```
//Si4734 i2C functions
//Roger Traylor 11.13.2011
//device driver for the si4734 chip.
//TODO: unify the properties if possible between modes
//TODO: unify power up commands...all 0x01?, think so, power ups look the same
//TODO: document how this is now running with interrupt mode
//      and not blind timing.

// header files
#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdlib.h>
#include <util/twi.h>
#include <avr/eeprom.h>
#include <util/delay.h>
#include "uart_functions.h"

#include "twi_master.h" //my defines for TWCR_START, STOP, RACK, RNACK, SEND
#include "si4734.h"

uint8_t si4734_wr_buf[9];          //buffer for holding data to send to the si47
34
uint8_t si4734_rd_buf[15];         //buffer for holding data recieved from the s
i4734
uint8_t si4734_tune_status_buf[8]; //buffer for holding tune_status data
uint8_t si4734_revision_buf[16];   //buffer for holding revision  data

enum radio_band{FM, AM, SW};
extern volatile enum radio_band current_radio_band;

volatile uint8_t STC_interrupt;  //flag bit to indicate tune or seek is done

 uint16_t eeprom_fm_freq;
 uint16_t eeprom_am_freq;
 uint16_t eeprom_sw_freq;
 uint8_t  eeprom_volume;

uint16_t current_fm_freq;
uint16_t current_am_freq;
uint16_t current_sw_freq;
uint8_t  current_volume;

//Used in debug mode for UART1
extern char uart1_tx_buf[40];      //holds string to send to crt
extern char uart1_rx_buf[40];      //holds string that recieves data from uart
//*************************************************************


//****************************************************************************
**
//                            get_int_status()
//
//Fetch the interrupt status available from the status byte.
//
//TODO: update for interrupts
//
uint8_t get_int_status(){

    si4734_wr_buf[0] = GET_INT_STATUS;
    twi_start_wr(SI4734_ADDRESS, si4734_wr_buf, 1); //send get_int_status comman
d
    while( twi_busy() ){}; //spin while previous TWI transaction finshes
    _delay_us(300);        //si4734 process delay
    twi_start_rd(SI4734_ADDRESS, si4734_rd_buf, 1); //get the interrupt status
    while( twi_busy() ){}; //spin while previous TWI transaction finshes
    return(si4734_rd_buf[0]);
}
//****************************************************************************
**
```



```
//****************************************************************************
**
//                            fm_tune_freq()
//
//takes current_fm_freq and sends it to the radio chip
//

void fm_tune_freq(){
  si4734_wr_buf[0] = 0x20;  //fm tune command
  si4734_wr_buf[1] = 0x00;  //no FREEZE and no FAST tune
  si4734_wr_buf[2] = (uint8_t)(current_fm_freq >> 8); //freq high byte
  si4734_wr_buf[3] = (uint8_t)(current_fm_freq);      //freq low byte
  si4734_wr_buf[4] = 0x00;  //antenna tuning capactior
  //send fm tune command
  STC_interrupt = FALSE;
  twi_start_wr(SI4734_ADDRESS, si4734_wr_buf, 5);
  while( ! STC_interrupt ){}; //spin until the tune command finishes
}
//****************************************************************************
**

//****************************************************************************
**
//                            am_tune_freq()
//
//takes current_am_freq and sends it to the radio chip
//

void am_tune_freq(){
  si4734_wr_buf[0] = AM_TUNE_FREQ; //am tune command
  si4734_wr_buf[1] = 0x00;         //no FAST tune
  si4734_wr_buf[2] = (uint8_t)(current_am_freq >> 8); //freq high byte
  si4734_wr_buf[3] = (uint8_t)(current_am_freq);      //freq low byte
  si4734_wr_buf[4] = 0x00;  //antenna tuning capactior high byte
  si4734_wr_buf[5] = 0x00;  //antenna tuning capactior low byte
  //send am tune command
  STC_interrupt = FALSE;
  twi_start_wr(SI4734_ADDRESS, si4734_wr_buf, 6);
  while( ! STC_interrupt ){}; //spin until the tune command finishes
}
//****************************************************************************
**

//****************************************************************************
**
//                            sw_tune_freq()
//
//takes current_sw_freq and sends it to the radio chip
//antcap low byte is 0x01 as per datasheet

void sw_tune_freq(){
  si4734_wr_buf[0] = 0x40;  //am tune command
  si4734_wr_buf[1] = 0x00;  //no FAST tune
  si4734_wr_buf[2] = (uint8_t)(current_sw_freq >> 8); //freq high byte
