Embedded Software

CS 145/145L

Caio Batista de Melo

Announcements (2022-04-21)

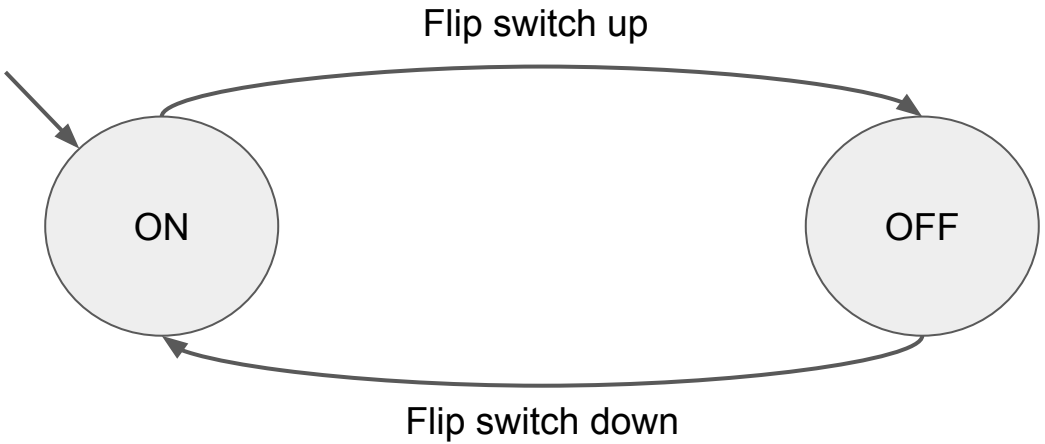
- Project 1 was due yesterday
- Homework 2 is due tomorrow
- Keypad detects the wrong button being pressed
  - Add a small delay (1ms) between writing a week 1 and reading it back
  - It should let the microcontroller read the final value “after things have settled”

What is a State Machine (SM)?

Collection of states with info on:

- how to go from one state to another one;
- what to do in each state.

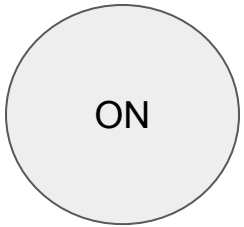
Example: ICS2-162 lights



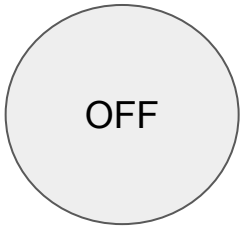
States

- Describes the *state* of the system;
- Can have actions attached to them.

Example: ICS2-162 lights



“give” power to light

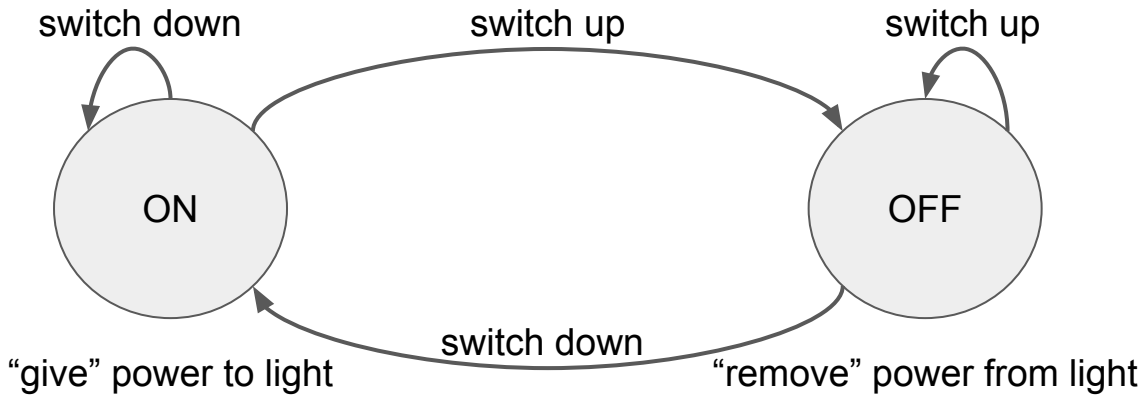


“remove” power from light

Transitions

- Describes what's the next state;
- The next state can be the same one!
- Can have conditions attached to them;
- Can have actions attached to them.

