Appl. Math. and Comp. Sci., 1994, vol.4, No.1, 139–152

AUTOMATIC SYNTHESIS OF SELF–CLOCKED ASYNCHRONOUS STATE MACHINES AND PARALLEL CONTROLLERS

FARHAD AGHDASI*

The classical methods for the design of asynchronous state machines are usually complicated due to critical races and hazards, and necessitate special state assignment techniques and hazard free combinational logic leading to extra hardware. The necessity for such special considerations prevents asynchronous designs to take advantage of the CAD tools developed for synchronous machines. The contribution of this paper is a novel approach to such designs which enables asynchronous state machines to be systematically synthesized with minimum state variables and arbitrary state encoding. Multiple input changes are allowed. The method uses a separate data driven clock per state variable. The combinational logic is hazard free by default allowing flexibility of minimization. Mealy and Moore outputs can be generated without hazards. Simple latches in master–slave configuration are used as memory elements rendering the method suitable for implementation in SSI or VLSI. It avoids extra delay elements often necessary in self–clocked circuits. The simplicity of this method enables the design equations to be derived direct from the Algorithmic State Machines rather than flow tables. The method is illustrated by its application to the design of a VMEbus requester. The methodology is then extended to parallel controllers represented by Petri nets and automated using state assignment techniques already developed for synchronous parallel controllers.

1. Introduction

Asynchronous sequential circuits offer improved speed of operation when compared with their synchronous counterparts since they respond directly to input changes and do not have to wait for the arrival of the system clock. Moreover, they do not suffer from the problem of metastability encountered when trying to synchronize inherently asynchronous signals such as in interfacing two subsystems each running on a separate clock. An asynchronous approach allows modular design and replacement of components and adapts well to technology scaling. However, asynchronous state machines are difficult to design correctly. The classical methods of design (Unger, 1969) require careful examination of the flow table to cater for possible critical races and hazards, complicating the design procedure and often leading to extra states and additional

* Department of Electrical Engineering, University of Zimbabwe, P.O.Box M.P.167, Harare, Zimbabwe

hardware. The necessity for critical–race–free state assignments, hazard free combinational logic designs and delay elements have persuaded many logic designers to avoid asynchronous designs particularly since such restrictions render the CAD tools of synchronous designs inappropriate for their asynchronous counterparts.

Over the past three decades a number of methods have been suggested to combine the advantages of asynchronous and synchronous sequential circuits by locally generating a clock and using it to self–synchronize the machine. Such clock signals have been generated whenever an input changes (Bredeson and Hulina, 1971; 1973; Ray and Vaucher, 1974) or by controlled excitation whenever a change of inputs necessitates a change of state (Chuang and Das, 1973) or whenever the present state and the next state are different (Huertas and Acha, 1976). The main disadvantage of these methods is the requirement of delay elements in the clock path which is difficult to calculate accurately, cumbersome to add to the design. Inertial delays (Chuang and Das, 1973) to allow multiple input changes have even more adverse side–effects. The use of edge–triggered flip–flops in such methods, although enables critical race considerations to be set aside, is unsuitable for VLSI designs where simple latches are preferred (Aghdasi, 1991a; 1991b). Specific methods have also been suggested for application with SSI logic (Aghdasi, 1989), Programmable Logic Devices (Aghdasi and Bolton, 1991) and Logic Cell Arrays (Aghdasi, 1990).

In all these methods hazard free Mealy outputs remain a problem unless excessive additional hardware is used (Bredeson, 1975) or cumbersome classical methods (Unger, 1969) are applied. Efforts to use mixed operating mode (Chiang and Radhakrishnan, 1990) in order to remove essential hazards individually and then search for critical race free state assignments lead to equally complicated design methods.

Recent advances in this area offer a design methodology (Nowick and Dill, 1991a) which allows hazard free outputs from minimal or near minimal number of states with a special case of multiple input changes. To prevent hazards, state minimization algorithms are developed and automated (Nowick and Dill, 1991b). The methodology continues to rely on delay elements to cover worst–case scenarios.

2. Overview

The general scepticism towards asynchronous circuits may be removed if the methodology compares well, in simplicity and reliability, with their synchronous counterparts. Towards this end the following issues should satisfactorily be addressed:

- No special state assignments should be necessary to avoid critical races.
- No special combinatorial minimizations should be necessary to avoid hazards.
- Delay elements (inertial or otherwise) should be avoided.
- Memory elements, if used, should be simple latches rather than edge–triggered flip–flops (for easy implementation in VLSI).
- The starting point of design should be comparable in complexity with the Algorithmic State Machines rather than primitive flow tables.

The contribution of this paper is to present a novel and automated method for the design of asynchronous state machines which satisfies the above requirements.

