SIK Guide Code -- ver. codebender08.12.2016

redirect to codebender SIK example code: www.sparkfun.com/sikcodebender

This is a complete print-able version of the example code used for the 16 circuits in the SparkFun Inventor's Kit. This version of the code differs slightly from the standard SIK guide code that is described in the booklet in that many of the comments have been simplified or removed for greater readability.

These files are on the cloud, and as such, they may change. The example code here was pulled on Aug. 12, 2016. Please note that there may be minor changes over time. Line numbers have been included in this version to provide reference points for you to provide instruction and guidance. If you have any questions, please e-mail us at education@sparkfun.com

Table of Contents

(This page intentionally left blank.)

Example sketch 01 -- BLINKING A LED

https://codebender.cc/sketch:77046
Fritzing diagram

```
/*************************************************************************
 * SparkFun Inventor's Kit
 * Example sketch 01 -- BLINKING A LED
 *
 * Turn an LED on for one second, off for one second, and repeat forever.
 *
 * This sketch was written by SparkFun Electronics, with lots of help from
 * the Arduino community. This code is completely free for any use.
 * Visit http://sparkfun.com/sikguide for SIK information.
 * Visit http://www.arduino.cc to learn about the Arduino.
 *
 * Version 2.0 6/2012 MDG
*************************************************************************/

void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);   // Turn on the LED
  delay(1000);               // Wait for one second
  digitalWrite(13, LOW);     // Turn off the LED
  delay(1000);               // Wait for one second
}

/*************************************************************************
 * Try changing the 1000 in the above delay() functions to different numbers
 * and see how it affects the timing. Smaller values will make the loop run
 * faster. (Why?)
 *
 * Other challenges:
 *      - Decrease the delay to 10 ms. Can you still see it blink?
 *      - Find the smallest delay that you can still see a blink.
 *      - What is this frequency?
 *      - Modify the code above to resemble a heartbeat.
*************************************************************************/
```

Example sketch 02 -- POTENTIOMETER

https://codebender.cc/sketch:77047

Fritzing diagram

```
/*****************************************************************************
 * SparkFun Inventor's Kit
 * Example sketch 02 -- POTENTIOMETER
 *
 * Measure the position of a potentiometer and use it to control the blink
 * rate of an LED. Turn the knob to make it blink faster or slower!
 *
 * This sketch was written by SparkFun Electronics, with lots of help from
 * the Arduino community. This code is completely free for any use.
 * Visit http://sparkfun.com/sikguide for SIK information.
 * Visit http://www.arduino.cc to learn about the Arduino.
 *
 * Version 2.0 6/2012 MDG
/*****************************************************************************/

int sensorValue;
const int sensorPin = A0;     // The potentiometer is connected to analog pin 0
const int ledPin = 13;        // The LED is connected to digital pin 13

void setup() // this function runs once when the sketch starts up
{
  pinMode(ledPin, OUTPUT);
}

void loop() // this function runs repeatedly after setup() finishes
{
  sensorValue = analogRead(sensorPin);

  digitalWrite(ledPin, HIGH);       // Turn the LED on
  delay(sensorValue);               // Pause for sensorValue in milliseconds
  digitalWrite(ledPin, LOW);        // Turn the LED off
  delay(sensorValue);               // Pause for sensorValue in milliseconds
}
```

Example sketch 03 -- RGB LED

https://codebender.cc/sketch:77048

Fritzing diagram

```
/*****************************************************************
 * SparkFun Inventor's Kit
 * Example sketch 03 -- RGB LED
 *
 * Make an RGB LED display a rainbow of colors!
 *
 * This sketch was written by SparkFun Electronics,
 * with lots of help from the Arduino community.
 * Visit http://learn.sparkfun.com/products/2 for SIK information.
 * Visit http://www.arduino.cc to learn about the Arduino.
 *
 * Version 2.0 6/2012 MDG
 * Version 2.1 9/2014 BCH
 *****************************************************************/
const int RED_PIN = 9;
const int GREEN_PIN = 10;
const int BLUE_PIN = 11;

const int DISPLAY_TIME = 1000;  // used in mainColors() to determine the
// length of time each color is displayed.
void setup()
{
  pinMode(RED_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(BLUE_PIN, OUTPUT);
}

void loop()
{
  mainColors();       // Red, Green, Blue, Yellow, Cyan, Purple, White
  //  showSpectrum(); // Gradual fade from Red to Green to Blue to Red
}

/*****************************************************************
 * void mainColors()
 * This function displays the eight "main" colors that the RGB LED
 * can produce. If you'd like to use one of these colors in your
 * own sketch, you can copy and paste that section into your code.
/*****************************************************************/
void mainColors()
{
  // all LEDs off
  digitalWrite(RED_PIN, LOW);
  digitalWrite(GREEN_PIN, LOW);
  digitalWrite(BLUE_PIN, LOW);
  delay(DISPLAY_TIME);

  // Red
  digitalWrite(RED_PIN, HIGH);
  digitalWrite(GREEN_PIN, LOW);
  digitalWrite(BLUE_PIN, LOW);
  delay(DISPLAY_TIME);

  // Green
  digitalWrite(RED_PIN, LOW);
  digitalWrite(GREEN_PIN, HIGH);
```

```
  digitalWrite(BLUE_PIN, LOW);
  delay(DISPLAY_TIME);

  // Blue
  digitalWrite(RED_PIN, LOW);
  digitalWrite(GREEN_PIN, LOW);
  digitalWrite(BLUE_PIN, HIGH);
  delay(DISPLAY_TIME);

  // Yellow (Red and Green)
  digitalWrite(RED_PIN, HIGH);
  digitalWrite(GREEN_PIN, HIGH);
  digitalWrite(BLUE_PIN, LOW);
  delay(DISPLAY_TIME);

  // Cyan (Green and Blue)
  digitalWrite(RED_PIN, LOW);
  digitalWrite(GREEN_PIN, HIGH);
  digitalWrite(BLUE_PIN, HIGH);
  delay(DISPLAY_TIME);

  // Purple (Red and Blue)
  digitalWrite(RED_PIN, HIGH);
  digitalWrite(GREEN_PIN, LOW);
  digitalWrite(BLUE_PIN, HIGH);
  delay(DISPLAY_TIME);

  // White (turn all the LEDs on)
  digitalWrite(RED_PIN, HIGH);
  digitalWrite(GREEN_PIN, HIGH);
  digitalWrite(BLUE_PIN, HIGH);
  delay(DISPLAY_TIME);
}

/*****************************************************************
 * void showSpectrum()
 *
 * Steps through all the colors of the RGB LED, displaying a rainbow.
 * showSpectrum() calls a function showRGB() that translates a number
 * from 0 to 767 where 0 = all RED, 767 = all RED
 *
 * Breaking down tasks down into individual functions like this
 * makes your code easier to follow, and it allows.
 * parts of your code to be re-used.
*****************************************************************/

void showSpectrum()
{
  for (int x = 0; x <= 766; x++)
  {
        RGB(x);    // Increment x and call RGB() to progress through colors.
        delay(10); // Delay for 10ms(1/100th of a sec) - to help the "smoothing"
  }
}

/*****************************************************************
 * void RGB(int color)
 *
 * RGB(###) displays a single color on the RGB LED.
 * Call RGB(###) with the number of a color you want
 * to display. For example, RGB(0) displays pure RED, RGB(255)
 * displays pure green.
```

```
 *
 * This function translates a number between 0 and 767 into a
 * specific color on the RGB LED. If you have this number count
 * through the whole range (0 to 767), the LED will smoothly
 * change color through the entire spectrum.
 *
 * The "base" numbers are:
 * 0   = pure red
 * 255 = pure green
 * 511 = pure blue
 * 767 = pure red (again)
 *
 * Numbers between the above colors will create blends. For
 * example, 640 is midway between 512 (pure blue) and 767
 * (pure red). It will give you a 50/50 mix of blue and red,
 * resulting in purple.
/*****************************************************************/
void RGB(int color)
{
  int redIntensity;
  int greenIntensity;
  int blueIntensity;

// constrain the input value to a range of values from 0 to 767
  color = constrain(color, 0, 767);

        // if statement breaks down the "color" into three ranges:
  if (color <= 255)          // RANGE 1 (0 - 255) - red to green
  {
        redIntensity = 255 - color;  // red goes from on to off
        greenIntensity = color;      // green goes from off to on
        blueIntensity = 0;           // blue is always off
  }
  else if (color <= 511)  // RANGE 2 (256 - 511) - green to blue
  {
        redIntensity = 0;                    // red is always off
        greenIntensity = 511 - color;        // green on to off
        blueIntensity = color - 256;         // blue off to on
  }
  else                        // RANGE 3 ( >= 512)- blue to red
  {
        redIntensity = color - 512;          // red off to on
        greenIntensity = 0;                  // green is always off
        blueIntensity = 767 - color;         // blue on to off
  }

  // "send" intensity values to the Red, Green, Blue Pins using analogWrite()
  analogWrite(RED_PIN, redIntensity);
  analogWrite(GREEN_PIN, greenIntensity);
  analogWrite(BLUE_PIN, blueIntensity);
}
```

