Georgia Tech College of Engineering
School of Electrical
and Computer Engineering

ECE 2020:

State Machine Implementation and Applications

Instructor: Samuel Talkington

October 31, 2024

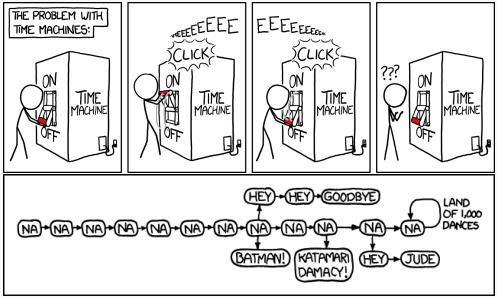
Logistics

- **Election Day next Tuesday:** Go vote! If you need time, you're excused.
- **Office hours:** Happened today. Friday office hours available.
- **Lab 2 Postponed:** Until next Thursday, Nov. 7th.
- **Old notes for this module** from a previous semester are now available on Canvas; our focus may differ, but I want you to have as many resources as possible.
- **Exam 3:** Tentatively November 14th. Practice tests now available.
- **Exam 2 solutions:** Released by tomorrow.

Building computers

Agenda: next 2 weeks

- State machine applications
- State machine implementation
- More computational science topics



Source: xkcd

Recap: FSMs

Mealy FSM

- Output depends on both present state and inputs
- The input / output is labeled along each transition arc
- The state change is synchronous to the clock, but the output changes whenever the input changes (asynchronous w clock)
- Less predictable because output changes asynchronously
- Less hardware requirements, harder design (less states)

Moore FSM

- The output depends on present state only
- The input is labeled along each transition arc
- Output is labeled inside the circle, i.e., **each state has a single output**.
- Both state and outputs change only on the clock edge
- Output changes only when the next state is reached
- More hardware requirements, easier design (more states)

Recap of FSMs

Last time, we covered JK flip flops and designing Mealy and Moore state machines.

Let's review the results from last time briefly (going to the past slides), and then pick up where we left off.

Implementation of state machines with sequential logic

1101 sequence detectors

Different types of encoding

Implementing state machines

- Thus far, our states $\mathcal{Q}$ can be quite abstract.
- To build state machines using *sequential logic*, we need to represent our states $\mathcal{Q}$ in binary
- That is, we need a system to *encode* the states into numbers.

Encoding

There are two primary ways to perform state encoding:

- Given a list of states $\mathcal{Q}$ with n elements, **we need at least** $\lceil \log_2(n) \rceil$ bits to encode each state in binary.
- The minimum number of bits $\lceil \log_2(n) \rceil$ is known as *compact encoding*.
- It is sometimes easier to assign each state its own bit, i.e., to use n bits.
- Using 1 bit per state is known as *one-hot encoding*. (sound familiar?)

Example: 1101 Sequence Detector

Suppose that we have:

- input $X(t) \in \{0,1\}$
- outputs $Z(t) \in \{0,1\}$, where

$$Z(t) = \begin{cases} 1 & \left(X(t-3) \ldots X(t)\right)_2 = (1101)_2 \\ 0 & \text{otherwise.} \end{cases}$$

Summary: **the system should output 1 when the last four inputs are 1101.**

Example: Compact Encoding for 1101 Sequence Detector

Fill in the *compact encoding* characteristic table.

$\lceil \log_2(5) \rceil = 1.58 \rightarrow 2$ bits

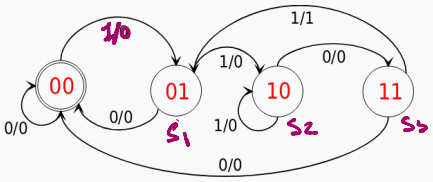


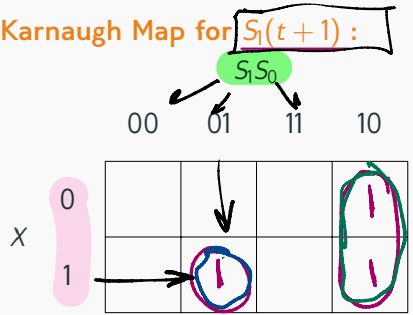
Figure 1: 1101 sequence detector (Mealy form, compact encoding)

input

$S_1(t)$	$S_0(t)$	$X(t)$	$S_1(t+1)$	$S_0(t+1)$	$Z(t)$
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	1	1	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	1	1

