Developing Software And Using the Spectrum Analyzer for the VS1053B

California PlugLoad Research Center
University of California, Irvine

DSP-FPGA

Author: Paolo Caraos
March 7, 2017

1 Introduction

This report explains the progress made by the DSP FPGA team on developing software and implementing open source libraries to work for the VS0153B. The team has two goals:

1. To implement the workflow for practical usage of the Spectrum Analyzer application
2. To learn how to develop custom plugin applications and run them on the VS1053B

Both of the goals were adequately met and general knowledge of the steps to recreate the workflow are documented in this report.

2 Required Items

2.1 Required Hardware

- VS1053B Breakout Board
- VS1053B Shield
- Arduino Uno
- Breadboard
- USB cable
- Wires
- Headphone/Earphones
- Microphone
- MicroSD card
- Headphone Jack

The VS1053B is an audio decoder and encoder designed by VLSI Solutions. Its ability to process audio either through microphone input or audio file while simultaneously running one of its special applications is the primary study of this research team. The audio files supported are listed in the VS1053B datasheet. In this project, the spectrum analyzer application is the main focus.

Adafruit and Sparkfun integrated VS1053B on a breakout board and a shield respectively, and they developed and publicized their own libraries. Since both boards have the same chip, the boards differ only in pin addressing. Most Arduino programs are translatable

between the two by simply changing the pin addresses. In this project, most of the applications were ran on the Sparkfun Shield.

In order to upload applications into the VS1053B, the direct UART approach or SPI Transfer via Arduino can be employed. This research team took the latter approach and therefore, only the SPI approach will be discussed. Connecting the shield onto the Arduino is straightforward, and connecting the Adafruit breakout board is described in A.1.

2.2 Required Software

- VS IDE (http://www.vlsi.fi/en/support/software/vside.html)
- Arduino IDE 1.8.0
- VS1053B Adafruit Library (https://github.com/adafruit/Adafruit_VS1053_Library)
- VS1053B Sparkfun Library (https://github.com/madsci1016/Sparkfun-MP3-Player-Shield-Arduino-Library)
- VLSI Plugin applications including the Spectrum Analyzer, Admoneq, Admixer (http://www.vlsi.fi/en/support/software/vs10xxapplications.html)
- VS1053B Patches (http://www.vlsi.fi/en/support/software/vs10xxpatches.html)
- HexEditor
- Plugin to .053 file perl script conversion (https://github.com/madsci1016/Sparkfun-MP3-Player-Shield-Arduino-Library/blob/master/plugins/vs_plg_to_bin.pl)
- ActivePerl-5.24
- Makeloadingtable.exe (http://www.vsdsp-forum.com/phpbb/viewtopic.php?f=7&t=1129&p=4659&hilit=aiaddr#p4647)

VLSI Solutions developed their own user-friendly IDE to compile custom application plugins. It must be noted, however, that some example applications in the IDE still do not work with workflow documented in this report. This IDE can compile the chosen C code into an .img or .plg file using certain configurations. Furthermore, after compilation, the command line in the IDE will specify the start address of the application which is a crucial piece of information when running your application.

Since the SPI transfer approach is employed, writing the Arduino code with open source libraries on VS1053B(Adafruit or Sparkfun) is necessary. The Arduino program will upload the plugin using the library functions and control the flow of running the application and receiving audio input either through microphone or audio file.

Some applications written by VLSI Solutions require certain patches. These patches are C code translated into hexadecimal machine code that extend or repair the intended functionalities of VS1053B when running specific applications. Because patches are application specific, it is recommended to only load a patch specified by the application notes of the plugin desired. For the Spectrum Analyzer, no patches are required

During the process of troubleshooting and debugging, the team used a hex editor to examine plugin files. While this may be supplemental to understanding how to program the chip, it is not completely necessary.

The perl script is used to extract the hexadecimal instructions in the plugin files and store it in a .053 file. This .053 file is then stored into the SD card where the Arduino program will pull the hexadecimal instructions and write into the instruction memory (RAM) of VS1053B. Consequently, ActivePerl-5.24 is needed to run this script.

makeloadingtable.exe is a program used alongside VS IDE to determine the start address of an application. This executable can be found attached to Staff Member Panu's comment in the provided link.