Example: ICS2-162 lights

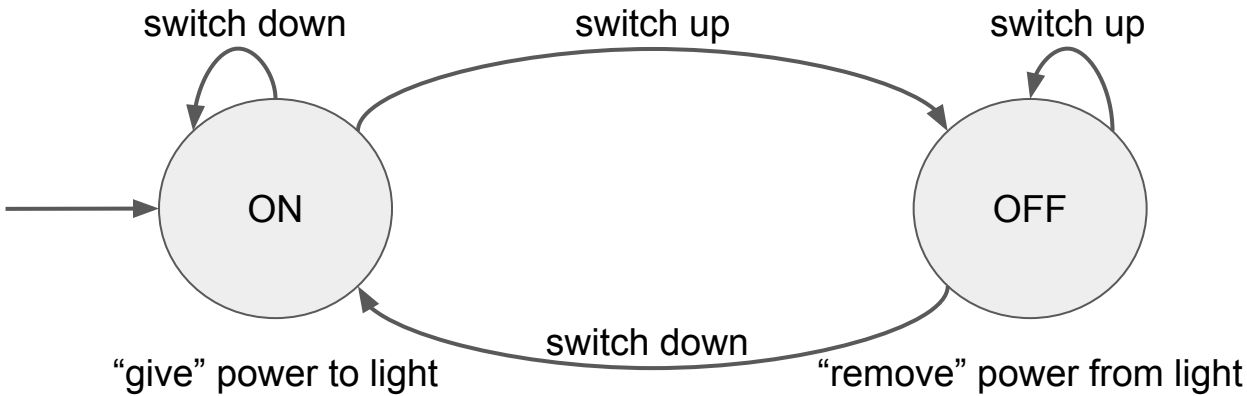


Initial State

Defines where do you start in the SM.

Can have some initialization for the system (e.g., clean variables).

Example: ICS2-162 lights



SM for Project 1

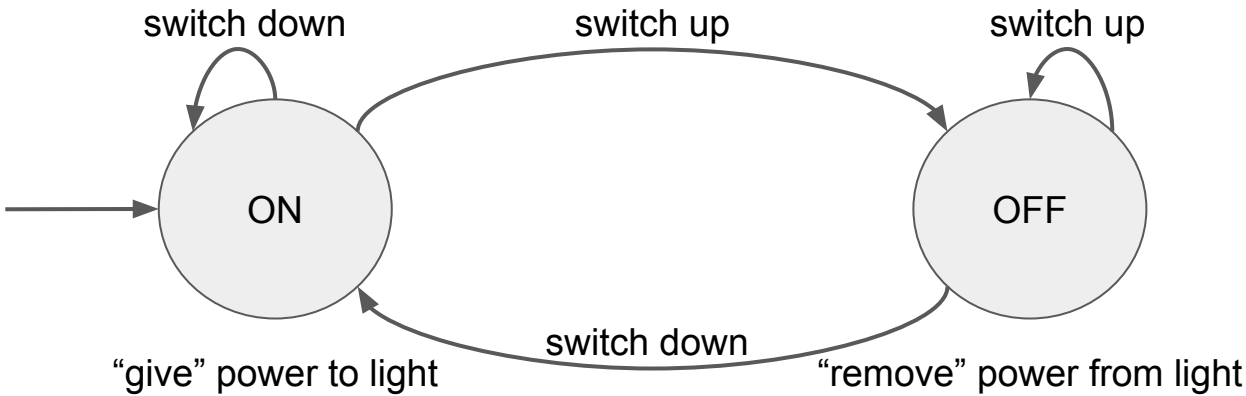
Project 1: blink an LED whenever a button is pressed.

How can we do that with a SM?

- What are the states?
- What are their actions?
- How do we switch between them?