Example: Karnaugh Map for $S_1(t+1)$

Karnaugh Map for $S_1(t+1)$:

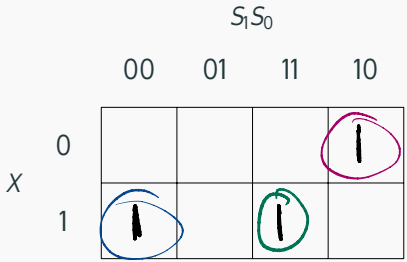


$$S_1(t+1) = S_1 \cdot \overline{S_0} + X \cdot \overline{S_1} \cdot S_0$$

$S_1(t)$	$S_0(t)$	$X(t)$	$S_1(t+1)$	$S_0(t+1)$	$Z(t)$
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	1	1	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	1	1

Example: Karnaugh Map for $S_0(t+1)$

Karnaugh Map for $S_0(t+1)$:

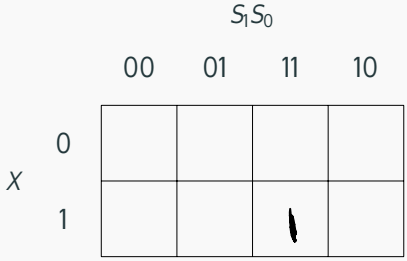


$S_1(t)$	$S_0(t)$	$X(t)$	$S_1(t+1)$	$S_0(t+1)$	$Z(t)$
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	1	1	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	1	1

$S_0(t+1) = X\cdot\overline{S_1}\cdot\overline{S_0} + X\cdot S_1\cdot S_0 + \overline{X}\cdot S_1\cdot\overline{S_0}$

Example: Karnaugh Map for $Z(t)$

Karnaugh Map for $Z(t)$:



$Z(t) = XS_1S_0$

$S_1(t)$	$S_0(t)$	$X(t)$	$S_1(t+1)$	$S_0(t+1)$	$Z(t)$
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	1	1	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	1	1

- Let's build the Mealy-form 1101 sequence detector with one-hot encoding.
- On the next slide, fill in the characteristic table for the Mealy-form 1101 sequence detector with one-hot encoding.
- **Note:** we are suppressing the dependence on t to make things fit. (Too many states!) Let S_i^+ denote the value of state S_i at time $t+1$.
- **Note:** the rows of the truth table are filled out a bit different from what we've seen before–we only have rows for each possible combination between the input X and each one-hot state.

$S_i^+ = S_i(t+1)$ for each i

Example: One-hot encoding for 1101 Sequence Detector

$q_i = (S_3 S_2 S_1 S_0)$ (indexed)

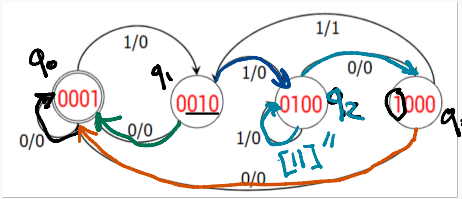


Figure 2: 1101 sequence detector (Mealy form, **one-hot** encoding.)

	S_3	S_2	S_1	S_0	X	S_3^+	S_2^+	S_1^+	S_0^+	Z
q_0	0	0	0	1	0	0	0	0	1	0
q_0	0	0	0	1	1	0	0	1	0	0
q_1	0	0	1	0	0	0	0	0	1	0
	0	0	1	0	1	0	1	0	0	0
q_2	0	1	0	0	0	1	0	0	0	0
q_2	0	1	0	0	1	0	1	0	0	0
q_3	1	0	0	0	0	0	0	0	1	0
q_3	1	0	0	0	1	0	0	1	0	1

(not)

The place of the 1 bit determines the state index.

if q_3 and $X=1 \rightarrow Z=1$

Example: Deriving state transitions for 1101 Sequence Detector

Wait... What are the state transitions? We can get these **by inspection!**

$$S_3^+ = \bar{S}_3 S_2 \bar{S}_1 \bar{S}_0 \bar{X} = S_2 \bar{X}$$

if $S_2 = 1 \Rightarrow S_i = 0 \ \forall \ i \neq 2$

$$S_2^+ = (S_2 + S_1) \cdot X$$

$$S_1^+ = (S_3 + S_0) \cdot X$$

$$S_0^+ = (S_0 + S_1 + S_3) \cdot \bar{X}$$

$$Z = S_3 \cdot X$$

S_3	S_2	S_1	S_0	X	S_3^+	S_2^+	S_1^+	S_0^+	Z
0	0	0	1	0	0	0	0	1	0
0	0	0	1	1	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	1	0	1	0	0	0
0	1	0	0	0	1	0	0	0	0
0	1	0	0	1	0	1	0	0	0
1	0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	0	1	0	1

Takeaway for one-hot encoding

- If we use **one-hot** encoding, we don't have to simplify things with K-maps!
- But there are trade-offs–what are the downsides?

