LOCALLY CLOCKED AFSMS WITH DYNAMIC LATCH IMPLEMENTATION

Jari Pasanen and Bengt Oelmann

Department of Information Technology, Mid Sweden University
S-851 70 Sundsvall, Sweden
{Jari.Pasanen@ite.mh.se}

ABSTRACT

Asynchronous Finite State Machines (AFSMs) have been proposed to be used in designs where the demands on high speed or low power consumption are high. In this paper we present a synthesis procedure for a type of AFSMs called locally clocked state machines with dynamic latch implementation. The use of dynamic latches makes it possible to reduce input capacitances and the number of transistors. It also enables efficient implementation of gates with monotonic output transitions which is important in AFSM design. We (1) show what implications the use of dynamic gates have on the synthesis procedure, (2) define state constraints and requirements on these circuits, and (3) present a complete procedure for implementing AFSMs through an example.

1. INTRODUCTION

Asynchronous (or self-timed) circuits [1] are sometimes proposed as the solution to problems related to the distribution of the global clock signal in a synchronous system. Several complex digital CMOS ICs have been designed using asynchronous circuit exclusively [2,3]. However, no complex asynchronous design has, in a convincingly way, demonstrated its superiority over a synchronous solution. For smaller and more specific problems asynchronous finite state machines have demonstrated better performance than its synchronous counterpart [4]. There are basically three situations where AFSMs can be used successfully; (1) for high speed circuits where it is not feasible to distribute a high frequency clock, (2) off-chip interfaces between devices that are not synchronized, and (3) where the requirements on response time are shorter than the clock period.

From research during the last three decades a large number of methods and design techniques for locally clocked sequential circuits have emerged. An extensive review of these is presented in [5]. In the 90's a design methodology for locally-clocked state machines was developed at HP Laboratories [6]. These AFSMs work accordingly to the burst-mode specification. One major advantage with this method is the use of selective clocking which results in circuits with very short response time that for some state changes only is a combinational response.

Dynamic CMOS techniques are widely used in digital VLSI design when high performance is targeted. A number of different dynamic logic families have been developed during the past years, such as C^2MOS [7], NORA [8], TSPC [9], and all-N logic TSPC [10]. In general, dynamic logic provides advantages such as high speed and area efficient circuit implementations, reduced input capacitance, and monotonic glitch-free operation.

In this work we have investigated the implications of using dynamic latches instead of static ones, that has in so far been used in locally-clocked state machines. The presentation is given in a general way that applies to all dynamic latches. The outline of the rest of the paper is as follows. First a brief description of burst-mode AFSM is given and then the proposed dynamic AFSM is presented. After that we put forward a synthesis procedure and finally we discuss the performance of the dynamic AFSM in relation to the static.

2. BURST-MODE AFSM

A burst-mode asynchronous state machine is specified by a form of state diagram [11] containing a finite number of states connected by a number of arcs. Each arc, representing a state transition, is labelled with a non-empty set of input signals (an *input burst*) and a set of output signals (an *output burst*). In a given state, when a complete input burst has arrived, the machine responds with an associated output burst and transits into the next state.

An input burst is permitted to have multiple input changes in random order, where only specified input changes may occur. Also, no input burst in a given state can be a subset of another so that the machine can unambiguously determine the complete input burst.

The burst-mode machine operates accordingly to the generalized *fundamental-mode* requirement; that is, new inputs are not allowed to arrive until the system is settled into its new state. Another restriction is that a given state must always be entered with the same set of input values.

A simple example of a burst-mode specification is shown in figure 1. This specification will be used as a design example throughout the paper.

0-7803-5682-9/99/$10.00©1999 IEEE.

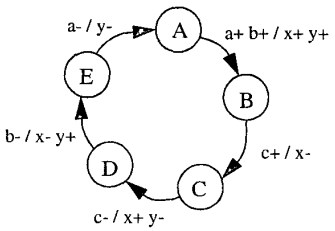

Fig. 1: Simple example specification

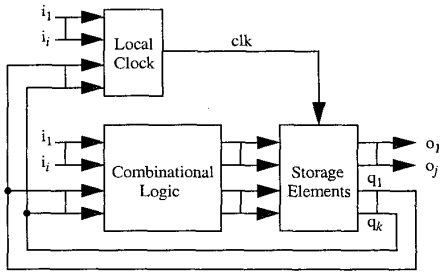

Fig. 2: Generalized block diagram of a locally-clocked AFSM

An asynchronous state machine that satisfies the burst-mode specification can be implemented as a locally-clocked machine. A generalized block diagram of such AFSM is shown in figure 2. The machine is a type of self-synchronous circuit in which synchronization is handled by a local clock generator unit. Additional circuitry is combinational logic and clock-controlled storage elements.

