iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 2 **Issue:** IX **Month of publication:** September 2014

DOI:

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

ISSN: 2321-9653

INTERNATIONAL JOURNAL FOR RESEARCH IN APPLIED SCIENCE AND ENGINEERING TECHNOLOGY (IJRASET)

Finite State Machines

Sakshi Ahuja[#1], Anjali Rajput[*2], Dipti Bhardwaj[#3]

[#]*CSE-CSE, Maharishi Dayanand University*

Abstract— Finite state machines (FSMs) are a common presence in digital circuit design. However, they can be very useful also for the software developer. Actual operating systems and application software are event-based and communication issues play a big role; these fields can be more easily handled with software based on finite state machines - software that is simpler and easier to understand, debug and modify. Embedded systems' software can also benefit from state machines because of their efficient way of using the limited resources of the system. The paper presents some basic concepts of finite state machines, some typical applications, with focus on Web technologies (modem control, FTP - File Transfer Protocol, remote access via Telnet console) and some implementation issues - programming finite state machines in Delphi for Windows, in microcontroller assembly language and C. Latest trends are also analyzed - the hardware implementation of state machines in silicon, like the new Texas Instruments MSP430 series of low power microcontrollers. These electronic packages offer some features like reduced power consumption, a single chip solution for complex applications and high functional flexibility.

Keywords— FSM(Finite State Machine), Mealy, Moore,

I. INTRODUCTION

A finite-state machine (FSM), or simply a state machine, is a mathematical model of computation used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of states. The machine is in only one state at a time; the state it is in at any given time is called the current state. It can change from one state to another when initiated by a triggering event or condition; this is called a transition. A particular FSM is defined by a list of its states, and the triggering condition for each transition.

Finite-state machines can model a large number of problems among which are electronic design automation communication protocol design, language parsing and other engineering applications. In biology and artificial intelligence research, state machines or hierarchies of state machines have been used to describe neurological systems and in linguistics to describe the grammars of natural languages. Considered as an abstract model of computation, the finite state machine is weak; it has less computational power than some other models of computation such as the Turing machine. That is, there are tasks which no FSM can do, but some Turing machines can. This is because the FSM has limited memory. The memory is limited by the number of states.

Finite State Machines are mainly of two types:-

1. Deterministic -- Deterministic FSM, meaning that given an input and the current state, the state transition can be predicted.
2. Non-Deterministic -- Non-deterministic finite state machine. This is where given the current state; the state transition is not predictable. It may be the case that multiple inputs are received at various times, means the transition from the current state to another state cannot be known until the inputs are received (event driven).Example : Use a random number generator to select a triggered rule.

II. INFORMAL DESCRIPTION

A finite state machine is usually specified in the form of a transition table, much like the one shown in Table 8.1 below.

TABLE II.1

CONDITION		EFFECT	
CURRENT STATE	IN	OUT	NEXT STATE
q0	–	1	q2
q1	–	0	q0
q2	0	0	q3
q2	1	0	q1
q3	0	0	q0
q3	1	0	q1

For each control state of the machine the table specifies a set of transition rules. There is one rule per row in the table, and usually more than one rule per state. The example table contains transition rules for control states named q0, q1, q2, and q3.

Each transition rule has four parts, each part corresponding to one of the four columns in the table. The first two are conditions that must be satisfied for the transition rule to be executable. They specify-

- The control state in which the machine must be
- A condition on the ‘‘environment’’ of the machine, such as the value of an input signal.

The last two columns of the table define the effect of the application of a transition rule. They specify-

- How the ‘‘environment’’ of the machine is changed, e.g., how the value of an output signal changes.
- The new state that the machine reaches if the transition rule is applied.

In the traditional finite state machine model, the environment of the machine consists of two finite and disjoint sets of signals: input signals and output signals. Each signal has an arbitrary, but finite, range of possible values. The condition that must be satisfied for the transition rule to be executable is then phrased as a condition on the value of each input signal, and the effect of the transition can be a change of the values of the output signals. The machine in Table II.1 illustrates that model. It has one input signal, named In, and one output signal, named Out.

