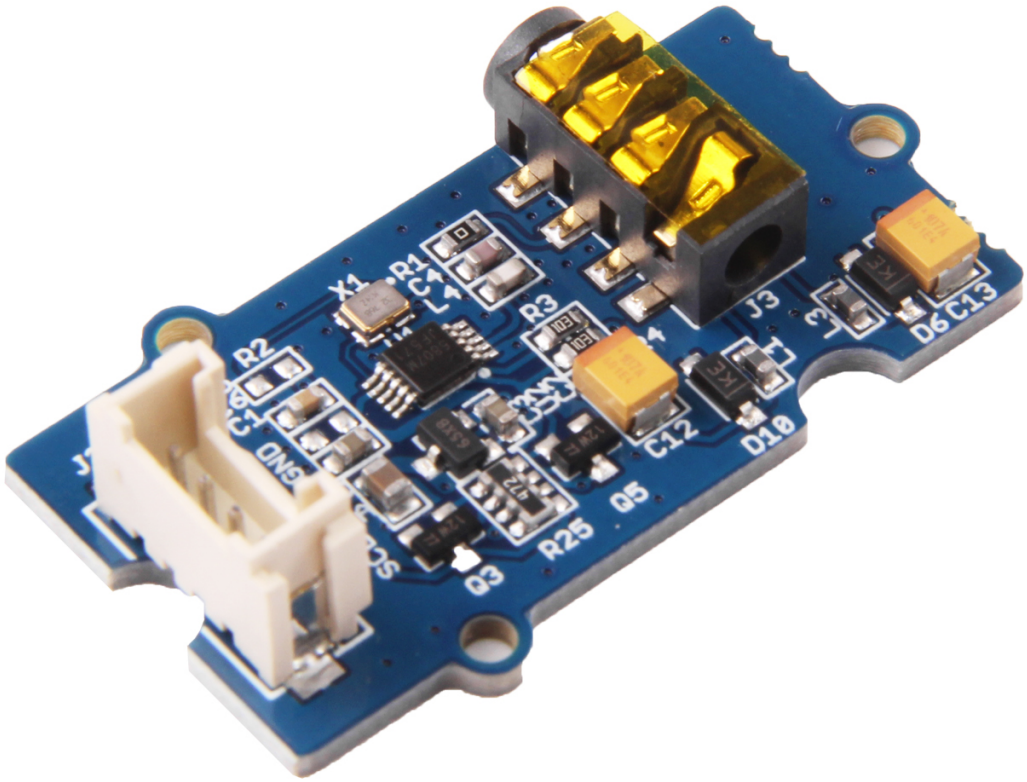
Grove - I2C FM Receiver v1.1



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.

Get One Now

[https://www.seeedstudio.com/Grove-FM-Receiver-V1.1-p-3076.html]

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 [https://wiki.seeedstudio.com/Grove_System/]

Platforms Supported

Arduino	Raspberry Pi	

Caution
The platforms mentioned above as supported is/are an indication of the module's software 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.

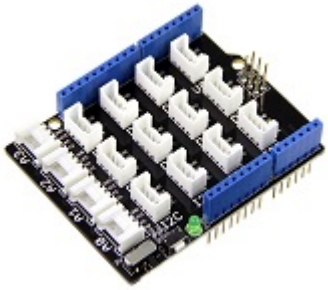
Getting Started

Note
If this is the first time you work with Arduino, we strongly recommend you to see Getting Started with Arduino [https://wiki.seeedstudio.com/Getting_Started_with_Arduino/] before the start.

