On The Decomposition Of Synchronous State Machines Using Sequence Invariant State Machines

K.Hebbalalu, S.Whitaker and K.Cameron
NASA Space Engineering Research Center
University of Idaho
Moscow, Idaho 83843

Abstract – **This paper presents a few techniques for the decomposition of Synchronous State Machines of medium to large sizes into smaller component machines. The methods are based on the nature of the transitions and sequences of states in the machine, and on the number and variety of inputs to the machine. The results of the decomposition, and of using the Sequence Invariant State Machine (SISM) Design Technique for generating the component machines, include great ease and quickness in the design and implementation processes. Furthermore, there is increased flexibility in making modifications to the original design, leading to negligible re-design time.**

1 Introduction

Most digital Very Large Scale Integrated (VLSI) circuits currently designed contain controllers which co-ordinate and control activities occuring inside and outside the chip. Usually, such controllers are comprised, partly or wholly, of synchronous sequential machines. Often, depending on the nature and complexity of the control action required, these sequential machines are of medium to large sizes (requiring five or more state variables). While it is always possible to design these machines as single units using random logic, this implementation presents the designer with a narrow choice of options, since a large number of state variables results in difficult-to-derive and unwieldy design equations. Also, the layout process for these single, large machines becomes complicated and even minor changes in the original flow Table may result in a complete re-design. In addition, large machines tend to be slower and not very amenable to fault diagonostics or to trouble-shooting.

Hence there is a need to decompose such large machines into smaller, more manageable component machines, which offer greater flexibility to the designer. This paper presents a few such decomposition techniques, resulting in faster and better designs. Furthermore, the Sequence Invariant State Machine (SISM) Design Technique [1] is used to generate the component machines, which enhances the ease of the design process, by not requiring the use of complicated design equations and K-Maps. The final and greatest advantage is that the layout is automated.

2 Notations and Definitions

Definition 1: An *active state machine* is one in which some outputs are valid and which is exercising control over the chip in part or whole.

Definition 2: An *inactive state machine* is one in which no output is valid.

Definition 3: A *ready protocol* between two state machines signifies that the machine which originates the *ready* signal is becoming inactive and is handing over control to the machine which receives the signal.

Definition 4: A *start-up state* is a state in which a machine waits until it receives a ready signal from another machine, indicating it is to become active and to go into its next valid state. In the example of Figure 1, states A, X' and X" are start-up states.

Definition 5: An *idle state* or *wait state* is one in which a machine raises a ready signal to another machine to become active, and perhaps waits in this state as long as the machine is inactive: *ie.,* until it receives a ready signal in its turn. In Figure 1, states F, X' and X" are wait states. A single state can function both as a start-up and as a wait state.

Definition 6: State splitting is the replication of a state from the original machine, into corresponding states in component machines. These states serve exactly the same function as the original state with the same outputs. States G' and G" in Figure 1, are split states of the original state G. State splitting is used to limit the interaction and to avoid unnecessary transfer of control between machines.

3 Decomposition and SISMs

Most of the state machines encountered during design, have their own individual peculiarities and constraints, and a formal, all-encompassing procedure for their design or decomposition is difficult. There would be several possible ways of achieving the results, and depending on the criteria, more than one optimal solution would be found. These criteria for finding the 'best' solution usually include speed and area considerations, cost, quickness and ease of implementation and testing, among others. The designer might choose one or more of these, in various orders of priority.

Some decomposition techniques are presented in the following sections, bearing in mind the implementation of the component machines using the SISM Design Technique [1]. There are several advantages to be gained, using this method:

- The structure and design of all the sub-state-machines are the same, provided, they all have equal number of state variables.
- The design of the SISMs is simple and easy, not involving the use of K-Maps or design equations.
- The layout process for these machines is uncomplicated and is automated, leading to very quick results[2]. Also, since the basic structure of the SISM is very regular, the layout is very dense and the area savings are considerable.
- The simulations for the circuits using SISMs are also easy and quick. Any changes to be incorporated are easily done with negligible re-design time.