  si4734_wr_buf[3] = (uint8_t)(current_sw_freq);      //freq low byte
  si4734_wr_buf[4] = 0x00;  //antenna tuning capactior high byte
  si4734_wr_buf[5] = 0x01;  //antenna tuning capactior low byte
  //send am tune command
  twi_start_wr(SI4734_ADDRESS, si4734_wr_buf, 6);
   _delay_ms(80); //TODO: FIX
}

//****************************************************************************
**
//                            fm_pwr_up()
//
void fm_pwr_up(){
//restore the previous fm frequency
```

```
 //current_fm_freq = eeprom_read_word(&eeprom_fm_freq); //TODO: only this one do
es not work
 //current_volume  = eeprom_read_byte(&eeprom_volume);  //TODO: only this one do
es not work

//send fm power up command
  si4734_wr_buf[0] = FM_PWR_UP; //powerup command byte
  si4734_wr_buf[1] = 0x50;      //GPO2O enabled, STCINT enabled, use ext. 32khz
osc.
  si4734_wr_buf[2] = 0x05;      //OPMODE = 0x05; analog audio output
  twi_start_wr(SI4734_ADDRESS, si4734_wr_buf, 3);
  _delay_ms(120);               //startup delay as specified
  //The seek/tune interrupt is enabled here. If the STCINT bit is set, a 1.5us
  //low pulse will be output from GPIO2/INT when tune or seek is completed.
  set_property(GPO_IEN, GPO_IEN_STCIEN); //seek_tune complete interrupt
}
//***************************************************************************
**

//***************************************************************************
**
//                            am_pwr_up()
//
void am_pwr_up(){
//restore the previous am frequency
  //current_am_freq = eeprom_read_word(&eeprom_am_freq);
  //current_volume  = eeprom_read_byte(&eeprom_volume); //TODO: only this one do
es not work

//send am power up command
  si4734_wr_buf[0] = AM_PWR_UP;
  si4734_wr_buf[1] = 0x51;//GPO2OEN and XOSCEN selected
  si4734_wr_buf[2] = 0x05;
  twi_start_wr(SI4734_ADDRESS, si4734_wr_buf, 3);
  _delay_ms(120);
  set_property(GPO_IEN, GPO_IEN_STCIEN);    //Seek/Tune Complete interrupt
}
//***************************************************************************
**

//***************************************************************************
**
//                            sw_pwr_up()
//

void sw_pwr_up(){
//restore the previous sw frequency
  //current_sw_freq = eeprom_read_word(&eeprom_sw_freq);
  //current_volume  = eeprom_read_byte(&eeprom_volume); //TODO: only this one do
es not work

//send sw power up command (same as am, only tuning rate is different)
    si4734_wr_buf[0] = AM_PWR_UP; //same cmd as for AM
    si4734_wr_buf[1] = 0x51;
    si4734_wr_buf[2] = 0x05;
    twi_start_wr(SI4734_ADDRESS, si4734_wr_buf, 3);
    _delay_ms(120);   //start up delay

  //set property to disable soft muting for shortwave broadcasts
  set_property(AM_SOFT_MUTE_MAX_ATTENUATION, 0x0000); //cut off soft mute
  //select 4khz filter BW and engage power line filter
  set_property(AM_CHANNEL_FILTER, (AM_CHFILT_4KHZ | AM_PWR_LINE_NOISE_REJT_FILTE
R));
  set_property(GPO_IEN, GPO_IEN_STCIEN); //Seek/Tune Complete interrupt
}
//***************************************************************************
**

//***************************************************************************
```



```
**
//                            radio_pwr_dwn()
//

void radio_pwr_dwn(){
/*
//save current frequency to EEPROM
switch(current_radio_band){
  case(FM) : eeprom_write_word(&eeprom_fm_freq, current_fm_freq); break;
  case(AM) : eeprom_write_word(&eeprom_am_freq, current_am_freq); break;
  case(SW) : eeprom_write_word(&eeprom_sw_freq, current_sw_freq); break;
  default  : break;
}//switch

  eeprom_write_byte(&eeprom_volume, current_volume); //save current volume level
*/
//send fm power down command
    si4734_wr_buf[0] = 0x11;
    twi_start_wr(SI4734_ADDRESS, si4734_wr_buf, 1);
   _delay_us(310); //power down delay
}
//***************************************************************************
**

//***************************************************************************
**
//                            fm_rsq_status()
//
//Get the status on the receive signal quality. This command returns signal stre
ngth
//(RSSI), signal to noise ratio (SNR), and other info. This function sets the
//FM_RSQ_STATUS_IN_INTACK bit so it clears RSQINT and some other interrupt flags
//inside the chip.