Additionally, it has been shown that the methodology can be extended to the design of parallel controllers represented by Petri nets using state assignment algorithms already developed for synchronous parallel controllers (Amroun and Bolton, 1989).

2.1. Multiple Input Change

Asynchronous sequential circuits respond to input changes as they occur and therefore most methods developed for such designs (Unger, 1969; Chiang and Radhakrishnan, 1990) place restrictions on the input changes to ensure that after the change of only one input no other input changes until the machine has had time to respond to that input change and has reached a steady state (fundamental mode operation). It is advantageous to be able to relax this restriction such that multiple input changes can be allowed. Obviously, reasonable restrictions can still be placed on the individual signals since no physical device can be expected to respond to input changes of arbitrary high frequency. Thus, although it may be assumed that some minimum time separates consecutive changes in each signal, if two or more inputs change at the same time the machine should respond in a predefined and desirable manner. It can be argued that there is no such thing as "at the same time", the machine should respond to two input changes which have occurred within a small window 'd' as if they had occurred at the same time. Methods that allow such multiple input changes (Chuang and Das, 1973; Aghdasi, 1991a) have to suppress the spurious pulses, due to such separation of the multiple input changes, by inertial delays. Such delays, as discussed earlier, should be avoided if possible. Methods have been developed to cater for such multiple input changes that when transition from one state to the next is dependent on the change of several inputs at arbitrary times and no input burst in a given state can be a subset of another (Nowick and Dill, 1991a). In fact, because of such a restriction, no state change is effected until all the component inputs have arrived. The state machine responds to such multiple input changes in the same manner as it would to a single input change which has several components ANDed together.

Our proposed solution to the issue of multiple input change is in the manner of the state machine specification and in its method of implementation. When a machine in a particular state should respond differently to the change of either of two inputs, the designer should establish that if the two changes are perceived by the machine to have occurred at the same time which of the two responses takes precedence. This decision can be incorporated in the state machine representation by testing the input with the higher priority first. This choice can always be made since the simultaneous arrival of the two inputs or precedence of one over the other (when their occurrences are very close) is arbitrary and dependent on the delays in the various paths. It should be noted that even in synchronous designs the condition for entering a successor state cannot be a subset of the condition for entering another successor state (Sandige, 1990). In the proposed method, transition from each state to the next one is achieved through generation of a pulse for each state bit which needs to change if the present state and input levels necessary for that transition have been established. Therefore, if a state change occurs and the new state finds that the input necessary to take it to the next state is already established the necessary transition will take place. If transition from one state to the next depends on the arrival of several inputs, no

transition will happen until all the necessary inputs have arrived. An example of the treatment of multiple input changes will be discussed later during the analysis of the VMEbus requester.

2.2. State Machine Specification

One of the obstacles in the classical design of asynchronous state machines (Unger, 1969) is the need to start from a primitive flow table. For practical designs of even moderate size such flow tables can be very large and hard to manage. If the state diagrams developed for synchronous designs can easily be used for asynchronous methods, the designer would be in a position to compare the advantages of the two approaches for various modules or the whole of the design. It is shown that our design method can be implemented from any state machine specification which is based on transition from one state to the next one when certain input levels are present. This requirement is so minimal that any representation from algorithmic state machines to Petri nets would be suitable. Such representations when prepared for synchronous machines can directly be transferred for asynchronous implementation merely by recognizing that in self–clocked methodology there is no global clock in the machine and therefore no fixed cycle time. The transition from each state to the next one takes place when the necessary input condition is established. Obviously under such circumstances counters would have a different meaning unless there are certain transitions that are being counted. In a synchronous counter, the count represents a duration of time which is a multiple of the clock period. In our self clocked circuit, in the absence of any external inputs, the movement from one count to the next one would take place as soon as the previous transition is completed and therefore, it is a measure of the speed of the system.

3. The Design Method

A block diagram of the proposed method for the synthesis of an asynchronous state machine is shown in Figure 1. The machine consists of combinational logic blocks $f_1, f_2, .., f_n$ and a pair of master–slave latches for each state variable. Each master–slave flip–flop is connected in the toggling mode such that every input clock pulse complements its output. Master latches are static, slave latches are dynamic. It should be noted that unlike synchronous designs there is no fixed cycle time. There are no clock pulses unless the established values of the inputs and present states require a change in the next state values and, even then, the master latches of those next state bits which do not need to change receive no pulses. The clock of each master latch is a function of the inputs X and present states Y. The output of the same latch is used to reset the clock forcibly without hazards once the change has been effected. For example the combinational block f_1 generates the clock C_1 as follow:

$$C_1 = \overline{y_{M1}} f(\Delta y_1) + y_{M1} f(\nabla y_1) \tag{1}$$

In this equation $f(\Delta y_1)$ represents the sum–of–products function of all the present state and input conditions which require a 0–1 transition of the state variable y_1.

