Si4735 Arduino Library

AUTHOR

Version 1.1.8

02/04/2020

Table of Contents

Deprecated List

Global SI4735::analogPowerUp (void)
Consider use radioPowerUp instead

Module Index

Modules

Here is a list of all modules:

File Index

File List

Here is a list of all files with brief descriptions:

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.

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

See also

Si47XX PROGRAMMING GUIDE; AN332; page 195.

Parameters

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

```
{
    si4735_digital_output_format df;
    df.refined.OSIZE = OSIZE;
    df.refined.OMONO = OMONO;
    df.refined.OMODE = OMODE;
    df.refined.OFALL = OFALL;
    sendProperty(DIGITAL_OUTPUT_FORMAT, df.raw);
}
```

void SI4735::digitalOutputSampleRate (uint16_t *DOSR*)

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

See also

Si47XX PROGRAMMING GUIDE; AN332; page 196.

Parameters

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

```
{
    sendProperty(DIGITAL_OUTPUT_SAMPLE_RATE, DOSR);
}
```

uint8_t SI4735::getVolume ()

Gets the current volume level.

See also

setVolume()

Returns

volume (domain: 0 - 63)

```
{
    return this->volume;
}
```

void SI4735::setAudioMute (bool *off*)

Returns the current volume level.

Sets the audio on or off.

See also

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

Parameters

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

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

void SI4735::setVolume (uint8_t *volume*)

RESP8 - Returns the Chip Revision (ASCII).

Sets volume level (0 to 63)

See also

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

Parameters

uint8_t	volume (domain: 0 - 63)

```
{
    sendProperty(RX_VOLUME, volume);
    this->volume = volume;
}
```

void SI4735::volumeDown ()

Set sound volume level Down

See also

setVolume()

```
{
    if (volume > 0)
        volume--;
    setVolume(volume);
}
```

void SI4735::volumeUp ()

Set sound volume level Up

See also

setVolume()

```
{
    if (volume < 63)
        volume++;
    setVolume(volume);
}
```

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::waitInterrupr (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

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

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

Author

PU2CLR - Ricardo Lima Caratti

By Ricardo Lima Caratti, Nov 2019.

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 ()
bool getRadioDataSystemInterrupt ()

Gets Received Signal Quality Interrupt(RSQINT)

bool getTuneCompleteTriggered ()

Gets Radio Data System (RDS) Interrupt.

bool getStatusError ()

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

bool getStatusCTS ()

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

bool getACFIndicator ()

Gets the Error flag of status response.

bool getBandLimit ()

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

bool getStatusValid ()

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

uint8_t getReceivedSignalStrengthIndicator ()

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

uint8_t getStatusSNR ()

Returns integer Received Signal Strength Indicator (dBÎ¼V).

uint8_t getStatusMULT ()

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

uint8_t getAntennaTuningCapacitor ()

Returns integer containing the multipath metric when tune is complete.

void getAutomaticGainControl ()

Returns integer containing the current antenna tuning capacitor value.

void setAvcAmMaxGain (uint8_t gain)

Sets the maximum gain for automatic volume control.

void setAvcAmMaxGain ()
uint8_t getCurrentAvcAmMaxGain ()
void setAmSoftMuteMaxAttenuation (uint8_t smattn)
void setAmSoftMuteMaxAttenuation ()
void setSsbSoftMuteMaxAttenuation (uint8_t smattn)
void setSsbSoftMuteMaxAttenuation ()
bool isAgcEnabled ()
uint8_t getAgcGainIndex ()
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 ()
uint8_t getCurrentSNR ()

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

bool getCurrentRssiDetectLow ()

current SNR metric (0–127 dB).

bool getCurrentRssiDetectHigh ()

RSSI Detect Low.

bool getCurrentSnrDetectLow ()

RSSI Detect High.

bool getCurrentSnrDetectHigh ()

SNR Detect Low.

bool getCurrentValidChannel ()

SNR Detect High.

bool getCurrentAfcRailIndicator ()

Valid Channel.

bool getCurrentSoftMuteIndicator ()

AFC Rail Indicator.

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

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

uint8_t getCurrentMultipath ()
Indicates stereo pilot presence.

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

bool getCurrentMultipathDetectLow ()
Signed frequency offset (kHz).

bool getCurrentMultipathDetectHigh ()
Multipath Detect Low.

bool getCurrentBlendDetectInterrupt ()
Multipath Detect High.

uint8_t getFirmwarePN ()
Blend Detect Interrupt.

uint8_t getFirmwareFWMAJOR ()
RESP1 - Part Number (HEX)

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

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

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

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

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

uint8_t getFirmwareCHIPREV ()
RESP7 - Returns the Component Minor Revision (ASCII).

void setVolume (uint8_t volume)
: *RESP8 - Returns the Chip Revision (ASCII).*

uint8_t getVolume ()
: *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 intialFreq, 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 setFunction (uint8_t FUNC)
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 spacingfor 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 RDSSYNCFOUND, uint8_t RDSSYNCLOST, 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 RDSEN, uint8_t BLETHA, uint8_t BLETHB, uint8_t BLETHC, uint8_t BLETHD)
RESP3 - RDS FIFO Used; Number of groups remaining in the RDS FIFO (0 if empty).

uint16_t getRdsPI (void)
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 setSBBSidebandCutoffFilter (uint8_t SBCUTFLT)
Sets SBB 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 waitInterrupr (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 Informaation.

char rds_time [20]

RDS Basic tuning and switching information (Type 0 groups)

int rdsTextAdress2A

RDS date time received information

int rdsTextAdress2B

rds_buffer2A current position

int rdsTextAdress0A

rds_buffer2B current position

int16_t deviceAddress = SI473X_ADDR_SEN_LOW

rds_buffer0A current position

uint8_t lastTextFlagAB

current I2C buss address

uint8_t 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]`

Gets the Error flag of status response.

```
00961 { return currentStatus.resp.AFCRL; };
```

uint8_t SI4735::getAgcGainIndex () `[inline]`

```
00983 { return currentAgcStatus.refined.AGCIDX; }; // Returns the current AGC gain
index.
```

uint8_t SI4735::getAntennaTuningCapacitor () `[inline]`

Returns integer containing the multipath metric when tune is complete.

```
00967 { return currentStatus.resp.READANTCAP; };
```

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

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

```
00962 { return currentStatus.resp.BLTF; };
```

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

Valid Channel.

```
00997 { return currentRqsStatus.resp.AFCRL; };
```

uint8_t SI4735::getCurrentAvcAmMaxGain () `[inline]`

```
00973 {return currentAvcAmMaxGain; };
```

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

Multipath Detect High.

```
01006 { return currentRqsStatus.resp.BLENDINT; };
```

uint8_t SI4735::getCurrentMultipath () `[inline]`

Indicates stereo pilot presence.

```
01002 { return currentRqsStatus.resp.MULT; };
```

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

Multipath Detect Low.

```
01005 { return currentRqsStatus.resp.MULTHINT; };
```

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

Signed frequency offset (kHz).

```
01004 { return currentRqsStatus.resp.MULTLINT; };
```

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

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

```
01001 { return currentRqsStatus.resp.PILOT; };
```

uint8_t SI4735::getCurrentRSSI () `[inline]`

```
00990 { return currentRqsStatus.resp.RSSI; };
```

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

RSSI Detect Low.

```
00993 { return currentRqsStatus.resp.RSSIHINT; };
```

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

current SNR metric (0–127 dB).

```
00992 { return currentRqsStatus.resp.RSSIILINT; };
```

uint8_t SI4735::getCurrentSignedFrequencyOffset () `[inline]`

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

```
01003 { return currentRqsStatus.resp.FREQOFF; };
```

uint8_t SI4735::getCurrentSNR () `[inline]`

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

```
00991 { return currentRqsStatus.resp.SNR; };
```

