University of Hull
Department of Computer Science
C4DI
Help for Bald Secret Agents

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).

The Problem

The problem we are solving this week is quite a simple one. We are working for the British Secret Service Technical Gadgets Division who need help for some of their agents. They want a reliable way of discovering if someone else has been looking at secret stuff.

The standard way of detecting whether or not a desk drawer has been opened is to use some saliva to paste a hair across the gap between the drawer and the side of the desk. If you come back to the drawer and the hair has been moved then this means that someone has been in your stuff.

Secret agents have been using this for years to detect when drawers and doors have been opened, but they are having a problem at the moment. For some reason a lot of their agents have no hair at all. Many are completely bald. So they need a device that can do this job for them.

They want a way of detecting when a drawer has been opened. They have a list of the following requirements:

- They would prefer it if there was no mechanical connection to the drawer
- The device should be portable, preferably battery powered
- If it could count how many times the drawer had been opened, this would be useful
- If it could sound an alarm when the drawer was opened, this would be useful too

In the session this week we are going to build such a device. Any ideas?

Suitable Sensors

The first thing we need to consider is what sensor we are going to use to detect when a drawer has been opened. The SparkFun kit that are using has a number of sensors that we might like to consider.

Switch Sensor

We can detect electrical signals. We could place a switch in the drawer so that when the drawer is opened the switch is triggered. The Arduino provides a digital input that could be made to change state when the drawer is opened and closed. A program in the Arduino could detect when this happens and use it to track when the drawer was opened and closed.

Flex Sensor

We would use this in a similar way to a switch. We could bend the sensor when the drawer moves and a program could detect the changes in the values it produces.

Light Sensor

We haven't used the light sensor yet. It provides a value which changes with the amount of light falling on it. We can assume that when the drawer is closed it will be dark inside, and that if drawer is opened it will let light in. We can't assume that the number in the dark will be small and the number in the light be large, but we can work on the basis that the value will change and that program could detect this.

Analogue or Digital?

We have seen that there are essentially two kinds of input signal that an Arduino can accept:

- Digital signals that are in one of two states. In a C program we use the convention that the value 0 means "false" and any other value (i.e. non-zero) means true.
- Analogue signals which are represented by a number of possible values in a particular range. The Arduino can convert an input signal into a number in the range 0 to 1023. Analogue signals are never "true" or "false", although you can use thresholds to make decisions about their state, in other words you could say "Any value more than 100 is true".

Which of the sensors above is analogue, and which is digital?

The Best Sensor to Use?

Often in programming there are lots of different ways of solving a problem. The best way to select a sensor is to make a list of the advantages and disadvantages of each one. At the end of the day the customer won't really care which one you use, as long as they get something that does what they want.

Quite often a hardware engineer will have to make a choice which balances performance and cost. They may well have to compromise on some aspects of their solution. In other words, the reason that some things don't work well in some respects is not that they were badly designed, but that the designer was forced to make some hard choices.

Which sensor would you use?

You can find my choice on the next page. See if you can predict which one I went for.

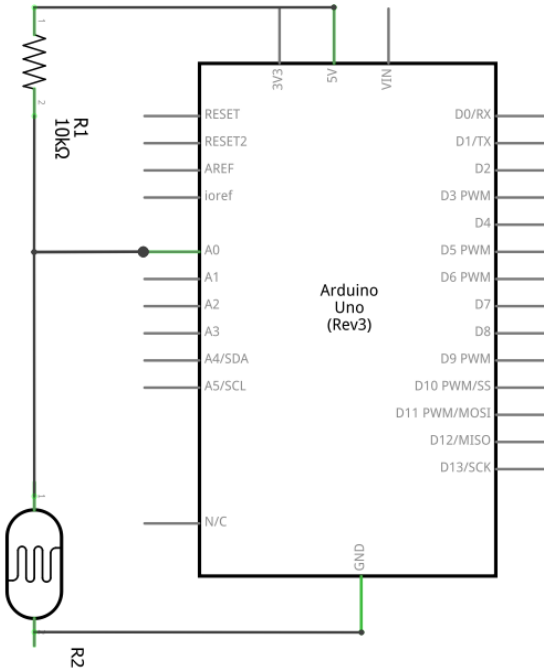
I decided to use the photocell. It is much easier to fit into the drawer than any of the mechanical switches and so our spy would find this much simpler to use. The only drawback is that if the drawer is opened in the dark our device might not detect this, but since it would not be possible to read the secret documents in the dark I reckon that this will be OK.

