Georgia Tech College of Engineering
School of Electrical
and Computer Engineering

ECE 2020: State Machines

Instructor: Samuel Talkington

October 24, 2024

Logistics

- **Exam 2 revisions due tonight:** Any questions about this?
- **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.

Mid-semester survey

Thank you all so much for the feedback on the mid-semester survey.

I am still going through the comments (the level of detail is extremely appreciated).

Here are some of the highlights for things I am trying to do better at:

Improvement highlights:

- **More practice exams:** $\times 2$ Exam 3 practice exams have now been posted.
- **More resources:** I'm updating the collection of past semester notes for this module in the "Past Semesters" folder on Canvas. Check back in later tonight for future material so you can read ahead.
- **More examples:** Multiple examples in this lecture :)

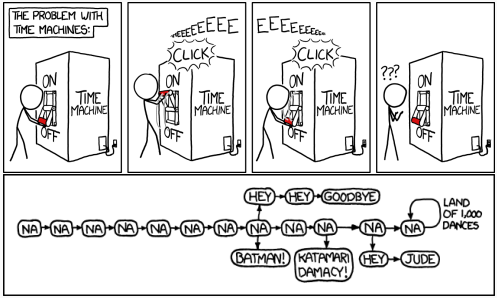
Coming soon

- **Problem set 4:** Now available, due Novermber 8th or 10th.
- **Problem set 5:** Coming soon; I advise you to start early on pset 4.
- **Prelab 2:** Released, officially due before lab, but you can turn it in later.
- **Lab 2:** Released, will occur in-class on October 31st, 2024—may potentially be pushed back depending how we feel after today's lecture
- **Exam 3:** Tentatively November 14th, in class.

Next up: the core of computers

Agenda: next 2 weeks

- Sequential logic ✓
- Latches ✓
- Flip flops ✓
- State machines ✓



Source: xkcd

Motivation: How does memory work

Todays agenda:

- JK flip flops + examples
- Finite state machines again
- Applications of finite state machines
- Examples with state machines

JK flip flops

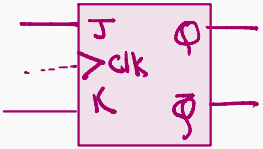
The JK flip flop

- The idea of a JK flip flop is that it combines the D, and T flip flops into one device, in some sense.
- It behaves like an SR latch without forbidden state, where $S \to J$ and $R \to K$.
- If the forbidden inputs occur, the device goes into toggle mode.
- Invented by an engineer named Jack Kilby…I wonder what the J and K stand for

$(SR) = (11) \Rightarrow$ forbidden!

$(JK) = (11) \longrightarrow$ toggle flip flop

The JK flip-flop



Positive edge triggered

Transition table

J	K	clk	Q	function
0	0	↑	Q	hold
0	1	↑	0	reset
1	0	↑	1	set
1	1	↑	$\overline{Q}$	toggle

forbidden on SR latches

Truth table with time

Characteristic table

For each timestep t:

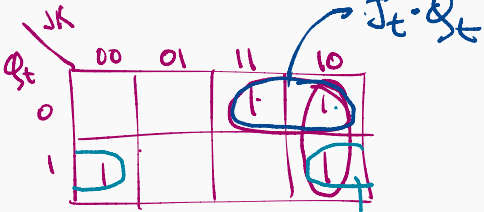
J_t	K_t	Q_t	Q_{t+1}	function
0	0	0	0	hold state
0	0	1	1	hold state
0	1	0	0	reset state
0	1	1	0	reset state
1	0	0	1	set state
1	0	1	1	set state
1	1	0	1	toggle state
1	1	1	0	toggle state

Kmap for JK flip flop

Derive the characteristic equation for the *JK* flip flop using a Karnaugh Map:

Characteristic equation

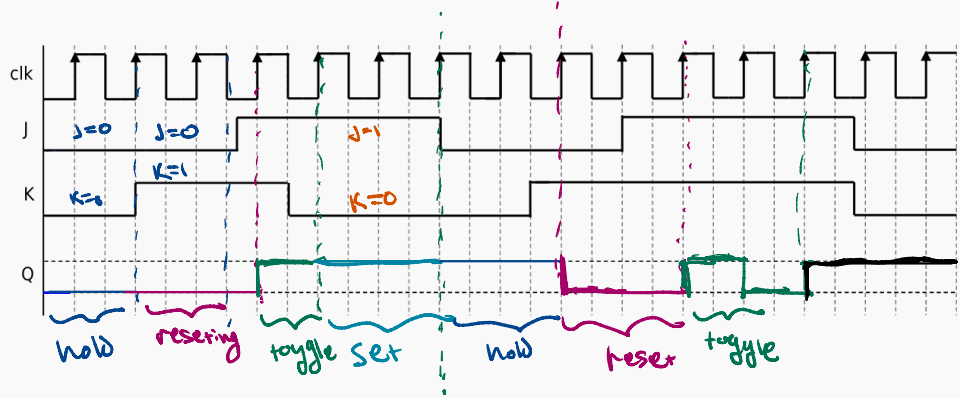
$$Q_{t+1} = J_t\overline{Q}_t + \overline{K}_tQ_t$$



Example: JK FF Timing

Look @ value just before clock cycle
↳ Is it the same for JK?