Similarly $f(\nabla y_1)$ represents the sum–of–products function of all the present state and input conditions which require a 1–0 transition of the state variable y_1.

The notable peculiarities of this arrangement are that each state variable is being clocked separately with its own data driven first clock phase only when that particular state variable bit should be complemented and then the transfer to the slave latches is synchronized (to ensure that possible critical races do not cause any problems) by a single NOR gate which generates the second clock phase after all the first clock phases have returned to their low level. The timing analysis and details of the critical races and hazards are considered later.

Those outputs of the state machine which do not need to be hazard–free (perhaps because they are being used only in other synchronous parts of the system and their values are tested at the rising edge of the system clock after they have finally settled) can be obtained in the standard manner (Unger, 1969) through combinational logic as a function of inputs and state variables (Mealy outputs) or as a function of state variables only (Moore outputs). This method of output generation has the usual problem of metastability associated with synchronous systems with independent external inputs which is why Mealy outputs are seldom used in synchronous systems. Our proposed method for hazard–free implementation of the outputs is shown in Figure 2. There is a master–slave latch assigned to each output such as Z_1 and is connected in the toggling mode so that the output is complemented on each arrival of the clock pulse. The clock pulse is generated by the combinational logic g_1 whenever the corresponding output should be complemented. This combinational logic is a function of the input variables X and state variables Y. The output of the corresponding master latch is used to forcibly reset the clock without hazards once the change has been effected. For example the combinational block g_1 generates the clock C_1 as follows

$$C_1 = \overline{z_{M1}} g(\Delta z_1) + z_{M1} g(\nabla z_1) \tag{2}$$

In this equation $g(\Delta z_1)$ represents the sum–of–products function of all the present state and input conditions which require a 0–1 transition of the output z_1. Similarly, $g(\nabla z_1)$ represents the sum–of–products function of all the present state and input conditions which require a 1–0 transition of the output z_1. The details of the hazard free operation of the clocks are considered later.

3.1. Races and Hazards

Having looked at the functional synthesis of our proposed method it is necessary to examine its operation at a lower level of logic and timing implementation to ensure its hazard free operation and to ascertain that possible critical races do not cause malfunction of the machine. First we consider the state machine in Figure 1. Assume the machine is in a stable state, no new input bursts have begun and all internal logic is stable. Any input burst may occur, with inputs arriving in arbitrary time and order. The following requirements have to be established:

1. If change of inputs does not require change of state, the output of the combinational logic blocks which provide the clock signals should remain at low level without hazards.

2. If a change of input requires change of state, there should be a hazard free 0–1 transition on the clocks of the corresponding master latches of those state bits which are required to change.
3. Once each clock pulse has effected the required change on the master latch, it should be reset without hazards.
4. To prevent possible critical races from causing the machine to malfunction, whenever a change of state requires the change of more than one state variable the transition from one state to the next of the affected state variables should be synchronized (McIntotsh and Weinberg, 1969).

The above requirements ensure that the operation of the machine is hazard free and that possible critical races do not cause the machine to malfunction. For the outputs shown in Figure 2, corresponding requirements 1 to 3 as above (any reference to state bits should be replaced by output variables) ensure hazard–free operation. Requirement 4 falls away since, unlike the state variables, the outputs are not fed back and therefore they do not need to be synchronized.

3.2. Logic Implementation

In this section it is shown that a simple AND–OR (sum–of–products) implementation of the combinational logic blocks of the state machine and the outputs is hazard–free by default without any special considerations and that all the above requirements to cater for hazards and critical races are satisfied. It is assumed that arbitrary fan–in AND blocks and OR blocks are available where a number of AND blocks feed directly into a single OR block. Multi–level realizations of these blocks are discussed later.

The AND–OR implementation of the combinational blocks have only 0–0 and 0–1 transitions during the input bursts. An AND–OR realization is hazard–free for all 0–0 and 0–1 transitions (Nowick and Dill, 1991a). For a 0–0 transition, each product term remains disabled since it must have some literal that remains 0 throughout the transition. If this were not true, then when certain inputs in the input burst have changed, every literal in a product term would become 1. This can only occur if the corresponding bit was supposed to change and therefore a 0–1 transition was intended which is not the case. Therefore, the requirement 1 above is satisfied. For a 0–1 transition, each product either makes a 0–0 or 0–1 transition. Using the same argument as above, a product making a 0–0 transition will be hazard–free. Remaining products make a 0–1 transition; since input changes are monotonic and the literals to such products will all be 1 only at the end of an input burst, no hazard is possible. Therefore, requirement 2 above is satisfied.

Requirement 3 above is satisfied since, following the rising edge of any of the clock signals and the toggling of the output of the corresponding master latch the clock signal is forcibly reset by the AND function of the toggling term and the output of the master latch as shown in equations 1 and 2.

