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

ECE 120: Introduction to Computing

The Design of the Lab FSM

What Problem Must be Solved?

In the lab, your task is
- to build a small FSM
- to **control a coin-operated vending machine**.

Inputs are produced by coins.

Outputs specify
- whether a coin should be accepted, and
- whether a product should be released.

The **design is extremely simple** so as to minimize the number of chips needed.

What Purpose Does the Lab Serve?

(from Prof. Doug Jones' view)

Help students to make the connection between lines and boxes on paper and wires and chips in a real system.

(from Prof. Steve Lumetta's view)

Help students to realize that the knowledge they have gained in ECE120 enables them to build real systems with sensors and actuators; in other words, to interface with the real world.

Background on the Lab FSM

Prof. Doug Jones designed the original FSM for the lab.

The derivation in these slides is somewhat different from the one in **Section 3.4** of the notes.*

Both approaches end up with the same design, which is what you must implement in the coming weeks.

*One is Lumetta's reverse-engineering of the design, and the other is Jones' explanation to Lumetta. Which is which? We forget.

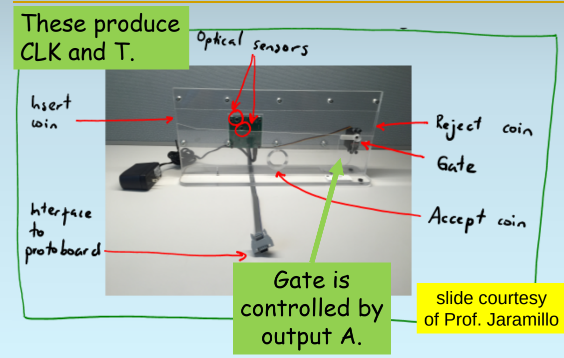
Background on the Physical System

Prof. Chris Schmitz designed and built the original prototype hardware, which worked well for the first few semesters.

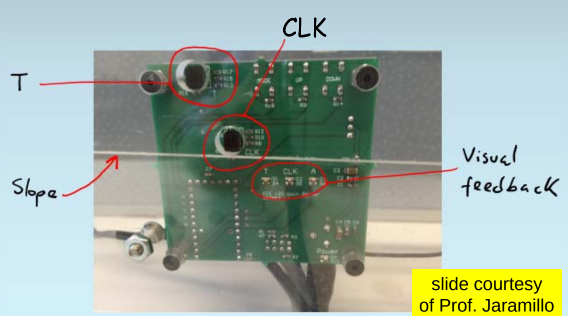
Prof. Volodymyr Kindratenko helped to evolve the design to meet the needs of more students.

Prof. Juan José Jaramillo and **Casey Smith** developed the current design to eliminate remaining issues and to enable you to do most of your work at home.

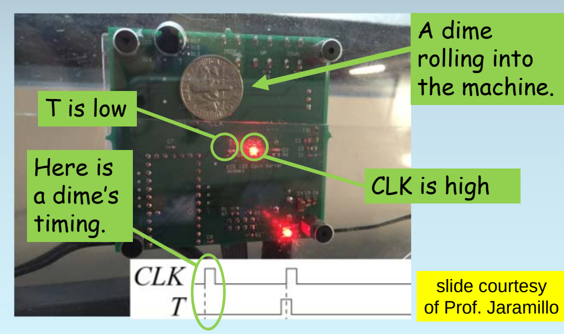
Let's Take a Look at the Physical System



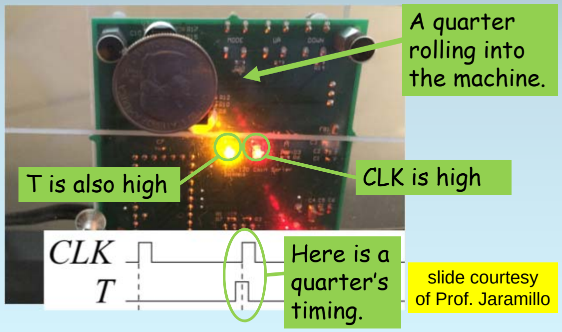
A Closer Look at Sensors and LED Feedback



What Happens When a Dime Rolls In?



What Happens When a Quarter Rolls In?



The Clock Signal is Unusual

The clock signal **CLK** is
- produced by an optical sensor
- when a coin rolls in front of it.

As a result, **CLK** is
- not a square wave, and
- not even periodic!
- The high pulses are coins.
- The pulse width depends on the coin's speed.
- The cycle time is the time between coins.