Play With Arduino

Hardware

Materials required

Seeeduino V4.2	Base Shield	Grove - I2C FM
		
Get One Now [https://www.seeedstudio.com/Seeeduino-V4.2-p-2517.html]	Get One Now [https://www.seeedstudio.com/Base-Shield-V2-p-1378.html]	Get One Now [https://www.s FM-Receiver-V

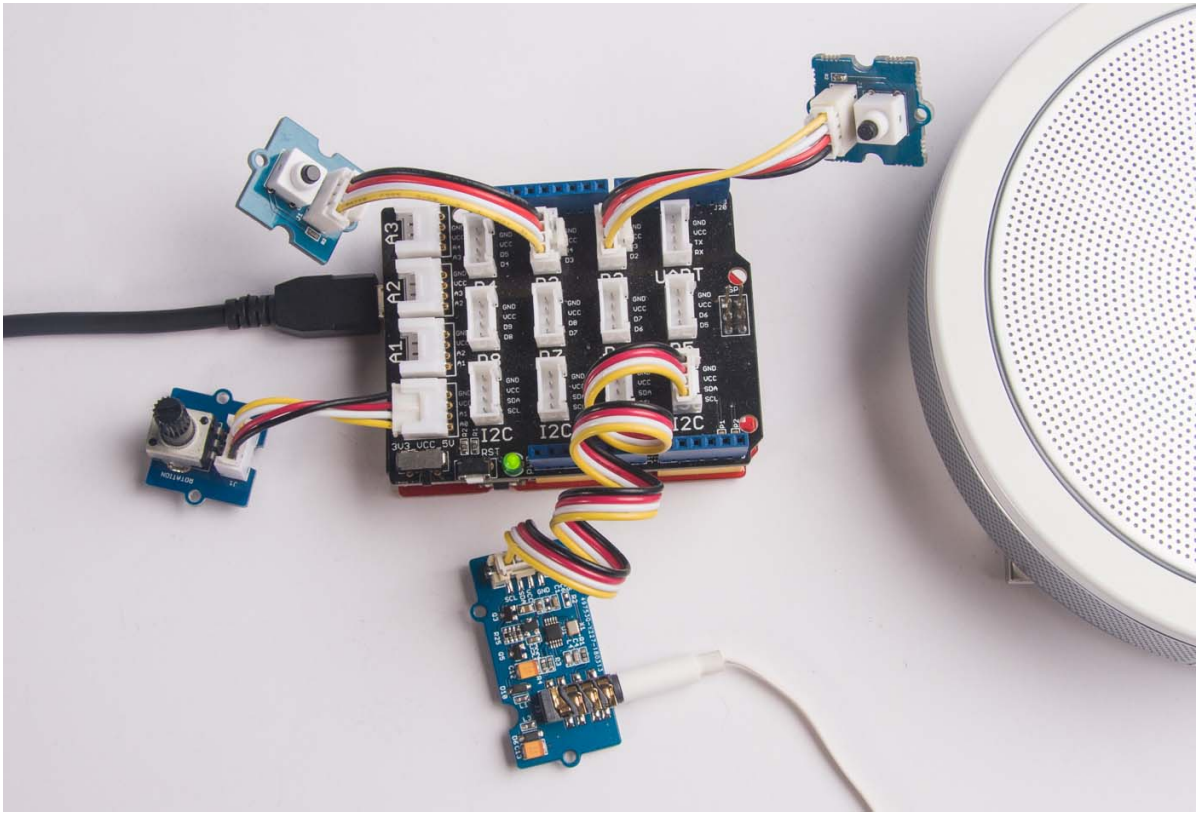
Note

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 [https://www.seeedstudio.com/Micro-USB-Cable-48cm-p-1475.html] 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 [https://www.seeedstudio.com/Grove-Universal-4-Pin-Buckled-20cm-Cable-%285-PCs-pack%29-p-936.html] to buy

- **Step 1.** Connect Grove - I2C FM Receiver v1.1 to port **IIC** of Grove-Base Shield. Plug your earphone or speaker into the 3.5mm jack of Grove - I2C FM Receiver v1.1.
- **Step 2.** Connect Grove - Button 1 to **D2** port and connect Grove - Button 2 to **D3** port.
- **Step 3.** Connect Grove - Rotary Angle Sensor to **A0** port of Grove-Base Shield.
- **Step 4.** Plug Grove - Base Shield into Seeeduino.
- **Step 5.** Plug the earphone or speaker to the 3.5mm jack of Grove - I2C FM Receiver v1.1.

- **Step 6.** 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 [https://github.com/mathertel/Radio/] and then install library.