Requirement 4 above is satisfied since the transfer of the changes from the master latches to the slave latches will only be effected after all the clocks have been reset and all the inputs of the NOR gate are at low level. Therefore, whenever a change of state requires the change of more than one state variable the transition from one state

to the next of the affected state variables is synchronized. It is, however, necessary to ensure that there are no unreasonable delays in any of the combinational logic blocks since by the time the first generated clock pulse has passed through the corresponding master latch, toggled its output, whose signal is fed back to the combinational logic and has reset the clock, the second clock pulse (if it is meant to appear) should have started its rising edge. In effect, if the maximum and minimum delays through the combinational blocks are δ_M and δ_m respectively, and the minimum delay through a latch is d_m, then the condition which needs to be satisfied is

$$\delta_M < 2\delta_m + d_m \tag{3}$$

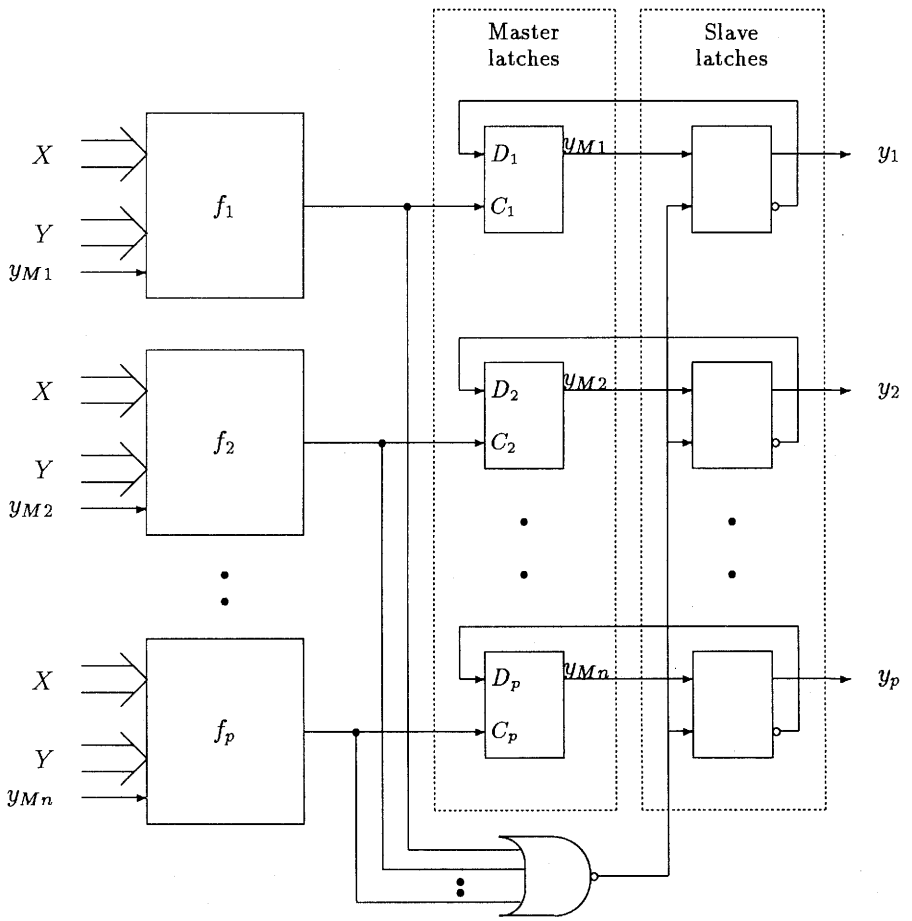

Fig. 1. Block diagram of an asynchronous state machine.

It is clear from the above discussion that the combinational logic blocks can be designed in any implementation style, including multi–level logic, so long as it avoids 0–0 and 0–1 hazards and (in the case of the state machine section) the very reasonable delay requirement stated above.