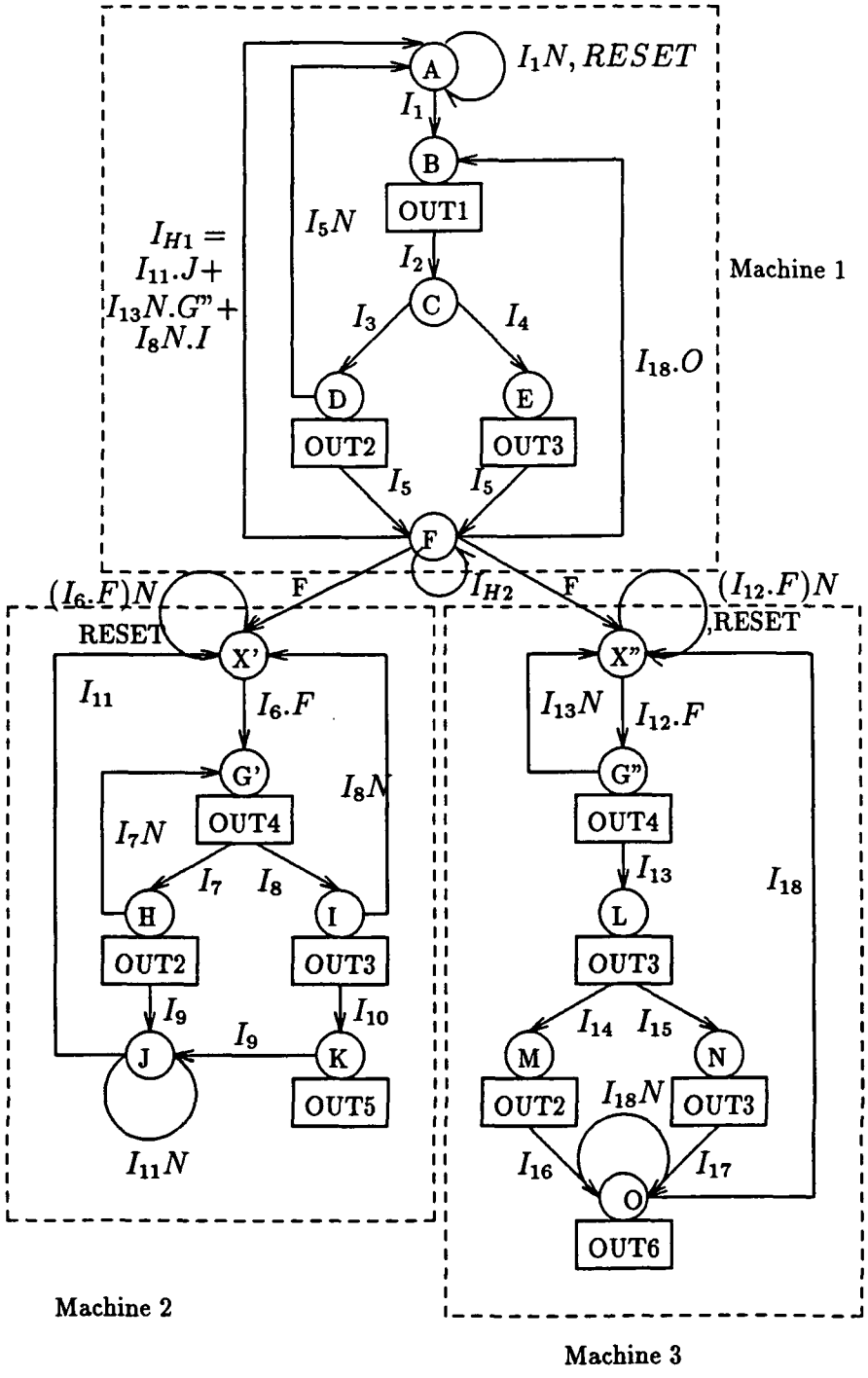

Figure 1: Example of Decomposed State Machine

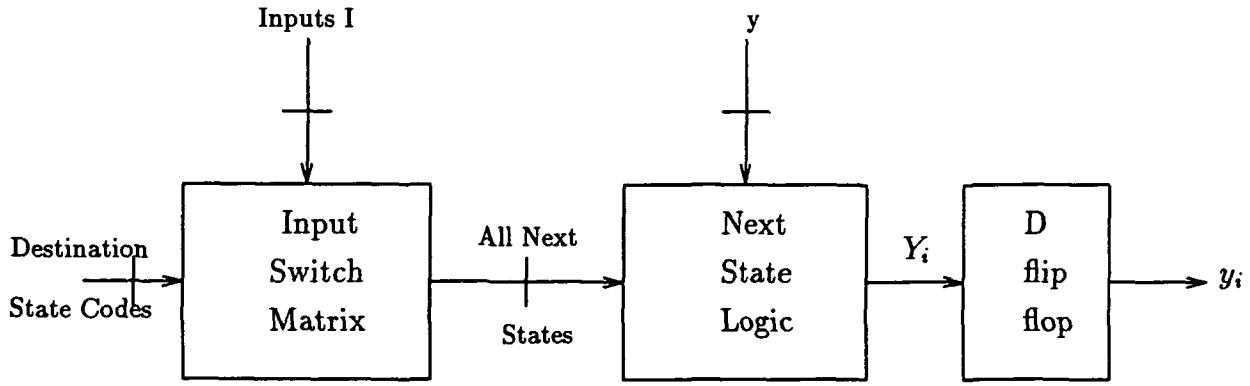

Figure 2: General Block Diagram of SISM

	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 1: General eight-state three-input flow table

3.1 The operation of SISMs

The driving idea behind the SISM architecture is to build a state machine given only the dimensions (number of inputs i and number of states s) of the flow Table. The state machine thus created must be capable of performing the operations of *any* sequence of states that are described in the i by s flow Table, to achieve sequence invariance.

An SISM has the basic block structure as shown in Figure 2. 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 decoder. This next state decoder selects one from the column of states presented to it, as the next state, depending on the present state. The hardware for the SISM depends only on the dimensions of the flow Table to be implemented. The only difference between state machines built for flow tabes with different sequences of states, but with the same dimensions, is in the programming of the destination state codes.

The operation of an SISM can be illustrated with the following example. Let Table 1 represent a general flow Table for a 3 state variable, 3 input state machine. I_1, I_2 and I_3 are the inputs, $S_0 ... S_7$ are the present states and N_{ij} are the next states. The set of N_{ij} also comprises of the destination codes. Let the state assignment be S_0 =000, S_1=001,.....S_7=111.

