Table of Contents

Overview

Blaster Circuit Prototyping

- Planning the Circuit
- Primary Functions
- Real-World Details
- Function Prototyping
- NeoPixel Tests
- NeoPixel with Toggle Switch Test
- Music Maker MP3 Tests
- Music Maker with Toggle Switch Test
- AudioFX Trigger Test
- Next Steps
Overview

Learn how to build your own Overwatch prop blaster with this multi-part guide! This 3D printed prop gun will include lights, sound effects, and background music, all played through a 20 watt amp and four speakers.

In part 1 we'll design the electronic functions and overall circuit design, and then build the individual functions as working prototypes.

In later parts in this series we'll combine the parts into a single system, planning how it will all fit by using a cardboard prototype, and ultimately create a 3D printed prop gun.
Blaster Circuit Prototyping
Planning the Circuit

I began by thinking about what I wanted the Blaster to be able to do. In researching how the gun works in the video game (a.k.a., playing video games for work) I decided on the must-have, primary functions, as well as the secondary, real-world considerations of how to activate these functions, provide power, adjust volume, and so on.

Primary Functions

The three primary functions of the Lucio Blaster are:

- Light up colorful lights
- Play background music
- Trigger sound effects

With those in mind I decided to approach them with an Arduino board of some kind to drive NeoPixels for the lights, a Music Maker MP3 shield on the Arduino board to play background music (BGM), and an Audio FX Sound Board to trigger the sound effects. (I wanted the BGM to continue playing during sound effects, so I couldn't easily use the Music Maker shield for both music and sound effects.)

Real-World Details

Since this prop gun will exist in the real world, here are the details that I needed to consider for both hardware and software:

- Tie together light and BGM modes
- Make lights react to shooting sounds
- Pull both left- and right-mouse button shots
- Activate "special" attack sound
- Engage reload sound effect
- Battery power and charging
- System ON/OFF
- System volume

This is a schematic drawing of the functions I created just to sketch out the parts I planned to use.
I sometimes like to add a very rough idea of how parts will connect. I tape a sheet of overhead transparency film to the page and use a wet erase marker to create these lines.

While no substitute for a real circuit diagram (which I'll create in Fritzing as I go) this is helpful for me to think about how the final system will work, which buttons and switches I'll need, and so on.
Function Prototyping

I built the circuit prototype in stages so that I could get a feel for each discreet subsystem before tying them all together. This is the fun, fairly simple part -- as you'll see, I only ran into difficulties once I started combining them. More about that later. Here are the individual functions as tested:

1. Drive NeoPixels from Arduino board (in this case, Adafruit METRO 328, which functions the same as the Arduino UNO)
2. Switch color modes of NeoPixels using toggle switch
3. Play background music with Music Maker MP3 shield
4. Switch background music with toggle switch
5. Trigger blaster sound effects on AudioFX shield with buttons and tilt switch

I like to test these early prototypes on breadboards with jumper wires so it's easy to change things around, rather than start soldering things too soon.

It's important to stay organized! I got my daughter to label these bins as I picked parts for testing.
Here's my initial NeoPixel test. No need to immediately jump into driving dozens of LEDs, just start out with a single strip or ring. This is simply a Metro board connected to a NeoPixel ring via pin 2 and GND, a 5V wall power supply running to Metro VIN and the NeoPixel ring VIN. I'm using a 1000uF electrolytic capacitor across the VIN and GND rails to smooth out any power spikes from frying the NeoPixels.

Once connected, I uploaded the "simple" sample sketch from the NeoPixel library examples (first adjusting the code to use pin 2, number of pixels to 12, and type to GBRW as seen below) to make sure it was all working.

```
/********************************************************
Blaster Circuit Prototype A01
```
NeoPixel with Toggle Switch Test

My second test was to add a toggle switch to trigger two different color modes on the NeoPixels. Add a switch to the circuit that connects pin 5 to ground when closed, then upload the following code.
/***********************************************
Blaster Circuit Prototype A02
Use Arduino UNO or Adafruit Metro
  - Lights a NeoPixel Ring on pin 2
  - Color controlled by switch on pin 5

written by John Park for Adafruit Industries
NeoPixel code based on library "simple" sketch by Shae Erisson
released under the GPLv3 license to match the rest of the AdaFruit NeoPixel library
***********************************************

#include <Adafruit_NeoPixel.h>
define NEOPIN 2
define NUMPIXELS 12
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUMPIXELS, NEOPIN, NEO_GRBW + NEO_KHZ800);
define MODESWITCHPIN 5

void setup() {
pixels.begin();
pixels.setBrightness(24);
pixels.show();
pinMode(MODESWITCHPIN, INPUT_PULLUP);
}

void loop() {
  // read switch value
  int modeSwitchVal = digitalRead(MODESWITCHPIN);
  if(modeSwitchVal == HIGH) {// green
    for(int i=0;i<NUMPIXELS;i++){
      pixels.setPixelColor(i, pixels.Color(0,150,0,5));
pixels.show();
delay(delayval);
    }
  } else {// yellow
    for(int i=NUMPIXELS-1;i>=0;i--){
Pixels.setPixelColor(i, pixels.Color(150,120,0,0));
pixels.show();
}
delay(delayval);
}
}

The next thing to get up and running was the MP3 playback for background music. I assembled the Music Maker shield, affixed it to a Metro board, and wired it to power, a 20W amplifier and a speaker.

