DESIGN OF SELF-SYNCHRONIZED COMPONENT FSMS FOR SELF-TIMED SYSTEMS.

Loc Bao Nguen, Marek Perkowski +, and Lech Jozwiak †.
Intel Supercomputer, Beaverton, USA; + Portland State University,
Dept. of Electr. Engng., Portland, Oregon 97207, mperkows@ee.pdx.edu,
† Technical University of Eindhoven, The Netherlands

1 INTRODUCTION.

Little has been published on *self-synchronized circuits*, a design style that is very useful in practical controller design; for instance in interfaces and controls of self-timed realizations. There are few researches on VLSI realization of low power (pass transistor-based) and other asynchronous self-synchronized sequential circuits [16, 10, 11, 1, 2]. The goal is to achieve fast circuits, with low power consumption, for deep sub-micron technologies.

The U.C. Berkeley project [1] proposes synchronization and communication between a number of processors that operate at varying clock rates and voltage levels. They propose the use of a *"data-driven asynchronous approach at the protocol level"*. This is combined with an *"locally synchronous island"* approach at the circuit level that allows for the reconfigurable interconnect network to operate in a self-timed mode at low voltage swing. "Processor modules can be operating in either synchronous or self-timed mode at arbitrary voltage levels. The combination of a data-driven communication protocol and the locally synchronous islands eliminates the occurrence of synchronization failures" [1].

However, no specialized set of EDA tools or PLDs are now available that can be used for fast prototyping of such systems and FPGA components in them. This paper proposes a new design style for self-synchronized state machines, demonstrates their usefulness by speed analysis, compares variants using a simple example, and proposes new types of EPLD/FPGA chips to aid in the board-level design of self-synchronized circuits.

2 BASIC DEFINITIONS

Single input change (SIC) mode: a machine can have many inputs but only one input is allowed to change level to cause the machine to enter the next state. **Multiple input change (MIC) mode:** more than one input level is allowed to change and all changes within some small interval are accepted as if they were simultaneous. A machine operates in **Unrestricted input change (UIC) mode** if there are no constraints in the possible input sequences. UIC mode can produce races between input variables. There are three types of races: **[1]** between two state variables, **[2]** between two input variables, **[3]** between state and input variables. (A general approach commmon to all these cases should be developed, since until now they are discussed separately in the literature.)

A machine operates in **Single output change (SOC) mode** if any input sequence causes only one state transition. All synchronous circuits operate in SOC mode. We will consider mostly the SOC mode in this paper. A machine operates in **Multiple output change (MOC) mode** if any input sequence causes the machine to perambulate through states before reaching the stable state (see [11] for details on MOC case).

The ultimate goal of the presented work is **to propose a new self-synchronized/mixed EPLD/FPGA device that will be better than the existing programmable devices adapted for realization of asynchronous machines**. The priorities should be: **[1]** faster, **[2]** allowing larger machines of more flexible structures, including controllers and data-flow sequential circuits. **[3]** not too expensive in the sense of the area. In this order of importance, with the highest priority to speed.

3 SELF-SYNCHRONIZED CIRCUIT STRUCTURE

Fig. 1 is a diagram by Rey and Vaucher [13] that shows how the self-synchronized machine operates. From the flow-chart, the operation can be summarized in the following steps: **[1)]** Detect the input change (a Change Detector). **[2)]** Let the inputs stablize by continuing to sample the input changes within a window with respect to the last input change. **[3)]** Trigger the state machine by creating a clock pulse. **[4)]** If the state variables are stable, go back to 1 (this is for the SOC case).

With respect to the hardware, a self-synchronized machine can be represented by the block diagram shown in Fig. 2. **The MOC case machine** can be represented by the block diagram from Fig. 3. Notice that the **MORE signal** has been added to tell the Clock Generator that more transitions are needed. The Clock Generator uses the state of MORE each time to generate an additional clock pulse. The signal MORE is produced by a combinational circuit that compares the overall state of the machine before the clock with the predetermined final overall state. If these states are not equal, MORE will be high. MORE is fed directly into a T flip-flop in the Clock Generator. Therefore, when the clock pulse occurs, the output of the T flip-flop changes. This change will be captured in the Change Detector to generate another clock pulse. If MORE is low at the clock pulse, the sequence ends. Let us observe that for the machine from Fig. 2 the combinational logic is the same as for the synchronous machine. For the MOC case the combinational logic includes, additionally, the MORE output function. The Clock Generator is added in both cases.