3 Developing a Custom Program

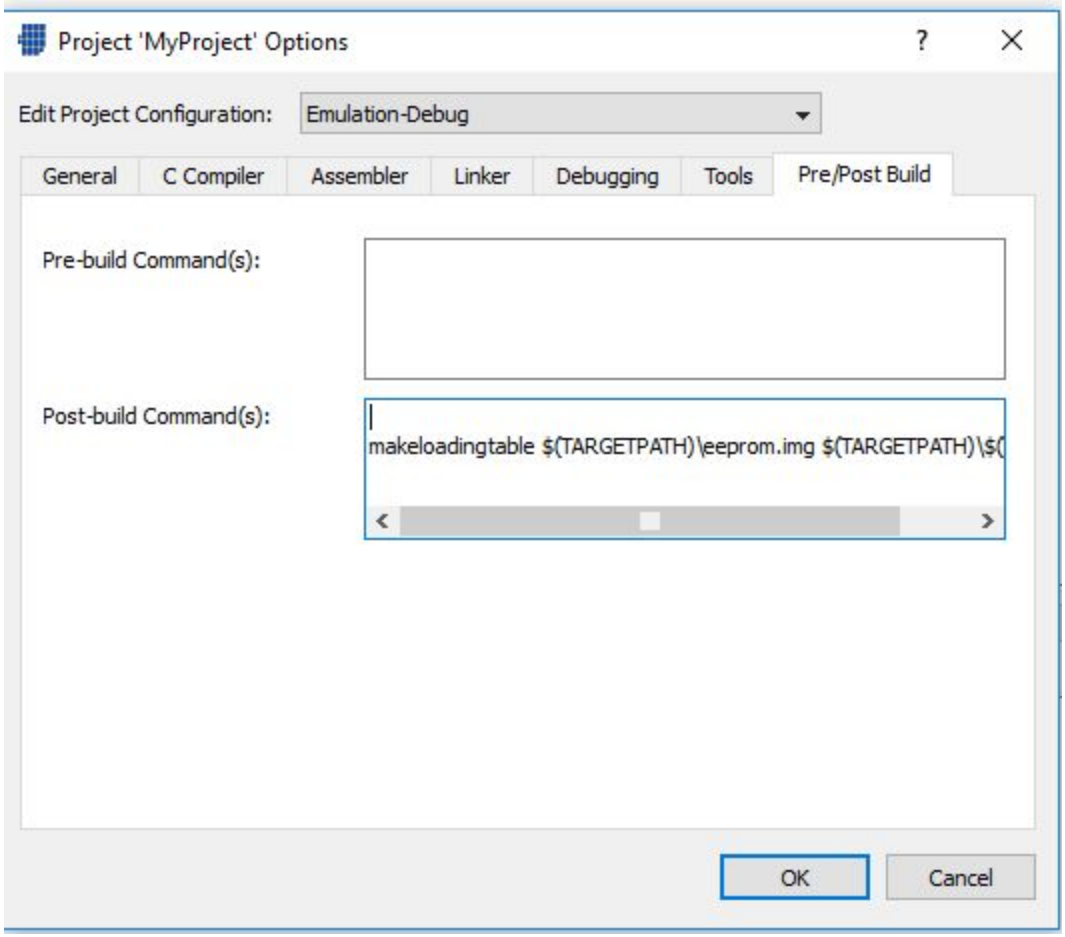
Since this is embedded programming, the applications are written in C and then compiled using the VS IDE. Simply create a new project on the IDE and start writing an application. For further information on how to start a project, please refer to the VS IDE User Manual.

