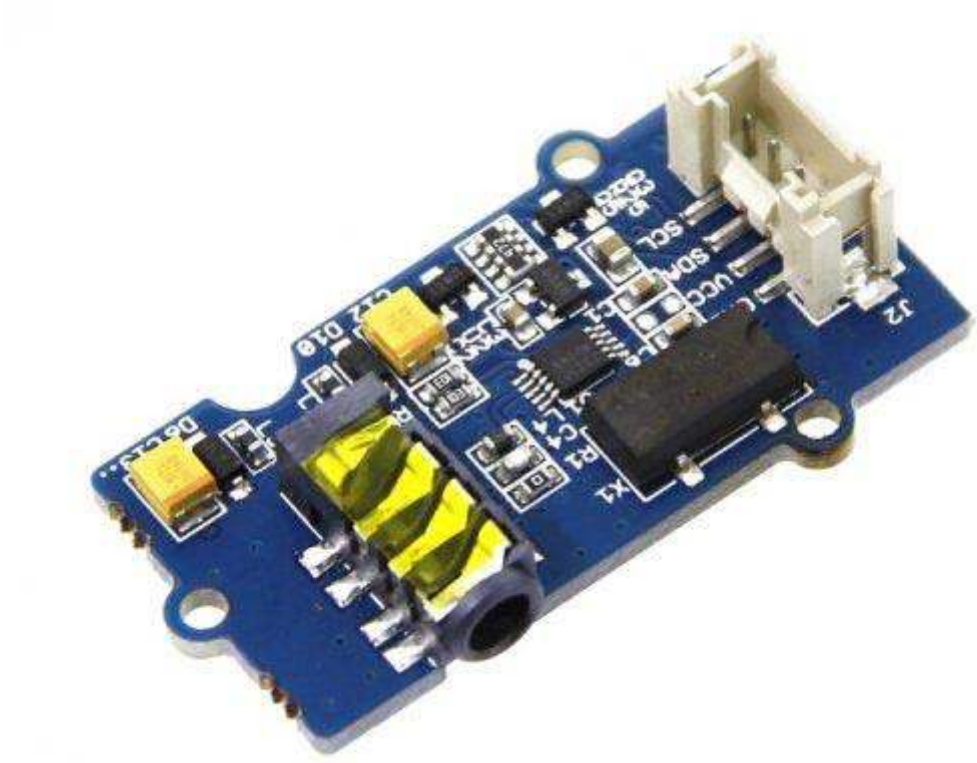
Grove - I2C FM Receiver



Grove - I2C FM Receiver is a wideband FM receiver module, this module is based on RDA5807M. The RDA5807M series are the latest generation single-chip broadcast FM stereo radio tuner with fully integrated synthesizer. The RDA5807M series have a powerful low-IF digital audio processor. The Grove - I2C FM Receiver has a headset jack, so it can connect to earphones or audio.

Version

Version	Change	Release date
Grove - I2C FM Receiver v1.0	Initial	May 18, 2017
Grove - I2C FM Receiver v1.1	Change some components to make the board more stable	April 17, 2018

Features

- Grove interface
- Supports worldwide frequency band: 50 - 115MHz
- Support RDS/RBDS
- Lower power consumption
- Headset interface
- Digital auto gain control
- Input voltage: 3.3V - 5V

Tip
More details about Grove modules please refer to Grove System

Platforms Supported

Arduino	Raspberry Pi	BeagleBone	Wio	LinkIt ONE
				

Caution
The platforms mentioned above as supported is/are an indication of the module's hardware or theoritical compatibility. We only provide software library or code examples for Arduino platform in most cases. It is not possible to provide software library / demo code for all possible MCU platforms. Hence, users have to write their own software library.

Getting Started

Note
If this is the first time you work with Arduino, we strongly recommend you to see Getting Started with Arduinobefore the start.