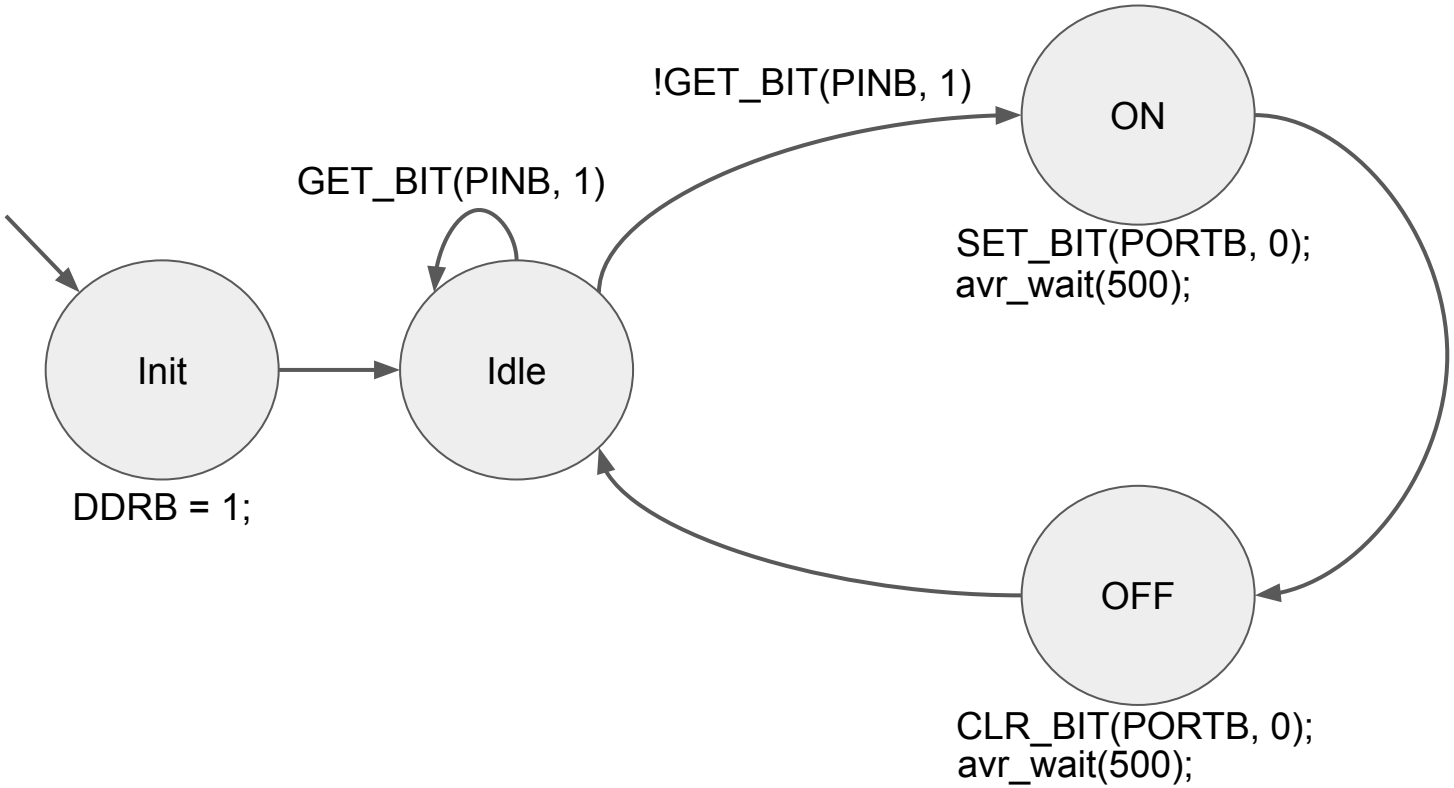
SM for Project 1

ICS2-162 lights:

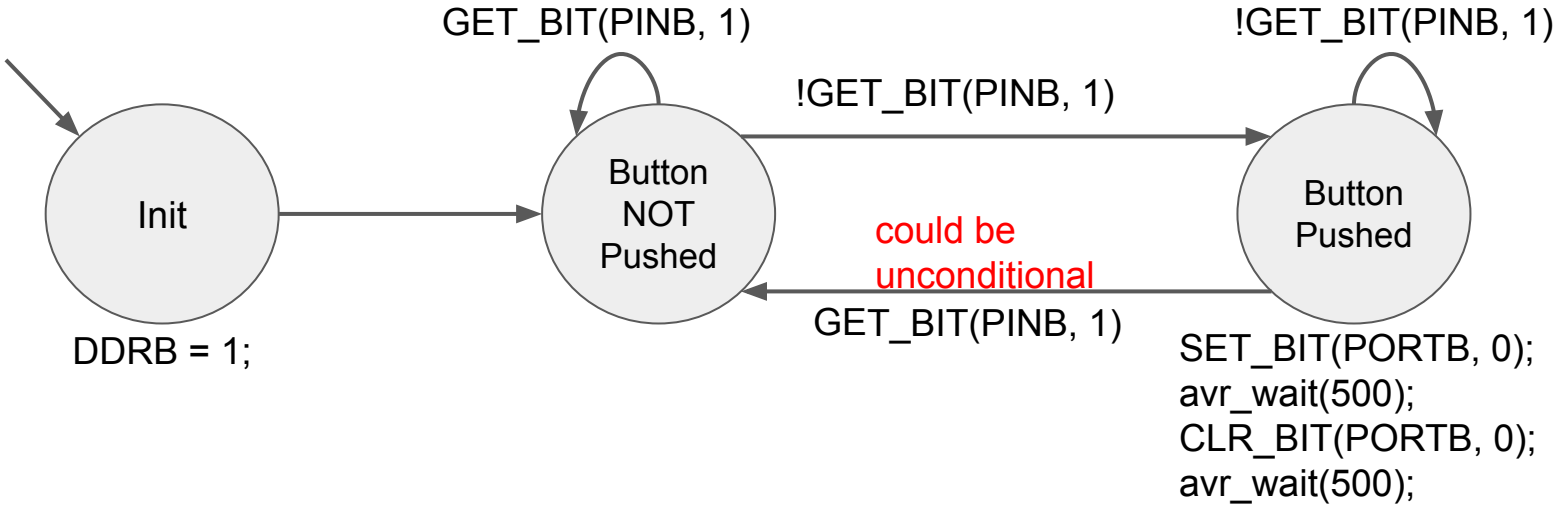


Not just flipping between them after an event,
might need more states and actions.

SM for Project 1 (v1)



SM for Project 1 (v2)



SMs in C-Code

There are libraries that can manage it for you:

- RIBS (zybooks)
- https://github.com/misje/stateMachine
- https://github.com/endurodave/C_StateMachine

But it's usually straightforward to implement them!

SMs in C-Code

```
int state = INITIAL_STATE;
while (1) {
  switch(state) {
    case 0:
      state_0_actions();
      state = state_0_transition();
      break;
    case 1:
      state_1_actions();
      state = state_1_transition();
      break;
    default:
      break;
  }
}
```

Start on initial state

Loop forever

Figure out current state

Execute actions for the state

Transition to next state

Why use SMs?

- easy to understand;
- easy to design;
- easy to implement;

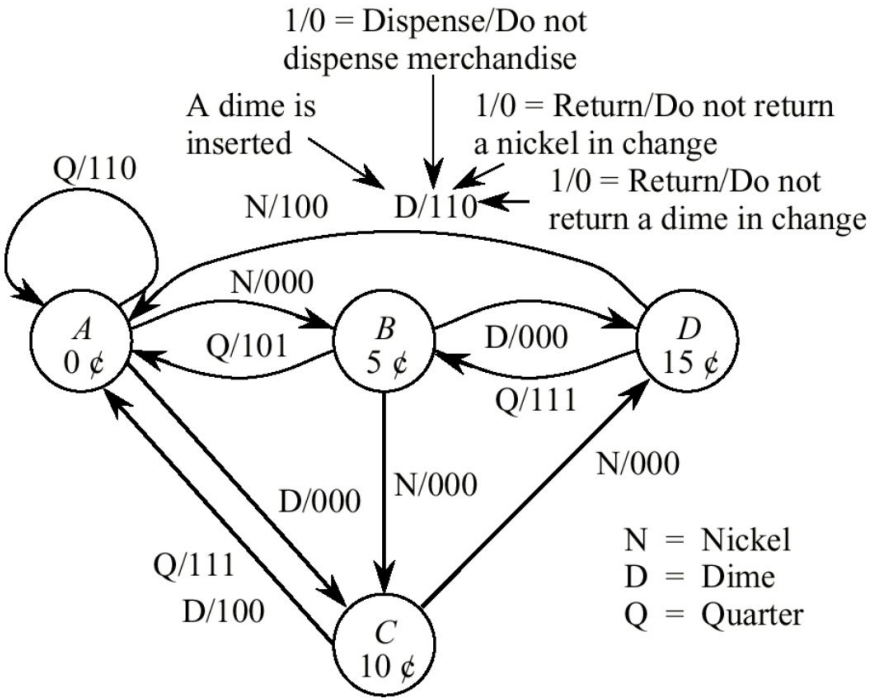
} usually…

- can describe complex systems;
- have a formal definition!

