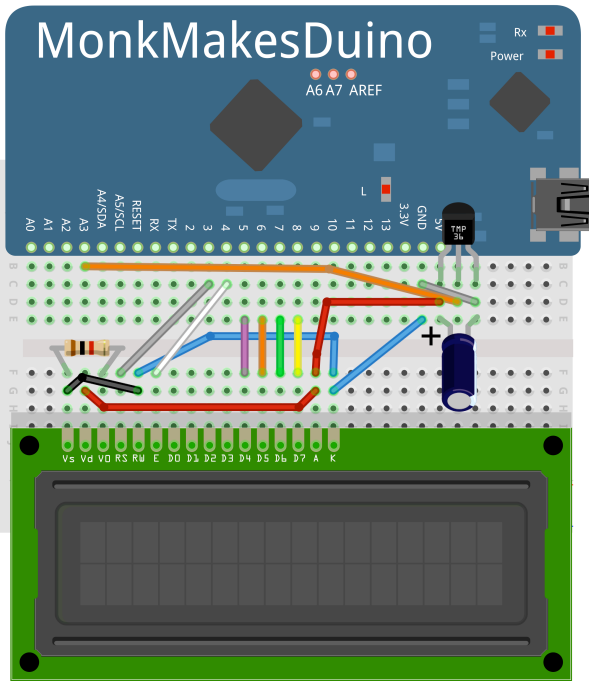


# Instructions:

**MONKMAKESDUINO**

**LCD KIT**

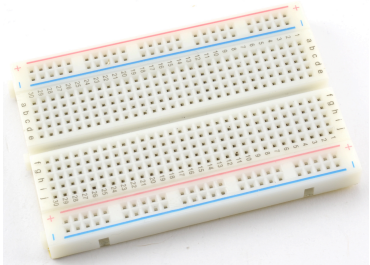
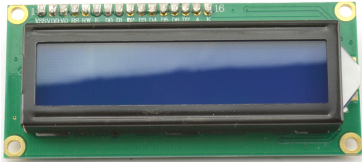




**v1A**










fritzing

# KIT CONTENTS

Before you do anything else, check that you kit includes the following items:

Quantity		
1	MonkMakesDuino board	
1	Micro USB lead	
1	Solderless breadboard	
1	16x2 LCD display module	
20	Male-male jumper wires	
1	TMP36 temperature sensor	
15	270Ω resistor (red, purple, brown)	
5	1k resistor (brown, black, red)	



1	10k trimpot	
1	Photoresistor	
12	Red LED	
1	RGB LED	
1	100μF capacitor	
1	Piezo buzzer	
2	Push switch	

The kit includes a few spare components for your own projects.

# GETTING STARTED

## Step 1. Connect the MonkMakesDuino

Use the USB lead to connect the MonkMakesDuino to a free USB port on your computer. This can be a Mac, Windows or Linux computer.

Note that if your MonkMakesDuino is already attached to the solderless breadboard you do not need to detach it from the breadboard. In fact its better to leave it in the breadboard as its easy to bend its pins trying to get it out.



As soon as you connect the MonkMakesDuino, the PWR LED should light and the 'L' LED start to slowly blink. This means that the MonkMakesDuino works and is drawing power from the USB port. If you have a Windows computer ignore the 'Found New Hardware' wizard and disconnect the MonkMakesDuino before the next step, which is to install USB drivers so that you can communicate with the MonkMakesDuino and program it.

## Step 2. Install the USB Driver

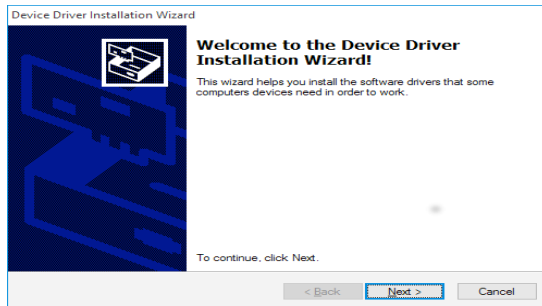
The procedure for installing the driver depends on which operating system your computer uses.

You will find VCP (Virtual Com Port) driver installers for your operating system on this webpage: <https://goo.gl/WofqQn>

Select the download for your operating system and download the ZIP file containing the driver installer ('Download VCP').

## Windows 7 to 10

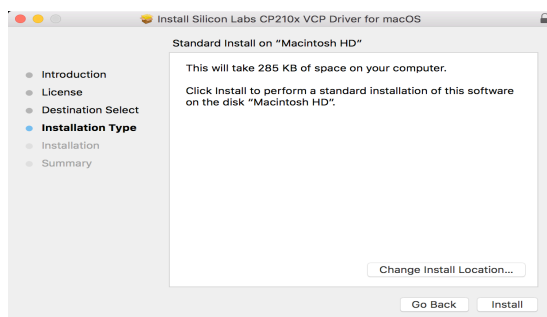
Unzip the ZIP file, which will be called something like CP210x\_Windows\_Drivers.zip and then double click on CP210xVCPInstaller\_x64.exe to launch the installer.



## Mac

To install the drivers on a Mac, scroll down Silicon Labs' download page until you get to the section 'Download for Macintosh OSX' and download the file which will be called something like 'Mac\_OSX\_VCP\_Driver.zip'.

Unzip the file by double-clicking on it. This will create a file called SiLabsUSBDriverDisk.dmg. Double click on this file in turn and then double-click on 'Silicon Labs VCP Driver' to run the installer for the driver.



## Linux

Most distributions of linux have the CP210x drivers pre-installed, so you shouldn't have to do anything.

### Step 3. Install the Arduino IDE

The other software that you must install is the Arduino Integrated Development Environment (IDE). This is the tool you will use to write and unload Arduino programs onto your MonkMakesDuino.

To install it, follow the instructions on the Arduino website, specific to your computer's operating system:

Windows: <https://www.arduino.cc/en/Guide/Windows>

Mac: <https://www.arduino.cc/en/Guide/MacOSX>

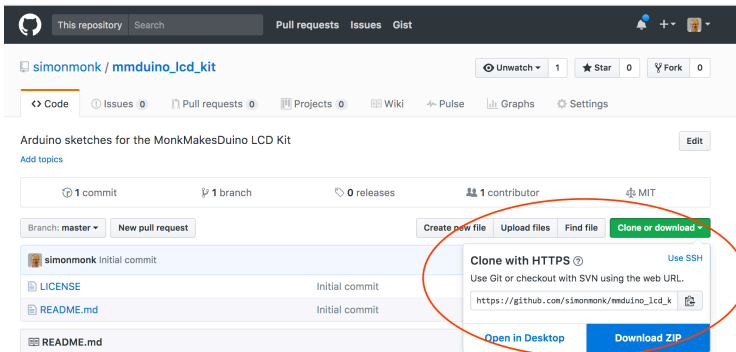
Linux: <https://www.arduino.cc/en/Guide/Linux>

You might notice that the Installer for your operating system offers to install USB drivers. These are the drivers for the official Arduino boards not the MonkMakesDuino, so installing them is a good idea as it will allow you to use official Arduino boards as well as the MonkMakesDuino.