Play With Arduino

Hardware

Materials required

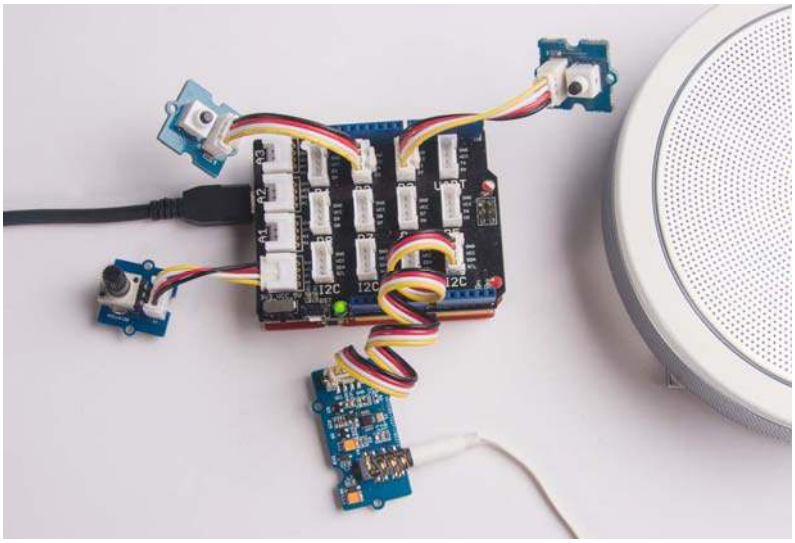
Seeeduino V4.2	Base Shield	Grove - I2C FM Receiver v1.1	Grove - Button x 2	Grove - Rotary Angle Sensor
				

Note

1 Please plug the USB cable gently, otherwise you may damage the port. Please use the USB cable with 4 wires inside, the 2 wires cable can't transfer data. If you are not sure about the wire you have, you can click here to buy

2 Each Grove module comes with a Grove cable when you buy. In case you lose the Grove cable, you can click here to buy

- **Step 1.** Connect Grove - I2C FM Receiver to port **IIC** of Grove-Base Shield.
- **Step 2.** Connect Grove - Button 1 to **D2** port and connect Grove - Button 2 to **D3** port.
- **Step 3.** Plug Grove - Base Shield into Seeeduino.
- **Step 4.** Connect Seeeduino to PC via a USB cable.



Note
If we don't have Grove Base Shield, We also can directly connect Grove - Temperature and Humidity Sensor Pro to Seeeduino as below.

Seeeduino	Grove - I2C FM Receiver v1.1
5V	Red
GND	Black
SDA	White
SCL	Yellow
Seeeduino	Grove - Button 1
5V	Red
GND	Black
Null	White
D2	Yellow
Seeeduino	Grove - Button 2
5V	Red
GND	Black
Null	White
D3	Yellow
Seeeduino	Grove - Rotary Angle Sensor
5V	Red
GND	Black
Null	White
A0	Yellow

Software

- **Step 1.** Download Grove-I2C FM Receiver library and then install library.
- **Step 2.** Refer to How to install library to install library for Arduino.
- **Step 3.** Copy the following code into you Arduino IDE, then save and compile.

