University of Hull
Department of Computer Science
C4DI
Interfacing with Arduinos

Vsn. 1.0 Rob Miles 2014

Introduction

Welcome to our Arduino hardware sessions.

Please follow the instructions carefully. If you get the wiring wrong your programs will not work and there is a good chance that you will destroy the delicate circuitry in the device that you are using. We are using RedBoard devices from SparkFun. These are interchangeable with Arduino devices, and so when you read Arduino I really mean the SparkFun RedBoard.

In this session you will learn a bit about electronics and how to control simple circuits using an Arduino device. Here are a few conventions used in the text.

This indicates a warning to be careful about this bit. If you get it wrong it might be time to buy a new device.

This indicates an activity you should perform in at this point in the text. You may be given precise instructions, or you may have to work something out for yourself.

This indicates something that you may want to think about later.

There are two parts to this work. We have to make the circuit (build the hardware) and then we have to create a program to use the devices (write the software).

Getting Started with the Arduino/RedBoard

SparkFun, who produced the RedBoard, have produced a very good set of web pages that take you through setting up your board and getting it to work.

The tutorial also tells you how to install and run the software that you will use to create programs for the

Find the tutorial and work through the first section

https://learn.sparkfun.com/tutorials/redboard-hookup-guide/introduction

At the end of this section you should have managed to make the LED on the RedBoard blink. Note that other Arduino boards have a light in this position too, although it might not be the same shade of purple.

Creating a simple circuit

The led connected to pin 13 on the RedBoard is wired directly to the pin on the device circuit board. Now we are going to discover how this wiring works by connecting our own LED to another digital pin.

We are going to start by creating a simple circuit to light up an LED. The circuit is as follows:

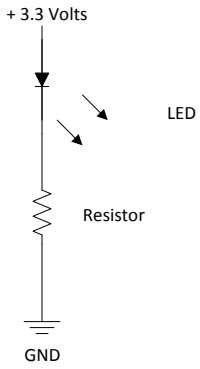

Figure 1: A very simple circuit

The current flows through the LED and causes it to light up. The resistor is there to reduce the flow of current so that that the LED receives just enough to light, but not so much that the LED overheats and fails. The 3.3 volt source is from the Arduino power supply. You might expect that the voltage you are going to use is 5 volts; after all, the Arduino is plugged into the USB port on your computer, and that produces 5 volts. However, it turns out that the actual Arduino computer runs on 3.3 volts to reduce the power consumption of the chip.

Computers work by storing patterns of 0 and 1 and then "twiddling" with them. In some computers the value of 1 is represented by a 5 volt signal, but in the Arduino this voltage is reduced to 3.3 volts to reduce power consumption. You might want to think about how this helps. The equation Power = Current * Voltage might be useful.

Breadboards and circuits

We are going to create our circuit using a *breadboard.* You can buy these in various shapes and sizes. They can be used to *prototype* a design and prove that it works, before you get the hardware actually manufactured. Alternatively, for very simple projects that you don't want to mass produce, you can use them to produce the finished product. After all, not many people look inside the box, right?

Breadboards are great for building simple circuits. They provide sockets that you can push wires and component leads into to link them together. Figure 2 shows the breadboard that we are going to use. The neat thing about a breadboard is that the sockets are connected underneath, so you can link things together without needing any (or at least many) wires.

Figure 2: Our empty breadboard

If you look very carefully at the breadboard in Figure 2 you will find that actually it is made up of three parts. The top and bottom rows are separate. This means that the breadboard has a central area for your components and then two connectors along the top and the bottom for delivering power and providing a ground connection.

Figure 3: Breadboard connections

Figure 3 shows how the sockets are connected together on the breadboard that we are going to use. I've only drawn in the top three rows, but the rest down the board are connected the same way. Note that there is no connection across the "river" in the middle, so we can put components across this bit and then use the sockets either side to connect to them.

Making a LED of Our Own

The first thing we are going to do does not involve any programming at all. We are simply going to make a circuit that will cause an LED to light up. We can start by considering how we are going to insert the components in the breadboard end. To do this we will need to use:

- The breadboard (obviously)
- The red led
- A 330 Ω resistor (from the helpfully labelled pack)
- A red jumper wire
- A black jumper wire

We are going to implement the circuit that lights the led. Then we are going to get the Arduino to control the led.