Reading the Photocell

The *photocell* (sometimes called a *photo detector*, or *light dependent resistor*) is a component that changes in resistance depending on how much light is falling on it. In the light it has a resistance of around 1,000 ohms. In the dark this rises to more than 10,000 ohms. The Arduino analogue inputs measure voltage, not resistance, so we need to create a circuit that will convert a resistance value into a voltage value.

Creating a "Potential Divider"

This should do the trick. At the top of the diagram you can see a 5 volt signal coming out of the Arduino. This goes through a resistor with a value of 10,000 ohms. An ohm is a measure of how hard it is for electricity to travel through a material. The higher the value, the more the material resists.



The signal then goes through the photocell and then reaches ground. The Arduino analogue input is connected to the midpoint, where the fixed resistor and our photocell are connected.

The way that electrical circuits work, the voltage across each component spreads out over the components in the ratio of their resistance.

What we have here is a *potential divider.* Potential is a posh word for voltage, and divide means that the voltage across the components is divided across each of them. The higher the resistance of a component, the greater the voltage will be across that component.

If that sounds hard to understand, consider what happens in the dark. In the dark the photocell has a resistance of 10,000 ohms (the same as the resistor). This means that half the voltage will be across the photocell and the other half will be across the resistor. This means that the A0 input should be at around 2.5 volts.

If we turn on the lights this will cause the resistance of the photocell to drop to around 1,000 ohms. The ratio of the resistances has changed from one to one (10,000 to 10,000) to around ten to one (10,000 to 1,000). This means that much more of the voltage will be across fixed resistor, causing the voltage at A0 to drop. Don't worry about this too much for now. As far as we are concerned the result of this little circuit is that a program can detect a change when we shine light on the photocell.

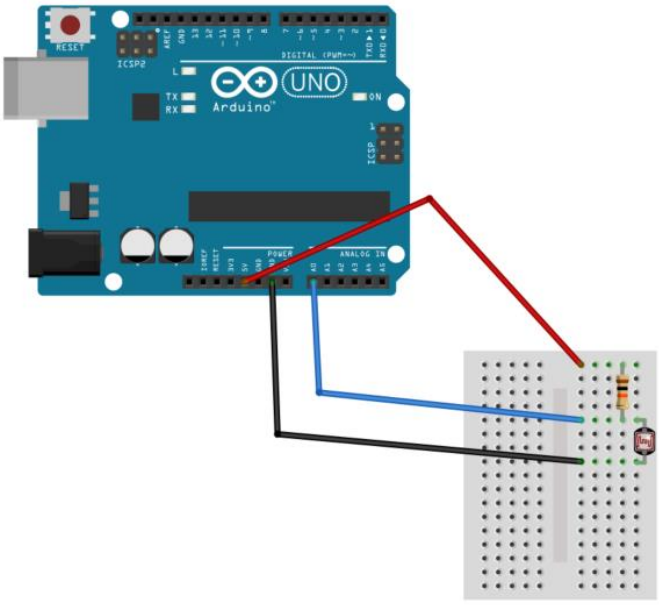
You can actually use a circuit like this to perform maths. If you think about this, if both resistors are the same value the voltage at the mid-point will be exactly half that coming into the circuit. By changing the values of the resistors we can divide by other numbers. This is the basis of *analogue* computing, which is performed by creating circuits like this. The advantage of an analogue

computer is that it performs calculations really fast (at the speed of electricity). The disadvantage is that to change what the computer does you actually have to re-wire its circuits.

As a little "thought experiment", you might want to consider how you would have to change the circuit if you wanted the voltage on A0 to *increase* when light falls on it.