4 CLOCK GENERATOR

The Clock Generator scheme presented here is detailed by Kirkpatrick [10, 11]. The Clock Generator consists of two blocks: the Change Detector and the delay element as shown in Fig. 4. The output of the Change Detector block is the signal DIFFER. The outputs of the delay element block are the signals CHANGE and CLOCK. The circuits behavior can be expressed by the following steps (see also Figures 11-13, they will be analyzed in detail in section 6).

[1)] DIFFER, CHANGE, and MORE are low. The

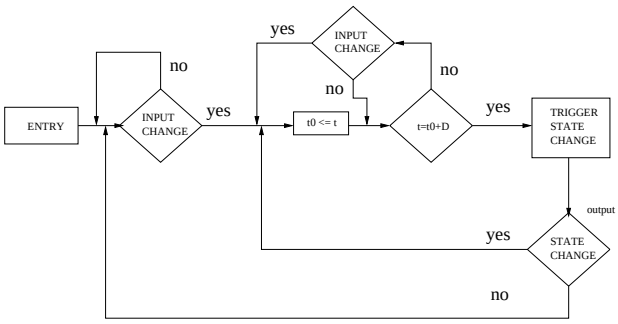

Figure 1: *Diagram of operation of self-synchronized machine, for MOC model [13].*

Change Detector and the machine are ready to accept input changes.

[2)] If there is an input change, DIFFER will go high to indicate that a change in inputs has been detected.

[3)] After a later time that is predictable by the delay, CHANGE emerges which is fed back to shut off the Change Detector. During this time, DIFFER is high and CHANGE is low and more input changes are allowed.

[4)] Eventually, DIFFER will go low but CHANGE will still be high. At this time, changes combined with the present state propagate through the combinational logic and setup the flip-flops as the next state of the machine. MORE is also updated, Fig. 3.

[5)] Finally, again through the delay, CHANGE goes low (DIFFER = CHANGE = low) and CLOCK goes high to trigger the machine and enable the machine again (SOC case). Note that in the MOC case, the signal MORE will cause more clock pulses so that the machine can perambulate through states until it finds a stable state. The Change Detector is kept off during the time of perambulation.

[6)] With DIFFER = CHANGE = MORE = low, the machine is now ready for another input excitation.

5 CHANGE DETECTOR

The Change Detector can be realized as shown in Fig. 5. In the beginning, the inputs I_i, $i \in \{1..n\}$ and the output of the latch are the same. Hence, the signal DIFFER is inactive (low). As soon as one or more of the inputs I_i, $i \in \{1..n\}$ changes levels, the corresponding EXOR gate will detect

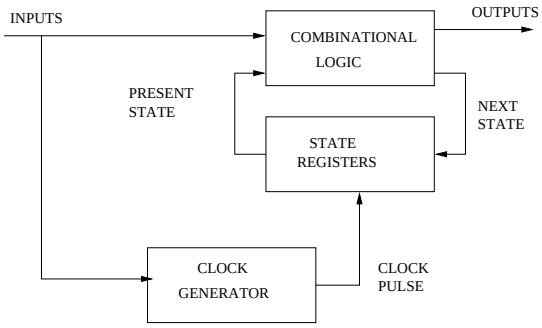

Figure 2: *Block diagram of self-synchronized machine.*

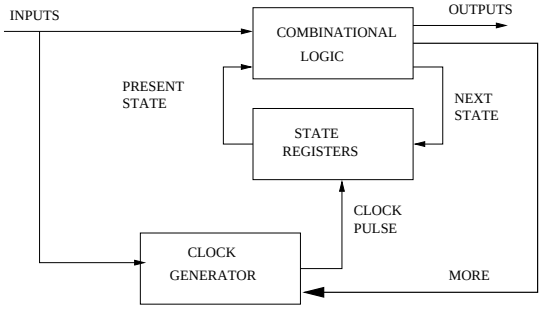

Figure 3: *Block diagram of MOC case.*

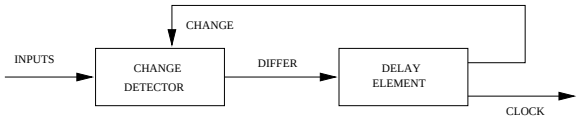

Figure 4: *Change Detector and delay element of the Clock Generator.*

the change and will go high. DIFFER will follow them. Later, CHANGE is generated to open up the latch. Then, the change from the input propagates through the latch to the EXOR gates. Eventually, DIFFER goes low and CHANGE goes low again to shut off the latch. This completes a sequence of input detection. As an example, an eight input Change Detector can be built using only two commercially available parts: one 74F373 eight-bit latch and one 74F521 eight-bit equality comparator as shown in Fig. 6.

