Using the SKYWORKS SUPERSET source code

1 Introduction

This document provides information on the SKYWORKS SUPERSET source code used to build DTV frontend applications using Skyworks DTV demodulators.

This code can be used to support any configuration using the API-controlled Skyworks DTV demodulators.
It's the recommended code for all future development efforts.

Change log

Rev 1.0 (2015/10/15) Initial version.

Rev 1.1 (2015/11/06) Adding software configuration functions. First release to customers.

Rev 1.2 (2015/11/12) Correcting typos. Adding software architecture view and basic flowcharts for locking & scanning.

Rev 1.3 (2015/11/16) Correcting references for Si2166/Si2167 in 'Part compatibility compilation flags'

Rev 1.4 (2015/12/03) Adding info on SiLabs_API_TS_Config function (as from SUPERSET V0.2.4.0/ wrapper 2.6.8)

Rev 1.5 (2016/03/03) Adding Allegro A8297 info

Rev 1.6 (2016/06/30) Adding x63 parts. Adding info on restrictions when using DUAL/TRIPLE/QUAD.

Rev 2.0 (2016/10/04) General rework to add much more details. Merging several separate docs.

Rev 2.1 (2017/03/28) Adding INDIRECT_I2C_CONNECTION details

Rev 2.2 (2017/03/30) Adding RDA16110E in 'Possible SAT tuners'

Rev 2.3 (2017/05/19) Adding TPS65233 in 'Possible SAT LNB Controllers'. Improving text in clock configuration functions.

Rev 2.4 (2018/05/15) Removing typo in chapter 7.3

Rev 2.5 (2019/06/24) Updating information on RDA16110E

Table of Contents

Annexes

1.1 Pros and cons of the SKYWORKS SUPERSET

1.1.1 Pros

It uses a single code base
- This is good for maintenance
  - Maintaining a single code is way easier than dealing with 18+ different codes
  - Improvements and new features will be much more easily added in a single code branch
- Everybody will be referring to the same function names
- *Details on the code used are provided in this document*

It supports single/dual/triple/quad configurations
- A single compilation flag controls the number of frontends
- *This is also the case with part-specific codes.*

It can be compiled for configurations where not all standards supported by a given part are required
- For instance, Skyworks ISDB-T capable parts also support DVB-T. The code can nevertheless be compiled only for ISDB-T support, not compiling the DVB-T code.
- *This was not possible with the part-specific codes.*

The same top-level API is used
- This is good for middleware adaptation.
- Once the work is completed once, very few changes are expected.
- *This is also the case with part-specific codes.*

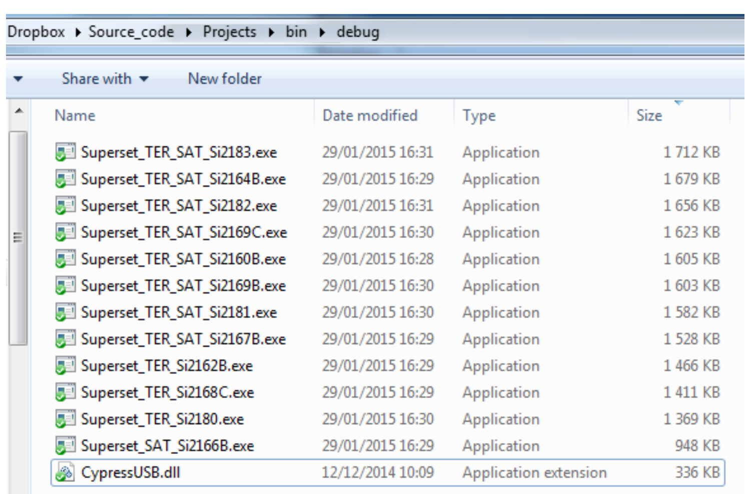
It's future-proof
- We'll keep building all our future code based on this type of approach.
- In the coming years, the same code base will still be in use.
- Moving from one part to another is trivial, even when more standards are supported by the newer part.
- Future tuners will be added when needed.

It's easy to compile the same code for various projects
- In the lab and for SW development, using Si2183 parts and allowing all standards allows testing the Skyworks DTV demodulators to their full extent, while the production code will be limited to the actual part used in production.
- Selecting the right compilation options is described later in this document

It's still as easy as 'pick and place' to add/remove a tuner
- Many SAT and TER tuners are currently supported
- Additional tuners will be added when needed, without required changes to the top-level API
- *This is also the case with part-specific codes.*

Building applications compatible with multiple HW is easy
- It uses configuration macros to match any HW
- Customers can define their own macros, named after their HW platforms.

It's efficient in terms of compiled code size
- Selecting different setting you get multiple applications fine-tuned to specific HW in minutes.
- As you can see in the snapshot on the right, the executable size varies according to the supported standards



1.1.2 Cons

(Counter arguments related to the points below are provided in *italic*)

The L1 and L2 function names will not match the part name

- No customer will actually buy Si2183, since it's an overkill for most applications.
- *On the other hand, the L3 function names are still the same, the middleware adaptation layer only uses calls to the L3, and so the underlying naming scheme is not critical…*

The source code is bigger

- For instance, ISDB-T code will be in even this standard is not supported by the HW
- *On the other hand, the compiled code will NOT contain the non-selected standards, and this is the important part as far as the final code size is concerned.*

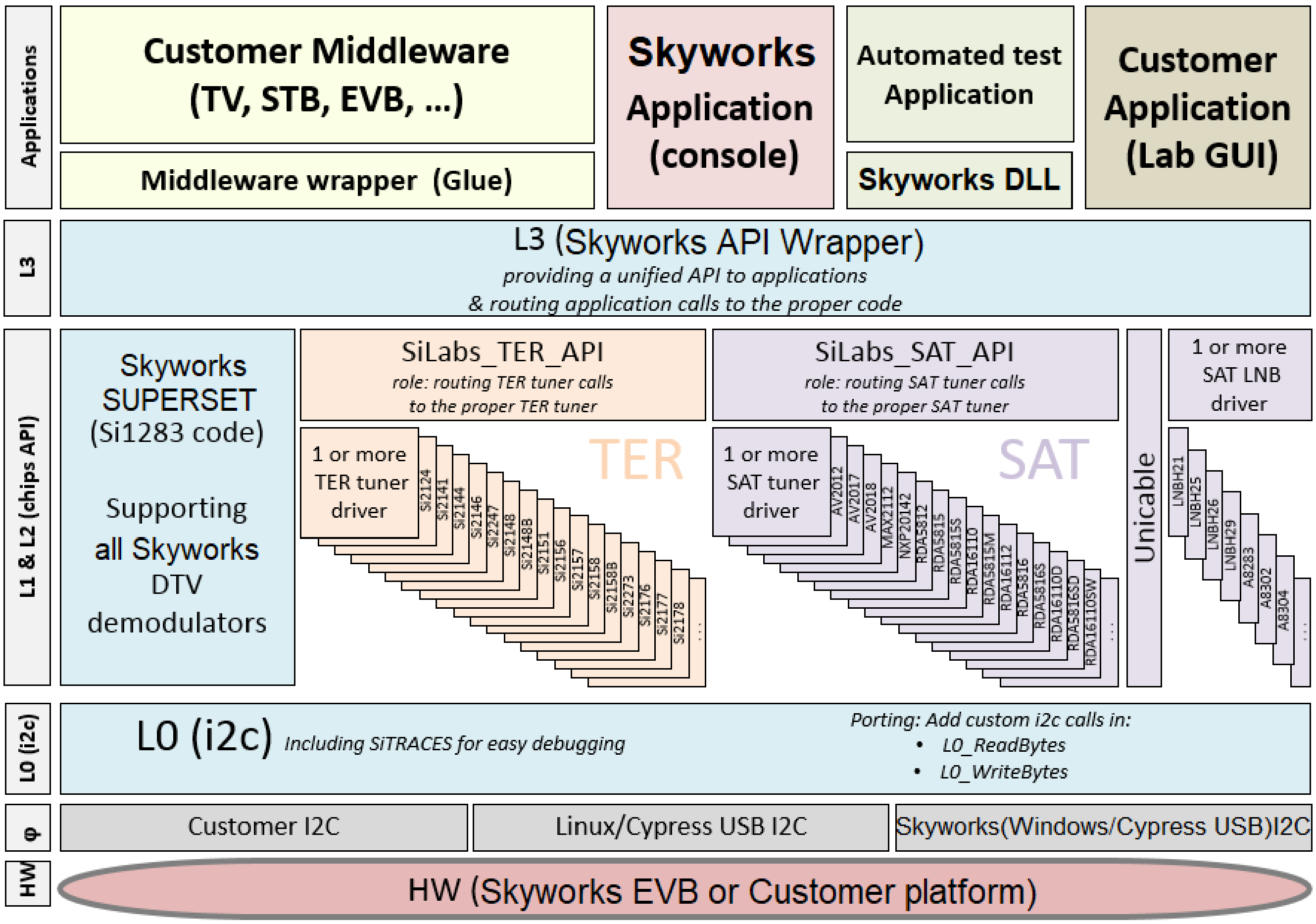
The source code contains lots of '#ifdef/#endif' lines

- To enable the versatility in the SKYWORKS SUPERSET, each standard-specific code is surrounded by tags to get it compiled or not. There are quite a lot of these lines.
- *Sure. But the added ease of use can't come without a cost. These tag-lines can also be helpful in identifying standard-specific code, since they contain comments to identify the tags.*

1.1.3 Conclusion

From a software development and maintenance point of view, the pros largely offset the cons, and the SKYWORKS SUPERSET is the best path to follow.

1.2 Software architecture (full)



This code architecture allows easily creation of any application using the Skyworks DTV demodulators.

- The main API interface is at L3 level.
- To fit any particular middleware implementation a 'middleware wrapper' can be added on top of the L3. Its role is to translate middleware calls into L3 calls and fill the middleware status parameters with values set by the L3 code. This porting work is specific to each middleware, and is done only once. When it exists, it will be re-usable for future platforms. If new standards are required, it will be upgraded.
- TER and SAT applications use the same L3 API.
- This architecture supports SAT-only, TER-only and SAT+TER applications.
- This architecture supports from 1 to 4 frontends.
- Any number of TER tuners can be used in the application.
- Any number of SAT tuners can be used in the application.
- Any number of LNB controllers can be used in the application.
- A single application can be used in the lab to stay compatible with many platforms over time.
- The 'release' version to be loaded on the final target is only compiled (using the same code) for the minimum set of parts, to keep its size as small as possible.

1.3 Source of the Skyworks DTV demodulator code

The source for all Skyworks DTV demodulator code is our FTP server (https://webftp.skyworksinc.com).
The various items required to build an application based on this code are available in subfolders of the FTP server.

1.4 Accessing the source code folders

Access to these FTP folders is granted to customers using Skyworks Solutions DTV demodulators upon request to the applications team.
Each FTP subfolder has a dedicated login information, made of a username and password pair. The login information has no expiration date, such that customers can regularly check the content of the folders to check for any update.

1.5 Software version check

When providing information on the application to Skyworks for debug, it's recommended to provide

- The version of the Si2183 code (check Si2183_L1_API_TAG_TEXT)
- The version of the L3 Wrapper code (check SiLabs_API_TAG_TEXT)
- The PART_INFO of the DTV demodulator (see the line with 'Full Info' in the Si2183_PowerUpWithPatch traces)
- The GET_REV information on the firmware version loaded in the DTV demodulator (see the line with 'Part running' in the Si2183_PowerUpWithPatch traces)

1.6 SKYWORKS SUPERSET code = Si2183 code

The SKYWORKS SUPERSET code is based on the Si2183 source code.
The reason for this is that the Si2183 has the most complete FW API, and this FW API is compatible with all other API-controlled Skyworks demodulators.

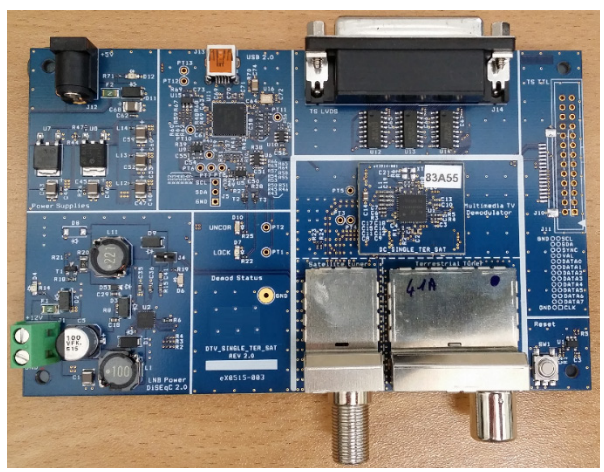
1.7 Example used in this document

In the present document, screenshots and references will be made to a SAT+TER configuration based on:

- SAT Tuner: AV2018
- SAT LNB Controller: LNBH25
- SAT Unicable
- TER Tuner: Si2141
- Si2183
- I2C

This is the configuration required to drive the most common Skyworks EVB as of writing:

DTV_SINGLE_TER_SAT_Rev2_0



This EVB fits most development efforts, since it can support SAT and TER reception and features a mother board with the I2c and Tuner parts with the demodulator on a daughter card.
The daughter card can be replaced in seconds to change the demodulator.
When fitted with Si2183, this EVB can get the most out of the Skyworks DTV reception capabilities.

Since it's more convenient to start from a full-featured project and then reduce its capabilities than adding features, this can be a good start point for any development work.

The example project builds the Skyworks console application under Windows. This application is compatible with all current Skyworks EVBs, depending on the compilation flags used.

Changing the TER tuner, SAT tuner and LNB controller in this project can make the application compatible with most Skyworks EVBs.

2 Development scenarios

2.1 SW development using a Skyworks Labs EVB

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

This application should be kept available at all times, for comparison purposes with the final HW and SW. Using the Skyworks traces it's easy to compare the behavior of different code versions.

Since it's easy to change the demodulator, the TER tuner, the SAT tuner and the SAT LNB controller using the SKYWORKS SUPERSET code, SW development can start very early in the project using the SKYWORKS SUPERSET code and the Skyworks EVB.

The best solution for customers willing to evaluate and develop applications for several standards is to request a DTV-SINGLE_TER_SAT_EVB with a daughter card fitted with the part covering all the requested standards (Si2183 being able to deal with all standards).

Using this EVB it's possible to prepare the SW in such a way that it will be validated when the final HW will be available.

There is no need to wait for customer HW to be developed before starting working on SW.

Once this is complete, only i2c and porting issues on the final platform should potentially occur. The console application will be usable on both platforms.

It's also possible to connect the Skyworks EVB I2c bus to the final platform's I2C to work on the I2C porting, still before the final HW is available. This bypasses the Cypress Fx2LP part.

When possible (at least in TS serial mode) it may be interesting to interconnect the I2C bus as well and get pictures with the SoC platform connected to the Skyworks EVB. In these conditions, the SW work could be 95% complete without custom HW availability. The remaining changes would be related to part changes and SW configuration, and this is no big deal.

2.1.1 Compatibility of SW application for lab use

The SW development application used in the lab can be kept compatible with several demodulator versions as well as with several HW. This enables managing a single code in the lab to cope with multiple designs.

2.1.2 Automated testing in the lab

A dedicated 'DLL' source code file is available (upon request to your Skyworks representative) to replace the console application by a Windows DLL.

This enables the developers to use any C-compatible automated test system (such as LabView, VeePro, etc.) to control the HW in a similar way to the console application.

It's therefore possible to run long and comprehensive test scenarios automatically, to test all areas of complex standards such as DVB-T and DVB-T2.

2.2 Using the console application to drive custom HW

It's possible to disconnect the Cypress FX2LP's I2C bus present on the Skyworks EVB from the rest of the EVB to hook it to customer HW, in order to use the existing console application to validate the HW without then need to have the SoC porting complete.

In this case, it may not be so easy to get pictures, but locking the new HW in all standards and checking the performance is possible, which is good for HW developers.
If automated tests have been set up on top of the Skyworks EVB they can still be used in this case to validate the new HW.

3 Typical application code

The typical application will be aiming at terrestrial (TER) reception, satellite (SAT) reception or both (TER+SAT).
Based on this, the code will come from various folders, as described below.

3.1 I2C code

In all cases, I2C communication with the parts needs to be allowed. This is achieved using the code in the si_i2c FTP folder (https://webftp.skyworksinc.com with username 'si_i2c').
There is only a single instance of the I2C code, used by all parts.
The I2C code is the 'Layer 0' (L0) code in DTV applications.
This I2C code can be used to drive any I2C control chip, and doesn't need to be duplicated for each part.
Each part will use its own L0 context, thus separating the settings from other parts.

The I2C code also provides tracing capability. A set of tracing functions enable the coders to follow and debug the application behavior based on clear text traces.
Tracing is an essential tool during application development, and needs to be properly implemented for easy development work.

3.2 TER code

The terrestrial (TER) code controls the TER tuner.

Each TER tuner has its own FTP folder (https://webftp.skyworksinc.com with the username being the TER tuner name in lowercase (i.e. 'si2151' for Si2151).

Customers need a way to easily select (and possibly change) the TER tuner they will use on their platforms. To make this easy, a wrapper layer has been implemented on top of the existing TER tuner drivers to allow an easy selection of any TER tuner. This code can handle all Skyworks Solutions DTV tuners.

The TER tuner wrapper is in a separate FTP folder (https://webftp.skyworksinc.com with username 'ter_tuner_wrapper').

3.3 SAT Code

The satellite (SAT) code controls the SAT tuner as well as the LNB controllers and optionally supports Unicable (I and II).

3.3.1 SAT tuners

Each SAT tuner has its own FTP folder (https://webftp.skyworksinc.com with the username being either the SAT tuner name in lowercase (i.e. 'rda5812' for RDA5812) or the supplier name for the recent SAT tuners.

Customers need a way to easily select (and possibly change) the SAT tuner they will use on their platforms. To make this easy, a wrapper layer has been implemented on top of the existing SAT tuner drivers to allow an easy selection of any SAT tuner.

The SAT tuner wrapper is in a separate FTP folder (https://webftp.skyworksinc.com with username 'sat_tuner_wrapper').

3.3.2 SAT LNB controllers

Each SAT LNB controller has different settings and ways to be controlled, so the L3 wrapper (which we discuss below) supports at the time of writing 9 different SAT LNB controllers.

3.3.3 SAT Unicable

SAT reception usually requires Unicable support as well, for independent reception of SAT signals on a single cable by up to 8 (Unicable I) or up to 32 (Unicable II) receivers.

The SAT Unicable code (compatible with Unicable I and II) is in a separate FTP folder (https://webftp.skyworksinc.com with username 'unicable').

3.4 Si2183/SUPERSET code

All demodulators are supported using a single code, named ‘SKYWORKS SUPERSET’.

The code used as the SKYWORKS SUPERSET is the Si2183 code because Si2183 supports all features possible with Skyworks Solutions DTV demodulators. The code is scalable at compilation time using compilation flags.

The SKYWORKS SUPERSET can be compiled for TER-only, SAT-only or TER+SAT applications.
The SKYWORKS SUPERSET can be compiled for a limited set of standards depending on the application’s requirements.

Customer applications are built on top of the SKYWORKS SUPERSET L3 layer, via a glue layer between an existing middleware or with direct access to L3 for new applications.

4 SW porting on final platform

SW porting consist in making I2C communication possible then configuring the code to match the HW.
I2C porting occurs at L0 level, while SW configuration uses the L3 functions. The application can be used without any change in the L1 and L2 layers.

4.1.1 I2C porting

Depending on the platform, several options are possible to achieve i2c communication.
Check the L0 documentation for detail on the I2C communication possibilities.
The code is prepared for the following use cases.

- Windows I2C over USB using the Cypress Fx2LP part
  - This is the default configuration, the one used with Skyworks EVBs.
- Windows I2C over the parallel port
  - This is a legacy mode, using the parallel port as an i2c interface. It relies on Philips drivers.
- Linux I2C over USB on a Linux PC, using the Cypress Fx2LP part
  - This mode can be used to connect a Skyworks EVB to a Linux PC
- Linux kernel I2C in STM SDK2
  - This is used when using the SKYWORKS SUPERSET code together with STM SDK2 distribution, with STM SOCs.
- Custom I2C (for all cases not listed above)
  - The SW developer will need to adapt the ‘CUSTOMER’ case inside L0_WriteBytes and L0_ReadBytes to access the platform’s i2c functions.

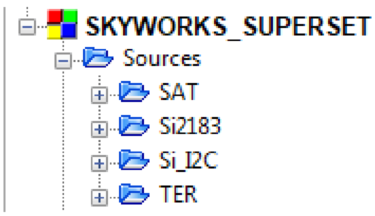
4.1.2 SW configuration

Because the final HW may differ from the Skyworks EVB, it’s required to:
- Add/Replace the TER tuner, SAT tuner and SAT LNB codes, if different from the EVB HW.
  - Adding codes is recommended for the lab test application, to keep the compatibility with the Skyworks EVB.
- Build with different compilation flags
  - To match the selected parts
  - To match the selected standards
  - To be compatible with different demodulators
- Create a new configuration macro for the new HW
  - It’s recommended to create one configuration macro named after the customer HW, such that the same code can support several configurations.

The following paragraphs provide details on choosing the code, selecting compilation flags and filling the configuration macros.

5 Source code selection

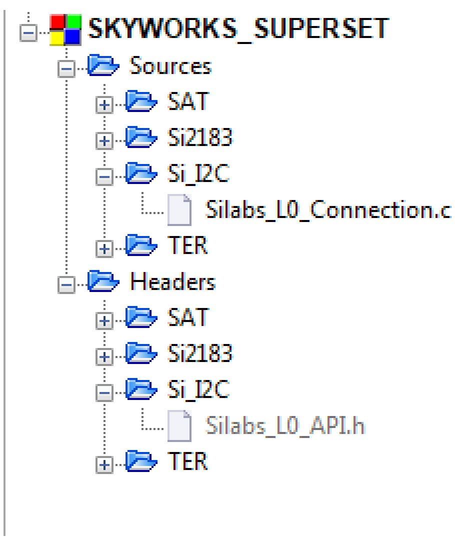
The source code tree is separated in 4 main parts:



Some parts are mandatory: Si2183 and Si_I2C
Others are only required depending on the application: SAT and TER
The following paragraphs provide additional information about all 4 parts.

5.1 I2C

Since sending a message over I2C is a generic process, there is a single copy of the I2C code to send messages to the DTV demodulator, the SAT and TER tuners and the possible LNB controller (for SAT applications).
All I2C messages are routed through this code (using different instances of the L0_Context).
Therefore, porting I2C is done by adding the necessary calls in the L0_WriteBytes and L0_readBytes functions once and for all.



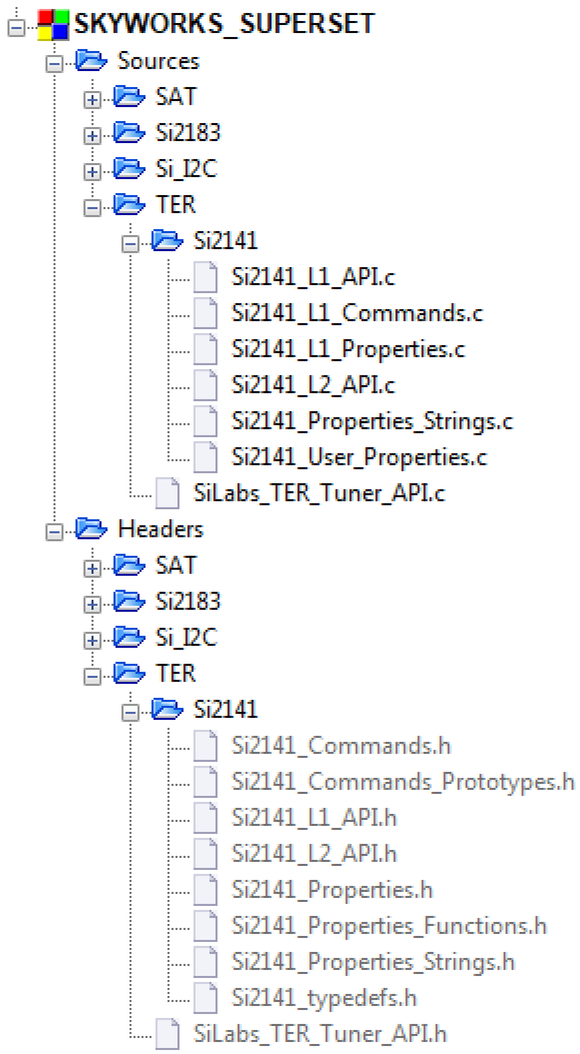
The Si2183 tuner wrapper code is retrieved from a dedicated FTP folder. It's a mandatory item.

NB: Links to the FTP folders are provided to customers with a valid NDA upon request to their Skyworks representative.

Once the project has been filled up with the required codes, it's time to select which standards will be supported.

5.2 TER

The source code for TER (Terrestrial reception) uses a 'TER tuner wrapper' layer to access the selected TER tuner.
It routes all calls to the TER tuner driver selected during SW configuration.



The TER tuner wrapper code is retrieved from a dedicated FTP folder. It's a mandatory item.
The TER tuner code is retrieved from a tuner-specific FTP folder. It depends on the TER tuner.

To use the TER code, 2 compilation flags are required:

- To build with the TER tuner wrapper code
  -D***TER_TUNER_SILABS***
- To select the TER tuner (depending on the TER tuner, here for Si2141)
  - Check 6.2.1 Possible TER tuners for all TER tuner possibilities

  -D***TER_TUNER_Si2141***

NB: Links to the FTP folders are provided to customers with a valid NDA upon request to their Skyworks representative.

Hint: It's possible to build projects using several TER tuners. This can be useful in the lab to keep using a single application for various HW platforms.

5.2.1 Possible TER tuners

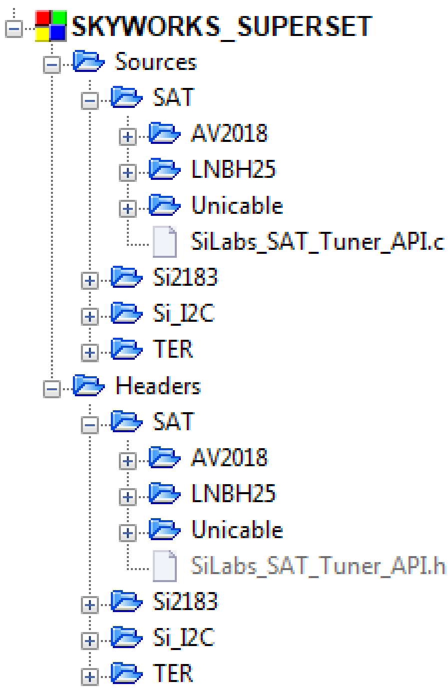
The available TER tuner flags are listed below, with the corresponding values used in the call to SiLabs_API_Select_TER_Tuner().

NB: You can declare several TER tuners and select the proper one at runtime using SiLabs_API_Select_TER_Tuner (front_end, ter_tuner_code, 0);

Part Number (single) / (dual)	TER Tuner flag	**SiLabs_API_Select_TER_Tuner** ter_tuner_code
Si2124	-D***TER_TUNER_Si2124***	0x2124
Si2141	-D***TER_TUNER_Si2141***	0x2141
Si2144	-D***TER_TUNER_Si2144***	0x2144
Si2146	-D***TER_TUNER_Si2146***	0x2146
Si2147	-D***TER_TUNER_Si2147***	0x2147
Si2148	-D***TER_TUNER_Si2148***	0x2148
Si2148B	-D***TER_TUNER_Si2148B***	0x2148B
Si2151	-D***TER_TUNER_Si2151***	0x2151
Si2156	-D***TER_TUNER_Si2156***	0x2156
Si2157	-D***TER_TUNER_Si2157***	0x2157
Si2158	-D***TER_TUNER_Si2158***	0x2158
Si2158B	-D***TER_TUNER_Si2158B***	0x2158B
Si2173	-D***TER_TUNER_Si2173***	0x2173
Si2176	-D***TER_TUNER_Si2176***	0x2176
Si2177	-D***TER_TUNER_Si2177***	0x2177
Si2178	-D***TER_TUNER_Si2178***	0x2178
Si2178B	-D***TER_TUNER_Si2178B***	0x2178B
Si2190	-D***TER_TUNER_Si2190***	0x2190
Si2191	-D***TER_TUNER_Si2191***	0x2191
Si2191B	-D***TER_TUNER_Si2191B***	0x2191B
Si2196	-D***TER_TUNER_Si2196***	0x2196

5.3 SAT

The source code for SAT (Satellite reception) uses a 'SAT tuner wrapper' layer to access the selected SAT tuner and LNB controller.
It routes all calls to the SAT tuner driver and SAT LNB controller selected during SW configuration, as well as to the SAT Unicable code if required.



The SAT tuner wrapper code is retrieved from a dedicated FTP folder. It's a mandatory item.
The SAT tuner code is retrieved from a tuner-specific (or tuner vendor specific) FTP folder. It depends on the SAT tuner.
The SAT LNB controller code is retrieved from an LNB-specific (or tuner vendor specific) FTP folder. It depends on the LNB part.
The SAT Unicable code is retrieved from a dedicated FTP folder. It's required if Unicable support is needed.

To use the SAT code, 3 compilation flags are required:

- To build with the SAT tuner wrapper code
  -D***SAT_TUNER_SILABS***
- To select the SAT tuner (depending on the SAT tuner, here for AV2018)
  - *Check 6.3.1* Possible SAT tuners *for all SAT tuner possibilities.*

  -D***SAT_TUNER_AV2018***
- To select the LNB controller (depending on the SAT LNB controller, here for LNBH25).
  - Check 6.3.2 Possible SAT LNB controllers for all LNB controller possibilities

  -D***LNBH25_COMPATIBLE***

To use the SAT Unicable code, a compilation flag is required: -D***UNICABLE_COMPATIBLE***
When using the SAT Unicable code, 1 additional compilation flag is required if compatibility with Unicable II is required: -D***UNICABLE_II_COMPATIBLE***

NB: Links to the FTP folders are provided to customers with a valid NDA upon request to their Skyworks representative.