Creating a Photocell Reader

We could sit down with a calculator and work out what values we could use to detect light and dark. We could read the datasheet for the photocell and do all kinds of sums to work out what numbers should appear when the device moves from light to dark. But that would be quite hard. Instead we are going to find out what the good values are by just looking at them. I reckon that this is actually the best way to design the system.



This is the potential divider circuit implemented on breadboard.

Create the circuit on your breadboard. Remember to use the 10K resistor. If you use the wrong one you won't do any damage, but the readings that you get won't be as useful.

Testing the Photocell Reader

We need to write a program to read the photocell and display the readings from it.

```
int photocellPin = 0;

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

void loop()
{
  int lightLevel = analogRead(photocellPin);
  Serial.print("Light level: ");
  Serial.println(lightLevel);
  delay(1000);
}
```

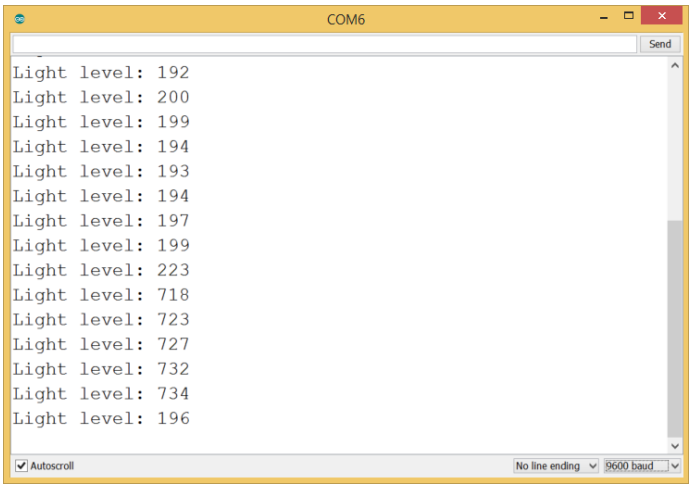
This is the program that I created to test the photocell. Remember that an Arduino program has a `setup` and a `loop` method. The `photocellPin` variable is there to make it easy to change the program to reflect changes in the hardware design. If we decide to use analogue input number 1 instead we just have to change that variable. There is another useful effect of creating a variable in this way, which is that it makes the program easier to understand. It is obvious from the call of `analogRead` that we are reading the photocell.

The `setup` method is called when the program starts running. In the program above it sets up the serial port with a particular data rate (in this case 9600 baud).

The `loop` method is called repeatedly. Each time the loop above runs it reads the light level and then prints it. We can then take a look at the values.

Start the Arduino development tool, create a brand new Arduino project and enter the above program into it.

Run the program and open up the Serial Monitor from the Tools menu. Make sure that the baud rate is set to 9600 (you can set it at the bottom right of the Serial Monitor window.



These are the readings I got when I ran the program using my sensor. The values of 700 or so were produced when I covered the sensor up with my finger.

It looks to me as if we could regard any value more than 700 as being in the dark and any value less than that as light.

We are *thresholding* the values. If you think about it, we are performing a very primitive "analogue to digital" conversion. We are taking a signal that can have a very large number of different values and then converting it into something that can be either true (above the threshold) or false (below the threshold). When the value we are seeing from the sensor crosses the threshold we want to trigger an action of some kind.

Making a `drawerClosed` function

We now know how to read the sensor, which is nice. We also know the meaning of the value that it produces. Any value larger than 700 means that the drawer is closed. The Arduino programming language lets us perform tests on values and make decisions based on these:

```
if(analogRead(photocellPin) > 700)
  Serial.println("Drawer Closed");
else
  Serial.println("Drawer Open");
```

However, it is often neater to create a *function* to deal with a particular aspect of program behaviour. This will perform the test and then just return whether the drawer is closed or not. This is way of hiding a bit of complexity from people that want to read the drawer status. Using a function can also make a program a bit smaller if we want to test the drawer status in lots of parts of the program.

