State machine with Arduino (Part 1)

Introduction

This paper is to introduce the concept of state machines and how to implement them in an Arduino sketch.

The use of state machines allows us to design easily complex tasks. We will start with simple machines to understand the concepts and the vocabulary of state machines and work our way to more and more challenging projects.

What is a state machine?

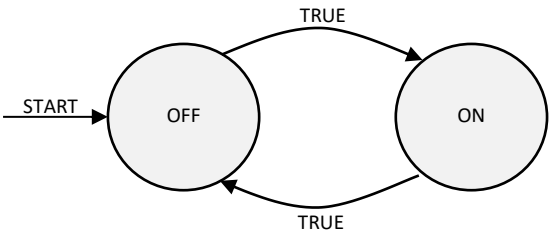
It is a way to describe what a system is designed to do (output) depending of the state is presently in. As an example, we could use an LED. An LED can only be in one of two states: {ON, OFF}. Each state is mutually exclusive of the other. An LED cannot be simultaneously ON and OFF. There has to be one of the states that have to be declared as the start state. We will also have to remember in which state the machine currently is (the current state).


```
enum LedStates{ON, OFF};  //The names of all states
LedStates ledState = ON;  //The start state (and the current state after that)
byte ledPin = 13;
```

Obviously, we want to be able to turn the LED ON or to turn it OFF. To go from a state to another state, we need a transition condition.

Toggling a LED

If we want to have the transition to happen unconditionally, we will write TRUE as the condition to go to the next state.



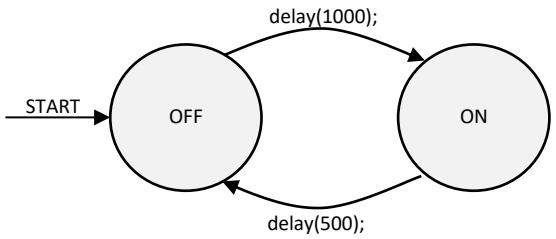

```
void toggleMachine() {
  switch (ledState) {                    //A switch construct will select the proper state
    case OFF: {                            //OFF
      digitalWrite(ledPin, HIGH);            //Bring the pin HIGH
      ledState = ON;                         //Change state
      break;                                 //Leave the switch construct
    }
    case ON : {                            //ON
      digitalWrite(ledPin, LOW);             //Bring the pin LOW
      ledState = OFF;                        //Change state
      break;                                 //Leave the switch construct
    }
  }
}
```

Here is the complete sketch:

```
enum LedStates{ON, OFF};
LedStates ledState = ON;
byte ledPin = 13;

void toggleMachine() {
  switch (ledState) {
    case OFF: {
      digitalWrite(ledPin, HIGH);
      ledState = ON;
      break;
    }
    case ON: {
      digitalWrite(ledPin, LOW);
      ledState = OFF;
      break;
    }
  }
}

void setup() {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);
}

void loop() {
  toggleMachine();
}
```

Blink with delay

With the previous example, the LED blinks so fast that we can't see it blinking. We will add a delay of 1 second (1000 milliseconds) as the condition to switch from the OFF state to the ON state, and will add a delay of ½ second (500 milliseconds) as the condition to switch from the ON to OFF state. The changes in the code are showed in blue.




```
void blinkMachine() {
  switch (ledState) {
    case OFF: {
      delay(1000);                    //A condition to get out of the state
      digitalWrite(ledPin, HIGH);     //Action to take on state change
      ledState = ON;                  //Change state
      break;
    }
    case ON : {
      delay(500);
      digitalWrite(ledPin, LOW);
      ledState = OFF;
      break;
    }
  }
}
```

Blink without delay

Our blink state machine is not as efficient as it could be because we use delay().

Inside Arduino, there is an internal clock that counts the number of milliseconds that have passed since the start of the sketch. This value is available using **`millis().`** It is just like asking the time.

With **`delay(1000)`** the processor does is:

- First, take note of how many milliseconds is showing on the internal clock.
- Keep looking at the clock until its value minus what was read in the previous step equals 1000.
- Terminate.

We don't have to let the processor sit idle looking at the internal clock for an entire second, doing nothing else. We can do it another way. We will use the global variable "chrono" to remember when the delay started.

