SP4x Development Kit

Quickstart Guide

About this document

Scope and purpose

This document shall explain all necessary steps to start a development for SP4x products using the SP4x development kit.

Intended audience

Intended audience are all firmware and hardware engineers working in a development involving the SP4x sensor.

Table of contents

1 Content of the Kit

The kit contains the following items:

- SP4x Development Board v5
- SP4x Prog-Adapter (soldered on the SP4x Development Board)
- Programming-adapter for programming of additional samples
- SP4x samples
- Documents containing important information

This document shall briefly explain the functionality and usage of the development Kit.

2 Setting up the software

There is several software involved in the SP4x debugger toolchain. The installation procedure must follow the order shown below:

1. µVsion Keil IDE minimum Version: 4
2. SP4xTargetGen2 Keil Plugin minimum Version: 1.0.0

2.1 Installation of Keil IDE

Keil µVision is an integrated development environment, which includes an editor, linker, compiler and a flash tool. A Keil µVision evaluation version may be downloaded from the internet (www.keil.com). There select '*C51 Compiler (Eval Tools)*' to install Keil µVision and follow the instructions in the installation routine.

2.2 Installation of the SP4x Keil Plugin

In order to make Keil able to communicate with the development board, a Keil plugin in needed. An installer is provided which takes care of installing the Keil plugin and adding it to the plugin-list of Keil. In order install the plugin, please execute the provided "SP4xTargetGen2_Installer.exe". The installer will search for the Keil installation path, check if the C51 toolchain is installed and finally copy the plugin-dll to the right place and modify Keil's TOOLS.INI file to make Keil recognize the new plugin. When the installer finished execution, make sure that everything worked as expected. The output of the window should like as shown in the next figure.

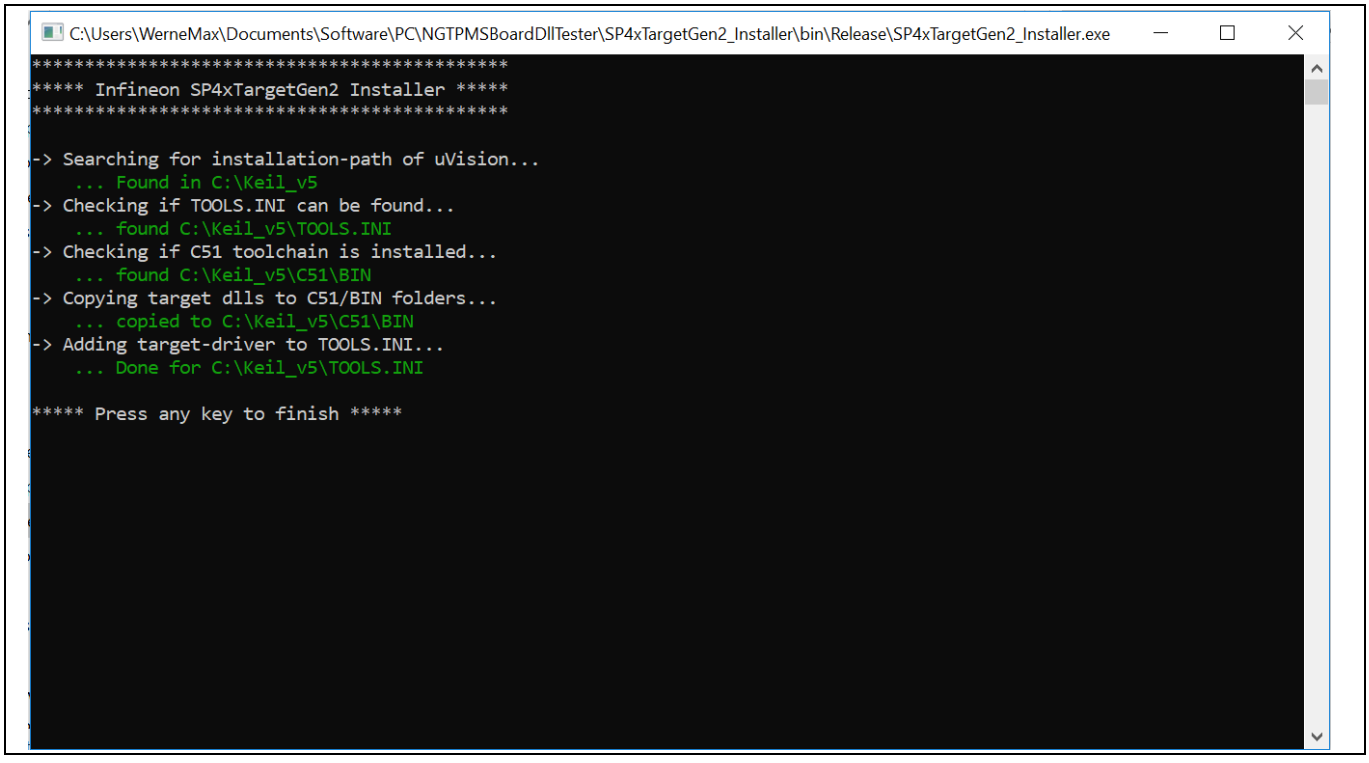

Figure 1 Output of the plugin installer

Should the output show any error, then the installation was not succesfull and the plugin will not be usable. In this case please contact us!

2.3 First Time Keil Configuration

When the complete toolchain itself is properly setup, Keil IDE must be configured as well. Therefore some settings and prerequisites in the Keil project must be done.

Each Keil project must be configured to use the correct target driver. Therefore open the project's settings right-clicking on "Options for Target" as shown in the next figure.