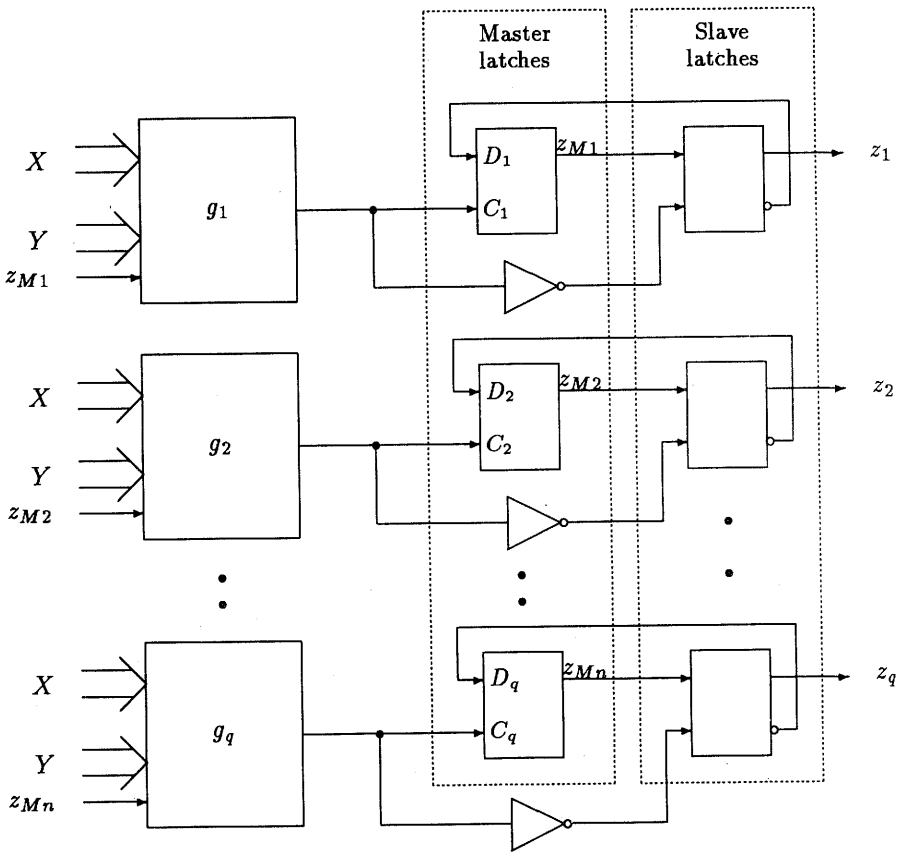

Fig. 2. Hazard–free outputs of an asynchronous state machine.

4. Design Example

4.1. VMEbus Requester

The method presented in this paper can be illustrated by a self–clocked design of a VMEbus Requester. Bus requesters are used in common bus systems that support multiple processors controlling bus transfers. The function of a bus requester is to request permission to control data bus transfers and to indicate to the master when control has been granted. The VME Bus is a common, high performance asynchronous bus that supports multiple bus masters. A self–clocked design approach to a VME bus

requester is appropriate because the VME Bus is asynchronous and high-performance. The bus request function is asynchronously initiated and sequential. A synchronous approach requires an external clock to synchronize and time the sequence. The VME Bus provides a 16 MHz system clock. Our proposed self-clocked design is much higher performance than a synchronous design using the system clock.

The state diagram for a simple VMEbus Requester is shown in Figure 3. All inputs and outputs are active low indicated by a dash following each signal name. Notice that since in the absence of any clock pulses the state variables retain their values in their respective latches, contrary to the classical asynchronous state diagrams (Sandige, 1990), each stable state does not need to have a path to itself in order to sustain the system in that state.

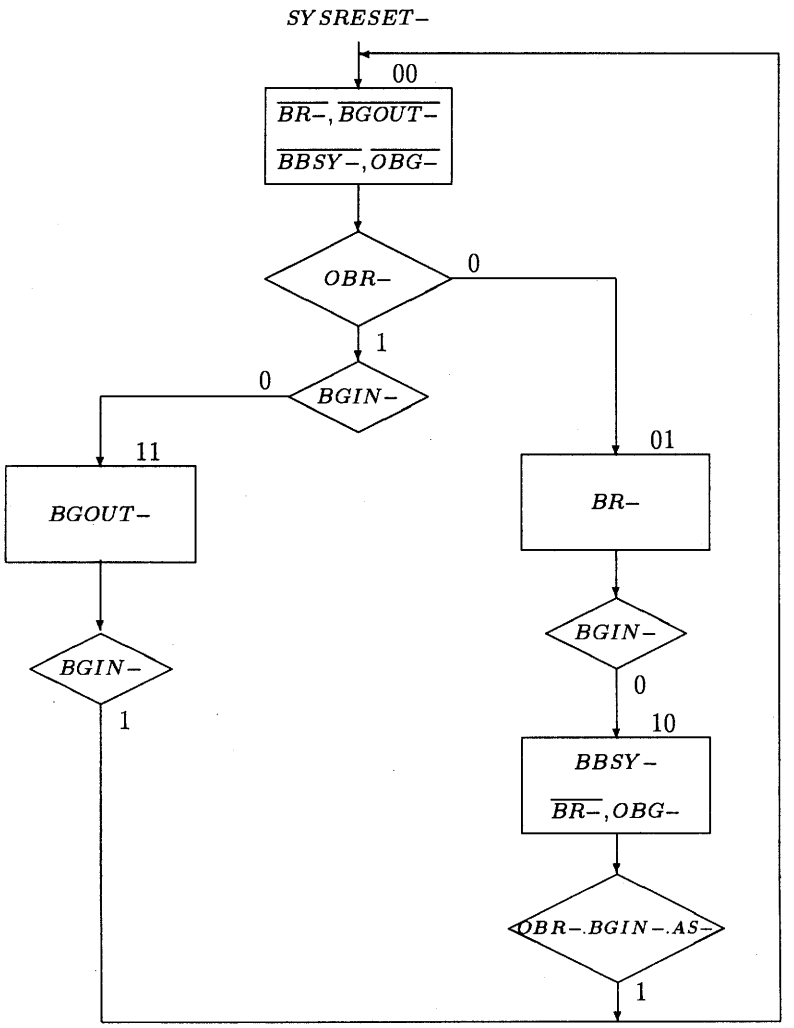

Fig. 3. VME bus requester state diagram.

