Automated Synthesis of Sequence Invariant State Machines

D. Buehler, S. Whitaker and J. Canaris
NASA Space Engineering Research Center
for VLSI System Design
University of Idaho
Moscow, Idaho 83843

Abstract - **A CAD tool for the design of VLSI synchronous sequential controllers is presented. Both the design and layout of the state machine are automatically generated. The program is process independent allowing a choice of design rules to base generation upon. An incremental layout creation approach has been implemented which makes the tool useful in a wide range of layout applications. Flow table descriptions are input to characterize the desired machine and a layout archive is output.**

1 Introduction

A state machine can be implemented as a programmable PLA based structure or as random logic. The realization of state machines based on random logic often results in the most compact and highest performance circuits, but the logic, which is a function of the state assignment, flip flop type and flow table, does not lend itself to easy automatic synthesis. Controllers implemented as PLA structures can be generated automatically, but are less area efficient and have reduced system performance.

An architecture that retains the traditional strengths of dedicated state machines, but offers the programmability of a microcontroller, was presented in [1]. This architecture produces controllers whose logic is invariant with respect to the actual sequence desired. State machines designed using this method approach the performance and size of random logic based state machines and have a programmability superior to a PLA based design.

This paper presents a CAD tool for generating VLSI dedicated controllers based on the sequence invariant architecture, described in Section 2. The tool, named "sm" (for SISM maker), is a flexible tool which not only provides the layout of complete state machines, but can generate stand alone sub-circuits as well. The layouts are design rule independent and correct by construction. Layout generation is done on the fly, with a process rule file being the minimum input required. The software, which is a pipeline of sub-circuit generators, provides great flexibility and speed, for an engineer and layout designer, in the implementation of CMOS controllers. A significant reduction in engineering design time is also attained, as "sm" performs all of the logic design tasks normally associated with state machines. The interesting and creative design task, flow table definition, remains in the hands of the engineer, while the time consuming portions are handled in software.

	I_1	I_2	I_3
A	C	B	A
B	D	C	B
C	E	D	C
D	F	E	D
E	A	F	E
F	B	A	F

Table 1: Example flow table.

Section 3 of this paper describes the approach "sm" takes to layout generation itself, while Section 4 provides a description of the input options available. Section 5 outlines the benefits of the CAD tool.

2 Sequence Invariant State Machines (SISMs)

2.1 What does "Sequence Invariant" mean?

State machines are used primarily as control structures in digital circuits. A traditional format for the description of a state machine is the flow table, such as shown in Table 1. The flow table has dimensions of i inputs wide by s states tall. Entries in the flow table determine the state transitions that are made during operation.

The driving idea behind the sequence invariant architecture is to build a state machine given only the dimensions (i and s) of the flow table. The state machine created must be capable of performing the operations of *any* sequence of states that are described in the i by s flow table.

2.2 How is "Sequence Invariance" accomplished?

The sequence invariant architecture is broken into functional blocks as shown in Figure 1. In effect, the Destination State Codes are a representation of the flow table to be implemented. The Input Switch Matrix selects a single column of the flow table and passes the next state information for the entire selected column to the Next State Logic. The function of the Next State Logic is to select which of the states from the column of states presented to it should be selected as the next state. This choice of next state is based upon the current state. The hardware implementation of the Input Switch Matrix and Next State Logic circuits is dependent only on the dimensions of the flow table to be implemented. The only difference between state machines with dimensions $i = x$, $s = y$ is the programming of the Destination State Codes.

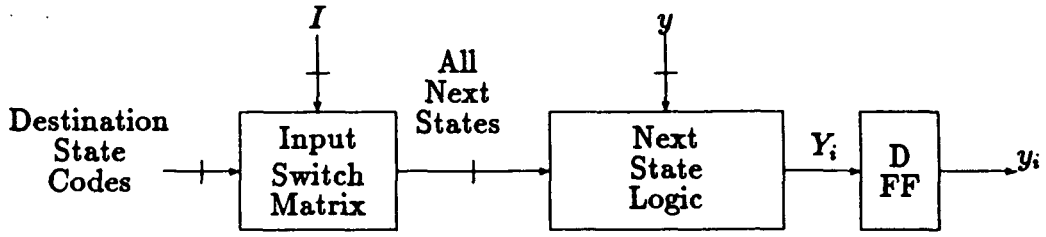

Figure 1: General block diagram.

