Si4735 Arduino Library

AUTHOR
Version 1.1.8
02/04/2020

Table of Contents

Table of contents

Module Index

Modules

Here is a list of all modules:

Class Index

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Module Documentation

Deal with Interrupt

Detailed Description

Deal with Interrupt

Deal with Interrupt

Classes

class SI4735
SI4735 Class.

Detailed Description

RDS Data types

Classes

union si47x_rqs_status
Radio Signal Quality data representation.
union si47x_rds_command
Data type for RDS Status command and response information.
union si47x_rds_status
Response data type for current channel and reads an entry from the RDS FIFO.
union si47x_rds_int_source
FM_RDS_INT_SOURCE property data type.
union si47x_rds_config
Data type for FM_RDS_CONFIG Property.
union si47x_rds_blocka
Block A data type.
union si47x_rds_blockb
Block B data type.
union si47x_rds_date_time

Detailed Description

Receiver Status and Setup

Classes

union si47x_agc_status
union si47x_agc_overrride
union si47x_bandwidth_config
union si47x_ssb_mode
union si4735_digital_output_format
Digital audio output format data structure (Property 0x0102. DIGITAL_OUTPUT_FORMAT).
struct si4735_digital_output_sample_rate
Digital audio output sample structure (Property 0x0104. DIGITAL_OUTPUT_SAMPLE_RATE).

Detailed Description

SI473X data types

SI473X data representation.

Classes

union si473x_powerup
Power Up arguments data type.
union si47x_frequency
Represents how the frequency is stored in the si4735.
union si47x_antenna_capacitor
Antenna Tuning Capacitor data type manupulation.
union si47x_set_frequency
AM Tune frequency data type command (AM_TUNE_FREQ command)
union si47x_seek
Seek frequency (automatic tuning)
union si47x_response_status
Response status command.
union si47x_firmware_information
Data representation for Firmware Information (GET_REV)
union si47x_firmware_query_library
Firmware Query Library ID response.
union si47x_tune_status
Seek station status.
union si47x_property
Data type to deal with SET_PROPERTY command.

Detailed Description

SI473X data representation.

The goal here is separate data from code. The Si47XX family works with many internal data that can be represented by data structure or defined data type in C/C++. These C/C++ resources have been used widely here.

This aproach made the library easier to build and maintain. Each data structure created here has its reference (name of the document and page on which it was based). In other words, to make the SI47XX device easier to deal, some defined data types were created to handle byte and bits to process commands, properties and responses. These data types will be usefull to deal with SI473X

Class Documentation

SI4735 Class Reference

SI4735 Class.

`#include <SI4735.h>`

Public Member Functions

SI4735 ()
void reset (void)
void waitToSend (void)
void setup (uint8_t resetPin, uint8_t defaultFunction)
void setup (uint8_t resetPin, int interruptPin, uint8_t defaultFunction, uint8_t audioMode=SI473X_ANALOG_AUDIO)
void setPowerUp (uint8_t CTSIEN, uint8_t GPO2OEN, uint8_t PATCH, uint8_t XOSCEN, uint8_t FUNC, uint8_t OPMODE)
void radioPowerUp (void)
void analogPowerUp (void)
void powerDown (void)
void setFrequency (uint16_t)
void getStatus ()
void getStatus (uint8_t, uint8_t)
uint16_t getFrequency (void)
uint16_t getCurrentFrequency ()
bool getSignalQualityInterrupt ()
bool getRadioDataSystemInterrupt ()
Gets Received Signal Quality Interrupt(RSQINT)

bool getTuneCompleteTriggered ()
Gets Radio Data System (RDS) Interrupt.

bool getStatusError ()
Seek/Tune Complete Interrupt; 1 = Tune complete has been triggered.

bool getStatusCTS ()
Return the Error flag (true or false) of status of the least Tune or Seek.

bool getACFIndicator ()
Gets the Error flag of status response.

bool getBandLimit ()
Returns true if the AFC rails (AFC Rail Indicator).

bool getStatusValid ()
Returns true if a seek hit the band limit (WRAP = 0 in FM_START_SEEK) or wrapped to the original frequency(WRAP = 1).

uint8_t getReceivedSignalStrengthIndicator ()
Returns true if the channel is currently valid as determined by the seek/tune properties (0x1403, 0x1404, 0x1108)

uint8_t getStatusSNR ()
Returns integer Received Signal Strength Indicator (dBÎ¼V).

uint8_t getStatusMULT ()
Returns integer containing the SNR metric when tune is complete (dB).

uint8_t getAntennaTuningCapacitor ()
Returns integer containing the multipath metric when tune is complete.

void getAutomaticGainControl ()
Returns integer containing the current antenna tuning capacitor value.

void setAvcAmMaxGain (uint8_t gain)
void setAutomaticGainControl (uint8_t AGCDIS, uint8_t AGCIDX)
void getCurrentReceivedSignalQuality (uint8_t INTACK)
void getCurrentReceivedSignalQuality (void)
uint8_t getCurrentSNR ()
current receive signal strength (0â€“127 dBÎ¼V).

bool getCurrentRssiDetectLow ()
current SNR metric (0–127 dB).

bool getCurrentRssiDetectHigh ()
RSSI Detect Low.

bool getCurrentSnrDetectLow ()
RSSI Detect High.

bool getCurrentSnrDetectHigh ()
SNR Detect Low.

bool getCurrentValidChannel ()
SNR Detect High.

bool getCurrentAfcRailIndicator ()
Valid Channel.

bool getCurrentSoftMuteIndicator ()
AFC Rail Indicator.

uint8_t getCurrentStereoBlend ()
Soft Mute Indicator. Indicates soft mute is engaged.

bool getCurrentPilot ()
Indicates amount of stereo blend in % (100 = full stereo, 0 = full mono).

uint8_t getCurrentMultipath ()
Indicates stereo pilot presence.

uint8_t getCurrentSignedFrequencyOffset ()
Contains the current multipath metric. (0 = no multipath; 100 = full multipath)

bool getCurrentMultipathDetectLow ()
Signed frequency offset (kHz).

bool getCurrentMultipathDetectHigh ()
Multipath Detect Low.

bool getCurrentBlendDetectInterrupt ()
Multipath Detect High.

uint8_t getFirmwarePN ()
Blend Detect Interrupt.

uint8_t getFirmwareFWMAJOR ()
RESP1 - Part Number (HEX)

uint8_t getFirmwareFWMINOR ()
RESP2 - Returns the Firmware Major Revision (ASCII).

uint8_t getFirmwarePATCHH ()
RESP3 - Returns the Firmware Minor Revision (ASCII).

uint8_t getFirmwarePATCHL ()
RESP4 - Returns the Patch ID High byte (HEX).

uint8_t getFirmwareCMPMAJOR ()
RESP5 - Returns the Patch ID Low byte (HEX).

uint8_t getFirmwareCMPMINOR ()
RESP6 - Returns the Component Major Revision (ASCII).

uint8_t getFirmwareCHIPREV ()

RESP7 - Returns the Component Minor Revision (ASCII).

void setVolume (uint8_t volume)

RESP8 - Returns the Chip Revision (ASCII).

uint8_t getVolume ()
void volumeDown ()
void volumeUp ()
void setAudioMute (bool off)

Returns the current volume level.

void digitalOutputFormat (uint8_t OSIZE, uint8_t OMONO, uint8_t OMODE, uint8_t OFALL)
void digitalOutputSampleRate (uint16_t DOSR)
void setAM ()
void setFM ()
void setAM (uint16_t fromFreq, uint16_t toFreq, uint16_t intialFreq, uint16_t step)
void setFM (uint16_t fromFreq, uint16_t toFreq, uint16_t initialFreq, uint16_t step)
void setBandwidth (uint8_t AMCHFLT, uint8_t AMPLFLT)
void setFrequencyStep (uint16_t step)
void setTuneFrequencyFast (uint8_t FAST)

Returns the FAST tuning status.

uint8_t getTuneFrequencyFreeze ()

FAST Tuning. If set, executes fast and invalidated tune. The tune status will not be accurate.

void setTuneFrequencyFreeze (uint8_t FREEZE)

Returns the FREEZE status.

void setTuneFrequencyAntennaCapacitor (uint16_t capacitor)

Only FM. Freeze Metrics During Alternate Frequency Jump.

void frequencyUp ()
void frequencyDown ()
bool isCurrentTuneFM ()
void getFirmware (void)
void seekStation (uint8_t SEEKUP, uint8_t WRAP)
void seekStationUp ()
void seekStationDown ()
void setSeekAmLimits (uint16_t bottom, uint16_t top)
void setSeekAmSpacing (uint16_t spacing)
void setSeekSrnThreshold (uint16_t value)
void setSeekRssiThreshold (uint16_t value)
void setFmBlendStereoThreshold (uint8_t parameter)
void setFmBlendMonoThreshold (uint8_t parameter)
void setFmBlendRssiStereoThreshold (uint8_t parameter)
void setFmBLendRssiMonoThreshold (uint8_t parameter)
void setFmBlendSnrStereoThreshold (uint8_t parameter)
void setFmBLendSnrMonoThreshold (uint8_t parameter)
void setFmBlendMultiPathStereoThreshold (uint8_t parameter)
void setFmBlendMultiPathMonoThreshold (uint8_t parameter)
void setFmStereoOn ()
void setFmStereoOff ()
void RdsInit ()

void setRdsIntSource (uint8_t RDSNEWBLOCKB, uint8_t RDSNEWBLOCKA, uint8_t RDSSYNCFOUND, uint8_t RDSSYNCLOST, uint8_t RDSRECV)
void getRdsStatus (uint8_t INTACK, uint8_t MTFIFO, uint8_t STATUSONLY)
void getRdsStatus ()
bool getRdsSyncLost ()
1 = FIFO filled to minimum number of groups

bool getRdsSyncFound ()
1 = Lost RDS synchronization

bool getRdsNewBlockA ()
1 = Found RDS synchronization

bool getRdsNewBlockB ()
1 = Valid Block A data has been received.

bool getRdsSync ()
1 = Valid Block B data has been received.

bool getGroupLost ()
1 = RDS currently synchronized.

uint8_t getNumRdsFifoUsed ()
1 = One or more RDS groups discarded due to FIFO overrun.

void setRdsConfig (uint8_t RDSEN, uint8_t BLETHA, uint8_t BLETHB, uint8_t BLETHC, uint8_t BLETHD)
RESP3 - RDS FIFO Used; Number of groups remaining in the RDS FIFO (0 if empty).

uint16_t getRdsPI (void)
uint8_t getRdsGroupType (void)
uint8_t getRdsFlagAB (void)
uint8_t getRdsVersionCode (void)
uint8_t getRdsProgramType (void)
uint8_t getRdsTextSegmentAddress (void)
char * getRdsText (void)
char * getRdsText0A (void)
char * getRdsText2A (void)
char * getRdsText2B (void)
char * getRdsTime (void)
void getNext2Block (char *)
void getNext4Block (char *)
void ssbSetup ()
void setSSBBfo (int offset)
void setSSBConfig (uint8_t AUDIOBW, uint8_t SBCUTFLT, uint8_t AVC_DIVIDER, uint8_t AVCEN, uint8_t SMUTESEL, uint8_t DSP_AFCDIS)
void setSSB (uint8_t usblsb)
void setSSBAudioBandwidth (uint8_t AUDIOBW)
void setSSBAutomaticVolumeControl (uint8_t AVCEN)
void setSBBSidebandCutoffFilter (uint8_t SBCUTFLT)
void setSSBAvcDivider (uint8_t AVC_DIVIDER)
void setSSBDspAfc (uint8_t DSP_AFCDIS)
void setSSBSoftMute (uint8_t SMUTESEL)

si47x_firmware_query_library queryLibraryId ()
void patchPowerUp ()
bool downloadPatch (const uint8_t *ssb_patch_content, const uint16_t ssb_patch_content_size)
bool downloadPatch (int eeprom_i2c_address)
void ssbPowerUp ()
void setI2CStandardMode (void)
Sets I2C buss to 10KHz.

void setI2CFastMode (void)
Sets I2C buss to 100KHz.

void setI2CFastModeCustom (long value=500000)
Sets I2C buss to 400KHz.

void setDeviceI2CAddress (uint8_t senPin)
int16_t getDeviceI2CAddress (uint8_t resetPin)
void setDeviceOtherI2CAddress (uint8_t i2cAddr)

Protected Member Functions

void waitInterrupr (void)
void sendProperty (uint16_t propertyValue, uint16_t param)
void sendSSBModeProperty ()
void disableFmDebug ()
void clearRdsBuffer2A ()
void clearRdsBuffer2B ()
void clearRdsBuffer0A ()

Protected Attributes

char rds_buffer2B [33]
RDS Radio Text buffer - Program Information.

char rds_buffer0A [9]
RDS Radio Text buffer - Station Informaation.

char rds_time [20]
RDS Basic tuning and switching information (Type 0 groups)

int rdsTextAdress2A
RDS date time received information

int rdsTextAdress2B
rds_buffer2A current position

int rdsTextAdress0A
rds_buffer2B current position

int16_t deviceAddress = SI473X_ADDR_SEN_LOW
rds_buffer0A current position

uint8_t lastTextFlagAB
current I2C buss address

uint8_t interruptPin
pin used on Arduino Board to RESET the Si47XX device

uint8_t currentTune
pin used on Arduino Board to control interrupt. If -1, interrupt is no used.

uint16_t currentMinimumFrequency
tell the current tune (FM, AM or SSB)

uint16_t currentMaximumFrequency
minimum frequency of the current band

uint16_t currentWorkFrequency
maximum frequency of the current band

uint16_t currentStep
current frequency

uint8_t lastMode = -1
current steps

uint8_t currentAvcAmMaxGain = 48
Store the last mode used.

si47x_frequency currentFrequency
Automatic Volume Control Gain for AM - Default 48.

si47x_set_frequency currentFrequencyParams
data structure to get current frequency

si47x_response_status currentStatus
current Radio SIgnal Quality status

si47x_firmware_information firmwareInfo
current device status

si47x_rds_status currentRdsStatus
firmware information

si47x_agc_status currentAgcStatus
current RDS status

si47x_ssb_mode currentSSBMode
current AGC status

si473x_powerup powerUp
indicates if USB or LSB

Detailed Description

SI4735 Class.

SI4735 Class definition

This class implements all functions to help you to control the Si47XX devices. This library was built based on “Si47XX PROGRAMMING GUIDE; AN332 ”. It also can be used on all members of the SI473X family respecting, of course, the features available for each IC version. These functionalities can be seen in the comparison matrix shown in table 1 (Product Family Function); pages 2 and 3 of the programming guide.

Author

PU2CLR - Ricardo Lima Caratti

Definition at line 873 of file SI4735.h.

Constructor & Destructor Documentation

SI4735::SI4735 ()

This is a library for the SI4735, BROADCAST AM/FM/SW RADIO RECEIVER, IC from Silicon Labs for the Arduino development environment. It works with I2C protocol. This library is intended to provide an easier interface for controlling the SI4735.

See also

documentation on https://github.com/pu2clr/SI4735.

Si47XX PROGRAMMING GUIDE; AN332

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; AMENDMENT FOR SI4735-D60 SSB AND NBFM PATCHES

ATTENTION: According to Si47XX PROGRAMMING GUIDE; AN332; page 207, "For write operations, the system controller next sends a data byte on SDIO, which is captured by the device on rising edges of SCLK. The device acknowledges each data byte by driving SDIO low for one cycle on the next falling edge of SCLK. The system controller may write up to 8 data bytes in a single 2-wire transaction. The first byte is a command, and the next seven bytes are arguments. Writing more than 8 bytes results in unpredictable device behavior". So, If you are extending this library, consider that restriction presented earlier.

ATTENTION: Some methods were implemented usin inline resource. Inline methods are implemented in SI4735.h

Author

PU2CLR - Ricardo Lima Caratti

By Ricardo Lima Caratti, Nov 2019. Construct a new SI4735::SI4735 object

Definition at line 31 of file SI4735.cpp.

```
00032 {
00033     // 1 = LSB and 2 = USB; 0 = AM, FM or WB
00034     currentSsbStatus = 0;
00035 }
```

Member Function Documentation

void SI4735::analogPowerUp (void)

Powerup in Analog Mode. It will be deprecated. Consider use radioPowerUp instead. Actually this function works fo Digital and Analog modes. You have to call setPowerUp method before.

Definition at line 226 of file SI4735.cpp.

```
00227 {
00228     radioPowerUp();
00229 }
```

References radioPowerUp().

void SI4735::clearRdsBuffer0A () [protected]

Clear RDS buffer 0A (text)

Definition at line 1233 of file SI4735.cpp.

```
01234 {
01235     for (int i = 0; i < 9; i++)
01236         rds_buffer0A[i] = ' '; // Station Name buffer
01237 }
```

References rds_buffer0A.

Referenced by getRdsStatus(), and RdsInit().

void SI4735::clearRdsBuffer2A () [protected]

Clear RDS buffer 2A (text)

Definition at line 1214 of file SI4735.cpp.

```
01215 {
01216     for (int i = 0; i < 65; i++)
01217         rds_buffer2A[i] = ' '; // Radio Text buffer - Program Information
01218 }
```

Referenced by getRdsStatus(), and RdsInit().

void SI4735::clearRdsBuffer2B () [protected]

Clear RDS buffer 2B (text)

Definition at line 1224 of file SI4735.cpp.

```
01225 {
01226     for (int i = 0; i < 33; i++)
01227         rds_buffer2B[i] = ' '; // Radio Text buffer - Station Informaation
01228 }
```

References rds_buffer2B.

Referenced by getRdsStatus(), and RdsInit().

void SI4735::digitalOutputFormat (uint8_t *OSIZE*, uint8_t *OMONO*, uint8_t *OMODE*, uint8_t *OFALL*)

Digital Audio Setup Configures the digital audio output format. Options: DCLK edge, data format, force mono, and sample precision.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 195.

Parameters

uint8_t	OSIZE Digital Output Audio Sample Precision (0=16 bits, 1=20 bits, 2=24 bits, 3=8bits).
uint8_t	OMONO Digital Output Mono Mode (0=Use mono/stereo blend).
uint8_t	OMODE Digital Output Mode (0=I2S, 6 = Left-justified, 8 = MSB at second DCLK after DFS pulse, 12 = MSB at first DCLK after DFS pulse).

uint8_t	OFALL Digital Output DCLK Edge (0 = use DCLK rising edge, 1 = use DCLK falling edge)

Definition at line 778 of file SI4735.cpp.

```
00779 {
00780     si4735_digital_output_format df;
00781     df.refined.OSIZE = OSIZE;
00782     df.refined.OMONO = OMONO;
00783     df.refined.OMODE = OMODE;
00784     df.refined.OFALL = OFALL;
00785     sendProperty(DIGITAL_OUTPUT_FORMAT, df.raw);
00786 }
```

References si4735_digital_output_format::OFALL, si4735_digital_output_format::OMODE, si4735_digital_output_format::OMONO, and sendProperty().

void SI4735::digitalOutputSampleRate (uint16_t *DOSR*)

Enables digital audio output and configures digital audio output sample rate in samples per second (sps).

See also

Si47XX PROGRAMMING GUIDE; AN332; page 196.

Parameters

uint16_t	DOSR Digital Output Sample Rate(32–48 ksps .0 to disable digital audio output).

Definition at line 795 of file SI4735.cpp.

```
00796 {
00797     sendProperty(DIGITAL_OUTPUT_SAMPLE_RATE, DOSR);
00798 }
```

References sendProperty().

void SI4735::disableFmDebug () [protected]

There is a debug feature that remains active in Si4704/05/3x-D60 firmware which can create periodic noise in audio. Silicon Labs recommends you disable this feature by sending the following bytes (shown here in hexadecimal form): 0x12 0x00 0xFF 0x00 0x00 0x00.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 299.

Definition at line 750 of file SI4735.cpp.

```
00751 {
00752     Wire.beginTransmission(deviceAddress);
00753     Wire.write(0x12);
00754     Wire.write(0x00);
00755     Wire.write(0xFF);
00756     Wire.write(0x00);
00757     Wire.write(0x00);
00758     Wire.write(0x00);
00759     Wire.endTransmission();
00760     delayMicroseconds(2500);
00761 }
```

References deviceAddress.

Referenced by setFM().

bool SI4735::downloadPatch (const uint8_t * *ssb_patch_content*, const uint16_t *ssb_patch_content_size*)

Transfers the content of a patch stored in a array of bytes to the SI4735 device. You must mount an array as shown below and know the size of that array as well.

It is importante to say that patches to the SI4735 are distributed in binary form and have to be transferred to the internal RAM of the device by the host MCU (in this case

Arduino). Since the RAM is volatile memory, the patch stored into the device gets lost when you turn off the system. Consequently, the content of the patch has to be transferred again to the device each time after turn on the system or reset the device.

The disadvantage of this approach is the amount of memory used by the patch content. This may limit the use of other radio functions you want implemented in Arduino.

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 64 and 215-220.

Example of content: const PROGMEM uint8_t ssb_patch_content_full[] = { // SSB patch for whole SSBRX full download 0x15, 0x00, 0x0F, 0xE0, 0xF2, 0x73, 0x76, 0x2F, 0x16, 0x6F, 0x26, 0x1E, 0x00, 0x4B, 0x2C, 0x58, 0x16, 0xA3, 0x74, 0x0F, 0xE0, 0x4C, 0x36, 0xE4, 0x16, 0x3B, 0x1D, 0x4A, 0xEC, 0x36, 0x28, 0xB7, 0x16, 0x00, 0x3A, 0x47, 0x37, 0x00, 0x00, 0x00, 0x15, 0x00, 0x00, 0x00, 0x00, 0x00, 0x9D, 0x29};

const int size_content_full = sizeof ssb_patch_content_full;

Parameters

ssb_patch_content	point to array of bytes content patch.
ssb_patch_content_size	array size (number of bytes). The maximum size allowed for a patch is 15856 bytes

Returns

false if an error is found.

Definition at line 2170 of file SI4735.cpp.

```
02171 {
02172     uint8_t content;
02173     register int i, offset;
02174     // Send patch to the SI4735 device
02175     for (offset = 0; offset < (int) ssb_patch_content_size; offset += 8)
02176     {
02177         Wire.beginTransmission(deviceAddress);
02178         for (i = 0; i < 8; i++)
02179         {
02180             content = pgm_read_byte_near(ssb_patch_content + (i + offset));
02181             Wire.write(content);
02182         }
02183         Wire.endTransmission();
02184
02185         // Testing download performance
02186         // approach 1 - Faster - less secure (it might crash in some
architectures)
02187         delayMicroseconds(MIN_DELAY_WAIT_SEND_LOOP); // Need check the
minimum value
02188
02189         // approach 2 - More control. A little more secure than approach 1
02190         /*
02191         do
02192         {
02193             delayMicroseconds(150); // Minimum delay founded (Need check the
minimum value)
02194             Wire.requestFrom(deviceAddress, 1);
02195         } while (!(Wire.read() & B10000000));
02196         */
02197
02198         // approach 3 - same approach 2
02199         // waitToSend();
02200
02201         // approach 4 - safer
02202         /*
02203         waitToSend();
02204         uint8_t cmd_status;
02205         Uncomment the lines below if you want to check erro.
02206         Wire.requestFrom(deviceAddress, 1);
02207         cmd_status = Wire.read();
02208         The SI4735 issues a status after each 8 byte transfered.
02209         Just the bit 7 (CTS) should be seted. if bit 6 (ERR) is seted, the
system halts.
02210         if (cmd_status != 0x80)
```

```
02211             return false;
02212         */
02213     }
02214     delayMicroseconds(250);
02215     return true;
02216 }
```

References deviceAddress.

bool SI4735::downloadPatch (int *eeprom_i2c_address*)

Under construction... Transfers the content of a patch stored in a eeprom to the SI4735 device.

TO USE THIS METHOD YOU HAVE TO HAVE A EEPROM WRITEN WITH THE PATCH CONTENT

See also

the sketch write_ssb_patch_eeprom.ino (TO DO)

Parameters

eeprom_i2c_address	

Returns

false if an error is found.

Definition at line 2229 of file SI4735.cpp.

```
02230 {
02231     int ssb_patch_content_size;
02232     uint8_t cmd_status;
02233     int i, offset;
02234     uint8_t eepromPage[8];
02235 
02236     union {
02237         struct
02238         {
02239             uint8_t lowByte;
02240             uint8_t highByte;
02241         } raw;
02242         uint16_t value;
02243     } eeprom;
02244 
02245     // The first two bytes are the size of the patches
02246     // Set the position in the eeprom to read the size of the patch content
02247     Wire.beginTransmission(eeprom_i2c_address);
02248     Wire.write(0); // writes the most significant byte
02249     Wire.write(0); // writes the less significant byte
02250     Wire.endTransmission();
02251     Wire.requestFrom(eeprom_i2c_address, 2);
02252     eeprom.raw.highByte = Wire.read();
02253     eeprom.raw.lowByte = Wire.read();
02254 
02255     ssb_patch_content_size = eeprom.value;
02256 
02257     // the patch content starts on position 2 (the first two bytes are the
size of the patch)
02258     for (offset = 2; offset < ssb_patch_content_size; offset += 8)
02259     {
02260         // Set the position in the eeprom to read next 8 bytes
02261         eeprom.value = offset;
02262         Wire.beginTransmission(eeprom_i2c_address);
02263         Wire.write(eeprom.raw.highByte); // writes the most significant byte
02264         Wire.write(eeprom.raw.lowByte);  // writes the less significant byte
02265         Wire.endTransmission();
02266 
02267         // Reads the next 8 bytes from eeprom
02268         Wire.requestFrom(eeprom_i2c_address, 8);
02269         for (i = 0; i < 8; i++)
02270             eepromPage[i] = Wire.read();
02271 
02272         // sends the page (8 bytes) to the SI4735
02273         Wire.beginTransmission(deviceAddress);
02274         for (i = 0; i < 8; i++)
```

```
02275             Wire.write(eepromPage[i]);
02276         Wire.endTransmission();
02277
02278         waitToSend();
02279
02280         Wire.requestFrom(deviceAddress, 1);
02281         cmd_status = Wire.read();
02282         // The SI4735 issues a status after each 8 byte transfered.
02283         // Just the bit 7 (CTS) should be seted. if bit 6 (ERR) is seted,
the system halts.
02284         if (cmd_status != 0x80)
02285             return false;
02286     }
02287     delayMicroseconds(250);
02288     return true;
02289 }
```

References deviceAddress, and waitToSend().

void SI4735::frequencyDown ()

Decrements the current frequency on current band/function by using the current step.

See also

setFrequencyStep

Definition at line 443 of file SI4735.cpp.

```
00444 {
00445
00446     if (currentWorkFrequency <= currentMinimumFrequency)
00447         currentWorkFrequency = currentMaximumFrequency;
00448     else
00449         currentWorkFrequency -= currentStep;
00450
00451     setFrequency(currentWorkFrequency);
00452 }
```

References currentMaximumFrequency, currentMinimumFrequency, currentStep, currentWorkFrequency, and setFrequency().

void SI4735::frequencyUp ()

Increments the current frequency on current band/function by using the current step.

See also

setFrequencyStep()

Definition at line 428 of file SI4735.cpp.

```
00429 {
00430     if (currentWorkFrequency >= currentMaximumFrequency)
00431         currentWorkFrequency = currentMinimumFrequency;
00432     else
00433         currentWorkFrequency += currentStep;
00434
00435     setFrequency(currentWorkFrequency);
00436 }
```

References currentMaximumFrequency, currentMinimumFrequency, currentStep, currentWorkFrequency, and setFrequency().

void SI4735::getAutomaticGainControl ()

Returns integer containing the current antenna tuning capacitor value.

Queries AGC STATUS

See also

Si47XX PROGRAMMING GUIDE; AN332; For FM page 80; for AM page 142.

AN332 REV 0.8 Universal Programming Guide Amendment for SI4735-D60 SSB and NBFM patches; page 18.

After call this method, you can call isAgcEnabled to know the AGC status and getAgcGainIndex to know the gain index value.

Definition at line 886 of file SI4735.cpp.

```
00887 {
00888     uint8_t cmd;
00889
00890     if (currentTune == FM_TUNE_FREQ)
00891     { // FM TUNE
00892         cmd = FM_AGC_STATUS;
00893     }
00894     else
00895     { // AM TUNE - SAME COMMAND used on SSB mode
00896         cmd = AM_AGC_STATUS;
00897     }
00898
00899     waitToSend();
00900
00901     Wire.beginTransmission(deviceAddress);
00902     Wire.write(cmd);
00903     Wire.endTransmission();
00904
00905     do
00906     {
00907         waitToSend();
00908         Wire.requestFrom(deviceAddress, 3);
00909         currentAgcStatus.raw[0] = Wire.read(); // STATUS response
00910         currentAgcStatus.raw[1] = Wire.read(); // RESP 1
00911         currentAgcStatus.raw[2] = Wire.read(); // RESP 2
00912     } while (currentAgcStatus.refined.ERR);    // If error, try get AGC
status again.
00913 }
```

References currentAgcStatus, currentTune, deviceAddress, and waitToSend().

uint16_t SI4735::getCurrentFrequency ()

Gets the current frequency saved in memory. Unlike getFrequency, this method gets the current frequency recorded after the last setFrequency command. This method avoids bus traffic and CI processing. However, you can not get others status information like RSSI.

See also

getFrequency()

Definition at line 830 of file SI4735.cpp.

```
00831 {
00832     return currentWorkFrequency;
00833 }
```

References currentWorkFrequency.

void SI4735::getCurrentReceivedSignalQuality (uint8_t *INTACK*)

Queries the status of the Received Signal Quality (RSQ) of the current channel. This method sould be called berore call getCurrentRSSI(), getCurrentSNR() etc. Command FM_RSQ_STATUS

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 75 and 141

Parameters

INTACK	Interrupt Acknowledge. 0 = Interrupt status preserved; 1 = Clears RSQINT, BLENDINT, SNRHINT, SNRLINT, RSSIHINT, RSSILINT, MULTHINT, MULTLINT.

Definition at line 975 of file SI4735.cpp.

```
00976 {
00977         uint8_t arg;
00978         uint8_t cmd;
00979         int sizeResponse;
00980
```

```
00981         if (currentTune == FM_TUNE_FREQ)
00982         { // FM TUNE
00983             cmd = FM_RSQ_STATUS;
00984             sizeResponse = 8; // Check it
00985         }
00986         else
00987         { // AM TUNE
00988             cmd = AM_RSQ_STATUS;
00989             sizeResponse = 6; // Check it
00990         }
00991 
00992         waitToSend();
00993 
00994         arg = INTACK;
00995         Wire.beginTransmission(deviceAddress);
00996         Wire.write(cmd);
00997         Wire.write(arg); // send B00000001
00998         Wire.endTransmission();
00999 
01000         // Check it
01001         // do
01002         //{
01003             waitToSend();
01004             Wire.requestFrom(deviceAddress, sizeResponse);
01005             // Gets response information
01006             for (uint8_t i = 0; i < sizeResponse; i++)
01007                 currentRqsStatus.raw[i] = Wire.read();
01008         //} while (currentRqsStatus.resp.ERR); // Try again if error found
01009 }
```

References currentTune, deviceAddress, and waitToSend().

void SI4735::getCurrentReceivedSignalQuality (void)

Queries the status of the Received Signal Quality (RSQ) of the current channel Command FM_RSQ_STATUS

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 75 and 141

Parameters

INTACK	Interrupt Acknowledge. 0 = Interrupt status preserved; 1 = Clears RSQINT, BLENDINT, SNRHINT, SNRLINT, RSSIHINT, RSSILINT, MULTHINT, MULTLINT.

Definition at line 1021 of file SI4735.cpp.

```
01022 {
01023     getCurrentReceivedSignalQuality(0);
01024 }
```

int16_t SI4735::getDeviceI2CAddress (uint8_t *resetPin*)

Scans for two possible addresses for the Si47XX (0x11 or 0x63) This function also sets the system to the found I2C bus address of Si47XX.

You do not need to use this function if the SEN PIN is configured to ground (GND). The default I2C address is 0x11. Use this function if you do not know how the SEN pin is configured.

Parameters

uint8_t	resetPin MCU Mater (Arduino) reset pin

Returns

int16_t 0x11 if the SEN pin of the Si47XX is low or 0x63 if the SEN pin of the Si47XX is HIGH or 0x0 if error.

Definition at line 64 of file SI4735.cpp.

```
00064                                                   {
00065     int16_t error;
00066 
```

```
00067     pinMode(resetPin, OUTPUT);
00068     delay(50);
00069     digitalWrite(resetPin, LOW);
00070     delay(50);
00071     digitalWrite(resetPin, HIGH);
00072
00073     Wire.begin();
00074     // check 0X11 I2C address
00075     Wire.beginTransmission(SI473X_ADDR_SEN_LOW);
00076     error = Wire.endTransmission();
00077     if ( error == 0 ) {
00078       setDeviceI2CAddress(0);
00079       return SI473X_ADDR_SEN_LOW;
00080     }
00081
00082     // check 0X63 I2C address
00083     Wire.beginTransmission(SI473X_ADDR_SEN_HIGH);
00084     error = Wire.endTransmission();
00085     if ( error == 0 ) {
00086       setDeviceI2CAddress(1);
00087       return SI473X_ADDR_SEN_HIGH;
00088     }
00089
00090     // Did find the device
00091     return 0;
00092 }
```

References setDeviceI2CAddress().

void SI4735::getFirmware (void)

Gets firmware information

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 66, 131

Definition at line 251 of file SI4735.cpp.

```
00252 {
00253     waitToSend();
00254
00255     Wire.beginTransmission(deviceAddress);
00256     Wire.write(GET_REV);
00257     Wire.endTransmission();
00258
00259     do
00260     {
00261         waitToSend();
00262         // Request for 9 bytes response
00263         Wire.requestFrom(deviceAddress, 9);
00264         for (int i = 0; i < 9; i++)
00265             firmwareInfo.raw[i] = Wire.read();
00266     } while (firmwareInfo.resp.ERR);
00267 }
```

References deviceAddress, firmwareInfo, and waitToSend().

uint16_t SI4735::getFrequency (void)

Device Status Information Gets the current frequency of the Si4735 (AM or FM) The method status do it an more. See getStatus below.

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 73 (FM) and 139 (AM)

Definition at line 810 of file SI4735.cpp.

```
00811 {
00812     si47x_frequency freq;
00813     getStatus(0, 1);
00814
00815     freq.raw.FREQL = currentStatus.resp.READFREQL;
00816     freq.raw.FREQH = currentStatus.resp.READFREQH;
00817
00818     currentWorkFrequency = freq.value;
00819     return freq.value;
```

```
00820 }
```

References currentStatus, currentWorkFrequency, si47x_frequency::FREQH, getStatus(), si47x_response_status::READFREQH, and si47x_response_status::READFREQL.

Referenced by seekStationDown(), and seekStationUp().

void SI4735::getNext2Block (char * *c*)

Process data received from group 2B

Parameters

c	char array reference to the "group 2B" text

Definition at line 1507 of file SI4735.cpp.

```
01508 {
01509     char raw[2];
01510     int i, j;
01511
01512     raw[1] = currentRdsStatus.resp.BLOCKDL;
01513     raw[0] = currentRdsStatus.resp.BLOCKDH;
01514
01515     for (i = j = 0; i < 2; i++)
01516     {
01517         if (raw[i] == 0xD || raw[i] == 0xA)
01518         {
01519             c[j] = '\0';
01520             return;
01521         }
01522         if (raw[i] >= 32)
01523         {
01524             c[j] = raw[i];
01525             j++;
01526         }
01527         else
01528         {
01529             c[i] = ' ';
01530         }
01531     }
01532 }
```

References si47x_rds_status::BLOCKDH, si47x_rds_status::BLOCKDL, and currentRdsStatus.

Referenced by getRdsText0A(), and getRdsText2B().

void SI4735::getNext4Block (char * *c*)

Process data received from group 2A

Parameters

c	char array reference to the "group 2A" text

Definition at line 1539 of file SI4735.cpp.

```
01540 {
01541     char raw[4];
01542     int i, j;
01543
01544     raw[0] = currentRdsStatus.resp.BLOCKCH;
01545     raw[1] = currentRdsStatus.resp.BLOCKCL;
01546     raw[2] = currentRdsStatus.resp.BLOCKDH;
01547     raw[3] = currentRdsStatus.resp.BLOCKDL;
01548     for (i = j = 0; i < 4; i++)
01549     {
01550         if (raw[i] == 0xD || raw[i] == 0xA)
01551         {
01552             c[j] = '\0';
01553             return;
01554         }
01555         if (raw[i] >= 32)
01556         {
01557             c[j] = raw[i];
01558             j++;
01559         }
01560         else
01561         {
```

```
01562             c[i] = ' ';
01563         }
01564     }
01565 }
```

References si47x_rds_status::BLOCKCH, si47x_rds_status::BLOCKCL, si47x_rds_status::BLOCKDH, si47x_rds_status::BLOCKDL, and currentRdsStatus.

Referenced by getRdsText(), and getRdsText2A().

uint8_t SI4735::getRdsFlagAB (void)

Returns the current Text Flag A/B

Returns

uint8_t

Definition at line 1441 of file SI4735.cpp.

```
01442 {
01443     si47x_rds_blockb blkb;
01444
01445     blkb.raw.lowValue = currentRdsStatus.resp.BLOCKBL;
01446     blkb.raw.highValue = currentRdsStatus.resp.BLOCKBH;
01447
01448     return blkb.refined.textABFlag;
01449 }
```

References si47x_rds_status::BLOCKBH, si47x_rds_status::BLOCKBL, and currentRdsStatus.

uint8_t SI4735::getRdsGroupType (void)

Returns the Group Type (extracted from the Block B)

Definition at line 1425 of file SI4735.cpp.

```
01426 {
01427     si47x_rds_blockb blkb;
01428
01429     blkb.raw.lowValue = currentRdsStatus.resp.BLOCKBL;
01430     blkb.raw.highValue = currentRdsStatus.resp.BLOCKBH;
01431
01432     return blkb.refined.groupType;
01433 }
```

References si47x_rds_status::BLOCKBH, si47x_rds_status::BLOCKBL, and currentRdsStatus.

Referenced by getRdsText0A(), getRdsText2A(), getRdsText2B(), and getRdsTime().

uint16_t SI4735::getRdsPI (void)

Returns the programa type. Read the Block A content

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 77 and 78

Returns

BLOCKAL

Definition at line 1413 of file SI4735.cpp.

```
01414 {
01415     if (getRdsReceived() && getRdsNewBlockA())
01416     {
01417         return currentRdsStatus.resp.BLOCKAL;
01418     }
01419     return 0;
01420 }
```

References si47x_rds_status::BLOCKAL, currentRdsStatus, and getRdsNewBlockA().

uint8_t SI4735::getRdsProgramType (void)

Returns the Program Type (extracted from the Block B)

See also

https://en.wikipedia.org/wiki/Radio_Data_System

Returns

program type (an integer betwenn 0 and 31)

Definition at line 1492 of file SI4735.cpp.

```
01493 {
01494     si47x_rds_blockb blkb;
01495
01496     blkb.raw.lowValue = currentRdsStatus.resp.BLOCKBL;
01497     blkb.raw.highValue = currentRdsStatus.resp.BLOCKBH;
01498
01499     return blkb.refined.programType;
01500 }
```

References si47x_rds_status::BLOCKBH, si47x_rds_status::BLOCKBL, and currentRdsStatus.

void SI4735::getRdsStatus ()

Gets RDS Status. Same result of calling getRdsStatus(0,0,0);

See also

SI4735::getRdsStatus(uint8_t INTACK, uint8_t MTFIFO, uint8_t STATUSONLY)

Please, call getRdsStatus(uint8_t INTACK, uint8_t MTFIFO, uint8_t STATUSONLY) instead getRdsStatus() if you want other behaviour

Definition at line 1398 of file SI4735.cpp.

```
01399 {
01400     getRdsStatus(0, 0, 0);
01401 }
```

void SI4735::getRdsStatus (uint8_t *INTACK*, uint8_t *MTFIFO*, uint8_t *STATUSONLY*)

Gets the RDS status. Store the status in currentRdsStatus member. RDS COMMAND FM_RDS_STATUS

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 55 and 77

Parameters

INTACK	Interrupt Acknowledge; 0 = RDSINT status preserved. 1 = Clears RDSINT.
MTFIFO	0 = If FIFO not empty, read and remove oldest FIFO entry; 1 = Clear RDS Receive FIFO.
STATUSONLY	Determines if data should be removed from the RDS FIFO.

Definition at line 1351 of file SI4735.cpp.

```
01352 {
01353     si47x_rds_command rds_cmd;
01354     static uint16_t lastFreq;
01355     // checking current FUNC (Am or FM)
01356     if (currentTune != FM_TUNE_FREQ)
01357         return;
01358
01359     if (lastFreq != currentWorkFrequency)
01360     {
01361         lastFreq = currentWorkFrequency;
01362         clearRdsBuffer2A();
01363         clearRdsBuffer2B();
01364         clearRdsBuffer0A();
01365     }
01366
01367     waitToSend();
01368
01369     rds_cmd.arg.INTACK = INTACK;
01370     rds_cmd.arg.MTFIFO = MTFIFO;
01371     rds_cmd.arg.STATUSONLY = STATUSONLY;
01372
01373     Wire.beginTransmission(deviceAddress);
01374     Wire.write(FM_RDS_STATUS);
```

```
01375     Wire.write(rds_cmd.raw);
01376     Wire.endTransmission();
01377
01378     do
01379     {
01380         waitToSend();
01381         // Gets response information
01382         Wire.requestFrom(deviceAddress, 13);
01383         for (uint8_t i = 0; i < 13; i++)
01384             currentRdsStatus.raw[i] = Wire.read();
01385     } while (currentRdsStatus.resp.ERR);
01386     delayMicroseconds(550);
01387 }
```

References clearRdsBuffer0A(), clearRdsBuffer2A(), clearRdsBuffer2B(), currentRdsStatus, currentTune, currentWorkFrequency, deviceAddress, and waitToSend().

char * SI4735::getRdsText (void)

Gets the RDS Text when the message is of the Group Type 2 version A

Returns

char* The string (char array) with the content (Text) received from group 2A

Definition at line 1573 of file SI4735.cpp.

```
01574 {
01575
01576     // Needs to get the "Text segment address code".
01577     // Each message should be ended by the code 0D (Hex)
01578
01579     if (rdsTextAdress2A >= 16)
01580         rdsTextAdress2A = 0;
01581
01582     getNext4Block(&rds_buffer2A[rdsTextAdress2A * 4]);
01583
01584     rdsTextAdress2A += 4;
01585
01586     return rds_buffer2A;
01587 }
```

References getNext4Block(), and rdsTextAdress2A.

char * SI4735::getRdsText0A (void)

Gets the station name and other messages.

Returns

char* should return a string with the station name. However, some stations send other kind of messages

Definition at line 1595 of file SI4735.cpp.

```
01596 {
01597     si47x_rds_blockb blkB;
01598
01599     // getRdsStatus();
01600
01601     if (getRdsReceived())
01602     {
01603         if (getRdsGroupType() == 0)
01604         {
01605             // Process group type 0
01606             blkB.raw.highValue = currentRdsStatus.resp.BLOCKBH;
01607             blkB.raw.lowValue = currentRdsStatus.resp.BLOCKBL;
01608
01609             rdsTextAdress0A = blkB.group0.address;
01610             if (rdsTextAdress0A >= 0 && rdsTextAdress0A < 4)
01611             {
01612                 getNext2Block(&rds_buffer0A[rdsTextAdress0A * 2]);
01613                 rds_buffer0A[8] = '\0';
01614                 return rds_buffer0A;
01615             }
01616         }
01617     }
01618     return NULL;
```

```
01619 }
```

References si47x_rds_status::BLOCKBH, si47x_rds_status::BLOCKBL, currentRdsStatus, getNext2Block(), getRdsGroupType(), rds_buffer0A, and rdsTextAdress0A.

char * SI4735::getRdsText2A (void)

Gets the Text processed for the 2A group

Returns

char* string with the Text of the group A2

Definition at line 1626 of file SI4735.cpp.

```
01627 {
01628     si47x_rds_blockb blkB;
01629
01630     // getRdsStatus();
01631     if (getRdsReceived())
01632     {
01633         if (getRdsGroupType() == 2 /* && getRdsVersionCode() == 0 */)
01634         {
01635             // Process group 2A
01636             // Decode B block information
01637             blkB.raw.highValue = currentRdsStatus.resp.BLOCKBH;
01638             blkB.raw.lowValue = currentRdsStatus.resp.BLOCKBL;
01639             rdsTextAdress2A = blkB.group2.address;
01640
01641             if (rdsTextAdress2A >= 0 && rdsTextAdress2A < 16)
01642             {
01643                 getNext4Block(&rds_buffer2A[rdsTextAdress2A * 4]);
01644                 rds_buffer2A[63] = '\0';
01645                 return rds_buffer2A;
01646             }
01647         }
01648     }
01649     return NULL;
01650 }
```

References si47x_rds_status::BLOCKBH, si47x_rds_status::BLOCKBL, currentRdsStatus, getNext4Block(), getRdsGroupType(), and rdsTextAdress2A.

char * SI4735::getRdsText2B (void)

Gets the Text processed for the 2B group

Returns

char* string with the Text of the group AB

Definition at line 1658 of file SI4735.cpp.

```
01659 {
01660     si47x_rds_blockb blkB;
01661
01662     // getRdsStatus();
01663     // if (getRdsReceived())
01664     // {
01665     // if (getRdsNewBlockB())
01666     // {
01667     if (getRdsGroupType() == 2 /* && getRdsVersionCode() == 1 */)
01668     {
01669         // Process group 2B
01670         blkB.raw.highValue = currentRdsStatus.resp.BLOCKBH;
01671         blkB.raw.lowValue = currentRdsStatus.resp.BLOCKBL;
01672         rdsTextAdress2B = blkB.group2.address;
01673         if (rdsTextAdress2B >= 0 && rdsTextAdress2B < 16)
01674         {
01675             getNext2Block(&rds_buffer2B[rdsTextAdress2B * 2]);
01676             return rds_buffer2B;
01677         }
01678     }
01679     //  }
```

```
01680     // }
01681     return NULL;
01682 }
```

References si47x_rds_status::BLOCKBH, si47x_rds_status::BLOCKBL, currentRdsStatus, getNext2Block(), getRdsGroupType(), rds_buffer2B, and rdsTextAdress2B.

uint8_t SI4735::getRdsTextSegmentAddress (void)

Returns the address of the text segment. 2A - Each text segment in version 2A groups consists of four characters. A messages of this group can be have up to 64 characters. 2B - In version 2B groups, each text segment consists of only two characters. When the current RDS status is using this version, the maximum message length will be 32 characters.

Returns

uint8_t the address of the text segment.

Definition at line 1461 of file SI4735.cpp.

```
01462 {
01463     si47x_rds_blockb blkb;
01464     blkb.raw.lowValue = currentRdsStatus.resp.BLOCKBL;
01465     blkb.raw.highValue = currentRdsStatus.resp.BLOCKBH;
01466
01467     return blkb.refined.content;
01468 }
```

References si47x_rds_status::BLOCKBH, si47x_rds_status::BLOCKBL, and currentRdsStatus.

char * SI4735::getRdsTime (void)

Gets the RDS time and date when the Group type is 4

Returns

char* a string with hh:mm +/- offset

Definition at line 1689 of file SI4735.cpp.

```
01690 {
01691     // Under Test and construction
01692     // Need to check the Group Type before.
01693     si47x_rds_date_time dt;
01694
01695     uint16_t minute;
01696     uint16_t hour;
01697
01698     if (getRdsGroupType() == 4)
01699     {
01700         char offset_sign;
01701         int offset_h;
01702         int offset_m;
01703
01704         // uint16_t y, m, d;
01705
01706         dt.raw[4] = currentRdsStatus.resp.BLOCKBL;
01707         dt.raw[5] = currentRdsStatus.resp.BLOCKBH;
01708         dt.raw[2] = currentRdsStatus.resp.BLOCKCL;
01709         dt.raw[3] = currentRdsStatus.resp.BLOCKCH;
01710         dt.raw[0] = currentRdsStatus.resp.BLOCKDL;
01711         dt.raw[1] = currentRdsStatus.resp.BLOCKDH;
01712
01713         // Unfortunately it was necessary to wotk well on the GCC compiler
on 32-bit
01714         // platforms. See si47x_rds_date_time (typedef union) and CGG
“Crosses boundary” issue/features.
01715         // Now it is working on Atmega328, STM32, Arduino DUE, ESP32 and
more.
01716         minute = (dt.refined.minute2 << 2) | dt.refined.minute1;
01717         hour = (dt.refined.hour2 << 4) | dt.refined.hour1;
01718
01719         offset_sign = (dt.refined.offset_sense == 1) ? '+' : '-';
01720         offset_h = (dt.refined.offset * 30) / 60;
01721         offset_m = (dt.refined.offset * 30) - (offset_h * 60);
01722         // sprintf(rds_time, "%02u:%02u %c%02u:%02u", dt.refined.hour,
dt.refined.minute, offset_sign, offset_h, offset_m);
```

```
01723         sprintf(rds_time, "%02u:%02u %c%02u:%02u", hour, minute,
offset_sign, offset_h, offset_m);
01724
01725         return rds_time;
01726     }
01727
01728     return NULL;
01729 }
```

References si47x_rds_status::BLOCKBH, si47x_rds_status::BLOCKBL, si47x_rds_status::BLOCKCH, si47x_rds_status::BLOCKCL, si47x_rds_status::BLOCKDH, si47x_rds_status::BLOCKDL, currentRdsStatus, getRdsGroupType(), and rds_time.

uint8_t SI4735::getRdsVersionCode (void)

Gets the version code (extracted from the Block B)

Returns

0=A or 1=B

Definition at line 1475 of file SI4735.cpp.

```
01476 {
01477     si47x_rds_blockb blkb;
01478
01479     blkb.raw.lowValue = currentRdsStatus.resp.BLOCKBL;
01480     blkb.raw.highValue = currentRdsStatus.resp.BLOCKBH;
01481
01482     return blkb.refined.versionCode;
01483 }
```

References si47x_rds_status::BLOCKBH, si47x_rds_status::BLOCKBL, and currentRdsStatus.

bool SI4735::getSignalQualityInterrupt () [inline]

STATUS RESPONSE Set of methods to get current status information. Call them after getStatus or getFrequency or seekStation See Si47XX PROGRAMMING GUIDE; AN332; pages 63

Definition at line 956 of file SI4735.h.

```
00956 { return currentStatus.resp.RSQINT; };
```

References currentStatus, and si47x_response_status::RSQINT.

void SI4735::getStatus ()

Gets the current status of the Si4735 (AM or FM)

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 73 (FM) and 139 (AM)

Definition at line 873 of file SI4735.cpp.

```
00874 {
00875     getStatus(0, 1);
00876 }
```

Referenced by getFrequency().

void SI4735::getStatus (uint8_t *INTACK*, uint8_t *CANCEL*)

Gets the current status of the Si4735 (AM or FM)

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 73 (FM) and 139 (AM)

Parameters

uint8_t	INTACK Seek/Tune Interrupt Clear. If set, clears the seek/tune complete interrupt status indicator;
uint8_t	CANCEL Cancel seek. If set, aborts a seek currently in progress;

Definition at line 842 of file SI4735.cpp.

```
00843 {
```

```
00844     si47x_tune_status status;
00845     uint8_t cmd = (currentTune == FM_TUNE_FREQ) ? FM_TUNE_STATUS :
AM_TUNE_STATUS;
00846
00847     waitToSend();
00848
00849     status.arg.INTACK = INTACK;
00850     status.arg.CANCEL = CANCEL;
00851
00852     Wire.beginTransmission(deviceAddress);
00853     Wire.write(cmd);
00854     Wire.write(status.raw);
00855     Wire.endTransmission();
00856     // Reads the current status (including current frequency).
00857     do
00858     {
00859         waitToSend();
00860         Wire.requestFrom(deviceAddress, 8); // Check it
00861         // Gets response information
00862         for (uint8_t i = 0; i < 8; i++)
00863             currentStatus.raw[i] = Wire.read();
00864     } while (currentStatus.resp.ERR); // If error, try it again
00865     waitToSend();
00866 }
```

References si47x_tune_status::CANCEL, currentStatus, currentTune, deviceAddress, and waitToSend().

uint8_t SI4735::getVolume ()

Gets the current volume level.

See also

setVolume()

Returns

volume (domain: 0 - 63)

Definition at line 1166 of file SI4735.cpp.

```
01167 {
01168     return this->volume;
01169 }
```

bool SI4735::isCurrentTuneFM ()

Returns true if the current function is FM (FM_TUNE_FREQ).

Returns

true if the current function is FM (FM_TUNE_FREQ).

Definition at line 593 of file SI4735.cpp.

```
00594 {
00595     return (currentTune == FM_TUNE_FREQ);
00596 }
```

References currentTune.

void SI4735::patchPowerUp ()

This method can be used to prepare the device to apply SSBRX patch Call queryLibraryId before call this method. Powerup the device by issuing the POWER_UP command with FUNC = 1 (AM/SW/LW Receive)

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 64 and 215-220 and

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE AMENDMENT FOR SI4735-D60 SSB AND NBFM PATCHES; page 7.

Definition at line 2091 of file SI4735.cpp.

```
02092 {
```

```
02093     waitToSend();
02094     Wire.beginTransmission(deviceAddress);
02095     Wire.write(POWER_UP);
02096     Wire.write(0b00110001);          // Set to AM, Enable External Crystal
Oscillator; Set patch enable; GPO2 output disabled; CTS interrupt disabled.
02097     Wire.write(SI473X_ANALOG_AUDIO); // Set to Analog Output
02098     Wire.endTransmission();
02099     delayMicroseconds(2500);
02100 }
```

References deviceAddress, and waitToSend().

void SI4735::powerDown (void)

Moves the device from powerup to powerdown mode. After Power Down command, only the Power Up command is accepted.

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 67, 132

Definition at line 237 of file SI4735.cpp.

```
00238 {
00239     waitToSend();
00240     Wire.beginTransmission(deviceAddress);
00241     Wire.write(POWER_DOWN);
00242     Wire.endTransmission();
00243     delayMicroseconds(2500);
00244 }
```

References deviceAddress, and waitToSend().

Referenced by queryLibraryId(), setAM(), and setFM().

si47x_firmware_query_library SI4735::queryLibraryId ()

SI47XX PATCH RESOURCES Call it first if you are applying a patch on SI4735. Used to confirm if the patch is compatible with the internal device library revision. See Si47XX PROGRAMMING GUIDE; AN332; pages 64 and 215-220.

Returns

a struct si47x_firmware_query_library (see it in SI4735.h) Query the library information

You have to call this function if you are applying a patch on SI47XX (SI4735-D60)

The first command that is sent to the device is the POWER_UP command to confirm that the patch is compatible with the internal device library revision. The device moves into the powerup mode, returns the reply, and moves into the powerdown mode. The POWER_UP command is sent to the device again to configure the mode of the device and additionally is used to start the patching process. When applying the patch, the PATCH bit in ARG1 of the POWER_UP command must be set to 1 to begin the patching process. [AN332 page 219].

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 214, 215, 216, 219

si47x_firmware_query_library in SI4735.h

Returns

si47x_firmware_query_library Library Identification

Definition at line 2054 of file SI4735.cpp.

```
02055 {
02056     si47x_firmware_query_library libraryID;
02057
02058     powerDown(); // Is it necessary
02059
02060     // delay(500);
02061
02062     waitToSend();
02063     Wire.beginTransmission(deviceAddress);
```

```
02064     Wire.write(POWER_UP);
02065     Wire.write(0b00011111);          // Set to Read Library ID, disable
interrupt; disable GPO2OEN; boot normaly; enable External Crystal Oscillator  .
02066     Wire.write(SI473X_ANALOG_AUDIO); // Set to Analog Line Input.
02067     Wire.endTransmission();
02068
02069     do
02070     {
02071         waitToSend();
02072         Wire.requestFrom(deviceAddress, 8);
02073         for (int i = 0; i < 8; i++)
02074             libraryID.raw[i] = Wire.read();
02075     } while (libraryID.resp.ERR); // If error found, try it again.
02076
02077     delayMicroseconds(2500);
02078
02079     return libraryID;
02080 }
```

References deviceAddress, powerDown(), and waitToSend().

void SI4735::radioPowerUp (void)

Powerup the Si47XX Before call this function call the setPowerUp to set up the parameters. Parameters you have to set up with setPowerUp

CTSIEN Interrupt anabled or disabled; GPO2OEN GPO2 Output Enable or disabled; PATCH Boot normally or patch; XOSCEN Use external crystal oscillator; FUNC defaultFunction = 0 = FM Receive; 1 = AM (LW/MW/SW) Receiver. OPMODE SI473X_ANALOG_AUDIO (B00000101) or SI473X_DIGITAL_AUDIO (B00001011)

See also

SI4735::setPowerUp()

Si47XX PROGRAMMING GUIDE; AN332; pages 64, 129

Definition at line 207 of file SI4735.cpp.

```
00207                                {
00208     // delayMicroseconds(1000);
00209     waitToSend();
00210     Wire.beginTransmission(deviceAddress);
00211     Wire.write(POWER_UP);
00212     Wire.write(powerUp.raw[0]); // Content of ARG1
00213     Wire.write(powerUp.raw[1]); // COntent of ARG2
00214     Wire.endTransmission();
00215     // Delay at least 500 ms between powerup command and first tune command
to wait for
00216     // the oscillator to stabilize if XOSCEN is set and crystal is used as
the RCLK.
00217     waitToSend();
00218     delay(10);
00219 }
```

References deviceAddress, powerUp, and waitToSend().

Referenced by analogPowerUp(), setAM(), setFM(), and setSSB().

void SI4735::RdsInit ()

RDS implementation Starts the control variables for RDS.

Definition at line 1202 of file SI4735.cpp.

```
01203 {
01204     clearRdsBuffer2A();
01205     clearRdsBuffer2B();
01206     clearRdsBuffer0A();
01207     rdsTextAdress2A = rdsTextAdress2B = lastTextFlagAB = rdsTextAdress0A =
0;
01208 }
```

References clearRdsBuffer0A(), clearRdsBuffer2A(), clearRdsBuffer2B(), lastTextFlagAB, rdsTextAdress0A, rdsTextAdress2A, and rdsTextAdress2B.

Referenced by setRdsConfig().

void SI4735::reset (void)

Reset the SI473X

See also

Si47XX PROGRAMMING GUIDE; AN332;

Definition at line 127 of file SI4735.cpp.

```
00128 {
00129     pinMode(resetPin, OUTPUT);
00130     delay(10);
00131     digitalWrite(resetPin, LOW);
00132     delay(10);
00133     digitalWrite(resetPin, HIGH);
00134     delay(10);
00135 }
```

Referenced by ssbSetup().

void SI4735::seekStation (uint8_t *SEEKUP*, uint8_t *WRAP*)

Look for a station

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 55, 72, 125 and 137

Parameters

SEEKUP	Seek Up/Down. Determines the direction of the search, either UP = 1, or DOWN = 0.
Wrap/Halt.	Determines whether the seek should Wrap = 1, or Halt = 0 when it hits the band limit.

Definition at line 1034 of file SI4735.cpp.

```
01035 {
01036     si47x_seek seek;
01037
01038     // Check which FUNCTION (AM or FM) is working now
01039     uint8_t seek_start = (currentTune == FM_TUNE_FREQ) ? FM_SEEK_START :
AM_SEEK_START;
01040
01041     waitToSend();
01042
01043     seek.arg.SEEKUP = SEEKUP;
01044     seek.arg.WRAP = WRAP;
01045
01046     Wire.beginTransmission(deviceAddress);
01047     Wire.write(seek_start);
01048     Wire.write(seek.raw);
01049
01050     if (seek_start == AM_SEEK_START)
01051     {
01052         Wire.write(0x00); // Always 0
01053         Wire.write(0x00); // Always 0
01054         Wire.write(0x00); // Tuning Capacitor: The tuning capacitor value
01055         Wire.write(0x00); //                   will be selected
automatically.
01056     }
01057
01058     Wire.endTransmission();
01059     delay(100);
01060 }
```

References currentTune, deviceAddress, si47x_seek::SEEKUP, and waitToSend().

Referenced by seekStationDown(), and seekStationUp().

void SI4735::seekStationDown ()

Search the previous station

See also

seekStation(uint8_t SEEKUP, uint8_t WRAP)

Definition at line 1079 of file SI4735.cpp.

```
01080 {
01081     seekStation(0, 1);
01082     delay(50);
01083     getFrequency();
01084 }
```

References getFrequency(), and seekStation().

void SI4735::seekStationUp ()

Search for the next station

See also

seekStation(uint8_t SEEKUP, uint8_t WRAP)

Definition at line 1067 of file SI4735.cpp.

```
01068 {
01069     seekStation(1, 1);
01070     delay(50);
01071     getFrequency();
01072 }
```

References getFrequency(), and seekStation().

void SI4735::sendProperty (uint16_t *propertyValue*, uint16_t *parameter*) [protected]

Sends (sets) property to the SI47XX This method is used for others to send generic properties and params to SI47XX

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 68, 124 and 133.

Definition at line 604 of file SI4735.cpp.

```
00605 {
00606     si47x_property property;
00607     si47x_property param;
00608
00609     property.value = propertyValue;
00610     param.value = parameter;
00611     waitToSend();
00612     Wire.beginTransmission(deviceAddress);
00613     Wire.write(SET_PROPERTY);
00614     Wire.write(0x00);
00615     Wire.write(property.raw.byteHigh); // Send property - High byte - most
significant first
00616     Wire.write(property.raw.byteLow);  // Send property - Low byte - less
significant after
00617     Wire.write(param.raw.byteHigh);    // Send the argments. High Byte -
Most significant first
00618     Wire.write(param.raw.byteLow);     // Send the argments. Low Byte - Less
significant after
00619     Wire.endTransmission();
00620     delayMicroseconds(550);
00621 }
```

References deviceAddress, and waitToSend().

Referenced by digitalOutputFormat(), digitalOutputSampleRate(), setAudioMute(), setAvcAmMaxGain(), setFmBlendMonoThreshold(), setFmBlendMultiPathMonoThreshold(), setFmBlendMultiPathStereoThreshold(), setFmBLendRssiMonoThreshold(), setFmBlendRssiStereoThreshold(), setFmBLendSnrMonoThreshold(), setFmBlendSnrStereoThreshold(), setFmBlendStereoThreshold(), setSeekAmLimits(), setSeekAmSpacing(), setSeekRssiThreshold(), setSeekSrnThreshold(), and setVolume().

void SI4735::sendSSBModeProperty () [protected]

Just send the property SSB_MOD to the device. Internal use (privete method).

Definition at line 2007 of file SI4735.cpp.

```
02008 {
02009     si47x_property property;
02010     property.value = SSB_MODE;
02011     waitToSend();
02012     Wire.beginTransmission(deviceAddress);
02013     Wire.write(SET_PROPERTY);
02014     Wire.write(0x00);                  // Always 0x00
02015     Wire.write(property.raw.byteHigh); // High byte first
02016     Wire.write(property.raw.byteLow);  // Low byte after
02017     Wire.write(currentSSBMode.raw[1]); // SSB MODE params; freq. high byte
first
02018     Wire.write(currentSSBMode.raw[0]); // SSB MODE params; freq. low byte
after
02019
02020     Wire.endTransmission();
02021     delayMicroseconds(550);
02022 }
```

References currentSSBMode, deviceAddress, and waitToSend().

Referenced by setSBBSidebandCutoffFilter(), setSSBAudioBandwidth(), setSSBAutomaticVolumeControl(), setSSBAvcDivider(), setSSBConfig(), setSSBDspAfc(), and setSSBSoftMute().

void SI4735::setAM ()

Sets the radio to AM function. It means: LW MW and SW.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 129.

Definition at line 459 of file SI4735.cpp.

```
00460 {
00461     // If you're already using AM mode, you don't need call powerDown and
radioPowerUp.
00462     // The other properties also should have the same value as the previous
status.
00463     if ( lastMode != AM_CURRENT_MODE ) {
00464         powerDown();
00465         setPowerUp(1, 1, 0, 1, 1, SI473X_ANALOG_AUDIO);
00466         radioPowerUp();
00467         setAvcAmMaxGain(currentAvcAmMaxGain); // Set AM Automatic Volume
Gain to 48
00468         setVolume(volume); // Set to previus configured volume
00469     }
00470     currentSsbStatus = 0;
00471     lastMode = AM_CURRENT_MODE;
00472 }
```

References currentAvcAmMaxGain, lastMode, powerDown(), radioPowerUp(), setAvcAmMaxGain(), setPowerUp(), and setVolume().

Referenced by setAM().

void SI4735::setAM (uint16_t *fromFreq*, uint16_t *toFreq*, uint16_t *initialFreq*, uint16_t *step*)

Sets the radio to AM (LW/MW/SW) function.

See also

setAM()

Parameters

fromFreq	minimum frequency for the band
toFreq	maximum frequency for the band
initialFreq	initial frequency
step	step used to go to the next channel

Definition at line 500 of file SI4735.cpp.

```
00501 {
00502
```

```
00503     currentMinimumFrequency = fromFreq;
00504     currentMaximumFrequency = toFreq;
00505     currentStep = step;
00506
00507     if (initialFreq < fromFreq || initialFreq > toFreq)
00508         initialFreq = fromFreq;
00509
00510     setAM();
00511     currentWorkFrequency = initialFreq;
00512     setFrequency(currentWorkFrequency);
00513 }
```

References currentMaximumFrequency, currentMinimumFrequency, currentStep, currentWorkFrequency, setAM(), and setFrequency().

void SI4735::setAudioMute (bool *off*)

Returns the current volume level.

Sets the audio on or off

See also

See Si47XX PROGRAMMING GUIDE; AN332; pages 62, 123, 171

Parameters

value	if true, mute the audio; if false unmute the audio.

Definition at line 1154 of file SI4735.cpp.

```
01154                                   {
01155     uint16_t value = (off)? 3:0; // 3 means mute; 0 means unmute
01156     sendProperty(RX_HARD_MUTE, value);
01157 }
```

References sendProperty().

void SI4735::setAutomaticGainControl (uint8_t *AGCDIS*, uint8_t *AGCIDX*)

If FM, overrides AGC setting by disabling the AGC and forcing the LNA to have a certain gain that ranges between 0 (minimum attenuation) and 26 (maximum attenuation); If AM/SSB, Overrides the AM AGC setting by disabling the AGC and forcing the gain index that ranges between 0 (minimum attenuation) and 37+ATTN_BACKUP (maximum attenuation);

See also

Si47XX PROGRAMMING GUIDE; AN332; For FM page 81; for AM page 143

Parameters

uint8_t	AGCDIS This param selects whether the AGC is enabled or disabled (0 = AGC enabled; 1 = AGC disabled);
uint8_t	AGCIDX AGC Index (0 = Minimum attenuation (max gain); 1 – 36 = Intermediate attenuation); if >greater than 36 - Maximum attenuation (min gain)).

Definition at line 927 of file SI4735.cpp.

```
00928 {
00929     si47x_agc_overrride agc;
00930
00931     uint8_t cmd;
00932
00933     cmd = (currentTune == FM_TUNE_FREQ) ? FM_AGC_OVERRIDE : AM_AGC_OVERRIDE;
00934
00935     agc.arg.AGCDIS = AGCDIS;
00936     agc.arg.AGCIDX = AGCIDX;
00937
00938     waitToSend();
00939
00940     Wire.beginTransmission(deviceAddress);
00941     Wire.write(cmd);
00942     Wire.write(agc.raw[0]);
```

```
00943     Wire.write(agc.raw[1]);
00944     Wire.endTransmission();
00945
00946     waitToSend();
00947 }
```

References currentTune, deviceAddress, and waitToSend().

void SI4735::setAvcAmMaxGain (uint8_t *gain*)

Sets the maximum gain for automatic volume control. If no parameter is sent, it will be consider 48dB.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 152

Parameters

uint8_t	gain Select a value between 12 and 192. Defaul value 48dB.

Definition at line 957 of file SI4735.cpp.

```
00957                                                {
00958     uint16_t aux;
00959     aux = ( gain > 12 && gain < 193 )? (gain * 340) : (48 * 340);
00960     currentAvcAmMaxGain =  gain;
00961     sendProperty(AM_AUTOMATIC_VOLUME_CONTROL_MAX_GAIN, aux);
00962 }
```

References currentAvcAmMaxGain, sendProperty(), and setAvcAmMaxGain().

Referenced by setAM(), and setAvcAmMaxGain().

void SI4735::setBandwidth (uint8_t *AMCHFLT*, uint8_t *AMPLFLT*)

Selects the bandwidth of the channel filter for AM reception. The choices are 6, 4, 3, 2, 2.5, 1.8, or 1 (kHz). The default bandwidth is 2 kHz. Works only in AM / SSB (LW/MW/SW)

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 125, 151, 277, 181.

Parameters

AMCHFLT	the choices are: 0 = 6 kHz Bandwidth 1 = 4 kHz Bandwidth 2 = 3 kHz Bandwidth 3 = 2 kHz Bandwidth 4 = 1 kHz Bandwidth 5 = 1.8 kHz Bandwidth 6 = 2.5 kHz Bandwidth, gradual roll off 7–15 = Reserved (Do not use).
AMPLFLT	Enables the AM Power Line Noise Rejection Filter.

Definition at line 558 of file SI4735.cpp.

```
00559 {
00560     si47x_bandwidth_config filter;
00561     si47x_property property;
00562
00563     if (currentTune == FM_TUNE_FREQ) // Only for AM/SSB mode
00564         return;
00565
00566     if (AMCHFLT > 6)
00567         return;
00568
00569     property.value = AM_CHANNEL_FILTER;
00570
00571     filter.param.AMCHFLT = AMCHFLT;
00572     filter.param.AMPLFLT = AMPLFLT;
00573
00574     waitToSend();
00575     this->volume = volume;
00576     Wire.beginTransmission(deviceAddress);
00577     Wire.write(SET_PROPERTY);
00578     Wire.write(0x00);                   // Always 0x00
00579     Wire.write(property.raw.byteHigh); // High byte first
00580     Wire.write(property.raw.byteLow);  // Low byte after
00581     Wire.write(filter.raw[1]);          // Raw data for AMCHFLT and
00582     Wire.write(filter.raw[0]);          // AMPLFLT
```

```
00583     Wire.endTransmission();
00584     waitToSend();
00585 }
```

References currentTune, deviceAddress, and waitToSend().

void SI4735::setDeviceI2CAddress (uint8_t *senPin*)

Sets the I2C Bus Address

ATTENTION: The parameter senPin is not the I2C bus address. It is the SEN pin setup of the schematic (eletronic circuit). If it is connected to the ground, call this function with senPin = 0; else senPin = 1. You do not need to use this function if the SEN PIN configured to ground (GND).

The default value is 0x11 (senPin = 0). In this case you have to ground the pin SEN of the SI473X. If you want to change this address, call this function with senPin = 1

Parameters

senPin	0 - when the pin SEN (16 on SSOP version or pin 6 on QFN version) is set to low (GND - 0V) 1 - when the pin SEN (16 on SSOP version or pin 6 on QFN version) is set to high (+3.3V)

Definition at line 109 of file SI4735.cpp.

```
00109                                                 {
00110     deviceAddress = (senPin)? SI473X_ADDR_SEN_HIGH : SI473X_ADDR_SEN_LOW;
00111 };
```

References deviceAddress.

Referenced by getDeviceI2CAddress().

void SI4735::setDeviceOtherI2CAddress (uint8_t *i2cAddr*)

Sets the onther I2C Bus Address (for Si470X) You can set another I2C address different of 0x11 and 0x63

Parameters

uint8_t	i2cAddr (example 0x10)

Definition at line 118 of file SI4735.cpp.

```
00118                                                     {
00119     deviceAddress = i2cAddr;
00120 };
```

References deviceAddress.

void SI4735::setFM ()

Sets the radio to FM function

See also

Si47XX PROGRAMMING GUIDE; AN332; page 64.

Definition at line 479 of file SI4735.cpp.

```
00480 {
00481     powerDown();
00482     setPowerUp(1, 1, 0, 1, 0, SI473X_ANALOG_AUDIO);
00483     radioPowerUp();
00484     setVolume(volume); // Set to previus configured volume
00485     currentSsbStatus = 0;
00486     disableFmDebug();
00487     lastMode = FM_CURRENT_MODE;
00488 }
```

References disableFmDebug(), lastMode, powerDown(), radioPowerUp(), setPowerUp(), and setVolume().

Referenced by setFM().

void SI4735::setFM (uint16_t *fromFreq*, uint16_t *toFreq*, uint16_t *initialFreq*, uint16_t *step*)

Sets the radio to FM function.

See also

setFM()

Parameters

fromFreq	minimum frequency for the band
toFreq	maximum frequency for the band
initialFreq	initial frequency (default frequency)
step	step used to go to the next channel

Definition at line 525 of file SI4735.cpp.

```
00526 {
00527
00528     currentMinimumFrequency = fromFreq;
00529     currentMaximumFrequency = toFreq;
00530     currentStep = step;
00531
00532     if (initialFreq < fromFreq || initialFreq > toFreq)
00533         initialFreq = fromFreq;
00534
00535     setFM();
00536
00537     currentWorkFrequency = initialFreq;
00538     setFrequency(currentWorkFrequency);
00539 }
```

References currentMaximumFrequency, currentMinimumFrequency, currentStep, currentWorkFrequency, setFM(), and setFrequency().

void SI4735::setFmBlendMonoThreshold (uint8_t *parameter*)

Sets RSSI threshold for mono blend (Full mono below threshold, blend above threshold). To force stereo set this to 0. To force mono set this to 127. Default value is 30 dBÎ¼V.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 56.

Parameters

parameter	valid values: 0 to 127

Definition at line 645 of file SI4735.cpp.

```
00646 {
00647     sendProperty(FM_BLEND_MONO_THRESHOLD, parameter);
00648 }
```

References sendProperty().

void SI4735::setFmBlendMultiPathMonoThreshold (uint8_t *parameter*)

Sets Multipath threshold for mono blend (Full mono above threshold, blend below threshold). To force stereo, set to 100. To force mono, set to 0. The default is 60.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 60.

Parameters

parameter	valid values: 0 to 100

Definition at line 722 of file SI4735.cpp.

```
00723 {
00724     sendProperty(FM_BLEND_MULTIPATH_MONO_THRESHOLD, parameter);
00725 }
```

References sendProperty().

void SI4735::setFmBlendMultiPathStereoThreshold (uint8_t *parameter*)

Sets multipath threshold for stereo blend (Full stereo below threshold, blend above threshold). To force stereo, set this to 100. To force mono, set this to 0. Default value is 20.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 60.

Parameters

parameter	valid values: 0 to 100

Definition at line 709 of file SI4735.cpp.

```
00710 {
00711     sendProperty(FM_BLEND_MULTIPATH_STEREO_THRESHOLD, parameter);
00712 }
```

References sendProperty().

void SI4735::setFmBLendRssiMonoThreshold (uint8_t *parameter*)

Sets RSSI threshold for mono blend (Full mono below threshold, blend above threshold). To force stereo, set this to 0. To force mono, set this to 127. Default value is 30 dBÎ¼V.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 59.

Parameters

parameter	valid values: 0 to 127

Definition at line 670 of file SI4735.cpp.

```
00671 {
00672     sendProperty(FM_BLEND_RSSI_MONO_THRESHOLD, parameter);
00673 }
```

References sendProperty().

void SI4735::setFmBlendRssiStereoThreshold (uint8_t *parameter*)

Sets RSSI threshold for stereo blend. (Full stereo above threshold, blend below threshold.) To force stereo, set this to 0. To force mono, set this to 127. Default value is 49 dBÎ¼V.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 59.

Parameters

parameter	valid values: 0 to 127

Definition at line 657 of file SI4735.cpp.

```
00658 {
00659     sendProperty(FM_BLEND_RSSI_STEREO_THRESHOLD, parameter);
00660 }
```

References sendProperty().

void SI4735::setFmBLendSnrMonoThreshold (uint8_t *parameter*)

Sets SNR threshold for mono blend (Full mono below threshold, blend above threshold). To force stereo, set this to 0. To force mono, set this to 127. Default value is 14 dB.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 59.

Parameters

parameter	valid values: 0 to 127

Definition at line 696 of file SI4735.cpp.

```
00697 {
```

```
00698     sendProperty(FM_BLEND_SNR_MONO_THRESHOLD, parameter);
00699 }
```

References sendProperty().

void SI4735::setFmBlendSnrStereoThreshold (uint8_t *parameter*)

Sets SNR threshold for stereo blend (Full stereo above threshold, blend below threshold). To force stereo, set this to 0. To force mono, set this to 127. Default value is 27 dB.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 59.

Parameters

parameter	valid values: 0 to 127

Definition at line 683 of file SI4735.cpp.

```
00684 {
00685     sendProperty(FM_BLEND_SNR_STEREO_THRESHOLD, parameter);
00686 }
```

References sendProperty().

void SI4735::setFmBlendStereoThreshold (uint8_t *parameter*)

Sets RSSI threshold for stereo blend (Full stereo above threshold, blend below threshold). To force stereo, set this to 0. To force mono, set this to 127.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 90.

Parameters

parameter	valid values: 0 to 127

Definition at line 632 of file SI4735.cpp.

```
00633 {
00634     sendProperty(FM_BLEND_STEREO_THRESHOLD, parameter);
00635 }
```

References sendProperty().

void SI4735::setFmStereoOff ()

Turn Off Stereo operation.

Definition at line 730 of file SI4735.cpp.

```
00731 {
00732     // TO DO
00733 }
```

void SI4735::setFmStereoOn ()

Turn Off Stereo operation.

Definition at line 738 of file SI4735.cpp.

```
00739 {
00740     // TO DO
00741 }
```

void SI4735::setFrequency (uint16_t *freq*)

Set the frequency to the corrent function of the Si4735 (FM, AM or SSB) You have to call setup or setPowerUp before call setFrequency.

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 70, 135

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; page 13

Parameters

uint16_t	freq Is the frequency to change. For example, FM => 10390 = 103.9 MHz; AM => 810 = 810 KHz.

Definition at line 377 of file SI4735.cpp.

```
00378 {
00379     waitToSend(); // Wait for the si473x is ready.
00380     currentFrequency.value = freq;
00381     currentFrequencyParams.arg.FREQH = currentFrequency.raw.FREQH;
00382     currentFrequencyParams.arg.FREQL = currentFrequency.raw.FREQL;
00383
00384     if (currentSsbStatus != 0)
00385     {
00386         currentFrequencyParams.arg.DUMMY1 = 0;
00387         currentFrequencyParams.arg.USBLSB = currentSsbStatus; // Set to LSB
or USB
00388         currentFrequencyParams.arg.FAST = 1;                  // Used just
on AM and FM
00389         currentFrequencyParams.arg.FREEZE = 0;                // Used just
on FM
00390     }
00391
00392     Wire.beginTransmission(deviceAddress);
00393     Wire.write(currentTune);
00394     Wire.write(currentFrequencyParams.raw[0]); // Send a byte with FAST and
FREEZE information; if not FM must be 0;
00395     Wire.write(currentFrequencyParams.arg.FREQH);
00396     Wire.write(currentFrequencyParams.arg.FREQL);
00397     Wire.write(currentFrequencyParams.arg.ANTCAPH);
00398     // If current tune is not FM sent one more byte
00399     if (currentTune != FM_TUNE_FREQ)
00400         Wire.write(currentFrequencyParams.arg.ANTCAPL);
00401
00402     Wire.endTransmission();
00403     waitToSend();                // Wait for the si473x is ready.
00404     currentWorkFrequency = freq; // check it
00405     delay(MAX_DELAY_AFTER_SET_FREQUENCY); // For some reason I need to delay
here.
00406 }
```

References si47x_set_frequency::ANTCAPH, si47x_set_frequency::ANTCAPL, currentFrequency, currentFrequencyParams, currentTune, currentWorkFrequency, deviceAddress, si47x_set_frequency::DUMMY1, si47x_set_frequency::FREEZE, si47x_frequency::FREQH, si47x_set_frequency::FREQH, si47x_set_frequency::FREQL, si47x_set_frequency::USBLSB, and waitToSend().

Referenced by frequencyDown(), frequencyUp(), setAM(), and setFM().

void SI4735::setFrequencyStep (uint16_t *step*)

Sets the current step value.

ATTENTION: This function does not check the limits of the current band. Please, don't take a step bigger than your legs.

Parameters

step	if you are using FM, 10 means 100KHz. If you are using AM 10 means 10KHz For AM, 1 (1KHz) to 1000 (1MHz) are valid values. For FM 5 (50KHz) and 10 (100KHz) are valid values.

Definition at line 418 of file SI4735.cpp.

```
00419 {
00420     currentStep = step;
00421 }
```

References currentStep.

void SI4735::setI2CFastModeCustom (long *value* = 500000) [inline]

Sets I2C buss to 400KHz.

Sets the I2C bus to a given value.

ATTENTION: use this function with cation

Parameters

value	in Hz. For example: The values 500000 sets the bus to 500KHz.

Definition at line 1150 of file SI4735.h.

```
01150 { Wire.setClock(value); };
```

void SI4735::setPowerUp (uint8_t *CTSIEN*, uint8_t *GPO2OEN*, uint8_t *PATCH*, uint8_t *XOSCEN*, uint8_t *FUNC*, uint8_t *OPMODE*)

Set the Power Up parameters for si473X. Use this method to chenge the defaul behavior of the Si473X. Use it before PowerUp()

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 65 and 129

Parameters

uint8_t	CTSIEN sets Interrupt anabled or disabled (1 = anabled and 0 = disabled)
uint8_t	GPO2OEN sets GP02 Si473X pin enabled (1 = anabled and 0 = disabled)
uint8_t	PATCH Used for firmware patch updates. Use it always 0 here.
uint8_t	XOSCEN sets external Crystal enabled or disabled
uint8_t	FUNC sets the receiver function have to be used [0 = FM Receive; 1 = AM (LW/MW/SW) and SSB (if SSB patch apllied)]
uint8_t	OPMODE set the kind of audio mode you want to use.

Definition at line 164 of file SI4735.cpp.

```
00165 {
00166     powerUp.arg.CTSIEN = CTSIEN;    // 1 -> Interrupt anabled;
00167     powerUp.arg.GPO2OEN = GPO2OEN; // 1 -> GPO2 Output Enable;
00168     powerUp.arg.PATCH = PATCH;      // 0 -> Boot normally;
00169     powerUp.arg.XOSCEN = XOSCEN;    // 1 -> Use external crystal oscillator;
00170     powerUp.arg.FUNC = FUNC;        // 0 = FM Receive; 1 = AM/SSB (LW/MW/SW)
Receiver.
00171     powerUp.arg.OPMODE = OPMODE;    // 0x5 = 00000101 = Analog audio outputs
(LOUT/ROUT).
00172
00173     // Set the current tuning frequancy mode 0X20 = FM and 0x40 = AM (LW/MW/
SW)
00174     // See See Si47XX PROGRAMMING GUIDE; AN332; pages 55 and 124
00175
00176     if (FUNC == 0)
00177     {
00178         currentTune = FM_TUNE_FREQ;
00179         currentFrequencyParams.arg.FREEZE = 1;
00180     }
00181     else
00182     {
00183         currentTune = AM_TUNE_FREQ;
00184         currentFrequencyParams.arg.FREEZE = 0;
00185     }
00186     currentFrequencyParams.arg.FAST = 1;
00187     currentFrequencyParams.arg.DUMMY1 = 0;
00188     currentFrequencyParams.arg.ANTCAPH = 0;
00189     currentFrequencyParams.arg.ANTCAPL = 1;
00190 }
```

References si47x_set_frequency::ANTCAPH, si47x_set_frequency::ANTCAPL, si473x_powerup::CTSIEN, currentFrequencyParams, currentTune, si47x_set_frequency::DUMMY1, si47x_set_frequency::FREEZE, si473x_powerup::GPO2OEN, si473x_powerup::OPMODE, si473x_powerup::PATCH, powerUp, and si473x_powerup::XOSCEN.

Referenced by setAM(), setFM(), and setSSB().

void SI4735::setRdsConfig (uint8_t *RDSEN*, uint8_t *BLETHA*, uint8_t *BLETHB*, uint8_t *BLETHC*, uint8_t *BLETHD*)

RESP3 - RDS FIFO Used; Number of groups remaining in the RDS FIFO (0 if empty).

Sets RDS property (FM_RDS_CONFIG) Configures RDS settings to enable RDS processing (RDSEN) and set RDS block error thresholds. When a RDS Group is received, all block errors must be less than or equal the associated block error threshold for the group to be stored in the RDS FIFO.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 104

MPORTANT: All block errors must be less than or equal the associated block error threshold for the group to be stored in the RDS FIFO. 0 = No errors. 1 = 1–2 bit errors detected and corrected. 2 = 3–5 bit errors detected and corrected. 3 = Uncorrectable. Recommended Block Error Threshold options: 2,2,2,2 = No group stored if any errors are uncorrected. 3,3,3,3 = Group stored regardless of errors. 0,0,0,0 = No group stored containing corrected or uncorrected errors. 3,2,3,3 = Group stored with corrected errors on B, regardless of errors on A, C, or D.

Parameters

uint8_t	RDSEN RDS Processing Enable; 1 = RDS processing enabled.
uint8_t	BLETHA Block Error Threshold BLOCKA.
uint8_t	BLETHB Block Error Threshold BLOCKB.
uint8_t	BLETHC Block Error Threshold BLOCKC.
uint8_t	BLETHD Block Error Threshold BLOCKD.

Definition at line 1266 of file SI4735.cpp.

```
01267 {
01268     si47x_property property;
01269     si47x_rds_config config;
01270
01271     waitToSend();
01272
01273     // Set property value
01274     property.value = FM_RDS_CONFIG;
01275
01276     // Arguments
01277     config.arg.RDSEN = RDSEN;
01278     config.arg.BLETHA = BLETHA;
01279     config.arg.BLETHB = BLETHB;
01280     config.arg.BLETHC = BLETHC;
01281     config.arg.BLETHD = BLETHD;
01282     config.arg.DUMMY1 = 0;
01283
01284     Wire.beginTransmission(deviceAddress);
01285     Wire.write(SET_PROPERTY);
01286     Wire.write(0x00);                  // Always 0x00 (I need to check it)
01287     Wire.write(property.raw.byteHigh); // Send property - High byte - most
significant first
01288     Wire.write(property.raw.byteLow);  // Low byte
01289     Wire.write(config.raw[1]);         // Send the argments. Most
significant first
01290     Wire.write(config.raw[0]);
01291     Wire.endTransmission();
01292     delayMicroseconds(550);
01293
01294     RdsInit();
01295 }
```

References si47x_rds_config::BLETHA, si47x_rds_config::BLETHB, si47x_rds_config::BLETHC, deviceAddress, si47x_rds_config::DUMMY1, RdsInit(), and waitToSend().

void SI4735::setRdsIntSource (uint8_t *RDSNEWBLOCKB*, uint8_t *RDSNEWBLOCKA*, uint8_t *RDSSYNCFOUND*, uint8_t *RDSSYNCLOST*, uint8_t *RDSRECV*)

Configures interrupt related to RDS

Use this method if want to use interrupt

See also

Si47XX PROGRAMMING GUIDE; AN332; page 103

Parameters

RDSRECV	If set, generate RDSINT when RDS FIFO has at least FM_RDS_INT_FIFO_COUNT entries.
RDSSYNCLOST	If set, generate RDSINT when RDS loses synchronization.
RDSSYNCFOUND	set, generate RDSINT when RDS gains synchronization.
RDSNEWBLOCKA	If set, generate an interrupt when Block A data is found or subsequently changed
RDSNEWBLOCKB	If set, generate an interrupt when Block B data is found or subsequently changed

Definition at line 1310 of file SI4735.cpp.

```
01311 {
01312     si47x_property property;
01313     si47x_rds_int_source rds_int_source;
01314
01315     if (currentTune != FM_TUNE_FREQ)
01316         return;
01317
01318     rds_int_source.refined.RDSNEWBLOCKB = RDSNEWBLOCKB;
01319     rds_int_source.refined.RDSNEWBLOCKA = RDSNEWBLOCKA;
01320     rds_int_source.refined.RDSSYNCFOUND = RDSSYNCFOUND;
01321     rds_int_source.refined.RDSSYNCLOST = RDSSYNCLOST;
01322     rds_int_source.refined.RDSRECV = RDSRECV;
01323     rds_int_source.refined.DUMMY1 = 0;
01324     rds_int_source.refined.DUMMY2 = 0;
01325
01326     property.value = FM_RDS_INT_SOURCE;
01327
01328     waitToSend();
01329
01330     Wire.beginTransmission(deviceAddress);
01331     Wire.write(SET_PROPERTY);
01332     Wire.write(0x00);                   // Always 0x00 (I need to check it)
01333     Wire.write(property.raw.byteHigh); // Send property - High byte - most
significant first
01334     Wire.write(property.raw.byteLow);  // Low byte
01335     Wire.write(rds_int_source.raw[1]); // Send the argments. Most
significant first
01336     Wire.write(rds_int_source.raw[0]);
01337     Wire.endTransmission();
01338     waitToSend();
01339 }
```

References currentTune, deviceAddress, si47x_rds_int_source::DUMMY1, si47x_rds_int_source::DUMMY2, si47x_rds_int_source::RDSNEWBLOCKA, si47x_rds_int_source::RDSNEWBLOCKB, si47x_rds_int_source::RDSSYNCFOUND, si47x_rds_int_source::RDSSYNCLOST, and waitToSend().

void SI4735::setSBBSidebandCutoffFilter (uint8_t *SBCUTFLT*)

Sets SBB Sideband Cutoff Filter for band pass and low pass filters: 0 = Band pass filter to cutoff both the unwanted side band and high frequency components > 2.0 kHz of the wanted side band. (default) 1 = Low pass filter to cutoff the unwanted side band. Other values = not allowed.

See also

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; page 24

Parameters

SBCUTFLT	0 or 1; see above

Definition at line 1915 of file SI4735.cpp.

```
01916 {
01917     currentSSBMode.param.SBCUTFLT = SBCUTFLT;
01918     sendSSBModeProperty();
01919 }
```

References currentSSBMode, si47x_ssb_mode::SBCUTFLT, and sendSSBModeProperty().

void SI4735::setSeekAmLimits (uint16_t *bottom*, uint16_t *top*)

Sets the bottom frequency and top frequency of the AM band for seek. Default is 520 to 1710.

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 127, 161, and 162

Parameters

uint16_t	bottom - the bottom of the AM band for seek
uint16_t	top - the top of the AM band for seek

Definition at line 1094 of file SI4735.cpp.

```
01095 {
01096     sendProperty(AM_SEEK_BAND_BOTTOM, bottom);
01097     sendProperty(AM_SEEK_BAND_TOP, top);
01098 }
```

References sendProperty().

void SI4735::setSeekAmSpacing (uint16_t *spacing*)

Selects frequency spacingfor AM seek. Default is 10 kHz spacing.

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 163, 229 and 283

Parameters

uint16_t	spacing - step in KHz

Definition at line 1107 of file SI4735.cpp.

```
01108 {
01109     sendProperty(AM_SEEK_FREQ_SPACING, spacing);
01110 }
```

References sendProperty().

void SI4735::setSeekRssiThreshold (uint16_t *value*)

Sets the RSSI threshold for a valid AM Seek/Tune. If the value is zero then RSSI threshold is not considered when doing a seek. Default value is 25 dBÎ¼V.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 127

Definition at line 1129 of file SI4735.cpp.

```
01130 {
01131     sendProperty(AM_SEEK_RSSI_THRESHOLD, value);
01132 }
```

References sendProperty().

void SI4735::setSeekSrnThreshold (uint16_t *value*)

Sets the SNR threshold for a valid AM Seek/Tune. If the value is zero then SNR threshold is not considered when doing a seek. Default value is 5 dB.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 127

Definition at line 1118 of file SI4735.cpp.

```
01119 {
01120     sendProperty(AM_SEEK_SNR_THRESHOLD, value);
01121 }
```

References sendProperty().

void SI4735::setSSB (uint8_t *usblsb*)

Set the radio to AM function. It means: LW MW and SW.

See also

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; pages 13 and 14

setAM()

void SI4735::setFrequency(uint16_t freq)

Parameters

usblsb	upper or lower side band; 1 = LSB; 2 = USB

Definition at line 1961 of file SI4735.cpp.

```
01962 {
01963     // Is it needed to load patch when switch to SSB?
01964     // powerDown();
01965     // It starts with the same AM parameters.
01966     setPowerUp(1, 1, 0, 1, 1, SI473X_ANALOG_AUDIO);
01967     radioPowerUp();
01968     // ssbPowerUp(); // Not used for regular operation
01969     setVolume(volume); // Set to previus configured volume
01970     currentSsbStatus = usblsb;
01971     lastMode = SSB_CURRENT_MODE;
01972 }
```

References lastMode, radioPowerUp(), setPowerUp(), and setVolume().

void SI4735::setSSBAudioBandwidth (uint8_t *AUDIOBW*)

SSB Audio Bandwidth for SSB mode

0 = 1.2 kHz low-pass filter* . (default) 1 = 2.2 kHz low-pass filter* . 2 = 3.0 kHz low-pass filter. 3 = 4.0 kHz low-pass filter. 4 = 500 Hz band-pass filter for receiving CW signal, i.e. [250 Hz, 750 Hz] with center frequency at 500 Hz when USB is selected or [-250 Hz, -750 1Hz] with center frequency at -500Hz when LSB is selected* . 5 = 1 kHz band-pass filter for receiving CW signal, i.e. [500 Hz, 1500 Hz] with center frequency at 1 kHz when USB is selected or [-500 Hz, -1500 1 Hz] with center frequency at -1kHz when LSB is selected* . Other values = reserved. Note: If audio bandwidth selected is about 2 kHz or below, it is recommended to set SBCUTFLT[3:0] to 0 to enable the band pass filter for better high- cut performance on the wanted side band. Otherwise, set it to 1.

See also

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; page 24

Parameters

AUDIOBW	the valid values are 0, 1, 2, 3, 4 or 5; see description above

Definition at line 1944 of file SI4735.cpp.

```
01945 {
01946     // Sets the audio filter property parameter
01947     currentSSBMode.param.AUDIOBW = AUDIOBW;
01948     sendSSBModeProperty();
01949 }
```

References currentSSBMode, and sendSSBModeProperty().

void SI4735::setSSBAutomaticVolumeControl (uint8_t *AVCEN*)

Sets SSB Automatic Volume Control (AVC) for SSB mode

See also

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; page 24

Parameters

AVCEN	0 = Disable AVC; 1 = Enable AVC (default).

Definition at line 1886 of file SI4735.cpp.

```
01887 {
01888     currentSSBMode.param.AVCEN = AVCEN;
01889     sendSSBModeProperty();
01890 }
```

References si47x_ssb_mode::AVCEN, currentSSBMode, and sendSSBModeProperty().

void SI4735::setSSBAvcDivider (uint8_t *AVC_DIVIDER*)

Sets AVC Divider

See also

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; page 24

Parameters

AVC_DIVIDER	SSB mode, set divider = 0; SYNC mode, set divider = 3; Other values = not allowed.

Definition at line 1899 of file SI4735.cpp.

```
01900 {
01901     currentSSBMode.param.AVC_DIVIDER = AVC_DIVIDER;
01902     sendSSBModeProperty();
01903 }
```

References si47x_ssb_mode::AVC_DIVIDER, currentSSBMode, and sendSSBModeProperty().

void SI4735::setSSBBfo (int *offset*)

Single Side Band (SSB) implementation

This implementation was tested only on Si4735-D60 device.

SSB modulation is a refinement of amplitude modulation that one of the side band and the carrier are suppressed.

See also

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; pages 3 and 5

First of all, it is important to say that the SSB patch content is not part of this library. The paches used here were made available by Mr. Vadim Afonkin on his Dropbox repository. It is important to note that the author of this library does not encourage anyone to use the SSB patches content for commercial purposes. In other words, this library only supports SSB patches, the patches themselves are not part of this library.

What does SSB patch means? In this context, a patch is a piece of software used to change the behavior of the SI4735 device. There is little information available about patching the SI4735.

The following information is the understanding of the author of this project and it is not necessarily correct.

A patch is executed internally (run by internal MCU) of the device. Usually, patches are used to fixes bugs or add improvements and new features of the firmware installed in the internal ROM of the device. Patches to the SI4735 are distributed in binary form and have to be transferred to the internal RAM of the device by the host MCU (in this case Arduino boards). Since the RAM is volatile memory, the patch stored into the device gets lost when you turn off the system. Consequently, the content of the patch has to be transferred again to the device each time after turn on the system or reset the device.

I would like to thank Mr Vadim Afonkin for making available the SSBRX patches for SI4735-D60 on his Dropbox repository. On this repository you have two files, amrx_6_0_1_ssbrx_patch_full_0x9D29.csg and amrx_6_0_1_ssbrx_patch_init_0xA902.csg. It is important to know that the patch content of the original files is constant hexadecimal representation used by the language C/C++. Actally, the original files are in ASCII format (not in binary format). If you are not using C/C++ or if you want to load the files directly to the SI4735, you must convert the values to numeric value of the hexadecimal constants. For example: 0x15 = 21 (00010101); 0x16 = 22 (00010110); 0x01 = 1 (00000001); 0xFF = 255 (11111111);

ATTENTION: The author of this project does not guarantee that procedures shown here will work in your development environment. Given this, it is at your own risk to continue with the procedures suggested here. This library works with the I²C communication protocol and it is designed to apply a SSB extension PATCH to CI SI4735-D60. Once again, the author disclaims any liability for any damage this procedure may cause to your SI4735 or other devices that you are using. Sets the SSB Beat Frequency Offset (BFO).

See also

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; pages 5 and 23

Parameters

offset	16-bit signed value (unit in Hz). The valid range is -16383 to +16383 Hz.

Definition at line 1791 of file SI4735.cpp.

```
01792 {
01793
01794     si47x_property property;
01795     si47x_frequency bfo_offset;
01796
01797     if (currentTune == FM_TUNE_FREQ) // Only for AM/SSB mode
01798         return;
01799
01800     waitToSend();
01801
01802     property.value = SSB_BFO;
01803     bfo_offset.value = offset;
01804
01805     Wire.beginTransmission(deviceAddress);
01806     Wire.write(SET_PROPERTY);
01807     Wire.write(0x00);                   // Always 0x00
01808     Wire.write(property.raw.byteHigh); // High byte first
01809     Wire.write(property.raw.byteLow);  // Low byte after
01810     Wire.write(bfo_offset.raw.FREQH);  // Offset freq. high byte first
01811     Wire.write(bfo_offset.raw.FREQL);  // Offset freq. low byte first
01812
01813     Wire.endTransmission();
01814     delayMicroseconds(550);
01815 }
```

References currentTune, deviceAddress, si47x_frequency::FREQH, and waitToSend().

void SI4735::setSSBConfig (uint8_t *AUDIOBW*, uint8_t *SBCUTFLT*, uint8_t *AVC_DIVIDER*, uint8_t *AVCEN*, uint8_t *SMUTESEL*, uint8_t *DSP_AFCDIS*)

Set the SSB receiver mode details: 1) Enable or disable AFC track to carrier function for receiving normal AM signals; 2) Set the audio bandwidth; 3) Set the side band cutoff filter; 4) Set soft-mute based on RSSI or SNR; 5) Enable or disbable automatic volume control (AVC) function.

See also

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; page 24

Parameters

AUDIOBW	SSB Audio bandwidth; 0 = 1.2KHz (default); 1=2.2KHz; 2=3KHz; 3=4KHz; 4=500Hz; 5=1KHz.
SBCUTFLT	SSB side band cutoff filter for band passand low pass filter if 0, the band pass

	filter to cutoff both the unwanted side band and high frequency component > 2KHz of the wanted side band (default).
AVC_DIVIDER	set 0 for SSB mode; set 3 for SYNC mode.
AVCEN	SSB Automatic Volume Control (AVC) enable; 0=disable; 1=enable (default).
SMUTESEL	SSB Soft-mute Based on RSSI or SNR.
DSP_AFCDIS	DSP AFC Disable or enable; 0=SYNC MODE, AFC enable; 1=SSB MODE, AFC disable.

Definition at line 1836 of file SI4735.cpp.

```
01837 {
01838     if (currentTune == FM_TUNE_FREQ) // Only AM/SSB mode
01839         return;
01840
01841     currentSSBMode.param.AUDIOBW = AUDIOBW;
01842     currentSSBMode.param.SBCUTFLT = SBCUTFLT;
01843     currentSSBMode.param.AVC_DIVIDER = AVC_DIVIDER;
01844     currentSSBMode.param.AVCEN = AVCEN;
01845     currentSSBMode.param.SMUTESEL = SMUTESEL;
01846     currentSSBMode.param.DUMMY1 = 0;
01847     currentSSBMode.param.DSP_AFCDIS = DSP_AFCDIS;
01848
01849     sendSSBModeProperty();
01850 }
```

References si47x_ssb_mode::AVC_DIVIDER, si47x_ssb_mode::AVCEN, currentSSBMode, currentTune, si47x_ssb_mode::DSP_AFCDIS, si47x_ssb_mode::DUMMY1, si47x_ssb_mode::SBCUTFLT, sendSSBModeProperty(), and si47x_ssb_mode::SMUTESEL.

void SI4735::setSSBDspAfc (uint8_t *DSP_AFCDIS*)

Sets DSP AFC disable or enable

See also

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; page 24

Parameters

DSP_AFCDIS	0 = SYNC mode, AFC enable; 1 = SSB mode, AFC disable

Definition at line 1859 of file SI4735.cpp.

```
01860 {
01861     currentSSBMode.param.DSP_AFCDIS = DSP_AFCDIS;
01862     sendSSBModeProperty();
01863 }
```

References currentSSBMode, si47x_ssb_mode::DSP_AFCDIS, and sendSSBModeProperty().

void SI4735::setSSBSoftMute (uint8_t *SMUTESEL*)

Sets SSB Soft-mute Based on RSSI or SNR Selection:

See also

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; page 24

Parameters

SMUTESEL	0 = Soft-mute based on RSSI (default); 1 = Soft-mute based on SNR.

Definition at line 1873 of file SI4735.cpp.

```
01874 {
01875     currentSSBMode.param.SMUTESEL = SMUTESEL;
01876     sendSSBModeProperty();
01877 }
```

References currentSSBMode, sendSSBModeProperty(), and si47x_ssb_mode::SMUTESEL.

void SI4735::setTuneFrequencyAntennaCapacitor (uint16_t *capacitor*)

Only FM. Freeze Metrics During Alternate Frequency Jump.

Selects the tuning capacitor value.

For FM, Antenna Tuning Capacitor is valid only when using TXO/LPI pin as the antenna input.

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 71 and 136

Parameters

capacitor	If zero, the tuning capacitor value is selected automatically. If the value is set to anything other than 0: AM - the tuning capacitance is manually set as 95 fF x ANTCAP + 7 pF. ANTCAP manual range is 1–6143; FM - the valid range is 0 to 191. According to Silicon Labs, automatic capacitor tuning is recommended (value 0).

Definition at line 344 of file SI4735.cpp.

```
00345 {
00346     si47x_antenna_capacitor cap;
00347
00348     cap.value = capacitor;
00349
00350     currentFrequencyParams.arg.DUMMY1 = 0;
00351
00352     if (currentTune == FM_TUNE_FREQ)
00353     {
00354         // For FM, the capacitor value has just one byte
00355         currentFrequencyParams.arg.ANTCAPH = (capacitor <= 191) ?
cap.raw.ANTCAPL : 0;
00356     }
00357     else
00358     {
00359         if (capacitor <= 6143)
00360         {
00361             currentFrequencyParams.arg.FREEZE = 0; // This parameter is not
used for AM
00362             currentFrequencyParams.arg.ANTCAPH = cap.raw.ANTCAPH;
00363             currentFrequencyParams.arg.ANTCAPL = cap.raw.ANTCAPL;
00364         }
00365     }
00366 }
```

References si47x_antenna_capacitor::ANTCAPH, si47x_set_frequency::ANTCAPH, si47x_set_frequency::ANTCAPL, currentFrequencyParams, currentTune, si47x_set_frequency::DUMMY1, and si47x_set_frequency::FREEZE.

void SI4735::setup (uint8_t *resetPin*, int *interruptPin*, uint8_t *defaultFunction*, uint8_t *audioMode* = `SI473X_ANALOG_AUDIO`)

Starts the Si473X device.

If the audio mode parameter is not entered, analog mode will be considered.

Parameters

uint8_t	resetPin Digital Arduino Pin used to RESET command
uint8_t	interruptPin interrupt Arduino Pin (see your Arduino pinout). If less than 0, iterrupt disabled
uint8_t	defaultFunction
uint8_t	audioMode default SI473X_ANALOG_AUDIO (Analog Audio). Use SI473X_ANALOG_AUDIO or SI473X_DIGITAL_AUDIO

Definition at line 280 of file SI4735.cpp.

```
00281 {
00282     uint8_t interruptEnable = 0;
00283     Wire.begin();
00284
00285     this->resetPin = resetPin;
00286     this->interruptPin = interruptPin;
00287
00288     // Arduino interrupt setup (you have to know which Arduino Pins can deal
with interrupt).
00289     if (interruptPin >= 0)
```

```
00290     {
00291         pinMode(interruptPin, INPUT);
00292         attachInterrupt(digitalPinToInterrupt(interruptPin),
interrupt_hundler, RISING);
00293         interruptEnable = 1;
00294     }
00295
00296     pinMode(resetPin, OUTPUT);
00297     digitalWrite(resetPin, HIGH);
00298
00299     data_from_si4735 = false;
00300
00301     // Set the initial SI473X behavior
00302     // CTSIEN   1 -> Interrupt anabled or disable;
00303     // GPO2OEN  1 -> GPO2 Output Enable;
00304     // PATCH    0 -> Boot normally;
00305     // XOSCEN   1 -> Use external crystal oscillator;
00306     // FUNC     defaultFunction = 0 = FM Receive; 1 = AM (LW/MW/SW)
Receiver.
00307     // OPMODE   SI473X_ANALOG_AUDIO or SI473X_DIGITAL_AUDIO.
00308     setPowerUp(interruptEnable, 0, 0, 1, defaultFunction, audioMode);
00309
00310     reset();
00311     radioPowerUp();
00312     setVolume(30); // Default volume level.
00313     getFirmware();
00314 }
```

References interruptPin.

void SI4735::setup (uint8_t *resetPin*, uint8_t *defaultFunction*)

Starts the Si473X device.

Use this setup if you are not using interrupt resource

Parameters

uint8_t	resetPin Digital Arduino Pin used to RESET command
uint8_t	defaultFunction

Definition at line 323 of file SI4735.cpp.

```
00324 {
00325     setup(resetPin, -1, defaultFunction);
00326     delay(250);
00327 }
```

void SI4735::setVolume (uint8_t *volume*)

RESP8 - Returns the Chip Revision (ASCII).

Sets volume level (0 to 63)

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 62, 123, 170, 173 and 204

Parameters

uint8_t	volume (domain: 0 - 63)

Definition at line 1141 of file SI4735.cpp.

```
01142 {
01143     sendProperty(RX_VOLUME, volume);
01144     this->volume = volume;
01145 }
```

References sendProperty().

Referenced by setAM(), setFM(), setSSB(), volumeDown(), and volumeUp().

void SI4735::ssbPowerUp ()

This function can be useful for debug and teste.

Definition at line 2117 of file SI4735.cpp.

```
02118 {
02119     waitToSend();
02120     Wire.beginTransmission(deviceAddress);
02121     Wire.write(POWER_UP);
02122     Wire.write(0b00010001); // Set to AM/SSB, disable interrupt; disable
GPO2OEN; boot normaly; enable External Crystal Oscillator  .
02123     Wire.write(0b00000101); // Set to Analog Line Input.
02124     Wire.endTransmission();
02125     delayMicroseconds(2500);
02126
02127     powerUp.arg.CTSIEN = 0;          // 1 -> Interrupt anabled;
02128     powerUp.arg.GPO2OEN = 0;         // 1 -> GPO2 Output Enable;
02129     powerUp.arg.PATCH = 0;           // 0 -> Boot normally;
02130     powerUp.arg.XOSCEN = 1;          // 1 -> Use external crystal
oscillator;
02131     powerUp.arg.FUNC = 1;            // 0 = FM Receive; 1 = AM/SSB
(LW/MW/SW) Receiver.
02132     powerUp.arg.OPMODE = 0b00000101; // 0x5 = 00000101 = Analog audio
outputs (LOUT/ROUT).
02133 }
```

References si473x_powerup::CTSIEN, deviceAddress, si473x_powerup::GPO2OEN, si473x_powerup::OPMODE, si473x_powerup::PATCH, powerUp, waitToSend(), and si473x_powerup::XOSCEN.

void SI4735::ssbSetup ()

Starts the Si473X device on SSB (same AM Mode). Same SI4735::setup optimized to improve loading patch performance

Definition at line 2106 of file SI4735.cpp.

```
02107 {
02108     // setPowerUp(powerUp.arg.CTSIEN, 0, 0, 1, 1, SI473X_ANALOG_AUDIO);
02109     reset();
02110     // radioPowerUp();
02111 }
```

References reset().

void SI4735::volumeDown ()

Set sound volume level Down

See also

setVolume()

Definition at line 1188 of file SI4735.cpp.

```
01189 {
01190     if (volume > 0)
01191         volume--;
01192     setVolume(volume);
01193 }
```

References setVolume().

void SI4735::volumeUp ()

Set sound volume level Up

See also

setVolume()

Definition at line 1176 of file SI4735.cpp.

```
01177 {
01178     if (volume < 63)
01179         volume++;
01180     setVolume(volume);
01181 }
```

References setVolume().

void SI4735::waitInterrupr (void) [protected]

If you setup interrupt, this function will be called whenever the Si4735 changes.

Definition at line 46 of file SI4735.cpp.

```
00047 {
00048     while (!data_from_si4735)
00049         ;
00050 }
```

void SI4735::waitToSend (void)

Wait for the si473x is ready (Clear to Send (CTS) status bit have to be 1).

This function should be used before sending any command to a SI47XX device.

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 63, 128

Definition at line 142 of file SI4735.cpp.

```
00143 {
00144     do
00145     {
00146         delayMicroseconds(MIN_DELAY_WAIT_SEND_LOOP); // Need check the
minimum value.
00147         Wire.requestFrom(deviceAddress, 1);
00148     } while (!(Wire.read() & B10000000));
00149 }
```

References deviceAddress.

Referenced by downloadPatch(), getAutomaticGainControl(), getCurrentReceivedSignalQuality(), getFirmware(), getRdsStatus(), getStatus(), patchPowerUp(), powerDown(), queryLibraryId(), radioPowerUp(), seekStation(), sendProperty(), sendSSBModeProperty(), setAutomaticGainControl(), setBandwidth(), setFrequency(), setRdsConfig(), setRdsIntSource(), setSSBBfo(), and ssbPowerUp().

The documentation for this class was generated from the following files:

SI4735/SI4735.h
SI4735/SI4735.cpp

---

si4735_digital_output_format Union Reference

Digital audio output format data structure (Property 0x0102. DIGITAL_OUTPUT_FORMAT).

`#include <SI4735.h>`

Detailed Description

Digital audio output format data structure (Property 0x0102. DIGITAL_OUTPUT_FORMAT).

Used to configure: DCLK edge, data format, force mono, and sample precision.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 195.

Definition at line 805 of file SI4735.h.

The documentation for this union was generated from the following file:

SI4735/SI4735.h

si4735_digital_output_sample_rate Struct Reference

Digital audio output sample structure (Property 0x0104. DIGITAL_OUTPUT_SAMPLE_RATE).

`#include <SI4735.h>`

Detailed Description

Digital audio output sample structure (Property 0x0104. DIGITAL_OUTPUT_SAMPLE_RATE).

Used to enable digital audio output and to configure the digital audio output sample rate in samples per second (sps).

See also

Si47XX PROGRAMMING GUIDE; AN332; page 196.

Definition at line 825 of file SI4735.h.

The documentation for this struct was generated from the following file:
SI4735/SI4735.h

si473x_powerup Union Reference

Power Up arguments data type.

`#include <SI4735.h>`

Detailed Description

Power Up arguments data type.

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 64 and 65

Definition at line 175 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

si47x_agc_overrride Union Reference

`#include <SI4735.h>`

Detailed Description

If FM, Overrides AGC setting by disabling the AGC and forcing the LNA to have a certain gain that ranges between 0 (minimum attenuation) and 26 (maximum attenuation). If AM, overrides the AGC setting by disabling the AGC and forcing the gain index that ranges between 0

See also

Si47XX PROGRAMMING GUIDE; AN332; For FM page 81; for AM page 143

Definition at line 737 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

---

si47x_agc_status Union Reference

```
#include <SI4735.h>
```

Detailed Description

AGC data types FM / AM and SSB structure to AGC

See also

Si47XX PROGRAMMING GUIDE; AN332; For FM page 80; for AM page 142

AN332 REV 0.8 Universal Programming Guide Amendment for SI4735-D60 SSB and NBFM patches; page 18.

Definition at line 708 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

---

si47x_antenna_capacitor Union Reference

Antenna Tuning Capacitor data type manupulation.

```
#include <SI4735.h>
```

Detailed Description

Antenna Tuning Capacitor data type manupulation.

Definition at line 209 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

---

si47x_bandwidth_config Union Reference

```
#include <SI4735.h>
```

Detailed Description

The bandwidth of the AM channel filter data type AMCHFLT values: 0 = 6 kHz Bandwidth

1 = 4 kHz Bandwidth 2 = 3 kHz Bandwidth 3 = 2 kHz Bandwidth 4 = 1 kHz Bandwidth 5 = 1.8 kHz Bandwidth 6 = 2.5 kHz Bandwidth, gradual roll off 7–15 = Reserved (Do not use)

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 125 and 151

Definition at line 764 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

si47x_firmware_information Union Reference

Data representation for Firmware Information (GET_REV)

`#include <SI4735.h>`

Detailed Description

Data representation for Firmware Information (GET_REV)

The part number, chip revision, firmware revision, patch revision and component revision numbers.

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 66 and 131

Definition at line 308 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

si47x_firmware_query_library Union Reference

Firmware Query Library ID response.

`#include <SI4735.h>`

Detailed Description

Firmware Query Library ID response.

Used to represent the response of a power up command with FUNC = 15 (patch)

To confirm that the patch is compatible with the internal device library revision, the library revision should be confirmed by issuing the POWER_UP command with Function = 15 (query library ID)

See also

Si47XX PROGRAMMING GUIDE; AN332; page 12

Definition at line 344 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

si47x_frequency Union Reference

Represents how the frequency is stored in the si4735.

```
#include <SI4735.h>
```

Detailed Description

Represents how the frequency is stored in the si4735.

It helps to convert frequency in uint16_t to two bytes (uint8_t) (FREQL and FREQH)

Definition at line 196 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

si47x_property Union Reference

Data type to deal with SET_PROPERTY command.

```
#include <SI4735.h>
```

Detailed Description

Data type to deal with SET_PROPERTY command.

Property Data type (help to deal with SET_PROPERTY command on si473X)

Definition at line 393 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

si47x_rds_blocka Union Reference

Block A data type.

```
#include <SI4735.h>
```

Detailed Description

Block A data type.

Definition at line 582 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

si47x_rds_blockb Union Reference

Block B data type.

`#include <SI4735.h>`

Detailed Description

Block B data type.

For GCC on System-V ABI on 386-compatible (32-bit processors), the following stands:

1) Bit-fields are allocated from right to left (least to most significant). 2) A bit-field must entirely reside in a storage unit appropriate for its declared type. Thus a bit-field never crosses its unit boundary. 3) Bit-fields may share a storage unit with other struct/union members, including members that are not bit-fields. Of course, struct members occupy different parts of the storage unit. 4) Unnamed bit-fields' types do not affect the alignment of a structure or union, although individual bit-fields' member offsets obey the alignment constraints.

See also

also Si47XX PROGRAMMING GUIDE; AN332; pages 78 and 79

also https://en.wikipedia.org/wiki/Radio_Data_System

Definition at line 612 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

si47x_rds_command Union Reference

Data type for RDS Status command and response information.

`#include <SI4735.h>`

Detailed Description

Data type for RDS Status command and response information.

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 77 and 78

Also https://en.wikipedia.org/wiki/Radio_Data_System

Definition at line 460 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

si47x_rds_config Union Reference

Data type for FM_RDS_CONFIG Property.

`#include <SI4735.h>`

Detailed Description

Data type for FM_RDS_CONFIG Property.

IMPORTANT: all block errors must be less than or equal the associated block error threshold for the group to be stored in the RDS FIFO. 0 = No errors; 1 = 1–2 bit errors detected and corrected; 2 = 3–5 bit errors detected and corrected; 3 = Uncorrectable. Recommended Block Error Threshold options: 2,2,2,2 = No group stored if any errors are uncorrected. 3,3,3,3 = Group stored regardless of errors. 0,0,0,0 = No group stored containing corrected or uncorrected errors. 3,2,3,3 = Group stored with corrected errors on B, regardless of errors on A, C, or D.

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 58 and 104

Definition at line 564 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

---

si47x_rds_date_time Union Reference

`#include <SI4735.h>`

Detailed Description

Group type 4A (RDS Date and Time) When group type 4A is used by the station, it shall be transmitted every minute according to EN 50067. This Structure uses blocks 2,3 and 5 (B,C,D)

ATTENTION: To make it compatible with 8, 16 and 32 bits platforms and avoid Crosses boundary, it was necessary to split minute and hour representation.

Definition at line 683 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

---

si47x_rds_int_source Union Reference

FM_RDS_INT_SOURCE property data type.

`#include <SI4735.h>`

Detailed Description

FM_RDS_INT_SOURCE property data type.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 103

also https://en.wikipedia.org/wiki/Radio_Data_System

Definition at line 533 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

---

si47x_rds_status Union Reference

Response data type for current channel and reads an entry from the RDS FIFO.

`#include <SI4735.h>`

Detailed Description

Response data type for current channel and reads an entry from the RDS FIFO.

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 77 and 78

Definition at line 478 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

---

si47x_response_status Union Reference

Response status command.

`#include <SI4735.h>`

Detailed Description

Response status command.

Response data from a query status command

See also

Si47XX PROGRAMMING GUIDE; pages 73 and

Definition at line 267 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

---

si47x_rqs_status Union Reference

Radio Signal Quality data representation.

`#include <SI4735.h>`

Detailed Description

Radio Signal Quality data representation.

Data type for status information about the received signal quality (FM_RSQ_STATUS and AM_RSQ_STATUS)

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 75 and

Definition at line 414 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

---

si47x_seek Union Reference

Seek frequency (automatic tuning)

`#include <SI4735.h>`

Detailed Description

Seek frequency (automatic tuning)

Represents searching for a valid frequency data type.

Definition at line 247 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

---

si47x_set_frequency Union Reference

AM Tune frequency data type command (AM_TUNE_FREQ command)

`#include <SI4735.h>`

Detailed Description

AM Tune frequency data type command (AM_TUNE_FREQ command)

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 135

Definition at line 225 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

si47x_ssb_mode Union Reference

```
#include <SI4735.h>
```

Detailed Description

SSB - datatype for SSB_MODE (property 0x0101)

See also

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; page 24

Definition at line 782 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

si47x_tune_status Union Reference

Seek station status.

```
#include <SI4735.h>
```

Detailed Description

Seek station status.

Status of FM_TUNE_FREQ or FM_SEEK_START commands or Status of AM_TUNE_FREQ or AM_SEEK_START commands.

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 73 and 139

Definition at line 376 of file SI4735.h.

The documentation for this union was generated from the following file:
SI4735/SI4735.h

Index

INDE