of P_m determines a probability that the FSM is in the state $a_m \in A$. For *dk*27, the following values of P_m are used: $P_1 = P_2 = 0.19$, $P_3 = 0.095$, $P_4 = 0.095$, $P_5 = 0.167$, $P_6 = 0.214$, $P_7 = 0.048$.

The summation of the direct edges' probabilities for each pair of states produces an undirected graph with edge weights (Figure 21). Now, it is necessary to minimize the Hamming distance between the codes $K(a_m)$ and $K(a_s)$ with the high transition probability.

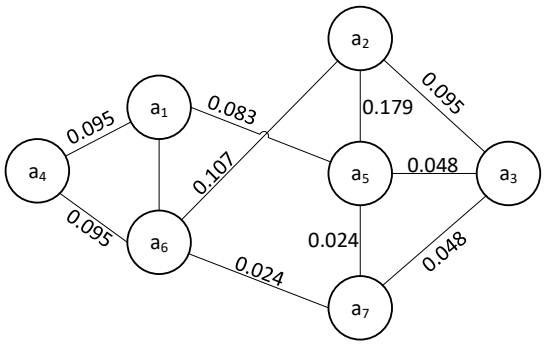

Figure 21. Undirected weighted graph for dk27 [122].

Using the algorithm [122] gives the binary tree for dk27 (Figure 22). The values zero and one correspond to state code bits. To find the code, we should move from the leaves to the tree root.

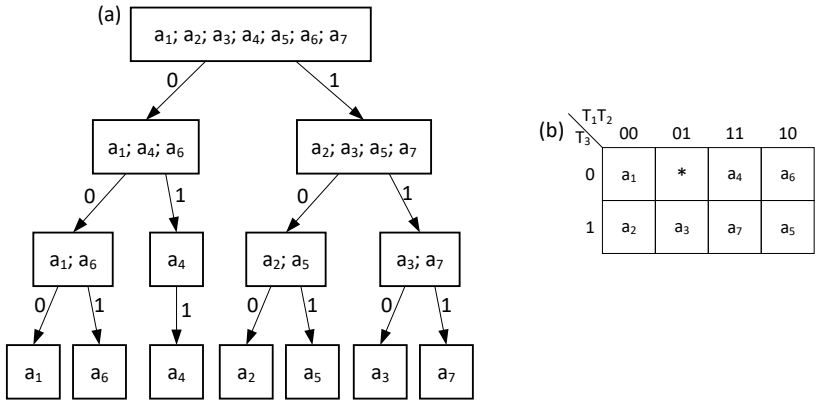

Figure 22. Binary tree for dk27 [122] (**a**) and Karnaugh map with state codes (**b**).

The resulting state codes are shown in the Karnaugh map (Figure 22b). R = 3 (this is the number of tree levels). For example, the following codes can be found: $K(a_1) = 000$, $K(a_2) = 001$, and so on.

To optimize the power consumption, it is necessary to take into account dependencies between states at some level of the tree. The optimization can be performed by swapping the nodes on the same tree level. It results in changing values of bit codes. For example, swapping state codes a_5 and a_2 (Figure 22a) produces state codes $K(a_5) = 001$ and $K(a_2) = 101$.

As shown in [122], the state assignment (Figure 22b) gives a sum of all Hamming distances equal to 16. Also, it gives a switching activity equal to 1.357. After swapping codes for pairs of states $\langle a_5, a_2 \rangle$ and $\langle a_3, a_7 \rangle$, the sum is equal to 15 and the average switching activity is equal to 1.19. Obviously, the less switching activity there is, the less the power consumption.

In [122], some results of experiments were shown. The system SIS [36] was used to calculate power consumption. The results were compared with results obtained using one-hot approach, JEDI, NOVA, and the one-level tree (OLT) algorithm [83]. The calculations were performed for the benchmarks from [82], with V_{dd} = 5 V and f_{op} = 20 MHz.

The results of these experiments showed the following values of power consumption and area (Table 2).

Table 2. Results of experiments [122].

Method	One-Hot	JEDI	NOVA	OLT	[122]
Power	8085	7754	9159	7732	7135
Area	6053	3280	3781	4012	3598

These results show that the method in [122] provides the minimum power consumption for the benchmarks from [82]. At the same time, the minimum area is provided by JEDI. The worst area characteristics are provided by the one-hot approach, whereas the maximum power is consumed by NOVA.

We can evaluate the overall efficiency of an algorithm by finding the product "Area × Power". In the discussed case, the following values of this product were found: 48.9×10^6 for one-hot, 25.4×10^6 for JEDI, 44.6×10^6 for NOVA, 31.02×10^6 for OLT, and 25.8×10^6 for [122]. A comparison these products show that both JEDI and [122] produce practically the same results.

All discussed state assignment methods are deterministic. To optimize power consumption in FPGA-based circuits, it is necessary to minimize the switching activity of flip-flops. This reduces the dynamic power consumption. To reduce the static power, it is necessary to reduce the chip area occupied by an FSM circuit. The chip area is determined by the number of LUTs and their interconnections. As noted in [92], the deterministic algorithms "are far from being optimal".

To improve the power consumption, various nondeterministic evolutionary methods have been developed. A survey of them can be found, for example, in [94]. All these methods deal with NP-complex problem, where NP stands for nondeterministic-polynomial time [123].

In [124], a genetic algorithm is proposed. For a given FSM, the algorithm optimizes the chip area occupied by its circuit. To get the optimal result, this algorithm uses a fitness function to evaluate the resulting chip area. In another genetic algorithm [125], the authors use literal counts as a cost function. In [93], both literal count (for area) and switching probability (for power) are used as an approximate cost function. In [111], the genetic algorithm optimizes both static and dynamic power consumption. To do it, the algorithm uses a fitness function based on the number of product terms, the switching activity, and Hamming distance among pairs of state codes. In [87,126], a genetic algorithm tries to optimize both static and dynamic power consumption. In [127], a multiobjective genetic algorithm optimizes both the area and power. To create a fitness function, it uses the number of product terms, the switching probability for state pairs, and the Hamming distance among pairs of states.

Some algorithms are based on simulated annealing for optimizing area and/or power. In [128], the approximate fitness function is used to optimize the area. In [129], both area and power are optimized. To do it, a fitness function is based on three characteristics: (1) the number of product terms, (2) the switching probability for state pairs, and (3) the Hamming distance. Also, approaches such as binary particle swarm and cuckoo search are used for optimizing the static power consumption (the circuit area) [130,131]. In [92], a probabilistic swap search state assignment algorithm is proposed. It is based on (1) assigning probabilities of each pair of code swaps and (2) probabilistically exploring pairwise code swaps. As a result, both area and power consumption are minimized for multi-level FSM circuits.