The next state logic is a general BTS circuit [3,4] with each path decoding a state. The input switch matrix passes the destination codes to the next state logic, as shown in Figure 3. The circuit works as follows: For an active input, I_i, all next state codes in the ith column of the input matrix are passed to the inputs of the next state logic. The present state variables, ys, select only one code among them and pass it on to the flip-flops. For example, if the machine is in state S_1 and input I_2 is asserted, then N_{12} would be passed to the inputs of the flip-flops. The current input 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 on the next clock pulse.

4 Decomposition Techniques

This section describes some methods of decomposing large state machines and at the same time, indicates the feasibility of each method for varying situations, with illustrations.

4.1 Functional and Hierarchical Decomposition

As is most often the case, the designer of a digital controller is provided with a word statement of the specifications and the sequence of operations that the controller should conform to, rather than a state diagram or a flow Table. In such cases, a popular method is to decompose the controller into functional blocks, with each block representing an operation in the main sequence of events. In other words, a functional block diagram is generated. Each functional block is progressively decomposed into smaller, more specific operations, culminating in state machines with sets of states, defining macro operations. The requisite flow Tables, design equations and outputs are generated for these machines and are then hierarchically combined to produce the final controller.

4.2 Decomposition using 'bottleneck' states

Many state diagrams reveal on inspection natural 'bottleneck' states which connect groups of states. In the example shown in Figure 1, state F is such a state. The state machine can be decomposed around such states, making the groups of states into sub-machines, with the bottleneck state included in any one of them.

