

Finite State Machines

Outline

- Finite State Machine Review
- Arduino Code Structure
- How to cleanly implement an FSM on the Arduino
 - switch() statements
 - #define statements
- Example
- Exercise
- Example 2 (with sensor)

Finite State Machine Review:

- Finite State Machine(FSM)
 - It is very useful that there are finite states (we can count them)
 - Finite states means we only need finite resources (they can be implemented in hardware)
 - Less Confusing (by definition they are only in one state at any given time)

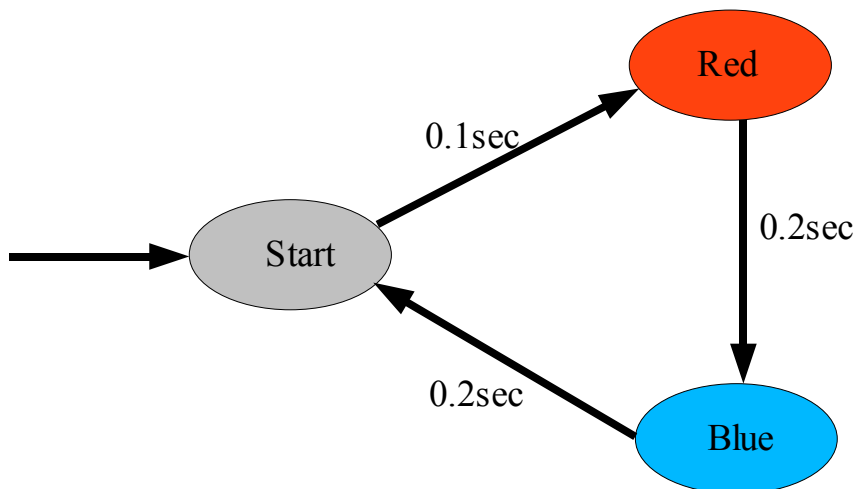


Diagram for FSM used in Example Section

Arduino Code Structure

The Arduino uses a modified form of C++. For this class we will be primarily using the features shared between C and C++. All this means is that we will be avoiding Object Oriented Programming which is a different structure for coding. That said, you can now completely forget the term "Object Oriented Programming" and it's time to look at the C-like features.

C is a language compiled completely before runtime. The notable advantages provided by this method are (1)more speed, (2)less memory requirements, and (3)can be compiled for different processors, such as our Atmega168 microcontrollers on the Arduino. The main disadvantages are that (1) code must be recompiled every time it is changed and (2) code only works on one processor.

Start out with using the following link for reference: <http://arduino.cc/en/Reference/HomePage>
When you get more comfortable/curious look at: <http://arduino.cc/en/Reference/Extended>

If you are ever trying to get true C examples working with the Arduino, you will need to know that there is a difference in how both languages handle the fundamental program structure:

Arduino program format:

```
void setup(){  
    //remember text after 2 slashes like this is just a comment  
    //some setup function calls here  
    //example: set pin as input/output  
}
```

```
void loop(){  
    //after setup() program goes here and runs this code as long as the arduino has power  
    //turn off lights, call new user defined functions, etc  
}
```

C program format:

```
int main(){  
    //notice "int" in the line above meaning this function returns an integer  
    //all code for arduino setup() and loop() goes here  
    return 1;//this is the integer returned  
}//end of program
```

For a little more context, the Arduino code really runs like this:

Real World C/Arduino Combination

```
int main(){

    setup();//arduino's optional setup function

    while(true){ //true is always true
        loop();//run code inside loop forever
    }

    //code never gets this far
    return 1;
} //end of program
```

How to cleanly implement an FSM on the Arduino

- There are two very useful C/C++ statements to help us
 1. *switch/case* statements
 - <http://www.arduino.cc/en/Reference/SwitchCase>
 - `switch()` can handle our FSM evaluation with minimal developer code
 - `case = state` (for our purposes)
 2. *#define* statement
 - `#define` statements create constants, you can't change the value during runtime
 - keeps code clean and understandable