A **Symmetrical delay** is a pure delay line which transforms (shifts) the input signal in time by the amount of time D. This delay can easily be realized using gates in series or using commercially available digital delay lines as shown in Fig. 7.

An **Asymmetrical delay** is a delay in which the

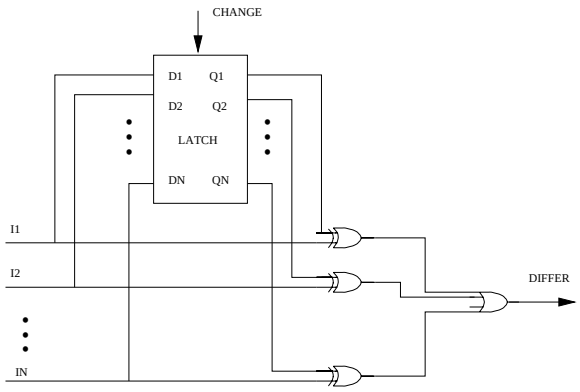

Figure 5: *Realization of Change Detector.*

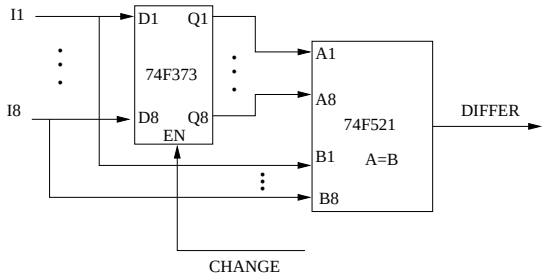

Figure 6: *74F373 eight-bit latch and 74F521 eight-bit equality comparator.*

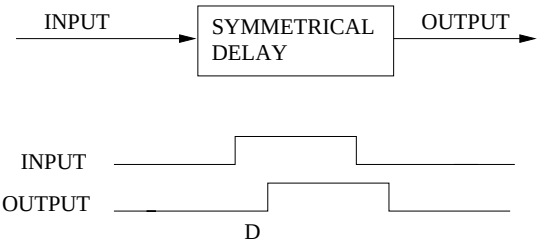

Figure 7: *Symmetrical delay.*

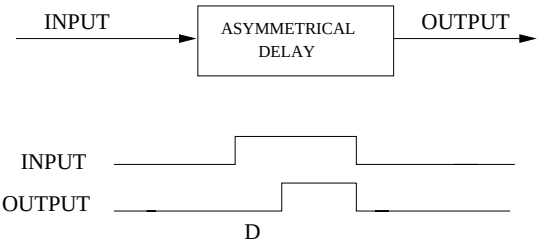

Figure 8: *Asymmetrical delay.*

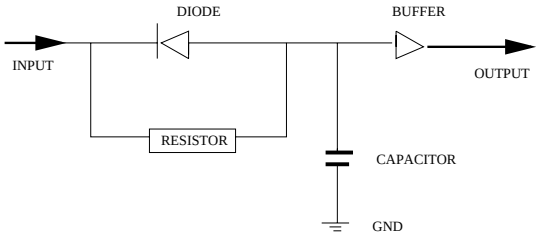

Figure 9: *Realization of asymmetrical delay.*

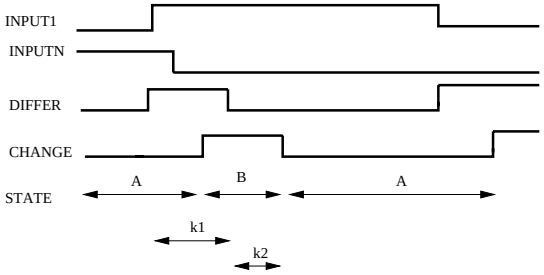

Figure 10: *Diagram using symmetrical delays.*

leading edge of the input change is delayed by the amount D. The trailing edge, however, is propagated without delay as shown in Fig. 8. The asymmetrical delay can be realized as shown in Fig. 9. Thus, the trailing edge speed is limited only by the technology. Different circuit implementations have been discussed [10, 11].