Hint: It's possible to build projects using several SAT tuners and/or several LNB controllers. This can be useful in the lab to keep using a single application for various HW platforms.

5.3.1 Possible SAT tuners

The available SAT tuner flags are listed below, with the corresponding values used in the call to SiLabs_API_Select_SAT_Tuner().

NB: You can declare several SAT tuners and select the proper one at runtime using
SiLabs_API_Select_SAT_Tuner (front_end, sat_tuner_code, 0);

Supplier	Part Number	SAT Tuner flag	**SiLabs_API_Select_SAT_Tuner** sat_tuner_code
Airoha	AV2012 AV2017* AV2018	-D***SAT_TUNER_AV2012*** -D***SAT_TUNER_AV2018*** -D***SAT_TUNER_AV2018***	0xA2012 0xA2017 0xA2018
Maxim	MAX2112	-D***SAT_TUNER_MAX2112***	0x2112
NXP	NXP20142	-D***SAT_TUNER_NXP20142***	0x20142
RDA	RDA5812 RDA5815 RDA5815S* RDA16110* RDA5815M RDA16112* RDA5816* RDA5816S RDA16110D* RDA16110E RDA5816SD RDA16110SW*	-D***SAT_TUNER_RDA5812*** -D***SAT_TUNER_RDA5815*** -D***SAT_TUNER_RDA5815*** -D***SAT_TUNER_RDA5815*** -D***SAT_TUNER_RDA5815M*** -D***SAT_TUNER_RDA5815M*** -D***SAT_TUNER_RDA5816S*** -D***SAT_TUNER_RDA5816S*** -D**SAT_TUNER_*****RDA5816S*** -D***SAT_TUNER_RDA16110E*** -D***SAT_TUNER_RDA5816SD*** -D***SAT_TUNER_RDA5816SD***	0x5812 0x5815 0x5815 0x5815 0x58150 0x58150 0x5816 0x5816 0x5816 0x16110E 0x58165D 0x58165D
Custom	*Any*	-D**SAT_TUNER_CUSTOMSAT**	SAT_TUNER_CUSTOMSAT_CODE (defined in Silabs_L1_RF_CUSTOMSAT_API.h)

NB: Please note that some Airoha tuners share a common driver, as well as some RDA tuners.

AV2017 is a specific case where the driver can be identical to the AV2018 driver, but the IQ output needs to be set differently. This is handled by the SAT tuner wrapper if you use the AV2018 driver but use '0xA2017' as the sat_tuner_code.

5.3.2 Possible SAT LNB controllers

SATELLITE_FRONT_END applications needs to drive an LNB control part.

You need to declare specific compilation flags to compile the corresponding code.

The available SAT LNB controller flags are listed below, with the corresponding values used in the call to SiLabs_API_SAT_Select_LNB_Chip().

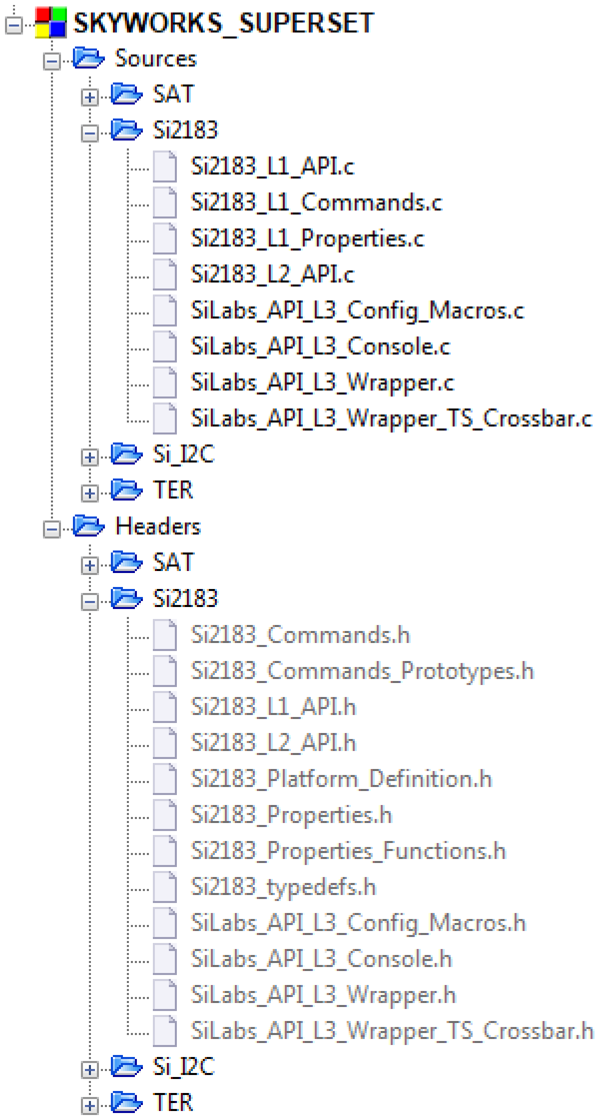
NB: You can declare several LNB controllers and select the proper one at runtime using ***SiLabs_API_SAT_Select_LNB_Chip (front_end, lnb_code, 0);***

Supplier	Part Number	SAT LNB controller flag	**SiLabs_API_SAT_Select_LNB_Chip** lnb_code
ST	LNBH21 LNBH25 LNBH26 LNBH29	-D***LNBH21_COMPATIBLE*** -D***LNBH25_COMPATIBLE*** -D***LNBH26_COMPATIBLE*** -D***LNBH29_COMPATIBLE***	21 25 26 29
Allegro	A8293* A8297* A8302* A8304*	-D***A8293_COMPATIBLE*** -D***A8297_COMPATIBLE*** -D***A8302_COMPATIBLE*** -D***A8304_COMPATIBLE***	0xA8293 0xA8297 0xA8302 0xA8304
Texas Instruments	TPS65233	-D**TPS65233_COMPATIBLE**	0x65233

(*) Untested drivers (no part available) which may require adjustments

5.4 Si1283/SUPERSET

The ‘SUPERSET’ code is the Si2183 source code, the code with the most complete API.



The Si2183 tuner wrapper code is retrieved from a dedicated FTP folder. It’s a mandatory item.

NB: Links to the FTP folders are provided to customers with a valid NDA upon request to their Skyworks representative.

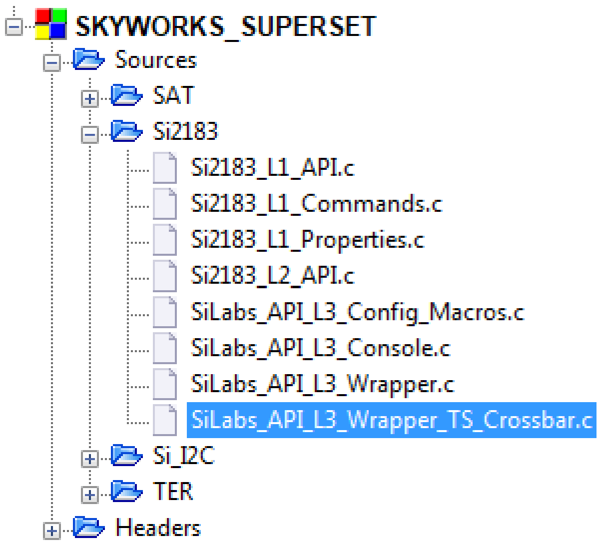
To use the Si2183 code as the SKYWORKS SUPERSET code, a compilation flag is required: -D***SILABS_SUPERSET***
To use the Si2183 code for SAT reception, a compilation flag is required: -D***SATELLITE_FRONT_END***
To use the Si2183 code for TER reception, a compilation flag is required: -D***TERRESTRIAL_FRONT_END***

NB: the above 2 flags are independent. This means that you can easily build the following applications:

- *SAT-only*
- *TER-only*
- *SAT+TER*

5.4.1 Dual-specific code: TS Crossbar

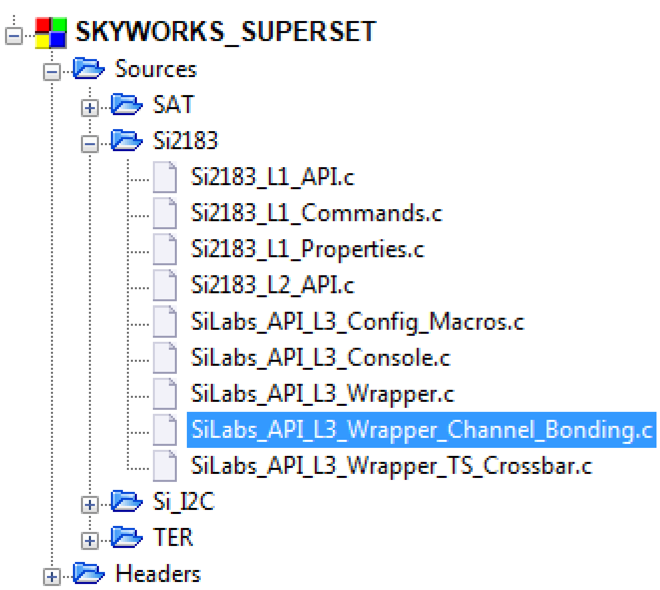
Duals (Si21xx2) demodulator parts support the TS crossbar feature, allowing to send TS_A and/or TS_B on the TS output buses A and B.
This part of the code requires adding the SiLabs_API_L3_Wrapper_TS_Crossbar code to your project.



To use the TS crossbar code, a dedicated compilation flag is required: -D***TS_CROSSBAR***
NB: This is not useful for 'single' configurations, it's only available with dual parts. i.e. Si21xx**2** parts.

5.4.2 Dual-specific code: Channel Bonding (DVB-S2X)

Duals (Si21xx2) demodulator parts support the Channel Bonding feature, allowing to generate a TS out of 2 or 3 different TS
The TS flow is from the part called 'SLAVE' to (optionally) a BRIDGE then finally to the MASTER whose TS output is connected to the SoC.
This part of the code requires adding the SiLabs_API_L3_Wrapper_TS_Channel_Bonding code to your project.



To use the Channel Bonding code, a dedicated compilation flag is required: -D***CHANNEL_BONDING***
NB: This is not useful for 'single' configurations, it's only available with dual parts. i.e. Si21xx**2** parts.

5.4.3 Multiple front-end code: INDIRECT I2C

To avoid I2C conflicts for dual/triple/quad frontends, it's required to route the tuner I2C carefully, with cases where fe[x] will need to have fe[y]'s I2C pass-through enabled to access its tuners.

This is handled by the code once a dedicated compilation flag is set, and calls to SiLabs_API_TER_tuner_I2C_connection (for TER) and SiLabs_API_SAT_tuner_I2C_connection (for SAT) are used to properly configure the I2C connection.

The ‘indirect I2C connection’ code is part of the L3 wrapper code.

To enable the indirect i2c connection, a dedicated compilation flag is required: -D***INDIRECT_I2C_CONNECTION***

NB: This is not useful for ‘single’ configurations.

5.4.3.1 Demodulator code related to the INDIRECT_I2C_CONNECTION feature

5.4.3.1.1 In the demodulator header file

```
typedef int  (*Si2183_INDIRECT_I2C_FUNC)  (void*);
```

5.4.3.1.2 In the demodulator structure

```
Si2183_INDIRECT_I2C_FUNC  f_TER_tuner_enable;
Si2183_INDIRECT_I2C_FUNC  f_TER_tuner_disable;
Si2183_INDIRECT_I2C_FUNC  f_SAT_tuner_enable;
Si2183_INDIRECT_I2C_FUNC  f_SAT_tuner_disable;
```

5.4.3.1.3 In the demodulator SW init function

```
char Si2183_L2_SW_Init            (Si2183_L2_Context *front_end
                                  , int demodAdd
                                  , int tunerAdd_Ter
 #ifdef    INDIRECT_I2C_CONNECTION
                                  , Si2183_INDIRECT_I2C_FUNC          TER_tuner_enable_func
                                  , Si2183_INDIRECT_I2C_FUNC          TER_tuner_disable_func
 #endif /* INDIRECT_I2C_CONNECTION */
                                  , int tunerAdd_Sat
#ifdef    UNICABLE_COMPATIBLE
 #ifdef    INDIRECT_I2C_CONNECTION
                                  , Si2183_INDIRECT_I2C_FUNC          SAT_tuner_enable_func
                                  , Si2183_INDIRECT_I2C_FUNC          SAT_tuner_disable_func
 #endif /* INDIRECT_I2C_CONNECTION */
                                  , void *p_context
                                  )
 #ifdef    INDIRECT_I2C_CONNECTION
   front_end->f_TER_tuner_enable  = TER_tuner_enable_func;
   front_end->f_TER_tuner_disable = TER_tuner_disable_func;
 #endif /* INDIRECT I2C CONNECTION */
#ifdef    INDIRECT_I2C_CONNECTION
   front_end->f_SAT_tuner_enable  = SAT_tuner_enable_func;
   front_end->f_SAT_tuner_disable = SAT_tuner_disable_func;
#endif /* INDIRECT I2C CONNECTION */
```

Using the above code, a pointer to the L3 functions used to enable/disable the i2c pass-through for either TER or SAT is stored in the demodulator context.
The demodulator then has the capability to close the i2c pass-through from another front-end is needed, in case the i2c connection is ‘INDIRECT’.

5.4.3.2 Wrapper code related to the INDIRECT_I2C_CONNECTION feature

5.4.3.2.1 In the SILABS_FE_Context structure

```
 int                    TER_tuner_I2C_connection;
 int                    SAT_tuner_I2C_connection;
```

5.4.3.2.2 In the Wrapper SW init function

```
#ifdef    INDIRECT_I2C_CONNECTION
  SiTRACE("INDIRECT_I2C_CONNECTION allowed.\n");
  for (i=0; i< FRONT_END_COUNT; i++) {
    if ( front_end   == &(FrontEnd_Table[i]) ) {
      front_end->TER_tuner_I2C_connection = i;
      front_end->SAT_tuner_I2C_connection = i;
    }
  }
  SiTRACE("Default front_end->TER_tuner_I2C_connection: %d\n", front_end->TER_tuner_I2C_connection );
  SiTRACE("Default front_end->SAT_tuner_I2C_connection: %d\n", front_end->SAT_tuner_I2C_connection );
#endif /* INDIRECT_I2C_CONNECTION */
```

5.4.3.2.3 Added Wrapper functions

The above code is storing default values (i.e. i2c connection via the current front_end). These values are later on updated during the SW configuration calling the 2 following functions

```
/************************************************************************************************************
  SiLabs_API_TER_tuner_I2C_connection function
  Use:        TER tuner I2C passthrough selection function
              Used to select which demodulator passthrough needs to be used to connect with the TER tuner I2C
  Behavior:   This function sets the TER_tuner_I2C_connection value in the front-end context
************************************************************************************************************/
int   SiLabs_API_TER_tuner_I2C_connection (SILABS_FE_Context *front_end, int fe_index)
{
  SiTRACE("API CALL CONFIG: SiLabs_API_TER_tuner_I2C_connection (front_end, %d);\n", fe_index);
  front_end->TER_tuner_I2C_connection = fe_index;
  return front_end->TER_tuner_I2C_connection;
}
/************************************************************************************************************
  SiLabs_API_SAT_tuner_I2C_connection function
  Use:        SAT tuner I2C passthrough selection function
              Used to select which demodulator passthrough needs to be used to connect with the SAT tuner I2C
  Behavior:   This function sets the SAT_tuner_I2C_connection value in the front-end context
************************************************************************************************************/
int   SiLabs_API_SAT_tuner_I2C_connection (SILABS_FE_Context *front_end, int fe_index)
{
  SiTRACE("API CALL CONFIG: SiLabs_API_SAT_tuner_I2C_connection (front_end, %d);\n", fe_index);
  front_end->SAT_tuner_I2C_connection = fe_index;
  return front_end->SAT_tuner_I2C_connection;
}
```

Once the SW init stage is done for each front_end, including a call to SiLabs_API_SW_Init followed by calls to SiLabs_API_TER_tuner_I2C_connection and SiLabs_API_SAT_tuner_I2C_connection (when applicable), each SILABS_FE_Context contains the index of the demodulator to use to connect the TER and SAT tuners.

5.4.3.3 Enabling the i2c pass-through during execution

A set of 4 functions (see below) exists to enable/disable I2C for both TER and SAT. We will only describe the behavior of one of them, since the code is similar for all 4 functions.

```
int   SiLabs_API_TER_Tuner_I2C_Enable      (SILABS_FE_Context *front_end);
int   SiLabs_API_TER_Tuner_I2C_Disable     (SILABS_FE_Context *front_end);
int   SiLabs_API_SAT_Tuner_I2C_Enable      (SILABS_FE_Context *front_end);
int   SiLabs_API_SAT_Tuner_I2C_Disable     (SILABS_FE_Context *front_end);
```

The SiLabs_API_TER_Tuner_I2C_Enable function checks in the table containing all SILABS_FE_Context pointers which one is the current one (passed as ‘front_end’), stores it as ‘requester’ then retrieves the value of front_end->TER_tuner_I2C_connection (stored as ‘connecter’).
Pointer checking is done to avoid using undeclared pointers.
In addition to calling the I2C enable L3 function for ‘connecter’, the L3 code also generates a SiTRACE message in case I2C connection is not direct, for debug purposes.

```
/************************************************************************************************************
  SiLabs_API_TER_Tuner_I2C_Enable function
  Use:        Demod Loop through control function,
              Used to switch the I2C loopthrough on, allowing communication with the tuners
              This function can control the I2C passthrough for any front-end in the front-end table,
               and is useful mainly in multi-front-end applications with dual tuners or dual demodulators,
               when the TER tuner I2C is not directly connected to the corresponding demodulator.
  Return:      the final mode (-1 if not known)
************************************************************************************************************/
int   SiLabs_API_TER_Tuner_I2C_Enable     (SILABS_FE_Context *front_end)
{
#ifdef    INDIRECT_I2C_CONNECTION
  int fe;
  int requester;
  int connecter;
  int fe_count;
  fe_count = FRONT_END_COUNT;
  for (fe=0; fe< fe_count; fe++) {
    if ( front_end  == &(FrontEnd_Table[fe]) ) {
      requester = fe;
      connecter = front_end->TER_tuner_I2C_connection;
      SiTRACE("-- I2C -- SiLabs_API_TER_Tuner_I2C_Enable  request for front_end %d via front_end %d\n", requester, connecter);
      if (connecter < fe_count) {
        if (requester != connecter) {
      SiTRACE("-- I2C -- Enabling  indirect TER tuner connection  for front_end %d via front_end %d\n", requester, connecter);
        }
        return SiLabs_API_Tuner_I2C_Enable(&(FrontEnd_Table[connecter]) );
      }
      break;
    }
  }
  SiTRACE("-- I2C -- SiLabs_API_TER_Tuner_I2C_Enable  request failed! Unable to find a match for the caller front_end! (0x%08x)\n", (int)front_end);
  SiERROR("-- I2C -- SiLabs_API_TER_Tuner_I2C_Enable  request failed! Unable to find a match for the caller front_end!\n");
  return 0;
#endif /* INDIRECT_I2C_CONNECTION */
  return SiLabs_API_Tuner_I2C_Enable(front_end);
}
```

A similar function exists to disable the I2C pass-through for the TER tuner.
A similar set of functions exists for SAT.

5.4.3.4 Use cases

In a situation where the application uses Si21832 (dual Si2183), with 2xTER tuners and 2 SAT tuners, the most common HW design connects all 4 tuners on the fe[0] i2c pass-through.
So, when fe[1] needs to access the tuners, it needs to call the L3 wrapper function to have it call the i2c enable function of fe[0].
The corresponding SW configuration is shown below.

5.4.3.4.1 Most common software configuration

This configuration corresponds to a QUAD design with all tuners accessed via fe[0].

```
  /* 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_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_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);
```

5.4.3.4.2 No i2c pass-through usage

In cases where all tuner I2C buses are directly connected to the SoC, i2c pass-through are not used and the code should be configured to reflect this.

This is achieved by using the specific value of 100 in the configuration, as follows (non I2c-related lines hidden for convenience):

```
  /* SW Init for front end 0 */\
  front_end                          = &(FrontEnd_Table[0]);\
. . .
  SiLabs_API_TER_tuner_I2C_connection  (front_end, 100);\
. . .
  SiLabs_API_SAT_tuner_I2C_connection  (front_end, 100);\
. . .
  /* SW Init for front end 1 */\
  front_end                           = &(FrontEnd_Table[1]);\
. . .
  SiLabs_API_TER_tuner_I2C_connection  (front_end, 100);\
. . .
  SiLabs_API_SAT_tuner_I2C_connection  (front_end, 100);\
. . .
```

5.4.3.4.3 I2c pass-through closed ‘once and for all’

In certain configurations it is required to access all tuners via a single pass-through and then keep this pass-through enabled at all times.

This is generally useful in multi-threaded applications to avoid the use of application-specific semaphores to avoid disabling a pass-through while the tuner is being accessed by another thread.

This is achieved by using the specific value of 100 in the configuration and closing the pass-through during the first init, as follows:

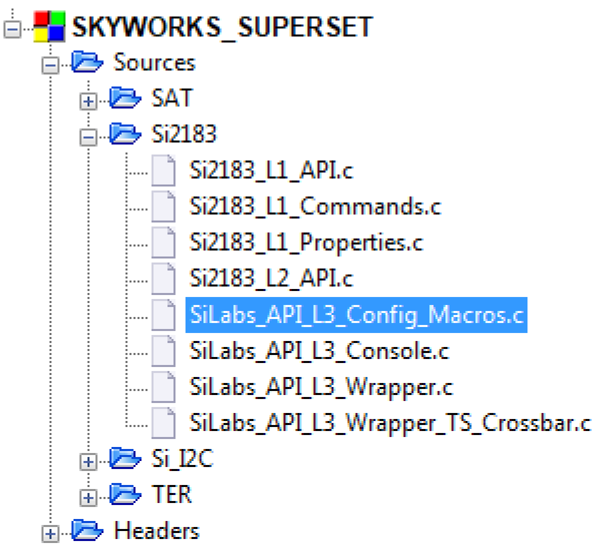
```
/* SW Init for front end 0 */\
  front_end                          = &(FrontEnd_Table[0]);\
. . .
  SiLabs_API_TER_tuner_I2C_connection  (front_end, 100);\
. . .
  SiLabs_API_SAT_tuner_I2C_connection  (front_end, 100);\
. . .
  /* SW Init for front end 1 */\
  front_end                            = &(FrontEnd_Table[1]);\
. . .
  SiLabs_API_TER_tuner_I2C_connection  (front_end, 100);\
. . .
  SiLabs_API_SAT_tuner_I2C_connection  (front_end, 100);\
. . .
  SiLabs_API_HW_Connect                (front_end, 1);
. . .
  front_end                          = &(FrontEnd_Table[0]);\
  SiLabs_API_Tuner_I2C_Enable          (front_end);
```

5.4.4 SW configuration macros code

To make it easy to configure the code for many HW configurations, a set of configuration macros are defined.
These are enabled as part of the SiLabs_API_L3_Config_Macros .c and .h files.
The .h files defines the macros, while the .c file allows the application to select the desired macro corresponding to the HW.

The amount of macros accessible to the console application depends on compilation flags:

To use the configuration code:
-D***CONFIG_MACROS***
To allow using macros prepared for Skyworks EVBs:
-D***SILABS_EVB_MACROS***



For development projects, we recommend using the existing configuration macros as examples to create a custom set of macros. A good macro naming would consist in using the HW nickname for the macro names, such that they get easily identified within the company.

6 Source code compilation flags

6.1 TER standards

The possible TER standards are:

- DVB-T
- DVB-T2 (on top of DVB-T)
- DVB-C
- DVB-C2 (on top of DVB-C)
- MCNS
- ISDB-T

The corresponding compilation flags are:

For DVB-T selection:
-D***DEMOD_DVB_T***
For DVB-T2 support, on top of the above:
-D**DEMOD_DVB_T2**

For DVB-C selection:
-D***DEMOD_DVB_C***
For DVB-C2 support, on top of the above:
-D**DEMOD_DVB_C2**
For MCNS selection:
-D***DEMOD_MCNS***

For ISDB-T selection:
-D***DEMOD_ISDB_T***

6.2 SAT standards

The possible SAT standards are:

- DVB-S
- DVB-S2 (on top of DVB-S)
- DVB-S2X (on top of DVB-S2)
- DSS

Since all SAT applications now require DVB-S2, this standard is always selected.
All DVB-S2 capable parts also need to support DVB-S as well, so DVB-S is also always selected.
Since DSS is very close to DVB-S, it's included with DVB-S support, so it's also always selected.
The only possible remaining selection is the DVB-S2X additions.

So, only 2 compilation flags are finally used for SAT:
For 'general' SAT selection:
-D***DEMOD_DVB_S_S2_DSS***
For DVB-S2X support, on top of the above:
-D**DEMOD_DVB_S2X**

6.3 Compatibility with specific demodulators

Apart from the media and standard selection, it's also required to '#include' the proper FW for the parts in use on the platform.

This selection is done also using compilation flags listed in the table below.

NB: Several flags can be declared, depending on the parts the code needs to be compatible with.

NB: Flags for ES (engineering Samples) are not listed here, since they would normally not be used in final products but only for early evaluation purposes.

Part compatibility compilation flags (to load the proper FW)

Part Number (single) / (dual)	Compatibility flag (for FW '#include')	type	**SiLabs_API_Frontend_Chip** demod_id
Si2160-A40 / Si21602-A40 Si2162-A40 / Si21622-A40 Si2164-A40 / Si21642-A40 Si2168-B40 / Si21682-B40 Si2169-B40 / Si21692-B40	-D***Si2164_A40_COMPATIBLE***	derivative derivative parent derivative derivative	0x2183
Si2168-A30 Si2169-A30	-D***Si2169_30_COMPATIBLE***	derivative parent	
Si2166-B20 Si2167-B20	-D***Si2167B_20_COMPATIBLE***	derivative parent	
Si2166-B22 / Si21662-B22 Si2167-B22 / Si21672-B22 Si21652-B22	-D***Si2167B_22_COMPATIBLE***	derivative parent derivative	
Si2167-B25	-D***Si2167_B25_COMPATIBLE***	parent	
Si2160-B50 / Si21602-B50 Si2162-B50 / Si21622-B50 Si2164-B50 / Si21642-B50 Si2166-C50 / Si21662-C50 Si2167-C50 / Si21672-C50 Si2168-C50 / Si21682-C50 Si2169-C50 / Si21692-C50 Si2180-A50 / Si21802-A50 Si2181-A50 / Si21812-A50 Si2182-A50 / Si21822-A50 Si2183-A50 / Si21832-A50	-D***Si2183_A50_COMPATIBLE***	derivative derivative derivative derivative derivative derivative derivative derivative derivative derivative parent	
Si2160-B55 / Si21602-B55 Si2162-B55 / Si21622-B55 Si2164-B55 / Si21642-B55 Si2166-C55 / Si21662-C55 Si2167-C55 / Si21672-C55 Si2168-C55 / Si21682-C55 Si2169-C55 / Si21692-C55 Si2180-A55 / Si21802-A55 Si2181-A55 / Si21812-A55 Si2182-A55 / Si21822-A55 Si2183-A55 / Si21832-A55	-D***Si2183_A55_COMPATIBLE***	derivative derivative derivative derivative derivative derivative derivative derivative derivative derivative parent	
Si2160-C5A / Si21602-C5A Si2162-C5A / Si21622-C5A Si2164-C5A / Si21642-C5A Si2166-D5A / Si21662-D5A Si2167-D5A / Si21672-D5A Si2168-D5A / Si21682-D5A Si2169-D5A / Si21692-D5A Si2180-B5A / Si21802-B5A Si2181-B5A / Si21812-B5A Si2182-B5A / Si21822-B5A Si2183-B5A / Si21832-B5A	-D***Si2183_B5A_COMPATIBLE***	derivative derivative derivative derivative derivative derivative derivative derivative derivative derivative parent	

Si2160-C60 / Si21602-C60 Si2162-C60 / Si21622-C60 Si2164-C60 / Si21642-C60 Si2166-D60 / Si21662-D60 Si2167-D60 / Si21672-D60 Si2168-D60 / Si21682-D60 Si2169-D60 / Si21692-D60 Si2180-B60 / Si21802-B60 Si2181-B60 / Si21812-B60 Si2182-B60 / Si21822-B60 Si2183-B60 / Si21832-B60	-D***Si2183_B60_COMPATIBLE***	derivative derivative derivative derivative derivative derivative derivative derivative derivative derivative parent	
Si2169-D63 / Si21692-D63	-D***Si2183_B63_COMPATIBLE***	parent	

6.4 Floats vs no floats

Some platforms do not allow using floats.
The code is supporting this thanks to a dedicated compilation flag: -D***NO_FLOATS_ALLOWED***

NB: when NO_FLOATS_ALLOWED is used, the status rates generally expressed as floats are only available as 'mant' and 'exp', and the C/N is available in 1/100 dB unit (in status->cn_100).

6.5 Frontend count

The number of SILABS_FE_Context instances is controlled by a compilation flag:
-D***FRONT_END_COUNT=n*** ***(where n=1, 2 , 3 or 4)***

NB: By default, i.e. if this flag is not set, it will be forced to 4 to match all possible Skyworks EVBs.