	I_1	I_2	I_3
S_0	N_{01}	N_{02}	N_{03}
S_1	N_{11}	N_{12}	N_{13}
S_2	N_{21}	N_{22}	N_{23}
S_3	N_{31}	N_{32}	N_{33}
S_4	N_{41}	N_{42}	N_{43}
S_5	N_{51}	N_{52}	N_{53}
S_6	N_{61}	N_{62}	N_{63}
S_7	N_{71}	N_{72}	N_{73}

Table 2: General eight-state three-input flow table.

2.3 Operation

The following illustrates specifically how this architecture works. Let Table 2 depict an example for a general 3 state variable, 3 input state machine. I_1, I_2 and I_3 are the inputs, $S_0 \ldots S_7$ are the present states, and $N_{S_0 I_1}, N_{S_0 I_2} \ldots N_{S_7 I_3}$ are the next states. This can be generalized so that $N_{S_i I_j}$ are the next states for S_i under input I_j. $N_{S_i I_j}$ has been abbreviated as N_{ij}. The set of N_{ij} also comprise the destination state codes. Let the state assignment be $S_0 = 000$, $S_1 = 001$, $S_2 = 010$, $\ldots$, $S_7 = 111$.

The next state logic is a general BTS circuit [2,3] with paths that decode each state. The input switch matrix is a pass transistor matrix, that passes the destination state codes to the next state pass network as shown in Figure 2. The circuit realization of this network operates in the following manner: All of the destination state codes N_{ij} are presented to the input switch matrix. For each input state I_i, all of the destination states in I_i are presented to the next state logic. The present state variables, y, select one and only one next state entry which is passed to the flip-flops. If the machine is in state S_1 and input I_2 is asserted, then N_{12_i} would be passed to the input of the flip-flop for next state variable Y_i. The current input state selects the set of potential next states that the circuit can assume (selects the input column) and the present state variables select the exact next state (row in the flow table) that the circuit will assume at the next clock pulse.