The state diagram of the VMEbus requester shown in Figure 3 shows how multiple input change is catered for in our proposed method. Transition from state 10 to 00 depends on the arrival of three inputs. The system remains in state 10 and no clock pulse is generated until all three inputs have assumed the desired values. The inputs can arrive at arbitrary time and order. Only after all the input conditions are established the machine moves to the next state. Next we consider the situation when the machine is in state 00. If a request is made (OBR–=0) the machine moves to state 01. If a request is not made (OBR–=1) but a grant signal is asserted (BGIN-=0), indicating that another device had requested the bus, the grant signal is passed down the daisy chain by moving the machine to state 11. A situation can arise that OBR– and BGIN– are both asserted at the same time. This can happen if another machine has requested the bus and exactly at the same time that the grant signal was being asserted the present machine has requested the bus. In this case since OBR– is being tested first, the VMEbus requester will be granted first. If the BGIN– was tested first, the other machine would have received the bus first. This decision is rather arbitrary since the sensing of the two inputs by the machine to be simultaneous is a function of the delays in their respective paths. It should be noted that the restrictions on the change of inputs amounts to the "fundamental mode" for multiple input changes. If input changes occur before the system is stable there is naturally the possibility of a metastable condition. However, the probability of such a condition occuring is a function of the frequency of the independent signals involved. In the absence of a high frequency clock the probability of a metastable condition is much less than in its synchronous counterpart.

4.2. Implementation

To implement this VMEbus Requester using our proposed method, the equations for the clocks (1) should be obtained. The state assignment is arbitrary and minimal. The inputs OBR–, BGIN– and AS– are shown as x_1, x_2 and x_3 respectively to conform to the notation of our model. Due to the simplicity of the method, the equations can be directly written from the state diagram or from the state and excitation table shown in Table 1.

Tab. 1. State and excitation for VMEbus requester.

Present state	Inputs			Next state	Excitations			
y_1y_2	x_1	x_2	x_3	y_1y_2	Δy_1	∇y_1	Δy_2	∇y_2
00	1	0	-	11	1	0	1	0
00	0	-	-	01	0	0	1	0
01	-	0	-	10	1	0	0	1
10	1	1	1	00	0	1	0	0
11	-	1	-	00	0	1	0	1

$$C_1 = x_1\overline{x}_2\overline{y}_1\overline{y}_{M1} + \overline{x}_2\overline{y}_1 y_2\overline{y}_{M1} + x_1x_2x_3y_1y_{M1} + x_2y_1y_2y_{M1} \quad (4)$$

$$C_2 = \overline{x}_1\overline{y}_1\overline{y}_2\overline{y}_{M2} + \overline{x}_2\overline{y}_1\overline{y}_2\overline{y}_{M2} + \overline{x}_2\overline{y}_1 y_2y_{M2} + x_2y_1y_2y_{M2} \quad (5)$$

To simplify the equations for the clocks C_1 and C_2, Karnaugh maps or any other logic simplification method can be used. The design must respond to the system reset signal, SYSRESET– , taking the state machine to state 00. Therefore, the asynchronous reset inputs of the flip–flops should be connected to SYSRESET–. The implementation of this design is shown in Figure 4.

