Implementing finite state machines in a computer based teaching system

Charles H. Hacker[a] & Renate Sitte[b]

[a]School of Engineering, C.Hacker@mailbox.gu.edu.au
[b]School of Information Technology, R.Sitte@mailbox.gu.edu.au
Griffith University, Gold Cast Campus, Qld. Australia. Ph: (07) 55948670

ABSTRACT

Finite State Machines (FSM) are models for functions commonly implemented in digital circuits such as timers, remote controls, and vending machines. Teaching FSM is core in the curriculum of many university digital electronic or discrete mathematics subjects. Students often have difficulties grasping the theoretical concepts in the design and analysis of FSM. This has prompted the author to developed an MS-Windows™ compatible software, *WinState*, that provides a tutorial style teaching aid for understanding the mechanisms of FSM. The animated computer screen is ideal for visually conveying the required design and analysis procedures. *WinState* complements other software for combinational logic previously developed by the author, and enhances the existing teaching package by adding sequential logic circuits. *WinState* enables the construction of a students own FSM, which can be simulated, to test the design for functionality and possible errors.

Keywords: Finite State Machines, Finite State Automata, Mealy and Moore machines, Boolean Algebra, Digital Electronic Simulation, Control Logic, Computer Based Teaching.

1. INTRODUCTION

Finite State Machines (FSM) are a model for digital electronic circuitry that are used in control logic applications. Examples of the types of systems that use FSM are: electronic timers, electronic clocks, remote controls, vending machines, and encryption circuits.

Teaching sequential digital logic circuits, using Finite State Machines, are part of the core in the curriculum of many university digital electronic and discrete mathematics subjects. Students in these subjects often have difficulties grasping the theoretical concepts in the design and analysis of Finite State Machines (FSM) and Finite State Automata.(FSA). For the first year digital electronics subject, lectured by the author, this problem is emphasised by the differing background of the students. The subject address students of electronic engineering and information theory, and thus their understanding is distinct in one being hardware based and the other being software based.

This has prompted the author to developed a Microsoft Windows™ compatible software, *WinState*, that provides a tutorial style teaching aid for Finite State Machines. This tutorial software assists undergraduate students in understanding the mechanisms of sequential machines, in particular the *Mealy* and *Moore* machines. The animated computer screen of Microsoft Windows provides an easy to use, user friendly, environment that is ideal for conveying the required design and analysis procedures, which enables students to better visualise the techniques.

The software provides a tutorial on the difference between the *Mealy* and *Moore* state machines. It also enables students to construct their own *Mealy* or *Moore* machine. The designed finite state machine can then be simulated, to test the design for functionality and possible errors.

The software was developed for both lecture demonstration and for students personal use. During lecture demonstrations detailed intricate explanations are necessary to sufficiently convey the Finite State Machine design and analysis strategies. During the explanation, it is easy for students to become confused and bewildered in the barrage of new information and technical terms. The designed program's animated output is ideal for anticipating possible confusion in the explanation, allowing students to observe the strategies during the explanation. The program was also intend for students personal use, allowing a student to build and test their own FSM design. This allows students to study the design procedures privately, at their own pace.

Part of the Conference on Education in Microelectronics and MEMS
Royal Pines Resort, Queensland, Australia • October 1999
SPIE Vol. 3894 • 0277-786X/99/$10.00

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 07/07/2016 Terms of Use: http://spiedigitallibrary.org/ss/TermsOfUse.aspx

Digital electronic subjects cover the two broad categories of combinational logic circuits and sequential logic circuits. Microsoft Windows software that provides a tutorial on combinational logic has been previously developed by the author. This software is known as *WinBoolean* and *BoolTut* (Hacker & Sitte, 1997; Hacker & Sitte, 1998). The Finite State Machines software, *WinState*, was developed to complement the existing software, and enhances the educational use of the existing teaching package, by adding sequential logic circuits.