CLK Signal is Sufficient for Our FSM's Needs

However, the **CLK** signal is **sufficient for our needs**.

You build with positive edge-triggered D flip-flops.

Because of the positioning of optical sensors, **T** is stable (0 for a dime, 1 for a quarter) when **CLK** rises.

Lab Machine as a Sequence Recognizer

A **sequence recognizer** looks for bit patterns in a serial input stream.

Previously, we developed a 01 sequence recognizer as an example of the difference between Mealy and Moore machines.

For the lab, we can
- treat the sequence of coins as a serial input (0 for dimes—$0.10; 1 for quarters—$0.25).
- then look for patterns that total $0.35.

A Process for Developing a 01/10 Recognizer

If **T** is the serial input of coin types, we must produce product release output **P = 1** whenever we see **01** or **10**.

We use the following process to develop the sequence recognizer:

1. Create a state for each bit in each sequence.
2. Complete the transitions not used in the desired sequences.
3. Minimize by merging redundant states.

Inputs, Output, and Notation

Inputs:

T **0 for dime**, **1 for quarter**

(transitions use black/red to denote T)

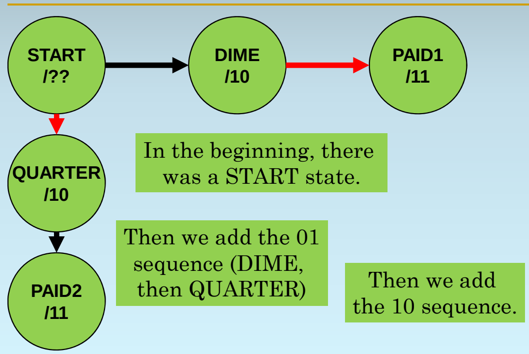
Outputs:

A 1 to accept the coin just inserted
(0 to reject it, returning to the user)

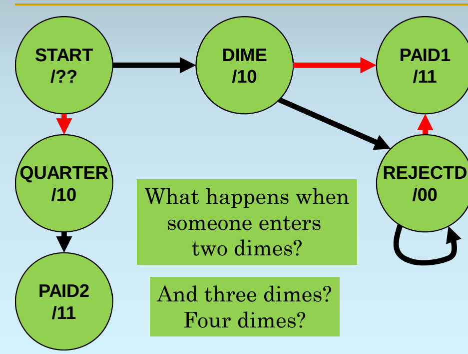
P 1 to release the product

States are marked with /AP.

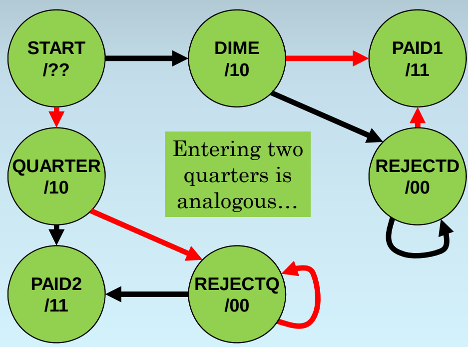
Step 1: Draw States for Both Sequences



Step 2: Complete the Transitions

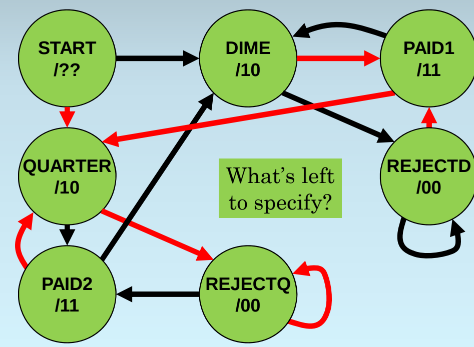


Step 2: Complete the Transitions



Entering two quarters is analogous…

Step 2: Complete the Transitions



What's left to specify?

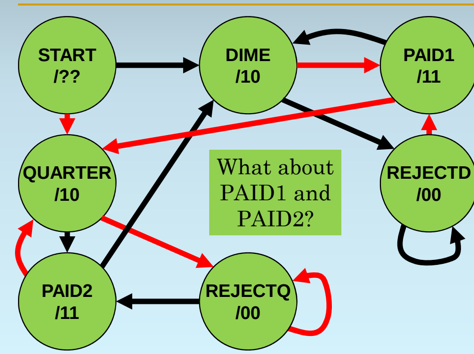
Step 3: Merge Redundant States

Now we can merge redundant states.

