International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org
Volume 6, Issue 1, January - February 2017 **ISSN 2278-6856**

Minimal finite-state automaton associated with a sequential circuit

Pierre MOUKELI MBINDZOUKOU

LAIMA - Institut Africain d'Informatique (IAI)
(African Institute of Computer Science),

Abstract

The tendency in the implementation of the sequential circuits is to specify the behavior of a circuit in the term of a finite automaton. Then to use a tool to transcribe this description in a specification language that will produce a simulation of the circuit before its implementation. This approach deprives the designer of the powerful mathematical tools developed in language theory, allowing significant reductions in complexity in term of the number of states and transitions. The present contribution proposes an approach making it possible to deduce from a sequential circuit specification the associated minimum automaton, the implementation of which allows a substantial reduction in the complexity of said circuit. Algorithms in this sense are proposed to decompile a sequential circuit, to construct the disjunctive canonical form of its transition function by using the canonical monomials composed of the states and input variables, and to deduce a minimal finite automaton. The implementation of this minimal finite automaton will give a less complex circuit as foreseen in the theory of languages.

Keywords: regular languages, finite-state automaton, sequential circuit simplification, transition function.

1. INTRODUCTION

Reducing logic circuits complexity is a well-known problem, for which, various simplification algorithms exist [2] [20] [21] [22] [27]. With regard to the *sequential circuits*, they are implementation of *finite-state automata* [1] [4] [16] [17] [18]. We know that any finite-state automaton admits a unique *minimal finite-state automaton* [5] [20] [21] [27] [28] [29], whose implementation leads to a reduced sequential circuit. Unfortunately, the world of language theory of which the finite-state automaton concerns, and that of computer architecture of which raise the sequential circuits, use the same mathematical objects (finite-state machines), and are often unaware of; so much that the others do not benefit from the powerful tools developed by the ones, in particular as regards simplification of the finite-state machines.

A good approach for the computers architects, to inherit the powerful tools developed by the theorists of languages would be first, to begin the design of a sequential circuit with the specification of the finite-state automaton of which it will be the implementation; and second to apply the reduction algorithms in order to obtain the related single minimal automaton [21] [24]. Unfortunately, it is not the use, owing to the fact that the logic circuit builders are unaware of languages theory algorithms. This is why, in the present contribution, we propose an approach which allows, starting from the specification of a sequential circuit, to obtain the associated minimal automaton; provided that this minimal automaton is unique for each sequential circuit. The pursued goal is that, instead of learning the languages theory algorithms, the computers architects apply our procedure to calculate the minimal automaton, whose implementation will contribute to reducing significantly the complexity of the initial sequential circuit.

This paper is organized as follow. In the second part, we point out some fundamental relating to finite-state automata and sequential circuits. In the third part, we detail the problem above stated. In the fourth part, we decline our solution which consists in deducing a finite-state machine from the sequential circuit. From this finite-state automaton, the unique minimal automaton associated with this sequential circuit will be calculated. In the firth part, we present an application example; and finally, in the conclusion, we outline the limits and the prospects for this work.

2. DEFINITIONS

The definitions given in this section are those usually found in the abundant literature on finite state machines [4] [7] [13] [20] [22] [29] and sequential circuits [3] [4] [14] [17] [18] [19] [22].

2.1 Definitions related to finite state automata

A **vocabulary** or alphabet is a nonempty finite set of atomic symbols. For example, the ten Indian digits and the Greek or Roman alphabets are vocabularies, unlike Roman numerals that do not represent an alphabet. A **string** or **word** over a vocabulary V is any combination (sequence) of vocabulary symbols; i.e. a finite length sequence of symbols from V. If V is a vocabulary,

V^* denotes the infinite set of all the words build from the concatenation (juxtaposition) of elements of V, including empty string denoted ε. We call **formal language** (or simply **language**, if there is no confusion), every part of V^*.

Formal languages, generated by grammars and recognized by automata, exist in several types and are classified according to the generation technique (Noam Chomsky's classification [7]) or the recognition technique. In this paper, we are only interested in regular languages, which are generated by regular grammars and recognized by finite-state automata (or finite-state machines).
Formally, a *finite-state automaton* is a 5-tuple: $A = < V_T, Q, q_0, F, \mu >$, such that:

- V_T denotes the *final vocabulary*; i.e. the vocabulary used to write the words recognized by the finite-state machine. We call *final symbol*, any element of V_T;
- Q denotes the set of states; i.e. a vocabulary such that: $V_T \cap Q = \varnothing$;
- $q_0 \in Q$ denotes the *initial state*; that is the state in which the automaton is set to start the recognition of a new word;
- $F \subset Q$ denotes the *set of final states*, provided that a final state (or accepting state) is a state in which the automaton ends the recognition of any word of the related regular language;

$\mu: \ Q \times V_T \to Q$ is the *transition function*; i.e. a mapping of a set of states and a set of final symbols onto the original set of states. It is a mathematical model of the automaton behavior, specifying which state the machine will go to after the recognition of an input final symbol. The interpretation of $\mu(q,a) = q'$ is that automaton A in state q with a as input final symbol will go in state q' after processing input atomic data a. The function μ admits an extension denoted: $\hat{\mu} : Q \times V_T^* \to Q,$ such that:

$$\forall (q,a,x) \in Q \times V_T \times V_T^*, \begin{cases} \hat{\mu}(q,ax) = \hat{\mu}\left(\mu(q,a),x\right) \\ \hat{\mu}(q,\varepsilon) = q \end{cases}$$

Behavior of an automaton: The automaton A switches from a state to another, as it recognizes each input final symbol from the input stream of the word to be recognized.

2.2 Definitions related to sequential circuits

We call ***Boolean* or *binary variable***, any numeric variable in the set: $B = \{0,1\}$. The set B is associated with two binary logic operators: ***AND*** (also called *conjunction*) and ***OR*** (also called *disjunction*) and one unary operator: ***NOT*** (called *complement*), noted respectively: $\wedge, \vee, \overline{\ \ }$. These three logic operators allow expressing all the other logic operators. AND and OR operators are implemented with AND and OR gates respectively and NOT is implemented with an *inverter* or NOT gate.

We call ***logic function***, a function on logic variables, associated by logic operators and possibly of other logic functions. A ***logic equation*** is an equation of logic functions and binary variables. Logic functions can be implemented with electronic circuits, called ***combinatorial circuits***.

We call ***monomial***, the conjunction of possibly individually complemented logic variables. A ***canonical monomial*** is a monomial comprising the totality of the logic variables used in a given logic function. For example, if x, y, z are Boolean variables, then $xy,\ y\overline{z},$ $x\overline{y}z,\ \overline{x}y\overline{z}$ are four monomials; among which $x\overline{y}z,\ \overline{x}y\overline{z}$ are two canonical monomials. When a function is expressed as a disjunction of canonical monomials, it is said to be in ***disjunctive canonical form***, or quite simply in the ***canonical*** or ***normal form***, if there is no ambiguity. Notice that each logic function admits a unique canonical form.