2. FINITE STATE MACHINES

Finite State Machines are abstract models of machines with a primitive internal memory. Each *state* of the Finite State Machine is a specific set of values stored in electronic registers, (the memory elements). The Finite State Machine accepts a recognisable (or legal) input and after at least one unit time delay produces an output. They consist of a set I of input symbols or "alphabet"; a set of O output symbols; a finite set S of states; a next state function f from $S \times I$ into S; and output function g from $S \times I$ into O; and an initial state $s \in S$.

$$O = g(S,I) \quad \text{and} \quad S_{next} = f(S,I) \tag{1}$$

A Finite State Machine can be represented in tabular form, which lists tables of *Present State*, *Inputs*, *Next State*, and *Outputs*. The Finite State Machine can also be represented as a directed inter-connected diagram. The diagram consists of nodes (circles) representing the states S, and linking arcs representing the transitions or next state functions. Input symbols are placed above the arcs, and output states are placed next to the node (for a Moore Machine) or underneath the input symbol (for a Mealy Machine). A linking arc is be drawn for each possible input symbol, linking one node with another. An example of the Finite State Machine being represented in tabular form, and as a directed diagram is shown in Figure 5.

The Finite State Machine transitions to a new state depending on the sequence of the inputs, which are entered as a string of symbols from the legal alphabet. The Finite State Machine then analyses each input symbol at a time (unit time) and moves to the next state as prescribed by the next state function, producing the designed output symbol. The sequence of instructions (input symbol string) determines the orderly path that is traversed through the sequence of states in the Finite State Machine. The next state to be traversed is determined by the *Next State Decoder*, with the applied *Inputs* and the *Present State*. Whether the *Outputs* depend on the present state alone, or the present state and the applied inputs, determines the type of Finite State Machine.

2.1. Moore Machine

The *Moore Machine* is a Finite State Machine where the derived outputs depend solely on the present state alone. In terms of the State Transition Diagram, (which is used for the design of Finite State Machines), to implement a Moore Machine the desired outputs are generated only with each individual state. An example block diagram of the Moore Machine is shown in Figure 1.

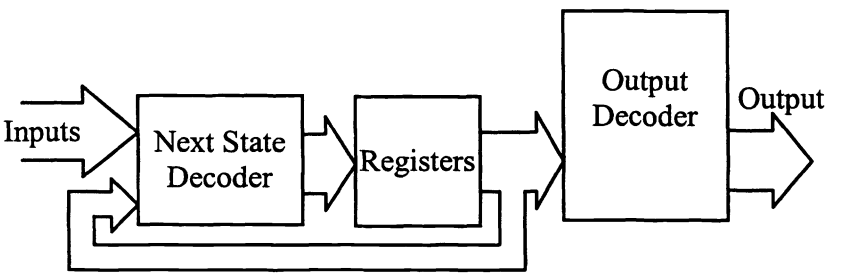

Figure 1: Flow diagram of the Moore machine

2.2. Mealy Machine

The *Mealy Machine* is a Finite State Machine where the derived outputs depend on the present state and the applied inputs. In terms of the State Transition Diagram, to implement a Mealy Machine the desired outputs are generated with each transition to the next state. An example block diagram of the Moore Machine is shown in Figure 2.

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 07/07/2016 Terms of Use: http://spiedigitallibrary.org/ss/TermsOfUse.aspx

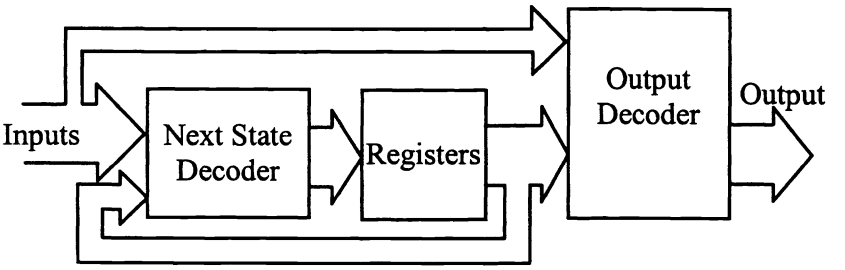

Figure 2: Flow diagram of the Mealy machine

3. TUTORIAL EXERCISE SOFTWARE

The software allows a choice of designing either the Moore or Mealy sequential state machines. An input symbol string to the sequential state machine can then be entered, which is equivalent to an instruction set or a rudimentary program. The input string is then finally run on the modelled sequential state machine. A diagram of the software's structure is shown in Figure 3.

