# A Simulator to Train for Finite State Machine Design

Domenico Ponta, Giuliano Donzellini
Department of Biophysical and Electronic Engineering, University of Genoa
Via Opera Pia 11A I-16145 Genoa - Italy
Phone +39-10-3532759, fax +39-10-3532175
E-mail ponta@dibe.unige.it, donzie@dibe.unige.it

## Abstract

*The paper presents a general purpose simulator of Finite State Machines (FSM), developed for training purposes. Algorithms, represented graphically as Algorithmic State Machine charts, can be drawn directly on the computer screen, and directly tested in the state and time domains without the necessity of synthesizing the FSM logic circuits. The simulator facilitates the transition from the pedagogical level of digital design, to the professional one, characterized by the use of CAD tools. It is currently used in introductory courses of digital electronics for several EE curricula.*

## Introduction: Teaching Finite State Machines

In a sequential network, the output is a function not only of the current inputs but also of a *state* that takes into account the *history* of previous inputs. Modern digital circuit design is essentially based on sequential logical networks, and their study is the most relevant issue in an introductory course on digital electronics. For teaching this kind of design, a formal and systematic approach is better than an intuitive one, because it insures quality e repeatability of the results. All the considerations that will be made in the following, refer to a pedagogical scenario where the learners have no previous introduction to digital electronics.

Design of digital systems is nowadays supported by several formal design methods, based essentially on the use of circuit description languages (HDL, VHDL). With these languages, sequential networks are conceived and designed as Finite State Machines and described with statements similar to the ones used by general purpose software programming languages.

We are convinced, though, that a pedagogical approach based on hardware description languages does not provide to the students the right background for developing their skills in digital circuits design. It is necessary, to our opinion, to keep a closer relation between the algorithm to be implemented and the FSM that implements it.

It is important, for example, to maintain, since the very beginning of the pedagogical action, a strict correlation between state sequences and time domain behavior of input/output signal of the system under consideration. A FSM state must be, more than an abstract concept, an observable logical condition of a network.

For such reason we believe it is still necessary to represent the FSM algorithms with graphical methods instead of moving up to a textual description language. In our course (Digital Systems Electronics, first digital design course of the curricula in Electronic Engineering and Computer Science) we use the ASM (Algorithmic State Machine) [1] method, applied to Moore and Mealy machines [2], as detailed in the following.

The use of a simulator especially conceived for pedagogical purposes has proven to be a useful support for the course, even in the first lectures introducing the fundamental concepts of FSM. In this paper we provide a short functional description of SIMFSM, conceived and developed within a more general strategy for using computer-based learning tools in the field of digital electronics [3]. The simulator is installed in the digital electronics lab, where students can access it or make a copy for personal use at home.

## Learning to design a Finite State Machine

The simulator presented here is the final step of a comprehensive computer-assisted methodology for teaching the design of Finite State Machines [4], founded on “learning tools” specifically developed for this purpose using Asymetrix ToolBook.

With the use of OpenScript, ToolBook programming language, the FSM algorithm can be simulated and associated to the ASM chart, that therefore becomes “alive”. The animated chart allows the learner to verify the evolution of the machine algorithm in the domains of states and time by setting the proper inputs and stimulating the FSM with clock and input signals. The logical networks’ schematics are designed to respond to input stimuli directly on the computer screen. Such representation of FSM algorithms has been used extensively in the course and has proved to be better than the paper-based traditional one, helping the student to understand the behavior of a given ASM [5].

Nevertheless, understanding an algorithm does not necessary mean to be able to develop one. A major challenge is therefore introducing the learner to the conception of the FSM algorithm. Development of this skill, that relies heavily on the designer intuition and

experience, is helped by the availability of an open simulation tool.

The first part of the academic course is targeted toward building the student's capability of applying the ASM method to well-known sequential structures, such as basic elements like flip-flops, registers, counters and other simple devices. The student is therefore made familiar with the design method and prepared for the next phase where he will make an active use of the ASM, designing non-standard digital structures described by a set of specifications. This active phase, consisting in the drawing of the ASM chart and its formal synthesis, develops the student's design capabilities.

