# Introduction

I'm always on the lookout for cool new (at least new to me) hardware components that I might incorporate into one of my electronic/computer projects. Recently while shopping online at SparkFun.com I came across an evaluation board for a digital FM receiver about the size of a postage stamp that sparked (no pun intended) my interest. Having grown up listening to FM radio I thought it would be fun to build an FM receiver of my own design. While this may be seen by many to be a very retro project what with Internet radios, music players and smart phones surrounding us, it has turned out to be quite useful. Now I have a remote controllable FM radio that I listen to with headphones while I am working. It sets on my desk along side the Desktop Contemplator and the Unique Digital Clock that I have written about previously. My desk is becoming cluttered with all the *useful* projects I have designed and built.

While I listen to my FM radio/receiver with headphones it can just as easily be plugged into a stereo amp with speakers to provide sound for a whole room. The fidelity of this receiver is quite good when tuned to strong FM stations. Because of its compact size you could even incorporate this FM receiver into a boom box of your own design.

This project can be built by anyone with basic electronic assembly and schematic reading experience. The software is somewhat complex but is made simpler by the use of pre-existing libraries for the major hardware components. The Arduino Uno sketch (program) for this FM receiver project is available on the Nuts and Volts website. Because you have access to the source code you can make changes to the design and/or add new features to the receiver. If you come up with a cool new feature please send me the code at calhjh@gmail.com so I can incorporate it into my receiver/radio as well.

Let's begin by discussing the hardware. Software will be discussed a little later.

# Hardware

The hardware is built around an Arduino Uno board running at 16 MHz and 5 volts. I've been finding these boards on eBay for around $13 each so I bought a few. Other Arduino's could be used but some of the I/O pin assignments might need to change. Using an Arduino running at a different clock speed will also impact the design especially in the IR (Infra Red) detection area. None of these problem are insurmountable but you will be in uncharted territory if you deviate from the design presented here.

I initially planned to have physical controls on the FM receiver. A power switch, a rotary encoder for channel selection and a couple of push buttons for mode selection but I quickly realized that I would then need physical proximity to the receiver to manipulate it. I then thought adding IR remote control would be interesting but soon realized that if I added remote control the other physical controls would be redundant and unnecessary. In the end I decided to go the no control route so all of the receiver's functionality is controlled through an IR remote. Quite convenient really.

While not technically a control, there is a contrast adjustment for the LCD display which should be set once and left alone. I used a 10 turn, screw driver adjustable trimmer but any 10K to 20K ohm potentiometer could be used in its place.

The parts list for this project is in shown in Figure One and a schematic of the hardware is shown in Figure Two. I built my radio using point to point wiring (can you say rats nest ?) but a prototyping shield could be used for a cleaner build.

The LCD display provides a 4 bit parallel data interface to the Arduino while the Si-4703 FM receiver is connected to the Arduino via an i2c interface. Since the Arduino Uno is a 5 volt part and the FM receiver is a 3.3 volt part, a bi-directional level converter must be used between them.

The LCD display runs on 5 volts. The backlight for the LCD display is directly controlled by an output pin from the Arduino.

The stereo audio output cable is used as the antenna for the FM receiver so you must pay attention to the length of the interconnect. The cord on the headphones I use works fine as an antenna as I can pick up all of the FM stations in my area. When strong stations are tuned in, the audio quality is top notch. When the receiver is receiving a stereo broadcast, the yellow stereo indicator LED lights up. Weaker stations are received in mono and the stereo indicator remains dark. A red, power on, LED lights when the radio is on.

An IR receiver (which is what Radio Shack calls it) is used to detect IR codes from the remote control. It filters out the 38 KHz IR carrier frequency from the received signal thereby making detection of the key codes more straight forward.

The radio is powered via a USB cable and a USB power supply. Alternatively the radio can be powered by connection to a USB port on your computer. There is no power switch. If you want to power off the radio, unplug the USB cable and/or power supply.