Notice that our "chrono" variable is of type unsigned long. This is because it is what **`millis()`** returns. Using this type of variable, the internal clock can count up to 4,294,967,295 milliseconds. That is almost 50 days. At that point, the clock will revert to 0 and start all over again.

```
unsigned long chrono = millis();

void loop() {
  if (millis() - chrono >= 1000) {
    chrono = millis();
    // Do something special
  }
  //Do something else in the meantime
}
```

On each loop, we do exactly what **`delay(1000)`** did. We look at the internal clock and see if the difference between that and when the delay started is greater than or equal to 1000. If it is, then we can do our *something special* thing. But now, we are free to do other things in the meantime in the rest of our sketch.

Depending on the complexity of the code in the loop (including the *do something in the meantime* code), the frequency at which the internal clock will be read can vary. The actual delay may be a bit longer than what we asked for, but by so little, that for most occasions, it will not matter. Notice that we used "**greater than or equal to"** as our logical operator. This is to make sure that we will react at the end of the delay even if it is a bit passed the time.

Notice that the first thing that we did when the count reached 1000, was to reset "chrono" to the current time. Doing so makes sure that our special code will be executed every 1000 milliseconds.

Here is the complete sketch with the changes in blue:

```
enum LedStates{ON, OFF};
LedStates ledState = ON;
byte ledPin = 13;
int delayOff = 1000;
int delayOn = 500;
unsigned long chrono = millis();

void blinkMachine() {
  switch (ledState) {
    case OFF: {
      if (millis() - chrono >= delayOff) {
        chrono = millis();
        digitalWrite(ledPin, HIGH);
        ledState = ON;
      }
      break;
    }
    case ON: {
      if (millis() - chrono >= delayOn) {
        chrono = millis();
        digitalWrite(ledPin, LOW);
        ledState = OFF;
      }
      break;
    }
  }
}

void setup() {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);
}

void loop() {
  blinkMachine();
}
```

Putting on a light show with 8 LEDs

Now, we will use tables to hold the values needed to blink all the 8 LEDs.

Tables are declared this way:
variableType variableName[numberOfElements] = {initialValue, initialValue,… ,initialValue};

```
byte ledPin[8] = {2, 3, 4, 5, 6, 7, 8, 9};
```

To access any of those elements, we use the the tables's name, followed, in brackets, by its index. Indexes start a **0** and go to the **number of elements – 1:**
ledPin[0] contains 2
ledpin[1] contains 3
…
ledPin[6] contains 8
ledPin[7] contains 9

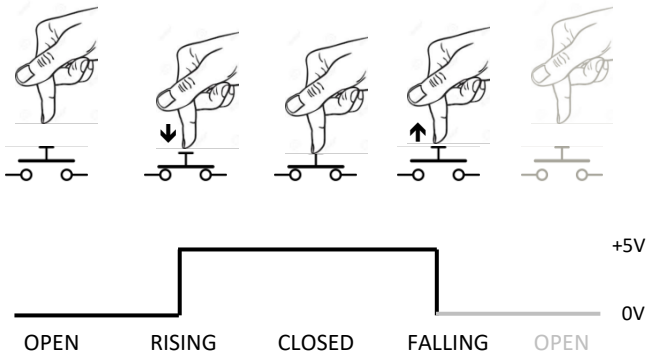
and ledPin[8] contains garbage. If you do that, you are reading outside of the array.

With arrays, our sketch looks like this:

```
enum LedStates{ON, OFF};
LedStates ledState[8] = {ON, ON, ON, ON, ON, ON, ON, ON};
byte ledPin[8] = {2, 3, 4, 5, 6, 7, 8, 9};
int delayOff[8] = {1000, 900, 800, 700, 600, 500, 400, 300};
int delayOn[8] = {500, 450, 400, 350, 300, 250, 200, 150};
unsigned long chrono[8];

void blinkMachine(byte i) {
  switch (ledState[i]) {
    case OFF: {
      if (millis() - chrono[i] >= delayOff[i]) {
        chrono[i] = millis();
        digitalWrite(ledPin[i], HIGH);
        ledState[i] = ON;
      }
      break;
    }
    case ON: {
      if (millis() - chrono[i] >= delayOn[i]) {
        chrono[i] = millis();
        digitalWrite(ledPin[i], LOW);
        ledState[i] = OFF;
      }
    break;
    }
  }
}

void setup() {
  for (byte i = 0 ; i < 8 ; i++) {
   pinMode(ledPin[i], OUTPUT);
   digitalWrite(ledPin[i], HIGH); }
}

void loop() {
  for (int i = 0 ; i < 8 ; i++) blinkMachine(i);
}
```

