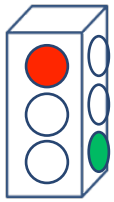
ECE 2534

State Machines

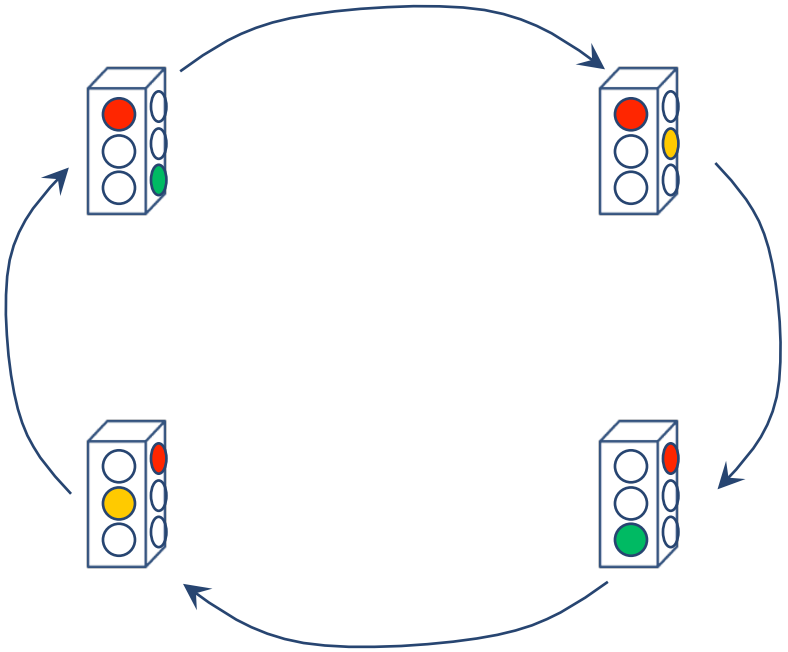
Finite State Machines (FSM)

- A FSM is an abstract representation of a system
  - The system is in one “state” at a time
  - Inputs to the system may cause state transitions
  - State-specific outputs/actions can take place
- “Machine” does not necessarily refer to a physical device
- Very common approach for structuring hardware or software control
  - Virtually all digital hardware control units are implemented as state machines
  - Embedded control software and device drivers are usually structured as state machines

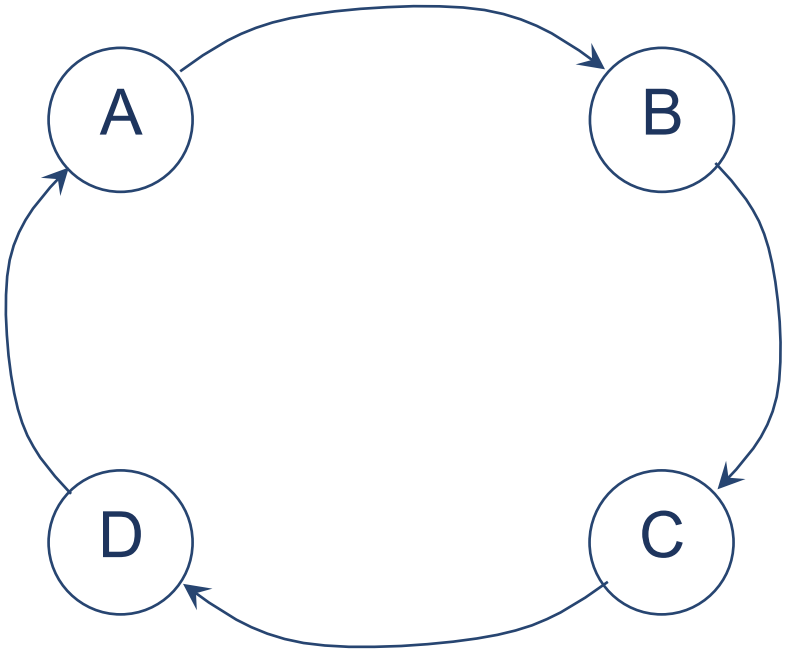
Example: simple traffic-light controller