Because in the real world of digital design very seldom a problem has a unique solution, the problem arises of verifying the validity of the specific implementation. There is no doubt that this essential part of the design job is often neglected in the educational environment, given the amount of time and energies required. As a result, many design flaws escape the student's attention.

Typically, state and output races belong to this category of unnoticed mistakes. In the traditional teaching environment the only solution would be to increase the interaction with the students and the time spent assisting them. The large number of electronic engineering students is often a very serious obstacle. The use of the FSM simulator takes care of the non-creative phases of design, making easier the verification of the network behavior.

With the introduction of a general purpose simulator the learners achieves a larger degree of independence, because they are able to design and test autonomously an unlimited number of networks of their own conception and, most important, to verify the correctness of their assumptions. On the same time, the use of a digital simulator represents a significant step on the way of getting acquainted with CAD techniques [6]. In our course the logic network simulator, conceived for pedagogical applications, is designed to facilitate the transition to a professional digital simulator, which is achieved in the last phase of the learning process.

## The Finite State Machine Simulator

The final stage of learning the FSM is achieved with the use of a general purpose Finite State Machine simulator, a tool developed in Microsoft Visual Basic with the purpose of allowing the transition from understanding a given FSM to designing one according to given specification. It represents also a transition from an essentially pedagogical tool (the animated FSM) to a quasi-professional design tool and, as such, a good introduction to CAD methodologies.

The simulator needs to be easy and intuitive to use, in line with the competence and ability that the student is building day by day. A professional digital simulator does not fit this necessity, because its many options and advanced features, necessary for the activity of the accomplished digital designer, are instead a serious obstacle for the beginner student.

Our FSM simulator is written in Microsoft Visual Basic and takes advantage of the Windows graphical interface. It is made of three main integrated modules: the graphic editor, the logic simulator and the map synthesizer, that are described in the following sub-sections.

### ASM Chart Editor

The graphic editor allows the construction of the FSM diagram, according to the rules of the ASM method (Figures 1, 2). An ASM diagram describes the output functions and the next-state functions of a state machine. It is composed of three basic elements, the state (rectangular box), the decision box (diamond) and the conditional output box (rounded rectangle). An ASM block is made of one state box, decision boxes and conditional output boxes. It has one entrance, but may have any number of exit paths, each of them connecting to a state box.

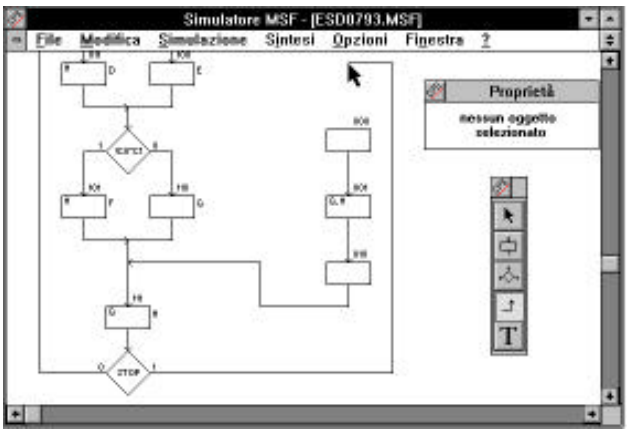


*Figure 1 ASM charts are drawn by dragging on the screen and connecting together the building blocks.*

The connection of several ASM blocks produces the ASM chart that describes the FSM algorithm in a way similar to the flow chart that is used for software algorithms, with the fundamental difference that each state block of the ASM represents a state of the machine, while the concept of state is not applicable to flow charts.