It also means that we can distribute work around a team. Once we have decided how a function is going to be used we can then give it to somebody else to write for us. When we create a function we need to give it a name, decide what (if anything) it works on and what (if anything) it returns. We have already created two functions, `setup` and `loop`. Now we are going to make a third one, which returns whether or not the drawer is closed.

```
boolean isDrawerClosed()
{
  if(analogRead(photocellPin) > 700)
      return true;
  else
      return false;
}
```

This is a `drawerClosed` function. It returns `true` if the drawer is closed, and `false` if not. The bool result of the function is not a numeric value, it is a value that can hold one of two states, true or false. This type of variable can be used directly in tests:

```
if (isDrawerClosed())
  Serial.println("Drawer Closed");
else
  Serial.println("Drawer Open");
```

Add a `drawerClosed` function to your program and use the above code in the loop method to test it.

Hiding the drawer testing behaviour in a function like this has another very important benefit. Remember that we had to decide on the sensor that we were going to use to determine whether or not the drawer was closed? Putting the reading behaviour inside a function like this means that we could change to a different sensor (perhaps a switch) very easily. All we would have to do is create a new version of `drawerClosed` that works with the new sensor design and the rest of the program would function exactly as before. This is a very important principle in program construction

Detecting Changes in the Drawer position

Our program can now obtain the state of the drawer at any instant, but what it really needs to do is detect when the state of the drawer is changed. To do this the program has to *remember* the previous state of the drawer, and then detect when the new state is different. Programs remember things using variables:

```
boolean oldDrawerClosed;
```

This is a variable called `oldDrawerClosed`. This is the state of the drawer the last time we looked at it. Remember that the `loop` method is called repeatedly when the program is running, so each time round the loop we can compare the previous state with the new state and do something if we detect a change:

```
boolean newDrawerClosed = isDrawerClosed();

if(newDrawerClosed != oldDrawerClosed)
  if(newDrawerClosed)
    drawerHasBeenClosed();
```

```
    else
      drawerHasBeenOpened();

  oldDrawerClosed = newDrawerClosed;
```

This code reads the drawer closed status and then compares it with the previous one. If the drawer has changed state it then works out whether it has just been closed or opened, and then calls one of two functions to deal with the event. Note that this shows another way that we can use functions. We can break our program down into sensible chunks, each of which deals with one part of the program.

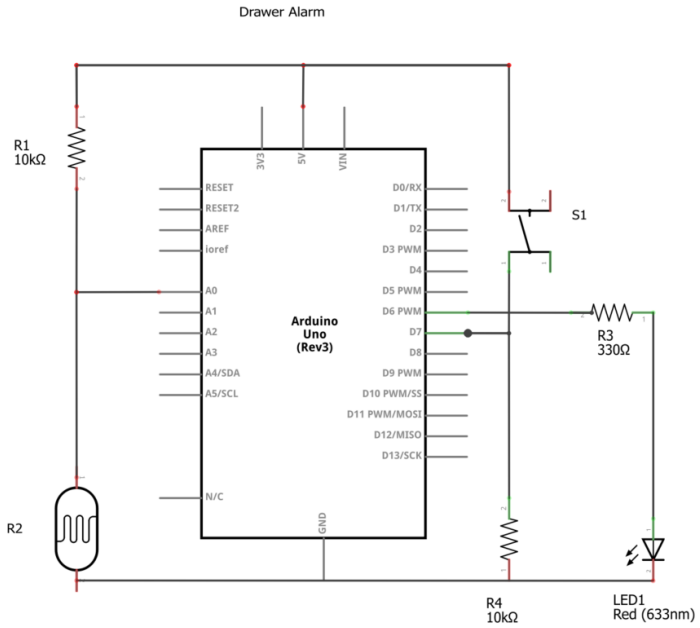
```
void drawerHasBeenClosed()
{
  Serial.println("Drawer has been closed");
}

void drawerHasBeenOpened()
{
  Serial.println("Drawer has been opened");
}
```

These are my first versions of the two drawer methods. At the moment they just print a message that I can use for testing.