3.1 Calculating the Application Start Address

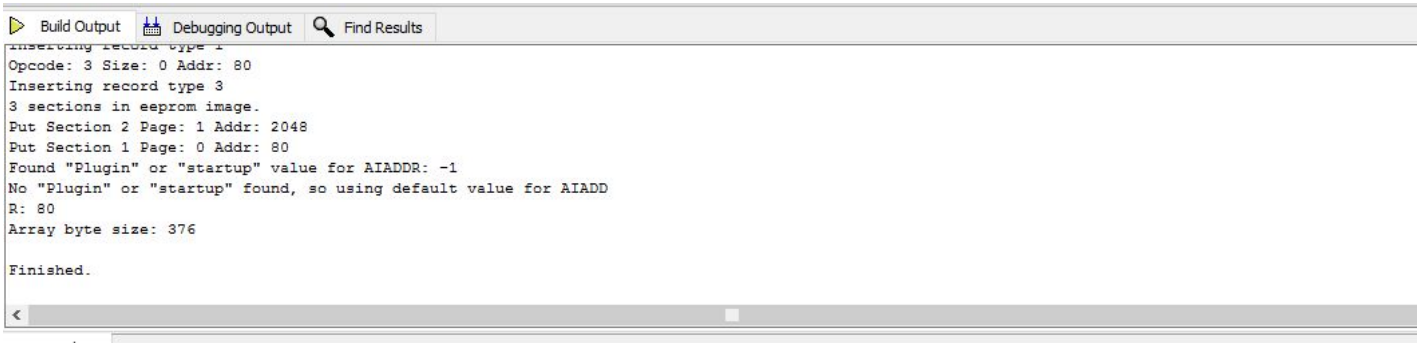
When the program is ready to be compiled into a plugin, the start address of the program must be determined in order to activate it. First build the entire solution to make sure the code does not need further modifications and is bug free. Then put the "makeloadingtable.exe" downloaded from the link above into VSIDE/bin in ProgramFiles(x86). After that under the "Project" tab in the IDE, go to "Properties". Under "Pre/Post Build" in the "Post Build" box, type in the following configuration

```
makeloadingtable $(TARGETPATH)\eeprom.img
$(TARGETPATH)\$(TARGET)
```

The result should appear as follows



Hit “Ok”. Then Build All. In the IDE Log Window, it should indicate the value of SCI_AIADDR, the start address of the application.



In the above example, the example HelloWorld program was compiled and the value of the start address is determined to be 80 in decimal or 0x050 hexadecimal. This value must be remembered.

3.2 Generating the Plugin

Once the start address has been found, certain settings must be configured first to generate the plugin. First, under the “Project” tab in the IDE, go to “Properties”. Under “Pre/Post Build” in the “Post Build” box, type in the following configuration

```
coff2allboot -x0x50 -iplugin $(TARGETPATH)/$(TARGET) -o
$(TARGETPATH)/$(PROJNAME).plg
coff2boot -x 0x50 $(TARGETPATH)\$(TARGET)
$(TARGETPATH)\eeprom.img
```

Take note of the number 0x050 which indicates the start address of plugin. For most applications made this is the normal start address. For some special applications this value will have to change. Compiling such special applications was not covered in this project. That value must be adjusted accordingly when a different start address is calculated using the process described in Section 3.1.