This method of decomposition is convienient and economical in that, it saves the generation of additional dummy states to help decomposition, or the resortion to state-splitting for the same purpose. The only criterion to be observed here is that the number of states in the groups connected by the bottleneck should be roughly the same, so that, when formed into sub-machines, they all need the same number of state variables. This method has been used in fragmenting the state machine in Figure 1, where there are 3 groups of states around state F, with each group having 6 states, needing 3 state variables each.

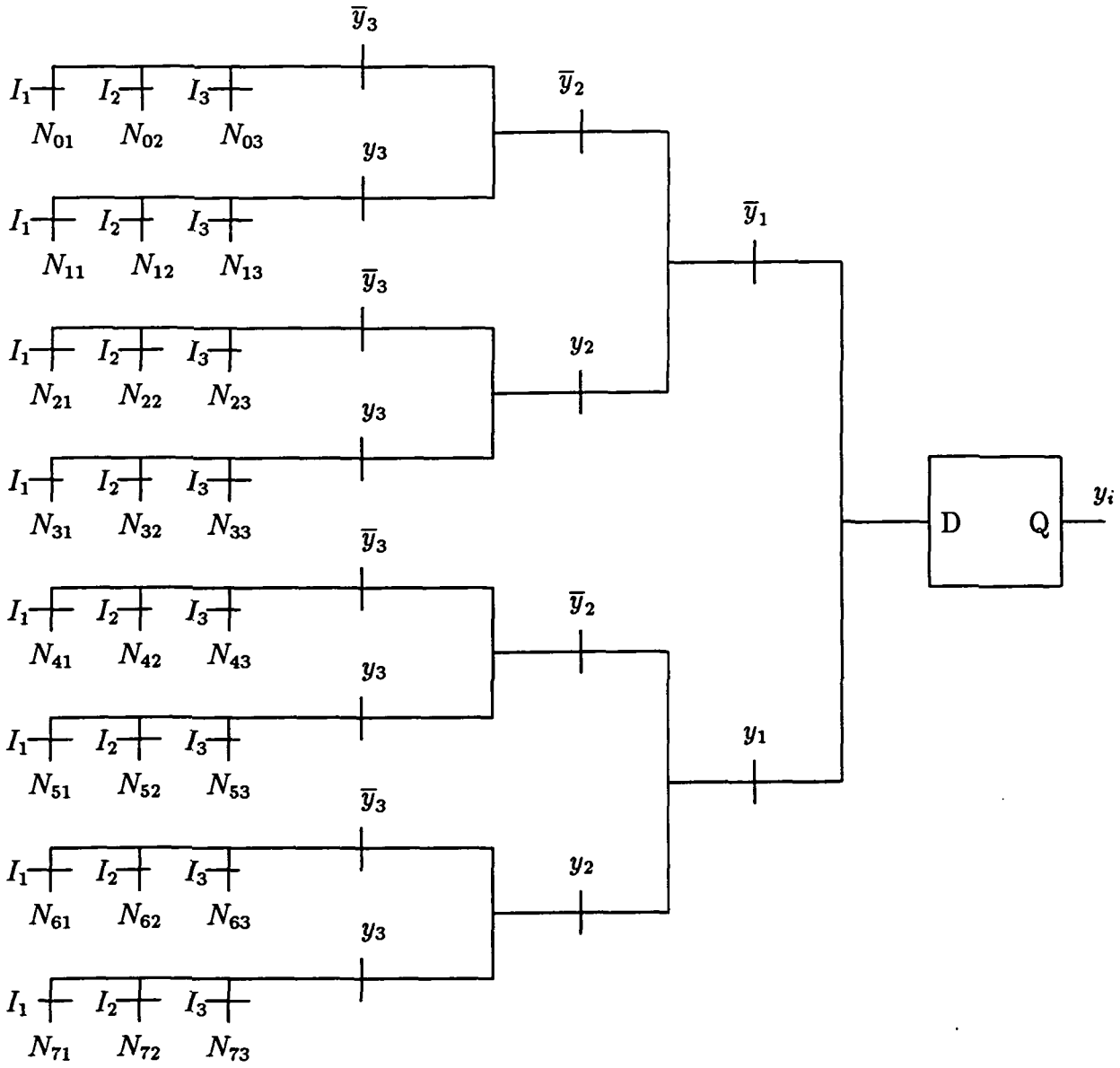

Figure 3: General 8 state 3 input next state equation circuit

	I_1	I_2	I_3	I_4	I_5	I_6
A	D	B	-	-	-	-
B	C	F	D	E	-	-
C	E	A	-	-	-	-
D	F	C	E	B	-	-
E	A	D	B	D	-	-
F	A	B	-	-	A	B

Table 2: Example flow table showing types of inputs

4.3 Decomposition using groups of inputs and states

The inputs to a state machine can be classified under three general categories: 1) those inputs under which *all* the states make transitions to other states in the machine; 2) those under which states of a *group* transition amongst themselves, and 3) inputs under which a *single* state transitions to other states. In Table 2, I_1 and I_2 belong to the first category, I_3 and I_4 to the second and I_5 and I_6 to the third.

Fragmentation of a large machine can be done, depending on the nature of inputs presented to it. An examination of the flow Table reveals which of the three classes of inputs discussed above are predominant, and decomposition can be done accordingly. The central idea here is to seek for *disjoint* groups of inputs, under which groups of states transition amongst themselves, and in addition, there should be little or no overlap between the groups; *ie.,* isolated groups of inputs with corresponding isolated groups of states should be identified. These can be formed into sub-machines.

It is clear that, if a large machine comprises of inputs which are used by *all* the states in the machine to make transitions, or if there is an even mix of the three types of inputs discussed above, as in Table 2, then breaking up of that machine is not viable, or at the most, minimal. This is because decomposing that machine would necessitate a duplication of either the inputs or the states or both in all the component machines leading to larger chip area and reduced speed. On the other hand, if a machine has inputs exclusively of the third class (under which single states transition to other states), then a maximal decomposition