Copied to clipboard

```
/*
 * I2C_FM.ino
 * Demo code for the Grove-I2C_FM_Receiver module
 *
 * Copyright (c) 2012 seeed technology inc.
 * Author     : Jack Shao (jacky.shaoxg@gmail.com)
 * Create Time: Jul 2014
 * Change Log :
 *
 * The MIT License (MIT)
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */
//
/*
 * Modifications to the I2C_FM.ino by Mel Patrick - Wabbit Wanch Design
 * Modified routines for scanning UP or DOWN through the FM band
 * Modified routine to test for signal strength of received station
 * Modified routines to support bass boost and MONO signal
 * RSSI, read it too soon after setting a station and you get a small value
 * so it's better to wait a bit (50ms) and try it. minSignalStrength will
 * skip locking on a station with a weak signal (you could set the MONO bit) to get
 * better reception on these stations.
 */
#include <Arduino.h>
#include <Wire.h>
#include <EEPROM.h>

#define BTNUP         2// used for seeking UP (normally CLOSED push button)
```

```
#define VOL_POT       A0// volume POT LOG taper 10K
#define BTNDN         3// used for seeking DOWN (normally CLOSED push button)

uint16_t gChipID = 0;
uint8_t RDA5807P_REGW[10];

#define I2C_ADDR      0x10

#define READ          1
#define WRITE         0

#define ADRW          0x20
#define ADRR          0x21
//

//#define              _SHARE_CRYSTAL_24MHz_
//#define              _SHARE_CRYSTAL_12MHz_
#define              _SHARE_CRYSTAL_32KHz_
//#define              _FM_STEP_50K_

//5807M,5807FP,5807NN,5807NP
uint8_t RDA5807N_initialization_reg[]={
#if defined(_SHARE_CRYSTAL_24MHz_)
  0xC4, 0x51, //02H:
#elif defined(_SHARE_CRYSTAL_12MHz_)
  0xC4, 0x11, //02H:
#elif defined(_SHARE_CRYSTAL_32KHz_)
  0xC4, 0x01,//change 01 to 05 enables the RDS/RBDS
#else
  0xC0, 0x01,
#endif
  0x00, 0x00,
  0x04, 0x00,
  0xC3, 0xad,  //05h
  0x60, 0x00,
  0x42, 0x12,
  0x00, 0x00,
  0x00, 0x00,
  0x00, 0x00,  //0x0ah
  0x00, 0x00,
  0x00, 0x00,
  0x00, 0x00,
  0x00, 0x00,
  0x00, 0x00,
  0x00, 0x00,  //0x10h
  0x00, 0x19,
  0x2a, 0x11,
  0xB0, 0x42,
  0x2A, 0x11,  //
  0xb8, 0x31,  //0x15h
  0xc0, 0x00,
  0x2a, 0x91,
  0x94, 0x00,
  0x00, 0xa8,
  0xc4, 0x00,  //0x1ah
  0xF7, 0xcF,
```

```
  0x12, 0x14,  //0x1ch
  0x80, 0x6F,
  0x46, 0x08,
  0x00, 0x86,  //10000110
  0x06, 0x61,  //0x20H
  0x00, 0x00,
  0x10, 0x9E,
  0x23, 0xC8,
  0x04, 0x06,
  0x0E, 0x1C,  //0x25H     //0x04 0x08
};

int16_t freq = 10110;
uint16_t vol = 1;
//
// added items - Mel
boolean bassBit = true;// bass boost
boolean monoBit = false;// force MONO not stereo
const boolean seekUP = true;
const boolean seekDN = false;
uint8_t minSignalStrength = 36;// anything below this probably set a MONO flag for better reception
uint8_t signalStrength;
long previousMillis = 0;// last time the function was called
long interval = 2000;// interval for the signal level function (2 seconds)
int8_t stationStep = 10;// kHz steps bewteen the stations (North America = 10)
boolean hasVolumePot = true;// flag if you have a POT attached or not
//
void setup()
{
  Wire.begin();
  loadDefaults();// load any defaults from previous radio settings
  Serial.begin(9600);
  Serial.println("Started");
  //=======================
  //rda5807 power on
  RDA5807P_PowerOnReset();
  RDA5807P_SetMute(false);

  //=======================
  pinMode(BTNUP, INPUT_PULLUP);
  pinMode(VOL_POT, INPUT);
  pinMode(BTNDN, INPUT_PULLUP);
  //=======================
  RDA5807P_SetVolumeLevel(vol);// use this if you don't have a POT for volume attached (0-15)
  RDA5807P_SetFreq( freq );
}

void loop()
{
  unsigned long currentMillis = millis();

  if(currentMillis - previousMillis > interval) {
    // save the last time you blinked the LED
    previousMillis = currentMillis;
    showSignalStrength();
  }
```

```
  //
  if (digitalRead(BTNUP) == 1)
  {
    delay(100);
    if (digitalRead(BTNUP) == 1)
      fmSeek(seekUP);
