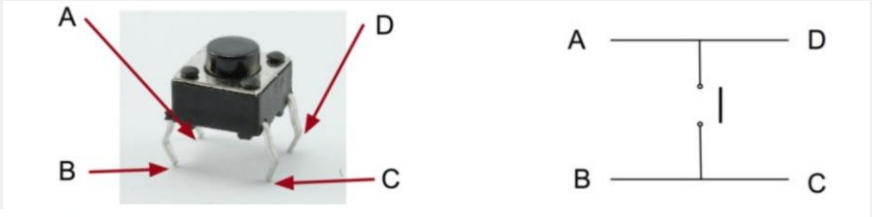
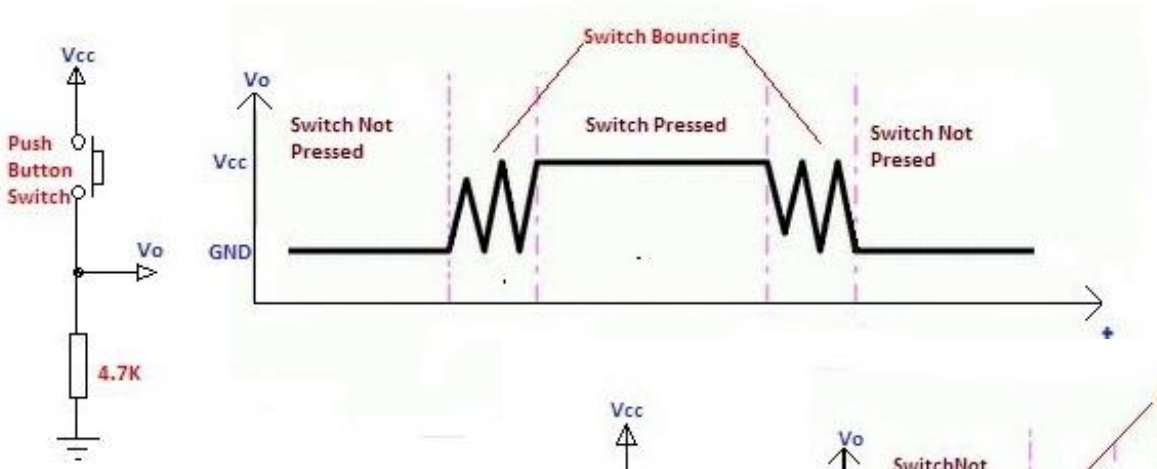
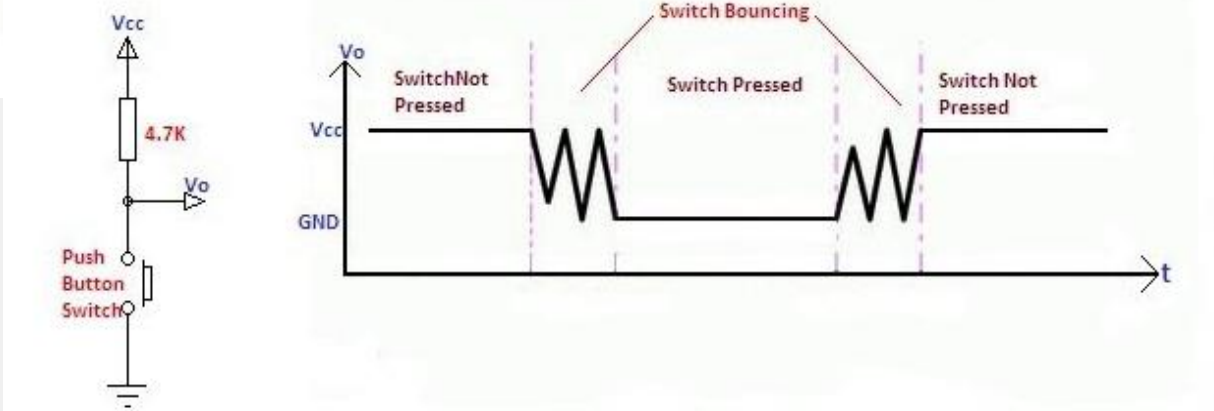
CH_5_Simple Digital and Analog Input

Switch Bouncing Problem

Switch bouncing



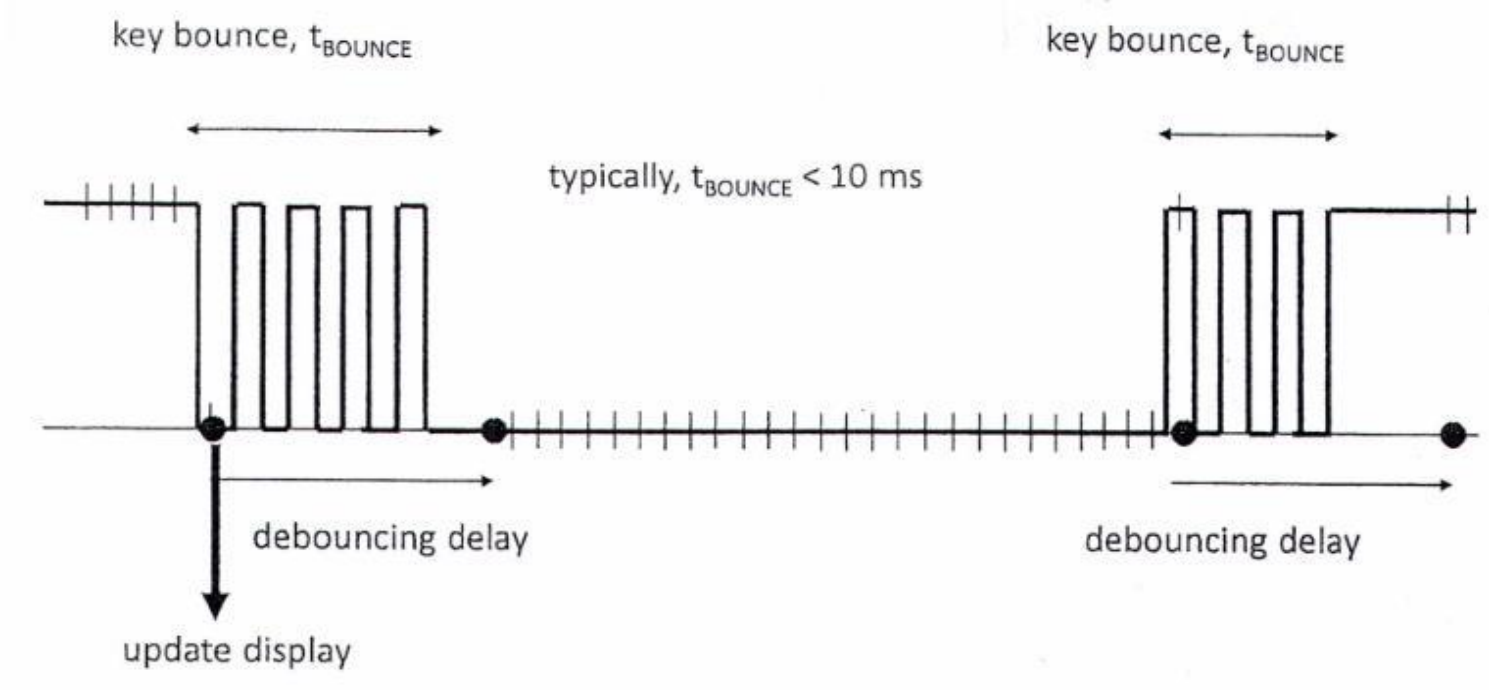




Debouncing Solutions

1) hardware solution (RC circuit).
2) Software solution (no circuit).

