University of Hull
Department of Computer Science
C4DI
Sound Effects

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

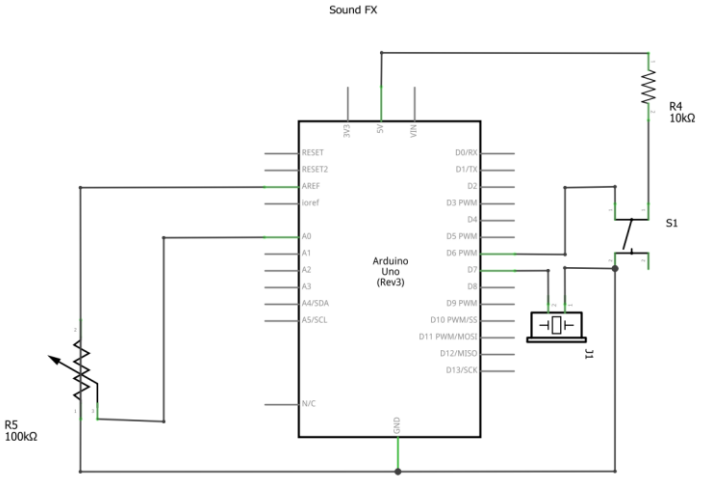
The problem we are solving this week is an interesting one. We have been employed by the director of a new cult Sci-Fi film called “Alien Space Invaders Who Come from Space”. She wants us to make sound effects for the numerous forms of death ray that are going to be used in the film. She wants sounds for the following:

- Intergalactic hyperdrive
- Raygun zap
- Anti-planetary gravity disrupter wave gun

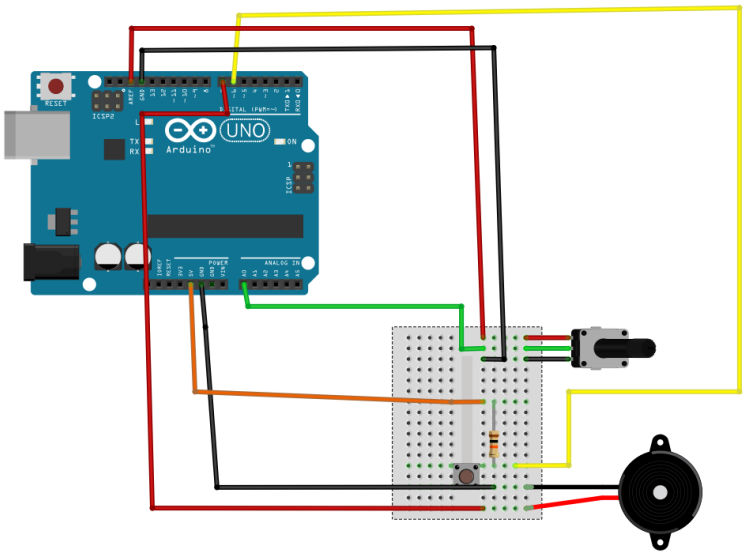
She is particularly interested in sounds that can be changed while they are played. For example she wants to be able to change the sound of the hyperdrive from “idling” to “full warp speed” when adding the soundtrack. And she also wants to be able to trigger a sound just by pressing a button. For some reason she is very keen on the sounds made by the sounder in the SparkFun kit. Which is just as well.

Arduino Hardware

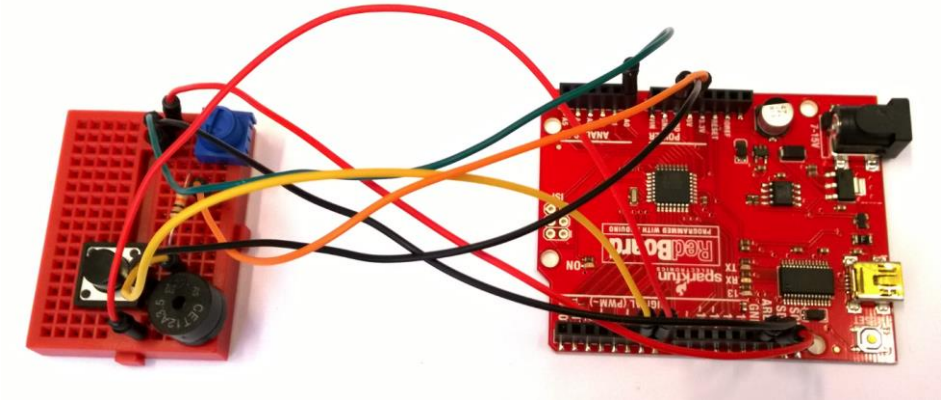
We can use a digital port from the Arduino to directly drive the sounder. We also need an analogue input that the program can read to allow the director to change the sound and we also need a button that we can press to trigger sound playback.



This is the circuit diagram that should do all this. The potentiometer is connected to A0, the sounder is connected to D7 and the button is connected to D6.



This shows how the components are connected. Note that the rotary control in the Sparkfun kit doesn't look like the one above, it is a small blue device with a knob on the top which you will fit directly into the circuit board. Also the sounder fits directly too.



This is the same circuit and the Arduino.

	Create the circuit on your breadboard.

Arduino Software

The Ardunio library provides a function called tone which will play a tone for a particular length of time. The program selects the pin to use, the frequency of the tone and the number of milliseconds duration. We can use this to produce sounds:

```
int dialPin = 0;
int speakerPin = 7;
int buttonPin = 6;

void setup()
{
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);
}

void loop()
{
  int val = analogRead(dialPin);
  if(!digitalRead(buttonPin))
  {
    int freq = val;
    int length = 30;
    int gap = 20;
    tone(speakerPin, freq, length);
    delay(gap);
  }
}
```

The program above sets up the input and output devices and then repeatedly reads the analogue input to get a value that is then used to control the frequency of the note that is played by the tone command. Note that the tone is only played when the button is pressed.

