State machine with Arduino (Part 1)

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

The use of a state machine allows us to design easily more and more complex tasks. We will start with very simple machines and work our way into more challenging projects.

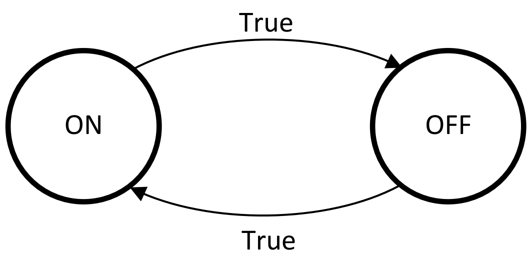
What is a state machine?

It is a way to describe what something does, depending on what state it is presently. As an example, we could use a LED. A LED can only be in one of two states: ON or OFF. Each state is mutually exclusive of the other (a LED cannot be simultaneously ON and OFF.

Obviously, we would want to turn the LED ON or OFF at will. To do this, we use transitions. We will use the first Sketch that is used to present Arduino: Blink. Before we start writing the sketch, we will make a diagram of our state machine:

The Toggle state machine diagram

We will start by creating a toggle state machine diagram.



With the two circles, we see that the LED has two states: ON and OFF. The arrows show the transitions between each state. True means that there is no condition. When in state ON, transition to state OFF. When in state OFF, transition to state ON.

Before starting to write a Sketch, we will always sit down with pen and paper and draw our diagram.

To remember in which state the machine currently is, and other stuff, we will add attributes to the machine.

Translating the diagram into an Arduino Sketch.

First, we have to decide if the state machine's attributes will be in global variables that can be seen by all parts of the sketch, or if they will be stored inside the machine, hidden away from the rest of the sketch. If there is only one machine that will be active, it makes sense to hide all the attributes inside the machine. If we know that there will be multiple instances of the same machine, it is better to make them global, so we can use the same code with all the instances.

We will name the LED's states and name the pin that the LED is attached to:

```
enum LedStates {ON, OFF};        //Give names to the LED's states
LedStates ledState = ON;         //The current state
byte ledPin = 13                 //The LED's pin
```

In the setup() section of the sketch, we will prepare the digital pin 13 for output:

```
void setup() {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);  //We declared ledState = ON
}
```

We need to create a new state machine, using a switch construct. The switch/case construct allows us to specify what is to be done when the machine is in that state and ignore the other states.

Each state may contain:

- instructions to be followed on entry;
- verification of conditions to :
    - remain in the same state;
    - leave to a new state;
- instructions to be followed on exit;

```
void toggle() {
  switch (ledState) {                 //Depending on the current state
   case ON: {                           //ON
                                          //On entry: Nothing to do
                                          //Condition to stay: None
                                          //Condition to leave: True -> None
     digitalWrite(ledPin, LOW);           //On exit: turn the LED OFF
     ledState = OFF;                      //Change the state to OFF
     break;                               //Get out of the switch construct
   }
   case OFF: {                          //OFF
                                          //On entry: Nothing to do
                                          //Condition to stay: None
                                          //Condition to leave: True -> None
     digitalWrite(ledPin, HIGH);          //On exit: turn LED ON
     ledState = ON;                       //Change the state to ON
     break;                               //Get out of the switch construct
   }
  }
}
```

Making it shorter:

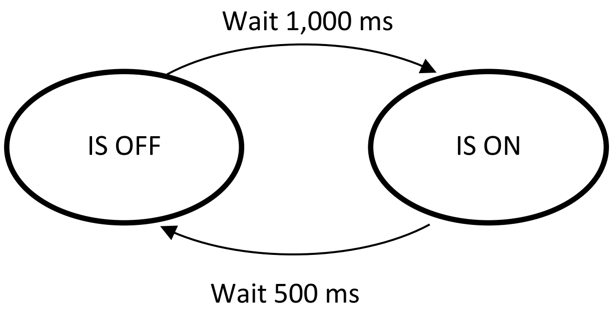
```
void toggle() {
  switch (ledState) {
    case ON  : { digitalWrite(ledPin, LOW);  ledState = OFF; break; }
    case OFF : { digitalWrite(ledPin, HIGH); ledState = ON;  break; }
  }
}
```

