2006-672: ASYNCHRONOUS FINITE STATE MACHINE DESIGN: A LOST ART?

Christopher Carroll, University of Minnesota-Duluth
Christopher R. Carroll earned his academic degrees from Georgia Tech and from Caltech. He is Director of Undergraduate Engineering in the College of Science and Engineering at the University of Minnesota Duluth and serves in the department of Electrical and Computer Engineering. His interests include special-purpose digital systems, VLSI, and microprocessor applications, especially in educational environments.

Asynchronous Finite State Machine Design: A Lost Art?

Abstract

As taught in most introductory digital circuit classes, design of sequential digital circuits is limited to a very strict set of restrictions, usually called "synchronous" finite state machine design. In synchronous design, there is *one* special signal called the "clock" which controls the timing of all state transitions, and *all* clock inputs on *all* sequential components in the digital system (flip-flops or other more complex components) *must* connect directly to that *one* clock signal in the system, without exception. Furthermore, *all* variables in the system are restricted to change *only* on the clock transition that causes state changes in the sequential components of the system. There is good reason to impose these restrictions in an introductory class. Working under the "synchronous" umbrella protects the designer from many timing problems that can occur in systems if these restrictions are not followed. However, examples abound where synchronous restrictions must be violated. For example, when a manually controlled switch input enters an otherwise synchronous system, the manual switch variable can change value at times unrelated to the clock transition, violating synchronous restrictions and causing headaches.

In fact, synchronous design techniques, and other related design regimens, are special cases of the more general design approach called "asynchronous" finite state machine design. In asynchronous design, there is no special clock signal to cause state changes. Instead, the state machine reacts to changes on the input variables from the environment. Flip-flops, for example, actually are not elementary building blocks of digital circuits as is taught in introductory courses. Flip-flops themselves are designed as asynchronous circuits. The memory in sequential circuits arises from the feedback present in asynchronous design to implement states in the state machine operation. In asynchronous sequential circuit design, there are no artificial restrictions on circuit behavior. As a result, however, there are many more concerns that must be addressed by the designer in order to ensure correct circuit operation, so asynchronous design is usually omitted from discussion in introductory digital courses.

This paper addresses some issues related to asynchronous finite state machine design, and includes some important examples of specific asynchronous circuits that have central significance in digital design. Techniques are included here to incorporate asynchronous design into lab experiments for advanced digital design courses. Without an understanding of the issues presented here, digital designers must work in a design environment that unnecessarily limits what they can do.

Background

In digital circuit design, sequential circuits, or finite state machines, are circuits containing some sort of memory, as opposed to combinational circuits that contain no memory. In a combinational circuit, the outputs of the circuit depend only on what the inputs are right now. In a sequential circuit, the outputs depend not only on what the inputs are now, but also on the past history of the input values, which implies that the circuit includes memory to record that history. Most commonly, memory is implemented with circuit elements called "flip-flops" that are "clocked" at specific times to update their contents. The clocking scheme used in a circuit design can be very strictly specified, as in synchronous design, where *every* flip-flop or sequential component's clock signal *must* connect to a *single* system clock signal so that all state in the system changes at the same time. The clocking scheme can be more relaxed, as in multi-phase clocking or a system of several clocks all derived from a common master clock, which allows state changes at controllable times in the system operation. The extreme case is the case where there are no clock signals to tell the system when to change state. This is the asynchronous finite state machine case, and is the most general approach to sequential circuit design.

Each relaxation of constraints on the clocking scheme used in a sequential circuit creates additional concerns that must be addressed by the circuit designer, generally regarding detailed timing of state transitions in the circuit. The strictest system, synchronous design, frees the designer from concern about all but the most basic timing details. The most general system, asynchronous design, requires that the designer face many potential timing flaws in the circuit. Consequently, introductory digital design courses generally focus on synchronous design. They sometimes allow slight relaxation of the synchronous constraints, but rarely address the design freedom and concerns of general asynchronous design. Textbooks used in today's digital design courses rarely if ever even mention asynchronous design. The most recent textbook of which this author is aware that carefully addresses asynchronous sequential circuit design is the classic text by Kohavi[1], copyrighted in 1978.

The pity is that asynchronous design is the foundation upon which all other sequential design techniques rest. Without a thorough understanding of asynchronous design techniques, the reasons for constraints imposed by less thorough techniques remain a mystery. Flip-flops are NOT fundamental circuit components, as is taught in introductory courses. Flip-flops are designed from more elementary components using asynchronous techniques. The purpose for this paper is to remind digital circuit designers that to understand fully the principles they use in their designs, they must apply knowledge of asynchronous sequential circuit design.