The value computed by a logic function must be stored in an elementary memory called ***latch*** or ***flip-flop***, which can store only one binary digit (i.e. ***bit***). There are several types of latches: D, RS, JK, T and so on. A latch can be synchronized with a clock intended to allow the latch to change its state. The content of a latch is available through its output q (often referred to as Q in the literature). A combinatorial circuit, provided with a memory is called ***sequential circuit***. In a sequential circuit, the memory output can be used as an input of the same circuit. Both combinatorial and sequential circuits are called ***logic circuits***.

A sequential circuit is formally defined as a quintuplet $C = < Z, I, Y, \delta, \omega >$ such that:

- Z denotes the set of states encoded in a set of latches;
- I refers to the set of input boolean variables; it can be regarded as a register containing the input value;
- Y denotes a set of circuit output values;

- δ is the transition function which determines circuit behavior; i.e. a combinatorial function which determines the circuit next state from the input variables and the latches output values. The transition function can incorporate a component called input-function, devoted to filtering input values;
- ω denotes the output function which uses the inputs variables and the latches outputs to determine the output value of the sequential circuit.

3 PROBLEM PRESENTATION

It is established that any sequential circuit is the implementation of a finite state automaton [1] [3] [4] [11] [14] [17] [18] [19] [22] and that all the equivalent finite-state automata admit one and only one minimal finite-state automaton [4] [5] [6] [7] [13] [20] [22] [29]. This automaton is minimal in term of possible states and transitions [12] [13]. This implies that any sequential circuit admits a unique minimal finite-state automaton.

Finite state machines can be represented graphically for an efficient visualization or design. Then, the visualization can be converted in a simulator using hardware description languages. So, the designer requires a tool to convert the visualized design to hardware description language code in order to simulate and implement it. Procedures to covert state diagrams to hardware description language code are addressed by many references [1] [3] [8] [30] [9] [10] [19] [23] [25] [26] [30].

Most circuit conception platforms represent finite state machines graphically for efficient visualization and design; this visualization in then converted in a simulator using hardware description languages, using appropriate tools. The procedure is addressed by many existing tools [15] [23] [26] [30].

Despite implementing a sequential circuit from its finite-state automaton, it is better to substitute this one by the associated minimal finite-state automaton; because the latter offers a minimal number of states and transitions, and the language theory guarantees that it is equivalent to the automaton of departure [12] [13], since they recognize both the same regular language. From then on, the sequential circuit can be implemented from the minimal automaton, with a minimal complexity.

But, given a sequential circuit, how can we compute its related minimal finite-state automaton? Ideally to do so, prior to conceive a sequential circuit, one must initially design the associated finite-state machine, then calculates its related minimal automaton, and finally implements this last as a sequential circuit using a well-known procedures [1] [3] [4] [8] [9] [10] [17] [18] [19] [22]. Unfortunately, this practice is not use, because the world of the finite-state machines (theory of the languages) and that of sequential circuits (computers architecture) cohabit while being unaware of.

Since sequential circuit designers are unwilling to use the languages theory powerful tools in order to simplify their finite-state machines, therefore pending sequential circuits, a solution would be to let them design their sequential circuits, then to provide them with a method to calculate the associated minimal automaton. The interest of this approach is obvious: the reduction of sequential circuit complexity, by exploiting the simplification potential offered by the minimal finite-state automata. However, there is no procedure to calculate the minimal finite-state automaton associated with a sequential circuit; so it is the aim of this contribution to provide such a procedure. Once a sequential circuit designed or specified, our method consists in disassembling this circuit in order to deduce an associated finite-state machine, then to calculate the single minimal automaton associated with this circuits.

4 MINIMAL FINITE AUTOMATON ASSOCIATED WITH A SEQUENTIAL CIRCUIT

Let $C =< Z, I, Y, \delta, \omega >$ denote a sequential circuit. We need to deduce from C, a related finite-state automaton $A_C =< V_T, Q, q_0, F, \mu >$, from which we can calculate the single minimal finite-state automaton $A_{C[\eta]}$ associated with C. Reconstituting the minimal finite-state machine $A_{C[\eta]}$ from the sequential circuit C amounts previously to deduce from the quintuplet $< Z, I, Y, \delta, \omega >$, a finite automaton $A_C =< V_T, Q, q_0, F, \mu >$ implemented as C. Once the finite-state automaton A_C is available, the next step will consist in applying to A_C the adequate algorithm in order to calculate its unique minimal automaton, which is to be the minimal automaton $A_{C[\eta]}$ of the sequential circuit C. It is the purpose of this section to show how A_C and $A_{C[\eta]}$ can be computed, provided C.

First, we propose an algorithm to disassemble a combinatorial circuit, in order to rebuild its unique disjunctive canonical function. Second, using this algorithm, we propose another algorithm to calculate a finite-state automaton associated with a sequential circuit. Third, we propose a version of the minimal automaton algorithm adapted to the case of sequential circuit.

4.1 Algorithm to disassemble a combinatorial circuit

Let $S=\sum_{i=1}^{k}\alpha_i$, be the equation of a combinatorial circuit, where k is the number of logic functions α_i whose sum gives the output of the combinatorial circuit.

Procedure 1: Calculation of the disjunctive form of a combinatorial function

```
Let $E_S=\{\alpha_1,\alpha_2,...,\alpha_k\}$ be the set of functions such that: $S=\sum_{i=1}^{k}\alpha_i$.
While ($\exists\alpha\in E_S\,/\,\alpha$ is not a monomial) do:
    1.Remove $\alpha$ from $E_S$;
    2.Transform $\alpha$ using AND, OR and NOT operators, such that: $\alpha=\sum_{i=1}^{n}\beta_i$, where
      $\beta_i$ is a function or a monomial, and $n$ is the number of $\beta_i$;
    3.Insert each $\beta_i$ in $E_S$. Naturally, duplicates are removed from $E_S$.
```

The transformations suggested in step 2 can be made using the same rules as those of logical equation simplification, in particular Morgan's laws [4] [14] [17] [24]. Procedure 1 ends, because any function can be decomposed using only AND, OR and NOT operators. At the end of procedure 1, the set E_S consists of k' monomials β_i such that:

$$\begin{cases} E_S=\{\beta_1,\beta_2,...,\beta_{k'}\} \\ S=\sum_{i=1}^{k'}\beta_i \end{cases} \qquad (1)$$

Let us note that the equality (1) was obtained by replacing every function α_i by its disjunctive form. The next step consists in normalizing S using the following procedure 2.

