8BitBox

Created by Mark Rudolph

https://learn.adafruit.com/8bitbox

Last updated on 2023-08-29 02:28:33 PM EDT
# Table of Contents

## Overview

## Parts Needed

### Housing

### Arduino Shield

- Wiring:
- Installing the pieces:

### Arduino Code

- Defining some Variables:
- Defining some Functions:
- Setting an LED:
- Making a blocking read:
- The essential behavior of the sketch:

### Android Code

- Take a step back
- What "interesting interactions" will our app perform?
- What commands will we be sending?
- Start Button
- Select Button
- Slider
- A and B Buttons

## Putting it all Together
Overview

The 8BitBox is a portable Arduino powered Bluetooth box, controlled via a Bluetooth enabled Android Device. This project has an RGB LED and a piezo buzzer to play music that you will be able to control remotely.

Once you have finished this project, you will have a better understanding of how to remotely control an Arduino project via Bluetooth, and have a pretty awesome nightlight.

Parts Needed

**Tools**

- Scissors
- Wire cutters
- Wire strippers
- Soldering iron
- Needle nosed pliers
- Drill
- 5/8” spade (or drill bit)
Housing

- 3D printed Mario Question Box
- Orange latching On/Off switch (http://adafru.it/1444)
- Quick connect (http://adafru.it/1152) button wire pair
- Vellum paper
- Aluminum foil

Electronics

- Arduino Uno
- Arduino proto shield
- 2500 mAh 3.7V LiPo battery (http://adafru.it/328)
- LiPo charging circuit (http://adafru.it/259)
- Adafruit EZ Link
- RGB LED (http://adafru.it/159)
- 220 Ohm resistors (x3)
- Piezo buzzer

Miscellaneous Parts

- Wires for soldering connections
- Solder
- Velcro squares / strips (3 sets approximately 1"x1"
- Scotch tape

Housing

Our enclosure will be a 3D printed Mario box from thingiverse: http://www.thingiverse.com/thing:37155. This box is the perfect size for the project as-is, and contains a printable lid to go with it. It is a remix of this thing. The original is smaller, but the author provides a tool to customize the box to whatever 8-bit design you can think of if you want to customize your project enclosure!

Step 1: Print a box and a lid!
Using the "standard medium" settings in MakerWare, the box takes about 6.5 hours to print, and the lid will take about 1.5 hours.

Step 2: Assemble the parts

The vellum paper is just used to diffuse the RGB LED in the box, so any translucent white paper should give good results. The aluminum foil will be used to help improve the lighting from the RGB LED.

You will need to cut the vellum paper into 4 3.5"x3.5" squares.

Step 3: Modify the housing
Using a pair or wire cutters, clip out all of the thin-walled grid to make a fully open question mark.

Drill a hole in the center of the lid; this is where we will install the On/Off switch. The On/Off switch here is 16 mm, so a 5/8" spade to drill through.

Step 4: Install housing accessories
Cut the vellum paper into squares, and tape them to the insides of the box.

Install the button through the hole.
Make a cut-out of aluminum foil to go around the top of the lid, and tape it in place. This is to reflect some of the LEDs light back, and provide better illumination for the box.

Step 5: Check it out!

Put the box and lid together, and check out your awesome work!

**Arduino Shield**

There are 4 main things we need to install to the proto-shield

1. RGB Led
2. Piezo buzzer
3. EZ-Link
4. LiPo charging circuit (with power switch pins)

WARNING: There will need to be a little voiding of the warranty to fit all of the electronics together. Depending on the exact shield/layout you use, you may/will need to remove the ICSP pins of your Arduino Uno! In this tutorial/configuration, we will need to do this!

Wiring:

The LEDs pins will be connected to the PWM pins 9 (RED), 10 (GREEN), 11 (BLUE). The LED used is common anode, and the corresponding Arduino code accounts for this. If you use a different style RGB LED, be sure to update the Arduino code accordingly!

The piezo buzzer will be connected to PWM pin 3.

The EZ-Link gets connected as in the EZ-Link guide, which can be found at https://learn.adafruit.com/introducing-bluefruit-ez-link

This project is intended to be battery powered. The EZ-Link is powered by a battery that is attached to the charging circuit; the EZLink then provides regulated power to Arduino. There will also be an On/Off switch between EZ-Link and Charging circuit to cut the power when you are not using your 8BitBox.
Installing the pieces:

The LED should be installed in the middle of the shield, and left as high, so it can shine light over the most area possible. The lower it is installed, then the more light the shield itself will block.

We hook up the common anode to the +5V rail, and break out the R/G/B pins. We will install 220 Ohm resistors in a later step, once we get the bulky components in place.
We will be hooking up the piezo buzzer to PWM pin 3, so we position it close to that end of the board, placing a pin into the ground rail. (It is only shown so high in the picture to illustrate where the pins are being placed).