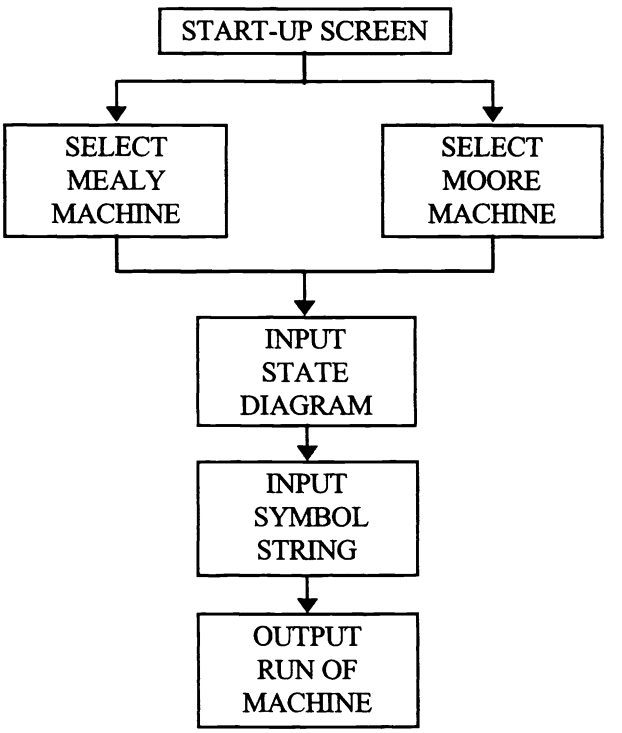

Figure 3: Tutorial exercise sequence

Sequential finite state machines can be either Moore or Mealy type. Consequently the software allows selecting either of these two types of finite state machines in the opening screen This opening screen is shown in Figure 4.

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 07/07/2016 Terms of Use: http://spiedigitallibrary.org/ss/TermsOfUse.aspx

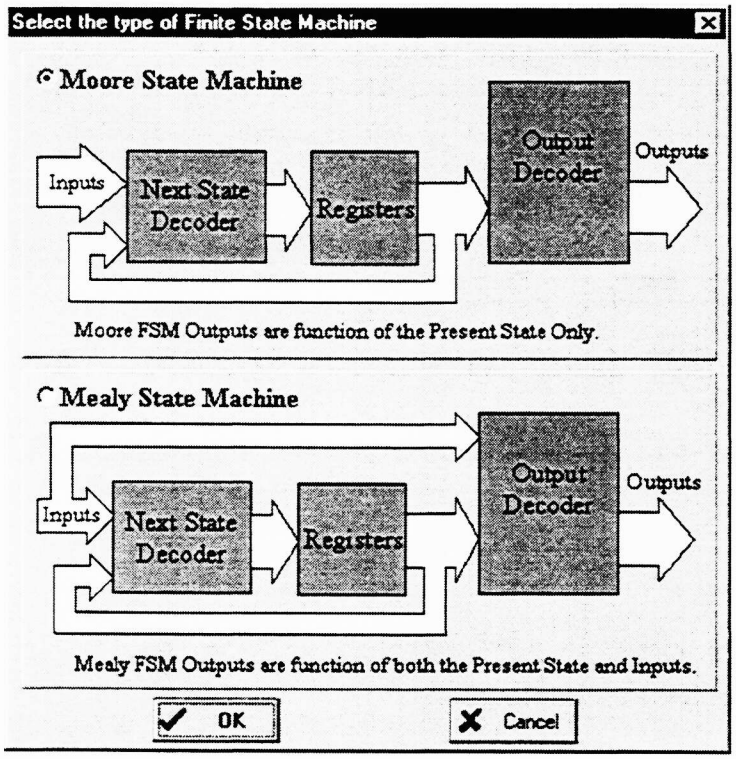

Figure 4: Initial Start-up Dialog

The software allows users to enter their own FSM design, by constructing the state diagram. Individual state node (circle) symbols can be selected from a tool bar, and randomly placed on the screen. The link button (with the arrow icon) is used to link the nodes with arrow segments. Hot spot areas make the link arrows jump to the node circles at an appropriate angle. A select pointer button is available for modifying the state names or link conditions, or for deleting user selected states or links. An option allows the user to choose between common keyboard characters or symbols characters for the state names or link conditions. A move-button allows the user to reposition states, or alter the position of links. For example state nodes can be selected and moved around the screen to a better position. Zooming of the complete state diagram is possible, allowing three different sizes - large, medium, or small. This helps when diagrams become large and complex. While the state diagram of the finite state machine is constructed on the screen, its finite state table is simultaneously filled in the left hand window.