6 FUNCTIONAL OPERATION

The operation of the self-synchronized circuits can easily be understood by studying the timing diagram shown in Fig. 10. **Notation:** $[A]$ means that the machine is ready to accept input changes. $[B]$ means that the inputs have to remain stable for the proper operation. $[k1]$ is the time interval for which several input signals may change. $[k2]$ is the time interval for which input signals may not change while the machine perambulates from one state to the goal state. If the input signals change during this interval, unpredictable behavior will occur. Hence, the machine may malfunction accordingly.

Case 1: Fig. 10 shows the diagram using symmetrical delays. The problem with symmetrical delays is that they operate correctly when we have the control of the inputs. Otherwise, the machine may malfunction if the change of an input occurs during state B. If an input change occurs during state B,

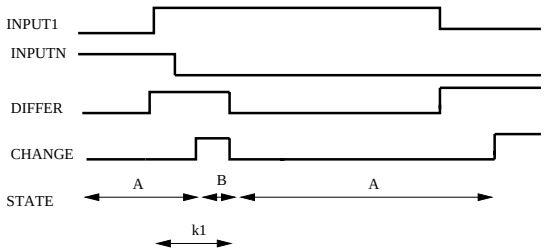

Figure 11: *Diagram using asymmetrical delays.*

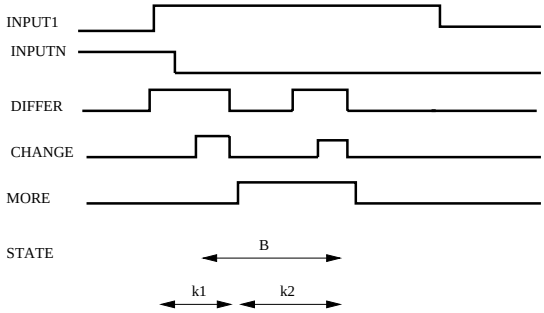

Figure 12: *Diagram of the MOC case.*

the inputs may change to a new state before the clock is generated to clock the flip-flop. Thus, the machine may enter a different state than it should enter. In addition, the speed of the machine is also slower due to this type of delay.

Case 2: Using asymmetrical delays the diagram is shown in Fig. 11. We can, therefore, minimize the problem mentioned above by using the asymmetrical delay elements. The speed of the circuit is now limited only by the chosen technology. For the MOC case the diagram is shown in Fig. 12. The signal MORE is high when the machine has not entered the final stable state.

7 TIMING ANALYSIS

The following notation will be used in this paper to evaluate the speed of the machine: $[D]$ the delay through the delay elements, $[d]$ the stray delays through the combinational logic, $[s]$ the set-up time for the register elements (flip-flops), $[f]$ the propagation delays through register elements, $[k1]$ the time interval for which several input signals may change, $[k2]$ the time interval for which input signals must remain stable.

ASYNCHRONOUS HUFFMAN MOORE MACHINE.

A MIC Huffman-Moore machine having a proper critical race-free assignment will, in general, still require delay elements for proper operation. The earliest that an input change can reach output logic is $k1 + dmax$ (**notation:** min = minimum, max = maximum, Dm = Delay element). Thus, the minimum valued for the delay element must be $Dmin \geq k1 + dmax - dmin$, or to be safe, $Dmin \geq k1 + dmax$. Hence, $k2$ is bound by $Dmin + dmin$ and $Dmax + dmax$.

For the SOC case: $k2 + dmin \geq dmax + (Dmax + dmax)$, hence $k2 \geq Dmax + 2dmax - dmin$ $(eq1)$. This is the period that inputs have to remain stable after the change.

In **the case of MOC,** we have another restriction. The time that each state changes is bound by $Dmin + dmin$ and $Dmax + dmax$. If n is the longest sequence of state transition in the machine to produce the output then $k2 + dmin \geq dmax + n(Dmax + dmax)$ or $k2 \geq nDmax + (n+1)dmax - dmin$ and the time between states: $k1 + k2 \geq k1 + n\ (Dmax + dmax) + (dmax - dmin)$.

