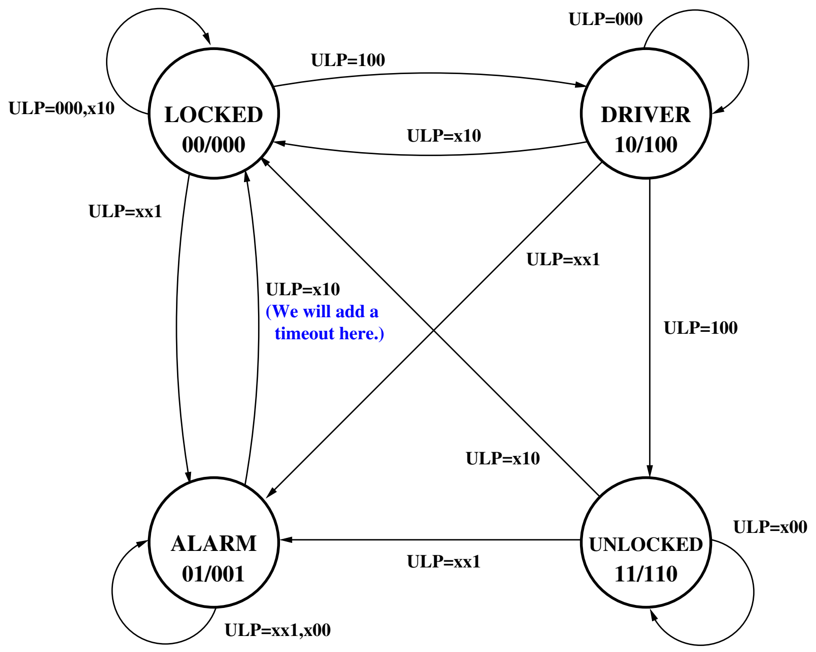
ECE120: Introduction to Computer Engineering

Notes Set 3.4 Extending Keyless Entry with a Timeout

This set of notes builds on the keyless entry control FSM that we designed earlier. In particular, we use a counter to make the alarm time out, turning itself off after a fixed amount of time. The goal of this extension is to illustrate how we can make use of components such as registers and counters as building blocks for our FSMs without fully expanding the design to explicitly illustrate all possible states.

To begin, let's review the FSM that we designed earlier for keyless entry. The state transition diagram for our design is replicated to the right. The four states are labeled with state bits and output bits, S_1S_0/DRA, where D indicates that the driver's door should be unlocked, R indicates that the rest of the doors should be unlocked, and A indicates that the alarm should be on. Transition arcs in the diagram are labeled with concise versions of the inputs ULP (using don't cares), where U represents an unlock button, L represents a lock button, and P represents a panic button.

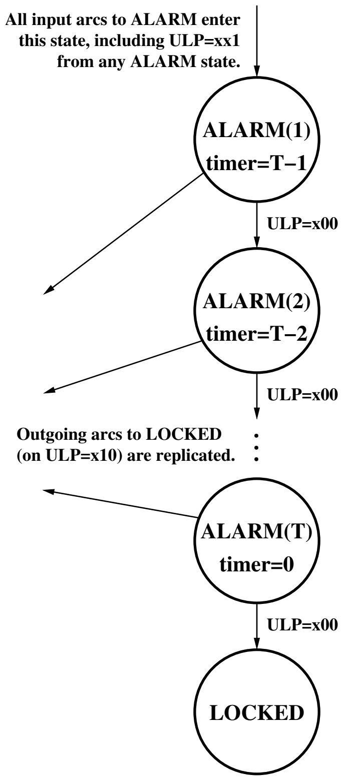


In this design, once a user presses the panic button P, the alarm sounds until the user presses the lock button L to turn it off. Instead of sounding the alarm indefinitely, we might want to turn the alarm off after a fixed amount of time. In other words, after the system has been in the ALARM state for, say, thirty or sixty seconds, we might want to move back to the LOCKED state even if the user has not pushed the lock button. The blue annotation in the diagram indicates the arc that we must adjust. But thirty or sixty seconds is a large number of clock cycles, and our FSM must keep track of the time. Do we need to draw all of the states?

Instead of following the design process that we outlined earlier, let's think about how we can modify our existing design to incorporate the new functionality. In order to keep track of time, we use a binary counter. Let's say that we want our timeout to be T cycles. When we enter the alarm state, we want to set the counter's value to $T-1$, then let the counter count down until it reaches 0, at which point a timeout occurs. To load the initial value, our counter should have a parallel load capability that sets the counter value when input $LD = 1$. When $LD = 0$, the counter counts down. The counter also has an output Z that indicates that the counter's value is currently zero, which we can use to indicate a timeout on the alarm. You should be able to build such a counter based on what you have learned earlier in the class. Here, we will assume that we can just make use of it.

