Finite state machines: abstraction

Lecture Topics

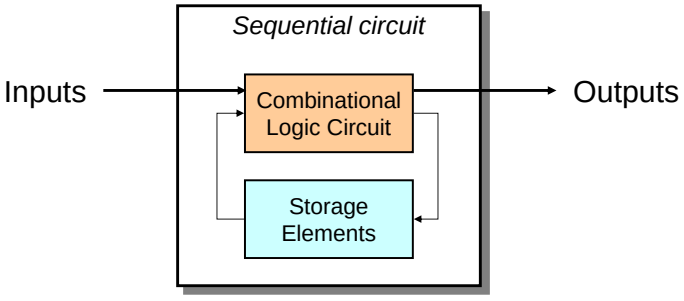
- Finite State Machine (FSM)
- Machine Models

Reading assignments

- Textbook § 3.6
- Prof. Lumetta's Notes Set 3.1: Serialization and Finite State Machines

Sequential logic

- combinational circuit: output is a function of its input ONLY
  - examples: logic gates, adder, MUX, etc.
- sequential circuit: output depends on the input AND on stored information (state)
  - examples: memory elements, finite state machine, etc.
  - state of a system is a "snapshot" of all relevant elements at a given moment in time
  - conceptually, sequential logic circuit consists of 1) a combinational logic circuit and 2) a storage element



(state machine)

- We will learn how to design sequential circuits by representing them as finite state machines

Finite State Machines

- A finite state machine (or FSM) is a model for understanding the behavior of a system by describing the system as occupying one of a finite set of states, moving between these states in response to external inputs, and producing external outputs.
- In any given state, a particular input may cause the FSM to move to another state
  - This combination is called a transition rule.
- An FSM consists of 5 elements
  - A finite number of *states*
  - A finite number of *external inputs*
  - A finite number of *external outputs*
  - An explicit specification of all *state transitions*
  - An explicit specification of what determines each external *output value*
- When an FSM is implemented as a digital system
  - all states must be represented as patterns using a fixed number of bits
  - all inputs must be translated into bits, and
  - all outputs must be translated into bits
  - For a digital FSM, transition rules must be complete
    - given any state of the FSM, and any pattern of input bits, a transition must be defined from that state to another state or itself.
  - Calculation of outputs for a digital FSM reduces to Boolean logic expressions.
- In this class, we will focus on clocked synchronous FSM implementations, in which the FSM's internal state bits are stored in flip-flops.
- FSM can be described by
  - List of abstract states
  - Next-state or state transition table
  - State transition or simply state diagram

- FSM can be implemented a s a sequential circuit consisting of
  - Combinational logic that computes external outputs and state transitions
  - Storage elements that store current state

Example: FSM for a keyless entry system for a car

Abstract model

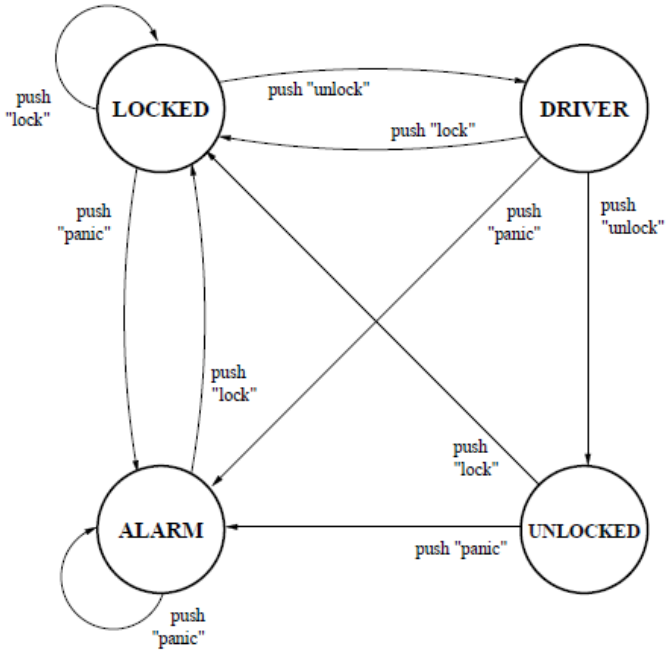
- **List of abstract states** for a keyless entry system for a car

meaning	state	driver's door	other doors	alarm on
vehicle locked	LOCKED	locked	locked	no
driver door unlocked	DRIVER	unlocked	locked	no
all doors unlocked	UNLOCKED	unlocked	unlocked	no
alarm sounding	ALARM	locked	locked	yes

