Arduino Lab

# Project 1: LED blink

This project will use single digital output pin to turn an LED on and off.

With the following circuit, calculate the value for resistor R1. (Trick – what is the output voltage of your Arduino?)

Arduino Pin 13

R1

Create the following circuit on the breadboard.

13

5V

GND

Arduino

R1

+

-

Open the Arduino software.

The steps in creating a program (Arduino sketch) are:

1. Write the code
2. Compile the code
3. Upload the code to the board
4. Run the code

When you upload the code, it automatically resets the board and begins running the code. The code will always run when the board is powered on. To stop and start it running, you would need to add a switch to your circuit, attach it to a pin on the board and watch its state.

1. Type the following code into a new sketch.

const int LED = 13; // The pin our LED is attached to

void setup()

{

pinMode(LED, OUTPUT); // Tells it pin 13 will be used for output

}

void loop()

{

digitalWrite(LED, HIGH);

delay(1000);

digitalWrite(LED, LOW);

delay(1000);

}

1. (and 3 + 4) Click on Upload, or press Ctrl+U to compile and upload the code to the Arduino board.

Your LED should now blink on and off with a 1 second interval. You can change the rate of the blink by changing the number in the call to delay().

# Project 2: LED fade

This project will use a digital PWM output to fake an analog output, giving us the ability to alter the brightness of an LED.

Change the circuit from Project 1 so that the LED is plugged in to pin 9 instead of pin 13. It should be connected as follows.

9

5V

GND

Arduino

R1

+

-

In the Arduino environment, create a new sketch (File -> New, or Ctrl+N). Type in the following code, and press Ctrl+U to upload the sketch.

const int LED = 9; // The pin our LED is attached to

void setup()

{

pinMode(LED, OUTPUT); // Tells it pin 9 will be used for output

}

void loop()

{

int loopCounter; // Counts how many times we’ve been through our loop

for (loopCounter = 0; loopCounter < 25; loopCounter++)

{

analogWrite(LED, loopCounter\*10);

delay(50);

}

for (/\* No initialization \*/; loopCounter >= 0; loopCounter--)

{

analogWrite(LED, loopCounter\*10);

delay(50);

}

}

## What does all the code do?

const int LED = 9;

This alone has a few concepts.

### Variables and types

A variable is a named value holder. We can have variables of different types: **int** (integer – whole numbers), **float** (floating point numbers – those with decimal places), **boolean** (true or false) and others. A variable can be given a value, and then later we can change that value. They allow us to hold values for use later on, store results of calculations etc.

In this line, we have declared an integer variable called LED. I have used all capitals here to indicate it is a constant (more on that later). Variable names can be upper or lower case, and they **cannot** contain spaces.

### Assignment (=)

Equals normally shows that something is equal. In programming, equals (=) is the assignment operator. This assigns a value to a variable.

In our line here, we are assigning the variable LED a value of 9. It is read as ‘LED is assigned 9’, and not ‘LED equals 9’.

To test for equality, we use double equals (==).

### const modifier

This tells us that this variable is actually a constant. Constants don’t change value. (No kidding!) So once the value has been assigned, it cannot be changed.

### setup()

This function does any setup work, and is run once when the sketch starts. For our program, it sets pin 9 (for our LED) to output mode, using the line

pinMode(LED, OUTPUT); // Tells it pin 9 will be used for output

### loop()

This loop runs forever while it is turned on.

int loopCounter; // Counts how many times we’ve been through our loop

This line declares an integer variable, which we will use to count how many times we have iterated through the loop. This is just what the comments say. Comments on a single line start with // and run until the end of the line. Comments are ignored and do not run as part of the program.

for (loopCounter = 0; loopCounter < 25; loopCounter++)

This is a **for loop**. A for loop has the following form:

for (initialization; condition when the loop stops; runs after each time through loop)

So in our case, we first initialize the loopCounter to 0. Then after each time through the loop, we increment the counter by 1 (++ operator). This does the same as loopCounter = loopCounter + 1. We check, after each time through the loop, to see if loopCounter is less than 25 – if it is, we continue through the loop again; if it is greater than 25, we exit the loop and continue with the rest of the code.

analogWrite(LED, loopCounter\*10);

This writes a value to the output pin between 0 and 255 (or in our case, 250). We have used the loop counter to help calculate this value.

delay(50);

This delays for 50ms, otherwise the code runs so fast the LED looks like it’s either on all the time or flickering slightly. (Try it!) You can speed up or slow down the rate of fade by changing this value, or altering how many times we iterate through the loop.

The second half of the code is the same, but in reverse.

# Project 3: Analog input

This project will read a varying voltage from a potentiometer using an analog input, and display it on a console.

Create the following circuit on the breadboard and connect it to the Arduino as shown.