6.6 FW download over SPI

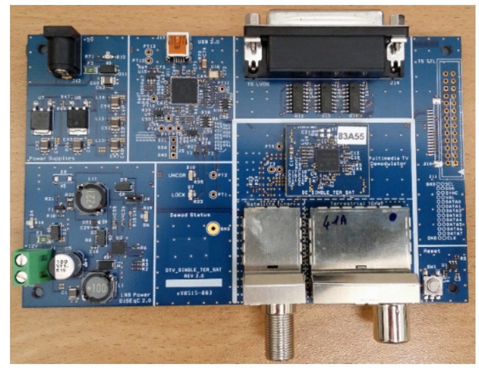
Skyworks DTV demodulators allow downloading FW over SPI.
To enable this feature in the code, a dedicated compilation flag is required: -D***FW_DOWNLOAD_OVER_SPI***

7 Quickly building the SKYWORKS SUPERSET console application

Below are a couple easy steps to build and run the Skyworks SKYWORKS SUPERSET console application on a Windows PC.

It is based on the assumption that you have access to the Skyworks DTV demodulator resources, either via links to the various FTP folders, or via an authorized access to the Skyworks DropBox folder where the project files are stored.

7.1 Preliminary requirements



- Access to the Skyworks Si2183 (SKYWORKS SUPERSET) code
  - On FTP (see you Skyworks representative to get access)

  Or
  - On DropBox (see you Skyworks representative to get access)
- A DTV_SINGLE_TER_SAT_Rev2_0 Skyworks EVB
- Completed CodeBlocks installation (or gcc compilation capability if using a makefile).

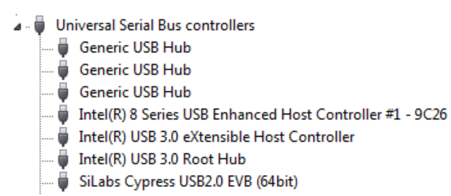
7.2 Retrieving SKYWORKS SUPERSET resources

Download the SKYWORKS_SUPERSET code from FTP or DropBox.

7.3 USB driver installation

NB: If you've already been using a Skyworks Broadcast Video EVB for evaluation purpose (using the provided GUI), you have completed this step already. You only need to check that the driver is properly installed on your machine.

7.3.1 Checking USB driver installation

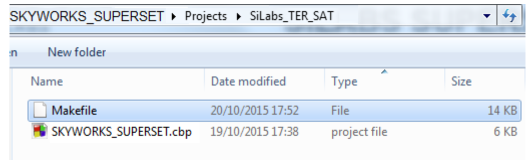


Connect the EVB to your machine and open the Windows Device Manager (under Windows, go to the control Panel then select 'Device Manager'.
Under 'Universal Serial Bus controllers' you should see the 'Skyworks Cypress USB2.0 EVB' if the installation is correct.
If the proper device appears in the Device Manager, you don't need to re-install the driver.

7.3.2 Installing the USB driver

If the 'Skyworks Cypress USB2.0 EVB' line doesn't appear, refer to Annex iv USB driver installation for details on the USB driver installation.

7.4 Building using 'make'



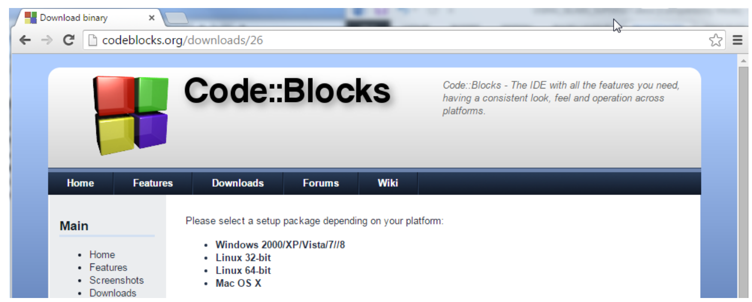
The SKYWORKS_SUPERSET/Projetcs/SiLabs_TER_SAT folder contains a makefile which can be used to compile and link the console application using the 'make' command.

This makefile corresponds to the compilation of an application for the following parts (the example application used in this document):

- SAT Tuner: AV2018
- SAT LNB Controller: LNBH25
- SAT Unicable
- TER Tuner: Si2141
- Si2183
- I2C

See Annex i Makefile content for details on the content of this makefile.

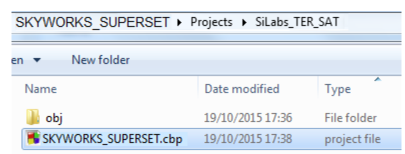
7.5 Opening the CodeBlocks project



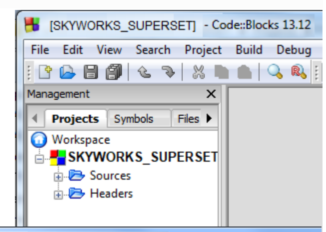
NB: we use CodeBlocks in this example, because it's a free IDE, available for both Windows and Linux, and it uses GCC/G++ to compile the code. Once you get familiar with the code, you can of course use your usual IDE. For the purpose of getting started quickly, using CodeBlocks is more convenient.

If you don't have CodeBlocks installed on your machine, you should preferably download it from http://codeblocks.org/downloads

Select the installer with mingw, since it will install the GCC compiler which is required for compiling.

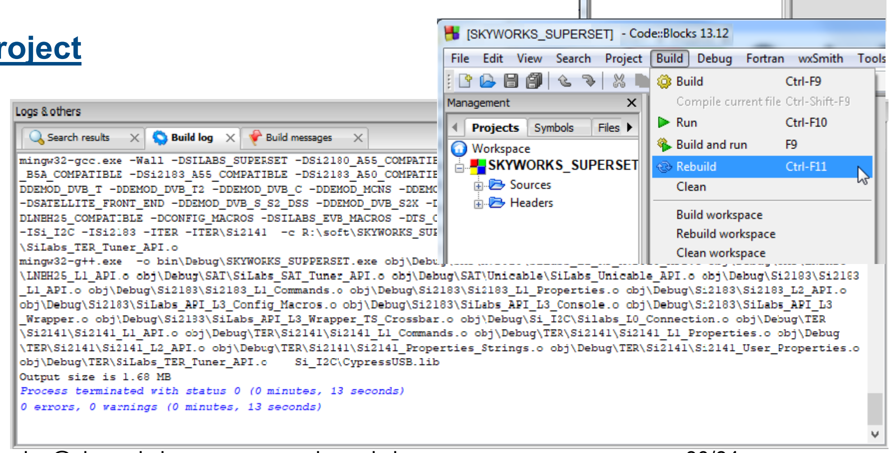


Once you have CodeBlocks installed, under you SKYWORKS_SUPERSET folder go to Projects\SiLabs_TER_SAT and select the SKYWORKS_SUPERSET.cbp CodeBlocks Project.



Double-click on the project to open it in CodeBlocks.

7.6 Building the CodeBlocks project



If you want to make sure you're rebuilding the entire project, select 'Build/Clean' first.

Select 'Build/Rebuild' to build the console application.

The final result is:

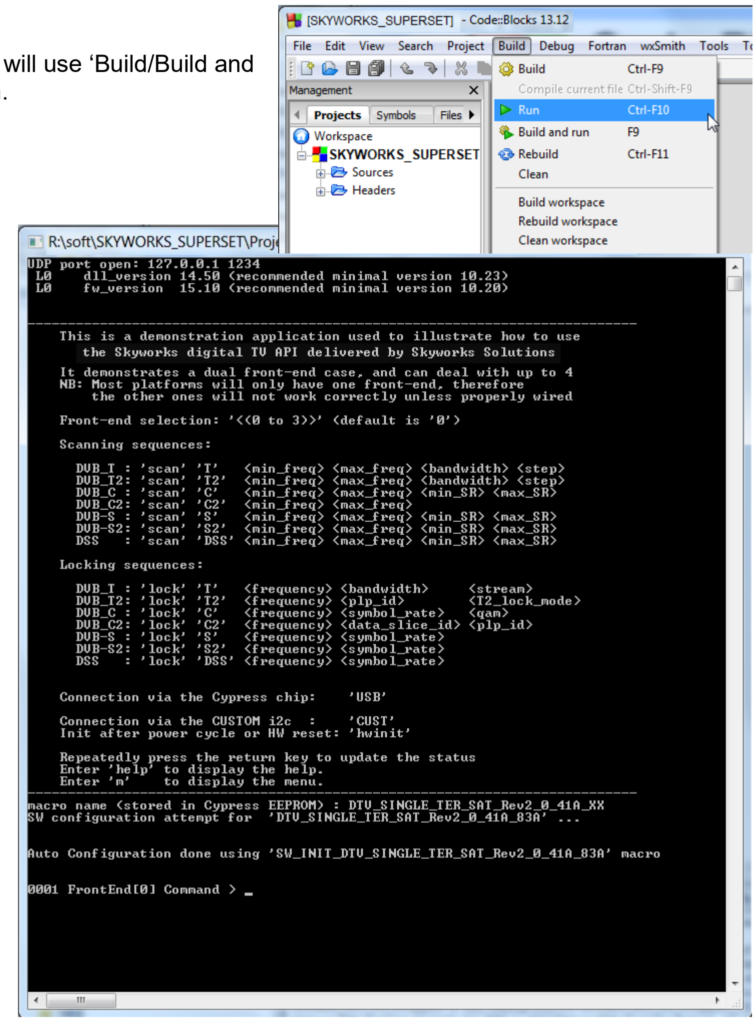
- An indication in the 'Build Log' window that the compiling and linking terminated with no error.
- The output file is under \bin\debug
- It's named SKYWORKS_SUPERSET.exe

You can note here:

- ✓ The last call to 'mingw32-gcc.exe' with all compilations flags. This is the compilation log for the last file which has been compiled.
- ✓ The call to 'mingw32-g++.exe' with all '.o' files. This is the linking stage, when all object files resulting for the compilation are linked to make the finale executable.

7.7 Running the CodeBlocks project

Select 'Build/Run' to run the console application. Later on, you will use 'Build/Build and run' to compile and run the application using a single operation.

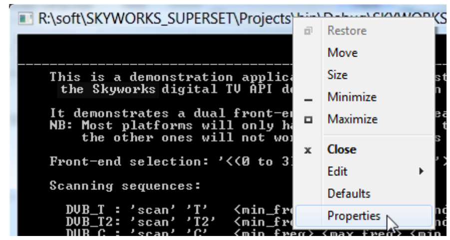


Since at this moment you should have a DTV_SINGLE_TER_SAT_Rev2_0 Skyworks EVB connected to your Windows machine with a proper USB driver installation, the application should be able to read the configuration string from the Cypress eeprom and use it to auto-configure the application to match the EVB.

In the command window we can see:

- The Cypress DLL version (here 14.50). This will be visible even if there is no EVB connected.
- The Cypress FW version (here 15.10). This will be visible only if the EVB is connected and the USB driver is properly installed. It can therefore be used as a first check of the USB installation.
- The macro name retrieved (using I2c over USB) from the Cypress eeprom is DTV_SINGLE_TER_SAT_Rev2_0_41A_XX. This is because the DTV_SINGLE_TER_SAT_Rev2_0 EVB features a motherboard which can be fitted with several TER tuners (here Si2141) and a daughter card which can support many different DTV demodulators. Based on that, the application will attempt to use the SW_INIT_DTV_SINGLE_TER_SAT_Rev2_0_41A_83A macro to lock the EVB.
- The fact that the application could read the configuration macro can be used as a further check that i2c is OK as far as the Cypress eeprom, so it's probably all ok.

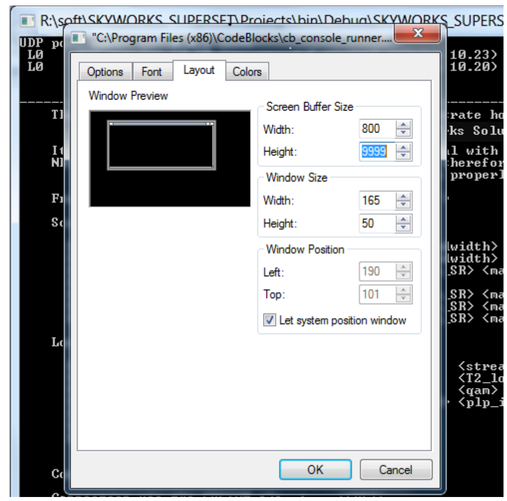
7.8 Configuring and using the Windows command window



At this stage, it can be convenient to configure the Windows command window to enable it to

- Display longer lines (the default line width is quite limited)
- Display more lines (the default number of line is small compared to what we may need)
- Be a bit easier to use for copy/pasting

The command windows settings are accessible when you move the mouse on the title bar and right-click, then select 'Properties'

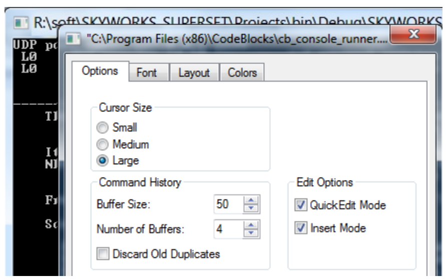


7.8.1 Windows command window settings

When running the application (especially when using traces for debug/learning purposes) some rather long lines will need to be displayed, so set the 'Screen Buffer Size/Width' to a large value (here, we use 800).

Since the number of lines can also be quite large, set the 'Screen Buffer Size/Height' to 9999 (It's the maximum possible value).

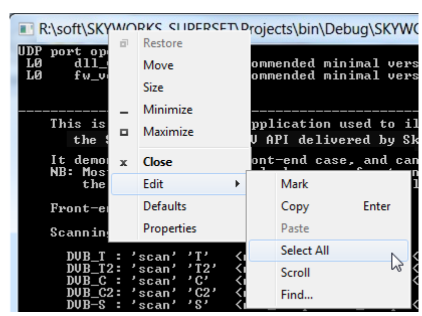
Press 'OK' to validate you changes.



Finally, under 'Options' tick the 'QuickEdit Mode' and 'Insert mode' boxes to allow easier copy/pasting in the command window.

Press 'OK' to validate you changes.

7.8.2 Windows command window copy/pasting



Once the above changes have been done, the Windows command window will display long lines and store maximum 9999 lines in its buffer.
During execution, it's then possible to select the entire content of the command window and store it into a text file. This text file can then be edited using a normal text editor and sent to Skyworks for analysis/debug.

Using text files is much easier than using screenshots, so we recommend to copy/paste the window content when needed.

To select all text in the command window, move the mouse on the title bar and right-click, then select 'Edit/Select All'. When doing this, the selected text wil appear with inverted colors in the command window.

NB: To select smaller parts of the command window's text, you can also select it with the mouse (with the left mouse button kept down). The selected text with appear with inverted colors.

To copy the selected text to the buffer, move the mouse on the title bar and right-click, then select 'Edit/Copy' (or press 'enter', as indicated in the menu).

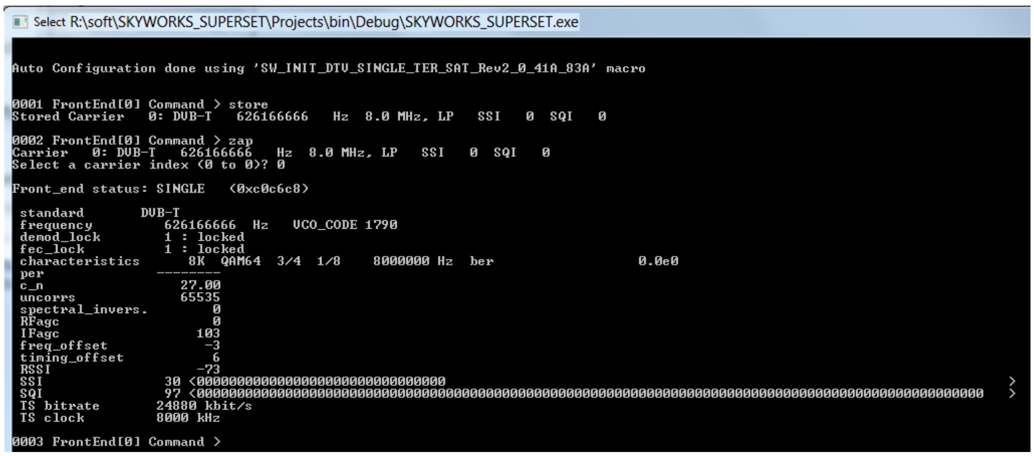
Then, open a text editor and 'paste' the selected text into a text file.

7.9 Locking the EVB

The console application is prepared to store an 8MHz DVB-T signal at 626166666 Hz, so you can use ‘store’ to have this channel information stored in the channels table.

To be able to use this, you need to have a valid 8MHz DVB-T signal at 626166666 Hz. If you are using another frequency, you need to use the ‘lock’ feature instead, and follow the on-screen instructions to get a lock on your channel. (It will work even at 626000000, due to the current afc range settings for DVB-T, so a signal at 626000000 is fine).

Then, since after using ‘store’ you have one channel in the channels list, you can use the ‘zap’ feature to lock the EVB on the first channel, channel 0.



Use ‘zap’ then ‘0’. This will trigger the first HW initialization of the EVB, and go through the first lock with the channel 0 parameters.

As visible in the screenshot here, lock is achieved directly on the DVB-T channel.

Using ‘u’ you can reset the uncorrs counter.

Pressing ‘enter’ will refresh the status.

Use ‘m’ to get information on the available features.

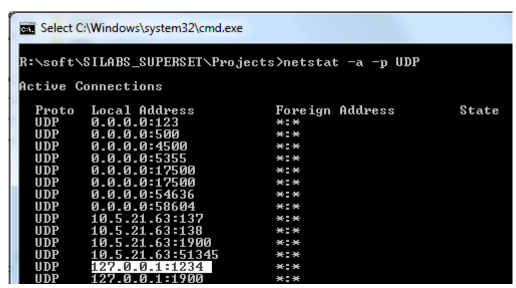
7.10 Activating TS over USB

The Skyworks EVBs allow transmitting the received TS data over USB to the PC (on UDP port 1234).

To activate this, use ‘TS’ then ‘GPIF’.

Not much will be visible, except than running ‘netstat –a –p UDP’ in a new command window will show UDP traffic on port 127.0.0.1:1234 (the default IP address and UDP port for the GPIF feature).

This can be used to confirm that TS data is now flowing from the frontend to the PC.



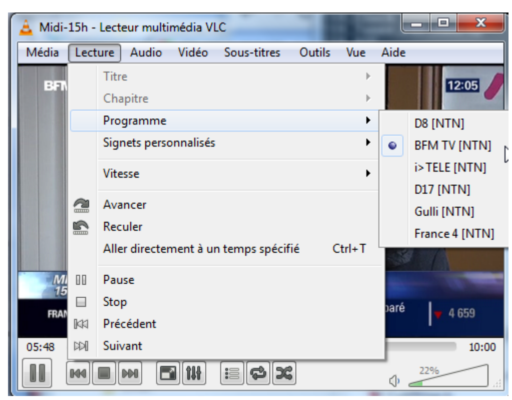
7.11 Starting video decoding in VLC

Once TS data is sent to 127.0.0.1:1234, you can use a media player such as VLC to play it.

Start VLC and open a network stream at UDP://@127.0.0.1:1234 to get the picture displayed inside VLC.

It’s then possible to select the program to be displayed via the VLC menus.

Obviously, to get video you need a valid DVB-T signal with proper TS content, including video and audio.



8 Software configuration functions

When using the SKYWORKS SUPERSET CODE, a set of functions is used to configure each frontend to match the HW design and SW source code.

Below is the configuration macro used for the DTV_SINGLE_TER_SAT_Rev2_0 EVB, which is showing all the configuration macros.

```
#define DTV_SINGLE_TER_SAT_Rev2_0(tuner_code,chip_code) \
  /* SW Init for front end 0 */\
  front_end                   = &(FrontEnd_Table[0]);\
  SiLabs_API_Frontend_Chip            (front_end, chip_code);\
  SiLabs_API_SW_Init                  (front_end, 0xc8, 0xc0, 0xC6);\
  SiLabs_API_SPI_Setup                (front_end, 0x00, 5, 0, 9, 1);\
  SiLabs_API_TS_Config                (front_end, 0, 0, 0, 0, 0, 0);\
  SiLabs_API_Select_TER_Tuner         (front_end, tuner_code, 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, 0xb, 1);\
  SiLabs_API_TER_AGC                  (front_end, 0x0, 0, 0xa, 0);\
  SiLabs_API_TER_Tuner_AGC_Input      (front_end, 1);\
  SiLabs_API_TER_Tuner_FEF_Input      (front_end, 1);\
  SiLabs_API_TER_Tuner_IF_Output      (front_end, 0);\
  SiLabs_API_Select_SAT_Tuner         (front_end, 0xA2018, 0);\
  SiLabs_API_SAT_Select_LNB_Chip      (front_end, 25, 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, 0xd, 1, 0x0, 1);\
  SiLabs_API_Set_Index_and_Tag        (front_end, 0, "fe[0]");\
  SiLabs_API_HW_Connect               (front_end, 1);\
  SiLabs_API_Cypress_Ports            (front_end, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00);\
```

Using these configuration macros the code can be adapted to cover all configurations, taking into account all possible cases for chip selection, i2c interconnection, clocks, AGCs, number of frontends, etc.

The same configuration code can be used for:

- Skyworks GUI configuration (when using the executable GUI provided with Skyworks EVBs)
- Source code macros (used in the console code)
- Final source code SW initialization
- Single, dual, triple, quad designs (one block of code per frontend)

The following paragraph goes through each of the above functions, providing information about the meaning of each field and the possible values.

Generally speaking, it's required to call the SW configuration functions in the order they are listed in in this document, especially starting with the 'general startup' functions (they perform the SW context memory allocation), followed by the SAT and TER functions (in any order), and finally calling the 'general completion' functions (they perform settings using the allocated contexts).

NB: No i2c traffic is generated during the SW configuration sequence.

8.1 Software clock configuration for DUALs

In DUALs the clock input pins are bonded together, and the clock is often shared between the frontends, so the code needs to be configured to avoid any change on the clock pins.

The simplest way to achieve this is to avoid switching the clock between TER and SAT.

This is the case if both the TER and SAT tuners on all frontends use the same clock source.

Below is a typical configuration for a TER+SAT DUAL frontend:

```
#define DTV_SINGLE_TER_SAT_Rev2_0(tuner_code,chip_code) \
  /* SW Init for front end 0 */\
  front_end                    = &(FrontEnd_Table[0]);\
. . .
  SiLabs_API_TER_Tuner_ClockConfig    (front_end, 1, 1);\
  SiLabs_API_TER_Clock                (front_end, 1, 44, 24, 1);\
. . .
  SiLabs_API_SAT_Clock                (front_end, 1, 44, 24, 1);\
. . .

  /* SW Init for front end 1 */\
  front_end                    = &(FrontEnd_Table[1]);\
. . .
  SiLabs_API_TER_Tuner_ClockConfig    (front_end, 0, 1);\
  SiLabs_API_TER_Clock                (front_end, 1, 44, 24, 1);\
. . .
  SiLabs_API_SAT_Clock                (front_end, 1, 44, 24, 1);\
. . .
```

Note that:

- All Tuners use the same clock source and clock frequency (here, the TER tuner clock from fe[0])
- In TER and SAT, the same clock source is used, so no risk of clock glitches.
- fe[0] is generating the clock and feeding it to fe[1]
- fe[0]'s clock is kept 'always on'

8.2 General startup configuration functions

8.2.1 front_end = (FrontEnd_Table[fe]);

This is not an actual function. It's used to select the frontend which is to be configured in the following lines.

- ***fe***

  The fe value is the index of the frontend in the frontend table.

  NB: The syntax here is different between TCL configuration scripts (used for GUIs) and the source code.
  TCL syntax: SelectFrontEnd fe
  C syntax: front_end =(FrontEnd_Table[fe]);

  Possible fe values
  The valid range is from 0 to FRONT_END_COUNT-1, so
  - 0 for 'single' designs
  - 0, 1 for 'dual' designs
  - 0, 1, 2 for 'triple' designs
  - 0, 1, 2, 3 for 'quad' designs

  fe value checks

  In HW: Check the number of frontends on your HW to know the range (from 0 to FRONT_END_COUNT-1).

  For multiple frontend designs: If the clock is coming from a single frontend and distributed to the other frontends, the frontend which is driving the master clock needs to be initialized first.

  In SW: it doesn't really matter which index in the list corresponds to which frontend, as long as:
  - All settings for each frontend are correct
  - The frontend generating the main clock is started first

8.2.2 SiLabs_API_Frontend_Chip (front_end, demod_id);

SiLabs_API_Frontend_Chip selects which demodulator source code the L3 layer must relay calls to.

- ***demod_id***

  demod_id is the code used in the Skyworks API L3 wrapper to select which source code is controlling the demodulator.

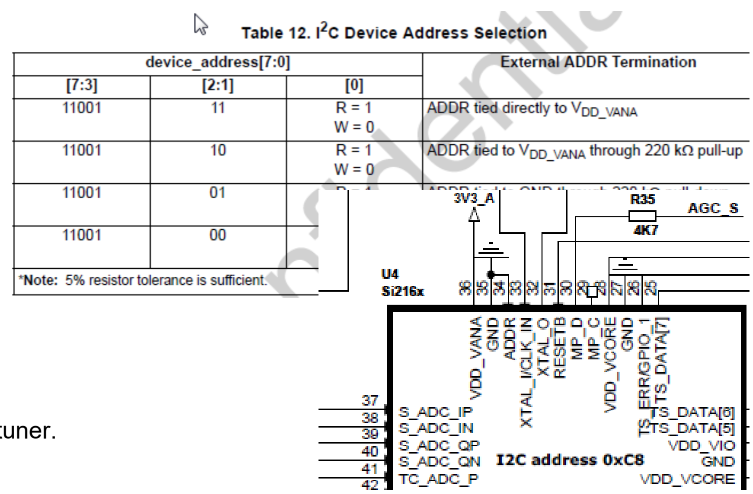
  Possible demod_id values

  For legacy codes (i.e. not the SKYWORKS SUPERSET) this code would vary depending on the source code tree. It's used at L3 level to route the L3 API calls to the corresponding demodulator code.

  For the SKYWORKS SUPERSET code, since there is a single source code there is only one valid value: 0x2183

8.2.3 SiLabs_API_SW_Init (front_end, demodAdd, tunerAdd_Ter, tunerAdd_Sat);

SiLabs_API_SW_Init is the main SW initialization function. It's used to initialize the DTV demodulator and tuner structures
This function performs all the steps necessary to initialize the DTV demodulator, tuner and all other instances (LNB, Unicable, etc.)



- ***demodAdd***

  demodAdd is the i2c address of the DTV demodulator.

  Possible demodAdd values
  0xc8, 0xca, 0xcc, 0xce for demodulators (it depends on the level on the ADDR pin)

  demodAdd value checks
  Check the HW diagram

- ***tuner Add_Ter***
  tuner Add_Ter is the i2c address of the TER tuner.

  Possible tuner Add_Ter values
  0xc0, 0xc2, 0xc4, 0xc6 for most TER tuners

  tuner Add_Ter value checks
  Check the HW diagram, i.e. the connection of the TER tuner address selection pin.

- ***tuner Add_Sat***
  tuner Add_Sat is the i2c address of the DTV demodulator.

  Possible tuner Add_Sat values
  0x-- for SAT tuners (check the SAT tuner datasheet)

  tuner Add_Sat value checks
  Check the HW diagram, i.e. the connection of the SAT tuner address selection pin(s).

8.2.4 SiLabs_API_SPI_Setup (front_end, send_option, clk_pin, clk_pola, data_pin, data_order);

SiLabs_API_SPI_Setup is not used in many designs, but is usable with the DTV_SINGLE_TER_SAT_Rev2_0 EVB.

It requires additional connections between the SoC and the DTV demodulator on top of I2C and TS (2 added pins for SPI clock and SPI serial data), plus dedicated support on the SoC side.
This function is used to configure the SPI download of FW to the DTV demodulators.
FW download over SPI is much faster than over i2c, so it's interesting to reduce the startup time when the HW allows it.
The SiLabs_API_SPI_Setup settings have to be checked with the HW designers.

- ***send_option***
  send_option is an optional SPI configuration byte, which can be used to configure the customer-specific SPI HW.

  send_option possible values
  With the DTV_SINGLE_TER_SAT_Rev2_0 EVB is needs to be set to 0x00.

