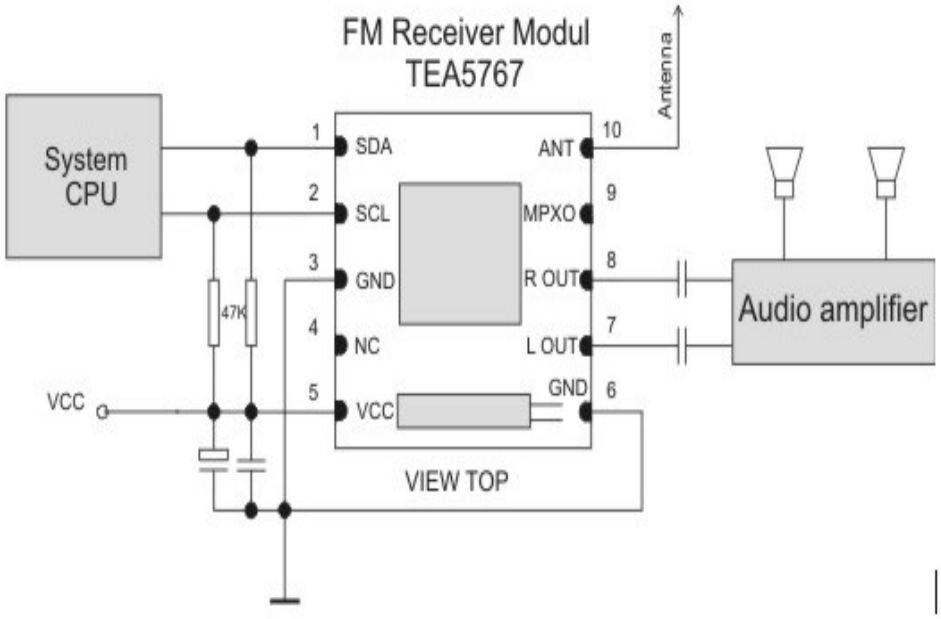
Radio Modul

http://www.voti.nl/docs/TEA5767.pdf



Funktionen

Auto Search

1. search tuning request (direction und „search stop level“ können eingestellt werden)

2. „ready flag“ checken; wenn gesetzt ist entweder ein Sender mit dem eingestellten Pegel gefunden oder das Frequenzband-Ende erreicht

Flag SM (search mode)

PLL[13:8][0:7] Synthesizer Counter Preset

SUD Search Up/Down

SSL[1:0] Search Stop Level (low,mid,high)

POR

Power on reset

Mute wird gesetzt; alle anderen Einstellungen müssen an das Modul übertragen werden