A0

5V

GND

Arduino

10k

+

-

In the Arduino environment, create a new sketch (File -> New, or Ctrl+N), and type in the following code.

const int POT\_INPUT = A0;

int value = 0;

void setup()

{

Serial.begin(9600); // Sets up serial communication and sets the rate

}

void loop()

{

value = analogRead(POT\_INPUT);

Serial.println(value);

delay(50);

}

There are a few new things in this code.

**1. Serial.begin(9600)**

This initializes serial communication between the Arduino and the computer. It will transmit at a rate of 9600 bits per second.

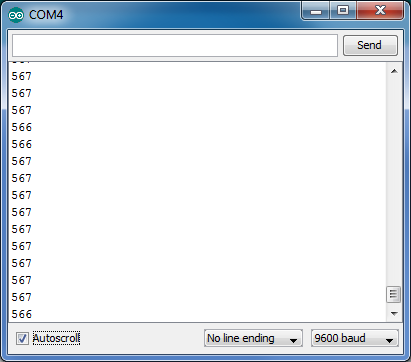
**2. value = analogRead(POT\_INPUT)**

This reads the current value of the potentiometer and stores it in the variable ‘value’. The value read will be on a scale of 0-1023. This is because the Analog to Digital Converter (ADC) is a 10-bit device, giving us 1024 possible values. So the voltage range of 0-5V DC will be scaled across the range 0-1023.

**3. Serial.println(value)**

This writes the value to the serial output, which we can then view, as we will see in a minute.

Compile the sketch and upload to the Arduino by pressing Ctrl+U. Once the program has uploaded, go to the Tools menu and select Serial Monitor, or simply press Ctrl+Shift+M. This will bring up the Serial Monitor in a new window. This allows us to see everything written to the serial output, and will look like this.



As you turn the potentiometer, you should see the value change. Now, how do we know what voltage it is? With maths!

Change the loop function to be the following.

void loop()

{

value = analogRead(POT\_INPUT);

float voltage = 5.0 \* ((float)value / 1024);

Serial.print(“value: “);

Serial.print(value);

Serial.print(“, voltage: “);

Serial.println(voltage);

delay(50);

}

We are now converting the value we read back into a voltage, by multiplying our maximum voltage by the fraction we are reading. Then we are printing a single line of output in steps. Notice only the final print call is println, so everything is written on a single line.

# Project 4: Potentiometer controlled LED dimmer

This project will use what we have done so far and combine it all together. We will read the value from the potentiometer, and control the brightness of an LED according to the current value.

Create the following circuit and connect it to the Arduino as shown. (Don’t forget the value of R1).

A0

5V

GND

Arduino

10k

+

-

R1

9

const int LED = 9; // The pin our LED is attached to

const int POT\_INPUT = A0;

int value = 0;

int ledBrightness = 0;

void setup()

{

pinMode(LED, OUTPUT); // Tells it pin 9 will be used for output

}

void loop()

{

value = analogRead(POT\_INPUT);

ledBrightness = map(value, 0, 1023, 0, 255);

analogWrite(LED, ledBrightness);

delay(20);

}

The map function has the following form:

int map(value, fromLow, fromHigh, toLow, toHigh)

It maps a value (value parameter) that is on a scale of fromLow -> fromHigh to a new scale of toLow -> toHigh. In our case, we are taking a value that we read and converting from a scale of 0-1023 to a scale of 0-255.

We need to do this so that we take our 10-bit analog input and convert that value to an 8-bit scale. We can then write that value (the LED brightness) directly to pin 9 as a PWM output.

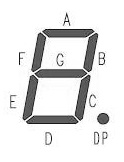
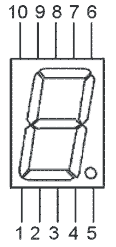
Compile and upload the sketch by pressing Ctrl+U. When the program runs, you should see the LED brightness change as you turn the potentiometer.

# Project 5: 7 Segment Display

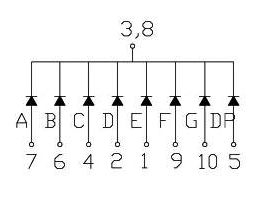
This project will connect a 7 segment display to the Arduino, allowing us to display the numbers 0-9.

## 7 Segment displays

7 segment displays have 7 LEDs arranged to form a figure 8. Each one can be turned on individually so as to form numbers or (some) letters. Their pinout is different to normal ICs, and is as follows.



Each segment is identified with a letter, and each letter corresponds to a pin, with two common pins. The internal circuit diagram looks like this.



## Driving the 7 segment display

With 7 LEDs (8 including the decimal point), each one requires a resistor so it doesn’t burn out, and each one requires an output pin so they can be turned on individually.

## Step 1: Draw the circuit diagram

Complete the following schematic for the circuit described here.

