

# **Si4735 Arduino Library**

AUTHOR  
Version 1.1.8  
04/04/2020



# Table of Contents

Table of contents

---

## Deprecated List

Global [SI4735::analogPowerUp](#) (void)

Consider use radioPowerUp instead

---

## Module Index

### Modules

Here is a list of all modules:

Audio setup.....	2
Deal with Interrupt.....	4
Deal with Interrupt and I2C bus.....	4
FM Mono Stereo audio setup.....	28
FM RDS/DBDS.....	31
Frequency and Si47XX device status.....	37
Host and slave MCU setup.....	41
RDS Data types.....	43
Receiver Status and Setup.....	50
SI473X data types.....	53
SI4735-D60 Single Side Band (SSB) support.....	60
SI47XX device Mode, Band and Frequency setup.....	67
SI47XX device information and start up.....	70
SI47XX filter setup.....	71
Tools method.....	72
Tune.....	73

---

## File Index

### File List

Here is a list of all files with brief descriptions:

<a href="#">SI4735/SI4735.cpp</a> .....	75
<a href="#">SI4735/SI4735.h</a> .....	75

---

# Module Documentation

## Audio setup

### Functions

void [SI4735::digitalOutputFormat](#) (uint8\_t OSIZE, uint8\_t OMONO, uint8\_t OMODE, uint8\_t OFALL)

*Configures the digital audio output format.*

void [SI4735::digitalOutputSampleRate](#) (uint16\_t DOSR)

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

void [SI4735::setVolume](#) (uint8\_t volume)

*RESP8 - Returns the Chip Revision (ASCII).*

void [SI4735::setAudioMute](#) (bool off)

*Returns the current volume level.*

uint8\_t [SI4735::getVolume](#) ()

*Gets the current volume level.*

void [SI4735::volumeUp](#) ()

*Set sound volume level Up*

void [SI4735::volumeDown](#) ()

*Set sound volume level Down*

---

## Detailed Description

---

### Function Documentation

void **SI4735::digitalOutputFormat** (uint8\_t **OSIZE**, uint8\_t **OMONO**, uint8\_t **OMODE**, uint8\_t **OFALL**)

Configures the digital audio output format.

## Audio setup

Options: DCLK edge, data format, force mono, and sample precision.

**See also**

Si47XX PROGRAMMING GUIDE; AN332; page 195.

**Parameters**

<i>uint8_t</i>	OSIZE Digital Output Audio Sample Precision (0=16 bits, 1=20 bits, 2=24 bits, 3=8bits).
<i>uint8_t</i>	OMONO Digital Output Mono Mode (0=Use mono/stereo blend ).
<i>uint8_t</i>	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).
<i>uint8_t</i>	OFALL Digital Output DCLK Edge (0 = use DCLK rising edge, 1 = use DCLK falling edge)

**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**

<i>uint16_t</i>	DOSR Digital Output Sample Rate(32–48 ksps .0 to disable digital audio output).
-----------------	---

**uint8\_t SI4735::getVolume ()**

Gets the current volume level.

**See also**

[setVolume\(\)](#)

**Returns**

volume (domain: 0 - 63)

**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**

<i>value</i>	if true, mute the audio; if false unmute the audio.
--------------	---

**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**

<code>uint8_t</code>	volume (domain: 0 - 63)
----------------------	-------------------------

**void SI4735::volumeDown ()**

Set sound volume level Down

**See also**

[setVolume\(\)](#)

**void SI4735::volumeUp ()**

Set sound volume level Up

**See also**

[setVolume\(\)](#)

## Deal with Interrupt

---

**Detailed Description**

Deal with Interrupt

## Deal with Interrupt and I2C bus

**Data Structures**

class [SI4735](#)

[SI4735](#) Class. [More...](#)

**Functions**

[SI4735::SI4735](#) ()

Construct a new [SI4735::SI4735](#) object.

void [SI4735::waitInterrupt](#) (void)

Interrupt handle.

int16\_t [SI4735::getDeviceI2CAddress](#) (uint8\_t [resetPin](#))

I2C bus address setup.

void [SI4735::setDeviceI2CAddress](#) (uint8\_t senPin)  
*Sets the I2C Bus Address.*

void [SI4735::setDeviceOtherI2CAddress](#) (uint8\_t i2cAddr)  
*Sets the onther I2C Bus Address (for Si470X)*

---

## Detailed Description

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

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.

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.

---

## Data Structure Documentation

### class SI4735

[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

## Public Member Functions

[SI4735](#) ()

*Construct a new [SI4735::SI4735](#) object.*



void [reset](#) (void)  
*Reset the SI473X*

void [waitToSend](#) (void)  
*Wait for the si473x is ready (Clear to Send (CTS) status bit have to be 1).*

void [setup](#) (uint8\_t [resetPin](#), uint8\_t defaultFunction)  
*Starts the Si473X device.*

void [setup](#) (uint8\_t [resetPin](#), int [interruptPin](#), uint8\_t defaultFunction, uint8\_t audioMode=[SI473X\\_ANALOG\\_AUDIO](#))  
*Starts the Si473X device.*

void [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.*

void [radioPowerUp](#) (void)  
*Powerup the Si47XX.*

void [analogPowerUp](#) (void)  
*You have to call setPowerUp method before.*

void [powerDown](#) (void)  
*Moves the device from powerup to powerdown mode.*

void [setFrequency](#) (uint16\_t)  
*Set the frequency to the corrent function of the Si4735 (FM, AM or SSB)*

void [getStatus](#) ()  
*Gets the current status of the Si4735 (AM or FM)*

void [getStatus](#) (uint8\_t, uint8\_t)  
*Gets the current status of the Si4735 (AM or FM)*

uint16\_t [getFrequency](#) (void)  
*Gets the current frequency of the Si4735 (AM or FM)*

uint16\_t [getCurrentFrequency](#) ()  
*Gets the current frequency saved in memory.*

bool [getSignalQualityInterrupt](#) ()  
*Get the Signal Quality Interrupt status.*

bool [getRadioDataSystemInterrupt](#) ()  
*Get the Radio Data System (RDS) Interrupt status.*

bool [getTuneCompleteTriggered](#) ()  
*Get the Tune Complete status.*

bool [getStatusError](#) ()  
*Get the Status Error.*

bool [getStatusCTS](#) ()  
*Gets the Error flag Clear to Send.*

bool [getACFIndicator](#) ()  
*Returns true if the AFC rails (AFC Rail Indicator).*

bool [getBandLimit](#) ()  
*Returns true if a seek hit the band limit.*

bool [getStatusValid](#) ()  
*Gets the channel status.*

uint8\_t [getReceivedSignalStrengthIndicator](#) ()  
*Returns the value of Received Signal Strength Indicator (dB $\hat{I}$ / $\hat{V}$ ).*

uint8\_t [getStatusSNR](#) ()  
*Gets the SNR metric when tune is complete (dB)*

uint8\_t [getStatusMULT](#) ()  
*Get the Status the M U L T.*

uint8\_t [getAntennaTuningCapacitor](#) ()  
*Get the Antenna Tuning Capacitor value.*

void [getAutomaticGainControl](#) ()  
*Queries Automatic Gain Control STATUS.*

void [setAvcAmMaxGain](#) ()  
*Queries Automatic Gain Control STATUS.*

void [setAvcAmMaxGain](#) (uint8\_t gain)  
*Sets the maximum gain for automatic volume control.*

uint8\_t [getCurrentAvcAmMaxGain](#) ()  
*Sets the maximum gain for automatic volume control.*

void [setAmSoftMuteMaxAttenuation](#) (uint8\_t smattn=0)

*Sets the Am Soft Mute Max Attenuation.*

void [setSsbSoftMuteMaxAttenuation](#) (uint8\_t smattn=0)  
*Sets the SSB Soft Mute Max Attenuation object.*

bool [isAgcEnabled](#) ()  
*Checks if the AGC is enabled.*

uint8\_t [getAgcGainIndex](#) ()  
*Gets the current AGC gain index.*

void [setAutomaticGainControl](#) (uint8\_t AGCDIS, uint8\_t AGCIDX)  
*Automatic Gain Control setup.*

void [getCurrentReceivedSignalQuality](#) (uint8\_t INTACK)  
*Queries the status of the Received Signal Quality (RSQ) of the current channel.*

void [getCurrentReceivedSignalQuality](#) (void)  
*Queries the status of the Received Signal Quality (RSQ) of the current channel (FM\_RSQ\_STATUS)*

uint8\_t [getCurrentRSSI](#) ()  
*Get the current receive signal strength (0â€‘127 dBÎ¼V)*

uint8\_t [getCurrentSNR](#) ()  
*Gets the current SNR metric (0–127 dB).*

bool [getCurrentRssiDetectLow](#) ()  
*Checks if RSSI detected is LOW.*

bool [getCurrentRssiDetectHigh](#) ()  
*Checks if RSSI detected is high.*

bool [getCurrentSnrDetectLow](#) ()  
*Checks if SNR detect is low.*

bool [getCurrentSnrDetectHigh](#) ()  
*Checks if SNR detect is high.*

bool [getCurrentValidChannel](#) ()  
*Checks if the current channel is valid.*

bool [getCurrentAfcRailIndicator](#) ()  
*AFC Rail Indicator.*

bool [getCurrentSoftMuteIndicator](#) ()  
*Soft Mute Indicator.*

uint8\_t [getCurrentStereoBlend](#) ()  
*Gets the value of the amount of stereo blend in % (100 = full stereo, 0 = full mono).*

bool [getCurrentPilot](#) ()  
*Checks the current pilot.*

uint8\_t [getCurrentMultipath](#) ()  
*Gets the current Multipath.*

uint8\_t [getCurrentSignedFrequencyOffset](#) ()  
 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](#) ()  
*Gets the current volume level.*

void [volumeDown](#) ()  
*Set sound volume level Down*

void [volumeUp](#) ()  
*Set sound volume level Up*

uint8\_t [getCurrentVolume](#) ()  
void [setAudioMute](#) (bool off)  
*Returns the current volume level.*

void [digitalOutputFormat](#) (uint8\_t OSIZE, uint8\_t OMONO, uint8\_t OMODE, uint8\_t OFALL)  
*Configures the digital audio output format.*

void [digitalOutputSampleRate](#) (uint16\_t DOSR)  
*Enables digital audio output and configures digital audio output sample rate in samples per second (sps).*

void [setAM](#) ()  
*Sets the radio to AM function. It means: LW MW and SW.*

void [setFM](#) ()  
*Sets the radio to FM function.*

void [setAM](#) (uint16\_t fromFreq, uint16\_t toFreq, uint16\_t initialFreq, uint16\_t step)  
*Sets the radio to AM (LW/MW/SW) function.*

void [setFM](#) (uint16\_t fromFreq, uint16\_t toFreq, uint16\_t initialFreq, uint16\_t step)  
*Sets the radio to FM function.*

void [setBandwidth](#) (uint8\_t AMCHFLT, uint8\_t AMPLFLT)  
*Selects the bandwidth of the channel filter for AM reception.*

void [setFrequencyStep](#) (uint16\_t step)  
*Sets the current step value.*

uint8\_t [getTuneFrequencyFast](#) ()  
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](#) ()  
*Increments the current frequency on current band/function by using the current step.*

void [frequencyDown](#) ()  
*Decrements the current frequency on current band/function by using the current step.*

bool [isCurrentTuneFM](#) ()  
*Returns true if the current function is FM (FM\_TUNE\_FREQ).*

void [getFirmware](#) (void)  
*Gets firmware information.*

void [seekStation](#) (uint8\_t SEEKUP, uint8\_t WRAP)  
*Look for a station (Automatic tune)*

void [seekStationUp](#) ()  
*Search for the next station.*

void [seekStationDown](#) ()  
*Search the previous station.*

void [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.*

void [setSeekAmSpacing](#) (uint16\_t spacing)  
*Selects frequency spacing for AM seek. Default is 10 kHz spacing.*

void [setSeekSrnThreshold](#) (uint16\_t value)  
*Sets the SNR threshold for a valid AM Seek/Tune.*

void [setSeekRssiThreshold](#) (uint16\_t value)  
*Sets the RSSI threshold for a valid AM Seek/Tune.*

void [setFmBlendStereoThreshold](#) (uint8\_t parameter)  
*Sets RSSI threshold for stereo blend (Full stereo above threshold, blend below threshold).*

void [setFmBlendMonoThreshold](#) (uint8\_t parameter)  
*Sets RSSI threshold for mono blend (Full mono below threshold, blend above threshold).*

void [setFmBlendRssiStereoThreshold](#) (uint8\_t parameter)  
*Sets RSSI threshold for stereo blend. (Full stereo above threshold, blend below threshold.)*

void [setFmBlendRssiMonoThreshold](#) (uint8\_t parameter)  
*Sets RSSI threshold for mono blend (Full mono below threshold, blend above threshold).*

void [setFmBlendSnrStereoThreshold](#) (uint8\_t parameter)  
*Sets SNR threshold for stereo blend (Full stereo above threshold, blend below threshold).*

void [setFmBlendSnrMonoThreshold](#) (uint8\_t parameter)  
*Sets SNR threshold for mono blend (Full mono below threshold, blend above threshold).*

void [setFmBlendMultiPathStereoThreshold](#) (uint8\_t parameter)  
*Sets multipath threshold for stereo blend (Full stereo below threshold, blend above threshold).*

void [setFmBlendMultiPathMonoThreshold](#) (uint8\_t parameter)  
*Sets Multipath threshold for mono blend (Full mono above threshold, blend below threshold).*

void [setFmStereoOn](#) ()  
*Turn Off Stereo operation.*

void [setFmStereoOff](#) ()  
*Turn Off Stereo operation.*

void [RdsInit](#) ()  
*Starts the control member variables for RDS.*

void [setRdsIntSource](#) (uint8\_t RDSNEWBLOCKB, uint8\_t RDSNEWBLOCKA, uint8\_t RDSSYNCFFOUND, uint8\_t RDSSYNCLST, uint8\_t RDSRECV)  
*Configures interrupt related to RDS.*

void [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.*

void [getRdsStatus](#) ()  
*Gets RDS Status.*

bool [getRdsReceived](#) ()  
 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 RDSSEN, uint8\_t BLETHA, uint8\_t BLETHB, uint8\_t BLETHC, uint8\_t BLETHD)  
*Sets RDS property.*

uint16\_t [getRdsPI](#) (void)  
*Returns the programa type.*

uint8\_t [getRdsGroupType](#) (void)  
*Returns the Group Type (extracted from the Block B)*

uint8\_t [getRdsFlagAB](#) (void)  
*Returns the current Text Flag A/B*

uint8\_t [getRdsVersionCode](#) (void)  
*Gets the version code (extracted from the Block B)*

uint8\_t [getRdsProgramType](#) (void)  
*Returns the Program Type (extracted from the Block B)*

uint8\_t [getRdsTextSegmentAddress](#) (void)  
*Returns the address of the text segment.*

char \* [getRdsText](#) (void)  
*Gets the RDS Text when the message is of the Group Type 2 version A.*

char \* [getRdsText0A](#) (void)  
*Gets the station name and other messages.*

char \* [getRdsText2A](#) (void)  
*Gets the Text processed for the 2A group.*



char \* [getRdsText2B](#) (void)  
*Gets the Text processed for the 2B group.*

char \* [getRdsTime](#) (void)  
*Gets the RDS time and date when the Group type is 4.*

void [getNext2Block](#) (char \*)  
*Process data received from group 2B.*

void [getNext4Block](#) (char \*)  
*Process data received from group 2A.*

void [ssbSetup](#) ()  
*Starts the Si473X device on SSB (same AM Mode).*

void [setSSBBfo](#) (int offset)  
*Sets the SSB Beat Frequency Offset (BFO).*

void [setSSBConfig](#) (uint8\_t AUDIOBW, uint8\_t SBCUTFLT, uint8\_t AVC\_DIVIDER, uint8\_t AVCEN, uint8\_t SMUTESEL, uint8\_t DSP\_AFCDIS)  
*Sets the SSB receiver mode.*

void [setSSB](#) (uint16\_t fromFreq, uint16\_t toFreq, uint16\_t intialFreq, uint16\_t step, uint8\_t usblsb)  
void [setSSB](#) (uint8\_t usblsb)  
*Set the radio to AM function.*

void [setSSBAudioBandwidth](#) (uint8\_t AUDIOBW)  
*SSB Audio Bandwidth for SSB mode.*

void [setSSBAutomaticVolumeControl](#) (uint8\_t AVCEN)  
*Sets SSB Automatic Volume Control (AVC) for SSB mode.*

void [setSSBSidebandCutoffFilter](#) (uint8\_t SBCUTFLT)  
*Sets SSB Sideband Cutoff Filter for band pass and low pass filters.*

void [setSSBAvcDivider](#) (uint8\_t AVC\_DIVIDER)  
*Sets AVC Divider.*

void [setSSBDspAfc](#) (uint8\_t DSP\_AFCDIS)  
*Sets DSP AFC disable or enable.*

void [setSSBSoftMute](#) (uint8\_t SMUTESEL)  
*Sets SSB Soft-mute Based on RSSI or SNR Selection:*

[si47x\\_firmware\\_query\\_library\\_queryLibraryId](#) ()  
*Query the library information of the Si47XX device.*

void [patchPowerUp](#) ()

*This method can be used to prepare the device to apply SSBRX patch.*

bool [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.*

bool [downloadPatch](#) (int eeprom\_i2c\_address)

*Transfers the content of a patch stored in a eeprom to the [SI4735](#) device.*

void [ssbPowerUp](#) ()

*This function can be useful for debug and test.*

void [setI2CLowSpeedMode](#) (void)

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)

*Sets the I2C Bus Address.*

int16\_t [getDeviceI2CAddress](#) (uint8\_t resetPin)

*I2C bus address setup.*

void [setDeviceOtherI2CAddress](#) (uint8\_t i2cAddr)

*Sets the onther I2C Bus Address (for Si470X)*

### **Protected Member Functions**

void [waitInterrupt](#) (void)

*Interrupt handle.*

void [sendProperty](#) (uint16\_t propertyValue, uint16\_t param)

*Sends (sets) property to the SI47XX.*

void [sendSSBModeProperty](#) ()

*Just send the property SSB\_MOD to the device. Internal use (privete method).*

void [disableFmDebug](#) ()

*There is a debug feature that remains active in Si4704/05/3x-D60 firmware which can create periodic noise in audio.*

void [clearRdsBuffer2A](#) ()

*Clear RDS buffer 2A (text)*

void [clearRdsBuffer2B](#) ()  
*Clear RDS buffer 2B (text)*

void [clearRdsBuffer0A](#) ()  
*Clear RDS buffer 0A (text)*

### Protected Attributes

char [rds\\_buffer2A](#) [65]  
char [rds\\_buffer2B](#) [33]  
*RDS Radio Text buffer - Program Information.*

char [rds\\_buffer0A](#) [9]  
*RDS Radio Text buffer - Station Information.*

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 [resetPin](#)  
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\\_rqs\\_status](#) [currentRqsStatus](#)  
[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*

uint8\_t [volume](#) = 32  
uint8\_t [currentSsbStatus](#)

---

## Member Function Documentation

**bool SI4735::getACFIndicator () [inline]**

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

**Returns**

true

**uint8\_t SI4735::getAgcGainIndex () [inline]**

Gets the current AGC gain index.

**Returns**

uint8\_t The current AGC gain index.

**uint8\_t SI4735::getAntennaTuningCapacitor () [inline]**

Get the Antenna Tuning Capacitor value.

Returns the current antenna tuning capacitor value.

**Returns**

uint8\_t capacitance

**bool SI4735::getBandLimit () [inline]**

Returns true if a seek hit the band limit.

(WRAP = 0 in FM\_START\_SEEK) or wrapped to the original frequency(WRAP = 1).

**Returns**

BLTF

**bool SI4735::getCurrentAfcRailIndicator () [inline]**

AFC Rail Indicator.

**Returns**

true or false

**uint8\_t SI4735::getCurrentAvcAmMaxGain () [inline]**

Sets the maximum gain for automatic volume control.

Get the current Avc Am Max Gain

**Returns**

uint8\_t Current AVC gain index value

**bool SI4735::getCurrentBlendDetectInterrupt () [inline]**

Multipath Detect High.

**uint8\_t SI4735::getCurrentMultipath () [inline]**

Gets the current Multipath.

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

**Returns**

uint8\_t value (0 to 100)

**bool SI4735::getCurrentMultipathDetectHigh () [inline]**

Multipath Detect Low.

**bool SI4735::getCurrentMultipathDetectLow () [inline]**

Signed frequency offset (kHz).

**bool SI4735::getCurrentPilot () [inline]**

Checks the current pilot.

Indicates stereo pilot presence.

**Returns**

true if stereo pilot presence has detected

**uint8\_t SI4735::getCurrentRSSI () [inline]**

Get the current receive signal strength (0â€“127 dBÎ¼V)

**Returns**

uint8\_t a value between 0 to 127

**bool SI4735::getCurrentRssiDetectHigh () [inline]**

Checks if RSSI detected is high.

**Returns**

true if RSSI detected is high

**bool SI4735::getCurrentRssiDetectLow () [inline]**

Checks if RSSI detected is LOW.

**Returns**

true if RSSI is low

**uint8\_t SI4735::getCurrentSignedFrequencyOffset () [inline]**

**uint8\_t SI4735::getCurrentSNR () [inline]**

Gets the current SNR metric (0–127 dB).

**Returns**

uint8\_t SNR value in dB (0-127)

**bool SI4735::getCurrentSnrDetectHigh () [inline]**

Checks if SNR detect is high.

**Returns**

true if SNR detect is high

**bool SI4735::getCurrentSnrDetectLow () [inline]**

Checks if SNR detect is low.

**Returns**

true if SNR detected is low

**bool SI4735::getCurrentSoftMuteIndicator () [inline]**

Soft Mute Indicator.

Indicates soft mute is engaged.

**Returns**

true if soft mute indicates is engaged.

**uint8\_t SI4735::getCurrentStereoBlend () [inline]**

Gets the value of the amount of stereo blend in % (100 = full stereo, 0 = full mono).

**Returns**

uint8\_t value (0 to 100)

**bool SI4735::getCurrentValidChannel () [inline]**

Checks if the current channel is valid.

**Returns**

true if the current channel is valid

**uint8\_t SI4735::getCurrentVolume () [inline]**

**uint8\_t SI4735::getFirmwareCHIPREV () [inline]**

RESP7 - Returns the Component Minor Revision (ASCII).

**uint8\_t SI4735::getFirmwareCMPMAJOR () [inline]**

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

**uint8\_t SI4735::getFirmwareCMPMINOR () [inline]**

RESP6 - Returns the Component Major Revision (ASCII).

**uint8\_t SI4735::getFirmwareFWMAJOR () [inline]**

RESP1 - Part Number (HEX)

**uint8\_t SI4735::getFirmwareFWMINOR () [inline]**

RESP2 - Returns the Firmware Major Revision (ASCII).

**uint8\_t SI4735::getFirmwarePATCHH () [inline]**

RESP3 - Returns the Firmware Minor Revision (ASCII).

**uint8\_t SI4735::getFirmwarePATCHL () [inline]**

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

**uint8\_t SI4735::getFirmwarePN () [inline]**

Blend Detect Interrupt.

**bool SI4735::getGroupLost () [inline]**

1 = RDS currently synchronized.

**uint8\_t SI4735::getNumRdsFifoUsed () [inline]**

1 = One or more RDS groups discarded due to FIFO overrun.

**bool SI4735::getRadioDataSystemInterrupt () [inline]**

Get the Radio Data System (RDS) Interrupt status.

#### **See also**

Si47XX PROGRAMMING GUIDE; AN332; pages 63

#### **Returns**

RDSINT status

**bool SI4735::getRdsNewBlockA () [inline]**

1 = Found RDS synchronization

Referenced by getRdsPI().

**bool SI4735::getRdsNewBlockB () [inline]**

1 = Valid Block A data has been received.

**bool SI4735::getRdsReceived () [inline]**

Referenced by getRdsPI(), getRdsText0A(), and getRdsText2A().

**bool SI4735::getRdsSync () [inline]**

1 = Valid Block B data has been received.

**bool SI4735::getRdsSyncFound () [inline]**

1 = Lost RDS synchronization



**bool SI4735::getRdsSyncLost () [inline]**

1 = FIFO filled to minimum number of groups

**uint8\_t SI4735::getReceivedSignalStrengthIndicator () [inline]**

Returns the value of Received Signal Strength Indicator (dB $\hat{I}$  $\frac{1}{4}$ V).

**Returns**

uint8\_t

**bool SI4735::getSignalQualityInterrupt () [inline]**

Get the Signal Quality Interrupt status.

STATUS RESPONSE Set of methods to get current status information. Call them after getStatus or getFrequency or seekStation

**See also**

Si47XX PROGRAMMING GUIDE; AN332; pages 63

Si47XX PROGRAMMING GUIDE; AN332; pages 63

**Returns**

RDSINT status

**bool SI4735::getStatusCTS () [inline]**

Gets the Error flag Clear to Send.

**Returns**

CTS

**bool SI4735::getStatusError () [inline]**

Get the Status Error.

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

**Returns**

Error flag

**uint8\_t SI4735::getStatusMULT () [inline]**

Get the Status the M U L T.

Returns the value containing the multipath metric when tune is complete.

**Returns**

uint8\_t

**uint8\_t SI4735::getStatusSNR () [inline]**

Gets the SNR metric when tune is complete (dB)

Returns the value of the SNR metric when tune is complete (dB).

**Returns**

uint8\_t

**bool SI4735::getStatusValid () [inline]**

Gets the channel status.

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

**Returns**

true

false

**bool SI4735::getTuneCompleteTriggered () [inline]**

Get the Tune Complete status.

Seek/Tune Complete Interrupt; 1 = Tune complete has been triggered.

**Returns**

STCINT status

**uint8\_t SI4735::getTuneFrequencyFast () [inline]**

**uint8\_t SI4735::getTuneFrequencyFreeze () [inline]**

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

**bool SI4735::isAgcEnabled () [inline]**

Checks if the AGC is enabled.

**Returns**

true if the AGC is enabled

**void SI4735::setAmSoftMuteMaxAttenuation (uint8\_t *smattn* = 0) [inline]**

Sets the Am Soft Mute Max Attenuation.

This function can be useful to disable Soft Mute. The value 0 disable soft mute.

Specified in units of dB. Default maximum attenuation is 8 dB.

**See also**

Si47XX PROGRAMMING GUIDE; AN332; page 158.

**Parameters**

<i>smattn</i>	Maximum attenuation to apply when in soft mute
---------------	--

**void SI4735::setAvcAmMaxGain () [inline]**

Queries Automatic Gain Control STATUS.

Sets the Avc Am Max Gain to 48dB

**void SI4735::setI2CFastMode (void ) [inline]**

Sets I2C buss to 100KHz.

**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**

<i>value</i>	in Hz. For example: The values 500000 sets the bus to 500KHz.
--------------	---

**void SI4735::setI2CLowSpeedMode (void ) [inline]**

**void SI4735::setI2CStandardMode (void ) [inline]**

Sets I2C buss to 10KHz.

**void SI4735::setSsbSoftMuteMaxAttenuation (uint8\_t smattn = 0) [inline]**

Sets the SSB Soft Mute Max Attenuation object.

Sets maximum attenuation during soft mute (dB). Set to 0 to disable soft mute.

Specified in units of dB. Default maximum attenuation is 8 dB.

**Parameters**

<i>smattn</i>	Maximum attenuation to apply when in soft mute.
---------------	---

**void SI4735::setTuneFrequencyFast (uint8\_t FAST) [inline]**

Returns the FAST tuning status.

**void SI4735::setTuneFrequencyFreeze (uint8\_t FREEZE) [inline]**

Returns the FREEZE status.

---

**Field Documentation**

[si47x\\_agc\\_status](#) SI4735::currentAgcStatus [protected]

current RDS status

**uint8\_t SI4735::currentAvcAmMaxGain = 48 [protected]**

Store the last mode used.

[si47x\\_frequency](#) SI4735::currentFrequency [protected]

Automatic Volume Control Gain for AM - Default 48.

[si47x\\_set\\_frequency](#) SI4735::currentFrequencyParams [protected]

data structure to get current frequency

**uint16\_t SI4735::currentMaximumFrequency** [protected]

minimum frequency of the current band

**uint16\_t SI4735::currentMinimumFrequency** [protected]

tell the current tune (FM, AM or SSB)

[\*\*si47x\\_rds\\_status\*\*](#) **SI4735::currentRdsStatus** [protected]

firmware information

[\*\*si47x\\_rqs\\_status\*\*](#) **SI4735::currentRqsStatus** [protected]

[\*\*si47x\\_ssb\\_mode\*\*](#) **SI4735::currentSSBMode** [protected]

current AGC status

**uint8\_t SI4735::currentSsbStatus** [protected]

[\*\*si47x\\_response\\_status\*\*](#) **SI4735::currentStatus** [protected]

current Radio Signal Quality status

**uint16\_t SI4735::currentStep** [protected]

current frequency

**uint8\_t SI4735::currentTune** [protected]

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

**uint16\_t SI4735::currentWorkFrequency** [protected]

maximum frequency of the current band

**int16\_t SI4735::deviceAddress** = [\*\*SI473X\\_ADDR\\_SEN\\_LOW\*\*](#) [protected]

rds\_buffer0A current position

[\*\*si47x\\_firmware\\_information\*\*](#) **SI4735::firmwareInfo** [protected]

current device status

**uint8\_t SI4735::interruptPin** [protected]

pin used on Arduino Board to RESET the Si47XX device

**uint8\_t SI4735::lastMode** = -1 [protected]

current steps

**uint8\_t SI4735::lastTextFlagAB** [protected]

current I2C buss address

[si473x\\_powerup](#) **SI4735::powerUp** [protected]

indicates if USB or LSB

**char SI4735::rds\_buffer0A[9]** [protected]

RDS Radio Text buffer - Station Information.

Referenced by clearRdsBuffer0A(), and getRdsText0A().

**char SI4735::rds\_buffer2A[65]** [protected]

Referenced by clearRdsBuffer2A(), getRdsText(), and getRdsText2A().

**char SI4735::rds\_buffer2B[33]** [protected]

RDS Radio Text buffer - Program Information.

Referenced by clearRdsBuffer2B(), and getRdsText2B().

**char SI4735::rds\_time[20]** [protected]

RDS Basic tuning and switching information (Type 0 groups)

Referenced by getRdsTime().

**int SI4735::rdsTextAddress0A** [protected]

rds\_buffer2B current position

Referenced by getRdsText0A().

**int SI4735::rdsTextAddress2A** [protected]

RDS date time received information

Referenced by getRdsText(), and getRdsText2A().

**int SI4735::rdsTextAddress2B** [protected]

rds\_buffer2A current position

Referenced by getRdsText2B().

**uint8\_t SI4735::resetPin** [protected]

**uint8\_t SI4735::volume = 32** [protected]

---

## Function Documentation

**int16\_t SI4735::getDeviceI2CAddress** (uint8\_t *resetPin*)

I2C bus address setup.

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

<i>uint8_t</i>	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.

### **void SI4735::setDeviceI2CAddress (uint8\_t senPin)**

Sets the I2C Bus Address.

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

<i>senPin</i>	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).
---------------	---

### **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

<i>uint8_t</i>	i2cAddr (example 0x10)
----------------	------------------------

### **SI4735::SI4735 ()**

Construct a new [SI4735::SI4735](#) object.

## **Deal with Interrupt and I2C bus**

### **void SI4735::waitInterrupr (void ) [protected]**

Interrupt handle.

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

## FM Mono Stereo audio setup

### Functions

void [SI4735::setFmBlendStereoThreshold](#) (uint8\_t parameter)

*Sets RSSI threshold for stereo blend (Full stereo above threshold, blend below threshold).*

void [SI4735::setFmBlendMonoThreshold](#) (uint8\_t parameter)

*Sets RSSI threshold for mono blend (Full mono below threshold, blend above threshold).*

void [SI4735::setFmBlendRssiStereoThreshold](#) (uint8\_t parameter)

*Sets RSSI threshold for stereo blend. (Full stereo above threshold, blend below threshold.)*

void [SI4735::setFmBlendRssiMonoThreshold](#) (uint8\_t parameter)

*Sets RSSI threshold for mono blend (Full mono below threshold, blend above threshold).*

void [SI4735::setFmBlendSnrStereoThreshold](#) (uint8\_t parameter)

*Sets SNR threshold for stereo blend (Full stereo above threshold, blend below threshold).*

void [SI4735::setFmBlendSnrMonoThreshold](#) (uint8\_t parameter)

*Sets SNR threshold for mono blend (Full mono below threshold, blend above threshold).*

void [SI4735::setFmBlendMultiPathStereoThreshold](#) (uint8\_t parameter)

*Sets multipath threshold for stereo blend (Full stereo below threshold, blend above threshold).*

void [SI4735::setFmBlendMultiPathMonoThreshold](#) (uint8\_t parameter)

*Sets Multipath threshold for mono blend (Full mono above threshold, blend below threshold).*

void [SI4735::setFmStereoOff](#) ()

*Turn Off Stereo operation.*

void [SI4735::setFmStereoOn](#) ()

*Turn Off Stereo operation.*

void [SI4735::disableFmDebug](#) ()

*There is a debug feature that remains active in Si4704/05/3x-D60 firmware which can create periodic noise in audio.*

---

### Detailed Description

---

## Function Documentation

### **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.

Referenced by SI4735::setFM().

### **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 dB $\hat{I}$ / $\frac{1}{4}$ V.

#### **See also**

Si47XX PROGRAMMING GUIDE; AN332; page 56.

#### **Parameters**

<i>parameter</i>	valid values: 0 to 127
------------------	------------------------

### **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**

<i>parameter</i>	valid values: 0 to 100
------------------	------------------------

### **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**

<i>parameter</i>	valid values: 0 to 100
------------------	------------------------

### **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 dB $\hat{I}$ / $\frac{1}{4}$ V.



**See also**

Si47XX PROGRAMMING GUIDE; AN332; page 59.

**Parameters**

<i>parameter</i>	valid values: 0 to 127
------------------	------------------------

**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 dB $\hat{1}$ /<sub>4</sub>V.

**See also**

Si47XX PROGRAMMING GUIDE; AN332; page 59.

**Parameters**

<i>parameter</i>	valid values: 0 to 127
------------------	------------------------

**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**

<i>parameter</i>	valid values: 0 to 127
------------------	------------------------

**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**

<i>parameter</i>	valid values: 0 to 127
------------------	------------------------

**void SI4735::setFmBlendStereoThreshold (uint8\_t *parameter*)**

Sets RSSI threshold for stereo blend (Full stereo above threshold, blend below threshold).

## FM Mono Stereo audio setup

To force stereo, set this to 0. To force mono, set this to 127.

**See also**

Si47XX PROGRAMMING GUIDE; AN332; page 90.

**Parameters**

<i>parameter</i>	valid values: 0 to 127
------------------	------------------------

**void SI4735::setFmStereoOff ()**

Turn Off Stereo operation.

TO DO

**void SI4735::setFmStereoOn ()**

Turn Off Stereo operation.

TO DO

## FM RDS/DBDS

### Functions

void [SI4735::RdsInit](#) ()

*Starts the control member variables for RDS.*

void [SI4735::clearRdsBuffer2A](#) ()

*Clear RDS buffer 2A (text)*

void [SI4735::clearRdsBuffer2B](#) ()

*Clear RDS buffer 2B (text)*

void [SI4735::clearRdsBuffer0A](#) ()

*Clear RDS buffer 0A (text)*

void [SI4735::setRdsConfig](#) (uint8\_t RDSSEN, uint8\_t BLETHA, uint8\_t BLETHB, uint8\_t BLETHC, uint8\_t BLETHD)

*Sets RDS property.*

void [SI4735::setRdsIntSource](#) (uint8\_t RDSNEWBLOCKB, uint8\_t RDSNEWBLOCKA, uint8\_t RDSSYNCFOUND, uint8\_t RDSSYNCCLOST, uint8\_t RDSRECV)

*Configures interrupt related to RDS.*

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.*

void [SI4735::getRdsStatus](#) ()

*Gets RDS Status.*

uint16\_t [SI4735::getRdsPI](#) (void)

*Returns the programa type.*

uint8\_t [SI4735::getRdsGroupType](#) (void)  
*Returns the Group Type (extracted from the Block B)*

uint8\_t [SI4735::getRdsFlagAB](#) (void)  
*Returns the current Text Flag A/B*

uint8\_t [SI4735::getRdsTextSegmentAddress](#) (void)  
*Returns the address of the text segment.*

uint8\_t [SI4735::getRdsVersionCode](#) (void)  
*Gets the version code (extracted from the Block B)*

uint8\_t [SI4735::getRdsProgramType](#) (void)  
*Returns the Program Type (extracted from the Block B)*

void [SI4735::getNext2Block](#) (char \*)  
*Process data received from group 2B.*

void [SI4735::getNext4Block](#) (char \*)  
*Process data received from group 2A.*

char \* [SI4735::getRdsText](#) (void)  
*Gets the RDS Text when the message is of the Group Type 2 version A.*

char \* [SI4735::getRdsText0A](#) (void)  
*Gets the station name and other messages.*

char \* [SI4735::getRdsText2A](#) (void)  
*Gets the Text processed for the 2A group.*

char \* [SI4735::getRdsText2B](#) (void)  
*Gets the Text processed for the 2B group.*

char \* [SI4735::getRdsTime](#) (void)  
*Gets the RDS time and date when the Group type is 4.*

---

## Detailed Description

---

## Function Documentation

**void SI4735::clearRdsBuffer0A () [protected]**

Clear RDS buffer 0A (text)

References SI4735::rds\_buffer0A.

Referenced by SI4735::getRdsStatus(), and SI4735::RdsInit().

**void SI4735::clearRdsBuffer2A () [protected]**

Clear RDS buffer 2A (text)

References SI4735::rds\_buffer2A.

Referenced by SI4735::getRdsStatus(), and SI4735::RdsInit().

**void SI4735::clearRdsBuffer2B () [protected]**

Clear RDS buffer 2B (text)

References SI4735::rds\_buffer2B.

Referenced by SI4735::getRdsStatus(), and SI4735::RdsInit().

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

Process data received from group 2B.

#### Parameters

<i>c</i>	char array reference to the "group 2B" text
----------	---

Referenced by SI4735::getRdsText0A(), and SI4735::getRdsText2B().

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

Process data received from group 2A.

#### Parameters

<i>c</i>	char array reference to the "group 2A" text
----------	---

Referenced by SI4735::getRdsText(), and SI4735::getRdsText2A().

**uint8\_t SI4735::getRdsFlagAB (void )**

Returns the current Text Flag A/B

#### Returns

uint8\_t current Text Flag A/B

**uint8\_t SI4735::getRdsGroupType (void )**

Returns the Group Type (extracted from the Block B)

### Returns

BLOCKBL

### 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

References SI4735::getRdsNewBlockA(), and SI4735::getRdsReceived().

### uint8\_t SI4735::getRdsProgramType (void )

Returns the Program Type (extracted from the Block B)

### See also

[https://en.wikipedia.org/wiki/Radio\\_Data\\_System](https://en.wikipedia.org/wiki/Radio_Data_System)

### Returns

program type (an integer between 0 and 31)

### void SI4735::getRdsStatus ()

Gets RDS Status.

Same result of calling getRdsStatus(0,0,0).

Please, call [getRdsStatus\(uint8\\_t INTACK, uint8\\_t MTFIFO, uint8\\_t STATUSONLY\)](#) instead [getRdsStatus\(\)](#) if you want other behaviour.

### See also

[SI4735::getRdsStatus\(uint8\\_t INTACK, uint8\\_t MTFIFO, uint8\\_t STATUSONLY\)](#)

### 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

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

References SI4735::clearRdsBuffer0A(), SI4735::clearRdsBuffer2A(), SI4735::clearRdsBuffer2B(), and SI4735::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  
References SI4735::getNext4Block(), SI4735::rds\_buffer2A, and SI4735::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  
References SI4735::getNext2Block(), SI4735::getRdsReceived(), SI4735::rds\_buffer0A, and SI4735::rdsTextAdress0A.

### **char \* SI4735::getRdsText2A (void )**

Gets the Text processed for the 2A group.

#### **Returns**

char\* string with the Text of the group A2  
References SI4735::getNext4Block(), SI4735::getRdsReceived(), SI4735::rds\_buffer2A, and SI4735::rdsTextAdress2A.

### **char \* SI4735::getRdsText2B (void )**

Gets the Text processed for the 2B group.

#### **Returns**

char\* string with the Text of the group AB  
References SI4735::getNext2Block(), SI4735::rds\_buffer2B, and SI4735::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.

**char \* SI4735::getRdsTime (void )**

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

**Returns**

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

References SI4735::rds\_time.

**uint8\_t SI4735::getRdsVersionCode (void )**

Gets the version code (extracted from the Block B)

**Returns**

0=A or 1=B

**void SI4735::RdsInit ()**

Starts the control member variables for RDS.

**FM RDS/DBDS**

RDS implementation

This method is called by [setRdsConfig\(\)](#)

**See also**

[setRdsConfig\(\)](#)

References SI4735::clearRdsBuffer0A(), SI4735::clearRdsBuffer2A(), and SI4735::clearRdsBuffer2B().

Referenced by SI4735::setRdsConfig().

**void SI4735::setRdsConfig (uint8\_t *RDSEN*, uint8\_t *BLETHA*, uint8\_t *BLETHB*, uint8\_t *BLETHC*, uint8\_t *BLETHD*)**

Sets RDS property.

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.

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.

Value	Description
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:

Exemples	Description
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; page 104

### Parameters

<i>uint8_t</i>	RDSSEN RDS Processing Enable; 1 = RDS processing enabled.
<i>uint8_t</i>	BLETHA Block Error Threshold BLOCKA.
<i>uint8_t</i>	BLETHB Block Error Threshold BLOCKB.
<i>uint8_t</i>	BLETHC Block Error Threshold BLOCKC.
<i>uint8_t</i>	BLETHD Block Error Threshold BLOCKD.

References SI4735::RdsInit(), and SI4735::waitToSend().

**void SI4735::setRdsIntSource (uint8\_t *RDSNEWBLOCKB*, uint8\_t *RDSNEWBLOCKA*, uint8\_t *RDSSYNCFFOUND*, uint8\_t *RDSSYNCFLOST*, 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

<i>RDSRECV</i>	If set, generate RDSINT when RDS FIFO has at least FM_RDS_INT_FIFO_COUNT entries.
<i>RDSSYNCFLOST</i>	If set, generate RDSINT when RDS loses synchronization.
<i>RDSSYNCFFOUND</i>	set, generate RDSINT when RDS gains synchronization.
<i>RDSNEWBLOCKA</i>	If set, generate an interrupt when Block A data is found or subsequently changed
<i>RDSNEWBLOCKB</i>	If set, generate an interrupt when Block B data is found or subsequently changed

References SI4735::waitToSend().

## Frequency and Si47XX device status

### Functions

uint16\_t [SI4735::getFrequency](#) (void)

*Gets the current frequency of the Si4735 (AM or FM)*



`uint16_t SI4735::getCurrentFrequency ()`  
*Gets the current frequency saved in memory.*

`void SI4735::getStatus (uint8_t, uint8_t)`  
*Gets the current status of the Si4735 (AM or FM)*

`void SI4735::getStatus ()`  
*Gets the current status of the Si4735 (AM or FM)*

`void SI4735::getAutomaticGainControl ()`  
*Queries Automatic Gain Control STATUS.*

`void SI4735::setAutomaticGainControl (uint8_t AGCDIS, uint8_t AGCIDX)`  
*Automatic Gain Control setup.*

`void SI4735::setAvcAmMaxGain (uint8_t gain)`  
*Sets the maximum gain for automatic volume control.*

`void SI4735::getCurrentReceivedSignalQuality (uint8_t INTACK)`  
*Queries the status of the Received Signal Quality (RSQ) of the current channel.*

`void SI4735::getCurrentReceivedSignalQuality (void)`  
*Queries the status of the Received Signal Quality (RSQ) of the current channel (FM\_RSQ\_STATUS)*

---

## Detailed Description

---

## Function Documentation

### **void SI4735::getAutomaticGainControl ()**

Queries Automatic Gain Control STATUS.

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

#### **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.

References `SI4735::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\(\)](#)

**void SI4735::getCurrentReceivedSignalQuality (uint8\_t INTACK)**

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

This method should be called before call [getCurrentRSSI\(\)](#), [getCurrentSNR\(\)](#) etc.  
Command FM\_RSQ\_STATUS

**See also**

Si47XX PROGRAMMING GUIDE; AN332; pages 75 and 141

**Parameters**

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

References SI4735::waitToSend().

**void SI4735::getCurrentReceivedSignalQuality (void )**

Queries the status of the Received Signal Quality (RSQ) of the current channel (FM\_RSQ\_STATUS)

**See also**

Si47XX PROGRAMMING GUIDE; AN332; pages 75 and 141

**Parameters**

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

**uint16\_t SI4735::getFrequency (void )**

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

## Frequency and Si47XX device status

Device Status Information

The method status do it an more. See `getStatus` below.

**See also**

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

## **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)

## **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**

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

References SI4735::waitToSend().

## **void SI4735::setAutomaticGainControl (uint8\_t AGCDIS, uint8\_t AGCIDX)**

Automatic Gain Control setup.

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**

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

References SI4735::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

[setAvcAmMaxGain\(\)](#)

### **Parameters**

<i>uint8_t</i>	gain Select a value between 12 and 192. Defaul value 48dB.
----------------	--

## Host and slave MCU setup

### Functions

void [SI4735::reset](#) (void)

*Reset the SI473X*

void [SI4735::waitToSend](#) (void)

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

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.*

void [SI4735::radioPowerUp](#) (void)

*Powerup the Si47XX.*

void [SI4735::analogPowerUp](#) (void)

*You have to call setPowerUp method before.*

void [SI4735::powerDown](#) (void)

*Moves the device from powerup to powerdown mode.*

---

### Detailed Description

---

### Function Documentation

#### **void SI4735::analogPowerUp (void )**

You have to call setPowerUp method before.

#### **Deprecated:**

Consider use radioPowerUp instead

#### **See also**

[SI4735::setPowerUp\(\)](#)

Si47XX PROGRAMMING GUIDE; AN332; pages 64, 129

References [SI4735::radioPowerUp\(\)](#).

#### **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

[radioPowerUp\(\)](#)

References SI4735::waitToSend().

Referenced by SI4735::queryLibraryId(), SI4735::setAM(), and SI4735::setFM().

**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

Parameter	Description
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

References SI4735::waitToSend().

Referenced by SI4735::analogPowerUp(), SI4735::setAM(), SI4735::setFM(), SI4735::setSSB(), and SI4735::setup().

**void SI4735::reset (void )**

Reset the SI473X

## Host and slave MCU setup

**See also**

Si47XX PROGRAMMING GUIDE; AN332;

Referenced by SI4735::setup(), and SI4735::ssbSetup().

**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 change the default behavior of the Si473X. Use it before PowerUp()

**See also**

Si47XX PROGRAMMING GUIDE; AN332; pages 65 and 129

**Parameters**

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

**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

Referenced by SI4735::downloadPatch(), SI4735::getAutomaticGainControl(), SI4735::getCurrentReceivedSignalQuality(), SI4735::getFirmware(), SI4735::getRdsStatus(), SI4735::getStatus(), SI4735::patchPowerUp(), SI4735::powerDown(), SI4735::queryLibraryId(), SI4735::radioPowerUp(), SI4735::seekStation(), SI4735::sendProperty(), SI4735::sendSSBModeProperty(), SI4735::setAutomaticGainControl(), SI4735::setBandwidth(), SI4735::setFrequency(), SI4735::setRdsConfig(), SI4735::setRdsIntSource(), SI4735::setSSBBfo(), and SI4735::ssbPowerUp().

## RDS Data types

**Data Structures**

union [si47x\\_rqs\\_status](#)

*Radio Signal Quality data representation. [More...](#)*

struct [si47x\\_rqs\\_status.resp](#)

union [si47x\\_rds\\_command](#)

*Data type for RDS Status command and response information. [More...](#)*

struct [si47x\\_rds\\_command.arg](#)

union [si47x\\_rds\\_status](#)

*Response data type for current channel and reads an entry from the RDS FIFO. [More...](#)*

struct [si47x\\_rds\\_status.resp](#)

union [si47x\\_rds\\_int\\_source](#)

*FM\_RDS\_INT\_SOURCE property data type. [More...](#)*

struct [si47x\\_rds\\_int\\_source.refined](#)

union [si47x\\_rds\\_config](#)

Data type for FM\_RDS\_CONFIG Property. [More...](#)

struct [si47x\\_rds\\_config.arg](#)

union [si47x\\_rds\\_blocka](#)

Block A data type. [More...](#)

struct [si47x\\_rds\\_blocka.refined](#)

struct [si47x\\_rds\\_blocka.raw](#)

union [si47x\\_rds\\_blockb](#)

Block B data type. [More...](#)

struct [si47x\\_rds\\_blockb.group0](#)

struct [si47x\\_rds\\_blockb.group2](#)

struct [si47x\\_rds\\_blockb.refined](#)

struct [si47x\\_rds\\_blockb.raw](#)

union [si47x\\_rds\\_date\\_time](#)

struct [si47x\\_rds\\_date\\_time.refined](#)

---

## Detailed Description

---

## Data Structure Documentation

### union si47x\_rqs\_status

Radio Signal Quality data representation.

## RDS Data types

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

### Data Fields:

uint8_t	raw[8]	
struct	resp	
<a href="#">si47x_rqs_status</a>		

### struct si47x\_rqs\_status.resp

### Data Fields:

uint8_t	AFCRL: 1	Valid Channel.
uint8_t	BLENDINT: 1	
uint8_t	CTS: 1	
uint8_t	DUMMY1: 1	
uint8_t	DUMMY2: 2	
uint8_t	DUMMY3: 1	Multipath Detect High.

uint8_t	DUMMY4: 1	AFC Rail Indicator.
uint8_t	DUMMY5: 4	Soft Mute Indicator. Indicates soft mute is engaged.
uint8_t	ERR: 1	
uint8_t	FREQOFF	RESP6 - Contains the current multipath metric. (0 = no multipath; 100 = full multipath)
uint8_t	MULT	RESP5 - Contains the current SNR metric (0–127 dB).
uint8_t	MULTHINT: 1	Multipath Detect Low.
uint8_t	MULTLINT: 1	SNR Detect High.
uint8_t	PILOT: 1	Indicates amount of stereo blend in% (100 = full stereo, 0 = full mono).
uint8_t	RDSINT: 1	
uint8_t	RSQINT: 1	
uint8_t	RSSI	Indicates stereo pilot presence.
uint8_t	RSSIHINT: 1	RSSI Detect Low.
uint8_t	RSSIILINT: 1	
uint8_t	SMUTE: 1	
uint8_t	SNR	RESP4 - Contains the current receive signal strength (0–127 dB $\frac{1}{4}$ V).
uint8_t	SNRHINT: 1	SNR Detect Low.
uint8_t	SNRLINT: 1	RSSI Detect High.
uint8_t	STBLEND: 7	
uint8_t	STCINT: 1	
uint8_t	VALID: 1	Blend Detect Interrupt.

### union si47x\_rds\_command

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](https://en.wikipedia.org/wiki/Radio_Data_System)

### Data Fields:

struct <a href="#">si47x_rds_command</a>	arg	
uint8_t	raw	

### struct si47x\_rds\_command.arg

### Data Fields:

uint8_t	dummy: 5	
uint8_t	INTACK: 1	
uint8_t	MTFIFO: 1	



uint8_t	STATUSONLY: 1	
---------	---------------	--

### union si47x\_rds\_status

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

### See also

Si47XX PROGRAMMING GUIDE; AN332; pages 77 and 78

### Data Fields:

uint8_t	raw[13]	
struct	resp	
<a href="#">si47x_rds_status</a>		

### struct si47x\_rds\_status.resp

### Data Fields:

uint8_t	BLEA: 2	
uint8_t	BLEB: 2	
uint8_t	BLEC: 2	
uint8_t	BLED: 2	RESP11 - RDS Block D; LOW byte.
uint8_t	BLOCKAH	RESP3 - RDS FIFO Used; Number of groups remaining in the RDS FIFO (0 if empty).
uint8_t	BLOCKAL	RESP4 - RDS Block A; HIGH byte.
uint8_t	BLOCKBH	RESP5 - RDS Block A; LOW byte.
uint8_t	BLOCKBL	RESP6 - RDS Block B; HIGH byte.
uint8_t	BLOCKCH	RESP7 - RDS Block B; LOW byte.
uint8_t	BLOCKCL	RESP8 - RDS Block C; HIGH byte.
uint8_t	BLOCKDH	RESP9 - RDS Block C; LOW byte.
uint8_t	BLOCKDL	RESP10 - RDS Block D; HIGH byte.
uint8_t	CTS: 1	
uint8_t	DUMMY1: 1	
uint8_t	DUMMY2: 2	
uint8_t	DUMMY3: 1	RDS Sync Found; 1 = Found RDS synchronization.
uint8_t	DUMMY4: 2	RDS New Block B; 1 = Valid Block B data has been received.
uint8_t	DUMMY5: 1	RDS Sync; 1 = RDS currently synchronized.
uint8_t	DUMMY6: 5	Group Lost; 1 = One or more RDS groups discarded due to FIFO overrun.
uint8_t	ERR: 1	
uint8_t	GRPLOST: 1	
uint8_t	RDSFIFOUSED	
uint8_t	RDSINT: 1	
uint8_t	RDSNEWBLOCKA: 1	
uint8_t	RDSNEWBLOCKB: 1	RDS New Block A; 1 = Valid Block A data has been received.
uint8_t	RDSRECV: 1	
uint8_t	RDSSYNC: 1	

uint8_t	RDSSYNCFIND: 1	RDS Sync Lost; 1 = Lost RDS synchronization.
uint8_t	RDSSYNCFIND: 1	RDS Received; 1 = FIFO filled to minimum number of groups set by RDSFIFOCNT.
uint8_t	RSQINT: 1	
uint8_t	STCINT: 1	

### union si47x\_rds\_int\_source

FM\_RDS\_INT\_SOURCE property data type.

### See also

Si47XX PROGRAMMING GUIDE; AN332; page 103

also [https://en.wikipedia.org/wiki/Radio\\_Data\\_System](https://en.wikipedia.org/wiki/Radio_Data_System)

### Data Fields:

uint8_t	raw[2]	
struct	refined	
<a href="#">si47x_rds_int_source</a>		

### struct si47x\_rds\_int\_source.refined

### Data Fields:

uint8_t	DUMMY1: 1	f set, generate RDSINT when RDS gains synchronization.
uint8_t	DUMMY2: 5	If set, generate an interrupt when Block B data is found or subsequently changed.
uint8_t	DUMMY3: 5	Reserved - Always write to 0.
uint8_t	RDSNEWBLOCKA: 1	Always write to 0.
uint8_t	RDSNEWBLOCKB: 1	If set, generate an interrupt when Block A data is found or subsequently changed.
uint8_t	RDSRECV: 1	
uint8_t	RDSSYNCFIND: 1	If set, generate RDSINT when RDS loses synchronization.
uint8_t	RDSSYNCFIND: 1	If set, generate RDSINT when RDS FIFO has at least FM_RDS_INT_FIFO_COUNT entries.

### union si47x\_rds\_config

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

**Data Fields:**

struct <a href="#">si47x_rds_config</a>	arg	
uint8_t	raw[2]	

**struct si47x\_rds\_config.arg****Data Fields:**

uint8_t	BLETHA: 2	Block Error Threshold BLOCKB.
uint8_t	BLETHB: 2	Block Error Threshold BLOCKC.
uint8_t	BLETHC: 2	Block Error Threshold BLOCKD.
uint8_t	BLETHD: 2	
uint8_t	DUMMY1: 7	1 = RDS Processing Enable.
uint8_t	RDSEN: 1	

**union si47x\_rds\_blocka**

Block A data type.

**Data Fields:**

struct <a href="#">si47x_rds_blocka</a>	raw	
struct <a href="#">si47x_rds_blocka</a>	refined	

**struct si47x\_rds\_blocka.refined****Data Fields:**

uint16_t	pi	
----------	----	--

**struct si47x\_rds\_blocka.raw****Data Fields:**

uint8_t	highValue	
uint8_t	lowValue	

**union si47x\_rds\_blockb**

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](https://en.wikipedia.org/wiki/Radio_Data_System)

**Data Fields:**

struct <a href="#">si47x_rds_blockb</a>	group0	
struct <a href="#">si47x_rds_blockb</a>	group2	
struct <a href="#">si47x_rds_blockb</a>	raw	
struct <a href="#">si47x_rds_blockb</a>	refined	

**struct si47x\_rds\_blockb.group0****Data Fields:**

uint16_t	address: 2	
uint16_t	DI: 1	
uint16_t	groupType: 4	
uint16_t	MS: 1	
uint16_t	programType: 5	
uint16_t	TA: 1	
uint16_t	trafficProgramCode: 1	
uint16_t	versionCode: 1	

**struct si47x\_rds\_blockb.group2****Data Fields:**

uint16_t	address: 4	
uint16_t	groupType: 4	
uint16_t	programType: 5	
uint16_t	textABFlag: 1	
uint16_t	trafficProgramCode: 1	
uint16_t	versionCode: 1	

**struct si47x\_rds\_blockb.refined****Data Fields:**

uint16_t	content: 4	
uint16_t	groupType: 4	
uint16_t	programType: 5	
uint16_t	textABFlag: 1	
uint16_t	trafficProgramCode: 1	
uint16_t	versionCode: 1	

**struct si47x\_rds\_blockb.raw****Data Fields:**

uint8_t	highValue	
uint8_t	lowValue	

**union si47x\_rds\_date\_time**

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.

**Data Fields:**

uint8_t	raw[6]	
struct <a href="#">si47x_rds_date_time</a>	refined	

**struct si47x\_rds\_date\_time.refined**

**Data Fields:**

uint8_t	hour1: 4	
uint8_t	hour2: 1	
uint8_t	minute1: 2	
uint8_t	minute2: 4	
uint32_t	mjd: 17	
uint8_t	offset: 5	
uint8_t	offset_sense: 1	

**Receiver Status and Setup**

**Data Structures**

union [si47x\\_agc\\_status](#)  
struct [si47x\\_agc\\_status.refined](#)  
union [si47x\\_agc\\_override](#)  
struct [si47x\\_agc\\_override.arg](#)  
union [si47x\\_bandwidth\\_config](#)  
struct [si47x\\_bandwidth\\_config.param](#)  
union [si47x\\_ssb\\_mode](#)  
struct [si47x\\_ssb\\_mode.param](#)  
union [si4735\\_digital\\_output\\_format](#)  
*Digital audio output format data structure (Property 0x0102.  
DIGITAL\_OUTPUT\_FORMAT). [More...](#)*  
  
struct [si4735\\_digital\\_output\\_format.refined](#)  
struct [si4735\\_digital\\_output\\_sample\\_rate](#)  
*Digital audio output sample structure (Property 0x0104.  
DIGITAL\_OUTPUT\_SAMPLE\_RATE). [More...](#)*

---

**Detailed Description**

---

**Data Structure Documentation**

**union si47x\_agc\_status**

**Receiver Status and Setup**

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.

**Data Fields:**

uint8_t	raw[3]	
struct	refined	
<a href="#">si47x_agc_status</a>		

**struct si47x\_agc\_status.refined****Data Fields:**

uint8_t	AGCDIS: 1	
uint8_t	AGCIDX	
uint8_t	CTS: 1	
uint8_t	DUMMY: 7	
uint8_t	DUMMY1: 1	
uint8_t	DUMMY2: 2	
uint8_t	ERR: 1	
uint8_t	RDSINT: 1	
uint8_t	RSQINT: 1	
uint8_t	STCINT: 1	

**union si47x\_agc\_override**

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

**Data Fields:**

struct	arg	
<a href="#">si47x_agc_override</a>		
uint8_t	raw[2]	

**struct si47x\_agc\_override.arg****Data Fields:**

uint8_t	AGCDIS: 1	
uint8_t	AGCIDX	
uint8_t	DUMMY: 7	

**union si47x\_bandwidth\_config**

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

**Data Fields:**

struct <a href="#">si47x_bandwidth_config</a>	param	
uint8_t	raw[2]	

**struct si47x\_bandwidth\_config.param****Data Fields:**

uint8_t	AMCHFLT: 4	
uint8_t	AMPLFLT: 1	
uint8_t	DUMMY1: 4	Selects the bandwidth of the AM channel filter.
uint8_t	DUMMY2: 7	Enables the AM Power Line Noise Rejection Filter.

**union si47x\_ssb\_mode**

SSB - datatype for SSB\_MODE (property 0x0101)

**See also**

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; page 24

**Data Fields:**

struct <a href="#">si47x_ssb_mode</a>	param	
uint8_t	raw[2]	

**struct si47x\_ssb\_mode.param****Data Fields:**

uint8_t	AUDIOBW: 4	
uint8_t	AVC_DIVIDER: 4	SSB side band cutoff filter for band pass and low pass filter.
uint8_t	AVCEN: 1	set 0 for SSB mode; set 3 for SYNC mode;
uint8_t	DSP_AFCDIS: 1	Always write 0;.
uint8_t	DUMMY1: 1	SSB Soft-mute Based on RSSI or SNR.
uint8_t	SBCUTFLT: 4	0 = 1.2KHz (default); 1=2.2KHz; 2=3KHz; 3=4KHz; 4=500Hz; 5=1KHz
uint8_t	SMUTESEL: 1	SSB Automatic Volume Control (AVC) enable; 0=disable; 1=enable (default);.

**union si4735\_digital\_output\_format**

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.

**Data Fields:**

uint16_t	raw	
struct <a href="#">si4735_digital_output_format</a>	refined	

**struct si4735\_digital\_output\_format.refined****Data Fields:**

uint8_t	dummy: 8	Digital Output DCLK Edge (0 = use DCLK rising edge, 1 = use DCLK falling edge)
uint8_t	OFALL: 1	Digital Output Mode (0000=I2S, 0110 = Left-justified, 1000 = MSB at second DCLK after DFS pulse, 1100 = MSB at first DCLK after DFS pulse).
uint8_t	OMODE: 4	Digital Output Mono Mode (0=Use mono/stereo blend ).
uint8_t	OMONO: 1	Digital Output Audio Sample Precision (0=16 bits, 1=20 bits, 2=24 bits, 3=8bits).
uint8_t	OSIZE: 2	

**struct si4735\_digital\_output\_sample\_rate**

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.

**Data Fields:**

uint16_t	DOSR	
----------	------	--

**SI473X data types**

SI473X data representation.

**Data Structures**

union [si473x\\_powerup](#)

*Power Up arguments data type. [More...](#)*

struct [si473x\\_powerup.arg](#)

union [si47x\\_frequency](#)

*Represents how the frequency is stored in the si4735. [More...](#)*

struct [si47x\\_frequency.raw](#)

union [si47x\\_antenna\\_capacitor](#)

*Antenna Tuning Capacitor data type manipulation. [More...](#)*



struct [si47x\\_antenna\\_capacitor.raw](#)  
union [si47x\\_set\\_frequency](#)

*AM Tune frequency data type command (AM\_TUNE\_FREQ command) [More...](#)*

struct [si47x\\_set\\_frequency.arg](#)  
union [si47x\\_seek](#)

*Seek frequency (automatic tuning) [More...](#)*

struct [si47x\\_seek.arg](#)  
union [si47x\\_response\\_status](#)

*Response status command. [More...](#)*

struct [si47x\\_response\\_status.resp](#)  
union [si47x\\_firmware\\_information](#)

*Data representation for Firmware Information (GET\_REV) [More...](#)*

struct [si47x\\_firmware\\_information.resp](#)  
union [si47x\\_firmware\\_query\\_library](#)

*Firmware Query Library ID response. [More...](#)*

struct [si47x\\_firmware\\_query\\_library.resp](#)  
union [si47x\\_tune\\_status](#)

*Seek station status. [More...](#)*

struct [si47x\\_tune\\_status.arg](#)  
union [si47x\\_property](#)

*Data type to deal with SET\_PROPERTY command. [More...](#)*

struct [si47x\\_property.raw](#)

---

## Detailed Description

SI473X data representation.

## SI473X data types

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 approach 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

---

## Data Structure Documentation

### union si473x\_powerup

Power Up arguments data type.

#### See also

Si47XX PROGRAMMING GUIDE; AN332; pages 64 and 65

#### Data Fields:

struct <a href="#">si473x_powerup</a>	arg	
uint8_t	raw[2]	

### struct si473x\_powerup.arg

#### Data Fields:

uint8_t	CTSIEN: 1	GPO2 Output Enable (0 = GPO2 output disabled; 1 = GPO2 output enabled).
uint8_t	FUNC: 4	
uint8_t	GPO2OEN: 1	Patch Enable (0 = Boot normally; 1 = Copy non-volatile memory to RAM).
uint8_t	OPMODE	CTS Interrupt Enable (0 = CTS interrupt disabled; 1 = CTS interrupt enabled).
uint8_t	PATCH: 1	Crystal Oscillator Enable (0 = crystal oscillator disabled; 1 = Use crystal oscillator and and OPMODE=ANALOG AUDIO) .
uint8_t	XOSCEN: 1	Function (0 = FM Receive; 1–14 = Reserved; 15 = Query Library ID)

### union si47x\_frequency

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)

#### Data Fields:

struct <a href="#">si47x_frequency</a>	raw	
uint16_t	value	

### struct si47x\_frequency.raw

#### Data Fields:

uint8_t	FREQH	Tune Frequency High byte.
uint8_t	FREQL	

### union si47x\_antenna\_capacitor

Antenna Tuning Capacitor data type manipulation.

**Data Fields:**

struct <a href="#">si47x_antenna_capacitor</a>	raw	
uint16_t	value	

**struct si47x\_antenna\_capacitor.raw****Data Fields:**

uint8_t	ANTCAPH	Antenna Tuning Capacitor High byte.
uint8_t	ANTCAPL	

**union si47x\_set\_frequency**

AM Tune frequency data type command (AM\_TUNE\_FREQ command)

**See also**

Si47XX PROGRAMMING GUIDE; AN332; pages 135

**Data Fields:**

struct <a href="#">si47x_set_frequency</a>	arg	
uint8_t	raw[5]	

**struct si47x\_set\_frequency.arg****Data Fields:**

uint8_t	ANTCAPH	ARG3 - Tune Frequency Low byte.
uint8_t	ANTCAPL	ARG4 - Antenna Tuning Capacitor High byte.
uint8_t	DUMMY1: 4	Valid only for FM (Must be 0 to AM)
uint8_t	FAST: 1	
uint8_t	FREEZE: 1	ARG1 - FAST Tuning. If set, executes fast and invalidated tune. The tune status will not be accurate.
uint8_t	FREQH	SSB Upper Side Band (USB) and Lower Side Band (LSB) Selection. 10 = USB is selected; 01 = LSB is selected.
uint8_t	FREQL	ARG2 - Tune Frequency High byte.
uint8_t	USBLSB: 2	Always set 0.

**union si47x\_seek**

Seek frequency (automatic tuning)

Represents searching for a valid frequency data type.

**Data Fields:**

struct <a href="#">si47x_seek</a>	arg	
uint8_t	raw	

## struct si47x\_seek.arg

### Data Fields:

uint8_t	RESERVED1: 2	
uint8_t	RESERVED2: 4	Determines the direction of the search, either UP = 1, or DOWN = 0.
uint8_t	SEEKUP: 1	Determines whether the seek should Wrap = 1, or Halt = 0 when it hits the band limit.
uint8_t	WRAP: 1	

## union si47x\_response\_status

Response status command.

Response data from a query status command

### See also

Si47XX PROGRAMMING GUIDE; pages 73 and

### Data Fields:

uint8_t	raw[8]	
struct <a href="#">si47x_response_status</a>	resp	

## struct si47x\_response\_status.resp

### Data Fields:

uint8_t	AFCRL: 1	Valid Channel.
uint8_t	BLTF: 1	
uint8_t	CTS: 1	Error. 0 = No error 1 = Error.
uint8_t	DUMMY1: 1	Seek/Tune Complete Interrupt; 1 = Tune complete has been triggered.
uint8_t	DUMMY2: 2	Received Signal Quality Interrupt; 0 = interrupt has not been triggered.
uint8_t	DUMMY3: 5	AFC Rail Indicator.
uint8_t	ERR: 1	
uint8_t	MULT	This byte contains the SNR metric when tune is complete (dB).
uint8_t	RDSINT: 1	
uint8_t	READANTCAP	Contains the multipath metric when tune is complete.
uint8_t	READFREQH	Reports if a seek hit the band limit.
uint8_t	READFREQL	Read Frequency High byte.
uint8_t	RSQINT: 1	Radio Data System (RDS) Interrupt; 0 = interrupt has not been triggered.
uint8_t	RSSI	Read Frequency Low byte.
uint8_t	SNR	Received Signal Strength Indicator (dB $\hat{1}/4V$ )
uint8_t	STCINT: 1	
uint8_t	VALID: 1	Clear to Send.

### union si47x\_firmware\_information

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

#### Data Fields:

uint8_t	raw[9]	
struct <a href="#">si47x_firmware_in formation</a>	resp	

### struct si47x\_firmware\_information.resp

#### Data Fields:

uint8_t	CHIPREV	RESP7 - Component Minor Revision (ASCII).
uint8_t	CMPMAJOR	RESP5 - Patch ID Low byte (HEX).
uint8_t	CMPMINOR	RESP6 - Component Major Revision (ASCII).
uint8_t	CTS: 1	
uint8_t	DUMMY1: 1	
uint8_t	DUMMY2: 2	
uint8_t	ERR: 1	
uint8_t	FWMAJOR	RESP1 - Final 2 digits of Part Number (HEX).
uint8_t	FWMINOR	RESP2 - Firmware Major Revision (ASCII).
uint8_t	PATCHH	RESP3 - Firmware Minor Revision (ASCII).
uint8_t	PATCHL	RESP4 - Patch ID High byte (HEX).
uint8_t	PN	
uint8_t	RDSINT: 1	
uint8_t	RSQINT: 1	
uint8_t	STCINT: 1	

### union si47x\_firmware\_query\_library

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

#### Data Fields:

uint8_t	raw[8]	
struct <a href="#">si47x_firmware_q uery_library</a>	resp	

## struct si47x\_firmware\_query\_library.resp

### Data Fields:

uint8_t	CHIPREV	RESP5 - Reserved, various values.
uint8_t	CTS: 1	
uint8_t	DUMMY1: 1	
uint8_t	DUMMY2: 2	
uint8_t	ERR: 1	
uint8_t	FWMAJOR	RESP1 - Final 2 digits of Part Number (HEX).
uint8_t	FWMINOR	RESP2 - Firmware Major Revision (ASCII).
uint8_t	LIBRARYID	RESP6 - Chip Revision (ASCII).
uint8_t	PN	
uint8_t	RDSINT: 1	
uint8_t	RESERVED1	RESP3 - Firmware Minor Revision (ASCII).
uint8_t	RESERVED2	RESP4 - Reserved, various values.
uint8_t	RSQINT: 1	
uint8_t	STCINT: 1	

## union si47x\_tune\_status

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

### Data Fields:

struct <a href="#">si47x_tune_status</a>	arg	
uint8_t	raw	

## struct si47x\_tune\_status.arg

### Data Fields:

uint8_t	CANCEL: 1	If set, clears the seek/tune complete interrupt status indicator.
uint8_t	INTACK: 1	
uint8_t	RESERVED2: 6	If set, aborts a seek currently in progress.

## union si47x\_property

Data type to deal with SET\_PROPERTY command.

Property Data type (help to deal with SET\_PROPERTY command on si473X)

### Data Fields:

struct <a href="#">si47x_property</a>	raw	
uint16_t	value	

## struct si47x\_property.raw

### Data Fields:

uint8_t	byteHigh	
uint8_t	byteLow	

## Si4735-D60 Single Side Band (SSB) support

### Functions

void [SI4735::setSSBBfo](#) (int offset)

*Sets the SSB Beat Frequency Offset (BFO).*

void [SI4735::setSSBConfig](#) (uint8\_t AUDIOBW, uint8\_t SBCUTFLT, uint8\_t AVC\_DIVIDER, uint8\_t AVCEN, uint8\_t SMUTESEL, uint8\_t DSP\_AFCDIS)

*Sets the SSB receiver mode.*

void [SI4735::setSSBDspAfc](#) (uint8\_t DSP\_AFCDIS)

*Sets DSP AFC disable or enable.*

void [SI4735::setSSBSoftMute](#) (uint8\_t SMUTESEL)

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

void [SI4735::setSSBAutomaticVolumeControl](#) (uint8\_t AVCEN)

*Sets SSB Automatic Volume Control (AVC) for SSB mode.*

void [SI4735::setSSBAvcDivider](#) (uint8\_t AVC\_DIVIDER)

*Sets AVC Divider.*

void [SI4735::setSBBSidebandCutoffFilter](#) (uint8\_t SBCUTFLT)

*Sets SBB Sideband Cutoff Filter for band pass and low pass filters.*

void [SI4735::setSSBAudioBandwidth](#) (uint8\_t AUDIOBW)

*SSB Audio Bandwidth for SSB mode.*

void [SI4735::setSSB](#) (uint8\_t usblsb)

*Set the radio to AM function.*

void [SI4735::setSSB](#) (uint16\_t fromFreq, uint16\_t toFreq, uint16\_t initialFreq, uint16\_t step, uint8\_t usblsb)

void [SI4735::sendSSBModeProperty](#) ()

*Just send the property SSB\_MOD to the device. Internal use (private method).*

[si47x\\_firmware\\_query\\_library SI4735::queryLibraryId](#) ()

*Query the library information of the Si47XX device.*

void [SI4735::patchPowerUp](#) ()

*This method can be used to prepare the device to apply SSBRX patch.*

void [SI4735::ssbSetup](#) ()  
*Starts the Si473X device on SSB (same AM Mode).*

void [SI4735::ssbPowerUp](#) ()  
*This function can be useful for debug and test.*

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.*

bool [SI4735::downloadPatch](#) (int eeprom\_i2c\_address)  
*Transfers the content of a patch stored in a eeprom to the [SI4735](#) device.*

---

## Detailed Description

---

## Function Documentation

**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.

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;

### See also

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

### Parameters

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

### Returns

false if an error is found.



### **bool SI4735::downloadPatch (int *eeeprom\_i2c\_address*)**

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

ATTENTION: Under construction...

#### **See also**

the sketch `write_ssb_patch_eeprom.ino` (TO DO)

#### **Parameters**

<i>eeeprom_i2c_addre</i> <i>ss</i>	
---------------------------------------	--

#### **Returns**

false if an error is found.

References `SI4735::waitToSend()`.

### **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.

References `SI4735::waitToSend()`.

### **[si47x\\_firmware\\_query\\_library](#) SI4735::queryLibraryId ()**

Query the library information of the Si47XX device.

#### **SI47XX PATCH RESOURCES**

Used to confirm if the patch is compatible with the internal device library revision.

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 64 and 215-220.

struct [si47x\\_firmware\\_query\\_library](#)

#### **Returns**

a struct [si47x\\_firmware\\_query\\_library](#) (see it in [SI4735.h](#))

References SI4735::powerDown(), and SI4735::waitToSend().

#### **void SI4735::sendSSBModeProperty () [protected]**

Just send the property SSB\_MOD to the device. Internal use (private method).

References SI4735::waitToSend().

Referenced by SI4735::setSBBSidebandCutoffFilter(), SI4735::setSSBAudioBandwidth(), SI4735::setSSBAutomaticVolumeControl(), SI4735::setSSBAvcDivider(), SI4735::setSSBConfig(), SI4735::setSSBDspAfc(), and SI4735::setSSBSoftMute().

#### **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**

<i>SBCUTFLT</i>	0 or 1; see above
-----------------	-------------------

References SI4735::sendSSBModeProperty().

#### **void SI4735::setSSB (uint16\_t *fromFreq*, uint16\_t *toFreq*, uint16\_t *initialFreq*, uint16\_t *step*, uint8\_t *usbIsb*)**

Set the radio to SSB (LW/MW/SW) function.

##### **See also**

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; pages 13 and 14

##### **Parameters**

<i>fromFreq</i>	minimum frequency for the band
<i>toFreq</i>	maximum frequency for the band
<i>initialFreq</i>	initial frequency
<i>step</i>	step used to go to the next channel
<i>usbIsb</i>	SSB Upper Side Band (USB) and Lower Side Band (LSB) Selection; value 2 (binary 10) = USB; value 1 (binary 01) = LSB.

#### **void SI4735::setSSB (uint8\_t *usbIsb*)**

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**

<i>usbIsb</i>	upper or lower side band; 1 = LSB; 2 = USB
---------------	--

References SI4735::radioPowerUp().

#### **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.

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**

<i>AUDIOBW</i>	the valid values are 0, 1, 2, 3, 4 or 5; see description above
----------------	--

References SI4735::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**

<i>AVCEN</i>	0 = Disable AVC; 1 = Enable AVC (default).
--------------	--

References SI4735::sendSSBModeProperty().

#### **void SI4735::setSSBAvcDivider (uint8\_t *AVC\_DIVIDER*)**

Sets AVC Divider.

#### **See also**

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; page 24

#### **Parameters**

<i>AVC_DIVIDER</i>	SSB mode, set divider = 0; SYNC mode, set divider = 3; Other values = not allowed.
--------------------	--

References SI4735::sendSSBModeProperty().

**void SI4735::setSSBBfo (int offset)**

Sets the SSB Beat Frequency Offset (BFO).

## Si4735-D60 Single Side Band (SSB) support

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 patches 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 fix 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++. Actually, 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 I<sup>2</sup>C 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.

### See also

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; pages 5 and 23

### Parameters

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

References SI4735::waitToSend().

**void SI4735::setSSBConfig (uint8\_t *AUDIOBW*, uint8\_t *SBCUTFLT*, uint8\_t *AVC\_DIVIDER*, uint8\_t *AVCEN*, uint8\_t *SMUTESEL*, uint8\_t *DSP\_AFCDIS*)**

Sets the SSB receiver mode.

You can use this method for:

- 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 disable automatic volume control (AVC) function.

**See also**

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; page 24

**Parameters**

<i>AUDIOBW</i>	SSB Audio bandwidth; 0 = 1.2KHz (default); 1=2.2KHz; 2=3KHz; 3=4KHz; 4=500Hz; 5=1KHz.
<i>SBCUTFLT</i>	SSB side band cutoff filter for band pass and 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).
<i>AVC_DIVIDER</i>	set 0 for SSB mode; set 3 for SYNC mode.
<i>AVCEN</i>	SSB Automatic Volume Control (AVC) enable; 0=disable; 1=enable (default).
<i>SMUTESEL</i>	SSB Soft-mute Based on RSSI or SNR.
<i>DSP_AFCDIS</i>	DSP AFC Disable or enable; 0=SYNC MODE, AFC enable; 1=SSB MODE, AFC disable.

References SI4735::sendSSBModeProperty().

**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**

<i>DSP_AFCDIS</i>	0 = SYNC mode, AFC enable; 1 = SSB mode, AFC disable
-------------------	--

References SI4735::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**

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

References [SI4735::sendSSBModeProperty\(\)](#).

### **void SI4735::ssbPowerUp ()**

This function can be useful for debug and test.

References [SI4735::waitToSend\(\)](#).

### **void SI4735::ssbSetup ()**

Starts the Si473X device on SSB (same AM Mode).

Same [SI4735::setup](#) optimized to improve loading patch performance

References [SI4735::reset\(\)](#).

## **Si47XX device Mode, Band and Frequency setup**

### **Functions**

void [SI4735::setTuneFrequencyAntennaCapacitor](#) (uint16\_t capacitor)

*Only FM. Freeze Metrics During Alternate Frequency Jump.*

void [SI4735::setFrequency](#) (uint16\_t)

*Set the frequency to the current function of the Si4735 (FM, AM or SSB)*

void [SI4735::setFrequencyStep](#) (uint16\_t step)

*Sets the current step value.*

void [SI4735::frequencyUp](#) ()

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

void [SI4735::frequencyDown](#) ()

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

void [SI4735::setAM](#) ()

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

void [SI4735::setFM](#) ()

*Sets the radio to FM function.*

void [SI4735::setAM](#) (uint16\_t fromFreq, uint16\_t toFreq, uint16\_t initialFreq, uint16\_t step)

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

void [SI4735::setFM](#) (uint16\_t fromFreq, uint16\_t toFreq, uint16\_t initialFreq, uint16\_t step)

*Sets the radio to FM function.*

bool [SI4735::isCurrentTuneFM](#) ()

*Returns true if the current function is FM (FM\_TUNE\_FREQ).*

---

## Detailed Description

---

## Function Documentation

### **void SI4735::frequencyDown ()**

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

#### **See also**

[setFrequencyStep\(\)](#)

### **void SI4735::frequencyUp ()**

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

#### **See also**

[setFrequencyStep\(\)](#)

### **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).

### **void SI4735::setAM ()**

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

Define the band range you want to use for the AM mode.

#### **See also**

Si47XX PROGRAMMING GUIDE; AN332; page 129.

References SI4735::powerDown(), and SI4735::radioPowerUp().

Referenced by SI4735::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**

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

References SI4735::setAM().

**void SI4735::setFM ()**

Sets the radio to FM function.

**See also**

Si47XX PROGRAMMING GUIDE; AN332; page 64.

References SI4735::disableFmDebug(), SI4735::powerDown(), and SI4735::radioPowerUp().

Referenced by SI4735::setFM().

**void SI4735::setFM (uint16\_t *fromFreq*, uint16\_t *toFreq*, uint16\_t *initialFreq*, uint16\_t *step*)**

Sets the radio to FM function.

Defines the band range you want to use for the FM mode.

**See also**

[setFM\(\)](#)

**Parameters**

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

References SI4735::setFM().

**void SI4735::setFrequency (uint16\_t *freq*)**

Set the frequency to the current 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**

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

References SI4735::waitToSend().

**void SI4735::setFrequencyStep (uint16\_t *step*)**



Sets the current step value.

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

#### Parameters

<i>step</i>	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.
-------------	--

**void SI4735::setTuneFrequencyAntennaCapacitor (uint16\_t *capacitor*)**

Only FM. Freeze Metrics During Alternate Frequency Jump.

Selects the tuning capacitor value.

## Si47XX device Mode, Band and Frequency setup

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

<i>capacitor</i>	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).
------------------	--

## Si47XX device information and start up

### Functions

void [SI4735::getFirmware](#) (void)

*Gets firmware information.*

void [SI4735::setup](#) (uint8\_t [resetPin](#), int [interruptPin](#), uint8\_t defaultFunction, uint8\_t audioMode=[SI473X\\_ANALOG\\_AUDIO](#))

*Starts the Si473X device.*

void [SI4735::setup](#) (uint8\_t [resetPin](#), uint8\_t defaultFunction)

*Starts the Si473X device.*

---

### Detailed Description

---

## Function Documentation

### **void SI4735::getFirmware (void )**

Gets firmware information.

## **Si47XX device information and start up**

### **See also**

Si47XX PROGRAMMING GUIDE; AN332; pages 66, 131

References SI4735::waitToSend().

Referenced by SI4735::setup().

### **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**

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

References SI4735::getFirmware(), SI4735::radioPowerUp(), and SI4735::reset().

### **void SI4735::setup (uint8\_t *resetPin*, uint8\_t *defaultFunction*)**

Starts the Si473X device.

Use this setup if you are not using interrupt resource

#### **Parameters**

<i>uint8_t</i>	resetPin Digital Arduino Pin used to RESET command
<i>uint8_t</i>	defaultFunction

## **Si47XX filter setup**

### Functions

void [SI4735::setBandwidth](#) (uint8\_t AMCHFLT, uint8\_t AMPLFLT)  
*Selects the bandwidth of the channel filter for AM reception.*

---

### Detailed Description

---

### Function Documentation

void **SI4735::setBandwidth** (uint8\_t *AMCHFLT*, uint8\_t *AMPLFLT*)

Selects the bandwidth of the channel filter for AM reception.

### Si47XX filter setup

The choices are 6, 4, 3, 2, 2.5, 1.8, or 1 (kHz). The default bandwidth is 2 kHz. It works only in AM / SSB (LW/MW/SW)

#### See also

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

#### Parameters

<i>AMCHFLT</i>	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).
<i>AMPLFLT</i>	Enables the AM Power Line Noise Rejection Filter.

References SI4735::waitToSend().

### Tools method

### Functions

void [SI4735::sendProperty](#) (uint16\_t propertyValue, uint16\_t param)  
*Sends (sets) property to the SI47XX.*

---

### Detailed Description

---

## Function Documentation

**void SI4735::sendProperty (uint16\_t *propertyValue*, uint16\_t *parameter*) [protected]**

Sends (sets) property to the SI47XX.

## Tools method

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.

References SI4735::waitToSend().

## Tune

### Functions

void [SI4735::seekStation](#) (uint8\_t SEEKUP, uint8\_t WRAP)

*Look for a station (Automatic tune)*

void [SI4735::seekStationUp](#) ()

*Search for the next station.*

void [SI4735::seekStationDown](#) ()

*Search the previous station.*

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.*

void [SI4735::setSeekAmSpacing](#) (uint16\_t spacing)

*Selects frequency spacing for AM seek. Default is 10 kHz spacing.*

void [SI4735::setSeekSrnThreshold](#) (uint16\_t value)

*Sets the SNR threshold for a valid AM Seek/Tune.*

void [SI4735::setSeekRssiThreshold](#) (uint16\_t value)

*Sets the RSSI threshold for a valid AM Seek/Tune.*

---

## Detailed Description

---

## Function Documentation

**void SI4735::seekStation (uint8\_t *SEEKUP*, uint8\_t *WRAP*)**

Look for a station (Automatic tune)

## Tune

### See also

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

### Parameters

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

References SI4735::waitToSend().

**void SI4735::seekStationDown ()**

Search the previous station.

### See also

[seekStation\(uint8\\_t SEEKUP, uint8\\_t WRAP\)](#)

**void SI4735::seekStationUp ()**

Search for the next station.

### See also

[seekStation\(uint8\\_t SEEKUP, uint8\\_t WRAP\)](#)

**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

<i>uint16_t</i>	bottom - the bottom of the AM band for seek
<i>uint16_t</i>	top - the top of the AM band for seek

**void SI4735::setSeekAmSpacing (uint16\_t *spacing*)**

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

### See also

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

### Parameters

<code>uint16_t</code>	spacing - step in KHz
-----------------------	-----------------------

### **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 dB $\frac{1}{4}$ V.

### See also

Si47XX PROGRAMMING GUIDE; AN332; page 127

### **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

---

## File Documentation

### SI4735/SI4735.cpp File Reference

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

---

### SI4735/SI4735.h File Reference

```
#include <Arduino.h>
```

```
#include <Wire.h>
```

### Data Structures

union [si473x\\_powerup](#)

*Power Up arguments data type. [More...](#)*

union [si47x\\_frequency](#)

*Represents how the frequency is stored in the si4735. [More...](#)*

union [si47x\\_antenna\\_capacitor](#)

*Antenna Tuning Capacitor data type manipulation. [More...](#)*

union [si47x\\_set\\_frequency](#)

*AM Tune frequency data type command (AM\_TUNE\_FREQ command) [More...](#)*

union [si47x\\_seek](#)  
*Seek frequency (automatic tuning) [More...](#)*

union [si47x\\_response\\_status](#)  
*Response status command. [More...](#)*

union [si47x\\_firmware\\_information](#)  
*Data representation for Firmware Information (GET\_REV) [More...](#)*

union [si47x\\_firmware\\_query\\_library](#)  
*Firmware Query Library ID response. [More...](#)*

union [si47x\\_tune\\_status](#)  
*Seek station status. [More...](#)*

union [si47x\\_property](#)  
*Data type to deal with SET\_PROPERTY command. [More...](#)*

union [si47x\\_rqs\\_status](#)  
*Radio Signal Quality data representation. [More...](#)*

union [si47x\\_rds\\_command](#)  
*Data type for RDS Status command and response information. [More...](#)*

union [si47x\\_rds\\_status](#)  
*Response data type for current channel and reads an entry from the RDS FIFO. [More...](#)*

union [si47x\\_rds\\_int\\_source](#)  
*FM\_RDS\_INT\_SOURCE property data type. [More...](#)*

union [si47x\\_rds\\_config](#)  
*Data type for FM\_RDS\_CONFIG Property. [More...](#)*

union [si47x\\_rds\\_blocka](#)  
*Block A data type. [More...](#)*

union [si47x\\_rds\\_blockb](#)  
*Block B data type. [More...](#)*

union [si47x\\_rds\\_date\\_time](#)  
union [si47x\\_agc\\_status](#)  
union [si47x\\_agc\\_override](#)  
union [si47x\\_bandwidth\\_config](#)  
union [si47x\\_ssb\\_mode](#)  
union [si4735\\_digital\\_output\\_format](#)  
*Digital audio output format data structure (Property 0x0102. DIGITAL\_OUTPUT\_FORMAT). [More...](#)*

struct [si4735\\_digital\\_output\\_sample\\_rate](#)  
*Digital audio output sample structure (Property 0x0104.*  
*DIGITAL\_OUTPUT\_SAMPLE\_RATE). [More...](#)*

class [SI4735](#)  
*[SI4735](#) Class. [More...](#)*

struct [si473x\\_powerup.arg](#)  
struct [si47x\\_frequency.raw](#)  
struct [si47x\\_antenna\\_capacitor.raw](#)  
struct [si47x\\_set\\_frequency.arg](#)  
struct [si47x\\_seek.arg](#)  
struct [si47x\\_response\\_status.resp](#)  
struct [si47x\\_firmware\\_information.resp](#)  
struct [si47x\\_firmware\\_query\\_library.resp](#)  
struct [si47x\\_tune\\_status.arg](#)  
struct [si47x\\_property.raw](#)  
struct [si47x\\_rqs\\_status.resp](#)  
struct [si47x\\_rds\\_command.arg](#)  
struct [si47x\\_rds\\_status.resp](#)  
struct [si47x\\_rds\\_int\\_source.refined](#)  
struct [si47x\\_rds\\_config.arg](#)  
struct [si47x\\_rds\\_blocka.refined](#)  
struct [si47x\\_rds\\_blocka.raw](#)  
struct [si47x\\_rds\\_blockb.group0](#)  
struct [si47x\\_rds\\_blockb.group2](#)  
struct [si47x\\_rds\\_blockb.refined](#)  
struct [si47x\\_rds\\_blockb.raw](#)  
struct [si47x\\_rds\\_date\\_time.refined](#)  
struct [si47x\\_agc\\_status.refined](#)  
struct [si47x\\_agc\\_override.arg](#)  
struct [si47x\\_bandwidth\\_config.param](#)  
struct [si47x\\_ssb\\_mode.param](#)  
struct [si4735\\_digital\\_output\\_format.refined](#)

## Macros

```
#define POWER\_UP\_FM 0  
#define POWER\_UP\_AM 1  
#define POWER\_UP\_WB 3  
#define POWER\_PATCH 15  
#define SI473X\_ADDR\_SEN\_LOW 0x11  
#define SI473X\_ADDR\_SEN\_HIGH 0x63  
#define POWER\_UP 0x01  
#define GET\_REV 0x10  
#define POWER\_DOWN 0x11  
#define SET\_PROPERTY 0x12  
#define GET\_PROPERTY 0x13  
#define GET\_INT\_STATUS 0x14  
#define FM\_TUNE\_FREQ 0x20  
#define FM\_SEEK\_START 0x21  
#define FM\_TUNE\_STATUS 0x22  
#define FM\_AGC\_STATUS 0x27  
#define FM\_AGC\_OVERRIDE 0x28  
#define FM\_RSQ\_STATUS 0x23  
#define FM\_RDS\_STATUS 0x24  
#define FM\_RDS\_INT\_SOURCE 0x1500  
#define FM\_RDS\_INT\_FIFO\_COUNT 0x1501
```



```

#define FM_RDS_CONFIG 0x1502
#define FM_RDS_CONFIDENCE 0x1503
#define FM_BLEND_STEREO_THRESHOLD 0x1105
#define FM_BLEND_MONO_THRESHOLD 0x1106
#define FM_BLEND_RSSI_STEREO_THRESHOLD 0x1800
#define FM_BLEND_RSSI_MONO_THRESHOLD 0x1801
#define FM_BLEND_SNR_STEREO_THRESHOLD 0x1804
#define FM_BLEND_SNR_MONO_THRESHOLD 0x1805
#define FM_BLEND_MULTIPATH_STEREO_THRESHOLD 0x1808
#define FM_BLEND_MULTIPATH_MONO_THRESHOLD 0x1809
#define AM_TUNE_FREQ 0x40
#define AM_SEEK_START 0x41
#define AM_TUNE_STATUS 0x42
#define AM_RSQ_STATUS 0x43
#define AM_AGC_STATUS 0x47
#define AM_AGC_OVERRIDE 0x48
#define GPIO_CTL 0x80
#define GPIO_SET 0x81
#define SSB_TUNE_FREQ 0x40
#define SSB_TUNE_STATUS 0x42
#define SSB_RSQ_STATUS 0x43
#define SSB_AGC_STATUS 0x47
#define SSB_AGC_OVERRIDE 0x48
#define DIGITAL_OUTPUT_FORMAT 0x0102
#define DIGITAL_OUTPUT_SAMPLE_RATE 0x0104
#define REFCLK_FREQ 0x0201
#define REFCLK_PRESCALE 0x0202
#define AM_DEEMPHASIS 0x3100
#define AM_CHANNEL_FILTER 0x3102
#define AM_AUTOMATIC_VOLUME_CONTROL_MAX_GAIN 0x3103
#define AM_MODE_AFC_SW_PULL_IN_RANGE 0x3104
#define AM_MODE_AFC_SW_LOCK_IN_RANGE 0x3105
#define AM_RSQ_INTERRUPTS 0x3200
#define AM_RSQ_SNR_HIGH_THRESHOLD 0x3201
#define AM_RSQ_SNR_LOW_THRESHOLD 0x3202
#define AM_RSQ_RSSI_HIGH_THRESHOLD 0x3203
#define AM_RSQ_RSSI_LOW_THRESHOLD 0x3204
#define AM_SOFT_MUTE_RATE 0x3300
#define AM_SOFT_MUTE_SLOPE 0x3301
#define AM_SOFT_MUTE_MAX_ATTENUATION 0x3302
#define AM_SOFT_MUTE_SNR_THRESHOLD 0x3303
#define AM_SOFT_MUTE_RELEASE_RATE 0x3304
#define AM_SOFT_MUTE_ATTACK_RATE 0x3305
#define AM_SEEK_BAND_BOTTOM 0x3400
#define AM_SEEK_BAND_TOP 0x3401
#define AM_SEEK_FREQ_SPACING 0x3402
#define AM_SEEK_SNR_THRESHOLD 0x3403
#define AM_SEEK_RSSI_THRESHOLD 0x3404
#define AM_AGC_ATTACK_RATE 0x3702
#define AM_AGC_RELEASE_RATE 0x3703
#define AM_FRONTEND_AGC_CONTROL 0x3705
#define AM_NB_DETECT_THRESHOLD 0x3900
#define AM_NB_INTERVAL 0x3901
#define AM_NB_RATE 0x3902
#define AM_NB_IIR_FILTER 0x3903
#define AM_NB_DELAY 0x3904
#define RX_VOLUME 0x4000
#define RX_HARD_MUTE 0x4001
#define GPO_IEN 0x0001
#define SSB_BFO 0x0100

```

```
#define SSB\_MODE 0x0101
#define SSB\_RSO\_INTERRUPTS 0x3200
#define SSB\_RSO\_SNR\_HI\_THRESHOLD 0x3201
#define SSB\_RSO\_SNR\_LO\_THRESHOLD 0x3202
#define SSB\_RSO\_RSSI\_HI\_THRESHOLD 0x3203
#define SSB\_RSO\_RSSI\_LO\_THRESHOLD 0x3204
#define SSB\_SOFT\_MUTE\_RATE 0x3300
#define SSB\_SOFT\_MUTE\_MAX\_ATTENUATION 0x3302
#define SSB\_SOFT\_MUTE\_SNR\_THRESHOLD 0x3303
#define SSB\_RF\_AGC\_ATTACK\_RATE 0x3700
#define SSB\_RF\_AGC\_RELEASE\_RATE 0x3701
#define SSB\_RF\_IF\_AGC\_ATTACK\_RATE 0x3702
#define SSB\_RF\_IF\_AGC\_RELEASE\_RATE 0x3703
#define LSB\_MODE 1
#define USB\_MODE 2
#define SI473X\_ANALOG\_AUDIO 0b00000101
#define SI473X\_DIGITAL\_AUDIO1 0b00001011
#define SI473X\_DIGITAL\_AUDIO2 0b10110000
#define SI473X\_DIGITAL\_AUDIO3 0b10110101
#define FM\_CURRENT\_MODE 0
#define AM\_CURRENT\_MODE 1
#define SSB\_CURRENT\_MODE 2
#define MAX\_DELAY\_AFTER\_SET\_FREQUENCY 30
#define MIN\_DELAY\_WAIT\_SEND\_LOOP 300
```

---

## Macro Definition Documentation

```
#define AM_AGC_ATTACK_RATE 0x3702

#define AM_AGC_OVERRIDE 0x48

#define AM_AGC_RELEASE_RATE 0x3703

#define AM_AGC_STATUS 0x47

#define AM_AUTOMATIC_VOLUME_CONTROL_MAX_GAIN 0x3103

#define AM_CHANNEL_FILTER 0x3102

#define AM_CURRENT_MODE 1

#define AM_DEEMPHASIS 0x3100

#define AM_FRONTEND_AGC_CONTROL 0x3705

#define AM_MODE_AFC_SW_LOCK_IN_RANGE 0x3105

#define AM_MODE_AFC_SW_PULL_IN_RANGE 0x3104

#define AM_NB_DELAY 0x3904

#define AM_NB_DETECT_THRESHOLD 0x3900

#define AM_NB_IIR_FILTER 0x3903

#define AM_NB_INTERVAL 0x3901

#define AM_NB_RATE 0x3902

#define AM_RSQ_INTERRUPTS 0x3200

#define AM_RSQ_RSSI_HIGH_THRESHOLD 0x3203

#define AM_RSQ_RSSI_LOW_THRESHOLD 0x3204

#define AM_RSQ_SNR_HIGH_THRESHOLD 0x3201

#define AM_RSQ_SNR_LOW_THRESHOLD 0x3202

#define AM_RSQ_STATUS 0x43

#define AM_SEEK_BAND_BOTTOM 0x3400

#define AM_SEEK_BAND_TOP 0x3401

#define AM_SEEK_FREQ_SPACING 0x3402
```

```
#define AM_SEEK_RSSI_THRESHOLD 0x3404

#define AM_SEEK_SNR_THRESHOLD 0x3403

#define AM_SEEK_START 0x41

#define AM_SOFT_MUTE_ATTACK_RATE 0x3305

#define AM_SOFT_MUTE_MAX_ATTENUATION 0x3302

#define AM_SOFT_MUTE_RATE 0x3300

#define AM_SOFT_MUTE_RELEASE_RATE 0x3304

#define AM_SOFT_MUTE_SLOPE 0x3301

#define AM_SOFT_MUTE_SNR_THRESHOLD 0x3303

#define AM_TUNE_FREQ 0x40

#define AM_TUNE_STATUS 0x42

#define DIGITAL_OUTPUT_FORMAT 0x0102

#define DIGITAL_OUTPUT_SAMPLE_RATE 0x0104

#define FM_AGC_OVERRIDE 0x28

#define FM_AGC_STATUS 0x27

#define FM_BLEND_MONO_THRESHOLD 0x1106

#define FM_BLEND_MULTIPATH_MONO_THRESHOLD 0x1809

#define FM_BLEND_MULTIPATH_STEREO_THRESHOLD 0x1808

#define FM_BLEND_RSSI_MONO_THRESHOLD 0x1801

#define FM_BLEND_RSSI_STEREO_THRESHOLD 0x1800

#define FM_BLEND_SNR_MONO_THRESHOLD 0x1805

#define FM_BLEND_SNR_STEREO_THRESHOLD 0x1804

#define FM_BLEND_STEREO_THRESHOLD 0x1105

#define FM_CURRENT_MODE 0

#define FM_RDS_CONFIDENCE 0x1503

#define FM_RDS_CONFIG 0x1502
```

```

#define FM_RDS_INT_FIFO_COUNT 0x1501

#define FM_RDS_INT_SOURCE 0x1500

#define FM_RDS_STATUS 0x24

#define FM_RSQ_STATUS 0x23

#define FM_SEEK_START 0x21

#define FM_TUNE_FREQ 0x20

#define FM_TUNE_STATUS 0x22

#define GET_INT_STATUS 0x14

#define GET_PROPERTY 0x13

#define GET_REV 0x10

#define GPIO_CTL 0x80

#define GPIO_SET 0x81

#define GPO_IEN 0x0001

#define LSB_MODE 1

#define MAX_DELAY_AFTER_SET_FREQUENCY 30

#define MIN_DELAY_WAIT_SEND_LOOP 300

#define POWER_DOWN 0x11

#define POWER_PATCH 15

#define POWER_UP 0x01

#define POWER_UP_AM 1

#define POWER_UP_FM 0

```

[SI4735](#) ARDUINO LIBRARY

Const, Data type and Methods definitions

#### **See also**

Si47XX PROGRAMMING GUIDE AN332

<https://www.silabs.com/documents/public/application-notes/AN332.pdf>

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE

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

#### **Author**

PU2CLR - Ricardo Lima Caratti

By Ricardo Lima Caratti, Nov 2019

## **SI473X data types**

## **RDS Data types**

## **Receiver Status and Setup**

## **Deal with Interrupt**

## **Deal with Interrupt and I2C bus**

## **Host and slave MCU setup**

## **Si47XX device information and start up**

## **Si47XX device Mode, Band and Frequency setup**

## **Si47XX filter setup**

## **Tools method**

## **FM Mono Stereo audio setup**

## **Audio setup**

## **Frequency and Si47XX device status**

## **Tune**

**FM RDS/DBDS**

**Si4735-D60 Single Side Band (SSB) support**

```

#define POWER_UP_WB 3

#define REFCLK_FREQ 0x0201

#define REFCLK_PRESCALE 0x0202

#define RX_HARD_MUTE 0x4001

#define RX_VOLUME 0x4000

#define SET_PROPERTY 0x12

#define SI473X_ADDR_SEN_HIGH 0x63

#define SI473X_ADDR_SEN_LOW 0x11

#define SI473X_ANALOG_AUDIO 0b00000101

#define SI473X_DIGITAL_AUDIO1 0b00001011

#define SI473X_DIGITAL_AUDIO2 0b10110000

#define SI473X_DIGITAL_AUDIO3 0b10110101

#define SSB_AGC_OVERRIDE 0x48

#define SSB_AGC_STATUS 0x47

#define SSB_BFO 0x0100

#define SSB_CURRENT_MODE 2

#define SSB_MODE 0x0101

#define SSB_RF_AGC_ATTACK_RATE 0x3700

#define SSB_RF_AGC_RELEASE_RATE 0x3701

#define SSB_RF_IF_AGC_ATTACK_RATE 0x3702

#define SSB_RF_IF_AGC_RELEASE_RATE 0x3703

#define SSB_RSQ_INTERRUPTS 0x3200

#define SSB_RSQ_RSSI_HI_THRESHOLD 0x3203

#define SSB_RSQ_RSSI_LO_THRESHOLD 0x3204

#define SSB_RSQ_SNR_HI_THRESHOLD 0x3201

#define SSB_RSQ_SNR_LO_THRESHOLD 0x3202

```



```
#define SSB_RSQ_STATUS 0x43

#define SSB_SOFT_MUTE_MAX_ATTENUATION 0x3302

#define SSB_SOFT_MUTE_RATE 0x3300

#define SSB_SOFT_MUTE_SNR_THRESHOLD 0x3303

#define SSB_TUNE_FREQ 0x40

#define SSB_TUNE_STATUS 0x42

#define USB_MODE 2
```

---

## Index

INDE