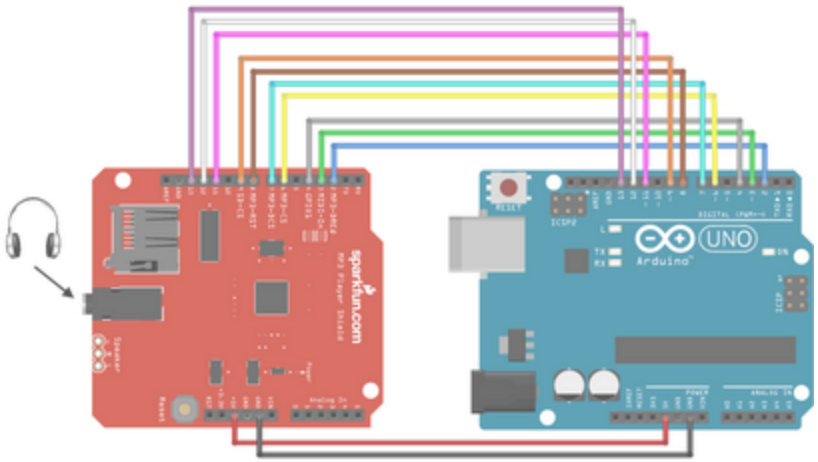
# Using the VS1053B Spectrum Analyzer and ADmonEQ Applications with Arduino




## 7 March 2017

**Author: Brandon Metcalf**

**UCI Henry Samueli**

**School of Engineering**

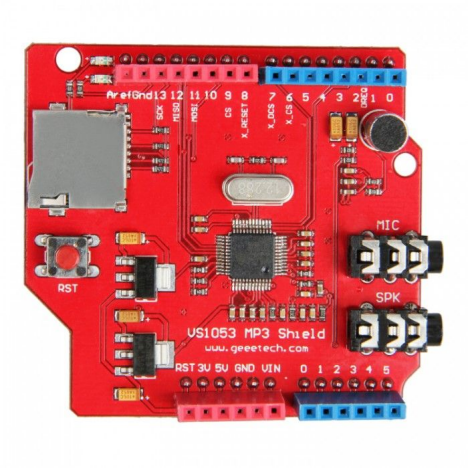
# TABLE OF CONTENTS

# 1 VS1053B Overview

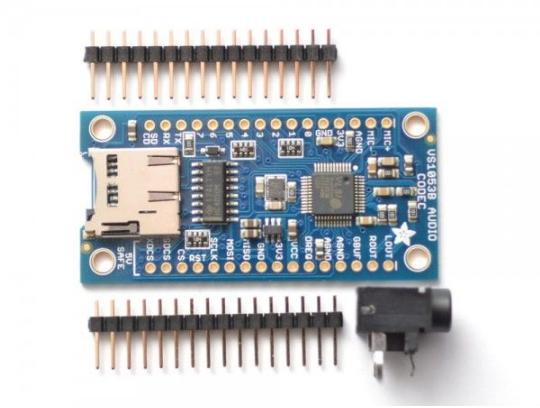
## 1.1 Introduction

The VS1053B is a DSP and audio processing chip, designed by VLSI. The chip can decode (play) .Mp3, .Ogg, MPEG4, .ACC, .WMA, and .WMV (and flac with the right patches) file types and can encode in .Ogg. The board comes in two major board designs (although there are others as well).

**VS1053B Sheild**




**VS1053B Breakout Board**



VSLI has designed uploadable “patches” and “plugins” to be uploaded to the VS1053B and utilized by another microprocessor. Note that a “patch” normally refers to an application that enhances or fixes already available functionality of the board, and a “plugin” refers to a program application (such as the equalizer or file player applications). This document attempts to fill the gap in information on how to utilize the Spectrum Analyzer plugin functionality for an Arduino Uno board for both board designs.

VSLI has provided documentation on how to utilize a plugin or patch. It is suggested that you become familiar with the spectrum analyzer and ADmonEQ plugin documents, as well as the basics of the VS1053B datasheet before attempting to implement an application. VLSI’s documentation assumes that the reader will be connecting with another processor from scratch. Not to worry, most of the work has already been done for you.

Sparkfun has created a VS1053B-friendly library, compatible with the Shield design. Adafruit has created a VS1053B breakout board library. Note that both libraries are well designed but are

SPECIFICALLY designed for only several plugins and may need modification for use in a more general sense.

### 1.2 Introduction to Spectrum Analyzer

The spectrum analyzer is a plugin designed to analyze the amplitude of sound produced a across a frequency spectrum for either a mic input or line-in (played file). Interestingly, the spectrum analyzer is a somewhat "passive application". Most major applications on VSLI's website are large and take up a large amount of memory on the registers. These applications require activation, whereas the spectrum analyzer does not. We suspect this has to do with the size of the source code used to produce the plugin and the small amount of data produced by the application in the chip's registers.