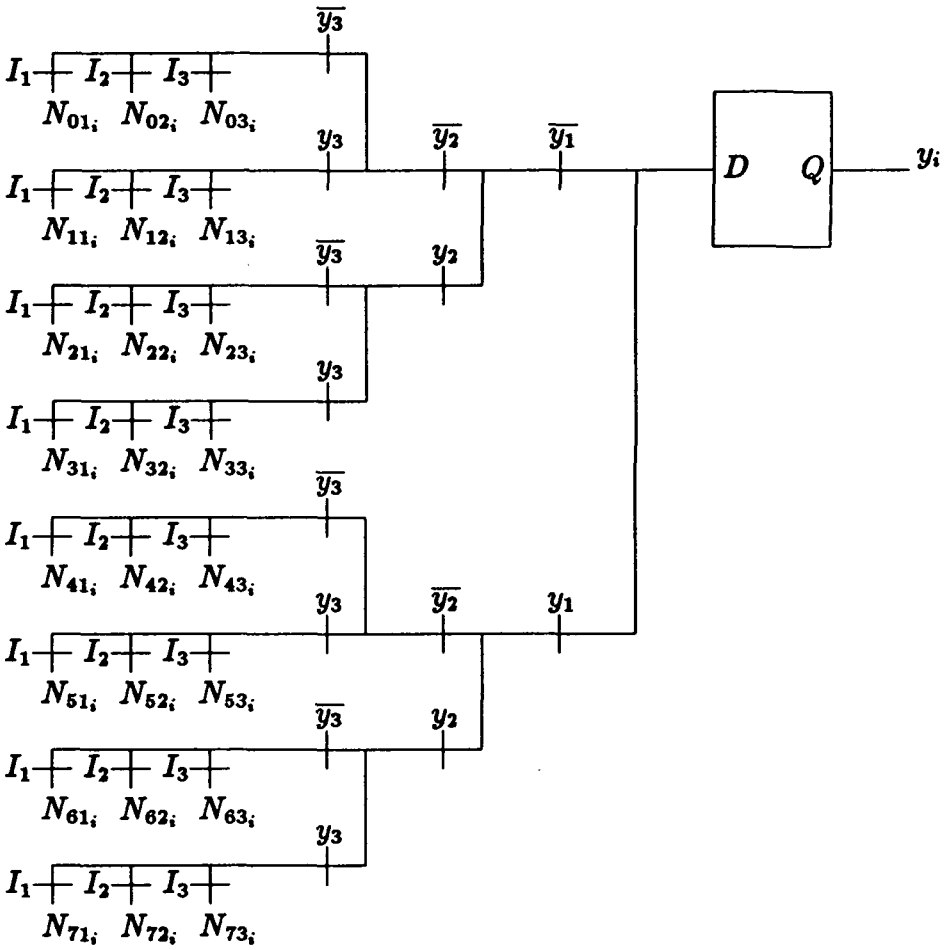

Figure 2: General eight-state three-input next state equation circuit.

2.4 SISM's and state machine output equations

It has been shown [4,5] that the same logic which is used to implement Sequence Invariant State Machines can also be used to generate the forming logic for the state machine output equations. In this case the feedback terms are removed. The present state information is provided by the logic blocks in the SISM. The same sequence invariance is available, as is the programmability. The CAD tool described here has the ability to generate logic blocks suitable for implementing these output equations.

3 Automating Layout

3.1 Layout approach

The structure of the SISM logic lends itself to two "natural" layout approaches. First, the regularity of poly spacings vertically and metal spacings horizontally, (see Figure 3), suggests a gate-matrix layout approach [6]. In this approach, horizontal poly-silicon and vertical metal pitchs are calculated, then contact and diffusion areas are placed to form transistors. The gate-matrix approach proves undesirable due to the inflexibility of the monolithic layout structure created. The second approach is also suggested by the regularity of the SISM logic. The layout can be divided into a number of tiles which can be calculated, created, then placed. The tile-laying methodology was chosen because it allows decomposition of the layout problem into smaller cell layout problems.

Although the ultimate goal is a complete state machine layout, this methodology allows useful layouts to be made available, as intermediate steps, in the compilation. The next section describes the layout decomposition.

3.2 Layout Decomposition

The organization of the SISM layout, (see Figure 4), parallels the functional blocks shown in Figure 1. Decomposition of the layout for creation purposes follows the functional decomposition shown. "sm" creates a cell implementing the Input Switch Matrix and a cell for the Next State Logic. Additional layout cells provide programmable connections for power and ground. Feedback taps for the state variables are also generated.

A single state bit cell can be created and replicated for each state bit because the Input Switch Matrix and Next State Logic are identical for each state variable bit (due to Sequence Invariance). After this base architecture is created, a programming mask is placed over the Input Switch Matrix to program the desired state transitions.

As described in Section 2.3, the Next State Logic block is a general BTS network and Figure 2 indicates that the Input Switch Network is a general pass transistor OR structure. Both of these functional blocks are found in logic implementations, other than state machines. Therefore, "sm" was designed to optionally create layouts of these sections independent of a particular SISM design. The user can then copy, connect, and program