As follows from this short analysis, the outcome of state assignment significantly influences the power characteristics of FSM circuits. In this survey, we mostly analyzed FPGA-based FSMs. As a rule, LUT-based FSM circuits are multi-level. To decrease the static power, it is necessary to diminish the number of literals in SBFs (5)–(6). This allows a reduction in the chip area occupied by an FSM circuit. To reduce the dynamic power consumption, it is necessary to optimize Hamming distances between state codes for pairs of states with a high switching probability.

6. Replacing LUTs by Embedded Memory Blocks

As we have shown before, even a single EMB can replace a lot of LUTs and interconnections (Figure 5). To optimize the resulting FSM circuit, it is necessary to find a configuration $\langle S_A^*, t_F^* \rangle$ which allows us to obtain a single-level EMB-based FSM circuit. In this case, there are very important relations among the EMB characteristics (S_A^*, t_F^*) and FSM parameters (L, N, and R). The LUT count of a LUT-based FSM circuit is not important for replacing LUTs by EMBs.

An EMB is a coarse-grained element compared to a LUT. Thus, the transition from LUT-based FSMs to EMB-based FSMs is a transition from fine-grained to coarse-grained elements. This is similar to the transition from radio components (transistors, capacitors, resistors, inductors, and so on) to integrated circuits. Such a transition improves the final product quality (reducing the size, increasing the performance, reducing the power consumption, increasing the reliability) by reducing the number of interconnections. Also, there is a simplification of complex tasks such as the mapping, placement, and routing. Thus, we can expect the same effect from replacing LUTs by EMBs.

All EMB-based FSM design methods originate in microprogram control units (MCUs). The idea of MCUs was proposed in 1951 by M. Wilkes [132,133]. The MCUs have been used to control the process of program execution in computers. The MCU design methods depend on the approach used for addressing the microinstructions (MIs). One of the first addressing methods is compulsory addressing [134,135].

To design an MCU circuit, it is necessary to represent an initial STG as a microprogram. A microprogram is an ordered set of microinstructions kept into a special control memory (CM). A microinstruction location into CM is determined by its address. A format of an MI with compulsory addressing includes four fields [134]. The field FY includes a code of the collection of outputs (COs) executed in a particular cycle of MCU performance. The field FX includes a code $K(x_l)$ of an FSM input $x_l \in X$ to be checked for determining the transition address. The field $FA0$ includes a transition address for the case when $x_l = 0$. The field $FA1$ includes a transition address for the case when $x_l = 1$. For unconditional transitions, the field FX is empty; in this case, the next address is determined by the contents of $FA0$.

The following rule determines the next microinstruction address:

$$A^{t+1} = \begin{cases} [FA0]^t & \text{if} \quad [FX]^t = \varnothing; \\ [FA0]^t & \text{if} \quad x_e^t = 0; \\ [FA1]^t & \text{if} \quad x_e^t = 1. \end{cases} \tag{16}$$

In (16), $t = 0, 1, 2, \ldots$ is a cycle time, A^{t+1} is the next address (an address of an MI executed in the cycle $t+1$), $[FX]^t$, $[FA0]^t$, $[FA1]^t$ are the contents of the corresponding fields in the current operation cycle, x_l^t is a value of an FSM input determined by the field $[FX]^t$. The unconditional transition is determined by the first line of (16).

The following blocks represent the circuit of an MCU with compulsory addressing: block addressing (BA), register of microinstruction address, RG, control memory, and control flip-flop (TF). A structural diagram of an MCU is shown in Figure 23.

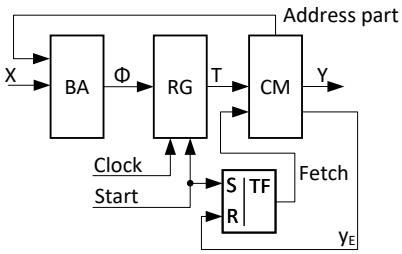

Figure 23. Structural diagram of an MCU.

The MCU (Figure 23) operates in the following manner. If $Start = 1$, then (1) the first address of the microprogram is loaded into RG and (2) $Fetch := 1$. If $Fetch = 1$, then the current MI is read from the CM. At the instant t, a microinstruction MI^t is fetched from CM. Its field FY is transformed into outputs $y_n \in Y$. They enter other blocks of the digital system. The address part of MI (fields FX, FA0, FA1) enters the BA, together with new values of FSM inputs. According to (16), BA generates variables $T_r \in T$ representing an address of the MI to be executed in the next cycle of MCU operation. The process is terminated if $y_E = 1$. In this case, $Fetch = 0$, and it is impossible to read MIs from the control memory.

In the first MCUs, the CM was implemented using read-only memory (ROM) blocks. The circuit of BA is implemented using gates and multiplexers [134]. If a microprogram includes M microinstructions, then the number of address bits is determined by (4). Obviously, the following relation holds: $|T| = |\Phi| = R$.

In the case of FPGA-based FSMs, the analog of the CM is implemented by EMBs, the analog of the addressing block is implemented using LUTs and dedicated multiplexers [46]. There is a significant difference between ROMs and EMBs. Namely, EMBs are synchronized blocks having a special control input which can be connected with the pulse *Clock* [46]. Thus, there is no need to have a separate register of addresses. The RG is hidden inside an EMB. Also, an EMB has a special control input to generate the zero code on its outputs. This input can be connected with the pulse *Start*. A typical EMB has S_A address inputs, t_F cell outputs, and three control inputs $(Clk,\ En,\ Cl)$ (Figure 24).

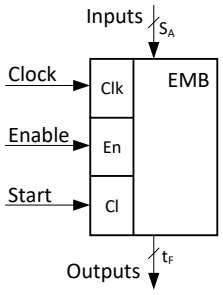

Figure 24. Organization of an embedded memory block.

If Clock = 1, then the outputs of a cell determined by the address inputs are loaded into the internal register of EMB. If Clear = 1, then RG = 0, and the address inputs are ignored. If Enable = 0, then the EMB operates in its standard mode. If Enable = 1, then the EMB outputs are in the third state. This means that the EMB is not connected with other existing blocks.

In [46], the authors propose to use EMBs for implementing Mealy FSM circuits. They propose an approach when the input MX cuts off the "don't care" FSM inputs (Figure 25).

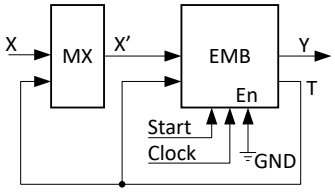

Figure 25. Structural diagram of an EMB-based Mealy FSM.