### Step 4. Download the Project Sketches

You can download the sketches (Arduino programs) for this kit's projects from GitHub.

Use the same computer that you just installed USB drivers and the Arduino IDE and direct your web browser to [https://github.com/simonmonk/mmdduino\\_lcd\\_kit](https://github.com/simonmonk/mmdduino_lcd_kit).

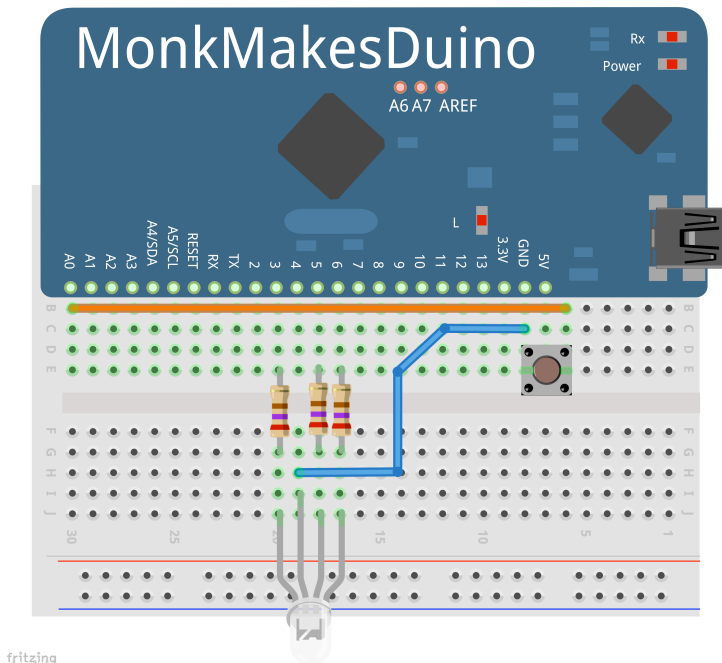


Click on the *Clone or download* button and then select *Download ZIP*. Save the ZIP file onto your desktop and then extract the contents of the ZIP archive. This will create a folder called `mmdduino_lcd_kit-master`. In here you will find a folder for each project. Inside each folder will be a file with the extension of `.ino` which when you double click on it open it in the Arduino IDE.

## CONSTRUCTION

## Breadboard

The projects in this kit are all built using 'Solderless Breadboard'. Breadboard is designed for making electronics prototypes, but is also a great tool for building electronic projects without having to get the soldering iron out. Here is the breadboard layout for project 2.



Underneath the holes in the plastic are metal clips that connect together groups of five holes in a row. So, if you start and look at the leftmost pin of the MonkMakesDunio (A0) this is plugged into one end of a group of five holes all connected together. So to connect a wire or component leg to A0, we can plug it into any of the four holes directly underneath A0. There is then a gap, followed by more sets of five holes all connected together.

At the bottom of the breadboard, you can see two long rows of holes stretching the whole width of the breadboard marked with blue and red lines. Each of these rows

has a single big long clip connecting all the holes together. This makes them useful for situations where you have many connections to make to spread out parts of the project, for instance, to supply power. In this case, the blue negative 'power rail' is used for the ground connection.

The colored lines on the breadboard indicate the use of jumper wires to connect one part of the design to another. Color coding the lines makes it easier to see which wire goes where. Its common to use red wires for the positive supply (5V) and blue or black wires for the ground (GND) connection.

## Parts

Most of the components are supplied individually in the plastic bag. The resistors are connected together with paper tape that can just be pulled off. The resistors and switches have two leads and can be connected either way around. It doesn't matter which way. However the LEDs and capacitor both have a positive lead which is the longer lead. You will find a little + symbol on the diagrams to help you get them the right way around.

The temperature sensor chip has one flat side that is shown on the diagram to make sure you get it the right way around.

## Building a Project

When building a project without risk of damaging the components of the MonkMakesDuino, it's a good idea to follow these steps:

1. Unplug the USB lead from your computer.
2. Remove any components from the breadboard that you don't need in the project you are making.
3. Plug the USB lead back into the computer and upload the sketch for the project you are making (see next section) and then unplug it again. This ensures that when you have finished making the project on the breadboard, the parts and the code will be in agreement.
4. Plug the components onto the breadboard.
5. Check that everything is in the right place and no component leads are touching each other before plugging in.

Before jumping in to any of the more advanced projects, now follow the instructions for Project 1 which will prepare you and your computer for later projects.

# PROJECT 1. BLINKING AN LED

When you first plugged your MonkMakesDuino into your computer's USB port it will have started blinking its 'L' LED. In this first project, you are going to do two things. First make the 'L' LED blink faster and then add an external LED to blink in time with the 'L' LED.

Start by finding the project files for the kit which you downloaded from Github in Step 4 of Getting Started. Find the folder *p1\_blink* and then double click on the only file inside it to open the sketch in the Arduino IDE, like this:

A screenshot of the Arduino IDE interface. The title bar reads "p1\_blink | Arduino 1.8.2". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for opening files, saving, uploading, and downloading. The main text area shows the code for the "p1\_blink" sketch. The code defines a constant integer "ledPin" as 13, sets the pin mode to OUTPUT in the setup function, and then in the loop function, it writes HIGH to the pin, delays for 500ms, writes LOW, and delays for 500ms. The status bar at the bottom indicates "12" and "Arduino/Genuino Uno on COM1".

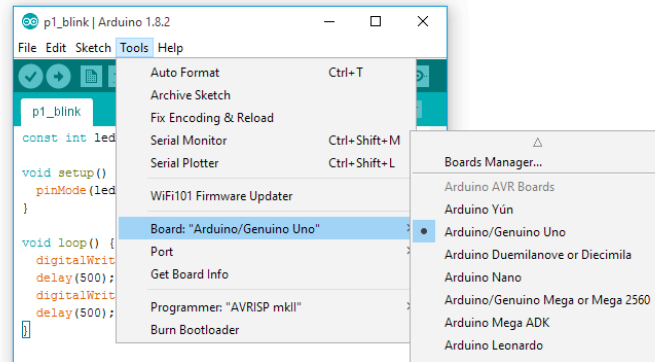
```
const int ledPin = 13;

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

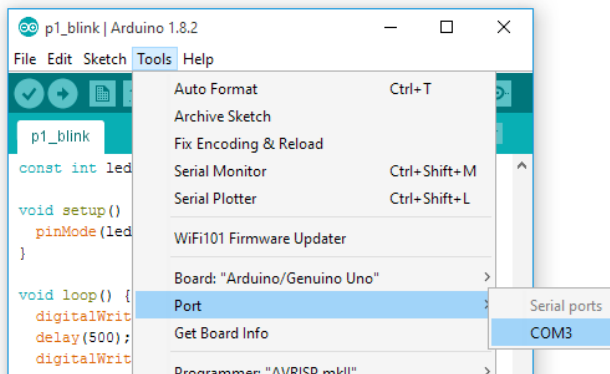
void loop() {
  digitalWrite(ledPin, HIGH);
  delay(500);
  digitalWrite(ledPin, LOW);
  delay(500);
}
```

The first time that you use the Arduino IDE, with a particular board such the MonkMakesDuino you have to tell the Arduino IDE what type of board you are using and which port it is connected to.