This crude test code isn't very efficient, constantly testing the switch pin. We'll use a better edge detection technique later.

Music Maker MP3 Tests
I put a couple of .mp3 song files on the SD card in the shield and uploaded the sample sketch to the Metro. It plays the two songs!

```cpp
#include <SPI.h>
#include <Adafruit_VS1053.h>
#include <SD.h>

#define BREAKOUT_RESET  9      // VS1053 reset pin (output)
#define BREAKOUT_CS     10     // VS1053 chip select pin (output)
#define BREAKOUT_DCS    8      // VS1053 Data/command select pin (output)
#define SHIELD_RESET  -1      // VS1053 reset pin (unused!)
#define SHIELD_CS     7      // VS1053 chip select pin (output)
#define SHIELD_DCS    6      // VS1053 Data/command select pin (output)
#define CARDCS 4     // Card chip select pin
#define DREQ 3       // VS1053 Data request, ideally an Interrupt pin

Adafruit_VS1053_FilePlayer musicPlayer =
  Adafruit_VS1053_FilePlayer(SHIELD_RESET, SHIELD_CS, SHIELD_DCS, DREQ, CARDCS);

void setup() {
  Serial.begin(9600);
  Serial.println("Adafruit VS1053 Simple Test");

  if (!musicPlayer.begin()) {
    Serial.println(F("Couldn't find VS1053, do you have the right pins defined?"));
    while (1);
  }
  Serial.println(F("VS1053 found"));
  SD.begin(CARDCS);  // initialise the SD card

  // Set volume for left, right channels. lower numbers == louder volume!
  musicPlayer.setVolume(20, 20);

  // Timer interrupts are not suggested, better to use DREQ interrupt!
  //musicPlayer.useInterrupt(VS1053_FILEPLAYER_TIMER0_INT); // timer int

  // If DREQ is on an interrupt pin (on uno, #2 or #3) we can do background
  // audio playing
  musicPlayer.useInterrupt(VS1053_FILEPLAYER_PIN_INT); // DREQ int

  // Play one file, don't return until complete
  Serial.println(F("Playing track s.mp3"));
  musicPlayer.playFullFile("s.mp3"); //substitute the name of an .mp3 file you have
  saved to SD card

  // Play another file in the background, REQUIRES interrupts!
  Serial.println(F("Playing track h.mp3"));
  musicPlayer.startPlayingFile("h.mp3"); //substitute the name of an .mp3 file you
  have saved to SD card
}
void loop() {
  // File is playing in the background (only after first song is finished and
  // second has begun)
  if (musicPlayer.stopped()) {
    Serial.println("Done playing music");
    while (1);
  }
  if (Serial.available()) {
    char c = Serial.read();
    // if we get an 's' on the serial console, stop!
    if (c == 's') {
      musicPlayer.stopPlaying();
    }
    // if we get an 'p' on the serial console, pause/unpause!
    if (c == 'p') {
      if (! musicPlayer.paused()) {
        Serial.println("Paused");
        musicPlayer.pausePlaying(true);
      } else {
        Serial.println("Resumed");
        musicPlayer.pausePlaying(false);
      }
    }
  }
  delay(100);
}

Music Maker with Toggle Switch Test

Building on to the previous test, I added a switch on pin 5 using an internal pullup
resistor to try switching between songs. This requires the use of interrupt pins, since
the system could become too busy playing music files to pay attention to your
switches if you aren't careful. See the code below, which is based upon the VS1053
library's player_interrupt example sketch.
/********************************************************
Blaster Circuit Prototype B02
Use Arduino UNO or Adafruit Metro with Music Maker MP3 Shield
- interrupt based .mp3 song file playback test
- Song choice based on switch on pin 5

based on playerInterrupt sketch
Written by Limor Fried/Ladyada for Adafruit Industries.
BSD license, all text above must be included in any redistribution
********************************************************/

// include SPI, MP3 and SD libraries
#include &lt;SPI.h&gt;
#include &lt;Adafruit_VS1053.h&gt;
#include &lt;SD.h&gt;

#define MODESWITCHPIN 5
int modeSwitchPushCounter = 0;   // counter for the number of button presses
bool modeSwitchState = 0;         // current state of the button
int lastModeButtonState = 0;     // previous state of the button

// These are the pins used for the breakout example
#define BREAKOUT_RESET  9      // VS1053 reset pin (output)
#define BREAKOUT_CS     10     // VS1053 chip select pin (output)
#define BREAKOUT_DCS    8      // VS1053 Data/command select pin (output)