Example sketch 04 -- MULTIPLE LEDs

https://codebender.cc/sketch:77049

Fritzing diagram

```
/*************************************************************
 * SparkFun Inventor's Kit
 * Example sketch 04 -- MULTIPLE LEDs
 *
 *   Make eight LEDs dance. Dance LEDs, dance!
 * This sketch was written by SparkFun Electronics,
 * with lots of help from the Arduino community.
 * Visit http://learn.sparkfun.com/products/2 for SIK information.
 * Visit http://www.arduino.cc to learn about the Arduino.
 *
 * Version 2.0 6/2012 MDG
 * Version 2.1 9/2014 BCH
/*************************************************************/
int ledPins[] = {2,3,4,5,6,7,8,9};   // Defines an array to store the pin numbers of the 8 LEDs.
// An array is like a list variable that can store multiple numbers.
// Arrays are referenced or "indexed" with a number in the brackets [ ]. See the example below.

void setup()
{
  // setup all 8 pins as OUTPUT - notice that the list is "indexed" with a base of 0.
  pinMode(ledPins[0],OUTPUT);  // ledPins[0] = 2
  pinMode(ledPins[1],OUTPUT);  // ledPins[1] = 3
  pinMode(ledPins[2],OUTPUT);  // ledPins[2] = 4
  pinMode(ledPins[3],OUTPUT);  // ledPins[3] = 5
  pinMode(ledPins[4],OUTPUT);  // ledPins[4] = 6
  pinMode(ledPins[5],OUTPUT);  // ledPins[5] = 7
  pinMode(ledPins[6],OUTPUT);  // ledPins[6] = 8
  pinMode(ledPins[7],OUTPUT);  // ledPins[7] = 9
}

void loop()
{
  oneAfterAnother();  // Light up all the LEDs in turn

  //oneOnAtATime();          // Turn on one LED at a time,

  //pingPong();              // Light the LEDs middle to the edges

  //marquee();               // Chase lights like you see on signs

  //randomLED();             // Blink LEDs randomly
}

/*************************************************************
 * oneAfterAnother()
 *
 * This function does exactly the same thing as oneAfterAnotherNoLoop(),
 * but it takes advantage of for() loops and the array to do it with
 * much less typing.
/*************************************************************/
void oneAfterAnother()
{
  int index;
  int delayTime = 100; // milliseconds to pause between LEDs
                       // make this smaller for faster switching

```

```
  // Turn all the LEDs on:
  for(index = 0; index <= 7; index = index + 1)  // step through index from 0 to 7
  {
        digitalWrite(ledPins[index], HIGH);
        delay(delayTime);
  }

  // Turn all the LEDs off:
  for(index = 7; index >= 0; index = index - 1)  // step through index from 7 to 0
  {
        digitalWrite(ledPins[index], LOW);
        delay(delayTime);
  }
}

/*****************************************************************
 * oneOnAtATime()
 *
 * This function will step through the LEDs, lighting only one at
 * a time. It turns each LED ON and then OFF before going to the
 * next LED.
/*****************************************************************/

void oneOnAtATime()
{
  int index;
  int delayTime = 100; // milliseconds to pause between LEDs
                       // make this smaller for faster switching

  for(index = 0; index <= 7; index = index + 1)   // step through the LEDs, from 0 to 7
  {
        digitalWrite(ledPins[index], HIGH);  // turn LED on
        delay(delayTime);                    // pause to slow down
        digitalWrite(ledPins[index], LOW);   // turn LED off
  }

  for(index = 7; index >= 0; index = index - 1)   // step through the LEDs, from 7 to 0
  {
        digitalWrite(ledPins[index], HIGH);  // turn LED on
        delay(delayTime);                    // pause to slow down
        digitalWrite(ledPins[index], LOW);   // turn LED off
  }
}


/*****************************************************************
 * pingPong()
 *
 * This function will step through the LEDs, lighting one at at
 * time in both directions. There is no delay between the LED off
 * and turning on the next LED. This creates a smooth pattern for
 * the LED pattern.
/*****************************************************************/
void pingPong()
{
  int index;
  int delayTime = 100; // milliseconds to pause between LEDs

  for(index = 0; index <= 7; index = index + 1)   // step through the LEDs, from 0 to 7
  {
        digitalWrite(ledPins[index], HIGH);  // turn LED on
        delay(delayTime);                    // pause to slow down
```

```
120         digitalWrite(ledPins[index], LOW);   // turn LED off
121   }
122
123   for(index = 7; index >= 0; index = index - 1)   // step through the LEDs, from 7 to 0
124   {
125         digitalWrite(ledPins[index], HIGH);  // turn LED on
126         delay(delayTime);                    // pause to slow down
127         digitalWrite(ledPins[index], LOW);   // turn LED off
128   }
129 }
130
131 /*****************************************************************
132  * marquee()
133  *
134  * This function will mimic "chase lights" like those around signs.
135 /*****************************************************************/
136 void marquee()
137 {
138   int index;
139   int delayTime = 200; // milliseconds to pause between LEDs
140
141   // Step through the first four LEDs
142   // (We'll light up one in the lower 4 and one in the upper 4)
143
144   for(index = 0; index <= 3; index++) // Step from 0 to 3
145   {
146         digitalWrite(ledPins[index], HIGH); // Turn a LED on
147         digitalWrite(ledPins[index+4], HIGH);  // Skip four, and turn that LED on
148         delay(delayTime);                    // Pause to slow down the sequence
149         digitalWrite(ledPins[index], LOW);  // Turn the LED off
150         digitalWrite(ledPins[index+4], LOW);   // Skip four, and turn that LED off
151   }
152 }
153
154 /*****************************************************************
155  * randomLED()
156  *
157  * This function will turn on random LEDs. Can you modify it so it
158  * also lights them for random times?
159 /*****************************************************************/
160 void randomLED()
161 {
162   int index;
163   int delayTime;
164
165   index = random(8);    // pick a random number between 0 and 7
166   delayTime = 100;
167
168   digitalWrite(ledPins[index], HIGH);  // turn LED on
169   delay(delayTime);                  // pause to slow down
170   digitalWrite(ledPins[index], LOW);   // turn LED off
171 }
172
173
```

Example sketch 05 -- PUSH BUTTON

https://codebender.cc/sketch:77050
Fritzing diagram

```
/*************************************************************
 * SparkFun Inventor's Kit
 * Example sketch 05 -- PUSH BUTTONS
 *
 *   Use pushbuttons for digital input
 *
 *   Connect one side of the pushbutton to GND, and the other
 *   side to a digital pin. When we press down on the pushbutton,
 *   the pin will be connected to GND, and therefore will be read
 *   as "LOW" by the Arduino.
 *
 * This sketch was written by SparkFun Electronics,
 * with lots of help from the Arduino community.
 * This code is completely free for any use.
 * Visit http://learn.sparkfun.com/products/2 for SIK information.
 * Visit http://www.arduino.cc to learn about the Arduino.
 *
 * Version 2.0 6/2012 MDG
 * Version 2.1 9/2014 BCH
/*************************************************************/

const int button1Pin = 2;  // pushbutton 1 pin
const int button2Pin = 3;  // pushbutton 2 pin
const int ledPin =  13;      // LED pin

int button1State, button2State;  // variables to hold the pushbutton states


void setup()
{
  // Set up the pushbutton pins to be an input:
  pinMode(button1Pin, INPUT);
  pinMode(button2Pin, INPUT);

  // Set up the LED pin to be an output:
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  button1State = digitalRead(button1Pin);
  button2State = digitalRead(button2Pin);

  // if button1 or button 2 are pressed (but not both)
  if (((button1State == LOW) && (button2State == HIGH)) || ((button1State == HIGH) && (button2State
== LOW)))
  {
       digitalWrite(ledPin, HIGH);  // turn the LED on
  }
  else
  {
       digitalWrite(ledPin, LOW);  // turn the LED off
  }
}
```

