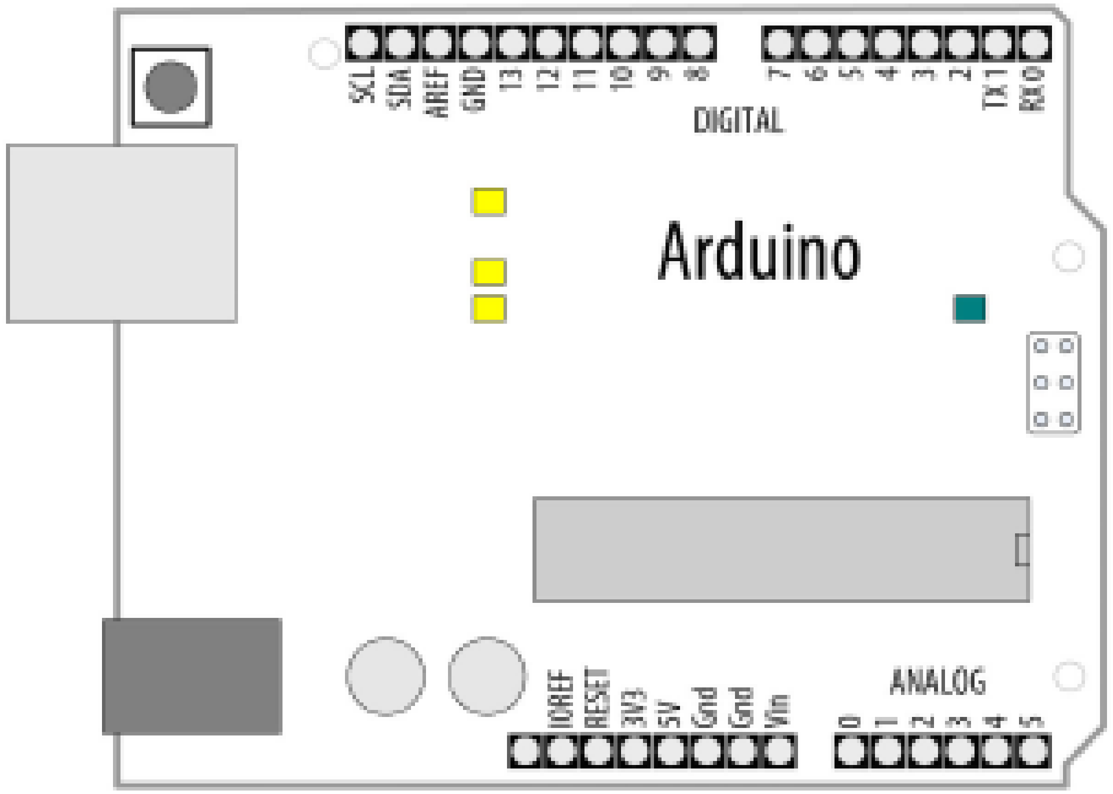
# Chapter 5

Simple Digital and Analog Input

## Introduction

- Arduino pins sense digital and analog inputs. Digital input pins sense the presence and absence of voltage on a pin. Analog input pins measure a range of voltages on a pin.



## Introduction (Contd.)

- Arduino function to detect digital input is digitalRead, and it tells your sketch if a voltage on a pin is HIGH or LOW.
- The Arduino function to configure a pin for reading input is pin Mode(pin, INPUT).
- There are 14 digital pins (numbered 0 to 13) 13 has a built in LED
- Pins 0 and 1 (marked RX and TX) are used for the USB serial connection and should be avoided for other uses.

## Introduction (Contd.)

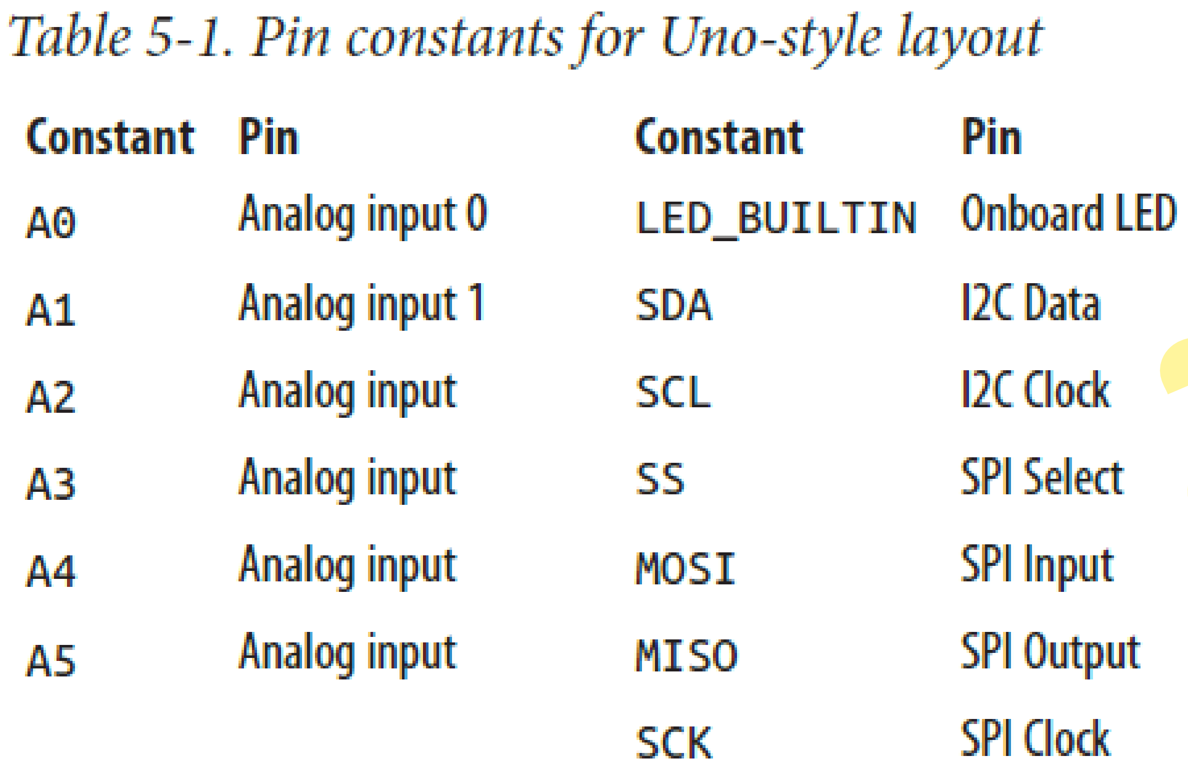


*Table 5-1. Pin constants for Uno-style layout*

| Constant | Pin | Constant | Pin |
|---|---|---|---|
| `A0` | Analog input 0 | `LED_BUILTIN` | Onboard LED |
| `A1` | Analog input 1 | `SDA` | I2C Data |
| `A2` | Analog input | `SCL` | I2C Clock |
| `A3` | Analog input | `SS` | SPI Select |
| `A4` | Analog input | `MOSI` | SPI Input |
| `A5` | Analog input | `MISO` | SPI Output |
| | | `SCK` | SPI Clock |

pinMode(i, INPUT_PULLUP) - HIGH all the time, LOW when device is connected

## Introduction

- Digital input sometimes use internal or external resistors to force the input pin to stay in a known state when the input is not engaged
- Without such a resistor, the pin's value would be in a state known as *floating*, and digitalRead might return HIGH at first, but then would return LOW milliseconds later.
- A *pull-up* resistor is so named because the voltage is "pulled up" to the logic level (5V or 3.3V) of the board.
- When you press the button in a pull-up configuration, digitalRead will return LOW. At all other times, it returns HIGH because the pull-up resistor is keeping high.

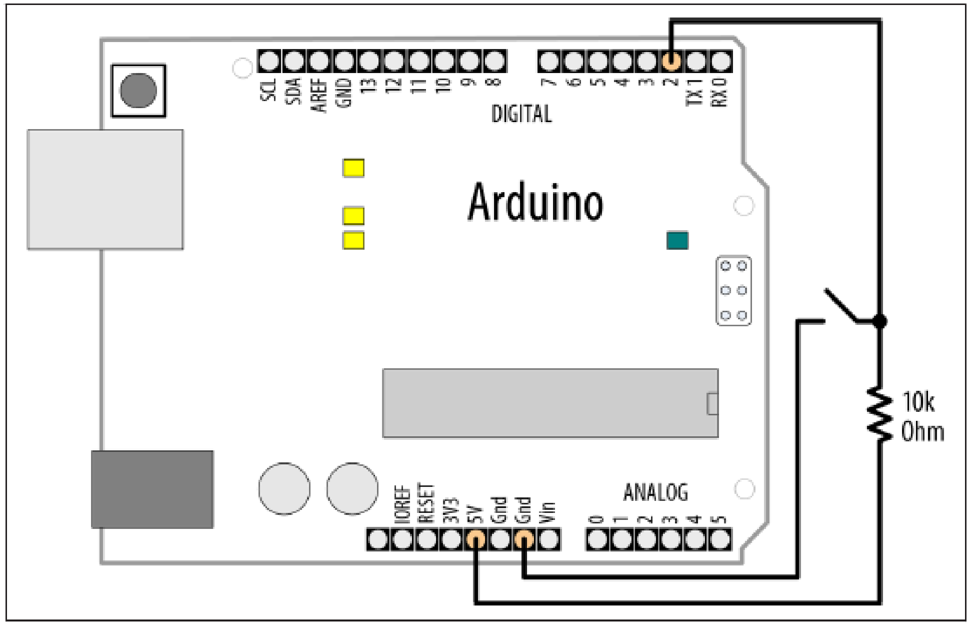


Figure 5-4. Switch connected using pull-up resistor

# Introduction

- *A pull-down* resistor pulls the pin down to 0 volts. In this configuration, digitalRead will return HIGH when the button is pressed. Although 10K ohms is a commonly used value for a pull-up or pull-down resistor, anything between 4.7K and 20K or more will work.
- Arduino boards have internal pull-up resistors that you can activate when you use the INPUT_PULLUP mode with pinMode.
- This eliminates the need for external pull-up resistors.