of the machine is possible. In other words, the machine can be fragmented into *one bit state machines*, each containing a state of the original machine, with corresponding inputs under which the state transitions to other states (in this case, to other one bit machines). This method is also called as *one hot coding*[5] and is quite popular in applications where the size of the machine is not very large, since it requires the use of one flip-flop for every state of the machine, and thus the area occupied is more. The main advantage here is ease and quickness in design, with little or no design equations, and can be used for standard cell designs. Also, in one bit machines, since there is only one state variable, the series and feedback delays are minimal and the speed of operation is enhanced.

5 Implementation using SISMs

In order to implement the sub-machines using SISMs, a few general guidelines have to be followed, during the decomposition: All the machines should have an equal number of state variables and a common maximum of the number of inputs, to take advantage of the repetitive and automated design and layout processes for SISMs. If any machine has fewer states or inputs than those of other machines, dummy states and inputs can be introduced in that particular machine, to preserve regularity. Every component machine must have a start-up/wait state, to facilitate the ready protocol and the handover of control between machines. Also, such states should have no valid outputs that are used in the normal activities of the controller *ie.,* the machine should be inactive, while it is in the start-up or wait state. If a machine does not have such states naturally, an additional state performing their functions should be introduced. An added constraint is that a machine can have only one wait state, though it can have more than one start-up state. Usually the start-up states of all machines are coded 000 or 111 to help reset the machines with a common reset signal, provided to the flip-flops.

Finally, it is a good practice to keep all states from the original machine, that transition among themselves, in a single sub-machine, rather than distribute them over a few machines. This saves unnecessary interaction and handing over of control between them, resulting in hardware savings and speed enhancement. If this is not feasible under certain circumstances (for example, when there is a single transition from a state in one group, to a state in another, both being separated by many states), then, state splitting can be used to limit the interaction.