Procedure 2: *Calculation of the normal disjunctive form of a combinatorial function*

```
Let $V_S=\{v_1,v_2,...,v_j\}$ be the set of $j$ Boolean variables constituting $E_S$ monomials.
While ($\exists\beta\in E_S\,/\,\beta$ is not a canonical monomial) do
    1.Remove $\beta$ from $E_S$;
    2.If $v\in V_S$ is a logic variable not present in $\beta$ then add $\beta v$ and $\beta\overline{v}$ to $E_S$.
        Duplicates are removed from $E_S$.
```

At the end of procedure 2, the set E_S consists now in a set of $k"$ canonical monomials such that:

$$\begin{cases} E_S=\{\beta_1,\beta_2,...,\beta_{k"}\} \\ S=\sum_{i=1}^{k"}\beta_i \end{cases}$$

Illustrative example: Let A,B,C be 3 Boolean variables and S a logic function such that:
$S=\overline{A+(B\oplus C)}+C$

Our aim is to find the canonical form of S using the above procedures 1 and 2. With procedure 1, we recognize that C, the last term of S, is already a monomial. So, at the thirst step of procedure 1 we have: $E_S=\{\overline{A+(B\oplus C)},C\}$. The following are the next steps of procedure 1 to transform the function $\overline{A+(B\oplus C)}$ in a disjunction of monomials:

$$\begin{aligned} E_S &=\{\overline{A+(B\oplus C)},C\}=\{\overline{A}\,\overline{B\oplus C)},C\} \\ &=\{\overline{A}(\overline{B\overline{C}+\overline{B}C}),C\} \\ &=\{\overline{A}(\overline{B\overline{C}}(\overline{\overline{B}C}),C\}=\{\overline{A}(\left(\overline{B}+\overline{\overline{C}}\right)\left(\overline{\overline{B}}+\overline{C}\right)),C\} \\ &=\{\overline{A}(\left(\overline{B}+C\right)\left(B+\overline{C}\right)),C\} \\ &=\left\{\overline{A}\left(\overline{B}\,\overline{C}+BC\right),C\right\} \\ &=\left\{\overline{A}\,\overline{B}\,\overline{C}+\overline{A}BC,C\right\}=\left\{\overline{A}\,\overline{B}\,\overline{C},\overline{A}BC,C\right\} \end{aligned}$$

Three monomials were obtained: ($\overline{A}\,\overline{B}\,\overline{C},\overline{A}BC,C$), two of which are canonical ($\overline{A}\,\overline{B}\,\overline{C},\overline{A}BC$). It thus remains to transform the monomial C into a set of canonical monomials. To do so, we apply procedure 2, introducing successively the variables A and B:

$$C=AC+\overline{A}C=ABC+A\overline{B}C+\overline{A}BC+\overline{A}\,\overline{B}C$$

So, we add the four monomials $(ABC, A\overline{B}C, \overline{A}BC, \overline{A}\overline{B}C)$ to E_S :

$$\Rightarrow E_S = \left\{\overline{A}\overline{B}\overline{C}, \overline{A}BC, ABC, A\overline{B}C, \overline{A}\overline{B}C\right\}$$

We can now deduce the canonical form of S :

$$S = \overline{A}\overline{B}\overline{C} + \overline{A}BC + ABC + A\overline{B}C + \overline{A}\overline{B}C .$$

4.2 Finite-state automaton associated with a sequential circuit

We are interested in sequential circuits with states coded in a set of latches. This excludes from our field, in particular token machines. Let $C =< Z, I, Y, \delta, \omega >$ denotes a sequential circuit. Calculating the finite-state machine $A_C =< V_T, Q, q_0, F, \mu >$ implemented as C, amounts determining a finite-state automaton using the quintuplet $< Z, I, Y, \delta, \omega >$. To do so, we will proceed in 5 steps, namely:

(1) Disassembling the transition function δ in order to obtain its constituting set of canonical monomials E_δ ;
(2) Determining final vocabulary V_T and the set of automaton states Q;
(3) Determining the initial state q_0 ;
(4) Determining the set of final states F ;
(5) Calculating the finite-state machine transition function μ .

These five steps are described below.

4.2.1Disassembling a transition function

Let us recall that the sequential circuit transition function is a combinatorial circuit which determines the successor of the current state, according to the value of entry variables and the aforesaid current state. Switching from the current state to its successor is called ***transition***. It is supposed that the sequential circuit states are coded using n latches. In such a circuit $C =< Z, I, Y, \delta, \omega >$, the transition function δ is in fact, a vector of boolean functions : $\delta = \left(\delta_1, \delta_2, ..., \delta_n\right)$, such that each δ_i is the transition function of latch L_i, whose output is noted q_i . Each boolean function δ_i operates on a couple of vectors $(\vec{Q}, \vec{I})$ such that (see fig. 1): $\vec{Q} = (q_1, q_2, ..., q_n)$ is the vector of the output of the n latches encoding each sequential circuit state; and $\vec{I} = (i_1, i_2, ..., i_m)$ is the vector of m circuit input variables. In the following, we suppose that $\omega = \vec{Q}$, which means that the output function of the sequential circuit is reduced to the outputs of the flip-flops encoding the sequential circuit states; because, as well as the input function, the output function is not taken into account in reducing a sequential circuit transition function.

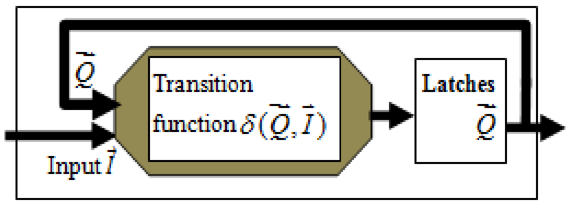

Figure 1: Sequential circuit components

In fact, the transition function δ is a vector of n Boolean functions:

$$\delta(\vec{Q}, \vec{I}) = \left(\delta_1(\vec{Q}, \vec{I}), \delta_2(\vec{Q}, \vec{I}), ..., \delta_n(\vec{Q}, \vec{I})\right);$$

Such that δ_i is the transition function of latch i , $\vec{Q}$ is the output vector of the n latches, and $\vec{I}$ is the input vector (see fig. 2).