Example: simple traffic-light controller



State diagram



A: red NS, green EW

B: red NS, yellow EW

C: green NS, red EW

D: yellow NS, red EW

Partial state table (state-transition table)

Current State	Current Input N	Next State
A	0	A
	1	B
B	0	B
	1	C
C	0	C
	1	D
D	0	D
	1	A

State table (state-transition table)

Current State	Current Input N	Next State	Current Output Λ
A	0	A	100001
	1	B	
B	0	B	100010
	1	C	
C	0	C	001100
	1	D	
D	0	D	010100
	1	A	

Formally, a Finite State Machine is …

- A finite, non-empty set of possible inputs Σ
- A finite, non-empty set of possible states S
- A state-transition function $\delta : S \times \Sigma \rightarrow S$
- An initial state $s_0 \in S$
- A finite, possibly empty set of outputs Λ

FSM-based software design

- Select a set of states S that can represent the system
- Decide what events should cause state transitions, and decide what outputs/actions are needed from the system
- Convert the specification to a state diagram or state-transition table
- Write code using conditional expressions to implement the FSM
  - if …
    else if …
    else if …
    else if …
    else …
  - switch …
    case …
    case …
    case …
    default …

FSM-based software design

- It is common to use an enumerated type (**enum**) for the set of states S

```
enum States {STATE_A, STATE_B, STATE_C, STATE_D};

enum States systemState;   // declare a state variable
systemState = STATE_A;     // initialize system state

.
.
.

systemState = STATE_B;     // update system state
.
.
.
```

FSM-based software design

- Can use conditional statements (**if/else** or **switch**) to implement the behavior specified for the FSM

```
switch (systemState)
{
case STATE_A:
      <statements>
      break;
case STATE_B:
      <statements>
      break;
case STATE_C:
      <statements>
      break;
. . .
default:
     <statements>
}
```

State-machine example: Dollar-bill change machine

Current State	Actions	Event	Next State
READY	Turn on ready light	Bill detected	LOAD_BILL
		<none>	READY
LOAD_BILL	Turn on wait light	Bill loaded	VERIFY_BILL
		Bill jam	REJECT_BILL
VERIFY_BILL	Scan bill	Scan passed	CHECK_COINS
		Scan failed	REJECT_BILL
REJECT_BILL	Turn off wait light Eject bill	Bill ejected	READY
CHECK_COINS	Check if sufficient coins remain	Sufficient coins	DISPENSE_COINS
		Insufficient coins	OUT_OF_CHANGE
DISPENSE_COINS	Turn off wait light Dispense change	All coins dispensed	READY
OUT_OF_CHANGE	Eject bill Turn off ready light	<none>	OUT_OF_CHANGE

```
enum states {
    READY,
    LOAD_BILL,
    VERIFY_BILL,
    REJECT_BILL,
    CHECK_COINS,
    DISPENSE_COINS,
    OUT_OF_CHANGE
};
enum states state;
char c;
```

```
state = READY;
while (1) {
    switch (state) {
    case READY:
        READY actions
        c = getInput();
        switch (c) {
            case c0: state = whatever; break;
            case c1: state = whatever; break;
              …
            default: state = whatever; break;
        }
        break;
    case LOAD_BILL:
        LOAD_BILL actions
        c = getInput();
        switch (c) {
            case c0: state = whatever; break;
            case c1: state = whatever; break;
                  …
        default: state = whatever; break;
        }
    break;
      …
    }
}
```

Summary

- Finite State Machines are commonly used in HW and SW design

- The use of an FSM encourages a systematic, methodical approach for designing a digital system
  - Good for designing the system
  - Good for debugging/testing the system

- Another name for the same thing:
  Finite-State Automaton

  (plural: Automata)