

University of Illinois at Urbana-Champaign  
Dept. of Electrical and Computer Engineering

## ECE 120: Introduction to Computing

### Extending Keyless Entry

ECE 120: Introduction to Computing

© 2016 Steven S. Lumetta. All rights reserved.

slide 1

## Combinational Logic Design Allows Use of Abstraction

Recall combinational logic design.

- One can always design from gates (or even transistors),
- but it's often **easier to build with components** such as adders, comparators, muxes, and decoders.

We **use common functionality**

- addition, comparison, selection, and identifying encoded values (respectively)
- **to abstract away details** of low-level logic.

ECE 120: Introduction to Computing

© 2016 Steven S. Lumetta. All rights reserved.

slide 2

## FSM Designs Also Allow Use of Abstraction

The same holds for FSMs:

- one can always design every state,
- but often we want to **organize an FSM hierarchically**,
- and **analyze states in groups** rather than individually.

We can use both combinational logic components and sequential logic components such as registers and shift registers to simplify the design task.

ECE 120: Introduction to Computing

© 2016 Steven S. Lumetta. All rights reserved.

slide 3

## Let's Extend Our Keyless Entry FSM

As you may recall, our FSM design only reacted to user input (the **ULP** buttons).

For example,

- if a user pushes the panic button **P**,
- and then does nothing more,
- the FSM stays in the **ALARM** state,
- and the alarm sounds forever (until the car battery dies).

Let's modify the design to **make the FSM turn the alarm off** after some time.

ECE 120: Introduction to Computing

© 2016 Steven S. Lumetta. All rights reserved.

slide 4

## A Quick Review of I/O for Keyless Entry

Outputs are as follows:

- D** driver door; 1 means unlocked
- R** remaining doors; 1 means unlocked
- A** alarm; 1 means alarm is sounding

And inputs are as follows:

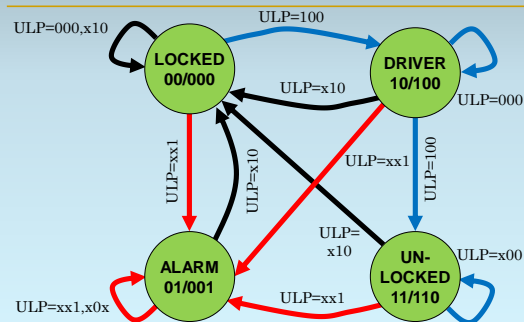
- U** unlock button; 1 means it's been pressed
- L** lock button; 1 means it's been pressed
- P** panic button; 1 means it's been pressed

## Also Review the State Table

The state table below gives the state IDs and the outputs for each state.

| meaning              | state    | $S_1S_0$ | D | R | A |
|----------------------|----------|----------|---|---|---|
| vehicle locked       | LOCKED   | 00       | 0 | 0 | 0 |
| driver door unlocked | DRIVER   | 10       | 1 | 0 | 0 |
| all doors unlocked   | UNLOCKED | 11       | 1 | 1 | 0 |
| alarm sounding       | ALARM    | 01       | 0 | 0 | 1 |

## And, Finally, the State Transition Diagram



## Start by Extending the Abstract Model

So what exactly do we want to change?

After a user turns on the alarm, the FSM should start measuring time.

Once a certain amount of time has passed, the FSM should turn off the alarm.

**In what unit can an FSM measure time?**

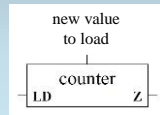
**In clock cycles.**

**What component can we use? A counter.**

## Use a Binary Counter to Measure Time

Let's use a binary down counter with a load input **LD**.

- When **LD = 1**, the counter loads a new value (from the above in the figure).
- The counter bits represent an unsigned value.
- The count (stored value) goes down towards 0.
- When the count reaches 0, the counter outputs **Z = 1**.



**You know how to build one.**

ECE 120: Introduction to Computing

© 2016 Steven S. Lametta. All rights reserved.

slide 9

## The Counter Creates New I/O Signals

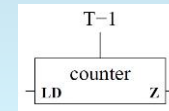
The counter gives our FSM new inputs and outputs.

Counter output **Z** is an input to the FSM.

To control the counter, the FSM must output

- **LD**, the counter load signal, and
- the counter input value.

Since we only want to use a fixed timeout, let's hardwire the value input.



ECE 120: Introduction to Computing

© 2016 Steven S. Lametta. All rights reserved.

slide 10

## How Big is the Counter?

The number of bits in the counter depends on **T**, which in turn depends on the clock speed.