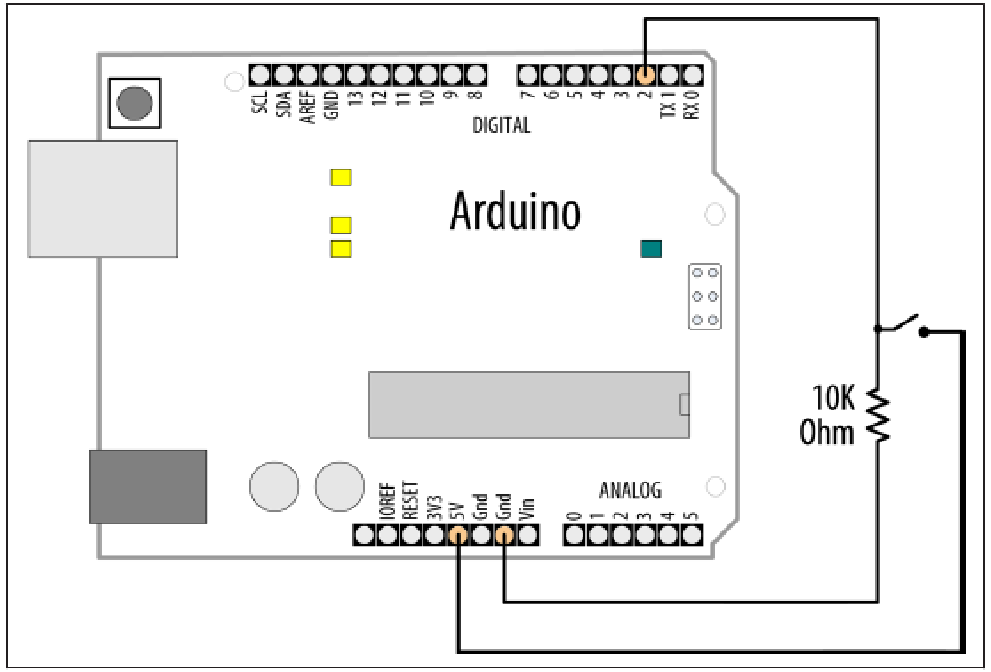
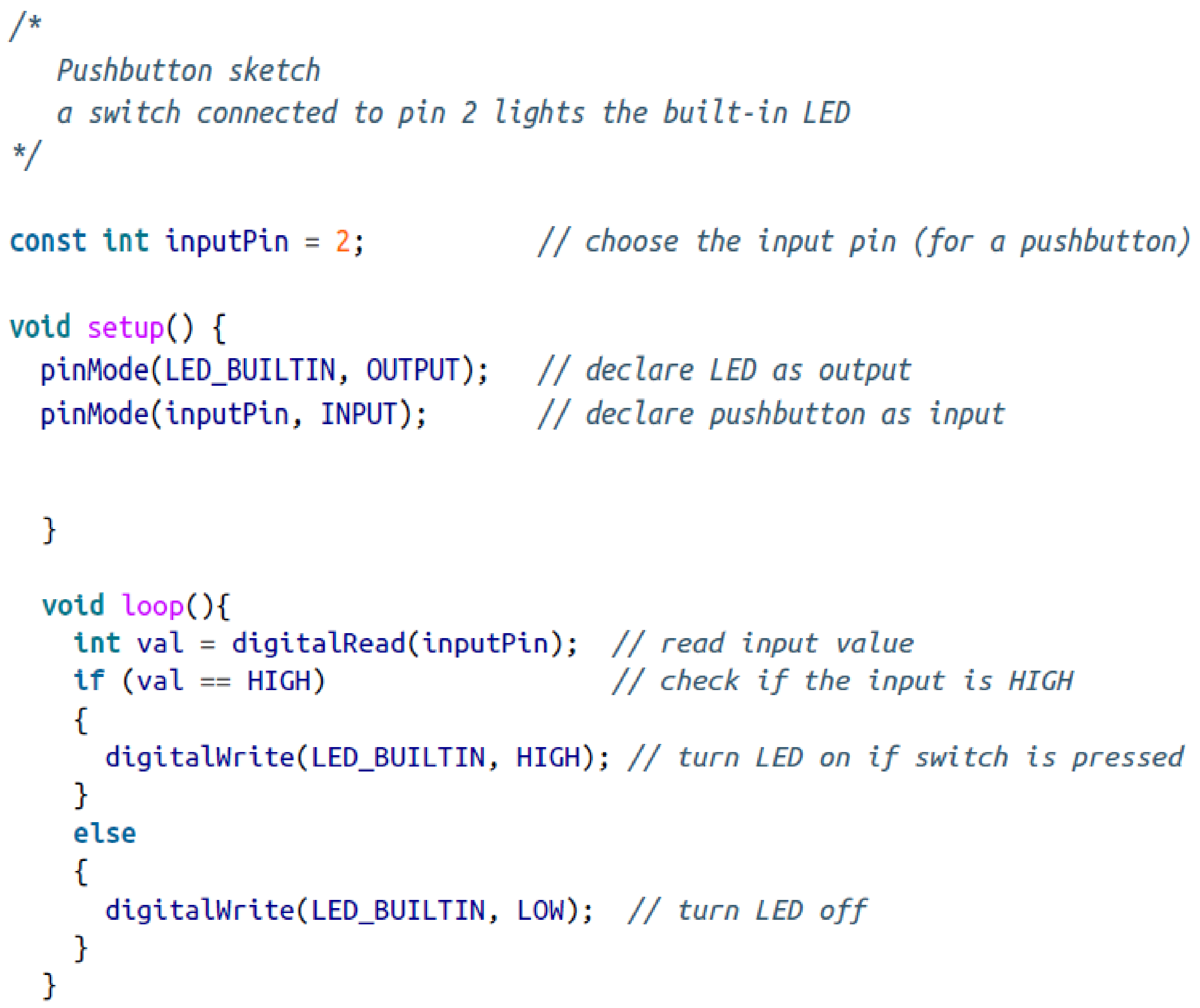


*Figure 5-3. Switch connected using pull-down resistor*

## Introduction

- Arduino code uses a function called analogRead to get a value proportional to the voltage it sees on one of its analog pins.
- The value will be 0 if there are 0 volts on the pin and 1,023 for 5 volts.
- Analog recipes use a *potentiometer* to vary the voltage on a pin. When choosing a potentiometer, a value of 10K is the best option for connecting to analog pins.
- Multimeter can measure voltage and resistance.

# Using a Switch



```
/*
   Pushbutton sketch
   a switch connected to pin 2 lights the built-in LED
*/

const int inputPin = 2;          // choose the input pin (for a pushbutton)

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);   // declare LED as output
  pinMode(inputPin, INPUT);       // declare pushbutton as input


  }

  void loop(){
    int val = digitalRead(inputPin);  // read input value
    if (val == HIGH)                  // check if the input is HIGH
    {
      digitalWrite(LED_BUILTIN, HIGH); // turn LED on if switch is pressed
    }
    else
    {
      digitalWrite(LED_BUILTIN, LOW);  // turn LED off
    }
  }
```

- Problem: You want your sketch to respond to the closing of an electrical contact; for example, a pushbutton or other switch or an external device that makes an electrical connection.

# Using a Switch

The pull-up code is similar to the pull-down version, but the logic is reversed

```
void loop()
{
  int val = digitalRead(inputPin);  // read input value
  if (val == HIGH)                  // check if the input is HIGH
  {
    digitalWrite(LED_BUILTIN, LOW);  // turn LED OFF
  }
  else
  {
    digitalWrite(LED_BUILTIN, HIGH); // turn LED ON
  }
}
```

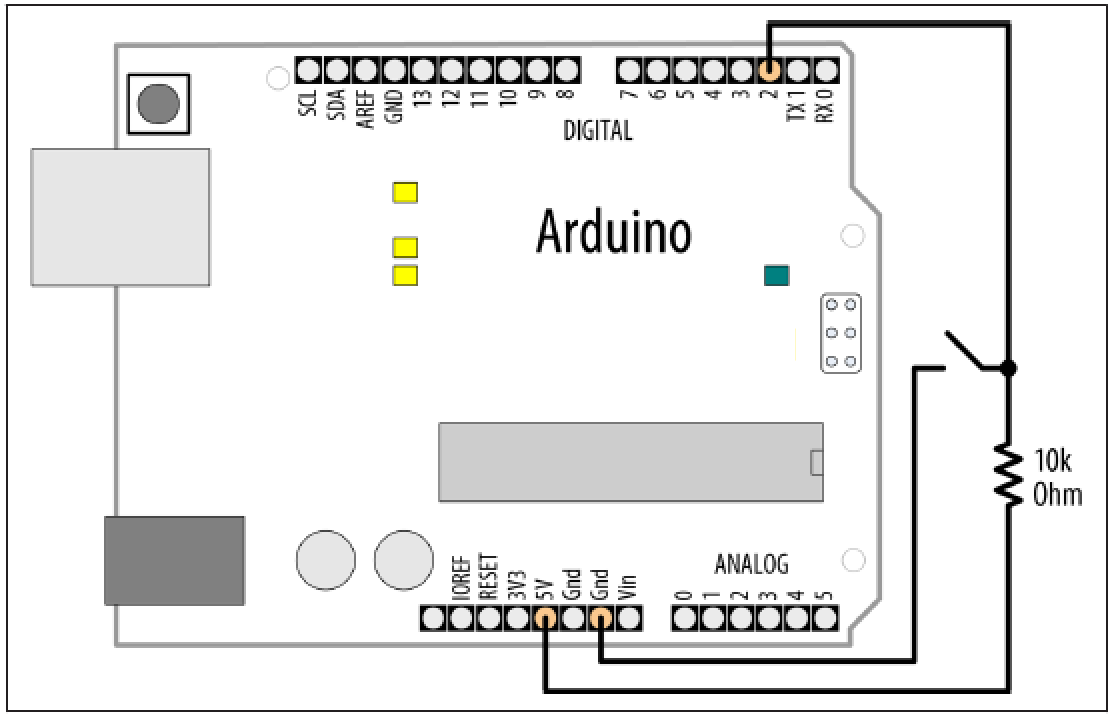


*Figure 5-4. Switch connected using pull-up resistor*

# Using a Switch Without External Resistors

## Problem

You want to simplify your wiring by eliminating external pull-up resistors when connecting switches.

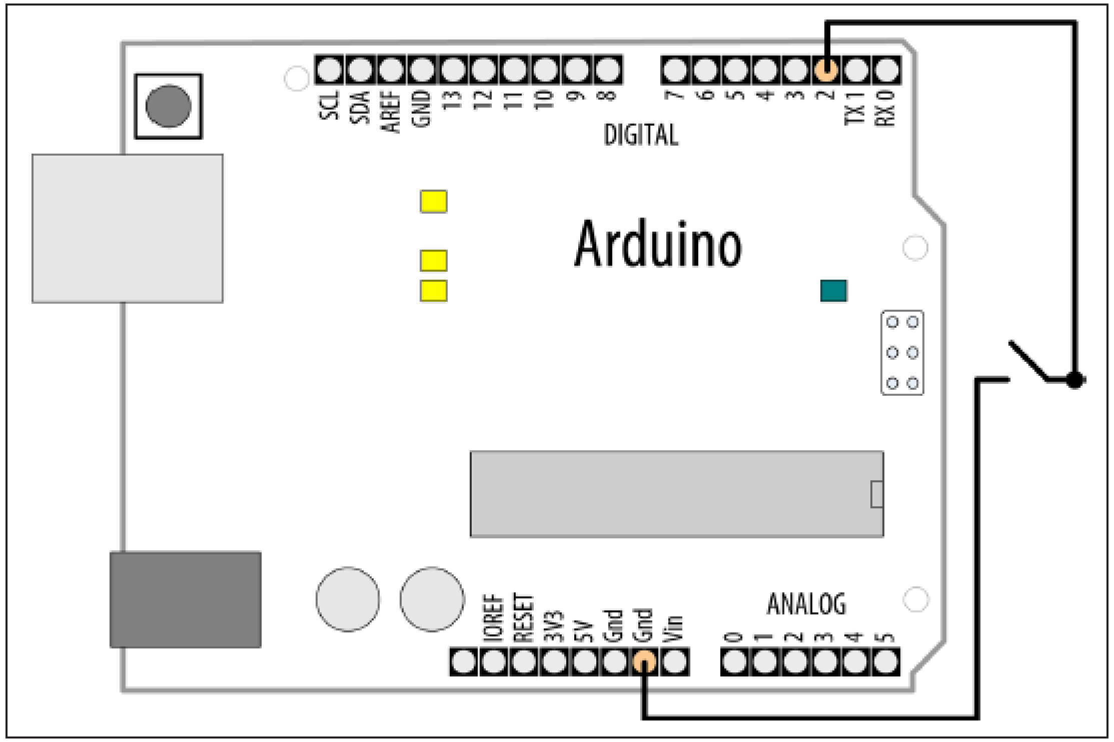
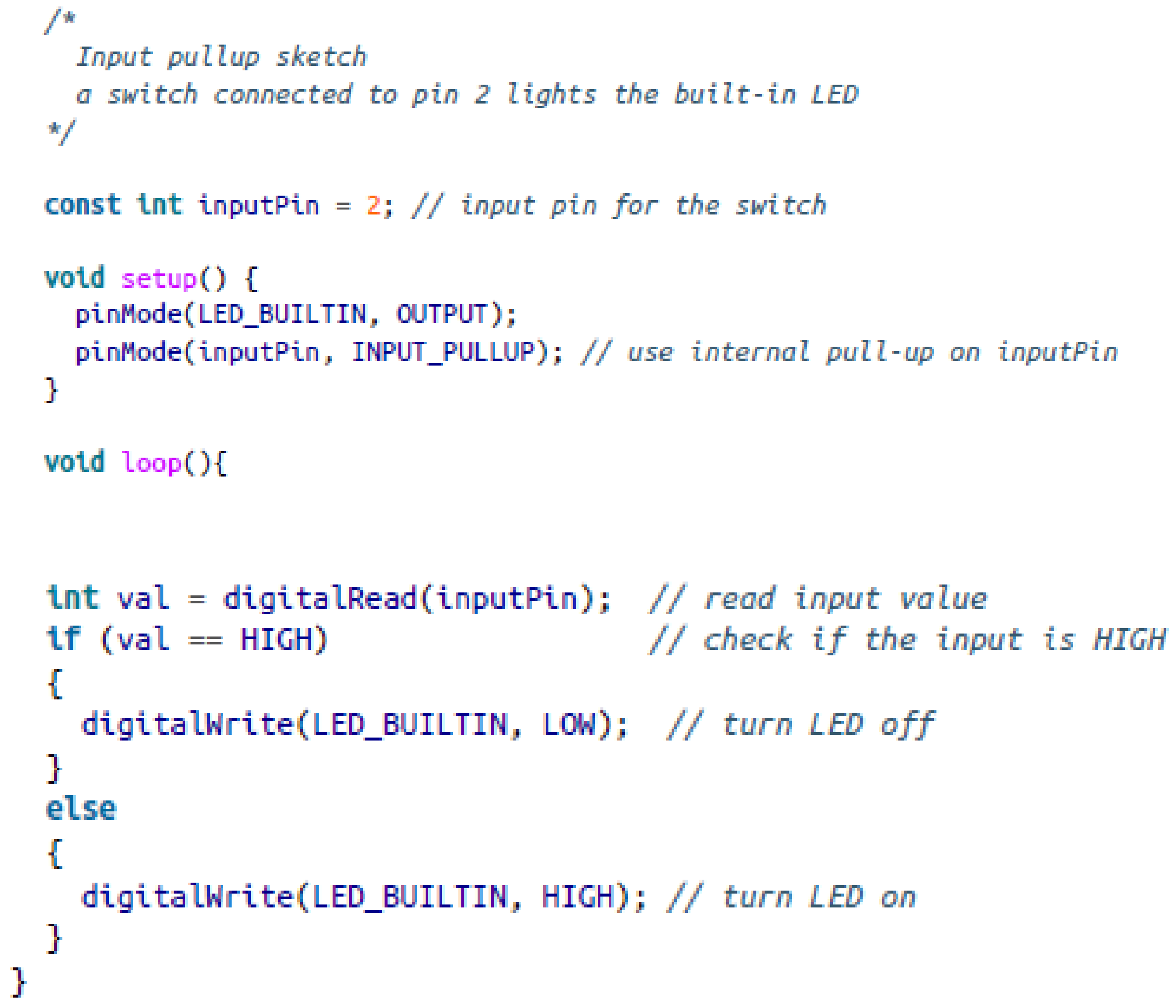