We connect our 220 Ohm resistors from PWM pins 9/10/11 to the R/G/B leads of the RGB LED.

We jumper a wire from the PWM pin 3 to the piezo buzzer.
We install a header to attach the EZ-Link to our board, so we don't have to solder it directly to the shield. We hook up the pins exactly exactly as described in https://learn.adafruit.com/introducing-bluefruit-ez-link

NOTE: The Bluetooth Module of the EZ-Link will be facing outwards! That is to say that the pin closest to the Digital IO breakouts is the GROUND pin, and the pin closest to the Analog IN / Power breakouts is the RESET pin. This is mainly done to have the LEDs of the EZ-Link facing outwards, as they are fairly bright!

We also install a jumper pin (near the piezo buzzer, as shown) to easily attach our On/Off switch to.
We will velcro the charging circuit to the bottom of the shield. This will save us some space, but we here we will need to clip the ICSP pins to have everything fit!

If you do not wish to remove your Arduino's ICSP pins, then you can have this attached with some length of wire, and perhaps velcro it to the project enclosure instead.

Again, we are using the EZ-Link to provide power to everything, so the On/Off power is connected to the power of the EZ-Link! We will attach this via the battery connector on the back of the EZ-Link.
The battery gets connected to charging circuit.

We attach a "battery connector" wire to our shield (shown in the next picture), to connect directly to the EZ-Link.

We attach the wires from the "Load" breakout on the charging circuit to our header breakout for the power switch: The - wire goes directly the the ground wire for the EZ-Link and the + goes to the On/Off switch pins.

Now that all of the parts have been assembled, velcro them together into their new package form factor, hook up the switch and and test it out to make sure everything is getting power.

---

Arduino Code

All of the code needed to run this project is located at [https://bitbucket.org/alterationx10/8bitbox-adafruit](https://bitbucket.org/alterationx10/8bitbox-adafruit)

The Arduino sketch is located (from the source root directory) at

/Arduino/_8BitBox_Adafruit/_8BitBox_Adafruit.ino

The code is fairly commented, and is best explained while looking right at it.

Some concepts are highlighted/discussed below, but you should check out the source for all the details.
Defining some Variables:

Here are the variables defined at the top of the sketch, and used throughout the code.

```cpp
// Define our pins
int buzzerPin = 3;  // Piezo buzzer is hooked to pwm 3
int redPin = 9;     // Red LED hooked to pwm 9
int greenPin = 10;  // Green LED hooked to pwm 10
int bluePin = 11;   // Blue LED hooked to pwm 11
int commandByte;   // A variable for command parsing

// We will store our favorite color in the EEPROM
// NOTE:
// From http://playground.arduino.cc/Code/EEPROM-Flash :
// "The ATmega328 has 32 KiB of Flash program memory.
// This memory supports at least 10,000 writes or upload cycles."
int FAV_RED = 0;
int FAV_GREEN = 1;
int FAV_BLUE = 2;
```

Defining some Functions:

Setting an LED:

Here is a snippet of a function used to set the value of the Red LED (similar functions are made for the G and B). Note that we use a common anode RGB LED, so we use `analogWrite(pin, 255 - colorValue)`!

```cpp
// Set the PWM of the Red LED
void setRed(int val) {
  // Sanitize the value
  if (val &gt; 255) {
    val = 255;
  }
  if (val &lt; 0) {
    val = 0;
  }
  analogWrite(redPin, 255 - val);
}
```

Making a blocking read:

Serial.read() will perform faster than we can send data to the Arduino, so we will need to know "how to speak" to the device using our own protocol. When we receive a known command of our protocol, we know there could be more data coming, so we need to do a blocking read to make sure we get it.

```cpp
// Serial.read() appears to go much faster than we can send text.
// Therefore, we add a method to do a blocking read.
```
int getNextByte() {
    while (Serial.available() == 0) {
        // BLOCK
    }
    return Serial.read();
}

The essential behavior of the sketch:

The `loop()` function of our sketch just checks to see if serial data is available, and if so, parse the command and perform an action.

Below is a snippet to show how it would parse for a command to set the Red LED to a certain value, using the two functions shown above.

In this example, we will send an "R" from the Android device as our protocol command to set the Red LED to a value, along with the value (0-255) to set it to.