In [46], the authors show that using EMBs instead of LUTs allows a reduction in the power consumption. They write that "although memory arrays have greater power consumption when compared to individual LUTs and flip-flops, for state machine which uses several flip-flops, LUTs, and significant routing resources, the EMB-based approach

has lower power consumption". In Figure 25, the pulse *Clock* is connected with the *Clk* input of EMB, the pulse *Start* is connected with *Cl*, and $En = 0$.

The results of experiments are presented in [46]. They were obtained using the chip XC2V250-6fg256 by Virtex-II (Xilinx) and the library of standard FSM benchmarks [82]. To calculate the main characteristics of the FSM circuits (the number of EMBs, LUT counts, and power consumption), the authors used an experimental flow based on the CAD tool SIS [36]. This flow is shown in Figure 26.

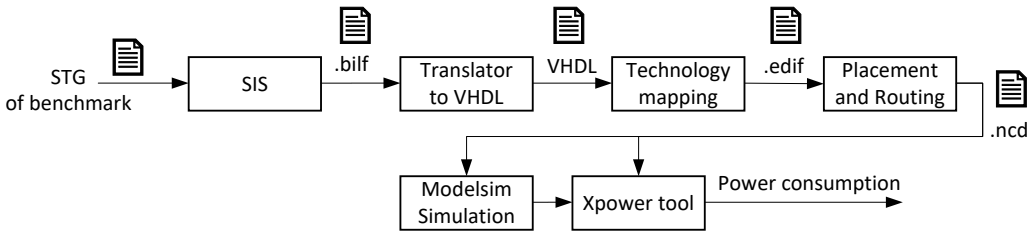

Figure 26. Experimental flow used in [46].

Using an initial STG, SIS generates a net-list of an FSM in *blif* format. This file describes the combinational circuit represented by (5)–(6) and the FSM state register. This file is transformed into a VHDL program using the translator from *blif* to VHDL. The tool Simplify-pro by Sinplicity executes the technology mapping. As a result, an *edif* file is generated. It describes LUTs, FFs, and their interconnections. To execute the placement and routing, they use the Xilinx ISE 4.2.03i design tool suite. To calculate the power dissipation, the *ncd* file enters Xpower tool. This tool also uses the *vcd* (value change dump) file produced by ModelSim simulator. These files are used to estimate the power dissipation.

The results of the experiments in [46] show that using EMBs leads to a significant area and power consumption improvement. The area was measured as a mutual LUT count and the number of flip-flops. Table 3 includes the results of experiments on some of the benchmarks from [82].

Table 3. Results of experiments [46].

Benchmark	LUT-Based FSMs			EMB-Based FSMs			% for 100 MHz
	LUT	FF	P (μW)	LUT	FF	P (μW)	
dk16	114	5	144.33	0	1	131.38	9.0
tbk	342	10	177.14	0	1	132.61	25.19
keyb	112	5	141.85	16	1	131.93	7.0
donfile	66	5	141.00	0	1	137.75	2.3
sand	263	10	185.08	20	2	162.83	12.2
styr	241	8	184.76	16	2	165.51	10.4
ex1	144	5	197.50	15	3	190.37	3.6
planet	320	12	224.39	18	3	199.85	11.0

The power was measured for different clock frequencies. But in Table 4, we show only results obtained for 100 MHz. The column "%" includes the percentage of power saving due to the replacement of LUTs by EMBs.

Also, in [46], the authors propose to enable the EMB input for further power saving. If an FSM does not change its state (this is an idle state), then the pulse *Clock* is disconnected from the synchronization input.

To save power, it is necessary to add some LUT-based logic for the timing control. For Mealy FSMs, this block is synthesized using some inputs $x_l \in X^1$, where $X^1 \subseteq X$, and some outputs $y_n \in Y^1$, where $Y^1 \subseteq Y$. The outputs can be used for situations where a state is not changed, but outputs are changed. This situation is shown in Figure 27.

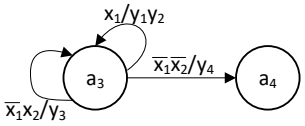

Figure 27. Changing outputs for unchangeable state a_3.

As follows from Figure 26, the state a_3 is not changed if there is either $x_1 = 1$ or $\overline{x_1}x_2 = 1$. But during these idle cycles, different collections of outputs are generated: $\lambda(a_3, x_1) = \{y_1, y_2\}$ and $\lambda(a_3, \overline{x_1}x_2) = \{y_3\}$.

The EMB-based Mealy FSM with the enabling *Clock* is shown in Figure 28. The block CL represents the clock logic. This block generates the function

$$En = f(T, X^1, Y^1). \tag{17}$$

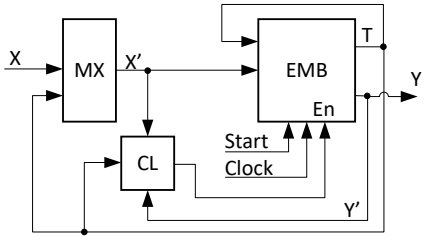

Figure 28. Structural diagram of an EMB-based Mealy FSM with clock logic [46].

As shown in [46], using CL allows a reduction in the power consumption compared with the Mealy FSM shown in Figure 25. But this approach is connected with the power overhead: the CL adds delay in performance and increases LUT count. Both saving power and area overhead are shown in Table 4.

Table 4. Saving and overhead from enabling EMB.

Benchmark	Power (µW) for 100 MHz	%	Area Overhead	
			LUTs	Slices
dk16	127.97	11.33	4	2
tbk	127.86	26.12	62	32
keyb	129.13	9.00	4	3
donfile	134.44	4.60	4	2
sand	158.07	14.50	47	25
styr	159.09	13.89	17	10
ex1	188.75	4.44	49	28
planet	190.47	15.20	9	13

In Table 4, the column "%" includes power saving for the frequency equal to 100 MHz. This saving takes into account the power consumed by the block CL. As follows from Table 4, using EMBs and CL allows a saving from 4% to 26% compared with equivalent LUT-based FSMs. Of course, these numbers are valid only for these experiments' conditions. But this approach may be used if the power saving is the most important issue of a particular project.

There is no need for the input register RGX for EMB-based FSMs. Due to the existence of the internal register inside EMB, the outputs $y_n \in Y$ are registered. The registering outputs have the same positive effect as the registering inputs $x_l \in X$. In both cases, the FSM outputs are stable. Also, using EMBs has a clear advantage over LUT-based approach. Namely, there is no need for using additional LUTs to create the input register RGX.