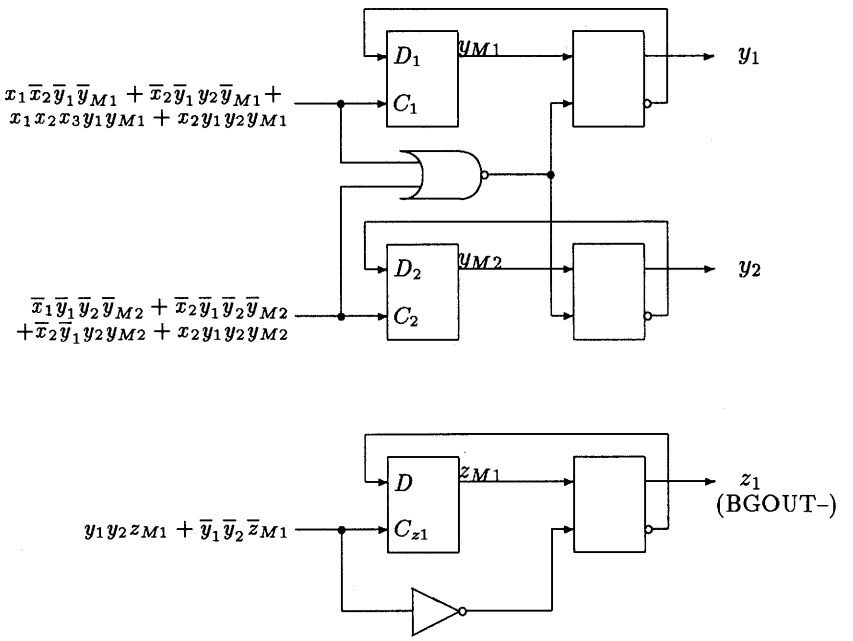

Fig. 4. Design of VMEbus requester state machine and output sample.

For the design of the outputs of the VMEbus requester it should be noted that BGOUT– is specifically required to be glitch free. Using the proposed method, the combinational logic for the clock signal of this output which is only asserted during 11 state can be derived in two ways as follows:

1. The output can be asserted ($z_1 = BGOUT- = 0$) when the machine has entered state 11 and released when the machine has entered state 00. This is the Moore type where the output is only the function of the present state.

$$C_{z1} = y_1y_2z_{M1} + \overline{y}_1\overline{y}_2\overline{z}_{M1} \quad (6)$$

2. The output can be asserted when the state and input conditions for entering the state 11 are established and it is released when the input condition for leaving the state 11 is established. This is the Mealy type where the output is a function of

the inputs and the present state. Outputs generated in this manner may require more combinational logic but are established faster. Moreover, when the output is not associated with any state and is Mealy type in the state diagram it has to be a function of the inputs.

$$C_{z1} = x_1\overline{x}_2\overline{y}_1\overline{y}_2 z_{M1} + x_2 y_1 y_2 \overline{z}_{M1} \tag{7}$$

The other outputs can either be derived in the same manner or as described in (Aghdasi and Bolton, 1991).

5. Parallel Controllers

A sequential controller has only one active state at any one time. When several such state machines are interacting they form a system of parallel controllers which can have several active local states simultaneously. Parallel controllers are best represented by interpreted Petri nets (Pardey and Bolton, 1991). Parallel controllers can be designed using our proposed self–clocked method with the state assignment algorithms or decomposition techniques developed for their synchronous counterparts. This ease of application is made possible by the fact that our self–clocked method behaves similar to a synchronous machine with the exception that there is no fixed clock cycle time. A clock pulse is generated as soon as the machine is ready to move to the next state and there are no restrictions on state assignment or minimization of the feedback logic due to critical races and hazards normally associated with asynchronous circuits. It is this similarity that allows the same state machine representation prepared for synchronous systems to be interpreted and used for our self–clocked systems.

5.1. Decomposition

A parallel controller represented by an interpreted Petri net can be decomposed into two or more subnets similar to FSMs where each state machine has only one active state at any one time (Pardey and Bolton, 1991). Each of the component state machines can then be designed using our self–clocked method.

5.2. Parallel State Assignment

A parallel controller represented by an interpreted Petri net can be implemented by state assignment algorithms which ensure orthogonality, thereby avoiding concurrent local states having conflicting state variables (Amroun and Bolton, 1989). This procedure can directly be implemented using our self–clocked method. The same state encoding can be used and transition from one state to the next one is defined by the present state and the required input values.

5.3. Isomorphic Encoding

The simplest state assignment of parallel controllers which have several local states active concurrently is isomorphic encoding. This assigns one flip–flop per state in the same manner as one–hot encoding of FSM synthesis, except that now several flip–flops can be active simultaneously. This method is particularly attractive with Logic Cell

Arrays where there are an abundance of flip–flops (Aghdasi, 1990). Our self–clocked method can directly be applied to parallel controllers with isomorphic encoding since, although each state flip–flop is being clocked separately, the transition from one state to the next one is synchronized.

6. Conclusions

A novel design method has been presented for asynchronous state machines and parallel controllers which does not require special considerations and non minimal state encoding to avoid critical races and hazards. Such race and hazard considerations which had long plagued asynchronous designs have been rendered irrelevant through the design approach. This feature makes asynchronous designs comparable in complexity and reliability with synchronous designs while retaining the basic advantages of asynchronous methodology. Through combining data and local clocking the need for delay elements often associated with self–clocked circuits has been eliminated. The method allows implementation from Algorithmic State Machines or comparable specifications used for sequential designs, or Petri net representation for parallel controllers. Mealy and Moore hazard–free outputs can be obtained. CMOS power consumption is minimal since only the bits which need to change receive a clock signal. The method uses combinational logic and simple latches and, therefore, is suitable for implementation in SSI or VLSI.