  - We have merely named the states rather than specifying the bit patterns to be used for each state—for this reason, we refer to them as abstract states.
  - The description of the states in the first column is an optional element often included in the early design stages for an FSM, when identifying the states needed for the design.
  - A list may also include the outputs for each state. Again, in the list below, we have specified these outputs abstractly. By including outputs for each state, we implicitly assume that outputs depend only on the state of the FSM.
    - We will return to this assumption in more detail later
- **State table** (or next-state table, or state transition table)
  - Maps the current state and input combination into the next state of the FSM.

state	action/input	next state
LOCKED	push "unlock"	DRIVER
DRIVER	push "unlock"	UNLOCKED
(any)	push "lock"	LOCKED
(any)	push "panic"	ALARM

  - This abstract state table outlines desired behavior at a high level, and is often ambiguous, incomplete, and even inconsistent
    - For example, what happens if a user pushes two buttons?
    - What happens if they push unlock while the alarm is sounding?
    - These questions should eventually be considered.
- **State transition diagram** (or transition diagram, or state diagram) illustrates the contents of the next-state table graphically
  - with each state drawn in a circle, and
  - arcs between states labeled with the input combinations that cause these transitions from one state to another.



- Implementing an FSM using digital logic requires that we
  - translate the design into bits,
  - eliminate any ambiguity, and
  - complete the specification.
- There are many questions to answer in order to implement this FSM as a digital circuit
  - How many internal bits should we use?
  - What are the possible input values, and how are their meanings represented in bits?
  - What are the possible output values, and how are their meanings represented in bits?
  - …

From abstract model to digital system

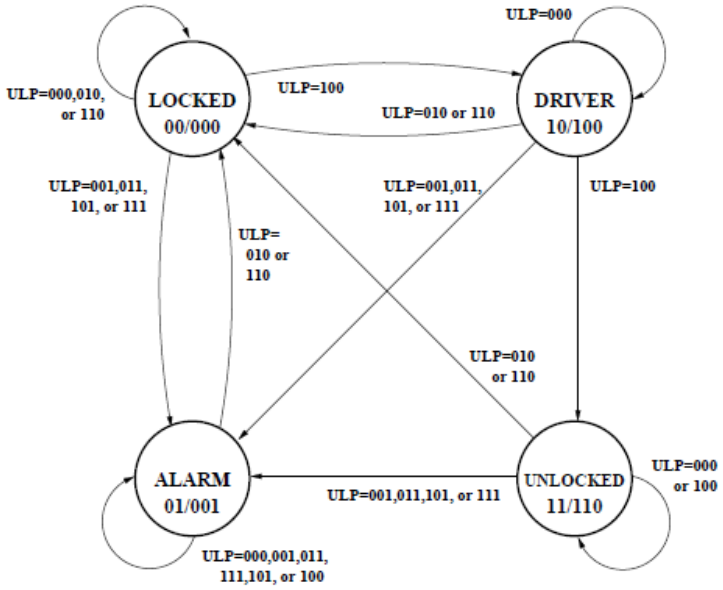
- Given four states, we need at least ⌈log2(4)⌉ = 2 bits of internal state, which we store in two flip-flops and call S_1S_0.
- Let's list input and output signals and define their meaning:
  - Outputs
    - D driver door; 1 means unlocked
    - R other doors (Remaining doors); 1 means unlocked
    - A alarm; 1 means alarm is sounding
  - Inputs
    - U unlock button; 1 means it has been pressed
    - L lock button; 1 means it has been pressed
    - P panic button; 1 means it has been pressed
- We can now choose a representation for our states and rewrite **the list of states**, using bits both for the states and for the outputs.
  - The order of states in the list is not particularly important, but should be chosen for convenience and clarity

meaning	state	S_1S_0	driver's door D	other doors R	alarm on 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