- **Step 2.** Refer to How to install library [https://wiki.seeedstudio.com/How_to_install_Arduino_Library] to install library for Arduino.
- **Step 3.** Copy the following code into you Arduino IDE, then save and compile.

```
/*
 * 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 obtainir
 * of this software and associated documentation files (the "Software",
 * in the Software without restriction, including without limitation th
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or
 * copies of the Software, and to permit persons to whom the Software i
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be inclu
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABIL
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SH
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTH
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARIS
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALIN
 * 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 v
 * so it's better to wait a bit (50ms) and try it. minSignalStrength wi
 * skip locking on a station with a weak signal (you could set the MONO
```

```
 * 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 butto

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 M
uint8_t signalStrength;
long previousMillis = 0;     // last time the function was called
long interval = 2000;        // interval for the signal level function
int8_t stationStep = 10;     // kHz steps bewteen the stations (North A
boolean hasVolumePot = true; // flag if you have a POT attached or not
//
```

```
129 void setup()
130 {
131     Wire.begin();
132     loadDefaults(); // load any defaults from previous radio settings
133     Serial.begin(9600);
134     Serial.println("Started");
135     //======================
136     //rda5807 power on
137     RDA5807P_PowerOnReset();
138     RDA5807P_SetMute(false);
139 
140     //======================
141     pinMode(BTNUP, INPUT_PULLUP);
142     pinMode(VOL_POT, INPUT);
143     pinMode(BTNDN, INPUT_PULLUP);
144     //======================
145     RDA5807P_SetVolumeLevel(vol); // use this if you don't have a POT j
146     RDA5807P_SetFreq(freq);
147 }
148 
149 void loop()
150 {
151     unsigned long currentMillis = millis();
152 
153     if (currentMillis - previousMillis > interval)
154     {
155         // save the last time you blinked the LED
156         previousMillis = currentMillis;
157         showSignalStrength();
158     }
159     //
160     if (digitalRead(BTNUP) == 1)
161     {
162         delay(100);
163         if (digitalRead(BTNUP) == 1)
164             fmSeek(seekUP);
165         while (digitalRead(BTNUP) == 1)
166             ;
167     }
168     if (digitalRead(BTNDN) == 1)
169     {
170         delay(100);
171         if (digitalRead(BTNDN) == 1)
172             fmSeek(seekDN);
173         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 accordi
    } while (signalStrength < minSignalStrength);  // minimum signal si
    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
{
    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++)
        {
```

```
264             Wire.write(*data++);
265         }
266         Wire.endTransmission();
267     }
268     return 0;
269 }
270 
271 /**
272  * @brief Reset RDA5807P while power on RDA5807P
273  * @author RDA RDA Ri'an Zeng
274  * @date 2008-11-05
275  * @param void
276  * @return void
277  * @retval
278  */
279 void RDA5807P_PowerOnReset(void)
280 {
281     RDA5807P_Intialization();
282 }
283 
284 /**
285  * @brief RDA5807P power off function
286  * @author RDA Ri'an Zeng
287  * @date 2008-11-05
288  * @param void
289  * @return void
290  * @retval
291  */
292 void RDA5807P_PowerOffProc(void)
293 {
294     RDA5807P_REGW[1] &= (~1);
295     OperationRDAFM_2w(WRITE, &(RDA5807P_REGW[0]), 2);
296 }
297 
298 /**
299  * @brief Set RDA5807P into mute mode
300  * @author RDA Ri'an Zeng
301  * @date 2008-11-05
302  * @param bool mute: if mute is true,then set mute; if mute is false,th
303  * @return void
304  * @retval
305  */
306 void RDA5807P_SetMute(boolean mute)
307 {
308     if (mute)
```

```
309         RDA5807P_REGW[0] &= ~(1 << 6);
310     else
311         RDA5807P_REGW[0] |= 1 << 6;
312     RDA5807P_REGW[0] |= monoBit << 5;
313     RDA5807P_REGW[0] |= bassBit << 4;
314     OperationRDAFM_2w(WRITE, &(RDA5807P_REGW[0]), 2); //RDA5807M_REGW