- ***clk_pin***
  clk_pin indicates where the SPI clock signal enters the DTV demodulator.

  clk_pin possible values
  When using a Si2183 derivative, the possibilities are:
```
#define Si2183_SPI_LINK_CMD_SPI_CONF_CLK_DISEQC_CMD  9
#define Si2183_SPI_LINK_CMD_SPI_CONF_CLK_DISEQC_IN   7
#define Si2183_SPI_LINK_CMD_SPI_CONF_CLK_DISEQC_OUT  8
#define Si2183_SPI_LINK_CMD_SPI_CONF_CLK_GPIO0       5
#define Si2183_SPI_LINK_CMD_SPI_CONF_CLK_GPIO1       6
#define Si2183_SPI_LINK_CMD_SPI_CONF_CLK_MP_A        1
#define Si2183_SPI_LINK_CMD_SPI_CONF_CLK_MP_B        2
#define Si2183_SPI_LINK_CMD_SPI_CONF_CLK_MP_C        3
#define Si2183_SPI_LINK_CMD_SPI_CONF_CLK_MP_D        4
```

- ***clk_pola***
  clk_pola configure the polarity of the SPI clock signal

  clk_pola possible values
```
#define Si2183_SPI_LINK_CMD_SPI_CLK_POLA_FALLING  1
#define Si2183_SPI_LINK_CMD_SPI_CLK_POLA_RISING   0
```

- ***data_pin***
  data_pin indicates where the SPI data signal enters the DTV demodulator

  data_pin possible values
```
#define Si2183_SPI_LINK_CMD_SPI_CONF_DATA_DISEQC_CMD  9
#define Si2183_SPI_LINK_CMD_SPI_CONF_DATA_DISEQC_IN   7
#define Si2183_SPI_LINK_CMD_SPI_CONF_DATA_DISEQC_OUT  8
#define Si2183_SPI_LINK_CMD_SPI_CONF_DATA_GPIO0       5
#define Si2183_SPI_LINK_CMD_SPI_CONF_DATA_GPIO1       6
#define Si2183_SPI_LINK_CMD_SPI_CONF_DATA_MP_A        1
#define Si2183_SPI_LINK_CMD_SPI_CONF_DATA_MP_B        2
#define Si2183_SPI_LINK_CMD_SPI_CONF_DATA_MP_C        3
#define Si2183_SPI_LINK_CMD_SPI_CONF_DATA_MP_D        4
```

- ***data_order***
  data_order indicates whether the SPI data signal comes in 'LSB first' or 'MSB first'.

  data_order possible values
```
#define Si2183_SPI_LINK_CMD_SPI_DATA_DIR_LSB_FIRST  1
#define Si2183_SPI_LINK_CMD_SPI_DATA_DIR_MSB_FIRST  0
```

8.2.5 SiLabs_API_TS_Config (front_end, clock_config, gapped, serial_clk_inv, parallel_clk_inv, ts_err_inv, serial_pin);

SiLabs_API_TS_Config allows configuring the TS from the configuration macro. This is useful when different platforms don't use the same TS settings. Please refer to 'TS_Design_Guide_Apps_Note_Rev0.9.pdf' for details on the possible values.

- ***clock_config***
  clock_config is used to select the clock mode between 2 possible automatic modes (AUTO_ADAPT and AUTO_FIXED) and a fixed MANUAL clock frequency.

  clock_config possible values
  - 0: AUTO_ADAPT
  - 1: AUTO_FIXED
  - otherwise:MANUAL clock set to the provided value in kHz unit. Example: use clock=20000 to get a fixed frequency 20 MHz TS clock.

- ***gapped***
  gapped selects if the clock is punctured (i.e. gapped) or not.

  gapped possible values
  - 0: constant clock
  - 1: gapped clock

- ***serial_clk_inv***
  serial_clk_inv selects whether the serial clock signal is inverted or not.

  serial_clk_inv possible values
  - 0: TS serial clock signal non-inverted
  - 1: TS serial clock signal inverted

- ***parallel_clk_inv***
  parallel_clk_inv selects whether the parallel clock signal is inverted or not.

  parallel_clk_inv possible values
  - 0: TS parallel clock signal non-inverted
  - 1: TS parallel clock signal inverted

- ***ts_err_inv***
  ts_err_inv selects whether the TS error signal is inverted or not.

  ts_err_inv possible values
  - 0: TS error signal non-inverted
  - 1: TS error signal inverted

- ***serial_pin***
  serial selects the TS Dx pin used to output the TS serial signal.

  serial possible values
  - From 0 to 7 (most of the time either 0 or 7)

8.2.6 SiLabs_API_TS_Strength_Shape (front_end, serial_strength, serial_shape, int parallel_strength, parallel_shape);

SiLabs_API_TS_Config allows configuring the TS signals shape and strength. This is useful to adapt the TS bus settings to the HW design. Please refer to 'TS_Design_Guide_Apps_Note_Rev0.9.pdf' for details on the possible values.

- ***serial_strength***
  - 0 to 15: applied value
  - otherwise: NO change

- ***serial_shape***
  - 0 to 3: applied value
  - otherwise: NO change
- ***parallel_strength***
  - 0 to 15: applied value

   - otherwise: NO change
- ***parallel_shape***
   - 0 to 3: applied value
   - Otherwise: NO change

8.3 TER configuration functions

The following functions are only required when building and application for TER reception.
When not using TER, they can be skipped during SW configuration.

8.3.1 SiLabs_API_Select_TER_Tuner (front_end, ter_tuner_code, ter_tuner_index);

SiLabs_API_Select_TER_Tuner is used to select which TER tuner is in use.

- ***ter_tuner_code***
  ter_tuner_code is the code used for the TER tuner selection, easily readable when in hexadecimal, as it matches the part name, i.e '0x2178b' for Si2178B. In our example we use '0x2141'.

  ter_tuner_code possible values
  Possible values (as of writing):
   - 0x2124
   - 0x2141, 0x2144, 0x2146, 0x2147, 0x2148, 0x2148B
   - 0x2151, 0x2156, 0x2157, 0x2158, 0x2158B
   - 0x2173, 0x2176, 0x2177, 0x2178, 0x2178B
   - 0x2190, 0x2190B, 0x2191, 0x2191B, 0x2196

  These values can vary according to the code used, in case the application uses the API compatibility. For example, the Si2178-B code can be used to drive the Si2148-B or Si2158-B, etc.

  ter_tuner_code value check
  In SiLabs_TER_Tuner_API.c, function SiLabs_TER_Tuner_Select_Tuner, look for the possible ter_tuner_code values.

```
signed  int   SiLabs_TER_Tuner_Select_Tuner
  SiTRACE_X("Select TER Tuner selecting Si%04x_Tu
#ifdef   TER_TUNER_CUSTOMTER
#ifdef   TER_TUNER_Si2124
#ifdef   TER_TUNER_Si2141
  if (ter_tuner_code                == 0x2141 ) {
#endif /* TER_TUNER_Si2141 */
#ifdef   TER_TUNER_Si2144
#ifdef   TER_TUNER_Si2146
#ifdef   TER_TUNER_Si2147
#ifdef   TER_TUNER_Si2148
#ifdef   TER_TUNER_Si2148B
#ifdef   TER_TUNER_Si2151
#ifdef   TER_TUNER_Si2156
#ifdef   TER_TUNER_Si2157
#ifdef   TER_TUNER_Si2158
#ifdef   TER_TUNER_Si2158B
#ifdef   TER_TUNER_Si2173
#ifdef   TER_TUNER_Si2176
#ifdef   TER_TUNER_Si2177
#ifdef   TER_TUNER_Si2178
#ifdef   TER_TUNER_Si2178B
#ifdef   TER_TUNER_Si2190
#ifdef   TER_TUNER_Si2190B
#ifdef   TER_TUNER_Si2191
#ifdef   TER_TUNER_Si2191B
#ifdef   TER_TUNER_Si2196
  SiTRACE("Select_Tuner     selected   Si%04x_Tun
  return (silabs_tuner->ter_tuner_code<<8)+silab
}
```

- ***ter_tuner_index***
  ter_tuner_index is a provision in case there would be an application using more than 1 TER tuner per DTV demodulator.
  Unless this situation exists, the ter_tuner_index value is 0.

8.3.2 SiLabs_API_TER_tuner_I2C_connection (front_end, fe_index);

SiLabs_API_TER_tuner_I2C_connection is used to select which demodulator pass-through needs to be used to connect with the TER tuner I2C.

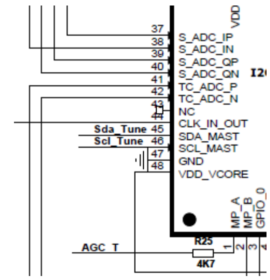
It's only active if ***INDIRECT_I2C_CONNECTION*** is declared in the compilation flags.

Background information related to tuner I2c connection
Dual demodulator includes an I2C switch to reduce the potential noise on satellite and terrestrial tuners during I2C accesses.
When using this I2C switch, SDA_MAST and SCL_MAST have to be connected to tuners I2C.
To prevent any conflict between both dies, I2C switch shall be controlled only by one demodulator, usually demodulator A.

- ***fe_index***
  fe_index is the index of the frontend whose demodulator i2c pass-through must be closed to access the TER tuner.

  For single designs this is generally the same value as the frontend index (i.e. 0).
  For duals, the second frontend tuners are often connected through the first demodulator. In this case, set fe_index to '0' for frontend[0] as well as frontend[1].
  For quads, values are often: '0' for frontend[0], '0' for frontend[1], '1' for frontend[2], '1' for frontend[3]

fe_index possible values
- *0 for single designs*
- *0, 1 for dual designs*
- *0, 1, 2 for triple designs*
- *0, 1, 2, 3 for quad designs*

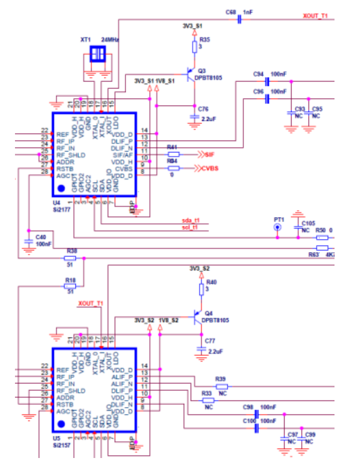


fe_index value check
Check the i2c routing diagram, this is set by HW design.
Check the i2c connections between demodulator(s) and TER tuners,
When using dual demodulators, tuners are generally connected through demodulator 'A'.

8.3.3 SiLabs_API_TER_Tuner_ClockConfig (front_end, xtal, xout);

SiLabs_API_TER_Tuner_ClockConfig is used to configure the TER tuner clock path inside the TER tuner.
This is required because Skyworks TER tuners feature several clock generation options.

NB: Note the functions named with 'TER Tuner' are configuring the TER tuner, not the DTV demodulator.



- ***xtal***
  xtal is a flag indicating if the tuner is driving a xtal or not. Set to '1' if the TER tuner drives an xtal, to '0' otherwise.

  xtal can be '0' for secondary tuners, when they receive their clock signal from another tuner.

  xtal possible values
  - 0: external clock
  - 1: a xtal is connected to the TER tuner

  xtal value check
  In the diagram, check the clock input for the TER tuner

- ***xout***
  xout is a flag indicating if the TER tuner provides an output clock to other parts.

  Set xout to '1' if the TER tuner clock output needs to be activated, to '0' otherwise.

  xout can be '0' for secondary tuners, when the clock signal is provided to the demodulator from another tuner.

  xout possible values
  - 0: no clock output from TER tuner
  - 1: a clock signal is output by the TER tuner

  xout value check
  In the diagram, check the connections between the TER tuner and the DTV demodulator.

8.3.4 SiLabs_API_TER_Clock (front_end, clock_source, clock_input, clock_freq, clock_control);

SiLabs_API_TER_Clock is used to configure the clock path used for TER reception by the DV demodulator.
The source of the clock can be a xtal, the TER tuner or the SAT tuner.
It configures the DTV demodulator, as opposed to SiLabs_API_TER_Tuner_ClockConfig which configures the TER tuner.

- ***clock_source***
  clock_source is the origin of the clock when locking in TER.
  This is used to know when to initialize the clock source, start the clock if it's used, and when to set the clock in standby.
  clock_source is taken into account together with clock_control to decide when and if the clock can be stopped.

clock_source possible values
- 0 for 'xtal'
- 1 for 'TER tuner'
- 2 for 'SAT tuner'

clock_source value check
In the diagram, check where the clock used for TER reception comes from.

- ***clock_input***
  clock_input is the clock input used by the demodulator for TER reception.

  NB: This is coded as pin numbers corresponding to single demodulator pinout, in an attempt to make reading the SW configuration easier.
  When using dual demodulators, use the same numbers as for single parts.

clock_input possible values
- 44 for 'CLKIO'
- 33 for 'Xtal_IN'
- 32 for 'Xtal' (driven by DTV demodulator)

clock_input value check
In the diagram, check the clock input pin used on the demodulator for TER reception.

- ***clock_freq***

  clock_freq is the clock frequency used by the demodulator for TER reception.

clock_freq possible values
- 16 for 16 MHz
- 24 for 24 MHz
- 27 for 27 MHz

clock_freq value check
In the diagram, check the clock frequency of the Xtal used as the clock source.

- ***clock_control***
  clock_control is the control mode for the TER tuner clock output.
  When the clock is used by another frontend, it must be 'always on'.
  When the clock is never used, it must be 'always off'.
  When the application should control it, it must be 'managed'

clock_control possible values
- 0 for 'ALWAYS_OFF'
- 1 for 'ALWAYS_ON'
- 2 for 'MANAGED'

clock_control value check
In the diagram, check if the TER tuner clock is used by another frontend. If yes, it should be kept 'ALWAYS ON' (use 1).
Check if the TER tuner clock is not going anywhere. If yes, it should be kept 'ALWAYS OFF' (use 0).
If you want the code to turn it ON/OFF when it's used/unused, use 2 ('MANAGED').

8.3.5 SiLabs_API_TER_Tuner_FEF_Input (front_end, dtv_fef_freeze_input);

SiLabs_API_TER_Tuner_FEF_Input is used to configure the TER tuner pin used for FEF freeze inside the TER tuner.
This is required because recent Skyworks TER tuners feature several possible FEF freeze input pins.

The FEF freeze signal goes from the DTV demodulator to the TER tuner. It's used to dynamically freeze the DTV AGC in the TER tuner during DVB-T2 FEF frames.

NB: Note the functions named with 'TER Tuner' are configuring the TER tuner, not the DTV demodulator.

- ***dtv_fef_freeze_input***

dtv_fef_freeze_input selects on the TER tuner side the input pin used for FEF FREEZE control.

dtv_fef_freeze_input possible values
Look into SiLabs_TER_Tuner_API.c/SiLabs_TER_Tuner_FEF_FREEZE_PIN_SETUP to see how the silabs_tuner->fef_freeze_pin value is used to setup the FEF management in the TER tuner (this depends on the TER tuner API). Unfortunately, different TER tuners use different APIs for FEF freeze, so it's required here to check the code for the possible dtv_fef_freeze_input values corresponding to each TER tuner.

Be aware that if this value is improperly set you may have a frozen AGC and therefore reception problems.

If in doubt, you can start using 'fef_mode = 0' in SiLabs_API_TER_FEF_Config to temporarily disable FEF freeze, then set this later on. The DTV demodulator may still be generating a FEF freeze signal, but it will not be taken into account by the TER tuner.

dtv_fef_freeze_input value check
In the diagram, check on which pin of the TER tuner the FEF freeze signal is connected to (it may be non-connected).

8.3.6 SiLabs_API_TER_FEF_Config (front_end, fef_mode, fef_pin, fef_level);

SiLabs_API_TER_FEF_Config is used to select in the DTV demodulator which pin is used to output the FEF freeze signal sent to the TER tuner.

The FEF freeze signal goes from the DTV demodulator to the TER tuner. It's used to dynamically freeze the DTV AGC in the TER tuner during DVB-T2 FEF frames.

- ***fef_mode***

fef_mode sets the selected mode for the FEF management.

During FEF periods it is required to avoid changing the TER tuner AGC. Recent tuners allow freezing their AGC through a pin. Other tuners allow slowing down their AGC after a tuner for a short period. Older tuners can only have their normal AGC slowed down when there is a possibility of having FEF frames in the transmitted signal

fef_mode possible values
- 0 for 'SLOW_NORMAL' (fallback if no better option)
- 1 for 'FREEZE_PIN' (if wired)
- 2 for 'SLOW_INITIAL' (depending on the tuner capabilities)
- 3 for 'TUNER_AUTO_FREEZE' (if tuner capable of DTV_AGC_AUTO_FREEZE)

fef_mode value check
FEF freeze interconnection is set by HW design and constrained by the tuner capabilities
Check if the FEF pin is wired in the diagram (if yes, use 'FREEZE_PIN'), or if the TER tuner has the SLOW_INITIAL capability in the TER tuner datasheet (if yes, use 'SLOW_INITIAL').
If none of the above is possible, use 'SLOW_NORMAL.

- ***fef_pin***

fef_pin selects the DTV demodulator output pin used to freeze the TER tuner AGC during DVB-T2 FEF frames.

NB: Only important when using the 'FREEZE_PIN' fef_mode.

fef_pin possible values
- 0 for 'unused'
- 0xa for 'MP_A'
- 0xb for 'MP_B'

- 0xc for ‘MP_C’
- 0xd for ‘MP_D’

fef_pin value check
FEF freeze interconnection is set by HW design and constrained by the tuner capabilities
Check which pin is used for FEF Freeze on the DTV demodulator side.

- ***fef_level***
fef_level is the logical level present on the output pin used to freeze the TER tuner AGC during FEF frames.

Only important when using the ‘FREEZE_PIN’ mode.

fef_level possible values
- 0 for ‘active low’
- 1 for ‘active high’

fef_level value check
FEF freeze interconnection is set by HW design and constrained by the tuner capabilities
Check the FEF pin wiring (if used), and select the value corresponding to the tuner freeze settings (set using SiLabs_TER_Tuner_FEF_FREEZE_PIN_SETUP in the TER tuner wrapper).

8.3.7 SiLabs_API_TER_Tuner_AGC_Input (front_end, dtv_agc_source);

SiLabs_API_TER_Tuner_AGC_Input is used to configure the TER tuner pin used for DTV AGC inside the TER tuner.
This is required because recent Skyworks TER tuners feature several possible DTV AGC input pins.

NB: Note the functions named with ‘TER Tuner’ are configuring the TER tuner, not the DTV demodulator.

- ***dtv_agc_source***
dtv_agc_source is the TER tuner pin selected for TER AGC control, if the TER AGC is controlled by the DTV demodulator.

dtv_agc_source possible values
Check in TER tuner API for values
Look into SiLabs_TER_Tuner_DTV_AGC_SOURCE to see how the value is written to the TER tuner, since this may depend on the TER tuner. This will lead you to the TER tuner API, where you will find the possible values.

The TER AGC ‘internal’ control is also selectable using dtv_agc_source.

dtv_agc_source value check
Check on which pin of the TER tuner the TER AGC is connected to (may be non-connected).

Runtime check: If AGC settings are incorrect, the AGC status will be either 0 or 255, no intermediate value, and locking may be difficult. It may still work on a limited input power range, though.

8.3.8 SiLabs_API_TER_AGC (front_end, agc1_mode, agc1_inversion, agc2_mode, agc2_inversion);

SiLabs_API_TER_AGC is used to configure the TER AGC in the DTV demodulator.

- ***agc1_mode***

agc1_mode selects the DTV demodulator pin used to send the TER AGC to the TER tuner, if using agc1 in the DTV demodulator (there are 2 AGC loops available in the DTV demodulators).
When not using agc1 for TER reception, use ‘0’.

agc1_mode possible values

- 0 for 'unused'
- 0xa for 'MP_A'
- 0xb for 'MP_B'
- 0xc for 'MP_C'
- 0xd for 'MP_D'

agc1_mode value check
Check from which pin of the DTV demodulator the TER tuner AGC is output (it may be non-connected).

- ***agc1_inversion***

agc1_inversion controls the TER AGC signal inversion, if using agc1 in the DTV demodulator for TER reception.
When 'agc1_mode' is '0', this value is unused.

agc1_inversion possible values
- 0 for 'non-inverted'
- 1 for 'inverted'

agc1_inversion value check
Check the connection of the TER AGC pin, if it has a pull-up or pull-down. Also check the TER tuner AGC characteristics.

Runtime check: If AGC settings are incorrect, the AGC status will be either 0 or 255, no intermediate value, and locking may be difficult. If the agc1_inversion is incorrect, the DVB-C blindscan will only detect a limited number of channels, with incorrect frequencies.

- ***agc2_mode***

agc2_mode selects the DTV demodulator pin used to send the TER AGC to the TER tuner, if using agc2 in the DTV demodulator (there are 2 AGC loops available in the DTV demodulators).
When not using agc2 for TER reception, use '0'.

agc2_mode possible values
- 0 for 'unused'
- 0xa for 'MP_A'
- 0xb for 'MP_B'
- 0xc for 'MP_C'
- 0xd for 'MP_D'

agc2_mode value check
Check from which pin of the DTV demodulator the TER tuner AGC is output (it may be non-connected).

- ***agc2_inversion***

agc2_inversion controls the TER AGC signal inversion, if using agc2 in the DTV demodulator for TER reception.
When 'agc2_mode' is '0', this value is unused.

agc2_inversion possible values
- 0 for 'non-inverted'
- 1 for 'inverted'

Agc2_inversion value check
Check the connection of the TER AGC pin, if it has a pull-up or pull-down. Also check the TER tuner AGC characteristics.

Runtime check: If AGC settings are incorrect, the AGC status will be either 0 or 255, no intermediate value, and locking may be difficult. If the agc2_inversion is incorrect, the DVB-C blindscan will only detect a limited number of channels, with incorrect frequencies.

8.3.9 SiLabs_API_TER_Tuner_IF_Output (front_end, dtv_out_type);

SiLabs_API_TER_Tuner_IF_Output is used to configure the TER tuner pin used for IF connection to the DTV demodulator inside the TER tuner.
The IF signal goes from the TER tuner to the DTV demodulator.
This is required because recent Skyworks TER tuners feature several possible DTV IF output pins.

The value set here will ultimately be sent to SiLabs_TER_Tuner_DTV_OUT_TYPE during the HW initialization.

There is a single TER IF input pin on the DTV demodulator side, so no configuration is required on the DTV demodulator as far as TER IF is concerned.

NB: Note the functions named with 'TER Tuner' are configuring the TER tuner, not the DTV demodulator.

- ***dtv_out_type***
  dtv_out_type is the TER tuner pin used to output the TER IF signal to the DTV demodulator.

  dtv_out_type possible values
  Look into SiLabs_TER_Tuner_API.c/ SiLabs_TER_Tuner_DTV_OUT_TYPE to see how the dtv_out_type value is used to setup the TER IF output in the TER tuner (this depends on the TER tuner API). Unfortunately, different TER tuners use different APIs for DTV IF selection, so it's required here to check the code for the possible dtv_out_type values corresponding to each TER tuner.

  dtv_out_type value check
  In the diagram, check on the TER tuner side which pin is used to output the TER IF signal to the DTV demodulator.

8.4 SAT configuration functions

The following functions are only required when building and application for SAT reception.
When not using SAT, they can be skipped during SW configuration.

8.4.1 SiLabs_API_Select_SAT_Tuner (front_end, sat_tuner_code, sat_tuner_index);

SiLabs_API_Select_SAT_Tuner is used to select which SAT tuner is in use.

- ***sat_tuner_code***
  sat_tuner_code is the code used for the SAT tuner selection, easily readable when in hexadecimal, as it matches the part name, i.e. '0x5812' for RDA5812. In our example we use '0xA2018'.

  sat_tuner_code possible values
  Possible values (as of writing):
  - 0xA2012 for AV2012
  - 0xA2017 for AV2017 (ultimately using AV2018 driver)
  - 0xA2018 for AV2018
  - SAT_TUNER_CUSTOMSAT_CODE, for a custom SAT driver accessed through the CUSTOMSAT driver
  - 0x2112 for MAX2112
  - 0x20142 for NXP TDA20142
  - 0x5812 for RDA5812
  - 0x5815 for RDA5815 / RDA5815S / RDA16110
  - 0x58150 for RDA5815M / RDA16112
  - 0x5816, for RDA5816S / RDA16110D
  - 0x58165D for RDA5816SD / RDA16110SW

```
int    SiLabs_SAT_Tuner_Select_Tuner     (SILABS_SAT
  SiTRACE_X("Select SAT Tuner with code 0x%x[%d]\
#ifdef     SAT_TUNER_AV2012
#ifdef     SAT_TUNER_AV2018
  if (sat_tuner_code                     == 0xA2018 ) {
#endif /* SAT_TUNER_AV2018 */
#ifdef     SAT_TUNER_CUSTOMSAT
#ifdef     SAT_TUNER_MAX2112
#ifdef     SAT_TUNER_NXP20142
#ifdef     SAT_TUNER_RDA5812
  if (sat_tuner_code                     == 0x5812 ) {
#endif /* SAT_TUNER_RDA5812 */
#ifdef     SAT_TUNER_RDA5815
#ifdef     SAT_TUNER_RDA5815M
#ifdef     SAT_TUNER_RDA5816S
#ifdef     SAT_TUNER_RDA5816SD
  if (silabs_tuner->sat_tuner_code != 0) {
  return (silabs_tuner->sat_tuner_code<<8)+silabs
}
```

  sat_tuner_code value check
  In SiLabs_SAT_Tuner_API.c, function SiLabs_SAT_Tuner_Select_Tuner, look for the possible sat_tuner_code values.
  In the diagram, check the SAT tuner part number.

- ***sat_tuner_index***

sat_tuner_index is a provision in case there would be an application using more than 1 SAT tuner per DTV demodulator. Unless this situation exists, the sat_tuner_index value is 0.

8.4.2 SiLabs_API_SAT_Select_LNB_Chip (front_end, lnb_code, lnb_chip_address);

SiLabs_API_SAT_Select_LNB_Chip is used to select which SAT LNB controller part is used, as well as its I2C address.

- ***lnb_code***
  lnb_code is the code used to select the SAT LNB controller part, easily readable (in decimal or hexadecimal, for historical reasons) as it matches the part name, i.e. '21' for LNBH21. In our example we use '25'.

lnb_code possible values

- 21 for LNBH21
- 25 for LNBH25
- 26 for LNBH26
- 29 for LNBH29
- 0xA8293 for A8293
- 0xA9297 for A8297
- 0xA8302 for A8302
- 0xA8304 for A8304

lnb_code value check
In SiLabs_API_L3_Wrapper.c, function SiLabs_API_SAT_Select_LNB_Chip, look for the possible lnb_code values.
In the diagram, check the SAT LNB controller part number.

- ***chip_address***
  chip_address is the SAT LNB controller's I2C address.

chip_address possible values
This depends on the SAT LNB chip in use. Check its datasheet.

chip_address value check
In the diagram, check the address configuration of the SAT LNB controller part.

8.4.3 SiLabs_API_SAT_LNB_Chip_Index (front_end, lnb_index);

SiLabs_API_SAT_LNB_Chip_Index is used to select the LNB controller chip index, when using a dual part.
So far, only LNBH26 and A8302 are dual LNB controllers.
If the SAT LNB controller in use is not a dual, this function is not required in the SW configuration.

- ***lnb_index***
  lnb_index is used to select the 'side' of the SAT LNB controller is used.

lnb_index possible values
Check SiLabs_API_SAT_LNB_Chip_Index and the underlying SAT LNB controller driver for possible values.
Since LNB controllers have very different APIs, checking the LNB driver is required here.

lnb_index value check
In the diagram, check which 'side' of the dual SAT LNB controller is used for the current frontend.

8.4.4 SiLabs_API_SAT_tuner_I2C_connection (front_end, fe_index);

SiLabs_API_SAT_tuner_I2C_connection is used to select which demodulator pass-through needs to be used to connect with the SAT tuner I2C.

It's only active if ***INDIRECT_I2C_CONNECTION*** is declared in the compilation flags.

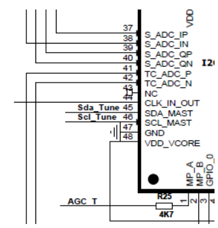
Background information related to tuner I2c connection

Dual demodulator includes an I2C switch to reduce the potential noise on satellite and terrestrial tuners during I2C accesses.
When using this I2C switch, SDA_MAST and SCL_MAST have to be connected to tuners I2C.
To prevent any conflict between both dies, I2C switch shall be controlled only by one demodulator, usually demodulator A.

- ***fe_index***
  fe_index is the index of the frontend whose demodulator i2c pass-through must be closed to access the SAT tuner.

  For single designs this is generally the same value as the frontend index (i.e. 0).
  For duals, the second frontend tuners are often connected through the first demodulator. In this case, set fe_index to '0' for frontend[0] as well as frontend[1].
  For quads, values are often: '0' for frontend[0], '0' for frontend[1], '1' for frontend[2], '1' for frontend[3]

  fe_index possible values
  - *0 for single designs*
  - *0, 1 for dual designs*
  - *0, 1, 2 for triple designs*
  - *0, 1, 2, 3 for quad designs*



  fe_index value check
  Check the i2c routing diagram, this is set by HW design.
  Check the i2c connections between demodulator(s) and TER tuners,
  When using dual demodulators, tuners are generally connected through demodulator 'A'.

8.4.5 SiLabs_API_SAT_Clock (front_end, clock_source, clock_input, clock_freq, clock_control);

SiLabs_API_SAT_Clock is used to configure the clock path used for SAT reception by the DV demodulator. **The source of the clock can be a xtal, the SAT tuner or the TER tuner**.

- ***clock_source***
  clock_source is the origin of the clock when locking in SAT.
  This is used to know when to initialize the clock source, start the clock if it's used, and when to set the clock in standby.
  clock_source is taken into account together with clock_control to decide when and if the clock can be stopped.

  clock_source possible values
  - 0 for 'xtal'
  - 1 for 'TER tuner'
  - 2 for 'SAT tuner'

  clock_source value check
  In the diagram, check where the clock used for SAT reception comes from.

- ***clock_input***
  clock_input is the clock input used by the demodulator for SAT reception.

  NB: This is coded as pin numbers corresponding to single demodulator pinout, in an attempt to make reading the SW configuration easier.
  When using dual demodulators, use the same numbers as for single parts.

  clock_input possible values
  - 44 for 'CLKIO'
  - 33 for 'Xtal_IN'
  - 32 for 'Xtal' (driven by DTV demodulator)

  clock_input value check

In the diagram, check the clock input pin used on the demodulator for SAT reception.

- ***clock_freq***
  clock_freq is the clock frequency used by the demodulator for SAT reception.

  clock_freq possible values
  - 16 for 16 MHz
  - 24 for 24 MHz
  - 27 for 27 MHz

  clock_freq value check
  In the diagram, check the clock frequency of the Xtal used as the clock source.