If we receive an "R", then we know that there is one more byte on it's way as a value to set the LED to (0-255), so we then perform a blocking read to get it, and then use our function to set the value to the Red LED.

```c
void loop() {
    // All our loop currently does is check for serial data
    // and parse serial commands
    if (Serial.available() > 0) {
        // Read our command byte
        commandByte = Serial.read();
        // See if it matches any of our commands
        // Set the Red LED
        if (commandByte == 'R') {
            int redVal = getNextByte();
            setRed(redVal);
        }
        // Parse for other commands in full sketch...
    }
}
```

Android Code

All of the code needed to run this project is located at [https://bitbucket.org/alterationx10/8bitbox-adafruit](https://bitbucket.org/alterationx10/8bitbox-adafruit)

NOTE: The Android code is targeted to a minimum of Android JellyBean (API 16)! You will have to do a bit of work to make is work with older Android devices (but it is
Take a step back
Let's take a moment to think about what we've done so far.

Hardware design: It might not be too advanced, but we laid out some components on a shield. It may only be an RGB LED and a buzzer, but that has implications for our firmware.

Firmware design: Based on our hardware design, we have laid out our firmware to connect to our components. We know we are going to be talking to a Bluetooth Android device, so we have even implemented our own communication protocol.

Software design: We are now at the point where we can make software to talk to the firmware on the hardware we made! How exciting! But this can be very frustrating too. You writing software (an App) with other peoples software (Android) on other peoples hardware (your phone/tablet) to communicate with the firmware you wrote for your own hardware. What happens when something doesn't work? Is it your software? Is it your firmware? Is it your hardware? Or is it theirs? It's probably a vicious mixture of all of the above :-) 

But, we should take a deep breath a step gently though each step, thinking about how each part interacts with each other to isolate and fix any issues that arise.

A full discussion of Android (and some of the problems you might run in to) is outside the scope of this tutorial. Much of the code is commented, or can be further commented if you ask, and the best understanding would come from looking right at the code and tweaking it yourself.

Below, we will think about the "interesting interactions" between your app and your 8BitBox, how the software is going to do it, and how you are going to get the user to tell the software to do it. Since it's an 8BitBox, and 8Bit style controller would make interesting interaction. Our app will be themed as such.

What "interesting interactions" will our app perform?

1. Connect/Disconnect to the 8BitBox
2. Send a command to the 8BitBox
3. Wait for the 8BitBox to respond
What commands will we be sending?

1. Setting an R/G/B color
2. Playing a note on the buzzer
3. Storing a favorite color in EEPROM

With all this in mind, let's look at our simple UI, and figure out which UI elements we are going to assign to which actions.
Let's lay out all the actions we want to perform, and what elements will trigger/show it

**Element - Function**

- Action Bar - Show status'
- Slider (TOP) - Adjust Red LED
- Slider (Middle) - Adjust Green LED
- Slider (Bottom) - Adjust Blue LED
- Start - Connect
- Select - Disconnect
- B - Play Song 1
- A - Play Song 2
- (Secret) Up, Down, Left, Right, B, A, Start - Store a favorite color to EEPROM if currently connected to the 8BitBox

Now that we have an overview of what the app does, let's discuss a few of the of them in a little more detail.

**Start Button**

This is what happens when we press to connect to the 8BitBox. We connect to the 8BitBox based on it's MAC address, and provide a custom CallBack to perform events based on the result. We try to connect using Serial Port Protocol, and if successful, we get the socket input output streams. Then it's just like usual RX/TX on the Arduino!

This is very much like Serial.begin() in an Arduino sketch, and once we are connected, we just start sending bytes of data back and forth.

```java
// An Interface will allow the user to run their own code when
// a certain even happens. This is one for Bluetooth Conenctions.
public interface BluetoothConnectCallback {
  public void doOnConnect();
  public void doOnConnectionFailed();
  public void doOnDisconnect();
}

public boolean btConnect(String MAC, BluetoothConnectCallback myCallback) {
  // Just in case BT wasn't enabled when we started the app
  if (btAdapater == null) {
    btAdapater = BluetoothAdapter.getDefaultAdapter();
  }

  // Are we already connected?
  if (btSocket != null && btSocket.isConnected()) {
    return true;
  }

  // MAC needs to be upper case. Help the user out.
  MAC = MAC.toUpperCase();

  // Get our device
  btDevice = btAdapater.getRemoteDevice(MAC);

  // Connect to the device
  btDevice.connect(myCallback);

  // Wait for connection
  while (btDevice.isConnected() == false) {
    try {
      Thread.sleep(200);
    } catch (InterruptedException ex) {
      ex.printStackTrace();
    }
  }

  // Successful connection, register the incoming data stream
  Serial.print("Connection successful!

device connected: 
name: 
MAC: 
IP: 
Connecting to Arduino...

// Start sending data
// Connect to the 8BitBox...
```

©Adafruit Industries
// UUID for Serial Port Protocol
UUID mUUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