Device-level state machine implementation

From FSMs to Circuits

Each **state bit** can be implemented using one *D*-flip flop.

- E.g., if we encode the states as S_2, S_1, S_0, we need 3 FFs.
- **Present state** = **output** of D-FF
- **Next state** = **input** of D-FF

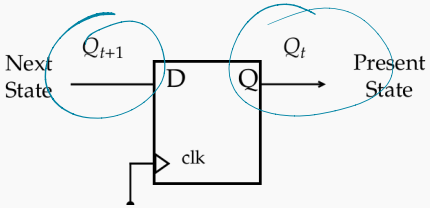


Figure 3: Relationship between D-FF and FSM states

From FSMs to Circuits

You need to use additional **combinational logic** to input:

- Next state updates
- Sequential logic circuit outputs
- E.g., HW4 “Flipping and Flopping” problem.

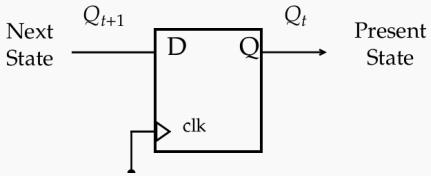
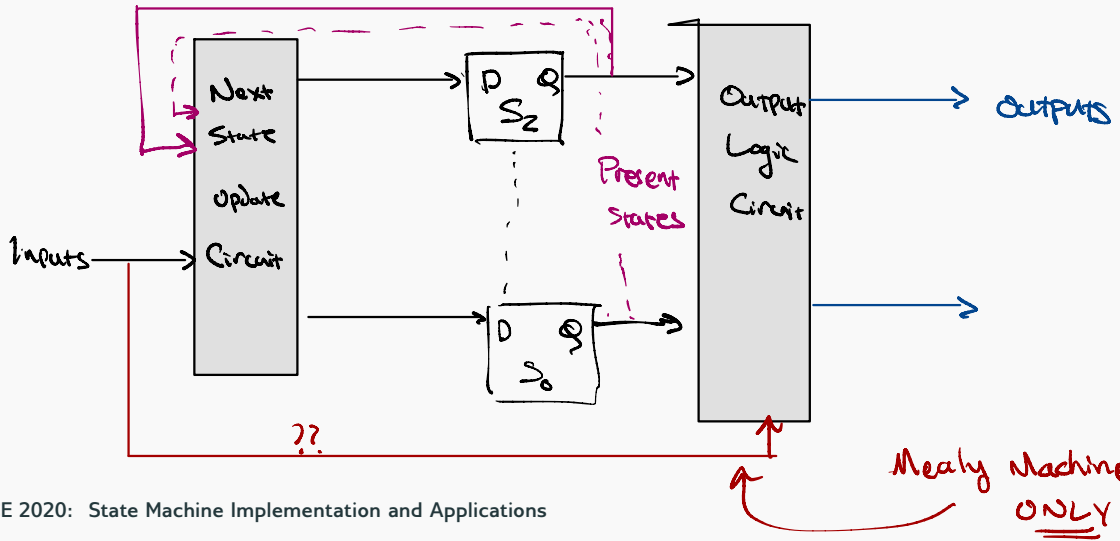