Sketch the JK flip flop timing diagram



State machines

How do computers actually work?

State machines

Latches and flip flops let us store information.

How do we perform actions based on what has happened in the past? (Our memory)

State machines... We've seen this this week! Let's remind ourselves:

Defining state diagrams explicitly

Definition: Finite state machine (FSM)

An n-dimensional finite state machine (FSM) is an abstract model of a computer, which is defined by a collection of **5 elements** $(\mathcal{Q}, \mathcal{X}, \delta, q_0, \mathcal{F})$, where

- the set of all available states is $\mathcal{Q} = \{q_1, q_2, ...\}$
- the set of all available inputs is $\mathcal{X} = \{x_1, x_2, ...\}$
- the *state transition function* is $\delta : \mathcal{Q} \times \mathcal{X} \to \mathcal{Q}$
- the *initial state* is $q_0 \in \mathcal{Q}$, and
- the set of final states is $\mathcal{F} \subseteq \mathcal{Q}$ (it can possibly be empty).

final states — subset of — all states

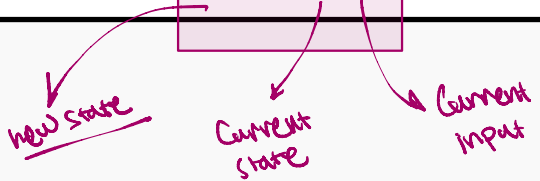
State transition

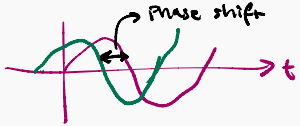
State transition function

The state transition function $\delta : \mathcal{Q} \times \mathcal{X} \to \mathcal{Q}$ takes in a current state and the present input and returns a new element $q_{t+1} \in \mathcal{Q}$ from the set of all states $\mathcal{Q}$.

We can write:

$$q_{t+1} = \delta(q_t, x_t).$$



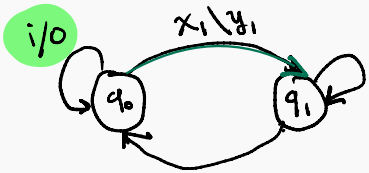


synchronous FSM

- Depends on inputs and state at *discrete* instances of time
- e.g. clocked CPU chips, flip-flops, chip registers etc.
- this is what we care about

asynchronous FSM

- Depends on inputs and states at any instance of time
- e.g. interrupt-driven computers, asynchronous communication systems, etc.



Mealy FSM

- Output depends on both present state and inputs
- The input / output is labelled along each transition arc

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.**

Applications of State Machines

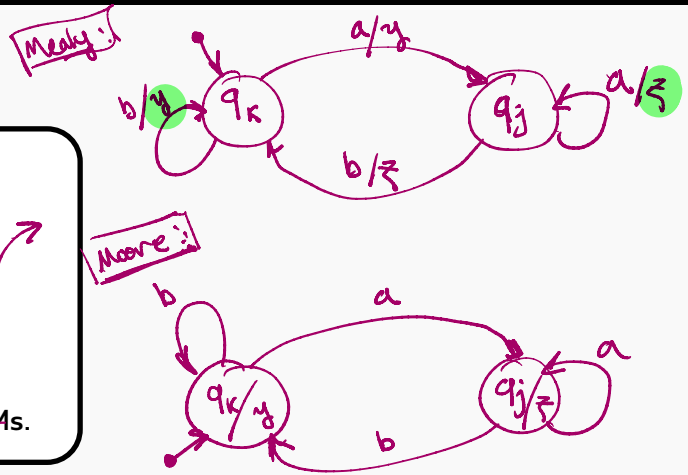
Example 1: Mealy and Moore State Machines

Example 1

Suppose that we have:

- states $\mathcal{Q} := \{q_k, q_j\}$
- inputs $\mathcal{X} := \{a, b\}$
- outputs $\mathcal{F}; = \{y, z\}$
- initial state $q(0) = q_k$

~~Draw the Mealy and Moore FSMs~~.