The spectrum analyzer plugin code itself, as well as its documentation can be downloaded from VSLI's website:

**Link:** http://www.vlsi.fi/en/support/software/vs10xxplugins.html

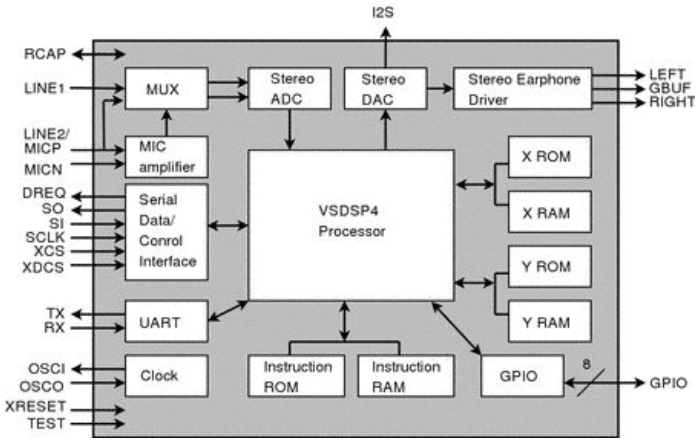
**Note:** There are many plugin files and applications provided in the spectrum analyzer zip file. The two we will be looking at are: "spectrumAnalyzerAppl1053b.plg" and "spectrumAnalyzerAppl1053b-2.plg". The spectrum analyzer document refers to the first as the "default" plugin, whereas appl-2 is the analyzer with flac decoding (it can play flac files). Both the default and the 2nd plugin work using any of the methods described in this document. Other than the 2nd plugins ability to read flac files, there is little difference between the two besides compatibility issues.

The spectrum analyzer plugin writes its results entirely to XRAM. This is the register that saves the frequency spectrum analysis and the register that we will be accessing.

### 1.3 Introduction to ADmonEQ

The admoneq plugin is an adaptation of the admixer plugin. It allows for playing input sound (through the mic) on top of already playing songs (thru line in). This plugin is extremely advantageous to us because it is compatible with the spectrum analyzer plugin. We will be using the admoneq plugin to activate the mic for spectrum analysis.

Unlike the spectrum analyzer, this plugin DOES require activation. We write a value to the AIADDR register address to activate a plugin. The specific value to write will be explored later in this document.




## 2 Pin Layout and Setup

### 2.1 SPI Brief

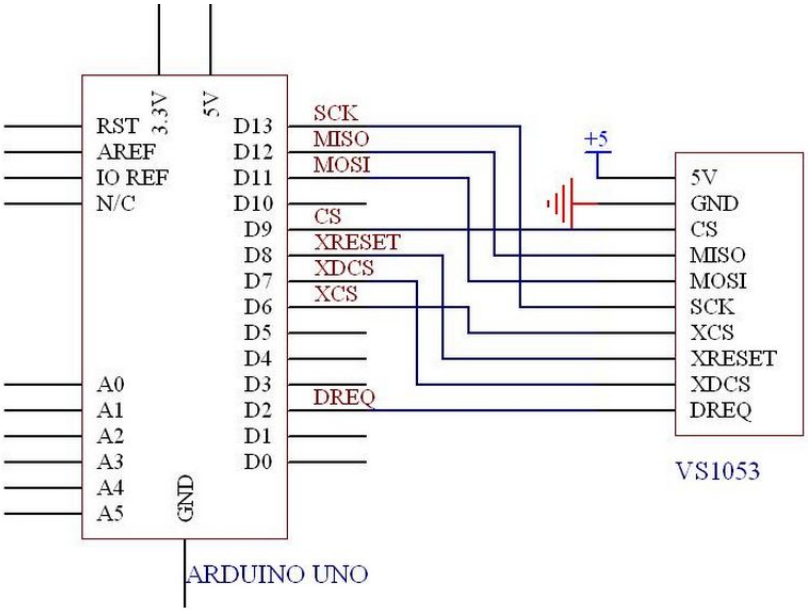
We found that the most convenient approach to connecting the Arduino and VS1053B is through Serial Peripheral Interface (SPI). Both the Sparkfun and Adafruit library solutions use SPI and allow for easily integrating the VS1053B into larger projects.

SPI is a communication method determined by clock cycles. There are typically 5 pins that are always associated with SPI: MISO, MOSI, SCLK, CS, RST. These pins determine the flow direction and timing of data between processors.

MISO: Master-In Slave-Out
MOSI: Master-Out Slave-In
SCLK: Clock
CS: Chip Select (DREQ in our case)
RST: Board Reset (XRESET)

Since the VS1053B also features an SD card, we must also connect XSC and XDSC to the Arduino.

Obviously, when you have attached the shield, we do not need to worry about connections, however, for connecting the breakout board, use the schematic below:




For a more in-depth tutorial on connecting the Arduino and breakout board, refer to the Adafruit "file player" tutorial:

**link:**
https://cdn-learn.adafruit.com/downloads/pdf/adafruit-vs1053-mp3-aac-ogg-midi-wav-play-and-record-codec-tutorial.pdf

## 3 Coding Overview

There are two plugin and patch loading methods. Uploading from the SD card, or uploading via "old user loading tables" (sending through SPI as an array of instructions). Both libraries are more compatible with the SD card loading method, however, since we are able to implement either, we will discuss both.