It is possible to ignore readings of the bouncing switch (During bounce interval which is about 10 micro-seconds), then consider readings of the switch state after it is stable !



Switch Bouncing Problem & Debouncing Solution

- **Problem:**
  - **You want to avoid false readings due to *contact bounce* (contact *bounce* produces faulty signals for a short time when the switch contacts close or open). The process of eliminating faulty signal reading is called *debouncing*.**

- **Solution:**
  - **There are *many* ways to solve this problem; here is one software solution using Recipe 5.1 for the wiring shown in Figure 5-3 :**

Switch Bouncing Problem & Debouncing Solution

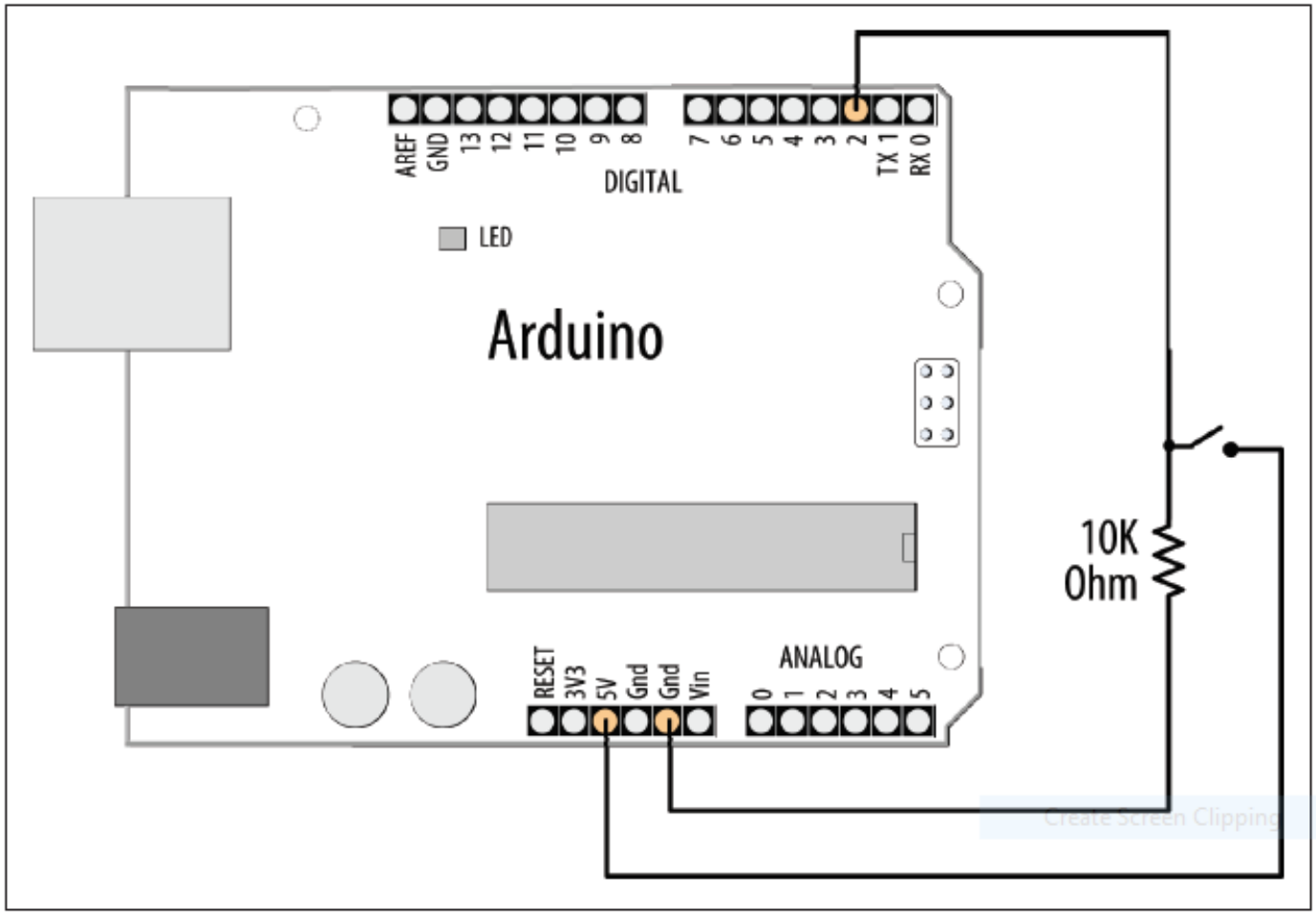


Figure 5-3. Switch connected using pull-down resistor

Switch Bouncing Problem & Debouncing Solution

```
/*
 * Debounce sketch
 * a switch connected to pin 2 lights the LED on pin 13
 * debounce logic prevents misreading of the switch state
 */

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 in 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
      }
  }
```

Switch Bouncing Problem & Debouncing Solution

```
  // here when the switch state has been stable longer than the debounce period
  return state;
}

void setup()
{
  pinMode(inputPin, INPUT);
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  if (debounce(inputPin))
  {
    digitalWrite(ledPin, HIGH);
  }
}
```

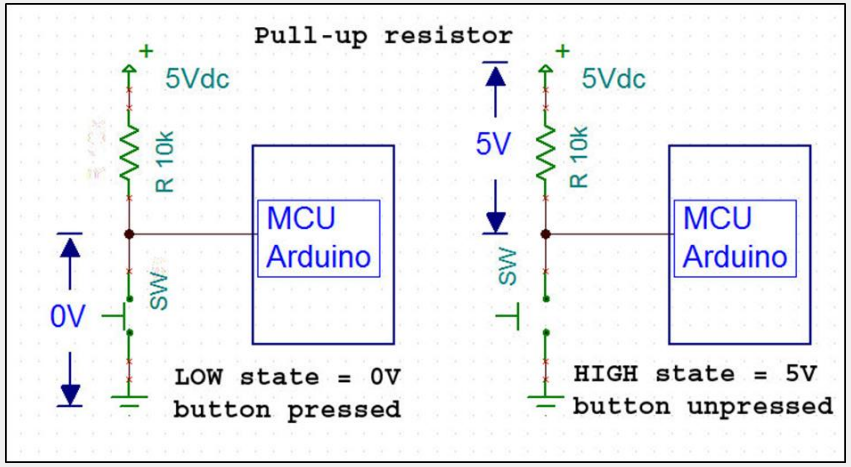
The debounce function is called (used) with the pin number of the switch you want to debounce; the function returns true if the switch is pressed and stable. It returns false if it is not pressed or not yet stable.