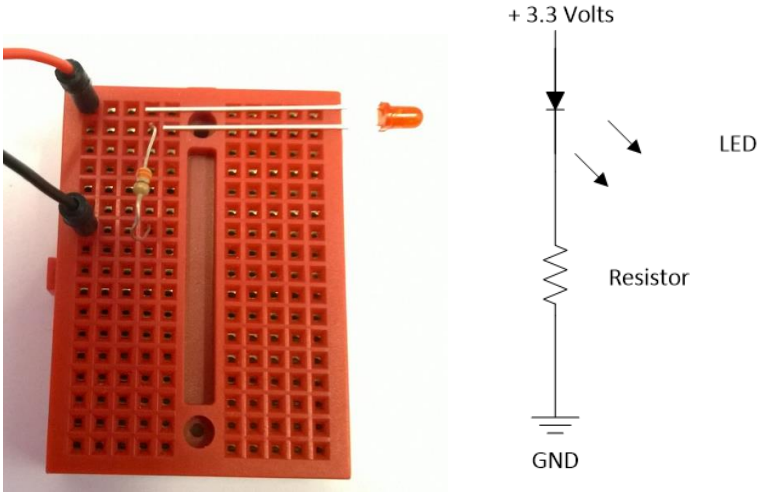

Figure 4: A circuit on a breadboard

Figure 4 shows how we take a circuit and implement it on a breadboard. The resistor makes the connection between the power socket and the LED bridges the gap between the two vertical rows. I've laid the LED on its side so you can see which socket the long side goes into.

Create a circuit using the arrangement show above. Make sure that the components all line up and push the led into the two holes and make sure that the rows with the plugs on line up correctly.

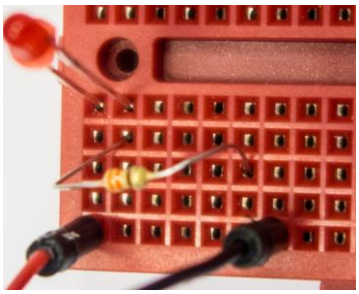

Figure 5: Something wrong here

Figure 5 shows a LED that is would not light up. Can you see what is wrong here?

Now we have connected the breadboard end we need to consider how to connect to the Arduino that is going to provide the power.

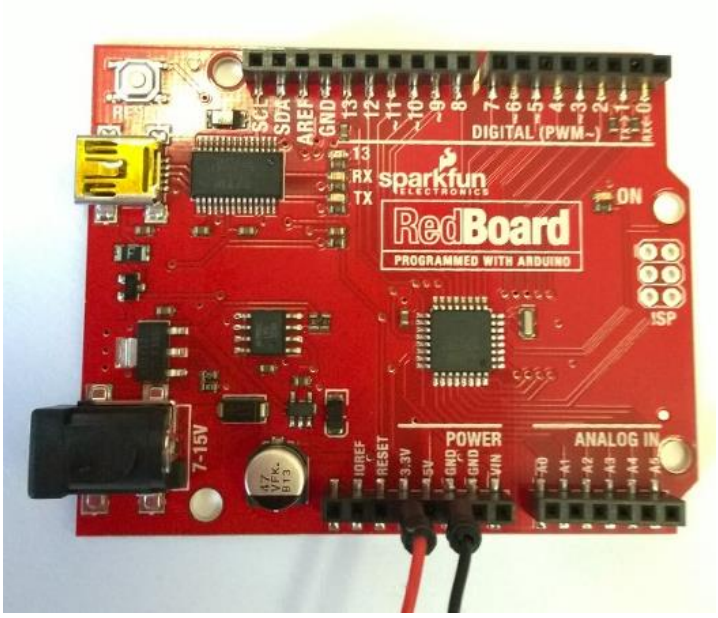

Figure 6: Connecting to the RedBoard

Figure 6 shows the connections to be used. We are going to use the 3.3 volt and the ground sockets. The pins on the RedBoard are all labelled with their function. However the labels are printed on the board and the connectors are quite long, so make sure that you line up the label correctly.

Put the red connector into the socket labelled 3.3 volts and the black connector into a socket labelled GND. There are three of these on the board, you can use any one of them you prefer.

Once you have made the connections, you can plug the red mini-usb cable into your laptop and power up the RedBoard. At this point the power sockets on the device should become live and the led should light up.

If this doesn't happen make sure that you have lined up all the connections on the breadboard correctly and that the led is plugged in the right way round.