*Figure 5-5. Switch wired for use with internal pull-up resistor*

# Using a Switch Without External Resistors

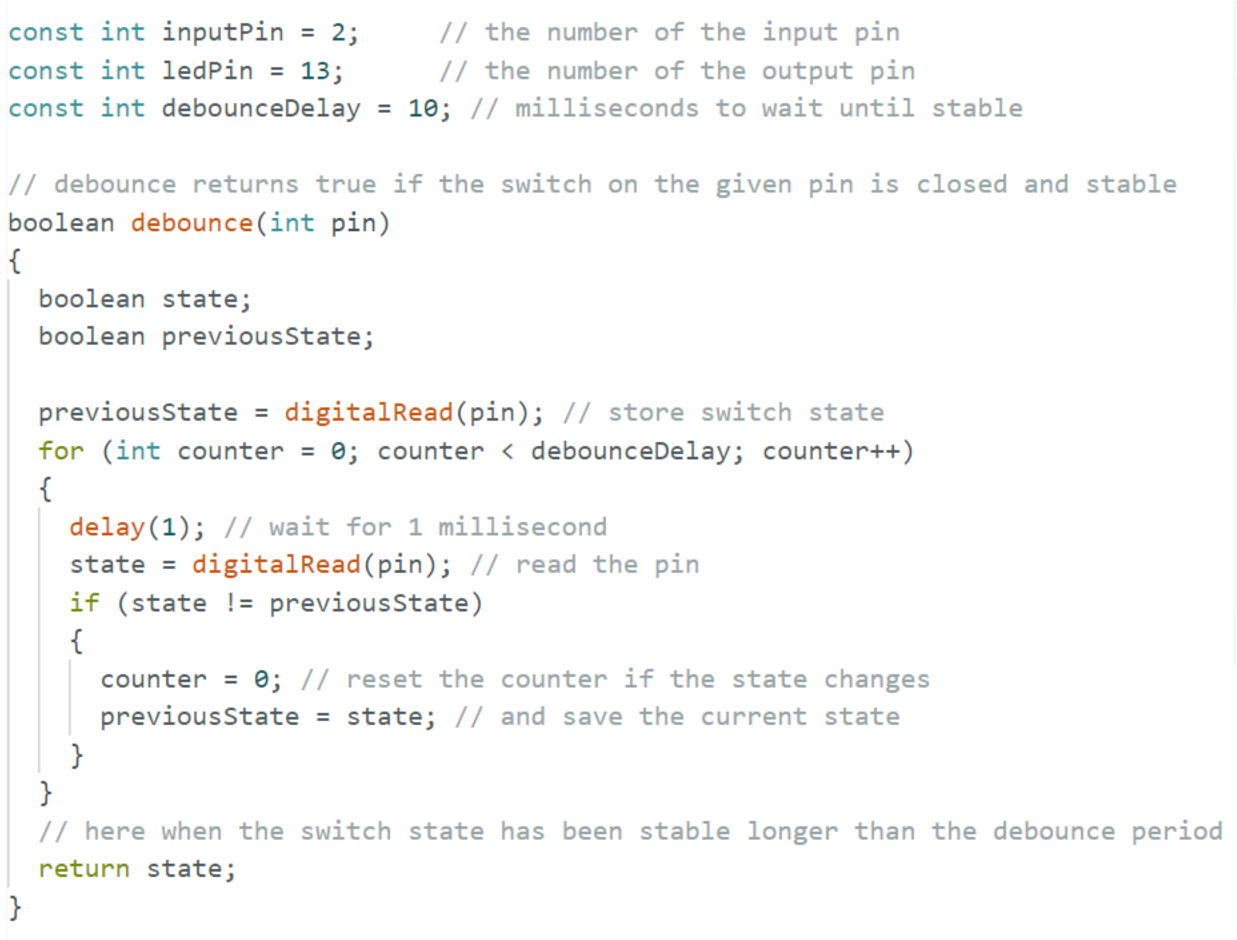


```
 /*
   Input pullup sketch
   a switch connected to pin 2 lights the built-in LED
 */

 const int inputPin = 2; // input pin for the switch

 void setup() {
   pinMode(LED_BUILTIN, OUTPUT);
   pinMode(inputPin, INPUT_PULLUP); // use internal pull-up on inputPin
 }

 void loop(){


 int val = digitalRead(inputPin);  // read input value
 if (val == HIGH)                  // check if the input is HIGH
 {
   digitalWrite(LED_BUILTIN, LOW);  // turn LED off
 }
 else
 {
   digitalWrite(LED_BUILTIN, HIGH); // turn LED on
 }
}
```

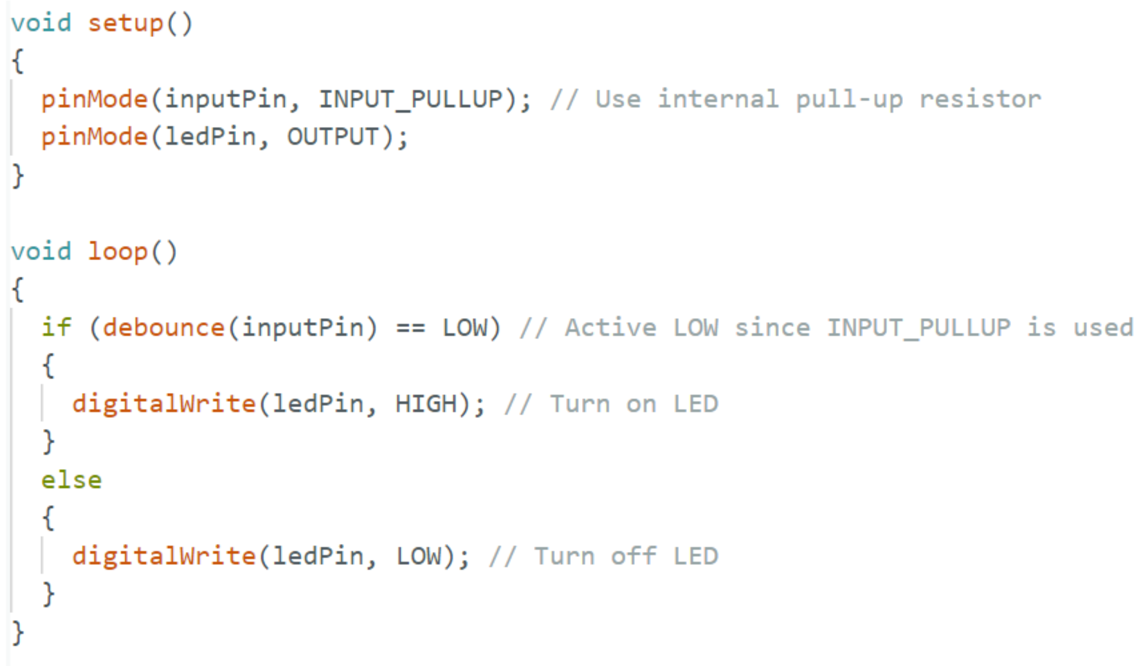
## Reliably Detect (Debounce) When a Switch Is Pressed

- When you press a button down, it may not immediately make a complete connection. In fact, it may make contact on one side – then both – and then the other side – until it finally settles down. This making and breaking contact is called bouncing.
- Contact bounce produces spurious signals at the moment the switch contacts close or open.
- The process of eliminating spurious readings is called *debouncing*.

# Reliably Detect (Debounce) When a Switch Is Pressed



```
const int inputPin = 2;     // the number of the input pin
const int ledPin = 13;      // the number of the output pin
const int debounceDelay = 10; // milliseconds to wait until stable

// debounce returns true if the switch on the given pin is closed and stable
boolean debounce(int pin)
{
  boolean state;
  boolean previousState;

  previousState = digitalRead(pin); // store switch state
  for (int counter = 0; counter < debounceDelay; counter++)
  {
    delay(1); // wait for 1 millisecond
    state = digitalRead(pin); // read the pin
    if (state != previousState)
    {
      counter = 0; // reset the counter if the state changes
      previousState = state; // and save the current state
    }
  }
  // here when the switch state has been stable longer than the debounce period
  return state;
}
```



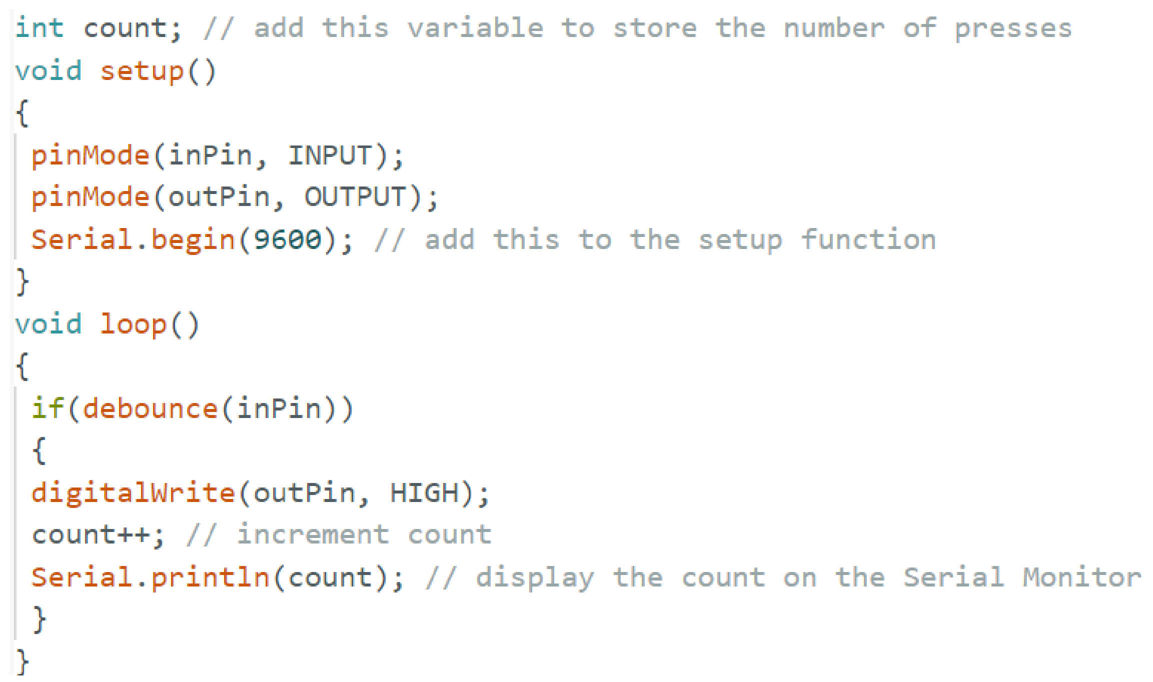
```
void setup()
{
  pinMode(inputPin, INPUT_PULLUP); // Use internal pull-up resistor
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  if (debounce(inputPin) == LOW) // Active LOW since INPUT_PULLUP is used
  {
    digitalWrite(ledPin, HIGH); // Turn on LED
  }
  else
  {
    digitalWrite(ledPin, LOW); // Turn off LED
  }
}
```

## Reliably Detect (Debounce) When a Switch Is Pressed

- When you want to reliably check for a button press, you should call the debounce function with the pin number of the switch you want to debounce.
- The debounce method checks to see if it gets the same reading from the switch after a delay that needs to be long enough for the switch contacts to stop bouncing.