The automated design procedure is being added to SIS, which is a recent version of the MISII logic synthesizer (Brayton *et al.*, 1987) supporting sequential circuit synthesis.

References

Aghdasi F. (1989): *Design of asynchronous sequential circuits using interface protocol asynchronous cell (IPAC) PAL device.* — Proc. Int. Symp. *Computer Architecture and Digital Signal Processing*, CA–DSP'89, Hong–Kong, pp.426–430.

Aghdasi F. (1990): *Application of logic cell arrays in design of self–clocked sequential circuits.* — Proc. IEEE *TENCON'90, Computer and Communication Systems*, Hong–Kong, pp.519–523.

Aghdasi F. (1991a): *Synthesis of asynchronous sequential machines for VLSI applications.* — Proc. Int. Conf. *Concurrent Engng. and Electronic Design Automation*, CEEDA'91, Bournemouth, U.K., pp.55–59.

Aghdasi F. (1991b): *Pass–transistor self–clocked asynchronous sequential circuits.* — Proc. Int. Conf. *Very Large Scale Integration*, VLSI 91, Edinburgh, Scotland, pp.9.1.1–9.1.9.

Aghdasi F. and Bolton M. (1991): *Self–clocked asynchronous state machine design with PAL22IP6 device.* — Microprocessors and Microsystems, v.15, No.1, pp.35–41.

Amroun A. and Bolton M. (1989): *Synthesis of controllers from Petri net description and application of Ella.* — Proc. IFIP Workshop *Applied Formal Methods for Correct VLSI Design*, North Holland, pp.57–74.

Brayton R.K., Rudell R., Sangiovanni–Vincentelli A. and Wang A.R. (1987): *MIS: A multiple–level logic optimization system.* — IEEE Trans. Comp.–Aided Design, v.6, No.6, pp.1062–1081.

Bredeson J. (1975): *Comments on 'synthesis of multiple input change asynchronous machines using controlled excitation and flip-flops,'* – H.Y.U. Chuang, S. Das *Author's reply.* — IEEE Trans. Comp., v.C–24, No.11, pp.1142–1144.

Bredeson J.G. and Hulina P.T. (1971): *Generation of a clock pulse for asynchronous sequential machines to eliminate critical races.* — IEEE Trans. Comp., v.C–20, pp.225–226.

Bredeson J.G. and Hulina P.T. (1973): *Synthesis of multiple input change asynchronous circuits using transition-sensitive flip-flops.* — IEEE Trans. Comp., v.C–22, No.5, pp.524–531.

Chiang J. and Radhakrishnan D. (1990): *Hazard-free design of mixed operating mode asynchronous sequential circuits.* — Int. J. Electronics, v.68, No.1, pp.23–37.

Chuang H.Y.H. and Das S. (1973): *Synthesis of multiple input change asynchronous machines using controlled excitation and flip-flops.* — IEEE Trans. Comp., v.C–22, No.12, pp.1103–1109.

Cypress Semiconductor (1988): *CY7C331 application example: asynchronous, self-timed VME bus requester.*

Huertas J.L. and Acha J.I. (1976): *Self-synchronization of asynchronous sequential circuits employing a general clock function.* — IEEE Trans. Comp., v.C–25, No.3, pp.297–300.

McIntotsh M.D. and Weinberg B.L. (1969): *On asynchronous machines with flip-flops.* — IEEE Trans. Comp., v.C–18, No.5, p.473.

Motorola (1982): *VMEbus Specification Manual.* — MVMEBS/D1.

Nowick S.M. and Dill D.L. (1991a): *Synthesis of asynchronous state machines using a local clock.* — Proc. Int. Conf. *Computer Design*, ICCD'91, IEEE Computer Society Press, pp.192–197.

Nowick S.M. and Dill D.L. (1991b): *Automatic synthesis of locally-clocked asynchronous state machines.* — Proc. Int. Conf. *Computer Aided Design*, ICCAD'91, IEEE Computer Society Press, pp.318–321.

Pardey J. and Bolton M. (1991): *Logic synthesis of synchronous parallel controllers.* — Proc. IEEE Int. Conf. *Computer Design*, pp.454–457.

Ray C.A. and Vaucher J. (1974): *Self-synchronized sequential machines.* — IEEE Trans. Comput., v.C–23, No.12, pp.1306–1311.

Sandige R.S. (1990): *Modern Digital Design.* — New York: McGraw–Hill.

Unger S.H. (1969): *Asynchronous Sequential Switching Circuits.* — New York: Wiley.

Unger S.H. (1971): *Asynchronous sequential switching circuits with unrestricted input changes.* — IEEE Trans. Comp., v.C–20, pp.1437–1444.

Received October 6, 1992
Revised September 7, 1993