A dash in one of the first two columns is used as a shorthand to indicate a ‘‘don’t care’’ condition (that always evaluates to the Boolean value true). A transition rule, then, with a dash in the first column applies to all states of the machine, and a transition rule with a dash in the second column applies to all possible values of the input signal. Dashes in the last two columns can be used to indicate that the execution of a transition rule does not change the environment. A dash in the third column means that the output signal does not change, and similarly, a dash in the fourth column means that the control state remains unaffected. In each particular state of the machine there can be zero or more transition rules that are executable. If no transition rule is executable, the machine is said to be in an endstate. If precisely one transition rule is executable, the machine makes a deterministic move to a new control state. If more than one transition rule is executable a nondeterministic choice is made to select a transition rule. A nondeterministic choice in this context means that the selection criterion is undefined.

III. TURING MACHNES

The above definition of a finite state machine is intuitively the simplest. There are many variants of this basic model that differ in the way that the environment of the machines is defined and thus in the definition of the conditions and the

effects of the transition rules. For truly finite state systems, of course, the environment must be finite state as well (e.g., it could be defined as another finite state machine). If this

requirement is dropped, we obtain the well-known Turing Machine model. It is used extensively in theoretical computer science as the model of choice in, for instance, the study of computational complexity. The Turing machine can be seen as a generalization of the finite state machine model, although Turing's work predates that of Mealy and Moore by almost two decades.

The ''environment'' in the Turing machine model is a tape of infinite length. The tape consists of a sequence of squares, where each square can store one of a finite set of tape symbols. All tape squares are initially blank. The machine can read or write one tape square at a time, and it can move the tape left or right, also by one square at a time. Initially the tape is empty and the machine points to an arbitrary square. The condition of a transition rule now consists of the control state of the finite state machine and the tape symbol that can be read from the square that the machine currently points to. The effect of a transition rule is the potential output of a new tape symbol onto the current square, a possible left or right move, and a jump to a new control state.

IV. COMMUNICATING FINITE STATE MACHINES

We assume that signals have a finite range of possible values and can change value only at precisely defined moments. The machine executes a two-step algorithm. In the first step, the input signal values are inspected and an arbitrary executable transition rule is selected. In the second step, the machine changes its control state in accordance with that rule and updates its output signals. These two steps are repeated

forever. If no transition rule is executable, the machine will continue cycling through its two-step algorithm without changing state, until a change in the input signal values, effected by another finite state machine, makes a transition possible. A signal, then, has a state, much like a finite state machine. It can be interpreted as a variable that can only be evaluated or assigned to at precisely defined moments.

We can build elaborate systems of interacting machines in this way, connecting the output signals of one machine to the input signals of another. The machines must share a common ''clock'' for their two-step algorithm, but they are not otherwise synchronized.

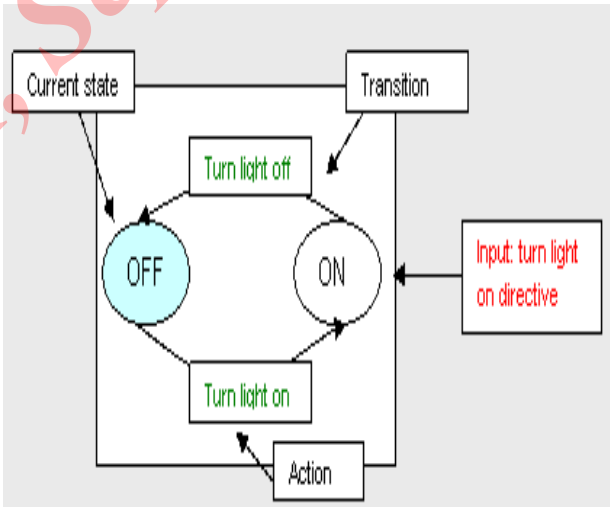
If further synchronization is required, it must be realized with a subtle system of handshaking on the signals connecting the machines. Most systems provide a designer with higher-level synchronization primitives to build a protocol. An example of such synchronization primitives are the send and receive operations defined in PROMELA.

V. MEALY AND MOORE NACHINES

Both these machine types follow the basic characteristics of state machines, but differ in the way that outputs are generated.

1. Mealy Machines -- Outputs are independent of the inputs, i.e., outputs are effectively produced from within the state of the state machine.