The result should appear as follows

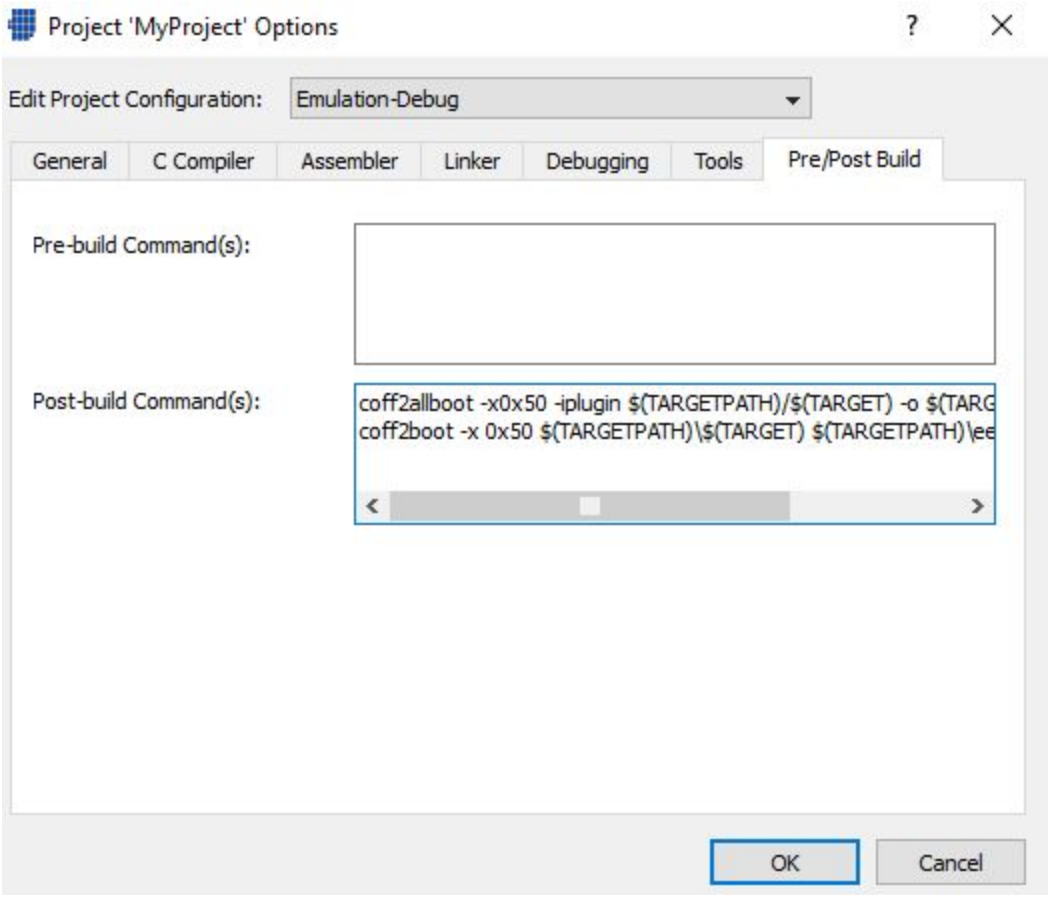


Figure 3.1

After this click Build All to compile the solution

This will generate a ".plg" file and an ".img" file in the solutions directory normally located in the user's Documents directory. When loading via array method (Section 4.2) this the header file that must be included in the Arduino code. For more information about how to load the array, see Section 4.2.

3.3 Converting the Plugin for SD Card Use

The Adafruit board uses .img files (generated by the steps described above) to load the plugin. Simply, save this file onto an SD card and move on to the next section to learn how to load a plugin.

On the other hand, the Sparkfun board uses a .053 file. In order to generate this file, it must be converted from a .plg file using a Perl script. The instruction are as described below.

```
1  This script uses PERL to convert a VS1053 .PLG file into a .053 RLE compressed file for the VS1053b.
2
3  1) Load ActivePerl-5.24 (or newer)
4  2) place file in a dedicated directory along with the perl conversion script
5  3) Open Administration level command prompt and run the following script (shown for the equilzer in a temp
   directory called "A"
6  3) perl vs_plg_to_bin.pl C:\A\spectrumAnalyzerAppl1053b.plg                    C:\A\spctan.053
```

After generating the .053 file, save the file onto a SD card.

4 Uploading the Plugin

The process of loading the plugin requires reading the contents of the plugin application from an SD card attached to the board (or an array from a header file) and writing that information into the instruction memory(RAM) of VS1053B where it will be ready for execution.

4.1 Loading from an SD Card

Save the plugin file made in Section 3 into an SD card and insert the SD card into the board. Then using the Sparkfun/Adafruit libraries, call the function that will load the application into the RAM by specifying the plugin name. In Sparkfun library the function name is

```
uint8_t VSLoadUserCode(char*);
```

And in the Adafruit library this is

```
//Loads user plugin in the form of an array
bool loadUserCode(const unsigned short plugin[]);
```

Both functions read from the SD card and will only operate when the SD card is available.
For Sparkfun shield, the application plugin must be .053 file and for Adafruit, the plugin must be in the form of an .img file. After successful execution of the above functions, the plugin is now in the instruction memory of VS1053B, awaiting activation.

4.2 Loading from an Array

In lieu of the an SD card, a loading array can be used. However, the Adafruit and Sparkfun libraries do not provide the function to facilitate this. The programmer must then implement their own plugin load function. Fortunately, the pseudo-code is provided in every application notes (under How to Load a Plugin) and .plg file generated by the VS IDE.

```
#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
```

In order to use this piece of code, simply include the plugin file generated by VS IDE in your Arduino program and remove the preprocessor macros that prevent compilation of this code. The write function must also be replaced with the corresponding write function in the library being used. Do not forget to move the plugin file into a directory that the Arduino can find.

5 Running the application

5.1 Starting an Application

As described in Section 9.6.10 of the VS1053B Datasheet, running an application required writing the value of the start address to SCI_AIADDR. This is referred to as an application hook that begins the plugin application loaded via the steps described in the prior sections. Therefore, loading the plugin using either method, writing to this register will activate the plugin. For more information regarding on how to write to registers see Section A.3. In this project, the AdmonEq is activated by writing the value 0x220 to SCI_AIADDR.