In the loop section, we just need call toggle():

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

The Blink state machine

Let's say that we want to have the LED to stay on 1000ms, and to stay off 500ms. We would change our diagram to look like this:



Before the setup() part of the sketch. we will define the two states and the two delays:

```
enum BlinkStates{IS_OFF, IS_ ON};   //Give names to the blink's states
BlinkStates blinkState = IS_ON
int offDelay = 1000;
int onDelay = 500;
byte ledPin = 13
```

The setup() section remains the same.

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

This is what our blink state machine looks like:

```
void blink() {
  switch (currentBlinkState) {             //Depending on the current state:
    case IS_ON: {                            //IS_ON
      delay(onDelay);                          //delay (from on to off delay)
      digitalWrite(ledPin, LOW);               //Turn off the LED
      currentBlinkState = IS_OFF;              //Change state to IS_OFF
      break;
    }
    case IS_OFF: {                           //IS_OFF
      delay(offDelay);                         //delay (from off to on delay)
      digitalWrite(ledPin, HIGH);              //Turn on the LED
      currentBlinkState = IS_ON;               //Change state to IS_ON
      break;
    }
  }
}
```

In the setup() section of the sketch, we will prepare the digital pin 13 for output:

```
void setup() {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);  //We declared ledState = ON
}
```

The loop() section just calls blink()

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

Having a blink state machine that can blink any pin, at two different rates

We will modify our blink state machine so that it can blink any pin. We will add parameters to the machine; so that when we call it, we can specify how it should behave:

The variable declaration remains the same.

```
enum BlinkStates{IS_OFF, IS_ ON};   //Give names to the blink's states
BlinkStates blinkState = IS_ON;
int onDelay = 1000
int offDelay = 500;
byte ledPin = 13
```

Modifying the blink() state machine so it can blink any pin, at two different intervals, using parameters.

```
void blink(byte pin, int delayOn, int delayOff) {
  switch (blinkState) {
    case IS_ON: {
      delay(delayOn);
      digitalWrite(pin, LOW);
      currentBlinkState = IS_OFF;
      break;
    }
    case IS_OFF: {
      delay(delayOff);
      digitalWrite(pin, HIGH);
      currentBlinkState = IS_ON;
      break;
    }
  }
}
```

In the setup() section of the sketch, we will prepare the digital pin 13 for output:

```
void setup() {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);  //We declared ledState = ON
}
```

The loop() section just calls blink() with the proper parameters:

```
void loop() {
  blink(ledPin, onDelay, offDelay);
}
```

Blink without delay()

Our blink() method has a serious flaw. It has the processor going idle for 99.99% of the time doing it's delay() thing. That prevents the sketch to do other tasks in the meantime.

To free the processor to do other things, we will add another attribute to our blink machine. A chronometer. Arduino has an internal clock that counts the number of milliseconds elapsed since the Sketch started. We will use that clock to count the milliseconds between state changes instead of delay(). A call to millis() is the same as asking time.

Suppose that we want to have a state that's duration is 1000ms. We could have an attribute called "chrono", and ask it to be equal to Arduino's clock (Same as setting chrono to "now"), and let it be our "Start counting signal". We then let Arduino do other things until some point where the difference between Arduino's clock and our "Start counting signal" goes equal or beyond 1000ms.

That, in Arduino's terms would be:

First, we will set "chrono" to remember what the time at the beginning of the delay is:

```
chrono = millis();
```

To find out if we are at or beyond that limit, we will use the "if" construct. (There may be some times that, at the moment that we ask, the delay could have already been passed, so we don't take any chances and also check if it is beyond the limit.)