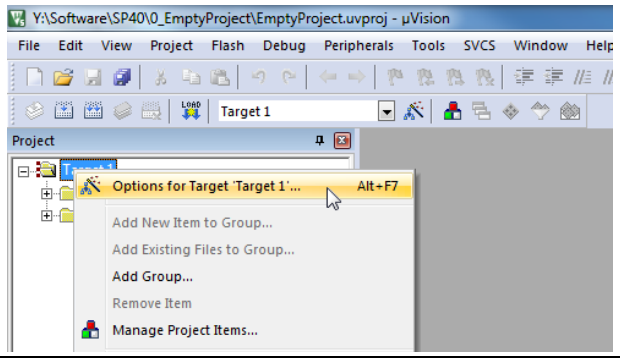

Figure 2 Opening settings of a Keil project

In the window that will open afterwards go to the Tab "Debug" and apply the settings as visible in the next figure.

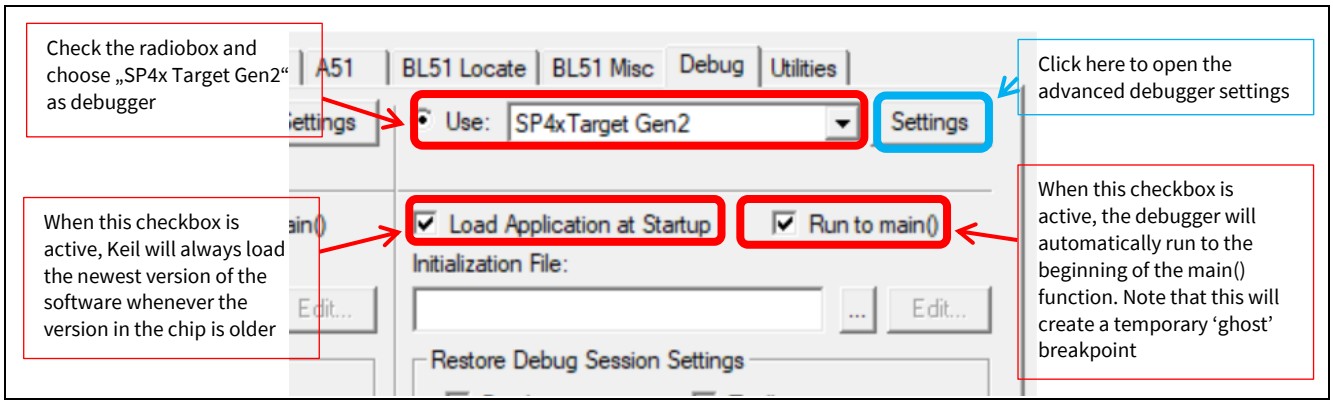

Figure 3 Basic debug settings in a Keil project

Afterwards click on "Settings" in order to open the advanced settings of the selected debugger. Here several settings have to be made, as shown in the next figure.

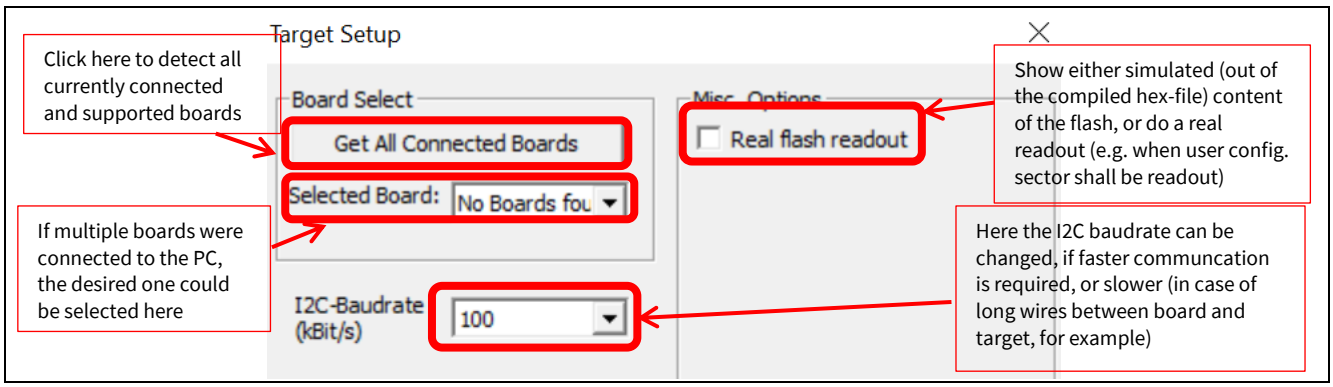

Figure 4 Advanced debug settings in a Keil project

In order that Debugging among the written C-code is possible, also options under “Output” in the project’s target options must be selected, as shown in the next figure.