Using EMBs makes sense only till there is no need for the cascading EMBs. Let the symbol S_{Amax} stand for the number of address inputs if $t_F = 1$. The cascading should be used if the following condition holds:

$$\left|X^1\right| + R > S_{Amax}. \quad (18)$$

This is quite possible than more than a single EMB is used for implementing an FSM circuit (even if (18) is violated). In this case, it is necessary to compare the characteristics of equivalent LUT- and EMB-based FSM circuits. Also, a mixed approach should be investigated too. In the case of a mixed approach, a circuit is represented as a network of LUTs and EMBs [22,48–54].

7. Conclusions

Modern digital systems should be power-efficient. They should consume as little power as possible [3]. This is true for each block of a digital system. Obviously, this is true for various sequential blocks which play very important roles in digital systems. For example, control units operate in each cycle of a digital system operation. Very often, the sequential blocks are represented by finite state machines. In this survey, we mostly analyzed known methods of saving power for FPGA-based FSMs.

There are two sources of power dissipation in CMOS-based circuits: static and dynamic. They have a different nature. The static power dissipation is connected with the imperfection of MOS transistors, which leads to the presence of leakage currents in the stable state of an FSM circuit. To decrease the static power consumption, it is necessary to reduce the chip area occupied by an FSM circuit. There are thousands and thousands of methods developed to solve this problem. Their analysis can be found, for example, in the survey [136]. The dynamic power consumption is connected with the existence of parasitic capacitors which must be charged or discharged during the state change of combinational and sequential elements creating FSM circuits. To reduce this component of the power consumption, it is necessary to diminish the switching activity of an FSM. Finally, the third approach for saving power is associated with an increase in the granularity of the circuit elements. In the case of FPGA-based FSMs, this path leads to the replacement of LUTs with embedded memory blocks.

The existing methods can be divided into two groups: coarse-grained and fine-grained methods. The coarse-grained methods are general for all CMOS-based systems. The most popular coarse-grained methods are the voltage scaling and clock frequency scaling. These methods have some disadvantages. The main ones are: (1) the area overhead (to execute scaling, it is necessary to have some additional circuitry) and (2) the time overhead (to switch from reduced voltages or frequencies to normal ones, some time is needed, which is added to the total operation time required to complete the task of a digital system). Therefore, the coarse-grained methods can be used if a digital system based on them is able to complete the required task in a given time (the time should not exceed some deadline).

The fine-grained methods take into account specifics of both FSMs and FPGAs. Three groups of these methods exist. The first of them is clock gating. The method is based on disconnecting synchronization pulses from some blocks of the FSM circuit. This can be achieved by either a decomposition of an initial FSM or by disabling the input. The second approach is based on a proper state assignment leading to reducing the switching activity of flip-flops. The third approach is based on replacing LUTs by EMBs.

All these methods were analyzed in our current survey. The power saving could be reached by a twofold state assignment [137,138], but this approach was out of the scope of our survey. We hope that the review will help broaden the horizons of experts in the field of sequential circuit design. A good knowledge and understanding of existing methods of reducing power consumption is a prerequisite for the development of new, more effective methods to solve this very important problem.

All these methods were presented in our current survey. We hope that the review provides an extensive analysis of the history and current state of affairs in the field of reducing power consumption in FSM-based blocks of digital systems. In this regard, we think that the review may be used by designers of digital systems to (1) select the method most suitable for a particular project and (2) develop new methods to solve this problem.

Author Contributions: Conceptualization, A.B. and L.T.; methodology, A.B.; software, J.B.; validation, A.B. and L.T.; formal analysis, A.B.; investigation, J.B. and K.K.; resources, J.B. and K.K.; data curation, A.B.; writing—original draft preparation, A.B. and L.T.; writing—review and editing, A.B. and J.B.; visualization, J.B.; supervision, A.B.; project administration, L.T. and J.B.; funding acquisition, L.T. and A.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Kharchenko, V.; Kondratenko, Y.; Kacprzyk, J. (Eds.) *Green IT Engineering: Social, Business and Industrial Applications*; Springer International Publishing: Cham, Switzerland, 2019. [CrossRef]
2. Barkalov, A. Microprogram control unit as composition of automate with programmable and hardwired logic. *Autom. Comput. Sci.* **1983**, *17*, 36–41.
3. Monteiro, J.C. Power Optimization Using Dynamic Power Management. In Proceedings of the XIIth Conference on Integrated Circuits and Systems Design, Natal, Brazil, 2 October 1999; pp. 134–139.
4. De Micheli, G. *Synthesis and Optimization of Digital Circuits*; McGraw–Hill: Sydney, Australia, 1994; p. 578.
5. Baillieul, J.; Samad, T. (Eds.) *Encyclopedia of Systems and Control*; Springer: London, UK, 2015. [CrossRef]
6. Baranov, S. *Logic and System Design of Digital Systems*; TUT Press: Tallinn, Estonia, 2008; p. 276.
7. Minns, P.; Elliot, I. *FSM-Based Digital Design Using Verilog HDL*; John Wiley and Sons: Hoboken, NJ, USA, 2008.
8. Czerwinski, R.; Kania, D. *Finite State Machine Logic Synthesis for Complex Programmable Logic Devices*; Lecture notes in electrical engineering; Springer: Berlin, Germany, 2013; Volume 231, p. 172. [CrossRef]
9. Sklyarov, V.; Skliarova, I.; Barkalov, A.; Titarenko, L. *Synthesis and Optimization of FPGA-Based Systems*; Lecture notes in electrical engineering; Springer International Publishing: Cham, Switzerland, 2014; Volume 294, p. 432. [CrossRef]
10. Kubica, M.; Opara, A.; Kania, D. Logic Synthesis for FPGAs Based on Cutting of BDD. *Microprocess. Microsystems* **2017**, *52*, 173–187. [CrossRef]
11. Kubica, M.; Kania, D.; Kulisz, J. A Technology Mapping of FSMs Based on a Graph of Excitations and Outputs. *IEEE Access* **2019**, *7*, 16123–16131. [CrossRef]
12. Opara, A.; Kubica, M.; Kania, D. Methods of Improving Time Efficiency of Decomposition Dedicated at FPGA Structures and Using BDD in the Process of Cyber-Physical Synthesis. *IEEE Access* **2019**, *7*, 20619–20631. [CrossRef]
13. Kubica, M.; Kania, D. Area-oriented technology mapping for LUT-based logic blocks. *Int. J. Appl. Math. Comput. Sci.* **2017**, *27*, 207–222. [CrossRef]
14. Barkalov, O.; Titarenko, L.; Mazurkiewicz, M. *Foundations of Embedded Systems*; Studies in Systems, Decision and Control; Springer International Publishing: Cham, Switzerland, 2019; Volume 195, p. 167.
15. Arora, M. *Embedded System Design: Introduction to SoC System Architecture*; Learning Bytes Publishing: Islamabad, Pakistan, 2016.
16. Jimenez, J.J.; Trojman, L.; Procel, L.M. Power and Area Reduction of MD5 based on Cryptoprocessor Using novel approach of Internal Counters on the Finite State Machine. In Proceedings of the 2019 IEEE Fourth Ecuador Technical Chapters Meeting (ETCM), Guayaquil, Ecuador, 11–15 November 2019. [CrossRef]
17. Brown, B.D.; Card, H.C. Stochastic neural computation. I. Computational elements. *IEEE Trans. Comput.* **2001**, *50*, 891–905. [CrossRef]
18. Ardakani, A.; Leduc-Primeau, F.; Onizawa, N.; Hanyu, T.; Gross, W.J. VLSI Implementation of Deep Neural Network Using Integral Stochastic Computing. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2017**, *25*, 2688–2699. [CrossRef]
19. Li, P.; Lilja, D.J.; Qian, W.; Riedel, M.D.; Bazargan, K. Logical Computation on Stochastic Bit Streams with Linear Finite-State Machines. *IEEE Trans. Comput.* **2014**, *63*, 1474–1486. [CrossRef]
20. Xie, Y.; Liao, S.; Yuan, B.; Wang, Y.; Wang, Z. Fully-Parallel Area-Efficient Deep Neural Network Design Using Stochastic Computing. *IEEE Trans. Circuits Syst. II Express Briefs* **2017**, *64*, 1382–1386. [CrossRef]