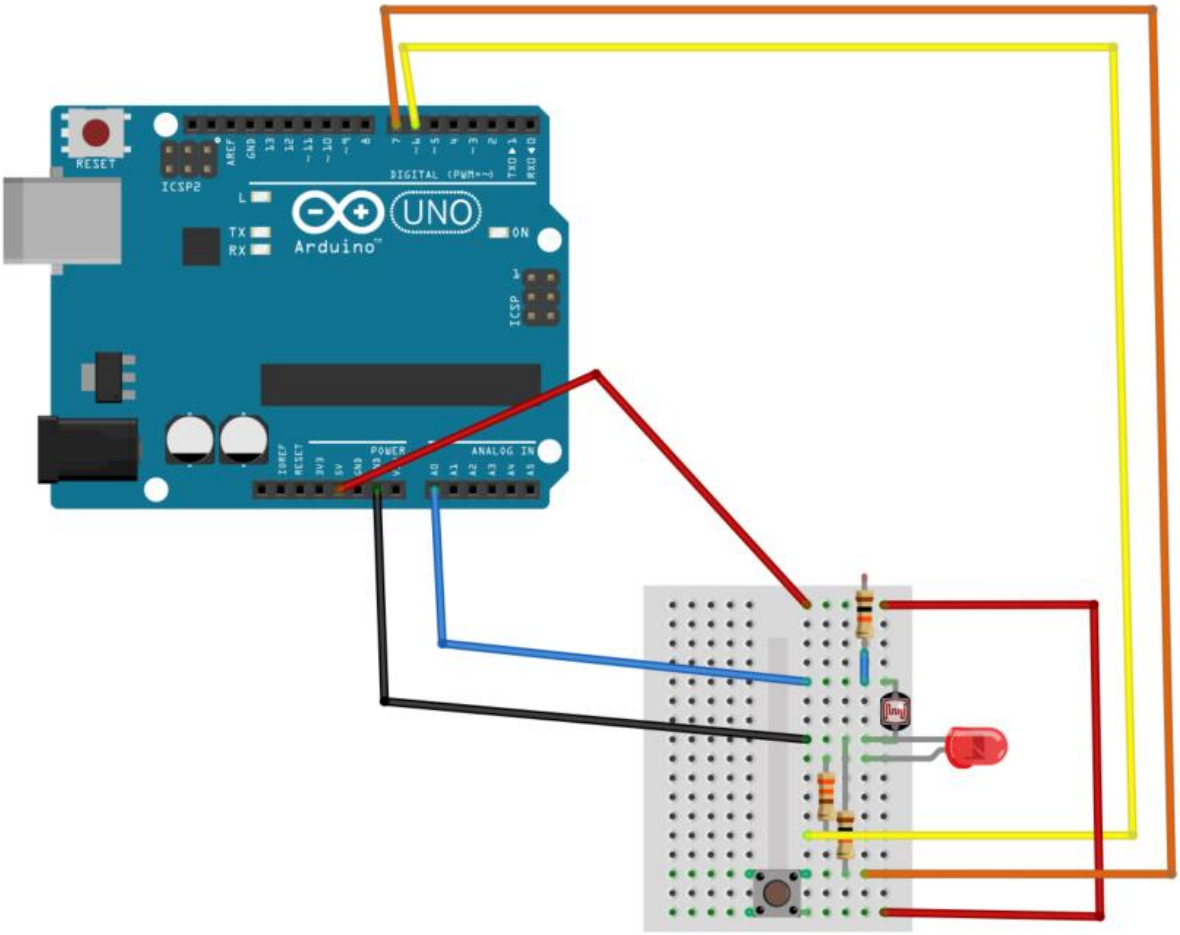
Add these functions to your program and prove that it can detect when the drawer is closed and opened.

Adding an Arm Button and an Indicator LED

The alarm needs to know when it is being armed and it also needs to have some way of displaying the output. We can use a button to arm the device and a light emitting diode (LED) to indicate if the drawer has been opened.



This is the circuit of the complete alarm. The button (S1) is connected to pin D7, which is configured as an input. The LED (LED1) is connected to pin D6 which is configured as an output.



This is the same circuit implemented on breadboard.

Wire your circuit up to match the diagrams above.