Given these diagrams

Example 2: Sequence recognizer

Example 2

Suppose that we have:

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

$$Z(t) = \begin{cases} 1 & (X_{t-3} \ldots X_t)_2 = (1101)_2 \\ 0 & \text{otherwise} \end{cases}$$

i.e., the system should output 1 when last four inputs are 1101.

Questions:

- Define the states
- How to do this in Mealy and Moore models?

Example 2 Approach

Looking for part 4 X's to look like $(1101)_2$

Time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$X(t)$	1	0	0	1	0	1	1	0	1	0	1	1	0	1	1	0	1
$Z(t)$	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1

Example 2 Approach

Time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$X(t)$	1	0	0	1	0	1	1	0	1	0	1	1	0	1	1	0	1
$Z(t)$	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1

- Need states that remember **at least** 3 bits of past data (past values of $X(t)$).
- Specifically, we need states that are at least [1], [1, 1], and [1, 1, 0].
- The actual states we use depend on whether we choose Mealy or Moore.

Example 2.1: Solution with Mealy machine

Example 2: Mealy solution

Define the states $\mathcal{Q} = \{q_0, q_1, q_2, q_3\}$, where:

- $q_0 = [\emptyset]$, initial state (empty)
- $q_1 = [1]$: sub-pattern 1 detected
- $q_2 = [11]$: sub-pattern 11 detected
- $q_3 = [110]$: sub-pattern 110 detected – what happens?

1 on the transition to q_1

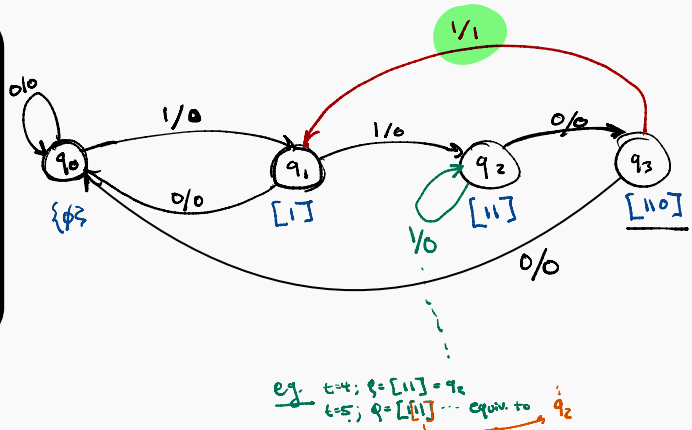
Example 2.1: Solution with Mealy machine

States

Define the states

$\mathcal{Q} = \{q_0, q_1, q_2, q_3\}$, where:

- $q_0 = [\emptyset]$, initial state
- $q_1 = [1]$: detected 1
- $q_2 = [11]$: detected 11
- $q_3 = [110]$: detected 110



Example 2.2: Moore machine pattern recognition

We can also approach pattern recognition with a Moore Machine.

In contrast with the Mealy approach, we need to **assign a unique output** to each state.

Therefore, for Example 2, we will need to add an additional state $q_4 = [1101]$.

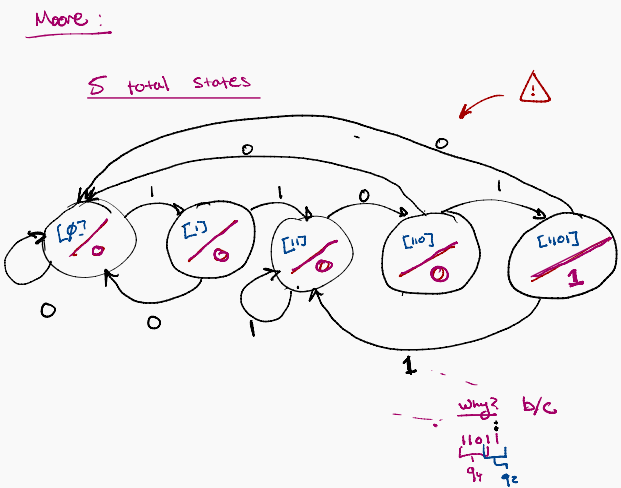
final moore

Example 2.2: Moore solution

Define the states

$\mathcal{Q} = \{q_0, q_1, q_2, q_3, q_4\}$, where:

- $q_0 = [\emptyset]$, initial state (empty)
- $q_1 = [1]$: sub-pattern 1 detected
- $q_2 = [11]$: sub-pattern 11 detected
- $q_3 = [110]$: sub-pattern 110 detected
- $q_4 = [1101]$: sub-pattern 1101 detected



General idea: pattern matching problem

Pattern matching problems

If your input is a binary signal X and you want to track the appearance of a particular pattern in X across time, do the following:

- Create a string matching tree
- State output = 1 if a pattern is contained along a path
- Add failure edges–pick the longest suffix of the string seen so far and transition to the corresponding prefix state.