We will use this technique to add other components to the circuit and connect the Arduino to them. Just remember how the sockets are connected. If it helps, you can think of the river that runs down

the middle of the breadboard as being at the bottom of a valley, with the rows of sockets all connected together as they run down towards it.

Controlling a LED from your program

At this point in the exercise you should have already downloaded the Blink program into your device and it should be merrily flashing your led. We are going to connect our red led to another pin on the Arduino and then change the program so that it controls that led.

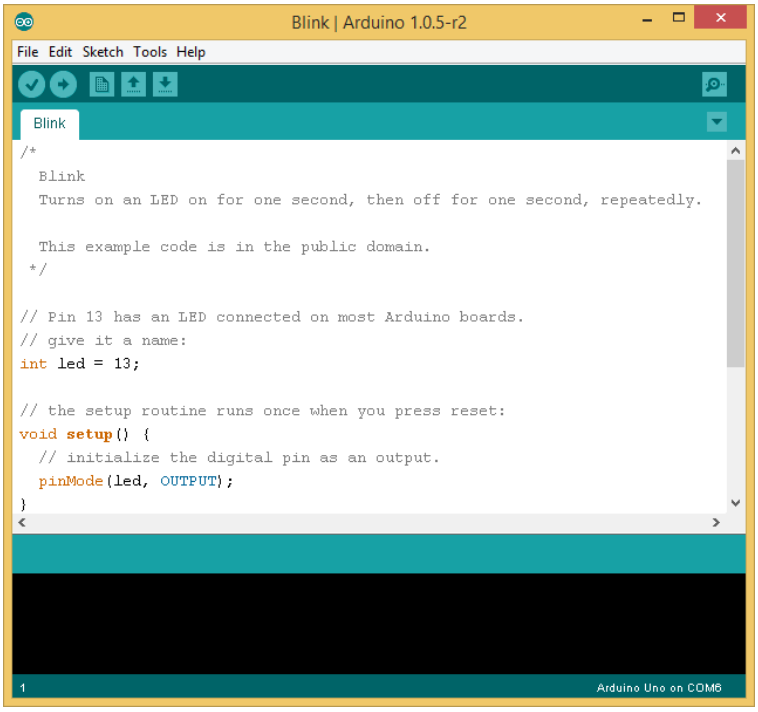

Figure 7: The Blink program

Figure 7 shows the Blink program inside the Arduino environment. You should have this up on your screen at the moment, and the Blink program should be running. Take a look at the statement:

```
int led = 13;
```

What do you think this statement does?

This is a line of C. C is a programming language. It is a way that we can tell programs what to do. There are lots of languages out there, C is one of the most powerful and popular ones. It can also be a bit tricky to work with, but we are going to just pick up the bits that we need to learn as we go along, so don't worry about this too much.

The statement creates a variable called `led`. You can think of a variable as a box with a name on it. Into this box we can put things. In the case of an `int` variable we can store integers there. The statement puts the value 13 into a box called led.

The program will look in this variable if it ever needs to know which of the digital ports is connected to the led.

What do you think would be the effect of changing the value 13 to 12?

The program would run fine. But the led on the Arduino would not flash. The program is now using digital connection 12 instead of 13.

Change the value of led to 12 and upload your program.

Note that the led on the Arduino no longer flashes.

Now we just need to move the power connection so that the led is controlled by digital 12.

Move the **red** cable connection from the 3.3 v pin across the Arduino to digital pin 12. You should see the led start to flash.

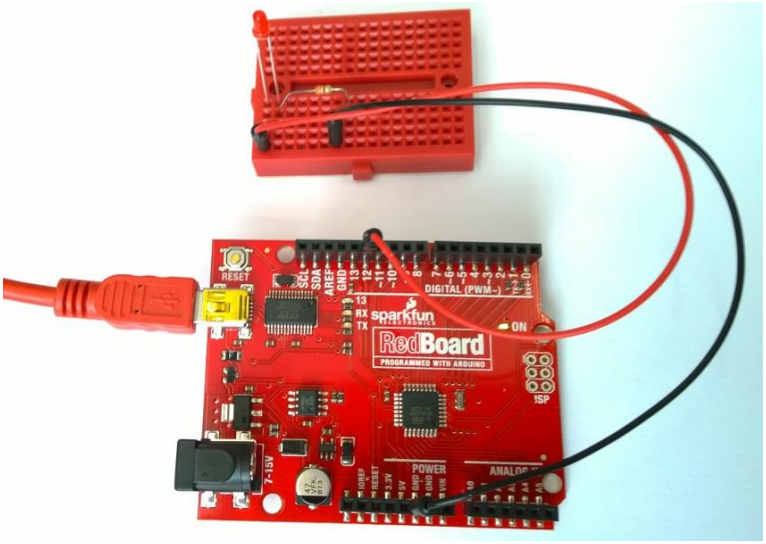

Figure 8: The complete connections

If your led isn't flashing, check your wiring against the arrangement in Figure 8.