Figure 4: Relationship between D-FF and FSM states

Concept diagram for implementing state machines in circuits



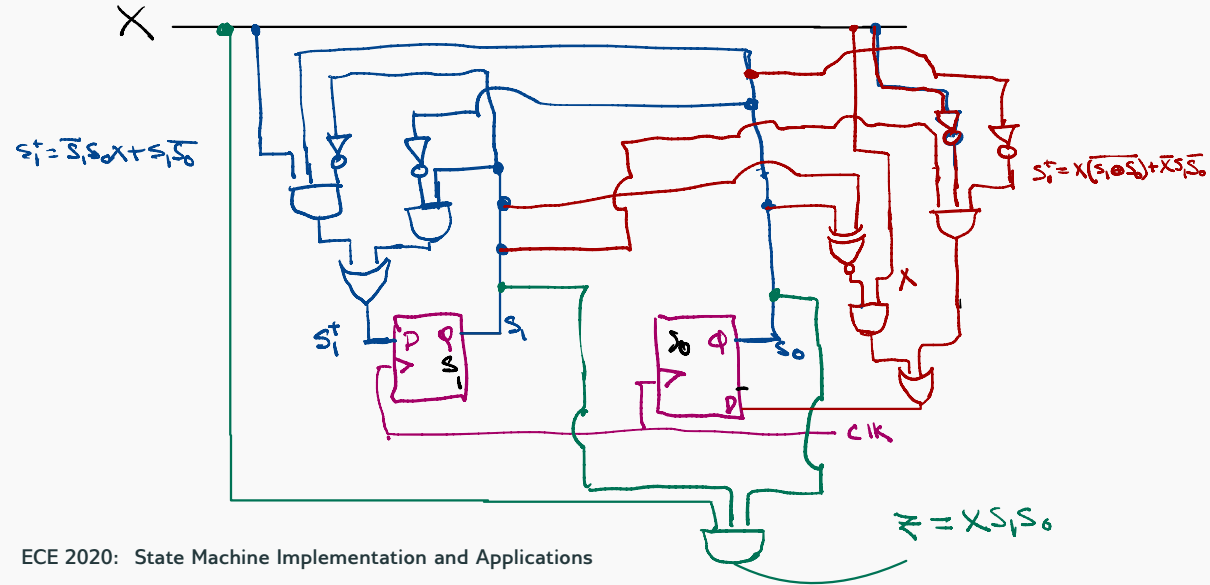
Example

Design a sequential logic circuit that implements our Mealy-form 1101 sequence detector with compact encoding.

$$S_1^+ = \bar{S}_1 S_0 X + S_1 \bar{S}_0$$
$$S_0^+ = \bar{S}_1 \bar{S}_0 X + S_1 \bar{S}_0 \bar{X} + S_1 S_0 X = X(\overline{S_1 \oplus S_0}) + \bar{X} S_1 \bar{S}_0$$
$$Z = S_1 S_0 X$$

2 bits!

Example: Design Mealy-Form 1101 sequence detector with compact encoding



FSM Implementation Summary: Step 1

Step 1: State diagram$\rightarrow$ State transition table

- Determine the # of states in state diagram (m)
- Encode each state into binary numbers using either compact encoding with $\lceil \log_2(m) \rceil$ bits or one-hot encoding with m bits
- For each state, write out the 2^n possible input rows, where $n =$ # of 1-bit inputs
- For m states and n 1-bit inputs, there are $m \times 2^n$ rows in the characteristic table.
- For each state/input pair, follow the arc of the state diagram to determine next state and output.

FSM Implementation Summary: Step 2

Step 2: State transition table$\rightarrow$ Boolean expressions

- Generate Boolean expressions for each external output and each state encoded bit.
- Simplify the Boolean expressions if necessary using the method of your choice.
- Convert to the desired gate form.

FSM Implementation Summary: Step 3

Step 3: Boolean expressions $\rightarrow$ circuit

- Draw a flip flop for each state encoding bit
- Each flip flop holds 1 bit of the present state value
- Draw logic circuits for external inputs and the inputs of each state encoding bit
- Next state = FF input, Current state = FF output.

Pair and share (15m):

How would you design a circuit to implement a **Mealy-form** FSM with **one-hot encoding** for the 1101 sequence detector?

Hints:

- How many bits do you need for the one-hot encoding of the Mealy form?
- Do you need K-maps? Why or why not?
- Use the answers to the above to derive the simplified expressions and draw the circuit.

Pair and Share: Moore-Form 1101 sequence detector with one-hot encoding

Take-home puzzle: Moore-Form 1101 sequence detector with compact encoding

Take-home puzzle

Before tomorrow at 11:59pm, design a **Moore form** of the 1101 sequence detector with **compact** encoding. Answer the following questions on Canvas (answers are provided after you submit). Remember, any submission receives full credit.

Follow the above steps of the FSM implementation summary:

- How many state bits are needed?
- Derive the simplified state transition expressions
- Derive the simplified output expression
- Draw the circuit schematic

pass: FSM