Montgomery Village Arduino Meetup

Dec 10, 2016

Making Microcontrollers Multitask

or

How to teach your Arduinos to walk
and chew gum at the same time
(metaphorically speaking)

Background

- My personal project is to prototype a computer-controlled model railroad layout.
- I've started a build blog for this project at `ModelRailroadElectronics.blog`
- A variety of Arduino-class microcontrollers will be used for the different parts of this system.
- Some of these Arduinos will need to be programmed to handle concurrent operations, which simple Arduino programming techniques don't do well.

Some Arduino varieties

Adafruit Motor Shield *

Arduino Uno

Arduino Mega 2560
(in clear case)

Arduino Gemma

Adafruit Trinket

Adafruit Pro Trinket

Arduino Micro

* *Shields* are peripheral boards that can be stacked onto some Arduino boards.

Arduino hardware

board	*power*	*flash (bytes)*	*RAM (bytes)*	*GPIO pins*	*analogWrite pins*	*analogRead pins*	*shields?*
Arduino Gemma	3.3v	8,192	512	3	2	1	no
Adafruit Trinket	3.3v or 5v	8,192	512	5	2	3	no
Adafruit Pro Trinket	3.3v or 5v	8,192	512	18	6	8	no
Arduino Micro	5v	32,768	2,048	18	8	6	no
Arduino Uno	5v	32,768	2,560	20	6	8	yes
Arduino Mega 2560	5v	262,144	8,192	54	15	16	yes

Arduino software

- All Arduinos are programmed with the Arduino IDE in C++.
- Applications written for the Arduino are called *sketches*.
- Software interfaces to hardware are provided by *libraries.*

Arduino software

- All Arduino sketches must have two functions.
  - `void setup() {…}`
    Called once after power-up or reset of board.
  - `void loop() {…}`
    Called repeatedly after setup completes.
- The user application defines the content of these functions.

Blink sketch

- Usually the first sketch a new Arduino user runs is “Blink”. It flashes the onboard LED (usually on pin 13) *about* once every two seconds (1 second on, 1 second off).
- “Blink” uses the built-in `delay()` function to provide the timing of the on and off states.

Blink sketch

```
16
17  // the setup function runs once when you press reset or power the board
18  void setup() {
19    // initialize digital pin 13 as an output.
20    pinMode(13, OUTPUT);
21  }
22
23  // the loop function runs over and over again forever
24  void loop() {
25    digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
26    delay(1000);              // wait for a second
27    digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
28    delay(1000);              // wait for a second
29  }
```

Blink sketch

- During the delays, the Arduino is doing nothing else but marking the passage of time.
- What if we want to drive 2 or more LEDs, but with different timings?
  - Answer: We can't do it using this sketch as a model. We need a different approach... which doesn't rely on `delay()`.

BlinkWithoutDelay, setup

```
24
25  // constants won't change. Used here to set a pin number :
26  const int ledPin =  13;      // the number of the LED pin
27
28  // Variables will change :
29  int ledState = LOW;             // ledState used to set the LED
30
31  // Generally, you should use "unsigned long" for variables that hold time
32  // The value will quickly become too large for an int to store
33  unsigned long previousMillis = 0;        // will store last time LED was updated
34
35  // constants won't change :
36  const long interval = 1000;           // interval at which to blink (milliseconds)
37
38  void setup() {
39    // set the digital pin as output:
40    pinMode(ledPin, OUTPUT);
41  }
42
```

BlinkWithoutDelay, loop

```
42
43  void loop() {
44    // here is where you'd put code that needs to be running all the time.
45
46    // check to see if it's time to blink the LED; that is, if the
47    // difference between the current time and last time you blinked
48    // the LED is bigger than the interval at which you want to
49    // blink the LED.
50    unsigned long currentMillis = millis();
51
52    if (currentMillis - previousMillis >= interval) {
53      // save the last time you blinked the LED
54      previousMillis = currentMillis;
55
56      // if the LED is off turn it on and vice-versa:
57      if (ledState == LOW) {
58        ledState = HIGH;
59      } else {
60        ledState = LOW;
61      }
62
63      // set the LED with the ledState of the variable:
64      digitalWrite(ledPin, ledState);
65    }
66  }
67
```