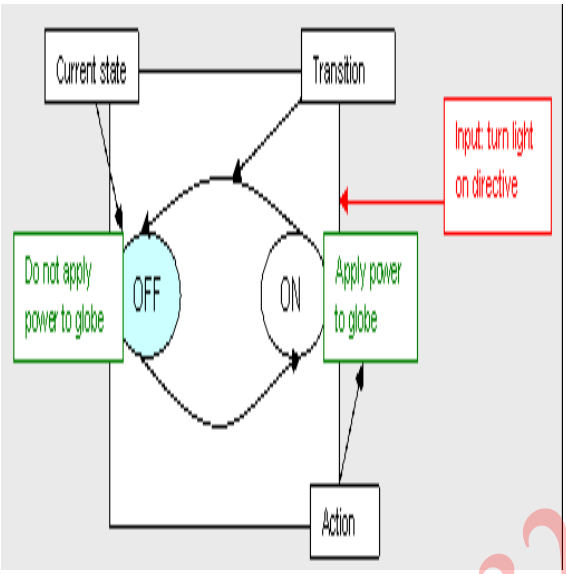
   Outputs are generated as products of the transitions between states. In example the light is affected by the process of changing states.



2. Moore Machines -- Outputs can be determined by the present state alone, or by the present state and the present inputs, ie outputs are produced as the machine makes a transition from one state to another.

   Outputs are generated as products of the states.

   In this example the states define what to do ; such as apply power to the light globe.



Moore and Mealy FSMs can be functionally equivalent

Mealy FSM has richer description and usually requires

smaller number of states, smaller circuit area. Mealy FSM computes outputs as soon as inputs change.

Mealy FSM responds one clock cycle sooner than

Equivalent Moore FSM.Moore FSM Has No

combinational path between inputs and outputs.

Moore FSM is less likely to have a shorter critical

Path.

VI. ADVANTAGES AND DISADVANTAGES

Simple, Predictable (deterministic FSM) - given a set of inputs and a known current state, the state transition can be predicted, allowing for easy testing. Due to their simplicity, FSMs are quick to design, quick to implement and quick in execution. FSM is an old knowledge representation and system modeling technique, and its been around for a long time, as such it is well proven even as an artificial intelligence technique, with lots of examples to learn from. Easy to transfer from a meaningful abstract representation to a coded implementation.

The predictable nature of deterministic FSMs can be unwanted in some domains such as computer games (non-DFSM tries to solve this). The conditions for state transitions are ridged, meaning they are fixed. Not suited to all problem domains, should only be used when a systems behavior can be decomposed into separate states with well defined conditions for state transitions. This means that all states, transitions and conditions need to be known up front and be well defined !!

AI example: Computer Game:

The goal: to use a computer game to illustrate the conceptual workings of a FSM based on a practical rather than theoretical implementation. First person computer game called Quake, under GNU General Public License. Quake makes extensive use of FSMs as a control mechanism governing the entities that exist in the game world. Quake is a good example, and a good learning tool that can show the power of both very simple finite state machines such as the rocket, and slightly more complex FSM made up of a hierarchy of FSM and motivated by goals, such as the Shambler monster.

VII. EXTENDED FINITE STATE MACHINES

The finite state machine models we have considered so far still fall short in two important aspects: the ability to model the manipulation of variables conveniently and the ability to model the transfer of arbitrary values. These machines where defined to work with abstract objects that can be appended to and retrieved from queues and they are only synchronized on the access to these queues. We make three changes to this basic finite state machine model. First, we introduce an extra primitive that is defined much like a queue: the variable. Variables have symbolic names and they hold abstract objects. The abstract objects, in this case, are integer values. The main difference from a real queue is that a variable can hold only one value at a time, selected from a finite range of possible values. Any number of values can be appended to a variable, but only the last value that was appended can be retrieved.

The second change is that we will now use the queues specifically to transfer integer values, rather than undefined abstract objects. Third, and last, we introduce a range of arithmetic and logical operators to manipulate the contents of variables.

TABLE VII.1

CURRENT STATE	IN	OUT	NEXT STATE
q0	s0	–	–
q0	s1	–	q1
q0	s2	–	q2
q0	rv	–	r0
q0			□– 0
q1	s0	–	q0
q1	s1	–	–
q1	s2	–	q2
q1	rv	–	r1
r1	–	1	q1
q2	s0	–	q0
q2	s1	–	q1
q2	s2	–	–
q2	rv	–	r2
r2	–	2	q2