## Reliability Detect (Contd.)

- ❑ add a count variable to display the number of presses



```
int count; // add this variable to store the number of presses
void setup()
{
 pinMode(inPin, INPUT);
 pinMode(outPin, OUTPUT);
 Serial.begin(9600); // add this to the setup function
}
void loop()
{
 if(debounce(inPin))
 {
 digitalWrite(outPin, HIGH);
 count++; // increment count
 Serial.println(count); // display the count on the Serial Monitor
 }
}
```

## Reliability Detect (Contd.)

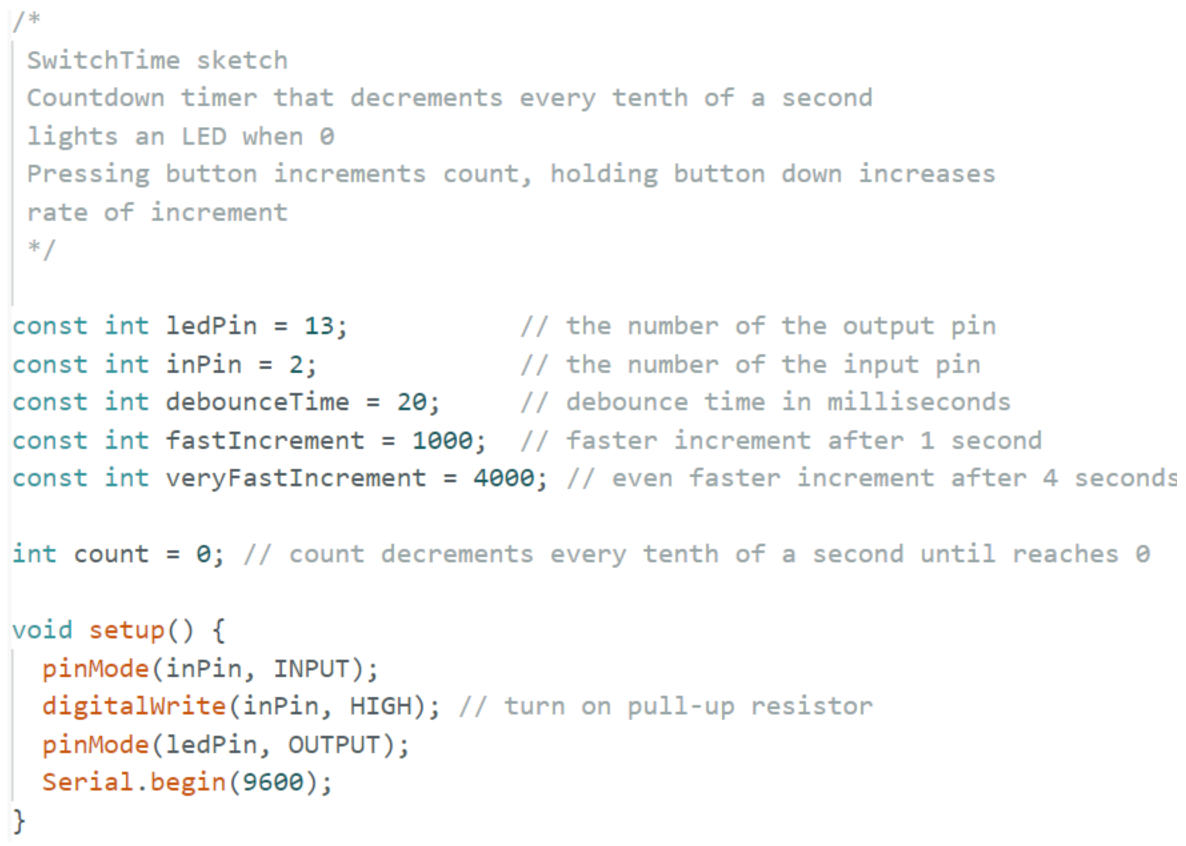
A potential disadvantage of this method for some applications is that from the time the debounce function is called, everything waits until the switch is stable. In most cases this doesn’t matter, but your sketch may need to be attending to other things while waiting for your switch to stabilize.

## Determining How Long a Switch Is Pressed

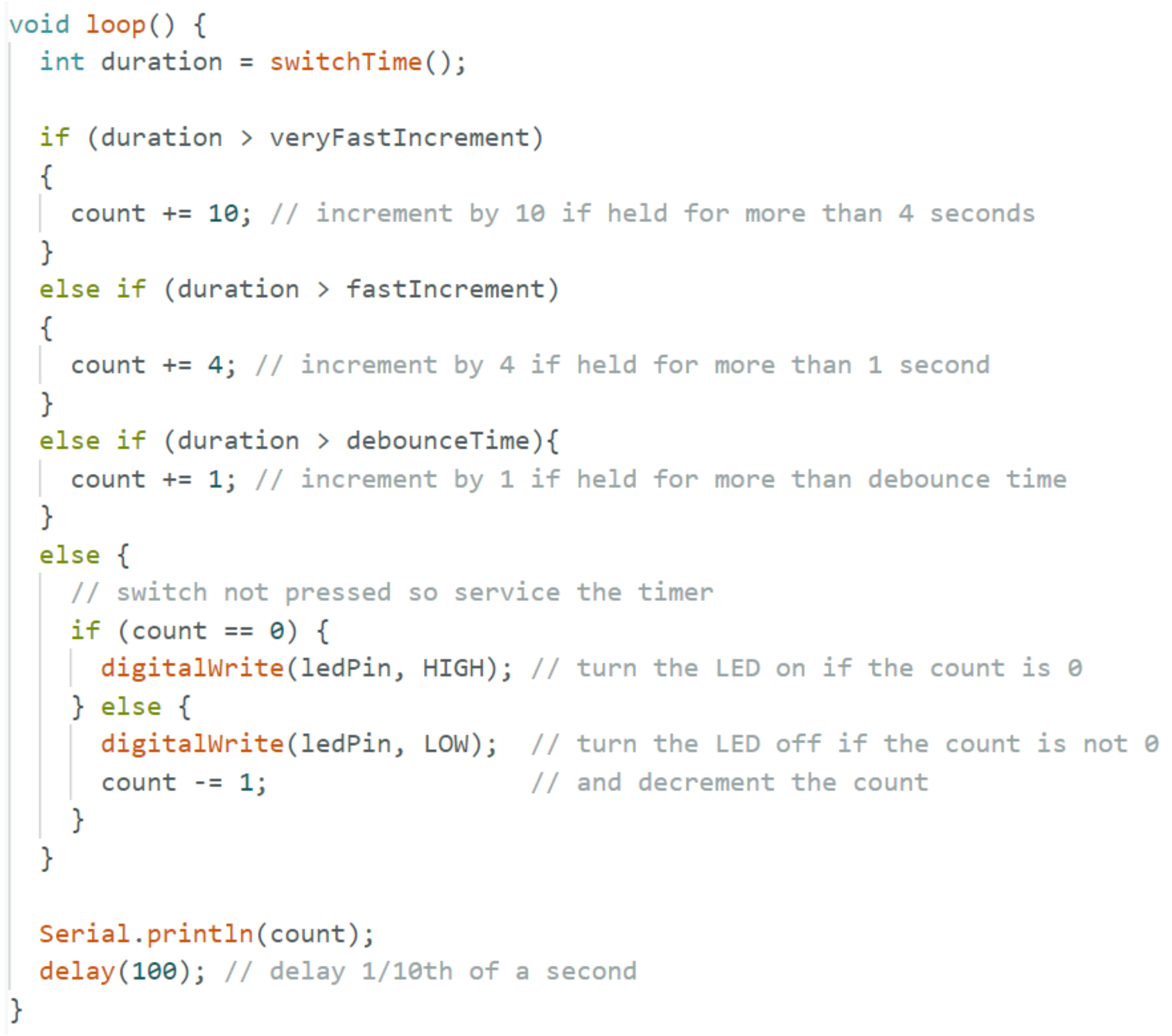
**Setting of a countdown timer:**

- Pressing a switch sets the timer by incrementing the timer count; releasing the switch starts the countdown
- The code debounces the switch and accelerates the rate at which the counter increases when the switch is held for longer periods.
- The timer count is incremented by one when the switch is initially pressed (after debouncing)
- Holding the switch for more than one second increases the increment rate by four; holding the switch for four seconds increases the rate by 10
- Releasing the switch starts the countdown, and when the count reaches zero, a pin is set HIGH

# Determining How Long a Switch Is Pressed

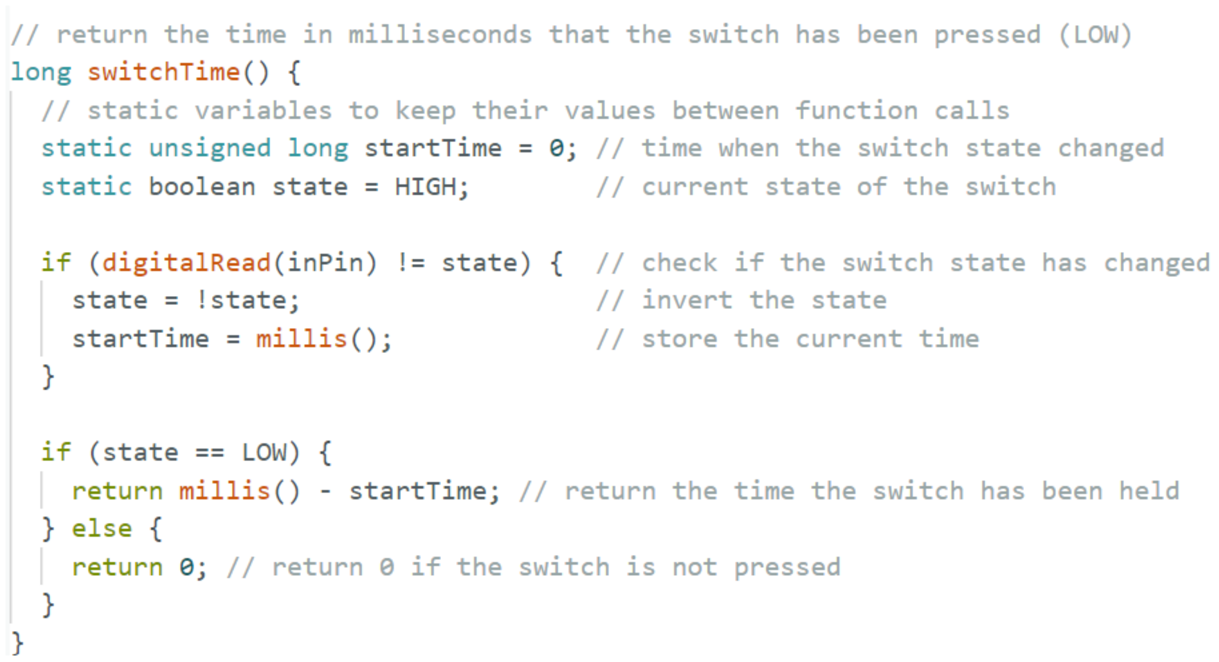


```
/*
 SwitchTime sketch
 Countdown timer that decrements every tenth of a second
 lights an LED when 0
 Pressing button increments count, holding button down increases
 rate of increment
 */

const int ledPin = 13;            // the number of the output pin
const int inPin = 2;              // the number of the input pin
const int debounceTime = 20;      // debounce time in milliseconds
const int fastIncrement = 1000;   // faster increment after 1 second
const int veryFastIncrement = 4000; // even faster increment after 4 seconds

int count = 0; // count decrements every tenth of a second until reaches 0

void setup() {
  pinMode(inPin, INPUT);
  digitalWrite(inPin, HIGH); // turn on pull-up resistor
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}
```



```
void loop() {
  int duration = switchTime();

  if (duration > veryFastIncrement)
  {
    count += 10; // increment by 10 if held for more than 4 seconds
  }
  else if (duration > fastIncrement)
  {
    count += 4; // increment by 4 if held for more than 1 second
  }
  else if (duration > debounceTime){
    count += 1; // increment by 1 if held for more than debounce time
  }
  else {
    // switch not pressed so service the timer
    if (count == 0) {
      digitalWrite(ledPin, HIGH); // turn the LED on if the count is 0
    } else {
      digitalWrite(ledPin, LOW);  // turn the LED off if the count is not 0
      count -= 1;                 // and decrement the count
    }
  }

  Serial.println(count);
  delay(100); // delay 1/10th of a second
}
```