From the Tools menu, select Board and then Arduino/Genuino Uno.

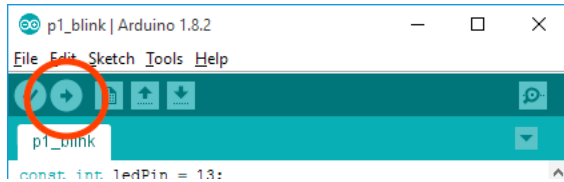


Then, also from the Tools menu, select Port. You will probably find that there is only one option in there, so select it. On Windows machines, the port will be called COM followed by a number. On Linux machines and Macs, the port will have a longer name something like: '/dev/cu.SLAB\_USBtoUART'.



Now we need to transfer the sketch in the Arduino IDE onto the MonkMakesDuino over the USB cable. To do this, click on the Upload button (circled in red below).





If all is well, the sketch will start to upload and the Rx (Receive) LED on the MonkMakesDuino start to flicker. As soon as the uploading is complete, you will see a message 'Done Uploading' appear in the status area of the Arduino IDE.

You should also notice that the LED is now blinking twice as fast as it was.

Lets try and make it blink even faster. Find the two lines in the sketch like this:

```
delay(500);
```

and change them both to be:

```
delay(100);
```

Upload the sketch again and you should see the LED blink really quickly.

Lets take a look at the sketch.

The first line

```
const int ledPin = 13;
```

.. gives a name 'ledPin' to the number (int) 13 and the optional 'const' word tells the Arduino IDE that the value stored in 'ledPin' will not change while the program is running. 13 was chosen because on the MonkMakesDuino board there is a small LED labeled 'L' that is controlled by pin 13.

Most lines of code end in a ';'.

The next block of code

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

.. contains a list of things to do every time a new sketch is uploaded to the MonkMakesDuino or whenever the Arduino is powered up. Every Arduino sketch must have a 'setup function' like this and all that is different between different sketches is what goes between { and }. In this case these is the single line:

```
pinMode(ledPin, OUTPUT);
```

This configures the MonkMakesDuino's pin 13 to be used as an output (it can also be used as an input) but to power an LED, we need it to be an output.

Next we come to the 'loop function'. Like 'setup', every Arduino sketch has to have one of these. In the case of 'loop' the lines inside the { and } are run over and over again until you unplug the MonkMakesDuino. Lets look at the four lines that will be run repeatedly.

```
digitalWrite(ledPin, HIGH);  
delay(100);  
digitalWrite(ledPin, LOW);  
delay(100);
```

The first digitalWrite command sets the LED pin HIGH (5V). The next line

```
delay(100);
```

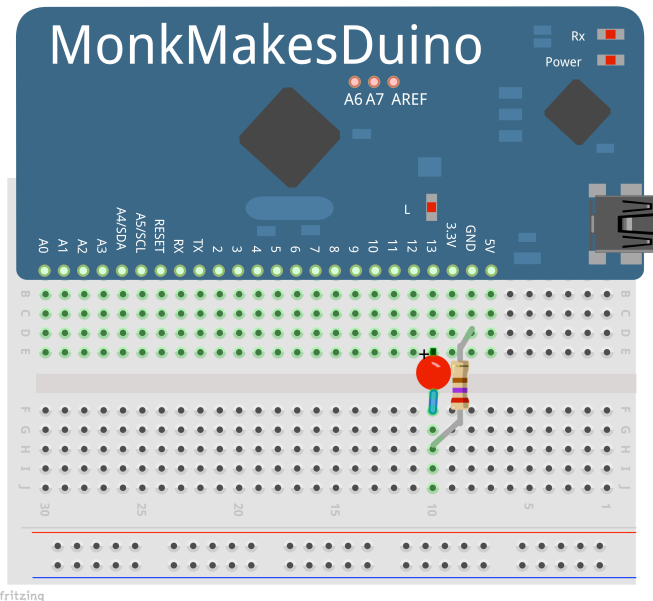
.. delays (does nothing) for 100 milliseconds. A millisecond is 1/1000 of a second, so 100 milliseconds is the same as 1/10 of a second.

The next line sets ledPin LOW to turn the LED off again and then after another delay of 1/10 second the cycle repeats itself.

Let's now make one of the red LEDs included in your kit blink in time with the built-in LED attached to pin 13 of the MonkMakesDuino.

## Component List:

- A red LED
- A 270Ω resistor (red, purple and brown stripes)
- Solderless breadboard



Start by identifying the positive lead of the LED. This will be the longer of its two leads. Connect the positive lead to the same breadboard line as pin 13 (as shown above) and the negative lead of the LED across the breadboard as shown.

Next connect one end of the resistor (it doesn't matter which) to the same line as GND (ground) on the MonkMakesDuino and the other on to the same line as the negative end of the LED.

The breadboard has metal clips under the plastic that connect together all the holes on the same line of five. This is how the negative lead of the LED and resistor are connected together. The resistor is needed to stop the LED drawing too much current.

Now, you should find that the LED blinks in time with the built-in 'L' LED on the MonkMakesDuino.

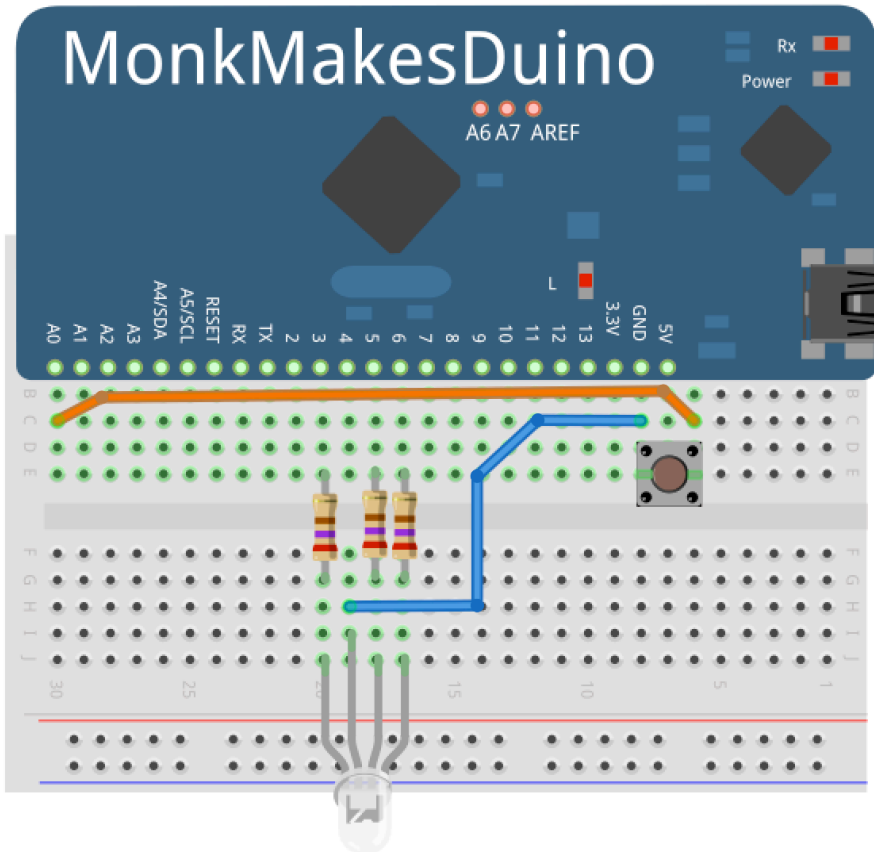
Incidentally, the breadboard drawings for these projects were all made using Fritzing ([fritzing.org](http://fritzing.org)). If you want to use the drawings yourself, you can find them all here: [https://github.com/simonmonk/mmduino\\_lcd\\_kit](https://github.com/simonmonk/mmduino_lcd_kit)