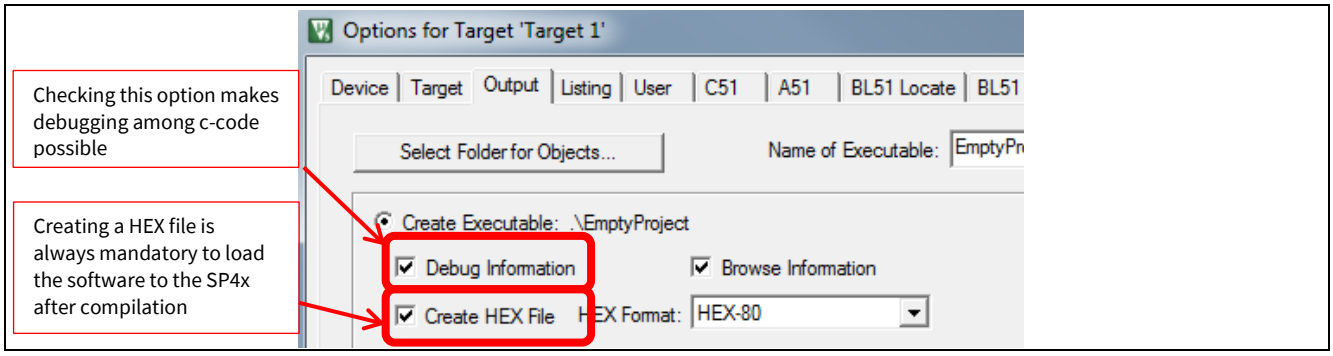

Figure 5 Output settings for debugging a Keil project among c-code

Since the generated hex-file must be loaded to the device prior to debugging it, this hex-file must be generated and furthermore the correct tool for flashing the file must be selected under “Utilities” in the project’s target option as illustrated in the next figure.

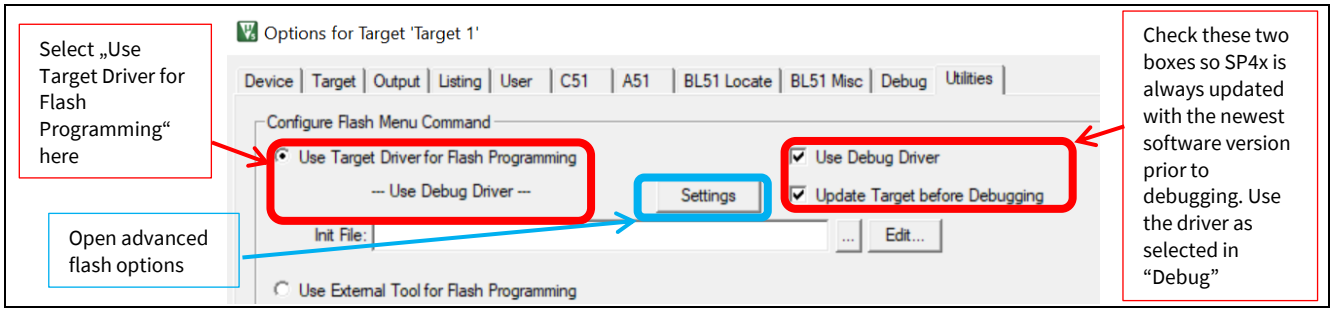

Figure 6 Settings in the "Utilities" section of a Keil project

After a click on “Settings” another window appears where the following settings must be applied.

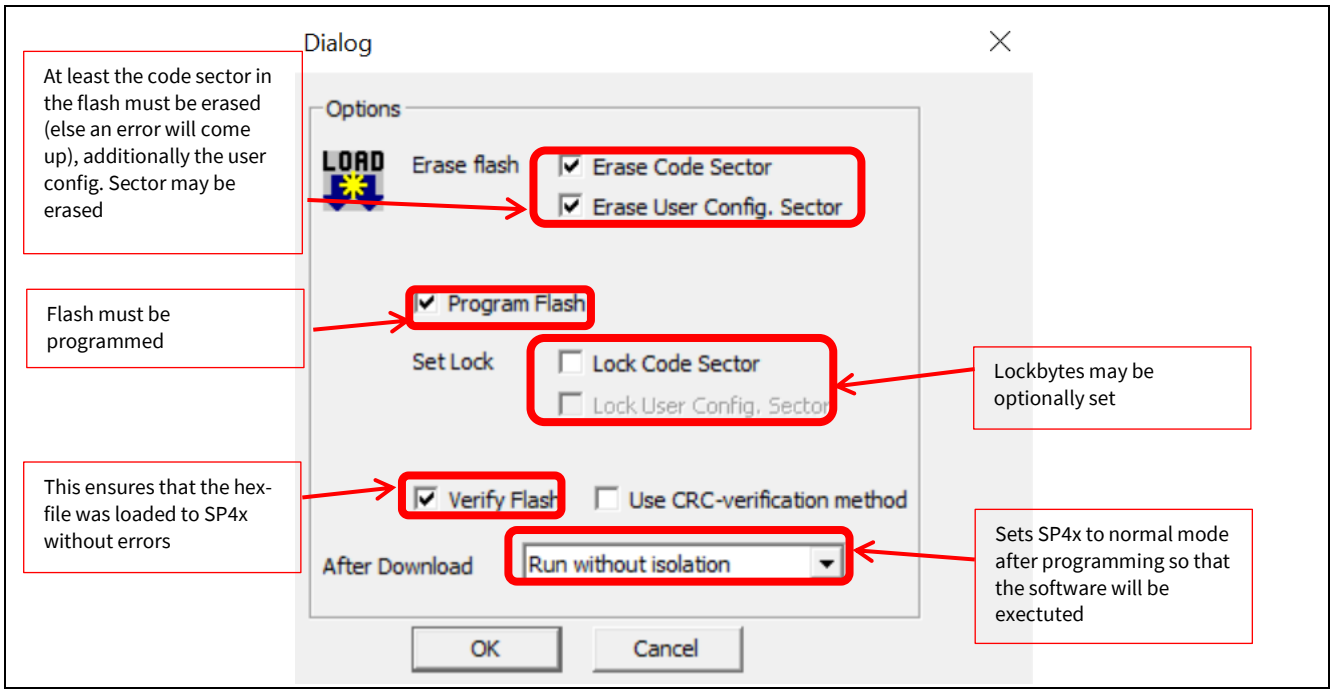

Figure 7 Settings of the SP4x target driver flash utility

In the last dialog, for option “After Download”, there are several option possible:

1. “Run without isolation”: This resets the SP4x to normal mode after flash downloading, with the PPx-pins NOT isolated from the programmer. This means that the PPx-pins are routed through a level-shifter towards the programming adapter and therefore UART for example still works. Current consumption of the device may be a bit higher due to some potential leakage current through that path.
2. “Run with isolation”: In contrast to point 1, the PPx-pins are isolated from the programming adapter. This means that UART cannot work anymore, but there is no additional leakage current. This setting is recommended for precise current-measurements.
3. “Do not run”: The device will remain in programming mode after programming was finished. The flashed application will not start to be executed.