### 3.1 Using Sparkfun's Library

Sparkfun designed a library compatible with .053 plugin and patch file types. So before diving into the code for this application, you will need to convert the .plg files provided by VSLI on their website.

Use the following instructions to convert the provided .plg file to a .053 file:

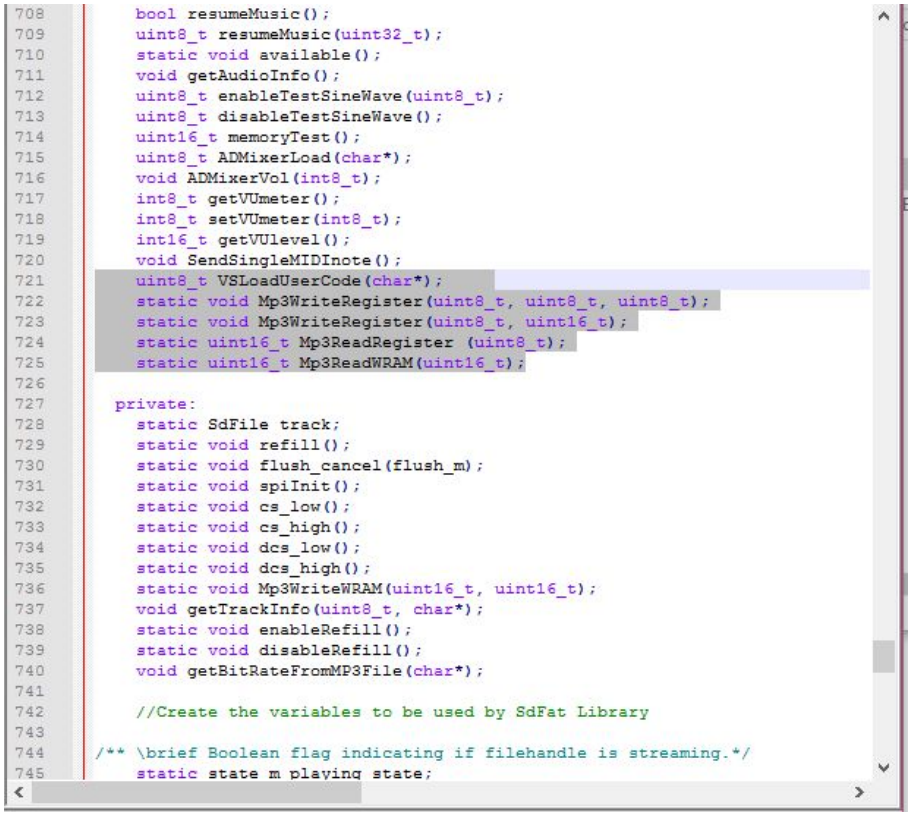
```
This script uses PERL to convert a VS1053 .PLG file
into a .053 RLE compressed file for the VS1053b.

1) Load ActivePerl-5.24 (or newer)
2) place file in a dedicated directory along with the perl conversion script
3) Open Administration level command prompt and run the following script
3) perl vs_plg_to_bin.pl C:\A\spectrumAnalyzerAppl1053b.plg  C:\A\spctan.053
```

Next download Sparkfun's shield library for the Arduino and open up your Arduino IDE to start coding.

**Link:** https://learn.sparkfun.com/tutorials/mp3-player-shield-hookup

Before using this library (with our code or with your own), there are a few immediate problems with the Sparkfun library code. In order for us to upload our own plugin, we need to use the read and write register functions. However, in Sparkfun's SFEMP3Shield.h file, they have declared their read and write functions as a private member of mp3player. We need to make those functions public, so that we have access to them. Move them to public like seen below.