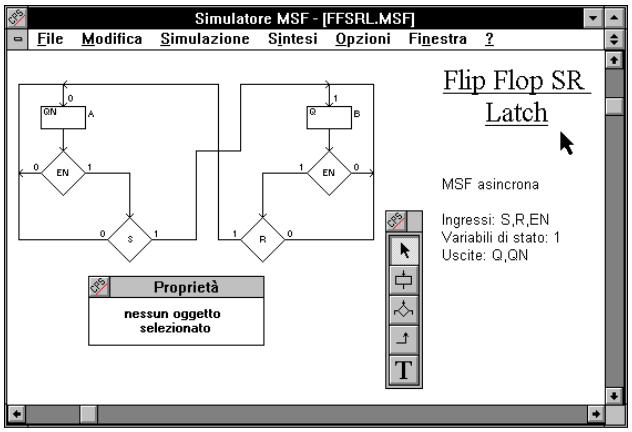


*Figure 2 The ASM chart of an asynchronous FSM, a Latch-type Flip-Flop.*

The editing phase, in which the ASM diagram is created, is assisted by a toolbox that contains the drawing tools, a window that shows the properties of the objects that have been selected and a syntactic/semantic corrector that identifies the most common mistakes, such as the diagram inconsistency, the duplication of names and codes, and insufficient number of state variables. A specific algorithm in charge of eliminating critical races is available. The editor will provide defaults for information not entered by the student, like names and number of state variables, allowing to proceed in the process. All logical expressions and variables' names entered in the ASM chart must be defined in this window (Figure 3). Later modifications are possible. The construction of the ASM chart is made by inserting the appropriate blocks, using the mouse and toolbox's buttons, and then interconnecting them. Captions and comments can also be inserted. ASM charts drawn with the editor can be exported and used in other applications.

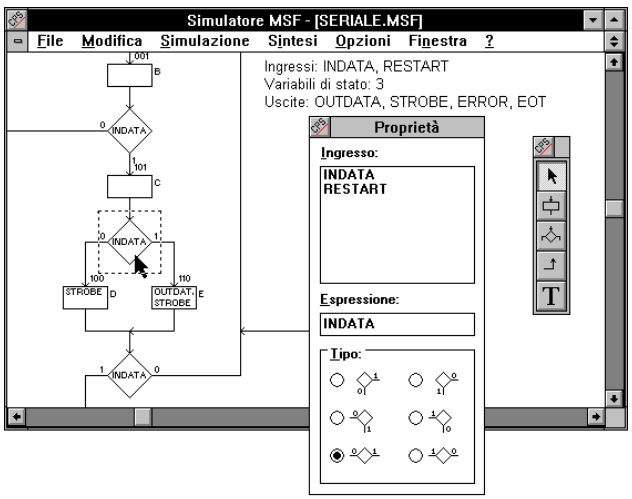


*Figure 3 Definition of input and outputs: in the figure the input INDATA is created on the ASM.*

## Logic simulator

This section of SIMFSM simulates the algorithm described by the ASM chart and presents it to the learner as a sequence of states or a timing diagram of the FSM state and output variables. The simulation is calculated by software directly from the ASM chart. This is an innovative feature, especially relevant in the pedagogical context, because it allows the preliminary simulation of the ASM chart, before going to the synthesis. The student can study the state sequence without the complication of the network synthesis, as it would be the case using a professional digital simulator.

To emphasize the relation between a state diagram symbology and time tracing, the current state and the path to the next one are highlighted also in the window containing the ASM chart (Figure 4). The resulting effect of visual animation is especially effective. The quality of animation is enhanced by another feature, the low-speed continuous clock generation. In such mode the learner can interact with the machine in "real time" by modifying the inputs while the algorithm is running.

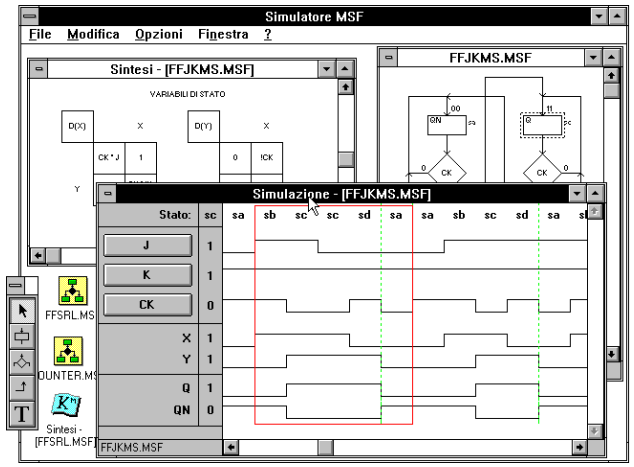


*Figure 4 The time-domain behavior of a JK Master Slave Flip Flop. In the background, the windows with the ASM chart and its synthesis.*

The simulator, therefore, fulfills the pedagogical need to demonstrate to the learner the relations between state sequence and time evolution (Figure 5), one of the most difficult issues to master for the beginner student.