The extension with variables, provided that they have a finite range of possible values, does not increase the computational power of finite state machines with bounded FIFO queues. A variable with a finite range can be simulated trivially by a finite state machine. Consider the six-state machine shown in Table VII.1, that models a variable with the range of values from zero to two. The machine accepts four different input

messages. Three are used to set the pseudo variable to one of its three possible values. The fourth message, rv, is used to test the current value of the pseudo variable. The machine will respond to the message rv by returning one of the three possible values as an output message. Thus, at the expense of a large number of states, we can model any finite variable

without extending the basic model, as a special purpose finite state machine. The extension with explicit variables, therefore, is no more than a modeling convenience. Recall that the transition rules of a finite state machine have two parts: a condition and an effect. The conditions of the transition rules are now generalized to include boolean expressions on the value of variables, and the effects (i.e. the actions) are generalized to include assignment to variables.

An extended finite state machine can now be defined as a tuple (Q,q0 ,M,A,T), where A is the set of variable names. Q, q0, and M are as defined before. The state transition relation T is unchanged. We have simply defined two extra types of actions:

boolean conditions on and assignments to elements of set A. A single assignment can change the value of only one variable. Expressions are built from variables and constant values, with the usual arithmetic and relational operators.

In the spirit of the validation language PROMELA, we can define a condition to be executable only if it evaluates to true, and let an assignment always be executable. Note carefully that the extended model of communicating finite state machines is a finite state model, and almost all results that apply to finite state machines also apply to this model.

VIII. SUMMARY

The formal model of a finite state machine was developed in the early 1950s for the study of problems in computational complexity and, independently, for the study of problems in the design of combinatorial and sequential circuits. There are almost as many variants of the basic model of a finite state

machine as there are applications. For the study of protocol design problems we need a formalism in which we can model the primitives of process interactions as succinctly as possible. With this in mind we developed an extended finite state machine model that can directly model message passing and the manipulation of variables. Its semantics are closely linked to the semantics of PROMELA.

There are three main criteria for evaluating the adequacy of formal modeling tools: Modeling power, Analytical power , Descriptive clarity

The main purpose of the modeling is to obtain a gain in analytical power. It should be easier to analyze the model than it is to analyze the original system being modelled. We have chosen the finite state machine as our basic model. There is a small set of useful properties that can easily be established with a static analysis of finite state machine models. More importantly, however, the manipulation of finite state machines can be automated, and more sophisticated dynamic analysis tools can be developed. The descriptive clarity of the

finite state machines is debatable. It can well be argued that they trade descriptive clarity for analytical power. By using PROMELA as an intermediate form of an extended finite state machine, however, we can circumvent this problem.

An FSM is considered to be deterministic which means its actions are easily predictable. Extensions to finite state machines such as random selection of transitions, and fuzzy state machines shows us another type of FSM called non-determinist where the systems actions were not as predictable, giving a better appearance of intelligence. Finite state machines are a simple and effective artificial intelligence technique for controlling a system and providing the appearance of intelligence. In some cases the perceived appearance of intelligence is more important than actual intelligence, and that FSMs are able to provide this perception.

IX. REFERENCES

[1].https://mailattachment.googleusercontent.com/attachment/u/0/?ui=2&ik=d13ce03a82&view=att&th=1487f1dc9c339664&attid=0.1&disp=safe&realattid=f_i05ctljp0&zw&saduie=AG9B_P_Of8WNiUXyXAPdBHdBKTsV&sadet=1410884290422&sads=5xYdEEPgq5ckFc6LKkkL8J-P1rk

[2].http://cse.iitkgp.ac.in/~debdeep/teaching/VLSI/slides/fsm.pdf

[3].http://www.gnu.org/copyleft/gpl.html

[4]. ftp://ftp.idsoftware.com/idstuff/source/q1source.zip

[5].ftp://ftp.idsoftware.com/idstuff/source/q1tools_gpl.tgz

[6]. http://www.iitg.ernet.in/asahu/cs221/Lects/Lec16.pdf

[7].http://embedded.eecs.berkeley.edu/research/hsc/class/ee249/lectures/l4-FSM-CFSM.pdf

[8].https://www.google.co.in/search?output=search&sclient=psyab&q=types%20of%20finite%20state%20machine&pbx=1&biw=1164&bih=606&dpr=1.1&pf=p&pdl=300&cad=cbv&sei=rIkhVMj6JojIuAT3yILQCQ