Example sketch 05 -- PUSH BUTTON (v2) RGB LED State Machine

https://codebender.cc/sketch:128600
Fritzing diagram

```
/*************************************************************
 * SparkFun Inventor's Kit
 * Example sketch 05 -- PUSH BUTTONS (v2)
 *
 *   Use pushbuttons for digital input
 *
 *   Connect one side of the pushbutton to GND, and the other
 *   side to a digital pin. When we press down on the pushbutton,
 *   the pin will be connected to GND, and therefore will be read
 *   as "LOW" by the Arduino.
 *
 *   This is a variation (v2) to the code in the SIK. This example shows
 *   how to create and use what is called a state-machine to increment
 *   your Arduino through several program states.
 *
 * This sketch was written by SparkFun Electronics,
 * with lots of help from the Arduino community.
 * This code is completely free for any use.
 * Visit http://learn.sparkfun.com/products/2 for SIK information.
 * Visit http://www.arduino.cc to learn about the Arduino.
 *
 * Version 2.0 6/2012 MDG
 * Version 2.1 9/2014 BCH
/*************************************************************/

const int buttonPin = 2;   // pushbutton pin
const int redPin = 9;      // red LED pin
const int greenPin = 10;   // green LED pin
const int bluePin = 11;    // blue LED pin

int buttonState;    // variables to hold the pushbutton states
int progState = 0;  // progState is used as a state variable

void setup()
{
  // Set up the pushbutton pins to be an input:
  pinMode(buttonPin, INPUT);

  // Set up the LED pin to be an output:
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}

void loop()
{
  buttonState = digitalRead(buttonPin);

  // if button pressed, increment the state machine
  if (buttonState == LOW)
  {
    progState = progState + 1;
    if (progState > 3)
    {
      progState = 0;
```

```
    }
    // this is a while() loop. It continues to run whatever code
    // is between its { } while the statement ( ) is TRUE
    // This example, we do nothing, but wait. This ensures that we don't
    // continue to increment the state machine until the *next* button press.

    while (digitalRead(buttonPin) == LOW)
    {
      // do nothing
    }
    delay(100);
    // Short 0.1 second delay to "de-bounce" the circuit. See what happens if
    // you comment this line out. Test with various short or long presses of
    // the button. Watch the sequence of colors. Is it still Red-White-Blue?
  }

  //*************************************************************//
  // This part is the state machine. It checks the state variable
  // and then does a different thing based on what state it is in.
  //*************************************************************//
  if (progState == 0)
  {
    // red
    analogWrite(9, 255);
    analogWrite(10, 0);
    analogWrite(11, 0);
  }
  else if (progState == 1)
  {
    // white
    analogWrite(9, 255);
    analogWrite(10, 255);
    analogWrite(11, 255);
  }
  else if (progState == 2)
  {
    // blue
    analogWrite(9, 0);
    analogWrite(10, 0);
    analogWrite(11, 255);
  }
  else if (progState == 3) // state 3
  {
    // turn off the RGB LED
    analogWrite(9, 0);
    analogWrite(10, 0);
    analogWrite(11, 0);
  }
}
```

Example sketch 06 -- PHOTORESISTOR

https://codebender.cc/sketch:77051
Fritzing diagram

```
/*************************************************************
 * SparkFun Inventor's Kit
 * Example sketch 06 -- PHOTORESISTOR
 *
 * Use a photoresistor (light sensor) to control the brightness
 * of a LED.
 *
 * This sketch was written by SparkFun Electronics,
 * with lots of help from the Arduino community.
 * This code is completely free for any use.
 * Visit http://learn.sparkfun.com/products/2 for SIK information.
 * Visit http://www.arduino.cc to learn about the Arduino.
 *
 * Version 2.0 6/2012 MDG
 * Version 2.1 9/2014 BCH
*************************************************************/

// As usual, we'll create constants to name the pins we're using.
// This will make it easier to follow the code below.

const int sensorPin = 0;
const int ledPin = 9;

// We'll also set up some global variables for the light level:
int lightLevel;
int calibratedlightLevel; // used to store the scaled / calibrated lightLevel
int maxThreshold = 0;        // used for setting the "max" light level
int minThreshold = 1023;   // used for setting the "min" light level

void setup()
{
  pinMode(ledPin, OUTPUT);   // Setup the LED pin to be an output.
  Serial.begin(9600);
}

void loop()
{
  lightLevel = analogRead(sensorPin);  // reads the voltage on the sensorPin
  Serial.print(lightLevel);
  //autoRange();  // autoRanges the min / max values you see in your room.

  // scale the lightLevel from 0 - 1023 range to 0 - 255 range and assign to
  // variable calibratedlightLevel. The map() function applies a linear
  // scale / offset. map(inputValue, fromMin, fromMax, toMin, toMax);
  calibratedlightLevel = map(lightLevel, 0, 1023, 0, 255);

  Serial.print("\t");           // tab character

  // prints out the calibrated value --
  // println prints an CRLF at the end (creates a new line after)
  Serial.println(calibratedlightLevel);

  // set the led level based on the input lightLevel.
  analogWrite(ledPin, calibratedlightLevel);
}
/*************************************************************
 * void autoRange()
```

```
 *
 * This function sets a minThreshold and maxThreshold value for the
 * light levels in your setting. Move your hand / light source / etc
 * so that your light sensor sees a full range of values. This will
 * "autoCalibrate" to your range of input values.
/*****************************************************************/

void autoRange()
{
  // minThreshold was initialized to 1023 -- so, if it's less, reset the threshold level.
  if (lightLevel < minThreshold)
       minThreshold = lightLevel;

  // maxThreshold was initialized to 0 -- so, if it's bigger, reset the threshold level.
  if (lightLevel > maxThreshold)
       maxThreshold = lightLevel;

  // Once we have the highest and lowest values, we can stick them
  // directly into the map() function.
  //
  // This function must run a few times to get a good range of bright and dark
  // values in order to work.

  lightLevel = map(lightLevel, minThreshold, maxThreshold, 0, 255);
  lightLevel = constrain(lightLevel, 0, 255);
}
```

Example sketch 07 -- TEMPERATURE SENSOR

https://codebender.cc/sketch:77052
Fritzing diagram

```
/*************************************************************
 * SparkFun Inventor's Kit
 * Example sketch 07 - TEMPERATURE SENSOR
 *
 *  Use the serial monitor window to read a temperature sensor (TMP36).
 *
 * This sketch was written by SparkFun Electronics,
 * with lots of help from the Arduino community.
 * This code is completely free for any use.
 * Visit http://learn.sparkfun.com/products/2 for SIK information.
 * Visit http://www.arduino.cc to learn about the Arduino.
 *
 *
 * Version 2.0 6/2012 MDG
/*************************************************************/

// We'll use analog input A0 to measure the temperature sensor's signal pin.
const int temperaturePin = A0;

void setup()
{
  // initialize Serial communication at speed of 9600 baud.
  Serial.begin(9600);
}

void loop()
{
  // declare three variables as a float.  A float (floating-point) is a data
  // type that can have a fractional number such as 0.2211, 1.42, 253.261, etc...
  // Notice that we can declare multiple variables of the same type on one line:
  float voltage, degreesC, degreesF;

  // getVoltage() is a custom function we wrote.
  voltage = getVoltage(temperaturePin);

  // Now we'll convert the voltage to degrees Celsius.
  // According to the datasheet, the sensor voltage on the TMP36 changes at a
  // rate of 0.010 V per deg C, and the sensor has an output of 0.750 V at 25 deg C.
  // Using a little algebra, we come up with this convenient formula:

  degreesC = (voltage * 100) - 50;

  // While we're at it, let's convert degrees Celsius to Fahrenheit.
  // This is the classic C to F conversion formula:

  degreesF = degreesC * (9.0 / 5.0) + 32.0;

  Serial.print("voltage: ");
  Serial.print(voltage);
  Serial.print("  deg C: ");
  Serial.print(degreesC);
  Serial.print("  deg F: ");

  // this last println() inserts a carriage-return & newline character
  Serial.println(degreesF);

  delay(1000); // repeat once per second (change as you wish!)
```

```
58  }
59
60  float getVoltage(int pin)
61  {
62    // by multiplying the analogRead value by (AREF / 1023), we can convert
63    // it from the range of 0 - 1023 to volts.
64    const float AREF = 5.0;
65    return (analogRead(pin) * AREF / 1023);
66  }
67
68  // Other things to try with this code:
69  //
70  //   Turn on an LED if the temperature is above or below a value.
71  //
72  //   Read that threshold value from a potentiometer - now you've
73  //   created a thermostat!
```