21. Glaser, J.; Damm, M.; Haase, J.; Grimm, C. TR-FSM: Transition-Based Reconfigurable Finite State Machine. *ACM Trans. Reconfigurable Technol. Syst.* **2011**, *4*, 23:1–23:14. [CrossRef]
22. Das, N.; Priya, P.A. FPGA Implementation of Reconfigurable Finite State Machine with Input Multiplexing Architecture Using Hungarian Method. *Int. J. Reconfigurable Comput.* **2018**, *2018*, 6831901. [CrossRef]
23. Altera. Available online: http://www.altera.com (accessed on 1 January 2024).
24. Atmel. Available online: http://www.atmel.com (accessed on 1 January 2024).
25. Xilinx. Available online: http://www.xilinx.com (accessed on 1 January 2024).
26. Rodriguez-Andina, J.J.; Valdes-Pena, M.D.; Moure, M.J. Advanced Features and Industrial Applications of FPGAs—A Review. *IEEE Trans. Ind. Informatics* **2015**, *11*, 853–864. [CrossRef]
27. Monmasson, E.; Cirstea, M. FPGA Design Methodology for Industrial Control Systems—A Review. *IEEE Trans. Ind. Electron.* **2007**, *54*, 1824–1842. [CrossRef]
28. Rabaey, J. *Low Power Design Essentials*; Integrated Circuits and Systems; Springer: Greer, SC, USA, 2009.
29. Baranov, S. *Logic Synthesis of Control Automata*; Kluwer Academic Publishers: Norwell, MA, USA, 1994; p. 312.
30. Gajski, D. *Principles of Digital Design*; Prentice-Hall International Editions; Prentice-Hall International: Hoboken, NJ, USA, 1997.
31. Opara, A.; Kubica, M.; Kania, D. Strategy of logic synthesis using MTBDD dedicated to FPGA. *Integration* **2018**, *62*, 142–158. [CrossRef]
32. Brayton, R.; Mishchenko, A. ABC: An academic industrial-strength verification tool. In Proceedings of the Computer Aided Verification, Edinburgh, UK, 15–19 July 2010; pp. 24–40. [CrossRef]
33. Mealy, G.H. A method for synthesizing sequential circuits. *Bell Syst. Tech. J.* **1955**, *34*, 1045–1079. [CrossRef]
34. Moore, E.F. Gedanken-Experiments on Sequential Machines. In *Automata Studies. (AM-34)*; Princeton University Press: Princeton, NJ, USA, 1956; pp. 129–154. [CrossRef]
35. Glushkov, V. *Synthesis of Digital Automata*; FTD-MT, Translation Division, Foreign Technology Division; Fizmatgiz: Moscow, Russia, 1965; p. 487.
36. Sentowich, E.; Singh, K.; Lavango, L.; Moon, C.; Murgai, R.; Saldanha, A.; Savoj, H.; P, P.S.; Bryton, R.; Sangiovanni-Vincentelli, A. *SIS: A System for Sequential Circuit Synthesis*; Technical report; University of California: Berkely, CA, USA, 1992.
37. ABC System. Available online: https://people.eecs.berkeley.edu/~alanmi/abc/ (accessed on 1 January 2024).
38. Baranov, S.; Skliarov, V. *Digital Devices with Programmable LSIs with Matrix Structure*; Radio and Sviaz: Moscow, Russia, 1986; p. 272. (In Russian)
39. Skliarov, V. *Synthesis of Automata with Matrix LSIs*; Nauka i Technika: Minsk, Belarus, 1984; p. 256. (In Russian)
40. Grout, I. *Digital Systems Design with FPGAs and CPLDs*; Elsevier Science: Amsterdam, The Netherlands, 2011; p. 718.
41. Scholl, C. *Functional Decomposition with Application to FPGA Synthesis*; Kluwer Academic Publishers: Boston, MA, USA, 2001.
42. Dahl, O.J.; Dijkstra, E.W.; Hoare, C.A.R. (Eds.) *Structured Programming*; Academic Press Ltd.: Cambridge, MA, USA, 1972; p. 234.
43. Feng, W.; Greene, J.; Mishchenko, A. Improving FPGA Performance with a S44 LUT Structure. In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, New York, NY, USA, 25–27 February 2018; FPGA '18, pp. 61–66. [CrossRef]
44. Kilts, S. *Advanced FPGA Design: Architecture, Implementation, and Optimization*; Wiley-IEEE Press: Hoboken, NJ, USA, 2007; p. 312.
45. Barkalov, A.; Titarenko, L.; Kołopieńczyk, M.; Mielcarek, K.; Bazydło, G. *Logic Synthesis for FPGA-Based Finite State Machines*; *Studies in Systems, Decision and Control*; Springer International Publishing: Cham, Switzerland; Heidelberg, Germany, 2015; Volume 38, p. 280.
46. Tiwari, A.; Tomko, K. Saving power by mapping finite-state machines into Embedded Memory Blocks in FPGAs. In Proceedings of the Conference on Design, Automation and Test in Europe, Paris, France, 16–20 February 2004; pp. 916–921.
47. Kołopieńczyk, M.; Barkalov, A.; Titarenko, L. Hardware reduction for RAM-based Moore FSMs. In Proceedings of the 7th International Conference on Human System Interactions—HSI 2014, Costa da Caparica, Portugal, 16–18 June 2014; pp. 255–260.
48. Kołopieńczyk, M.; Titarenko, L.; Barkalov, A. Design of EMB-Based Moore FSMs. *J. Circuits Syst. Comput.* **2017**, *26*, 1–23. [CrossRef]
49. Garcia-Vargas, I.; Senhadji-Navarro, R. Finite State Machines With Input Multiplexing: A Performance Study. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2015**, *34*, 867–871. [CrossRef]
50. Garcia-Vargas, I.; Senhadji-Navarro, R.; Jiménez-Moreno, G.; Civit-Balcells, A.; Guerra-Gutierrez, P. ROM-based finite state machine implementation in low cost FPGAs. In Proceedings of the IEEE International Symposium on Industrial Electronics ISIE 2007, Vigo, Spain, 4–7 June 2007; pp. 2342–2347.
51. Senhadji-Navaro, R.; Garcia-Vargas, I. High-Speed and Area-Efficient Reconfigurable Multiplexer Bank for RAM-Based Finite State Machine Implementations. *J. Circuits Syst. Comput.* **2015**, *24*, 1550101. [CrossRef]
52. Senhadji-Navarro, R.; Garcia-Vargas, I. High-Performance Architecture for Binary-Tree-Based Finite State Machines. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2018**, *37*, 796–805. [CrossRef]
53. Senhadji-Navarro, R.; Garcia-Vargas, I.; Jiménez-Moreno, G.; Civit-Balcells, A.; Guerra-Gutierrez, P. ROM-based FSM implementation using input multiplexing in FPGA devices. *Electron. Lett.* **2004**, *40*, 1249–1251. [CrossRef]
54. Sklyarov, V. Synthesis and Implementation of RAM-based Finite State Machines in FPGAs. In Proceedings of the Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing, Villach, Austria, 27–30 August 2000; pp. 718–727. [CrossRef]