//TODO: Dang, thats a big delay, could cause problems, best check out.
//
void fm_rsq_status(){

    si4734_wr_buf[0] = FM_RSQ_STATUS;            //fm_rsq_status command
    si4734_wr_buf[1] = FM_RSQ_STATUS_IN_INTACK;  //clear STCINT bit if set
    twi_start_wr(SI4734_ADDRESS, si4734_wr_buf, 2);
    while(twi_busy()){}; //spin while previous TWI transaction finshes
    _delay_us(300);      //delay for si4734 to process
    //This is a blind wait. Waiting for CTS interrupt here would tell you
    //when the command is received and has been processed.
    //get the fm tune status
    twi_start_rd(SI4734_ADDRESS, si4734_tune_status_buf, 8);
    while(twi_busy()){}; //spin while previous TWI transaction finshes
}


//***************************************************************************
**
//                            fm_tune_status()
//
//Get the status following a fm_tune_freq command. Returns the current frequency
,
//RSSI, SNR, multipath and antenna capacitance value. The STCINT interrupt bit
//is cleared.
//TODO: Dang, thats a big delay, could cause problems, best check out.
//
void fm_tune_status(){

    si4734_wr_buf[0] = FM_TUNE_STATUS;            //fm_tune_status command
    si4734_wr_buf[1] = FM_TUNE_STATUS_IN_INTACK;  //clear STCINT bit if set
    twi_start_wr(SI4734_ADDRESS, si4734_wr_buf, 2);
    while(twi_busy()){}; //spin while previous TWI transaction finshes
    _delay_us(300);        //delay for si4734 to process
    //get the fm tune status
    twi_start_rd(SI4734_ADDRESS, si4734_tune_status_buf, 8);
```

```
    while( twi_busy() ){}; //spin till TWI read transaction finshes
}

//***************************************************************************
**
//                            am_tune_status()
//
//TODO: could probably just have one tune_status() function
//TODO: Dang, thats a big delay, could cause problems, best check out.

void am_tune_status(){

    si4734_wr_buf[0] = AM_TUNE_STATUS;            //fm_tune_status command
    si4734_wr_buf[1] = AM_TUNE_STATUS_IN_INTACK;  //clear STCINT bit if set
    twi_start_wr(SI4734_ADDRESS, si4734_wr_buf, 2);
    while(twi_busy()){}; //spin while previous TWI transaction finshes
    _delay_us(300);        //delay for si4734 to process command
    //get the am tune status
    twi_start_rd(SI4734_ADDRESS, si4734_tune_status_buf, 8);

}
//***************************************************************************
**
//                            am_rsq_status()
//
//TODO: Dang, thats a big delay, could cause problems, best check out.

void am_rsq_status(){

    si4734_wr_buf[0] = AM_RSQ_STATUS;            //am_rsq_status command
    si4734_wr_buf[1] = AM_RSQ_STATUS_IN_INTACK;  //clear STCINT bit if set
    twi_start_wr(SI4734_ADDRESS, si4734_wr_buf, 2);
    while(twi_busy()){}; //spin while previous TWI transaction finshes
    _delay_us(300);        //delay for si4734 to process command
    //get the fm tune status
    twi_start_rd(SI4734_ADDRESS, si4734_tune_status_buf, 8);
}

//***************************************************************************
**
//                            set_property()
//
//The set property command does not have a indication that it has completed. Thi
s
//command is guarnteed by design to finish in 10ms.