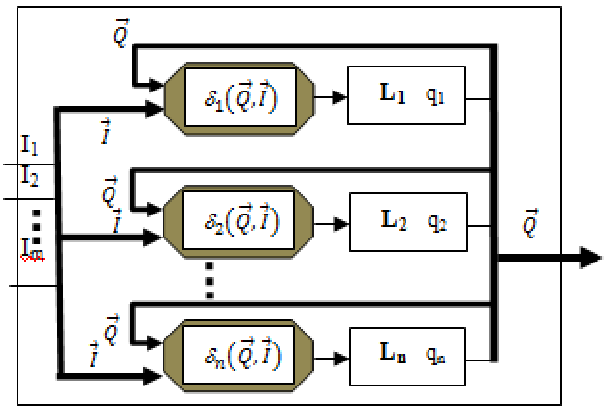

Figure 2: Structure of the transition

The circuit transition function δ is disassembled using the following procedure:

Procedure 3: Disassemble the transition function δ

1. *For every latch L_i, calculate $S_i = \delta_i\left(\vec{Q}, \vec{I}\right)$ and E_{S_i} applying the previous procedures 1 and 2, in order to disassemble the transition function δ_i and to build its canonical monomials merging all the variables of $\vec{Q}$ and $\vec{I}$. For each latch L_i, we get:*

$$\begin{cases} E_{S_i} = \{M_1, M_2, \ldots M_{k_i}\} \\ \quad\quad and \\ S_i = \delta_i(\vec{Q}, \vec{I}) = \sum_{j=1}^{k_i} M_j \end{cases}$$

Where each M_j is a canonical monomial and k_i is the number of these canonical monomials in $\delta_i(\vec{Q}, \vec{I})$ disjunctive canonical form.

2. *Rewrite each canonical monomial using the commutativity of the AND operator, so that the input variables are left and the outputs of flip-flops right.*

3. *The transition function δ and its canonical monomials set E_δ are given by:*

$$\begin{cases} E_\delta = E_S = \bigcup_{i=1}^{n} E_{S_i} \\ \quad\quad and \\ \delta(\vec{Q}, \vec{I}) = S = (S_1, S_2, \ldots, S_n) \end{cases}$$

4.2.2Determination of the terminal vocabulary V_T and the set of states Q

In the section 4.2.1 above, we have calculated the set of canonical monomials $E_\delta = E_S$ and the normal form of the transition function $\delta = S$. Each canonical monomial is actually made up of two parts, namely, a conjunction of all the entry variables possibly complemented (prefix), and a conjunction of all latches output possibly complemented too (suffix). Which means:

$$\forall M \in E_\delta, \exists(v, q) \in \vec{I} \times \vec{Q} / M = vq\,;$$

Where v is a canonical monomial of the input-variables and q is a canonical monomial of latches output. The proof of this property is established by the fact that every canonical monomial of the transition function is built with all the sequential circuit input variables and latches outputs, possibly complemented individually.

Let's now show how to compute V_T and Q.

Procedure 4: Calculate terminal vocabulary V_T and the set of states Q

Let P_{E_δ} denote the set of the canonical monomial prefixes; that is, the vectors of the input variables, and S_{E_δ}, the set of suffixes of the canonical monomials (the vectors of the outputs of the flip-flops). In fact, in each canonical monomial $M = vq$, the vector of the input variables is nothing but a possible input value, that is, the code of an input data; Similarly, the vector of the outputs of the flip-flops is nothing but a possible state value, that is, the code of a state. Therefore:

$$\begin{cases} P_{E_\delta} = \{v \in \vec{I} / \exists q \in \vec{Q}, vq \in E_\delta\} \\ S_{E_\delta} = \{q \in \vec{Q} / \exists v \in \vec{I}, vq \in E_\delta\} \end{cases} \Rightarrow \begin{cases} P_{E_\delta} = V_T \\ S_{E_\delta} = Q \end{cases}$$

We have thus determined V_T the terminal vocabulary and Q the set of states of the finite automaton associated with our sequential circuit.

4.2.3 Determination of initial state q_0

The initial state q_0 of a sequential circuit is the state taken by that circuit when it is turned on. This state is encoded by the initial content of each latch L_i; knowing that for all n flip-flops, the initial value is set by the latches SET (put in 1) or RESET (put in 0) inputs.

Procedure 5: *Determine the initial state q_0*

The initial state q_0 is coded in the n flip-flops according to the preset of SET or RESET entries:

$$q_0 = b_1 b_2 \ldots b_n / \begin{cases} Set(L_i) \Rightarrow b_i = 1 \\ Reset(L_i) \Rightarrow b_i = 0 \end{cases}$$

In this notation, $Set(L_i)$ stands for "entry SET of latch L_i" is connected to 1, as well as $Reset(L_i)$ means that "entry RESET of latch L_i" is connected to 0. Note that: $q_0 \in Q$ is a canonical monomial, because the initial state of a sequential circuit is always counted among the states. Therefore, it is always associated with at least one initial input data (value) in order to determine the successor of initial state. At least one canonical monomial $v_0 q_0$ resulting from this association is thus found in E_δ where $v_0 \in P_{E_\delta}$. Therefore:

$$\begin{cases} v_0 q_0 \in E_\delta \\ v_0 \in P_{E_\delta} \quad \Rightarrow q_0 \in Q \\ q_0 \in S_{E_\delta} \end{cases}.$$

4.2.4 Determination of the final state set F

In principle in sequential circuits, each state eventually produces an output value, because these circuits are seldom used to implement sequences or word recognition automats. In this case we can consider that: $F = Q$. However, a sequential circuit can be associated with a special output function ω, proceeding with input variables and the output of the flip-flops. This necessitates disassembling ω to obtain E_ω a set of canonical monomials from which to deduce F :

$$F = \left\{ q \in Q / \exists v \in \vec{I}, vq \in E_\omega \right\}$$

In the following, without losing the generality, we assume that $F = Q$.

4.2.5 Determination of transition function μ

In this section, we focus on μ, the fifth element of the finite-state automaton quintuplet :
$A_C = < V_T, Q, q_0, F, \mu >$, associated with the sequential circuit $C = < Z, I, Y, \delta, \omega >$.

We have to calculate μ from δ the sequential circuit transition function; that is, we must determine the set of triplets: $(a,q,q') \in V_T \times Q \times Q / \mu(q,a) = q'$; which means: $\delta(q,a) = q'$. In other words, if the circuit is in the state q and it receives as input the data a, then it switches into the state q'.

At this step, we assume that, using above procedure 3, δ the sequential circuit transition function has been already decomposed in set of canonical monomials E_δ. We know that each of these monomials consists in the conjunction of an element of V_T (let us denote it a) and an element of Q (let us denote it q). Every canonical monomial $aq \in \vec{I} \times \overrightarrow{Q}$ (c.f. procedure 4) imposes on the circuit a new transition, that is to say a new state q'. We thus have for each canonical monomial two elements (a and q) of the triplet (a,q,q'). We must now determine the third element: q'. To do so, we must determine the state q' in which the circuit switches when it is in the state q and it receives the input data a.

Yet we know, according to procedure 3, that if a canonical monomial $aq \in E_\delta$ belongs to E_{S_i} (the canonical monomial set of a latch L_i), then monomial aq sets latch L_i to 1; elsewhere is sets L_i to 0. Gathering the values set by aq on each latch, we obtain a vector of bits which corresponds to the code of the state q' successor of state q with input data a. That is:

$$\delta(q,a) = q' \Rightarrow \mu(q,a) = q'.$$

In fact, q' corresponds to the right part of the truth table for the line of the monomial aq. The following algorithm formalizes that process.