## PROJECT 2. RGB COLOR MIXER

### Component List:

### Sketch: p2\_color\_mixer

- RDB LED
- 3 x 270Ω resistors (red, purple, brown stripes)
- Push switch
- 2 x jumper wires



Pressing the button on the switch will cause the color of the LED to change to one of 10 preset colors.

The RGB LED is actually 3 LEDs in one package. One red, one green and one

blue. By varying the strength of each LED you can mix up different colors. The negative leads of all 3 LEDs are connected to the longest lead of the LED (shown as longer above).

## The Code for Project 2

You may find it helpful to open the file `p2_color_mixer` in the Arduino IDE for this section.

Constants are provided for all the pins used in the project and then you find this line:

```
const int numColors = 10;
```

.. this specifies the number of colors that the LED will display. The actual colors are defined in a data structure called an array. Everything after `//` on the line is 'comment'. That is it helps to explain the code to anyone reading it, but is actually ignored when the sketch is uploaded.

```
int colors[numColors][3] = {
  {0, 0, 0}, // black - off
  {255, 255, 255}, // white
  {255, 0, 0}, // red
  {0, 255, 0}, // green
  {0, 0, 255}, // blue
  {255, 255, 0}, // yellow
  {255, 127, 0}, // orange
  {0, 255, 255}, //
  {255, 0, 255}, //
  {0, 0, 0},
};
```

The array is actually an array of 10 arrays. Each of the 10 elements representing a color as a trio of red, green and blue color intensities between 0 and 255.

If you wish to mix up some more colors, you can add these onto the end of the array, but remember to change 'numColors' to agree with the number of colors in the array.

The variable 'color' holds the position in the array of the current color. In Arduino C, the index positions of an array start at 0 rather than 1.

```
int color = 0;
```

The 'setup' function sets the red, green and blue LED control pins to be outputs and the switchPin to be `INPUT_PULLUP`. Specifying `INPUT_PULLUP` rather than just `INPUT` means that the input is biased towards 5V (HIGH) and so the switching action occurs when the input is connected to GND.

```
void setup() {  
  pinMode(redLedPin, OUTPUT);  
  pinMode(greenLedPin, OUTPUT);  
  pinMode(blueLedPin, OUTPUT);  
  pinMode(switchPin, INPUT_PULLUP);  
}
```

The 'loop' function is where most of the action takes place.

```
void loop() {  
  int red = colors[color][0];  
  int green = colors[color][1];  
  int blue = colors[color][2];  
  
  analogWrite(redLedPin, red);  
  analogWrite(greenLedPin, green);  
  analogWrite(blueLedPin, blue);  
  
  if (digitalRead(switchPin) == LOW) {  
    color++;  
    if (color == numColors) {  
      color = 0;  
    }  
    delay(200);  
  }  
}
```

First of all, the color array for the currently selected color is split into separate red, green and blue values. The 'analogWrite' Arduino command is then used to set the intensity of each color channel.

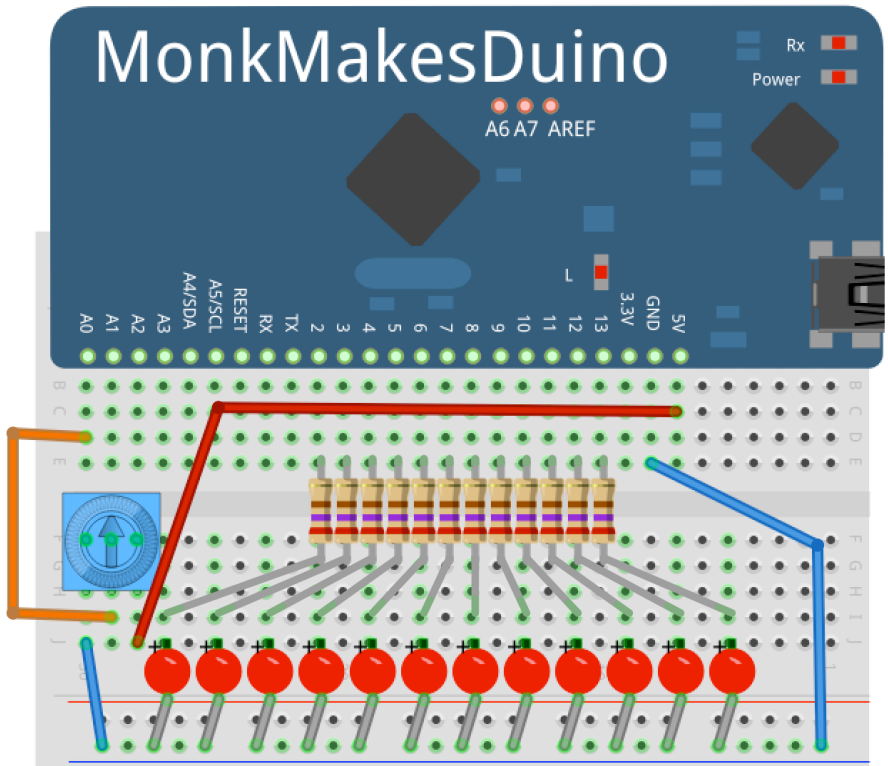
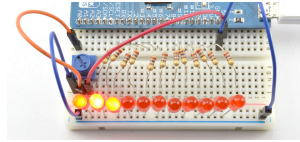
Finally, the 'loop' function checks to see if the switch has been pressed and if it has, it adds 1 to the color index 'color'. If 'color' goes above 'numColors' then 'color' is set back to 0 to begin the cycle of colors again. The 'delay' command ensures that the colors don't change too fast when you have the finger on the button.