Working with the LED Flashing program

The program that is controlling the led looks like this:

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 12;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

The top part is just a comment that tells us about the program. It says what the program does. This is followed by the creation of our led variable. Note that the comments don't agree with what the program does, we probably need to fix that.

The led setup is followed by the creation of a method called `setup`. A method is a set of C statements that have been given a name. The `setup` method is used when the hardware needs to be set up. This is just after the Arduino is switched on.

```
// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}
```

This version of the `setup` method calls another method, `pinMode`, to tell the Arduino system that the led in is going to be an output. The `pinMode` method is given two pieces of information. The number of the pin that is being controlled (in this case the value in `led`) and the mode for that pin. The value `OUTPUT` is actually built into the Arduino software. There is a corresponding value which can be used to set a pin as an input. I will leave you to work out what this is.

The second method in our program is called `loop`. This is called repeatedly when the Arduino is switched on.

```
// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

You can probably work out what the statements in the `loop` method do without much further explanation. The text at the right of each line, after the `//` characters, is comment text. The `digitalWrite` method writes a value to a particular pin and the `delay` method pauses for a particular number of thousandths of a second.

How would I change the program to make the light flash twice as fast?

By changing the delay we can change the rate at which the light flashes.

Change the delay value to 500 and upload your program.

This should make the light flash twice as quickly.

We can control multiple leds by connecting them to other pins.

Add a second led (the yellow one) and connect it to pin 11. For extra style you could change the variables so that you have a `redLed` variable with `12` in it and a `yellowLed` variable with `11` in it. Make both the leds flash together.

Remember that each led that you add will need its own 330 Ω resistor and connection to ground. Remember also that you need to make sure the led is connected the right way round, with the long lead going towards the positive signal (the output pin).

Change your program so that when one led is lit the other is off, so that the light seems to “bounce” between them.

Everything we do from now on will be a mix of wiring together a circuit on the little breadboard and then changing the software to make it drive the hardware in a particular way.

It is important that you don’t get programs and the hardware they are connected to mixed up, otherwise your program might send signals the wrong way into devices and damage them or the Arduino.

Making Sounds

We can re-purpose the blink program and make some irritating noises. If you look at the code you see that the value in the call of the delay method controls the rate at which the lights flash. By changing the delay we can change the rate of flash.

What do you think would happen if we made both of the delay values 1?

If we make the delay very short the led will still flash, but it will flash so rapidly that we won't see it change. It will just seem to be lit, but not quite as brightly as it was before. A delay of 1 millisecond for the on and the off parts of the wave will produce a signal wave that is going up and down 500 times a second. The technical term for this is a sound wave with a frequency of 500 Hertz.

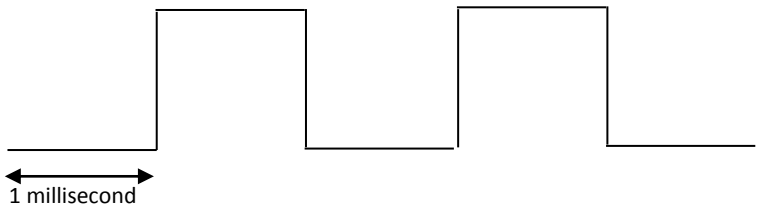

Figure 9: The output that our program produces

Figure 9 shows what the output from our program looks like. This kind of signal is often called a *square wave* because the shape of the edges of the waveforms is a square.

We can't see a signal like this, but we can hear it. If we replace the led with a sounder we can actually hear the 500 tone. The good news is that we actually have a sounder in the RedBoard kit.

Figure 10: Connecting to the sounder