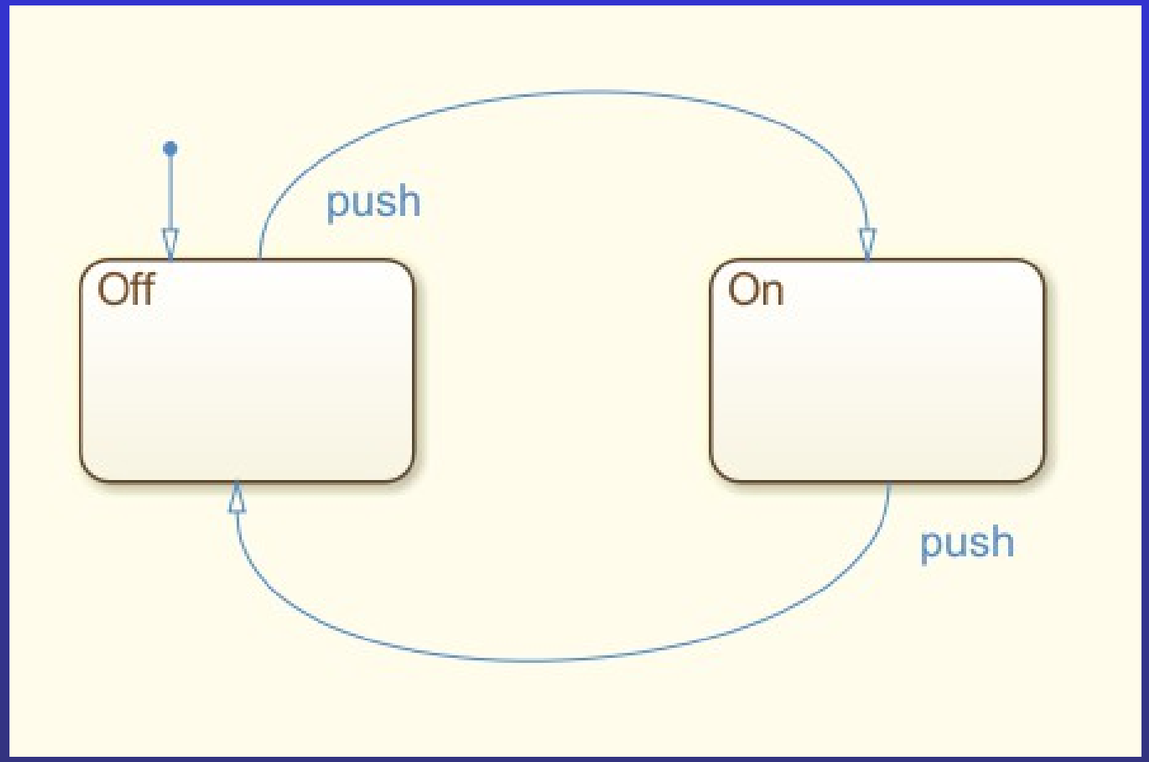
BlinkWithoutDelay sketch

- With this approach, the processor never gets locked up in a `delay()`, and could be modified to allow more than one LED to be flashed, and with different timings.
- We will now skip a lot of steps and jump into...

State machines

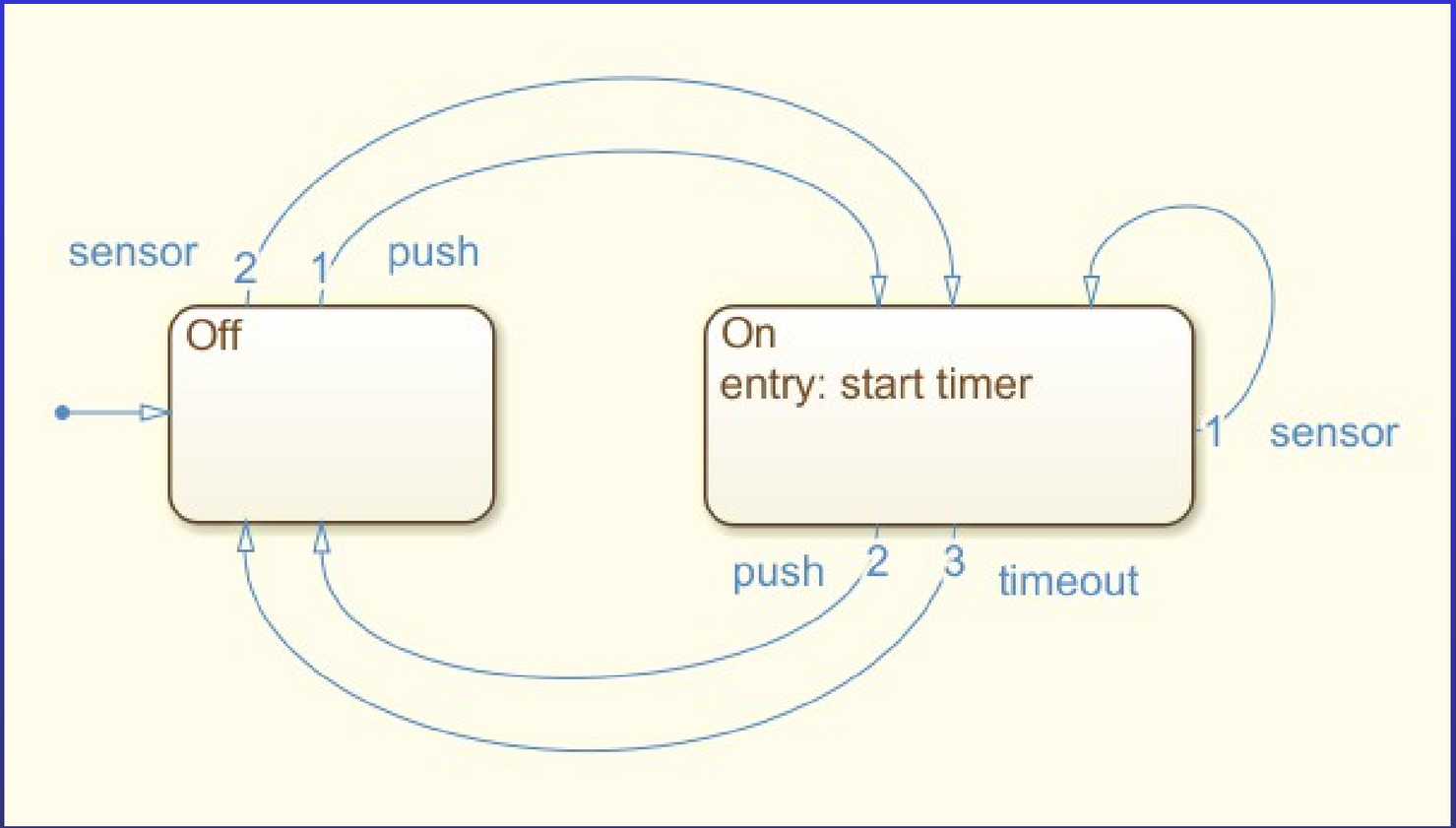
- A *state machine* (full name *finite state machine* or *FSM*) is a mechanism, in this case purely software, which at any given moment is in exactly *one state* out of several possible.
- A state machine has:
  - states
  - actions within states
  - transitions between states