Procedure 6: Determine the initial state

```
Vector $\vec{q}$;
        $\mu = \varnothing$;        // Empty set
        For-all $(aq \in E_\delta)$ do:
        For-all $(i \in [1-n])$ do:
    If $(aq \in E_{S_i})$
    Then $\vec{q}[i] = 1$;
    Else $\vec{q}[i] = 0$;
    Insert $(a, q, \vec{q})$ in $\mu$.
```

4.2.6 Minimal finite automaton associated with a sequential circuit

At this level, we assume that the finite automaton obtained in the preceding step is deterministic. If this is not the case, it is appropriate to apply known algorithms to make it deterministic [11] [20] [21] [22].

The algorithm for computing a minimal finite automaton associated with a deterministic finite automaton is well known in the literature [5] [6] [20] [21] [27] [28] [29]. We have adapted this algorithm to the specific case of a sequential circuit. But let us first present some theoretical rudiments.

1. Let $A = < V_T, Q, q_0, F, \mu >$ be a determinist finite-state automaton. We call *family of the words recognized from state* $q \in Q$, the set L_q such that:
$$L_q = \left\{ x \in V_T^* / \hat{\mu}(q,x) \in F \right\}.$$

In fact, this is the set of words recognized by the automaton: $A_q =< V_T, Q, q, F, \mu >, \forall q \in Q.$

2. Let $A =< V_T, Q, q_0, F, \mu >$ be a determinist finite-state automaton. We call *equivalence* or *congruence of Nerode* [13] [20], the relation η defined by:

$$\forall (q, q') \in Q^2, \eta(q) = q' \Leftrightarrow L_q = L_{q'}.$$

The relation η is an equivalence relation on Q, whose equivalence classes are denoted by $Q_{[\eta]}$. For any state $q \in Q$, we denote $[q]_\eta$, the equivalence class of q for the relation η. We denote $q \equiv q'_{[\eta]}$ to signify that q is equivalent to q' modulo η.

The equivalence of Nerode makes it possible in fact to determine groups (classes) of states, as in each group, the automata A_q associated with the states $q \in Q$ recognize the same language L_q. In other words, two states are equivalent, if the sequences recognized from one are the same as those recognized from the other. That is to say, formally:

$$\forall (q, q') \in Q^2, q \equiv q'_{[\eta]} \Leftrightarrow \forall x \in V_T^*, \hat{\mu}(q, x) = \hat{\mu}(q', x).$$

Procedure 7: Determine the minimal finite automaton associated with a sequential circuit

Let $A_C =< V_T, Q, q_0, F, \mu >$ denote a determinist finite-state automaton calculated from sequential circuit $C =< Z, I, Y, \delta, \omega >$ using the above procedures 3 to 6. The minimal finite-state automaton $A_{C[\eta]} =< V_T, Q_{[\eta]}, q_{0[\eta]}, F_{[\eta]}, \mu_{[\eta]} >$, associated with C, is processed as follows:

- $Q_{[\eta]}$ denoting the quotient set of the equivalence classes of the relation η on Q, is calculated as follows:

```
1. $Q_1 = \{F, Q - F\}$;   // Even if we assume that $F = Q$.
2. $Q_{i+1} = Q_i$;
   For-all ( $X \in Q_{i+1}$ ) do   // $X$ is a set.
   $X' = \varnothing$;
   $\forall (q, q') \in X^2 / \exists a \in V_T, \mu(q, a) \neq \mu(q', a) \Rightarrow$
       $q \notin X$   //remove q from X
       $q \in X'$   //insert q into X'
   Insert $X'$ in $Q_{i+1}$.
3. Repeat (2) until the sequence is stationed at a term k such that : $Q_{k+1} = Q_k$.
4. If $k$ is the term at which the sequence (2) is stationary then $Q_k$ is the Nerode partition, of which $Q_{[\eta]}$ is the quotient set [13] [20].
```

- To build $Q_{[\eta]}$ from Q_k, we keep one state $q_{[\eta]}$ from each equivalence class $[q]_\eta$ and suppress all the others states of $[q]_\eta$. This state is the quotient of that class. This is to say : $Q_{[\eta]} = \{ q_{[\eta]} \in Q / q_{[\eta]} \in [q]_\eta \}$.
- $q_{0[\eta]}$ is the quotient the of class of q_0.
- $F_{[\eta]}$ is the set of quotient of the final state classes: $F_{[\eta]} = \{ q_{[\eta]} \in F / q_{[\eta]} \in [q]_\eta \}$.
- The transition function $\mu_{[\eta]}$ is constructed by removing all the states and transitions from μ except quotient of classes, as follows:
$$\forall (q, q', a) \in Q^2 \times V_T, \mu_{[\eta]}(q_{[\eta]}, a) = q'_{[\eta]} \Leftrightarrow \mu(q, x) = q'.$$

Comments: *The calculation of the automaton is carried out in two steps. The first one consists in building the classes of equivalence Q_k, where k is the term at which the sequence 3-4 is stationary, and then in deducing $Q_{[\eta]}$. To do so:*

- *We start Q_1 with two classes: F and $M_Q - F$.* Note that often $M_Q = F$ in sequential circuits, in which case: $M_Q - F = \varnothing$.
- *To obtain Q_{i+1}, we proceed in a refinement of each of the equivalence classes of Q_i : if two states of the same class in Q_i have different transitions with the same terminal symbol, then they will be in two distinct classes in Q_{i+1}; this leads to a break-up of the class to which the two elements belonged.*
- *When this sequence is stationary (which is always the case because the number of states is finite), we obtain a partition of Q which is in fact a partition of Nerode whose quotient is $Q_{[\eta]}$.*

The second step consists in constructing the minimal automaton by exploiting $Q_{[\eta]}$ *; to do so, all the states belonging to the same equivalence class are grouped into a single state. The minimal automaton is thus obtained, which is unique for the associated sequential circuit. This uniqueness is established by the fact that the minimal automaton does not depend on the finite automaton from which it was calculated, but from the associated regular language.*

5. Application Example

Let $C =< Z, I, Y, \delta, \omega >$ be the sequential circuit in figure 3, whose states are coded by three latches L1, L2, L3 having respective outputs Q_1, Q_2, Q_3. We will apply the previous algorithms (procedures 3 to 7) to compute the minimal finite automaton $A_{C[\eta]}$ associated with this circuit. Provided: $A_{C[\eta]} =< V_T, Q_{Q[\eta]}, q_{0[\eta]}, F_{[\eta]}, \mu_{[\eta]} >$

Note that we have: $\begin{cases} \vec{I} = I \\ \vec{Q} = (Q_1, Q_2, Q_3) \end{cases}$