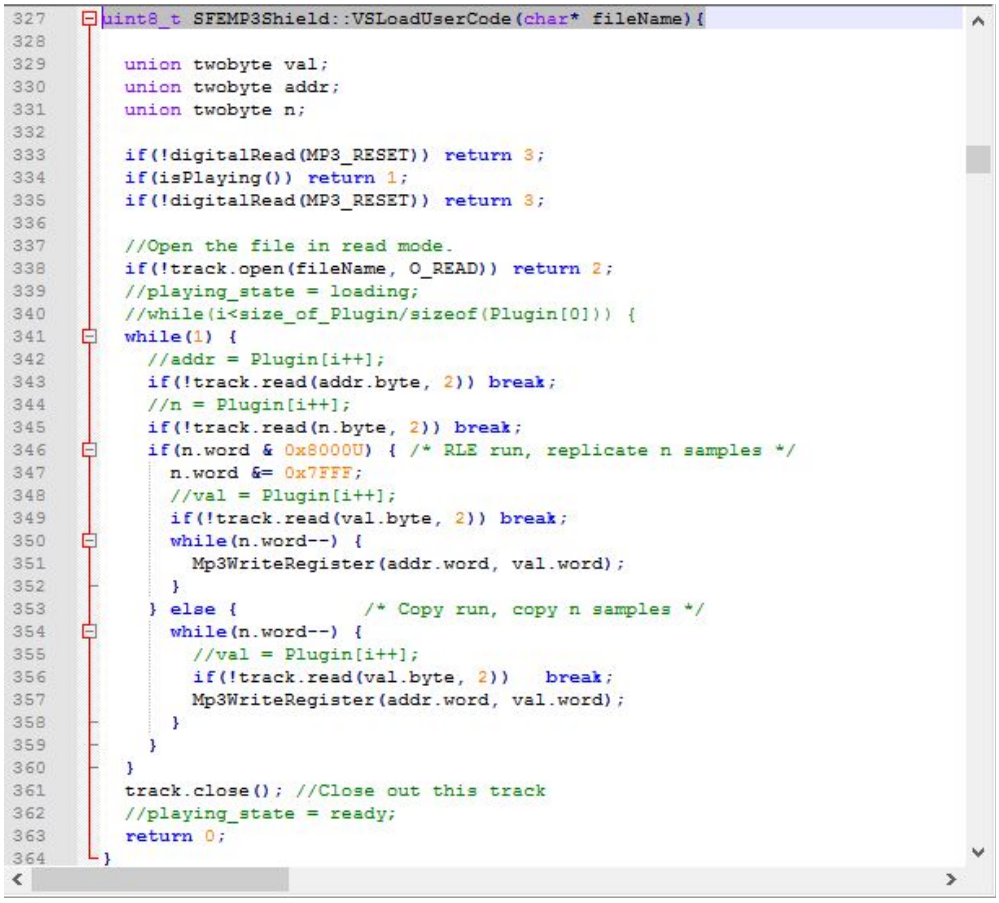
```
    bool resumeMusic();
    uint8_t resumeMusic(uint32_t);
    static void available();
    void getAudioInfo();
    uint8_t enableTestSineWave(uint8_t);
    uint8_t disableTestSineWave();
    uint16_t memoryTest();
    uint8_t ADMixerLoad(char*);
    void ADMixerVol(int8_t);
    int8_t getVUmeter();
    int8_t setVUmeter(int8_t);
    int16_t getVUlevel();
    void SendSingleMIDInote();
    uint8_t VSLoadUserCode(char*);
    static void Mp3WriteRegister(uint8_t, uint8_t, uint8_t);
    static void Mp3WriteRegister(uint8_t, uint16_t);
    static uint16_t Mp3ReadRegister (uint8_t);
    static uint16_t Mp3ReadWRAM(uint16_t);

  private:
    static SdFile track;
    static void refill();
    static void flush_cancel(flush_m);
    static void spiInit();
    static void cs_low();
    static void cs_high();
    static void dcs_low();
    static void dcs_high();
    static void Mp3WriteWRAM(uint16_t, uint16_t);
    void getTrackInfo(uint8_t, char*);
    static void enableRefill();
    static void disableRefill();
    void getBitRateFromMP3File(char*);

    //Create the variables to be used by SdFat Library

/** \brief Boolean flag indicating if filehandle is streaming.*/
    static state_m playing_state;
```

Next, if we want to use sparkfun's "VSLoadUserCode()" function to load our own user tables, we have to comment out a few things to get it to work. Modify the function like seen below:



```
uint8_t SFEMP3Shield::VSLoadUserCode(char* fileName){

  union twobyte val;
  union twobyte addr;
  union twobyte n;

  if(!digitalRead(MP3_RESET)) return 3;
  if(isPlaying()) return 1;
  if(!digitalRead(MP3_RESET)) return 3;

  //Open the file in read mode.
  if(!track.open(fileName, O_READ)) return 2;
  //playing_state = loading;
  //while(i<size_of_Plugin/sizeof(Plugin[0])) {
  while(1) {
    //addr = Plugin[i++];
    if(!track.read(addr.byte, 2)) break;
    //n = Plugin[i++];
    if(!track.read(n.byte, 2)) break;
    if(n.word & 0x8000U) { /* RLE run, replicate n samples */
      n.word &= 0x7FFF;
      //val = Plugin[i++];
      if(!track.read(val.byte, 2)) break;
      while(n.word--) {
        Mp3WriteRegister(addr.word, val.word);
      }
    } else {           /* Copy run, copy n samples */
      while(n.word--) {
        //val = Plugin[i++];
        if(!track.read(val.byte, 2))   break;
        Mp3WriteRegister(addr.word, val.word);
      }
    }
  }
  track.close(); //Close out this track
  //playing_state = ready;
  return 0;
}
```