Reading a switch

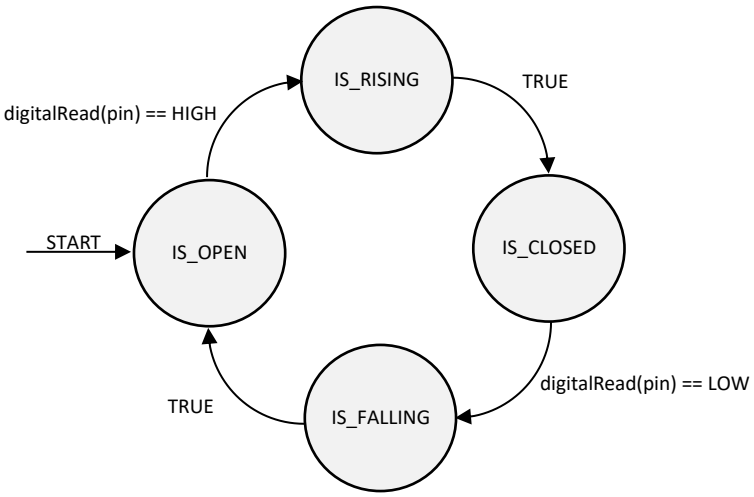
There are four states that a switch can be in:



Here, we have a normally open switch. Working from left to right:

- The switch is not pressed, and the pin is LOW. **It is OPEN**
- The user presses the switch and the pin goes from LOW to HIGH. **It is RISING**
- The user keeps the switch pressed and the pin is HIGH. **It is CLOSED**
- The user releases the switch and the pin goes from HIGH to LOW. **It is FALLING**
- The switch is not pressed again and the pin is LOW. **It is OPEN again**.

This is can be made into a state machine with this schematic:



NOTE: If the switch is connected to Arduino using its internal input pullup resistor, the switch reads HIGH when not pressed and LOW when it is pressed. We will take it into account for our switch state machine, inverting what we have read at the pin if we have a switch with a pullup resistor.

As we did with the LED, we will define the machine's states and assign it a variable to its start and current state.

```
enum SwitchStates {IS_OPEN, IS_RISING, IS_CLOSED, IS_FALLING};
SwitchStates switchState = IS_OPEN;
byte switchPin = 10;

enum SwitchModes {PULLUP, PULLDOWN};
SwitchModes switchMode = PULLDOWN;
```

We will use the switchMode in the setup()

```
void setup() {
  if (switchMode == PULLDOWN) pinMode(switchPin, INPUT);
  else                        pinMode(switchPin, INPUT_PULLUP);
}
```

Making the state machine is straight forward. We just have to follow our schematic.

```
void switchMachine() {
  byte pinIs = digitalRead(switchPin);
  if (switchMode == PULLUP) pinIs = !pinIs;
  switch (switchState) {
    case IS_OPEN:    { if(pinIs == HIGH) switchState = IS_RISING;  break; }
    case IS_RISING:  {                   switchState = IS_CLOSED;  break; }
    case IS_CLOSED:  { if(pinIs == LOW)  switchState = IS_FALLING; break; }
    case IS_FALLING: {                   switchState = IS_OPEN;    break; }
  }
}
```

Often, we just want to know if a switch has been pressed to trigger an event. The user presses the switch, and then releases it. It is called a click. For this to happen, the switch (originally in the OPEN state) will go to the RISING state as the user presses it, then will go immediately in the CLOSED state and stay there as long as the user still has the switch pressed. When the user releases the switch, it will go in the FALLING state. This is exactly when we want to take action.

In the loop(), we could write:

```
void loop() {
  switchMachine();
  if(switchState == IS_FALLING) {
  //Do something
  }
}
```

NOTE: If the switches' state is used to trigger other events, do not evoke the switchMachine again in the same loop as it would make it go to the next state (OPEN). Just use the "if" construct.