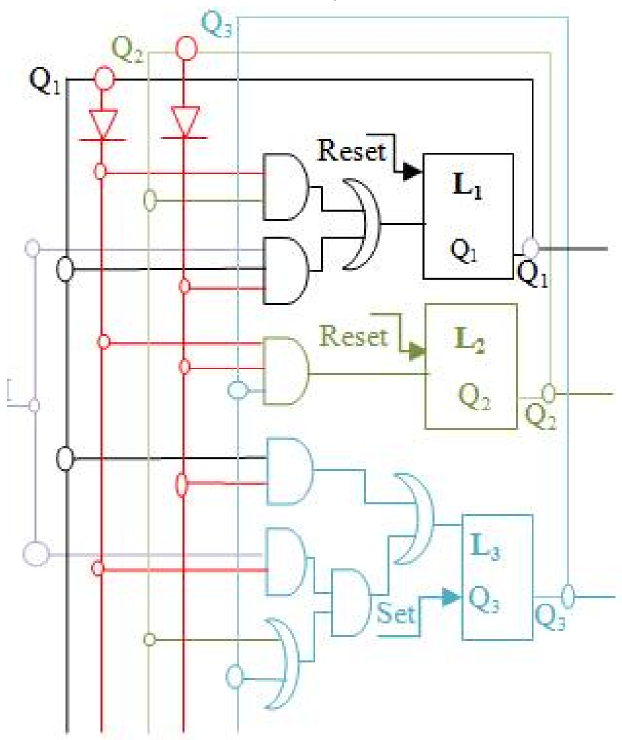

Figure 3: An example of sequential circuit

1. Disassembling δ the sequential circuit transition function:

   Let us calculate $\delta_1, \delta_2, \delta_3$ the canonical transition functions of latches L1, L2, L3 respectively:

$$\begin{aligned}
\delta_1 &= g_1 + g_1 = \bar{Q}_1 Q_2 + I Q_1 \bar{Q}_2 \\
&= I\bar{Q}_1 Q_2 + \bar{I}\bar{Q}_1 Q_2 + I Q_1 \bar{Q}_2 Q_3 + I Q_1 \bar{Q}_2 \bar{Q}_3 \\
&= I\bar{Q}_1 Q_2 Q_3 + I\bar{Q}_1 Q_2 \bar{Q}_3 + I\bar{Q}_1 Q_2 Q_3 + \\
&\quad \bar{I}\bar{Q}_1 Q_2 \bar{Q}_3 + I Q_1 \bar{Q}_2 Q_3 + I Q_1 \bar{Q}_2 \bar{Q}_3 \\
&\Rightarrow E_{\delta_1} = \{I\bar{Q}_1 Q_2 Q_3, I\bar{Q}_1 Q_2 \bar{Q}_3, I\bar{Q}_1 Q_2 Q_3, \\
&\qquad \bar{I}\bar{Q}_1 Q_2 \bar{Q}_3, I Q_1 \bar{Q}_2 Q_3, I Q_1 \bar{Q}_2 \bar{Q}_3\}
\end{aligned}$$

$$\delta_2 = \bar{Q}_1 \bar{Q}_2 Q_3 = I\bar{Q}_1 \bar{Q}_2 Q_3 + \bar{I}\bar{Q}_1 \bar{Q}_2 Q_3$$

$$\Rightarrow E_{\delta_2} = \left\{ I\bar{Q}_1 \bar{Q}_2 Q_3, \bar{I}\bar{Q}_1 \bar{Q}_2 Q_3 \right\}$$

$$\begin{aligned}
\delta_3 &= g_3 + g_4 = Q_1 \bar{Q}_2 + g_5 g_6 \\
&= Q_1 \bar{Q}_2 + I\bar{Q}_1 (Q_2 + Q_3) \\
&= Q_1 \bar{Q}_2 + I\bar{Q}_1 Q_2 + I\bar{Q}_1 Q_3 \\
&= I Q_1 \bar{Q}_2 + \bar{I} Q_1 \bar{Q}_2 + I\bar{Q}_1 Q_2 + I\bar{Q}_1 Q_3
\end{aligned}$$

We can now deduce the set of canonical monomials E_δ and the canonical transition function:

$$\begin{aligned}
E_\delta &= E_{\delta_1} \cup E_{\delta_2} \cup E_{\delta_3} \\
&= \{I Q_1 \bar{Q}_2 Q_3, I Q_1 \bar{Q}_2 \bar{Q}_3, \bar{I} Q_1 \bar{Q}_2 Q_3, \bar{I} Q_1 \bar{Q}_2 \bar{Q}_3, \\
&\qquad I\bar{Q}_1 Q_2 Q_3, I\bar{Q}_1 Q_2 \bar{Q}_3, I\bar{Q}_1 \bar{Q}_2 Q_3, \bar{I}\bar{Q}_1 Q_2 \bar{Q}_3, \\
&\qquad \bar{I}\bar{Q}_1 \bar{Q}_2 Q_3, \bar{I}\bar{Q}_1 Q_2 \bar{Q}_3, I Q_1 \bar{Q}_2 Q_3, I Q_1 \bar{Q}_2 \bar{Q}_3\} \\
&= I Q_1 \bar{Q}_2 Q_3 + I Q_1 \bar{Q}_2 \bar{Q}_3 + \bar{I} Q_1 \bar{Q}_2 Q_3 + \bar{I} Q_1 \bar{Q}_2 \bar{Q}_3 \\
&\quad + I\bar{Q}_1 Q_2 Q_3 + I\bar{Q}_1 Q_2 \bar{Q}_3 + I\bar{Q}_1 \bar{Q}_2 Q_3 \\
&\Rightarrow E_{\delta_3} = \{I Q_1 \bar{Q}_2 Q_3, I Q_1 \bar{Q}_2 \bar{Q}_3, \bar{I} Q_1 \bar{Q}_2 Q_3, \bar{I} Q_1 \bar{Q}_2 \bar{Q}_3, \\
&\qquad I\bar{Q}_1 Q_2 Q_3, I\bar{Q}_1 Q_2 \bar{Q}_3, I\bar{Q}_1 \bar{Q}_2 Q_3\}
\end{aligned}$$

And $\delta = (\delta_1, \delta_3, \delta_3)$.