Wiring and wire routing for this project are non-critical. Be advised that as with all digital circuitry keeping the wires as short as possible/practical is always a good idea. Using a consistent color scheme for the wires is also a good idea. I used red for 5 volts, white for 3.3 volts and black for all ground connections. Various other wire colors are used for data and clock signals.

## *Packaging*

I packaged my radio using two 4”x6” pieces of clear 1/8” acrylic plastic in a sandwich like arrangement held together by wooden 1 1/2” dowel spacers in the corners. I like the naked electronics look. The LCD display is mounted to the front acrylic piece and the Arduino Uno and receiver board are mounted to the rear piece. The finished radio is free standing with this packaging approach. See the photos for details.

# Software

All of the radio's software was developed using the Arduino IDE (Integrated Development Environment) version 1.0.5 for OSX. Windows versions of the IDE are available if that is your chosen computing environment. Make sure you have the Board type set to Arduino Uno and the Serial Port set appropriately in the IDE. Of course you will need to plug the receiver's Arduino Uno into your computer via a USB cable to download the provided firmware via the IDE.

Three libraries are used in this project to ease the software development task. The Liquid Crystal library which comes standard with the Arduino IDE, an IR remote library for IR code detection and a library for controlling the Si-4703 digital FM receiver. The Resources section has pointers/links as to where these libraries can be obtained.

The Liquid Crystal library is preinstalled when you download the IDE so you need not do anything else to use it other than including *LiquidCrystal.h* in your sketches. To install the IRremote library you must first unzip the downloaded IRremote library file somewhere on your computer and then copy its contents to your arduino\libraries directory. Finally, rename the Arduino-IRremote-master directory to IRremote and you should be good to go.

Installing the Si-4703_Breakout library is similar. First download the library, unzip it, copy its contents to the arduino\libraries directory and then rename the directory called Arduino-Si4703-Library-libcode-only to Si4703_Breakout.

The Arduino IDE must be shut down while installing new libraries. Once restarted the IDE should pick up any newly installed libraries. You can check for proper installation by clicking *Examples* from the *File* menu. There you should see an *IRremote* entry with a bunch of example sketches along with a *Si4703_Breakout* entry with its example sketch.

Whereas the Liquid Crystal and the IRremote libraries can be used as is, the Si4703_Breakout library must be modified before being used in our application. This library was not built with the idea that someone might want to extend it with functionality the library didn't directly provide. For our application I needed the ability to mute/unmute the FM receiver and to poll the stereo reception indicator, functionality the library doesn't provide. Luckily the change to the library is trivial and only involves editing the header file, *Si4703_Breakout.h*. Again the Arduino IDE must be shut down while editing is being performed. Bring up the file, *Si4703_Breakout.h,* in a text editor of some kind. On my MAC I use TextEdit; on Windows you could us Notepad.

The change involves moving the following entries from the private section of the interface definition up to the public section. Once you have moved these four lines of text, save the file and we should be ready to go.

```
void readRegisters();

byte updateRegisters();

int getChannel();

uint16_t si4703_registers[16]; //There are 16 registers, each 16 bits large
```

With this change we are able to extend the functionality of the Si4703 library as required for our application.

## *Software Overview*

You may want to consult the DigitalFMRadio.ino sketch during the discussion to follow.

The sketch is broken up into numerous sections for organizational purposes. At the top of the sketch are the global definitions followed by the hardware definitions where all of the I/O pins for the radio are assigned. Following that there are sections for the major hardware elements of the design in the following order: IR Receiver , the LCD display, the SI-4703 FM receiver, EEPROM and finally miscellaneous functions. Following that are the *setup()* and *loop()* Arduino functions.

In the IR receiver section an instance of IRrecv called *IrReceiver* is declared and assigned the IR_RECEIVER_PIN defined earlier. This assignment connects the IR receiver hardware to the software that will manage it. Next, the IR remote control key codes for the AdaFruit remote (see Figure Three) are declared. I listed all of the key codes available from the remote control not just the ones used in this sketch. Finally, two convenience functions are defined which wrap functions in the IR receiver library for our ease of use.