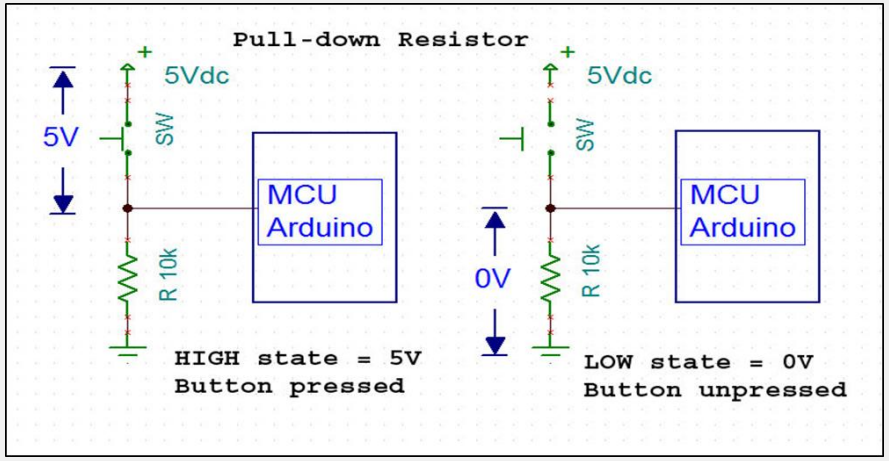
Switch Bouncing Problem & Debouncing Solution

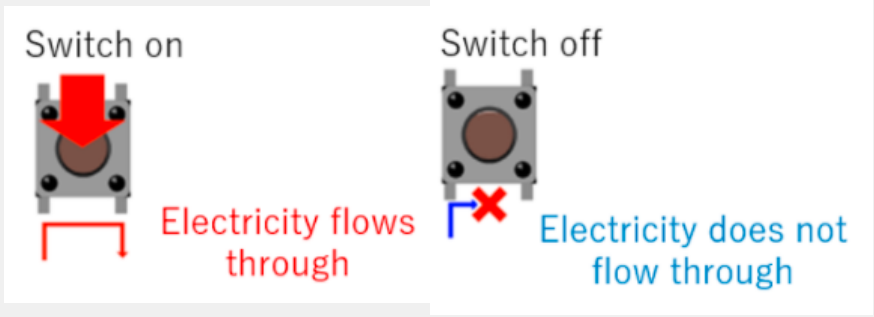
This debounce() function will work for any number of switches, but you must ensure that the pins used are in input mode.

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. You can use the code shown in Recipe 5.4 to overcome this problem.

How to avoid the floating state ?







Determining How Long a Switch Is Pressed:

- **Problem:**
  - **Your application wants to determine how long a switch has been in its current state. Or you want to increment a value while a switch is pushed. Or you want some rate to increase as long as a switch is being held (the way many electronic clocks are set). Or you want to know if a switch has been pressed long enough for the reading to be stable (see Recipe 5.3).**
- **Solution:**
  - **The following sketch demonstrates the setting of a countdown timer. The wiring is the same as in Figure 5-5 from Recipe 5.2. 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 ten.**

Determining How Long a Switch Is Pressed:

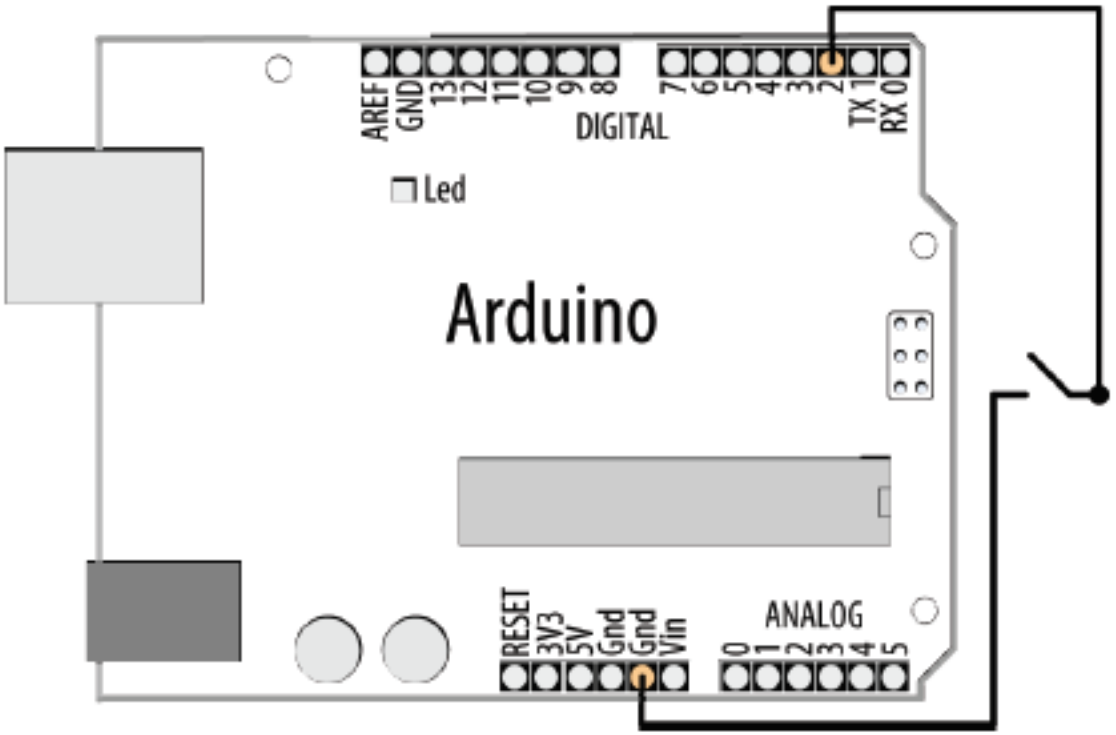


Figure 5-5. Switch wired for use with internal pull-up resistor

Determining How Long a Switch Is Pressed:

Releasing the switch starts the countdown, and when the count reaches zero, a pin is set HIGH (in this example, lighting an LED):

```
/*
 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;           // the time in milliseconds required
                                        // for the switch to be stable
const int  fastIncrement = 1000;        // increment faster after this many
                                        // milliseconds
const int  veryFastIncrement = 4000;    // and increment even faster after
                                        // this many milliseconds
int count = 0;                          // count decrements every tenth of a
                                        // second until reaches 0
```

Determining How Long a Switch Is Pressed:

```
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 = count + 10;
  else if ( duration > fastIncrement)
    count = count + 4;
  else if ( duration > debounceTime)
    count = count + 1;

  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 = count - 1;           // and decrement the count
    }
  }
```

Determining How Long a Switch Is Pressed:

```
  Serial.println(count);
  delay(100);
}

// return the time in milliseconds that the switch has been in pressed (LOW)
long switchTime()
{
  // these variables are static  - see Discussion for an explanation
  static unsigned long startTime = 0;  // the time the switch state change was
first detected
  static boolean state;                // the current state of the switch

  if(digitalRead(inPin) != state) // check to see if the switch has changed state
  {
    state = ! state;       // yes, invert the state
    startTime = millis();  // store the time
  }
  if( state == LOW)
    return millis() - startTime;   // switch pushed, return time in milliseconds
  else
    return 0; // return 0 if the switch is not pushed (in the HIGH state);
}
```

Reading a Keypad:

- **Problem:**
  - **You have a matrix keypad and want to read the key presses in your sketch. For example, you have a telephone-style keypad similar to the SparkFun 12-button keypad (Spark-Fun COM-08653).**

- **Solution:**
  - **Wire the rows and columns from the keypad connector to the Arduino, as shown in Figure 5-6.**

Reading a Keypad:

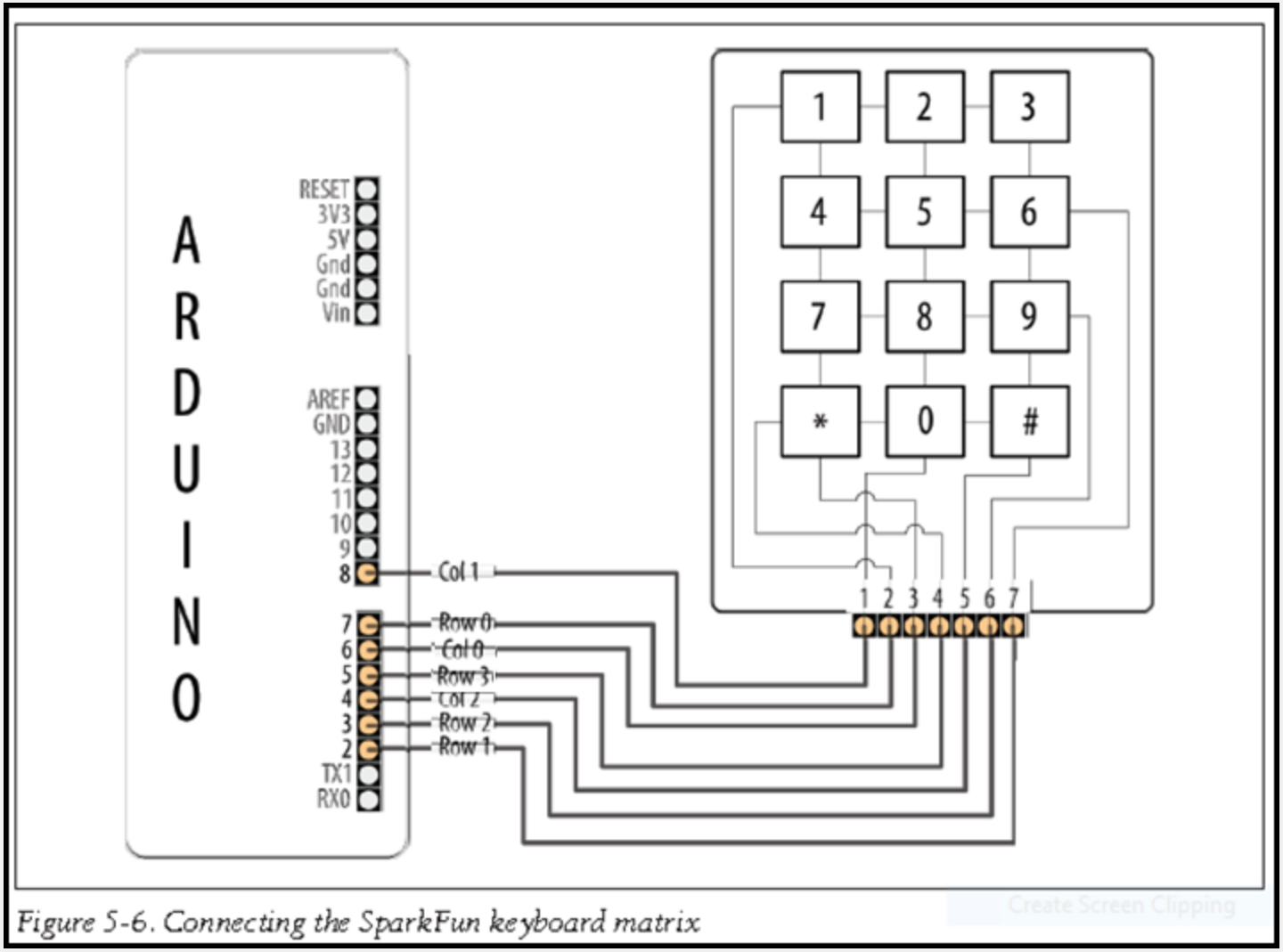


Figure 5-6. Connecting the SparkFun keyboard matrix

Reading a Keypad:

If you've wired your Arduino and keypad as shown in Figure 5-6, the following sketch will print key presses to the Serial Monitor:

```
/*
  Keypad sketch
  prints the key pressed on a keypad to the serial port
*/

const int numRows = 4;       // number of rows in the keypad
const int numCols = 3;       // number of columns
const int debounceTime = 20; // number of milliseconds for switch to be stable
```

```
// keymap defines the character returned when the corresponding key is pressed
const char keymap[numRows][numCols] = {
  { '1', '2', '3'  } ,
  { '4', '5', '6'  } ,
  { '7', '8', '9'  } ,
  { '*', '0', '#'  }
};

// this array determines the pins used for rows and columns
const int rowPins[numRows] = { 7, 2, 3, 5 }; // Rows 0 through 3
const int colPins[numCols] = { 6 , 8, 4 };    // Columns 0 through 2
```

Reading a Keypad:

```
void setup()
{
  Serial.begin(9600);
  for (int row = 0; row < numRows; row++)
  {
    pinMode(rowPins[row],INPUT);        // Set row pins as input
    digitalWrite(rowPins[row],HIGH);    // turn on Pull-ups
  }
  for (int column = 0; column < numCols; column++)
  {
    pinMode(colPins[column],OUTPUT);     // Set column pins as outputs
                                         // for writing
    digitalWrite(colPins[column],HIGH);  // Make all columns inactive
  }
}
```