The clock, which is used to control the state changes of the machine, depends only on the current inputs and state. For a given input burst, the machine reacts by generating the corresponding output burst and, if a state change is necessary, the clock is fired which in turn updates the state variables.

3. DYNAMIC LOGIC IMPLEMENTATION

A locally-clocked burst-mode AFSM can be implemented with dynamic latches. A dynamic latch is depicted as a clocked gate which incorporates both a logic and a latch function; it also comes in two flavors to be used in an alternating fashion (see [7,8,9,10]). Thus, dynamic latches can substitute the combinational logic and storage elements of figure 2.

A block diagram of the proposed dynamic AFSM, representing the simple example specified in figure 1, is shown in figure 3. The *phase-1* and *phase-2* latches are controlled by the clock, where the clock logic itself is unlatched and static. Each *output variable* has a single phase-1 latch while each *state variable* has a pair of phase-1 and phase-2 latches. Hence, this pair-constellation behaves like an edge-triggered flip-flop; it also forms the next-state function in two steps.

Which clocking strategy to be used depends on the choice of dynamic technique. For example, using the

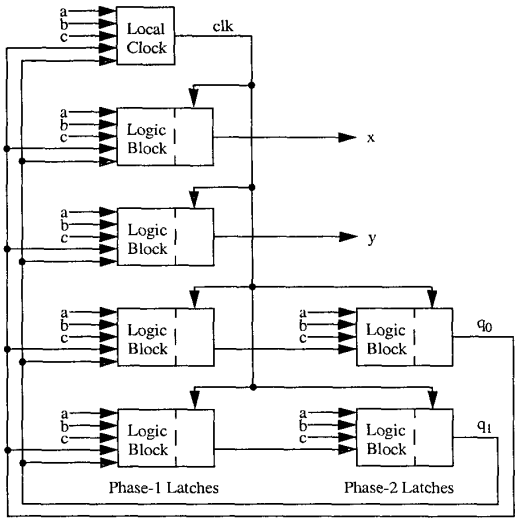

Fig. 3: Block diagram of a dynamic AFSM

TSPC technique [9] would require a true-single-phase clock and a pipeline system of N-block and P-block structures. In this case, a correct implementation is possible if using N-block structures as phase-1 latches and P-block structures as phase-2 latches.

A dynamic latch can only be in either a *precharge* or an *evaluation* phase. During a precharge phase the latch is holding its current output value while during an evaluation phase the output is determined by the gate inputs. When the machine is in a stable state, the phase-1 latches are in evaluation phase and the phase-2 latches are in precharge phase. During a state change of the machine these phases will be swapped.

A phase-1 latch is in evaluation phase during an input burst; it may also remain in this state between two consecutive bursts. This imposes that a phase-1 latch must resemble the transparent behavior of the general machine (see figure 2). However, due to the monotonic nature of a dynamic latch input burst arriving in sequential order may inflict an erroneous behavior. To illustrate this behavior, an example of a non-inverting dynamic latch is offered in figure 4.

The operational principle of the latch is addressed as follows: When *CLK* is *low* (the precharge phase), PMOS *P1* is on, node *A* is *high*, PMOS *P2* is *off*. Because NMOS *N2* is *off*, the output node *OUT* holds the previous value. When *CLK* is *high* (the evaluation phase), NMOS *N1* and *N2* are *on*. If the *N-logic* is logically *0*, node *A* remains *high* and causes node *OUT* to fall to *ground*. If the *N-logic* is logically *1*, node *A* is discharged to *ground* and node *OUT* is *high*. However, once the node *A* has been discharged the *N-logic* can no longer affect the output value (it will remain *high*). This logical *stuck-at-one* state demonstrates the *monotonic* (glitch-free) behavior of a dynamic circuit.

Hence, for sequential input changes (starting with node *A high*) the logic block is only allowed to make the

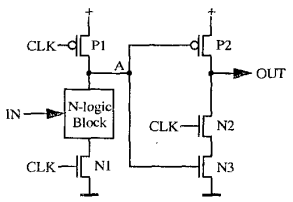

Fig. 4: Example of a dynamic latch (TSPC-1)

logical transitions 0→0, 0→1 and 1→1 to resemble a static behavior. A transition of 1→0 would require that a new precharge phase must take place (i.e. the latch must be clocked) before this transition actually can alter the output value; this is denoted as the *recharge requirement* of a dynamic circuit.