Example sketch 08-1 -- SERVO SWEEP

https://codebender.cc/sketch:77055
Fritzing diagram

```
/*************************************************************
 * SparkFun Inventor's Kit
 * Example sketch 08-1 SERVO SWEEP
 *
 *  Sweep a servo back and forth through its full range of motion.
 *
 * This sketch was written by SparkFun Electronics,
 * with lots of help from the Arduino community.
 * This code is completely free for any use.
 * Visit http://learn.sparkfun.com/products/2 for SIK information.
 * Visit http://www.arduino.cc to learn about the Arduino.
 *
 * Version 2.0 6/2012 MDG
/*************************************************************/

// The servo motor is controlled by a PWM signal. As the pulse width changes,
// the servo will move to a different position. To control this pulse width
// signal for controlling the servo, we use a built-in library called Servo.h

#include <Servo.h>  // includes the servo library

// With the Servo.h library included, we can now access new methods and objects
// Here, we create a servo object called servo1. You can create as many (up to 12)
// servo objects as you need for your project.

Servo servo1;  // servo control object

void setup()
{
  // this initializes the servo object and attaches it to pin 9 for signal control
  // the second and third parameters define the minimum and maximum pulse width
  // for this servo as 900 microseconds and 2100 microseconds.

  servo1.attach(9, 900, 2100);
}


void loop()
{
  // to command the servo to a position, you use the .write() method
  // the parameter defines the angle from 0 to 180 of the servo arm.

  servo1.write(90);    // Tell servo to go to 90 degrees
  delay(1000);         // Pause to get it time to move

  servo1.write(180);   // Tell servo to go to 180 degrees
  delay(1000);         // Pause to get it time to move

  servo1.write(0);     // Tell servo to go to 0 degrees
  delay(1000);         // Pause to get it time to move

  // Change position at a slower speed:

  // When you command the servo to move, it does so immediately. To slow down the
  // speed of the servo, we'll move the servo in small increments with a short
  // delay in between each movement using a for() loop.
  //
```

```
58    // The for() loop is used to increment a variable and loop a specific number
59    // of times. In this case, it will move 90 steps (0 to 180 in 2 deg increments).
60
61    for(int position = 0; position < 180; position += 2)
62    {
63      servo1.write(position);  // Move to next position
64      delay(20);               // Short pause to allow it to move
65    }
66
67    // Here we tell servo to go to back to 0 degrees, by stepping by one degree
68    // increments (slower). This will move in 180 steps instead of just 90.
69
70    for(int position = 180; position >= 0; position -= 1)
71    {
72      servo1.write(position);  // Move to next position
73      delay(20);               // Short pause to allow it to move
74    }
75  }
```

Example sketch 08-2 -- SERIAL SERVO

https://codebender.cc/sketch:77053
Fritzing diagram

```
/*****************************************************************
 * SparkFun Inventor's Kit
 * Example sketch 08-2 -- SERIAL SERVO
 *
 *  Control the motion of a servo using the Serial Monitor
 *
 * This sketch was written by SparkFun Electronics,
 * with lots of help from the Arduino community.
 * This code is completely free for any use.
 * Visit http://learn.sparkfun.com/SIK for SIK information.
 * Visit http://www.arduino.cc to learn about the Arduino.
 *
 * Version 2.0 6/2012 MDG
 * Version 2.1 9/2014 BCH
*****************************************************************/
// The servo motor is controlled by a PWM signal. As the pulse width changes,
// the servo will move to a different position. To control this pulse width
// signal for controlling the servo, we use a built-in library called Servo.h

#include <Servo.h>  // includes the servo library

// With the Servo.h library included, we can now access new methods and objects
// Here, we create a servo object called servo1. You can create as many (up to 12)
// servo objects as you need for your project.

Servo servo1;  // servo control object

void setup()
{
  // this initializes the servo object and attaches it to pin 9 for signal control
  // the second and third parameters define the minimum and maximum pulse width
  // for this servo as 1000 microseconds and 2000 microseconds. You can adjust
  // these ranges to maximize the motion of your servo.

  servo1.attach(9, 1000, 2000);

  // initialize Serial communications at 9600 baud.
  Serial.begin(9600);

  // print out some information for the user.
  Serial.println("Type an angle (0-180) into the box above,");
  Serial.println("then click [send] or press [return]");
  Serial.println();  // Print a blank line
}

void loop()
{
  serialServo();
}

/*****************************************************************************
 * serialServo()
 *
 * This is a custom function that prints information to the Serial Monitor and
 * then waits for user input.
*****************************************************************************/
 void serialServo()
```

```
{
  int angle;
  // First we check to see if incoming data is available:
    while (Serial.available() > 0)
    {
    // If it is, we'll use parseInt() to pull out any numbers:
    angle = Serial.parseInt();

    // We'll print out a message to let you know that the number was received:
    Serial.print("Setting angle to ");
    Serial.println(angle);

    // set the angle on the servo motor.
    servo1.write(angle);
    }
}
```

Example sketch 09 -- FLEX SENSOR

https://codebender.cc/sketch:77057
Fritzing diagram

```
/*********************************************************************
 * SparkFun Inventor's Kit
 * Example sketch 09 - FLEX SENSOR
 *
 *  Use the "flex sensor" to change the position of a servo. In the previous
 *  sketch, we learned how to command a servo to mode to different positions.
 *  In this sketch, we'll introduce a new sensor, and use it to control the servo.
 *
 * Version 2.0 6/2012 MDG
/*********************************************************************/

// The servo motor is controlled by a PWM signal. As the pulse width changes,
// the servo will move to a different position. To control this pulse width
// signal for controlling the servo, we use a built-in library called Servo.h

#include <Servo.h>  // includes the servo library

Servo servo1;  // Decalare a servo control object called servo1

// Define the analog input pin to measure flex sensor position:
const int flexpin = A0;

int minFlex = 600;
int maxFlex = 900;

void setup()
{
  // Use the serial monitor window to help debug our sketch:
  Serial.begin(9600);

  // Enable & initialize control of a servo on pin 9. 900 and 2100 are calibration
  // parameters that specify the min and max pulse width in microSeconds for the servo
  servo1.attach(9, 900, 2100);
}


void loop()
{
  int flexposition;   // Input value from the analog pin.
  int servoposition;   // Output value to the servo.

  // Read the position of the flex sensor (0 to 1023):
  flexposition = analogRead(flexpin);

  // The map() function applies a linear translation to map a value from one range
  // to another range.
  servoposition = map(flexposition, minFlex, maxFlex, 0, 180);

  // The map() function may map to a value less than 0 or greater than 180.
  // The constrain() function ensures that the value is between 0 and 180.
  servoposition = constrain(servoposition, 0, 180);

  // Now we'll command the servo to move to that position:

  servo1.write(servoposition);

  // Because every flex sensor has a slightly different resistance,
```

```
58    // the 600 - 900 range may not exactly cover the flex sensor's
59    // output. To help tune our program, we'll use the serial port to
60    // print out our values to the serial monitor window:
61
62    Serial.print("sensor: ");
63    Serial.print(flexposition);
64    Serial.print("  servo: ");
65    Serial.println(servoposition);
66
67    delay(20);  // wait 20ms between servo updates
68  }
69
70  /*****************************************************************************
71   * After you upload the sketch, open the serial monitor. You'll be able to see
72   * the sensor values. Bend the flex sensor and note its minimum and maximum
73   * values.
74   *
75   * Change the minFlex and maxFlex values (600 and 900) above, you'll be able to
76   * exactly match the flex sensor's range with the servo's range.
77  *****************************************************************************/
```

Example sketch 10 -- SOFT POT

https://codebender.cc/sketch:77058
Fritzing diagram

```
/*****************************************************************************
 * SparkFun Inventor's Kit
 * Example sketch 10 SOFT POTENTIOMETER
 *
 * Use the soft potentiometer to change the color of the RGB LED
 *
 * This sketch was written by SparkFun Electronics, with lots of help from
 * the Arduino community. This code is completely free for any use.
 * Visit http://learn.sparkfun.com/SIK for SIK information.
 * Visit http://www.arduino.cc to learn about the Arduino.
 *
 * Version 2.0 6/2012 MDG
 *****************************************************************************/
const int RED_PIN = 9;       // Red LED Pin
const int GREEN_PIN = 10; // Green LED Pin
const int BLUE_PIN = 11;  // Blue LED Pin

const int SENSOR_PIN = A0;   // Analog input pin

// Global variables for PWM brightness values for the RGB LED.
// These are global so both loop() and setRainbow() can see them.
int redValue, greenValue, blueValue;

void setup()
{
  pinMode(RED_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(BLUE_PIN, OUTPUT);
}

void loop()
{
  int sensorValue;

  // Read the voltage from the softpot (0-1023)
  sensorValue = analogRead(SENSOR_PIN);

  // We've written a new function called setRainbow() (further down in the
  // sketch) that decodes sensorValue into a position on the RGB "rainbow", and
  // sets the RGB LED to that color.
  setRainbow(sensorValue);
}

/*****************************************************************************
 * rainbow()
 * Set a RGB LED to a position on the "rainbow" of all colors. Colors transitions
 * from green to red to blue back to green. RGBposition should be in the range
 * of 0 to 1023 (such as from an analog input).
 *****************************************************************************/
void setRainbow(int RGBposition)
{
  int brightness;
  if(RGBposition <= 341)
  {
      // green --> red
       brightness = map(RGBposition, 0, 341, 0, 255);
       analogWrite(RED_PIN, brightness);
```

```
58          analogWrite(GREEN_PIN, 255 - brightness);
59          analogWrite(BLUE_PIN, 0);
60    }
61    else if (RGBposition <= 682)
62    {
63        // red --> blue
64          brightness = map(RGBposition, 341, 682, 0, 255);
65          analogWrite(RED_PIN, 255 - brightness);
66          analogWrite(GREEN_PIN, 0);
67          analogWrite(BLUE_PIN, brightness);
68    }
69    else
70    {
71        // blue --> green
72          brightness = map(RGBposition, 682, 1023, 0, 255);
73          analogWrite(RED_PIN, 0);
74          analogWrite(GREEN_PIN, brightness);
75          analogWrite(BLUE_PIN, 255 - brightness);
76    }
77  }
```