The sounder is the little black round thing with CET12A3.5 written on it (no, I don't know what that means). You can connect it as shown in Figure 10 about. It works loudest without the resistor.

It is important that you plug the sounder in the right way round. Otherwise you might hear a silent alarm that means it is time to buy a new sounder and/or Arduino. Connect it with the tiny embossed + sign towards the positive connection, as shown above.

When we connect the sounder the program will send the square wave into it, which will cause it to make an irritating buzzing noise.

Remove the LEDs and resistors from the board. Put them back into their little bags for safe keeping.

Plug the sounder into the breadboard and run the program.

You can change the pitch of the note that is played by increasing the delay values.

What do you think would happen if we made both of the delay values 2?

The longer the delay, the slower the waveform and the lower the pitch of the note.

Change the delay values and notice how the pitch changes.

Getting user input from switches

At the moment we can only send outputs. We might want to write programs that get inputs too. The RedBoard kit contains a rather nice push button that we can use for input. To get the input to work we need to connect the switch to a digital pin on the Arduino.

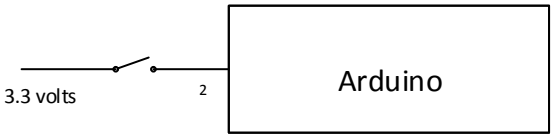

Figure 11: Badly connecting an input signal

You might think that you could just wire up the switch as you see in Figure 11 above. Although the title of the figure might make you think again. The circuit looks OK, when we press the switch it sends 3.3 volts into digital pin 2. So the signal will go high when we press the button and low when it is released.

The problem with this approach is that the input is very susceptible to stray voltages which might be induced into the wire. These might be mistaken for inputs by the software. What we need is something that will pull the signal into one state, and then when the switch is closed the signal is pulled into the

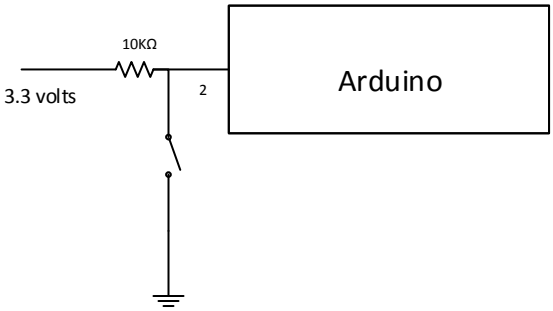

Figure 12:Properly connecting an input signal

Figure 12 is far more sensible. The power signal is connected to the input pin via a resistor which is called a “pull up resistor”. This holds the input at 3.3 volts. When the switch is closed it makes a connection to ground, causing the input to be made low. Now the input signal is always being held at a steady value and there is no chance of noisy inputs causing problems. The RedBoard kit is supplied with a number of 10KΩ resistors which are to be used as pull up resistors.

Note that this means that the input to the Arduino will be high until we press the switch. Our program will have to be written to handle this.

I’ve used pin 2 in the circuit about because there just happens to be an example program in the Arduino program examples that reads a button and uses it to control the state of the led on the Arduino board.

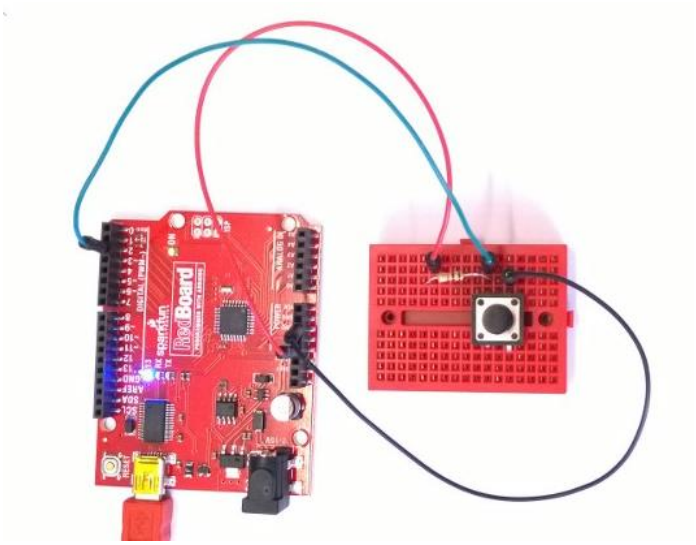

Figure 13:How to connect the switch to the Arduino

Figure 13 shows the circuit we are creating. The red wire is power, the black wire is ground and I'm using the green wire to send the signal to pin 2.

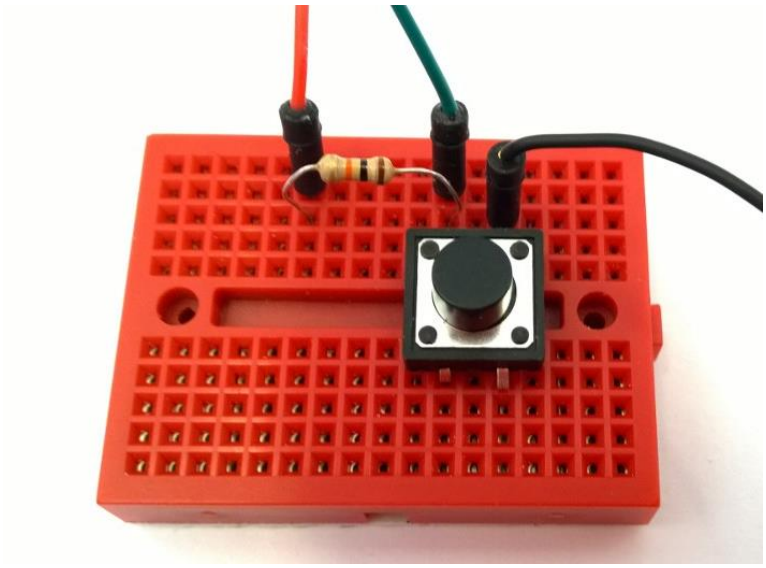

Figure 14:Breadboard details

You can see a better view of the circuit here. Note that I've put the switch across the "river" in the middle of the breadboard, but I'm using the connections on just one side. Note also that I've not done a very good job of pushing in the connector on the bottom right of the switch, make sure that when you push the switch home you get the connector all the way into the socket. If you are viewing this in black and white the wires going into the board are coloured red, green and black respectively, going from left to right. The arrangement of the components is actually a realisation of the circuit in Figure 12. There are lots of ways of doing this, I quite like mine because it works.

Put the circuit above into your breadboard and load the Button program from **Examples>02 Digital>Button**

Run the program and note that when you press the button the light goes off.

If this doesn't happen, get help.

You should now have a way of telling a program that a button has been pressed.

How would we change the program so that the light came on when the button was pressed?

Making Decisions

The Button program contains a new program construction that we might find useful.