The characteristics of the dynamic latch will affect the synthesis procedure described in the next section.

4. SYNTHESIS PROCEDURE

We now present a complete synthesis procedure for a dynamic AFSM. The procedure is outlined by the properties of dynamic logic and exemplified by the simple example described earlier (see figure 1 and 3).

4.1 Functional Synthesis

Given a burst-mode specification, the first steps in synthesis is to merge compatible states of the specification, assign state codes, and generate boolean expressions for the clock and each output and state variable.

4.1.1 State Encoding

For state minimization, the specification is translated into a *flow table* [12], indicating output and next-state values. Each row of this table represent a node in the original specification and each column represents a unique combination of input signals. A stable state is represented in the table if a next-state value is the same as that of the current row, otherwise this is an unstable state. An input burst begins at a stable entry point of some row. Other entries in the same row may be visited during the input burst.

For stable behavior, all reachable entries must have specified output and next-state values; remaining entries may be specified don't-cares.

At this point, the incompletely specified flow table can be minimized using standard techniques [12,13]. The result is a set of *maximal compatible* states; a collection of the largest sets of state rows which can be merged. The set of compatible states selected to be merged indicate a new minimized state. The final choice of minimized states is however constrained by the *recharge requirement*. Refining this requirement it states that, for some output transitions it is necessary to change the state and therefore some compatible states may not be merged.

Finally, merged states are assigned unique state codes, and the minimized flow table is used to compute the *sum-of-products* (SOP) boolean expressions for the clock and each output and state variable. These expressions can be described in, and synthesized from output- and transition-tables (see figure 5 for tables generated for the simple example) using standard synthesis procedures [14].

4.1.2 Output Table Construction

The output table is generated as follows. (1) Every table entry that corresponds to a *stable state* must be specified. If an input burst causes a state change; (2) all reachable entries during the input burst must be specified unless there is a *stuck-at-one* condition, (3) except for the entry representing the *complete* input burst which must be specified to avoid *stuck-at-one*. (4) If an input burst does not cause a state change; all reachable entries must be specified unless there is a *stuck-at-one* condition. (5) Remaining entries are don't cares.

4.1.3 Transition Table Construction

Transition-table entries are specified only when they represent a *completed* input burst. Remaining entries are specified don't-cares, including transient states visited during feedback transitions.

4.1.4 Clock Table Construction

The clock table is generated as follows. The clock is stable during an input burst. If a completed input burst causes a state change the clock is fired, otherwise it remains quiescent. Remaining entries are don't cares; these include transient states visited during feedback transitions.

Outputs x y / Inputs a b c

States q_0 q_1	000	001	011	010	110	111	100	100
00	00 (A)	--	--	00	11 (B)	--	--	00
01	--	--	--	--	--	01 (C)	--	--
11	--	--	--	--	10 (D)	--	--	--
10	0-	--	--	--	--	--	--	01 (E)

(a) Output table

Next-state q_0^* q_1^* / Inputs a b c

States q_0 q_1	000	001	011	010	110	111	100	100
00	-- (A)	--	--	--	-- (B)	01	--	--
01	--	--	--	--	11	-- (C)	--	--
11	--	--	--	--	-- (D)	--	--	10
10	00	--	--	--	--	--	--	-- (E)

(b) Transition table

Clock clk / Inputs a b c

States q_0 q_1	000	001	011	010	110	111	100	100
00	1 (A)	-	-	1	1 (B)	0	-	1
01	-	-	-	-	0	1 (C)	-	-
11	-	-	-	-	1 (D)	-	-	0
10	0	-	-	-	-	-	-	1 (E)

(c) Clock table

Fig. 5: Function tables for the simple example

4.2 Implementation Requirements

To insure correct operation of an implementation, a number of conditions are presented below. First, a couple of timing requirements must be met to avoid race conditions through the latches.

Requirement 1. *The minimum propagation delay through the clock logic must be greater than the maximum propagation delay through the latches of every output and state variable.*

Requirement 2. *The delay between disabling the phase-1 latches and enabling the phase-2 latches must be less than the minimum delay in the feedback path.*

The next requirement insures correct logic implementation.

Requirement 3. *The output and clock logic must be free of logic hazards for every permitted input burst in every state.*

And finally, the last requirements insures that the clock resets correctly.