Example sketch 11 -- BUZZER

https://codebender.cc/sketch:77059
Fritzing diagram

```
/*****************************************************************************
 * SparkFun Inventor's Kit
 * Example sketch 11 BUZZER
 *
 * This sketch uses the buzzer to play songs. The Arduino's tone() command will
 * play notes of a given frequency.
 *
 * This sketch was written by SparkFun Electronics, with lots of help from
 * the Arduino community. This code is completely free for any use.
 * Visit http://learn.sparkfun.com/SIK for SIK information.
 * Visit http://www.arduino.cc to learn about the Arduino.
 *
 * Version 2.0 6/2012 MDG
 * Version 2.1 9/2014 BCH
*****************************************************************************/

const int buzzerPin = 9;     // connect the buzzer to pin 9 and GND
const int songLength = 18;  // sets the number of notes of the song.

// notes[] is an array of text characters corresponding to the notes in your
// song. A space represents a rest (no tone). This array is 18 notes long.
//
// An array is a list of items that consists of the same data type. Arrays are
// a convenient way to organize data that is all of the same type.
char notes[songLength] = {
  'c', 'd', 'f', 'd', 'a', ' ', 'a', 'g', ' ', 'c', 'd', 'f', 'd', 'g', ' ', 'g', 'f', ' '};

// beats[] is another array. This stores values that represent the length of each
// note. A "1" represents a quarter-note, 2 a half-note, etc. Don't forget
// that the rests (spaces) need a length as well.

int beats[songLength] = {
  1, 1, 1, 1, 1, 1, 4, 4, 2, 1, 1, 1, 1, 1, 1, 4, 4, 2};

// The tempo is how fast to play the song. This is the number of milliseconds for
// a quarter-note.
int tempo = 150;

void setup()
{
  pinMode(buzzerPin, OUTPUT);  // sets the pin as an OUTPUT
}

void loop()
{
  int duration;

  // for loop is used to index through the arrays
  for (int index = 0; index < songLength; index++)
  {
        duration = beats[index] * tempo;  // length of note/rest in ms

        if (notes[index] == ' ')     // is this a rest?
        delay(duration);             // then pause for a moment
        else                         // otherwise, play the note
        {
        tone(buzzerPin, frequency(notes[index]), duration);  // play the note
```

```
58          delay(duration);      // wait for tone to finish
59          }
60          delay(tempo/10);      // brief pause between notes
61    }
62
63    // We only want to play the song once, so we'll pause forever.
64    while(true)
65    {}
66    // If you'd like your song to play over and over, remove the while(true)
67    // statement and the curly braces above.
68  }
69
70  /*****************************************************************************
71   * frequency() is a custom function that takes a note as a character (a-g),
72   * and returns the corresponding frequency in Hz for the tone() function.
73  *****************************************************************************/
74  int frequency(char note)
75  {
76    int i;
77    const int numNotes = 8;  // number of notes we're storing
78
79    // This is the look-up table for notes and frequencies:
80    char names[numNotes] = {
81          'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C'};
82    int frequencies[numNotes] = {
83          262, 294, 330, 349, 392, 440, 494, 523};
84
85    // Now we'll search through the letters in the array, and if we find it, we'll
86    // return the frequency for that note.
87    for (i = 0; i < numNotes; i++)  // Step through the notes
88    {
89          if (note == names[i])         // Is this the one?
90          {
91          return(frequencies[i]);       // Yes! Return the frequency and exit function.
92          }
93    }
94    return(0);  // Returning a 0 indicates didn't find a note.
95  }
```

Example sketch 12 -- MOTOR SPIN

https://codebender.cc/sketch:77060
Fritzing diagram

```
/*************************************************************************
 * SparkFun Inventor's Kit
 * Example sketch 12 SPINNING A MOTOR
 *
 * This example requires that you drive your motor using a switching
 * transistor. The Arduino is only capable of sourcing about 40 mA of
 * current per pin and a motor requires upwards of 150 mA.
 *
 * Look at the wiring diagram in the SIK Guide - Circuit #12 or read the
 * notes in the readme tab for more information on wiring.
 *
 * This sketch was written by SparkFun Electronics, with lots of help from
 * the Arduino community. This code is completely free for any use.
 * Visit http://learn.sparkfun.com/SIK for SIK information.
 * Visit http://www.arduino.cc to learn about the Arduino.
 *
 * Version 2.0 6/2012 MDG
 * Version 2.1 8/2014 BCH
*************************************************************************/

// connect the base pin of the transistor (center pin) to pin 9 even though it's
// not directly connected to the motor we'll still call it the 'motorPin'.
const int motorPin = 9;

void setup()
{
  pinMode(motorPin, OUTPUT);  // setup the pin as an OUTPUT
  Serial.begin(9600);          // initialize Serial communications.
}


void loop()
{ // this example basically replicates a blink, but with the motorPin instead.
  int onTime = 3000;  // milliseconds to turn the motor on
  int offTime = 3000; // milliseconds to turn the motor off

  analogWrite(motorPin, 255); // turn the motor on (full speed)
  delay(onTime);              // delay for onTime milliseconds
  analogWrite(motorPin, 0);  // turn the motor off
  delay(offTime);             // delay for offTime milliseconds

  // Uncomment the functions below by taking out the //. Try each of these out
  // one at a time. Look below for the code.

  // speedUpandDown();
  // serialSpeed();
}

/*************************************************************************
 * speedUpandDown()
 * This function accelerates the motor to full speed, then decelerates back
 * down to a stop slowly.
*************************************************************************/
void speedUpandDown()
{
  int delayTime = 20; // milliseconds between each speed step

```

```
  // accelerate the motor to full speed in approximately 5 seconds
  for(int speed = 0; speed <= 255; speed++)
  {
        analogWrite(motorPin,speed);    // set the new speed
        delay(delayTime);                    // delay between speed steps
  }
  // decelerate the motor slowly -- notice that the delay is (2*delayTime)
  for(int speed = 255; speed >= 0; speed--)
  {
        analogWrite(motorPin,speed);    // set the new speed
        delay(2*delayTime);                  // delay between speed steps
  }
}

/*****************************************************************************
 * serialSpeed()
 * Open the Serial Monitor and enter a speed value (0 to 255) to set the motor.
/*****************************************************************************/
void serialSpeed()
{
  int speed;

  Serial.println("Type a speed (0-255) into the box above,");
  Serial.println("then click [send] or press [return]");
  Serial.println();  // Print a blank line

  // In order to type out the above message only once,
  // we'll run the rest of this function in an infinite loop:

  while(true)  // "true" is always true, so this will loop forever.
  {
        // Check to see if incoming data is available:
        while (Serial.available() > 0)
        {
        // parseInt() reads in the first integer value from the Serial Monitor.
        speed = Serial.parseInt();

        // constrains the speed between 0 and 255 because analogWrite()
        speed = constrain(speed, 0, 255);

        analogWrite(motorPin, speed);  // sets the speed of the motor.

        // feedback and prints out the speed that you entered.
        Serial.print("Setting speed to ");
        Serial.println(speed);
        }
  }
}
```

Example sketch 13 -- RELAYS

https://codebender.cc/sketch:77061
Fritzing diagram

```
/*****************************************************************
 * SparkFun Inventor's Kit
 * Example sketch 13 RELAYS
 *
 * A relay is a electrically-controlled mechanical switch. It has an
 * electro-magnetic coil that either opens or closes a switch. Because
 * it is a physical switch, a relay can turn on and off large devices
 * like BIG motors, spot lights, and lamps.
 *
 * To create enough current to excite the electro-magnet, we need to use
 * the transistor circuit from the last example. Each time we excite the relay
 * you should hear an audible clicking sound of the switch.
 *
 * This sketch was written by SparkFun Electronics,
 * with lots of help from the Arduino community.
 * Visit http://sparkfun.com/sikguide for SIK information.
 * Visit http://www.arduino.cc to learn about the Arduino.
 *
 * Version 2.0 6/2012 MDG
 * Version 2.1 8/2014 BCH
 *****************************************************************/

const int relayPin = 2;      // use this pin to drive the transistor (which drives the relay)
const int timeDelay = 1000; // delay in ms for on and off phases

// You can make timeDelay shorter, but note that relays, being mechanical devices,
// will wear out quickly if you try to drive them too fast or too often.

void setup()
{
  pinMode(relayPin, OUTPUT);  // set pin as an output
}


void loop()
{
  digitalWrite(relayPin, HIGH);  // turn the relay on
  delay(timeDelay);           // wait for one second
  digitalWrite(relayPin, LOW);   // turn the relay off
  delay(timeDelay);           // wait for one second
}
```