# Determining How Long a Switch Is Pressed



```
// return the time in milliseconds that the switch has been pressed (LOW)
long switchTime() {
  // static variables to keep their values between function calls
  static unsigned long startTime = 0; // time when the switch state changed
  static boolean state = HIGH;        // current state of the switch

  if (digitalRead(inPin) != state) {  // check if the switch state has changed
    state = !state;                   // invert the state
    startTime = millis();             // store the current time
  }

  if (state == LOW) {
    return millis() - startTime; // return the time the switch has been held
  } else {
    return 0; // return 0 if the switch is not pressed
  }
}
```

## Determining How Long a Switch Is Pressed

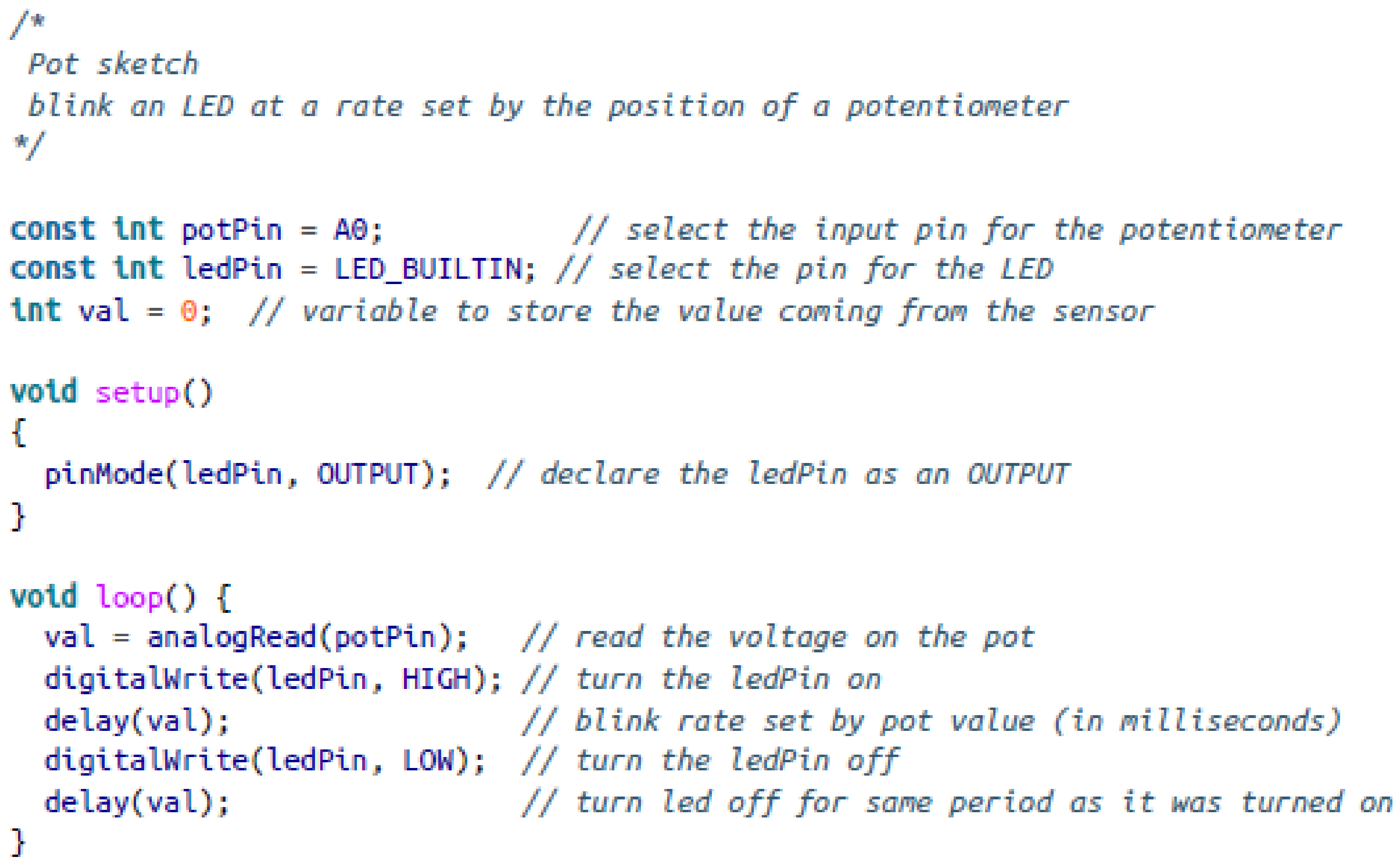
- To retain the values of variables defined in the function, this sketch uses *static variables*.
- Static variables within a function provide permanent storage for values that must be maintained between function calls
- A value assigned to a static variable is retained even after the function returns.
- But unlike global variables, static variables declared in a function are only accessible within that function. The benefit of static variables is that they cannot be accidentally modified by some other function.

## Reading Analog Values

- The voltage is measured using analogRead, which provides a value proportional to the actual voltage on the analog pin.
- A pot has three pins; two are connected to a resistive material and the third pin (usually in the middle) is connected to a wiper that can be rotated to make contact anywhere on the resistive material.
- As the potentiometer rotates, the resistance between the wiper and one of the pins increases, while the other decreases.
- As the wiper moves down, the voltage on the analog pin will decrease (to a minimum of 0 volts). Moving the wiper upward will have the opposite effect, and the voltage on the pin will increase (up to a  maximum of 5 or 3.3 volts).

# Reading Analog Values

- Problem: You want to read the voltage on an analog pin. Perhaps you want a reading from a potentiometer (pot), a variable resistor, or a sensor that provides a varying voltage.



```
/*
 Pot sketch
 blink an LED at a rate set by the position of a potentiometer
*/

const int potPin = A0;           // select the input pin for the potentiometer
const int ledPin = LED_BUILTIN; // select the pin for the LED
int val = 0;  // variable to store the value coming from the sensor

void setup()
{
  pinMode(ledPin, OUTPUT);  // declare the ledPin as an OUTPUT
}

void loop() {
  val = analogRead(potPin);   // read the voltage on the pot
  digitalWrite(ledPin, HIGH); // turn the ledPin on
  delay(val);                 // blink rate set by pot value (in milliseconds)
  digitalWrite(ledPin, LOW);  // turn the ledPin off
  delay(val);                 // turn led off for same period as it was turned on
}
```

## Reading Analog Values

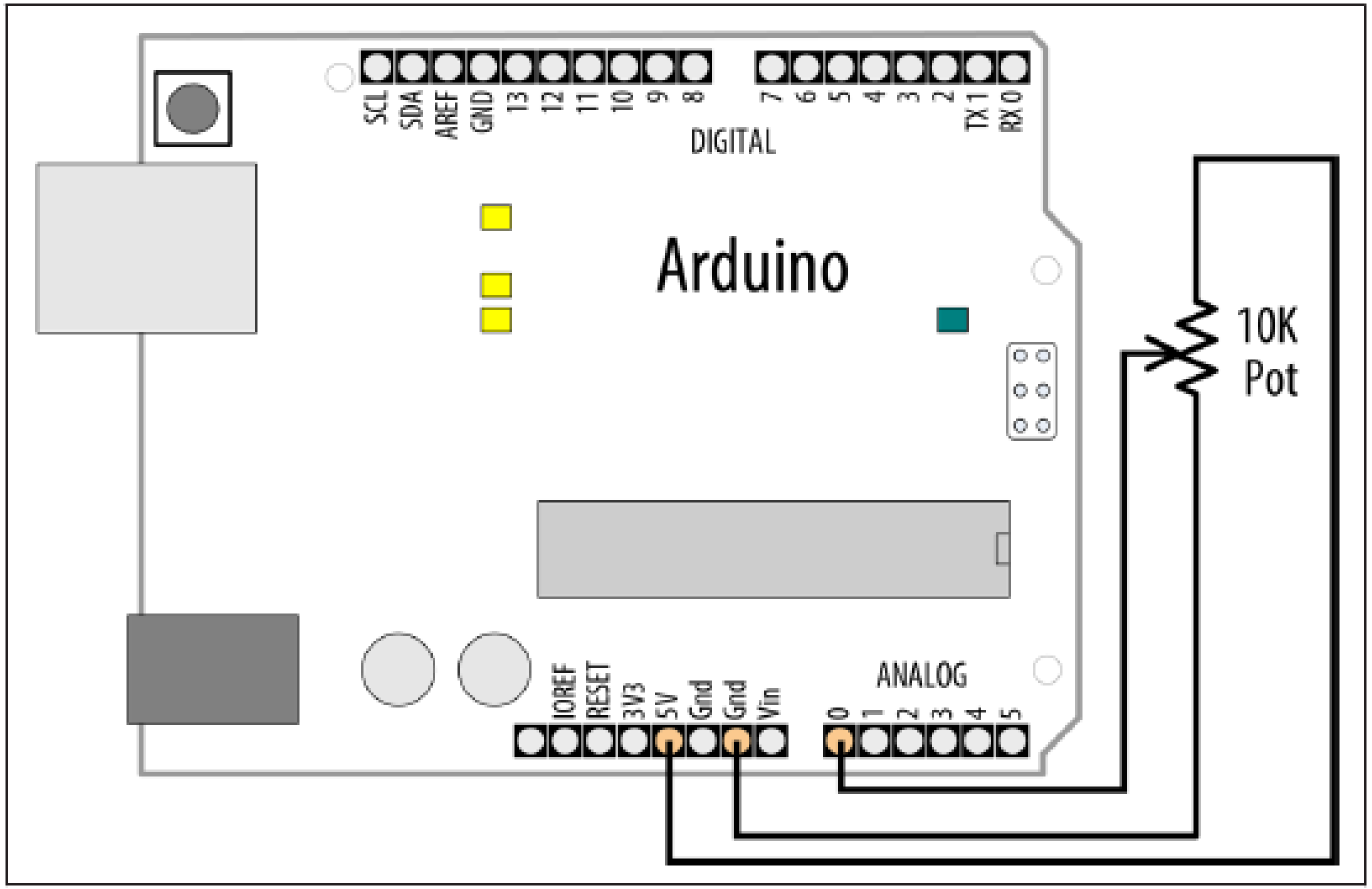


*Figure 5-7. Connecting a potentiometer to Arduino*

## Reading More than Six Analog Inputs

### Problem

You have more analog inputs to monitor than you have available analog pins. A standard Arduino board has six analog inputs (the Mega has 16) and there may not be enough analog inputs available for your application. Perhaps you want to adjust eight parameters in your application by turning knobs on eight potentiometers.

### **Solution**

- Use a multiplexer chip to select and connect multiple voltage sources to one analog input.
- By sequentially selecting from multiple sources, you can read each source in turn.
- Connect your analog inputs (such as a pot or resistive sensor) to the 4051 pins marked Ch 0 to Ch 7. Make sure the voltage on the channel input pins is never higher than 5 volts. If you are not using an input pin, you must connect it to ground with a 10K resistor