// Try to connect
try {
    btSocket = btDevice.createInsecureRfcommSocketToServiceRecord(mUUID);
} catch (IOException e) {
    Log.d(TAG,"Failed to open the bluetooth socket...");
    // Trigger the fail method that the user has implemented
    myCallback.doOnConnectionFailed();
    return false;
}

// Connect ot the socket
try {
    btSocket.connect();
    iStream = btSocket.getInputStream();
    oStream = btSocket.getOutputStream();
    // Trigger the success event that the user has implemented
    myCallback.doOnConnect();
} catch (IOException e) {
    Log.d(TAG, "Failed to connect to the bluetooth socket, or open the I/O streams...");
    // Trigger the fail method
    myCallback.doOnConnectionFailed();
    return false;
}

// If we've made it this far, everything is good to go!
// This is for a job queue to process commands in a
// First-In-First-Out manner.
commService = Executors.newSingleThreadExecutor();
return true;

Select Button

When the select button is pressed, we want to disconnect from the device. This is the type of code you would run for that event

```java
public boolean btDisconnect(BluetoothConnectCallback myCallback) {
    // Don't bother to disconnect if we're not already connected
    if (btSocket != null && btSocket.isConnected()) {
        try {
            // Try and close down the I/O streams and close the socket.
            iStream.close();
            oStream.close();
            btSocket.close();
            if (commService != null && !commService.isShutdown()) {
                commService.shutdown();
            }
        } catch (IOException e) {
            Log.d(TAG,"Had trouble properly closing down the I/O and Bluetooth socket...");
        }
    } else {
        // Return false if we weren't connected
        return false;
    }

    // Run the user specified code
    return true;
}
```
Slider

We will use the slider (SeekBar) to adjust our RGB values. By setting the maximum possible value to 255, every time it is changed, we can just send a command with the progress value, and adjust the LED.

// Get the reference to the SeekBar from the layout XML file
SeekBar redSeeker = findViewById(R.id.seekRed);
    // This is used to set an LED, so we only want it to go between
    // 0 and 255
    redSeeker.setMax(255);
    redSeeker.setProgress(0);
    redSeeker.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
        @Override
        public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
            // When the SeekBar is changed, we get the 0-255 progress value,
            // and send it over the serial port.
            // BoxConstants.RED[0] is "R", the command of our protocol to
            // Adjust the Red LED.
            // This means that setLevel looks something like
            // [0x52, 0xff] if the slider was to the max
            byte[] setLevel = {BoxConstants.RED[0], (byte) progress};
            myService.writeData(setLevel);

            // This method just sets the color of the picture on the screen
            // to be the same as the 8BitBox
            setBitBoxColor();
        }
        @Override
        public void onStartTrackingTouch(SeekBar seekBar) {
        }
        @Override
        public void onStopTrackingTouch(SeekBar seekBar) {
        }
    });

A and B Buttons

These buttons simply send a preconfigured array of bytes to play a song over the piezo buzzer. The interesting thing here, is that music presents a challenge due to the timing. It's only a quarter note if it's played for a quarter of the time...

This is where our protocol comes in handy! We don't expect a response when we change the LED, because the timing isn't so important, but for this it is. For the case of beeping out some music, we don't want to send the next note until the last note has played. Playing a note is started with the command "Z", and on the Arduino side, once
we are finished playing a note, we will return a "z" to indicate that we're done and ready for the next note.

```java
// For the length of the array of all the notes to play (underworld.length)
for (int i=0; i < underworld.length; i++) {
    // To play a musical note, we use command “Z” (BoxConstants.MUSIC[0])
    // Once the note plays, it will return “z”,
    // and we will know that we can play the next note

    // Our (musical) note is composed of “Z”,
    // a 16 bit integer value (as two bytes)
    // and a duration to play the note for
    byte[] note =
        {BoxConstants.MUSIC[0],
         Tone.toneMSB(underworld[i]),
         Tone.toneLSB(underworld[i]),
         (byte) underworld_tempo[i]};

    // We send the command data
    myBox.writeData(note);

    // For everything but the last note
    if (i != underworld.length -1) {
        while(myBox.rawRead() != BoxConstants.MUSIC_RESPONSE[0]) { // Block until we get feedback that we’re ready for the next note
            }
    }
}
```

---

**Putting it all Together**

We have all the parts sandwiched together, and the power is working!
We velcro the electronics to the middle of the bottom of the box (we don't want it to slide around).

We have the EZ-Link facing on side of the box, which we will call the "status panel side". Since the EZ-Link has LEDs on it, it will be always lit when the power is applied.

Put the lid on, and turn on the power!

Start programming your Arduino wirelessly, and interacting with it via your Android device!

If you haven't already, get the source code and start modifying it!
https://bitbucket.org/alterationx10/8bitbox-adafruit ()