## PROJECT 3. LARSON SCANNER

### Component List:

- 12 x red LED
- 12 x 270Ω resistors (red, purple, brown stripes)
- 10k Trimpot
- 4 x jumper wires

### Sketch:p3\_larson\_scanner



The Larson scanner is named after Glen A Larson creator of the Knight Rider and Battle Star Galactica TV shows. The lights scan back and forth at a speed controlled by the knob of the variable resistor (blue thing on the left)

You may find it helpful to open the file p3\_larson\_scanner in the Arduino IDE for this section.

The LED pins are defined in an array to make it easy to step up and down the LEDs lighting each in turn. Although the LED numbers are actually sequential using an array allows the flexibility to use more LEDs (perhaps using the A1 to A5 pins).

```
const int ledPins[] = {2, 3, 4, 5, 6, 7, 8,  
                      9, 10, 11, 12, 13};  
const int potPin = A0;  
const int n = 12;
```

'potPin' refers to the pin to be used as an analog input to read the voltage of the variable resistor (pot). The pins A0 to A5 on the MonkMakesDuino that can be used to measure voltage.

```
void setup() {  
  for (int i = 0; i < n; i++) {  
    pinMode(ledPins[i], OUTPUT);  
  }  
}
```

The 'setup' function loops around all 12 of the pins setting them to be outputs.

The 'loop' function contains two 'for' loops that loop over each LED in turn, turning the LED on, delaying for a period set by the function 'calculateDelay'. One loop mores from left to right, the other right to left.

```
void loop() {  
  for (int i = 0; i < n; i++) {  
    digitalWrite(ledPins[i], HIGH);  
    delay(calculateDelay());  
    digitalWrite(ledPins[i], LOW);  
  }  
  for (int i = n-1; i > 0; i--) {  
    digitalWrite(ledPins[i], HIGH);  
    delay(calculateDelay());  
    digitalWrite(ledPins[i], LOW);  
  }  
}
```

The 'calculateDelay' function converts the knob position to a delay period between 10 and 200 milliseconds.

```
int calculateDelay() {  
  int knobPosition = analogRead(potPin);  
  int t = map(knobPosition, 0, 1023, 10, 200);  
  return t;  
}
```

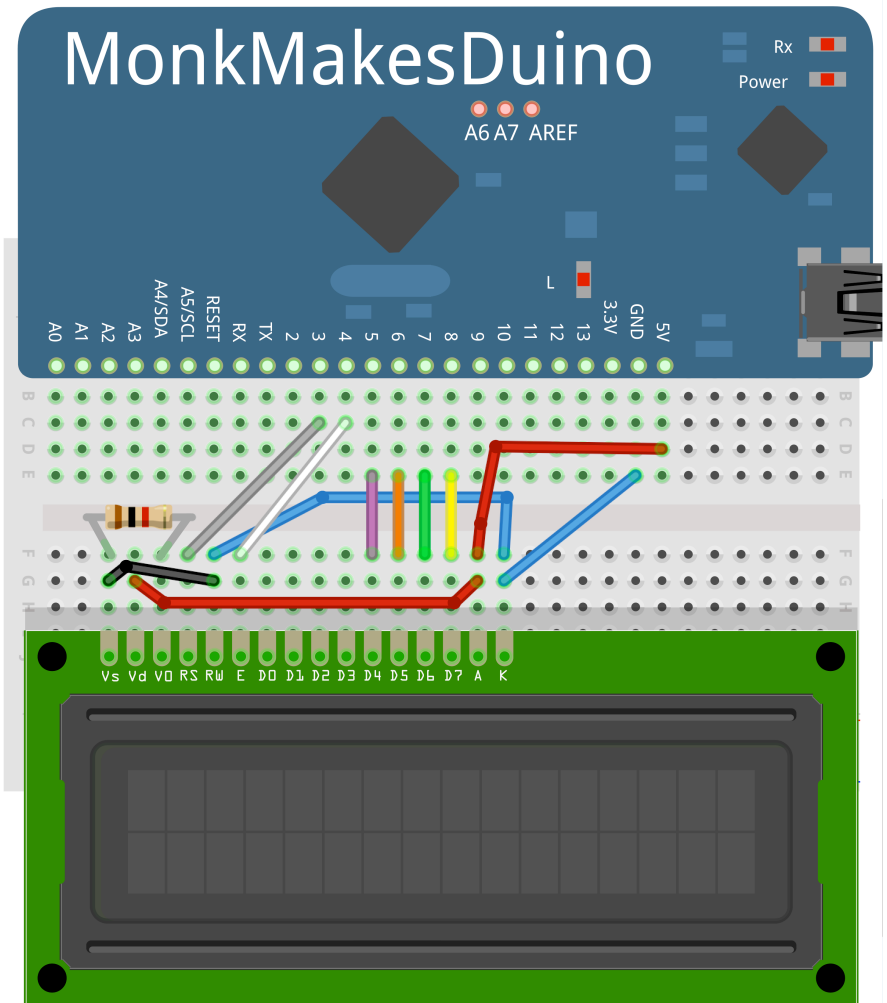


## PROJECT 4. LCD MESSAGE BOARD

### Component List:

- LCD Display
- 1k resistor (brown, black, red stripes)
- 11 x jumper wires

### Sketch: p4\_lcd\_message\_board

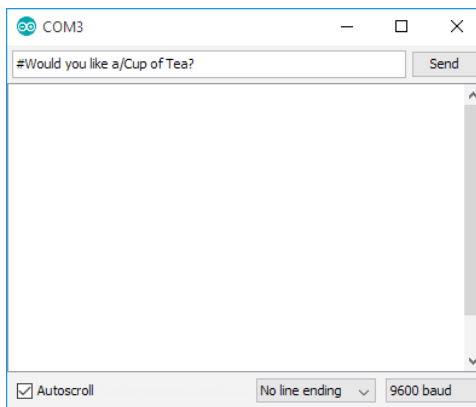


The Trimpot adjusts the contrast of the display. So, if when you first plug in the project, you might not see anything on the display until you have adjusted the knob on the trimpot.

This project displays an initial message, but you can send your own messages to display from your computer. To do this, you first need to open the Arduino IDE Serial Monitor by clicking on the icon for it circled below.



This will open a separate window (the Serial Monitor) which is used to communicate with the MonkMakesDuino.



Make sure that the drop down list at the bottom right of the Serial Monitor has '9600 baud' selected. This is the communication speed between your computer and the MonkMakesDuino.

You can now enter messages in the text area at the top of the Serial Monitor and send them to the MonkMakesDuino by pressing the 'Send' button. The character # clears the display and / starts a new line on the LCD display.

## The Code for Project 4

The code for the project starts with two lines that are needed for all the projects that use the LCD display.

```
#include <LiquidCrystal.h>

//          RS EN D4 D5 D6 D7
LiquidCrystal lcd(3, 4, 5, 6, 7, 8);
```

The 'include' statement tells the Arduino IDE that the LiquidCrystal library is needed by the sketch. Arduino libraries are code modules for special purposes, such as different types of display, sound etc.

The next line, that starts with // is a comment to remind you which of the LCD display's control pins are being specified on the line underneath it.