# Reading More than Six Analog Inputs

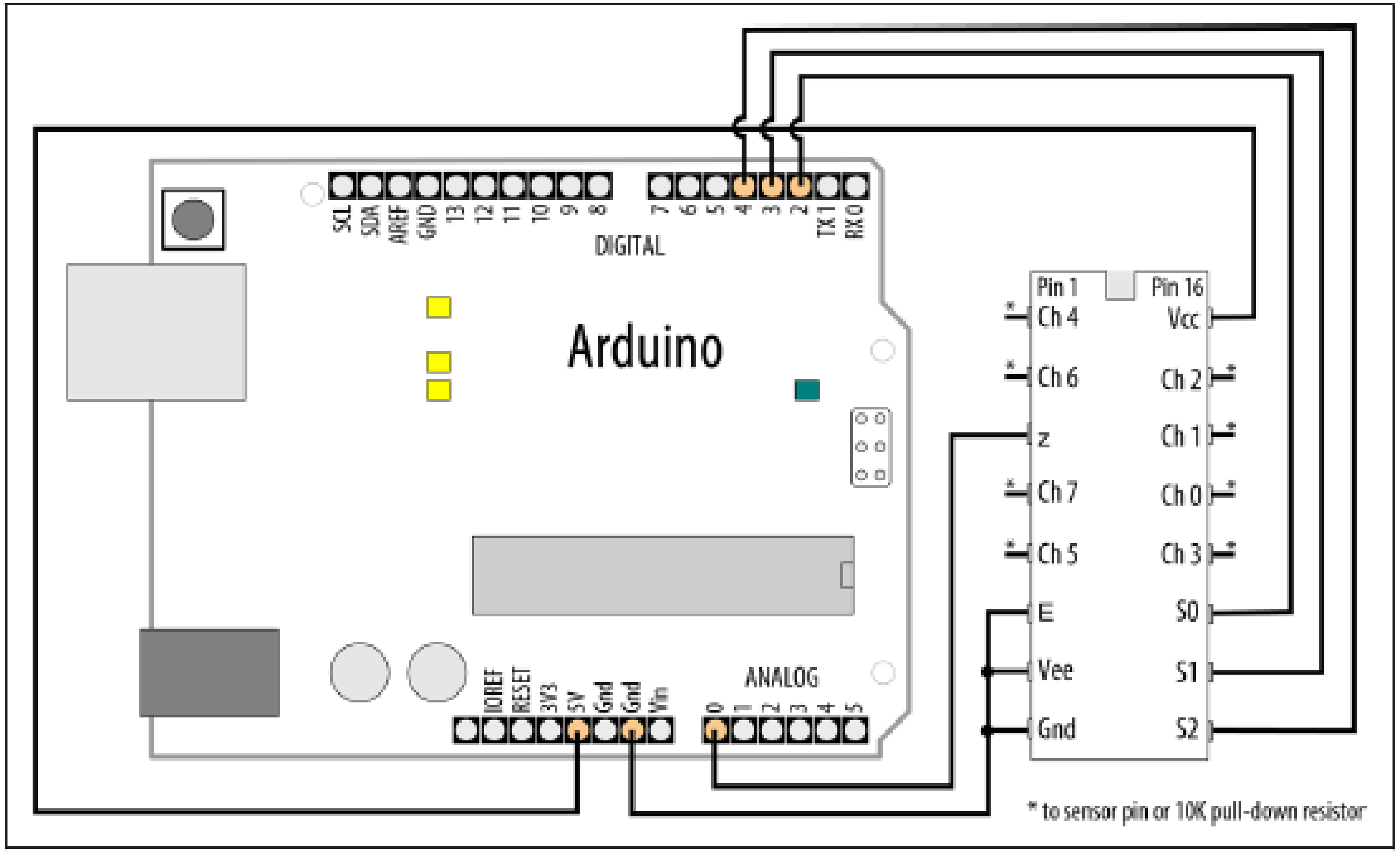
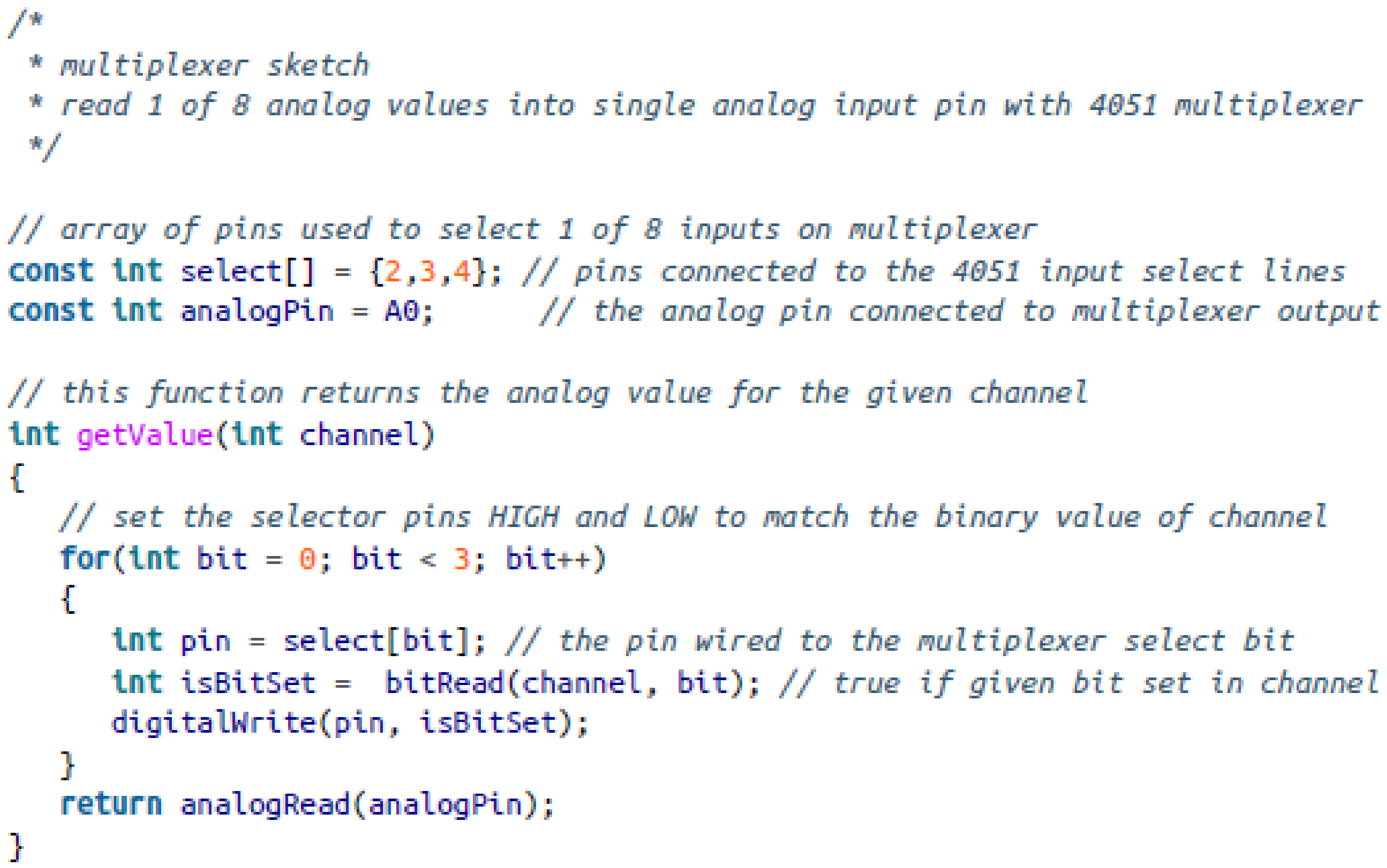


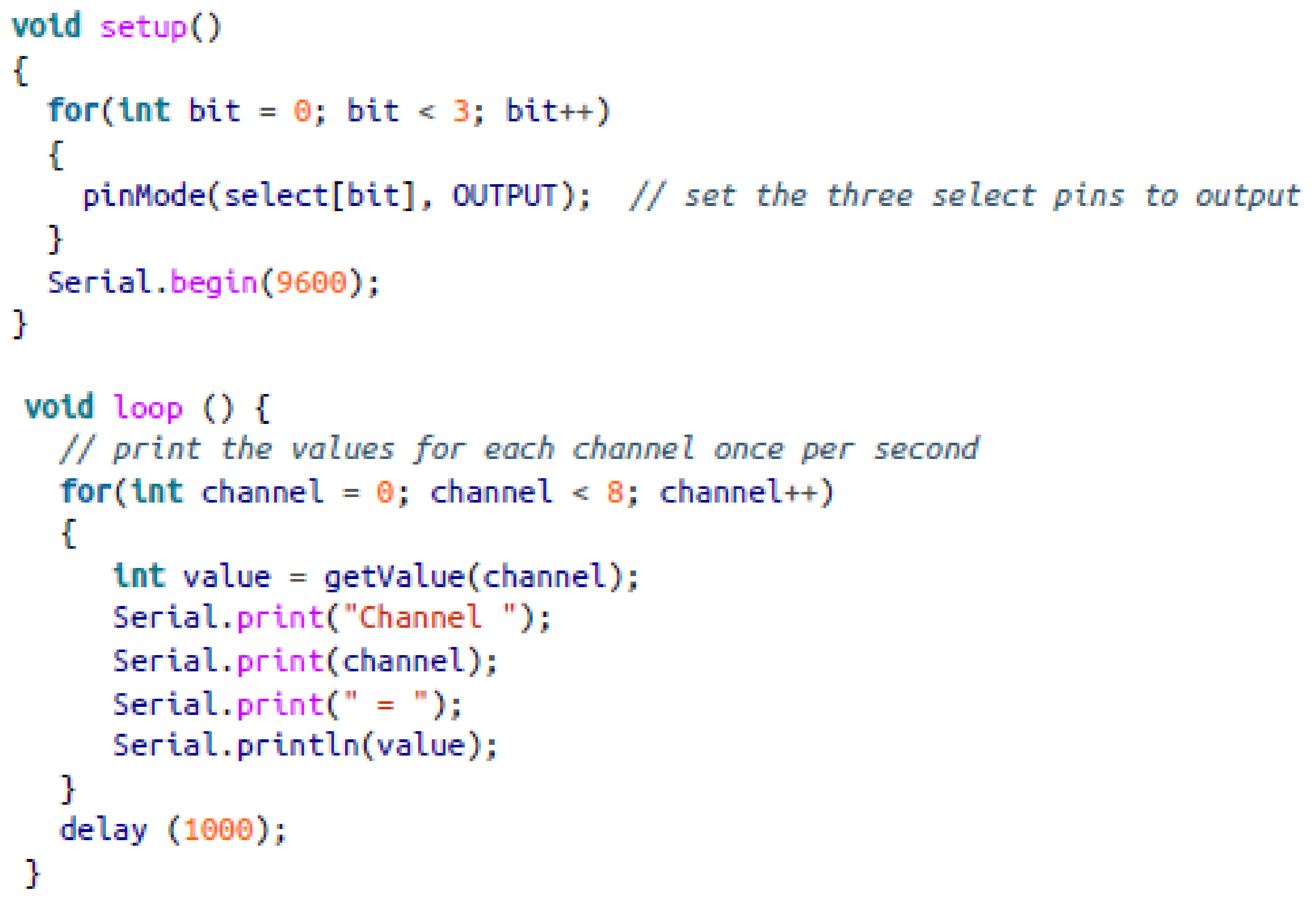
*Figure 5-8. The 4051 multiplexer connected to Arduino*

# Reading More than Six Analog Inputs



```
/*
 * multiplexer sketch
 * read 1 of 8 analog values into single analog input pin with 4051 multiplexer
 */

// array of pins used to select 1 of 8 inputs on multiplexer
const int select[] = {2,3,4}; // pins connected to the 4051 input select lines
const int analogPin = A0;      // the analog pin connected to multiplexer output

// this function returns the analog value for the given channel
int getValue(int channel)
{
   // set the selector pins HIGH and LOW to match the binary value of channel
   for(int bit = 0; bit < 3; bit++)
   {
      int pin = select[bit]; // the pin wired to the multiplexer select bit
      int isBitSet =  bitRead(channel, bit); // true if given bit set in channel
      digitalWrite(pin, isBitSet);
   }
   return analogRead(analogPin);
}
```

# Reading More than Six Analog Inputs

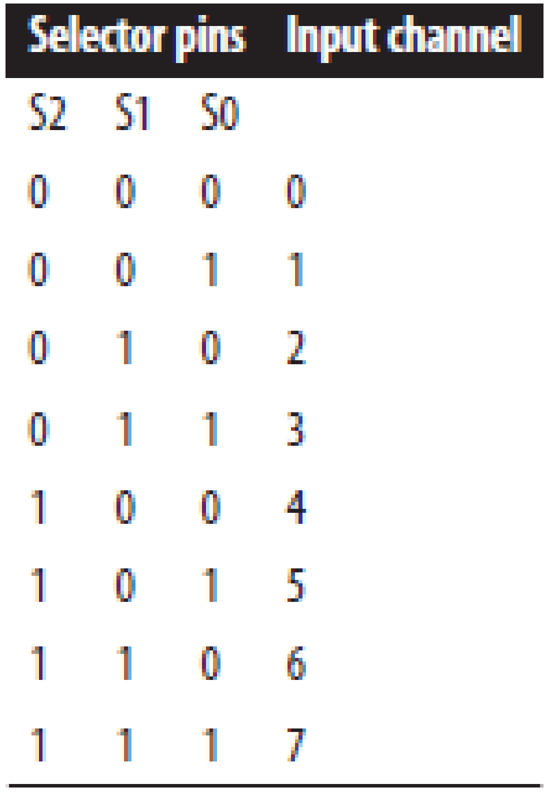


```
void setup()
{
  for(int bit = 0; bit < 3; bit++)
  {
    pinMode(select[bit], OUTPUT);  // set the three select pins to output
  }
  Serial.begin(9600);
}

 void loop () {
   // print the values for each channel once per second
   for(int channel = 0; channel < 8; channel++)
   {
      int value = getValue(channel);
      Serial.print("Channel ");
      Serial.print(channel);
      Serial.print(" = ");
      Serial.println(value);
   }
   delay (1000);
 }
```

# Reading More than Six Analog Inputs

- Analog multiplexers are digitally controlled analog switches.
- The 4051 selects one of eight inputs through three selector pins (S0, S1, and S2). There are eight different combinations of values for the three selector pins, and the sketch sequentially selects each of the possible bit patterns.