State machine:
Push-on/Push-off switch



- Two states, **Off** and **On**
- Initial state is **Off**
- **Push** action causes change of state

State machine:
Motion-sensor switch



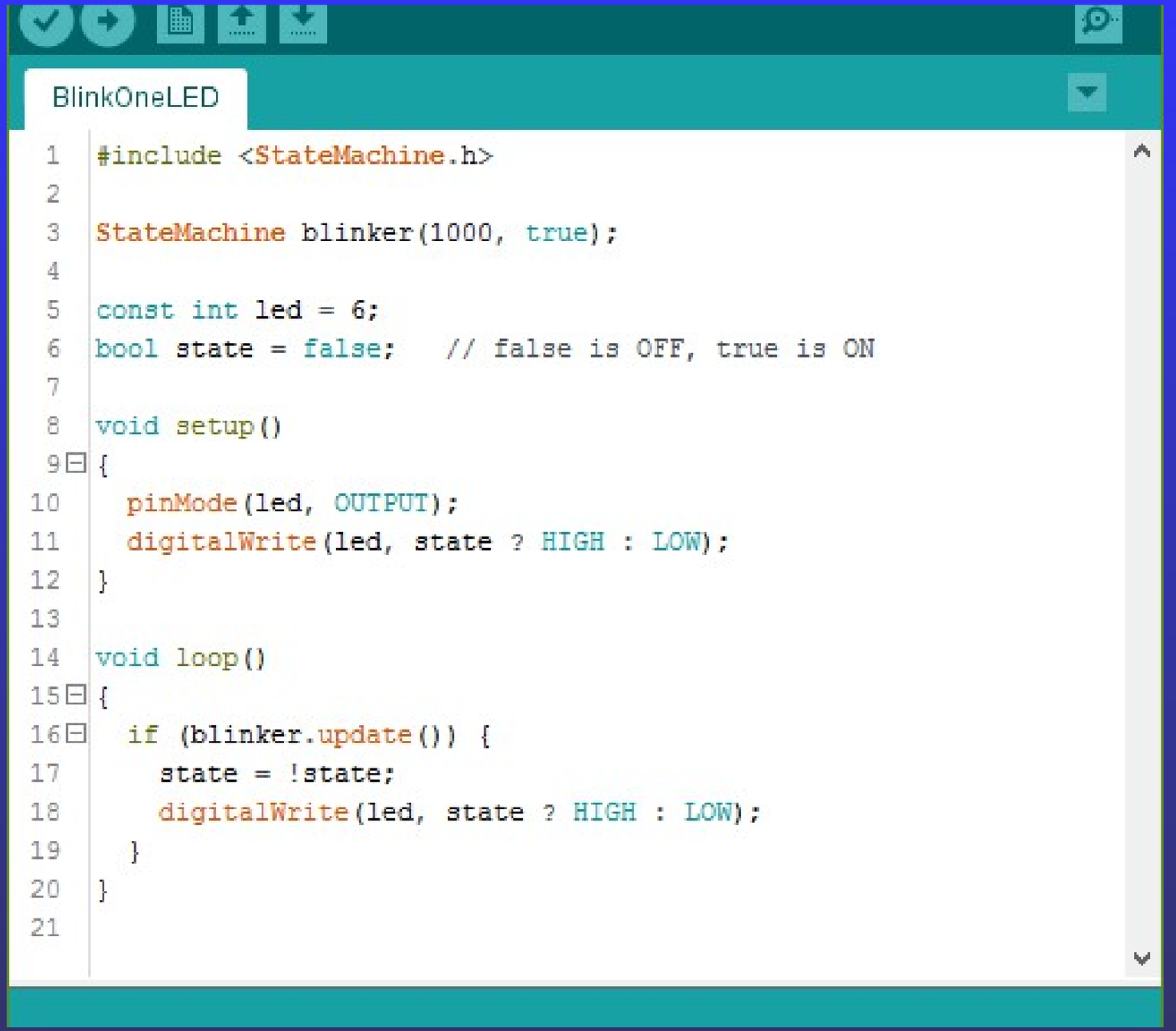
- Sensor will turn switch **On**, starting timer
- Sensor will keep switch **On**, restarting timer
- Timer running out will turn switch **Off**