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

SNR Detect Low.

```
00995 { return currentRqsStatus.resp.SNRHINT; };
```

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

RSSI Detect High.

```
00994 { return currentRqsStatus.resp.SNRLINT; };
```

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

AFC Rail Indicator.

```
00998 { return currentRqsStatus.resp.SMUTE; };
```

uint8_t SI4735::getCurrentStereoBlend () `[inline]`

Soft Mute Indicator. Indicates soft mute is engaged.

```
01000 { return currentRqsStatus.resp.STBLEND; };
```

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

SNR Detect High.

```
00996 { return currentRqsStatus.resp.VALID; };
```

uint8_t SI4735::getCurrentVolume () `[inline]`

```
01028 { return volume; };
```

uint8_t SI4735::getFirmwareCHIPREV () `[inline]`

RESP7 - Returns the Component Minor Revision (ASCII).

```
01021 { return firmwareInfo.resp.CHIPREV; };
```

uint8_t SI4735::getFirmwareCMPMAJOR () `[inline]`

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

```
01019 { return firmwareInfo.resp.CMPMAJOR; };
```

uint8_t SI4735::getFirmwareCMPMINOR () `[inline]`

RESP6 - Returns the Component Major Revision (ASCII).

```
{ return firmwareInfo.resp.CMPMINOR; };
```

uint8_t SI4735::getFirmwareFWMAJOR () `[inline]`

RESP1 - Part Number (HEX)

```
{ return firmwareInfo.resp.FWMAJOR; };
```

uint8_t SI4735::getFirmwareFWMINOR () `[inline]`

RESP2 - Returns the Firmware Major Revision (ASCII).

```
{ return firmwareInfo.resp.FWMINOR; };
```

uint8_t SI4735::getFirmwarePATCHH () `[inline]`

RESP3 - Returns the Firmware Minor Revision (ASCII).

```
{ return firmwareInfo.resp.PATCHH; };
```

uint8_t SI4735::getFirmwarePATCHL () `[inline]`

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

```
{ return firmwareInfo.resp.PATCHL; };
```

uint8_t SI4735::getFirmwarePN () `[inline]`

Blend Detect Interrupt.

```
{ return firmwareInfo.resp.PN;};
```

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

1 = RDS currently synchronized.

```
{ return currentRdsStatus.resp.GRPLOST; };
```

uint8_t SI4735::getNumRdsFifoUsed () `[inline]`

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

```
{ return currentRdsStatus.resp.RDSFIFOUSED; };
```

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

Gets Received Signal Quality Interrupt(RSQINT)

```
{ return currentStatus.resp.RDSINT; };
```

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

1 = Found RDS synchronization

```
{ return currentRdsStatus.resp.RDSNEWBLOCKA; };
```

Referenced by getRdsPI().

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

1 = Valid Block A data has been received.

```
{ return currentRdsStatus.resp.RDSNEWBLOCKB; };
```

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

```
{ return currentRdsStatus.resp.RDSRECV; };
```

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

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

1 = Valid Block B data has been received.

```
01086 { return currentRdsStatus.resp.RDSSYNC; };
```

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

1 = Lost RDS synchronization

```
01083 { return currentRdsStatus.resp.RDSSYNCFOUND; };
```

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

1 = FIFO filled to minimum number of groups

```
01082 { return currentRdsStatus.resp.RDSSYNCLOST; };
```

uint8_t SI4735::getReceivedSignalStrengthIndicator () `[inline]`

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

```
00964 { return currentStatus.resp.RSSI; };
```

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

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

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

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

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

```
00960 { return currentStatus.resp.CTS; };
```

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

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

```
00959 { return currentStatus.resp.ERR; };
```

uint8_t SI4735::getStatusMULT () `[inline]`

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

```
00966 { return currentStatus.resp.MULT; };
```

uint8_t SI4735::getStatusSNR () `[inline]`

Returns integer Received Signal Strength Indicator (dBÎ¼V).

```
00965 { return currentStatus.resp.SNR; };
```

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

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

```
00963 { return currentStatus.resp.VALID; };
```

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

Gets Radio Data System (RDS) Interrupt.

```
00958 { return currentStatus.resp.STCINT; };
```

uint8_t SI4735::getTuneFrequencyFast () [inline]

```
{ return currentFrequencyParams.arg.FAST; };
```

uint8_t SI4735::getTuneFrequencyFreeze () [inline]

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

```
{ return currentFrequencyParams.arg.FREEZE; };
```

bool SI4735::isAgcEnabled () [inline]

```
{ return !currentAgcStatus.refined.AGCDIS; };      // Returns true if the
AGC is enabled
```

void SI4735::setAmSoftMuteMaxAttenuation () [inline]

```
{sendProperty(AM_SOFT_MUTE_MAX_ATTENUATION, 0);};
```

void SI4735::setAmSoftMuteMaxAttenuation (uint8_t *smattn*) [inline]

```
{sendProperty(AM_SOFT_MUTE_MAX_ATTENUATION, smattn);};
```

void SI4735::setAvcAmMaxGain () [inline]

```
{ sendProperty(AM_AUTOMATIC_VOLUME_CONTROL_MAX_GAIN, ((currentAvcAmMaxGain =
48) * 340));};
```

void SI4735::setFunction (uint8_t *FUNC*)

void SI4735::setI2CFastMode (void) [inline]

Sets I2C buss to 100KHz.

```
{ Wire.setClock(400000); };
```

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

Sets I2C buss to 400KHz.

Sets the I2C bus to a given value.

ATTENTION: use this function with cation

Parameters

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

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

void SI4735::setI2CLowSpeedMode (void) [inline]

```
{ Wire.setClock(10000); };
```

void SI4735::setI2CStandardMode (void) [inline]

Sets I2C buss to 10KHz.

```
{ Wire.setClock(100000); };
```

void SI4735::setSsbSoftMuteMaxAttenuation () [inline]

```
{sendProperty(SSB_SOFT_MUTE_MAX_ATTENUATION, 0);};
```

void SI4735::setSsbSoftMuteMaxAttenuation (uint8_t *smattn*) [inline]

```
{sendProperty(SSB_SOFT_MUTE_MAX_ATTENUATION, smattn);};
```

void SI4735::setTuneFrequencyFast (uint8_t *FAST*) [inline]

Returns the FAST tuning status.

```
{ currentFrequencyParams.arg.FAST = FAST; };
```

void SI4735::setTuneFrequencyFreeze (uint8_t *FREEZE*) [inline]

Returns the FREEZE status.

```
{ currentFrequencyParams.arg.FREEZE = FREEZE; };
```

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 Informaation.

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::rdsTextAdress0A `[protected]`

rds_buffer2B current position

Referenced by getRdsText0A().

int SI4735::rdsTextAdress2A `[protected]`

RDS date time received information

Referenced by getRdsText(), and getRdsText2A().

int SI4735::rdsTextAdress2B `[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

uint8_t	resetPin MCU Mater (Arduino) reset pin

Returns

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

```
                                                    {
    int16_t error;

    pinMode(resetPin, OUTPUT);
    delay(50);
    digitalWrite(resetPin, LOW);
    delay(50);
    digitalWrite(resetPin, HIGH);

    Wire.begin();
    // check 0X11 I2C address
    Wire.beginTransmission(SI473X_ADDR_SEN_LOW);
    error = Wire.endTransmission();
    if ( error == 0 ) {
      setDeviceI2CAddress(0);
      return SI473X_ADDR_SEN_LOW;
    }

    // check 0X63 I2C address
    Wire.beginTransmission(SI473X_ADDR_SEN_HIGH);
    error = Wire.endTransmission();
    if ( error == 0 ) {
      setDeviceI2CAddress(1);
      return SI473X_ADDR_SEN_HIGH;
    }

    // Did find the device
    return 0;
}
```

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

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

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

void SI4735::setDeviceOtherI2CAddress (uint8_t *i2cAddr*)

Sets the onther I2C Bus Address (for Si470X)

You can set another I2C address different of 0x11 and 0x63

Parameters

uint8_t	i2cAddr (example 0x10)

```
                                                    {
    deviceAddress = i2cAddr;
};
```

SI4735::SI4735 ()

Construct a new SI4735::SI4735 object.