3 Hardware description

The SP4x development board consists of several blocks that shall help to develop applications for the SP4x TPMS device. An overview over these blocks shall be given in **Error! Reference source not found.**.

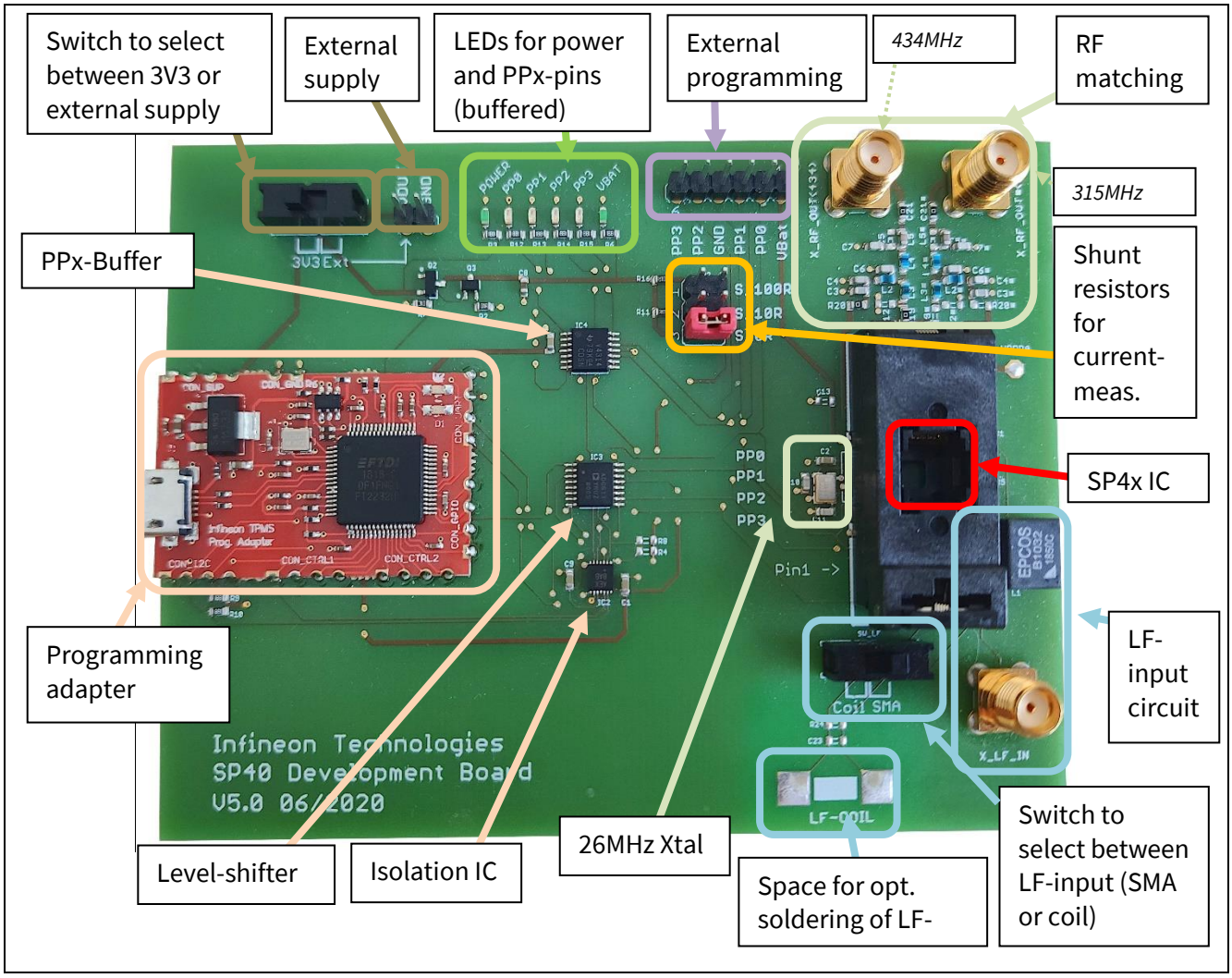

Figure 8 SP4x development board blocks

Additionally to the default setup of the board, determined by several 0Ω resistors, some changes may be applied.

3.1 Selecting an appropriate matching

For only RF testing (like for instance checking if appropriate RF telegram being transmitted), both matching can be let soldered on the board. If instead RF performance is planned to be tested, only one matching shall be present. In order to disconnect the unused matching, the resistors R19 and R20 of the unused matching shall be unsoldered as shown in the next figure.