* Label each LED of the seven segment display with A-G.
* Label each LED with its corresponding pin number.
* Connect each LED to an individual digital output pin on the Arduino, through a resistor.
* Label each digital output 1-7 for display segments A-G (so 1 connects to A, and so on).
* Connect the cathode to the appropriate power rail.
* From the 7 segment display datasheet, calculate the resistor value needed. (The Arduino output will either be 5V or 3.3V, depending on which one you have.)

5V

GND

Arduino

+

-

## Step 2: Connect the dots

A connection table helps us to plan ahead and make sure that the right things are connected together, before we plug things in and blow things up.

Fill in the following table, showing what connections are being made. This will also help later when programming.

|  |  |  |
| --- | --- | --- |
| Segment | Arduino pin | 7-segment pin |
| A |  |  |
| B |  |  |
| C |  |  |
| D |  |  |
| E |  |  |
| F |  |  |
| G |  |  |
| DP |  |  |
| - | GND | 3 |
| - | GND | 8 |

## Step 3: Build the circuit

Once you have checked your work, build the circuit on the breadboard. Be careful to make the right connections, and make sure resistors are not touching.

## Step 4: Making numbers

The next step, of course, is to actually make a number appear on the display. But what segments do you need to turn on to make each number?

Below, draw each digit 0-9, in segments, to show what they would look like. 0 is given as an example.

Fill in the following table, putting a 1 (on) or 0 (off) in the column for each segment, for each number. Use the diagram of the 7-segment display for reference. 0 is completed as an example.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Digit** | **A** | **B** | **C** | **D** | **E** | **F** | **G** |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |

## Step 5: The code

Now we can translate all this into some code to make the numbers show on the display.

## A short diversion

For this program, we will use arrays. Arrays are a collection of variables that are referred to by one name. Their values are retrieved by using an index to access each one. If a variable is your mailbox at home, arrays are the Post Office boxes, and the index in the array is the address of the PO Box.

Integer Variable: value

0

1

2

3

4

5

6

7

8

9

Integer Array: values

Here, *value* is a single integer variable, and *values* is an array of 10 integers. We can access the items in the array by putting the index in square brackets after the name of the array. For example, to put the value 5 in the first element of the array, we would use the following:

values[0] = 5;

If we want to multiply every element in the array by 5, we could write the following:

for (int i = 0; i < 10; i++)

{

values[i] \*= 5;

}

Why does the index start at 0 and not 1? This relates to the way items are referenced in the underlying assembly language and machine instructions. It’s interesting. Ask me if you’re interested.

## Back to the code…

We will actually be using an array of arrays – that is, an array, where each of its elements is an array. You can easily imagine this looking exactly like the table above. Each row (element of our array) is an array, indicating which segments are on or off.

Create a new sketch, and declare the 2-dimensional array like so:

int digit\_segments[10][7] = {

{1, 1, 1, 1, 1, 1, 0}, // This is digit 0, as per the table

{…}, // Digit 1

…,

{…} // Digit 9

}

For the array for each digit, fill in the patterns of 1’s and 0’s, so segment A is element 0, through to segment G as element 7.

In the setup() function, you will need to configure your IO pins (from your wiring table in step 2) as digital outputs.

Now we will write a function that will display whichever digit we tell it to.

void writeDigit(int digit)

{

int startPin = 1;

for (int i = 0; i < 7; i++)

{

digitalWrite(startPin + i, digit\_segments[digit][i]);

}

}

This function allows us to specify where our outputs start (startPin – pin 1). We iterate through each segment and write its output to the correct pin.

digitalWrite(startPin + i, digit\_segments[digit][i]);

This uses startPin as a base, and then adds an index, so if we start at one, we will loop through and write output to pins 1-7.

The value written to the pin is from our 2-dimensional array. Our row is indexed by digit – the number we passed in that we want to write. The value we need to write to the pin is retrieved using the segment index.

Now in the loop() function, write each digit to the display in order, with a 1 second delay between each, like this:

writeDigit(0);

delay(1000);

…

Upload the sketch, and after it resets, you should see the display count up from 0 to 9.

# Project 6.1: Potentiometer to 7 Segment Display

This project will again combine past projects and allow us to display a number from 0-9 by turning the potentiometer.

Connect the potentiometer as it was in Project 3 (potentiometer output in to Analog Input 1).

Copy the sketch from Project 5 to a new sketch. Remove all of the code from the loop() function.

Add new code to the loop() function to

* Read the value from analog input 1
* Convert the value to a scale of 0-9
* Write the scaled value to the 7 segment display (using the function you wrote in Project 5)
* Wait 50ms

# Project 6.2: Keypad to 7 Segment Displays

This project will again combine past projects and allow us to display a number from 0-9 by using the keypad.