2. Calculation of a finite-state automaton $A_C =< V_T, Q, q_0, F, \mu >$ associated with C:

   From above E_δ and δ, we have to deduce the quintuple $< V_T, Q, q_0, F, \mu >$:

   - Determining the terminal vocabulary V_T and the set of states Q. To this end, we apply the procedure 4. We decompose the elements of E_δ into couples $(\vec{I}\vec{Q})$, such as $\vec{I}$ is a vector of the input variables, and $\vec{Q}$ a vector of the latches output:

$$\begin{cases} V_T = \{I, \bar{I}\} \\ Q = \{\bar{Q}_1\bar{Q}_2Q_3, \bar{Q}_1Q_2\bar{Q}_3, \bar{Q}_1Q_2Q_3, \\ \quad Q_1\bar{Q}_2\bar{Q}_3, Q_1\bar{Q}_2Q_3\} \end{cases}$$

For sake of simplicity, we rename Q elements: $Q = \{q_1, q_2, q_3, q_4, q_5\}$

- Determining the initial state q_0 with procedure 5. According to figure 3, the initial configuration of the 3 latches is (RESET, RESET, SET); which gives vector (001) which is the code of state $\bar{Q}_1\bar{Q}_2Q_3$, renamed as q_1. Definitely, q_1 is the initial state of our finite-state automaton A_C.
- Determining the set of final states F. Since we have no precision on the nature of the automaton, we abstract the set of final states: $F = Q$.
- Calculation of the finite-state automaton transition function μ applying procedure 6:

Table 1: Transition function

$Q \setminus V_T$	$\bar{I}$	I
$\bar{Q}_1\bar{Q}_2Q_3$	$\bar{Q}_1Q_2\bar{Q}_3$	$\bar{Q}_1Q_2Q_3$
$\bar{Q}_1Q_2\bar{Q}_3$	$Q_1\bar{Q}_2\bar{Q}_3$	$Q_1\bar{Q}_2Q_3$
$\bar{Q}_1Q_2Q_3$	$Q_1\bar{Q}_2\bar{Q}_3$	$Q_1\bar{Q}_2Q_3$
$Q_1\bar{Q}_2\bar{Q}_3$	$\bar{Q}_1\bar{Q}_2Q_3$	$Q_1\bar{Q}_2Q_3$
$Q_1\bar{Q}_2Q_3$	$\bar{Q}_1\bar{Q}_2Q_3$	$Q_1\bar{Q}_2Q_3$

Table 2: Renamed state transition function

$Q \setminus V_T$	$\bar{I}$	I
q_1	q_2	q_3
q_2	q_4	q_5
q_3	q_4	q_5
q_4	q_1	q_5
q_5	q_1	q_5

Ultimately,

$$A_C =< \{I, \bar{I}\}_T, \{q_1, q_2, q_3, q_4, q_5\}, q_1, \{q_1, q_2, q_3, q_4, q_5\}, \mu >$$

with μ given by the above table.

3. Minimal finite-state automaton $A_{C[\eta]}$ associated with C using procedure 7:

- Determining the partition of Nerode: let's calculate the sequence (Q).

$$Q_1 = \{\{q_1, q_2, q_3, q_4, q_5\}\}$$
$$Q_2 = \{\{q_1\}, \{q_2, q_3, q_4, q_5\}\}$$
$$Q_3 = \{\{q_1\}, \{q_2, q_3\}, \{q_4, q_5\}\}$$
$$Q_4 = \{\{q_1\}, \{q_2, q_3\}, \{q_4, q_5\}\}$$

The sequence (Q) is stationary at step 3, so: $Q_k = Q_3$.

- Determining $Q_{[\eta]}$ the state set of $A_{C[\eta]}$: the quotient set of Q_3 is $\{q_1, q_2, q_4\}$. Therefore:
$$Q_{[\eta]} = \{q_1, q_2, q_4\}$$
- Determining $q_{0[\eta]}$ the initial state of $A_{C[\eta]}$: q_1 the initial state of A_C is a singleton in its class, so: $q_{0[\eta]} = q_1$.
- Determining $F_{[\eta]}$ the set state of final states of $A_{C[\eta]}$: since $F = Q$ according to our assumption, then $F_{[\eta]} = Q_{[\eta]} = \{q_1, q_2, q_4\}$.
- Determining $\mu_{[\eta]}$ the transition function of $A_{C[\eta]}$: To obtain $\mu_{[\eta]}$W, we reduce μ by replacing respectively q_3 by q_2 and q_5 by q_4; which give the following $\mu_{[\eta]}$ transition table:

Table 3: Reduced transition function

$Q_{[\eta]} \setminus V_T$	$\bar{I}$	I
q_1	q_2	q_2
q_2	q_4	q_4
q_4	q_1	q_4

- Finally:
$$A_{C[\eta]} =< \{I, \bar{I}\}, \{q_1, q_2, q_4\}, q_1, \{q_1, q_2, q_4\}, \mu_{[\eta]} >$$

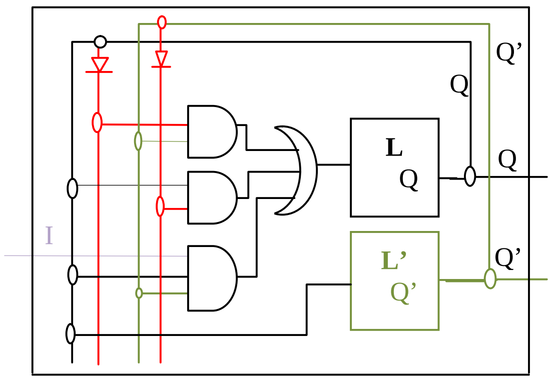

Figure 4: Minimal circuit

We thus notice the reduction of the number of states from 5 to 3 states, and then transitions from 10 to 6 transitions. A sequential circuit implementing this minimal automaton is given in figure 4, with 2 latches (despite of 3 in the original circuit), and the number of basic gates (AND, OR) was reduced from 9 to 4.

6. CONCLUSION

We have presented a technique for computing the minimal finite automaton associated with a sequential circuit whose states are coded by a set of flip-flops. The advantage of this contribution is to enable the designers of the sequential circuits to take advantage of the powerful tools developed by the language theorists, with a view of simplifying the complexity of the circuits. However, there are two shortcomings. The first is that it does not cover the case of token machines, widely used for the implementation of sequencers; it requires adaptation for this purpose. The second is that often each state of a sequential circuit generates a set of information useful to other circuits such as control commands, data, or synchronization signals. The suppression of the states and the transitions induced by the computation of the minimal finite automaton will lead to the loss of the information generated by the said states and transitions. Solutions are being studied to solve these two problems in order to complete the present contribution.

Bibliography

[1] ABDEL-HAMID A. T., ZAKI M., TAHAR S.: A Tool Converting Finite State Machine to VHDL, Proc. of IEEE Canadian Conference on Electrical & Computer Engineering (CCECE'04), Niagara Falls, pp. 1907--1910, 2004

[2] ALMEIDA M., MOREINA N., REIS R. : On the performance of automata minimization algorithms, Technical Report Series: DCC-2007-03, Version 1.0 June 2007, Departamento de Ciência de Computadores, Laboratorio de Inteligência Artificial e Ciência de Computadores, Universidade do Porto, Portugal, http://www.dcc.fc.up.pt/dcc/Pubs/TReports/TR07/dcc-2007-03.pdf.