*Table 5-3. Truth table for 4051 multiplexer*



| Selector pins | | | Input channel |
|---|---|---|---|
| S2 | S1 | S0 | |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 |
| 1 | 1 | 1 | 7 |

# Measuring Voltages Up to 5V

## Problem

You want to monitor and display the value of a voltage between 0 and 5 volts. For example, suppose you want to display the voltage of a single 1.5V cell on the Serial Monitor.

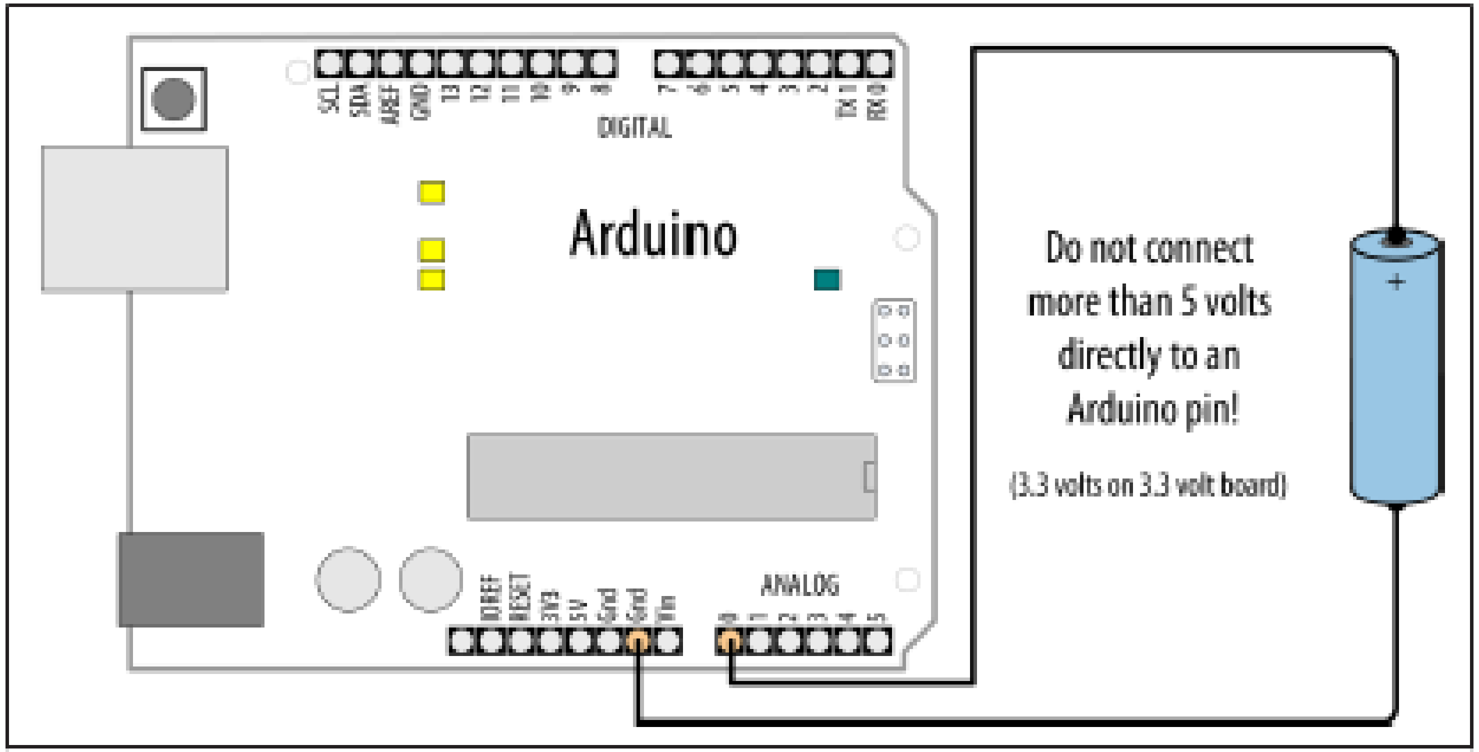
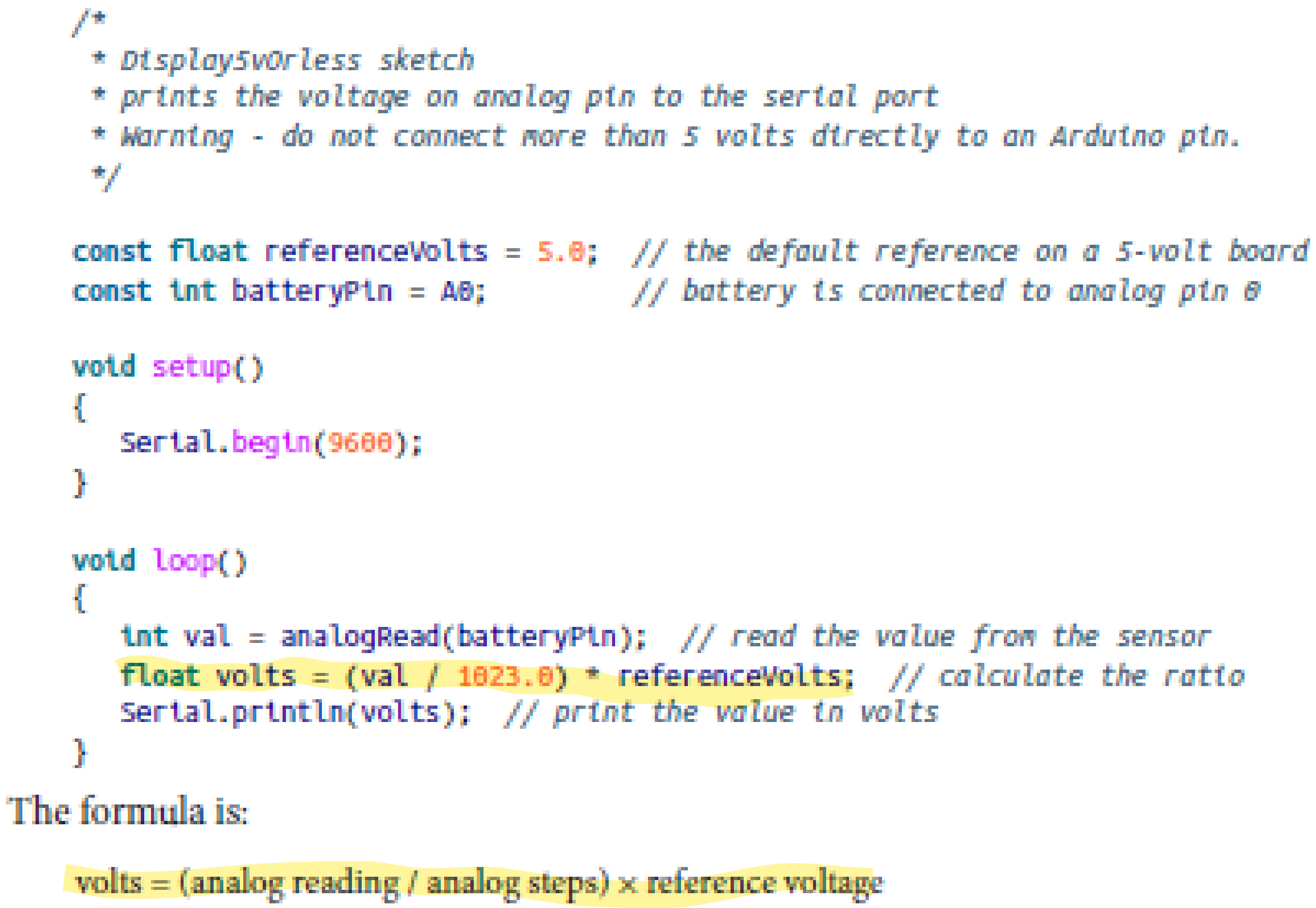

SCL
SDA
AREF
GND
13
12
11
10
9
8
7
6
5
4
3
2
TX 1
RX 0
DIGITAL
Arduino
IOREF
RESET
3V3
5V
Gnd
Gnd
Vin
ANALOG
0
1
2
3
4
5
Do not connect more than 5 volts directly to an Arduino pin!
(3.3 volts on 3.3 volt board)
+

*Figure 5-9. Measuring voltages up to 5 volts using 5V board*

## Measuring Voltages Up to 5V

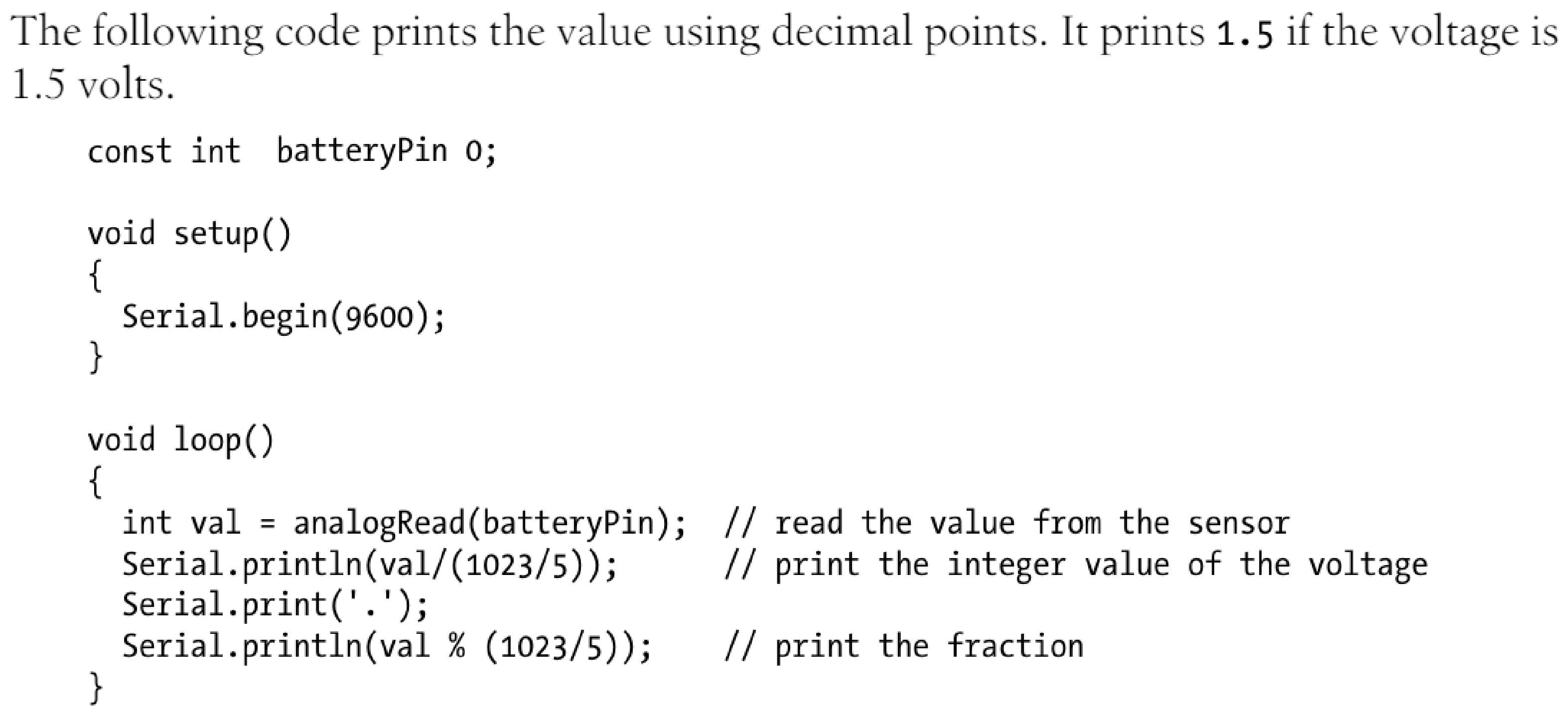


```
/*
 * Display5vOrless sketch
 * prints the voltage on analog pin to the serial port
 * Warning - do not connect more than 5 volts directly to an Arduino pin.
 */

const float referenceVolts = 5.0;  // the default reference on a 5-volt board
const int batteryPin = A0;         // battery is connected to analog pin 0

void setup()
{
   Serial.begin(9600);
}

void loop()
{
   int val = analogRead(batteryPin);  // read the value from the sensor
   float volts = (val / 1023.0) * referenceVolts;  // calculate the ratio
   Serial.println(volts);  // print the value in volts
}
```

The formula is:

volts = (analog reading / analog steps) × reference voltage

# Measuring Voltages Up to 5V



The following code prints the value using decimal points. It prints `1.5` if the voltage is 1.5 volts.

```
const int  batteryPin 0;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  int val = analogRead(batteryPin);  // read the value from the sensor
  Serial.println(val/(1023/5));      // print the integer value of the voltage
  Serial.print('.');
  Serial.println(val % (1023/5));    // print the fraction
}
```

# Responding to Changes in Voltage

## Problem

You want to monitor one or more voltages and take some action when the voltage rises or falls below a threshold. For example, you want to flash an LED to indicate a low battery level—perhaps to start flashing when the voltage drops below a warning threshold and increasing in urgency as the voltage drops further.