An example to illustrate the techniques described so far, is shown in Figure 1. The original machine with 14 states and 6 outputs has been broken up around the natural bottleneck state F, into 3 sub-machines M_1, M_2 and M_3. All the machines have 6 states each and can be coded with 3 state variables. States A and F are the natural start-up and idle states for machine M_1. State X' is the combined start-up and idle state for M_2, while X" serves the same purpose for M_3. States X' and X" have been deliberately added to facilitate the ready protocol. State G of the original machine, common to group HIJK and to group LMNO, has been split into states G' and G", to eliminate any interaction between M_2 and M_3. It can also be observed that there are no common inputs among all the three machines and that there is a maximum of 3 inputs to any state.

In the normal sequence of events, at power-up, a common reset signal is applied, which leaves all the machines in their start-up states, namely, A, X' and X". Machine M_1 is activated first and proceeds through its sequence of states. On reaching state F, it waits there, and depending on the occurance of I_6 or I_{12}, either state G' or G" will be entered, and the corresponding machine activated. The return of control to M_1 is achieved when M_2 sinks back to state X' or when M_3 returns to state X", at which time, M_1 moves from state F to either of states A or B, depending on where the ready signal originated. (The ready signal is just the transition between states; for example, state J going to state X', on receipt of I_{11}, will be a ready signal for machine M_1 to go from state F to state A.)

The generation of the hardware for the machines begins with the construction of the flow Tables for the individual machines. As an example, the Table for machine M_1 is shown in