Special case for Huffman-Moore machine: If the machine is in SOC mode and has no essential hazard, then $D = 0$. Thus $k2 \geq 2dmax - dmin$ $(eq2)$.

SELF-SYNCHRONIZED MACHINES.

For the machine built using this structure, the clock edge to the register elements (flip-flop2) must not arrive before the input changes have gone through the combinational logic, reached the state-variable flip-flops, and met the set-up time requirements. Thus, $Dmin \geq k1 + dmax + s$ and similarly,

$k2 + Dmin \geq Dmax + fmax + dmax + s$ $(eq3)$

from which we obtain

$k2 \geq fmax + dmax + s + (Dmax - Dmin)$ $(eq4)$

Thus the input changes are separated by the time interval:

$k1 + k2 \geq k1 + (fmax + dmax + s) + (Dmax - Dmin)$ $(eq5)$

For MOC case:

$k1 + k2 \geq k1 + n(fmax + dmax + s) + (Dmax - Dmin)$

By comparison between equations $(eq2)$ and $(eq5)$, the Huffman-Moore machine will always be faster if the machine operates in SOC and has no essential hazards. Otherwise, the combination circuit will dictate the speed of the circuit in the Huffman-Moore machines. The more complex the machine, the bigger the combination circuit due to the complicated state assignment to avoid races and hazards. This leads to larger $k1$. On the other hand, the state assignment in self-synchronized circuits can be arbitrary. Thus the combinational logic can be made much simpler. Consequently, the speed of the self-synchronized machine can be faster than that of the asynchronous Huffman-Moore machine, and this trade-off should be analyzed by CAD tools.

Self synchronized circuits extended to Unrestricted Input Change (UIC) case.

Almost all asynchronous designs assume that the machine will operate in the fundamental mode - once the input-state change is perceived by the machine, the machine will reach a final stable state before another input-state change is allowed. When the machine operates in UIC mode, the fundamental mode assumption is violated. Since the timing relationships between the inputs are not constrained, ambiguous input-state states will result. This may cause the machine to malfunction. As described in Kirkpatrick [11], the extension of the self-synchronized machines to the UIC case is straightforward. All we have to do is to add a transparent latch, such as 74F373, to the input signals. While the machine is in a stable state, the latch is enabled. Thus, the machine is ready to accept input changes. Once the machine detects new inputs via DIFFER going high, this input latch is disabled and is freezing the input state in the latch. Next, this input-state is processed and once the machine returns to the stable state, the input latch is again enabled to accept new input changes.

It should be noted that this UIC input latch will exhibit the metastable behavior due to the input changes not meeting the set-up and hold-time requirements for the latch. To compensate for this, an *additional delay* has to be added to k1 (normally four times the propagation delay of the latch). The general structure of the UIC self-synchronized machine would look, therefore, like in Fig. 13. The UIC_LE signal stands for UIC Latch Enable.

Thus, the speed of the circuit is:

For SOC:

$k1 + k2 \geq k1 + (5fmax + dmax + s) + (Dmax - Dmin)$

For MOC:

$k1 + k2 \geq k1 + n(5fmax + dmax + s) + (Dmax - Dmin)$

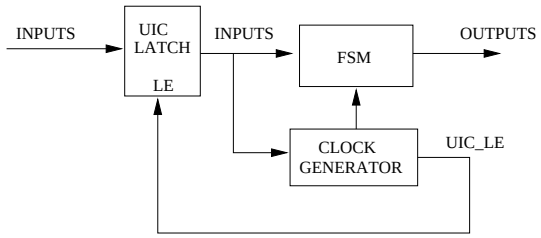

Figure 13: *UIC self-synchronized machine.*

STATE \ X1 X2	00	01	11	10	y1 y2
1	(1,0)	2,0	4,0	(1,0)	0 0
2	(2,0)	(2,0)	3,0	3,1	0 1
3	1,0	2,0	(3,1)	(3,1)	1 0
4	2,0	2,0	(4,0)	(4,0)	1 1

state, z

Figure 14: *Flow matrix by Unger.*

8 COMPARISON BETWEEN CIRCUITS.

In this section we will compare synchronous, asynchronous and self-synchronized circuits. To make the analysis using real data for one technology, in the following example we will assume most pessimistically the 74FXX technology, and also assume that each FXX gate delay is 3ns, and 10ns for a minimum and a maximum, respectively ($dmin = 3$, $dmax = 10$). For the PAL 16L8B and 16R4B, the set-up time is 15ns, the clock to output time is 12ns, and the propagation delay time is 15ns.