This line initializes the LCD library specifying which pins of the MonkMakesDuino are connected to which pins of the LCD module. So, from left to right: pin 3 of the MonkMakesDuino is connected to the RS connection of the LCD module; pin 4 to EN and so on. Rather confusingly, the LCD module has pins named D4 to D7 like the MonkMakesDuino.

```
void setup() {  
  Serial.begin(9600);  
  lcd.begin(2, 16);  
  lcd.clear();  
  lcd.setCursor(0,0);  
  lcd.print("MonkMakes");  
  lcd.setCursor(0,1);  
  lcd.print("LCD Kit");  
}
```

The 'setup' function begins serial communication. The number 9600 refers to the communication speed, which has to match whatever speed you set on the Serial Monitor.

The next line tells the LCD library how many rows and columns of characters the display has. In this case its 2 rows of 16 characters.

The remainder of the 'setup' function is concerned with displaying a default message before you start sending your own messages.

The 'loop' function first checks to see if any characters are available to be read as a result of being sent from the Serial Monitor. If there are, they are read and written to the LCD display. The special characters of # and / are also handled.

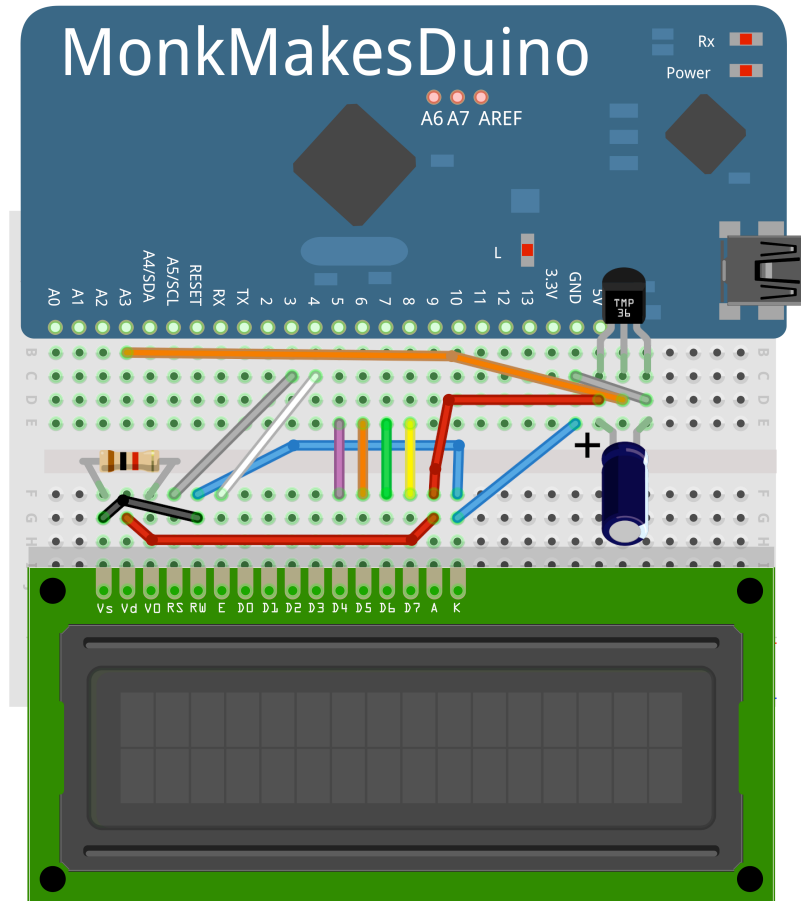
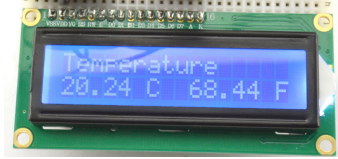
```
void loop() {  
  if (Serial.available()) {  
    char ch = Serial.read();  
    if (ch == '#') {  
      lcd.clear();  
    }  
    else if (ch == '/') {  
      lcd.setCursor(0,1);  
    }  
    else {  
      lcd.write(ch);  
    }  
  }  
}
```

## PROJECT 5. LCD THERMOMETER

### Component List:

- LCD Display
- 1k resistor (brown, black, red stripes)
- 12 x jumper wires
- TMP36 temperature sensor
- 100µF capacitor

### Sketch: p5\_lcd\_thermometer



This project displays the temperature in both degrees C and F.

The temperature sensor chip (TMP36) has three pins. One is connected to GND another to 5V to supply it with power and the middle 'OUT' pin will produce a voltage between 0 and 5V that is proportional to the temperature. This output is connected to pin A3 of the MonkMakesDuino, which is capable of measuring voltages.

The capacitor is needed to help keep the supply voltage from the USB connection to the MonkMakesDuino as stable as possible, so as not to add errors to the measured temperature.

## The Code for Project 5

The code to import and initialize the LCD library is the same as for the previous project.

The 'loop' function sets the cursor position to the start of the second line, reads the temperature in degrees C, converts that to degrees F and then writes both out to the LCD display. The half-second delay ensures that small fluctuations in the temperature do not cause the display to flicker unreadably.

```
void loop() {  
  lcd.setCursor(0, 1);  
  float tempC = readTemp();  
  float tempF = tempC * 9.0 / 5.0 + 32.0;  
  lcd.print(tempC);  
  lcd.print(" C ");  
  lcd.print(tempF);  
  lcd.print(" F");  
  delay(500);  
}
```

The 'readTemp' function reads the 'raw' input value from the 'tempPin' (A3). This will return a value between 0 and 1023. 0 for 0V and 1023 for 5V. This is used to first calculate the voltage ('volts'). The value 205.0 comes from the range 1023 divided by 5V.

The temperature in degrees C can then be calculated from the voltage as the voltage \* 100 minus 50. For example, if the voltage is 0.7 the temperature is :

$0.7 * 100 - 50 = 20$  degrees C.