https://github.com/andykarpov/TEA5767/tree/master/examples/SimpleRadioFM

```
/*
```

```
* TEA5767.h
* defintions for TEA5767 FM radio module.
* Created by Andrey Karpov
* Based on this project: http://kalum.posterous.com/arduino-with-tea5767-single-
chip-radio-and-no
*
* This program is free software; you can redistribute it and/or modify it
* under the terms of the GNU General Public License as published by the
* Free Software Foundation; either version 2, or (at your option) any
* later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/

#ifndef _TEA5767_H_
#define _TEA5767_H_

/*****************************
* Write mode register values *
*****************************/

/* First register */
#define TEA5767_MUTE 0x80 /* Mutes output */
#define TEA5767_SEARCH 0x40 /* Activates station search */
/* Bits 0-5 for divider MSB */

/* Second register */
/* Bits 0-7 for divider LSB */

/* Third register */

/* Station search from botton to up */
#define TEA5767_SEARCH_UP 0x80

/* Searches with ADC output = 10 */
#define TEA5767_SRCH_HIGH_LVL 0x60

/* Searches with ADC output = 10 */
#define TEA5767_SRCH_MID_LVL 0x40

/* Searches with ADC output = 5 */
#define TEA5767_SRCH_LOW_LVL 0x20

/* if on, div=4*(Frf+Fif)/Fref otherwise, div=4*(Frf-Fif)/Freq) */
#define TEA5767_HIGH_LO_INJECT 0x10

/* Disable stereo */
#define TEA5767_MONO 0x08

/* Disable right channel and turns to mono */
#define TEA5767_MUTE_RIGHT 0x04

/* Disable left channel and turns to mono */
#define TEA5767_MUTE_LEFT 0x02
```

```
#define TEA5767_PORT1_HIGH 0x01

/* Fourth register */
#define TEA5767_PORT2_HIGH 0x80
/* Chips stops working. Only I2C bus remains on */
#define TEA5767_STDBY 0x40

/* Japan freq (76-108 MHz. If disabled, 87.5-108 MHz */
#define TEA5767_JAPAN_BAND 0x20

/* Unselected means 32.768 KHz freq as reference. Otherwise Xtal at 13 MHz */
#define TEA5767_XTAL_32768 0x10

/* Cuts weak signals */
#define TEA5767_SOFT_MUTE 0x08

/* Activates high cut control */
#define TEA5767_HIGH_CUT_CTRL 0x04

/* Activates stereo noise control */
#define TEA5767_ST_NOISE_CTL 0x02

/* If activate PORT 1 indicates SEARCH or else it is used as PORT1 */
#define TEA5767_SRCH_IND 0x01

/* Fifth register */

/* By activating, it will use Xtal at 13 MHz as reference for divider */
#define TEA5767_PLLREF_ENABLE 0x80

/* By activating, deemphasis=50, or else, deemphasis of 50us */
#define TEA5767_DEEMPH_75 0X40

/*****************************
* Read mode register values *
*****************************/

/* First register */
#define TEA5767_READY_FLAG_MASK 0x80
#define TEA5767_BAND_LIMIT_MASK 0X40
/* Bits 0-5 for divider MSB after search or preset */

/* Second register */
/* Bits 0-7 for divider LSB after search or preset */

/* Third register */
#define TEA5767_STEREO_MASK 0x80
#define TEA5767_IF_CNTR_MASK 0x7f

/* Fourth register */
#define TEA5767_ADC_LEVEL_MASK 0xf0

/* should be 0 */
#define TEA5767_CHIP_ID_MASK 0x0f

/* Fifth register */
/* Reserved for future extensions */
#define TEA5767_RESERVED_MASK 0xff

/* internal constants */
#define TEA5767_SEARCH_DIR_UP 1
#define TEA5767_SEARCH_DIR_DOWN 2
```

```
#include <Arduino.h>

struct tea5767_ctrl
{
  unsigned int port1:1;
  unsigned int port2:1;
  unsigned int high_cut:1;
  unsigned int st_noise:1;
  unsigned int soft_mute:1;
  unsigned int japan_band:1;
  unsigned int deemph_75:1;
  unsigned int pllref:1;
  unsigned int xtal_freq;
};

class TEA5767 {
private:
tea5767_ctrl ctrl_data;
int HILO;
protected:
int hilo_optimal (unsigned long freq);
void set_frequency (int hilo, double freq);
int ready (unsigned char *buf);
int bl_reached (unsigned char *buf);
public:
TEA5767();
TEA5767(double initial_freq);
void set_frequency (double freq);
int read_status (unsigned char *buf);
int signal_level (unsigned char *buf);
int stereo (unsigned char *buf);
double frequency_available (unsigned char *buf);
void search_up (unsigned char *buf);
void search_down (unsigned char *buf);
int process_search (unsigned char *buf, int search_dir);
void init();
};

#endif // _TEA5767_H_


#include <stdio.h>
#include <avr/pgmspace.h>
#include <Arduino.h>
#include <Wire.h>
#include "TEA5767.h"

TEA5767::TEA5767() {
        Wire.begin();
        HILO = 1;
}

TEA5767::TEA5767(double initial_freq) {
        Wire.begin();
        HILO = 1;
        set_frequency(initial_freq);
}

//calculate the optimial hi or lo injection mode for the freq is in hz
//return 1 if high is the best, or 0 for low injection
int TEA5767::hilo_optimal (unsigned long freq) {
```

```
        int signal_high = 0;
        int signal_low = 0;
        unsigned char buf[5];

        set_frequency (1, (double) (freq + 450000) / 1000000);
        delay (30);

        // Read the signal level
        if (read_status (buf) == 1) {
                signal_high = signal_level (buf);
        }

        set_frequency (0, (double) (freq - 450000) / 1000000);
        delay (30);

        if (read_status (buf) == 1) {
                signal_low = signal_level (buf);
        }

        return (signal_high < signal_low) ? 1 : 0;
}

void TEA5767::set_frequency (int hilo, double freq) {
        unsigned char buffer[5];
        unsigned div;
        int rc;

        memset (buffer, 0, 5);

        buffer[2] = 0;
        buffer[2] |= TEA5767_PORT1_HIGH;

        if (hilo == 1)
        buffer[2] |= TEA5767_HIGH_LO_INJECT;

        buffer[3] = 0;

        if (ctrl_data.port2)
                buffer[3] |= TEA5767_PORT2_HIGH;

        if (ctrl_data.high_cut)
                buffer[3] |= TEA5767_HIGH_CUT_CTRL;

        if (ctrl_data.st_noise)
                buffer[3] |= TEA5767_ST_NOISE_CTL;

        if (ctrl_data.soft_mute)
                buffer[3] |= TEA5767_SOFT_MUTE;

        if (ctrl_data.japan_band)
                buffer[3] |= TEA5767_JAPAN_BAND;

        buffer[3] |= TEA5767_XTAL_32768;
        buffer[4] = 0;

        if (ctrl_data.deemph_75)
                buffer[4] |= TEA5767_DEEMPH_75;