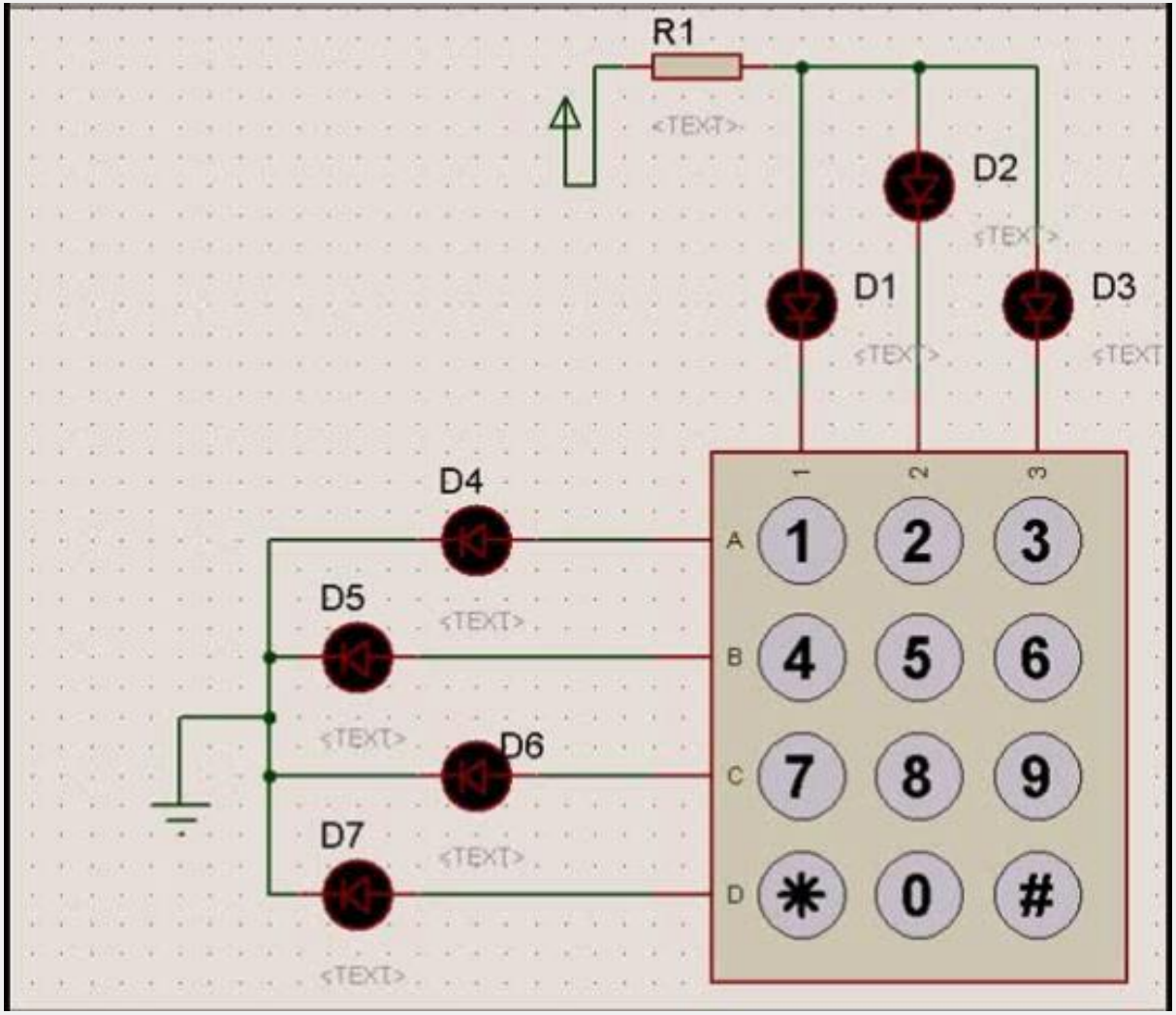
Reading a Keypad:

```
void loop()
{
  char key = getKey();
  if( key != 0) {       // if the character is not 0 then
                        // it's a valid key press
    Serial.print("Got key ");
    Serial.println(key);
  }
}

// returns with the key pressed, or 0 if no key is pressed
char getKey()
{
  char key = 0;                                  // 0 indicates no key pressed

  for(int column = 0; column < numCols; column++)
  {
    digitalWrite(colPins[column],LOW);          // Activate the current column.
    for(int row = 0; row < numRows; row++)      // Scan all rows for
                                                // a key press.
    {
      if(digitalRead(rowPins[row]) == LOW)      // Is a key pressed?
      {
        delay(debounceTime);                    // debounce
        while(digitalRead(rowPins[row]) == LOW)
            ;                                   // wait for key to be released
```

Reading a Keypad:

```
        key = keymap[row][column];              // Remember which key
                                                // was pressed.
      }
    }
    digitalWrite(colPins[column],HIGH);     // De-activate the current column.
  }
  return key;  // returns the key pressed or 0 if none
}
```

Reading a Keypad:



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. This recipe uses the popular 4051 chip connected to Arduino as shown in Figure 5-8. Your analog inputs get connected 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:**

Reading More Than Six Analog Inputs:

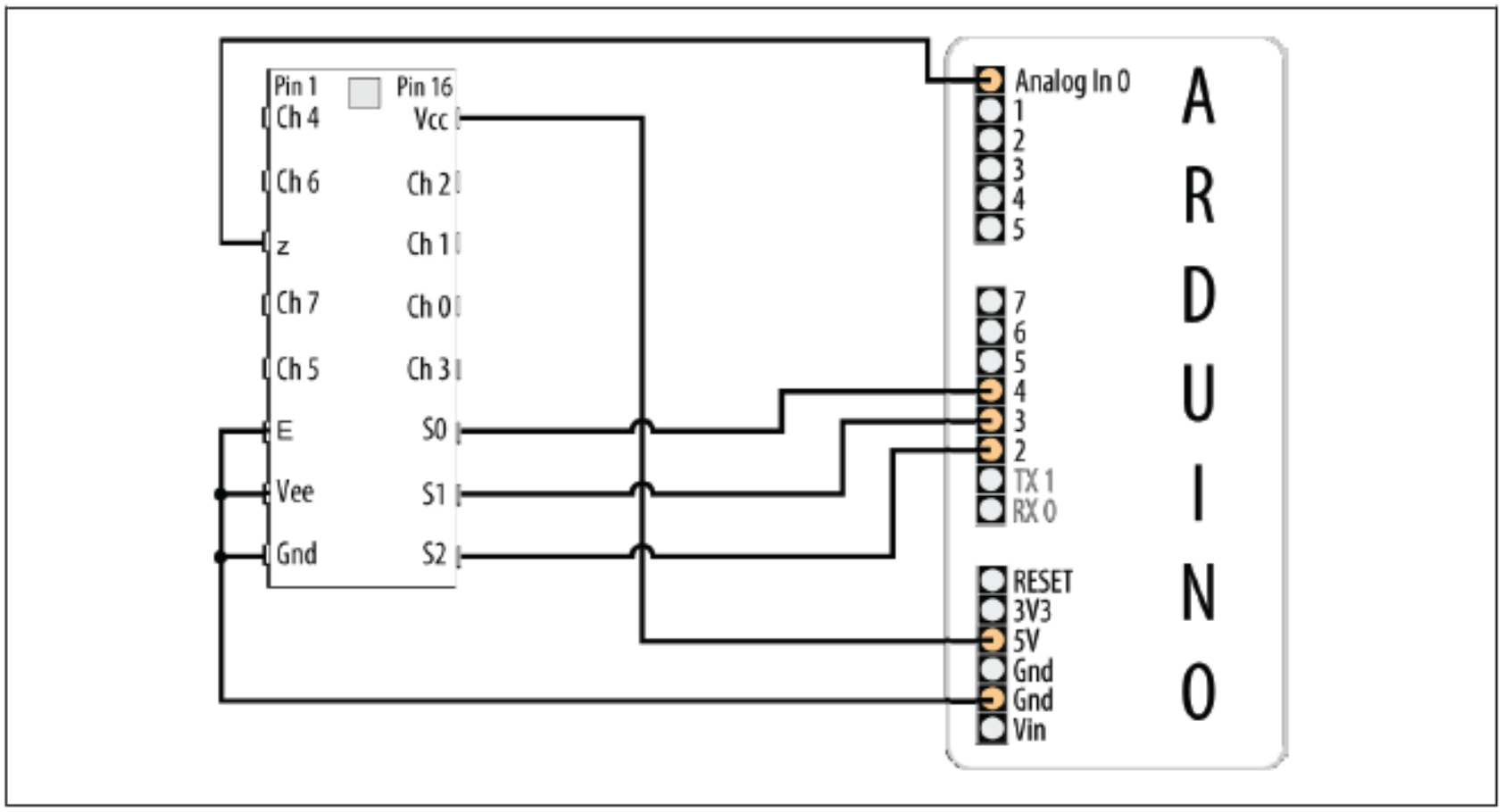


Figure 5-8. The 4051 multiplexer connected to Arduino

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 = 0;      // 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
```

Reading More Than Six Analog Inputs:

```
      int isBitSet =  bitRead(channel, bit); // true if given bit set in channel
      digitalWrite(pin, isBitSet);
   }
   return analogRead(analogPin);
}

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:

Discussion

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; see Table 5-3.

Table 5-3. Truth table for 4051 multiplexer

Selector pins			Selected input
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

Displaying 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.**

- **Solution:**
  - **Use AnalogRead to measure the voltage on an analog pin. Convert the reading to a voltage by using the ratio of the reading to the reference voltage (5 volts), as shown in Figure 5-9.**

Displaying Voltages Up to 5V:

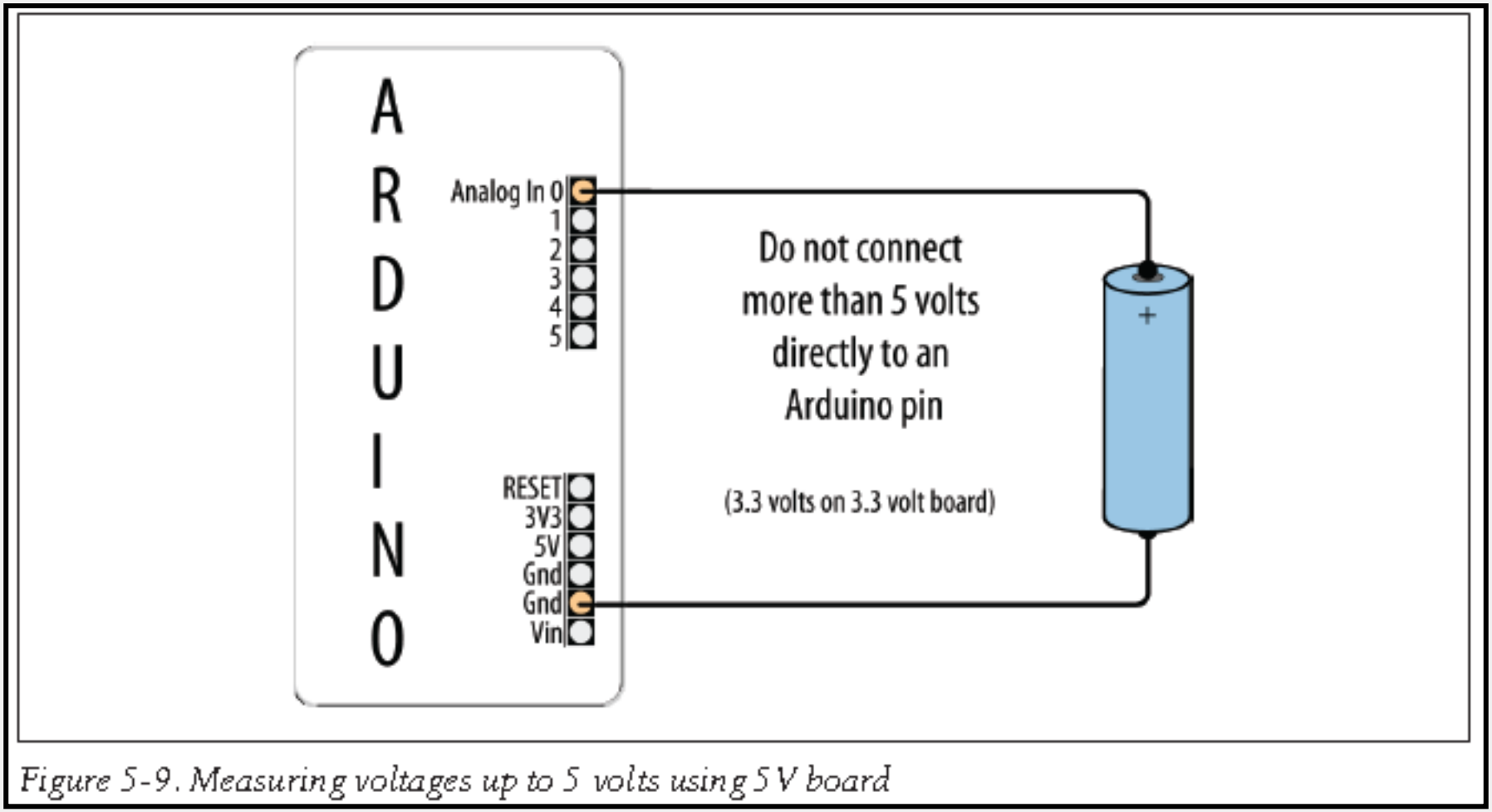


Figure 5-9. Measuring voltages up to 5 volts using 5V board

Displaying Voltages Up to 5V:

The simplest solution uses a floating-point calculation to print the voltage; this example sketch calculates and prints the ratio as a voltage:

```
/*
 * 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 = 0;          // battery is connected to analog pin 0

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

Displaying Voltages Up to 5V:

```
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

Printing a floating-point value to the serial port with println will format the value to two decimal places.