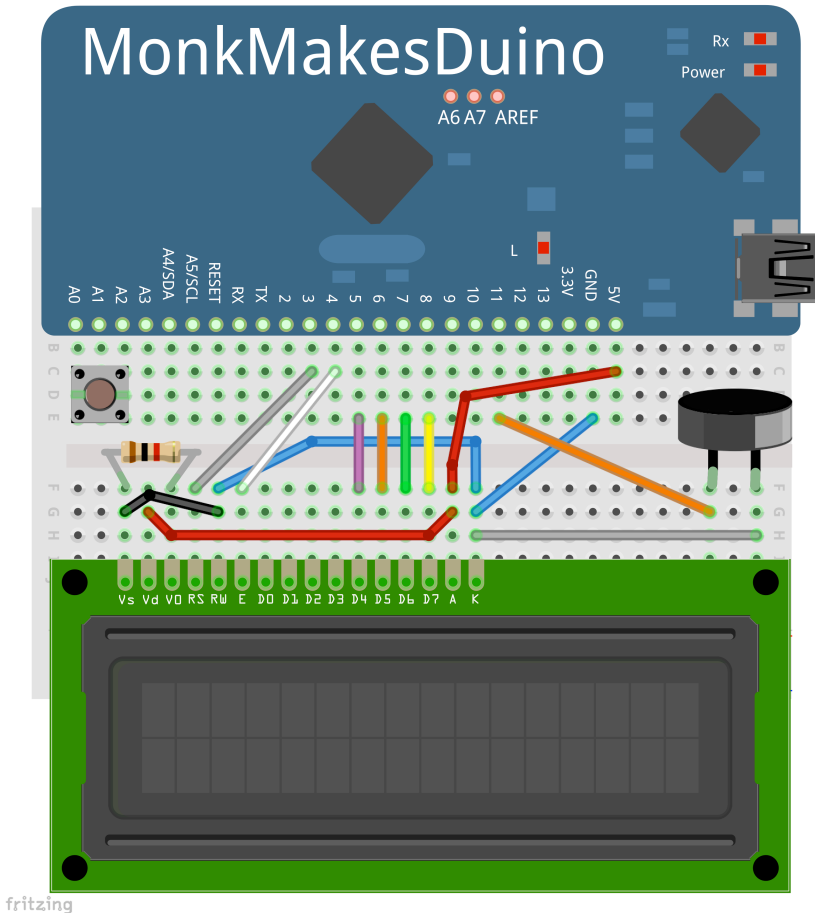
```
float readTemp() {  
  int raw = analogRead(tempPin);  
  float volts = raw / 205.0;  
  float tempC = 100.0 * volts - 50;  
  return tempC;  
}
```

# PROJECT 6. LCD COUNTDOWN TIMER

## Component List:

- LCD Display
- 1k resistor (brown, black, red stripes)
- 13 x jumper wires
- Push switch
- Piezo buzzer

## Sketch: p6\_lcd\_timer



This project is good as a kitchen timer. You use short presses of the button to set the number of minutes and then a long press to start the timer running. When the countdown is complete, the buzzer will sound. This can be cancelled by pressing the button again.

To simplify the wiring, the button is connected between pins A0 and A2. A2 is permanently set LOW and A1 read as a digital input to see if the button was pressed.

## The Code for Project 6

This is the most complicated sketch of all the projects.

After the usual LCD imports and pin definitions there are some more values defined:

```
const int SET_MODE = 1;
const int RUN_MODE = 2;
const int ALARM_MODE = 3;

const int SHORT_PRESS = 1;
const int LONG_PRESS = 2;
```

These constants are used to keep track of which mode the timer is in ('set' the minutes, 'run' or 'alarm' when the timer has got to zero.

The values SHORT\_PRESS and LONG\_PRESS are used to indicate the type of button press.

After this, there are some variables defined:

```
int mode = SET_MODE;
int mins = 4;
int secs = 0;
long lastUpdateTime = 0;
```

'mode' indicates the current mode that the timer is in and this is initialized to SET\_MODE. The variables 'mins' and 'secs' are used to keep track of the minutes and seconds of the timer.

'lastUpdateTime' is used to keep track of when the counter last ticked by a second, so that the sketch knows when to do it again.

The 'loop' function first displays the 'time' and then calls one of three different handler functions depending on which mode the timer is in.

```
void loop() {
  displayTime();
  if (mode == SET_MODE) {
    handleSetMode();
  }
  else if (mode == RUN_MODE) {
```

```

    handleRunMode();
}
else if (mode == ALARM_MODE) {
    handleAlarmMode();
}
}
}

```

Lets take the first of these handlers (handleSetMode).

```

void handleSetMode() {
    int switchPress = checkSwitch(switchPin1);
    if (switchPress == LONG_PRESS) {
        mode = RUN_MODE;
        return;
    }
    if (switchPress == SHORT_PRESS) {
        mins++;
        if (mins > 10) {
            mins = 0;
        }
    }
}
}

```

This first calls the function 'checkSwitch' and assigns the result to 'switchPress'. If the switch press is long, then the mode variable is set to RUN and return is called to exit the function. If, on the other hand, the press is short then 1 is added to 'mins' and if that makes 'mins' greater than 10, its set back to 0 to cycle around.

The other handlers follow a similar pattern. Here is the handler for run mode:

```

void handleRunMode() {
    if (checkSwitch(switchPin1) == SHORT_PRESS) {
        mode = SET_MODE;
        return;
    }
    if (mins == 0 && secs == 0) {
        mode = ALARM_MODE;
        tone(buzzerPin, 1000);
        return;
    }
    updateTime();
}
}

```

This starts by checking for a short button press to send it back to 'set' mode. It then checks to see if the timer has counted down to zero and if it has, sets the mode to 'alarm' and uses the 'tone' command to set 'buzzerPin' oscillating at 1000 times a second to make the buzzer sound. Finally the function calls 'updateTime' to decrease the second and minute count.

```

void updateTime() {
    long now = millis();
    if (now > lastUpdateTime + 1000) {

```



```

secs --;
if (secs < 0) {
    secs = 59;
    mins --;
}
lastUpdateTime = now;
}
}

```

'updateTime' uses 'millis' to get the current number of milliseconds since the MonkMakesDuino last reset and compares this with 'lastUpdateTime' and if 1000 milliseconds (1 second) has elapsed, it subtracts 1 from seconds. If this makes 'secs' negative then 'secs' is set back to 59 and 'mins' is decreased by 1.

The 'checkSwitch' function is something that you might want to copy and use in your own programs. It takes the pin to monitor for presses as a parameter so that you can use it for multiple switches if you want to.