// These are the pins used for the music maker shield
#define SHIELD_RESET  -1      // VS1053 reset pin (unused!)
#define SHIELD_CS     7      // VS1053 chip select pin (output)
#define SHIELD_DCS    6      // VS1053 Data/command select pin (output)

// These are common pins between breakout and shield
#define CARDCS 4     // Card chip select pin
// DREQ should be an Int pin, see http://arduino.cc/en/Reference/attachInterrupt
#define DREQ 3       // VS1053 Data request, ideally an Interrupt pin

Adafruit_VS1053_FilePlayer musicPlayer =
    // create shield-example object!
    Adafruit_VS1053_FilePlayer(SHIELD_RESET, SHIELD_CS, SHIELD_DCS, DREQ, CARDCS);

const char* SONGS[] = {"h.mp3", "s.mp3"};
int songPick = 0;
const int VOL = 80; // volume higher numbers are quieter
void setup() {
  pinMode(MODESWITCHPIN, INPUT_PULLUP);
  Serial.begin(9600);
  Serial.println("Adafruit VS1053 Simple Test");
  if (!musicPlayer.begin()) { // initialise the music player
    Serial.println(F("Couldn't find VS1053, do you have the right pins defined?"));
    while (1);
  }
  Serial.println(F("VS1053 found"));
  musicPlayer.sineTest(0x44, 200);// Make a tone to indicate VS1053 is working
  if (!SD.begin(CARDCS)) {
    Serial.println(F("SD failed, or not present"));
    while (1);  // don't do anything more
  }
  Serial.println(F("SD OK!"));
  // Set volume for left, right channels. lower numbers == louder volume!
  musicPlayer.setVolume(20, 20);
  // This option uses a pin interrupt. No timers required! But DREQ
  // must be on an interrupt pin. For Uno/Duemilanove/Diecimilla
  // that's Digital #2 or #3
  // See http://arduino.cc/en/Reference/attachInterrupt for other pins
  // *** This method is preferred
  if (!musicPlayer.useInterrupt(VS1053_FILEPLAYER_PIN_INT))
    Serial.println(F("DREQ pin is not an interrupt pin"));
}

void loop() {
  // Start playing a file, then we can do stuff while waiting for it to finish
  if (!musicPlayer.startPlayingFile(SONGS[songPick])) {
    Serial.println(F("Could not open file"));
    while (1);
  }
  while (musicPlayer.playingMusic) {
    // file is now playing in the 'background' so now's a good time
    // to do something else like handling LEDs or buttons :
    modeSwitchState = digitalRead(MODESWITCHPIN); //read switch input pin
    if (modeSwitchState != lastModeButtonState) { // compare the modeSwitchState
      // to its previous state
      if (modeSwitchState == HIGH) {
        // if the current state is HIGH then the button
        // went from left to right :
        modeSwitchPushCounter++;
        songPick=0;
        musicPlayer.stopPlaying(); //does this so it'll restart the player with
        // the new song choice
      }
      else {
        songPick=1;
        musicPlayer.stopPlaying(); //restart the player with the new song choice
      }
      delay(50);
    }
    // save the current state as the last state,
    // for next time through the loop
  lastModeButtonState = modeSwitchState;
  }

The third of the main functions to test was blaster sound effects. I wanted to initiate two of the shooting sounds by pulling the primary and secondary triggers (just like LMB and RMB in the game), a reload sounds by tilting the gun upwards, and the special, wide radius blast attack sound by pointing the gun at the ground. Rather than get tricky using an accelerometer or even more advanced IMU, I went for tried and true tilt ball switches.

I uploaded the sounds (named T01.wav for trigger pin 1 and so on, no code required as this board is set up for very simple operation) to the AudioFX Sound Board and then wired the buttons between ground and their respective pins on the board. I connected the output of the board to the amplifier and tested out the sweet blaster sounds.
Next Steps

In the next part of this guide we'll combine these subsystems to create the full circuit for the Lucio Blaster effects, lights, and music, as well as take a closer look at powering the system. It's gonna get a bit messier before it gets neater!

I'll also be working on the design of the 3D printed blaster parts in parallel. Here's a miniature, low resolution, single-piece print I made to start planning the parts layout and design.
Stay tuned for more!