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.**

- **Solution:**
  - **You can use the connections shown in Figure 5-7 in Recipe 5.9, but here we'll compare the value from analogRead to see if it drops below a threshold. This example starts flashing an LED at 1.2 volts and increases the on-to-off time as the voltage decreases below the threshold. If the voltage drops below a second threshold, the LED stays lit:**

Responding to Changes in Voltage:

```
long warningThreshold = 1200;
long criticalThreshold = 500;

const int batteryPin = 0;
const int ledPin = 13;

void setup()
{
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  int val = analogRead(batteryPin);
  Serial.println(val);
  if(val<((warningThreshold*1023L)/5000) && val>((criticalThreshold*1023L)/5000))
  {
    flash(val);
  }
  else if(val<=((criticalThreshold*1023L)/5000))
  {
    digitalWrite(ledPin,HIGH);
  }
  else
  {
    digitalWrite(ledPin,LOW);
  }
}
```

Responding to Changes in Voltage:

```
30  void flash(int percent)
31  {
32    digitalWrite(ledPin,HIGH);
33    delay(percent+1);
34    digitalWrite(ledPin,LOW);
35    delay(percent+1);
36  }
```

Responding to Changes in Voltage:

Discussion

The highlighted line in this sketch calculates the ratio of the value read from the analog port to the value of the threshold voltage. For example, with a warning threshold of 1 volt and a reference voltage of 5 volts, you want to know when the analog reading is one-fifth of the reference voltage. The expression `1023L` tells the compiler that this is a long integer (a 32-bit integer; see Recipe 2.2), so the compiler will promote all the variables in this expression to long integers to prevent overflowing the capacity of an `int` (a normal 16-bit integer).

When reading analog values, you can work in the units that are returned from `analogRead`—ranging from 0 to 1023—or you can work in the actual voltages they represent (see Recipe 5.7). As in this recipe, if you are not displaying voltage, it's simpler and more efficient to use the output of `analogRead` directly.

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.**

- **Solution:**
  - **Use a solution similar to Recipe 5.9, but connect the voltage through a voltage divider (see Figure 5-10). For voltages up to 10 volts, you can use two 4.7K ohm resistors. For higher voltages, you can determine the required resistors using Table 5-4.**

Measuring Voltages More Than 5V (Voltage Dividers):

Table 5-4. Resistor values

Max voltage	R1	R2	Calculation R2/(R1 + R2)	value of `resistorFactor`
5	Short (+V connected to analog pin)	None (Gnd connected to Gnd)	None	1023
10	1K	1K	1(1 + 1)	511
15	2K	1K	1(2 + 1)	341
20	3K	1K	1(3 + 1)	255
30	4K (3.9K)	1K	1(4 + 1)	170

Measuring Voltages More Than 5V (Voltage Dividers):