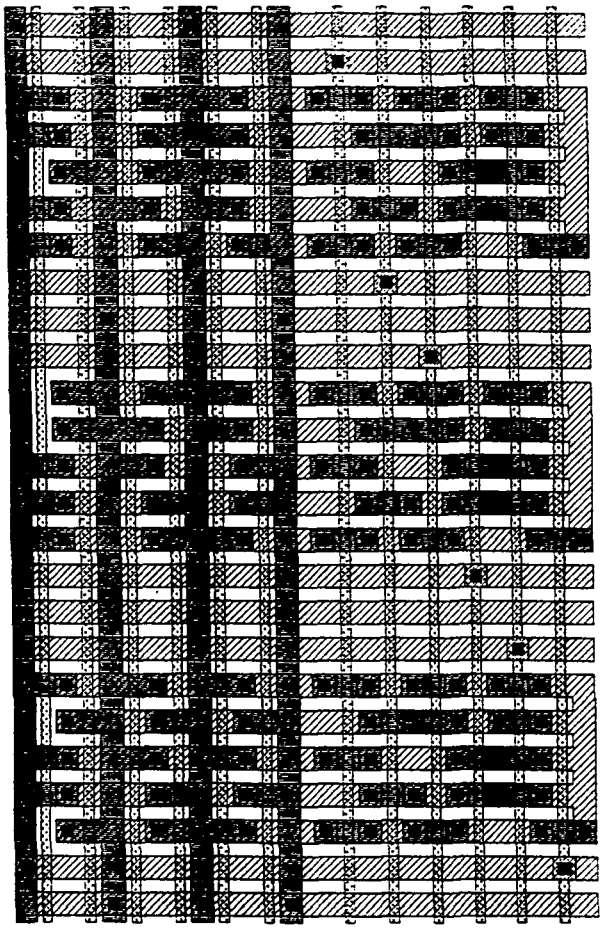

Figure 3: Three input, five state SISM layout

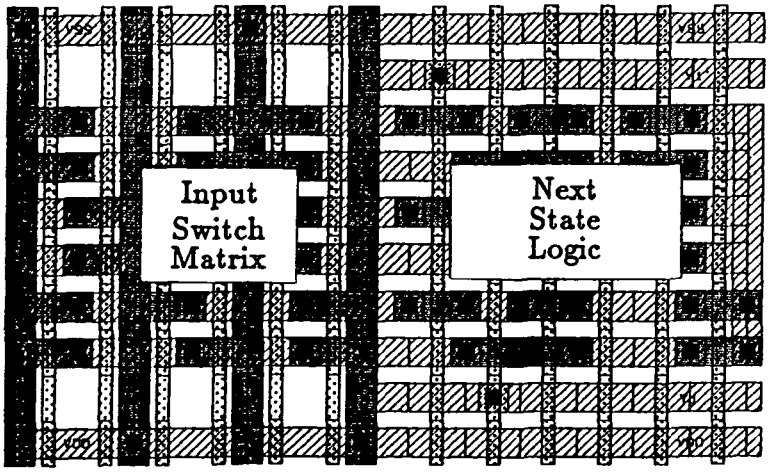

Figure 4: Layout showing functional areas

```
NAME: MACH1
PROCESS: CMOS34
a:  c b a.
b:  d c b.
c:  e d c.
d:  f e d.
e:  a f e.
f:  b a f.
```

Table 3: Flow table entry format

them manually. This allows more flexibility in the layout process, and provides automatic generation of structures which may have uses in applications other than SISMs.

4 "sm"

4.1 Layout options

The amount of layout generated is very flexible. The user can choose to specify the number of states and generate the Next State Logic cell. Specifying the number of inputs in addition to the number of states allows the user to generate an unprogrammed Input Switch Matrix and/or a Next State Logic cell, or a complete, unprogrammed state machine. These options allow the layout designer to create blocks which generate the output signals of a state machine, as described in Section 2.4. To generate a complete, programmed state machine the user must create a file with a flow table description of the state machine.

4.2 Flow table description

When the user requests creation of a complete, programmed state machine, "sm" requires a flow table description of the desired state machine. The flow table description is a simple ASCII text file consisting of state transition information. Table 3 shows the flow table entry format used to create a state machine corresponding to the flow table shown in Table 1. State declarations are followed by a colon and the state transition information for each input, then closed with a period. Undefined or "don't care" states are not allowed. States are assigned sequentially, beginning with the first state declared, which is assigned state variable value zero. It is often advantageous to assign this state to the reset state.

4.3 Process choices

The CAD tool described here is process independent. Layouts are created on the fly, using a tiling algorithm. Fabrication of VLSI circuits, however, is not process independent.

When targeting a specific process, "sm" requires a Design Rule file for that particular process. "sm" currently has two Design Rule files available:

- Hewlett-Packard 1.6μm CMOS-40 rules.
- Hewlett-Packard 1.0μm CMOS-34 rules.
- Implementation of the MOSIS Scalable Rule set is planned.

4.4 Transistor width

The transistors used in the SISM layout default to a minimum transistor width, which is the size of a metal/diffusion contact plus diffusion overlaps on each side. If the user wishes, the transistors can be created larger than this minimum. The resolution of transistor widths allowed is 0.2μm .

5 "sm" and SISM Benefits

Use of "sm" and sequence invariant state machines has many benefits over PLA and random logic state machine implementations. Running on an HP 9000/375 "sm" took two seconds real time (half a second CPU time) to create the layout shown in Figure 5. This layout implements the state machine from the flow table description shown in Table 3. The layout created is correct by construction and will pass Design Rule Checks (DRC). A layout designer would require half a day to lay out, check continuity, and run DRC verification on a hand crafted SISM state machine layout.