315     delay(50);                                        //Dealy 50 ms
316 }
317 //
318 /*********************************************
319  * @brief Set frequency function
320  * @author RDA Ri'an Zeng
321  * @date 2008-11-05
322  * @param int16_t curFreq:frequency value
323  * @return void
324  * @retval
325  *********************************************/
326 void RDA5807P_SetFreq(int16_t curFreq)
327 {
328     uint16_t curChan;
329     curChan = RDA5807P_FreqToChan(curFreq);
330 
331     if ((curFreq >= 6500) && (curFreq < 7600))
332     {
333         RDA5807P_REGW[3] = 0x0c;
334     }
335     else if ((curFreq >= 7600) && (curFreq < 10800))
336     {
337         RDA5807P_REGW[3] = 0x08; // sets the BAND bits (00xx = 87-108,
338                                  // for north america this must be set
339     }
340     //SetNoMute
341     RDA5807P_REGW[0] |= 1 << 6;
342     RDA5807P_REGW[0] |= monoBit << 5;
343     RDA5807P_REGW[0] |= bassBit << 4;
344     //handleBits();
345     RDA5807P_REGW[2] = curChan >> 2;
346     RDA5807P_REGW[3] = (((curChan & 0x0003) << 6) | 0x10) | (RDA5807P_F
347 
348     OperationRDAFM_2w(WRITE, &(RDA5807P_REGW[0]), 4);
349     delay(50); //Delay five ms
350     showRadioStation();
351 }
352 //
353 /**
```

```
354   * @brief Station judge for auto search
355   * @In auto search mode,uses this function to judge the frequency if hc
356   * @author RDA Ri'an Zeng
357   * @date 2008-11-05
358   * @param int16_t freq:frequency value
359   * @return bool: if return true,the frequency has a true station;otherv
360   * @retval
361   */
362  boolean RDA5807P_ValidStop(int freq)
363  {
364      uint8_t RDA5807P_reg_data[4] = {
365          0};
366      uint8_t falseStation = 0;
367      uint8_t i = 0;
368      uint16_t curChan;
369
370      if ((freq >= 6500) && (freq < 7600))
371      {
372          RDA5807P_REGW[3] = 0x0c;
373      }
374      else if ((freq >= 7600) && (freq < 10800))
375      {
376          RDA5807P_REGW[3] = 0x08; // sets the BAND bits (00xx = 87-108,
377                                   // for north america this must be set
378      }
379      curChan = RDA5807P_FreqToChan(freq);
380      //SetNoMute bit 9 is seek direction (0=seek down, 1=seek up).
381      //02H 14
382      RDA5807P_REGW[0] |= 1 << 6; // reg zero is bits 15 to bit 8 (this s
383      RDA5807P_REGW[0] |= monoBit << 5;
384      RDA5807P_REGW[0] |= bassBit << 4;
385      //handleBits();
386      RDA5807P_reg_data[0] = RDA5807P_REGW[0];
387      RDA5807P_reg_data[1] = RDA5807P_REGW[1];
388      RDA5807P_reg_data[2] = curChan >> 2;
389      RDA5807P_reg_data[3] = (((curChan & 0x0003) << 6) | 0x10) | (RDA58(
390      OperationRDAFM_2w(WRITE, &(RDA5807P_reg_data[0]), 4);
391
392      delay(50); //Dealy 25 ms
393
394      if (0x5808 == gChipID)
395          OperationRDAFM_2w(READ, &(RDA5807P_reg_data[0]), 4); //
396      else
397      {
398          do
```

```
399         {
400             i++;
401             if (i > 5)
402                 return 0;
403 
404             delay(30);
405             //read REG0A&0B
406             OperationRDAFM_2w(READ, &(RDA5807P_reg_data[0]), 4);
407         } while ((RDA5807P_reg_data[0] & 0x40) == 0);
408     }
409 
410     //check FM_TRUE
411     if ((RDA5807P_reg_data[2] & 0x01) == 0)
412         falseStation = 1; //0B 8  FM TRUE
413 
414     if (freq == 9600)
415         falseStation = 1; // North America - if scanning DOWN, the radi
416     delay(50);
417     if (falseStation == 1)
418         return 0;
419     else
420         return 1;
421 }
422 
423 /**
424  * @brief Get the signal level(RSSI) of the current frequency
425  * @author RDA Ri'an Zeng
426  * @date 2008-11-05
427  * @param int16_t curf:frequency value
428  * @return uint8_t: the signal level(RSSI)
429  * @retval
430  */
431 uint8_t RDA5807P_GetSigLvl(int16_t curf)
432 {
433     uint8_t RDA5807P_reg_data[4] = {
434         0};
435     OperationRDAFM_2w(READ, &(RDA5807P_reg_data[0]), 4);
436     delay(50);                          //Delay 50 ms
437     return (RDA5807P_reg_data[2] >> 1); /*??rssi*/
438 }
439 
440 /**
441  * @brief Set FM volume
442  * @It has better use the system volume operation to replace this funci
443  * @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],
    delay(50);