- We can also rewrite the **next-state table** in terms of bits.
  - We use Gray code order on both axes, as these orders make it more convenient to use K-maps.
  - The values represented in this table are the next FSM state given the current state S_1S_0 and the inputs U, L, and P.
  - Our symbols for the next-state bits are S_1^+ and S_0^+.
    - The "+" superscript is a common way of expressing the next value in a discrete series
  - The next-sate table should be sufficient to derive expressions for $S_1^+(S_1, S_0,U,L, P)$ and $S_0^+(S_1, S_0,U,L, P)$ as well as expressions for the output logic $D(S_1, S_0)$,$R(S_1, S_0)$, and $A(S_1, S_0)$.

current state	ULP							
S_1S_0	000	001	011	010	110	111	101	100
00	00	01	01	00	00	01	01	10
01	01	01	01	00	00	01	01	01
11	11	01	01	00	00	01	01	11
10	10	01	01	00	00	01	01	11

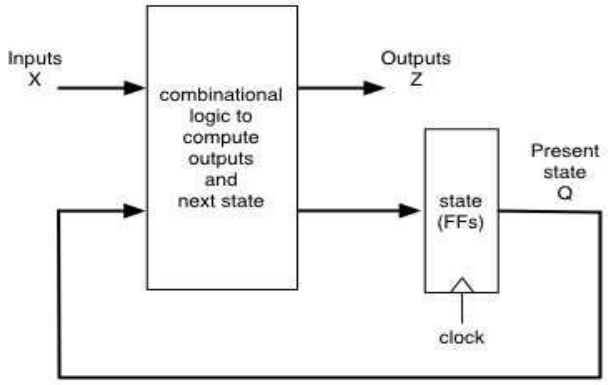
- In the process of writing out the next-state table, we have made decisions for all of the questions that we asked earlier regarding the abstract state table. These decisions are also reflected in the complete state transition diagram shown below.



- Note that the states have been extended with state bits and output bits, as S_1S_0/DRA.
- What's left is to write out Boolean expressions for the next-state variables S_1^+ and S_0^+, and for the outputs D, R, and A.

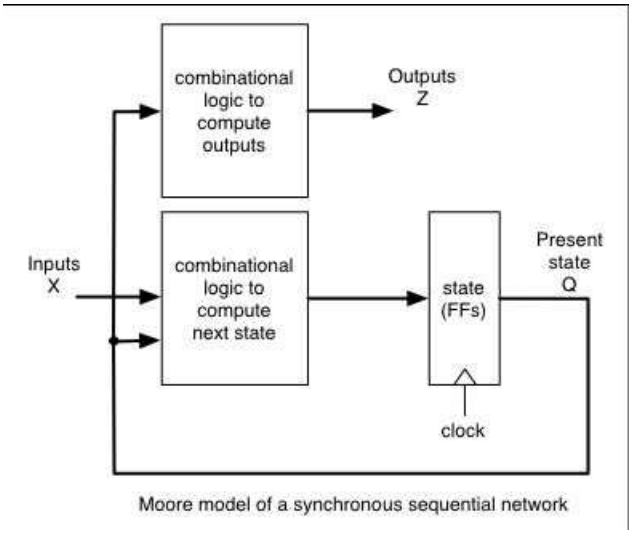
Machine Models

- There are two types of sequential circuit models:
  - **Mealy machine**. The outputs depend on the present state and the inputs. Mealy outputs are asynchronous: outputs can change in response to input changes - independent of the clock.



Mealy model of a synchronous sequential network

  - **Moore machine**. The outputs depend only on the present state. Moore outputs are synchronous: outputs change only with the clock edge.