        if (ctrl_data.pllref)
                buffer[4] |= TEA5767_PLLREF_ENABLE;
```

```
        if (hilo == 1)
                div = (4 * (freq * 1000 + 225)) / 32.768;
        else
                div = (4 * (freq * 1000 - 225)) / 32.768;

        buffer[0] = (div >> 8) & 0x3f;
        buffer[1] = div & 0xff;

        Wire.beginTransmission (0x60);

        for (int i = 0; i < 5; i++)
                Wire.write (buffer[i]);

        Wire.endTransmission ();
}

/* Freq should be specifyed at X M hz */
void TEA5767::set_frequency (double freq) {
        HILO = hilo_optimal ((unsigned long) (freq * 1000000));
        set_frequency (HILO, freq);
}

//read functions

int TEA5767::read_status (unsigned char *buf) {
        memset (buf, 0, 5);
        Wire.requestFrom (0x60, 5);    //reading TEA5767

        if (Wire.available ()) {
                for (int i = 0; i < 5; i++) {
                        buf[i] = Wire.read ();
                }
        return 1;
        } else {
        return 0;
        }
}

int TEA5767::signal_level (unsigned char *buf) {
        int signal = ((buf[3] & TEA5767_ADC_LEVEL_MASK) >> 4);
        return signal;
}

int TEA5767::stereo (unsigned char *buf) {
        int stereo = (buf[2] & TEA5767_STEREO_MASK);
        return stereo ? 1 : 0;
}

//returns 1 if tuning completed or BL reached
int TEA5767::ready (unsigned char *buf) {
        return (buf[0] & 0x80) ? 1 : 0;
}

//returns 1 if band limit is reached during searching
int TEA5767::bl_reached (unsigned char *buf) {
        return (buf[0] & 0x40) ? 1 : 0;
}

//returns freq available in Hz
double TEA5767::frequency_available (unsigned char *buf) {
        double freq_available;
        if (HILO == 1)
```

```
        freq_available = (((buf[0] & 0x3F) << 8) + buf[1]) * 32768 / 4 - 225000;
        else
        freq_available = (((buf[0] & 0x3F) << 8) + buf[1]) * 32768 / 4 + 225000;
        return freq_available;
}

void TEA5767::search_up (unsigned char *buf) {
        unsigned div;
        double freq_av;

        freq_av = frequency_available (buf);

        div = (4 * (((freq_av + 98304) / 1000000) * 1000000 + 225000)) / 32768;

        buf[0] = (div >> 8) & 0x3f;
        buf[1] = div & 0xff;

        buf[0] |= TEA5767_SEARCH;

        buf[2] = 0;

        buf[2] |= TEA5767_SEARCH_UP;
        buf[2] |= TEA5767_SRCH_MID_LVL;
        buf[2] |= TEA5767_HIGH_LO_INJECT;

        //buf[3] = 0x18;
        buf[3] = 0;

        if (ctrl_data.port2)
        buf[3] |= TEA5767_PORT2_HIGH;

        if (ctrl_data.high_cut)
        buf[3] |= TEA5767_HIGH_CUT_CTRL;

        if (ctrl_data.st_noise)
        buf[3] |= TEA5767_ST_NOISE_CTL;

        if (ctrl_data.soft_mute)
        buf[3] |= TEA5767_SOFT_MUTE;

        if (ctrl_data.japan_band)
        buf[3] |= TEA5767_JAPAN_BAND;

        buf[3] |= TEA5767_XTAL_32768;

        buf[4] = 0;

        if (ctrl_data.deemph_75)
        buf[4] |= TEA5767_DEEMPH_75;

        if (ctrl_data.pllref)
        buf[4] |= TEA5767_PLLREF_ENABLE;

        Wire.beginTransmission (0x60);

        for (int i = 0; i < 5; i++)
        Wire.write (buf[i]);

        Wire.endTransmission ();
        HILO = 1;
}
```

```
void TEA5767::search_down (unsigned char *buf)
{
        unsigned div;
        double freq_av;

        freq_av = frequency_available (buf);

        div = (4 * (((freq_av - 98304) / 1000000) * 1000000 + 225000)) / 32768;

        buf[0] = (div >> 8) & 0x3f;
        buf[1] = div & 0xff;

        buf[0] |= TEA5767_SEARCH;

        buf[2] = 0;

        buf[2] |= TEA5767_SRCH_MID_LVL;
        buf[2] |= TEA5767_HIGH_LO_INJECT;

        buf[3] = 0;

        if (ctrl_data.port2)
        buf[3] |= TEA5767_PORT2_HIGH;

        if (ctrl_data.high_cut)
        buf[3] |= TEA5767_HIGH_CUT_CTRL;

        if (ctrl_data.st_noise)
        buf[3] |= TEA5767_ST_NOISE_CTL;

        if (ctrl_data.soft_mute)
        buf[3] |= TEA5767_SOFT_MUTE;

        if (ctrl_data.japan_band)
        buf[3] |= TEA5767_JAPAN_BAND;

        buf[3] |= TEA5767_XTAL_32768;

        buf[4] = 0;

        if (ctrl_data.deemph_75)
        buf[4] |= TEA5767_DEEMPH_75;

        if (ctrl_data.pllref)
        buf[4] |= TEA5767_PLLREF_ENABLE;

        Wire.beginTransmission (0x60);

        for (int i = 0; i < 5; i++)
        Wire.write (buf[i]);

        Wire.endTransmission ();

        HILO = 1;
}

//Returns 1 if search is finished, 0 if wrapped and new search initiated
//TODO - To prevent endless looping add a static variable to abort if it has
searched for more than 2 loops
int TEA5767::process_search (unsigned char *buf, int search_dir)
{
        if (ready (buf) == 1) {
```

```
        if (bl_reached (buf) == 1) {
                if (search_dir == TEA5767_SEARCH_DIR_UP) {
                        //wrap down
                        set_frequency (87.5);
                        read_status (buf);
                        search_up (buf);
                        return 0;
                } else if (search_dir == TEA5767_SEARCH_DIR_DOWN) {
                        //wrap up
                        set_frequency (108);
                        read_status (buf);
                        search_down (buf);
                        return 0;
                }
        } else {
                // search finished - round up the pll word and feed it back as
recommended
                double rounded_freq;
                double freq_available = frequency_available (buf);
                rounded_freq = floor (freq_available / 100000 + .5) / 10;
                set_frequency (rounded_freq);
                return 1;
        }
    }
}

void TEA5767::init() {
  ctrl_data.port1 = 1;
  ctrl_data.port2 = 1;
  ctrl_data.high_cut = 1;
  ctrl_data.st_noise = 1;
  ctrl_data.soft_mute = 1;
  ctrl_data.deemph_75 = 0;
  ctrl_data.japan_band = 0;
  ctrl_data.pllref = 0;

//  unsigned long freq = 87500000;
//  set_frequency((float) freq / 1000000);

}
```