A traditional state machine requires a significant amount of work. The engineering tasks required include flow table construction, state assignments, choosing a flip-flop, designing input forming logic, designing output forming logic, and circuit design. The layout task is one of the most time consuming in a VLSI design. An experienced layout designer may be able to draw about 8 transistors/hour of random logic. Traditional controller designs tend to have large amounts of random logic associated with them. A traditional implementation of the flow table, Table 3, might require several days of layout.

Time is saved, using "sm", because no state machine design time, other than the flow table construction, is required. The sequence invariant implementation will yield additional savings in the event of design changes which require modification of the state flow sequence or the addition of new states.

"sm", while remaining a simple program to use, provides flexibility to the layout designer, as well as time savings to the design engineer.

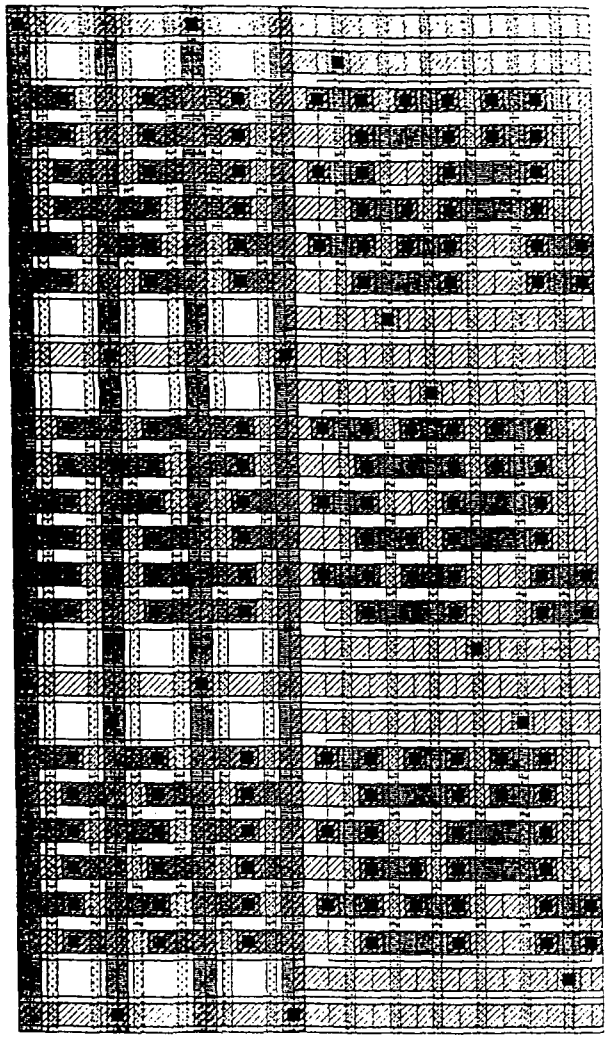

Figure 5: "sm" generated layout

References

[1] S. Whitaker, S. Manjunath and G. Maki, "Sequence Invariant State Machines", submitted to the IEEE Journal of Solid State Circuits.

[2] G. Peterson and G. Maki, "Binary Tree Structured Logic Circuits: Design and Fault Detection," Proceedings of IEEE International Conference on Computer Design: VLSI in Computers, pp. 139-144, Oct. 1984.

[3] D. Radhakrishnan, S. Whitaker and G. Maki, "Formal Design Procedures for Pass-Transistor Switching Circuits," IEEE Journal of Solid State Circuits, pp. 531-536, Apr. 1985.

[4] S. Whitaker and G. Maki, "Pass Transistor Asynchronous Sequential Circuits", IEEE

JSSC, Vol. SC-24, pp. 71-78, February 1989.

[5] S. Whitaker, G. Maki and M. Canaris, "A Programmable Architecture for CMOS Sequential Circuits", Proceedings of the NASA SERC 1990 Symposium on VLSI Design, Moscow, Idaho, pp. 223-229, January 1990.

[6] O. Wing, S. Huang and R. Wang, "Gate Matrix Layout", IEEE Transactions on Computer-Aided Design, pp. 220-231, July 1985.

This research was supported in part by NASA under the NASA Space Engineering Research Center grant NAGW-1406 and by the Idaho State Board of Education under grants 88-038 and 89-041.