- ***clock_control***
  clock_control is the control mode for the SAT tuner clock output.
  When the clock is used by another frontend, it must be 'always on'.
  When the clock is never used, it must be 'always off'.
  When the application should control it, it must be 'managed'

  clock_control possible values
  - 0 for 'ALWAYS_OFF'
  - 1 for 'ALWAYS_ON'
  - 2 for 'MANAGED'

  clock_control value check
  In the diagram, check if the SAT tuner clock is used by another frontend. If yes, it should be kept 'ALWAYS ON' (use 1).
  Check if the SAT tuner clock is not going anywhere. If yes, it should be kept 'ALWAYS OFF' (use 0).
  If you want the code to turn it ON/OFF when it's used/unused, use 2 ('MANAGED').

8.4.6 SiLabs_API_SAT_Spectrum (front_end, spectrum_inversion);

SiLabs_API_SAT_Spectrum is used to configure the SAT ZIF spectrum inversion.

- ***spectrum_inversion***
  spectrum_inversion is a flag indicating if the SAT signal appears inverted to the DTV demodulator.
  Note: I/Q swap on the SAT ZIF lines will result in an artificial SAT spectrum inversion.
  This situation is possible if there is an I/Q swap in the HW design, for easier routing
  It is perfectly OK to do that to route easily, but the SW needs to be informed of this inversion.
  NB: The additional inversion added by Unicable equipment should not be taken into account here. The spectrum_inversion set here corresponds to the 'normal' mode.

  spectrum_inversion possible values
  - 0 for 'non-inverted'
  - 1 for 'inverted'

  spectrum_inversion value check
  In the diagram, check the connection of the SAT I/Q signals. A swap between I/Q signals between the SAT tuner and the demodulator is allowed, but it requires inverting the spectrum_inversion flag.
  Runtime check: If the spectrum_inversion is incorrect, the SAT blindscan will only detect a limited number of channels, with incorrect frequencies, and later locking on these channels will be difficult and will lead to a huge frequency offset being displayed in the status.

8.4.7 SiLabs_API_SAT_AGC (front_end, agc1_mode, agc1_inversion, agc2_mode, agc2_inversion);

SiLabs_API_SAT_AGC is used to configure the SAT AGC in the DTV demodulator.

- ***agc1_mode***

agc1_mode selects the DTV demodulator pin used to send the SAT AGC to the SAT tuner, if using agc1 in the DTV demodulator (there are 2 AGC loops available in the DTV demodulators).
When not using agc1 for SAT reception, use ‘0’.

agc1_mode possible values
- 0 for ‘unused’
- 0xa for ‘MP_A’
- 0xb for ‘MP_B’
- 0xc for ‘MP_C’
- 0xd for ‘MP_D’

agc1_mode value check
Check from which pin of the DTV demodulator the SAT tuner AGC is output (it may be non-connected).

- ***agc1_inversion***

agc1_inversion controls the SAT AGC signal inversion, if using agc1 in the DTV demodulator for SAT reception.
When ‘agc1_mode’ is ‘0’, this value is unused.

agc1_inversion possible values
- 0 for ‘non-inverted’
- 1 for ‘inverted’

agc1_inversion value check
Check the connection of the SAT AGC pin, if it has a pull-up or pull-down. Also check the SAT tuner AGC characteristics.

Runtime check: If AGC settings are incorrect, the AGC status will be either 0 or 255, no intermediate value, and locking may be difficult.

- ***agc2_mode***

agc2_mode selects the DTV demodulator pin used to send the SAT AGC to the SAT tuner, if using agc2 in the DTV demodulator (there are 2 AGC loops available in the DTV demodulators).
When not using agc2 for SAT reception, use ‘0’.

agc2_mode possible values
- 0 for ‘unused’
- 0xa for ‘MP_A’
- 0xb for ‘MP_B’
- 0xc for ‘MP_C’
- 0xd for ‘MP_D’

agc2_mode value check
Check from which pin of the DTV demodulator the SAT tuner AGC is output (it may be non-connected).

- ***agc2_inversion***

agc2_inversion controls the SAT AGC signal inversion, if using agc2 in the DTV demodulator for SAT reception.
When ‘agc2_mode’ is ‘0’, this value is unused.

agc2_inversion possible values
- 0 for ‘non-inverted’
- 1 for ‘inverted’

Agc2_inversion value check
Check the connection of the SAT AGC pin, if it has a pull-up or pull-down. Also check the SAT tuner AGC characteristics.

Runtime check: If AGC settings are incorrect, the AGC status will be either 0 or 255, no intermediate value, and locking may be difficult.

8.5 General completion configuration functions

8.5.1 SiLabs_API_Set_Index_and_Tag (front_end, index, tag);

SiLabs_API_Set_Index_and_Tag is used to improve tracing during application development and debug.

Since an increasing number of applications are dealing with 'multiple' frontend designs, it's very interesting to be able to identify the source (i.e. which frontend) of the traces.

When tracing the tag is enabled (see the L0 documentation for further details on 'SiTRACES', the Skyworks traces), each trace line generated by a frontend will contain the custom 'tag' followed by 'DTV' 'TER' or 'SAT' depending on the case, such that it can be easily identified as coming from this frontend.

Usually, tags will be of the form "fe[0]", but they can be customized to use any text less than SILABS_TAG_SIZE characters long (SILABS_TAG_SIZE is 20 by default).

- ***index***
  index is currently not used.
- ***tag***
  tag is a text string which must be smaller than SILABS_TAG_SIZE characters long (SILABS_TAG_SIZE is 20 by default). It can be any text, as long as it easily helps identifying the frontends.

8.5.2 SiLabs_API_HW_Connect (front_end, connection_mode);

SiLabs_API_HW_Connect is used to connect all elements of the current frontend in the selected mode.
The Skyworks I2c Layer (the 'L0') allows dynamically switching between several communication modes.

Which communication modes are available depend on the platform (Windows / Linux / other) and on the HW capabilities.

- ***connection_mode***

connection_mode is the current communication mode.
It will be applied to all parts in the frontend (DTV demodulator, TER tuner, SAT tuner, SAT LNB Controller).

connection_mode possible values

- 0 for 'SIMU'
- 1 for 'USB'
- 2 for 'CUSTOMER'
- 3 for 'LINUX_I2C' (Linux I2C using custom I2C functions to be ported by customer)
- 4 for 'LINUX_USB' (Linux userspace I2C using the Cypress FX2LP chip as interface)
- 5 for 'LINUX_KERNEL_SDK2_I2C' (Linux kernelspace I2C based on STM SDK2 kernel i2c functions)
- *Any other value which can be added during i2c porting.*

connection_mode value check
The communication mode to select depends on the platform.
Obviously, when the communication mode is incorrect I2C communication will fail, and it will not be possible to perform the HW initialization.
It may be useful to enable the L0 (byte level) traces and check the text corresponding to I2C messages during execution.
This text should correspond to the selected communication mode.
When properly ported, I2c communication should work and the traced bytes should correspond to the actual I2C signals on the bus.

8.5.3 SiLabs_API_Cypress_Ports (front_end, OEA, IOA, OEB, IOB, OED, IOD);

SiLabs_API_Cypress_Ports is a very specific function used on Skyworks EVBs to set up the Cypress FX2LP (USB/I2C interface) part according to the EVB design.
In the package used on Skyworks EVBs, the FX2LP provides access to three 8 bit ports: A, B and D.

For each of these buses, it's possible to enable/disable the port outputs (using the OEx = 'Output Enable A/B/C' byte).
Each 'OEx.n' bit set to '1' configures the corresponding pin (port x bit n) as an output. Otherwise, the pin can be used as an input to check the current level on the pin.

When a given pin is enabled as an output, the corresponding (IOx.n = 'Input Output A/B/C'.n) bit selects the logical level on the pin.

Annex i Makefile content

```
# ===============================================================================
# | The present makefile is used for SKYWORKS SUPERSET compilation under Windows  |
# | It can be used to compile using                                              |
# | "C:\Program Files (x86)\CodeBlocks\MinGW\bin\mingw32-make.exe"               |
# | Look for '+++<porting>+++' to locate areas where changes may be required     |
# | Such areas are ending with '---<porting>---'                                 |
# ===============================================================================

CC = gcc.exe
LD = g++.exe

I2C_DIR       = ..\\..\\Si_I2C
DTV_DIR       = ..\\..\\Si2183
SAT_DIR       = ..\\..\\SAT
TER_DIR       = ..\\..\\TER

LIB           = $(I2C_DIR)\\CypressUSB.lib
OBJDIR_DEBUG  = obj\\Debug
OUT_DEBUG     = ..\\bin\\Debug\\SKYWORKS_SUPERSET.exe

#==============================================================
#include path for skyworks_superset internal files
#--------------------------------------------------------------
ccflags-y+= -I$(I2C_DIR)
ccflags-y+= -I$(DTV_DIR)
#==============================================================

#==============================================================
# General compilation flags for skyworks_superset
#--------------------------------------------------------------
ccflags-y+= -Wall
ccflags-y+= -DSILABS_SUPERSET
ccflags-y+= -DFRONT_END_COUNT=1
ccflags-y+= -DCONFIG_MACROS -DSILABS_EVB_MACROS
ccflags-y+= -DTS_CROSSBAR
#==============================================================


#==============================================================
# +++<porting>+++ possible FWs to load in demodulators
#                  (depending on part number)
# Several values can be used together, for compatibility
#   with several versions.
# The more FW files you include, the bigger the code size
# Check Si2183_L2_API.c file for possible flags
#  (between change log and first function implementation)
#--------------------------------------------------------------
ccflags-y+= -DSi2180_A55_COMPATIBLE
ccflags-y+= -DSi2180_A50_COMPATIBLE
ccflags-y+= -DSi2183_B60_COMPATIBLE
ccflags-y+= -DSi2183_B5A_COMPATIBLE
ccflags-y+= -DSi2183_A55_COMPATIBLE
ccflags-y+= -DSi2183_A50_COMPATIBLE
# ---<porting>--- End of FWs selection
#==============================================================

#==============================================================
# General SKYWORKS SUPERSET source objects
#--------------------------------------------------------------
OBJFILES += $(OBJDIR_DEBUG)\\Si_I2C\\Silabs_L0_Connection.o
OBJFILES += $(OBJDIR_DEBUG)\\Si2183\\Si2183_L1_API.o
OBJFILES += $(OBJDIR_DEBUG)\\Si2183\\Si2183_L1_Commands.o
OBJFILES += $(OBJDIR_DEBUG)\\Si2183\\Si2183_L1_Properties.o
OBJFILES += $(OBJDIR_DEBUG)\\Si2183\\Si2183_L2_API.o
OBJFILES += $(OBJDIR_DEBUG)\\Si2183\\SiLabs_API_L3_Wrapper_TS_Crossbar.o
OBJFILES += $(OBJDIR_DEBUG)\\Si2183\\SiLabs_API_L3_Wrapper.o
OBJFILES += $(OBJDIR_DEBUG)\\Si2183\\SiLabs_API_L3_Config_Macros.o
OBJFILES += $(OBJDIR_DEBUG)\\Si2183\\SiLabs_API_L3_Console.o
#==============================================================

#==============================================================
#        TER compilation flags
#--------------------------------------------------------------
# +++<porting>+++ TER tuner selection
#                  (use 'none' for no_TER compilation)
# Check TER/SiLabs_TER_Tuner_API.c/SiLabs_TER_Tuner_SW_Init
#           for possible values
# Also add the corresponding TER tuner code under TER/
#  the current TER tuner being selected in the SW configuration
#  using SiLabs_API_Select_TER_Tuner
#--------------------------------------------------------------
TER_TUNER=Si2141
# ---<porting>--- End of TER tuner selection
#--------------------------------------------------------------
ifneq (none, $(TER_TUNER))
#--------------------------------------------------------------
# +++<porting>+++        TER standards selection
#                  (comment unused standards)
# Restrictions related to the TER standards selection:
#     If DVB-T2 support is required, DVB-T support is also mandatory
#     If DVB-C2 support is required, DVB-C support is also mandatory
#     If MCNS   support is required, DVB-C support is also mandatory
ccflags-y+=-DDEMOD_DVB_T   # for DVBT
ccflags-y+=-DDEMOD_DVB_T2  # for DVBT2
ccflags-y+=-DDEMOD_DVB_C   # for DVBC_ANNEX_AC
ccflags-y+=-DDEMOD_DVB_C2  # for DVBC2
ccflags-y+=-DDEMOD_MCNS    # for DVBC_ANNEX_B
ccflags-y+=-DDEMOD_ISDB_T  # for ISDBT
# ---<porting>--- End of TER standards selection
#--------------------------------------------------------------
#include path for skyworks_superset TER files
ccflags-y+= -I$(TER_DIR)
ccflags-y+= -I$(TER_DIR)\\$(TER_TUNER)
```

```
# TER compilation flags
ccflags-y+= -DTERRESTRIAL_FRONT_END -DTER_TUNER_SILABS -DTER_TUNER_$(TER_TUNER)
#object files for TER tuner
OBJFILES += $(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)\\$(TER_TUNER)_L1_API.o
OBJFILES += $(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)\\$(TER_TUNER)_L1_Commands.o
OBJFILES += $(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)\\$(TER_TUNER)_L1_Properties.o
OBJFILES += $(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)\\$(TER_TUNER)_Properties_Strings.o
OBJFILES += $(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)\\$(TER_TUNER)_User_Properties.o
OBJFILES += $(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)\\$(TER_TUNER)_L2_API.o
OBJFILES += $(OBJDIR_DEBUG)\\TER\\SiLabs_TER_Tuner_API.o
endif
#-------------------------------------------------------------
# End of TER compilation flags
#=============================================================


#=============================================================
#        SAT compilation flags
#-------------------------------------------------------------
# +++<porting>+++        SAT tuner selection
#                        (use 'none' for no_SAT compilation)
# Check SAT\\SiLabs_SAT_Tuner_API.c\\SiLabs_SAT_Tuner_SW_Init
#          for possible values
# Also add the corresponding SAT tuner code under SAT\\
#  the current SAT tuner being selected in the SW configuration
#  using SiLabs_API_Select_SAT_Tuner
SAT_TUNER=AV2018
SAT_LNB=LNBH25
# ---<porting>--- End of SAT tuner selection
#-------------------------------------------------------------
ifneq (none, $(SAT_TUNER))
#-------------------------------------------------------------
# +++<porting>+++        SAT standards selection
#                       (only one option: comment DVB-S2X if unused)
ccflags-y+= -DDEMOD_DVB_S_S2_DSS  # for DVBS, DVBS2 and DSS
ccflags-y+= -DDEMOD_DVB_S2X       # if support for DVB-S2X is required
# ---<porting>--- End of SAT standards selection
#-------------------------------------------------------------
#include path for skyworks_superset SAT files
ccflags-y+= -I$(SAT_DIR)
ccflags-y+= -I$(SAT_DIR)\\Unicable
ccflags-y+= -I$(SAT_DIR)\\$(SAT_TUNER)
# SAT compilation flags
ccflags-y+= -DSATELLITE_FRONT_END -DSAT_TUNER_SILABS -DSAT_TUNER_$(SAT_TUNER)
ccflags-y+= -DUNICABLE_COMPATIBLE
#object file for SAT tuner
OBJFILES += $(OBJDIR_DEBUG)\\SAT\\$(SAT_TUNER)\\SiLabs_L1_RF_$(SAT_TUNER)_API.o
#object file for SAT tuner wrapper
OBJFILES += $(OBJDIR_DEBUG)\\SAT\\SiLabs_SAT_Tuner_API.o
#object file  for SAT Unicable
OBJFILES += $(OBJDIR_DEBUG)\\SAT\\Unicable\\SiLabs_Unicable_API.o
endif

ifneq (none,  $(SAT_LNB))
ccflags-y+= -D$(SAT_LNB)_COMPATIBLE
ccflags-y+= -I$(SAT_DIR)\\$(SAT_LNB)

OBJFILES +=   $(OBJDIR_DEBUG)\\SAT\\$(SAT_LNB)\\$(SAT_LNB)_L1_API.o
endif
#-------------------------------------------------------------
# End of SAT compilation flags
#=============================================================

CFLAGS_DEBUG  = $(ccflags-y) -g
LIB_DEBUG     = $(LIB)

all: debug

clean: clean_debug

before_debug:
            cmd /c if not exist ..\\bin\\Debug                     md ..\\bin\\Debug
            cmd /c if not exist $(OBJDIR_DEBUG)\\Si2183            md $(OBJDIR_DEBUG)\\Si2183
            cmd /c if not exist $(OBJDIR_DEBUG)\\Si_I2C            md $(OBJDIR_DEBUG)\\Si_I2C
ifneq (none, $(TER_TUNER))
            cmd /c if not exist $(OBJDIR_DEBUG)\\TER               md $(OBJDIR_DEBUG)\\TER
            cmd /c if not exist $(OBJDIR_DEBUG)\\TER\\$(TER_TUNER) md $(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)
endif
ifneq (none, $(SAT_TUNER))
            cmd /c if not exist $(OBJDIR_DEBUG)\\SAT               md $(OBJDIR_DEBUG)\\SAT
            cmd /c if not exist $(OBJDIR_DEBUG)\\SAT\\$(SAT_TUNER) md $(OBJDIR_DEBUG)\\SAT\\$(SAT_TUNER)
            cmd /c if not exist $(OBJDIR_DEBUG)\\SAT\\$(SAT_LNB)   md $(OBJDIR_DEBUG)\\SAT\\$(SAT_LNB)
            cmd /c if not exist $(OBJDIR_DEBUG)\\SAT\\Unicable     md $(OBJDIR_DEBUG)\\SAT\\Unicable
endif

after_debug:

debug: before_debug out_debug after_debug

test:  before_debug out_test  after_debug

out_debug: before_debug  $(OBJFILES)
            $(LD) -o $(OUT_DEBUG)  $(OBJFILES) $(LIB_DEBUG)

$(OBJDIR_DEBUG)\\Si2183\\SiLabs_API_L3_Wrapper.o: $(DTV_DIR)\\SiLabs_API_L3_Wrapper.c
            $(CC) $(CFLAGS_DEBUG) -c $(DTV_DIR)\\SiLabs_API_L3_Wrapper.c -o $(OBJDIR_DEBUG)\\Si2183\\SiLabs_API_L3_Wrapper.o

$(OBJDIR_DEBUG)\\Si2183\\SiLabs_API_L3_Wrapper_TS_Crossbar.o: $(DTV_DIR)\\SiLabs_API_L3_Wrapper_TS_Crossbar.c
            $(CC) $(CFLAGS_DEBUG) -c $(DTV_DIR)\\SiLabs_API_L3_Wrapper_TS_Crossbar.c -o $(OBJDIR_DEBUG)\\Si2183\\SiLabs_API_L3_Wrapper_TS_Crossbar.o

$(OBJDIR_DEBUG)\\Si_I2C\\Silabs_L0_Connection.o: $(I2C_DIR)\\Silabs_L0_Connection.c
            $(CC) $(CFLAGS_DEBUG) -c $(I2C_DIR)\\Silabs_L0_Connection.c -o $(OBJDIR_DEBUG)\\Si_I2C\\Silabs_L0_Connection.o

$(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)\\$(TER_TUNER)_L1_API.o: $(TER_DIR)\\$(TER_TUNER)\\$(TER_TUNER)_L1_API.c
            $(CC) $(CFLAGS_DEBUG) -c $(TER_DIR)\\$(TER_TUNER)\\$(TER_TUNER)_L1_API.c -o $(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)\\$(TER_TUNER)_L1_API.o

$(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)\\$(TER_TUNER)_L1_Commands.o: $(TER_DIR)\\$(TER_TUNER)\\$(TER_TUNER)_L1_Commands.c
            $(CC) $(CFLAGS_DEBUG) -c $(TER_DIR)\\$(TER_TUNER)\\$(TER_TUNER)_L1_Commands.c -o $(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)\\$(TER_TUNER)_L1_Commands.o
```

```
$(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)\\$(TER_TUNER)_L1_Properties.o: $(TER_DIR)\\$(TER_TUNER)\\$(TER_TUNER)_L1_Properties.c
	$(CC) $(CFLAGS_DEBUG) -c $(TER_DIR)\\$(TER_TUNER)\\$(TER_TUNER)_L1_Properties.c -o $(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)\\$(TER_TUNER)_L1_Properties.o

$(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)\\$(TER_TUNER)_L2_API.o: $(TER_DIR)\\$(TER_TUNER)\\$(TER_TUNER)_L2_API.c
	$(CC) $(CFLAGS_DEBUG) -c $(TER_DIR)\\$(TER_TUNER)\\$(TER_TUNER)_L2_API.c -o $(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)\\$(TER_TUNER)_L2_API.o

$(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)\\$(TER_TUNER)_Properties_Strings.o: $(TER_DIR)\\$(TER_TUNER)\\$(TER_TUNER)_Properties_Strings.c
	$(CC) $(CFLAGS_DEBUG) -c $(TER_DIR)\\$(TER_TUNER)\\$(TER_TUNER)_Properties_Strings.c -o
$(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)\\$(TER_TUNER)_Properties_Strings.o

$(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)\\$(TER_TUNER)_User_Properties.o: $(TER_DIR)\\$(TER_TUNER)\\$(TER_TUNER)_User_Properties.c
	$(CC) $(CFLAGS_DEBUG) -o $(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)\\$(TER_TUNER)_User_Properties.o -c $(TER_DIR)\\$(TER_TUNER)\\$(TER_TUNER)_User_Properties.c

$(OBJDIR_DEBUG)\\TER\\SiLabs_TER_Tuner_API.o: $(TER_DIR)\\SiLabs_TER_Tuner_API.c
	$(CC) $(CFLAGS_DEBUG) -c $(TER_DIR)\\SiLabs_TER_Tuner_API.c -o $(OBJDIR_DEBUG)\\TER\\SiLabs_TER_Tuner_API.o

$(OBJDIR_DEBUG)\\SAT\\$(SAT_TUNER)\\SiLabs_L1_RF_$(SAT_TUNER)_API.o: $(SAT_DIR)\\$(SAT_TUNER)\\SiLabs_L1_RF_$(SAT_TUNER)_API.c
	$(CC) $(CFLAGS_DEBUG) -c $(SAT_DIR)\\$(SAT_TUNER)\\SiLabs_L1_RF_$(SAT_TUNER)_API.c -o $(OBJDIR_DEBUG)\\SAT\\$(SAT_TUNER)\\SiLabs_L1_RF_$(SAT_TUNER)_API.o

$(OBJDIR_DEBUG)\\SAT\\$(SAT_LNB)\\$(SAT_LNB)_L1_API.o: $(SAT_DIR)\\$(SAT_LNB)\\$(SAT_LNB)_L1_API.c
	$(CC) $(CFLAGS_DEBUG) -c $(SAT_DIR)\\$(SAT_LNB)\\$(SAT_LNB)_L1_API.c -o $(OBJDIR_DEBUG)\\SAT\\$(SAT_LNB)\\$(SAT_LNB)_L1_API.o

$(OBJDIR_DEBUG)\\SAT\\SiLabs_SAT_Tuner_API.o: $(SAT_DIR)\\SiLabs_SAT_Tuner_API.c
	$(CC) $(CFLAGS_DEBUG) -c $(SAT_DIR)\\SiLabs_SAT_Tuner_API.c -o $(OBJDIR_DEBUG)\\SAT\\SiLabs_SAT_Tuner_API.o

$(OBJDIR_DEBUG)\\SAT\\Unicable\\SiLabs_Unicable_API.o: $(SAT_DIR)\\Unicable\\SiLabs_Unicable_API.c
	$(CC) $(CFLAGS_DEBUG) -c $(SAT_DIR)\\Unicable\\SiLabs_Unicable_API.c -o $(OBJDIR_DEBUG)\\SAT\\Unicable\\SiLabs_Unicable_API.o

$(OBJDIR_DEBUG)\\Si2183\\Si2183_L1_API.o: $(DTV_DIR)\\Si2183_L1_API.c
	$(CC) $(CFLAGS_DEBUG)   -c $(DTV_DIR)\\Si2183_L1_API.c -o $(OBJDIR_DEBUG)\\Si2183\\Si2183_L1_API.o

$(OBJDIR_DEBUG)\\Si2183\\Si2183_L1_Commands.o: $(DTV_DIR)\\Si2183_L1_Commands.c
	$(CC) $(CFLAGS_DEBUG)        -c $(DTV_DIR)\\Si2183_L1_Commands.c -o $(OBJDIR_DEBUG)\\Si2183\\Si2183_L1_Commands.o

$(OBJDIR_DEBUG)\\Si2183\\Si2183_L1_Properties.o: $(DTV_DIR)\\Si2183_L1_Properties.c
	$(CC) $(CFLAGS_DEBUG)          -c $(DTV_DIR)\\Si2183_L1_Properties.c -o $(OBJDIR_DEBUG)\\Si2183\\Si2183_L1_Properties.o

$(OBJDIR_DEBUG)\\Si2183\\Si2183_L2_API.o: $(DTV_DIR)\\Si2183_L2_API.c
	$(CC) $(CFLAGS_DEBUG) -c $(DTV_DIR)\\Si2183_L2_API.c -o $(OBJDIR_DEBUG)\\Si2183\\Si2183_L2_API.o

$(OBJDIR_DEBUG)\\Si2183\\SiLabs_API_L3_Config_Macros.o: $(DTV_DIR)\\SiLabs_API_L3_Config_Macros.c
	$(CC) $(CFLAGS_DEBUG) -c $(DTV_DIR)\\SiLabs_API_L3_Config_Macros.c -o $(OBJDIR_DEBUG)\\Si2183\\SiLabs_API_L3_Config_Macros.o

$(OBJDIR_DEBUG)\\Si2183\\SiLabs_API_L3_Console.o: $(DTV_DIR)\\SiLabs_API_L3_Console.c
	$(CC) $(CFLAGS_DEBUG) -c $(DTV_DIR)\\SiLabs_API_L3_Console.c -o $(OBJDIR_DEBUG)\\Si2183\\SiLabs_API_L3_Console.o

clean_debug:
	cmd /c del /f $(OBJFILES) $(OUT_DEBUG)
	cmd /c if exist $(OBJDIR_DEBUG)\\Si2183             rd $(OBJDIR_DEBUG)\\Si2183
	cmd /c if exist $(OBJDIR_DEBUG)\\Si_I2C             rd $(OBJDIR_DEBUG)\\Si_I2C
	cmd /c if exist $(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)   rd $(OBJDIR_DEBUG)\\TER\\$(TER_TUNER)
	cmd /c if exist $(OBJDIR_DEBUG)\\TER                rd $(OBJDIR_DEBUG)\\TER
	cmd /c if exist $(OBJDIR_DEBUG)\\SAT\\$(SAT_TUNER)   rd $(OBJDIR_DEBUG)\\SAT\\$(SAT_TUNER)
	cmd /c if exist $(OBJDIR_DEBUG)\\SAT\\$(SAT_LNB)     rd $(OBJDIR_DEBUG)\\SAT\\$(SAT_LNB)
	cmd /c if exist $(OBJDIR_DEBUG)\\SAT                rd $(OBJDIR_DEBUG)\\SAT
	cmd /c if exist $(OBJDIR_DEBUG)\\SAT\\Unicable       rd $(OBJDIR_DEBUG)\\SAT\\Unicable


.PHONY: before_debug after_debug clean_debug
```

Adapting this makefile to other configurations:

Search for ‘<porting>’ to locate areas where you may need to apply changes.

To avoid compiling for TER reception, set TER_TUNER=none
To compile for another TER tuner, set TER_TUNER=<TER-tuner-name>
(Also add the new TER tuner code under TER)

To avoid compiling for SAT reception, set SAT_TUNER=none
To compile for another SAT tuner, set SAT_TUNER=<SAT-tuner-name>
(Also add the new SAT tuner code under SAT)

To compile for another SAT LNB, set SAT_LNB=<SAT-LNB-name>

Select the standards to be supported for TER and SAT by commenting lines in the respective TER and SAT standards selection areas.

Annex ii Compilation flags for each DTV demodulator

The table below allows copy/pasting the compilation flags for each part in project makefiles.