Basics

Sequential circuits, or finite state machines, are designed around the templates shown on the next page. Figure 1 shows the template for synchronous circuits, where the memory in the circuits is implemented with flip-flops that are clocked by the special system clock signal. Figure 2 shows the template for asynchronous circuits. The memory in such circuits is composed of a simple delay. This delay must be non-zero, but is otherwise unspecified in duration, and usually is implemented with the gate delays inherent in the combinational logic portion of the circuit, so there is generally no separate hardware that implements the delay. The heavy arrows in the templates indicate bundles of signals grouped into the four labeled groups. Asynchronous circuits are characterized by the presence of these feedback paths in the design where some combinational logic produces an output that depends on itself. This feedback, and the associated delay inherent in the logic, implements the memory of the asynchronous finite state machine.

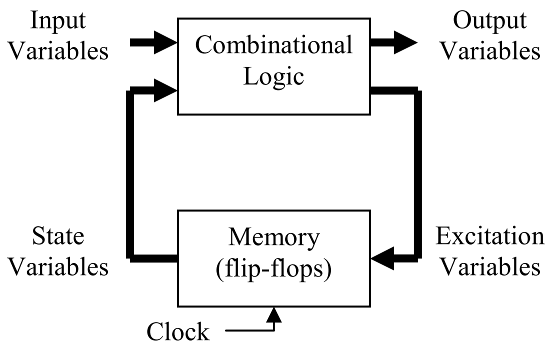

Figure 1. Synchronous template

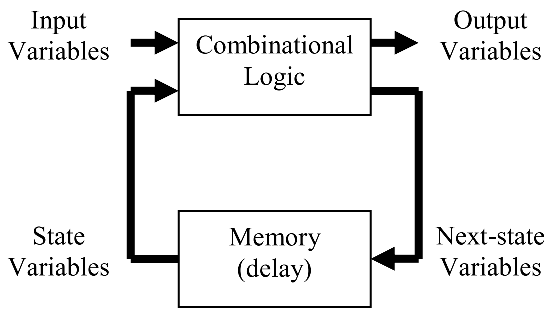

Figure 2. Asynchronous template

Figure 3 below shows a classic SR latch, the most fundamental memory circuit studied in introductory digital circuit courses. Figure 4 shows exactly the same circuit, but drawn differently to emphasize the single feedback path, which holds the one state variable in the circuit. The circuit remembers which of the two input variables, S or R, was most recently a 1, by recording on the output variable, Q, a 1 if it was S or a 0 if it was R. By realizing that this SR latch, the most fundamental memory circuit in any static memory device, is actually an asynchronous finite state machine, one realizes the fundamental nature of this topic.

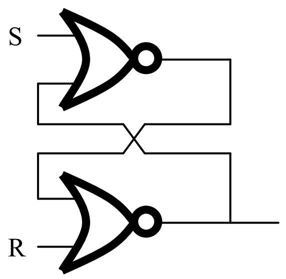

Figure 3. Classic SR latch

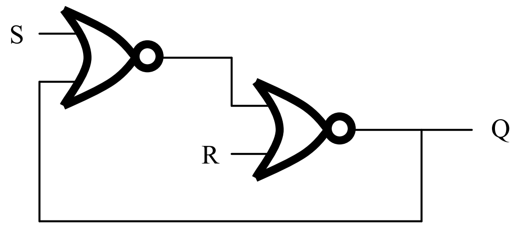

Figure 4. SR latch re-drawn

In the SR latch pictured above, the “memory” in the circuit is actually implemented in the delay inherent in the gates themselves. If the gates somehow had zero delay, this circuit could fail.

Hazards

One interesting characteristic of digital circuits that must be addressed in asynchronous designs but that can be ignored in synchronous designs is hazards. A hazard is present in a circuit if a signal in the circuit changes unexpectedly in response to a change in a single input variable. This unexpected change can be either a brief “glitch” to the other value when a signal is supposed to stay constant during the transition (a so-called static hazard) or multiple changes on a value when the signal is supposed to change just once in response to the transition (a so-called dynamic hazard). Dynamic hazards arise because of the presence of static hazards, so removing the static hazards in a circuit also removes any dynamic hazards. An example of a circuit where hazards cause trouble is shown in Figure 5. This is an elementary gated transparent D latch, where the Q output follows the D input as long as the control input, C, is 1, but latches the current value on D when C becomes 0. Looking at the circuit in Figure 5, clearly when C = 1, the bottom AND gate