Is the time difference between right now "millis()" and when we set "chrono" equal to or greater than a 1000?:

```
if(millis() - chrono) >= 1000) {
  //do something
}
```

The first thing that we will do when it is time to toggle the pin, is to reset our chrono to be ready for the next blink.

```
if(millis() - chrono) >= 1000) {
  chrono = millis(); //reset chrono
  //do something else
}
```

Now, we will modify our blink() method to be super-efficient, requiring only millionths of a second to do its job instead of the 1½ seconds it used to.

NOTE : Unlike standard longs, unsigned longs won't store negative numbers, making their range from 0 to 4,294,967,295. This allows the millis() function to go on for almost 50 days before rolling back to 0. Since millis() returns an unsigned long value, chrono will also have to be an unsigned long.

The variable declaration has a new attribute:

```
enum BlinkStates{IS_OFF, IS_ ON};   //Give names to the blink's states
BlinkStates blinkState = IS_ON;
unsigned long chrono;
int onDelay = 1000
int offDelay = 500;
byte ledPin = 13
```

We will modify the blink() state machine as follows:

```
void blink(byte pin, int delayOn, int delayOff) {
  switch (blinkState) {
    case IS_ON: {
      if (millis() - chrono >= delayOn {
        chrono = millis();
        digitalWrite(pin, LOW);
        currentBlinkState = IS_OFF;
      }
      break;
    }
    case IS_OFF: {
      if (millis() - chrono >= delayOff {
        chrono = millis();
        digitalWrite(pin, HIGH);
        currentBlinkState = IS_ON;
      }
      break;
    }
  }
}
```

In the setup() section remains the same:

```
void setup() {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);  //We declared ledState = ON
}
```

The loop() section remains the same:

```
void loop() {
  blink(ledPin, onDelay, offDelay);
}
```

Putting on a light show with 8 LEDs

Now, we want to use 8 LEDs. The use of parameters will make our sketch cluttered. Instead, we will use tables.

Tables are declared this way:

variableType variableName[numberOfElements] = {initialValue, initialValue,…, initialValue};

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

To access any element, we use the table's name, followed by its index. Indexes start from 0 and go to the number of elements minus 1: pins[0], pin[1], pins[2], pins[3], pins[4], pins[5], pins[6] and pins[7].

The sketch will look like this:

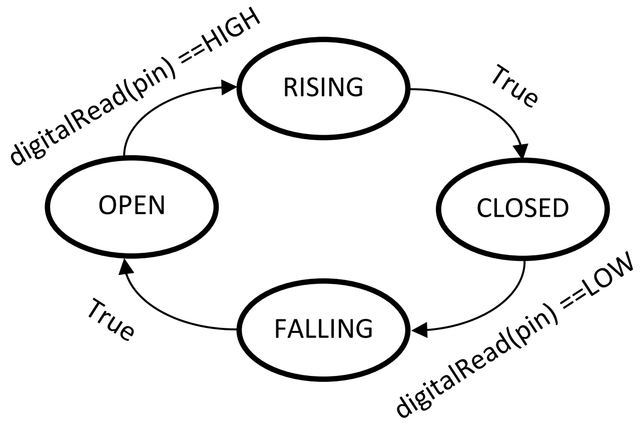
```
enum BlinkStates {IS_ON, IS_OFF};
BlinkStates blinkStates[8] = {IS_ON, IS_ON, IS_ON, IS_ON, IS_ON, IS_ON, IS_ON, IS_ON}
byte ledPins[8] = {2, 3, 4, 5, 6 ,7, 8, 9};
int onDelays[8] = {1000, 900, 800, 700, 600, 500, 400, 300};
int offDelays[8] = {500, 450, 400, 350, 300, 250, 200, 150};
unsigned long chronos[8];

void blink(byte i) {
  switch (blinkStates[i]) {
    case IS_ON: {
      if (millis() - chronos[i] >= onDelays[i] {
        chronos[i] = millis();
        digitalWrite(ledPins[i], LOW);
        blinkStates[i] = IS_OFF;
      }
      break;
    }
    case IS_OFF: {
      if (millis() - chronos[i] >= offDelays[i] {
        chronos[i] = millis();
        digitalWrite(ledPins[i], HIGH);
        blinkStates[i] = IS_ON;
      }
      break;
    }
  }
}

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

void loop() {
  for (byte i = 0 ; i < 8 ; i++) {
    blink(i);
  }
}
```