Parts	Compilation flags for all standards supported by the parts
Si2160 Si21602	-DSILABS_SUPERSET -DTERRESTRIAL_FRONT_END -DTER_TUNER_SILABS -DDEMOD_DVB_T -DDEMOD_DVB_C -DDEMOD_DVB_C2 -DDEMOD_MCNS -DSATELLITE_FRONT_END -DSAT_TUNER_SILABS -DDEMOD_DVB_S_S2_DSS -DDEMOD_DVB_S2X
Si2162 Si21622	-DSILABS_SUPERSET -DTERRESTRIAL_FRONT_END -DTER_TUNER_SILABS -DDEMOD_DVB_T -DDEMOD_DVB_T2 -DDEMOD_DVB_C -DDEMOD_DVB_C2 -DDEMOD_MCNS
Si2164 Si21642	-DSILABS_SUPERSET -DTERRESTRIAL_FRONT_END -DTER_TUNER_SILABS -DDEMOD_DVB_T -DDEMOD_DVB_T2 -DDEMOD_DVB_C -DDEMOD_DVB_C2 -DDEMOD_MCNS -DSATELLITE_FRONT_END -DSAT_TUNER_SILABS -DDEMOD_DVB_S_S2_DSS -DDEMOD_DVB_S2X
Si21652	-DSILABS_SUPERSET -DTERRESTRIAL_FRONT_END -DTER_TUNER_SILABS -DDEMOD_DVB_T -DDEMOD_DVB_C -DDEMOD_MCNS
Si2166 Si21662	-DSILABS_SUPERSET -DSATELLITE_FRONT_END -DSAT_TUNER_SILABS -DDEMOD_DVB_S_S2_DSS -DDEMOD_DVB_S2X
Si2167 Si21672	-DSILABS_SUPERSET -DTERRESTRIAL_FRONT_END -DTER_TUNER_SILABS -DDEMOD_DVB_T -DDEMOD_DVB_C -DDEMOD_MCNS -DSATELLITE_FRONT_END -DSAT_TUNER_SILABS -DDEMOD_DVB_S_S2_DSS -DDEMOD_DVB_S2X
Si2168 Si21682	-DSILABS_SUPERSET -DTERRESTRIAL_FRONT_END -DTER_TUNER_SILABS -DDEMOD_DVB_T -DDEMOD_DVB_T2 -DDEMOD_DVB_C -DDEMOD_MCNS
Si2169 Si21692	-DSILABS_SUPERSET -DTERRESTRIAL_FRONT_END -DTER_TUNER_SILABS -DDEMOD_DVB_T -DDEMOD_DVB_T2 -DDEMOD_DVB_C -DDEMOD_MCNS -DSATELLITE_FRONT_END -DSAT_TUNER_SILABS -DDEMOD_DVB_S_S2_DSS -DDEMOD_DVB_S2X
Si2180 Si21802	-DSILABS_SUPERSET -DTERRESTRIAL_FRONT_END -DTER_TUNER_SILABS -DDEMOD_DVB_T -DDEMOD_ISDB_T -DDEMOD_DVB_C -DDEMOD_MCNS
Si2181 Si21812	-DSILABS_SUPERSET -DTERRESTRIAL_FRONT_END -DTER_TUNER_SILABS -DDEMOD_DVB_T -DDEMOD_ISDB_T -DDEMOD_DVB_C -DDEMOD_MCNS -DSATELLITE_FRONT_END -DSAT_TUNER_SILABS -DDEMOD_DVB_S_S2_DSS -DDEMOD_DVB_S2X
Si2182 Si21822	-DSILABS_SUPERSET -DTERRESTRIAL_FRONT_END -DTER_TUNER_SILABS -DDEMOD_DVB_T -DDEMOD_DVB_T2 -DDEMOD_ISDB_T -DDEMOD_DVB_C -DDEMOD_MCNS -DSATELLITE_FRONT_END -DSAT_TUNER_SILABS -DDEMOD_DVB_S_S2_DSS -DDEMOD_DVB_S2X
Si2183 Si21832	-DSILABS_SUPERSET -DTERRESTRIAL_FRONT_END -DTER_TUNER_SILABS -DDEMOD_DVB_T -DDEMOD_DVB_T2 -DDEMOD_ISDB_T -DDEMOD_DVB_C -DDEMOD_DVB_C2 -DDEMOD_MCNS -DSATELLITE_FRONT_END -DSAT_TUNER_SILABS -DDEMOD_DVB_S_S2_DSS -DDEMOD_DVB_S2X

NB: It's possible to remove standards listed above are follows, to reduce the executable code footprint:

- If DVB_T2 is required, keep DVB-T (All DVB-T2 products also need to support DVB-T)
- If DVB_C2 is required, keep DVB-C (All DVB-C2 products also need to support DVB-C)
- If DVB_S2 is required, use DEMOD_DVB_S_S2_DSS (All DVB-S2 products also need to support DVB-S)
- If DVB_S2X is required, use DEMOD_DVB_S_S2_DSS + DEMOD_DVB_S2X (All DVB-S2X products also need to support DVB-S2)
- ISDB_T-only is possible
- DVB_T-only is possible (not much sense)
- DVB_C-only is possible (so, without MCNS)

Annex iii SW Configuration Check List

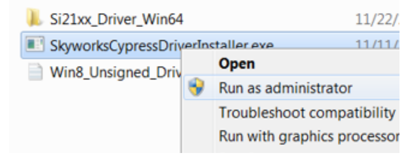
The table below summarizes all SW configuration settings.
It can be printed out and used during discussions with the HW engineers, using the electrical diagram as a reference, to prepare the SW configuration. Once this table is filled out, the SW configuration is trivial.

Function or item	Parameter name	Value	Comments
SelectFrontEnd	fe_index		
SiLabs_API_Frontend_Chip	demod_id	0x2183	Forced for SKYWORKS SUPERSET
SiLabs_API_SW_Init	demodAdd		
	tunerAdd_Ter		
	tunerAdd_Sat		
SiLabs_API_Select_TER_Tuner	ter_tuner_code		
	ter_tuner_index	0	Forced to 0
SiLabs_API_TER_tuner_I2C_connection	fe_index		Only for dual/triple/quad
SiLabs_API_TER_Tuner_ClockConfig	xtal		
	xout		
SiLabs_API_TER_Clock	clock_source		
	clock_input		
	clock_freq		
	clock_control		
SiLabs_API_TER_Tuner_FEF_Input	dtv_fef_freeze_input		
SiLabs_API_TER_FEF_Config	fef_mode		
	fef_pin		
	fef_level		
SiLabs_API_TER_Tuner_AGC_Input	dtv_out_type		
SiLabs_API_TER_AGC	agc1_mode		
	agc1_inversion		
	agc2_mode		
	agc2_inversion		
SiLabs_API_TER_Tuner_IF_Output	dtv_out_type		
SiLabs_API_Select_SAT_Tuner	sat_tuner_code		
	sat_tuner_index	0	Forced to 0
SiLabs_API_SAT_Select_LNB_Chip	lnb_code		
	lnb_chip_address		
SiLabs_API_SAT_LNB_Chip_Index	lnb_index		Only for dual LNB controllers
SiLabs_API_SAT_tuner_I2C_connection	fe_index		Only for dual/triple/quad
SiLabs_API_SAT_Clock	clock_source		
	clock_input		
	clock_freq		
	clock_control		
SiLabs_API_SAT_Spectrum	spectrum_inversion		
SiLabs_API_SAT_AGC	agc1_mode		
	agc1_inversion		
	agc2_mode		
	agc2_inversion		
SiLabs_API_Set_Index_and_Tag	index		Better traces for dual/triple/quad
	tag		
SiLabs_API_HW_Connect	connection_mode		
SiLabs_API_Cypress_Ports	OEA		Only for Skyworks EVBs
	IOA		
	OEB		
	IOB		
	OED		
	IOD		

Legend

	Function/Value required for all applications
	Function/Value required for TER applications
	Function/Value required for SAT applications

Annex iv USB driver installation

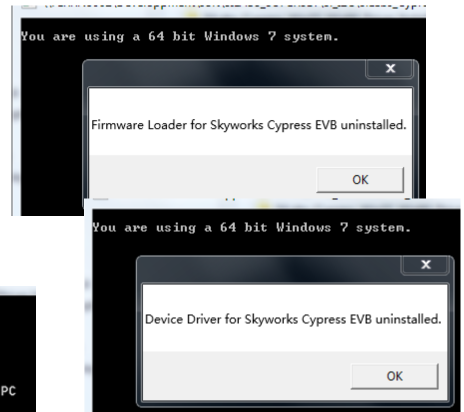


Under your SKYWORKS_SUPERSET folder, go to Si_I2C\Skyworks_Cypress_Windows_Driver_Installer.

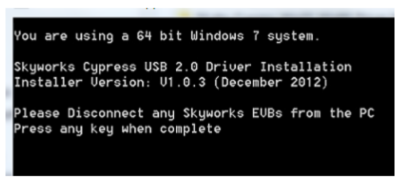
Right-click on the SkyworksCypressDriverInstaller.exe and select 'run as administrator'.

Then, follow the on-screen instructions.

First of all, the installer will display information on your system (Windows version and 32/64 bit), then it will uninstall previous drivers for Skyworks EVBs. Just press 'OK' on both messages to acknowledge the actions.

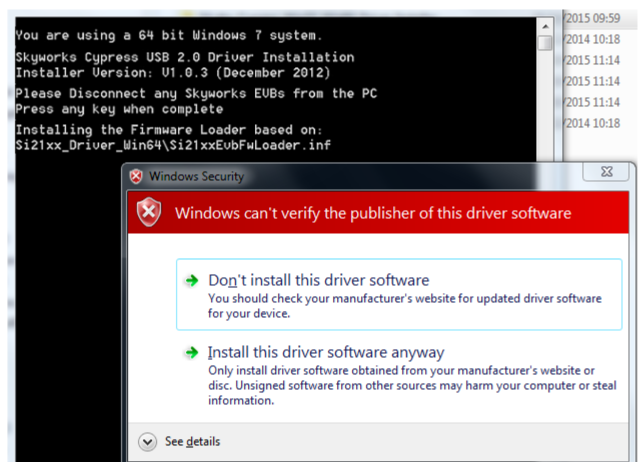


Then, the installer will prompt you to disconnect any Skyworks EVB.
Please do so, and press 'enter' to continue.



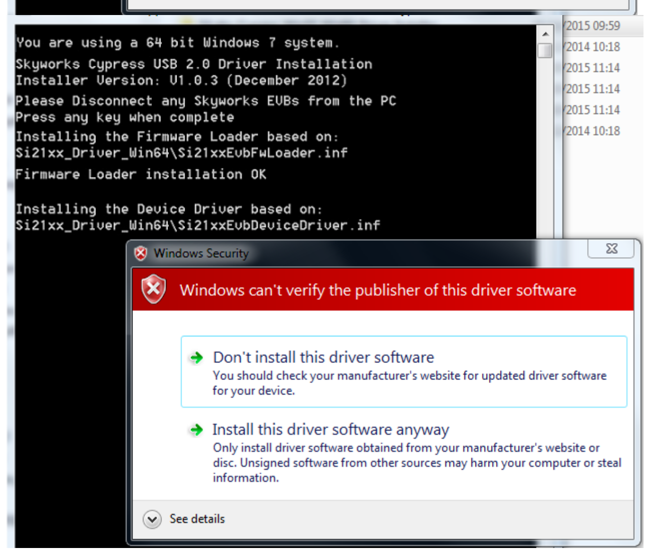
Next, the installer will install the Firmware loader on your machine.

If you get a message indicating that the software publisher can't be verified, select the <u>'Install the driver anyway'</u> option to bypass it.



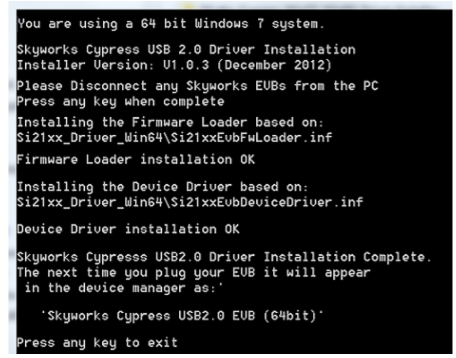
Next, the installer will install the Device driver on your machine.

If you get a message indicating that the software publisher can't be verified, select the <u>'Install the driver anyway'</u> option to bypass it.

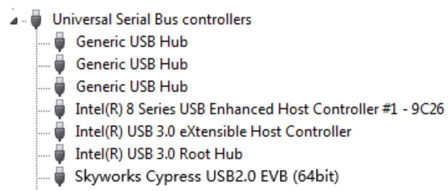


Finally, the installer will indicate proper installation of the Skyworks Cypress USB2.0 Driver.

Press ‘enter’ to acknowledge this message and terminate the installer.

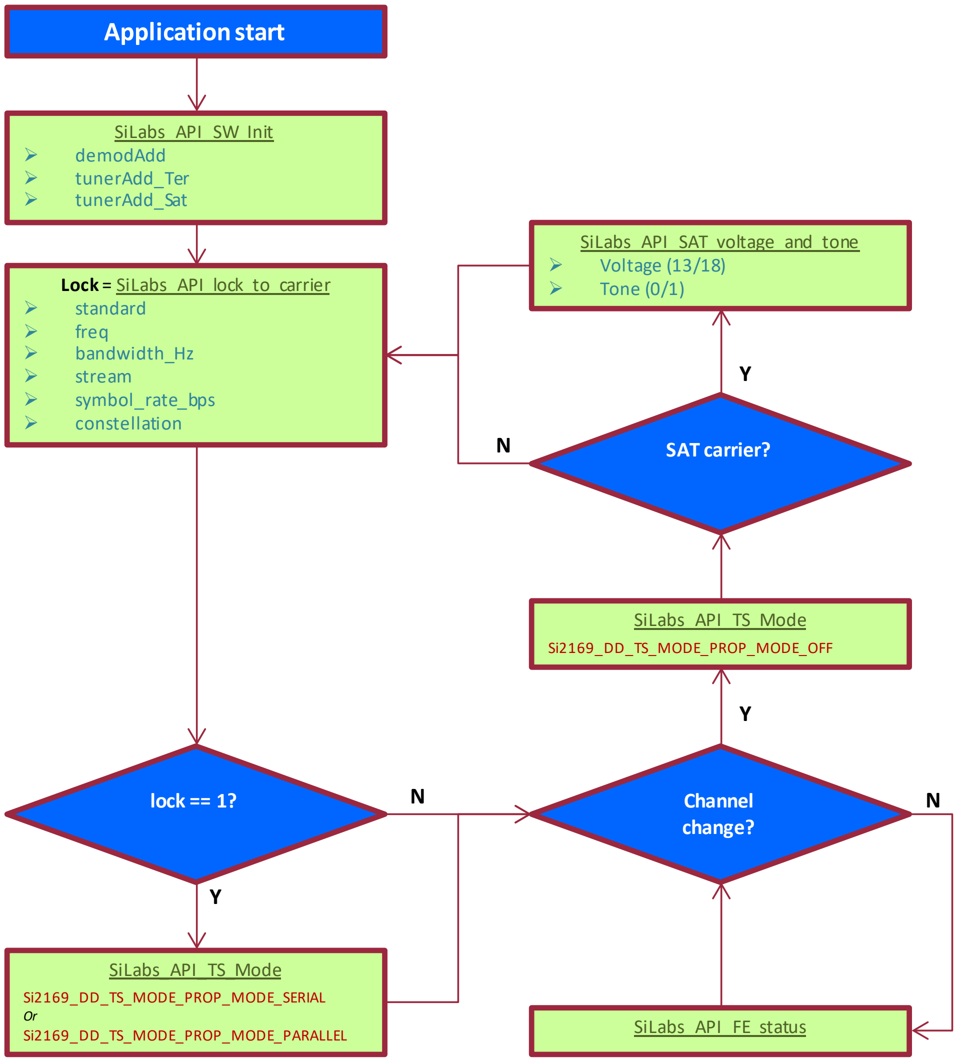


You can now connect the EVB to your machine and open the Windows Device Manager (under Windows, go to the control Panel then select ‘Device Manager’).
Under ‘Universal Serial Bus controllers’ you should see the ‘Skyworks Cypress USB2.0 EVB’ if the installation is correct.



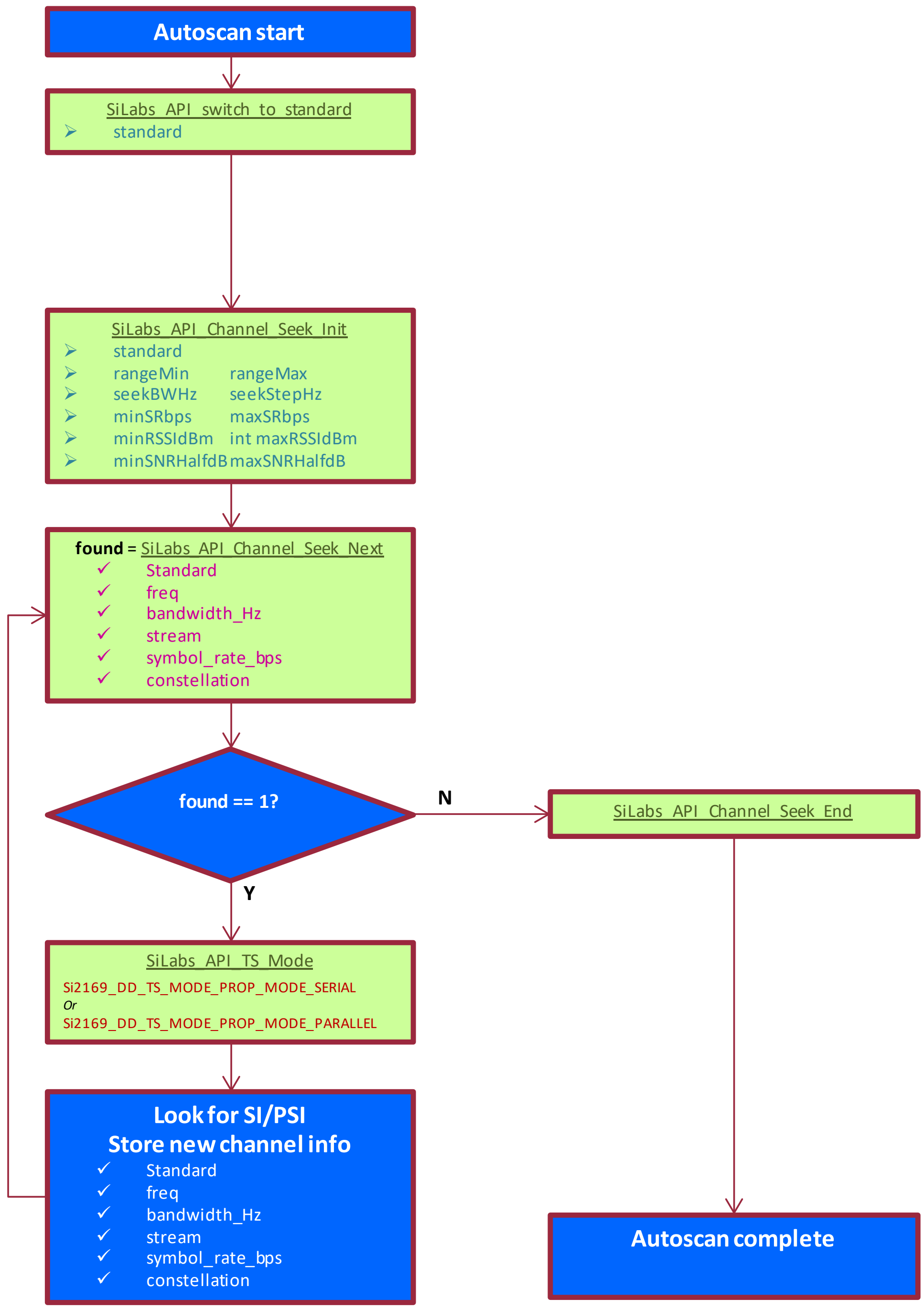
Annex v Application flowchart: channel lock

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



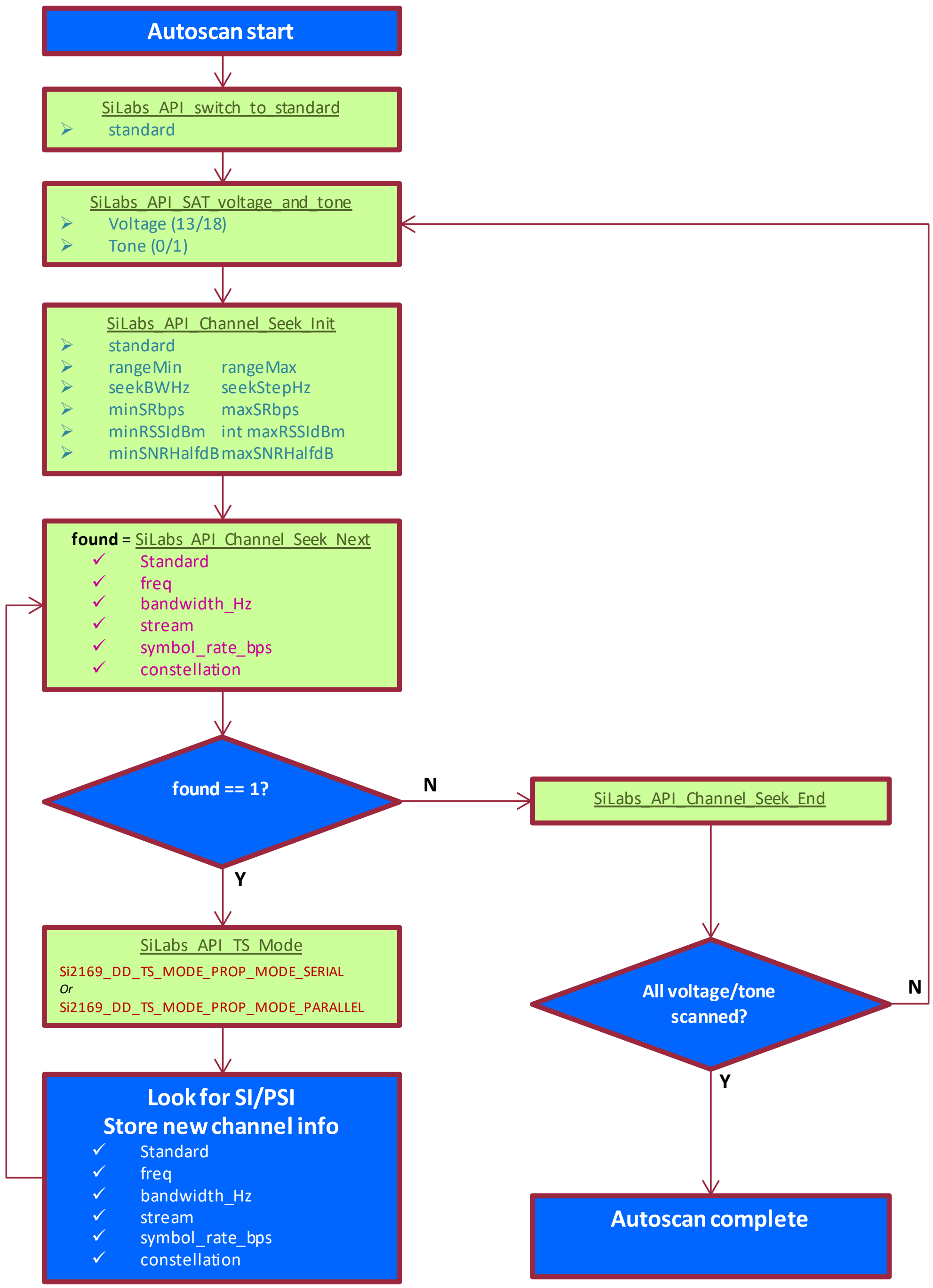
Annex vi TER auto-scan flowchart (DVB-T/T2, DVB-C)

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.



Annex vii SAT auto-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.



9 Using the API

Following the software initialization (which is allocating the necessary items and setting the parameters to match the hardware), the applications is now ready to be used to lock and monitor the frontend status.
This is described in the following paragraphs

9.1 Locking the frontend

After the software initialization is complete, a single function can be used to lock in any standard: SiLabs_API_lock_to_carrier. SiLabs_API_lock_to_carrier will go through the hardware initialization procedure (if not already done), prepare the frontend to lock on the required standard (tuning and setting the demodulator), then wait for the lock.

```
/************************************************************************************************************
  SiLabs_API_lock_to_carrier function
  Use:      relocking function
            Used to relock on a channel for the required standard
  Parameter: standard the standard to lock to
  Parameter: freq                 the frequency to lock to    (in Hz for TER, in kHz for SAT)
  Parameter: bandwidth_Hz         the channel bandwidth in Hz (only for DVB-T, DVB-T2, ISDB-T)
  Parameter: dvb_t_stream         the HP/LP stream            (only for DVB-T)
  Parameter: symbol_rate_bps      the symbol rate in baud/s   (for DVB-C, MCNS and SAT)
  Parameter: dvb_c_constellation the DVB-C constellation      (only for DVB-C)
  Parameter: polarization         the SAT LNB polarization    (only for SAT)
  Parameter: band                 the SAT LNB band selection  (only for SAT)
  Parameter: data_slice_id        the DATA SLICE Id           (only for DVB-C2 when num_dslice  > 1)
  Parameter: plp_id               the PLP Id                  (only for DVB-T2 and DVB-C2 when num_plp >
         1, or used as isi_id in DVB-S2)
  Parameter: T2_lock_mode         the DVB-T2 lock mode        (0='ANY', 1='T2-Base', 2='T2-Lite')
  Return:    1 if locked, 0 otherwise
***********************************************************************************************************/
```

9.1.1 Locking in TER

To lock in a TER standard, just call SiLabs_API_lock_to_carrier with the parameters required for the selected standard.

9.1.1.1 DVB-T/ DVB-T2 auto detection

In case you'd like to automatically lock in T or T2 when the signal is one of these standards, use SiLabs_API_TER_AutoDetect (frontend, 1); to activate the T/T2 auto detect feature. This is convenient during the installation procedure. To disable this, call SiLabs_API_TER_AutoDetect (frontend, 0);

9.1.2 Locking in SAT

To lock in a SAT standard, it's required to allow controlling the LNB controller first. There is a chicken and egg situation there, where you can only control the LNB 22 kHz tone if the demodulator is running, and you can't lock the demodulator (using SiLabs_API_lock_to_carrier) until the LNB provides the required SAT signal.

To get out of this, it's required to first perform the frontend initialization (once) then call SiLabs_API_lock_to_carrier as below:

- Call SiLabs_API_switch_to_standard for the desired standard (→demodulator init done, LNB 22 kHz tone can be used).
- Call SiLabs_API_lock_to_carrier with the parameters required for the selected standard (including the polarization and band).

9.1.2.1 DVB-S/ DVB-S2 (/ DSS) auto detection

In case you'd like to automatically lock in S or S2 when the signal is one of these standards, use SiLabs_API_SAT_AutoDetect (frontend, 1); to activate the S/S2 auto detect feature. This is convenient during the installation procedure. To disable this, call SiLabs_API_SAT_AutoDetect (frontend, 0);

NB: In case you'd like to automatically lock on DSS signal as well, setting the current standard as SILABS_DSS will allow auto detection between DVB-S / DVB-S2 / DSS signals.

9.2 Checking the frontend status

A single function is used to monitor the frontend status, SiLabs_API_FE_status_selection. This is an extension of the initial SiLabs_API_FE_status function (still available for compatibility purposes), with control over the statuses which will be refreshed.

```
/************************************************************************************************
  SiLabs_API_FE_status_selection function
  Use:        Front-End status function, with selection of 'all' or partial statuses
              Used to retrieve the status of the front-end in a structure
  Parameter:  front_end, a pointer to the SILABS_FE_Context context
  Parameter:  status, a pointer to the status structure (configurable in CUSTOM_Status_Struct)
  Parameter:  status_selection, an 8 bit field used to control which items to refresh. Use 0x00 for
         'all'.
  Return:     1 if successful, 0 otherwise
************************************************************************************************/
signed   int  SiLabs_API_FE_status_selection        (SILABS_FE_Context *front_end,
         CUSTOM_Status_Struct *status, unsigned char status_selection);
```

SiLabs_API_FE_status_selection will update the CUSTOM_Status_Struct members depending on the status_selection flag.
NB: status_selection = 0x00 will status everything possible for the current standard. This is the behavior of SiLabs_API_FE_status.

The possible values for status_selection are:

```
typedef enum  _STATUS_SELECTION {
  FE_LOCK_STATE     = 0x01, /* demod_lock, fec_lock,  uncorrs, TS_bitrate_kHz, TS_clock_kHz  */
  FE_LEVELS         = 0x02, /* RSSI, RFagc, IFagc                                            */
  FE_RATES          = 0x04, /* BER, PER, FER (depending on standard)                         */
  FE_SPECIFIC       = 0x08, /* symbol_rate, stream, constellation, c/n, freq_offset,
                                 timing_offset, code_rate, t2_version, num_plp, plp_id
                                 ds_id, cell_id, etc. (generally one command per standard)   */
  FE_QUALITY        = 0x10, /* SSI, SQI                                                      */
  FE_FREQ           = 0x20, /* freq                                                          */
} STATUS_SELECTION;
```

This is useful to limit I2C traffic in case the application requires monitoring of only a subset of the status fields.
It removes the need for several separate monitoring functions.

9.3 Standard specific features

9.3.1 DVB-T2

9.3.1.1 DVB-T2 PLP management

After locking on a DVB-T2 channel for the first time, it is recommended to check the number of PLPs in this channel.

The number of PLPs in a DVB-T2 channel is provided as the 'status->num_plp' member of the status structure, which is updated upon each call to SiLabs_API_FE_status_selection with status_selection containing 'FE_SPECIFIC'.

The num_plp value is then used in a loop of calls from plp_index = 0 to plp_index = num_plp -1 to SiLabs_API_Get_PLP_ID_and_TYPE (ds_id is not used in DVB-T2).

Each call to SiLabs_API_Get_PLP_ID_and_TYPE returns the plp_id and plp_type values for this PLP.

If the plp_type value is different from SILABS_PLP_TYPE_COMMON (0), then the PLP is a DATA PLP which needs to be stored as a valid PLP in the channel list.

The plp_id value will be used later on in future calls to SiLabs_API_lock_to_carrier to lock on this PLP.

Typical DVB-C2 data slice and PLP loop:

```
if (status->standard == SILABS_DVB_T2) {
  for (plp_index =0; plp_index <status->num_plp; plp_index ++) {
    SiLabs_API_Get_PLP_ID_and_TYPE  (front_end, 0, plp_index, &plp_id, &plp_type);
  }
```

```
}
```

9.3.2 DVB-C2

9.3.2.1 DVB-C2 Dataslice and PLP management

After locking on a DVB-C2 channel for the first time, it is recommended to check the number of data slices in this channel, then to check the number of PLs in each data slice.

The number of data slices in a DVB-C2 channel is provided as the 'status->num_data_slice member of the status structure, which is updated upon each call to SiLabs_API_FE_status_selection with status_selection containing 'FE_SPECIFIC'.

The number of PLPs in a DVB-T2 channel is provided as the 'status->num_plp' member of the status structure, which is updated upon each call to SiLabs_API_FE_status_selection with status_selection containing 'FE_SPECIFIC'.
SiLabs_API_Get_DS_ID_Num_PLP_Freq
The num_data_slice value is then used in a loop of calls from ds_index = 0 to plp_index = num_plp -1 to SiLabs_API_Get_PLP_ID_and_TYPE (ds_id is not used in DVB-T2).