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

void SI4735::waitInterrupr (void) [protected]

Interrupt handle.

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

```
{
    while (!data_from_si4735)
        ;
}
```

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.

```
{
    Wire.beginTransmission(deviceAddress);
    Wire.write(0x12);
    Wire.write(0x00);
    Wire.write(0xFF);
    Wire.write(0x00);
    Wire.write(0x00);
    Wire.write(0x00);
    Wire.endTransmission();
    delayMicroseconds(2500);
}
```

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 dBÎ¼V.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 56.

Parameters

parameter	valid values: 0 to 127

```
{
    sendProperty(FM_BLEND_MONO_THRESHOLD, parameter);
}
```

void SI4735::setFmBlendMultiPathMonoThreshold (uint8_t *parameter*)

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

To force stereo, set to 100. To force mono, set to 0. The default is 60.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 60.

Parameters

parameter	valid values: 0 to 100

```
{
    sendProperty(FM_BLEND_MULTIPATH_MONO_THRESHOLD, parameter);
}
```

void SI4735::setFmBlendMultiPathStereoThreshold (uint8_t *parameter*)

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

To force stereo, set this to 100. To force mono, set this to 0. Default value is 20.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 60.

Parameters

parameter	valid values: 0 to 100

```
{
    sendProperty(FM_BLEND_MULTIPATH_STEREO_THRESHOLD, parameter);
}
```

void SI4735::setFmBLendRssiMonoThreshold (uint8_t *parameter*)

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

To force stereo, set this to 0. To force mono, set this to 127. Default value is 30 dBÎ¼V.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 59.

Parameters

parameter	valid values: 0 to 127

```
{
    sendProperty(FM_BLEND_RSSI_MONO_THRESHOLD, parameter);
}
```

void SI4735::setFmBlendRssiStereoThreshold (uint8_t *parameter*)

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

To force stereo, set this to 0. To force mono, set this to 127. Default value is 49 dBÎ¼V.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 59.

Parameters

parameter	valid values: 0 to 127

```
{
    sendProperty(FM_BLEND_RSSI_STEREO_THRESHOLD, parameter);
}
```

void SI4735::setFmBLendSnrMonoThreshold (uint8_t *parameter*)

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

To force stereo, set this to 0. To force mono, set this to 127. Default value is 14 dB.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 59.

Parameters

parameter	valid values: 0 to 127

```
{
    sendProperty(FM_BLEND_SNR_MONO_THRESHOLD, parameter);
}
```

void SI4735::setFmBlendSnrStereoThreshold (uint8_t *parameter*)

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

To force stereo, set this to 0. To force mono, set this to 127. Default value is 27 dB.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 59.

Parameters

parameter	valid values: 0 to 127

```
{
    sendProperty(FM_BLEND_SNR_STEREO_THRESHOLD, parameter);
}
```

void SI4735::setFmBlendStereoThreshold (uint8_t *parameter*)

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

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

See also

Si47XX PROGRAMMING GUIDE; AN332; page 90.

Parameters

parameter	valid values: 0 to 127

```
{
    sendProperty(FM_BLEND_STEREO_THRESHOLD, parameter);
}
```

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 RDSEN, uint8_t BLETHA, uint8_t BLETHB, uint8_t BLETHC, uint8_t BLETHD)

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

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

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

References SI4735::rds_buffer0A.

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

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

Clear RDS buffer 2A (text)

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

References SI4735::rds_buffer2A.

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

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

Clear RDS buffer 2B (text)

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

References SI4735::rds_buffer2B.

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

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

Process data received from group 2B.

Parameters

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

```
{
    char raw[2];
    int i, j;

    raw[1] = currentRdsStatus.resp.BLOCKDL;
    raw[0] = currentRdsStatus.resp.BLOCKDH;

    for (i = j = 0; i < 2; i++)
    {
        if (raw[i] == 0xD || raw[i] == 0xA)
        {
            c[j] = '\0';
            return;
        }
        if (raw[i] >= 32)
        {
            c[j] = raw[i];
            j++;
        }
        else
        {
            c[i] = ' ';
        }
    }
}
```

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

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

Process data received from group 2A.

Parameters

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

```
{
    char raw[4];
    int i, j;

    raw[0] = currentRdsStatus.resp.BLOCKCH;
    raw[1] = currentRdsStatus.resp.BLOCKCL;
    raw[2] = currentRdsStatus.resp.BLOCKDH;
    raw[3] = currentRdsStatus.resp.BLOCKDL;
    for (i = j = 0; i < 4; i++)
    {
        if (raw[i] == 0xD || raw[i] == 0xA)
        {
            c[j] = '\0';
            return;
        }
        if (raw[i] >= 32)
        {
            c[j] = raw[i];
            j++;
        }
        else
        {
            c[i] = ' ';
        }
    }
}
```

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

```
{
    si47x_rds_blockb blkb;

    blkb.raw.lowValue = currentRdsStatus.resp.BLOCKBL;
    blkb.raw.highValue = currentRdsStatus.resp.BLOCKBH;

    return blkb.refined.textABFlag;
}
```

uint8_t SI4735::getRdsGroupType (void)

Returns the Group Type (extracted from the Block B)

Returns

BLOCKBL

```
{
    si47x_rds_blockb blkb;

    blkb.raw.lowValue = currentRdsStatus.resp.BLOCKBL;
    blkb.raw.highValue = currentRdsStatus.resp.BLOCKBH;

    return blkb.refined.groupType;
}
```

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

```
{
    if (getRdsReceived() && getRdsNewBlockA())
    {
        return currentRdsStatus.resp.BLOCKAL;
    }
    return 0;
}
```

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

Returns

program type (an integer betwenn 0 and 31)

```
{
    si47x_rds_blockb blkb;

    blkb.raw.lowValue = currentRdsStatus.resp.BLOCKBL;
    blkb.raw.highValue = currentRdsStatus.resp.BLOCKBH;

    return blkb.refined.programType;
}
```

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)

```
{
    getRdsStatus(0, 0, 0);
}
```

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

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

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 55 and 77

Parameters

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

```
{
    si47x_rds_command rds_cmd;
    static uint16_t lastFreq;
    // checking current FUNC (Am or FM)
    if (currentTune != FM_TUNE_FREQ)
        return;

    if (lastFreq != currentWorkFrequency)
    {
        lastFreq = currentWorkFrequency;
        clearRdsBuffer2A();
        clearRdsBuffer2B();
        clearRdsBuffer0A();
    }

    waitToSend();

    rds_cmd.arg.INTACK = INTACK;
    rds_cmd.arg.MTFIFO = MTFIFO;
    rds_cmd.arg.STATUSONLY = STATUSONLY;

    Wire.beginTransmission(deviceAddress);
    Wire.write(FM_RDS_STATUS);
    Wire.write(rds_cmd.raw);
    Wire.endTransmission();

    do
    {
        waitToSend();
        // Gets response information
        Wire.requestFrom(deviceAddress, 13);
        for (uint8_t i = 0; i < 13; i++)
            currentRdsStatus.raw[i] = Wire.read();
    } while (currentRdsStatus.resp.ERR);
    delayMicroseconds(550);
}
```

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

```
{

    // Needs to get the "Text segment address code".
    // Each message should be ended by the code 0D (Hex)

    if (rdsTextAdress2A >= 16)
        rdsTextAdress2A = 0;

    getNext4Block(&rds_buffer2A[rdsTextAdress2A * 4]);

    rdsTextAdress2A += 4;

    return rds_buffer2A;
}
```

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

```
{
    si47x_rds_blockb blkB;

    // getRdsStatus();

    if (getRdsReceived())
    {
        if (getRdsGroupType() == 0)
        {
            // Process group type 0
            blkB.raw.highValue = currentRdsStatus.resp.BLOCKBH;
            blkB.raw.lowValue = currentRdsStatus.resp.BLOCKBL;

            rdsTextAdress0A = blkB.group0.address;
            if (rdsTextAdress0A >= 0 && rdsTextAdress0A < 4)
            {
                getNext2Block(&rds_buffer0A[rdsTextAdress0A * 2]);
                rds_buffer0A[8] = '\0';
                return rds_buffer0A;
            }
        }
    }
    return NULL;
}
```

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

```
{
    si47x_rds_blockb blkB;

    // getRdsStatus();
    if (getRdsReceived())
    {
        if (getRdsGroupType() == 2 /* && getRdsVersionCode() == 0 */)
        {
            // Process group 2A
            // Decode B block information
            blkB.raw.highValue = currentRdsStatus.resp.BLOCKBH;
            blkB.raw.lowValue = currentRdsStatus.resp.BLOCKBL;
            rdsTextAdress2A = blkB.group2.address;

            if (rdsTextAdress2A >= 0 && rdsTextAdress2A < 16)
            {
                getNext4Block(&rds_buffer2A[rdsTextAdress2A * 4]);
                rds_buffer2A[63] = '\0';
                return rds_buffer2A;
            }
        }
    }
    return NULL;
}
```

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

```
{
    si47x_rds_blockb blkB;

    // getRdsStatus();
    // if (getRdsReceived())
    // {
    // if (getRdsNewBlockB())
    // {
    if (getRdsGroupType() == 2 /* && getRdsVersionCode() == 1 */)
    {
        // Process group 2B
        blkB.raw.highValue = currentRdsStatus.resp.BLOCKBH;
        blkB.raw.lowValue = currentRdsStatus.resp.BLOCKBL;
        rdsTextAdress2B = blkB.group2.address;
        if (rdsTextAdress2B >= 0 && rdsTextAdress2B < 16)
        {
            getNext2Block(&rds_buffer2B[rdsTextAdress2B * 2]);
            return rds_buffer2B;
        }
    }
    //  }
    // }
    return NULL;
}
```

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.

```
{
    si47x_rds_blockb blkb;
    blkb.raw.lowValue = currentRdsStatus.resp.BLOCKBL;
    blkb.raw.highValue = currentRdsStatus.resp.BLOCKBH;

    return blkb.refined.content;
}
```

char * SI4735::getRdsTime (void)

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

Returns

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

```
{
    // Under Test and construction
    // Need to check the Group Type before.
    si47x_rds_date_time dt;

    uint16_t minute;
    uint16_t hour;

    if (getRdsGroupType() == 4)
    {
        char offset_sign;
        int offset_h;
        int offset_m;

        // uint16_t y, m, d;

        dt.raw[4] = currentRdsStatus.resp.BLOCKBL;
        dt.raw[5] = currentRdsStatus.resp.BLOCKBH;
        dt.raw[2] = currentRdsStatus.resp.BLOCKCL;
        dt.raw[3] = currentRdsStatus.resp.BLOCKCH;
        dt.raw[0] = currentRdsStatus.resp.BLOCKDL;
        dt.raw[1] = currentRdsStatus.resp.BLOCKDH;

        // Unfortunately it was necessary to wotk well on the GCC compiler on 32-bit
        // platforms. See si47x_rds_date_time (typedef union) and CGG "Crosses boundary" issue/features.
        // Now it is working on Atmega328, STM32, Arduino DUE, ESP32 and more.
        minute = (dt.refined.minute2 << 2) | dt.refined.minute1;
        hour = (dt.refined.hour2 << 4) | dt.refined.hour1;

        offset_sign = (dt.refined.offset_sense == 1) ? '+' : '-';
        offset_h = (dt.refined.offset * 30) / 60;
        offset_m = (dt.refined.offset * 30) - (offset_h * 60);
        // sprintf(rds_time, "%02u:%02u %c%02u:%02u", dt.refined.hour, dt.refined.minute, offset_sign, offset_h, offset_m);
        sprintf(rds_time, "%02u:%02u %c%02u:%02u", hour, minute, offset_sign, offset_h, offset_m);

        return rds_time;
    }

    return NULL;
}
```
References SI4735::rds_time.

uint8_t SI4735::getRdsVersionCode (void)

Gets the version code (extracted from the Block B)

Returns

0=A or 1=B

```
{
    si47x_rds_blockb blkb;

    blkb.raw.lowValue = currentRdsStatus.resp.BLOCKBL;
    blkb.raw.highValue = currentRdsStatus.resp.BLOCKBH;

    return blkb.refined.versionCode;
}
```

void SI4735::RdsInit ()

Starts the control member variables for RDS.

RDS implementation

This method is called by setRdsConfig()

See also

setRdsConfig()

```
{
    clearRdsBuffer2A();
    clearRdsBuffer2B();
    clearRdsBuffer0A();
    rdsTextAdress2A = rdsTextAdress2B = lastTextFlagAB = rdsTextAdress0A =
0;
}
```
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*)

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

Sets RDS property (FM_RDS_CONFIG)

Configures RDS settings to enable RDS processing (RDSEN) and set RDS block error thresholds.

When a RDS Group is received, all block errors must be less than or equal the associated block

error threshold for the group to be stored in the RDS FIFO.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 104

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.

Parameters

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

```
{
    si47x_property property;
    si47x_rds_config config;

    waitToSend();

    // Set property value
    property.value = FM_RDS_CONFIG;

    // Arguments
    config.arg.RDSEN = RDSEN;
    config.arg.BLETHA = BLETHA;
    config.arg.BLETHB = BLETHB;
    config.arg.BLETHC = BLETHC;
    config.arg.BLETHD = BLETHD;
    config.arg.DUMMY1 = 0;

    Wire.beginTransmission(deviceAddress);
    Wire.write(SET_PROPERTY);
    Wire.write(0x00);                  // Always 0x00 (I need to check it)
    Wire.write(property.raw.byteHigh); // Send property - High byte - most
significant first
    Wire.write(property.raw.byteLow);  // Low byte
    Wire.write(config.raw[1]);         // Send the argments. Most
significant first
    Wire.write(config.raw[0]);
    Wire.endTransmission();
    delayMicroseconds(550);

    RdsInit();
}
```

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

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

Configures interrupt related to RDS.

Use this method if want to use interrupt

See also

Si47XX PROGRAMMING GUIDE; AN332; page 103

Parameters

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

```
{
    si47x_property property;
    si47x_rds_int_source rds_int_source;

    if (currentTune != FM_TUNE_FREQ)
        return;

    rds_int_source.refined.RDSNEWBLOCKB = RDSNEWBLOCKB;
    rds_int_source.refined.RDSNEWBLOCKA = RDSNEWBLOCKA;
    rds_int_source.refined.RDSSYNCFOUND = RDSSYNCFOUND;
    rds_int_source.refined.RDSSYNCLOST = RDSSYNCLOST;
    rds_int_source.refined.RDSRECV = RDSRECV;
    rds_int_source.refined.DUMMY1 = 0;
    rds_int_source.refined.DUMMY2 = 0;

    property.value = FM_RDS_INT_SOURCE;

    waitToSend();

    Wire.beginTransmission(deviceAddress);
    Wire.write(SET_PROPERTY);
    Wire.write(0x00);                    // Always 0x00 (I need to check it)
    Wire.write(property.raw.byteHigh); // Send property - High byte - most
significant first
    Wire.write(property.raw.byteLow);  // Low byte
    Wire.write(rds_int_source.raw[1]); // Send the argments. Most
significant first
    Wire.write(rds_int_source.raw[0]);
    Wire.endTransmission();
    waitToSend();
}
```

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 ()

Returns integer containing the current antenna tuning capacitor value.

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 ()

Returns integer containing the current antenna tuning capacitor value.

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.

```
{
    uint8_t cmd;

    if (currentTune == FM_TUNE_FREQ)
    { // FM TUNE
        cmd = FM_AGC_STATUS;
    }
    else
    { // AM TUNE - SAME COMMAND used on SSB mode
        cmd = AM_AGC_STATUS;
    }

    waitToSend();

    Wire.beginTransmission(deviceAddress);
    Wire.write(cmd);
    Wire.endTransmission();

    do
    {
        waitToSend();
        Wire.requestFrom(deviceAddress, 3);
        currentAgcStatus.raw[0] = Wire.read(); // STATUS response
        currentAgcStatus.raw[1] = Wire.read(); // RESP 1
        currentAgcStatus.raw[2] = Wire.read(); // RESP 2
    } while (currentAgcStatus.refined.ERR);    // If error, try get AGC 
status again.
}
```

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()

```
{
    return currentWorkFrequency;
}
```

void SI4735::getCurrentReceivedSignalQuality (uint8_t *INTACK*)

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

This method sould be called berore call getCurrentRSSI(), getCurrentSNR() etc. Command FM_RSQ_STATUS

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 75 and 141

Parameters

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

```
{
        uint8_t arg;
        uint8_t cmd;
        int sizeResponse;

        if (currentTune == FM_TUNE_FREQ)
        { // FM TUNE
            cmd = FM_RSQ_STATUS;
            sizeResponse = 8; // Check it
        }
        else
        { // AM TUNE
            cmd = AM_RSQ_STATUS;
            sizeResponse = 6; // Check it
        }

        waitToSend();

        arg = INTACK;
        Wire.beginTransmission(deviceAddress);
        Wire.write(cmd);
        Wire.write(arg); // send B00000001
        Wire.endTransmission();

        // Check it
        // do
        //{
            waitToSend();
            Wire.requestFrom(deviceAddress, sizeResponse);
            // Gets response information
            for (uint8_t i = 0; i < sizeResponse; i++)
                currentRqsStatus.raw[i] = Wire.read();
        //} while (currentRqsStatus.resp.ERR); // Try again if error found
}
```

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

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

```
{
    getCurrentReceivedSignalQuality(0);
}
```

uint16_t SI4735::getFrequency (void)

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

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)

```
{
    si47x_frequency freq;
    getStatus(0, 1);

    freq.raw.FREQL = currentStatus.resp.READFREQL;
    freq.raw.FREQH = currentStatus.resp.READFREQH;

    currentWorkFrequency = freq.value;
    return freq.value;
}
```

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)

```
{
    getStatus(0, 1);
}
```

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

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

See also

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

Parameters

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

```
{
    si47x_tune_status status;
    uint8_t cmd = (currentTune == FM_TUNE_FREQ) ? FM_TUNE_STATUS :
AM_TUNE_STATUS;

```

```
    waitToSend();

    status.arg.INTACK = INTACK;
    status.arg.CANCEL = CANCEL;

    Wire.beginTransmission(deviceAddress);
    Wire.write(cmd);
    Wire.write(status.raw);
    Wire.endTransmission();
    // Reads the current status (including current frequency).
    do
    {
        waitToSend();
        Wire.requestFrom(deviceAddress, 8); // Check it
        // Gets response information
        for (uint8_t i = 0; i < 8; i++)
            currentStatus.raw[i] = Wire.read();
    } while (currentStatus.resp.ERR); // If error, try it again
    waitToSend();
}
```

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

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

```
{
    si47x_agc_overrride agc;

    uint8_t cmd;

    cmd = (currentTune == FM_TUNE_FREQ) ? FM_AGC_OVERRIDE : AM_AGC_OVERRIDE;

    agc.arg.AGCDIS = AGCDIS;
    agc.arg.AGCIDX = AGCIDX;

    waitToSend();

    Wire.beginTransmission(deviceAddress);
    Wire.write(cmd);
    Wire.write(agc.raw[0]);
    Wire.write(agc.raw[1]);
    Wire.endTransmission();

    waitToSend();
}
```

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

Parameters

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

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

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

```
{
    radioPowerUp();
}
```

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()

```
{
    waitToSend();
    Wire.beginTransmission(deviceAddress);
    Wire.write(POWER_DOWN);
    Wire.endTransmission();
    delayMicroseconds(2500);
}
```

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

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

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

References SI4735::waitToSend().

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

void SI4735::reset (void)

Reset the SI473X

See also

Si47XX PROGRAMMING GUIDE; AN332;

```
{
    pinMode(resetPin, OUTPUT);
    delay(10);
    digitalWrite(resetPin, LOW);
    delay(10);
    digitalWrite(resetPin, HIGH);
    delay(10);
}
```

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 chenge the defaul behavior of the Si473X. Use it before PowerUp()

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 65 and 129

Parameters

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

```
{
    powerUp.arg.CTSIEN = CTSIEN;   // 1 -> Interrupt anabled;
    powerUp.arg.GPO2OEN = GPO2OEN; // 1 -> GPO2 Output Enable;
    powerUp.arg.PATCH = PATCH;     // 0 -> Boot normally;
    powerUp.arg.XOSCEN = XOSCEN;   // 1 -> Use external crystal oscillator;
    powerUp.arg.FUNC = FUNC;       // 0 = FM Receive; 1 = AM/SSB (LW/MW/SW) Receiver.
    powerUp.arg.OPMODE = OPMODE;   // 0x5 = 00000101 = Analog audio outputs (LOUT/ROUT).

    // Set the current tuning frequancy mode 0X20 = FM and 0x40 = AM (LW/MW/SW)
    // See See Si47XX PROGRAMMING GUIDE; AN332; pages 55 and 124

    if (FUNC == 0)
    {
        currentTune = FM_TUNE_FREQ;
        currentFrequencyParams.arg.FREEZE = 1;
    }
    else
    {
        currentTune = AM_TUNE_FREQ;
        currentFrequencyParams.arg.FREEZE = 0;
    }
    currentFrequencyParams.arg.FAST = 1;
    currentFrequencyParams.arg.DUMMY1 = 0;
    currentFrequencyParams.arg.ANTCAPH = 0;
    currentFrequencyParams.arg.ANTCAPL = 1;
}
```

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

```
{
    do
    {
        delayMicroseconds(MIN_DELAY_WAIT_SEND_LOOP); // Need check the minimum value.
        Wire.requestFrom(deviceAddress, 1);
    } while (!(Wire.read() & B10000000));
}
```

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.

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 si47x_rqs_status	resp	

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 (0â€“127 dBÎ¼V).
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

Data Fields:

struct si47x_rds_command	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 si47x_rds_status	resp	

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	RDSSYNCFOUND: 1	RDS Sync Lost; 1 = Lost RDS synchronization.
uint8_t	RDSSYNCLOST: 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

Data Fields:

uint8_t	raw[2]	
struct si47x_rds_int_source	refined	

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	RDSSYNCFOUND: 1	If set, generate RDSINT when RDS loses synchronization.
uint8_t	RDSSYNCLOST: 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 si47x_rds_config	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 si47x_rds_blocka	raw	
struct si47x_rds_blocka	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

Data Fields:

struct si47x_rds_blockb	group0	
struct si47x_rds_blockb	group2	
struct	raw	

si47x_rds_blockb		
struct si47x_rds_blockb	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 si47x_rds_date_ti	refined	

me		

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_overrride
struct si47x_agc_overrride.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

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	

si47x_agc_status		

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_overrride

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 si47x_agc_overrride	arg	
uint8_t	raw[2]	

struct si47x_agc_overrride.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 si47x_bandwidth_config	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 si47x_ssb_mode	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 passand 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 si4735_digital_output_format	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 manupulation. 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.

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

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

Data Structure Documentation

union si473x_powerup

Power Up arguments data type.

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 64 and 65

Data Fields:

struct si473x_powerup	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 si47x_frequency	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 manupulation.

Data Fields:

struct si47x_antenna_capacitor	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 si47x_set_frequency	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 si47x_seek	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 si47x_response_status	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) Interrup; 0 = interrupt has not been triggered.
uint8_t	RSSI	Read Frequency Low byte.
uint8_t	SNR	Received Signal Strength Indicator (dBÎ¼V)
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 si47x_firmware_information	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 si47x_firmware_query_library	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 si47x_tune_status	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 si47x_property	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 intialFreq, uint16_t step, uint8_t usblsb)
void SI4735::sendSSBModeProperty ()
Just send the property SSB_MOD to the device. Internal use (privete 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

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

Returns

false if an error is found.

```
{
    uint8_t content;
    register int i, offset;
    // Send patch to the SI4735 device
    for (offset = 0; offset < (int) ssb_patch_content_size; offset += 8)
    {
        Wire.beginTransmission(deviceAddress);
        for (i = 0; i < 8; i++)
        {
            content = pgm_read_byte_near(ssb_patch_content + (i + offset));
            Wire.write(content);
        }
        Wire.endTransmission();

        // Testing download performance
        // approach 1 - Faster - less secure (it might crash in some architectures)
        delayMicroseconds(MIN_DELAY_WAIT_SEND_LOOP); // Need check the minimum value

        // approach 2 - More control. A little more secure than approach 1
        /*
        do
        {
            delayMicroseconds(150); // Minimum delay founded (Need check the minimum value)
            Wire.requestFrom(deviceAddress, 1);
        } while (!(Wire.read() & B10000000));
        */

        // approach 3 - same approach 2
        // waitToSend();

        // approach 4 - safer
        /*
        waitToSend();
        uint8_t cmd_status;
        Uncomment the lines below if you want to check erro.
        Wire.requestFrom(deviceAddress, 1);
        cmd_status = Wire.read();
        The SI4735 issues a status after each 8 byte transfered.
        Just the bit 7 (CTS) should be seted. if bit 6 (ERR) is seted, the system halts.
        if (cmd_status != 0x80)
           return false;
        */
    }
    delayMicroseconds(250);
    return true;
}
```

bool SI4735::downloadPatch (int *eeprom_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

eeprom_i2c_address	

Returns

false if an error is found.

```
{
    int ssb_patch_content_size;
    uint8_t cmd_status;
    int i, offset;
    uint8_t eepromPage[8];

    union {
        struct
        {
            uint8_t lowByte;
            uint8_t highByte;
        } raw;
        uint16_t value;
    } eeprom;

    // The first two bytes are the size of the patches
    // Set the position in the eeprom to read the size of the patch content
    Wire.beginTransmission(eeprom_i2c_address);
    Wire.write(0); // writes the most significant byte
    Wire.write(0); // writes the less significant byte
    Wire.endTransmission();
    Wire.requestFrom(eeprom_i2c_address, 2);
    eeprom.raw.highByte = Wire.read();
    eeprom.raw.lowByte = Wire.read();

    ssb_patch_content_size = eeprom.value;

    // the patch content starts on position 2 (the first two bytes are the size of the patch)
    for (offset = 2; offset < ssb_patch_content_size; offset += 8)
    {
        // Set the position in the eeprom to read next 8 bytes
        eeprom.value = offset;
        Wire.beginTransmission(eeprom_i2c_address);
        Wire.write(eeprom.raw.highByte); // writes the most significant byte
        Wire.write(eeprom.raw.lowByte);  // writes the less significant byte
        Wire.endTransmission();

        // Reads the next 8 bytes from eeprom
        Wire.requestFrom(eeprom_i2c_address, 8);
        for (i = 0; i < 8; i++)
            eepromPage[i] = Wire.read();

        // sends the page (8 bytes) to the SI4735
        Wire.beginTransmission(deviceAddress);
        for (i = 0; i < 8; i++)
            Wire.write(eepromPage[i]);
        Wire.endTransmission();

        waitToSend();

        Wire.requestFrom(deviceAddress, 1);
        cmd_status = Wire.read();
        // The SI4735 issues a status after each 8 byte transfered.
        // Just the bit 7 (CTS) should be seted. if bit 6 (ERR) is seted, the system halts.
        if (cmd_status != 0x80)
            return false;
    }
    delayMicroseconds(250);
    return true;
}
```

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.

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

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)

```
{
    si47x_firmware_query_library libraryID;

    powerDown(); // Is it necessary

    // delay(500);

    waitToSend();
    Wire.beginTransmission(deviceAddress);
    Wire.write(POWER_UP);
    Wire.write(0b00011111);          // Set to Read Library ID, disable
interrupt; disable GPO2OEN; boot normaly; enable External Crystal Oscillator  .
    Wire.write(SI473X_ANALOG_AUDIO); // Set to Analog Line Input.
    Wire.endTransmission();

    do
    {
        waitToSend();
        Wire.requestFrom(deviceAddress, 8);
        for (int i = 0; i < 8; i++)
            libraryID.raw[i] = Wire.read();
    } while (libraryID.resp.ERR); // If error found, try it again.

    delayMicroseconds(2500);

    return libraryID;
}
```

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

void SI4735::sendSSBModeProperty () [protected]

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

```
{
    si47x_property property;
    property.value = SSB_MODE;
    waitToSend();
    Wire.beginTransmission(deviceAddress);
    Wire.write(SET_PROPERTY);
    Wire.write(0x00);                  // Always 0x00
    Wire.write(property.raw.byteHigh); // High byte first
    Wire.write(property.raw.byteLow);  // Low byte after
    Wire.write(currentSSBMode.raw[1]); // SSB MODE params; freq. high byte
first
    Wire.write(currentSSBMode.raw[0]); // SSB MODE params; freq. low byte
after

    Wire.endTransmission();
    delayMicroseconds(550);
}
```

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

SBCUTFLT	0 or 1; see above

```
{
    currentSSBMode.param.SBCUTFLT = SBCUTFLT;
    sendSSBModeProperty();
}
```

References SI4735::sendSSBModeProperty().

void SI4735::setSSB (uint16_t *fromFreq*, uint16_t *toFreq*, uint16_t *initialFreq*, uint16_t *step*, uint8_t *usblsb*)

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

See also

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; pages 13 and 14

Parameters

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

```
{
    currentMinimumFrequency = fromFreq;
    currentMaximumFrequency = toFreq;
    currentStep = step;

    if (initialFreq < fromFreq || initialFreq > toFreq)
        initialFreq = fromFreq;

    setSSB(usblsb);

    currentWorkFrequency = initialFreq;
    setFrequency(currentWorkFrequency);
    delayMicroseconds(550);
}
```

void SI4735::setSSB (uint8_t *usblsb*)

Set the radio to AM function.

It means: LW MW and SW.

See also

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; pages 13 and 14 setAM()

void SI4735::setFrequency(uint16_t freq)

Parameters

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

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

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

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

```
{
    // Sets the audio filter property parameter
    currentSSBMode.param.AUDIOBW = AUDIOBW;
    sendSSBModeProperty();
}
```

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

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

```
{
    currentSSBMode.param.AVCEN = AVCEN;
    sendSSBModeProperty();
}
```

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

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

```
{
    currentSSBMode.param.AVC_DIVIDER = AVC_DIVIDER;
    sendSSBModeProperty();
}
```

References SI4735::sendSSBModeProperty().

void SI4735::setSSBBfo (int *offset*)

Sets the SSB Beat Frequency Offset (BFO).

Single Side Band (SSB) implementation

This implementation was tested only on Si4735-D60 device.

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

See also

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; pages 3 and 5

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

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

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

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

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

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

See also

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; pages 5 and 23

Parameters

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

```
{

    si47x_property property;
    si47x_frequency bfo_offset;

    if (currentTune == FM_TUNE_FREQ) // Only for AM/SSB mode
        return;

    waitToSend();

    property.value = SSB_BFO;
    bfo_offset.value = offset;

    Wire.beginTransmission(deviceAddress);
    Wire.write(SET_PROPERTY);
    Wire.write(0x00);                   // Always 0x00
    Wire.write(property.raw.byteHigh); // High byte first
    Wire.write(property.raw.byteLow);  // Low byte after
    Wire.write(bfo_offset.raw.FREQH);  // Offset freq. high byte first
    Wire.write(bfo_offset.raw.FREQL);  // Offset freq. low byte first

    Wire.endTransmission();
    delayMicroseconds(550);
}
```

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 disbable automatic volume control (AVC) function.

See also

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; page 24

Parameters

AUDIOBW	SSB Audio bandwidth; 0 = 1.2KHz (default); 1=2.2KHz; 2=3KHz; 3=4KHz; 4=500Hz; 5=1KHz.
SBCUTFLT	SSB side band cutoff filter for band passand low pass filter if 0, the band pass filter to cutoff both the unwanted side band and high frequency component > 2KHz of the wanted side band (default).
AVC_DIVIDER	set 0 for SSB mode; set 3 for SYNC mode.
AVCEN	SSB Automatic Volume Control (AVC) enable; 0=disable; 1=enable (default).
SMUTESEL	SSB Soft-mute Based on RSSI or SNR.
DSP_AFCDIS	DSP AFC Disable or enable; 0=SYNC MODE, AFC enable; 1=SSB MODE, AFC disable.

```
{
    if (currentTune == FM_TUNE_FREQ) // Only AM/SSB mode
        return;

    currentSSBMode.param.AUDIOBW = AUDIOBW;
    currentSSBMode.param.SBCUTFLT = SBCUTFLT;
    currentSSBMode.param.AVC_DIVIDER = AVC_DIVIDER;
    currentSSBMode.param.AVCEN = AVCEN;
    currentSSBMode.param.SMUTESEL = SMUTESEL;
    currentSSBMode.param.DUMMY1 = 0;
    currentSSBMode.param.DSP_AFCDIS = DSP_AFCDIS;

    sendSSBModeProperty();
}
```

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

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

```
{
    currentSSBMode.param.DSP_AFCDIS = DSP_AFCDIS;
    sendSSBModeProperty();
}
```

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

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

```
{
    currentSSBMode.param.SMUTESEL = SMUTESEL;
    sendSSBModeProperty();
}
```

References SI4735::sendSSBModeProperty().

void SI4735::ssbPowerUp ()

This function can be useful for debug and test.

```
{
    waitToSend();
    Wire.beginTransmission(deviceAddress);
    Wire.write(POWER_UP);
    Wire.write(0b00010001); // Set to AM/SSB, disable interrupt; disable
GPO2OEN; boot normaly; enable External Crystal Oscillator  .
    Wire.write(0b00000101); // Set to Analog Line Input.
    Wire.endTransmission();
    delayMicroseconds(2500);

    powerUp.arg.CTSIEN = 0;          // 1 -> Interrupt anabled;
    powerUp.arg.GPO2OEN = 0;         // 1 -> GPO2 Output Enable;
    powerUp.arg.PATCH = 0;           // 0 -> Boot normally;
    powerUp.arg.XOSCEN = 1;          // 1 -> Use external crystal
oscillator;
    powerUp.arg.FUNC = 1;            // 0 = FM Receive; 1 = AM/SSB
(LW/MW/SW) Receiver.
    powerUp.arg.OPMODE = 0b00000101; // 0x5 = 00000101 = Analog audio
outputs (LOUT/ROUT).
}
```

References SI4735::waitToSend().

void SI4735::ssbSetup ()

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

Same SI4735::setup optimized to improve loading patch performance

```
{
    // setPowerUp(powerUp.arg.CTSIEN, 0, 0, 1, 1, SI473X_ANALOG_AUDIO);
    reset();
    // radioPowerUp();
}
```

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 corrent 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 intialFreq, 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()

```
{

    if (currentWorkFrequency <= currentMinimumFrequency)
        currentWorkFrequency = currentMaximumFrequency;
    else
        currentWorkFrequency -= currentStep;

    setFrequency(currentWorkFrequency);
}
```

void SI4735::frequencyUp ()

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

See also

setFrequencyStep()

```
{
    if (currentWorkFrequency >= currentMaximumFrequency)
        currentWorkFrequency = currentMinimumFrequency;
    else
        currentWorkFrequency += currentStep;

    setFrequency(currentWorkFrequency);
}
```

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).

```
{
    return (currentTune == 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.

```
{
    // If you're already using AM mode, it is not necessary to call powerDown and radioPowerUp.
    // The other properties also should have the same value as the previous status.
    if ( lastMode != AM_CURRENT_MODE ) {
        powerDown();
        setPowerUp(1, 1, 0, 1, 1, SI473X_ANALOG_AUDIO);
        radioPowerUp();
        setAvcAmMaxGain(currentAvcAmMaxGain); // Set AM Automatic Volume Gain to 48
        setVolume(volume); // Set to previus configured volume
    }
    currentSsbStatus = 0;
    lastMode = AM_CURRENT_MODE;
}
```

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

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

```
{

    currentMinimumFrequency = fromFreq;
    currentMaximumFrequency = toFreq;
    currentStep = step;

    if (initialFreq < fromFreq || initialFreq > toFreq)
        initialFreq = fromFreq;

    setAM();
    currentWorkFrequency = initialFreq;
    setFrequency(currentWorkFrequency);
}
```

References SI4735::setAM().

void SI4735::setFM ()

Sets the radio to FM function.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 64.

```
{
    powerDown();
    setPowerUp(1, 1, 0, 1, 0, SI473X_ANALOG_AUDIO);
    radioPowerUp();
    setVolume(volume); // Set to previus configured volume
    currentSsbStatus = 0;
    disableFmDebug();
    lastMode = FM_CURRENT_MODE;
}
```

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

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

```
{

    currentMinimumFrequency = fromFreq;
    currentMaximumFrequency = toFreq;
    currentStep = step;

    if (initialFreq < fromFreq || initialFreq > toFreq)
        initialFreq = fromFreq;

    setFM();

    currentWorkFrequency = initialFreq;
    setFrequency(currentWorkFrequency);
}
```

References SI4735::setFM().

void SI4735::setFrequency (uint16_t *freq*)

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

You have to call setup or setPowerUp before call setFrequency.

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 70, 135

AN332 REV 0.8 UNIVERSAL PROGRAMMING GUIDE; page 13

Parameters

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

```
{
    waitToSend(); // Wait for the si473x is ready.
    currentFrequency.value = freq;
    currentFrequencyParams.arg.FREQH = currentFrequency.raw.FREQH;
    currentFrequencyParams.arg.FREQL = currentFrequency.raw.FREQL;

    if (currentSsbStatus != 0)
    {
        currentFrequencyParams.arg.DUMMY1 = 0;
        currentFrequencyParams.arg.USBLSB = currentSsbStatus; // Set to LSB or USB
        currentFrequencyParams.arg.FAST = 1;                  // Used just on AM and FM
        currentFrequencyParams.arg.FREEZE = 0;                // Used just on FM
    }

    Wire.beginTransmission(deviceAddress);
    Wire.write(currentTune);
    Wire.write(currentFrequencyParams.raw[0]); // Send a byte with FAST and FREEZE information; if not FM must be 0;
    Wire.write(currentFrequencyParams.arg.FREQH);
    Wire.write(currentFrequencyParams.arg.FREQL);
    Wire.write(currentFrequencyParams.arg.ANTCAPH);
    // If current tune is not FM sent one more byte
    if (currentTune != FM_TUNE_FREQ)
        Wire.write(currentFrequencyParams.arg.ANTCAPL);

    Wire.endTransmission();
    waitToSend();               // Wait for the si473x is ready.
    currentWorkFrequency = freq; // check it
    delay(MAX_DELAY_AFTER_SET_FREQUENCY); // For some reason I need to delay here.
}
```

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

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

```
{
    currentStep = step;
}
```

void SI4735::setTuneFrequencyAntennaCapacitor (uint16_t *capacitor*)

Only FM. Freeze Metrics During Alternate Frequency Jump.

Selects the tuning capacitor value.

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

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 71 and 136

Parameters

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

```
{
    si47x_antenna_capacitor cap;

    cap.value = capacitor;

    currentFrequencyParams.arg.DUMMY1 = 0;

    if (currentTune == FM_TUNE_FREQ)
    {
        // For FM, the capacitor value has just one byte
        currentFrequencyParams.arg.ANTCAPH = (capacitor <= 191) ? cap.raw.ANTCAPL : 0;
    }
    else
    {
        if (capacitor <= 6143)
        {
            currentFrequencyParams.arg.FREEZE = 0; // This parameter is not used for AM
            currentFrequencyParams.arg.ANTCAPH = cap.raw.ANTCAPH;
            currentFrequencyParams.arg.ANTCAPL = cap.raw.ANTCAPL;
        }
    }
}
```

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.

See also

Si47XX PROGRAMMING GUIDE; AN332; pages 66, 131

```
{
    waitToSend();

    Wire.beginTransmission(deviceAddress);
    Wire.write(GET_REV);
    Wire.endTransmission();

    do
    {
        waitToSend();
        // Request for 9 bytes response
        Wire.requestFrom(deviceAddress, 9);
        for (int i = 0; i < 9; i++)
            firmwareInfo.raw[i] = Wire.read();
    } while (firmwareInfo.resp.ERR);
}
```

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

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

```
{
    uint8_t interruptEnable = 0;
    Wire.begin();

    this->resetPin = resetPin;
    this->interruptPin = interruptPin;

    // Arduino interrupt setup (you have to know which Arduino Pins can deal with interrupt).
    if (interruptPin >= 0)
    {
        pinMode(interruptPin, INPUT);
        attachInterrupt(digitalPinToInterrupt(interruptPin), interrupt_hundler, RISING);
        interruptEnable = 1;
    }

    pinMode(resetPin, OUTPUT);
    digitalWrite(resetPin, HIGH);

    data_from_si4735 = false;

    // Set the initial SI473X behavior
    // CTSIEN   1 -> Interrupt anabled or disable;
    // GPO2OEN  1 -> GPO2 Output Enable;
    // PATCH    0 -> Boot normally;
    // XOSCEN   1 -> Use external crystal oscillator;
    // FUNC     defaultFunction = 0 = FM Receive; 1 = AM (LW/MW/SW)
Receiver.
    // OPMODE   SI473X_ANALOG_AUDIO or SI473X_DIGITAL_AUDIO.
    setPowerUp(interruptEnable, 0, 0, 1, defaultFunction, audioMode);

    reset();
    radioPowerUp();
    setVolume(30); // Default volume level.
    getFirmware();
}
```

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

uint8_t	resetPin Digital Arduino Pin used to RESET command
uint8_t	defaultFunction

```
{
    setup(resetPin, -1, defaultFunction);
    delay(250);
}
```

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.

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

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

```
{
    si47x_bandwidth_config filter;
    si47x_property property;

    if (currentTune == FM_TUNE_FREQ) // Only for AM/SSB mode
        return;

    if (AMCHFLT > 6)
        return;

    property.value = AM_CHANNEL_FILTER;

    filter.param.AMCHFLT = AMCHFLT;
    filter.param.AMPLFLT = AMPLFLT;

    waitToSend();
    this->volume = volume;
    Wire.beginTransmission(deviceAddress);
    Wire.write(SET_PROPERTY);
    Wire.write(0x00);                  // Always 0x00
    Wire.write(property.raw.byteHigh); // High byte first
    Wire.write(property.raw.byteLow);  // Low byte after
    Wire.write(filter.raw[1]);         // Raw data for AMCHFLT and
    Wire.write(filter.raw[0]);         // AMPLFLT
    Wire.endTransmission();
    waitToSend();
}
```

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.

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.

```
{
    si47x_property property;
    si47x_property param;

    property.value = propertyValue;
    param.value = parameter;
    waitToSend();
    Wire.beginTransmission(deviceAddress);
    Wire.write(SET_PROPERTY);
    Wire.write(0x00);
    Wire.write(property.raw.byteHigh); // Send property - High byte - most significant first
    Wire.write(property.raw.byteLow);  // Send property - Low byte - less significant after
    Wire.write(param.raw.byteHigh);    // Send the argments. High Byte - Most significant first
    Wire.write(param.raw.byteLow);     // Send the argments. Low Byte - Less significant after
    Wire.endTransmission();
    delayMicroseconds(550);
}
```

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 spacingfor 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)

See also

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

Parameters

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

```
{
    si47x_seek seek;

    // Check which FUNCTION (AM or FM) is working now
    uint8_t seek_start = (currentTune == FM_TUNE_FREQ) ? FM_SEEK_START : AM_SEEK_START;

    waitToSend();

    seek.arg.SEEKUP = SEEKUP;
    seek.arg.WRAP = WRAP;

    Wire.beginTransmission(deviceAddress);
    Wire.write(seek_start);
    Wire.write(seek.raw);

    if (seek_start == AM_SEEK_START)
    {
        Wire.write(0x00); // Always 0
        Wire.write(0x00); // Always 0
        Wire.write(0x00); // Tuning Capacitor: The tuning capacitor value
        Wire.write(0x00); //                   will be selected automatically.
    }

    Wire.endTransmission();
    delay(100);
}
```

References SI4735::waitToSend().

void SI4735::seekStationDown ()

Search the previous station.

See also

seekStation(uint8_t SEEKUP, uint8_t WRAP)

```
{
    seekStation(0, 1);
    delay(50);
    getFrequency();
}
```

void SI4735::seekStationUp ()

Search for the next station.

See also

seekStation(uint8_t SEEKUP, uint8_t WRAP)

```
{
    seekStation(1, 1);
    delay(50);
    getFrequency();
}
```

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

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

See also

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

Parameters

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

```
{
    sendProperty(AM_SEEK_BAND_BOTTOM, bottom);
    sendProperty(AM_SEEK_BAND_TOP, top);
}
```

void SI4735::setSeekAmSpacing (uint16_t *spacing*)

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

See also

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

Parameters

uint16_t	spacing - step in KHz

```
{
    sendProperty(AM_SEEK_FREQ_SPACING, spacing);
}
```

void SI4735::setSeekRssiThreshold (uint16_t *value*)

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

If the value is zero then RSSI threshold is not considered when doing a seek. Default value is 25 dBÎ¼V.

See also

Si47XX PROGRAMMING GUIDE; AN332; page 127

```
{
    sendProperty(AM_SEEK_RSSI_THRESHOLD, value);
}
```

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

```
{
    sendProperty(AM_SEEK_SNR_THRESHOLD, value);
}
```

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 manupulation. 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_overrride
union si47x_bandwidth_config
union si47x_ssb_mode
union si4735_digital_output_format
    Digital audio output format data structure (Property 0x0102. DIGITAL_OUTPUT_FORMAT). 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_overrride.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_RSQ_INTERRUPTS  0x3200
#define SSB_RSQ_SNR_HI_THRESHOLD  0x3201
#define SSB_RSQ_SNR_LO_THRESHOLD  0x3202
#define SSB_RSQ_RSSI_HI_THRESHOLD  0x3203
#define SSB_RSQ_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

```
#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