55. Rawski, M.; Selvaraj, H.; Łuba, T. An application of functional decomposition in ROM-based FSM implementation in FPGA devices. *J. Syst. Archit.* **2005**, *51*, 423–434. [CrossRef]
56. Rawski, M.; Tomaszewicz, P.; Borowski, G.; Łuba, T. Logic Synthesis Method of Digital Circuits Designed for Implementation with Embedded Memory Blocks on FPGAs. In *Design of Digital Systems and Devices. LNEE 79*; Adamski, M., Barkalov, A., Węgrzyn, M., Eds.; Springer: Berlin, Germany, 2011; pp. 121–144.
57. Rafla, N.I.; Gauba, I. A reconfigurable pattern matching hardware implementation using on-chip RAM-based FSM. In Proceedings of the 2010 53rd IEEE International Midwest Symposium on Circuits and Systems, Seattle, WA, USA, 1–4 August 2010; pp. 49–52. [CrossRef]
58. Maxfield, C. *The Design Warrior's Guide to FPGAs*; Academic Press, Inc.: Orlando, FL, USA, 2004.
59. Maxfield, C. *FPGAs: Instant Access*; Elsevier: Newnes, Australia, 2008.
60. Smith, M. *Application-Specific Integrated Circuits*; Addison-Wesley: Boston, MA, USA, 1997.
61. Veendrick, H.J.M. Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits. *IEEE J. Solid-State Circuits* **1984**, *19*, 468–473. [CrossRef]
62. Butts, J.A.; Sohi, G.S. A static power model for architects. In Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture. Association for Computing Machinery, Monterey, CA, USA, 1 December 2000; pp. 191–201. [CrossRef]
63. Tsui, C.y.; Monteiro, J.; Pedram, M.; Devadas, S.; Despain, A.; Lin, B. Power Estimation Methods for Sequential Logic Circuits. *Very Large Scale Integr. (VLSI) Syst. IEEE Trans.* **1995**, *3*, 404–416. [CrossRef]
64. Gajski, D.D.; Abdi, S.; Gerstlauer, A.; Schirner, G. *Embedded System Design: Modeling, Synthesis and Verification*, 1st ed.; Springer Publishing Company, Incorporated: New York, NY, USA, 2009.
65. Smarr, L. Project GreenLight: Optimizing Cyber-infrastructure for a Carbon-Constrained World. *Computer* **2010**, *43*, 22–27. [CrossRef]
66. *2011 International Technology Roadmap for Semiconductors (ITRS)*; Technical report; Semiconductor Industry Association: Washington, DC, USA, 2011.
67. Ashford, L.; Sanjit Arunkumar, S. *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed.; The MIT Press: Cambridge, MA, USA, 2016.
68. Marwedel, P. *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things*, 3rd ed.; Springer International Publishing: Cham, Switzerland, 2018; p. 423. [CrossRef]
69. Mittal, S. A survey of techniques for improving energy efficiency in embedded computing systems. *Int. J. Comput. Aided Eng. Technol.* **2014**, *6*, 440. [CrossRef]
70. Oelmann, B.; Tammemäe, K.; Kruus, M.; O'Nils, M. Automatic FSM Synthesis for Low-power Mixed Synchronous/Asynchronous Implementation. *VLSI Des.* **2001**, *12*, 167–186. [CrossRef]
71. Machalec, M.; Stastny, J. Synchronous FSM Design Methodology for Low Power Smart Sensors and RFID Devices. *Electro. Rev.* **2010**, *1*, 46–54.
72. Shabel, J. *Analysis of Clock Trees*; Technical report; SNUG: Boston, MA, USA, 2005.
73. Monteiro, J.C.; Oliveira, A.L. Finite State Machine Decomposition for Low Power. In Proceedings of the 35th Annual Design Automation Conference, San Francisco, CA, USA, 15–19 June 1998; pp. 758–763. [CrossRef]
74. Alidina, M.; Monteiro, J.; Devadas, S.; Ghosh, A.; Papaefthymiou, M. Precomputation-based sequential logic optimization for low power. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **1994**, *2*, 426–436. [CrossRef]
75. Knuth, D.E. *The Art of Computer Programming, Volume II: Seminumerical Algorithms*; Addison-Wesley: Boston, MA, USA, 1969.
76. Benini, L.; De Micheli, G. Automatic synthesis of low-power gated-clock finite-state machines. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **1996**, *15*, 630–643. [CrossRef]
77. Chow, S.H.; Ho, Y.C.; Hwang, T.; Liu, C.L. Low Power Realization of Finite State Machines—a Decomposition Approach. *ACM Trans. Des. Autom. Electron. Syst.* **1996**, *1*, 315–340. [CrossRef]
78. Hartmanis, J. Symbolic analysis of a decomposition of information processing machines. *Inf. Control* **1960**, *3*, 154–178. [CrossRef]
79. Devadas, S.; Newton, A. Decomposition and factorization of sequential finite state machines. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **1989**, *8*, 1206–1217. [CrossRef]
80. Levin, I.S. Decompositional Design of Automata Based on PLA with Memory. *Autom. Control Comput. Sci.* **1986**, *20*, 61–68.
81. Levin, I.S. A Hierarchical Model of the Interaction of Microprogrammed Automata. *Autom. Control Comput. Sci.* **1987**, *21*, 67–73.
82. McElvain, K. LGSynth93 Benchmark Set. Version 4.0., 1993. Available online: https://people.engr.ncsu.edu/brglez/CBL/benchmarks/LGSynth93/LGSynth93.tar (accessed on 1 February 2018).
83. Bacchetta, P.; Daldos, L.; Sciuto, D.; Silvano, C. Low-power state assignment techniques for finite state machines. In Proceedings of the 2000 IEEE International Symposium on Circuits and Systems (ISCAS'2000), Geneva, Switzerland, 28–31 May 2000; Volume 2, pp. 641–644.
84. Agrawal, R.; Borowczak, M.; Vemuri, R. A State Encoding Methodology for Side-Channel Security vs. Power Trade-off Exploration. In Proceedings of the 2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID), Delhi, India, 5–9 January 2019; pp. 70–75.
85. Benini, L.; De Micheli, G. State assignment for low power dissipation. *IEEE J. Solid-State Circuits* **1995**, *30*, 258–268. [CrossRef]
86. Benini, L.; De Micheli, G.; Macii, E. Designing low-power circuits: Practical recipes. *IEEE Circuits Syst. Mag.* **2001**, *1*, 6–25. [CrossRef]