    error_ind = OperationRDAFM_2w(READ, (uint8_t *)&RDA5807P_REGR[0], 1
```

```
489     delay(50);
490 
491     gChipID = RDA5807P_REGR[8];
492     gChipID = ((gChipID << 8) | RDA5807P_REGR[9]);
493 
494     Serial.print("Chip ID: 0x");
495     Serial.println(gChipID, HEX);
496 
497     for (i = 0; i < 8; i++)
498     {
499         RDA5807P_REGW[i] = RDA5807N_initialization_reg[i];
500     }
501 
502     error_ind = OperationRDAFM_2w(WRITE, (uint8_t *)&RDA5807N_initiali
503     delay(600);
504     //Serial.println(sizeof(RDA5807N_initialization_reg));
505     error_ind = OperationRDAFM_2w(WRITE, (uint8_t *)&RDA5807N_initiali
506 
507     delay(50); //Dealy 50 ms
508 
509     if (error_ind)
510         return 0;
511     else
512         return 1;
513 }
514 //
515 /**
516  * @brief Cover the frequency to channel value
517  * @author RDA Ri'an Zeng
518  * @date 2008-11-05
519  * @param uint16 frequency:covered frequency
520  * @return uint16: channel value
521  * @retval
522  * In the United States, frequency-modulated broadcasting stations oper
523  * for a total of 20.2 MHz. It is divided into 101 channels, each 0.2 M
524  * In actual practice, no one (except the FCC) uses these channel numbe
525  */
526 uint16_t RDA5807P_FreqToChan(uint16_t frequency)
527 {
528     uint8_t channelSpacing = 10;
529     uint16_t channel = 0;
530 
531     if ((frequency >= 6500) && (frequency < 7600))
532     {
533         channel = (frequency - 6500) / channelSpacing;
```

```
534     }
535     else if ((frequency >= 7600) && (frequency < 10800))
536     {
537         channel = (frequency - 7600) / channelSpacing;
538     }
539     return (channel);
540 }
541 //
542 void loadDefaults()
543 {
544     char myCode[9] = "Grove_FM";
545     char myInit[9] = "blank123";
546     /*
547    * byte map in EEPROM
548     * 8, 9 the default frequency for a reboot
549     * 10, 11 current preset volume of the radio (used if no pot is attac
550     */
551     for (int i = 0; i < 8; i++)
552     {
553         myInit[i] = EEPROM.read(i); // read out to see if the thing is
554     }
555     if (strcmp(myCode, myInit) == 0)
556     {                           // if this is ZERO (we previously wrote sc
557         freq = epReadINT(8); // read back the INT for frequency from ee
558         if (!hasVolumePot)
559             vol = epReadINT(10); // read back the volume setting but dc
560     }
561     else // we don't have any defaults, so we have to save some first
562     {
563         for (int i = 0; i < 8; i++)
564         {
565             EEPROM.write(i, myCode[i]); // write this to EEPROM to show
566         }
567         saveDefaults(); // write the current default settings
568     }
569 }
570 //
571 void saveDefaults()
572 {
573     epWriteINT(8, freq); // write the two bytes for INT for a reboot
574     epWriteINT(10, vol); // write the current volume POT setting
575 }
576 //
577 void epWriteINT(int where, int theVal)
578 {
```

```
579     union uData {
580         byte stuff[2];
581         int f1; // 2 bytes of memory
582     } u;
583     u.f1 = theVal; // copy into the union
584     for (int j = 0; j < 2; j++)
585     {                                           // now we have to write ou
586         EEPROM.write(where + j, u.stuff[j]); // write it to EEPROM
587     }
588 }
589 //
590 long epReadINT(int where)
591 {
592     union uData {
593         byte stuff[2];
594         int f1; // 2 bytes of memory
595     } u;
596     for (int j = 0; j < 2; j++)
597     {
598         u.stuff[j] = EEPROM.read(where + j); // read back the 2 bytes o
599     }
600     return u.f1;
601 }
602 //
603 void epWriteLong(int where, long theVal)
604 {
605     union uData {
606         byte stuff[4];
607         long f1; // 4 bytes of memory
608     } u;
609     u.f1 = theVal; // copy into the union
610     for (int j = 0; j < 4; j++)
611     {                                           // now we have to write ou
612         EEPROM.write(where + j, u.stuff[j]); // write it to EEPROM
613     }
614 }
615 //
616 long epReadLong(int where)
617 {
618     union uData {
619         byte stuff[4];
620         long f1; // 4 bytes of memory
621     } u;
622     for (int j = 0; j < 4; j++)
623     {
```

```
624           u.stuff[j] = EEPROM.read(where + j); // read back the 4 bytes i
625       }
626       return u.f1;
627   }
```