How many bits do we need in our counter? The answer depends on T. If we add the counter to our design, the state of the counter is technically part of the state of our FSM, but we can treat it somewhat abstractly. For example, we only plan to make use of the counter value in the ALARM state, so we ignore the counter bits in the three other states. In other words, $S_1S_0 = 10$ means that the system is in the LOCKED state regardless of the counter's value.

We expand the ALARM state into T separate states based on the value of the counter. As shown to the right, we name the states ALARM(1) through ALARM(T). All of these alarm states use $S_1S_0 = 01$, but they can be differentiated using a "timer" (the counter value).


All input arcs to ALARM enter this state, including ULP=xx1 from any ALARM state.
ALARM(1) timer=T−1
ULP=x00
ALARM(2) timer=T−2
ULP=x00
Outgoing arcs to LOCKED (on ULP=x10) are replicated.
ALARM(T) timer=0
ULP=x00
LOCKED

We need to make design decisions about how the arcs entering and leaving the ALARM state in our original design should be used once we have incorporated the timeout. As a first step, we decide that all arcs entering ALARM from other states now enter ALARM(1). Similarly, if the user presses the panic button P in any of the ALARM(t) states, the system returns to ALARM(1). Effectively, pressing the panic button resets the timer.

The only arc leaving the ALARM state goes to the LOCKED state on $ULP = x10$. We replicate this arc for all ALARM(t) states: the user can push the lock button at any time to silence the alarm.

Finally, the self-loop back to the ALARM state on $ULP = x00$ becomes the countdown arcs in our expanded states, taking ALARM(t) to ALARM(t+1), and ALARM(T) to LOCKED.

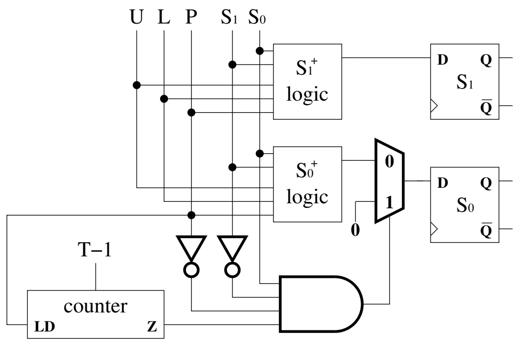
Now that we have a complete specification for the extended design, we can implement it. We want to reuse our original design as much as possible, but we have three new features that must be considered. First, when we enter the ALARM(1) state, we need to set the counter value to $T - 1$. Second, we need the counter value to count downward while in the ALARM state. Finally, we need to move back to the LOCKED state when a timeout occurs—that is, when the counter reaches zero.

The first problem is fairly easy. Our counter supports parallel load, and the only value that we need to load is $T - 1$, so we apply the constant bit pattern for $T - 1$ to the load inputs and raise the LD input whenever we enter the ALARM(1) state. In our original design, we chose to enter the ALARM state whenever the user pressed P, regardless of the other buttons. Hence we can connect P directly to our counter's LD input.

The second problem is handled by the counter's countdown functionality. In the ALARM(t) states, the counter will count down each cycle, moving the system from ALARM(t) to ALARM(t+1).

The last problem is slightly trickier, since we need to change S_1S_0. Notice that $S_1S_0 = 01$ for the ALARM state and $S_1S_0 = 00$ for the LOCKED state. Thus, we need only force S_0 to 0 when a timeout occurs. We can use a single 2-to-1 multiplexer for this purpose. The "0" input of the mux comes from the original S_0^+ logic, and the "1" input is a constant 0. All other state logic remains unchanged. When does a timeout occur? First, we must be in the ALARM(T) state, so $S_1S_0 = 01$ and the counter's Z output is raised. Second, the input combination must be $ULP = xx0$—notice that both $ULP = x00$ and $ULP = x10$ return to LOCKED from ALARM(T). A single, four-input AND gate thus suffices to obtain the timeout signal, $\bar{S_1}S_0Z\bar{P}$, which we connect to the select input of the mux between the S_0^+ logic and the S_0 flip-flop.

The extension thus requires only a counter, a mux, and a gate, as shown below.


U L P S1 S0
S1+ logic
D Q S1 Q̄
S0+ logic
0 1
0
D Q S0 Q̄
T−1
counter
LD Z