In the LCD section an instance of LiquidCrystal called *lcd* is defined and it is passed the hardware I/O pins used in this design. Again this connects the hardware to the managing software. A single function is then defined that will clear a specified row of the LCD display when called.

The SI-4703 section is a little more complicated in that we have to extend the functionality of the Si4703_Breakout library. First though, an instance of the library is created called *radio* that is passed the I/O pins used in this design. The Si-4703 FM receiver is controlled using a series of 16 16 bit registers. In a typical operation the registers are read from the chip, values are changed and the updated register values are sent back causing the receiver chip to react. The function *updateStereoIndicator* is a little different in that we are only polling the status of the receiver to determine if a stereo signal is being received to determine if the stereo indicator LED should be lit or not. In the *muteRadio* function we read the registers, set or clear the DMUTE bit depending upon whether or not we are muting and then send the modified registers back to the chip.

The EEPROM section has functions for reading and writing 8 and 16 bit unsigned integers from the EEPROM contained within the Arduino's processor. Values written to the EEPROM survive loss of power and will be available indefinitely until changed.

The Miscellaneous section has functions for displaying the FM station frequency on row 0 of the LCD display, for displaying row 1 messages on the LCD display, for retrieving and storing FM station presets and for processing presets. The *processSetPreset* function waits for a IR key to be pressed on the remote control and then stores the channel/station that is currently being listen to in a corresponding preset in EEPROM.

The Arduino *setup()* function prepares the hardware for operation. Here, the backlight, the power on and the stereo LED control pins are configured as outputs. The IR receiver is enabled, the LCD display is configured for 16 character by 2 row operation, some volume and channel defaults are installed and the EEPROM is prepared for use.

To prevent bogus preset values from being used I wanted to do a one time initialization of the EEPROM setting each preset value storage location to zero. To make sure this is only done once, the code looks for a signature in the first two bytes of the EEPROM. If the signature is not found the two byte signature 0xAA, 0x55 is written and all 10 preset locations are set to zero. If the signature is found, which is usually the case, EEPROM initialization is skipped.

The Arduino *loop()* function is where the radio is controlled. It consists of a large switch statement that is driven by the key codes received from the remote control. The table below details which keys do what.

| **Key** | **Function** |
|---|---|
| Vol- | Mutes the audio |
| Play Pause | Turns the radio off and on |
| Vol+ | Un-mutes the audio |
| Up Arrow | Volume Up |
| Down Arrow | Volume Down |
| Left Arrow | Scan Down for a station/channel |
| Right Arrow | Scan Up for a station/channel |
| Enter Save | Sets up for saving a preset. Tune in the desired station, press Enter Save and then a key 1 .. 9 to save the station as a preset. |
| Keys 1 .. 9 | Tunes station if preset set, otherwise does nothing |

It is important to note that when power is applied to the radio it appears to be off though it really isn't. Instead, the firmware is constantly looking for the Play Pause key code to be received to virtually turn the radio on. A variable called *radioOn* tracks whether the radio is off or on. The processing of all key codes is conditional on this variable as I didn't want the radio responding to commands while it was supposed to be off. Most cases have similar structure. First the *radioOn* variable is check and if the radio is on, a function is performed and a message is written to the LCD display. If *radioOn* is false, the received key code is ignored. Processing of the Play Pause key is the most complex as numerous steps are necessary to virtually turn the radio off or on. The comments in the code should make it clear what is happening.

The cases used for retrieving a preset should be mentioned. There are 9 presets available numbered 1 through 9. When a preset key on the remote is clicked, the corresponding preset is retrieved from the EEPROM and its value is examined. If a value of zero is returned, the preset has never been set so the radio does not respond. If however a non zero value is returned, that channel is set.