Requirement 4. *The clock logic must be stable before it resets.*

Requirement 5. *Each resetting clock transition must be free of all hazards.*

Requirements 1, 2 and 4 can easily be satisfied by adding delay elements to the clock output and the feedback lines. Requirements 3 and 5 can always be satisfied by correct functional synthesis of the clock and output functions (see section 4.1 and [13,14]).

5. CONCLUSION

A synthesis procedure has been presented enabling design of locally-clocked AFSMs with dynamic latch implementation.

Dynamic logic has several properties that can be exploited when designing efficient AFSMs. First, the next-state logic and output logic can efficiently be implemented with a small number of transistors, which leads to reduced power consumption and good speed performance. Second, the monotonic transition of the output of a dynamic gate removes the need of redundant logic for generating hazard-free signals. This contributes to the reduction of the logic expressions by leaving some table-entries don't-care. Third, half-cycle pipelining in phase-1 and phase-2 latches improves the speed performance. For complex AFSMs the computation of the next-state can be partitioned and performed in both phase-1 and phase-2. The complexity of the logic tree can be simplified and thus the gate delay will be reduced. We have described that the dynamic implementation requires more state changes than a static one would do and it is a consequence of the recharge requirement of the dynamic latch. Since every state change requires a clock generation, we have seen that the clock generation logic will in most cases become larger for the dynamic implementation.

6. REFERENCES

[1] S. B. Furber, P. Day, J. D. Garside, N. C. Paver and J. V. Woods, "A micropipelined ARM", *Proc. of the IFIP TC10/WG 10.5 Int. Conf. on Very Large Scale Integration*, Grenoble, France, Sept. 1993, Ed. T. Yanagawa and P.A. Ivey, Pub. North Holland

[2] K. van Berkel, R. Burgess, J. Kessels, A. Peeters, M. Roncken and F. Schalij, "A fully-asynchronous low-power error corrector for DCC player", *Proc. Int. Conf. Solid State Circuits*, Feb. 1994, pp. 99-106

[3] B. Oelmann, H. Martijn, and H. Tenhunen, "VLSI implementation of DS-CDMA receiver using asynchronous design techniques", *Proc. of the 6th IEEE Int. Conf. on Personal Indoor Mobile Radio Communication, Toronto*, Canada, Sept. 1995

[4] S. M. Nowick, M. E. Dean, D. L. Dill and M. Horowitz, "The design of a high-performance cache controller: a case study in asynchronous synthesis", *Integration, the VLSI J.*, vol. 15, no. 3, Oct. 1993, pp. 241-262

[5] F. Aghdasi, "Survey of self-clocked controllers", *Microelectronics J.*, 26 (1995), pp. 659 - 682

[6] S. M. Nowick and D. L. Dill, "Synthesis of asynchronous state machines using a local clock", *Proc. of the 1991 IEEE Int. Conf. on Computer Design*, IEEE Computer Society Press, Oct. 1991, pp. 192-197

[7] Y. Suzuki, K. Nogami, and T. Abe, "Clocked CMOS calculator circuitry", *IEEE J.*, vol. SC-8, Dec. 1973, pp. 462-469

[8] N. F. Gonclaves and H. J. DeMan, "NORA: A racefree dynamic CMOS technique for pipelined logic structures", *IEEE J. Solid-State Circuits*, vol. SC-18, June 1983, pp. 261-266

[9] J. Yuan and C. Svensson, "High-speed CMOS circuit technique", *IEEE J. Solid-State Circuits*, vol. 24, Feb. 1989, pp. 62-70

[10] R. X. Gu and M. I. Elmasry, "An all-N-logic high-speed single-phase dynamic CMOS logic", *IEEE Int. Conf. on Circuits and Systems*, vol. 4, 1994, pp. 7-10

[11] B. Coates, A. Davis, and K. Stevens, "The post office experience: designing a large asynchronous chip", *Proc. of the 26th Annual Hawaii Int. Conf. on System Sciences*, vol. 1, Jan. 1993, pp. 409-418

[12] S. H. Unger, *Asynchronous sequential switching circuits*, Wiley-Interscience, New York, 1969

[13] E. J. McCluskey, *Logic design principles*, Prentice-Hall, 1986

[14] S. M. Nowick and D. L. Dill, "Exact two-level minimization of hazard-free logic with multiple-input changes", *Proc. of the 1992 IEEE Int. Conf. on Computer-Aided Design*, IEEE Computer Society Press, Oct. 1992, pp. 192-197