Each call to SiLabs_API_Get_PLP_ID_and_TYPE returns the plp_id and plp_type values for this PLP.

If the plp_type value is different from SILABS_PLP_TYPE_COMMON (0), then the PLP is a DATA PLP which needs to be stored as a valid PLP in the channel list.

The plp_id value will be used later on in future calls to SiLabs_API_lock_to_carrier to lock on this PLP.

Typical DVB-C2 data slice and PLP loop:

```
if (status->standard == SILABS_DVB_C2) {
  for (ds_index = 0; ds_index <status->num_data_slice; ds_index ++) {
    SiLabs_API_Get_DS_ID_Num_PLP_Freq (front_end, ds_index, &data_slice_id, &num_plp, &freq);
    for (plp_index=0; plp_index <num_plp; plp_index ++) {
      SiLabs_API_Get_PLP_ID_and_TYPE  (front_end, ds_index, plp_index, &plp_id, &plp_type);
    }
  }
}
```

9.3.3 ISDB-T

9.3.3.1 ISDB-T A B C Layers transmission information

```
/************************************************************************************************************
  NAME: SiLabs_API_TER_ISDBT_Layer_Info
  DESCRIPTION: information function for the TER ISDBT Layer information
  Parameter: front_end,    Pointer to SILABS_FE_Context
  Parameter: layer          the ISDB-T Layer to retrieve info about (0xA='LAYER_A', 0xB='LAYER_B', 0xC='LAYER_C')
  Parameter: *constellation  Pointer to the ISDB-T constellation
  Parameter: *code_rate      Pointer to the ISDB-T code rate
  Parameter: *il             Pointer to the ISDB-T interleaving code
  Parameter: *nb_seg         Pointer to the ISDB-T number of segments
  Returns:    0 if OK (-1 otherwise)
************************************************************************************************************/
signed  int  SiLabs_API_TER_ISDBT_Layer_Info      (SILABS_FE_Context *front_end,
                                                   signed   int layer,
                                                   signed   int *constellation,
                                                   signed   int *code_rate,
                                                   signed   int *il,
                                                   signed   int *nb_seg);
```

To retrieve the transmission information for the ISDB-T A, B and C Layers, call SiLabs_API_TER_ISDBT_Layer_Info with 'layer' equal to 0xA, 0xB, 0xC respectively.

The following values for the selected layer will then be updated:

- constellation
- code_rate
- il
- nb_seg

9.3.3.2 ISDB-T A B C Layers monitoring

In addition to the general status items for all other standards, ISDB-T specific items have been added to the status structure to store the ISDB-T layer information:

_Wrapper\SiLabs_API_L3_Wrapper.h

```
  /* Start of ISDB-T specifics   */
  signed   int  isdbt_system_id;
  signed   int  nb_seg_a;
  signed   int  nb_seg_b;
  signed   int  nb_seg_c;
  signed   int  fec_lock_a;
  signed   int  fec_lock_b;
  signed   int  fec_lock_c;
  signed   int  partial_flag;
  signed   int  emergency_flag;
  signed   int  constellation_a;
  signed   int  constellation_b;
  signed   int  constellation_c;
  signed   int  code_rate_a;
  signed   int  code_rate_b;
  signed   int  code_rate_c;
  signed   int  uncorrs_a;
  signed   int  uncorrs_b;
  signed   int  uncorrs_c;
  signed   int  il_a;
  signed   int  il_b;
  signed   int  il_c;
  signed   int  ber_window_a;
  signed   int  ber_window_b;
  signed   int  ber_window_c;
  signed   int  ber_count_a;
  signed   int  ber_count_b;
  signed   int  ber_count_c;
#ifndef  NO_FLOATS_ALLOWED
  double        ber_a;
  double        ber_b;
  double        ber_c;
#endif /* NO_FLOATS_ALLOWED */
  /* End of ISDB-T specifics   */
```

The general items such as ber, constellation, code_rate and uncorrs reflect the status of the currently monitored layer, while the layer-specific values such as ber_x, constellation_x, code_rate_x and uncorrs_x store the values per level.

The SiLabs_API_TER_ISDBT_Monitoring_mode function controls the way ISDB-T layer information is statused.

```
/************************************************************************************************************************
  NAME: SiLabs_API_TER_ISDBT_Monitoring_mode
  DESCRIPTION: selection function for the TER ISDBT C/N monitoring (the one which will reflect in the c_n status) mode
  Parameter: front_end, Pointer to SILABS_FE_Context
  Parameter: layer_mon       the ISDB-T monitoring mode (0='ALL', 0xA='LAYER_A', 0xB='LAYER_B', 0xC='LAYER_C', 0xABC="loop mode')
  Returns:    the current state of the ISDBT C/N monitoring mode field (using a value out of the above returns the current value)
************************************************************************************************************************/
signed   int  SiLabs_API_TER_ISDBT_Monitoring_mode  (SILABS_FE_Context *front_end,    signed   int layer_mon);
```

To monitor the transmission quality for the ISDB-T A, B and C Layers, call SiLabs_API_TER_ISDBT_Monitoring_mode with ‘layer_mon’ equal to 0xA, 0xB, 0xC respectively, then call SiLabs_API_FE_status_selection to get the status for the selected layer. NB: The BER information is not immediately available. When it’s unavailable it’s returning as ‘1’.

To monitor values (such as the BER) for ALL layers together, call SiLabs_API_TER_ISDBT_Monitoring_mode with ‘layer_mon’ equal to 0xA, 0xB, 0xC respectively, then call SiLabs_API_FE_status_selection to get the global status for ALL layers. NB: The BER information is not immediately available. When it’s unavailable it’s returning as ‘1’.

To monitor values (such as the BER) for layers A, B and C in a ‘loop mode’ fashion, call SiLabs_API_TER_ISDBT_Monitoring_mode with ‘layer_mon’ equal to 0xABC respectively, then call SiLabs_API_FE_status_selection to have the status for each active layer updated as soon as possible, one layer at a time.

The ‘loop mode’ strategy consists in:

- Updating the information for the current layer in layer-specific status members
- Checking if the BER is ‘available’ for the current layer.
- When the BER for a layer is available, store it and select the next active layer
- When changing for a new layer, check its constellation and adapt the BER depth to get a fast C/N update.
- Loop this, one step during each call to the status function.

As a consequence, several calls to SiLabs_API_FE_status_selection will be necessary to have all 3 layers’ information up to date following a lock. These will then be maintained and regularly updated when the loop will select the next layer. The number of calls with mainly depend on the delay between calls to SiLabs_API_FE_status_selection.

10 Frequently Asked Questions / FAQ

10.1 Why use a wrapper for Skyworks applications?

Legacy Skyworks digital demodulators (Si2165D/Si2167A/Si2169A) being pin-pin compatible with the current API-controlled parts but not controlled via the same method ('register' mode for legacy, 'command/property' mode for recent demodulators), the wrapper proposes a uniform front-end API, thus making the transition between Skyworks demodulators easy. This is valid for all Skyworks demodulators available to the date of writing, and should remain true over time.

The wrapper also makes it easy to select any terrestrial or satellite tuner without the need to change the top-level application, thanks to dedicated TER Tuner and SAT tuner wrappers.

10.2 Why use a set of wrappers for TER and SAT tuners?

At the time the initial demodulator source code releases were issued, there was only a limited number of tuners, and the first implementations used a set of macros per tuner to wrap the tuner function calls, and handle tuner specific settings. This leads to having a dedicated h file per tuner in the demodulator's code.

Over time, the number of supported tuners increased, with the need to stay compatible with an increasing list of tuners.

This meant that:

- There was an increasing number of tuner-related files in each demodulator code, and this number was much higher than the number of files directly related to the demodulators.
- Upon arrival of each new tuner, each demodulator driver required updates to support the new tuner, and therefore software updates occurred because a new tuner was supported, while otherwise the demodulator code could have been left unchanged.
- It became more and more difficult and complex to make sure that any change in the demod-tuner files was properly ported to all demodulator codes.

To avoid the above consequences related to an ever-increasing number of tuners, it has been decided to separate the tuner code from the demodulator code using one wrapper for all TER tuners and one wrapper for all SAT tuners.
With this organization, the situation is now:

- The number of files in each demodulator code is constant and much lower than what it was, at least for recent demodulators. 'Legacy' demodulators still have the original list of tuner files, for compatibility purposes with previous applications.
- No demodulator source code update is required when a new tuner gets in.
- The support of any new tuner is only added in the related tuner wrapper code.
- Each tuner is controlled through a unique/common location
- Adding one new tuner in one of the tuner wrapper makes it immediately available to all applications, without any other change than in the SW initialization calls, which needs to use the new tuner code if required.
- At the date of writing, there are 22 Skyworks TER tuners and 9 SAT tuner drivers supported by these wrappers. Some SAT tuners (i.e. Airoha tuners and RDA tuners) are valid for several parts, such that a total of 16 SAT tuners can be used.
- It is also possible to control any TER or SAT tuner thanks to the 'CUSTOMTER' and 'CUSTOMSAT' access.
- 9 LNB controllers are also available for SAT applications. These are not controlled via a dedicated wrapper but directly from the L3 code.

10.3 How to use my company's enums for 'standard', 'qam', 'hierarchy', etc?

These are defined in SiLabs_API_L3_Wrapper.h, the Skyworks API wrapper header:

```
/* Standard code values used by the top-level application              */
/* <porting> set these values to match the top-level application values */
```

```
typedef enum   CUSTOM_Standard_Enum {
  SILABS_ANALOG = 4,
  SILABS_DVB_T  = 0,
  SILABS_DVB_C  = 1,
  SILABS_DVB_S  = 2,
  SILABS_DVB_S2 = 3,
  SILABS_DVB_T2 = 5,
  SILABS_DSS    = 6,
  SILABS_MCNS   = 7,
  SILABS_DVB_C2 = 8,
  SILABS_ISDB_T = 9,
  SILABS_SLEEP  = 100,
  SILABS_OFF    = 200
} CUSTOM_Standard_Enum;

typedef enum   CUSTOM_Constel_Enum  {
. . .
typedef enum   CUSTOM_Stream_Enum   {
. . .
typedef enum   CUSTOM_TS_Mode_Enum  {
. . .
typedef enum   CUSTOM_FFT_Mode_Enum {
. . .
typedef enum   CUSTOM_GI_Enum {
. . .
typedef enum   CUSTOM_Coderate_Enum {
. . .
typedef enum   CUSTOM_Hierarchy_Enum {
. . .
typedef enum   CUSTOM_Video_Sys_Enum {
. . .
typedef enum   CUSTOM_Transmission_Mode_Enum {
. . .
typedef enum   CUSTOM_Color_Enum {
. . .
```

Change the values in the above enums to match what your application is expecting.

10.4 How are my company's enums translated into values specific to a tuner or a demodulator?

For each enum, a translation function named 'Silabs_<enumName>Code' is used to transform your company's enum into the Skyworks value for the corresponding chip.

Below is the example for the standardCode:

```
int   Silabs_standardCode        (SILABS_FE_Context* front_end, CUSTOM_Standard_Enum standard)
{
#ifdef    Si2183_COMPATIBLE
  if (front_end->chip ==   0x2183 ) {
    switch (standard) {
#ifdef    DEMOD_DVB_T
      case SILABS_DVB_T : return Si2183_DD_MODE_PROP_MODULATION_DVBT;
#endif /* DEMOD_DVB_T */
#ifdef    DEMOD_ISDB_T
      case SILABS_ISDB_T: return Si2183_DD_MODE_PROP_MODULATION_ISDBT;
#endif /* DEMOD_ISDB_T */
#ifdef    DEMOD_DVB_T2
```

```
      case SILABS_DVB_T2: return Si2183_DD_MODE_PROP_MODULATION_DVBT2;
#endif /* DEMOD_DVB_T2 */
#ifdef    DEMOD_MCNS
      case SILABS_MCNS  : return Si2183_DD_MODE_PROP_MODULATION_MCNS;
#endif /* DEMOD_MCNS */
#ifdef    DEMOD_DVB_C
      case SILABS_DVB_C : return Si2183_DD_MODE_PROP_MODULATION_DVBC;
#endif /* DEMOD_DVB_C */
#ifdef    DEMOD_DVB_C2
      case SILABS_DVB_C2: return Si2183_DD_MODE_PROP_MODULATION_DVBC2;
#endif /* DEMOD_DVB_C2 */
#ifdef    DEMOD_DVB_S_S2_DSS
      case SILABS_DVB_S : return Si2183_DD_MODE_PROP_MODULATION_DVBS;
      case SILABS_DVB_S2: return Si2183_DD_MODE_PROP_MODULATION_DVBS2;
      case SILABS_DSS   : return Si2183_DD_MODE_PROP_MODULATION_DSS;
#endif /* DEMOD_DVB_S_S2_DSS */
      case SILABS_ANALOG: return Si2183_DD_MODE_PROP_MODULATION_ANALOG;
      case SILABS_SLEEP : return 100;
      case SILABS_OFF   : return 200;
      default           : return -1;
    }
  }
#endif /* Si2183_COMPATIBLE */
  return -1;
}
```

10.5 How are Skyworks internal values translated into my company’s values?

For each enum, a translation function named ‘Custom_<enumName>Code’ is used to transform your company’s enum into the Skyworks value for the corresponding chip.

Below is the example for the standardCode:

```
int   Custom_standardCode        (SILABS_FE_Context* front_end, int standard)
{
#ifdef    Si2183_COMPATIBLE
  if (front_end->chip ==   0x2183 ) {
    switch (standard) {
#ifdef    DEMOD_DVB_T
      case Si2183_DD_MODE_PROP_MODULATION_DVBT : return SILABS_DVB_T ;
#endif /* DEMOD_DVB_T */
#ifdef    DEMOD_ISDB_T
      case Si2183_DD_MODE_PROP_MODULATION_ISDBT: return SILABS_ISDB_T;
#endif /* DEMOD_ISDB_T */
#ifdef    DEMOD_DVB_T2
      case Si2183_DD_MODE_PROP_MODULATION_DVBT2: return SILABS_DVB_T2;
#endif /* DEMOD_DVB_T2 */
#ifdef    DEMOD_DVB_C
      case Si2183_DD_MODE_PROP_MODULATION_DVBC : return SILABS_DVB_C ;
#endif /* DEMOD_DVB_C */
#ifdef    DEMOD_DVB_C2
      case Si2183_DD_MODE_PROP_MODULATION_DVBC2: return SILABS_DVB_C2;
#endif /* DEMOD_DVB_C2 */
#ifdef    DEMOD_MCNS
      case Si2183_DD_MODE_PROP_MODULATION_MCNS : return SILABS_MCNS  ;
```

```
#endif /* DEMOD_MCNS */
#ifdef    DEMOD_DVB_S_S2_DSS
      case Si2183_DD_MODE_PROP_MODULATION_DVBS : return SILABS_DVB_S ;
      case Si2183_DD_MODE_PROP_MODULATION_DVBS2: return SILABS_DVB_S2;
      case Si2183_DD_MODE_PROP_MODULATION_DSS  : return SILABS_DSS   ;
#endif /* DEMOD_DVB_S_S2_DSS */
      default                                  : return -1;
    }
  }
#endif /* Si2183_COMPATIBLE */
  return -1;
}
```

10.6 Which functions does the wrapper include?

The Skyworks API wrapper includes all functions necessary to control a digital television front-end:

- SW init
- SW configuration
- Scanning
- Locking
- Monitoring
- Control of the traces mechanism

Below is the list of Skyworks API functions at the time of writing:

```
signed   char SiLabs_API_SW_Init                    (SILABS_FE_Context *front_end,    signed   int
         demodAdd, signed   int  tunerAdd_Ter, signed   int  tunerAdd_Sat);
signed   int  SiLabs_API_Set_Index_and_Tag          (SILABS_FE_Context *front_end,    unsigned char
         index, char* tag);
signed   int  SiLabs_API_Frontend_Chip              (SILABS_FE_Context *front_end,    signed   int
         demod_id);
signed   int  SiLabs_API_Handshake_Setup            (SILABS_FE_Context *front_end,    signed   int
         handshake_mode, signed   int  handshake_period_ms);
signed   int  SiLabs_API_SPI_Setup                  (SILABS_FE_Context *front_end,    unsigned int
         send_option, unsigned int  clk_pin, unsigned int  clk_pola, unsigned int  data_pin, unsigned
         int  data_order);
signed   int  SiLabs_API_Demods_Broadcast_I2C       (SILABS_FE_Context *front_ends[], signed   int
         front_end_count);
signed   int  SiLabs_API_Broadcast_I2C              (SILABS_FE_Context *front_ends[], signed   int
         front_end_count);
signed   int  SiLabs_API_XTAL_Capacitance           (SILABS_FE_Context *front_end,    signed   int
         xtal_capacitance);
/* TER tuner selection and configuration functions (to be used after SiLabs_API_SW_Init) */
signed   int  SiLabs_API_TER_Possible_Tuners        (SILABS_FE_Context *front_end,    char *tunerList);
signed   int  SiLabs_API_Select_TER_Tuner           (SILABS_FE_Context *front_end,    signed   int
         ter_tuner_code, signed   int  ter_tuner_index);
void*         SiLabs_API_TER_Tuner                  (SILABS_FE_Context *front_end);
signed   int  SiLabs_API_TER_tuner_I2C_connection   (SILABS_FE_Context *front_end,    signed   int
         fe_index);
signed   int  SiLabs_API_TER_Address                (SILABS_FE_Context *front_end,    signed   int
         add);
signed   int  SiLabs_API_TER_Broadcast_I2C          (SILABS_FE_Context *front_ends[], signed   int
         front_end_count);
signed   int  SiLabs_API_TER_Tuner_ClockConfig      (SILABS_FE_Context *front_end,    signed   int
         xtal, signed   int  xout);
```

```
signed  int  SiLabs_API_TER_Clock_Options         (SILABS_FE_Context *front_end,    char
         *clockOptions);
signed  int  SiLabs_API_TER_Clock                 (SILABS_FE_Context *front_end,    signed  int
         clock_source, signed  int  clock_input, signed  int  clock_freq, signed  int
         clock_control);
signed  int  SiLabs_API_TER_FEF_Options           (SILABS_FE_Context *front_end,    char* fefOptions);
signed  int  SiLabs_API_TER_FEF_Config            (SILABS_FE_Context *front_end,    signed  int
         fef_mode, signed  int  fef_pin, signed  int  fef_level);
signed  int  SiLabs_API_TER_AGC_Options           (SILABS_FE_Context *front_end,    char *agcOptions);
signed  int  SiLabs_API_TER_AGC                   (SILABS_FE_Context *front_end,    signed  int
         agc1_mode,   signed  int  agc1_inversion, signed  int  agc2_mode, signed  int
         agc2_inversion);
signed  int  SiLabs_API_TER_Tuner_AGC_Input       (SILABS_FE_Context *front_end,    signed  int
         dtv_agc_source);
signed  int  SiLabs_API_TER_Tuner_GPIOs           (SILABS_FE_Context *front_end,    signed  int
         gpio1_mode, signed  int gpio2_mode);
signed  int  SiLabs_API_TER_Tuner_FEF_Input       (SILABS_FE_Context *front_end,    signed  int
         dtv_fef_freeze_input);
signed  int  SiLabs_API_TER_Tuner_IF_Output       (SILABS_FE_Context *front_end,    signed  int
         dtv_out_type);
/* SAT tuner selection and configuration functions (to be used after SiLabs_API_SW_Init) */
signed  int  SiLabs_API_SAT_Possible_Tuners       (SILABS_FE_Context *front_end,    char *tunerList);
signed  int  SiLabs_API_Select_SAT_Tuner          (SILABS_FE_Context *front_end,    signed  int
         sat_tuner_code, signed  int  sat_tuner_index);
signed  int  SiLabs_API_SAT_Tuner_Sub             (SILABS_FE_Context *front_end,    signed  int
         sat_tuner_sub);
signed  int  SiLabs_API_SAT_tuner_I2C_connection  (SILABS_FE_Context *front_end,    signed  int
         fe_index);
signed  int  SiLabs_API_SAT_Address               (SILABS_FE_Context *front_end,    signed  int
         add);
signed  int  SiLabs_API_SAT_Clock_Options         (SILABS_FE_Context *front_end,    char
         *clockOptions);
signed  int  SiLabs_API_SAT_Clock                 (SILABS_FE_Context *front_end,    signed  int
         clock_source, signed  int  clock_input, signed  int  clock_freq, signed  int
         clock_control);
signed  int  SiLabs_API_SAT_AGC_Options           (SILABS_FE_Context *front_end,    char *agcOptions);
signed  int  SiLabs_API_SAT_AGC                   (SILABS_FE_Context *front_end,    signed  int
         agc1_mode,   signed  int  agc1_inversion, signed  int  agc2_mode, signed  int
         agc2_inversion);
signed  int  SiLabs_API_SAT_Spectrum              (SILABS_FE_Context *front_end,    signed  int
         spectrum_inversion);
signed  int  SiLabs_API_SAT_Possible_LNB_Chips    (SILABS_FE_Context *front_end,    char *lnbList);
signed  int  SiLabs_API_SAT_Select_LNB_Chip       (SILABS_FE_Context *front_end,    signed  int
         lnb_code, signed  int  lnb_chip_address);
signed  int  SiLabs_API_SAT_LNB_Chip_Index        (SILABS_FE_Context *front_end,    signed  int
         lnb_index);

/* Front_end info, control and status functions                                    */
signed  int  SiLabs_API_Infos                     (SILABS_FE_Context *front_end,    char *infoString);
signed  int  SiLabs_API_Config_Infos              (SILABS_FE_Context *front_end,    char*
         config_function, char *infoString);
signed  char SiLabs_API_HW_Connect                (SILABS_FE_Context *front_end,    CONNECTION_TYPE
         connection_mode);
signed  char SiLabs_API_bytes_trace               (SILABS_FE_Context *front_end,    unsigned char
         track_mode);
signed  int  SiLabs_API_ReadString                (SILABS_FE_Context *front_end,    char *readString,
         unsigned char *pbtDataBuffer);
signed  char SiLabs_API_WriteString               (SILABS_FE_Context *front_end,    char
         *writeString);
signed  int  SiLabs_API_communication_check       (SILABS_FE_Context *front_end);
```

```
signed   int  SiLabs_API_switch_to_standard          (SILABS_FE_Context *front_end,    signed   int
         standard, unsigned char force_full_init);
signed   int  SiLabs_API_set_standard                (SILABS_FE_Context *front_end,    signed   int
         standard);
signed   int  SiLabs_API_lock_to_carrier             (SILABS_FE_Context *front_end,
                                           signed   int  standard,
                                           signed   int  freq,
                                           signed   int  bandwidth_Hz,
                                           unsigned int  stream,
                                           unsigned int  symbol_rate_bps,
                                           unsigned int  constellation,
                                           unsigned int  polarization,
                                           unsigned int  band,
                                           signed   int  data_slice_id,
                                           signed   int  plp_id,
                                           unsigned int  T2_lock_mode
                                           );
signed   int  SiLabs_API_Tune                        (SILABS_FE_Context *front_end,    signed   int
         freq);
signed   int  SiLabs_API_Channel_Lock_Abort          (SILABS_FE_Context *front_end);
signed   int  SiLabs_API_Demod_Standby               (SILABS_FE_Context *front_end);
signed   int  SiLabs_API_Demod_Silent                (SILABS_FE_Context *front_end,    signed   int
         silent);
signed   int  SiLabs_API_Demod_ClockOn               (SILABS_FE_Context *front_end);
signed   int  SiLabs_API_Reset_Uncorrs               (SILABS_FE_Context *front_end);
signed   int  SiLabs_API_TS_Strength_Shape           (SILABS_FE_Context *front_end,    signed   int
         serial_strength, signed   int serial_shape, signed   int parallel_strength, signed   int
         parallel_shape);
signed   int  SiLabs_API_TS_Config                   (SILABS_FE_Context *front_end,    signed   int
         clock_config, signed   int gapped, signed   int serial_clk_inv, signed   int parallel_clk_inv,
         signed   int ts_err_inv, signed   int serial_pin);
signed   int  SiLabs_API_TS_Mode                     (SILABS_FE_Context *front_end,    signed   int
         ts_mode);
signed   int  SiLabs_API_Get_TS_Dividers             (SILABS_FE_Context *front_end,    unsigned int
         *div_a, unsigned int *div_b);
signed   int  Silabs_API_TS_Tone_Cancel              (SILABS_FE_Context* front_end,    signed   int
         on_off);
signed   int  SiLabs_API_Tuner_I2C_Enable            (SILABS_FE_Context *front_end);
signed   int  SiLabs_API_Tuner_I2C_Disable           (SILABS_FE_Context *front_end);
/* Scan functions                                                                     */
signed   int  SiLabs_API_Channel_Seek_Init           (SILABS_FE_Context *front_end,
                                            signed   int  rangeMin,     signed   int  rangeMax,
                                            signed   int  seekBWHz,     signed   int  seekStepHz,
                                            signed   int  minSRbps,     signed   int  maxSRbps,
                                            signed   int  minRSSIdBm,   signed   int  maxRSSIdBm,
                                            signed   int  minSNRHalfdB, signed   int  maxSNRHalfdB);
signed   int  SiLabs_API_Channel_Seek_Next           (SILABS_FE_Context *front_end,    signed   int
         *standard, signed   int *freq, signed   int *bandwidth_Hz, signed   int *stream, unsigned int
         *symbol_rate_bps, signed   int *constellation, signed   int *polarization, signed   int *band,
         signed   int *num_data_slice, signed   int *num_plp, signed   int *T2_base_lite);
signed   int  SiLabs_API_Channel_Seek_Abort          (SILABS_FE_Context *front_end);
signed   int  SiLabs_API_Channel_Seek_End            (SILABS_FE_Context *front_end);
#ifdef    SATELLITE_FRONT_END
signed   int  SiLabs_API_SAT_AutoDetectCheck         (SILABS_FE_Context *front_end);
signed   int  SiLabs_API_SAT_AutoDetect              (SILABS_FE_Context *front_end,    signed   int
         on_off);
signed   int  SiLabs_API_SAT_Tuner_Init              (SILABS_FE_Context *front_end);
signed   int  SiLabs_API_SAT_Tuner_SetLPF            (SILABS_FE_Context *front_end,    signed   int
         lpf_khz);
```

```
signed   int  SiLabs_API_SAT_voltage                 (SILABS_FE_Context *front_end,    signed   int
         voltage);
signed   int  SiLabs_API_SAT_tone                    (SILABS_FE_Context *front_end,    unsigned char
         tone);
signed   int  SiLabs_API_SAT_voltage_and_tone        (SILABS_FE_Context *front_end,    signed   int
         voltage, unsigned char tone);
signed   int  SiLabs_API_SAT_prepare_diseqc_sequence(SILABS_FE_Context *front_end,    signed   int
         sequence_length, unsigned char *sequence_buffer, unsigned char cont_tone, unsigned char
         tone_burst, unsigned char burst_sel, unsigned char end_seq, signed   int *flags);
signed   int  SiLabs_API_SAT_trigger_diseqc_sequence(SILABS_FE_Context *front_end,    signed   int
         flags);
signed   int  SiLabs_API_SAT_send_diseqc_sequence   (SILABS_FE_Context *front_end,    signed   int
         sequence_length, unsigned char *sequence_buffer, unsigned char cont_tone, unsigned char
         tone_burst, unsigned char burst_sel, unsigned char end_seq);
signed   int  SiLabs_API_SAT_read_diseqc_reply      (SILABS_FE_Context *front_end,    signed   int
         *reply_length  , unsigned char *reply_buffer   );
signed   int  SiLabs_API_SAT_Tuner_Tune             (SILABS_FE_Context *front_end,    signed
         intfreq_kHz);
signed   int  SiLabs_API_SAT_Get_AGC                (SILABS_FE_Context *front_end);
#ifdef    UNICABLE_COMPATIBLE
signed   int  SiLabs_API_SAT_Unicable_Config        (SILABS_FE_Context *front_end,    signed   int
         unicable_mode, signed  int unicable_spectrum_inversion,  unsigned  int ub,  unsigned  int
         Fub_kHz,  unsigned  int Fo_kHz_low_band,  unsigned  int Fo_kHz_high_band );
signed   int  SiLabs_API_SAT_Unicable_Install       (SILABS_FE_Context *front_end);
signed   int  SiLabs_API_SAT_Unicable_Uninstall     (SILABS_FE_Context *front_end);
signed   int  SiLabs_API_SAT_Unicable_Tune          (SILABS_FE_Context *front_end,    signed   int
         freq_kHz);
#endif /* UNICABLE_COMPATIBLE */
#endif /* SATELLITE_FRONT_END */
#ifdef    TERRESTRIAL_FRONT_END
#ifdef    DEMOD_DVB_T
signed   int  SiLabs_API_Get_DVBT_Hierarchy         (SILABS_FE_Context *front_end,    signed   int
         *hierarchy);
#endif /* DEMOD_DVB_T */
signed   int  SiLabs_API_Get_DS_ID_Num_PLP_Freq     (SILABS_FE_Context *front_end,    signed   int
         ds_index, signed   int *data_slice_id, signed   int *num_plp, signed   int *freq);
signed   int  SiLabs_API_Get_PLP_ID_and_TYPE        (SILABS_FE_Context *front_end,    signed   int
         ds_id, signed   int plp_index, signed   int *plp_id, signed   int *plp_type);
signed   int  SiLabs_API_Get_PLP_Group_Id           (SILABS_FE_Context *front_end,    signed   int
         recall, signed   int plp_index, signed   int *group_id);
signed   int  SiLabs_API_Select_PLP                 (SILABS_FE_Context *front_end,    signed   int
         plp_id);
signed   int  SiLabs_API_Get_AC_DATA                (SILABS_FE_Context *front_end,    unsigned char
         segment, unsigned char filtering, unsigned char read_offset, signed   intbit_count, unsigned
         char *AC_data);
signed   int  SiLabs_API_TER_AutoDetect             (SILABS_FE_Context *front_end,    signed   int
         on_off);
signed   int  SiLabs_API_TER_T2_lock_mode           (SILABS_FE_Context *front_end,    signed   int
         T2_lock_mode);
signed   int  SiLabs_API_TER_ISDBT_Monitoring_mode  (SILABS_FE_Context *front_end,    signed   int
         layer_mon);
signed   int  SiLabs_API_TER_ISDBT_Layer_Info       (SILABS_FE_Context *front_end,    signed   int
         layer, signed   int *constellation, signed   int *code_rate, signed   int *il, signed   int
         *nb_seg);
signed   int  SiLabs_API_TER_Tuner_Fine_Tune        (SILABS_FE_Context *front_end,    signed   int
         offset_500hz);
signed   int  SiLabs_API_TER_Tuner_Init             (SILABS_FE_Context *front_end);
signed   int  SiLabs_API_TER_Tuner_SetMFT           (SILABS_FE_Context *front_end,    signed   int
         nStep);
signed   int  SiLabs_API_TER_Tuner_Text_status      (SILABS_FE_Context *front_end,    char *separator,
         char *msg);
```

```
signed   int  SiLabs_API_TER_Tuner_ATV_Text_status  (SILABS_FE_Context *front_end,    char *separator,
         char *msg);
signed   int  SiLabs_API_TER_Tuner_DTV_Text_status  (SILABS_FE_Context *front_end,    char *separator,
         char *msg);
signed   int  SiLabs_API_TER_Tuner_ATV_Tune         (SILABS_FE_Context *front_end);
signed   int  SiLabs_API_TER_Tuner_Block_VCO        (SILABS_FE_Context *front_end,    signed   int
         vco_code);
signed   int  SiLabs_API_TER_Tuner_Block_VCO2       (SILABS_FE_Context *front_end,    signed   int
         vco_code);
signed   int  SiLabs_API_TER_Tuner_Block_VCO3       (SILABS_FE_Context *front_end,    signed   int
         vco_code);
#endif /* TERRESTRIAL_FRONT_END */
#ifndef   NO_FLOATS_ALLOWED
signed   int  SiLabs_API_SSI_SQI                    (signed   int  standard, signed   int  locked,
         signed   int  Prec, signed   int  constellation, signed   int  code_rate, double CNrec, double
         ber, signed   int *ssi, signed   int *sqi);
#endif /* NO_FLOATS_ALLOWED */
signed   int  SiLabs_API_SSI_SQI_no_float           (signed   int  standard, signed   int  locked,
         signed   int  Prec, signed   int  constellation, signed   int  code_rate, signed   int
         c_n_100, signed   int  ber_mant, signed   int  ber_exp, signed   int *ssi, signed   int *sqi);
#ifdef    Si2183_DVBS2_PLS_INIT_CMD
signed   int  SiLabs_API_SAT_Gold_Sequence_Init     (signed   int gold_sequence_index);
signed   int  SiLabs_API_SAT_PLS_Init               (SILABS_FE_Context *front_end,    signed   int
         pls_init);
#endif /* Si2183_DVBS2_PLS_INIT_CMD */
signed   int  SiLabs_API_Demod_Dual_Driving_Xtal    (SILABS_FE_Context  *front_end_driving_xtal,
         SILABS_FE_Context  *front_end_receiving_xtal);
signed   int  SiLabs_API_Demods_Kickstart           (void);
signed   int  SiLabs_API_TER_Tuners_Kickstart       (void);
signed   int  SiLabs_API_Cypress_Ports              (SILABS_FE_Context *front_end,    unsigned char OEA,
         unsigned char IOA, unsigned char OEB, unsigned char IOB, unsigned char OED, unsigned char IOD
         );
signed   int  SiLabs_API_SSI_SQI_no_floats          (signed   int  standard, signed   int  locked,
         signed   int  Prec, signed   int  constellation, signed   int  code_rate, signed   int
         CNrec_1000, signed   int  ber_mant, signed   int  ber_exp, signed   int *ssi, signed   int
         *sqi);
void          SiLabs_API_Demod_reset                (SILABS_FE_Context *front_end);
#ifdef    GUI_SPECIFIC
signed   int  SiLabs_API_Skip_HW_Init               (SILABS_FE_Context *front_end,    signed   int
         standard);
#endif /* GUI_SPECIFIC */
signed   int  SiLabs_API_Store_FW                   (SILABS_FE_Context *front_end,    firmware_struct
         fw_table[], signed   int  nbLines);
signed   int  SiLabs_API_Store_SPI_FW               (SILABS_FE_Context *front_end,    unsigned char
         fw_table[], signed   int  nbBytes);

signed   int  SiLabs_API_Auto_Detect_Demods         (L0_Context* i2c, signed   int *Nb_FrontEnd, signed
         int  demod_code[4], signed   int  demod_add[4], char *demod_string[4]);

signed   int  SiLabs_API_TER_Tuner_I2C_Enable       (SILABS_FE_Context *front_end);
signed   int  SiLabs_API_TER_Tuner_I2C_Disable      (SILABS_FE_Context *front_end);
signed   int  SiLabs_API_SAT_Tuner_I2C_Enable       (SILABS_FE_Context *front_end);
signed   int  SiLabs_API_SAT_Tuner_I2C_Disable      (SILABS_FE_Context *front_end);
signed   int  SiLabs_API_Get_Stream_Info            (SILABS_FE_Context *front_end,    signed   int
         isi_index, signed   int *isi_id, signed   int *isi_constellation, signed   int
         *isi_code_rate);
signed   int  SiLabs_API_Select_Stream              (SILABS_FE_Context *front_end,    signed   int
         stream_id);
```

```
signed   int  SiLabs_API_TER_Tuner_Dual_Driving_Xtal(SILABS_FE_Context  *front_end_driving_xtal,
         SILABS_FE_Context  *front_end_receiving_xtal);

#ifdef    SILABS_API_TEST_PIPE
signed   int  Silabs_API_Test                      (SILABS_FE_Context *front_end,    char *target, char
         *cmd, char *sub_cmd, double dval, double *retdval, char **rettxt);
#endif /* SILABS_API_TEST_PIPE */

char*         SiLabs_API_TAG_TEXT                  (void);
```