Sparkfun's library comes with several example sketches for use with the Arduino, however, they are all built for the file-player plugin. We developed a sketch off of the "FilePlayer" example sketch to implement the spectrum analyzer plugin and ADMonEQ plugins. This sketch is extremely useful, since we want to also be able to play a song and analyzer its frequency spectra in real time. You should have access to our sketch. If not, a screenshot of how our example sketch uploads the plugins is displayed below:

```
else if (key_command == 'S') {
 Serial.println(F("Attempting to load Spectrum Analyzer plugin"));
 char pluginname[] = "specAn_2.053"; //Name of plugin
 result = MP3player.VSLoadUserCode(pluginname); //load plugin
 if (result != 0) {
   Serial.print(F("Error code: "));   // return Any Value other than zero indicates a problem occurred.
   Serial.print(result);
   Serial.println(F("  Load function terminated"));
   }
   else {
     Serial.println(F("Plugin Loaded!"));
   }
 SetBands(frequency,nBands);
 GetAnalysis(data,frequency);
 readFreq();
 }


else if (key_command == 's') {
 Serial.println(F("Attempting to load Spectrum Analyzer plugin"));
 char pluginname[] = "specAn_1.053"; //Name of plugin
 result = MP3player.VSLoadUserCode(pluginname); //load plugin
 if (result != 0) {
   Serial.print(F("Error code: "));   // return Any Value other than zero indicates a problem occurred.
```

### 3.2 Using Adafruit Library

Adafruit's library uses the more-common IMG file type to upload plugins. Usually copies of the plugin files are provided as an .img file by VSLI when you download a zip file from their site. However, if one is not provided, there are searchable solutions for properly converting to an .img file online.

Download Adafruit's library for the VS1053B and open up your Arduino IDE to start coding.
**Link:** https://github.com/adafruit/Adafruit_VS1053_Library

Before using their library, we must address one of the function we will be using: prepareRecordOgg(). Now typically, we would want to use a function that is less obviously specific to a particular plugin. We will not be using the Ogg encoder plugin for this project. However, this code does a few things that we need for uploading a plugin in general (for example, it clears the Bass register and disables all interrupts). So we will modify this function to be less specific to the Ogg encoder plugin so we can use it to upload anything.

Comment out the sections of code as seen below. These lines of code essentially confirm whether or not the plugin you just uploaded was the Ogg encoder. Obviously, we don't care, nor want to check for that, so comment it out and the function should run just fine.

```
boolean Adafruit_VS1053::prepareRecordOgg(char *plugname) {
  sciWrite(VS1053_REG_CLOCKF, 0xC000);  // set max clock
  delay(1);    while (! readyForData() );

  sciWrite(VS1053_REG_BASS, 0);  // clear Bass

  softReset();
  delay(1);    while (! readyForData() );

  sciWrite(VS1053_SCI_AIADDR, 0);
  // disable all interrupts except SCI
  sciWrite(VS1053_REG_WRAMADDR, VS1053_INT_ENABLE);
  sciWrite(VS1053_REG_WRAM, 0x02);

  int pluginStartAddr = loadPlugin(plugname);
  //if (pluginStartAddr == 0xFFFF)   //hex 0xFFFF, Dec 65535 (all bits are 1:  0b1111111111111111)
  //{
  //#ifdef DEBUGLOAD
  //   Serial.print("F: Plugin Start (chk for not 0xFFFF) at $"); Serial.println(pluginStartAddr, HEX);
  //#endif
  //return false;
  //}

//  if (pluginStartAddr != 0x34) //hex 0x34, Dec 52, 4 in Ascii, 0b110100
//  {
//  #ifdef DEBUGLOAD
//    Serial.print("F: Plugin Start (chk for not 0x34) at $"); Serial.println(pluginStartAddr, HEX);
//  #endif
//  return false;
//  }
  #ifdef DEBUGLOAD
  Serial.print("T: Plugin Start (general case) at $"); Serial.println(pluginStartAddr, HEX);
  #endif
  return true;
}
```

**Warnings**

We spent most of our time trying to get the spectrum analyzer working on the VS1053B shield. Although we have been able to load other plugins using this method, we never confirmed that the spectrum analyzer works using the method described above, so it is not yet proven. Read the documentation on the spectrum analyzer very carefully to make in the case that there are shortcomings to the method described above.

### 3.3 SD card Method:

Sparkfun: Copy the .053 file (the file we converted using the perl script earlier) onto a micro SD card and plug the card into the VS1053B. Run the program and use the Arduino's serial monitor to activate the plugin.

Adafruit: Copy the IMG file onto the micro SD card and plug the card into the VS1053B. Run the program and use the Arduino's serial monitor to activate the plugin.

### 3.4 Loading Table Method:

VSLI has provided, not only the plugin in c code, but also a brief loading function to incorporate into your code. You can either copy the code below directly into your sketch or include the file as a header file for your sketch and call the function accordingly.

The code below is generalized for any microprocessor. You'll notice that the code uses functions that are not defined for Arduino (WriteVS10XXRegister()). Luckily, Sparkfun's and Adafruit's libraries have already defined read and write functions for us.