Example

Example FSM(flashing red and blue police car leds, for toy car?)

```
//first define states with obvious names and different values
#define START 0
#define RED 1
#define BLUE 2

//next define which pin is which
#define RED_PIN 9
#define BLUE_PIN 10

int state = START; //create state variable and initialize it to START state
```

```

void setup(){
  pinMode(RED_PIN, OUTPUT);
  pinMode(BLUE_PIN, OUTPUT);
  digitalWrite(BLUE_PIN,LOW); //turn blue led off
  digitalWrite(RED_PIN,LOW); //turn red led off
}

void loop(){
  //FSM time
  switch(state){
    case START:
      digitalWrite(BLUE_PIN,LOW); //turn blue led off
      digitalWrite(RED_PIN,LOW); //turn red led off
      delay(100); //wait 100ms
      state = RED; //transition to red state
      break; //end of START case

    case RED:
      digitalWrite(BLUE_PIN,LOW); //turn blue led off
      digitalWrite(RED_PIN,HIGH); //turn red led on
      delay(200); //wait 200ms
      state = BLUE; //transition to blue state
      break; //end of RED case

    case BLUE:
      digitalWrite(RED_PIN,LOW); //turn red led off
      digitalWrite(BLUE_PIN,HIGH); //turn blue led on
      delay(200); //wait 200ms
      state = START; //transition to start state
      break; //end of BLUE case

  }
}

```

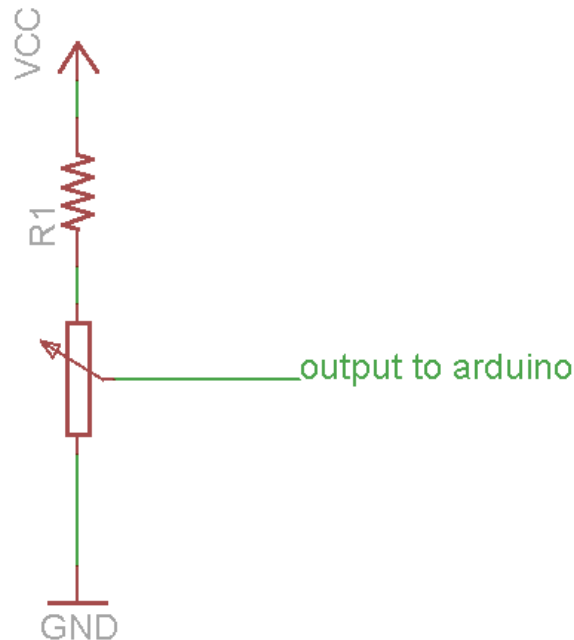
Exercise

- Using switch/case and #define statements as in the example, implement a FSM on an Arduino.
- Use at least 3 different colors to convey state information visually
- In addition to simple delay actions causing state change(as in the example above), use at least one sensor (thermistor, potentiometer, photoresistor, switch...) to cause a state change.
- To do this you will should construct a simple voltage divider and use an analog input.

Example 2

Example FSM similar to the previous one. In this FSM the user can use some sort of sensor, such as a phototransistor, photocell, thermistor, potentiometer, modified servo, etc to cause the the state change. This sensor will be replacing the delay that caused the transition from the START to RED state in the previous example.

As a reminder, the schematic for hooking up a resistance based sensor(a potentiometer in this example) to one of the arduino's **analog** input pins is shown:



Example 2 code starts here:

```
//first define states with obvious names and different values
```

```
#define START 0
```

```
#define RED 1
```

```
#define BLUE 2
```

```
//next define which pin is which
```

```
#define RED_PIN 9
```

```
#define BLUE_PIN 10
```

```
//the analog pin for the sensor causing a state transition
```

```
#define SENSOR_PIN 1
```

```
//the value for the sensor to cause a state change
```

```
#define SENSOR_THRESH 512
```

```

int state = START; //create state variable and initialize it to START state

void setup(){
  pinMode(RED_PIN, OUTPUT);
  pinMode(BLUE_PIN, OUTPUT);
  digitalWrite(BLUE_PIN,LOW); //turn blue led off
  digitalWrite(RED_PIN,LOW); //turn red led off
}

void loop(){
  //FSM time
  switch(state){
    case START:
      digitalWrite(BLUE_PIN,LOW); //turn blue led off
      digitalWrite(RED_PIN,LOW); //turn red led off

      //loop while the sensor value is less than the threshold
      while( analogRead(SENSOR_PIN) < SENSOR_THRESH){
        delay(100) // only check every 100 ms
      }

      //at this point the sensor value was higher than the threshold requirement

      state = RED; //transition to red state
      break; //end of START case

    case RED:
      digitalWrite(BLUE_PIN,LOW); //turn blue led off
      digitalWrite(RED_PIN,HIGH); //turn red led on
      delay(200); //wait 200ms
      state = BLUE; //transition to blue state
      break; //end of RED case

    case BLUE:
      digitalWrite(RED_PIN,LOW); //turn red led off
      digitalWrite(BLUE_PIN,HIGH); //turn blue led on
      delay(200); //wait 200ms
      state = START; //transition to start state
      break; //end of BLUE case

  }
}

```