    while(digitalRead(BTNUP) == 1);
  }
  if (digitalRead(BTNDN) == 1)
  {
    delay(100);
    if (digitalRead(BTNDN) == 1)
      fmSeek(seekDN);
    while(digitalRead(BTNDN) == 1);
  }
  if (hasVolumePot == true) setVolume();// use this to read the POT
}
//
void setVolume() {
  unsigned int temp_vol;
  temp_vol = analogRead( VOL_POT );
  if (abs(temp_vol - vol)>5)
  {
    if (vol != temp_vol) {// don't bother changing the volume if unless the pot moves
      vol = temp_vol;
      unsigned char hex_vol = map(vol, 0, 1023, 0, 0xf);
      RDA5807P_SetVolumeLevel(hex_vol);
      saveDefaults();// save new volume to EEPROM
    }
  }
}
//
void fmSeek(boolean theDir) {
  int signalStrength;
  if (!theDir) {
    Serial.println("Start seeking down...");
  }
  else
  {
    Serial.println("Start seeking up...");
  }
  do {
    do{
      if (theDir == seekUP) {
        freq += stationStep;
      }
      else
      {
        freq -= stationStep;
      }
      if (freq > 10800) freq = 8800;
      if (freq < 8800) freq = 10800;
      //Serial.println(freq);
    }
    while(!RDA5807P_ValidStop(freq));
    delay(50);
```

```
    signalStrength = RDA5807P_GetSigLvl(freq);// max is 63 according to Data sheet, but I've seen more
  }
  while (signalStrength < minSignalStrength);// minimum signal strength, keep looking
  showRadioStation();
  saveDefaults();// save new station selection to EEPROM
}
//
void showRadioStation() {
  Serial.print("Stable Freq:");
  Serial.print(((float)freq)/100.0f);
  Serial.println("MHz");
}
//
void showSignalStrength() {
  signalStrength = RDA5807P_GetSigLvl(freq);// max is 63...as noted
  Serial.print("Signal Strength: ");
  Serial.println(signalStrength);
}

//=============================================================
// FM functions
//=============================================================
unsigned char OperationRDAFM_2w(unsigned char operation, unsigned char *data, int numBytes)
{
  if(operation == READ)
  {
    Wire.requestFrom(I2C_ADDR, numBytes);
    for(int i=0;i<numBytes;i++)
    {
      *data++ = Wire.read();
    }
  }
  else
  {
    Wire.beginTransmission(I2C_ADDR);
    for(int i=0;i<numBytes;i++)
    {
      Wire.write(*data++);
    }
    Wire.endTransmission();
  }
  return 0;
}


/**
 * @brief Reset RDA5807P while power on RDA5807P
 * @author RDA RDA Ri'an Zeng
 * @date 2008-11-05
 * @param void
 * @return void
 * @retval
 */
void  RDA5807P_PowerOnReset(void)
{
  RDA5807P_Intialization();
```

```
}

/**
 * @brief RDA5807P power off function
 * @author RDA Ri'an Zeng
 * @date 2008-11-05
 * @param void
 * @return void
 * @retval
 */
void  RDA5807P_PowerOffProc(void)
{
  RDA5807P_REGW[1] &= (~1);
  OperationRDAFM_2w(WRITE, &(RDA5807P_REGW[0]), 2);
}

/**
 * @brief Set RDA5807P into mute mode
 * @author RDA Ri'an Zeng
 * @date 2008-11-05
 * @param bool mute: if mute is true,then set mute; if mute is false,then set no mute
 * @return void
 * @retval
 */
void RDA5807P_SetMute(boolean mute)
{
  if(mute)
    RDA5807P_REGW[0] &=  ~(1<<6);
  else
    RDA5807P_REGW[0] |= 1<<6;
  RDA5807P_REGW[0] |= monoBit<<5;
  RDA5807P_REGW[0] |= bassBit<<4;
  OperationRDAFM_2w(WRITE, &(RDA5807P_REGW[0]), 2);//RDA5807M_REGW
  delay(50);    //Dealy 50 ms
}
//
/************************************************
 * @brief Set frequency function
 * @author RDA Ri'an Zeng
 * @date 2008-11-05
 * @param int16_t curFreq:frequency value
 * @return void
 * @retval
 ************************************************/
void RDA5807P_SetFreq(int16_t curFreq)
{
  uint16_t curChan;
  curChan=RDA5807P_FreqToChan(curFreq);

  if((curFreq >= 6500)&&(curFreq < 7600))
  {
    RDA5807P_REGW[3] = 0x0c;
  }
  else if((curFreq >= 7600)&&(curFreq < 10800))
  {
```

```
    RDA5807P_REGW[3] = 0x08;// sets the BAND bits (00xx = 87-108, 01xx=76-91, 10xx=76-108,
11xx=65-76
    // for north america this must be set to 10xx for some unknown reason
  }
  //SetNoMute