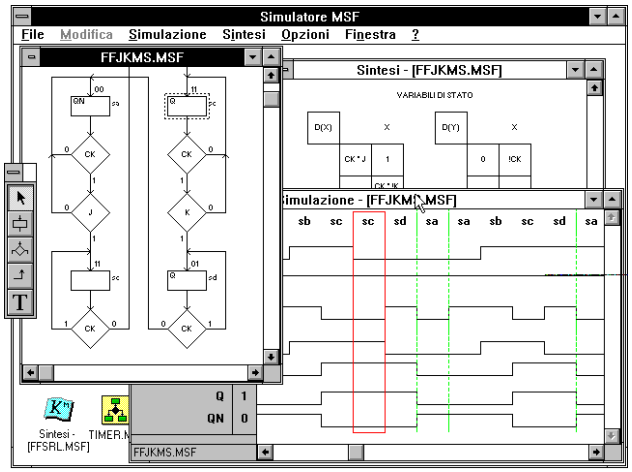


*Figure 5 State and time domains windows presented in the same screen.*

**FSM synthesis**

Derivation of next state equations directly from the ASM diagram is a standard procedure similar to the synthesis of a combinational network from truth table. The logic synthesizer(Figure 4, background) provides next state and output Karnaugh maps (K-maps) and equations from the FSM diagrams. The student has the choice to fill the K-maps manually, deriving the next state equations from the ASM chart, or to get them filled by the synthesizer. In any case, the simulator knows the solution and is therefore able to check step-by-step the validity of the Boolean expressions entered in the K-maps. The student then proceeds to derive the minimized expressions of next state and output equations. Boolean equations can be exported, to be used externally.

The synthesizer allows to choose the desired flip-flop type for the state register, to allow the student to compare the resulting synthesis. The module can update the maps if the ASM chart is modified. A context-sensitive Help guides the learner through all simulator operations.

## Evaluation and conclusions

An objective evaluation of the effectiveness of SIMFSM as learning tool goes beyond our possibilities. The same problem is common to most, if not all, attempts to introduce CBL in the field of engineering education, especially when, as it is in our case, CBL is tested in parallel with traditional education.

A possible empirical measure of validation could be found by identifying the effectiveness of a product with the extent of its use. Under this point of view, we can state without doubts that the simulator has been so far the single most successful piece of our courseware for digital electronics. Students use it not only to design new state machines, but also to check the correctness of their work and exercises. The editor has become a common tool for drawing ASM charts.

A major limitation of SIMFSM is its “closeness”, i.e. the impossibility of adding external components to the FSM or to embed the simulated FSM in a more general digital systems. This limitation is being removed by a current project that integrates into a single tool, SYSSIM (system simulator), the three existing tools SIMFSM, a digital simulator and a microprocessor emulator.

## Link:

- Finite State Machine Simulator

## Acknowledgments

The ASM chart editor and simulator have been developed by Fabio Terrile and the synthesis and minimisation algorithms by ClaudioCastellini.

## References

1. Clare, C.R. “Designing Logic Systems using State Machines”, MacGraw-Hill, 1973.
2. Tinder, R.F., Digital Engineering Design, Prentice-Hall International, 1991.
3. Ponta, D., Donzellini, G., “Learning Electronics with Hypermedia and Computer Tools”. Proceedings of CALISCE ‘94, International Conference on Computer Aided Learning and Instruction in Science and Engineering. Telecom Paris, France, 1994.
4 Ponta D., Donzellini G., Hypertext and Computer Tools for Teaching Finite State Machines, Proceedings of Hypermedia in Vaasa ‘94 Conference on Computers and Hypermedia in Engineering Education, M. Linna and P. Ruotsala editors,Vaasa, Finland, June 8-11, 1994.
5. Schank, R.C., “Active Learning through Multimedia”, IEEE MultiMedia, Spring 1994, pp.69-78, 1994.
6. Ponta, G. Donzellini, G. Parodi, An Integrated Computer-Based Course bridges the gap between design theory and professional CAD for digital electronic systems, proceedings of IFIP WG 3.4 International Working Conference,Soest, Germany, 12-17 July 1993