Example sketch 14 -- SHIFT REGISTER

https://codebender.cc/sketch:77060
Fritzing diagram

```
/*************************************************************************
 * SparkFun Inventor's Kit
 * Example sketch 14 SHIFT REGISTER
 *
 * Use a shift register to turn three pins into eight (or more!) outputs. Shift
 * registers can be cascaded together so that you can drive any number of LEDs
 * with just the three pins on the Arduino.
 *
 * To cascade shift registers, connect the data out (QH*) to the Data In of the
 * next Shift Register. Connect together the latch and clock pins for all shift
 * registers.
 *
 * This sketch was written by SparkFun Electronics, with lots of help from
 * the Arduino community. This code is completely free for any use.
 * Visit http://sparkfun.com/sikguide for SIK information.
 * Visit http://www.arduino.cc to learn about the Arduino.
 *
 * Version 2.0 6/2012 MDG
/*************************************************************************/

// The 74HC595 shift register uses a type of serial connection called SPI
// (Serial Peripheral Interface) that requires three pins:

int datapin = 2;
int clockpin = 3;
int latchpin = 4;

// Declare a global variable for the data we're sending to the shift register.

byte data = 0;

void setup()
{
  // Set the three SPI pins to be outputs:
  pinMode(datapin, OUTPUT);
  pinMode(clockpin, OUTPUT);
  pinMode(latchpin, OUTPUT);
}

void loop()
{

  binaryCount();

  // These are each functions that we demonstrated in Example #4 with the multiple
  // LEDs. These are now each done using the shift register instead.
  //
  //oneAfterAnother();       // All on, all off
  //oneOnAtATime();          // Scroll down the line
  //pingPong();              // Like above, but back and forth
  //randomLED();             // Blink random LEDs
  //marquee();

}

/*************************************************************
 * binaryCount()
```

```
 *
 * Numbers are stored internally in the Arduino as arrays of "bits",
 * each of which is a 1 or 0. Just like the base-10 numbers we use
 * every day, The position of the bit affects the magnitude of its
 * contribution to the total number:
 *
 * Bit position   Contribution
 *     0              1
 *     1              2
 *     2              4
 *     3              8
 *     4              16
 *     5              32
 *     6              64
 *     7              128
 *
 * To build any number from 0 to 255 from the above 8 bits, just
 * select the contributions you need to make. The bits will then be
 * 1 if you use that contribution, and 0 if you don't.
/*************************************************************/
void binaryCount()
{
  int delayTime = 1000; // time (milliseconds) to pause between LEDs
                        // make this smaller for faster switching

  // Send the data byte to the shift register:

  shiftOut(datapin, clockpin, MSBFIRST, data);

  // Toggle the latch pin to make the data appear at the outputs:

  digitalWrite(latchpin, HIGH);
  digitalWrite(latchpin, LOW);

  // Add one to data, and repeat!
  data++;

  // Delay so you can see what's going on:
  delay(delayTime);
}

/*************************************************************************
 * shiftWrite()
 * This function sets the desired bit (0 - 7) to a desired state (HIGH/LOW) through
 * the shift register.
/*************************************************************************/

void shiftWrite(int desiredBit, boolean desiredState)
{
  // First we'll alter the global variable "data", changing the
  // desired bit to 1 or 0:
  bitWrite(data, desiredBit,desiredState);

  // shiftOut() moves a byte (8-bits) of data out to the shift register.
  // MSBFIRST indicates that it moves the Most Significant Bit out first.
  shiftOut(datapin, clockpin, MSBFIRST, data);

  // Once the data has been shifted in, it must be "latched" in order for it to
  // show on the OUTPUTS. We do this by making a HIGH-to-LOW transition on the latch pin.
  digitalWrite(latchpin, HIGH);
  digitalWrite(latchpin, LOW);
}
```

```
120
121 /*****************************************************************
122  * oneAfterAnother()
123  *
124  * This function does exactly the same thing as oneAfterAnotherNoLoop(),
125  * but it takes advantage of for() loops and the array to do it with
126  * much less typing.
127 *****************************************************************/
128 void oneAfterAnother()
129 {
130   int index;
131   int delayTime = 100; // Time (milliseconds) to pause between LEDs
132                        // Make this smaller for faster switching
133
134   // Turn all the LEDs on:
135
136   // This for() loop will step index from 0 to 7
137   // (putting "++" after a variable means add one to it)
138   // and will then use digitalWrite() to turn that LED on.
139
140   for(index = 0; index <= 7; index++)
141   {
142         shiftWrite(index, HIGH);
143         delay(delayTime);
144   }
145
146   // Turn all the LEDs off:
147
148   // This for() loop will step index from 7 to 0
149   // (putting "--" after a variable means subtract one from it)
150   // and will then use digitalWrite() to turn that LED off.
151
152   for(index = 7; index >= 0; index--)
153   {
154         shiftWrite(index, LOW);
155         delay(delayTime);
156   }
157 }
158
159
160 /*****************************************************************
161  * oneOnAtATime()
162  *
163  * This function will step through the LEDs, lighting only one at
164  * a time. It turns each LED ON and then OFF before going to the
165  * next LED.
166 *****************************************************************/
167 void oneOnAtATime()
168 {
169   int index;
170   int delayTime = 100; // Time (milliseconds) to pause between LEDs
171                        // Make this smaller for faster switching
172
173   // step through the index from 0 to 7 and turn the LEDs on then off in order.
174   for(index = 0; index <= 7; index++)
175   {
176         shiftWrite(index, HIGH);   // turn LED on
177         delay(delayTime);            // pause to slow down the sequence
178         shiftWrite(index, LOW);      // turn LED off
179   }
180 }
181
```

```
182
183 /*****************************************************************
184  * pingPong()
185  *
186  * This function will step through the LEDs, lighting one at at
187  * time in both directions. There is no delay between the LED off
188  * and turning on the next LED. This creates a smooth pattern for
189  * the LED pattern.
190 *****************************************************************/
191 void pingPong()
192 {
193   int delayTime = 100; // time (milliseconds) to pause between LEDs
194                        // make this smaller for faster switching
195
196   // step through the index from 0 to 7 to turn the LEDS ON then OFF
197   for(int index = 0; index <= 7; index++)
198   {
199         shiftWrite(index, HIGH);  // turn LED on
200         delay(delayTime);         // pause to slow down the sequence
201         shiftWrite(index, LOW);       // turn LED off
202   }
203
204   // step through the index from 7 to 0 to turn the LEDS ON then OFF
205   for(int index = 7; index >= 0; index--)
206   {
207         shiftWrite(index, HIGH);  // turn LED on
208         delay(delayTime);         // pause to slow down the sequence
209         shiftWrite(index, LOW);       // turn LED off
210   }
211 }
212
213 /*****************************************************************
214  * marquee()
215  *
216  * This function will mimic "chase lights" like those around signs.
217 *****************************************************************/
218
219 void marquee()
220 {
221   int delayTime = 200; // Time (milliseconds) to pause between LEDs
222                        // Make this smaller for faster switching
223
224   // Step through an index of 0 to 3. We'll use this index to light up the
225   // lower 4 LEDs and the upper 4 LEDs together.
226   for(int index = 0; index <= 3; index++)
227   {
228         shiftWrite(index, HIGH);      // Turn a LED on
229         shiftWrite(index + 4, HIGH);  // Skip four, and turn on LEDs 4 - 7
230
231         delay(delayTime);         // Pause to slow down the sequence
232         shiftWrite(index, LOW);       // Turn LED off
233         shiftWrite(index + 4, LOW); // Turn 2nd LED off
234   }
235 }
236
237 /*****************************************************************
238  * randomLED()
239  *
240  * This function will turn on random LEDs. Can you modify it so it
241  * also lights them for random times?
242 *****************************************************************/
243
```

```
244 void randomLED()
245 {
246   int index;
247   // delay time (milliseconds) to pause between LEDs.  Make this smaller for faster switching
248   int delayTime = 100;
249 
250   // pick a random number between 0 and 7 (upperBound is exclusive)
251   index = random(0, 8);
252 
253   shiftWrite(index, HIGH);    // turn LED on
254   delay(delayTime);           // pause to slow down the sequence
255   shiftWrite(index, LOW);    // turn LED off
256 }
```