Beispiel

```
#include <TEA5767.h>
#include <Wire.h>
#include <Button.h>
#include <LiquidCrystal.h>

TEA5767 Radio;
LiquidCrystal lcd(12,11,10,9,8,7);
double old_frequency;
double frequency;
int search_mode = 0;
int search_direction;
unsigned long last_pressed;

Button btn_forward(3, PULLUP);
Button btn_backward(2, PULLUP);

void setup() {
```

```
  Wire.begin();
  Radio.init();
  Radio.set_frequency(105.4);
  Serial.begin(9600);
  lcd.begin(16,2);
  lcd.clear();
}

void loop() {

  unsigned char buf[5];
  int stereo;
  int signal_level;
  double current_freq;
  unsigned long current_millis = millis();

  if (Radio.read_status(buf) == 1) {
    current_freq =  floor (Radio.frequency_available (buf) / 100000 + .5) / 10;
    stereo = Radio.stereo(buf);
    signal_level = Radio.signal_level(buf);
    lcd.setCursor(0,0);
    lcd.print("FM: "); lcd.print(current_freq);
    lcd.setCursor(0,1);
    if (stereo) lcd.print("STEREO "); else lcd.print("MONO ");
    //lcd.print(signal_level);
  }

  if (search_mode == 1) {
      if (Radio.process_search (buf, search_direction) == 1) {
          search_mode = 0;
      }
  }

  if (btn_forward.isPressed()) {
    last_pressed = current_millis;
    search_mode = 1;
    search_direction = TEA5767_SEARCH_DIR_UP;
    Radio.search_up(buf);
    delay(300);
  }

  if (btn_backward.isPressed()) {
    last_pressed = current_millis;
    search_mode = 1;
    search_direction = TEA5767_SEARCH_DIR_DOWN;
    Radio.search_down(buf);
    delay(300);
  }
  //delay(20);
  delay(50);
}
```