Enter the program and run it. Save it in a sketch called "HyperDrive". You should hear sounds when you press the button. If you adjust the rotary control you should be able to change the frequency of the sound.

When the control is all the way over to the one side there is no sound at all. The director doesn't like this. It is because the analogue input returns 0 in this situation and a sound wave with a frequency of 0 is not really a sound at all.

How do you fix this?

A program can look a variable and decide to do something if the value of variable satisfies a particular condition. This what the if construction is used for. Perhaps we could change the program so that the `freq` value is never allowed to go below 20.

We want the program to do something if the value of the freq is less than 20. What would the test look like?

In a program this is called *clamping*. We clamp the value of a variable so that it can't go outside a particular range. It is exactly the same construction that would stop you moving your bat off the screen in a game of video tennis. If the player tries to move the bat too far the program will "clamp" the value at the edge of the screen.

```
if(freq < 20) freq = 20;
```

This is the statement that you need to add to your program. If the frequency value drops below 20 it will be clamped to 20.

Add the above test to your program. You will need to figure out where it needs to be placed. A hint: it should be just after the program acquires the value of `freq`, but just before it is used in the `tone` method.

Actually, if you read the Arduino documentation you find that you shouldn't really make sounds less than 31 Hz, so you might want to change the software to meet this requirement.

Making a Raygun

The director is quite happy with the buzzing sound, and reckons it will make a fine Intergalactic Hyperdrive, but now she wants something for the raygun sound. This should make a single zap sound when the button is pressed. To do this the program must detect when the button is pressed, that's when the sound should be produced. Essentially we need to detect when the state of the button input changes.

```
int dialPin = 0;
int speakerPin = 7;
int buttonPin = 6;
boolean oldButtonState;

void setup()
{
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);
  oldButtonState = digitalRead(buttonPin);
}

void loop()
{
  boolean newButtonState = digitalRead(buttonPin);
  if(oldButtonState != newButtonState)
  {
    // Button has changed state
    if (!newButtonState)
    {
      // Button has just been pressed
      int freq = 1000;
      int length = 100;
      tone(speakerPin, freq,length);
      delay(length+1);
    }
  }
  oldButtonState = newButtonState;
}
```

The program above does this. It uses a variable called `oldButtonState` to record the state of the button from last time the loop method was called (remember that loop is called repeatedly when your program runs. When the old and the new state are different we know that we might be firing the raygun.

Start a new Arduino sketch, called Raygun, and put the code in it. If you run the program it should beep each time that the button is pressed.

Making more Interesting Sounds

The director will not be impressed by a simple beep. What she wants is some kind of "whoosh", the sound starts with a low frequency and then changes rapidly to a high pitched one as the gun charges and then fires. To achieve this we need to consider how the tone function is used:

```
int freq = 1000;
int length = 100;
tone(speakerPin, freq,length);
delay(length+1);
```

The first parameter is the `speakerPin`, this identifies the output to be produced. The second is the frequency. The third is the length.

You might be wondering about the reason for the call of the `delay` method after we have started the tone playing. This is because the `tone` method is rather clever. It plays the sound in the background, while our program continues running. We are going to make a sequence of tones when create our raygun sound, and we need to make sure that each tone has finished before we start playing the next one. We add one millisecond to this delay to make sure that the tone really has finished before we play the next one.

Note that this is a lovely example of a "kludge", in that from a programming point of view we shouldn't need this extra millisecond, but from a "making it work properly" perspective we definitely do.

If you are not convinced, try removing the delay and see what happens.

We can make a more interesting sound by playing a sequence of tones of different frequencies:

```
if (!newButtonState)
{
  // Button has just been pressed
  int freq;
  int length = 30;
  for ( freq=100; freq < 2000; freq=freq+200)
  {
    tone(speakerPin, freq,length);
    delay(length+1);
  }
}
```

This code replaces the simple tone generation with a for loop that takes the frequency from 100 to 2000 in steps of 200.

Add the tone generation code above and note how it makes a more interesting whoosh.

This is a reasonable whoosh, but it is not quite what the director wants. She'd like the whoosh to go a bit quicker and maybe start from a lower frequency.

Change the code to make a more interesting whoosh. You could even add a second for loop after the first one which make the sound go back down again. There might be prize for the best sounding raygun….

Adding Randomness

You can make sounds much more interesting by adding a little randomness to them. The Arduino library has a random function that will return a random number in a given range.

```
int dialPin = 0;
int speakerPin = 7;
int buttonPin = 6;
boolean oldButtonState;

void setup()
{
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);
  oldButtonState = digitalRead(buttonPin);
}

void loop()
{
  boolean newButtonState = digitalRead(buttonPin);
  if(oldButtonState != newButtonState)
  {
    // Button has changed state

    // Update the old button state
    oldButtonState = newButtonState;

    // See if the button is now down
    if (!newButtonState)
    {
      // Button has just been pressed
      int freq;
      int length = 30;
      int i;
      for ( i=0; i < 50; i=i+1)
      {
        freq = random(1000,2000);
        length = random(30,40);
        tone(speakerPin, freq,length);
        delay(length+1);
      }
    }
  }
}
```

This program uses the random function to play sounds of different frequencies and lengths.

Start a new Arduino sketch called “Disrupter” and put the above program in it. You can vary the high and the low frequencies to get different sound effects.

You play the above program to the director and she quite likes it, but she would like to hear a kind of low pitched buzz after each sound. See if you can do this. You could also increase the length of the sound and even start to use the analogue control to change the sound as it is playing

Rob Miles 2014