Managing the input Button

We can deal with the button in exactly the same way as we have dealt with the drawer signal:

```
boolean isButtonPressed()
{
  if(digitalRead(buttonPin))
    return true;
  else
    return false;
}
```

This is the function that will read the button. It returns `true` if the button is pressed and `false` if not. Note that I am using the same convention for naming the function (the previous one was called `isDrawerClosed`. This is good programming practice, as it makes it easier for someone reading my program to work out what is going on.

If you think about it, the program really only wants to know when the button is pressed, just like we only really wanted to know when the drawer was opened and closed. Again, we can manage this behaviour in just the same way as we created the drawer software.

```
boolean oldButtonPressed;

void updateButton()
{
  boolean newButtonPressed = isButtonPressed();

  if ( newButtonPressed != oldButtonPressed)
    if(newButtonPressed)
      buttonPressedDown();
    else
      buttonReleased();

  oldButtonPressed = newButtonPressed;
}
```

This is the code that looks after the button. If you compare it with the drawer open/close code you will find that it is exactly the same pattern, just using a different input. Note that I've made one addition though. I've put the whole behaviour into a function which updates the button. This means that I can just call this from the loop method to update the button for me. The idea behind this is that I want to create a library of useful functions that I can plug into any program that wants to use buttons and light sensors.

Making the alarm work

Our system can now detect when the drawer is opened and closed. We now need a way of making the whole device. I propose this sequence of operation:

1. Secret agent takes our device out of their secret agent briefcase.
2. Secret agent presses the "arm" button on the device. The "Opened" counter in the device is reset to 0.
3. Secret agent puts the device in the drawer and closes it.
4. The device now counts the number of times that the drawer is opened and only lights the led the second (and greater) time that the drawer is opened.
5. If the agent opens the drawer and sees a red light they know that the drawer has been opened before.

This actually boils down to a few very simple behaviours:

- If the Arduino sees the arm button pressed it sets the counter to zero.
- If the Arduino sees the "drawer opened" event it increases the counter by 1 and then turns the led on if the counter is greater than 1.
- If the Arduino sees the "drawer closed" event it turns off the led.

I've produced a "starter" program that has all the required elements. At the moment all it does is light the led when it detects that the drawer is opened and turn the led off when the button is pressed. It is up to you to add the rest. You can find the program at the end of this document.

You can add enhancements if you like. You could make the device flash the led to indicate the number of times that the drawer has been opened.

Rob Miles
May 2014

```
int photocellPin = 0;
int buttonPin = 7;
int ledPin = 6;

// LED control

void ledOn()
{
  digitalWrite(ledPin,HIGH);
}

void ledOff()
{
  digitalWrite(ledPin,LOW);
}

// Drawer

boolean isDrawerClosed()
{
  if(analogRead(photocellPin) > 700)
      return true;
  else
      return false;
}

void drawerHasBeenClosed()
{
  Serial.println("Drawer has been closed");
}

void drawerHasBeenOpened()
{
  Serial.println("Drawer has been opened");
  ledOn();
}

boolean oldDrawerClosed;

void updateDrawer()
{
  boolean newDrawerClosed = isDrawerClosed();

  if(newDrawerClosed != oldDrawerClosed)
    if(newDrawerClosed)
      drawerHasBeenClosed();
    else
      drawerHasBeenOpened();

  oldDrawerClosed = newDrawerClosed;
```

```
}

// Button

boolean isButtonPressed()
{
  if(digitalRead(buttonPin))
    return true;
  else
    return false;
}

void buttonPressedDown()
{
    Serial.println("Button has been pressed");
    ledOff();
}

void buttonReleased()
{
    Serial.println("Button has been released");
}

boolean oldButtonPressed;

void updateButton()
{
  boolean newButtonPressed = isButtonPressed();

  if ( newButtonPressed != oldButtonPressed)
    if(newButtonPressed)
      buttonPressedDown();
    else
      buttonReleased();

  oldButtonPressed = newButtonPressed;
}
void setup()
{
  Serial.begin(9600);
  pinMode(buttonPin,INPUT);
  pinMode(ledPin,OUTPUT);
}

void loop()
{
  updateButton();
  updateDrawer();
  delay(10);
}
```