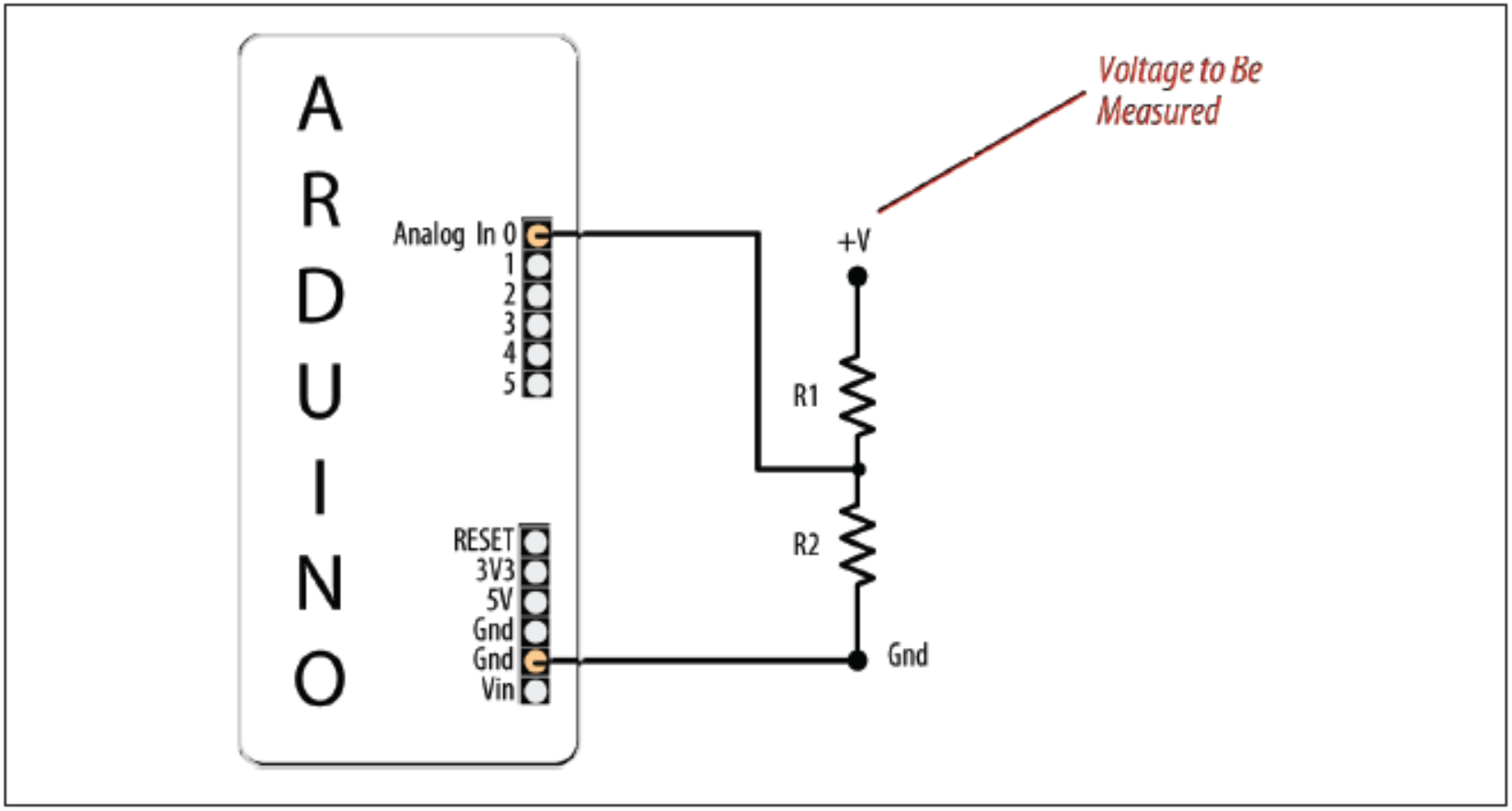
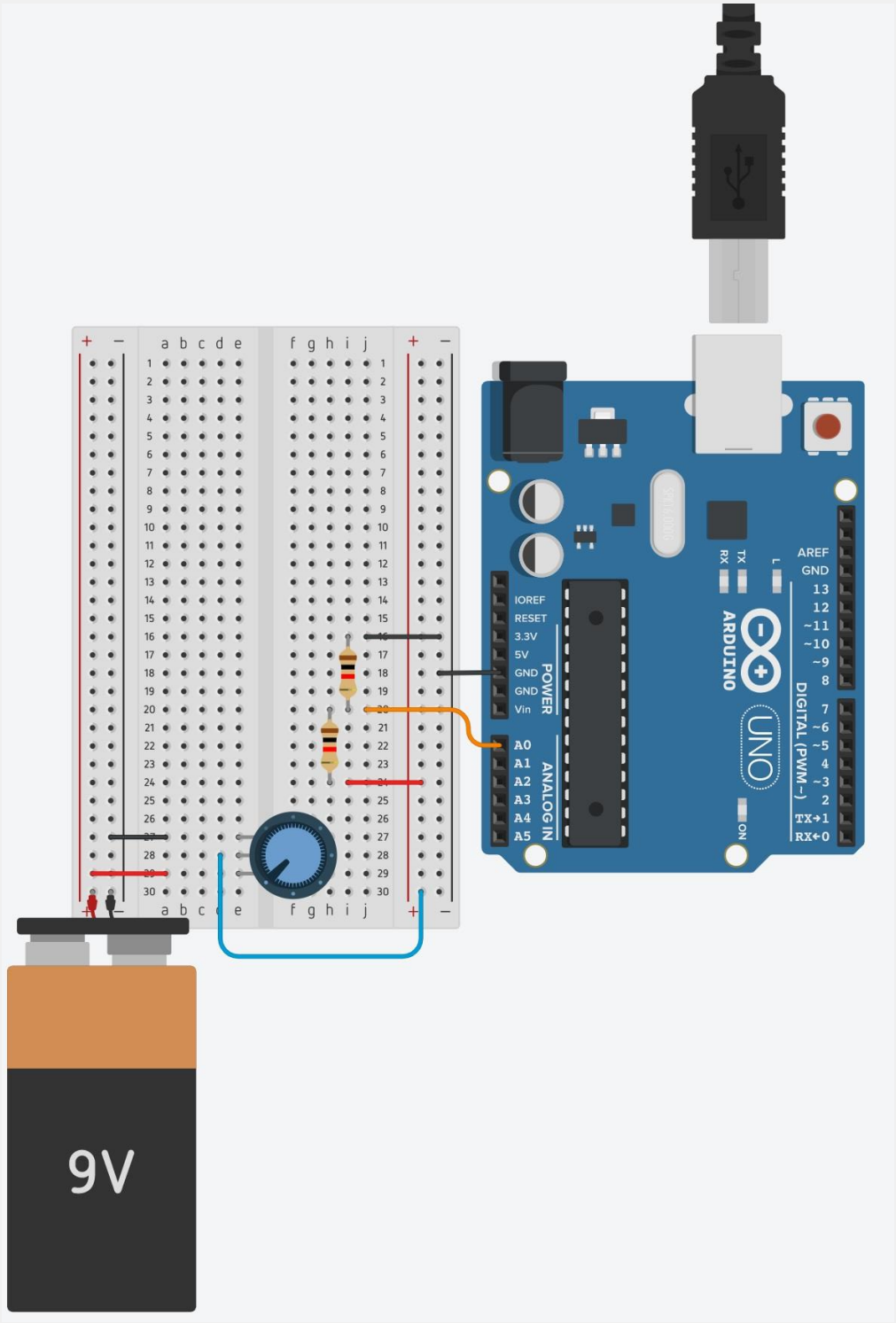


Figure 5-10. Voltage divider for measuring voltages greater than 5 volts


9V
ARDUINO
UNO
POWER
ANALOG IN
DIGITAL (PWM~)
IOREF
RESET
3.3V
5V
GND
GND
Vin
A0
A1
A2
A3
A4
A5
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
TX
RX
L
ON

Measuring Voltages More Than 5V (Voltage Dividers):

Select the row with the highest voltage you need to measure to find the values for the two resistors:

```
/*
  DisplayMoreThan5V sketch
  prints the voltage on analog pin to the serial port
  Do not connect more than 5 volts directly to an Arduino pin.
*/

const float referenceVolts = 5;        // the default reference on a 5-volt board
//const float referenceVolts = 3.3;  // use this for a 3.3-volt board

const float R1 = 1000; // value for a maximum voltage of 10 volts
const float R2 = 1000;
// determine by voltage divider resistors, see text
const float resistorFactor = 1023.0 / (R2/(R1 + R2));
const int batteryPin = 0;          // +V from battery is connected to analog pin 0
```

Measuring Voltages More Than 5V (Voltage Dividers):

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

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

Measuring Voltages More Than 5V (Voltage Dividers):

Discussion

Like the previous analog recipes, this recipe relies on the fact that the `analogRead` value is a ratio of the measured voltage to the reference. But because the measured voltage is divided by the two dropping resistors, the `analogRead` value needs to be multiplied to get the actual voltage. The code here is similar to that in Recipe 5.7, but the value of `resistorFactor` is selected based on the voltage divider resistors as shown in Table 5-4:

```
const int resistorFactor = 511;    // determine by voltage divider resistors,
see Table 5-3
```

The value read from the analog pin is divided not by 1,023, but by a value determined by the dropping resistors:

```
float volts = (val / resistorFactor) * referenceVolts ;  // calculate the ratio
```

The calculation used to produce the table is based on the following formula: the output voltage is equal to the input voltage times R2 divided by the sum of R1 and R2. In the example where two equal-value resistors are used to drop the voltage from a 9V battery by half, `resistorFactor` is 511 (half of 1,023), so the value of the `volts` variable will be twice the voltage that appears on the input pin. With resistors selected for 10 volts, the analog reading from a 9V battery will be approximately 920.