Example sketch 15 -- Liquid Crystal Display (LCD)

https://codebender.cc/sketch:77063
Fritzing diagram

```
/*********************************************************************************
 * SparkFun Inventor's Kit
 * Example sketch 15 LIQUID CRYSTAL DISPLAY (LCD)
 *
 * This sketch will show you how to connect an LCD to your Arduino
 * and display any data you wish.
 *
 * Once everything is connected, load this sketch into the Arduino, and adjust
 * the potentiometer until the display is clear.
 * This sketch was written by SparkFun Electronics, with lots of help from
 * the Arduino community. This code is completely free for any use.
 * Visit http://sparkfun.com/sikguide for SIK information.
 * Visit http://www.arduino.cc to learn about the Arduino.
 *
 * Version 1.0 2/2013 MDG
 * Version 1.1 7/2016 BCH
/*********************************************************************************/

// Load the LiquidCrystal library. This gives us commands to interface to the LCD
#include <LiquidCrystal.h>

// LiquidCrystal lcd(RS, Enable, d4, d5, d6, d7); -- 6 control / signal pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
  // This initializes the lcd as a 16 x 2 display. The same library works with
  // other sizes as well.
  lcd.begin(16, 2);

  // This clears the display.
  lcd.clear();

  // When the display powers up, the cursor (invisible) starts on the top row
  // and first column. That's were we're going to start writing out text.

  lcd.print("hello, world!");  // prints "hello, world!" to the LCD.
}

void loop()
{
  // You can move the invisible cursor to any location on the
  // LCD before sending data. Counting starts at 0. The top
  // line is line 0 and the bottom line is line 1.
  // Columns range from 0 on the left side, to 15 on the right.

  // Move the cursor to the first column (column 0) of the second line (line 1):
  lcd.setCursor(0, 1);

  // Now we'll print the number of seconds (millis() / 1000)
  // since the Arduino last reset:

  lcd.setCursor(0,1);   // Reset the cursor to the original position
  lcd.print(millis()/1000); // Print our value
}

/*********************************************************************************
```

```
 * TIP: Since the numeric data we're sending is always growing in length, new
 * values will always overwrite the previous ones. However, if you want to
 * display varying or decreasing numbers like a countdown, you'll find that
 * the display will leave "orphan" characters when the new value is shorter
 * than the old one.
 *
 * To prevent this, you'll need to erase the old number before writing the
 * new one. You can do this by overwriting the last number with spaces. If you
 * erase the old number and immediately write the new one, the momentary erase
 * won't be noticeable. Here's a typical sequence of code:

  lcd.setCursor(0,1);   // Set the cursor to the position
  lcd.print("         "); // Erase the largest possible number
  lcd.setCursor(0,1);   // Reset the cursor to the original position
  lcd.print(millis()/1000); // Print our value

/*********************************************************************
 * NEXT STEPS:
 *
 * Now you know the basics of hooking up an LCD to the Arduino, and sending text
 * and numeric data to the display!
 *
 * The LCD library has many commands for turning the cursor on and off,
 * scrolling the screen, etc. See: http://arduino.cc/en/Reference/LiquidCrystal
 * for more information.
 *
 * Arduino also comes with a number of built-in examples showing off the features
 * of the LiquidCrystal library. These are located under
 * File>Examples>LiquidCrystal menu.
 *
 * Have fun, and let us know what you create!
 * Your friends at SparkFun.
*********************************************************************/
```

Example sketch 16 -- Simon Game*

https://codebender.cc/sketch:77064
Fritzing diagram

```
/*****************************************************************************
 * SparkFun Inventor's Kit
 * Example sketch 16 Simon Game
 *
 * Simon Says is a memory game. Start the game by pressing one of the four
 * buttons. When a button lights up, press the button, repeating the sequence.
 * The sequence will get longer and longer. The game is won after 13 rounds.
 *
 * This sketch was written by SparkFun Electronics, with lots of help from
 * the Arduino community. This code is completely free for any use.
 * Visit http://sparkfun.com/sikguide for SIK information.
 * Visit http://www.arduino.cc to learn about the Arduino.
 *
 * SparkFun Electronics
 * Oct. 7, 2014
*****************************************************************************/

#include "SIK_circuit16_simonGame.h" // public constants used in the code

// Game state variables
byte gameMode = MODE_MEMORY; //By default, let's play the memory game
byte gameBoard[32]; //Contains the combination of buttons as we advance
byte gameRound = 0; //Counts the number of succesful rounds the player has made it through

void setup()
{
  // Setup hardware inputs/outputs. These pins are defined in the hardware_versions header file

  // Enable Internal pull up resistors on button inputs.
  pinMode(BUTTON_RED, INPUT_PULLUP);
  pinMode(BUTTON_GREEN, INPUT_PULLUP);
  pinMode(BUTTON_BLUE, INPUT_PULLUP);
  pinMode(BUTTON_YELLOW, INPUT_PULLUP);

  // Setup LED pins as OUTPUTs
  pinMode(LED_RED, OUTPUT);
  pinMode(LED_GREEN, OUTPUT);
  pinMode(LED_BLUE, OUTPUT);
  pinMode(LED_YELLOW, OUTPUT);

  // Buzzer is connected to two pins - this enables the sound to be louder.
  pinMode(BUZZER1, OUTPUT);
  pinMode(BUZZER2, OUTPUT);

  // Mode checking
  gameMode = MODE_MEMORY; // By default, we're going to play the memory game

  // Check to see if the lower right button is pressed
  if (checkButton() == CHOICE_YELLOW) play_beegees();

  // Check to see if upper right button is pressed
  if (checkButton() == CHOICE_GREEN)
  {
        gameMode = MODE_BATTLE; // Put game into battle mode

        // Turn on the upper right (green) LED
        setLEDs(CHOICE_GREEN);
```

```
        toner(CHOICE_GREEN, 150);

        setLEDs(CHOICE_RED | CHOICE_BLUE | CHOICE_YELLOW); // Turn on the other LEDs until you release button

        while(checkButton() != CHOICE_NONE) ; // Wait for user to stop pressing button

        // Now do nothing. Battle mode will be serviced in the main routine
  }

  play_winner(); // After setup is complete, say hello to the world
}

void loop()
{
  attractMode(); // Blink lights while waiting for user to press a button

  // Indicate the start of game play
  setLEDs(CHOICE_RED | CHOICE_GREEN | CHOICE_BLUE | CHOICE_YELLOW); // Turn all LEDs on
  delay(1000);
  setLEDs(CHOICE_OFF); // Turn off LEDs
  delay(250);

  if (gameMode == MODE_MEMORY)
  {
        // Play memory game and handle result
        if (play_memory() == true)
        play_winner(); // Player won, play winner tones
        else
        play_loser(); // Player lost, play loser tones
  }

  if (gameMode == MODE_BATTLE)
  {
        play_battle(); // Play game until someone loses

        play_loser(); // Player lost, play loser tones
  }
}

//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
//The following functions are related to game play only

// Play the regular memory game
// Returns 0 if player loses, or 1 if player wins
boolean play_memory(void)
{
  randomSeed(millis()); // Seed the random generator with random amount of millis()

  gameRound = 0; // Reset the game to the beginning

  while (gameRound < ROUNDS_TO_WIN)
  {
        add_to_moves(); // Add a button to the current moves, then play them back

        playMoves(); // Play back the current game board

        // Then require the player to repeat the sequence.
        for (byte currentMove = 0 ; currentMove < gameRound ; currentMove++)
        {
        byte choice = wait_for_button(); // See what button the user presses