The complete sketch:

```
enum SwitchStates {IS_OPEN, IS_RISING, IS_CLOSED, IS_FALLING};
SwitchStates switchState = IS_OPEN;
byte switchPin = 2;

enum SwitchModes {PULLUP, PULLDOWN};
SwitchModes switchMode = PULLDOWN;

void switchMachine() {
  byte pinIs = digitalRead(switchPin);
  if (switchMode == PULLUP) pinIs = !pinIs;
  switch (switchState) {
    case IS_OPEN:    { if(pinIs == HIGH) switchState = IS_RISING;  break; }
    case IS_RISING:  {                   switchState = IS_CLOSED;  break; }
    case IS_CLOSED:  { if(pinIs == LOW)  switchState = IS_FALLING; break; }
    case IS_FALLING: {                   switchState = IS_OPEN;    break; }
  }
}

void setup() {
  if (switchMode == PULLDOWN) pinMode(switchPin, INPUT);
  else                        pinMode(switchPin, INPUT_PULLUP);
}

void loop() {
  switchMachine();
  if(switchState == IS_FALLING) {
  //Do something
  }
}
```

Making the machine do things

Normally, the state machine is probed once per loop. We could do something every time the machine is probed, like reading some sensors or the like. Moreover, when the machine is in a specific state, we can ask it to do things when:

- entering the state;
- every time the state is probed;
- exiting the state;

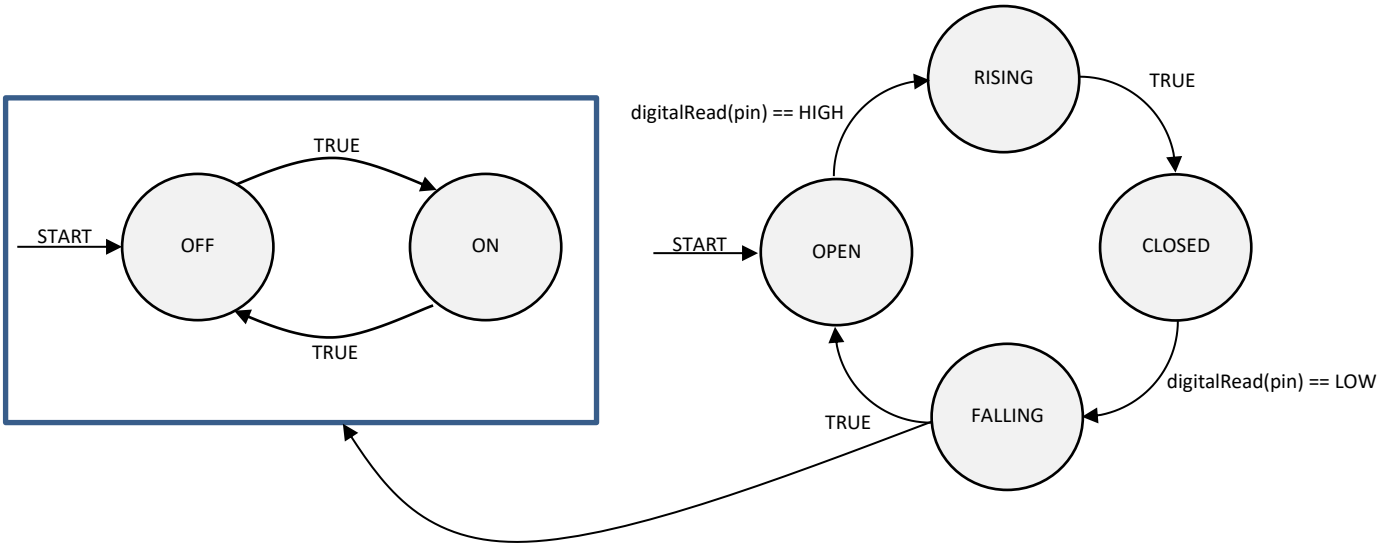
With the blinkMachine, when exiting the OFF state, we wrote HIGH to the pin and when exiting the ON state, we wrote LOW to the pin. Here is some pseudocode to show one way to implement all the possible combinations.

```
machineEntry() {
  Do: on every machine probe code (read sensors…)
  Switch (state) {
    Case A: {
      static bool entering = true;
      if (entering) { Do: entering this state code; entering = false;}
      Do: whenever probed in this state code;
      if (exit condition == true) {
        Do: exiting this state code;
        entering = true;
        set next state;
      }
    }
  }
}
```

For our next sketch, we will have two state machines interact. This time, instead of acting on its own machine part, one machine will call upon another machine when exiting one of its states.

Making a new state machine using two state machines

We will use our switchMachine and the toggleMachine that we designed in the beginning of this tutorial. We will toggle a LED with a switch.



The switchMachine will call the ToggleMachine when exiting its FALLING state.