For some programs, such as the Spectrum Analyzer, this method is not required as this special application becomes active when it is loaded successfully. Because of this nature, the spectrum analyzer can ran in tandem with another application that is mentioned to be compatible with it. In this project that said application is the AdmonEq.

5.2 Using the Spectrum Analyzer

5.2.1 Necessary Functions

Loading the spectrum analyzer prepares the VS1053B to process audio input. After processing audio input, certain registers are updated with the results. To retrieve these values, the programmer must first implement certain functions into the Arduino program. The programmer can define the frequencies at which the audio will be analyzed by implementing the settingBands code described in Section 1.3 of the Spectrum Analyzer application notes. The source code implemented into the Arduino for this project is shown below.

```
// 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(SCI_WRAMADDR, BASE + 0x68);
  for(i = 0; i < nBands; i++)
    MP3player.Mp3WriteRegister(SCI_WRAM, frequency[i]);
  if(i < 23)
    MP3player.Mp3WriteRegister(SCI_WRAM, 25000); // Terminate partial frequency lists with 25000

  //Reactivate Analyzer
  MP3player.Mp3WriteRegister(SCI_WRAMADDR, BASE+1);
  MP3player.Mp3WriteRegister(SCI_WRAM, 0);
}
```

Please pay attention to the value of BASE and adjust it according to what the application notes specifies. It is 0x1800 for the first version of the Spectrum Analyzer and 0x1810 in the second version. The frequencies set at these bands can be analyzed with the function mentioned in Section 1.4 of the Spectrum Analyzer applications notes. The implementation used for this project is shown below.

```
void readFreq(){
  MP3player.Mp3WriteRegister(SCI_WRAMADDR, BASE + 1);
  short int frequency[14] = {0};
  int rate = MP3player.Mp3ReadRegister(SCI_WRAM);
  int bands = MP3player.Mp3ReadRegister(SCI_WRAM);

  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);
    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(" ");
  }
}
```

In order to analyze the amplitude of the input, the function described in Section 1.2 of the Spectrum Analyzer application notes was implemented into the Arduino program. The implementation used for this project is shown below. Note that that the results of the amplitude analysis should not exceed the value 32. Thus a modulo function is done onto the result read from the SCI_WRAM register. (Modulo 32 is also the same as bitwise & 31).

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

  //Get nBands
  MP3player.Mp3WriteRegister(SCI_WRAMADDR, BASE + 2);
  int nBands = MP3player.Mp3ReadRegister(SCI_WRAM);

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

  //Read analysis
  MP3player.Mp3WriteRegister(SCI_WRAMADDR, BASE + 4);
  for(int i = 0; i < nBands; i++) {
    data[i] = MP3player.Mp3ReadRegister(SCI_WRAM) & 31; //Current value is in bits 0-5
    Serial.print("Spectrum Data: ");
    Serial.println(data[i]);
  }
```

These functions are called everytime the Spectrum Analyzer is loaded into the chip, so in this project, the menu option of loading the Spectrum Analyzer includes these three functions in the following order.

```
  }
SetBands(frequency,nBands);
GetAnalysis(data,frequency);
readFreq();
}
```

A Fileplayer function must also be implemented into the workflow of the Arduino program. This can be easily borrowed from one of the examples in the libraries. In this project the FilePlayer example from Sparkfun was used.

5.2.2 Analyzing Audio Files

In order to analyze audio files the following control flow must be implemented

1. Load the Spectrum Analyzer
2. Play the Audio File
3. Load the Spectrum Analyzer again

Since retrieving results of the Spectrum Analyzer is also included in the loading function that is implemented in this project, the results shown after the first step, before any audio input, will be meaningless. Note that with the way FilePlayer is implemented, playing an audio file also requires saving the audio file into the SD card. It is recommended that the file be saved as an Mp3. By default the audio input is set to be taken from LINEIN. After the third step the results should display the amplitude and the frequencies at the bands specified.

5.2.3 Analyzing Microphone Input

In order to analyze audio files the following control flow must be implemented

1. Load and activate the AdmonEq
2. Input Audio through the mic
3. Load the Spectrum Analyzer

In this project, the lines needed to activate AdmonEq along with the microphone input are written right after loading the AdmonEq plugin. These lines are shown below.