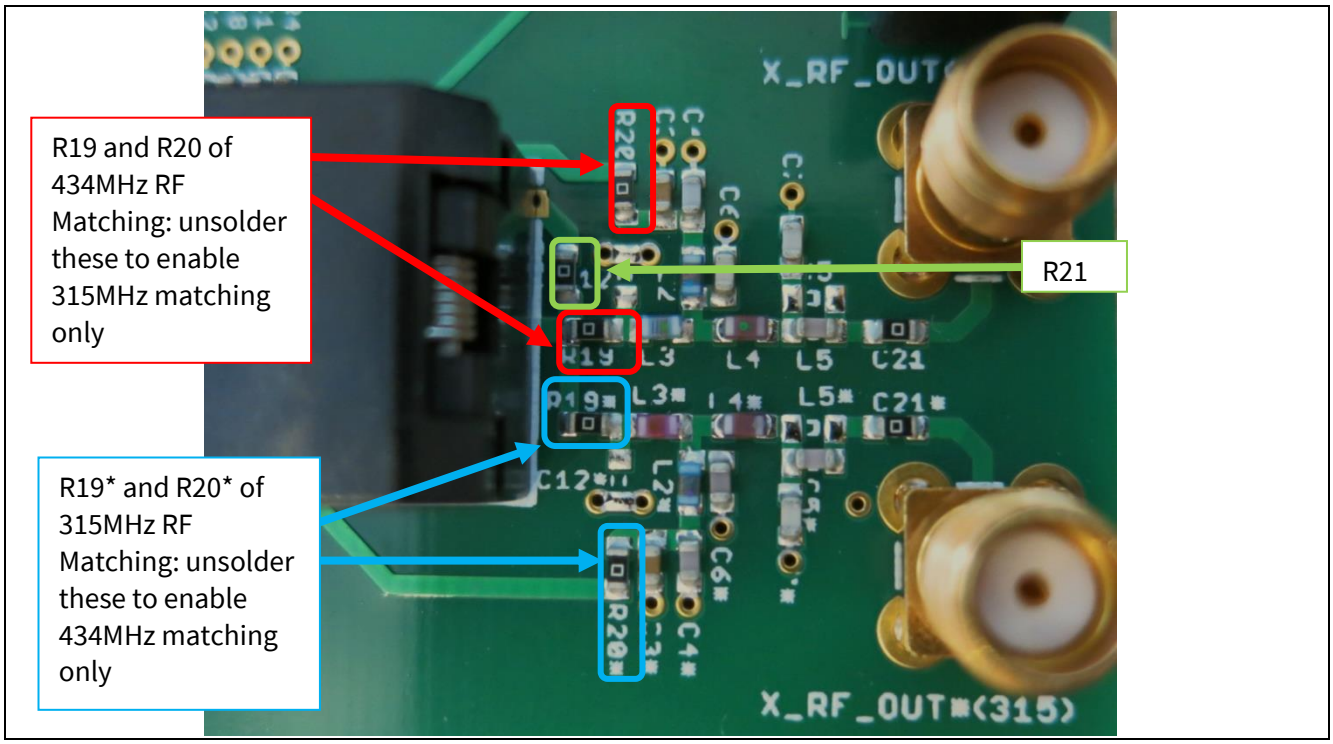

Figure 9 Two different matchings on the board

3.2 Soldering the SP4x directly on the board

One further option is to solder the SP4x device directly onto the board. Therefore R21 (see Figure 9) must be unsoldered. Note that due to this change the impedance of the matching changes at the PA pin, thus an adoption of the matching must be done afterwards. If not done so, a negative effect on the output power is most likely.