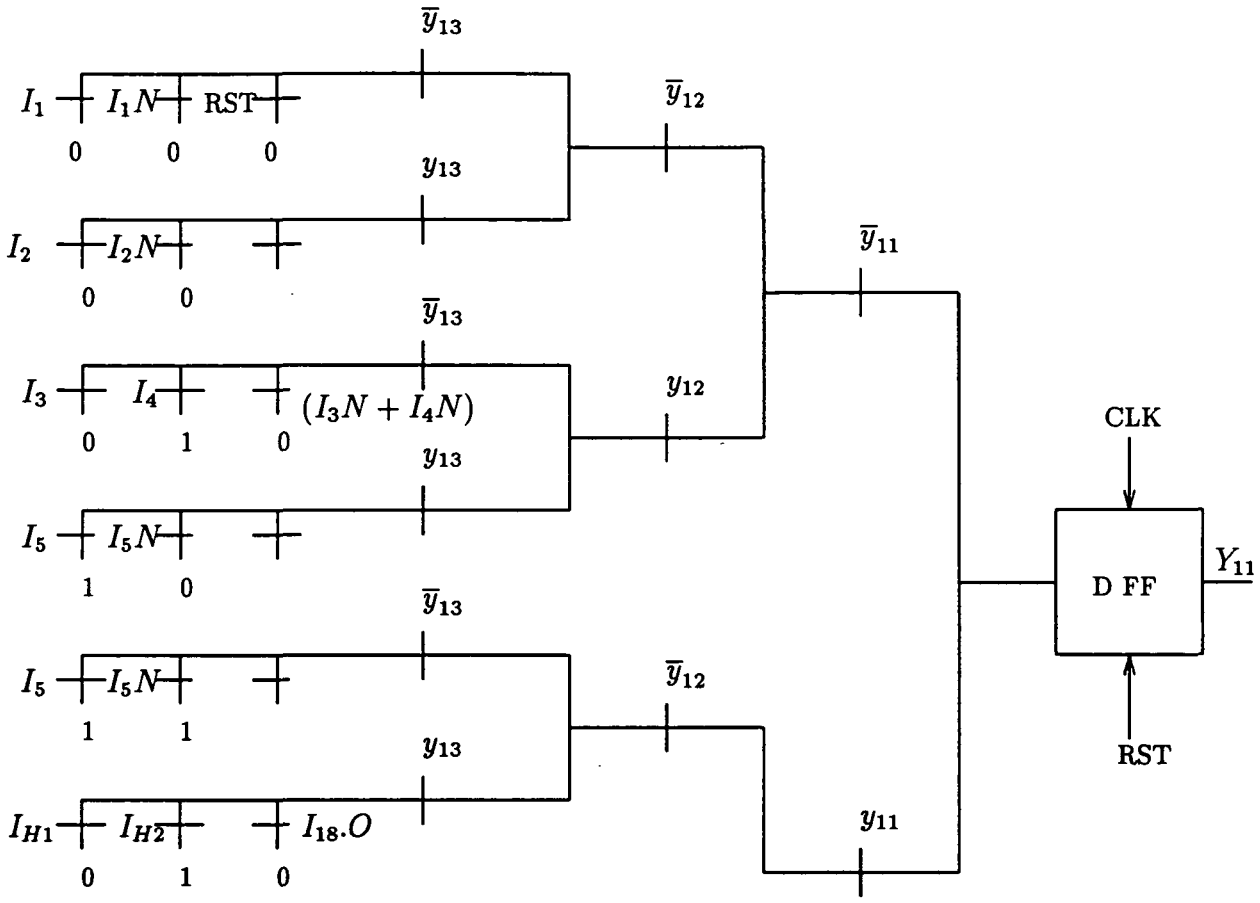

Figure 4: Next state equation circuit for Y_{11} of Table 3

Table 3. Next the circuits for the machines are generated using the SISM procedure. This is illustrated in Figure 4, for one of the next state variables, Y_{11}, of the flow Table shown in Table 3. In Figure 4, the complementary signals of all inputs (I_1N .....I_5N) have been used to prevent the input to the flip-flop from going into a high- impedance state, when the inputs are not active. For example, if the machine is in state B and I_2 has not occured for a clock cycle, and if the complementary signal, I_2N were not present as an additional input, then, the flip-flop would be presented with a high impedance at its input, thereby making its output unpredicTable on the next clock edge. However, if it can be assured externally that the input always occurs within the clock cycle, the complementary signal can be dispensed with. The output forming logic is generated as usual, using the present state codes of the states for their corresponding outputs.

6 Conclusions

The paper discussed the need for the decomposition of large state machines and a few techniques for achieving the same. The use of SISMs for implementing the component machines and the general requirements for doing so, were examined in detail. The methodology pro-

y_{11}	y_{12}	y_{13}		I_1	I_1N	I_2	I_2N	I_3	I_3N	I_4	I_4N	I_5	I_5N	I_{H1}	I_{H2}	RST
0	0	0	A	B	A	-	-	-	-	-	-	-	-	-	-	A
0	0	1	B	-	-	C	B	-	-	-	-	-	-	-	-	A
0	1	0	C	-	-	-	-	D	C	E	C	-	-	-	-	A
0	1	1	D	-	-	-	-	-	-	-	-	F	A	-	-	A
1	0	0	E	-	-	-	-	-	-	-	-	F	-	-	-	A
1	0	1	F	-	-	-	-	-	-	-	-	F	-	A	F	A

$$I_{H1} = I_{11}.J + I_{13}N.G" + I_8N.I$$

$$I_{H2} = \overline{I_{H1} + I_{18}.O}$$

Table 3: Flow Table for the component machine M_1

vided in the paper gives a simple and easy way of implementing the decomposition of large state machines. The process is also flexible, quick and automated, leading to reduced time and cost of design.

References

[1] S. Whitaker, Manjunath Shamanna and G. Maki, "Sequence-Invariant State Machines," *IEEE Journal of Solid State Circuits*, Vol. 26, August 1991, pp. 1145– 1161.

[2] D.Buehler, S. Whitaker, and J.Canaris, "Automated Synthesis of Sequence Invariant State Machines," *2nd NASA SERC Symposium on VLSI Design*, Moscow, Idaho, November 1990, pp. 4.4.1– 4.3.9.

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

[4] S. Whitaker, and G. Maki, "Pass Transistor Asynchronous Sequential Circuits," *IEEE Journal of Solid State Circuits*, Vol. SC-24, February 1989, pp. 71–78.

[5] Lee.A.Hollaar, "Direct Implementation of Asynchronous Control Units," *IEEE Transactions on Computers*, vol. C-31, NO.12, December 1982, pp. 1133–1141.

This research was supported in part by NASA under Space Engineering Research Center Grant NAGW-1406.