For Sparkfun, we can use Mp3WriteRegister() to implement this code. Or you can user their pre-defined VSLoadUserCode() function described earlier. For Adafruit, we can use spiwrite() to implement this.

```
/* User application code loading tables for VS10xx */

#if 0
void LoadUserCode(void) {
  int i = 0;

  while (i<sizeof(plugin)/sizeof(plugin[0])) {
    unsigned short addr, n, val;
    addr = plugin[i++];
    n = plugin[i++];
    if (n & 0x8000U) { /* RLE run, replicate n samples */
      n &= 0x7FFF;
      val = plugin[i++];
      while (n--) {
        WriteVS10xxRegister(addr, val);
      }
    } else {           /* Copy run, copy n samples */
      while (n--) {
        val = plugin[i++];
        WriteVS10xxRegister(addr, val);
      }
    }
  }
}
#endif

#ifndef SKIP_PLUGIN_VARNAME
#define PLUGIN_SIZE 1002
const unsigned short plugin[1002] = { /* Compressed plugin */
#endif
  0x0007, 0x0001, 0x8050, 0x0006, 0x0004, 0x2800, 0x2f40, 0x0000, /*    0 */
  0x0024, 0x0007, 0x0001, 0x8052, 0x0006, 0x00d6, 0x3e12, 0xb817, /*    8 */
  0x3e12, 0x3815, 0x3e05, 0xb814, 0x3615, 0x0024, 0x0000, 0x800a, /*   10 */
  0x3e10, 0x3801, 0x0006, 0x0000, 0x3e10, 0xb803, 0x0000, 0x0303, /*   18 */
  0x3e11, 0x3805, 0x3e11, 0xb807, 0x3e14, 0x3812, 0xb884, 0x130c, /*   20 */
  0x3410, 0x4024, 0x4112, 0x10d0, 0x4010, 0x008c, 0x4010, 0x0024, /*   28 */
  0xf400, 0x4012, 0x3000, 0x3840, 0x3009, 0x3801, 0x0000, 0x0041, /*   30 */
  0xfe02, 0x0024, 0x2900, 0x8880, 0x48b2, 0x0024, 0x36f3, 0x0844, /*   38 */
  0x6306, 0x8845, 0xae3a, 0x8840, 0xbf8e, 0x8b41, 0xac32, 0xa846, /*   40 */
  0xffc8, 0xabc7, 0x3e01, 0x7800, 0xf400, 0x4480, 0x6090, 0x0024, /*   48 */
  0x6090, 0x0024, 0xf400, 0x4015, 0x3009, 0x3446, 0x3009, 0x37c7, /*   50 */
  0x3009, 0x1800, 0x3009, 0x3844, 0x48b3, 0xe1e0, 0x4882, 0x4040, /*   58 */
  0xfeca, 0x0024, 0x5ac2, 0x0024, 0x5a52, 0x0024, 0x4cc2, 0x0024, /*   60 */
  0x48ba, 0x4040, 0x4eea, 0x4801, 0x4eca, 0x9800, 0xff80, 0x1bc1, /*   68 */
  0xf1eb, 0xe3e2, 0xf1ea, 0x184c, 0x4c8b, 0xe5e4, 0x48be, 0x9804, /*   70 */
  0x488e, 0x41c6, 0xfe82, 0x0024, 0x5a8e, 0x0024, 0x525e, 0x1b85, /*   78 */
```

Upon loading the code, the plugin should activate the spectrum analyzer plugin automatically.

**Note:** Any patches can uploaded in the same methods described above, however, it is not recommended to attempt loading these patches in tandem with the spectrum analyzer, as many of them are incompatible with each other. The spectrum analyzer document notes this, however, nowhere else online were able to find any indication as to which patches and plugins are compatible with each other. So we recommend avoiding patches altogether if at all possible.

### 3.5 Loading Order

There is an order to uploading the spectrum analyzer plugin to get it to work properly with the Arduino. Before loading any plugins, make sure that your code uses the debug methods (and code functions) in section 4. Obviously, results of spectrum analysis are not immediately sent to the Arduino. They must be read first. With this in mind, make sure to load the plugins in the following order:

1) When loading the spectrum analyzer for a line in input, first run the spectrum analyzer (you should not get any values reported back to you). Then play an mp3 or other sound file and run the spectrum analyzer again. You should get results every time the plugin is run after this.
2) When loading the spectrum analyzer for a mic input, first run the ADmonEQ plugin and activate it. You should hear a small amount of static playing through the output jack. Run the spectrum analyzer and you should see results from the mic input (provided you have included code to display them).