The global part:

```
enum LedStates{ON, OFF};
LedStates ledState = ON;
byte ledPin = 13;

enum SwitchStates {IS_OPEN, IS_RISING, IS_CLOSED, IS_FALLING};
SwitchStates switchState = IS_OPEN;
byte switchPin = 10;

enum SwitchModes {PULLUP, PULLDOWN};
SwitchModes switchMode = PULLUP;
```

The toggleMachine (The one we designed in the beginning of this tutorial):

```
void toggleMachine() {
  switch (ledState) {
    case OFF: { digitalWrite(ledPin, HIGH); ledState = ON;  break; }
    case ON : { digitalWrite(ledPin, LOW);  ledState = OFF; break; }
  }
}
```

The switch machine (Notice the change in the IS_FALLING state):

```
void switchMachine() {
  byte pinIs = digitalRead(switchPin);
  if (switchMode == PULLUP) pinIs = !pinIs;
  switch (switchState) {
    case IS_OPEN:    { if(pinIs == HIGH) switchState = IS_RISING;  break; }
    case IS_RISING:  {                   switchState = IS_CLOSED;  break; }
    case IS_CLOSED:  { if(pinIs == LOW)  switchState = IS_FALLING; break; }
    case IS_FALLING: { toggleMachine();  switchState = IS_OPEN;    break; }
  }
}
```

The setup:

```
void setup() {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);
  if (switchMode == PULLDOWN) pinMode(switchPin, INPUT);
  else                        pinMode(switchPin, INPUT_PULLUP);
}
```

The loop:

```
void loop() {
  switchMachine();
}
```

About debouncing

Whenever we read a switch, we should never trust digitalRead(). When the switch is pressed or released, the pin goes between HIGH and LOW frantically for a certain amount of time before settling to its new state. It is called bouncing. This is why a single read of the pin is insufficient. If you tried the previous sketch, you might have noticed that most of the time it works, but sometimes it doesn't. That is due to bouncing.

There are many ways to debounce a switch. It can be done with hardware (using a capacitor or a capacitor and a resistor, or a capacitor and two resistors, or…). It can also be done in software.

The most popular way to do it with Arduino aficionados in software is:

```
digitalRead(pin)
if (it is not the same as before, "then the pin IS_RISING or IS_FALLING" {
  use the blink without delay code to wait a little bit;
  read the pin again until "little bit passed";
  if the result is the same, then "the switch really have changed state and return the new state"
}
```

This algorithm has to be applied to each and every switch that our setup counts.

There are some debounce libraries out there that makes it easy to debounce switches without you having to add this (or any other) code to every single switch in your setup. Any debounce library will do the job. Save yourself the grief of such a trivial task.

I happen to have written one that is called "EdgeDebounceLite". And, as a plus, it does also filter Electromagnetic field interferences. You can download it here: https://github.com/j-bellavance/EdgeDebounceLite.

In the beginning of the sketch we write:

```
#include <EdgeDebounceLite.h>
EdgeDebounceLite debounce;
```

Now, instead of using:

```
aVariable = digitalRead(pin);
```

We use:

```
aVariable = debounce.pin(pin);
```

Making a new state machine using two state machines… debounced

```
#include <EdgeDebounceLite.h>
EdgeDebounceLite debounce;

enum LedStates{ON, OFF};
LedStates ledState = ON;
byte ledPin = 13;

enum SwitchStates {IS_OPEN, IS_RISING, IS_CLOSED, IS_FALLING};
SwitchStates switchState = IS_OPEN;
byte switchPin = 10;

enum SwitchModes {PULLUP, PULLDOWN};
SwitchModes switchMode = PULLUP;

void toggleMachine() {
  switch (ledState) {
    case OFF: { digitalWrite(ledPin, HIGH); ledState = ON;  break; }
    case ON : { digitalWrite(ledPin, LOW);  ledState = OFF; break; }
  }
}

void switchMachine() {
  byte pinIs = debounce.pin(switchPin);
  if (switchMode == PULLUP) pinIs = !pinIs;
  switch (switchState) {
    case IS_OPEN:    { if(pinIs == HIGH) switchState = IS_RISING;  break; }
    case IS_RISING:  {                   switchState = IS_CLOSED;  break; }
    case IS_CLOSED:  { if(pinIs == LOW)  switchState = IS_FALLING; break; }
    case IS_FALLING: { toggleMachine();  switchState = IS_OPEN;    break; }
  }
}

void setup() {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);
  if (switchMode == PULLDOWN) pinMode(switchPin, INPUT);
  else                        pinMode(switchPin, INPUT_PULLUP);
}

void loop() {
  switchMachine();
}
```

Now it works every time.

In part 2, we will create a vending machine state machine.