Some Applications of SMs

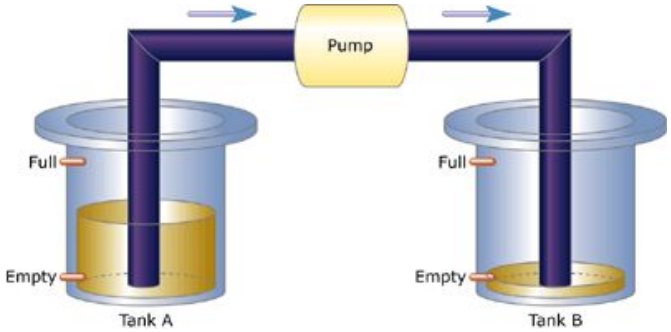
- Embedded systems
- Model checking
- Games
- Object Detection

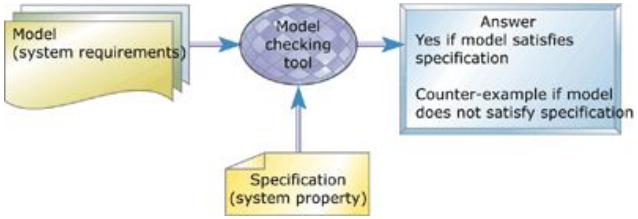
SMs in Action (Embedded)



https://www.csee.umbc.edu/courses/undergraduate/313/Fall03/cpatel2/slides/slides20.pdf

SMs in Action (Model Checking)





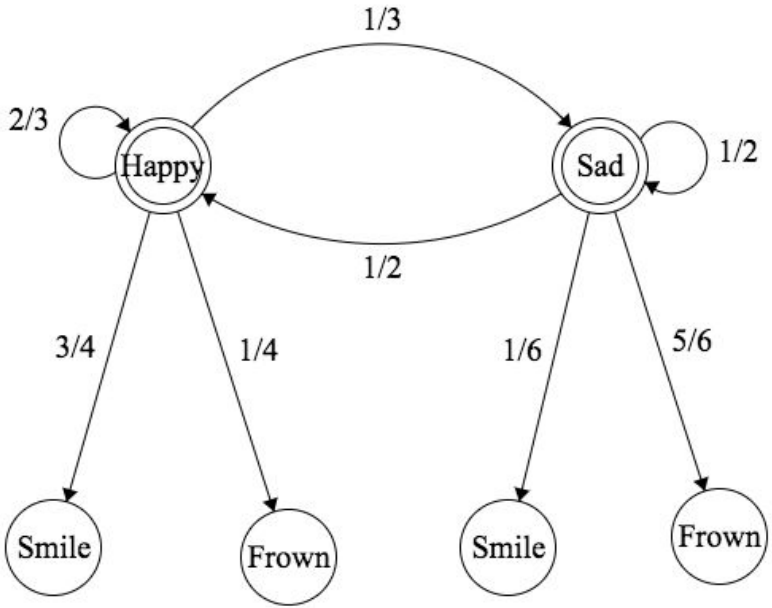
- pump is always off if ground tank is empty or up tank is full
- it is always possible to reach a state when the up tank is ok or full

  - AG AF ((level_a = empty | level_b = full) -> pump = off)
  - AG (EF (level_b = ok | level_b = full))