```

```
119         if (choice == 0) return false; // If wait timed out, player loses
120
121         if (choice != gameBoard[currentMove]) return false; // If the choice is incorect, player
***  loses
122         }
123
124         delay(1000); // Player was correct, delay before playing moves
125     }
126
127     return true; // Player made it through all the rounds to win!
128 }
129
130 // Play the special 2 player battle mode
131 // A player begins by pressing a button then handing it to the other player
132 // That player repeats the button and adds one, then passes back.
133 // This function returns when someone loses
134 boolean play_battle(void)
135 {
136     gameRound = 0; // Reset the game frame back to one frame
137
138     while (1) // Loop until someone fails
139     {
140         byte newButton = wait_for_button(); // Wait for user to input next move
141         gameBoard[gameRound++] = newButton; // Add this new button to the game array
142
143         // Then require the player to repeat the sequence.
144         for (byte currentMove = 0 ; currentMove < gameRound ; currentMove++)
145         {
146         byte choice = wait_for_button();
147
148         if (choice == 0) return false; // If wait timed out, player loses.
149
150         if (choice != gameBoard[currentMove]) return false; // If the choice is incorect, player
***  loses.
151         }
152
153         delay(100); // Give the user an extra 100ms to hand the game to the other player
154     }
155
156     return true; // We should never get here
157 }
158
159 // Plays the current contents of the game moves
160 void playMoves(void)
161 {
162     for (byte currentMove = 0 ; currentMove < gameRound ; currentMove++)
163     {
164         toner(gameBoard[currentMove], 150);
165
166         // Wait some amount of time between button playback
167         // Shorten this to make game harder
168         delay(150); // 150 works well. 75 gets fast.
169     }
170 }
171
172 // Adds a new random button to the game sequence, by sampling the timer
173 void add_to_moves(void)
174 {
175     byte newButton = random(0, 4); //min (included), max (exluded)
176
177     // We have to convert this number, 0 to 3, to CHOICEs
178     if(newButton == 0) newButton = CHOICE_RED;
```

```
  else if(newButton == 1) newButton = CHOICE_GREEN;
  else if(newButton == 2) newButton = CHOICE_BLUE;
  else if(newButton == 3) newButton = CHOICE_YELLOW;

  gameBoard[gameRound++] = newButton; // Add this new button to the game array
}

//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
//The following functions control the hardware

// Lights a given LEDs
// Pass in a byte that is made up from CHOICE_RED, CHOICE_YELLOW, etc
void setLEDs(byte leds)
{
  if ((leds & CHOICE_RED) != 0)
        digitalWrite(LED_RED, HIGH);
  else
        digitalWrite(LED_RED, LOW);

  if ((leds & CHOICE_GREEN) != 0)
        digitalWrite(LED_GREEN, HIGH);
  else
        digitalWrite(LED_GREEN, LOW);

  if ((leds & CHOICE_BLUE) != 0)
        digitalWrite(LED_BLUE, HIGH);
  else
        digitalWrite(LED_BLUE, LOW);

  if ((leds & CHOICE_YELLOW) != 0)
        digitalWrite(LED_YELLOW, HIGH);
  else
        digitalWrite(LED_YELLOW, LOW);
}

// Wait for a button to be pressed.
// Returns one of LED colors (LED_RED, etc.) if successful, 0 if timed out
byte wait_for_button(void)
{
  long startTime = millis(); // Remember the time we started the this loop

  while ( (millis() - startTime) < ENTRY_TIME_LIMIT) // Loop until too much time has passed
  {
        byte button = checkButton();

        if (button != CHOICE_NONE)
        {
        toner(button, 150); // Play the button the user just pressed

        while(checkButton() != CHOICE_NONE) ;  // Now let's wait for user to release button

        delay(10); // This helps with debouncing and accidental double taps

        return button;
        }

  }

  return CHOICE_NONE; // If we get here, we've timed out!
}

// Returns a '1' bit in the position corresponding to CHOICE_RED, CHOICE_GREEN, etc.
```

```
241 byte checkButton(void)
242 {
243   if (digitalRead(BUTTON_RED) == 0) return(CHOICE_RED);
244   else if (digitalRead(BUTTON_GREEN) == 0) return(CHOICE_GREEN);
245   else if (digitalRead(BUTTON_BLUE) == 0) return(CHOICE_BLUE);
246   else if (digitalRead(BUTTON_YELLOW) == 0) return(CHOICE_YELLOW);
247 
248   return(CHOICE_NONE); // If no button is pressed, return none
249 }
250 
251 // Light an LED and play tone
252 // Red, upper left:   440Hz - 2.272ms - 1.136ms pulse
253 // Green, upper right:  880Hz - 1.136ms - 0.568ms pulse
254 // Blue, lower left:  587.33Hz - 1.702ms - 0.851ms pulse
255 // Yellow, lower right: 784Hz - 1.276ms - 0.638ms pulse
256 void toner(byte which, int buzz_length_ms)
257 {
258   setLEDs(which); //Turn on a given LED
259 
260   //Play the sound associated with the given LED
261   switch(which)
262   {
263   case CHOICE_RED:
264         buzz_sound(buzz_length_ms, 1136);
265         break;
266   case CHOICE_GREEN:
267         buzz_sound(buzz_length_ms, 568);
268         break;
269   case CHOICE_BLUE:
270         buzz_sound(buzz_length_ms, 851);
271         break;
272   case CHOICE_YELLOW:
273         buzz_sound(buzz_length_ms, 638);
274         break;
275   }
276 
277   setLEDs(CHOICE_OFF); // Turn off all LEDs
278 }
279 
280 // Toggle buzzer every buzz_delay_us, for a duration of buzz_length_ms.
281 void buzz_sound(int buzz_length_ms, int buzz_delay_us)
282 {
283   // Convert total play time from milliseconds to microseconds
284   long buzz_length_us = buzz_length_ms * (long)1000;
285 
286   // Loop until the remaining play time is less than a single buzz_delay_us
287   while (buzz_length_us > (buzz_delay_us * 2))
288   {
289         buzz_length_us -= buzz_delay_us * 2; //Decrease the remaining play time
290 
291         // Toggle the buzzer at various speeds
292         digitalWrite(BUZZER1, LOW);
293         digitalWrite(BUZZER2, HIGH);
294         delayMicroseconds(buzz_delay_us);
295 
296         digitalWrite(BUZZER1, HIGH);
297         digitalWrite(BUZZER2, LOW);
298         delayMicroseconds(buzz_delay_us);
299   }
300 }
301 
302 // Play the winner sound and lights
```

```
303 void play_winner(void)
304 {
305   setLEDs(CHOICE_GREEN | CHOICE_BLUE);
306   winner_sound();
307   setLEDs(CHOICE_RED | CHOICE_YELLOW);
308   winner_sound();
309   setLEDs(CHOICE_GREEN | CHOICE_BLUE);
310   winner_sound();
311   setLEDs(CHOICE_RED | CHOICE_YELLOW);
312   winner_sound();
313 }
314
315 // Play the winner sound
316 // This is just a unique (annoying) sound we came up with, there is no magic to it
317 void winner_sound(void)
318 {
319   // Toggle the buzzer at various speeds
320   for (byte x = 250 ; x > 70 ; x--)
321   {
322         for (byte y = 0 ; y < 3 ; y++)
323         {
324         digitalWrite(BUZZER2, HIGH);
325         digitalWrite(BUZZER1, LOW);
326         delayMicroseconds(x);
327
328         digitalWrite(BUZZER2, LOW);
329         digitalWrite(BUZZER1, HIGH);
330         delayMicroseconds(x);
331         }
332   }
333 }
334
335 // Play the loser sound/lights
336 void play_loser(void)
337 {
338   setLEDs(CHOICE_RED | CHOICE_GREEN);
339   buzz_sound(255, 1500);
340
341   setLEDs(CHOICE_BLUE | CHOICE_YELLOW);
342   buzz_sound(255, 1500);
343
344   setLEDs(CHOICE_RED | CHOICE_GREEN);
345   buzz_sound(255, 1500);
346
347   setLEDs(CHOICE_BLUE | CHOICE_YELLOW);
348   buzz_sound(255, 1500);
349 }
350
351 // Show an "attract mode" display while waiting for user to press button.
352 void attractMode(void)
353 {
354   while(1)
355   {
356         setLEDs(CHOICE_RED);
357         delay(100);
358         if (checkButton() != CHOICE_NONE) return;
359
360         setLEDs(CHOICE_BLUE);
361         delay(100);
362         if (checkButton() != CHOICE_NONE) return;
363
364         setLEDs(CHOICE_GREEN);
```

```
        delay(100);
        if (checkButton() != CHOICE_NONE) return;

        setLEDs(CHOICE_YELLOW);
        delay(100);
        if (checkButton() != CHOICE_NONE) return;
    }
}

//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
// The following functions are related to Beegees Easter Egg only

// Notes in the melody. Each note is about an 1/8th note, "0"s are rests.
int melody[] = {
  NOTE_G4, NOTE_A4, 0, NOTE_C5, 0, 0, NOTE_G4, 0, 0, 0,
  NOTE_E4, 0, NOTE_D4, NOTE_E4, NOTE_G4, 0,
  NOTE_D4, NOTE_E4, 0, NOTE_G4, 0, 0,
  NOTE_D4, 0, NOTE_E4, 0, NOTE_G4, 0, NOTE_A4, 0, NOTE_C5, 0};

int noteDuration = 115; // This essentially sets the tempo, 115 is just about right for a disco groove :)
int LEDnumber = 0; // Keeps track of which LED we are on during the beegees loop

// Do nothing but play bad beegees music
// This function is activated when user holds bottom right button during power up
void play_beegees()
{
  //Turn on the bottom right (yellow) LED
  setLEDs(CHOICE_YELLOW);
  toner(CHOICE_YELLOW, 150);

  setLEDs(CHOICE_RED | CHOICE_GREEN | CHOICE_BLUE); // Turn on the other LEDs until you release button

  while(checkButton() != CHOICE_NONE) ; // Wait for user to stop pressing button

  setLEDs(CHOICE_NONE); // Turn off LEDs

  delay(1000); // Wait a second before playing song

  digitalWrite(BUZZER1, LOW); // setup the "BUZZER1" side of the buzzer to stay low, while we play the tone on the other pin.

  while(checkButton() == CHOICE_NONE) //Play song until you press a button
  {
        // iterate over the notes of the melody:
        for (int thisNote = 0; thisNote < 32; thisNote++) {
        changeLED();
        tone(BUZZER2, melody[thisNote],noteDuration);
        // to distinguish the notes, set a minimum time between them.
        // the note's duration + 30% seems to work well:
        int pauseBetweenNotes = noteDuration * 1.30;
        delay(pauseBetweenNotes);
        // stop the tone playing:
        noTone(BUZZER2);
        }
  }
}

// Each time this function is called the board moves to the next LED
void changeLED(void)
{
```

```
424    setLEDs(1 << LEDnumber); // Change the LED
425
426    LEDnumber++; // Goto the next LED
427    if(LEDnumber > 3) LEDnumber = 0; // Wrap the counter if needed
428  }
```