Preset 10 is a special case. When the radio is virtually turned off, the channel that was being listened to is written to preset 10. Then, when the radio is virtually turned back on, preset 10 is read and that station/channel is reinstated.

## Conclusions

Building your own FM receiver is cool and easy even if listening to over the air FM radio seems retro.

## Resources

The following sites may be of interest to those seeking more information on the topics described in this article.

Information about the Si-4703 digital FM receiver can be found at the manufactures website at: www.silabs.com. AN230, AN231, AN243 are application notes concerning the Si-470x series parts. AN332 provides example code useful for programmers.

The free Arduino IDE development tool for Windows, OSX and Linux are available at: http://arduino.cc/en/Main/Software.

The IRremote library is available at: github.com/shirriff/Arduino-IRremote. There is a *Download ZIP* button in the lower right side of this page. Click it to get the library.

The Si4703_Breakout library is available at: github.com/infomaniac50/Arduino-Si4703-Library. Again, click the *Download ZIP* button to get the library.

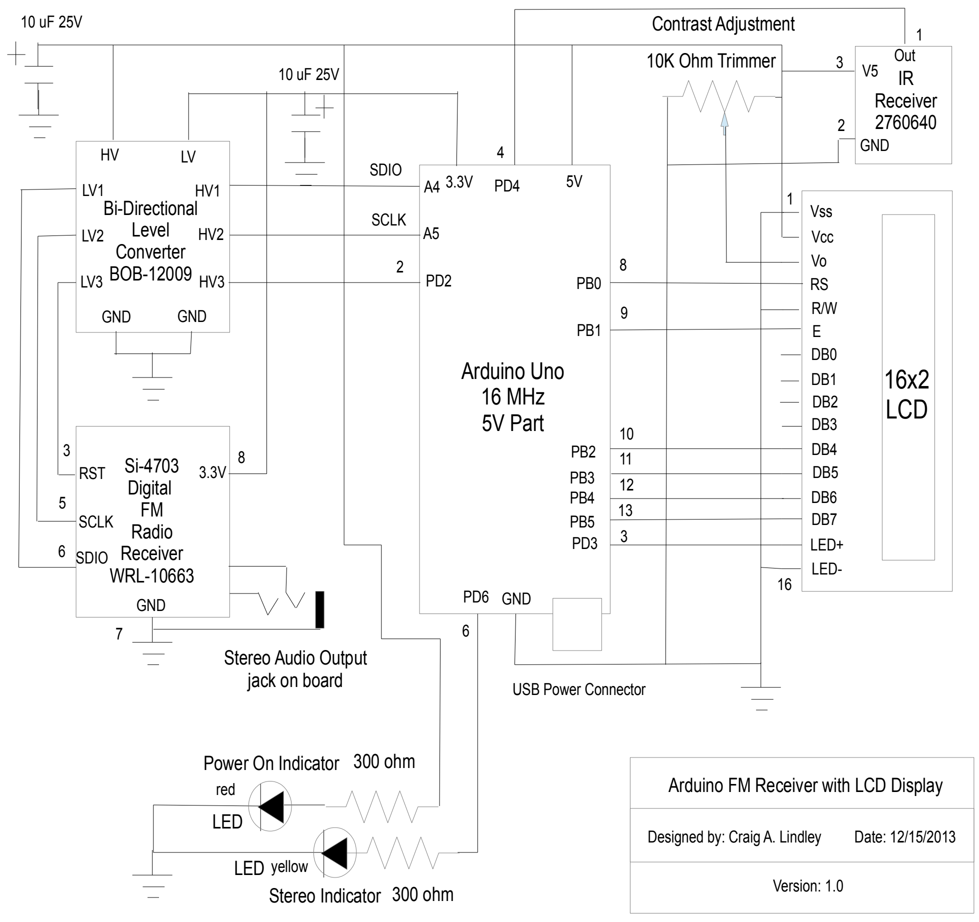
The Arduino sketch described in this article is available from the Nuts and Volts website and is called DigitalFMRadio.ino.