  RDA5807P_REGW[0] |= 1<<6;
  RDA5807P_REGW[0] |= monoBit<<5;
  RDA5807P_REGW[0] |= bassBit<<4;
  //handleBits();
  RDA5807P_REGW[2]=curChan>>2;
  RDA5807P_REGW[3]=(((curChan&0x0003)<<6)|0x10) | (RDA5807P_REGW[3]&0x0f);    //set tune bit

  OperationRDAFM_2w(WRITE, &(RDA5807P_REGW[0]), 4);
  delay(50);     //Delay five ms
  showRadioStation();
}
//
/**
 * @brief Station judge for auto search
 * @In auto search mode,uses this function to judge the frequency if has a station
 * @author RDA Ri'an Zeng
 * @date 2008-11-05
 * @param int16_t freq:frequency value
 * @return bool: if return true,the frequency has a true station;otherwise doesn't have a station
 * @retval
 */
boolean RDA5807P_ValidStop(int freq)
{
  uint8_t RDA5807P_reg_data[4]={
    0                                       };
  uint8_t falseStation = 0;
  uint8_t i=0;
  uint16_t curChan;

  if((freq >= 6500)&&(freq < 7600))
  {
    RDA5807P_REGW[3] = 0x0c;
  }
  else if((freq >= 7600)&&(freq < 10800))
  {
    RDA5807P_REGW[3] = 0x08;// sets the BAND bits (00xx = 87-108, 01xx=76-91, 10xx=76-108,
11xx=65-76
    // for north america this must be set to 10xx for some unknown reason
  }
  curChan=RDA5807P_FreqToChan(freq);
  //SetNoMute bit 9 is seek direction (0=seek down, 1=seek up).
  //02H 14
  RDA5807P_REGW[0] |=   1<<6;// reg zero is bits 15 to bit 8 (this shifts to bit 14)
  RDA5807P_REGW[0] |= monoBit<<5;
  RDA5807P_REGW[0] |= bassBit<<4;
  //handleBits();
  RDA5807P_reg_data[0]=RDA5807P_REGW[0];
  RDA5807P_reg_data[1]=RDA5807P_REGW[1];
  RDA5807P_reg_data[2]=curChan>>2;//03H 15:8 CHAN
  RDA5807P_reg_data[3]=(((curChan&0x0003)<<6)|0x10) | (RDA5807P_REGW[3]&0x0f);//
  OperationRDAFM_2w(WRITE,&(RDA5807P_reg_data[0]), 4);
```

```
  delay(50);    //Dealy 25 ms

  if (0x5808 == gChipID)
    OperationRDAFM_2w(READ,&(RDA5807P_reg_data[0]), 4); //
  else
  {
    do
    {
      i++;
      if(i>5) return 0;

      delay(30);
      //read REG0A&0B
      OperationRDAFM_2w(READ,&(RDA5807P_reg_data[0]), 4);
    }
    while((RDA5807P_reg_data[0]&0x40)==0);
  }

  //check FM_TRUE
  if((RDA5807P_reg_data[2] &0x01)==0) falseStation=1;//0B 8  FM TRUE

  if(freq==9600) falseStation=1;// North America - if scanning DOWN, the radio will lock on 9600 for some
reason!
  delay(50);
  if (falseStation==1)
    return 0;
  else
    return 1;
}

/**
 * @brief Get the signal level(RSSI) of the current frequency
 * @author RDA Ri'an Zeng
 * @date 2008-11-05
 * @param int16_t curf:frequency value
 * @return uint8_t: the signal level(RSSI)
 * @retval
 */
uint8_t RDA5807P_GetSigLvl( int16_t curf )
{
  uint8_t RDA5807P_reg_data[4]={
    0                                       };
  OperationRDAFM_2w(READ,&(RDA5807P_reg_data[0]), 4);
  delay(50);    //Delay 50 ms
  return  (RDA5807P_reg_data[2]>>1);  /*??rssi*/
}

/**
 * @brief Set FM volume
 * @It has better use the system volume operation to replace this function
 * @author RDA Ri'an Zeng
 * @date 2008-11-05
 * @param uint8_t level: volume value
 * @return void
 * @retval
```

```
 */
void RDA5807P_SetVolumeLevel(uint8_t level)
{
  uint8_t RDA5807P_reg_data[8];
  uint8_t i = 0;

  for (i=0;i<8;i++)
    RDA5807P_reg_data[i] = RDA5807P_REGW[i];

  RDA5807P_reg_data[7]=(( RDA5807P_REGW[7] & 0xf0 ) | (level & 0x0f));

  RDA5807P_reg_data[3] &= (~(0x10));//disable tune

  OperationRDAFM_2w(WRITE, &(RDA5807P_reg_data[0]), 8);
  delay(50);    //Dealy 50 ms
}

/**
 * @brief Initialize RDA5807P
 * @author RDA Ri'an Zeng
 * @date 2008-11-05
 * @param void
 * @return bool:if true,the operation is successful;otherwise is failed
 * @retval
 **/
boolean  RDA5807P_Intialization(void)
{
  uint8_t error_ind = 0;
  uint8_t RDA5807P_REGR[10]={
    0x0                                         };
  uint8_t i = 0;

  RDA5807P_REGW[0] = 0x00;
  RDA5807P_REGW[0] |= monoBit<<5;
  RDA5807P_REGW[0] |= bassBit<<4;
  RDA5807P_REGW[1] = 0x02;