An example of the user design of a Finite State Machine is shown in Figure 5. The example demonstrates that as the state diagram is constructed in the right side window, the state table entries appear on the left side window.

The keyboard is used to enter the identification symbols of the nodes (for example S_o in the above example), and output symbols (such as 0 in the above example). The repertoire of state symbols and output symbols is not limited to the keyboard characters, it also includes a range of symbols. For example the symbol Δ is popularly used to signal an end to a string (null), which is not directly available from the keyboard. To enable user access to these non-standard characters, a pop-up window can be activated that lists the full character and symbol set. The user can then select any desired character or symbol from this window. This is shown in Figure 6.

Once the finite state machine has been implemented, a input symbol string can then be input for execution. The string to be executed is input by the user in the string dialog window. The string is entered by the keyboard, and non-keyboard characters can be entered by the special character input table, which is available by selection of the *ASCII table* button. Each string element may be up to three symbols long, and new input string elements are advanced by the enter key. The arrow-keys allow the moving to previously input string elements, and thus the individual string elements can be modified. The string can be saved to disk, or can be loaded from disk.

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 07/07/2016 Terms of Use: http://spiedigitallibrary.org/ss/TermsOfUse.aspx

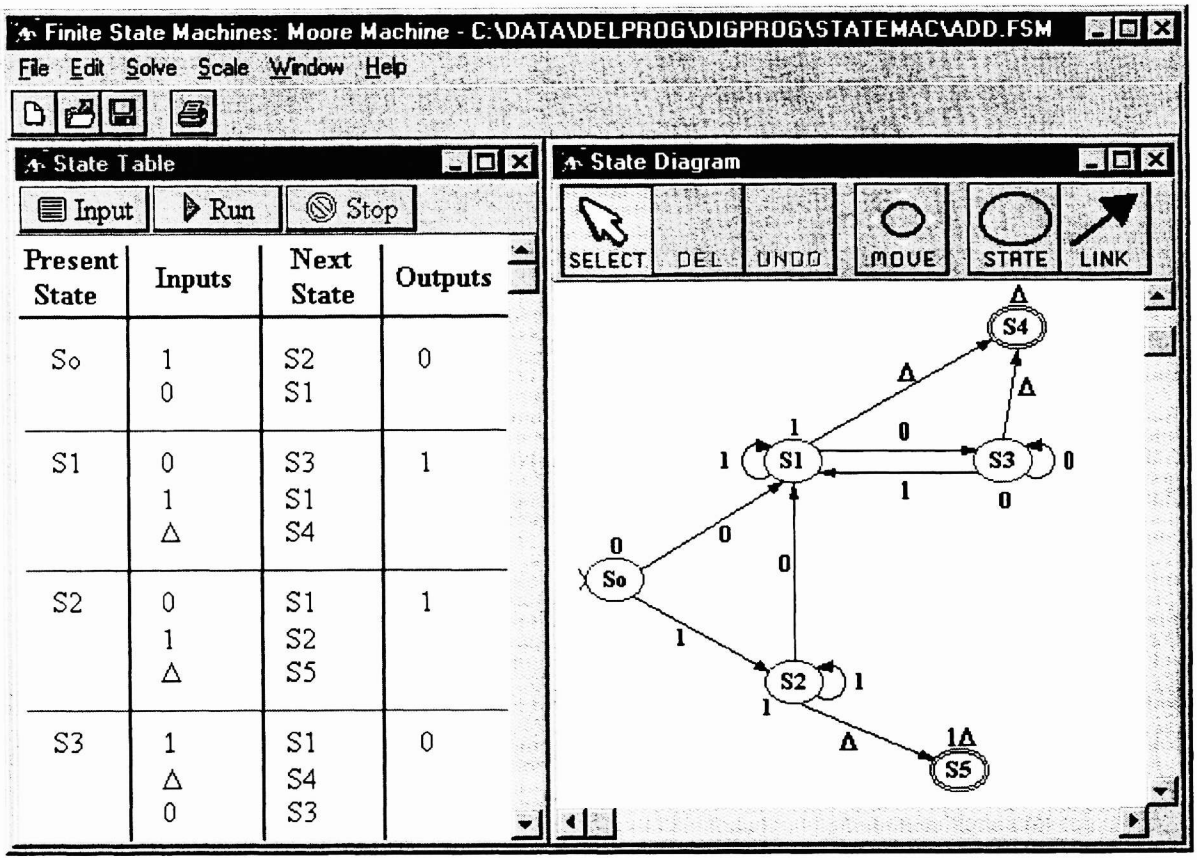

Figure 5: Finite State Machine Input

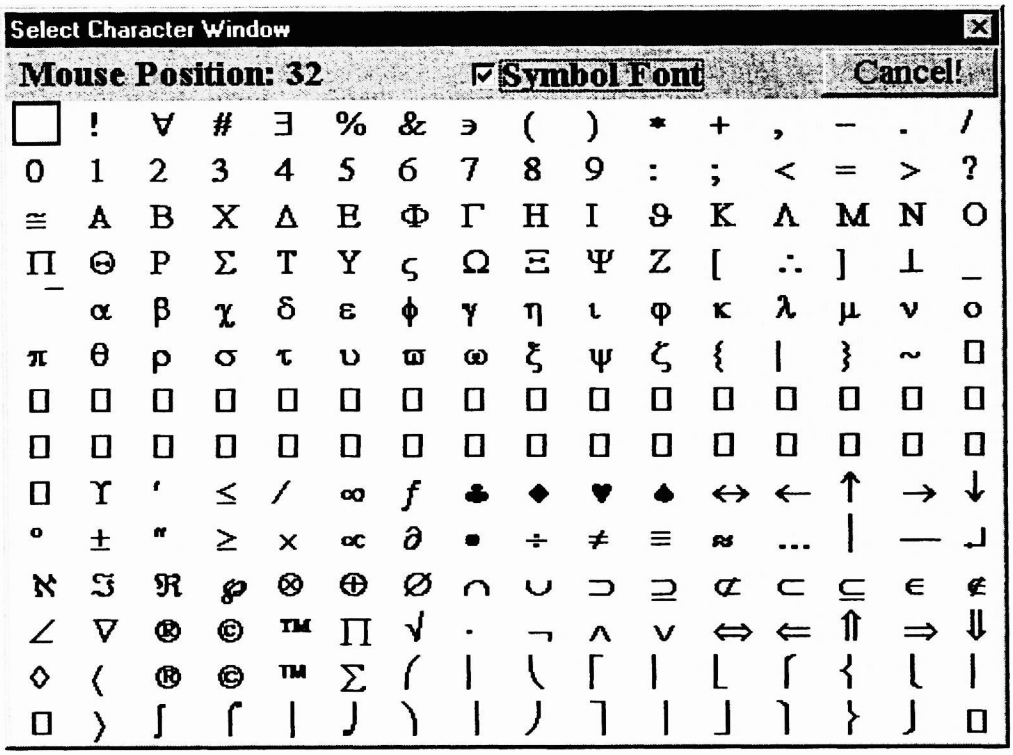

Figure 6: Special Character input table

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 07/07/2016 Terms of Use: http://spiedigitallibrary.org/ss/TermsOfUse.aspx