10.7 Why are int types explicitly 'signed int' or 'unsigned int'?

This is because the Tizen OS compiler (promoted by Samsung) behaves differently from other compilers.
For all compilers, 'int' means 'signed int'
For Tizen, 'int' means 'unsigend int'.
Since this creates porting issues on Tizen platforms, the 'int' types have been explicitly replaced by 'signed int'.

10.8 How to build a project using the Skyworks API Wrapper?

To build a project using the wrapper, open your favorite IDE and:

1. Add the files required for your application.
   Each project requires drivers for:
   - The i2c communication layer
   - The terrestrial tuner (if TER reception is required)
   - The TER tuner wrapper (if TER reception is required)
   - The satellite tuner (if SAT reception is required)
   - The SAT tuner wrapper (if SAT reception is required)
   - The demodulator (the demodulator folder will also contain the demodulator-specific wrapper files)
   - The SAT LNB power controller (if SAT reception is required)
   - Unicable (if required for SAT applications)

2. Add each driver folder in your project settings, to allow the compiler to locate all the headers files.

For TER reception:

3. Define that you will build a terrestrial application.
   - Add 'TERRESTRIAL_FRONTEND' in your build options.
4. Define that you will be controlling TER tuners via the Skyworks TER tuner wrapper
   - Add 'TER_TUNER_SILABS' in your build options.
5. Define which terrestrial tuner your application will use
   - Add 'TER_TUNER_xxxx' in your build options. (example: TER_TUNER_Si2196)
6. Define which TER standards you will support.
   - Add 'DEMOD_DVB_T' in your build options if supporting DVB-T
   - Add 'DEMOD_DVB_T2' in your build options if supporting DVB-T2
   - Add 'DEMOD_DVB_C' in your build options if supporting DVB-C
   - Add 'DEMOD_DVB_C2' in your build options if supporting DVB-C2
   - Add 'DEMOD_MCNS' in your build options if supporting MCNS
   - Add 'DEMOD_ISDB_T' in your build options if supporting ISDB-T

For SAT reception:

7. Define that you will build a satellite application.
   - Add 'SATELLITE_FRONTEND' in your build options.
8. Define that you will be controlling SAT tuners via the Skyworks SAT tuner wrapper
   - Add 'SAT_TUNER_SILABS' in your build options.
9. Define which satellite tuner your application will use
   - Add 'SAT_TUNER_xxxx' in your build options. (example: SAT_TUNER_RDA5812)
10. Define which SAT standards you will support.

- Add 'DEMOD_DVB_S_S2_DSS' in your build options if supporting DVB-S, DVB-S2 and DSS
- Add 'DEMOD_DVB_S2X' in your build options if supporting DVB-S2X

11. Define whether you support Unicable or not.
- Add 'UNICABLE_COMPATIBLE' in your build options if supporting Unicable

12. Define which LNB controller you support.
- Add 'LNBxxx_COMPATIBLE' in your build options (example LNBH25_COMPATIBLE)

For all applications:

13. Define that you will be using the SKYWORKS SUPERSET code
- Add 'SILABS_SUPERSET' in your build options.

10.9 Why are there some strange text lines in the header files, generating compilation errors?

We use dedicated lines to check that the required compilation flags have been defined.
Generally, these lines are only reached by the compiler if none of the possible options is selected.
For instance, the Si2183 code can only work if one of the following flags is defined:

```
#ifndef    Si2183_B60_COMPATIBLE
 #ifndef    Si2183_A55_COMPATIBLE
  #ifndef    Si2183_A50_COMPATIBLE
   #ifndef    Si2183_ES_COMPATIBLE
    #ifndef    Si2180_B60_COMPATIBLE
     #ifndef    Si2180_A55_COMPATIBLE
      #ifndef    Si2180_A50_COMPATIBLE
       #ifndef    Si2167B_22_COMPATIBLE
        #ifndef    Si2167B_20_COMPATIBLE
         #ifndef    Si2169_30_COMPATIBLE
          #ifndef    Si2167_B25_COMPATIBLE
           #ifndef    Si2164_A40_COMPATIBLE
     "If you get a compilation error on these lines, it means that no Si2183 version has been
         selected.";
        "Please define Si2183_B60_COMPATIBLE, Si2183_A55_COMPATIBLE, Si2183_A50_COMPATIBLE,
         Si2183_ES_COMPATIBLE,";
                      "Si2180_B60_COMPATIBLE, Si2180_A55_COMPATIBLE, Si2180_A50_COMPATIBLE,";
                      "Si2167B_22_COMPATIBLE, Si2167B_20_COMPATIBLE, Si2169_30_COMPATIBLE,";
                      "Si2167_B25_COMPATIBLE or Si2164_A40_COMPATIBLE at project level!";
     "Once the flags will be defined, this code will not be visible to the compiler anymore";
     "Do NOT comment these lines, they are here to help, showing if there are missing project flags";
           #endif /* Si2164_A40_COMPATIBLE */
          #endif /* Si2167_B25_COMPATIBLE */
         #endif /* Si2169_30_COMPATIBLE */
        #endif /* Si2167B_20_COMPATIBLE */
       #endif /* Si2167B_22_COMPATIBLE */
      #endif /* Si2180_A50_COMPATIBLE */
     #endif /* Si2180_A55_COMPATIBLE */
    #endif /* Si2180_B60_COMPATIBLE */
   #endif /* Si2183_ES_COMPATIBLE */
  #endif /* Si2183_A50_COMPATIBLE */
 #endif /* Si2183_A55_COMPATIBLE */
#endif /* Si2183_B60_COMPATIBLE */
```

The above block of code will trigger an error on the text line only if none of the supported option is defined.
The good way to get rid of the error is NOT to comment the text line, but rather to define the compilation flag at project level, as the message states. This will make sure that you select at least one FW to be loaded in the demodulator.

10.10 How to use common source code for a Skyworks EVB and my own HW?

To allow the same source code to be used for a Skyworks EVB and customer HW, settings for the Skyworks EVBs are surrounded by '#ifdef SILABS_EVB'.

- To compile for a Skyworks EVB, add 'SILABS_EVB' in your project build options.
- To compile for custom HW, remove 'SILABS_EVB' from your project build options.

10.11 How to build a multi-front-end application using the wrapper?

Look for the FRONT_END_COUNT declaration in the wrapper console code and increase this number.
You can also define this in your project options.
The L3 wrapper console application can accommodate up to 4 front-ends.

The underlying code can in fact handle an unlimited number of front-ends, the limits being the HW capability to manage various i2c addresses and the memory size of the platform.

```
#define FRONT_END_COUNT 1
```

10.12 Where to find an example initialization code for my custom application?

Several options are possible:

- Use the existing configuration macros as a reference, and copy/rename/adapt them to your configuration. This process is described in '*How to use configuration macros for different HW platforms?*'
- Another option consists in compiling the console application and use its 'manual configuration' feature to generate the required function calls. This process is described in '*How to use the console application to generate my initialization code?*'.

10.13 How to add Unicable support to my application?

- Download the Unicable code from FTP:
- Add the code in a 'Unicable' folder to the project
- Add the Unicable folder to your project 'include directories'
- Define 'UNICABLE_COMPATIBLE' at project level.
- The application will be able to select Unicable or not using SiLabs_API_SAT_Unicable_Install / SiLabs_API_SAT_Unicable_Uninstall

10.14 How to use the console code as an example initialization code?

- If you are building your own application, you did not include the L3 console code and you must replace it by your own code. Nevertheless, you can use this code as a reference for the initialization phase (the software initialization).
- To avoid using 'malloc' in the code, the necessary structures are allocated at the very beginning of the code. Copy/paste this code in your application.

Below is the sample code when using the wrapper:

```
/* define how many front-ends will be used */
#define FRONT_END_COUNT  1
SILABS_FE_Context     FrontEnd_Table    [FRONT_END_COUNT];
```

```
SILABS_FE_Context    *front_end;
CUSTOM_Status_Struct  FE_Status;
CUSTOM_Status_Struct *custom_status;
```

- Change FRONT_END_COUNT from 1 to 4, depending on the number of front-ends you intend to manage.
- Look into the 'SiLabs_SW_config_from_macro' function of the L3 config macros code for the SW initialization code, and search for the available configuration macros in the L2 headers. These configuration macros match the available Skyworks EVBs, and can be used as examples of the SW init required for single/dual/triple/quad configurations. The same code can be copy/pasted in your own application later on. You can define several configuration macros for several HW platforms and use a single driver code for several configurations.

❖ NB1: The software initialization should only be executed once.

❖ NB2: During software initialization, no I2C traffic is done, it is only a phase where the required structures are allocated.

Below is the typical SW init code for a 'single' TER+SAT configuration:

```
#define SW_INIT_Si216x_EVB_Rev3_0_Si2164\
  /* SW Init for front end 0 */\
  front_end  = &(FrontEnd_Table[0]);\
  SiLabs_API_Frontend_Chip            (front_end, 0x2164);\
  SiLabs_API_SW_Init                  (front_end, 0xc8, 0xc0, 0x18);\
  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, 2);\
  SiLabs_API_TER_FEF_Config           (front_end, 1, 0xb, 1);\
  SiLabs_API_TER_AGC                  (front_end, 0x0, 0, 0xa, 0);\
  SiLabs_API_Select_SAT_Tuner         (front_end, 0x5815, 0);\
  SiLabs_API_SAT_Select_LNB_Chip      (front_end, 25, 0x10);\
  SiLabs_API_SAT_tuner_I2C_connection (front_end, 0);\
  SiLabs_API_SAT_Clock                (front_end, 1, 44, 24, 2);\
  SiLabs_API_SAT_Spectrum             (front_end, 1);\
  SiLabs_API_SAT_AGC                  (front_end, 0xd, 1, 0x0, 1);\
  SiLabs_API_HW_Connect               (front_end, 1);
```

10.15 How to use configuration macros for different HW platforms?

While the number of configuration macros increased, the corresponding code has been separated from the rest of the wrapper code, to make controlling it easier.

This is achieved in files:
- SiLabs_API_L3_Config_Macros.c
- SiLabs_API_L3_Config_Macros.h

The set of macros which can be accessed can be limited using 4 flags:

SiLabs_API_Wrapper\SiLabs_API_L3_Config_Macros.c

```
extern SILABS_FE_Context    *front_end;

#define   MACROS_FOR_SINGLE_FRONTENDS
#define   MACROS_FOR_DUAL_FRONTENDS
#define   MACROS_FOR_TRIPLE_FRONTENDS
#define   MACROS_FOR_QUAD_FRONTENDS

```

It consists in a set of 2 functions, using code folding to be more easily readable.

SiLabs_SW_config_possibilities is a text-based function which will return the names of available macros:

```
int   SiLabs_SW_config_possibilities   (char *config_macros) {
  int macro_count;
  sprintf(config_macros, "%s", "");
  macro_count = 0;

#ifdef    MACROS_FOR_SINGLE_FRONTENDS
  sprintf(config_macros,   "%s-----------single------------"      , config_macros);
 #ifdef    SW_INIT_Si2176_DVBTC_DC_Rev1_0
  sprintf(config_macros, "%s\n  Si2176_DVBTC_DC_Rev1_0"          , config_macros); macro_count++;
 #endif /* SW_INIT_Si2176_DVBTC_DC_Rev1_0 */
 #ifdef    SW_INIT_Si216x_EVB_Rev3_0_Si2164
 #ifdef    SW_INIT_Si216x_EVB_Rev3_0_Si2169
 #ifdef    SW_INIT_Si216x_SOC_EVB_Rev1_0_Si2164
 #ifdef    SW_INIT_Si2169_67_76_EVB_Rev1_1_Si2169
 #ifdef    SW_INIT_Si2169_67_76_EVB_Rev1_1_Si2167B
#endif /* MACROS_FOR_SINGLE_FRONTENDS */

#ifdef    MACROS_FOR_DUAL_FRONTENDS

#ifdef    MACROS_FOR_TRIPLE_FRONTENDS

#ifdef    MACROS_FOR_QUAD_FRONTENDS

  return macro_count;
}
```

SiLabs_SW_config_from_macro will execute the software initialization depending on the macro name it receives:

```
int   SiLabs_SW_config_from_macro     (char *macro_name)    {
  printf("SW configuration attempt for  '%s'  ...\n", macro_name);

#ifdef    MACROS_FOR_SINGLE_FRONTENDS
 #ifdef    SW_INIT_Si2176_DVBTC_DC_Rev1_0
  if ( strcmp_nocase( macro_name, "Si2176_DVBTC_DC_Rev1_0"            )==0) { SW_INIT_Si2176_DVBTC_DC_Rev1_0;          return   2165;  }
 #endif /* SW_INIT_Si2176_DVBTC_DC_Rev1_0 */
 #ifdef    SW_INIT_Si216x_EVB_Rev3_0_Si2164
 #ifdef    SW_INIT_Si216x_EVB_Rev3_0_Si2169
 #ifdef    SW_INIT_Si216x_SOC_EVB_Rev1_0_Si2164
 #ifdef    SW_INIT_Si2169_67_76_EVB_Rev1_1_Si2169
 #ifdef    SW_INIT_Si2169_67_76_EVB_Rev1_1_Si2167B
#endif /* MACROS_FOR_SINGLE_FRONTENDS */

#ifdef    MACROS_FOR_DUAL_FRONTENDS

#ifdef    MACROS_FOR_TRIPLE_FRONTENDS

#ifdef    MACROS_FOR_QUAD_FRONTENDS

  printf("invalid command line argument '%s'\n", macro_name);
  return 0;
}
```

As visible in the above code, each macro is accessible only when it is defined.
The macros are generally not defined in the SiLabs_API_L3_Config_Macros files but rather in the L2 header file of the source code they refer to. This automatically reduces the list of macros returned by SiLabs_SW_config_possibilities to the ones actually available.

However, an example QUAD macro using a Si2183-A (SW_INIT_quad_Si2169A) is provided for reference in SiLabs_API_L3_Config_Macros.h, as this is the most complex setup which can easily be achieved with the I2C addressing capabilities of tuners and demodulators.

10.16 How to use the console application to generate my initialization code?

When running the application code, if none of the macros listed at startup match your HW you can decide to go through the 'manual configuration' phase.
During this phase, you will be asked all questions related to your configuration.
Based on this information, the following actions will be taken:

The software initialization will be done according to your input.
The source code calls required to perform an identical initialization will be traced in the console window. If you wish, you'll be able to copy/paste these into your source code to use it either as a new macro or as the final initialization code for your application.

10.17 Can I automatically run the console code for a given configuration?

Yes, just create a .bat (batch) file with the name of the console executable followed by the name of the configuration macro.
It will then be automatically used to perform the SW init.

10.18 How to retrieve plp information in DVB-T2 or DVB-C2?

Imagine you have a MPL stream with:

- Data PLP with plp_id 7
- Common PLP with plp_id 5
- Data PLP with plp_id 12
- Data PLP with plp_id 23

After locking, the demodulator FW would store the plp ids in a table, in an unknown order (i.e. depending on the order used in the transmission Head-End). What we know is that the first index will be 0 and the last would be equal to then number of plps minus 1.
Think of this as a set of tables such as:

```
unsigned char PLP_ID[256];   /* the system allows max 256 PLPs */
unsigned char PLP_TYPE[256]; /* the system allows max 256 PLPs */
PLP_ID[0] = 7;    PLP_TYPE[0] = DATA_PLP;
PLP_ID[1] = 5;    PLP_TYPE[1] = COMMON_PLP;
PLP_ID[2] = 12;   PLP_TYPE[2] = DATA_PLP;
PLP_ID[3] = 23;   PLP_TYPE[3] = DATA_PLP;
```

Then we call the status function, to get some information on the T2 signal. During this, as the standard is T2, Si2183_L1_DVBT2_STATUS will be called.
Then,

- status->num_plp is the number of PLPs in the stream (it would be 4 in the example).
- status->plp_id is the id of the current PLP. If locking in 'auto' mode, it will correspond to the plp_id of the first DATA PLP listed in the table. (it would be 7 in the example)

From this point, we know that there are 4 PLPs, and the application needs to retrieve the corresponding plp_ids.
To do this, we use (as in Silabs_UserInput_SeekNext):

```
      for (i=0; i<num_plp; i++) {
        SiLabs_API_Get_PLP_ID_and_TYPE   (front_end, 0, i, &plp_id, &plp_type);
```

```
        if (plp_id == -1) {
          printf ("ERROR retrieving PLP info for plp index %d\n", i);
        } else {
          if (plp_type != SILABS_PLP_TYPE_COMMON) {
            carrier_index = SiLabs_Scan_Table_AddOneCarrier (standard, freq, bandwidth_Hz, stream,
         symbol_rate_bps, constellation, polarization, band, data_slice_id, plp_id, T2_base_lite);
          }
        }
      }
```

The final result is that we have added to our channel list 3 new channels, with the corresponding plp_id values 7, 12 and 23. COMMON PLPs are not added to the channel list.
Later on, when locking in 'non-auto' mode on these channels we will provide the plp_id to 'lock_to_carrier', and the demodulator will output the TS data resulting from the recombination of the selected DATA plp with the COMMON plp.

10.19 How to port the I2C layer to my platform?

The L0 layer is handling i2c communication.

It allows selecting between several communication modes.

Usual communication modes are:

```
typedef enum CONNECTION_TYPE
{
    SIMU = 0,
    USB,
    CUSTOMER,
    none
}  CONNECTION_TYPE;
```

- SIMU is useful to test the application's startup without sending any i2c data.
- USB is used to communicate via the Cypress USB/I2C interface implemented on Skyworks EVBs.
- CUSTOMER is the custom mode fitting the customer applications.

1. To adapt the source code to your application, you need to adapt the case 'CUSTOMER' in the functions shown below.
2. You will also need to change the communication mode to CUSTOMER during the first call to SiLabs_API_HW_Connect (when using the API Wrapper) or the corresponding function handling the HW connection in your application.

Abstract of L0_ReadBytes:

```
int     L0_ReadBytes         (L0_Context    *i2c,
                              unsigned int   iI2CIndex,
                              int            iNbBytes,
                              unsigned char *pucDataBuffer
                              ) {
. . .
  switch (i2c->connectionType) {
#ifdef    USB_Capability
    case USB:
. . .
#endif /* USB_Capability */
```

```
    case CUSTOMER:
      /* <porting> Insert here whatever is needed to
      read iNbBytes bytes
      from the chip whose i2c address is i2c->address,
      starting at index iI2CIndex,
      with an index on i2c->indexSize bytes
      the data bytes being stored in pucDataBuffer.

      Make it such that on success nbReadBytes = iNbBytes
      and on failure nbReadBytes = 0.
      */
#ifdef    CUSTOMER_ReadI2C
                          ADD your code HERE
#endif /*  CUSTOMER_ReadI2C */
      break;
    case SIMU:
. . .
    default:
. . .
  }
. . .
    return nbReadBytes;
}
```

Abstract of L0_WriteBytes:

```
int     L0_WriteBytes        (L0_Context    *i2c,
                              unsigned int   iI2CIndex,
                              int            iNbBytes,
                              unsigned char *pucDataBuffer
                              ) {
. . .
  switch (i2c->connectionType) {
#ifdef    USB_Capability
    case USB:
. . .
#endif /* USB_Capability */
    case CUSTOMER:
      nbWrittenBytes = 0;
        /* <porting> Insert here whatever is needed to
        (option 1)
        write iNbBytes bytes
        to the chip whose i2c address is i2c->address,
        starting at index iI2CIndex,
        with an index on i2c->indexSize bytes
        the data bytes being stored in pucDataBuffer.

        (option 2)
        Another option is to
        write iNbBytes + i2c->indexSize bytes
        to the chip whose i2c address is i2c->address,
        the index bytes and data bytes all being stored in pucBuffer.

        Make it such that on success nbWrittenBytes = iNbBytes + i2c->indexSize
        and on failure write_error is incremented.
```

```
        */
#ifdef    CUSTOMER_WriteI2C
                          ADD your code HERE
#endif /* CUSTOMER_WriteI2C */
        break;
    case SIMU:
. . .
    default:
      break;
  }
. . .
  return nbWrittenBytes - i2c->indexSize;
}
```

10.20 How to validate the I2C porting?

With a demodulator:

- In most applications where a demodulator is used, the i2c can only reach the tuners via the demodulator's i2c pass-through. Therefore, i2c validation should be done while testing communication with the demodulator.
- In many applications, the demodulator will be using the clock signal from one of the tuners, and even possibly change its clock source when switching from a terrestrial standard to a satellite standard or vice-versa. This means that to perform the demodulator initialization the tuner providing the clock signal to the demodulator will need to output a clock.
- The above 2 points represent a situation where is becomes complex to start the application, as the demodulator clock may come from a tuner, and this tuner can only be accessed if the application can 'enable' the demodulator's i2c pass-through.
- To handle this situation, the demodulator i2c pass-through can be set without the demodulator receiving a clock signal (it uses the i2c SCL signal instead of a regular clock). This is the reason why we will use it to validate I2C communication.
    - With a 'command' mode demodulator
        1. Enabling the demodulator's I2C pass-through.
            - Write '0xc0 0x0d 0x01' in the demodulator (usual base-address 0xc8, depending on the demodulator). This is similar to calling the *demodulator*_L2_Tuner_I2C_Enable function.
        2. Reading the demodulator's status byte
            - Read one byte from the demodulator.
            - This should return '0x80', meaning that the CTS bit is set and there were no errors.
            - '0x00' or '0xff' would mean that i2 communication is not working.
        3. Reading the tuner's status byte
            - Read one byte from the tuner (no index).
            - This should return '0xC0' or '0x80', meaning that the CTS bit is '1'.
            - '0x00' or '0xff' would mean that i2 communication is not working.

With a tuner and **NO** demodulator:

1. Reading the tuner's status byte.
    - Read one byte from the tuner (no index).
    - This should return '0xC0' or '0x80', meaning that the CTS bit is '1'.
    - '0x00' or '0xff' would mean that i2 communication is not working.

10.21 Why are there SiERROR lines in addition to SiTRACE lines in the code?

The SiTRACE and SiERROR features are described in details in the L0 documentation. Please refer to this documentation for all details.

In brief, the SiTRACE messages allow tracing the code behavior during execution, while the SiERROR messages specifically deal with errors.

During the initialization phase, we try to make the code stop and return an error on the first error, as a failed initialization will forbid the application to work anyway.

The purpose is to make sure errors are detected and dealt with before going further.

If initialization fails, the application can get useful information on the reason for the fail calling the L0_ErrorMessage function and displaying it to the user.

As the SiTRACE lines can be removed from the compilation, it is recommended to keep the SiERROR lines in the code during the development phase, to help detect possible errors.

10.22 How to remove SiTRACE messages from the source code?

To remove all traces from the source code a single line needs to be commented in Silabs_L0_API.h:

```
/* Uncomment the following line to activate all traces in the code */
/*#define SiTRACES*/
```

NB: We recommend keeping the SiTRACE lines in the code, in order that they can be re-activated at will for debug purposes. Reading the SiTRACE output will save time compared to capturing the i2c traffic to reverse-engineer it to know what happens during execution.

10.23 How to remove SiERROR messages from the source code?

To remove all error traces from the source code a single line needs to be commented in Silabs_L0_API.h:

```
#define SiERROR            L0_StoreError
```

NB: We recommend keeping the SiERROR lines in the code, in order that they can be re-activated at will for debug purposes. Reading the L0_ErrorMessage output will save time to identify errors during execution.