Example: The Crumb Road Problem. The problem is the design of a sequential machine to control the traffic at the intersection of Crumb Road and Route 1. (For a complete description of the problem, see Unger, 1969). Unger derived the following flow matrix (Fig. 14). The corresponding circuit is:

$Z = x_1\ \overline{y_1}\ y_2 + x_1\ y_1\ \overline{y_2}$

$Y_1 = x_1\ x_2\ \overline{y_1}\ \overline{y_2} + x_1\ \overline{y_1}\ y_2 + x_1\ y_1$

$Y_2 = \overline{x_1}\ x_2 + y_1\ y_2 + \overline{x_1}\ y_2 + x_1\ x_2\ \overline{y_1}\ \overline{y_2}$

Asynchronous machine. The speed (total propagation time) of the asynchronous machine assuming PAL16L8B is 15ns, which corresponds to 66.6 Mhz.

Synchronous machine. Using PAL 16R4B, the synchronous machine version of this example has the maximum clock rate $= Tsetup + Tclock - to - output$ is equal to 15ns + 12ns = 27ns. The maximum speed is, therefore, 27ns or 37 Mhz.

Self-synchronized machine. The circuit realization for the above problem is shown in Fig. 15. First, let us understand the operation of the circuit. Assume on power on, everything is stable (we intentionally ignore the additional circuitry to bring the circuit to a known state upon power-up or reset condition). In this state, DIFFER, CHANGE are low and CLOCK is high, the latch $U1$ is disabled. The circuit is ready to accept any input changes. If any or both inputs $x1$ and $x2$ change, the changes will go to the PAL 16R4B and also will go through $U2$ to cause DIFFER to go to high. After the delay, CHANGE will go high to enable the latch $U1$. CLOCK then goes low. Next, the input will go through $U1$, $U2$ to turn off DIFFER, then through the delay to turn off CHANGE. Finally, the latch $U1$ is shut off and CLOCK goes high to clock the PAL 16R4B. Now, the state machine is ready for another input change.

Next we have to determine what is the delay in the delay line of the circuit, before we can calculate the circuit's speed. The worst case timing analysis is as follows. There are two paths in this circuit. Path 1, P1, is from the inputs to the PAL 16R4B (Fig. 15). The other path, P2, is from the inputs through the clock generator. The only constraint is that the input change has to arrive at the PAL16R4B at least within the minimum set-up time, 12ns, before the CLOCK is generated, going from low to high. Hence, the minimum delay through the clock generator block must be equal or greater than the set-up time requirement of the PAL. We have, therefore, the following inequality:

$tU2min + tDmin + tU1min + tU2min + tDmin + tU3min \geq tsetup$

$3 + tDmin + 3 + 3 + tDmin + 3 \geq 15$

$tDmin \geq 1.5\ ns.$

(We can use a non-inverting buffer as the delay in this case). Suppose, we use the F08 AND-gate as the delay in this example, then $tDmin = 3$ns. Then the speed of the circuit is:

$Speed = 2tDmin + 2tU2min + tU1min + tU3min = 2*3 + 2*3 + 3 + 3$

$Speed = 18\ ns\ or\ 55.5\ Mhz$

We can see, therefore, that the self-synchronized

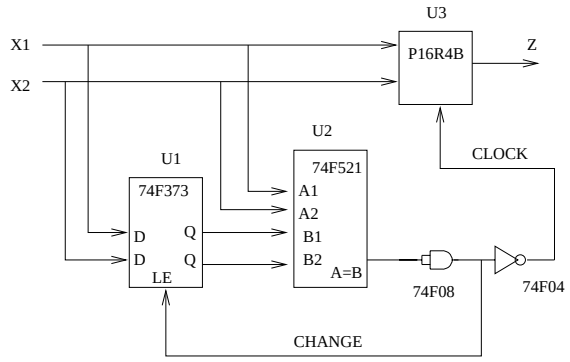

Figure 15: *Self-synchronized circuit for Crumb Road problem.*

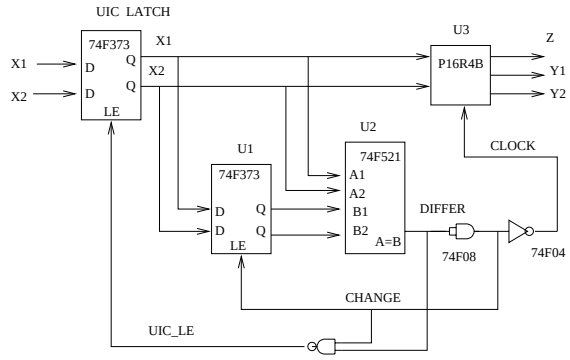

Figure 16: *UIC case for Crumb Road problem.*

circuit in this implementation is faster than the synchronous circuit by about 33%.

- Asynchronous Huffman-Moore machine : = 66.6 Mhz,
- Self-Synchronized machine : = 55.5 Mhz.
- Synchronous machine : = 37.0 Mhz.

For the UIC case: The UIC latch is added to the self-synchronized circuit and the synchronizer has to be added to the synchronous machine. The speed difference will be less apparent because the self-synchronized circuit will be slower by the extra UIC latch plus the compensation for metastability. On the other hand, the synchronous machine has to wait for an extra clock or two to synchronize the inputs. With the above example, the realization for the UIC case is as in Fig. 16.

As mentioned above, the UIC latch may exhibit the metastable condition. We allow 4 Tpd to allow the latch to recover. Thus the speed is

$Speed = 2tDmin+2tU2min+tU1min+tU3min+TUIClatch = 2*3+2*3+3+3+40.$

$Speed = 58\ ns\ or\ 17.24\ Mhz.$

For the synchronous machine, the metastable problem also has to be taken into account. Hence,

$Speed = 27+40 = 67\ ns\ or\ 14.9\ Mhz.$

9 SELF-SYNCHRONIZED CPLDS AND FPGAS.

It is an interesting issue whether the asynchronous designs, when mature, will become only a domain of VLSI processor design market, or they will affect also FPGA and EPLD markets. Above-introduced Clock Generator circuit can be used with EPLD devices for direct implementation of high-level specification of self-synchronized circuits. It was shown above by us and tested experimentally that the self-synchronized circuits can be designed using commercially available PALs or EPLDs and TTL parts. We demonstrated also that the self-synchronized circuits are faster than the synchronous circuits when implemented with PALs.

Finally, although we do not elaborate on it because of the lack of space, it should be obvious from the example, that it is relatively easy to convert standard synchronous optimization tools for self-synchronized circuits. The biggest advantage here is that one will be able to use standard methods of state assignments and logic reduction of synchronous machines. **Thus traditional methods with minimum modification are preserved while the speed can be improved.** Thus, any PAL/PLD/FPGA standard popular development system can be used by the designer, and the design slightly modified, or the tool can be easily modified by its developers. Although asynchronous circuit is still faster, only time-critical sub-machines could be realized as asynchronous if other are too difficult or do not lead to improvements. The other machines can be self-synchronized, as presented, and still lead to total power reduction and speed improvements of entire systems.

When using standard PALs or EPLDs, there are still a lot of extra components, 5 extra chips, besides the PAL needed to implement a self-synchronized circuit. Therefore we propose to **design a front-end VLSI chip,** called the *CLOCK GENERATOR,* so that one can build the self-synchronized circuits with only three components: Clock Generator, PALs (EPLDs, FPGAs), and a resistor with a capacitor. No PLD device is, to our knowl-

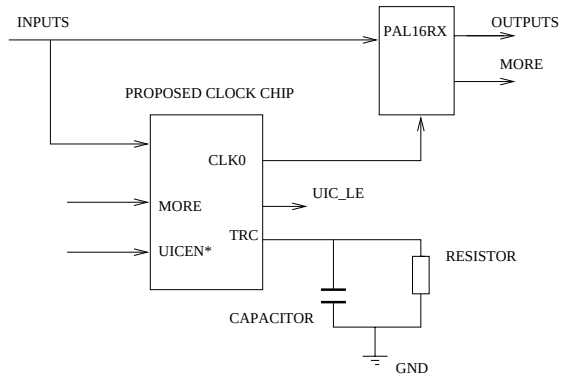

Figure 17: *Self-synchronized state machine with proposed chip and PAL.*

edge, currently produced by companies that would help to decrease the chip count when implementing self-timed or self-synchronized circuits, so our idea should become of interest to the EPLD/FPGA industry. The pair of resistor and capacitor will set the time delay. The asymmetrical delay element and the UIC mode, if selected, will be taken by the clock generator chip. This chip is fairly small and simple to design. The complete schematic of the self-synchronized state machine with the proposed chip and PAL is shown in Fig. 17.