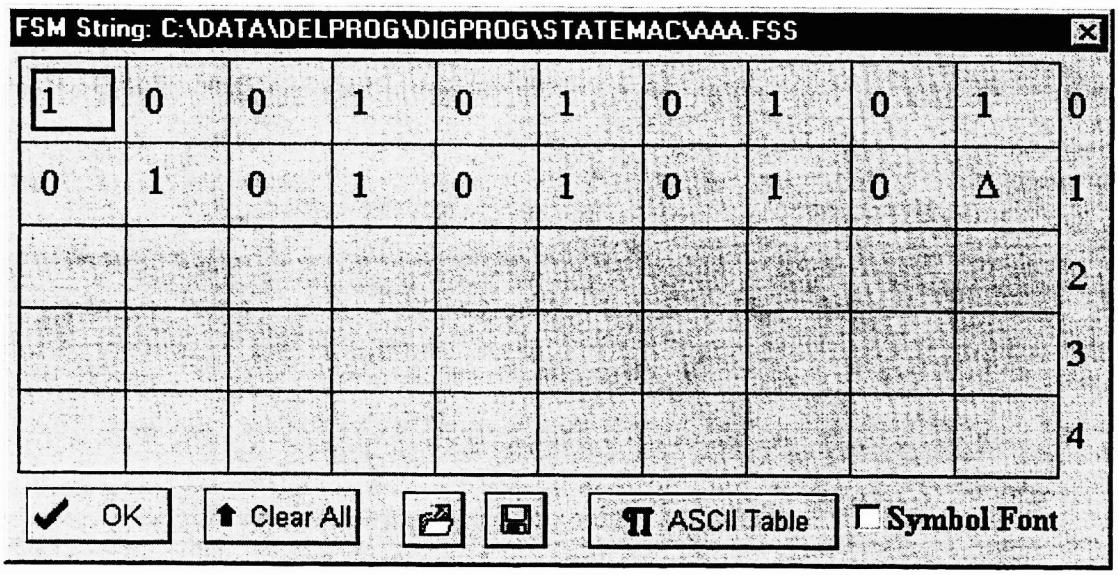

Figure 7: String Input

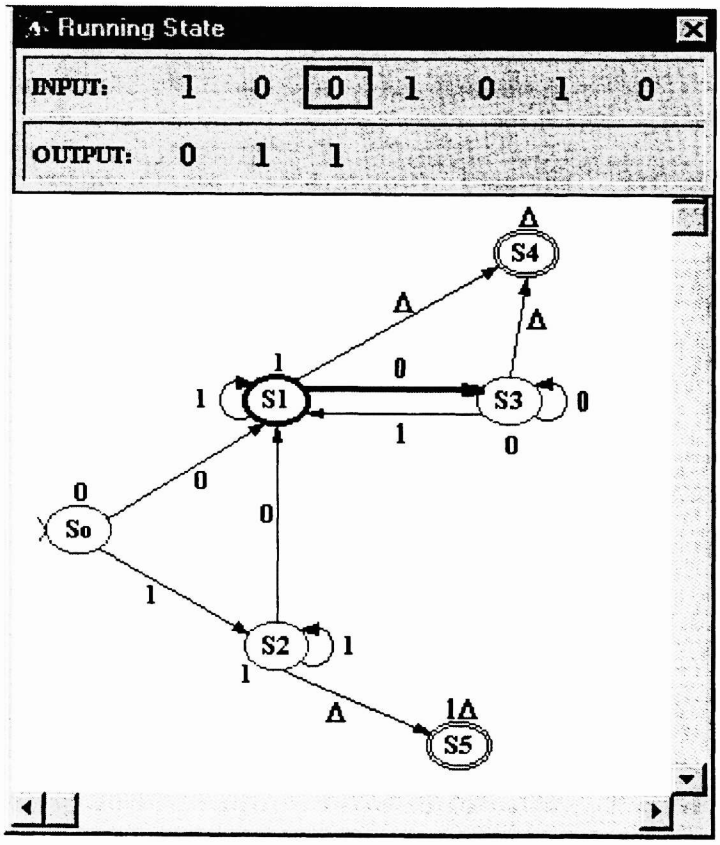

Figure 8: Example State Machine Execution

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 07/07/2016 Terms of Use: http://spiedigitallibrary.org/ss/TermsOfUse.aspx

An example of the user entry of an input string is shown in Figure 7. The example demonstrates that the input string starts on the upper line and continues to lower lines. The Δ symbol signals the end of the string (as a null character), and the square identifies the current input symbol being edited.

The input string is then executed on the constructed finite state machine. During execution a pop-up window shows the user entered input string on one line, with a red highlight square to show the present execution point. The output from the execution builds on the line below the input string. To aid the user in visualising the execution, the current state and link conditions are highlighted on the finite state machine by the use of thicker lines for the node and arc. This highlight occurs while the program parses and executes the current input string and produces the output string.

An example run of a user input string on a constructed finite state machine is shown in Figure 8. The example is currently executing the transition from *S1* to *S3*, due to an input of *0*, and produces on output of *1* from state *S1*.