```
buttonState = digitalRead(buttonPin);

// check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
```

The if construction has a test followed by a statement which is obeyed if the result of the test is true. It can also contain (as it does above) an else part that is obeyed if the test is not true. In the above code it turns the LED on if the button state is high. The value of the `buttonState` variable is read from the button pin by the `digitalRead` method.

If all this seems a bit hard to take in, remember what we are trying to achieve. We want the state of the led to be controlled by the state of the input button.

Note that the curly brackets – the { and } characters – are used to mark the start and end of the sequence of statements that are controlled by the condition.

The test `buttonState == High` has **two** equals symbols so that the Arduino knows that this is at test to see if one thing is the same as another. If there is only one equals character the Arduino will think that you want to put a value into the variable, which is quite a different operation. Beware of getting these two mixed up.

Challenges

Now that you can make sound and read inputs we can have some fun. You can figure out from the program how it works. Here are some things to have a go at:

On and Off

I want two lights. One should be lit when the button is pressed and the other must be lit when the button is released. When I press the button I want the lights to change state. You will need to add another led to your hardware design and decide which pin it is being connected to.

Light Delay

I want a delayed light. Press the button and the light comes on for five seconds and then goes out. Another variation of this is a light that lights one second after the button has been pressed. Take a good look at the delay method to do this one.

Sound and Light

I want the light to light and a buzzer to sound when the button is held down. You will need to change the wiring again.

Button Flash

I want the light to flash when the button is held down. Release the button and the light stops flashing.

Mega Challenge: Push On- Push Off

This is a very tricky one. I want the light to come on when the button is pressed and released. And then go off when I press and release the button again. In other words, I want the light to work like the one in your bathroom, where each time you pull the cord the light changes state.

A heavy hint here, the lamp needs to be able to detect when the button has changed state, and it also needs to know whether the light is presently lit or not. You can create your own variables to keep track of these states.

Remember that if you want to keep your Arduino programs to work on later you have to use the Save function in the editor.

You are not allowed to change the contents of the example programs that we are using as a starting point, but you can use the Save As command to save them in a different location.

Rob Miles

March 2014