87. Chattopadhyay, S. Area conscious state assignment with flip-flop and output polarity selection for finite state machines synthesis—A genetic algorithm. *Comput. J.* **2005**, *48*, 443–450. [CrossRef]
88. Chattopadhyay, S.; Chaudhuri, P. Genetic algorithm based approach for integrated state assignment and flipflop selection in finite state machines synthesis. In Proceedings of the IEEE International Conference on VLSI Design, Chennai, India, 4–7 January 1998; pp. 522–527.
89. Chen, C.; Zhao, J.; Ahmadi, M. A semi-Gray encoding algorithm for low-power state assignment. In Proceedings of the 2003 International Symposium on Circuits and Systems, IEEE, Bangkok, Thailand, 25–28 May 2003; Volume 5, pp. 389–392.
90. Devadas, S.; Ma, H.; Newton, A.; Sangiovanni-Vincentelli, A. MUSTANG: State assignment of finite state machines targeting multilevel logic implementation. *IEEE Trans. Comput.-Aided Des.* **1988**, *7*, 1290–1300. [CrossRef]
91. Du, X.; Hachtel, G.; Lin, B.; Newton, A. MUSE: A multilevel symbolic encoding algorithm for state assignment. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **1991**, *10*, 28–38. [CrossRef]
92. El-Maleh, A. A Finite State Machine-based Fault Tolerance Technique with Enhanced Area and Power of Synthesized Sequential Circuits. *IET Comput. Digit. Tech.* **2017**, *11*, 159–164. [CrossRef]
93. El-Maleh, A.; Sait, S.; Khan, F. Finite state machine state assignment for area and power minimization. In Proceedings of the 2006 IEEE International Symposium on Circuits and Systems, IEEE, Kos, Greece, 21–24 May 2006; pp. 5303–5306.
94. El-Maleh, A.H. A probabilistic pairwise swap search state assignment algorithm for sequential circuit optimization. *Integr. VLSI J.* **2017**, *56*, 32–43. [CrossRef]
95. Grześ, T.N.; Solov'ev, V.V. Minimization of power consumption of finite state machines by splitting their internal states. *J. Comput. Syst. Sci. Int.* **2015**, *54*, 367–374. [CrossRef]
96. Gupta, B.; Narayanan, H.; Desai, M. A state assignment scheme targeting performance and area. In Proceedings of the 12th International Conference on VLSI Design, Goa, India, 7–10 January 1999; pp. 378–383.
97. Hu, H.; Xue, H.; Bian, J. A heuristic state assignment algorithm targeting area. In Proceedings of the 5th International Conference on ASIC, Beijing, China, 21–24 October 2003; Volume 1, pp. 93–96.
98. Huang, J.; Jou, J.; Shen, W. ALTO: An Iterative Area/Performance Algorithms for LUT-based FPGA Technology Mapping. *IEEE Trans. VLSI Syst.* **2000**, *18*, 392–400. [CrossRef]
99. Iranli, A.; Rezvani, P.; Pedram, M. Low power synthesis of finite state machines with mixed D and T flip-flops. In Proceedings of the Asia and South Pacific–DAC, Kitakyushu, Japan, 21–24 January 2003; pp. 803–808.
100. Kubatova, H.; Becvar, M. FEL–Code: FSM Internal State Encoding Method. In Proceedings of the 5th International Workshop on Boolean Problems, Freiberg, Germany, 19–20 September 2002; pp. 109–114.
101. Nöth, W.; Kolla, R. Spanning tree based state encoding for low power dissipation. In Proceedings of the Conference on Design, automation and test in Europe, Association for Computing Machinery, Munich, Germany, 1 January 1999; p. 37.
102. Park, S.; Cho, S.; Yang, S.; Ciesielski, M. A new state assignment technique for testing and low power. In Proceedings of the 41st annual Design Automation Conference. Association for Computing Machinery, San Diego, CA, USA, 7–11 June 2004; pp. 510–513.
103. Park, S.; Yang, S.; Cho, S. Optimal state assignment technique for partial scan designs. *Electron. Lett.* **2000**, *36*, 1527–1529. [CrossRef]
104. Pedram, C.; Despain, A. Low-power state assignment targeting two- and multilevel logic implementations. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **1998**, *17*, 1281–1291.
105. Pomerancz, I.; Cheng, K. STOIC: State assignment based on output/input functions. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **1993**, *12*, 1123–1131. [CrossRef]
106. Salauyou, V.; Grześ, T. FSM State Assignment Methods for Low-Power Design. In Proceedings of the 6th International Conference on Computer Information Systems and Industrial Management Applications (CISIM'07), Elk, Poland, 28–30 June 2007; pp. 345–350. [CrossRef]
107. Shiue, W. Novel state minimization and state assignment in finite state machine design for low-power portable devices. *Integr. VLSI J.* **2005**, *38*, 549–570. [CrossRef]
108. Solov'ev, V.V. Changes in the length of internal state codes with the aim at minimizing the power consumption of finite-state machines. *J. Commun. Technol. Electron.* **2012**, *57*, 642–648. [CrossRef]
109. Sutter, G.; Todorovich, E.; López-Buedo, S.; Boemo, E. Low-power FSMs in FPGA: Encoding alternatives. In *Integrated Circuit Design. Power and Timing Modeling, Optimization and Simulation*; Springer: Seville, Spain, 2002; pp. 363–370.
110. Wu, X.; Pedram, M.; Wang, L. Multi-code state assignment for low-power design. *IEEE Proc. Circuits Devices Syst. IET* **2000**, *147*, 271–275. [CrossRef]
111. Xia, Y.; Almani, A. Genetic algorithm based state assignment for power and area optimization. *IEEE Proc. Comput. Digit. Tech.* **2002**, *149*, 128–133. [CrossRef]
112. Xilinx. *XC4000E and XC4000X Series Field Programmable Gate Arrays*; Technical report; Xilinx: San Jose, CA, USA, 1999.
113. Available online: http://www.cypress.com (accessed on 1 January 2016).
114. Trivedi, K.S. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*; Prentice Hall: Montgomery, IL, USA, 1982.
115. Agerwala, T. Microprogram optimization: A survey. *IEEE Trans. Comput.* **1976**, *25*, 962–973. [CrossRef]