- **Step 4.** Upload the demo. If you do not know how to upload the code, please check How to upload code [https://wiki.seeedstudio.com/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:

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

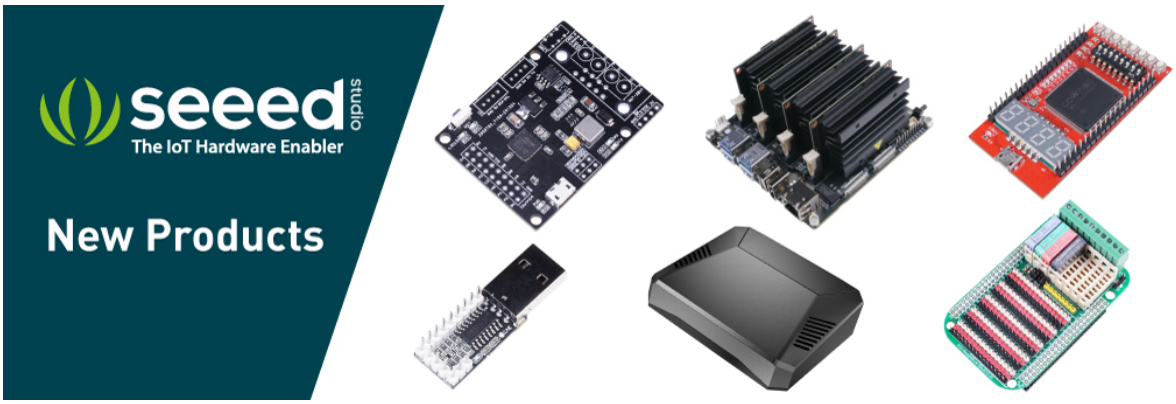
Schematic Online Viewer

Resources

- **[Zip]** Grove-I2C FM Receiver v1.1 eagle file [https://files.seeedstudio.com/wiki/Grove-I2C_FM_Receiver_v1.1/res/I2C%20FM%20Receiver%20v1.1.zip]
- **[PDF]** RDA5807 Datasheet [https://files.seeedstudio.com/wiki/Grove-I2C_FM_Receiver_v1.1/res/RDA5807%20Datasheet.pdf]

Tech Support

If you have any technical issue. Or submit the issue into our forum [https://forum.seeedstudio.com/].



[https://www.seeedstudio.com/act-4.html?utm_source=wiki&utm_medium=wikibanner&utm_campaign=newproducts]