USE a binary counter (counts down, let's say) with LD input, parallel load (when $\text{ULP}=1$; $\text{ULP}=0$ does count down), and output $z=1$ when counter reaches zero.

all inputs to ALARM
expand ALARM as follows...

T is timeout length in cycles

all input arcs go to this state, including ULP = x\times1 from any ALARM state (but not ULP = *x00)

why reset timer when panic held? Seems like the right decision, but that’s all it is—a decision

timer bits only relevant in ALARM states

timers & output signals timeout—we’ll use it

note: All timer states will use $S_{i\rightarrow o} = 01$

special case

solve as before, but add

1. set timer to $T$ when entering ALARM1
2. move to LOCKED on timeout

Let’s not even bother to re-solve K-maps!

Just use a mux for 2!
ALARM is $S_S^0 = 0$
LOCKED is $S_S^0 = 0$

```
force $S_S^0$ to 0 when

1. in ALARM state ($S_S^0 = 0$)
2. $ULP = \times 00$
3. timeout occurred ($Z = 1$)

$\Rightarrow$ when $S_S^0 \bar{LP} Z$
```
another keyless entry extension
the farmer's dilemma
memory & decoders

these are examples of using abstraction to build FSMs

we trade a bit of optimization potential for simple & clean, & easy-to-build correctly designs...

HW #9 in, solutions & HW #10 out

Note: lab problem—compare with your group!
3 buttons

- transmit as sequence of 0s & 1s
- decode to "buttons" at receiver

- let's "clean up" at transmitter (transmission costs energy!): send only "lock," "unlock," or "panic" (not all combos)

Let students assign to 2-bit space.

What happens if we receive a bit incorrectly?

See if a different button? Ops.

What if we send 3 bits instead?
Can you avoid this problem?
Show 2-bit spacing, talk about Hamming distance

Use original diagram and add odd parity

Using a parity bit gives a Hamming distance 2 "code" (even or odd)

But 2 bit errors still break it.

Larger Hamming distances allow more bits of error to be detected, or you can do some correction.

3 bits for 1 bit of data? (gets better with longer messages)

Used ubiquitously in digital systems today; maybe more at end of our course.

We'll use 100 unlock
010 lock
001 panic

Another concern we'll address: back-to-back transmissions

Errors in

100 001

I lock? each command?
Let's say that our (analog) receiver gives us a signal $R=1$ when we start seeing bits coming in, and a bit $B$ every cycle starting in the $R=1$ cycle.

We need to produce ULP (input to keyless entry) in exactly one cycle in response to bit patterns shown earlier.

Let's think about components:
- Shift register to hold received bits
- Counter to manage timeout/delay (can reuse to count $A$ bits shifted)

High-level states

- **WAIT** for first bit ($R=1$)
- **SHIFT** up to 3 bits
- **EXECUTE** if command is valid, output ULP
- **DELAY** for $T$ cycles

Counter - binary countdown, parallel load based on $L$ & data inputs, and $Z$ output to indicate reaching 0

Shift reg - 3 bits, left shift always ($B$ goes in) calls bit $R_2 R_1 R_0$
handles mapping from EXECUTE state \((E=1)\) and bit pattern received \((R)\) to ULP outputs to keyless entry FSM

Let's write one output \((U)\):
- \(E\) must be 1
- \(R\) must be 100
- answer? \(ER, R, R_0\)

Other outputs are similar

Now let's build the FSM:
- inputs \(R, S\)
- outputs \(L0, S, E\)
STATE TRANSITIONS

\[
\begin{align*}
S_i & \rightarrow S_0 \\
S_0 & \rightarrow S_1 \\
S_1 & \rightarrow \text{Exit}
\end{align*}
\]

STATE VECTORS

\[
\begin{align*}
S_0 & = (0, 0, 0, 0) \\
S_1 & = (0, 1, 1, 0) \\
S_2 & = (1, 1, 1, 1) \\
S_3 & = (1, 0, 0, 0)
\end{align*}
\]

STATE EQUATIONS

\[
\begin{align*}
S_i^{+} & = S_0 \oplus S_1 \\
S_0^{+} & = S_1 \oplus S_2 \\
S_1^{+} & = S_2 \oplus S_3 \\
S_2^{+} & = \overline{S_1} \cdot S_0 + \overline{S_1} \cdot S_3
\end{align*}
\]