### 3.6 Activating a plugin

Normally a plugin requires a certain value to be written to the AIADDR register address to activate it. The ADmonEQ plugin is one of these such plugins. 0x0220 must be written to AIADDR to activate the ADMonEQ plugin after you have uploaded it. The code snapshot below is how we activate the ADMonEq in our code. The other read and write address in the snapshot can be read about in the ADmon documentation under “how to load”.

```
  Serial.println(F("Activating loaded plugins"));
 //Settings for Plugin control
  int ADM_volume= -3; //set volume level between -3 and -31 (-31 is min)
//already defined in the SFEMP3shield.h file
  #define SCI_AIADDR            0x0A //register address (dec of 10)
  #define SM_LINE1            0x4000
  #define SCI_AICTRL0           0x0C //this is one of 3
  union twobyte MP3AIADDR;
  union twobyte MP3AICTRL0;
 MP3AIADDR.word = MP3player.Mp3ReadRegister(SCI_AIADDR); //two byte response
 if((ADM_volume > -3) || (-31 > ADM_volume)) {
    // Disable Mixer Patch -
    MP3AIADDR.word = 0x0221; //value to disable
    MP3player.Mp3WriteRegister(SCI_MODE, (MP3player.Mp3ReadRegister(SCI_MODE) & SM_LINE1)); //set to MIC input
    MP3player.Mp3WriteRegister(SCI_AIADDR, MP3AIADDR.word);
  } else {
    // Set Volume
    MP3AICTRL0.word = MP3player.Mp3ReadRegister(SCI_AICTRL0);
    MP3AICTRL0.byte[1] = (uint8_t) ADM_volume; // upper byte appears to have no affect
    MP3AICTRL0.byte[0] = (uint8_t) ADM_volume; //lower byte seems to have the effect
    MP3player.Mp3WriteRegister(SCI_AICTRL0, MP3AICTRL0.word);
   // Enable ADMon Patch
    MP3AIADDR.word = 0x0220; //value to enable
    MP3player.Mp3WriteRegister(SCI_AIADDR, MP3AIADDR.word);
  }
 }
```

**3.7 Writing and Reading Registers**

There are several main register addresses, key to understanding how communication between the VS1053B and other microprocessors works. WRAM, WRAMADDR, and AIADDR are the three most important register addresses.

Taken from Sparkfun's SFEMP3Shield.h:

1) SCI_AIADDR : is a Read/Write register indicates the start address of the application code written earlier with SCI_WRAMADDR and SCI_WRAM registers. If no application code is used, this register should not be initialized,or it should be initialized to zero.
2) SCI_WRAMADDR : is a Write only register used to set the program address for following SCI_WRAM writes/reads. Use an address offset from the following table to access X, Y, I or peripheral memory.
3) SCI_WRAM is a Read/Write register used to upload application programs and data to instruction and data RAMs. The start address must be initialized by writing to SCI_WRAMADDR prior to the first write/read of SCI_WRAM. As 16 bits of data can be transferred with one SCI_WRAM write/read, and the instruction word is 32 bits long, two consecutive writes/reads are needed for each instruction word. The byte order is big-endian (i.e. most significant words first). After each full-word write/read, the internal pointer is autoincremented.

For using choosing between using the MIC or LineIn input for the spectrum analyzer, we need to activate one or the other by writing to SCI_LINE1.

Taken from Sparkfun's SFEMP3Shield.h:

1) SM_LINE_IN : is used to select the left-channel input for ADPCM recording. If '0', differential microphone input pins MICP and MICN are used; if '1', line-level MICP/LINEIN1 pin is used.

Look back up to the snapshot of our code under section 3.6. Our code illustrates how to choose between line input and mic input.

## 4 Debugging/Results

We suggest attaching headphones to the output jack and listening to the output everytime you upload a plugin. Uploading a plugin incorrectly or in the wrong order often results in unexpected sounds and distortions (undefined behavior). The audio jack is sensitive to changes in operation of the chip, so if you are stuck and you are unsure whether anything is happening, this will help you.

The spectrum analyzer documentation has provided code for you to use to retain results from spectrum analysis. This code will display the results of your analysis to the Arduino serial monitor. Below are our own implementations of this code, using the sparkfun library. Note that if you are using the Adafruit library, use sciWrite() instead of Mp3WriteRegister().

**Note:** all 3 functions read from different register addresses depending on if you're using the spectrum analyzer default plugin or the 2nd plugin.

GetAnalysis() : Gets the results of running the spectrum analysis on a given input.

```
// Function modified from user 'CheeseBurger' from vsdsp-forum.com
void GetAnalysis(unsigned int data[14],const short int frequency[14]) {
  if(!frequency) return;
  delay(100);

  //Get nBands
  //MP3player.Mp3WriteRegister(SCI_WRAMADDR, BASE + 2);
  //MP3player.Mp3WriteRegister(7, 6146); // specAn1
  MP3player.Mp3WriteRegister(7, 6162); // specAn2
  //int nBands = MP3player.Mp3ReadRegister(SCI_WRAM);
  int nBands = MP3player.Mp3ReadRegister(6); //direct integer write of address

  Serial.print("There are ");
  Serial.print(nBands,DEC);
  Serial.println(" bands\n");
  //nBands = 14; // Ignore incorrect nBands

  //Read analysis
  MP3player.Mp3WriteRegister(7, 6164);
  for(int i = 0; i < nBands; i++) {
    //data[i] = MP3player.Mp3ReadRegister(6);
    data[i] = MP3player.Mp3ReadRegister(6) & 31; //Current value is in bits 0-5
    Serial.print("Spectrum Data: ");
    Serial.println(data[i]);
  }
}
```