[3] AYVAZYAN V., REHLICH K., SIMROCK S. N., STURM N.: Finite State Machine Implementation to Automate RF Operation at the TESLA Test Facility, Proceedings of the 2001 Particle Accelerator Conference, Chicago, pp. 286-288

[4] BELLOT P., SAKAROVITCH J.: Logique et Automates, Manuel d'Informatique, Ellipses,1998

[5] BRZOZOWSKI J. A.: Canonical regular expressions and minimal state graphs for definite events, in J. Fox, editor, Proc. of the Sym. on Mathematical Theory of Automata, volume 12 of MRI Symposia Series, pages 529–561, NY, 1963. Polytechnic Press of the Polytechnic Institute of Brooklyn., http://www.cs.hmc.edu/~keller/cs60book/12%20Finite-State%20Machines.pdf

[6] CARTON O.: Langages formels, Calculabilité et Complexité, 2007/2008, Formation interuniversitaire en informatique de l'École Normale Supérieure, http://www.liafa.jussieu.fr/~carton/Enseignement/Complexite/ENS/Support/.

[7] Chomsky N.: Three models for the description of language », IRE, Transactions on Information Theory, n^{o} 2, 1956, p. 113–124

[8] CUMMINGS C. E.: Coding And Scripting Techniques For FSM Designs With Synthesis-Optimized, Glitch-Free Outputs', SNUG-2000, Boston, MA, 2000.

[9] CUMMINGS C. E.: Synthesizable Finite State Machine Design Techniques Using the New SystemVerilog 3.0 Enhancements, SNUG-2003, San Jose, CA, 2003.

[10] DUFFNER S.: Qfsm - The Finite State Machine Designer, 2015. [Online]. Available: http://qfsm.sourceforge.net/

[11] GUREVICH Y.: Sequential Abstract State Machines Capture Sequential Algorithms, Microsoft Research, ACM Transactions on Computational Logic, vol. 1, no. 1 (July 2000), pages 77-111. http://research.microsoft.com/en-us/um/people/gurevich/Opera/141.pdf

[12] HOPCROFT J.: An N LOG N algorithm for minimizing states in finite automaton, STAN-CS-71-190, January 1971, COMPUTER SCIENCE DEPARTMENT, School of Humanities and Sciences, STANFORD UNIVERSITY

[13] HOPCROFT J., MOTWANI R., ULLMAN J. D.: Introduction to Automata Theory, Languages and Computation, Addison Wesley, 2000.

[14] HUFFMAN D. A.: The synthesis of sequential switching circuits, The Journal of Symbolic Logic, 20(1):69–70, 1955.

[15] KAESLIN H.: Digital integrated circuits design, from VLSI architectures to CMOS fabrication", Cambridge University Press, http://books.google.ga/books?id=gdRStcYgf2oC&pg=PA787&dq=medvedev+fsm&hl=fr#v=onepage&q=medvedev%20fsm&f=false
[16] MEALY G.H.: A method for synthesizing sequential circuits, The Bell System Technical Journal, 34, 5, pp. 1045-1079, September 1955.
[17] MIT Open Course Ware: Introduction to Electrical Engineering and Computer Science, chap 4, April 25, 2011, 6.01SC, https://www.google.fr/url?sa=t&rct=j&q=&esrc=s&source=web&cd=13&ved=0ahUKEwi4t8u9habRAhXINhoKHTC0CZM4ChAWCCQwAg&url=https%3A%2F%2Focw.mit.edu%2Fcourses%2Felectrical-engineering-and-computer-science%2F6-01sc-introduction-to-electrical-engineering-and-computer-science-i-spring-2011%2Funit-1-software-engineering%2Fstate-machines%2FMIT6_01SCS11_chap04.pdf&usg=AFQjCNFIgVLi1nzzhRGo-QcR-D8YG4pxTQ&cad=rja
[18] MOORE E. F.: Gedanken-experiments on sequential machines, pp 129-153 in Shannon and McCarthy (eds.), Automata Studies, Annals of Mathematics Studies, Number 34, Princeton University Press, 1956.
[19] PATT Y. N., PATEL S. J.: Introduction to Computing Systems, from bits and gates to C and beyond, McGraw Hill International Editions, 2001
[20] PERRIN D.: Finite automata, in J. van Leeuwen, editor, Handbook of Theoretical Computer Science, volume B, chapter 1, pages 1–57. Elsevier, 1990
[21] STERN J.: Fondements mathématiques de l'informatique, McGraw-Hill, 1990.
[22] STRAUBING H.: Finite Automata, Formal Logic and Circuits Complexity, Progress in theoretical computer science. Birkhauser, 1994.
[23] TCHOUMATCHENKO V. P.: Web Based Tool for State Machines Design, ANNUAL JOURNAL OF ELECTRONICS, 2015, ISSN 1314-0078, PP 69-71
[24] TEMAN O.: Architecture des Ordinateurs, http://pages.saclay.inria.fr/olivier.temam/teaching/x/texts/text_01_05.pdf
[25] UMA R.: Qualitative Analysis of Hardware Description Languages: VHDL and Verilog (IJCSIS) International Journal of Computer Science and Information Security, Vol. 9, No. 4, pp-127-135, April 2011.
[26] UMA R., DHAVACHELVAN P.: Synthesis optimization for finite state machine design in FPGAS, International Journal of VLSI design & Communication Systems (VLSICS) Vol.3, No.6, December 2012, DOI : 10.5121/vlsic.2012.3607 79
[27] WATSON B. W., DACIUK J.: An efficient DFA minimization algorithm, Natural Language Engineering, pages 49–64, 2003.
[28] WEATHERSPOON H.: State and Finite State Machines, CS 3410, Spring 2013, Computer Science, Cornell University
[29] WRIGHT D.R. : Finite State Machines, CSC216, (Summer 2005), http://www4.ncsu.edu/~drwrigh3/docs/courses/csc216/fsm-notes.pdf
[30] ZIMMER P.: Fizzim – an open-source fsm design environment, 2015, Available: http://www.fizzim.com/

AUTHOR

Born in 1958 in Gabon, Pierre MOUKELI MBINDZOUKOU received a doctorate in computer science in 1992 from Claude Bernard University, Lyon – France. From 1988 to 1992, he was a researcher-student member of LIP (Laboratoire de l'Informatique du Parallélisme) at ENS-Lyon (Ecole Normale Supérieure de Lyon). Back to Gabon, he joined the African Institute of Computer Science (IAI), an inter-African engineer-degree school, where he is teacher and a researcher at the LAIMA (Laboratoire Africain d'Informatique et de Mathématiques Appliquées – IAI). He is also a teacher at INPTIC (National Institute of Post and Information and Communication Technologies) at Libreville – Gabon. Since 2009, Pierre MOUKELI is Adviser in Computer Science of the President of the Republic of Gabon; and since January 2017, he helds these positions with the post of Director of Education of the African Institute of Computer Science (IAI). Among other activities, he is a consultant and auditor with public and private Gabonese companies. His research topics include parallel computing, operating system, language theory, document processing and e-Government.