If the button is not pressed, it just returns immediately with a value of 0. If the previous button state was 0 then this indicates the button being newly pressed and the function effectively starts a timer for 1 second. If the button is released before the timer times out SHORT\_PRESS is returned, otherwise as soon as the timer exceeds a second of button press, the function returns LONG\_PRESS.

```

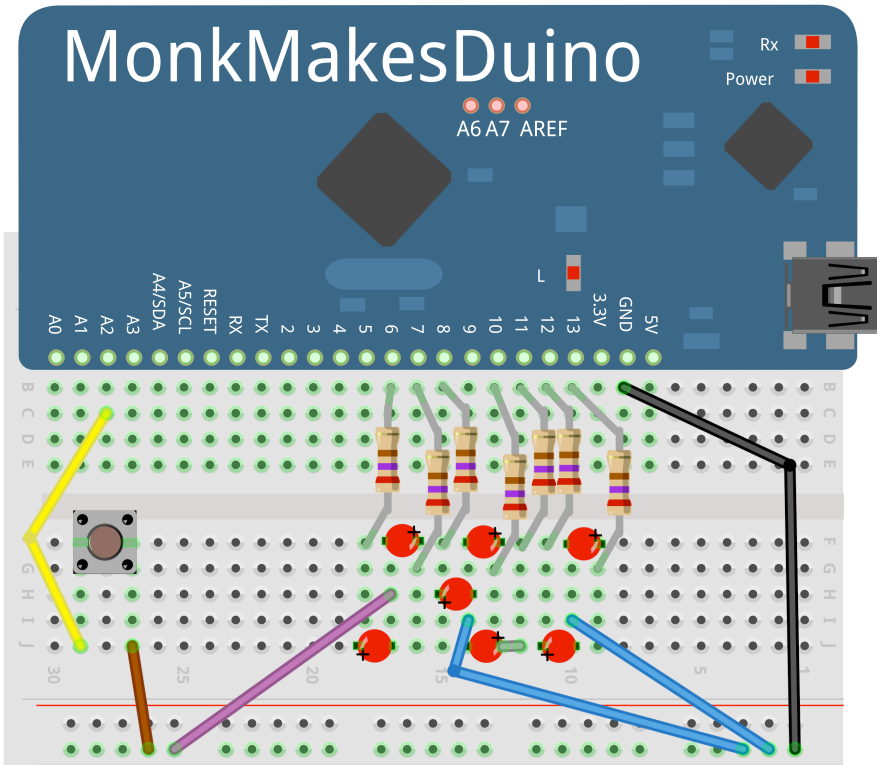
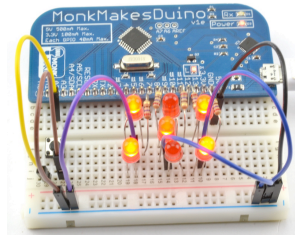
int checkSwitch(int pin) {
    static int previousState = 0;
    if (digitalRead(pin) == HIGH) {
        previousState = 0;
        return 0;
    }
    else if (previousState == 0) // and button down
    {
        long t0 = millis();
        while (digitalRead(pin) == LOW) {
            if ((millis() - t0) > 1000) {
                previousState = LONG_PRESS;
                return LONG_PRESS;
            }
        }
        previousState = SHORT_PRESS;
        return SHORT_PRESS;
    }
    return 0;
}

```

## PROJECT 7. LED DICE

### Component List:

- 7 x red LED
- 7 x 270Ω resistor (red, purple and brown stripes)
- 6 x Jumper wires
- Push switch



fritzing

When building up this circuit take particular care to make sure that none of the component leads are touching. Also note that the LEDs are not all the same way around.

Although this project doesn't have the LED 'pips' in quite the right positions, bending

the LED leads a little will allow you to make a dice-like layout.

When you press the button the LEDs will flash for a short time before the final throw of the dice is shown.

## The Code for Project 7

As with project 3, the pins used to control the LEDs are contained in an array. To make it easier to see which pin controls which of the 7 LEDs, the array is layed-out in a dice shape.

```
const int ledPins[] = {  
  13,    12,  
  10,   8,  11,  
   7,    6  
};
```

This shows how the Arduino IDE really doesn't care about where you place spaces and new lines. They are really just there to make the code easier to understand.

The possible end results of throwing the dice are also held in an array:

```
const byte throws[] = {  
  // 6543210  
  0b00001000, // 1 thrown  
  0b01000001, // 2  
  0b01001001, // 3  
  0b01100011, // 4  
  0b01101011, // 5  
  0b01110111, // 6  
};
```

In this case, each of the six possible throws (from 1 to 6) are each represented as a byte. A byte is made up of eight bits, and each bit can either be 0 or 1 (off or on). Each bit corresponds to one of the LEDs, with the spare bit at the beginning (left to right) of the byte always 0.

The 'setup' function sets the pin modes as usual, but also on the last line starts the random number generator on a seed value taken from the analog reading of the unused analog input A0.

```
void setup() {  
  pinMode(switchPin, INPUT_PULLUP);  
  for (int i = 0; i < 7; i++) {  
    pinMode(ledPins[i], OUTPUT);  
  }  
  randomSeed(analogRead(A0));  
}
```

If you were to try removing this line, you would find that the dice made the same sequence of throws each time you reset your MonkMakesDuino. This is because the numbers generated are not truly random, but just from a large series of numbers, that has a random distribution (ask a mathematician).

The main 'loop' function uses random numbers in two ways.

```
void loop() {
  if (digitalRead(switchPin) == LOW) {
    int shakes = random(10, 30);
    for (int i = 0; i < shakes; i++) {
      int x = random(6);
      show(x);
      delay(100);
    }
  }
}
```

Whenever the button is pressed, it first decides on a random number of 'shakes' between 10 and 30. For that number of times it comes up with another random number between 1 and 6 and display that on the LEDs using the 'show' function. The last of these 'shakes' will be the throw displayed on the LEDs when all the blinking has stopped.

The 'show' function looks up the 'bitPattern' of on/off LEDs that it will need to display for that throw from the 'throws' array, and then for each bit of the pattern sets the corresponding led control pin on or off.

```
void show(int x) {
  byte bitPattern = throws[x];
  for (int i = 0; i < 7; i++) {
    digitalWrite(ledPins[i], bitRead(bitPattern, i));
  }
}
```

## TROUBLESHOOTING

**Problem:** The Power LED of the MonkMakesDuino does not light when you plug it in.

**Solution:** Disconnect the MonkMakesDuino from the breadboard and try again. If the LED lights now, then something is wrong with your wiring. Try a different USB lead.

**Problem:** The port for the MonkMakesDuino does not appear in the list of ports in the Arduino IDE.

**Solution:** Check that the drivers are installed ok (see page 4). Try rebooting your computer. If you are using your own USB lead, try using the one supplied with the kit, because some leads are charge only and will not work.

**Problem:** One of the LEDs doesn't light.

**Solution:** Make sure its the right way around. If that doesn't work carefully check the wiring.

**Problem:** You can't see any text on the LCD display.

**Solution:** Check the wiring to the LCD. If the LCD backlight is not lit, check the connections near the LCD display's A and K pins.

For other support questions, please check out the product's web page (<http://monkmakes.com/mmd>) and you can send support questions to [support@monkmakes.com](mailto:support@monkmakes.com)

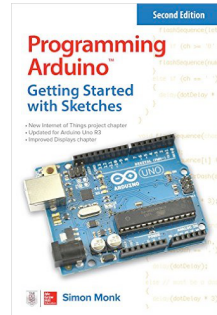
# WHAT NEXT?

## Arduino Programming

If you want to learn more about programming Arduino, then you should consider my best selling book 'Programming Arduino: Getting Started with Sketches', which is available from all major book sellers.

You can find out more about my books at:  
<http://simonmonk.org> or follow me on Twitter where I am @simonmonk2

You will also find tutorial series on [learn.adafruit.com](http://learn.adafruit.com) which while written for Arduino will work equally well on MonkMakesDuino: <https://goo.gl/7TEYDr>



## MonkMakes

For more information on the MonkMakesDuino, the projects home page is here:  
<https://monkmakes.com/mmd>

As well as this kit, MonkMakes makes all sorts of kits and gadgets to help with your maker projects. Find out more, as well as where to buy here:  
<https://monkmakes.com> you can also follow MonkMakes on Twitter @monkmakes.



From left to right: Basic Components Kit, Hacking Electronics Kit and MonkMakes Protoboard.