A = for all
E = exists
F = future
G = always

https://www.embedded.com/an-introduction-to-model-checking/

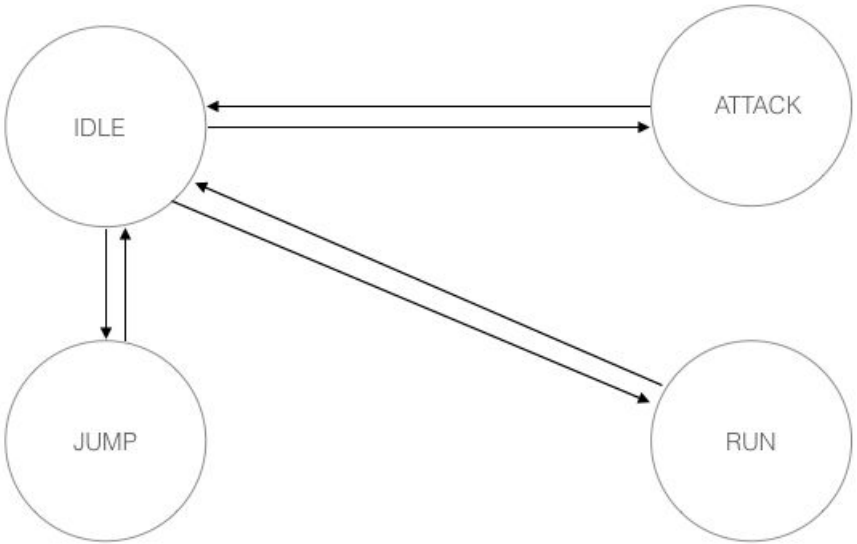
SMs in Action (Probabilistic Model Checking)

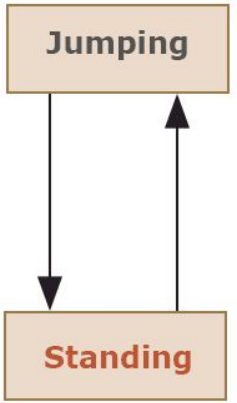


https://bookdown.org/probability/beta/markov-chains.html

This can be used in a lot of areas! https://www.prismmodelchecker.org/casestudies/index.php

SMs in Action (Games)





http://howtomakeanrpg.com/a/state-machines.html

https://gamedevelopertips.com/finite-state-machine-game-developers/

SMs in Action (Object Detection)

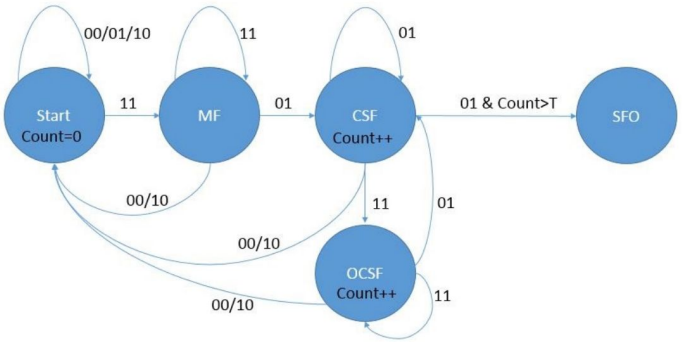


Fig. 3: State diagram of the pixel-based Finite State Machine. MF stands for Moving Foreground, CSF stands for Candidate Static Foreground, OCSF stands for Occluded Static Foreground, SFO stands for Static Foreground Object.

https://ieeexplore.ieee.org/document/8486464

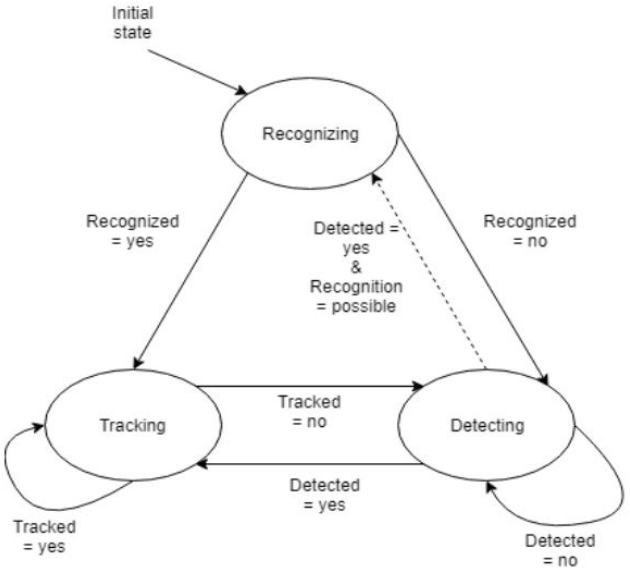


Figure 7. Three-State FSM proposed

http://www.ijeei.org/docs-10175997155e11e583808fb.pdf

See you next time :)

Q & A