For example,

- if we want a **5-minute timeout (300 seconds)**,
- and the clock speed is **16 MHz ( $1.6 \times 10^7$  cycles / second)**,
- we need  **$T = 4.8 \times 10^9$  cycles**,
- and a **33-bit counter**.

ECE 120: Introduction to Computing

© 2016 Steven S. Lametta. All rights reserved.

slide 11

## The Counter Bits are FSM State

Let's use the counter bits (denoted **timer**) to split the **ALARM** states into many states.

Whenever the user turns on the ALARM, the system will enter the **ALARM(0)** state by setting **timer = T - 1** (by setting **LD = 1**).

Then the counter counts down...



ECE 120: Introduction to Computing

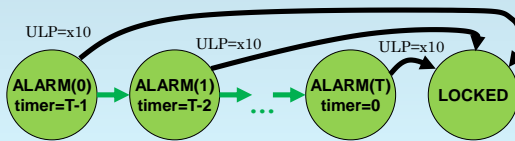
© 2016 Steven S. Lametta. All rights reserved.

slide 12

## Replicate Outgoing Arcs

We replicate outgoing arcs from **ALARM**.

So each of the states below has an arc labeled **ULP=x10** to the **LOCKED** state.



ECE 120: Introduction to Computing

© 2016 Steven S. Lametta. All rights reserved.

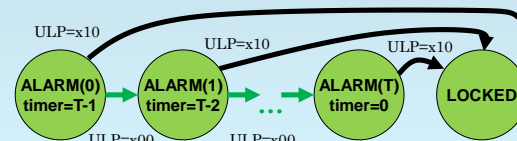
slide 13

## Time for Some Design Decisions

What if the user pushes panic (P)?

Just keep counting? Or reset the timer?

Let's **reset the timer**. So all transitions with **ULP=xx1** (not shown) enter **ALARM(0)**.



ECE 120: Introduction to Computing

© 2016 Steven S. Lametta. All rights reserved.

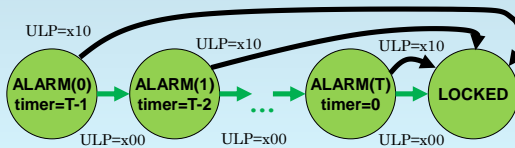
slide 14

## When the Timeout Happens, the FSM Turns Off the Alarm

What happens when **timer** reaches 0?

The counter outputs **Z = 1**, which the FSM can use to leave the **ALARM** state.

Where should it go? Let's say **LOCKED**.



ECE 120: Introduction to Computing

© 2016 Steven S. Lametta. All rights reserved.

slide 15

## Treat the Other Three States as Single States

The **timer** bits are part of the FSM state.

What about the other three states in the original design (**LOCKED**, **DRIVER**, and **UNLOCKED**)?

These states are also split into many new states!

But their **behavior is independent of the timer bits**.

So we **continue to treat them as single states**.

ECE 120: Introduction to Computing

© 2016 Steven S. Lametta. All rights reserved.

slide 16

## Two Issues Need to be Addressed by the Implementation

Now let's think about implementation.

Can we reuse the old design? Yes!

We have two issues to address:

1. Set **timer** = **T-1** when entering **ALARM(0)**.
2. Move from **ALARM** to **LOCKED** when **Z = 1**.

The counter handles the transitions from **ALARM(t)** to **ALARM(t-1)**.

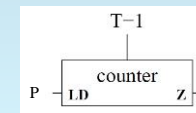
## When Should the Counter Load a New Value?

1. Set **timer** = **T-1** when entering **ALARM(0)**.

Recall that we enter **ALARM(0)** iff the **P** button is pressed. So...

(What should we do?)

...and we're done!



## Simplify the Implementation with a Mux

2. Move from **ALARM** to **LOCKED** when **Z = 1**.

**ALARM** is  $S_1S_0 = 01$ .

**LOCKED** is  $S_1S_0 = 00$ .

So we only need to change  $S_0^+$ . **How?**

Let's **use a mux**:

- the 0 input comes from the original  $S_0^+$  logic.
- the 1 input is 0 (to reach  $S_1S_0 = 00$ ).

## Calculating the Mux Select is the Hardest Part

**What controls the mux select?**

We want to force the **ALARM** to **LOCKED** transition when ...

- The system is in **ALARM** ( $S_1S_0 = 01$ ),
- AND **ULP** = **x00**,
- AND **Z** = **1**.

So the mux select signal is  $S_1'S_0L'P'Z$ .

But if we press **L**, we also move to **LOCKED**.

So we can simplify to  $S_1'S_0P'Z$ .

## Here's the Extended Implementation