An AmForth version of the firmware is also available. Contract the author for details.

Figure One
Parts List

| Item | Part Number/Description | Source |
|---|---|---|
| Arduino Uno | 16Mhz 5 volt part | SparkFun, AdaFruit, Radio Shack, eBay |
| Bi-directional Level Converter | BOB-12009 | SparkFun |
| Evaluation Board for Si4703 FM Tuner | WRL-10663 | SparkFun |
| 16x2 line LCD display | Any LCD noted to be Arduino compatible should work. | SparkFun, AdaFruit, eBay |
| IR Receiver | #2760640 | Radio Shack |
| Red LED | many | anywhere |
| Yellow LED | many | anywhere |
| 2 x 300 ohm ¼ w resistor | many | anywhere |
| 10K ohm 10 turn trimmer | many | anywhere |
| 2 x 10 uF @ 25V capacitor | many | anywhere |
| Mini Remote Control | ID: 389 | AdaFruit |
| USB cable for Arduino to USB power supply connection | many | SparkFun, AdaFruit, Radio Shack, eBay |
| USB power supply 500 mA or greater | many | SparkFun, AdaFruit, Radio Shack, eBay |
| .1” male break away header pins for Arduino connectors | many | SparkFun, AdaFruit, eBay |
| Wire, solder, packaging, etc | | |

Figure Two
Schematic Diagram



| Arduino FM Receiver with LCD Display | |
|---|---|
| Designed by: Craig A. Lindley | Date: 12/15/2013 |
| Version: 1.0 | |