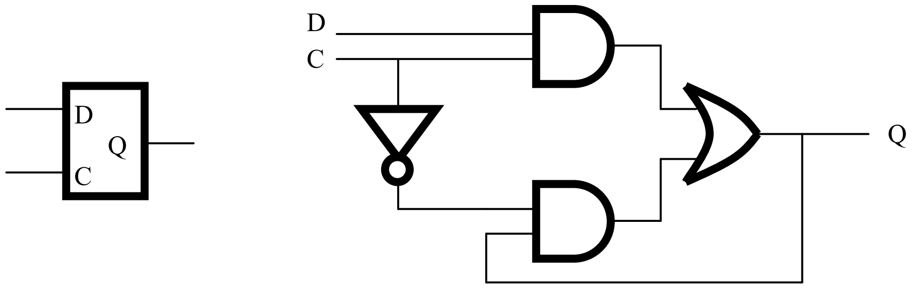

Figure 5. Gated transparent D latch, improperly designed

output is 0, and the top AND gate passes through the value on the D input, so the Q output is equal to the D input. When C = 0, the D input is blocked by the top AND gate, and whatever value is on the Q output is re-circulated so the Q output remains at its current value regardless of D. This sounds right. However, examine the circuit if initially D=C=Q=1 and then C changes from 1 to 0. Because of the delay in the inverter, the top AND gate output may become 0 before the bottom AND gate output becomes 1, which means that Q goes to 0, which then prevents the bottom AND gate output from going to 1 as it should. The circuit erroneously changes state from Q = 1 to Q = 0 because of a hazard in the circuit. Hazards must be avoided.

Fortunately, static hazards (and thus dynamic hazards) can always be avoided, but sometimes at the cost of slightly more complicated circuits. In this case, Figure 6 shows a modified design for the gated transparent D latch that avoids the hazard, and operates correctly. As can be seen in

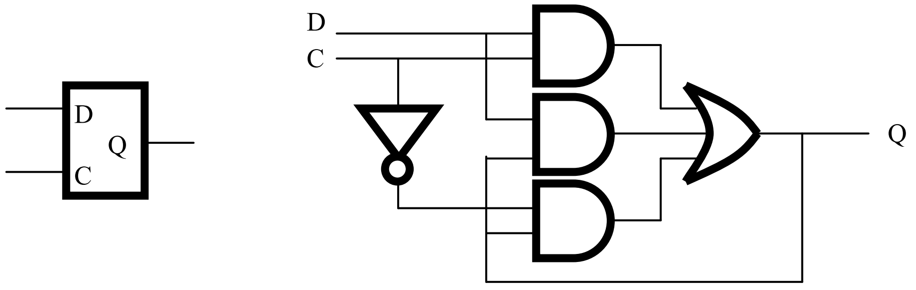

Figure 6. Gated transparent D latch, hazard removed

Figure 6, the circuit requires an additional gate, but is now hazard free and will operate correctly. Hazards are unimportant in synchronous designs, because as long as variables have settled by the time the next clock transition occurs, the circuit operates correctly. In asynchronous design, there is no clock to say when to look, so variable values must be valid at all times, including during transitions.

Races

A more challenging problem faced by asynchronous finite state machine designers is the problem of races. A “race” occurs whenever two or more state variables must change during a transition from one state to another. The state variables are the signals that feed back through the delay in the template of Figure 2. If only two states are present in a design, thus requiring only one state variable to distinguish them, no races are possible. However, in designs with more than two states, thus requiring more than one state variable, races are a concern. The difficulty arises because there is no control on the amount of delay in the feedback path. If the designer could guarantee that if, say, two state variables are changing during a transition, they would change at precisely the same time, races would not be an issue. However, because the feedback delay experienced by one state variable is guaranteed to be different from that experienced by another state variable, one of the variables will change before the other, taking the state machine to an unanticipated intermediate state that may well lead to improper machine behavior.

At a given time, an asynchronous state machine is either stable, or unstable. The machine is stable if the next state produced by the combinational logic is the same as the present state represented by the value of the state variables. In other words, there are no transitions in the process of propagating through the delay. If any next-state variable is different from its corresponding state variable, the machine is unstable, meaning that at least one variable transition is propagating through the delay that will eventually result in a state change. Normal operation of asynchronous state machines begins in a stable state. Then some input variable changes, possibly changing some next-state variable and putting the machine in an unstable state. When that changing value emerges from the delay and changes a state variable, ideally the machine is once again stable.