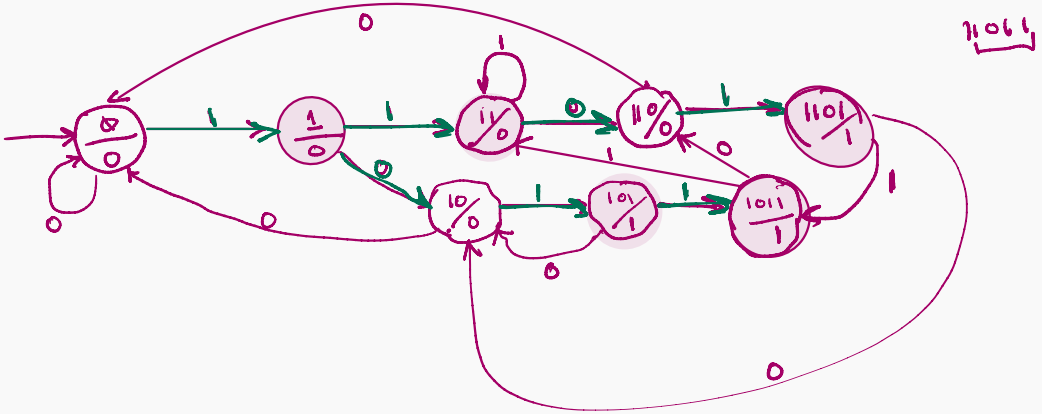
Example 3: General Pattern Recognition

Example 3

Let $X(t) \in \{0,1\}$ be a streaming binary input. Create a Moore Machine that outputs 1 if any of the following patterns are detected:

- 1101
- 1011
- 101

Example 3: Detect 1101, 1011, 101



Implementation of state machines with sequential logic

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 to *encode* them into numbers.

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 4: 1101 Sequence Detector Compact Encoding

Recall that 1101 sequence detector for a streaming binary signal. Fill in the *compact encoding* characteristic table.

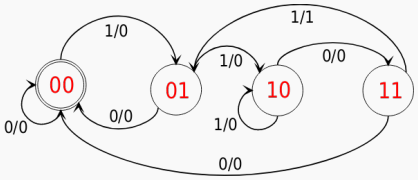
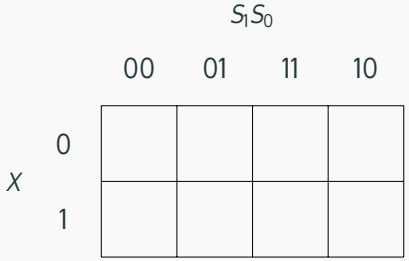


Figure 1: 1101 sequence detector (Mealy form)

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

Example 4: Karnaugh Map for $S_1(t+1)$

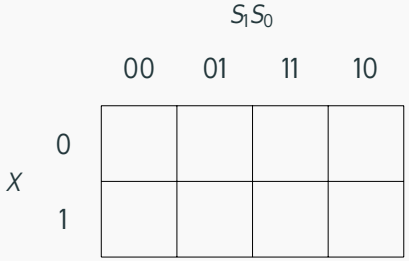
Karnaugh Map for $S_1(t+1)$:



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

Example 4: Karnaugh Map for $S_0(t+1)$

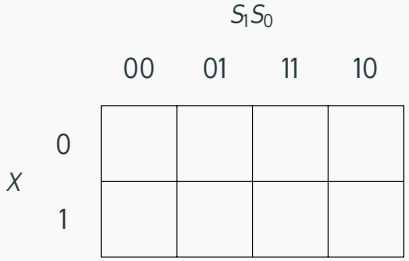
Karnaugh Map for $S_0(t+1)$:



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

Example 4: Karnaugh Map for $Z(t)$

Karnaugh Map for $Z(t)$:



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

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

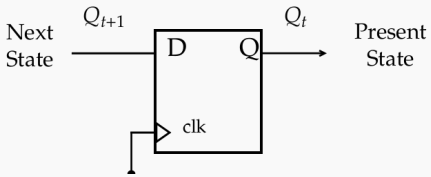


Figure 2: Relationship between D-FF and FSM states

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.

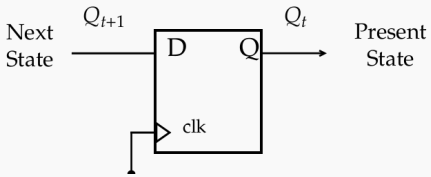


Figure 3: Relationship between D-FF and FSM states