//
void set_property(uint16_t property, uint16_t property_value){

    si4734_wr_buf[0] = SET_PROPERTY;                    //set property command
    si4734_wr_buf[1] = 0x00;                            //all zeros
    si4734_wr_buf[2] = (uint8_t)(property >> 8);        //property high byte
    si4734_wr_buf[3] = (uint8_t)(property);             //property low byte
    si4734_wr_buf[4] = (uint8_t)(property_value >> 8); //property value high byt
e
    si4734_wr_buf[5] = (uint8_t)(property_value);       //property value low byte
    twi_start_wr(SI4734_ADDRESS, si4734_wr_buf, 6);
    _delay_ms(10);  //SET_PROPERTY command takes 10ms to complete
}//set_property()
/*
//***************************************************************************
//                            get_rev()
//
//TODO: UNTESTED!
//Report the chip revision info via uart1. UART1 be setup and connected to
//a dumb terminal. e.g.: screen /dev/cu.usbserial-A800fh27 9600
//
void get_rev(){
    si4734_wr_buf[0] = GET_REV;                   //get rev command
    twi_start_wr(SI4734_ADDRESS, si4734_wr_buf, 1);
```



```
    while( twi_busy() ){}; //spin till TWI read transaction finshes
    _delay_us(300);        //wait for processing delay
//get the revision info
    twi_start_rd(SI4734_ADDRESS, si4734_revision_buf, 8);
    while( twi_busy() ){}; //spin till TWI read transaction finshes
//use TABs instead?
    uart1_puts("Si4734 Rev:  last 2 digits of part no.   chip rev     \n\r");
    uart1_puts("             -------------------------   --------     \n\r");
    uart1_puts("                              "); itoa((int)si4734_revision_buf[1],
uart1_tx_buf, 10); uart1_puts(uart1_tx_buf);
    uart1_puts("             "); itoa((int)si4734_revision_buf[2], uart1_tx_buf,
 10); uart1_puts(uart1_tx_buf); uart1_puts("\n\r");
}

//***************************************************************************
//                            get_fm_rsq_status()
//
// TODO: UNTESTED!
//Report the fm rsq status via uart1. Requires that UART1 be setup and connected
 to
//a terminal. e.g.: screen /dev/cu.usbserial-A800fh27 9600
//Also requires that a fm_tune_status has been previously called.

void get_fm_rsq_status(){
  uint8_t disp_freq;  //temp holding variable
  char    str[40];    //temp for building strings

  uart1_puts("FM_RSQ_STATUS: ");
  uart1_puts("status byte   :");   itoa((int)si4734_tune_status_buf[0], uart1_tx
_buf, 16);   uart1_puts(uart1_tx_buf); uart1_puts("\n\r");
  uart1_puts("resp1         :");   itoa((int)si4734_tune_status_buf[1], uart1_tx
_buf, 10);   uart1_puts(uart1_tx_buf); uart1_puts("\n\r");
  disp_freq = si4734_tune_status_buf[2];      //load high frequency byte
  disp_freq = (disp_freq << 8); //shift upper byte to upper 8 bits
  disp_freq |= si4734_tune_status_buf[3];     //load low high frequency byte
  uart1_puts("freq          :");   itoa(disp_freq, uart1_tx_buf, 10);   uart1_pu
ts(uart1_tx_buf); uart1_puts("\n\r");
  uart1_puts("freq high     :");   itoa((int)si4734_tune_status_buf[2], str, 16)
;   uart1_puts(str); uart1_puts("\n\r");
  uart1_puts("freq low      :");   itoa((int)si4734_tune_status_buf[3], str, 16)
;   uart1_puts(str); uart1_puts("\n\r");
  uart1_puts("rssi          :");   itoa((int)si4734_tune_status_buf[4], uart1_tx
_buf, 16);   uart1_puts(uart1_tx_buf); uart1_puts("\n\r");
}*/
```