In order to avoid problems caused by races, the designer must carefully assign values of state variables to represent the states of the system in such a way that two states which transition between each other differ in state assignment on exactly one state variable. Thus, to change from one state to another, only one state variable must change, and there is no race. Finding such a state assignment is tricky, and is not always possible. In the synchronous design case, state assignment is arbitrary, and any choice will lead to a functioning solution. In the asynchronous design case, state assignment is crucial to avoid races.

When a designer finds a situation in which no state assignment is possible to avoid races, there are options to try. Sometimes a race is a “non-critical” race in which the final state reached is correct regardless of the order in which state variables change. Non-critical races can be tolerated, but races in which the final state depends on the order in which variables change, or “critical races,” must be avoided to guarantee proper machine operation. Another option is the introduction of “cycles,” or sequences of unstable states through which transitions pass, in order to avoid a troublesome transition that leads to a critical race. These solutions require insight and creativity on the part of the designer, rather than the mechanical application of some design process as taught in introductory digital design classes. The good news is that a solution that avoids races is always possible, although it may lead to an increase in circuit complexity. The ability to handle asynchronous sequential circuit design is one quality that distinguishes good digital circuit designers from run-of-the-mill designers.

More!

The troubles faced by asynchronous finite state machine designers do not stop here. There is another type of hazard called an "essential hazard" that sometimes occurs in these designs. Essential hazards arise because of delays in the combinational logic that swamp the delays in the feedback part of the state machine, leading to improper operation. Finding and fixing essential hazards, unfortunately, requires detailed analysis of worst-case delay paths through the combinational logic portion of the circuitry, and sometimes requires adding delay padding to the feedback path of the circuit to ensure that the feedback delay overcomes other delays in the circuit.

Asynchronous state machines are designed under "fundamental mode" restrictions, which specify that only one input variable can change at a time, and only when the machine is stable. No other input variable may change until the state machine finishes its response to the first change and reaches a new stable state. This restriction is the origin of the "set-up" and "hold" time specifications always seen on data sheets for flip-flops or other sequential devices. Sometimes the fundamental mode restrictions can be relaxed a bit, but if they are not followed, unpredictable or non-deterministic behavior of the state machine can result.

Incorporating Asynchronous Design in the Curriculum

In the course "Digital Computer Circuits" at UMD, a second course in digital circuits, the last three weeks of the semester are devoted to asynchronous sequential circuit design, and students perform two lab exercises in which they design and build asynchronous finite state machines. These students already have experience with synchronous design, and also with various departures from the synchronous limitations, but the asynchronous design material relaxes those limitations even further to allow students to understand the fundamental nature of digital sequential design and to see the origins of the restrictions imposed by more rigid design strategies. Students appreciate the opportunity to see how information they have learned from previous courses fits into the "big picture" of digital design.

The first lab exercise performed by students in the area of asynchronous design involves designing and building a circuit that needs just two states, and therefore just one state variable, or feedback variable. This eliminates the need to be concerned about races in the circuit, and students can focus instead on avoiding hazards and minimizing circuit complexity. The circuit typically uses two debounced switch inputs, and must respond to some specific sequence of switch closures by activating an output variable appropriately.

The second asynchronous lab exercise involves designing and building a circuit that must use more than two states, and thus has more than one state variable, or feedback variable, so that races on the state variables during transitions must be avoided. For example, students have designed circuits that capture data from one input variable any time a second input variable changes, either 0 to 1 or 1 to 0, comparable to a D flip-flop that triggers on *both* edges of the "clock" input rather than just the rising *or* falling edge. Labs must be designed carefully to avoid assignments that involve essential hazards because debugging a circuit with essential hazards requires detailed timing analysis and simulation beyond the scope of the class.

Asynchronous finite state machine design, once understood, is accomplished with very simple circuits involving only individual gates, with no sequential components or higher levels of integration involved. This is considerably simpler than the complex circuits students have been building earlier in the semester, such as circuits implementing a multiplier, stack, FIFO, and other structures. Students are relieved to find that asynchronous circuits, although tricky to understand conceptually, are easily implemented on their breadboards. These lab exercises are a comfortable, relaxing way to end the semester in the lab for this class.

Summary

Designers of asynchronous finite state machines must consider many characteristics of their designs that can be ignored by designers working under more restrictive limitations. Synchronous design, as taught in introductory classes, relieves the designer from these concerns, but limits the kinds of results that can be achieved. Only by understanding the fundamental, asynchronous design process can digital designers fully realize the flexibility of digital circuits. The asynchronous state machine design process is a crucial foundation upon which digital design must rest.

References

1. Kohavi, Zvi, Swithcing and Finite Automata Theory, 2nd edition, McGraw-Hill, Inc., 1978.