4. IMPLEMENTATION

The implementation of the finite state machine network requires advanced programming techniques. To establish a link between the on the screen state diagram and the software function, the finite state machine is implemented via a system of linked lists. This requires two types of data structures: one for the state nodes, and one for the linking arcs. The linked list elements for the nodes have three fields: the state information, the arc-pointer link, and the next-node-pointer. For the linking arcs the linked list elements require two fields: the node the link points to, and the pointer to the next-arc. An element of the node-list uses its arc-pointer field to point to the arc-list. The arc list in turn uses its node-pointer field to point to the node-list. At the same time the node list uses its next-node field to point to its own next element, and conversely the arc-list uses its next-arc field to point to its next element in the list. The technique and algorithm is not new but complex. It is described in more detail by Tenenbaum & Augenstein, 1981.

Figure 9 shows an example Finite State Machine, as a diagram of a directed graph (the state diagram) and a diagram of the computerised data structure as a system of linked lists.

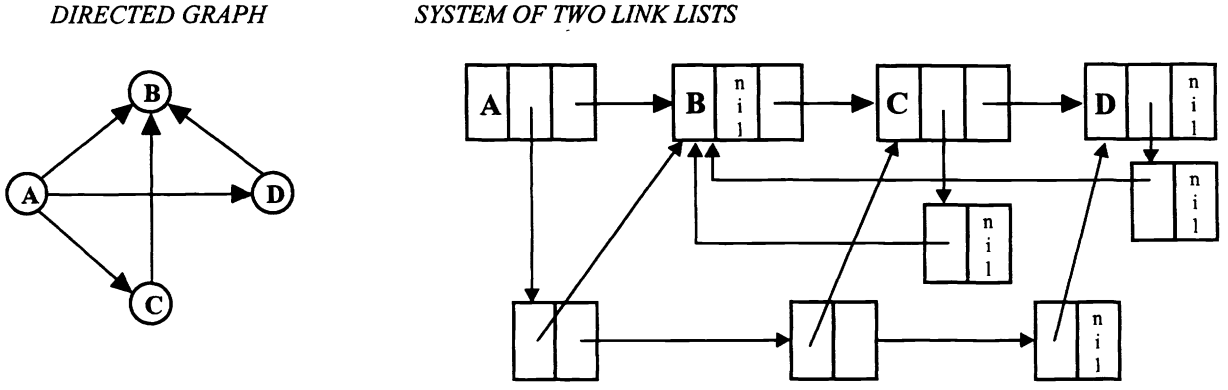

Figure 9: Example of a simple Finite State Machine as a Directed Graph and the associated Linked List structure

5. CONCLUSION

This paper outlined the development of a computer based tutorial for implementing Finite State Machines. The implementation requires advanced computing skills. This software forms an addition to software previously developed by the author for the teaching of combinational logic. The software extends the capability of the full teaching software package, to cover the full range of topics covered in the introductory digital electronic subjects of Combinational logic and Sequential state machines.

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 07/07/2016 Terms of Use: http://spiedigitallibrary.org/ss/TermsOfUse.aspx

REFERENCES

1. Advanced Micro Devices (AMD), *State Machine Design*, Advanced Micro Devices (AMD) databook, Publication Number: 90005, 1993
2. J. L. Gersting, *Mathematical structures for computer science*, Freeman Press, New York, 1987
3. C. Hacker, and R. Sitte, *A Computer Based Tutorial, For Demonstrating the Solving of Digital Electronic Circuits*, Proceedings of the 10th Annual Australasian Association for Engineering Education (AaeE98), Waves of Change, September 1998, pp. 509-519, 1998
4. C. Hacker, and R. Sitte, *Development of a computer program, to electronically design Digital Logic Circuits using Boolean Algebra*, Proceedings of the 9th Annual Australasian Association for Engineering Education (AaeE97), December 1997, pp. 353-357, 1997
5. C. Hacker, and R. Sitte, *Implementing the 'Espresso - two level logic minimiser' algorithm in the MS-Windows environment*, 2nd Asia-Pacific Forum on Engineering & Technology Education, UNESCO International Conference in Engineering Education (UICEE99), July 1999, pp. 124-127, 1999
6. R. Skvarcius and W. B. Robinson, *Discrete mathematics with computer science applications*, Benjamin / Cummings Pub. Co., Menlo Park, Califionia, 1986
7. A.M. Tenenbaum, and M.J. Augenstein, *Data Structures Using Pascal*, Prentice Hall Inc, 1981

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 07/07/2016 Terms of Use: http://spiedigitallibrary.org/ss/TermsOfUse.aspx