Reading a switch



Here, we have a normally open switch that is connected to the Arduino. A normally open switch conducts current only when it is pressed. Let's examine the four states of that switch.

- At the start, it is in the state OPEN (digitalRead(pin) returns LOW).
- When (digitalRead(pin) == HIGH), (just went from LOW to HIGH) it means that it is no longer OPEN. We will call that state RISING
- Follows immediately the CLOSED state (was HIGH, and stays HIGH) it is not rising anymore.
- When (digitalRead(pin) == LOW), (just went from HIGH to LOW) it means that it is no longer CLOSED. We will call that state FALLING
- Follows immediately the OPEN state again (was LOW and stays LOW) it is not falling anymore

NOTE: If the switch is connected to Arduino using its internal pullup resistor, then going from OPEN state to RISING state would happen when (digitalRead(pin) == LOW) and going from CLOSED state to FALLING state would happen when (digitalRead(pin)==HIGH).

Before the setup() we will write:

```
#define PULLUP 0
#define PULLDOWN 1
byte switchPin = 2;
byte switchMode = PULLDOWN;
enum switchStates {OPEN, RISING, CLOSED, FALLING};
switchStates switchCurrentState;
```

In the setup() we will write:

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

We can now create the readSwitch() state machine

```
void readSwitch() {
  byte switchStatus = digitalRead(switchPin);
  if (switchMode == PULLUP) switchStatus = !switchStatus;
  switch (switchCurrentState) {
    case OPEN: { if(switchStatus == HIGH) switchCurrentState = RISING; break; }
    case RISING: { switchCurrentState = CLOSED; break; }
    case CLOSED: { if(switchStatus) == LOW) switchCurrentState = FALLING; break;}
    case FALLING: { switchCurrentState = OPEN; break; }
  }
}
```

We can see that going from the diagram to Arduino code is done in a straightforward way.

Here, when we change the status of the switch, we have nothing special to do. We just have to see if the transition condition is met, and just change to the next status if it is.

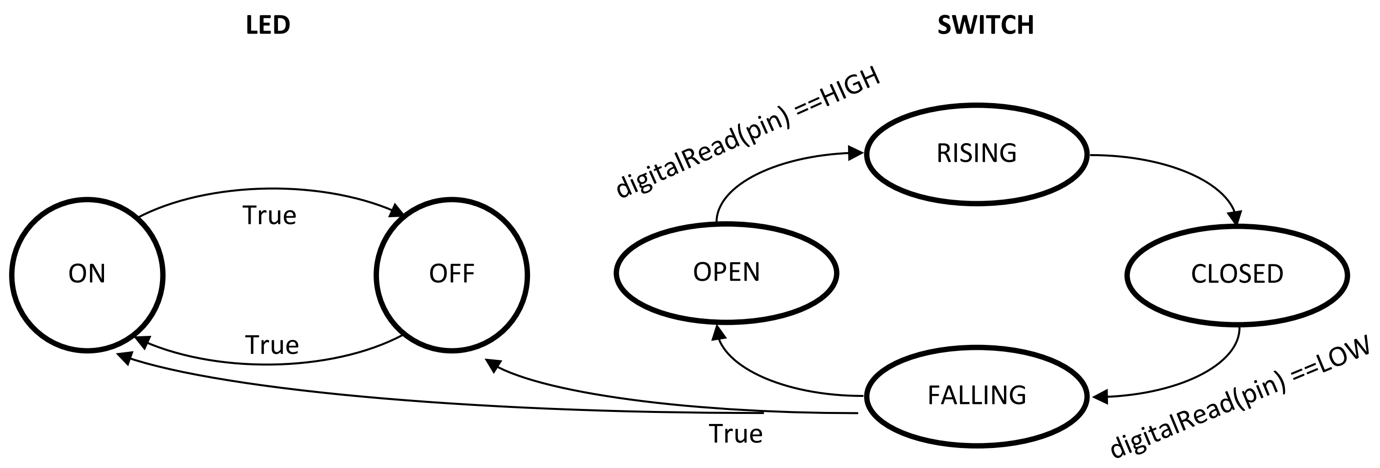
Very often, we just want to know if a switch has been pressed to trigger an event. The user presses the switch then releases it. That is what is often referred to as a click. For this to happen, the switch (originally in a OPEN state) will go to the RISING state when the user presses the switch, then go and stay in the CLOSED state until the user releases the switch, at which point it will be in the FALLING state. This is exactly when we want to take action.