  error_ind = OperationRDAFM_2w(WRITE, (uint8_t *)&RDA5807P_REGW[0], 2);//soft reset
  delay(50);

  error_ind = OperationRDAFM_2w(READ, (uint8_t *)&RDA5807P_REGR[0], 10);
  delay(50);

  gChipID = RDA5807P_REGR[8];
  gChipID = ((gChipID << 8) | RDA5807P_REGR[9]);

  Serial.print("Chip ID: 0x");
  Serial.println(gChipID, HEX);

  for (i=0;i<8;i++) {
    RDA5807P_REGW[i] = RDA5807N_initialization_reg[i];
  }

  error_ind = OperationRDAFM_2w(WRITE, (uint8_t *)&RDA5807N_initialization_reg[0], 2); //power up
  delay(600);
  //Serial.println(sizeof(RDA5807N_initialization_reg));
```

```
  error_ind = OperationRDAFM_2w(WRITE, (uint8_t *)&RDA5807N_initialization_reg[0],
sizeof(RDA5807N_initialization_reg));

  delay(50);        //Dealy 50 ms

  if (error_ind )
    return 0;
  else
    return 1;
}
//
/**
 * @brief Cover the frequency to channel value
 * @author RDA Ri'an Zeng
 * @date 2008-11-05
 * @param uint16 frequency:covered frequency
 * @return uint16: channel value
 * @retval
 * In the United States, frequency-modulated broadcasting stations operate in a frequency band extending
from 87.8 MHz to 108.0 MHz,
 * for a total of 20.2 MHz. It is divided into 101 channels, each 0.2 MHz wide, designated "channel 200"
through "channel 300."
 * In actual practice, no one (except the FCC) uses these channel numbers; the frequencies are used
instead.
 */
uint16_t RDA5807P_FreqToChan(uint16_t frequency) {
  uint8_t channelSpacing = 10;
  uint16_t channel = 0;

  if((frequency >= 6500)&&(frequency < 7600))
  {
    channel = (frequency - 6500)/channelSpacing;
  }
  else if((frequency >= 7600)&&(frequency < 10800))
  {
    channel = (frequency - 7600)/channelSpacing;
  }
  return (channel);
}
//
void loadDefaults() {
  char myCode[9] = "Grove_FM";
  char myInit[9] = "blank123";
  /*
  * byte map in EEPROM
   * 8, 9 the default frequency for a reboot
   * 10, 11 current preset volume of the radio (used if no pot is attached)
   */
  for (int i=0; i < 8; i++) {
    myInit[i] = EEPROM.read(i);// read out to see if the thing is INITIALIZED
  }
  if (strcmp(myCode, myInit) == 0) {// if this is ZERO (we previously wrote some), then read the values
    freq = epReadINT(8);// read back the INT for frequency from eeprom 8 and 9 (two bytes for an INT)
    if (!hasVolumePot) vol = epReadINT(10);// read back the volume setting but don't use it unless flag is
false
  }
```

```
  else// we don't have any defaults, so we have to save some first
  {
    for (int i=0; i < 8; i++) {
      EEPROM.write(i, myCode[i]);// write this to EEPROM to show we have it saved
    }
    saveDefaults();// write the current default settings
  }
}
//
void saveDefaults() {
  epWriteINT(8, freq);// write the two bytes for INT for a reboot
  epWriteINT(10, vol);// write the current volume POT setting
}
//
void epWriteINT(int where, int theVal) {
  union uData
  {
    byte stuff[2];
    int f1;// 2 bytes of memory
  }
  u;
  u.f1 = theVal;// copy into the union
  for (int j=0; j < 2; j++) {// now we have to write out 2 bytes of memory
    EEPROM.write(where + j, u.stuff[j]);// write it to EEPROM
  }
}
//
long epReadINT(int where) {
  union uData
  {
    byte stuff[2];
    int f1;// 2 bytes of memory
  }
  u;
  for (int j=0; j < 2; j++) {
    u.stuff[j]=EEPROM.read(where + j);// read back the 2 bytes at this memory location
  }
  return u.f1;
}
//
void epWriteLong(int where, long theVal) {
  union uData
  {
    byte stuff[4];
    long f1;// 4 bytes of memory
  }
  u;
  u.f1 = theVal;// copy into the union
  for (int j=0; j < 4; j++) {// now we have to write out 4 bytes of memory
    EEPROM.write(where + j, u.stuff[j]);// write it to EEPROM
  }
}
//
long epReadLong(int where) {
  union uData
  {
```

```
    byte stuff[4];
    long f1;// 4 bytes of memory
  }
  u;
  for (int j=0; j < 4; j++) {
    u.stuff[j]=EEPROM.read(where + j);// read back the 4 bytes to this memory location
  }
  return u.f1;
}
```