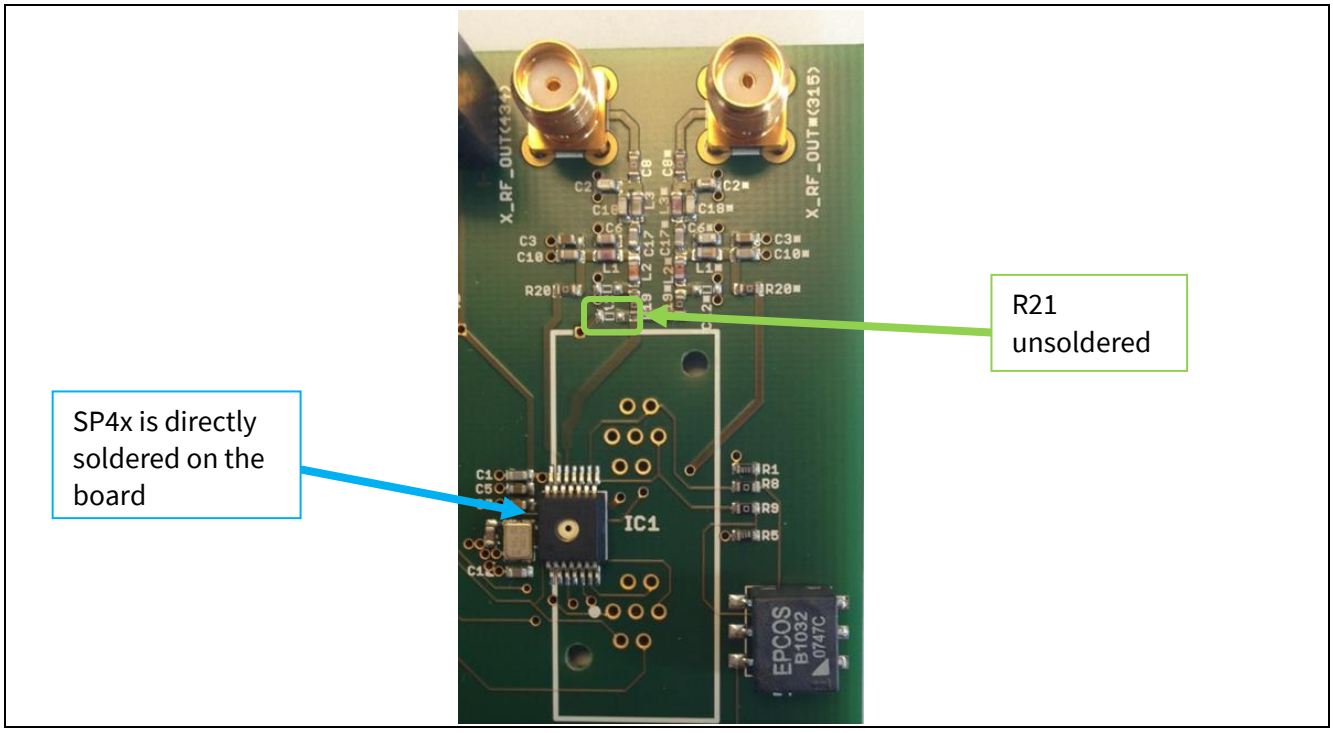

Figure 10 SP4x soldered directly on the development board

4 Setting up the hardware

In order to be able to use the SP4x Development board, some arrangements have to be made. First of all the SP4x IC must be placed properly in the socket. Here it is important that each PIN of the IC is placed on the respecting connector pin inside the socket, else debugging and programming may result in an error (like “Flash erase failed”). The correct orientation of the IC is shown in the next figure.

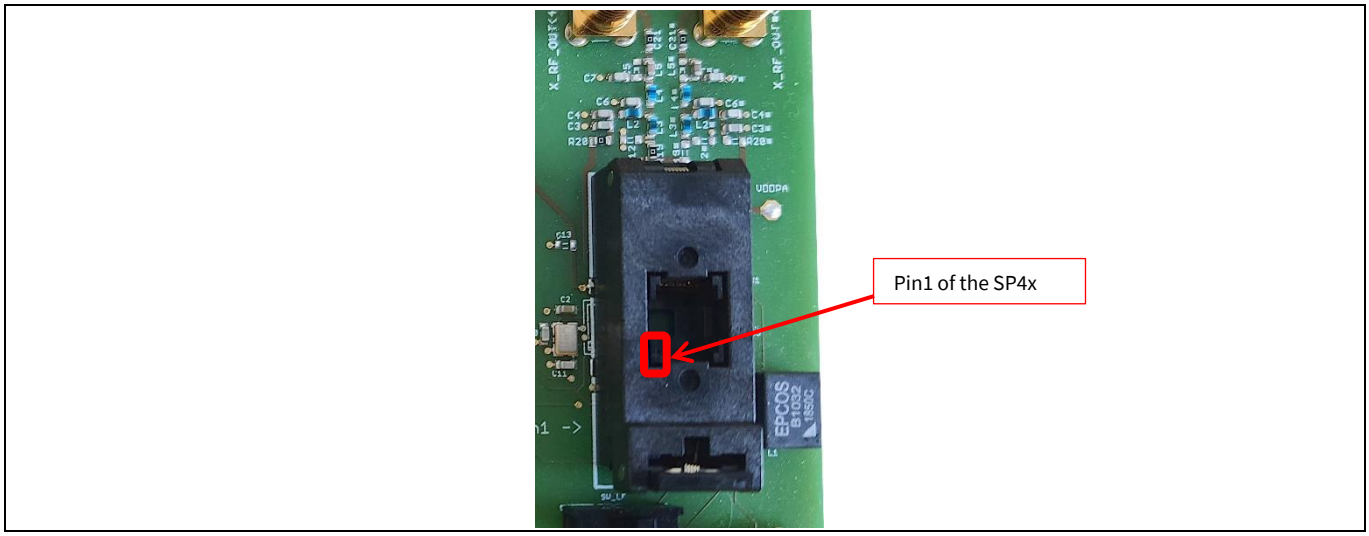

Figure 11 Correct orientation of the SP4x IC inside the holder

Afterwards a jumper must be set, as illustrated in red in the next figure.