Figure Three
IR Mini Remote Control

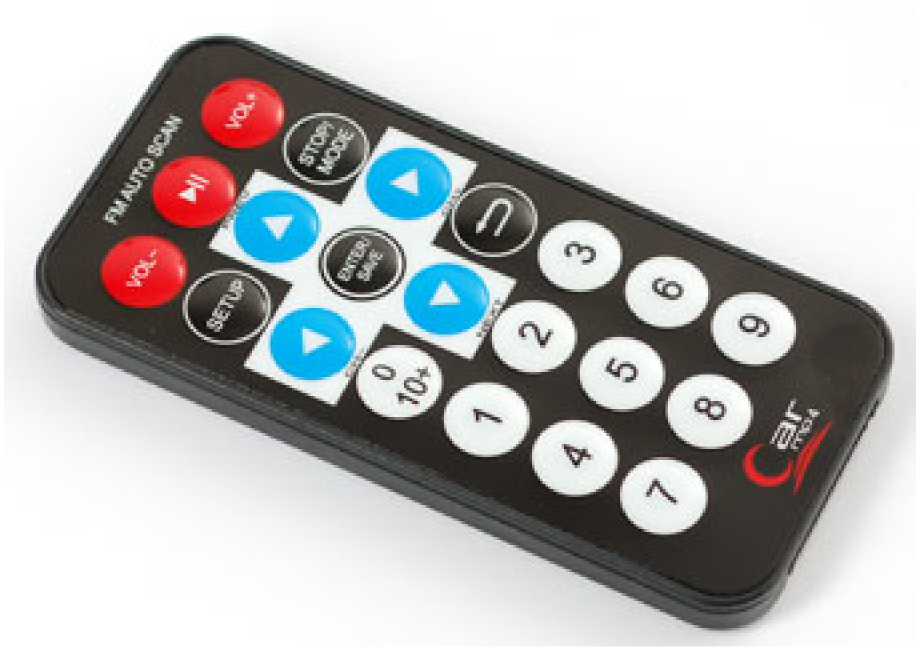


Photo One
Three Components – LCD, Arduino Uno and Receiver Board

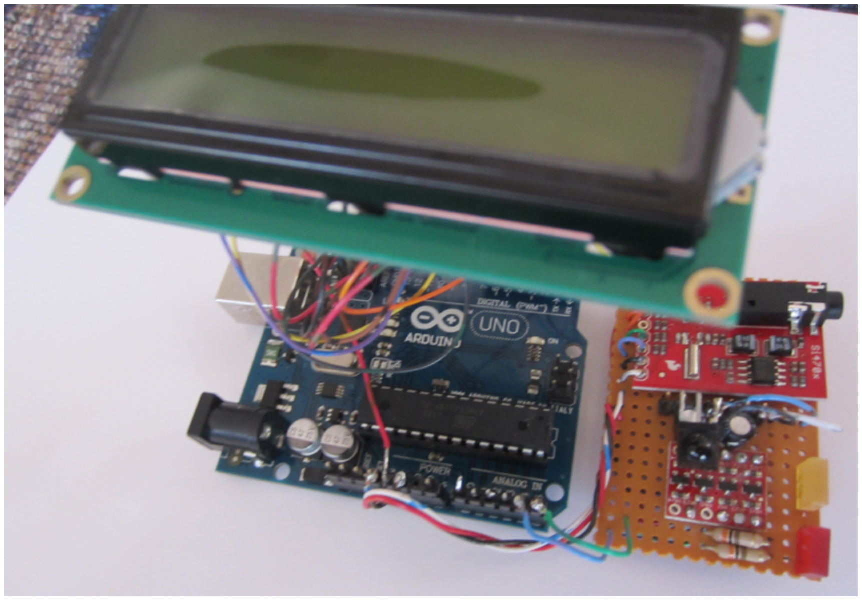


Photo Two
Receiver Board Close Up

At the top is the Si-4703 evaluation board with the black 1/8" stereo output jack, in the middle is the IR receiver facing upward along with some filter caps, towards the bottom is the four channel level converter, the yellow rectangular LED is the stereo indicator and the red LED is the power on indicator

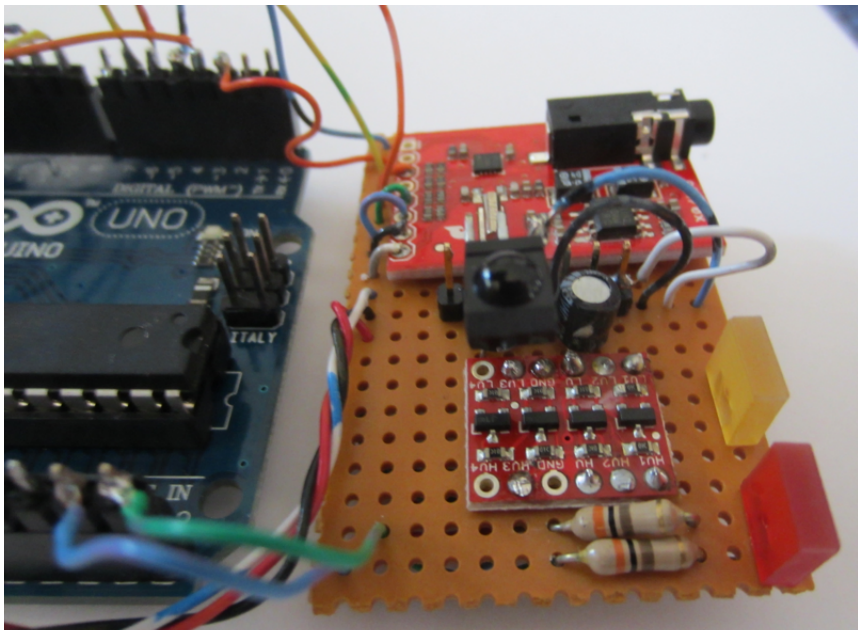


Photo Three
The Working Receiver in its prototype packaging

On the left is the USB cable powering the receiver. On the right I have my headphones plugged onto the receiver. The contrast adjusting trimmer can be seen on the left top of the LCD display.