```
#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 = 0x0F01; //value to disable
  MP3player.Mp3WriteRegister(SCI_MODE, (MP3player.Mp3ReadRegister(SCI_MODE) & SM_LINE1)); //set to MIC inpu
  MP3player.Mp3WriteRegister(SCI_AIADDR, MP3AIADDR.word);
} else {
  // Set Volume
  MP3AICTRL0.word = MP3player.Mp3ReadRegister(SCI_AICTRL0);//read then overwritten, may consider doingd sor
  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 Mixer Patch
  MP3AIADDR.word = 0x0F00; //value to enable
  MP3player.Mp3WriteRegister(SCI_AIADDR, MP3AIADDR.word);
}
```

The microphone input is the continuously applied until the step 3 is finished.

6 References

6.1 Documents

- VS1053B Datasheet
- Spectrum Analyzer Application Notes (http://www.vlsi.fi/fileadmin/software/VS10XX/spectrumAnalyzer.pdf)
- VS1053B SPI Notes (http://www.vlsi.fi/fileadmin/app_notes/vs10XXan_spi.pdf)
- VS1053B Patches Application Notes (http://www.vlsi.fi/fileadmin/software/VS10XX/vs1053b-patches.pdf)
- VS IDE User Manual (http://www.vlsi.fi/fileadmin/manuals_guides/vside_um.pdf)

6.2 Forum Links

- AdmonEq plugin (http://www.vsdsp-forum.com/phpbb/viewtopic.php?t=354)
- Running the spectrum analyzer (http://www.vsdsp-forum.com/phpbb/viewtopic.php?f=11&t=482&p=1919&hilit=spectrum+analyzer+vs1053b#p1919)

A Appendix

A.1 Connecting the Breakout Board

Connecting the breakout board to the Arduino is described in the link

(https://learn.adafruit.com/adafruit-vs1053-mp3-aac-ogg-midi-wav-play-and-record-codec-tutorial/downloads-and-links?view=all).

Because of this pin setup, the pin addressing is slightly different between the Adafruit and the Sparkfun library. In order to make Adafruit Arduino programs work on the Sparkfun Shield, the following configurations must be done to the preprocessor definitions.

```
//Default SPI Pins for Arduino Uno
#define RESET 8       // VS1053 reset pin (output)
#define CS 6         // VS1053 chip select pin (output)
#define DCS 7         // VS1053 Data/command select pin (output)
#define CARDCS 9      // Card chip select pin
#define DREQ 2        // VS1053 Data request, ideally an Interrupt pin
```

A.2 SPI Transfer

SPI transfer is a way for the Arduino and the V1053B board to communicate with each other serially. In the source code, SPI transfer is used specifically when writing or reading to registers. However, in order to do this successfully certain configurations must be set. These configurations include clock phase, clock polarity, speed, and endianness. Fortunately, the open source libraries have saved future programmers some time by writing their write and read functions according to the configurations of VS1053B. If for any reason information about these configurations are required, the VS1053B Datasheet provides it all.

For more information about SPI Transfer, a concise tutorial can be found in this link (http://arduino.stackexchange.com/questions/16348/how-do-you-use-spi-on-an-arduino).

A.3 Reading/Writing to VS1053B Registers

Both libraries have implemented their own read and write functions that uses SPI Transfer. These functions can be used to replace the generic “WriteSCI” functions that may be encountered in other open source applications written by VLSI Solutions. Since communication is serial for this project, the data must be sent in a specific order. The address to be written to is sent first followed by the high byte then the lowbyte (because this processor works with big endian). When reading, just the address byte is sent and the value read is returned by the function. Both read and write functions must respect the rules about DREQ which can be read in the processor datasheet.

The common use for these functions are reading and writing to RAM. This process requires writing the address of the place to be read/write into the register SCI_WRAMMADDR and then reading/writing the value from/into SCI_WRAM. For example, in order to retrieve the amplitude results described in the GetAnalysis function in Section 5.2.1, SCI_WRAMMADDR should point to the location of the value desired in memory which is BASE+4. The register SCI_WRAM will then be updated with the information in that memory location every processor clock cycle. The results can then be retrieved by reading SCI_WRAM.