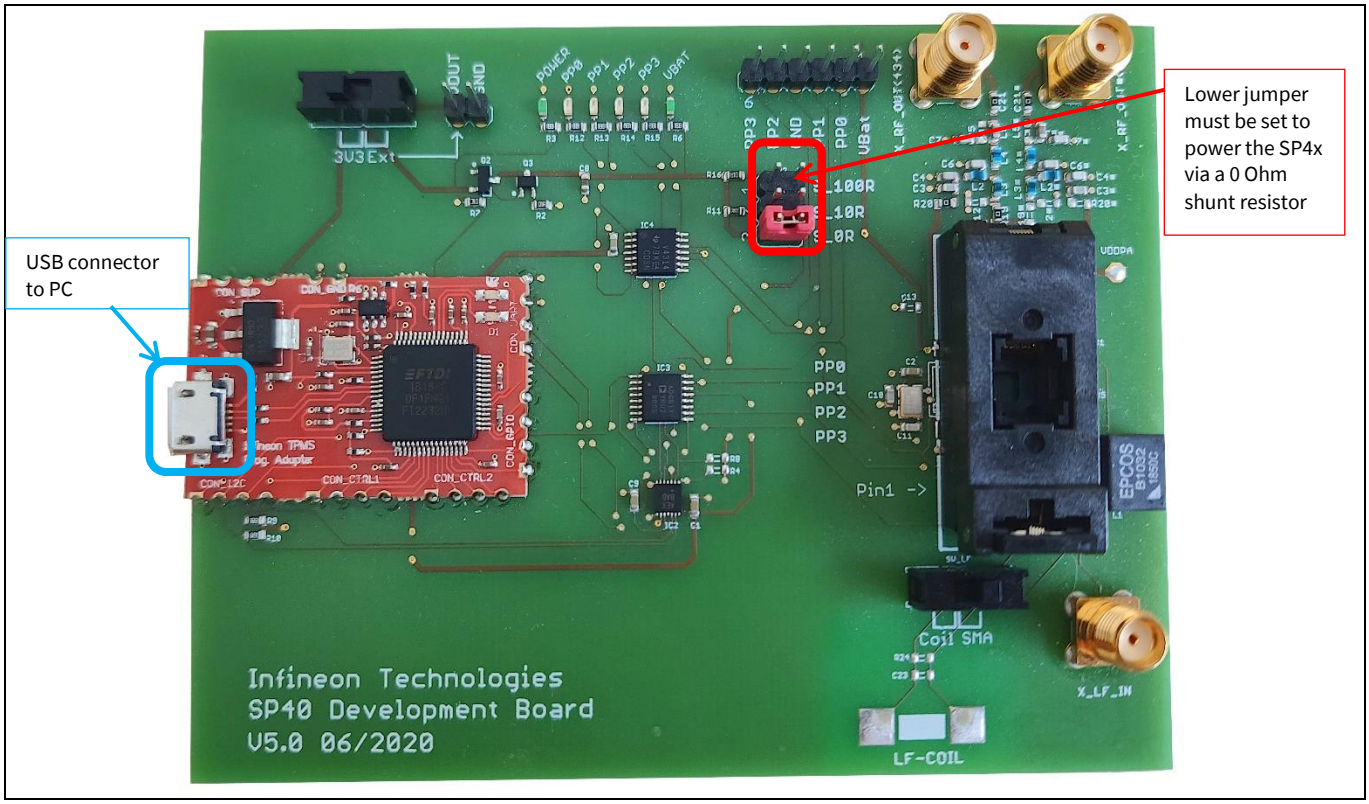

Figure 12 Setting of the required jumper on the SP4x development board

5 Example programs

The programs described in the following are to be downloaded from the MyICP library. They allow an easy start with the SP4x.

5.1 Preparation

Copy the source code of the examples from the folder '*Software Examples'* in the MyICP library to your local drive.

Note: If set, remove the write protection of the files. Otherwise Keil µVision has no write access to the files in the projects and causes an error message.

The example programs are organised in separate Keil Projects, where each program resides in a separate folder. To open an example program, open the according folder and double click on the file with the extension *".uvproj"*. The Keil IDE opens and shows the project. Afterwards check the settings of the project, according to chapter **Error! Reference source not found.**.

5.2 Example 0: EmptyProject

The purpose of this project is to have an empty project with all basic Keil settings and all needed header and library files already included. This is the starting point for every software development.

5.3 Example 1: Blinky

After opening the project with Keil, rebuild the project as shown in **Error! Reference source not found.**.

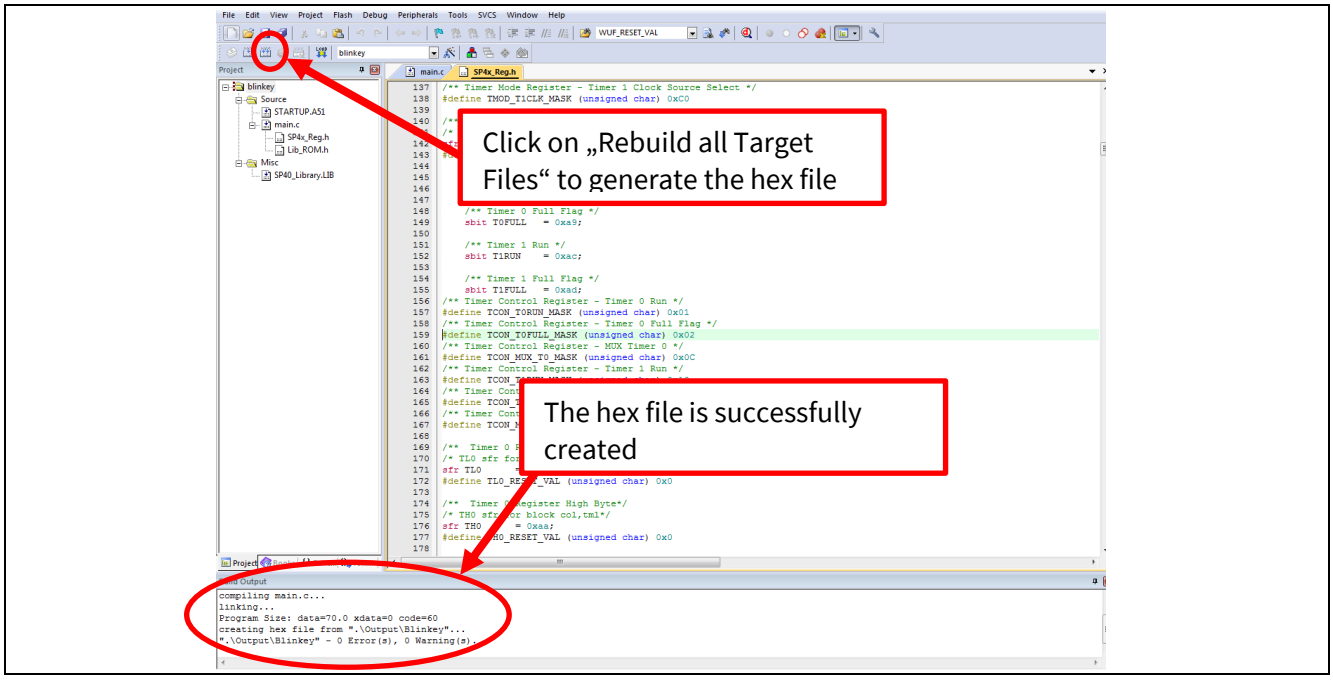

Figure 13 Rebuild the project in Keil

After successful compilation, the program must be loaded inside the SP4x Flash. To achieve this, download the program inside the SP4x Flash by clicking on the button "*Download to Flash Memory*", as shown in **Error! Reference source not found.**.