StateMachine library

- I've encapsulated the framework of a generic state machine into an Arduino library named `StateMachine`.
- This library can be downloaded from my GitHub: github.com/twrackers/StateMachine-library
- Instructions on installing this library and others are at: github.com/twrackers/MyDocuments/blob/master/Installation_to_Arduino.md

BlinkOneLED sketch

- This sketch uses the StateMachine library to flash a single LED, letting a state machine handle the timing while the sketch controls the LED based on the current state.
- This sketch is in the “examples” directory of the StateMachine library.


BlinkOneLED
#include <StateMachine.h>
StateMachine blinker(1000, true);
const int led = 6;
bool state = false;   // false is OFF, true is ON
void setup()
{
  pinMode(led, OUTPUT);
  digitalWrite(led, state ? HIGH : LOW);
}
void loop()
{
  if (blinker.update()) {
    state = !state;
    digitalWrite(led, state ? HIGH : LOW);
  }
}

BlinkThreeLEDs sketch

- This sketch builds upon “BlinkOneLED” to drive three LEDs at the same time, each with its own timing.
- This sketch is in the “examples” directory of the StateMachine library.

BlinkThreeLEDs

```ino
#include <StateMachine.h>

StateMachine blinker1(1000, true);
StateMachine blinker2(1010, true);
StateMachine blinker3(1020, true);

const int led1 = 6;
const int led2 = 5;
const int led3 = 3;

bool state1 = false;   // false is OFF, true is ON
bool state2 = false;   // false is OFF, true is ON
bool state3 = false;   // false is OFF, true is ON

void setup()
{
  pinMode(led1, OUTPUT);
  digitalWrite(led1, state1 ? HIGH : LOW);
  pinMode(led2, OUTPUT);
  digitalWrite(led2, state2 ? HIGH : LOW);
  pinMode(led3, OUTPUT);
  digitalWrite(led3, state3 ? HIGH : LOW);
}

```

```ino
void loop()
{
  if (blinker1.update()) {
    state1 = !state1;
    digitalWrite(led1, state1 ? HIGH : LOW);
  }
  if (blinker2.update()) {
    state2 = !state2;
    digitalWrite(led2, state2 ? HIGH : LOW);
  }
  if (blinker3.update()) {
    state3 = !state3;
    digitalWrite(led3, state3 ? HIGH : LOW);
  }
}

```

- There's a lot of repetition in this sketch, though. Can we improve upon that?
  - Answer: Definitely.

Pulser library

- This library defines a *Pulser* object, which is a StateMachine object with two new parameters:
  - the on-time in milliseconds
  - the off-time in milliseconds
- The Pulser alternates automatically between its on and off states.
- The Pulser stores internally its current state (on or off) and the clock time when it last changed state.
- github.com/twrackers/Pulser-library

LED_Pulser sketch

- This sketch defines a *Strobe* object, which is a Pulser that has a GPIO pin associated with it.
- Now all we need to do is create Strobe objects, each with a pin number, an on-time, and an off-time.
- We could easily create a Strobe library from this sketch.
- github.com/twrackers/Pulser-library/tree/master/Pulser/examples/LED_Pulser

State machine benefits

- By using the StateMachine library, and libraries and classes derived from it, we get the greatest reuse of code, which makes the downloaded code more compact.
- Because our sketch never gets tied up in delay calls, it can remain responsive to external inputs.