SetBands() : Sets the frequency bands you wish to read values from for a given input. There is memory for up to 15 set bands to read from for the 2nd spectrum analyzer plugin. The default should be able to store up to 23 bands.

```
// Function modified from user 'CheeseBurger' from vsdsp-forum.com
void SetBands(const short int frequency[14], int nBands) {
  if(!frequency || nBands > 23) return;

  //Set band frequencies
  int i;
  MP3player.Mp3WriteRegister(7, 0x1868);
  for(i = 0; i < nBands; i++)
    MP3player.Mp3WriteRegister(6, frequency[i]);
  if(i < 23)
    MP3player.Mp3WriteRegister(6, 25000); // Terminate partial frequency lists with 25000

  //Reactivate Analyzer
  MP3player.Mp3WriteRegister(7, 0x1811); //for specApp2
  //MP3player.Mp3WriteRegister(7, 0x1801); // for specApp1
  MP3player.Mp3WriteRegister(6, 0);
}
```

readFreq() : reads the frequencies of the spectrum being analyzed. If you have defined your own frequencies, then you should see values close to your defined frequencies displayed, using this function.

```
void readFreq(){
  //MP3player.Mp3WriteRegister(7, 0x1801); // for specApp1
  MP3player.Mp3WriteRegister(7, 0x1811); // for specApp2
  //MP3player.Mp3WriteRegister(SCI_WRAMADDR, BASE+1); //using preset macros provided
  short int frequency[14] = {0};
  unsigned int rate = MP3player.Mp3ReadRegister(6);
  int bands = MP3player.Mp3ReadRegister(6);

  Serial.print(" rate = "); Serial.println(rate);
  Serial.print(" bands = "); Serial.println(bands);

  int i;
  for(i = 0; i < bands ; i++){
    int a;
    //MP3player.Mp3WriteRegister(SCI_WRAMADDR, BASE + 0x1c + 3 * i); // from doc for specApp1
    MP3player.Mp3WriteRegister(SCI_WRAMADDR, BASE + 0x15 + 3 * i); //from forum for specApp2
    a = MP3player.Mp3ReadRegister(SCI_WRAM);
    frequency[i] = (long) rate * a >> 11;

    Serial.print("i = "); Serial.print(i); Serial.print(" ");
    Serial.print(", a = ");Serial.print(a); Serial.print(" ");
    Serial.print(", freq[i] = ");Serial.print(frequency[i]); Serial.println(" ");
  }
}
```

**4.2 Results:**

Below is one of our results from running the spectrum analyzer:

Using spectrum analyzer App2 and ADMonEQ
(analyzing sound through mic input):

```
There are 16 bands

Spectrum Data: 6
Spectrum Data: 7
Spectrum Data: 7
Spectrum Data: 5
Spectrum Data: 4
Spectrum Data: 3
Spectrum Data: 2
Spectrum Data: 2
Spectrum Data: 3
Spectrum Data: 1
Spectrum Data: 0
Spectrum Data: 0
Spectrum Data: 4
Spectrum Data: 2
Spectrum Data: 0
Spectrum Data: 18
 rate = 65021
 bands = 16
i = 0 , a = 1 , freq[i] = 31
i = 1 , a = 2 , freq[i] = 63
i = 2 , a = 4 , freq[i] = 126
i = 3 , a = 6 , freq[i] = 190
i = 4 , a = 9 , freq[i] = 285
i = 5 , a = 15 , freq[i] = 476
i = 6 , a = 24 , freq[i] = 761
i = 7 , a = 40 , freq[i] = 1269
i = 8 , a = 60 , freq[i] = 1904
i = 9 , a = 96 , freq[i] = 3047
i = 10 , a = 160 , freq[i] = 5079
i = 11 , a = 224 , freq[i] = 7111
i = 12 , a = 384 , freq[i] = 12191
i = 13 , a = 640 , freq[i] = 20319
i = 14 , a = 768 , freq[i] = 24382
i = 15 , a = 0 , freq[i] = 0
```

Note: The last frequency reading 0 is an error. The unction was accidently reading beyond the scope of saved frequency data. Other than that, this data works!

## 5 Useful References:

Make sure to read all necessary datasheets and documentation on these applications. I have also included useful links to the VSLI forum, where you can find information about people's previous work on projects similar to this:

Documents:
http://www.vlsi.fi/fileadmin/software/VS10XX/spectrumAnalyzer.pdf
https://www.sparkfun.com/datasheets/Components/SMD/vs1053.pdf

Forums and website:
http://www.vlsi.fi/en/products/vs1053.html
http://www.vsdsp-forum.com/phpbb/viewtopic.php?t=1421
http://www.vsdsp-forum.com/phpbb/viewtopic.php?f=11&t=478