Moore model of a synchronous sequential network

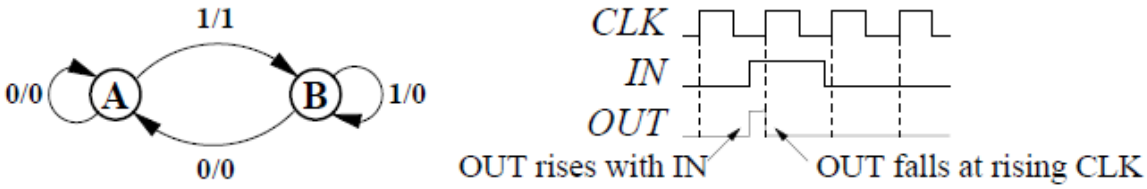
  - There are also machines which are mixed Mealy/Moore. These have both Mealy outputs and Moore outputs.
- Mealy-Moore Differences
  - In general, a Moore machine has more states than an equivalent Mealy machine because different states are required for different outputs.
  - The behavior of the Moore machine differs slightly from the Mealy.
    - A Mealy machine can have different output values within a single state.

- The Moore machine will have a single output value with each state.
- So the Moore machine typically has more states and the output may be slightly delayed.
- Also, a Mealy machine may have "false outputs" or "glitches", due to the timing of input changes.

- Advantages / Disadvantages:
  - The Moore machine has a number of advantages over Mealy:
    - the output can be read during the entire clock period,
    - there are no output glitches, and
    - Moore machines are easier to compose.
  - However, the big advantage of Mealy machines is that they require fewer states - and for this reason they continue to be popular with designers.

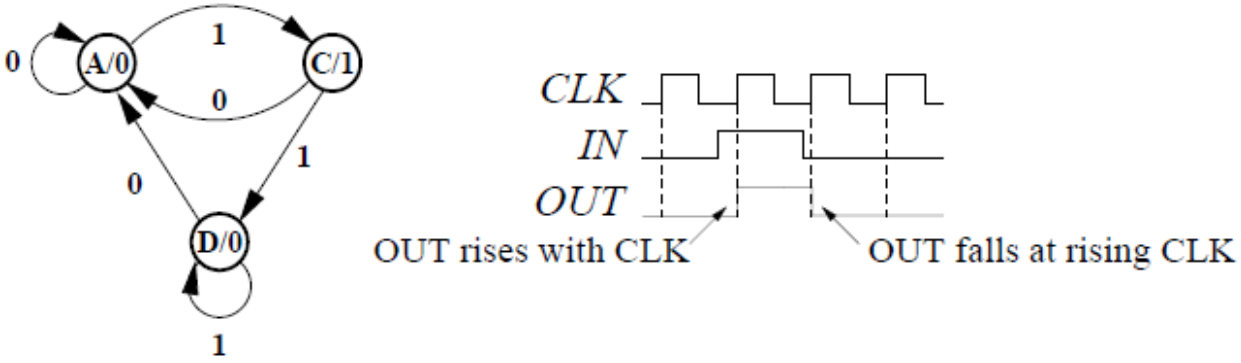
Example: 01 sequence recognizer

- Let's develop a model for a recognizer of the pattern of a 0 followed by a 1 on a single input that outputs a 1 when it observes the pattern.
- **Mealy** machine



  - The machine occupies state A when the last bit seen was a 0, and state B when the last bit seen was a 1.
  - The transition arcs in the state diagram are labeled with two values instead of one:
    - Since outputs can depend on input values as well as state, transitions in a Mealy machine are labeled with **input/output** combinations, while states are labeled only with their internal bits (or just their names).
  - Notice that the outputs indicated on any given transition hold only until that transition is taken (at the rising clock edge), as is apparent in the timing diagram.
    - When inputs are asynchronous, that is, not driven by the same clock signal, output pulses from a Mealy machine can be arbitrarily short, which can lead to problems.

- **Moore** machine



- For a Moore machine, we must create a special state in which the output is high.
- Doing so requires that we split state B into two states
  - a state C in which the last two bits seen were 01, and
  - a state D in which the last two bits seen were 11.
  - Only state C generates output 1.
  - State D also becomes the starting state for the new state machine.
- The state diagram illustrates the changes, using the transition diagram style that we introduced earlier to represent Moore machines.
- Notice in the associated timing diagram that the output pulse lasts a full clock cycle.