# Responding to Changes in Voltage

```
/*
 * RespondingToChanges sketch
 * flash an LED to indicate low voltage levels
 */

long batteryFull       = 1500;  // millivolts for a full battery
long warningThreshold  = 1200;  // Warning level in millivolts - LED flashes
long criticalThreshold = 1000;  // Critical voltage level - LED stays on

const int batteryPin = A0;
const int ledPin = LED_BUILTIN;

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

```
 // function to flash an led with specified on/off time
 void flash(int blinkDelay)
 {
   digitalWrite(ledPin, HIGH);
   delay(blinkDelay);
   digitalWrite(ledPin, LOW);
   delay(blinkDelay);
 }
```

mapped_value = map(value, fromLow, fromHigh, toLow, toHigh);
mapped_value = (value - fromLow) * (toHigh - toLow) / (fromHigh - fromLow) + toLow

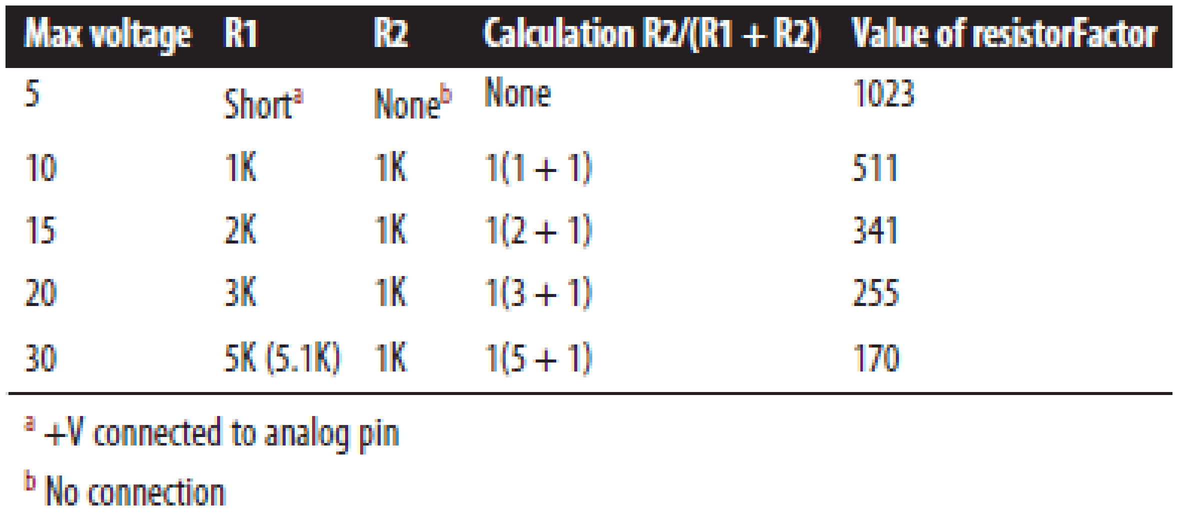
```
void loop()
{ // Read an analog value (0 to 1023)
  int val = analogRead(batteryPin);    // read the value from the sensor
  int mv = map(val, 0, 1023, 0, 5000);
  if(mv < criticalThreshold) {
    digitalWrite(ledPin, HIGH);
  }
  else if (mv < warningThreshold) {
    int blinkDelay = map(mv, criticalThreshold, batteryFull, 0, 250);
    flash(blinkDelay);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
  delay(1);
}
```

# Measuring Voltages More than 5V (Voltage Dividers)

- Problem

You want to measure voltages greater than 5 volts. For example, you want to display the voltage of a 9V battery and trigger an alarm LED when the voltage falls below a certain level.

*Table 5-4. Resistor values*



| Max voltage | R1 | R2 | Calculation R2/(R1 + R2) | Value of resistorFactor |
|---|---|---|---|---|
| 5 | Short[a] | None[b] | None | 1023 |
| 10 | 1K | 1K | 1(1 + 1) | 511 |
| 15 | 2K | 1K | 1(2 + 1) | 341 |
| 20 | 3K | 1K | 1(3 + 1) | 255 |
| 30 | 5K (5.1K) | 1K | 1(5 + 1) | 170 |

[a] +V connected to analog pin

[b] No connection

# Measuring Voltages More than 5V (Voltage Dividers)

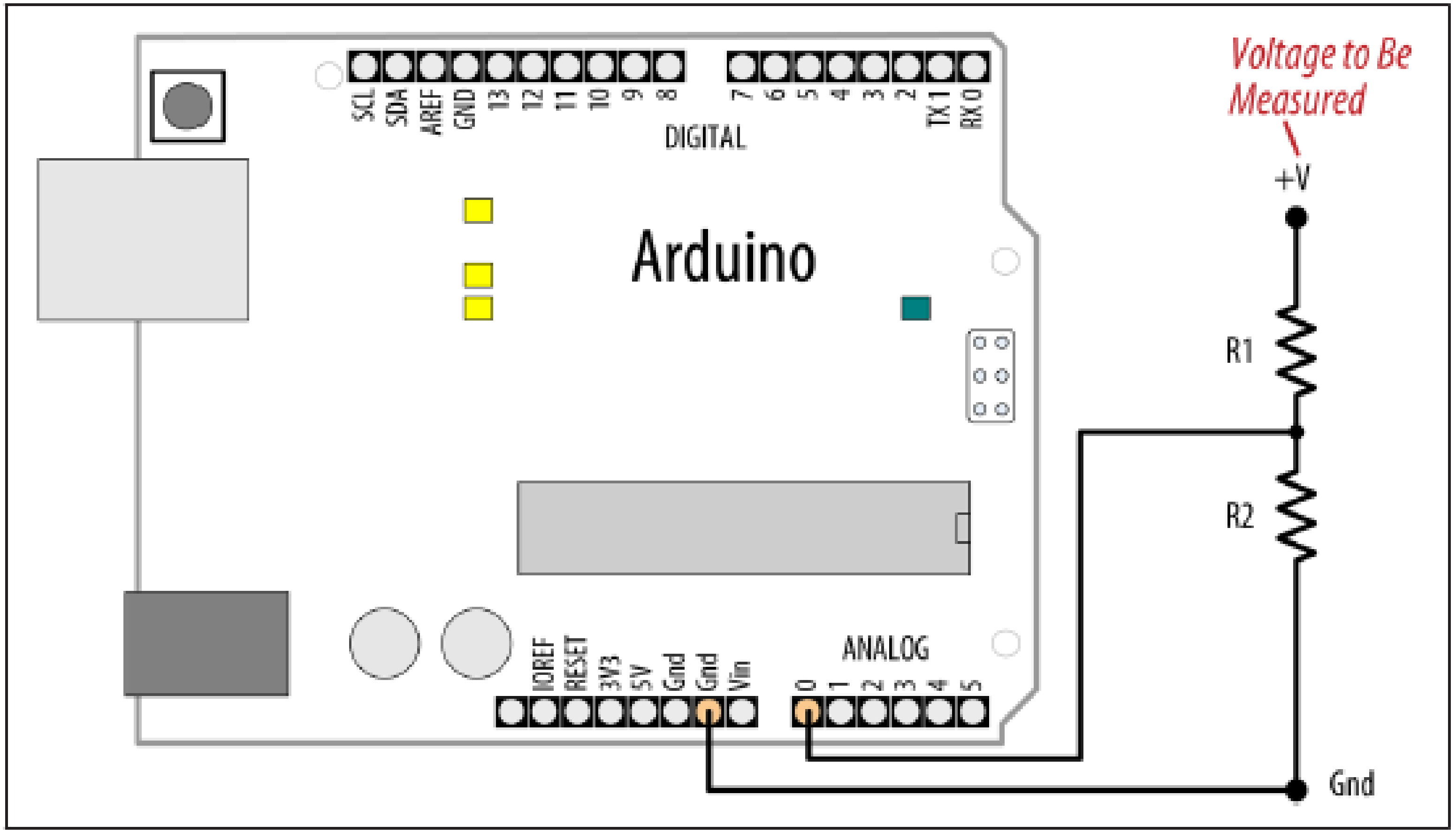
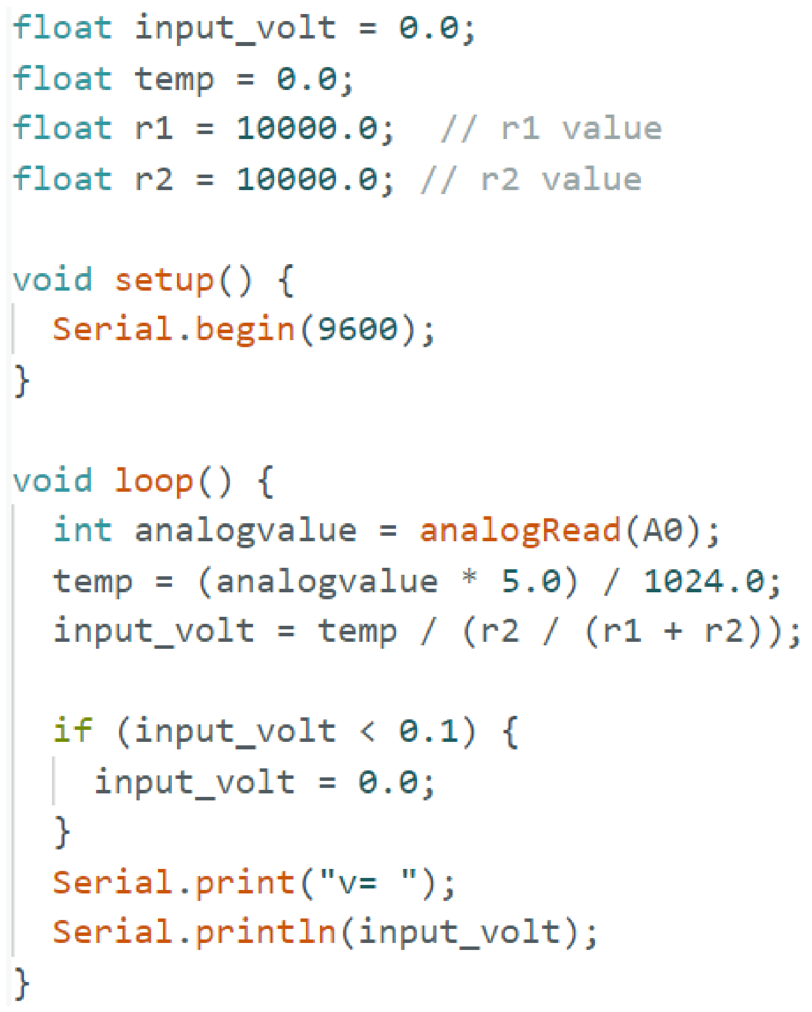


*Figure 5-10. Voltage divider for measuring voltages greater than 5 volts*

# Measuring Voltages More than 5V (Voltage Dividers)



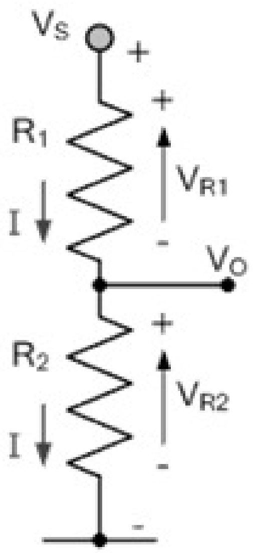
```
float input_volt = 0.0;
float temp = 0.0;
float r1 = 10000.0;  // r1 value
float r2 = 10000.0; // r2 value

void setup() {
  Serial.begin(9600);
}

void loop() {
  int analogvalue = analogRead(A0);
  temp = (analogvalue * 5.0) / 1024.0;
  input_volt = temp / (r2 / (r1 + r2));

  if (input_volt < 0.1) {
    input_volt = 0.0;
  }
  Serial.print("v= ");
  Serial.println(input_volt);
}
```

# Measuring Voltages More than 5V (Voltage Dividers)

A **voltage divider** is a simple circuit which turns a large voltage into a smaller one. Using just two series resistors and an input voltage, we can create an output voltage that is a fraction of the input. Voltage dividers are one of the most fundamental circuits in electronics.

Voltage Divider Equation

**Vout=Vin. R2/R1+R2**     Vin == VR2



*Resistive Type*

The above equation states that the Vout (o/p voltage) is directly proportional to the Vin (input voltage) and the ratio of two resistors R1 and R2.