116. Mishchenko, A.; Brayton, R.; Jiang, J.H.R.; Jang, S. Scalable Don't-Care-Based Logic Optimization and Resynthesis. *ACM Trans. Reconfigurable Technol. Syst.* **2011**, *4*, 1–23. [CrossRef]
117. Pradhan, S.N.; Kumar, M.T.; Chattopadhyay, S. Low power finite state machine synthesis using power-gating. *Integration* **2011**, *44*, 175–184. [CrossRef]
118. Hartmanis, J.S.R. Some gangers in the state reduction of sequential machines. *Inform. Control* **1962**, *5*, 252–260. [CrossRef]
119. Devadas, S.; Ma, H.K.; Newton, A.; Sangiovanni-Vincentelli, A. Irredundant sequential machines via optimal logic synthesis. In Proceedings of the Computer-Aided Design of Integrated Circuits and Systems, Kailua-Kona, HI, USA, 2–5 January 1990; Volume 9, pp. 417–426. [CrossRef]
120. Pfleeger, C. State Reduction in Incompletely Specified Finite-State Machines. *IEEE Trans. Comput.* **1973**, *C-22*, 1099–1102. [CrossRef]
121. Salauyou, V.; Klimowicz, A.; Grześ, T.; Bulatowa, I.; Dimitrowa-Grekow, T. Synthesis methods of finite state machines implemented in package ZUBR. In Proceedings of the 6th International Conference on Computer-Aided Design in Discrete Devices (CAD DD'7), Minsk, Belarus, 14–15 November 2007; pp. 53–56.
122. Kajstura, K.; Kania, D. Low Power Synthesis of Finite State Machines — State Assignment Decomposition Algorithm. *J. Circuits, Syst. Comput.* **2017**, *27*, 185004–185014. [CrossRef]
123. Sait, S.Y.H *Iterative Computer Algorithms with Application in Engineering: Solving Combinatorial Optimization Problems*; California: IEEE Computer Society Press: Los Alamitos, CA, USA, 1999.
124. Amaral, J.; Tumer, K.; Ghosh, J. Designing genetic algorithms for the state assignment problem. *IEEE Trans. Syst. Man, Cybern.* **1995**, *25*, 687–694. [CrossRef]
125. Almaini, A.; Miller, J.; Thomson, P.; Billina, S. State assignment of finite state machines using a genetic algorithm. *IEE Proc. Comput. Digit. Tech.* **1995**, *142*, 279. [CrossRef]
126. Chu, Y.C. *Computer Organization and Microprogramming*; Prentice Hall: Hoboken, NJ, USA, 1972.
127. Jassani, B.A.; Urquhart, N.; Almaini, A. State assignment for sequential circuits using multi-objective genetic algorithm. *IET Comput. Digit. Tech.* **2011**, *5*, 296. [CrossRef]
128. Aly, W.M. Solving the State Assignment Problem Using Stochastic Search Aided with Simulated Annealing. *Am. J. Eng. Appl. Sci.* **2009**, *2*, 703–707. [CrossRef]
129. Yang, M. State Assignment for Finite State Machine Synthesis. *J. Comput.* **2013**, *8*. [CrossRef]
130. El-Maleh, A.H.; Sheikh, A.T.; Sait, S.M. Binary particle swarm optimization (BPSO) based state assignment for area minimization of sequential circuits. *Appl. Soft Comput.* **2013**, *13*, 4832–4840. [CrossRef]
131. El-Maleh, A.H.; Sait, S.M.; Bala, A. State assignment for area minimization of sequential circuits based on cuckoo search optimization. *Comput. Electr. Eng.* **2015**, *44*, 13–23. [CrossRef]
132. Wilkes, M. The best way to design an automatic calculating machine. In Proceedings of the Manchester University Computer Inaugural Conference, Manchester, UK, 9–12 July 1951.
133. Wilkes, M.V.; Stringer, J.B. Micro-programming and the design of the control circuits in an electronic digital computer. *Math. Proc. Camb. Philos. Soc.* **1953**, *49*, 230–238. [CrossRef]
134. Barkalov, A.; Titarenko, L. *Logic Synthesis for Compositional Microprogram Control Units*; Springer: Berlin, Germany, 2008; Volume 22.
135. Barkalov, A.; Węgrzyn, M. *Design of Control Units With Programmable Logic*; University of Zielona Gora Press: Zielona Gora, Poland, 2006; p. 150.
136. Barkalov, A.; Titarenko, L.; Krzywicki, K. Structural Decomposition in FSM Design: Roots, Evolution, Current State—A Review. *Electronics* **2021**, *10*, 1174. [CrossRef]
137. Barkalov, O.; Titarenko, L.; Mielcarek, K. Hardware reduction for LUT-based Mealy FSMs. *Int. J. Appl. Math. Comput. Sci.* **2018**, *28*, 595–607. [CrossRef]
138. Barkalov, O.; Titarenko, L.; Mielcarek, K. Improving characteristics of LUT-based Mealy FSMs. *Int. J. Appl. Math. Comput. Sci.* **2020**, *30*, 745–759. [CrossRef]