Even higher integration can be achieved when several complete self-synchronized machines are put into a single VLSI chip. Thus CPLD and FPGA chips can also be fabricated that will include the proposed CLOCK generator, a (partitioned) PAL or distributed CLBs, together with programmable delays, in a single enclosure. More research is needed to define the best structures for such self-synchronized arrays. Perhaps they should be product-oriented rather than general purpose.

References

[1] U.C. Berkeley. "Project Goals: Ultra-Low Power DSP Processing based on Heterogeneous Co-Processors," http://infopad.-eecs.berkeley.edu/research/reconfigurable-/texts/overview/overview.html.

[2] D.S. Bormann, and P.Y.K. Cheung, "Asynchronous Wrapper for Heterogeneous Systems," *Proc. ICCD'97.*

[3] G. Borriello, "Synthesis of mixed synchronous/asynchronous control logic," *Proc. ISCAS'89,* pp. 762-765.

[4] J.G. Bredeson, and P.T. Hulina, "Generation of a clock pulse for asynchronous sequential machines to eliminate critical races," *IEEE TC,* Vol. C-20, pp. 225 - 226, Feb. 1971.

[5] R. Breuninger, and W. Thompson, "Metastability Evaluation of Logic Technologies," *Texas Instruments Inc.,* 1986.

[6] H.Y.H. Chuang, and S. Das, "Synthesis of multiple-input change asynchronous machines using controlled excitation and flip-flops," *IEEE TC,* Vol. C-22, pp. 1103 - 1109, Dec. 1974

[7] F.A. Franklin, and D.F. Wann, "Asynchronous and clocked control structures for VLSI based interconnection networks," *Proc. Symp. on Comp. Arch.,* IEEE, Apr. 1982, Austin, Texas, pp. 50-59.

[8] L.A. Hollaar, "Direct Implementation of Asynchronous Control Units," *IEEE TC,* Vol. C-31, No. 12., pp. 1133-1141, Dec., 1982.

[9] J.L. Huertas, and J.I. Acha, "Self-synchronization of asynchronous sequential circuits employing a general clock function," *IEEE TC,* pp. 297 - 300, March 1976.

[10] D.C. Kirkpatrick, and V.M. Powers, "An asynchronous design style to achieve ultimate operating speed," *5th Ann. Intern. Phoenix Conf. on Comp. and Comm. PCCC'86,* pp. 662 - 673, 1986.

[11] D.C. Kirkpatrick, "Design of Self-Synchronized Asynchronous Sequential State Machines Using Asymmetrical Delay Elements," *Ph.D. Diss., OSU,* 1985.

[12] C. Molnar, T. Fang, and F. Rosenberger, "Synthesis of Delay-Insensitive Modules," *Proc. Chapel Hill Conf. VLSI,* May 1985.

[13] C.A. Rey, and J. Vaucher, "Self-synchronized asynchronous sequential machines," *IEEE TC,* Vol. C-23, pp. 1306-1311, Dec. 1974.

[14] S.H. Unger, "Self-synchronizing circuits and nonfundamental-mode operation," *IEEE TC,* Vol. C-26, pp. 278 - 281, March 1977.

[15] S.H. Unger, "The Essence of Logic Circuits," *Prentice-Hall, Inc.*, 1989.

[16] S. Whitaker, and G.K. Maki, "Self Synchronized Asynchronous Sequential Pass Transistor Circuits," *IEEE TC*, Vol. 41, Oct. 1992, pp. 1344-1348.

[17] O. Yenersoy, "Synthesis of Asynchronous Machines Using Mixed-Operation Mode," *IEEE TC*, Vol. C-26, pp. 325-329, Apr. 1979.