To merge states, it suffices to
- find two states
- with identical outputs
- and identical next states.

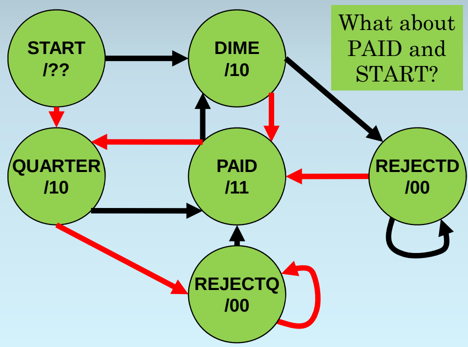
Let's take a look.

Step 3: Merge Redundant States

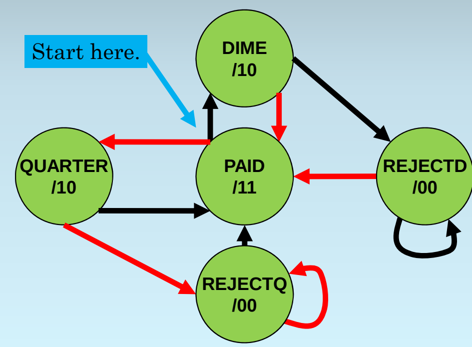


What about PAID1 and PAID2?

Any More Redundant States?



Our Final Abstract Model



Use Human Information to Define the Representation

We need 3 bits for 5 states.

Let's **use human information** to define the representation.

Think about the sequence of coins that has been inserted into the machine.

Let's call the last coin $\mathbf{T_0}$.

And the one before that $\mathbf{T_{-1}}$.

And so forth.

State Bit Definition

Define the state bits as follows:

$\mathbf{S_2}$ is $\mathbf{T_0}$, the last coin type.

$\mathbf{S_1}$ is 1 iff one or more **quarters** were inserted
- before the last coin ($\mathbf{T_{-1}}$, $\mathbf{T_{-2}}$, and so on)
- but after the last product release.

$\mathbf{S_0}$ is 1 iff one or more **dimes** were inserted
- before the last coin ($\mathbf{T_{-1}}$, $\mathbf{T_{-2}}$, and so on)
- but after the last product release.

Does that Approach Work? Check the State IDs

For **DIME**, the only coin since the last payment is a single dime.

So we have…

$S_2 = 0$ (the dime)

$S_1 = 0$ (no quarters at all)

$S_0 = 0$ (the dime is unique)

DIME	000
QUARTER	
REJECTD	
REJECTQ	
PAID	

Calculate the State ID for QUARTER

For **QUARTER**, the only coin since the last payment is a single quarter.

So we have…

$S_2 = 1$ (the quarter)

$S_1 = 0$ (the quarter is unique)

$S_0 = 0$ (no dimes at all)

DIME	000
QUARTER	100
REJECTD	
REJECTQ	
PAID	

Calculate the State ID for REJECTD

For **REJECTD**, we have seen two or more dimes but no quarters.

So we have…

$S_2 = 0$ (the last dime)

$S_1 = 0$ (no quarters at all)

$S_0 = 1$ (extra dimes)

DIME	000
QUARTER	100
REJECTD	001
REJECTQ	
PAID	

Calculate the State ID for REJECTQ

For **REJECTQ**, we have seen two or more quarters but no dimes.

So we have…

$S_2 = 1$ (the last quarter)

$S_1 = 1$ (extra quarters)

$S_0 = 0$ (no dimes at all)

DIME	000
QUARTER	100
REJECTD	001
REJECTQ	110
PAID	

What is the State ID for PAID?

For **PAID**, we could have gotten either coin last! Before the last coin, we got one or more of the other kind.

For a last quarter, we have…

S_2 = 1 (the last quarter)

S_1 = 0 (no extra quarters)

S_0 = 1 (at least one dime)

DIME	000
QUARTER	100
REJECTD	001
REJECTQ	110
PAID	101

What is the State ID for PAID?

But what about a last dime?

S_2 = 0 (the last dime)

S_1 = 1 (at least one quarter)

S_0 = 0 (no extra dimes)

So we have two bit patterns!

DIME	000
QUARTER	100
REJECTD	001
REJECTQ	110
PAID	101
PAID	010

That's All!

You will finish the rest of the design and implement it in the lab…