[9].u.cs.biu.ac.il/~veredm/89-689/**FSM**MorVered.ppt

[10].http://en.wikipedia.org/wiki/Finite-state_machine

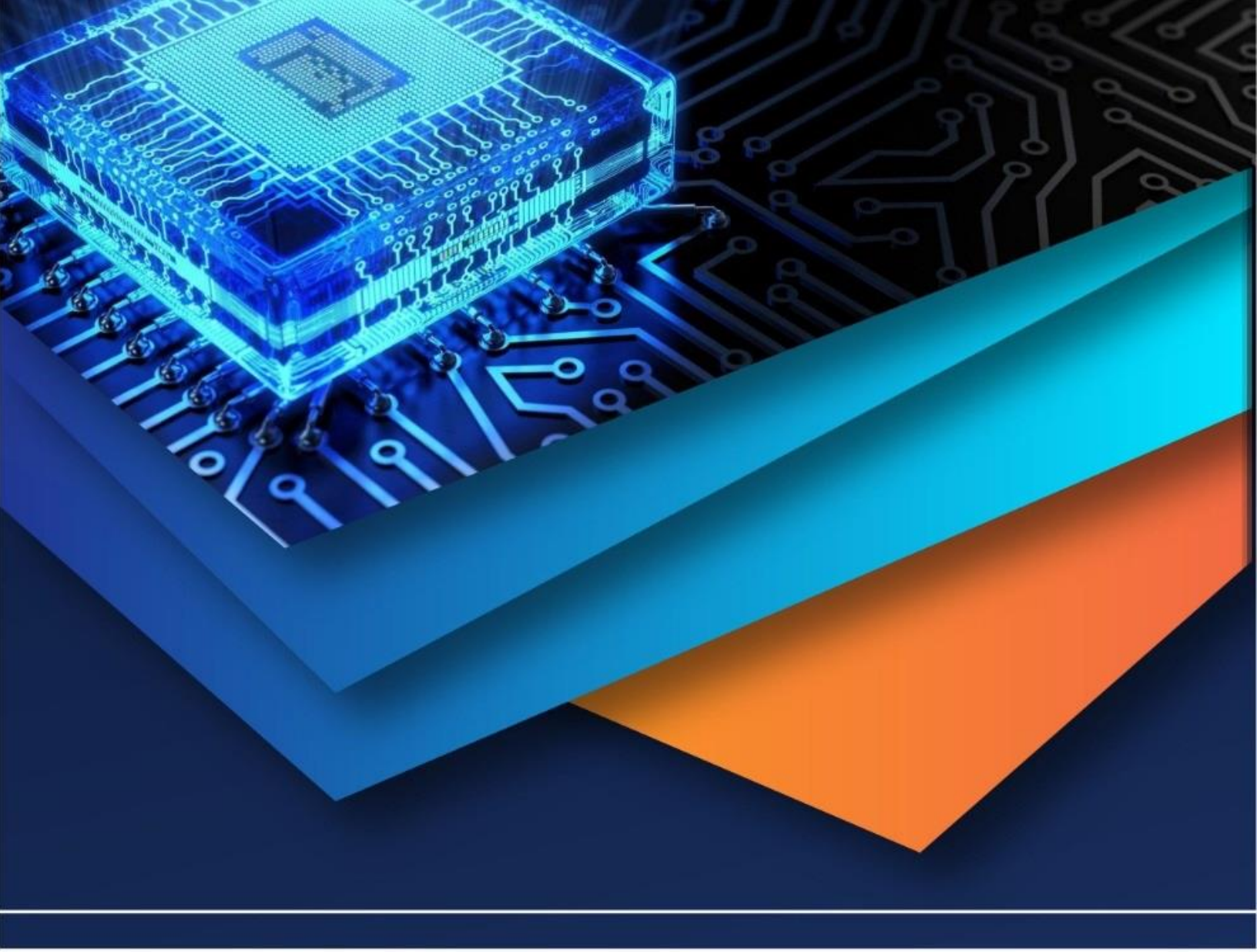
[11].faculty.kfupm.edu.sa/coe/aimane/coe202/Mealy_Moore.pdf

[12].http://www.cse.chalmers.se/~coquand/AUTOMATA/book.pdf

[13].www2.**research**.att.com/~**fsm**tools/**fsm**/

[14].www.spinroot.com/spin/Doc/Book91_PDF/F8.pdf

[15].iasir.net/AIJRSTEM**papers**/AIJRSTEM13-147.pdf



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY