Si2183 Software

Si2183 MULTI-STANDARD DEMODULATOR (DVB-T/T2/C/C2/S/S2/S2X MCNS DSS ISDB-T) Getting Started With the Delivered Software

Features

- ANSI-C Si2183 driver
- Porting indications
- Test application sample code
- RF drivers for customer-selected tuner upon request
- DVB-T, DVB-T2, ISDB-T DVB-C2 scan
- DVB-C, DVB-S/S2 & DSS blindscan
- Allows I2C communication in USB mode

General Description

- The code is designed for optimum readability and minimal porting efforts.
- The software architecture allows multiple front-ends to be instantiated.
- The provided source compiles as ANSI C.
- The provided source compiles under Windows and Linux.
- The provided source compiles with gcc or VisualStudio, or any other compiler.

Disclaimer

- It is important to note that this delivery does not contain a total application, rather the Core driver that can either be used under Windows, Linux or ported onto an Embedded CPU core on a Source Decoder engine.
- This software is provided as is, free of charge.
- Skyworks Solution shall not be made liable for any use of such software, or any part of it.

Applications

- Digital terrestrial, cable & satellite STB and NIM
- Digital terrestrial, cable & satellite iDTV set
- Personal Video Recorder (DVD or HDD-based)
- Digital terrestrial & satellite PC-TV tuner peripherals
- Portable DVB-T/T2 receiver / DVD player

1 Introduction

The Si2183 is a multi-standard demodulator for DVB-T, DVB-T2, DVB-C, DVB-C Annex B, DVB-C2, DVB-S/S2, ISDB-T and DSS DTV standards. The user selects the appropriate demodulation standard via a software I2C command.

Many parts are derived from Si2183 for different applications as well as for dual/triple/quad applications. The source code is identical between single and dual/triple/quad applications.

Si2183 supports DVB-T, DVB-T2 and DVB-C2 fast channel scanning and DVB-C/SAT blindscan. The use of the DVB-C/SAT blindscan reduces channel discovery time at device setup, and runs autonomously on Si2183 (small software burden on the MPEG/host processor).

An I^2C host bus interface is used to configure and monitor all internal parameters/fields. An internal pass-through switch acts as an I^2C repeater, and can be configured to only pass selected I^2C commands to a secondary (tuner-side) I^2C bus. This feature can provide a 'quiet' I^2C bus to the RF front end.

1.1 Content of the delivery

The delivery consists in:

1. The current 'getting started' guide
2. A zipped file containing the source code and the related documentation.

Table of Contents

Table of figures

2 Getting started

2.1 Recommended approach

2.1.1 Software development on the Evaluation Board

It is recommended that the software developers get familiar with the delivered code on a Windows platform connected to a Skyworks Evaluation Board, to get a quick start.

When the code compiles and the example console application works fine, it can scan the DVB-T, DVB-T2, DVB-C, DVB-C Annex B (MCNS) DVB-C2, DVB-S, DVB-S2, ISDB-T bands and detects the available channels. The software developer can then develop his application using the wrapper API, using the test application as an example of how the front-end can be managed.

The console code illustrates the lock and scan algorithms for all standards, as well as handling a channel list.

It is therefore possible to develop the complete application using the evaluation board, in order to have the final application available when the first hardware is ready.

NB: These steps are described in paragraphs 15 First use with an EVB up to 17 Porting the EVB code to the final application .

Please contact Skyworks to get help in establishing communication with your hardware in case of difficulties.

3 Architecture Overview

The architecture of the SI2183 EVB API is illustrated in the diagram below.
This diagram represents a more complex project than generally built, since it shows access to several ranges of demodulators as well as tuner for TER and SAT.

The current document intends to guide the software developer through the delivered set of source files.

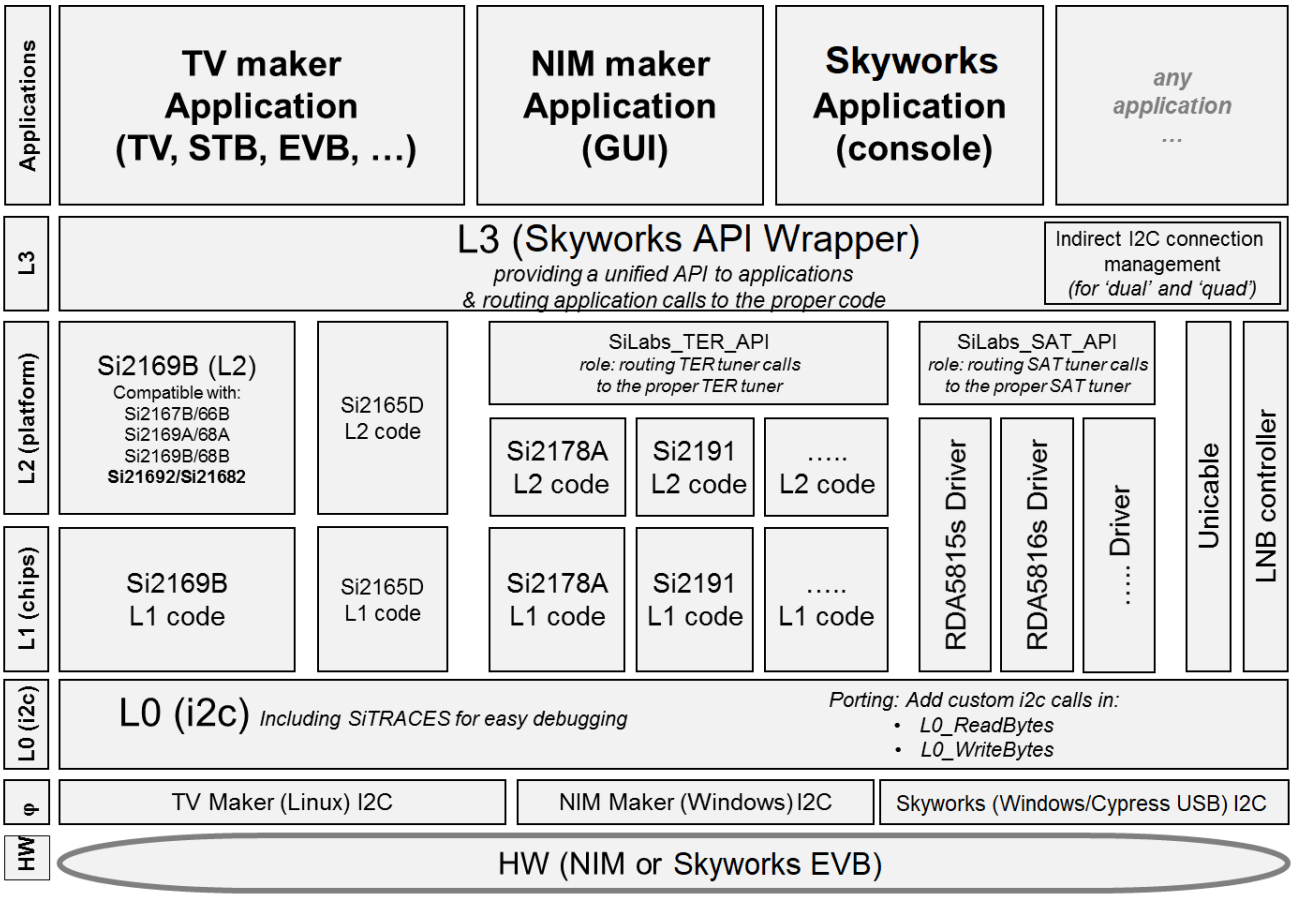


Figure 1: Software Architecture used by the Si2183 software

4 Code Layers

The various layers are split according to the following rules:

- Console application demonstrating the Front-End initialization then tuning and scanning. It should be replaced by the Customer application.
- Layer 3 is a wrapper application allowing easy configuration of the values used in the code and easy transition to another Skyworks demodulator.
- Layer 2 contains functions requiring access to more than one component (typically, scanning requires control over a tuner and the Si2183).
- Layer 1 contains all low-level command and property functions for the Si2183, more generally functions requiring access to a single component (one driver for each tuner, one driver for the Si2183).
- Layer 0 contains all generic communication functions. It will be adapted to the customer communication method.

4.1 Console application

It is a very simple application demonstrating the function calls used to initialize the Evaluation Board, tune it to a fixed frequency, then perform a DVB-T, DVB-T2, DVB-C, DVB-C2, DVB-S, DVB-S2 or ISDB-T scan in a given frequency range.

4.2 Layer 3

The L3 wrapper code is used to provide an identical API for all Skyworks demodulators and all Skyworks terrestrial tuners. This allows for easy transition between demodulators and tuners, depending on the final application.
The top-level application remains identical, although it will only allow locking on standards supported by the demodulator and the terrestrial tuner.

4.3 Layer 2

The Evaluation board driver contains all functions where access to the tuners and the Si2183 is required.
It also contains the front-end start-up and configuration functions.

4.4 Layer 1

4.4.1 Tuner Drivers

The Satellite and Terrestrial Tuner drivers are delivered in source code format.

To allow using as many tuners as possible without changing the demodulator code, a TER tuner wrapper and a SAT tuner wrapper code are used.

If other tuners than the ones delivered on the evaluation board are targeted, please require the tuner drivers you need to your Skyworks Solutions representative. As much as possible, we will add these to the corresponding tuner wrapper.

It is also possible to use the delivered source code as a framework to support the customer tuner.

The tuner driver should present the same API to Layer 2 and use the defined calls to Layer 0 to work properly.

4.4.2 Si2183 Driver

The Layer 1 Driver targets the Si2183 device and all its derivatives.

For the Si2183, the L1 code contains all command and property low-level functions. These functions perform the transition between the Si2183 API fields and the necessary i2c bytes required by the Si2183.
This layer is coded using an automated process, and therefore the coding is similar for all commands and for all properties.

4.5 Layer 0

The layer 0 code is delivered with simulator capabilities and can be adapted to the customer communication layer.

This code is documented in SW_Skywors_Broadcast_Video_Layer_0_ vx.x.x.doc. (See the ‘Doc’ folder of the Si-I2C code).

This part is probably where the biggest effort is required in the initial phase, to get proper communication using a dedicated method.

In case of difficulties getting communication with the HW, please contact Skyworks Solutions to get help from the software team.

Initially, it is possible to select between 3 communication modes during execution:

- ‘SIMU’ mode, to work without any hardware.
- ‘USB’ mode, to communicate with the Skyworks Solutions evaluation board.
- ‘CUSTOMER’ mode. This mode needs to be adapted to allow communication using the customer communication layer. Refer to SW_Skyworks_Broadcast_Video_Layer_0_vx.x.x for details on how to do the i2c porting.

The layer 0 communication layer being ‘generic’ to all Skyworks Solutions Broadcast Video products, it is not related to any particular IC and can be used for any i2c component, whether using 0, 1 or 2 bytes indexing.

5 Structures

5.1 Si2183_L2_Context

The Si2183_L2_Context structure is the top-level structure. It is defined in Si2183_L2_API.h.
To make it easily adaptable to various terrestrial and satellite tuners it uses generic names defined in the tuner wrapper files.

```
typedef struct Si2183_L2_Context {
   L1_Si2183_Context *demod;
   TER_TUNER_CONTEXT *tuner_ter;
   Si2183_INDIRECT_I2C_FUNC  f_TER_tuner_enable;
   Si2183_INDIRECT_I2C_FUNC  f_TER_tuner_disable;
   SAT_TUNER_CONTEXT *tuner_sat;
   Si2183_INDIRECT_I2C_FUNC  f_SAT_tuner_enable;
   Si2183_INDIRECT_I2C_FUNC  f_SAT_tuner_disable;
   L1_Si2183_Context  demodObj;
   TER_TUNER_CONTEXT  tuner_terObj;
   SAT_TUNER_CONTEXT  tuner_satObj;
   int                first_init_done;
   int                Si2183_init_done;
   int                TER_init_done;
   int                TER_tuner_init_done;
   unsigned char      auto_detect_TER;
   int                SAT_init_done;
   int                SAT_tuner_init_done;
   unsigned char      auto_detect_SAT;
   unsigned char      satellite_spectrum_inversion;
   unsigned char      lnb_type;
   unsigned char      unicable_spectrum_inversion;
   Si2183_WRAPPER_TUNER_TUNE_FUNC     f_tuner_tune;
   Si2183_WRAPPER_UNICABLE_TUNE_FUNC  f_Unicable_tune;
   void              *callback;
   int                standard;
   int                detected_rf;
   int                previous_standard;
   int                tuneUnitHz;
   int                rangeMin;
   int                rangeMax;
   int                seekBWHz;
   int                seekStepHz;
   int                minSRbps;
   int                maxSRbps;
   int                minRSSIdBm;
   int                maxRSSIdBm;
   int                minSNRHalfdB;
   int                maxSNRHalfdB;
   int                seekAbort;
   int                lockAbort;
   int                searchStartTime;
   int                timeoutStartTime;
   int                handshakeUsed;
   int                handshakeStart_ms;
   int                handshakePeriod_ms;
   unsigned char      handshakeOn;
} Si2183_L2_Context;
```

Members of this structure are:

- L1_Si2183_Context *demod, a pointer to the Si2183 demodulator structure. This demodulator structure is defined in Si2183_L1_Context.h. See (ref) for details about this structure.
- TER_TUNER_CONTEXT *tuner_ter, A pointer to the terrestrial tuner structure. This structure is defined in the TER tuner wrapper header file SiLabs_TER_Tuner_API.h. This file also contains the terrestrial tuner functions used by the application.

- f_TER_tuner_enable, A pointer to the wrapper function used to connect the TER tuner to the demodulator. If the TER tuner is not directly connected to the current demodulator, this function will close the required i2c pass-through.
- f_TER_tuner_disable, A pointer to the wrapper function used to disconnect the TER tuner from the demodulator. If the TER tuner is not directly connected to the current demodulator, this function will open the required i2c pass-through.
- SAT_TUNER_CONTEXT *tuner_sat, A pointer to the satellite tuner structure. This structure is defined in the SAT tuner wrapper header file SiLabs_SAT_Tuner_API.h. This file also contains the satellite tuner functions used by the application.
- L1_Si2183_Context demodObj, A Si2183 demodulator structure. During the software init phase, the demod pointer will be set to point at this structure. This will allow using pointers in the functions and avoids the need for dynamic allocation of the Si2183 context in the lower layers.
- TER_TUNER_CONTEXT tuner_terObj, A terrestrial tuner structure matching the selected terrestrial tuner. During the software init phase, the tuner_ter pointer will be set to point at this structure. This will allow using pointers in the functions and avoids the need for dynamic allocation of the TER_TUNER_CONTEXT in the lower layers.
- SAT_TUNER_CONTEXT tuner_satObj, A satellite tuner structure matching the selected satellite tuner. During the software init phase, the tuner_sat pointer will be set to point at this structure. This will allow using pointers in the functions and avoids the need for dynamic allocation of the SAT_TUNER_CONTEXT in the lower layers.
- first_init_done, A flag indicating if the front-end initialization has already been done. This is different from the next flag as if the front-end is first init for ATV reception the Si2183 will not be init on this very first call.
- Si2183_init_done, A flag indicating if the Si2183 hardware initialization has already been done.
- TER_init_done, A flag indicating if the terrestrial demodulator hardware initialization has already been done.
- TER_tuner_init_done, A flag indicating if the terrestrial tuner hardware initialization has already been done.
- auto_detect_TER, A flag indicating if the TER auto-detection (between T and T2) is active or not.
- SAT_init_done, A flag indicating if the satellite demodulator hardware initialization has already been done.
- SAT_tuner_init_done, A flag indicating if the satellite tuner hardware initialization has already been done.
- auto_detect_SAT, A flag indicating if the SAT auto-detection (between S and S2) is active or not.
- satellite_spectrum_inversion, A flag indicating if the SAT spectrum is inverted or not (this can be the case if I/Q signals are swapped in HW).
- lnb_type, A flag indicating if the SAT reception is done using a normal LNB or a Unicable LNB (see declaration of UNICABLE_LNB_TYPE_NORMAL/ UNICABLE_LNB_TYPE_UNICABLE).
- unicable_spectrum_inversion, A flag indicating if the Unicable LNB adds a spectrum inversion (this is generally true).
- f_tuner_tune, A pointer to the wrapper 'normal' SAT tune function. This function will be called when tuning in SAT in non-Unicable mode.
- f_Unicable_tune, A pointer to the wrapper 'Unicable' SAT tune function. This function will be called when tuning in SAT in Unicable mode.
- void *callback, a pointer to the L3 frontend context. It's used when calling L3 Wrapper functions.
- standard, The current standard, at front-end level. NB: This one can correspond to an ATV standard, while this is not possible in the lower-level standard in the L1_Si2183_Context.
- previous_standard, The value of the standard used during the last successful initialization.
- tuneUnitHz A value used to store the tune unit (1 Hz in TER, 1000 Hz in SAT).
- detected_rf Used to store the frequency of the signal detected during a seek.
- previous_standard Used in Si2183_L2_switch_to_standard to perform the minimum operations to switch from one standard to another.
- tuneUnitHz Used to store the tune unit (Hz for TER, kHz for SAT).
- rangeMin Min Seek RF limit
- rangeMax Max Seek RF limit
- seekBWHz Seek BW
- seekStepHz Seek Step (Generally equal to seekBWHz)
- minSRbps Min Seek symbol rate (in baud)
- maxSRbps Min Seek symbol rate (in baud)
- minRSSIdBm Unused. Provision for limiting the lock to a given RSSI range.
- maxRSSIdBm Unused. Provision for limiting the lock to a given RSSI range.
- minSNRHalfdB Unused. Provision for limiting the lock to a given SNR range.
- maxSNRHalfdB Unused. Provision for limiting the lock to a given SNR range.

- seekAbort A flag used to abort a seek, when set by the above layers.
- lockAbort A flag used to abort a lock, when set by the above layers.
- searchStartTime This field is used to trace the time spent trying to detect a channel. It is set differently from seekStartTime when returning from a handshake
- timeoutStartTime This field is used in blind mode to make sure the FW is correctly responding. timeoutStartTime is used in non-blind mode to manage the timeout when trying to lock on a channel.
- handshakeUsed A flag indicating if the handshake feature is used or not.
- handshakeStart_ms; Used to store the start time of the handshake
- handshakePeriod_ms; Used to store the duration of the handshake period
- handshakeOn; A flag indicating if the application is within a handshake period.
- seekRunning; A flag indicating is a seek is running.

5.2 L1_Si2183_Context

The L1_Si2183_Context structure contains all the variables used by the Si2183 L1 code.

```
typedef struct L1_Si2183_Context {
  L0_Context                  *i2c;
  L0_Context                   i2cObj;
  Si2183_CmdObj               *cmd;
  Si2183_CmdObj                cmdObj;
  Si2183_CmdReplyObj          *rsp;
  Si2183_CmdReplyObj           rspObj;
  Si2183_PropObj              *prop;
  Si2183_PropObj               propObj;
  Si2183_COMMON_REPLY_struct *status;
  Si2183_COMMON_REPLY_struct  statusObj;
           int standard;
           int media;
           int propertyWriteMode; /* Selection of 'DOWNLOAD_ALWAYS' or 'DOWNLOAD_ON_CHANGE' for
properties. Use 'DOWNLOAD_ALWAYS' only for debugging purposes */
  unsigned int  tuner_ter_clock_source; /* TER clock source configuration            */
  unsigned int  tuner_ter_clock_freq;   /* TER clock frequency configuration         */
  unsigned int  tuner_ter_clock_input;  /* TER clock input pin configuration         */
  unsigned int  tuner_ter_clock_control /* TER clock management configuration        */
  unsigned int  tuner_ter_agc_control;  /* TER AGC output and polarity configuration */
           int  TER_tuner_agc_input;    /* TER AGC input on tuner side configuration */
           int  TER_tuner_if_output;    /* TER IF output on tuner side configuration */
  unsigned int  fef_selection;          /* Preferred FEF mode. May not be possible   */
  unsigned int  fef_mode;               /* Final FEF mode, taking into account HW limitations */
  unsigned int  fef_pin;                /* FEF pin used for TER tuner AGC freez      */
  unsigned int  fef_level;              /* GPIO state on FEF_pin when used (during FEF periods) */
  unsigned int  tuner_sat_clock_source; /* SAT clock source configuration            */
  unsigned int  tuner_sat_clock_freq;   /* SAT clock frequency configuration         */
  unsigned int  tuner_sat_clock_input;   /* SAT clock input pin configuration         */
  unsigned int  tuner_sat_clock_control; /* SAT clock management configuration        */
  unsigned int  spi_download;     /* Flag indicating if SPI is used  */
  unsigned int  spi_send_option;  /* SPI download option (depends on SPI implementation) */
  unsigned int  spi_clk_pin;      /* Pin used for SPI clock */
  unsigned int  spi_clk_pola;     /* SPI clock polarity     */
  unsigned int  spi_data_pin;     /* Pin used for SPI data  */
  unsigned int  spi_data_order;   /* SPI data polarity      */
} L1_Si2183_Context;
```

5.3 L0_Context

The L0_Context structure contains all the variables used by the Layer 0 code to manage i2c communication. It is defined in Silabs_L0_API.h.
See the L0 documentation for implementation details.

```
typedef struct L0_Context
{
    unsigned char   address;
    int             indexSize;
    CONNECTION_TYPE connectionType;
    int             trackWrite;
    int             trackRead;
    int             mustReadWithoutStop;
    unsigned char   tag_index;
    char            tag[SILABS_TAG_SIZE];
} L0_Context;
```

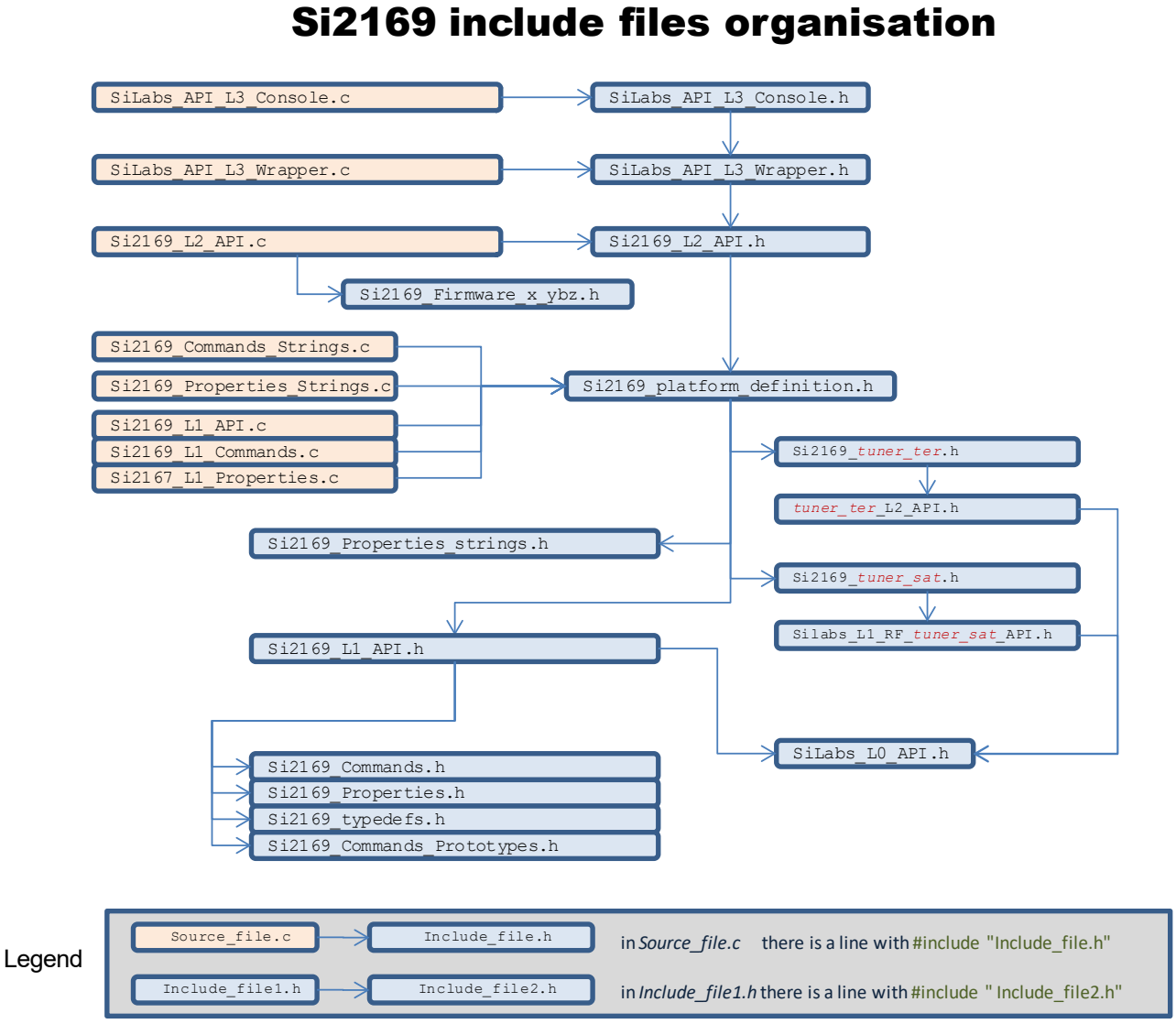
Members of this structure are:

- unsigned char address; the i2c address of the chip (0xc8 for the Si2183 base address)
- int indexSize; the number of bytes used to index the i2c bytes. 2 for the Si2183, usually 1 for most tuners.
- CONNECTION_TYPE connectionType; the value of the current connection mode. Set to 'USB' for Skyworks EVBs, it can also be 'SIMU' for software debug purposes or 'CUSTOMER' for customer hardware.
- int trackWrite; a flag to control whether i2c traces are generated or not during write operations.
- int trackRead; a flag to control whether i2c traces are generated or not during read operations.
- int mustReadWithoutStop; a flag indicating whether a read operation can include a STOP condition (the normal i2c behavior) or not. (So far, only the RDA5812 satellite tuner does not allow the normal mode and this flag must be set as 1).
- unsigned char tag_index; the value of the frontend index displayed in the tag text.
- Char tag[SILABS_TAG_SIZE]; the text used for tagging

6 Include scheme

The top-level file to include in an application is **SiLabs_API_L3_wrapper.h**.

The following diagram illustrates the Si2183 include scheme.



7 Access to Skyworks source code

To get access to Skyworks source code customers need to have a valid NDA in place.

- Please contact your Skyworks sales representative to set the NDA in place.

Request delivery of the source code access links once the NDA is signed by both parties.

- The list of parts used in your design is useful to get a full list of source code links.

Then, access to Skyworks FTP folders containing the necessary source code can be requested via your local support representative. Once granted, access to these folders is no-expiring, so customers can check on FTP the availability of newer source code or documentation.

Upon request, customer email addresses can be added to dedicated distribution lists for source code update notification.

- Send a request to get your email address added/removed from any notification list.

8 Documentation organization

The current document provides Si2183 specific information on the L1 and L2 layers.
Additional Information is available for other layers:

Refer to the L0 documentation for details of i2c porting and traces:

- **SW_Skyworks_Broadcast_Video_Layer_0_Vx.y.z.pdf** (in the Si_I2C folder)

Refer to the L3 documentation for details on the general application:

- **USING_SKYWORKS_SUPERSET.pdf**
- **Skyworks_API_Wrapper_SW_Release_Note.pdf**

9 Files organization

9.1 Si2183 console application

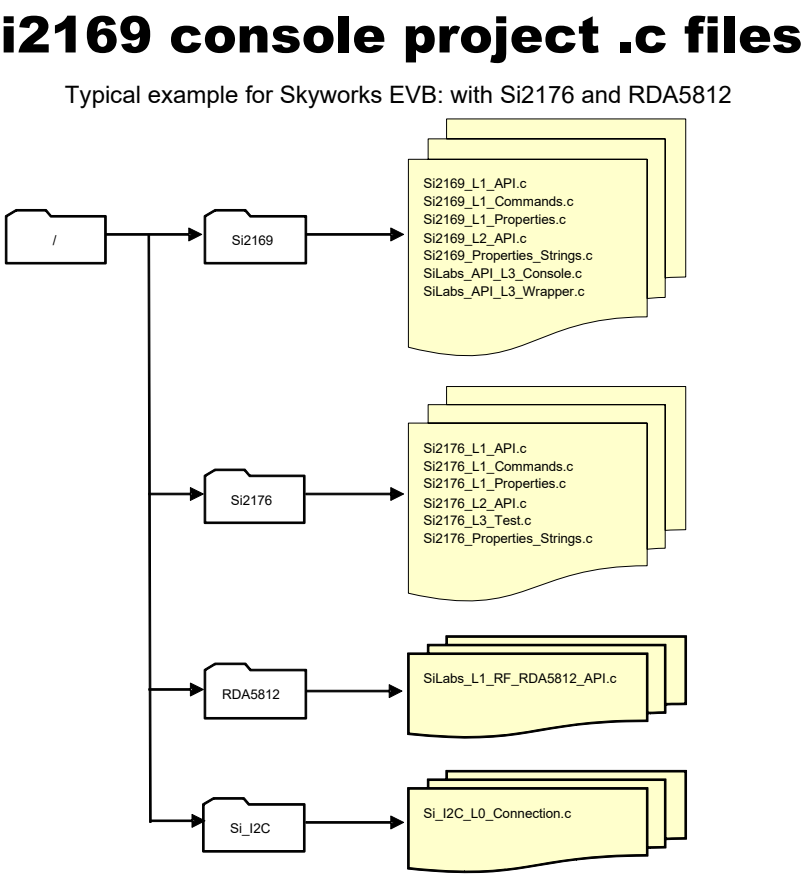


Figure 2: Files Organization

9.1.1 Compilation flags for the typical console application

To compile the typical console application for the Si2183/Si2176/RDA5812 EVB, the following compilation flags must be defined at project level:

- TER_TUNER_SILABS
  Flag used to use the TER tuner wrapper API. This is the only possibility provided with the Si2183, as it allows adding new tuners in the TER tuner wrapper and directly using them in a Si2183 application without changes to the Si2183 code.
- TER_TUNER_Si2178
  Flag used to select the Si2178 as the terrestrial tuner in the TER tuner wrapper code.
- SAT_TUNER_SILABS
  Flag used to use the SAT tuner wrapper API. This is the only possibility provided with the Si2183, as it allows adding new tuners in the SAT tuner wrapper and directly using them in a Si2183 application without changes to the Si2183 code.
- SAT_TUNER_RDA5812
  Flag used to select the RDA5812 as the satellite tuner in the SAT tuner wrapper code.
- DEMOD_Si2183
  Flag used to select the Si2183 as the demodulator (in the wrapper code)

- Si2183_0B_COMPATIBLE
  Flag used to allow compatibility with the Si2183_0B part in the Si2183 code.
- CONFIG_MACROS (if building a console application)
  The configuration macros defined in the Si2183_L2_API.h file match the Skyworks EVBs HW. They can be copied to new macros to fit any new HW. The configuration macro code demonstrates how to select different macros at runtime, based on the available HW.

NB: Dedicated lines are included in the code to avoid forgetting some of the above flags. These lines will trigger compilation errors if the flags are not defined, and contain messages indicating which flag is missing. Commenting these lines is not recommended, the compilation errors MUST be solved by defining the required flags at project level.

9.2 Si2183 source files used in a customer application

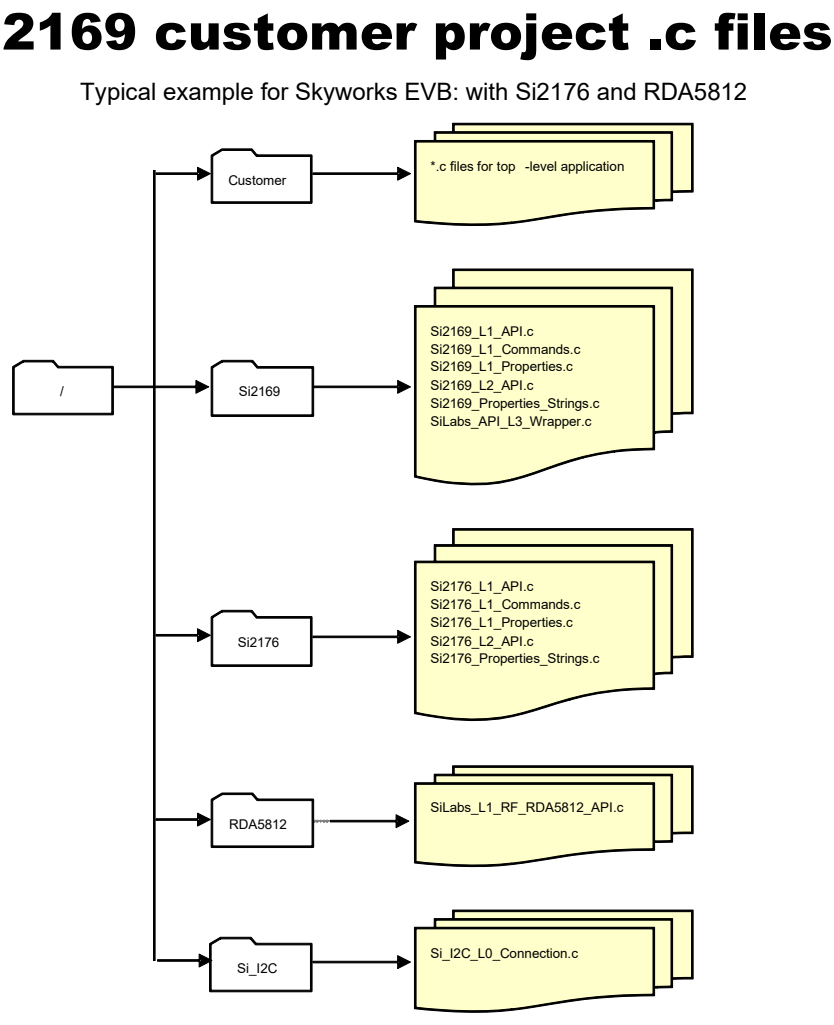


Figure 3: Customer Application Si2183 files

9.2.1 Compilation flags for the typical console application

To compile the typical customer application for customer Si2183/Si2176/RDA5812 hardware, the following compilation flags must be defined at project level:

- TER_TUNER_SILABS
  Flag used to use the TER tuner wrapper API. This is the only possibility provided with the Si2183, as it allows adding new tuners in the TER tuner wrapper and directly using them in a Si2183 application without changes to the Si2183 code.
- TER_TUNER_Si2178
  Flag used to select the Si2178 as the terrestrial tuner in the TER tuner wrapper code.
- SAT_TUNER_SILABS
  Flag used to use the SAT tuner wrapper API. This is the only possibility provided with the Si2183, as it allows adding new tuners in the SAT tuner wrapper and directly using them in a Si2183 application without changes to the Si2183 code.
- SAT_TUNER_RDA5812
  Flag used to select the RDA5812 as the satellite tuner in the SAT tuner wrapper code.
- DEMOD_Si2183
  Flag used to select the Si2183 as the demodulator (in the wrapper code)
- Si2183_0B_COMPATIBLE
  Flag used to allow compatibility with the Si2183_0B part in the Si2183 code.
- CONFIG_MACROS (if building a console application)
  The configuration macros defined in the Si2183_L2_API.h file match the Skyworks EVBs HW. They can be copied to new macros to fit any new HW. The configuration macro code demonstrates how to select different macros at runtime, based on the available HW. In a final application, it is possible to copy/paste the configuration macro into the software initialization part of the code.

NB: Dedicated lines are included in the code to avoid forgetting some of the above flags. These lines will trigger compilation errors if the flags are not defined, and contain messages indicating which flag is missing. Commenting these lines is not recommended, the compilation errors MUST be solved by defining the required flags at project level.

10 Software initialization

The Si2183 software initialization is demonstrated in the configuration macros present in Si2183_L2_API.h.
The following example corresponds to a dual design based on Si21832.

```
#define SW_INIT_Si216x2_EVB_Rev1_x_Si2178_Si2183\
 /* SW Init for front end 0 */\
 front_end                    = &(FrontEnd_Table[0]);\
 SiLabs_API_Frontend_Chip            (front_end, 0x2183);\
 SiLabs_API_SW_Init                  (front_end, 0xc8, 0xc0, 0x14);\
 SiLabs_API_Select_TER_Tuner         (front_end, 0x2178, 0);\
 SiLabs_API_TER_tuner_I2C_connection (front_end, 0);\
 SiLabs_API_TER_Tuner_ClockConfig    (front_end, 1, 1);\
 SiLabs_API_TER_Clock                (front_end, 1, 44, 24, 1);\
 SiLabs_API_TER_FEF_Config           (front_end, 1, 0xa, 1);\
 SiLabs_API_TER_AGC                  (front_end, 0x0, 0, 0xc, 0);\
 SiLabs_API_TER_Tuner_AGC_Input      (front_end, 1);\
 SiLabs_API_TER_Tuner_IF_Output      (front_end, 0);\
 SiLabs_API_Select_SAT_Tuner         (front_end, 0x5816, 0);\
 SiLabs_API_SAT_Select_LNB_Chip      (front_end, 26, 0x10);\
 SiLabs_API_SAT_tuner_I2C_connection (front_end, 0);\
 SiLabs_API_SAT_Clock                (front_end, 2, 33, 27, 1);\
 SiLabs_API_SAT_Spectrum             (front_end, 0);\
 SiLabs_API_SAT_AGC                  (front_end, 0xc, 1, 0x0, 0);\
 SiLabs_API_Set_Index_and_Tag        (front_end, 0, "fe[0]");\
 SiLabs_API_HW_Connect               (front_end, 1);\
 /* SW Init for front end 1 */\
 front_end                    = &(FrontEnd_Table[1]);\
 SiLabs_API_Frontend_Chip            (front_end, 0x2183);\
 SiLabs_API_SW_Init                  (front_end, 0xce, 0xc6, 0x16);\
```

```
  SiLabs_API_Select_TER_Tuner          (front_end, 0x2178, 0);\
  SiLabs_API_TER_tuner_I2C_connection (front_end, 0);\
  SiLabs_API_TER_Tuner_ClockConfig     (front_end, 0, 1);\
  SiLabs_API_TER_Clock                 (front_end, 1, 44, 24, 0);\
  SiLabs_API_TER_FEF_Config            (front_end, 1, 0xb, 1);\
  SiLabs_API_TER_AGC                   (front_end, 0x0, 0, 0xd, 0);\
  SiLabs_API_TER_Tuner_AGC_Input       (front_end, 1);\
  SiLabs_API_TER_Tuner_IF_Output       (front_end, 0);\
  SiLabs_API_Select_SAT_Tuner          (front_end, 0x5816, 0);\
  SiLabs_API_SAT_Select_LNB_Chip       (front_end, 26, 0x10);\
  SiLabs_API_SAT_tuner_I2C_connection (front_end, 0);\
  SiLabs_API_SAT_Clock                 (front_end, 2, 33, 27, 0);\
  SiLabs_API_SAT_Spectrum              (front_end, 0);\
  SiLabs_API_SAT_AGC                   (front_end, 0xd, 1, 0x0, 0);\
  SiLabs_API_Set_Index_and_Tag         (front_end, 1, "fe[1]");\
  SiLabs_API_HW_Connect                (front_end, 1);
```

After initializing the front-ends using the above code, the application can access each front-end using a single pointer: front_end[fe], fe being the index of the front-end.

11 Si2183 Top level application

11.1 Description

The application should perform the software initialization function (SiLabs_API_SW_Init) only once, to allocate the memory structures and initialize the underlying structures with proper values.

It should then call the SiLabs_API_lock_to_carrier function to lock on any channel.

The necessary firmware loading and configuration for the demodulator and the tuners will be automatically handled by the code in an optimal manner.
The required Si2183 firmware will be loaded only on the first call, then the TER and SAT configuration will be done on the first attempt to lock on a TER or SAT carrier, respectively.

The lock will return a fail/success value as fast as possible, using all the capabilities built in the Si2183 with optimum settings.

If lock is achieved (SiLabs_API_lock_to_carrier returns ‘1’), the TS output can be enabled using SiLabs_API_TS_Mode for the required mode (serial or parallel).

To monitor the channel status, call SiLabs_API_FE_status and all status values will be set in the status structure.

11.2 Lock information display while debugging

It can be interesting to call SiLabs_API_Text_status right after SiLabs_API_FE_status and to trace the resulting string to get the same information as in the console application.
This is useful when communicating with Skyworks to get help, since it displays all the important lock state information required to check the IF and AGC settings.

11.3 Debug information while debugging

Enabling the SiTRACES and activating them during software debugging is the best solution to get a rapid feedback from Skyworks in case you have any issue starting your HW.

12 Hardware initialization

The Si2183 applications can use various hardware configurations.

1. It may be decided to use the tuner clock signals for the demodulator.
2. This means that the clock source may be different between TER and SAT modes.
3. If the tuners are providing the Si2183 clock, they need to be initialized BEFORE the Si2183, in order to provide a clock signal to the Si2183.

The hardware design usually uses the Si2183 'i2c passthrough' feature to avoid having i2c signals reaching the tuners when not specifically required.

4. In the case where tuners are connected via the Si2183 i2c passthrough, the passthrough needs to be closed before being able to send i2c to tuners.

The above points 3 and 4 describe a 'chicken and egg' situation, where the application needs to be able to control the i2c_passthru of the Si2183 while the Si2183 is still not yet receiving a valid clock signal.

Also, the chip mode of the Si2183 needs to be controlled even without a valid clock, if the application is to set it to the proper value to select the clock

This is only possible thanks to a specific management of both the i2c passthrough and clock mode in the Si2183 (they use the i2c clock signal):

- The clock mode can be controlled even without a valid clock
- The i2c pass-through can be controlled even without a valid clock
- All other commands and properties require a valid clock to succeed

This means that switching between a given standard and another standard can be complex, especially when the clock source is changing.

To make it easy to manage the standard switches, a dedicated function is used: **Si2183_L2_switch_to_standard**.

This function is a very important part of the application. For this reason, it is described in details in the paragraphs below.

```
int  Si2183_L2_switch_to_standard (Si2183_L2_Context   *front_end,
                                   int new_standard,
                                   unsigned char force_full_init);
```

12.1 Si2183_L2_switch_to_standard function

12.1.1 Description

The Si2183_L2_switch_to_standard function uses a set of global values plus internal flags to sleep or wake up the front-end components depending on the transition (from '0' to '1' or from '1' to '0') of each front-end component and clock source.
Tuners are managed first, in case they are used as clock sources for the DTV demodulator.
Based on the previous state, the minimum set of actions is performed to reduce the transition time to a minimum.
For instance:

- The terrestrial tuner full initialization is only performed once or when using the 'force' mode.
- The DTV demodulator full initialization is only performed once or when using the 'force' flag.
- The tuners are only put in sleep mode when they were previously unused.
- The tuners are only re-started when they were previously in sleep mode.

The following naming conventions apply to the Si2183_L2_switch_to_standard function description and flowchart:

- 'TER' or 'TERRESTRIAL' means 'digital terrestrial reception'.
- 'SAT' or 'SATELLITE' means 'digital satellite reception'.
- 'ANALOG' means 'analog terrestrial reception'.
- 'previous_standard' means the current standard BEFORE calling Si2183_L2_switch_to_standard.

- 'new_standard' means the standard that Si2183_L2_switch_to_standard should switch to.

12.1.2 Global flags and variables

The following flags are stored in the Si2183_L2_Context, to be easily accessible even in multiple front-end applications.

- **Si2183_init_done** indicates if FW download has already been done.
- **first_init_done** indicates if Si2183_L2_switch_to_standard has already been called. It is used to trigger the full init on the very first call.
- **Si2183_init_done** indicates if the Si2183 init has been completed. It is used to avoid a Si2183 complete initialization on every channel change. It is particularly useful if the standard used for the first call was an ATV standard.
- **TER_init_done** indicates if the demodulator initialization for TER reception has been completed. It is used to avoid doing it more than once during execution.
- **TER_tuner_init_done** indicates if the terrestrial tuner initialization has been completed. It is used to avoid a complete initialization of the terrestrial tuner on every channel change.
- **SAT_init_done** indicates if the demodulator initialization for SAT reception has been completed. It is used to avoid doing it more than once during execution.
- **SAT_tuner_init_done** indicates if the satellite tuner initialization has been completed. It is used to avoid a complete initialization of the satellite tuner on every channel change.
- **previous_standard** is used to store the last standard used. It is the main value used by Si2183_L2_switch_to_standard to know which steps to go through. NB: reading the standard value in the Si2183 structure or in the Si2183 itself can't be used here, because the front-end is also capable of ANALOG demodulation, and this is not handled by the Si2183 driver.

12.1.3 Clocks sources definition

The clock settings are now grouped in a set of L3 functions. Check the **USING_SKYWORKS_SUPERSET.pdf** document for details about each configuration field, and also refer to the configuration macros in Si2183_L2_API.h the content of the pre-defined macros.

12.1.4 Internal flags

- **dtv_demod_already_used** indicates if the Si2183 is in use when calling Si2183_L2_switch_to_standard. It will be set when the previous standard is 'TER' or 'SAT'.
- **ter_tuner_already_used** indicates if the terrestrial tuner is in use when calling Si2183_L2_switch_to_standard.
- **sat_tuner_already_used** indicates if the satellite tuner is in use when calling Si2183_L2_switch_to_standard.
- **ter_clock_already_used** indicates if the terrestrial tuner clock is in use when calling Si2183_L2_switch_to_standard.
- **sat_clock_already_used** indicates if the satellite tuner clock is in use when calling Si2183_L2_switch_to_standard.
- **dtv_demod_needed** indicates if the Si2183 must be used for the new standard. It will be set when the new standard is 'TER' or 'SAT'.
- **ter_tuner_needed** indicates if the terrestrial tuner must be used for the new standard. It will be set when the new standard is 'TER' or 'ANALOG', and also for 'SAT' if the terrestrial tuner is used as the clock source for 'SAT'.
- **sat_tuner_needed** indicates if the satellite tuner must be used for the new standard. It will be set when the new standard is 'SAT', and also for 'TER' if the satellite tuner is used as the clock source for 'TER'.
- **ter_clock_needed** indicates if the terrestrial tuner clock must be used for the new standard.
- **sat_clock_needed** indicates if the satellite tuner clock must be used for the new standard.

12.1.5 Behavior

1. Set all internal flags to '0'
2. Set internal flags representing the previous state
3. Set internal flags representing the new state
4. In case a full initialization is required by the caller, set all internal flags to trigger a full initialization.
5. In case the DTV demodulator clock source is going to change, set a flag to request it to sleep first.

6. In case the DTV demodulator is going to transition from ‘1’ to ‘0’, set a flag to request it to sleep first.
7. If a DTV demodulator sleep is requested, sleep the DTV demodulator.
8. Allow communication with the tuners
9. Sleep the SAT tuner if it must transition from ‘1’ to ‘0’
10. Sleep the TER tuner if it must transition from ‘1’ to ‘0’
11. Wake up the SAT tuner if it must transition from ‘0’ to ‘1’
    a. Send the SAT clock to the DTV demodulator if required
12. Wake up the TER tuner if it must transition from ‘0’ to ‘1’
    a. Send the TER clock to the DTV demodulator if required
13. Stop communication with the tuners
14. If the standard changes, set the internal flag to trigger a DTV demodulator wake up
15. Wake up the DTV demodulator if it must transition from ‘0’ to ‘1’
    a. If the DTV demodulator initialization has never been done, do it.
    b. If the Si2183 TER init as not been done and the new standard is a TER standard, initialize the demodulator for TER reception.
    c. If the new standard is a TER standard, set the FEF control ‘on’ for DVB-T2 and ‘off’ otherwise.
    d. If the Si2183 SAT init as not been done and the new standard is a TER standard, do it.
16. If the Si2183 is used, set the standard in the demodulator and reset the demodulator.
17. Store the current standard for next call to Si2183_L2_switch_to_standard

12.1.6 Use

- Calling Si2183_L2_switch_to_standard with the ‘force_full_init’ flag set as ‘1’ makes the tuners and demodulator ready for reception of the requested standard, performing a complete init of all components in use.
- Calling Si2183_L2_switch_to_standard with the ‘force_full_init’ flag set as ‘0’ makes the tuners and demodulator ready for reception of the requested standard, performing the minimal set of operations to switch to the requested standard.

  Si2183_L2_switch_to_standard includes flags to automatically perform a ‘full’ init upon the very first call, to avoid the need to manage this flag from the top-level application.

12.1.7 Profiling

A set of macros are included in the code for profiling uses. It can be disabled by commenting the following line:

```
#define PROFILING
```

The profiling code is not shown in the current document, for ease of understanding.
Profiling provide a view of how much time is spent for each portion (TER tuner, SAT tuner, DTV demodulator) of the function.

12.1.8 Si2183_L2_switch_to_standard traces

For ease of understanding, the traces lines included in Si2183_L2_switch_to_standard are not shown in the current document.

12.1.9 Si2183_L2_switch_to_standard flowchart

Below is the Si2183_L2_switch_to_standard flowchart.
Blocks in light blue are described in further details in the following paragraphs.

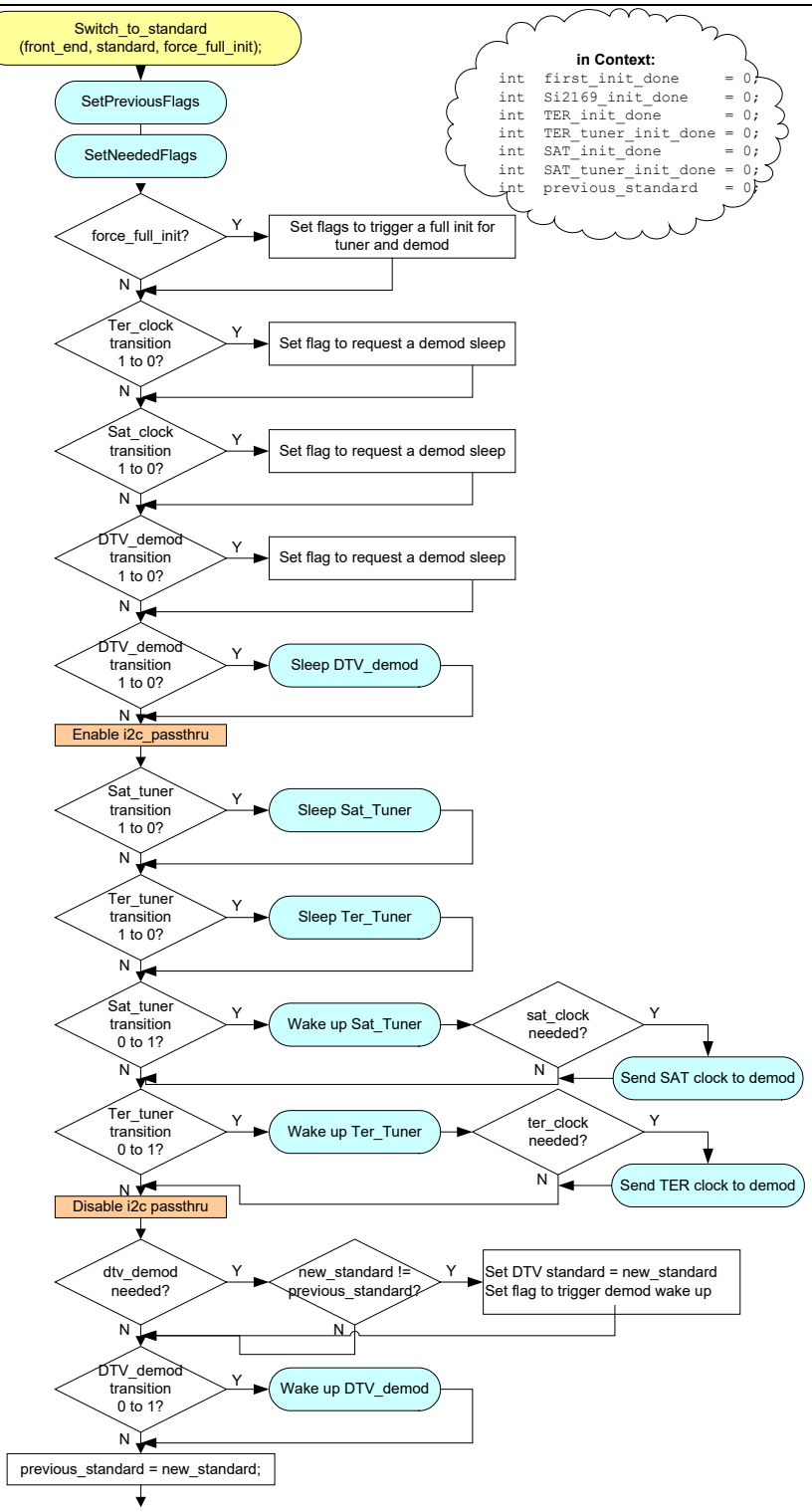


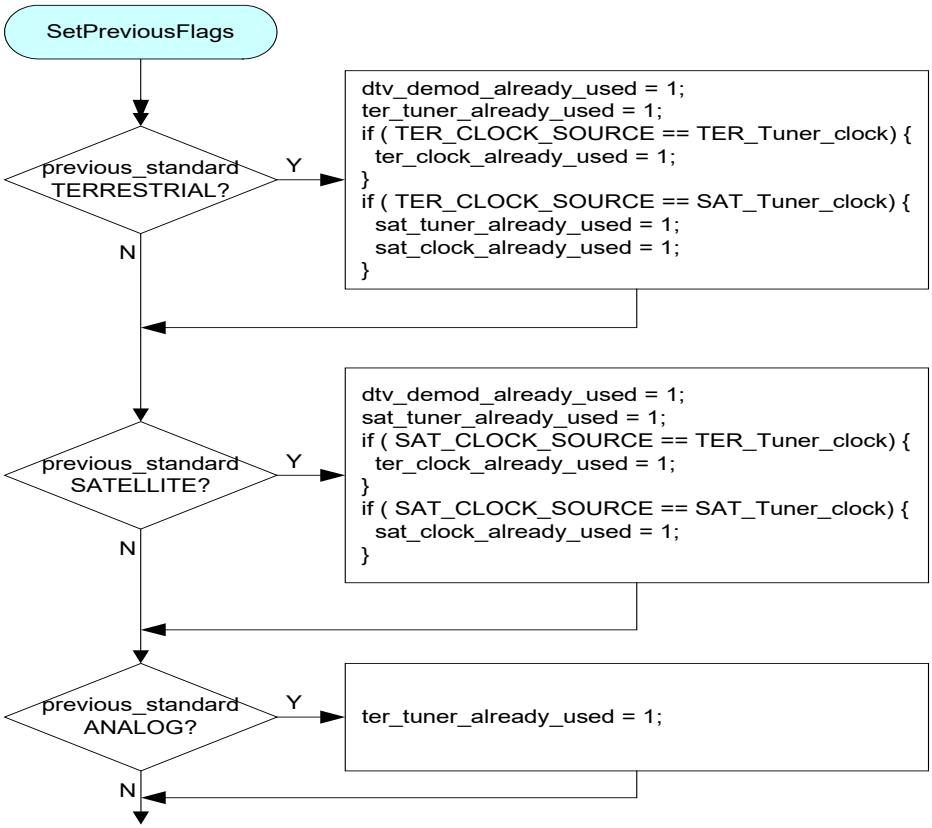
Figure 4: Si2183_L2_switch_to_standard flowchart

NB: the following description assumes that all required tuner functions are defined, and therefore the code to easily remove these during the development phase is not shown.

12.1.10 Set Previous Flags

This block of code is used to set flags describing the previous state of the front-end (all flags not set here are '0').

12.1.10.1 Flowchart




```
/* ------------------------------------------------------------ */
/* Set Previous Flags                                           */
/* Setting flags representing the previous state                */
/* NB: Any value not matching a known standard will init as ATV */
/* Logic applied:                                               */
/*  dtv demod was used for TERRESTRIAL and SATELLITE reception  */
/*  ter tuner was used for TERRESTRIAL reception                */
/*   and for SATELLITE reception if it is the SAT clock source  */
/*  sat tuner was used for SATELLITE reception                  */
/*   and for TERRESTRIAL reception if it is the TER clock source*/
/* ------------------------------------------------------------ */
switch (front_end->previous_standard) {
  case Si2183_DD_MODE_PROP_MODULATION_DVBT :
  case Si2183_DD_MODE_PROP_MODULATION_ISDBT :
  case Si2183_DD_MODE_PROP_MODULATION_DVBT2:
  case Si2183_DD_MODE_PROP_MODULATION_MCNS :
  case Si2183_DD_MODE_PROP_MODULATION_DVBC : {
    dtv_demod_already_used = 1;
    ter_tuner_already_used = 1;
    if ( front_end->demod->tuner_ter_clock_source == Si2183_TER_Tuner_clock) {
      ter_clock_already_used = 1;
    }
    if ( front_end->demod->tuner_ter_clock_source == Si2183_SAT_Tuner_clock) {
      sat_tuner_already_used = 1;
```

```
      sat_clock_already_used = 1;
    }
    break;
  }
  case Si2183_DD_MODE_PROP_MODULATION_DVBC2: {
    dtv_demod_already_used = 1;
    ter_tuner_already_used = 1;
    if ( front_end->demod->tuner_ter_clock_source == Si2183_TER_Tuner_clock) {
      ter_clock_already_used = 1;
    }
    if ( front_end->demod->tuner_ter_clock_source == Si2183_SAT_Tuner_clock) {
      sat_tuner_already_used = 1;
      sat_clock_already_used = 1;
    }
    break;
  }
  case Si2183_DD_MODE_PROP_MODULATION_DVBS  :
  case Si2183_DD_MODE_PROP_MODULATION_DVBS2 :
  case Si2183_DD_MODE_PROP_MODULATION_DSS   : {
    dtv_demod_already_used = 1;
    sat_tuner_already_used = 1;
    if ( front_end->demod->tuner_sat_clock_source == Si2183_TER_Tuner_clock) {
      ter_tuner_already_used = 1;
      ter_clock_already_used = 1;
    }
    if ( front_end->demod->tuner_sat_clock_source == Si2183_SAT_Tuner_clock) {
      sat_clock_already_used = 1;
    }
    break;
  }
  case Si2183_DD_MODE_PROP_MODULATION_ANALOG: {
    ter_tuner_already_used = 1;
    dtv_demod_sleeping = 1;
    break;
  }
  default : /* SLEEP */   {
    ter_tuner_already_used = 0;
    dtv_demod_sleeping = 1;
    break;
  }
}
```

12.1.11 Set Needed Flags

This block of code is used to set flags describing the new state of the front-end (all flags not set here are '0').

```
  /* ------------------------------------------------------------ */
/* Set Needed Flags                                             */
/* Setting flags representing the new state                     */
/* Logic applied:                                               */
/*  dtv demod is needed for TERRESTRIAL and SATELLITE reception */
/*  ter tuner is needed for TERRESTRIAL reception               */
/*   and for SATELLITE reception if it is the SAT clock source  */
/*  sat tuner is needed for SATELLITE reception                 */
/*   and for TERRESTRIAL reception if it is the TER clock source*/
/* ------------------------------------------------------------ */
switch (new_standard) {
  case Si2183_DD_MODE_PROP_MODULATION_DVBT :
  case Si2183_DD_MODE_PROP_MODULATION_ISDBT :
  case Si2183_DD_MODE_PROP_MODULATION_DVBT2:
  case Si2183_DD_MODE_PROP_MODULATION_MCNS :
  case Si2183_DD_MODE_PROP_MODULATION_DVBC : {
    dtv_demod_needed = 1;
    ter_tuner_needed = 1;
    if ( front_end->demod->tuner_ter_clock_source == Si2183_TER_Tuner_clock) {
      ter_clock_needed = 1;
    }
```

```
      if ( front_end->demod->tuner_ter_clock_source == Si2183_SAT_Tuner_clock) {
        sat_clock_needed = 1;
        sat_tuner_needed = 1;
      }
      break;
    }
    case Si2183_DD_MODE_PROP_MODULATION_DVBC2: {
      dtv_demod_needed = 1;
      ter_tuner_needed = 1;
      if ( front_end->demod->tuner_ter_clock_source == Si2183_TER_Tuner_clock) {
        ter_clock_needed = 1;
      }
      if ( front_end->demod->tuner_ter_clock_source == Si2183_SAT_Tuner_clock) {
        sat_tuner_needed = 1;
        sat_clock_needed = 1;
      }
      break;
    }
    case Si2183_DD_MODE_PROP_MODULATION_DVBS :
    case Si2183_DD_MODE_PROP_MODULATION_DVBS2:
    case Si2183_DD_MODE_PROP_MODULATION_DSS  : {
      dtv_demod_needed = 1;
      sat_tuner_needed = 1;
      if ( front_end->demod->tuner_sat_clock_source == Si2183_TER_Tuner_clock) {
        ter_clock_needed = 1;
        ter_tuner_needed = 1;
      }
      if ( front_end->demod->tuner_sat_clock_source == Si2183_SAT_Tuner_clock) {
        sat_clock_needed = 1;
      }
      break;
    }
    case Si2183_DD_MODE_PROP_MODULATION_ANALOG: {
      ter_tuner_needed = 1;
      break;
    }
    default : /* SLEEP */   {
      ter_tuner_needed = 0;
      break;
    }
  }
```

12.1.12 Sleep DTV Demod

This block of code is used to sleep the DTV demodulator. It relies on the function provided in the demodulator's API. The cases when the demodulator is required to sleep are:

- When the standard is 'analog'
- When the standard is unknown, which menas 'sleep'
- When the clock source changes

```
    /* ------------------------------------------------------------ */
    /* Request demodulator sleep if its clock will be stopped       */
    /* ------------------------------------------------------------ */
  #ifdef    TERRESTRIAL_FRONT_END
    SiTRACE("ter_tuner_already_used %d, ter_tuner_needed %d\n",
ter_tuner_already_used,ter_tuner_needed);
    SiTRACE("ter_clock_already_used %d, ter_clock_needed %d\n",
ter_clock_already_used,ter_clock_needed);
    if ((ter_tuner_already_used == 1) & (ter_tuner_needed == 0) ) { SiTRACE("TER tuner 1->0 "); }
    if ((ter_tuner_already_used == 0) & (ter_tuner_needed == 1) ) { SiTRACE("TER tuner 0->1 "); }
    if ((ter_clock_already_used == 1) & (ter_clock_needed == 0) ) { SiTRACE("TER clock 1->0 ");
dtv_demod_sleep_request = 1; }
    if ((ter_clock_already_used == 0) & (ter_clock_needed == 1) ) { SiTRACE("TER clock 0->1 ");
dtv_demod_sleep_request = 1; }
  #endif /* TERRESTRIAL_FRONT_END */
  #ifdef    SATELLITE_FRONT_END
```

```
    SiTRACE("sat_tuner_already_used %d, sat_tuner_needed %d\n",
sat_tuner_already_used,sat_tuner_needed);
    SiTRACE("sat_clock_already_used %d, sat_clock_needed %d\n",
sat_clock_already_used,sat_clock_needed);
    if ((sat_tuner_already_used == 1) & (sat_tuner_needed == 0) ) { SiTRACE("SAT tuner 1->0 "); }
    if ((sat_tuner_already_used == 0) & (sat_tuner_needed == 1) ) { SiTRACE("SAT tuner 0->1 "); }
    if ((sat_clock_already_used == 1) & (sat_clock_needed == 0) ) { SiTRACE("SAT clock 1->0 ");
dtv_demod_sleep_request = 1; }
    if ((sat_clock_already_used == 0) & (sat_clock_needed == 1) ) { SiTRACE("SAT clock 0->1 ");
dtv_demod_sleep_request = 1; }
  #endif /* SATELLITE_FRONT_END */
    SiTRACE("\n");
    /* ------------------------------------------------------------ */
    /* Request demodulator sleep if transition from '1' to '0'      */
    /* ------------------------------------------------------------ */
    if ((dtv_demod_already_used == 1) & (dtv_demod_needed == 0) ) { dtv_demod_sleep_request = 1; }
    SiTRACE(" %s-->%s switch flags    dtv_demod_already_used %d, dtv_demod_needed %d,
dtv_demod_sleep_request %d, dtv_demod_sleeping %d\n", Si2183_standardName(front_end->previous_standard),
Si2183_standardName(new_standard),
                                  dtv_demod_already_used,    dtv_demod_needed,
dtv_demod_sleep_request,    dtv_demod_sleeping   );
    /* ------------------------------------------------------------ */
    /* Sleep dtv demodulator if requested                           */
    /* ------------------------------------------------------------ */
    if (dtv_demod_sleep_request == 1) {
      SiTRACE("Sleep DTV demod\n");
      /* To avoid issues with the FEF pin when switching from T2 to ANALOG, set the demodulator for DVB-
T/non auto detect reception before POWER_DOWN */
      if (new_standard                        == Si2183_DD_MODE_PROP_MODULATION_ANALOG) {
        if ( ( (front_end->previous_standard == Si2183_DD_MODE_PROP_MODULATION_DVBT  )
             & (front_end->demod->prop->dd_mode.auto_detect ==
Si2183_DD_MODE_PROP_AUTO_DETECT_AUTO_DVB_T_T2) )
             | (front_end->previous_standard == Si2183_DD_MODE_PROP_MODULATION_DVBT2 ) ) {
          front_end->demod->prop->dd_mode.modulation  = Si2183_DD_MODE_PROP_MODULATION_DVBT;
          front_end->demod->prop->dd_mode.auto_detect = Si2183_DD_MODE_PROP_AUTO_DETECT_NONE;
          Si2183_L1_SetProperty2(front_end->demod, Si2183_DD_MODE_PROP_CODE);
          Si2183_L1_DD_RESTART  (front_end->demod);
        }
      }
      /* If the demod is not needed, it means it's either going to ANALOG or SLEEP   */
      /*  In this case, set the demod silent, putting all possible pins to tristate  */
      /* To nicely recover from this, Si2183_L2_Configure will need to be re-applied */
      /*  so set the flags to allow this                                             */
      if (dtv_demod_needed == 0) {
        Si2183_L2_SILENT(front_end, 1);
      }
      Si2183_STANDBY (front_end->demod);
      dtv_demod_sleeping = 1;
      SiTRACE(" %s-->%s switch now   dtv_demod_sleeping %d\n", Si2183_standardName(front_end-
>previous_standard), Si2183_standardName(new_standard), dtv_demod_sleeping   );
      DTV_DELAY
    }
```

12.1.13 Sleep Sat Tuner

This block of code is used to sleep the satellite tuner.

```
  /* ------------------------------------------------------------ */
  /* Sleep Sat Tuner                                              */
  /* Sleep satellite   tuner if transition from '1' to '0'        */
  /* ------------------------------------------------------------ */
  if ((sat_tuner_already_used == 1) & (sat_tuner_needed == 0) ) {
   #ifdef    INDIRECT_I2C_CONNECTION
    SiTRACE("Connect SAT tuner i2c to put it in sleep mode?\n");
    if (sat_i2c_connected==0) {
      SiTRACE("-- I2C -- Connect SAT tuner i2c to put it in sleep mode\n");
      front_end->f_SAT_tuner_enable(front_end->callback);
      sat_i2c_connected++;
```

```
    }
    DTV_DELAY
  #endif /* INDIRECT_I2C_CONNECTION */
    SiTRACE("Sleep satellite tuner\n");
    #ifdef    SAT_TUNER_CLOCK_OFF
    if (front_end->demod->tuner_sat_clock_control != Si2183_CLOCK_ALWAYS_ON) {
      if ( (res = SAT_TUNER_CLOCK_OFF(front_end->tuner_sat)) !=0 ) {
        SiTRACE("Satellite tuner CLOCK OFF error: 0x%02x\n",res );
        return 0;
      }
    }
    #endif /* SAT_TUNER_CLOCK_OFF */
    #ifdef    SAT_TUNER_STANDBY
    if ((res= SAT_TUNER_STANDBY(front_end->tuner_sat)) !=0 ) {
      SiTRACE("Satellite tuner Standby error: 0x%02x\n",res );
      return 0;
    }
    #endif /* SAT_TUNER_STANDBY */
    SAT_DELAY
  }
```

12.1.14 Wake Up Sat Tuner

This block of code is used to wake up the satellite tuner

```
    /* ------------------------------------------------------------ */
  /* Wakeup Sat Tuner                                             */
  /* Wake up satellite   tuner if transition from '0' to '1'      */
  /* ------------------------------------------------------------ */
  if ((sat_tuner_already_used == 0) & (sat_tuner_needed == 1)) {
   #ifdef    INDIRECT_I2C_CONNECTION
    if (sat_i2c_connected==0) {
      SiTRACE("-- I2C -- Connect SAT tuner i2c to init/wakeup it\n");
      front_end->f_SAT_tuner_enable(front_end->callback);
      sat_i2c_connected++;
    }
    DTV_DELAY
   #endif /* INDIRECT_I2C_CONNECTION */
    if (front_end->SAT_tuner_init_done==0) {
      SiTRACE("Init satellite tuner\n");
      #ifdef    SAT_TUNER_INIT
      if ((res= SAT_TUNER_INIT(front_end->tuner_sat)) !=0) {
        SiTRACE("Satellite tuner HW init error: 0x%02x\n",res );
        return 0;
      }
      #endif /* SAT_TUNER_INIT */
      front_end->SAT_tuner_init_done = 1;
    } else {
      SiTRACE("Wakeup satellite tuner\n");
      #ifdef    SAT_TUNER_WAKEUP
      if ((res= SAT_TUNER_WAKEUP(front_end->tuner_sat)) !=0) {
        SiTRACE("Satellite tuner wake up error: 0x%02x\n",res );
        return 0;
      }
      #endif /* SAT_TUNER_WAKEUP */
    }
    SAT_DELAY
  }
```

12.1.15 Send SAT clock to demod

This block of code is used to turn the satellite tuner's clock output on for the DTV demodulator.

```
  /* ------------------------------------------------------------ */
  /* If the satellite tuner's clock is required, activate it      */
  /* ------------------------------------------------------------ */
  SiTRACE("sat_clock_needed %d\n",sat_clock_needed);
  if (sat_clock_needed) {
```

```
    #ifdef    SAT_TUNER_CLOCK_ON
     #ifdef    INDIRECT_I2C_CONNECTION
      if (sat_i2c_connected==0) {
        SiTRACE("-- I2C -- Connect SAT tuner i2c to start its clock\n");
        front_end->f_SAT_tuner_enable(front_end->callback);
        sat_i2c_connected++;
      }
      DTV_DELAY
     #endif /* INDIRECT_I2C_CONNECTION */
    if (front_end->demod->tuner_sat_clock_control != Si2183_CLOCK_ALWAYS_OFF) {
      SiTRACE("Turn satellite tuner clock on\n");
      if ((res= SAT_TUNER_CLOCK_ON(front_end->tuner_sat) ) !=0) {
        SiTRACE("Satellite tuner CLOCK ON error: 0x%02x\n",res );
        return 0;
      }
    }
    #endif /* SAT_TUNER_CLOCK_ON */
    SAT_DELAY
}
```

12.1.16 Sleep Ter Tuner

This block of code is used to sleep the terrestrial tuner. It will first switch the clock output off then set the terrestrial tuner in standby mode.

```
  /* ------------------------------------------------------------ */
  /* Sleep Ter Tuner                                              */
  /* Sleep terrestrial tuner  if transition from '1' to '0'       */
  /* ------------------------------------------------------------ */
  if ((ter_tuner_already_used == 1) & (ter_tuner_needed == 0) ) {
   #ifdef    INDIRECT_I2C_CONNECTION
    if (ter_i2c_connected==0) {
      SiTRACE("-- I2C -- Connect TER tuner i2c to sleep it\n");
      front_end->f_TER_tuner_enable(front_end->callback);
      ter_i2c_connected++;
    }
    DTV_DELAY
   #endif /* INDIRECT_I2C_CONNECTION */
    SiTRACE("Sleep terrestrial tuner\n");
    #ifdef    TER_TUNER_CLOCK_OFF
    if (front_end->demod->tuner_ter_clock_control != Si2183_CLOCK_ALWAYS_ON) {
      SiTRACE("Terrestrial tuner clock OFF\n");
      if ((res= TER_TUNER_CLOCK_OFF(front_end->tuner_ter)) !=0 ) {
        SiTRACE("Terrestrial tuner CLOCK OFF error: 0x%02x : %s\n",res, TER_TUNER_ERROR_TEXT(res) );
        SiERROR("Terrestrial tuner CLOCK OFF error!\n");
        return 0;
      }
    }
    #endif /* TER_TUNER_CLOCK_OFF */
    #ifdef    TER_TUNER_STANDBY
    SiTRACE("Terrestrial tuner STANDBY\n");
    if ((res= TER_TUNER_STANDBY(front_end->tuner_ter)) !=0 ) {
      SiTRACE("Terrestrial tuner Standby error: 0x%02x : %s\n",res, TER_TUNER_ERROR_TEXT(res) );
      SiERROR("Terrestrial tuner Standby error!\n");
      return 0;
    }
    #endif /* TER_TUNER_STANDBY */
    TER_DELAY
  }
```

12.1.17 Wake Up Ter Tuner

This block of code is used to initialize the terrestrial tuner.
Depending on the TER_tuner_init_done flag, it will perform a full init of the terrestrial tuner or a simple ‘wake up’.

```
  /* ------------------------------------------------------------ */
  /* Wakeup Ter Tuner                                             */
  /* Wake up terrestrial tuner if transition from '0' to '1'      */
  /* ------------------------------------------------------------ */
  if ((ter_tuner_already_used == 0) & (ter_tuner_needed == 1)) {
   #ifdef    INDIRECT_I2C_CONNECTION
    if (ter_i2c_connected==0) {
      SiTRACE("-- I2C -- Connect TER tuner i2c to init/wakeup it\n");
      front_end->f_TER_tuner_enable(front_end->callback);
      ter_i2c_connected++;
    }
    DTV_DELAY
   #endif /* INDIRECT_I2C_CONNECTION */
    /* Do a full init of the Ter Tuner only if it has not been already done */
    if (front_end->TER_tuner_init_done==0) {
      SiTRACE("Init terrestrial tuner\n");
      #ifdef    TER_TUNER_INIT
      if ((res= TER_TUNER_INIT(front_end->tuner_ter)) !=0) {
        #ifdef    TER_TUNER_ERROR_TEXT
        SiTRACE("Terrestrial tuner HW init error: 0x%02x : %s\n",res, TER_TUNER_ERROR_TEXT(res) );
        #endif /* TER_TUNER_ERROR_TEXT */
        SiERROR("Terrestrial tuner HW init error!\n");
        return 0;
      }
      #ifdef    TER_TUNER_AGC_EXTERNAL
      if (front_end->demod->tuner_ter_agc_control) {
        TER_TUNER_AGC_EXTERNAL(front_end->tuner_ter);
      }
      #endif /* TER_TUNER_AGC_EXTERNAL */
      #endif /* TER_TUNER_INIT */
      front_end->TER_tuner_init_done =1;
    } else {
      SiTRACE("Wakeup terrestrial tuner\n");
      #ifdef    TER_TUNER_WAKEUP
      if ((res= TER_TUNER_WAKEUP(front_end->tuner_ter)) !=0) {
        SiTRACE("Terrestrial tuner wake up error: 0x%02x : %s\n",res, TER_TUNER_ERROR_TEXT(res) );
        SiERROR("Terrestrial tuner wake up error!\n");
        return 0;
      }
      #endif /* TER_TUNER_WAKEUP */
    }
    TER_DELAY
  }
```

12.1.18 Send TER clock to demod

This block of code is used to turn the terrestrial tuner's clock output on for the DTV demod.

```
    /* ------------------------------------------------------------ */
    /* If the terrestrial tuner's clock is required, activate it    */
    /* ------------------------------------------------------------ */
  SiTRACE("ter_clock_needed %d\n",ter_clock_needed);
  if (ter_clock_needed) {
    SiTRACE("Turn terrestrial tuner clock on\n");
    #ifdef    TER_TUNER_CLOCK_ON
     #ifdef    INDIRECT_I2C_CONNECTION
      if (ter_i2c_connected==0) {
        SiTRACE("-- I2C -- Connect TER tuner i2c to start its clock\n");
        front_end->f_TER_tuner_enable(front_end->callback);
        ter_i2c_connected++;
      }
      DTV_DELAY
     #endif /* INDIRECT_I2C_CONNECTION */
    if (front_end->demod->tuner_ter_clock_control != Si2183_CLOCK_ALWAYS_OFF) {
      SiTRACE("Terrestrial tuner CLOCK ON\n");
      if ((res= TER_TUNER_CLOCK_ON(front_end->tuner_ter) ) !=0) {
        SiTRACE("Terrestrial tuner CLOCK ON error: 0x%02x : %s\n",res, TER_TUNER_ERROR_TEXT(res) );
        SiERROR("Terrestrial tuner CLOCK ON error!\n");
        return 0;
```

```
        }
      }
      #endif /* TER_TUNER_CLOCK_ON */
      TER_DELAY
    }
    if ((front_end->previous_standard != new_standard) & (dtv_demod_needed == 0) & (front_end->demod-
>media == Si2183_TERRESTRIAL)) {
     if (front_end->demod->media == Si2183_TERRESTRIAL) {
      #ifdef    TER_TUNER_ATV_LO_INJECTION
       TER_TUNER_ATV_LO_INJECTION(front_end->tuner_ter);
      #endif /* TER_TUNER_ATV_LO_INJECTION */

     }
    }
```

12.1.19 Change DTV Demod standard and adapt TER tuner

```
    /* ------------------------------------------------------------ */
    /* Change Dtv Demod standard, adapt TER tuner if required       */
    /* ------------------------------------------------------------ */
    if ((front_end->previous_standard != new_standard) & (dtv_demod_needed == 1)) {
      SiTRACE("Store demod standard (%d)\n", new_standard);
      front_end->demod->standard = new_standard;
      DTV_DELAY
  #ifdef    TERRESTRIAL_FRONT_END
      if (front_end->demod->media == Si2183_TERRESTRIAL) {
      #ifdef    TER_TUNER_DTV_LO_INJECTION
       TER_TUNER_DTV_LO_INJECTION(front_end->tuner_ter);
      #endif /* TER_TUNER_DTV_LO_INJECTION */
      #ifdef    TER_TUNER_DTV_LIF_OUT_AMP
      /* Adjusting LIF signal for cable or terrestrial reception */
        switch (new_standard) {
          case Si2183_DD_MODE_PROP_MODULATION_DVBT :
          case Si2183_DD_MODE_PROP_MODULATION_DVBC2:
          case Si2183_DD_MODE_PROP_MODULATION_DVBT2:
          {
            TER_TUNER_DTV_LIF_OUT_AMP(front_end->tuner_ter, 0);
            break;
          }
         case Si2183_DD_MODE_PROP_MODULATION_MCNS :
         case Si2183_DD_MODE_PROP_MODULATION_DVBC : {
            TER_TUNER_DTV_LIF_OUT_AMP(front_end->tuner_ter, 1);
           break;
         }
         default: break;
        }
      #endif /* TER_TUNER_DTV_LIF_OUT_AMP */
      }
  #endif /* TERRESTRIAL_FRONT_END */
    }
```

12.1.20 Wake up DTV Demod

This block of code is used to init the DTV demodulator.
Depending on the Si2183_init_done flag, it will perform a full init of the Si2183 or a simple re-init.
NB: The Si2183 API provides two init functions:

- Si2183_L1_Demod_init_after_reset is used to perform a full init.
- Si2183_L1_Demod_Re_init is used to switch the Si2183 to another standard.

```
     /* ------------------------------------------------------------ */
    /* Wakeup Dtv Demod                                             */
    /*  if it has been put in 'standby mode' and is needed          */
    /* ------------------------------------------------------------ */
    if (front_end->Si2183_init_done) {
      SiTRACE("dtv_demod_sleeping %d\n", dtv_demod_sleeping);
```

```
      if ((dtv_demod_sleeping == 1) & (dtv_demod_needed == 1) ) {
        if (dtv_demod_already_used == 0) {
          SiTRACE("Take DTV demod out of SILENT mode\n");
          Si2183_L2_SILENT(front_end, 0);
        } else {
          SiTRACE("Wake UP DTV demod\n");
          if (Si2183_WAKEUP (front_end->demod) == NO_Si2183_ERROR) {
            SiTRACE("Wake UP DTV demod OK\n");
          } else {
            SiERROR("Wake UP DTV demod failed!\n");
            SiTRACE("Wake UP DTV demod failed!\n");
            return 0;
          }
        }
      }
    }
```

12.1.21 Setup DTV Demod

This block of code is trigger the FW download in the DTV demodulator if required, and to set all demod parameters according to the new standard

```
    /* ------------------------------------------------------------ */
    /* Setup Dtv Demod                                              */
    /* Setup dtv demodulator if transition from '0' to '1'          */
    /* ------------------------------------------------------------ */
    if (dtv_demod_needed == 1) {
      /* Do the 'first init' only the first time, plus if requested  */
      /* (when 'force' flag is 1, Si2183_init_done is set to '0')    */
      if (!front_end->Si2183_init_done) {
        SiTRACE("Init demod\n");
        if (Si2183_Init(front_end->demod) == NO_Si2183_ERROR) {
          front_end->Si2183_init_done = 1;
          SiTRACE("Demod init OK\n");
        } else {
          SiTRACE("Demod init failed!\n");
          SiERROR("Demod init failed!\n");
          return 0;
        }
      }
  #ifdef    TERRESTRIAL_FRONT_END
      if (front_end->demod->media == Si2183_TERRESTRIAL) {
        SiTRACE("front_end->demod->media Si2183_TERRESTRIAL\n");
        if (front_end->TER_init_done == 0) {
          SiTRACE("Configure demod for TER\n");
          if (Si2183_Configure(front_end->demod) == NO_Si2183_ERROR) {
            /* set dd_mode.modulation again, as it is overwritten by Si2183_Configure */
            front_end->demod->prop->dd_mode.modulation = new_standard;
            front_end->TER_init_done = 1;
          } else {
            SiTRACE("Demod TER configuration failed !\n");
            SiERROR("Demod TER configuration failed !\n");
            return 0;
          }
        }
        DTV_DELAY
  #endif /* TERRESTRIAL_FRONT_END */
        /* ------------------------------------------------------------ */
        /* Manage FEF mode in TER tuner                                 */
        /* ------------------------------------------------------------ */
        if (new_standard == Si2183_DD_MODE_PROP_MODULATION_DVBT2) {
          Si2183_L2_TER_FEF_SETUP (front_end, 1);
        } else {
          Si2183_L2_TER_FEF_SETUP (front_end, 0);
        }
  #ifdef    TERRESTRIAL_FRONT_END
        TER_DELAY
      }
  #endif /* TERRESTRIAL_FRONT_END */
```

```
      if (front_end->demod->media == Si2183_SATELLITE  ) {
        SiTRACE("front_end->demod->media Si2183_SATELLITE\n");
        if (front_end->SAT_init_done == 0) {
          SiTRACE("Configure demod for SAT\n");
          if (Si2183_Configure(front_end->demod) == NO_Si2183_ERROR) {
            /* set dd_mode.modulation again, as it is overwritten by Si2183_Configure */
            front_end->demod->prop->dd_mode.modulation      =
Si2183_DD_MODE_PROP_MODULATION_AUTO_DETECT;
            front_end->demod->prop->dd_mode.auto_detect     =
Si2183_DD_MODE_PROP_AUTO_DETECT_AUTO_DVB_S_S2;
            front_end->SAT_init_done = 1;
          } else {
            SiTRACE("Demod SAT configuration failed !\n");
            SiERROR("Demod SAT configuration failed !\n");
            return 0;
          }
        }
      }
    }
    front_end->demod->prop->dd_mode.invert_spectrum = Si2183_L2_Set_Invert_Spectrum(front_end);
    if (Si2183_L1_SetProperty2(front_end->demod, Si2183_DD_MODE_PROP_CODE)==0) {
      Si2183_L1_DD_RESTART(front_end->demod);
    } else {
      SiTRACE("Demod restart failed !\n");
      return 0;
    }
    DTV_DELAY
```

13 Software development scenarios

Some options can prove very useful during development, as described below.
Note that all the source code based options can run exactly the same source code, compiled using different compilation flags.

13.1 Skyworks console application running on Skyworks EVB

This means running the delivered source code on the EVB, as Skyworks does it to validate the source code.

This is the reference code used with the reference hardware. It should always be used as the primary comparison point in terms of code behavior and hardware performances.

This step is a must to get better knowledge of the source code and how to use it. It is also a good opportunity to get familiar with the traces and how to configure them to help in the debug.

13.2 Skyworks console application running on customer hardware

This step is interesting to validate new hardware using known software.
It is always very difficult to introduce at the same time new software and new hardware.

13.3 Customer application using Skyworks Solutions hardware

This means using the Si2183 evaluation board to work on the final code, in order to develop the final application while customer hardware is still under development.

Advantages:

No need for the software team to wait for the hardware before starting code development and validation.
Possible use of Skyworks Solutions GUI for comparative testing with the EVB. This can be done by connecting the I2C interface from an EVB to the customer HW (0 Ohm resistors on the EVB can be removed to disconnect the front-end from the I2C interface). The EVB is then used only as an I2C interface.

Requirements:

Use the Skyworks Solutions Layer 0, the single part that will need to be replaced when connecting the final hardware.

13.4 Skyworks Solutions GUI and API on customer hardware

This can be useful to validate new hardware using the GUI and API delivered by Skyworks Solutions.
It helps separating possible software issues from hardware issues.

Advantages:

The EVB GUI can work on the new hardware, provided that the proper values are entered in the 'system' tab, mainly for clock input and frequency selection.

If an automated test suite has been developed based on Skyworks Solutions EVB, it can be re-run on the new hardware.

The people who evaluated the Skyworks EVB using the Skyworks GUI will be using the same GUI to validate the new hardware. In the context of a NIM, there is immediate need to develop a custom GUI.

Requirements:

Properly wrapped Layer 0 functions (as will be needed for the final application).

One compatible tuner driver. Please contact Skyworks Solutions to check if such a driver is already available, if not delivered with the EVB. Building a new tuner driver is relatively easy, and a typical framework can be delivered if needed.

13.5 Customer application on customer hardware

This is the final situation

Going to this from the beginning, i.e. introducing simultaneously new code and new HW is highly error prone and is very difficult to debug until all HW and SW issues are cleared, if any.

It is highly recommended to go through some of the above steps before connecting new code with new hardware, and to refer to the 'mixed' applications for comparison in case of development issues.

We consider the following steps as being very valuable in terms of getting familiar with the code or as comparison points:

- Skyworks console application running on Skyworks EVB (for code knowledge: this is THE reference).
- Skyworks console application running on customer hardware (for hardware debugging)
- Customer application using Skyworks Solutions hardware (for software debugging)

14 Required configuration for PC compilation of the original code

The required configuration is a PC running Microsoft Windows (XP or 7), with USB2.0 capability.
USB2.0 compatibility is only required to be able to use the Cypress USB2.0 interface chip implemented on Skyworks EVBs.

Once I2C porting to the customer hardware is achieved, it is still possible to connect the Skyworks EVB to the i2c bus.

The delivered code is ANSI compliant, and compiles as C code, with no Windows specifics for the compilation.

The code is easily portable to other platforms such as Linux, and should compile under Visual Studio as well as with GCC.

15 First use with an EVB

In order to get familiar with the code the best solution is to rebuild it and run it on one Skyworks evaluation board, ideally one board for which you already have the evaluation software.

Here is the recommended step by step approach:

15.1 EVB validation using the EVB's GUI

Check that the board is working, using Skyworks evaluation GUI software. Make sure you can lock onto at least one channel. Remember the RF frequency of at least one valid channel.

15.2 Rebuilding the sample application

- Include all the source files in your development environment. Refer to 10.1Si2183 console application to get a list of the source files involved.
- Configure your project to have it link with the Cypress USB DLL. Refer to **SW_Skyworks_Broadcast_Video_Layer_0_ vx.x.x.doc** for details on this part.
- Add all folders containing source code .h files in the search list of your compiler.
- Rebuild the console application.

15.3 Running the sample application on a Skyworks EVB

Run the sample application executable, to see what you can do with it.

For instance, power-cycle the board and relock the demodulator using the sample application on one valid channel:

- From the start, try to lock on a valid channel using 'lock'.
- Select the reception standard using either 'T', 'T2', 'C', 'C2', 'S', 'S2' or 'ISDBT'
- Enter the frequency (in Hz for TER, in kHz for SAT).
- Enter the remaining values as requested by the application (standard-dependent).
- Check the demodulator status using the 'return' key.
- Enter 'm' to redisplay the menu at any time.

16 Porting the EVB code on customer hardware

After going through the following steps, the application should be working on customer hardware as it works on the EVB.
It is highly recommended to validate the code at this stage, making sure it can drive the customer hardware using the console application.
This step will ensure a smooth integration in the final application.

16.1 Using the EVB's i2c bus

Contact Skyworks to get the diagram of your EVB, then disconnect the SDA/SCL signals from the Demodulator. Then, you have an i2c communication bus which you can connect to your new HW. Using either the same configuration macro as with the EVB (if both HW match) or using a new macro with a modified configuration you should be able to control your new HW even before having ported the i2c layer.
This can also be convenient since you will be able to use Skyworks GUIs to control your HW as well.

16.2 i2c porting

Refer to SW_Skyworks_Broadcast_Video_Layer_0_ vx.x.x.doc for details on how to port the i2c layer, especially how to communicate using your custom i2c mode.

Ideally, keep the ability to easily switch between USB and CUSTOMER modes, as this will be useful to run comparative tests between your HW and the EVB.

NB: once this porting is complete and validated, it will be used for all i2c components used on the platform.

HINT: it is highly recommended to validate i2c communication first before any attempt at locking the new hardware.

16.3 Software customization

The Si2183 applications can use various hardware configurations, and the source code is built to be easily adaptable to many hardware designs using the configuration macros, without changing anything in the delivered source code except a couple parameters in the configuration macro.
Refer to the **USING_SKYWORKS_SUPERSET.pdf** document for details about each configuration field, and also refer to the configuration macros in Si2183_L2_API.h the content of the pre-defined macros.

17 Porting the EVB code to the final application

When the i2c and tuner code has been validated on the customer hardware, the last step is to use this code in the final application.

A single file needs to be removed from the final application: SiLabs_API_L3_Console.c.
This is not useful anymore, as it will be replaced by the final application management code.

The final application should call the same functions as the sample application to be able to init and lock the new platform for each standard, following the flowchart provided in 29 Annex A: application flowchart: channel lock.

18 Running automated tests using the source code

It is possible to easily run automated tests on top of the source code in any test environment such as VEEPro, LabView or any other C application running on a Windows machine.

To achieve this, please contact your Skyworks Solutions representative and ask for the C DLL resource code available to make it easy to build a Windows DLL using the source code.

NB: This code is identical for all Skyworks demodulators, so this work will be re-usable for other projects as well.

19 Checking the code configuration using the console application

Use the 'Infos' options and check the displayed lines to make sure the code configuration matches your hardware setup. Pay special attention to the clock input and frequency information.

```
0003 FrontEnd[0] Command > Infos

-------------------------------------
Si_I2C               Source code V3.4.7
Demod                Si2183  at 0xc8
Demod                Source code V0.0.1.0
Terrestrial tuner    Skyworks    at 0xc0
Satellite tuner via SiLabs_SAT_Tuner at 0x14
TER clock from  TER Tuner (24 MHz)
TER clock input CLKIO
FEF mode 'FREEZE PIN'
SAT clock from  SAT Tuner (27 MHz)
SAT clock input XTAL_IN
Compiled with UNICABLE compatibility
UNICABLE LNB not installed
-------------------------------------
Wrapper              Source code V2.4.5
LNB CHIP             LNBH21 at 0x10
LNB CHIP             Source code TAGNAME
-------------------------------------
0004 FrontEnd[0] Command >
```

20 Managing the traces display using the console application

The following example uses the 'r' (reset) option to illustrate the traces management, as this uses a single command.
NB: The SiLabs_API_HW_Connect function is called on every call to Silabs_demoloop. This is to allow lab testing with boards possibly disconnected/reconnected.

20.1 Checking the current traces configuration

```
  0011 FrontEnd[0] Command >  traces
  Si2183\SiLabs_API_L3_Wrapper.c            1554 SiLabs_API_HW_Connect          SiLabs_API_HW_Connect in
mode 1
  Enter the traces configuration string (-<param> ,<value>):  status
   -output file  -name Skyworks_Traces.txt -file  on -line  on -function  on -time off -verbose  on
```

20.2 Checking possible traces options

This is possible using 'traces' then '<enter>'.

```
  0004 FrontEnd[0] Command > traces
Enter the traces configuration string (-<param> ,<value>): help
-------------------------------------------------------------------------------------------
Possible traces commands:
-------------------------------------------------------------------------------------------
traces  -output   <memory/stdout/file/none>. select the trace destination (don't use 'stdout' in wish)
traces  -file     <on/off> . . . . . . . . . select file name   mode (adding C source file name)
traces  -line     <on/off> . . . . . . . . . select line number mode (adding C source file line number)
traces  -function <on/off> . . . . . . . . . select function    mode (adding C source file function
name)                                traces  -time     <on/off> . . . . . . . . . select time tag
mode (adding elapsed time as hh:mm:ss.ms)                                  traces  -tag      <on/off> . .
. . . . . . . select tag         mode (adding user-defined tag, often with frontend index)
traces  -level    <on/off> . . . . . . . . . select level       mode (adding trace level indication)s
traces  -name     <file_name>. . . . . . . . select traces file name (default 'Skyworks_traces.txt' and
'' gives the current file used)     traces  -verbose  <on/off> . . . . . . . . . select verbose mode
(traces in console even if '-output' not 'stdout')                        traces  show . . . . . . . . .
. . . . . . . display buffered traces in console (not visible in wish)
traces  get. . . . . . . . . . . . . . . . . display buffered traces (visible in wish)
traces  count. . . . . . . . . . . . . . . . display number of traces since start
traces  suspend. . . . . . . . . . . . . . . suspend tracing until 'resume'
traces  resume . . . . . . . . . . . . . . . resume  tracing after 'suspend'
traces  lost . . . . . . . . . . . . . . . . display number of traces lost since start
traces  save . . . . . . . . . . . . . . . . save    buffered traces to file
traces  flush. . . . . . . . . . . . . . . . erase   buffered traces
traces  erase. . . . . . . . . . . . . . . . erase   file content
traces  status . . . . . . . . . . . . . . . display the current traces flags
traces  help . . . . . . . . . . . . . . . . display this help
>
```

20.3 Default traces

The default traces configuration is controlled in the main function of the console application:

```
  int  main(int argc, char *argv[])
  {
  . . .
  SiTraceDefaultConfiguration();
  SiTraceConfiguration("traces -output file -file on -verbose on -function on -line on\n");
  . . .
```

20.4 Default traces during a reset:

The following traces show the process used to reset the demod:

```
  0004 FrontEnd[0] Command >  r
  Si2183\SiLabs_API_L3_Wrapper.c            1554 SiLabs_API_HW_Connect          SiLabs_API_HW_Connect in
mode 1
  11\Si2183\Si2183_L1_Commands.c             731 Si2183_L1_DD_RESTART           Si2183 DD_RESTART

  0005 FrontEnd[0] Command >
```

Here the DD_RESTART command is called.
The trace message comes from file Si2183_L1_Commands.c, in function Si2183_L1_DD_RESTART, on line 731. This allows a very accurate tracing of the code execution and can be very useful to compare the code behavior between the GUI, the console application and the customer code.

20.5 Traces without file name, function name or line number

The following screenshot shows the trace configuration to avoid displaying file/function/line and the resulting traces during a reset.

```
  0005 FrontEnd[0] Command >  traces
  Si2183\SiLabs_API_L3_Wrapper.c            1554 SiLabs_API_HW_Connect          SiLabs_API_HW_Connect in
mode 1
  Enter the traces configuration string (-<param> ,<value>):  -file off -function off -line off
   -output file  -name Skyworks_Traces.txt -file off -line off -function off -time off -verbose  on

  0006 FrontEnd[0] Command >  r
  SiLabs_API_HW_Connect in mode 1
  Si2183 DD_RESTART

  0007 FrontEnd[0] Command >
```

20.6 Traces with i2c bytes

It is possible to trace the i2c bytes sent over the i2c bus in addition to the function calls.
The corresponding option is 'trace', which toggles the i2c traces on/off.
The following screenshot shows the option to activate such tracing and the resulting traces still during a reset.

```
  0007 FrontEnd[0] Command >  trace
  SiLabs_API_HW_Connect in mode 1

  0008 FrontEnd[0] Command >  r
  SiLabs_API_HW_Connect in mode 1
  Si2183 DD_RESTART
  writing 0xc8  USB> 0x85
  reading 0xc9  <USB 0x80

  0009 FrontEnd[0] Command >
```

This shows the result of calling Si2183_L1_DD_RESTART: the DD_RESTART command code is sent to the demodulator, then CTS is checked.

20.7 Traces turned off

It is possible to totally disable the traces, to get less lines in the console output.
The following screenshot shows the corresponding option in use, as well as the traces displayed during a reset in this configuration: all traces are disabled.

```
  0009 FrontEnd[0] Command >  traces
  SiLabs_API_HW_Connect in mode 1
  Enter the traces configuration string (-<param> ,<value>):  -output none
   -output none  -name Skyworks_Traces.txt -file off -line off -function off -time off -verbose  on

  0010 FrontEnd[0] Command >  r
```

```
0011 FrontEnd[0] Command >
```

20.8 More details on traces?

More details on traces are available in the L0 documentation, delivered in the Si_I2C 'Doc' folder.

21 Locking on a DVB-T channel using the console application

```
0014 FrontEnd[0] Command >  lock
Standard ( T T2 C S S2 DSS )?  T
tuning frequency (in Hz)?  474166666
Bandwidth (in MHz) (6, 7 or 8)?       8
DVB-T Stream (HP LP)?  hp

Front_end status:

 standard       DVB-T
 frequency         474166666  Hz
 demod_lock        1 : locked
 fec_lock          1 : locked
 characteristics      8K  QAM64  3/4  1/8  8.0 MHz
 ber              0.00e+000
 per              0.00e+000
 c_n                 32.75
 uncorrs            590024
 RFagc                   0
 IFagc                 117
 freq_offset             0
 timing_offset           2
 RSSI             -42.000000
 SSI              100
 SQI              0

0015 FrontEnd[0] Command >
```

22 Locking on a DVB-T2 channel using the console application

```
0017 FrontEnd[0] Command >  lock
Standard ( T T2 C S S2 DSS )?  T2
tuning frequency (in Hz)?  474166666
Bandwidth (in MHz) (1.7, 5, 6, 7 or 8)?  8

/* At the time of writing, no DVB-T2 signal is available */
```

23 Locking on a DVB-C channel using the console application

```
0015 FrontEnd[0] Command >  lock
Standard ( T T2 C S S2 DSS )?  C
tuning frequency (in Hz)?  70000000
Symbol rate (in Msymb/s)? 6.9
     --> SR in baud 6900000
 DVB-C qam (auto(0) qam16(7) qam32(8) qam64(9) qam128(10) qam256(11)?  0

Front_end status:

 standard       DVB-C
 frequency         70000000  Hz
 demod_lock        1 : locked
 fec_lock          1 : locked
```

```
 characteristics     QAM64  6.900 Mbps
 ber              0.00e+000
 per              0.00e+000
 c_n                 35.50
 uncorrs            720909
 RFagc                   0
 IFagc                 181
 freq_offset             2
 timing_offset          34
 RSSI             -38.000000
 SSI              0
 SQI              0

0016 FrontEnd[0] Command >
```

24 Locking on a DVB-S channel using the console application

```
0010 FrontEnd[0] Command >  lock
Standard ( T T2 C S S2 DSS )?  S
tuning frequency (in kHz)?  1027000
Symbol rate (in Msymb/s)? 22
     --> SR in baud 22000000

Front_end status:

 standard       DVB-S
 frequency         1027000 kHz
 demod_lock        1 : locked
 fec_lock          1 : locked
 characteristics     QPSK  22.000 Mbps  5/6
 ber              0.00e+000
 per              0.00e+000
 c_n                 13.50
 uncorrs               140
 RFagc                   0
 IFagc                 167
 freq_offset           -79
 timing_offset         -16
 RSSI             0.000000
 SSI              0
 SQI              0

0011 FrontEnd[0] Command >
```

25 Locking on a DVB-S2 channel using the console application

```
0013 FrontEnd[0] Command >  lock
Standard ( T T2 C S S2 DSS )?  S2
tuning frequency (in kHz)?  1569000
Symbol rate (in Msymb/s)? 27.5
     --> SR in baud 27500000

Front_end status:

 standard       DVB-S2
 frequency         1569000 kHz
 demod_lock        1 : locked
 fec_lock          1 : locked
 characteristics     8PSK  27.500 Mbps  2/3  ON
 fer              0.00e+000
 per              0.00e+000
 c_n                  0.00
 uncorrs                 1
 RFagc                   0
 IFagc                 232
 freq_offset          -543
```

```
  timing_offset         -17
  RSSI              0.000000
  SSI               0
  SQI               0

0014 FrontEnd[0] Command >
```

26 DVB-T/T2 blindscan using the console application

```
0019 FrontEnd[0] Command >  scan
Standard ( T T2 C S S2 DSS )?  T
Seek Start tuning frequency (in Hz)?  474000000
Seek Stop  tuning frequency (in Hz)?  850000000
Bandwidth (in MHz) (6, 7 or 8)?       8
Seek Step (in Hz)?                    8000000
Channel detected.
Carrier   0: DVB-T   474167000   Hz  8.0 MHz, LP
Use the status functions to check the channel characteristics
Call SeekNext to search the next channel
Channel detected.
Carrier   1: DVB-T   498167000   Hz  8.0 MHz, LP
Use the status functions to check the channel characteristics
Call SeekNext to search the next channel
Channel detected.
Carrier   2: DVB-T   522167000   Hz  8.0 MHz, LP
Use the status functions to check the channel characteristics
Call SeekNext to search the next channel
Channel detected.
Carrier   3: DVB-T   602167000   Hz  8.0 MHz, LP
Use the status functions to check the channel characteristics
Call SeekNext to search the next channel
Channel detected.
Carrier   4: DVB-T   626167000   Hz  8.0 MHz, LP
Use the status functions to check the channel characteristics
Call SeekNext to search the next channel
Channel detected.
Carrier   5: DVB-T   698167000   Hz  8.0 MHz, LP
Use the status functions to check the channel characteristics
Call SeekNext to search the next channel

   Seek complete  now call SeekEnd to finish properly the scan
Carrier   0: DVB-T   474167000   Hz  8.0 MHz, LP
Carrier   1: DVB-T   498167000   Hz  8.0 MHz, LP
Carrier   2: DVB-T   522167000   Hz  8.0 MHz, LP
Carrier   3: DVB-T   602167000   Hz  8.0 MHz, LP
Carrier   4: DVB-T   626167000   Hz  8.0 MHz, LP
Carrier   5: DVB-T   698167000   Hz  8.0 MHz, LP

0020 FrontEnd[0] Command >
```

27 DVB-C blindscan using the console application

```
0005 FrontEnd[0] Command >  scan
Standard ( T T2 C S S2 DSS )?  C
Seek Start tuning frequency (in Hz)?  60000000
Seek Stop  tuning frequency (in Hz)?  200000000
Min Symbol rate (in Msymb/s)? 2
     --> SR in baud 2000000
 Max Symbol rate (in Msymb/s)? 7.5
     --> SR in baud 7500000
 Channel detected.
Carrier   0: DVB-C    69992000   Hz  6.90 Mbps QAM64
Use the status functions to check the channel characteristics
Call SeekNext to search the next channel
```

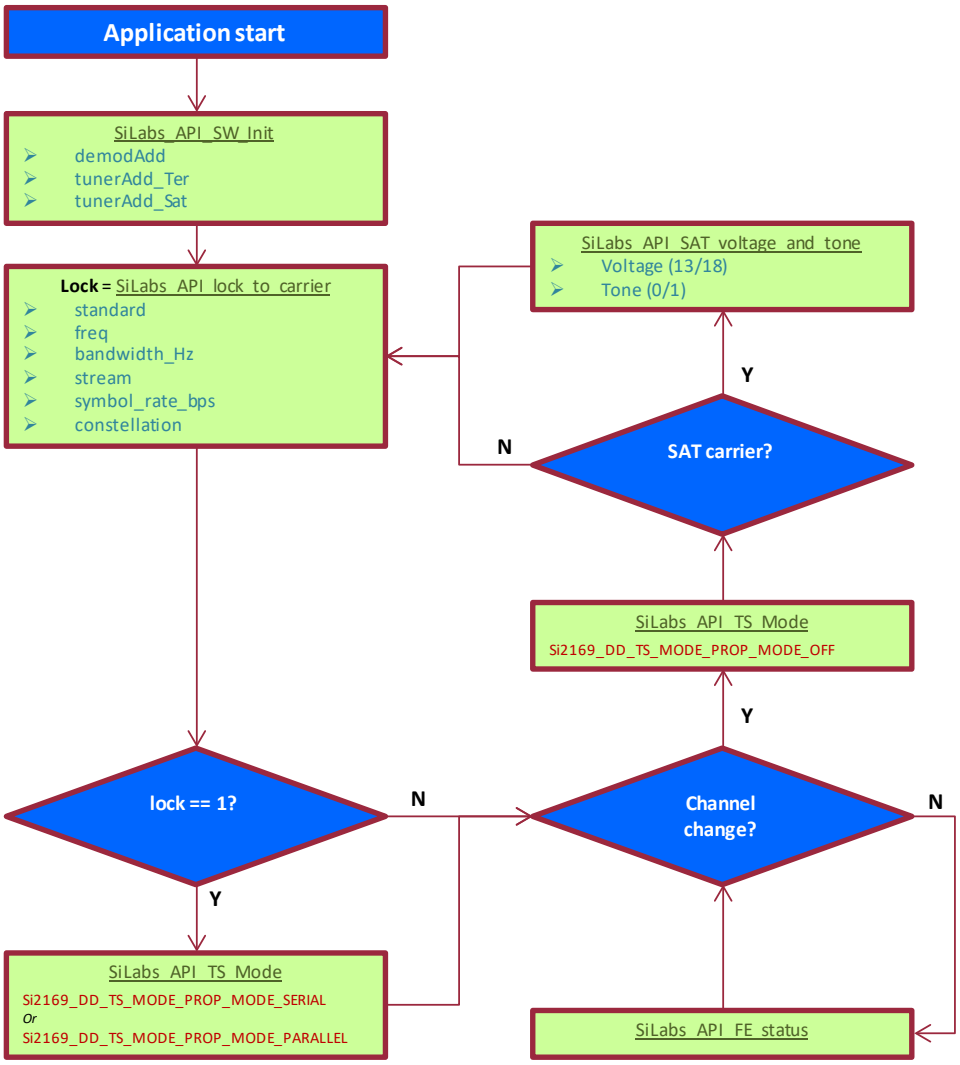
```
     Seek complete  now call SeekEnd to finish properly the scan
  Carrier   0: DVB-C    69992000   Hz  6.90 Mbps QAM64

  0006 FrontEnd[0] Command >
```

28 DVB-S/S2 blindscan using the console application

```
0030 FrontEnd[0] Command >  scan
Standard ( T T2 C S S2 DSS )?  S
Seek Start tuning frequency (in kHz)?  950000
Seek Stop  tuning frequency (in kHz)?  2150000
Min Symbol rate (in Msymb/s)? 4
     --> SR in baud 4000000
 Max Symbol rate (in Msymb/s)? 45
     --> SR in baud 45000000
 Channel detected.
Carrier   0: DVB-S      997392 kHz 22.00 Mbps
Use the status functions to check the channel characteristics
Call SeekNext to search the next channel
Channel detected.
Carrier   1: DVB-S     1026883 kHz 22.00 Mbps
Use the status functions to check the channel characteristics

. . .

Channel detected.
Carrier  21: DVB-S     2040463 kHz 22.00 Mbps
Use the status functions to check the channel characteristics
Call SeekNext to search the next channel
Channel detected.
Carrier  22: DVB-S     2099445 kHz 22.00 Mbps
Use the status functions to check the channel characteristics
Call SeekNext to search the next channel

   Seek complete  now call SeekEnd to finish properly the scan
Carrier   0: DVB-S      997392 kHz 22.00 Mbps
Carrier   1: DVB-S     1026883 kHz 22.00 Mbps
Carrier   2: DVB-S     1085931 kHz 22.00 Mbps
Carrier   3: DVB-S     1139408 kHz 27.50 Mbps
Carrier   4: DVB-S     1178664 kHz 27.50 Mbps
Carrier   5: DVB-S     1217396 kHz 27.50 Mbps
Carrier   6: DVB-S     1295384 kHz 27.50 Mbps
Carrier   7: DVB-S     1334378 kHz 27.50 Mbps
Carrier   8: DVB-S     1373437 kHz 27.50 Mbps
Carrier   9: DVB-S2    1412431 kHz 27.50 Mbps
Carrier  10: DVB-S     1529413 kHz 27.50 Mbps
Carrier  11: DVB-S2    1568407 kHz 27.50 Mbps
Carrier  12: DVB-S2    1607401 kHz 27.50 Mbps
Carrier  13: DVB-S     1646460 kHz 27.50 Mbps
Carrier  14: DVB-S     1724448 kHz 27.50 Mbps
Carrier  15: DVB-S     1763442 kHz 27.50 Mbps
Carrier  16: DVB-S     1802436 kHz 27.50 Mbps
Carrier  17: DVB-S2    1841430 kHz 29.70 Mbps
Carrier  18: DVB-S     1880424 kHz 27.50 Mbps
Carrier  19: DVB-S     1922433 kHz 22.00 Mbps
Carrier  20: DVB-S     1981415 kHz 22.00 Mbps
Carrier  21: DVB-S     2040463 kHz 22.00 Mbps
Carrier  22: DVB-S     2099445 kHz 22.00 Mbps

0031 FrontEnd[0] Command
```

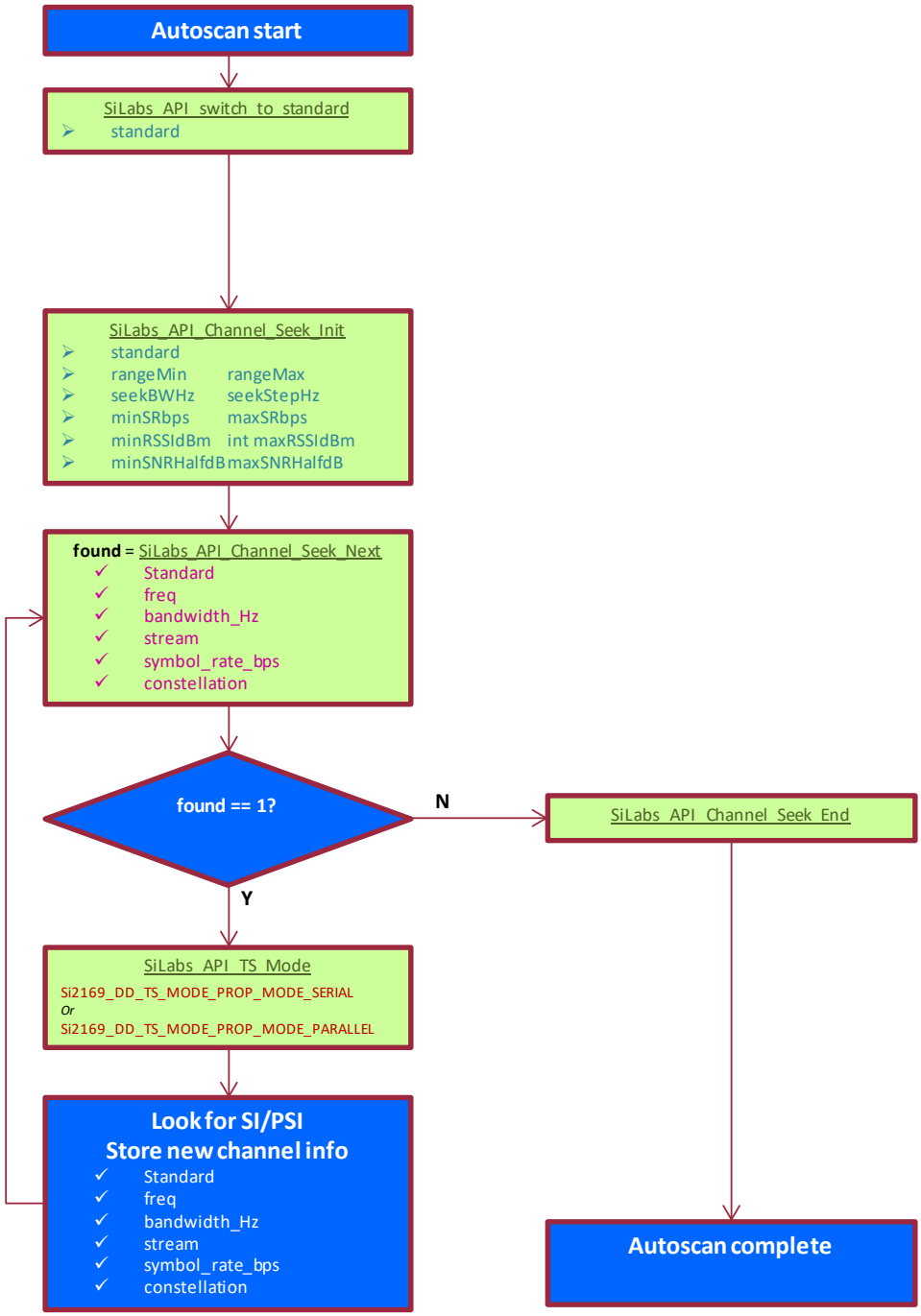
29 Annex A: application flowchart: channel lock

The diagram below illustrates the top-level application flowchart when locking on any DTV channel.



30 Annex B: Si2183 TER auto-scan flowchart (DVB-T/T2, DVB-C)

The diagram below illustrates the top-level auto-scan flowchart in TER. The only difference with SAT auto-scan is that there is no LNB control so the loop is used once for DVB-T/T2 and once for DVB-C.



31 Annex C: Si2183 SAT blind-scan flowchart (DVB-S/S2)

The diagram below illustrates the top-level blind-scan flowchart in SAT. Compared with the TER auto-scan there is then need for LNB control so the loop is used 4 times for DVB-S/S2.