- **Step 4.** Upload the demo. If you do not know how to upload the code, please check How to upload code.
- **Step 5.** Open the **Serial Monitor** of Arduino IDE by click **Tool-> Serial Monitor**. Or tap the Ctrl + Shift + M key at the same time. if every thing goes well, you will get the result.

The result should be like:

```
 1Started
 2Chip ID: 0x5808
 3Stable Freq:102.60MHz
 4Signal Strength: 46
 5Signal Strength: 46
 6Signal Strength: 45
 7Signal Strength: 45
 8Signal Strength: 45
 9Signal Strength: 45
10Start seeking down...
11Stable Freq:94.00MHz
12Signal Strength: 44
13Signal Strength: 51
14Signal Strength: 51
15Signal Strength: 50
16Signal Strength: 50
17Signal Strength: 51
```

Now you can hear the FM station, and you can press the Grove- button 1 and Grove- button 2 to change the radio stations. And you can rotate the Grove - Rotary Angle Sensor to adjust the volume.

Have fun~

Resources

- **[Zip]** Grove - I2C FM Receiver v1.0 Eagle File

https://raw.githubusercontent.com/SeeedDocument/GroveI2C_FM_Receiver/master/res/Grove_I2C_FM_Receiver_v1.0.zip

- **[PDF]** RDA5807 Datasheet

https://github.com/SeeedDocument/Grove-I2C_FM_Receiver_v1.1/raw/master/res/RDA5807%20Datasheet.pdf

Tech Support

Please do not hesitate to contact techsupport@seeed.cc if you have any technical issue. Or submit the issue into our forum.