In the loop(), we could write:

```
void loop() {
  readSwitch();
  if(switchCurrentState == FALLING) {
    //Do something
  }
}
```

NOTE: Of course, if the switch's state is used to trigger other events, do not invoke “readSwitch()” again in the same loop, as it would bring it in the next state (OPEN). Just use the “if” construct.

Using the switch to toggle a LED



We want to toggle the LED whenever we click the switch.

We need:

- a toggle state machine,
- a Switch state machine, that we will modify to send the LED state machine that it is time to toggle.

Before the setup() part of the sketch, we need to create the attributes for the two state machines:

```
ledPin = 13;
enum ledStates {ON, OFF};
ledStates ledCurrentState;
switchPin = 2;
switchMode = PULLUP;
enum switchStates {OPEN, RISING, CLOSED, FALLING};
switchStates switchCurrentState = OPEN;
```

The ToggleLed state machine is the same as the toggle() state machine in the beginning of this tutorial:

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

We will modify the FALLING case of the switch state machine to toggle the LEDs:

```
void readSwitch() {
  byte switchStatus = digitalRead(switchPin);
  if (switchMode == PULLUP) switchStatus = !switchStatus;
  switch (switchCurrentState) {
    case OPEN: { if(switchStatus == HIGH) switchCurrentState = RISING; break; }
    case RISING: { switchCurrentState = CLOSED; break; }
    case CLOSED: { if(switchStatus) == LOW) switchCurrentState = FALLING; break;}
    case FALLING: { toggleLed(); switchCurrentState = OPEN; break; }
  }
}
```

The loop() will contain this:

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

About debouncing

We should never use digitalRead() alone. This function only reads the pin once. Switches bounce as they are closed or open. It is always a good idea to debounce the switch. There are many libraries that will do that for you. Anyone will do. I happen to have written one that is called "EdgeDebounceLite". It will 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(switchPin);
```

We use:

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

Using the switch to toggle a LED debounced

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

ledPin = 13;
enum ledStates {ON, OFF};
ledStates ledCurrentState;
switchPin = 2;
switchMode = PULLUP;
enum switchStates {OPEN, RISING, CLOSED, FALLING};
switchStates switchCurrentState = OPEN;

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

void readSwitch() {
  byte switchStatus = debounce.pin(switchPin);
  if (switchMode == PULLUP) switchStatus = !switchStatus;
  switch (switchCurrentState) {
    case OPEN: { if(switchStatus == HIGH) switchCurrentState = RISING; break; }
    case RISING: { switchCurrentState = CLOSED; break; }
    case CLOSED: { if(switchStatus) == LOW) switchCurrentState = FALLING; break;}
    case FALLING: { toggleLed(); switchCurrentState = OPEN; break; }
  }
}

void setup() {
  pinMode(switchPin, INPUT_PULLUP);
  switchCurrentState = OPEN;
}

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