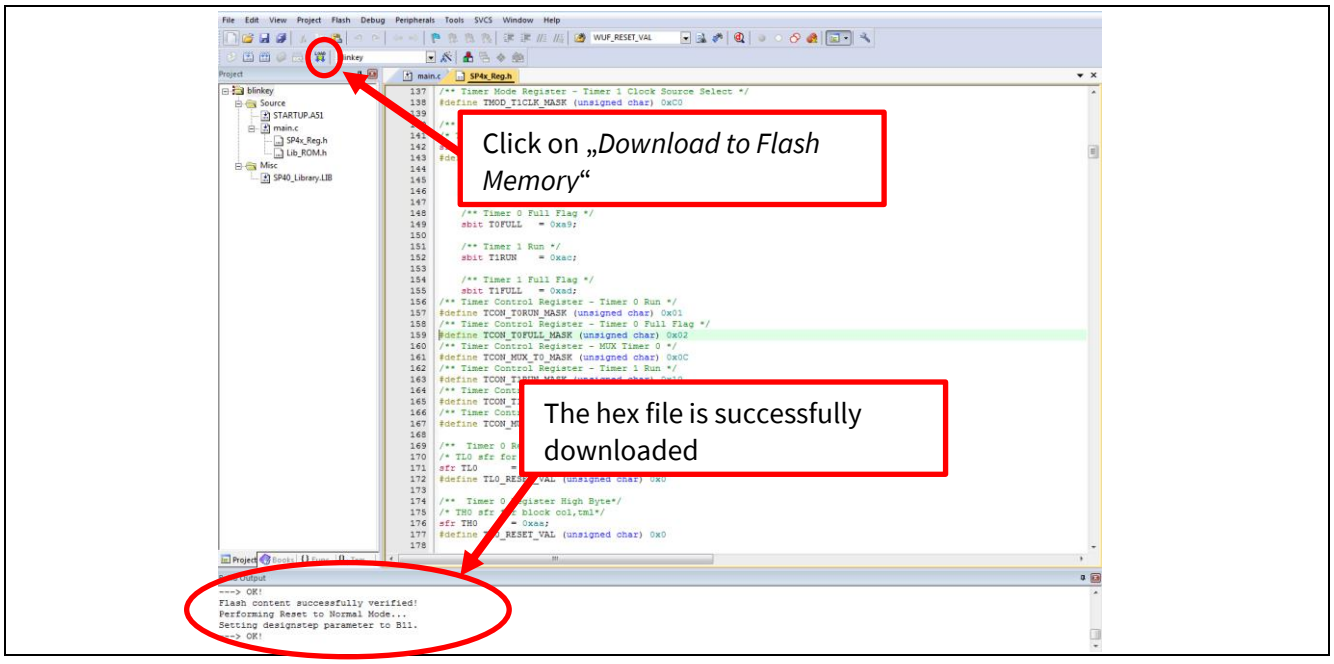

Figure 14 Download the program to the SP4x

The purpose of this program is to use the GPIOs of SP4x by changing the output state. As the GPIOs are connected to the LEDs, it is possible to observe easily the level of the GPIOs. For this example, it is the output state of PP2 that is modified in different interval by usage of the auxiliary-library function Aux_Blink_LED().

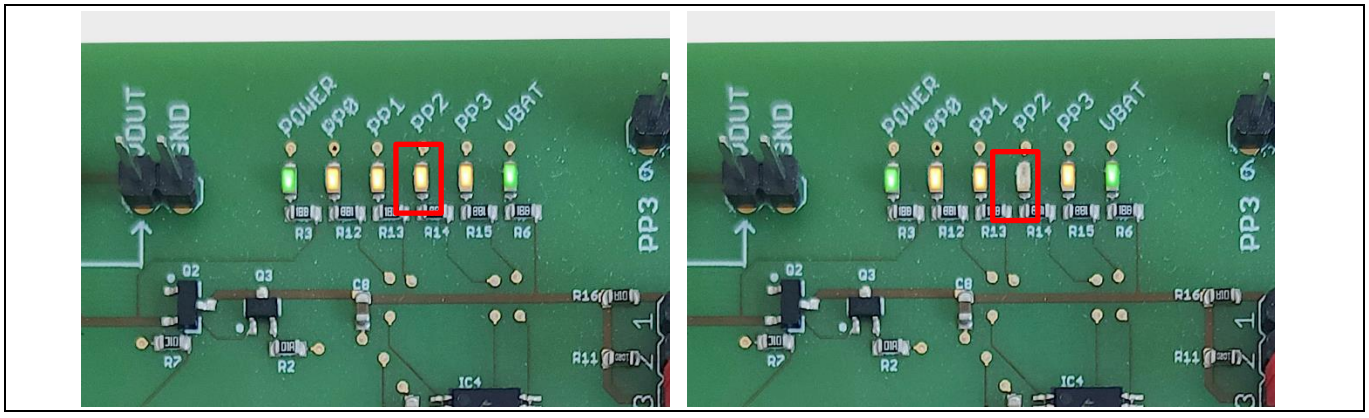

Figure 15 Observe the LED blinking at PP2

5.4 Example 2: LowPowerStates

Purpose of this program is to set the SP4x into all different low power states. The state can be defined via a pre-compiler option (Powerdown, Deep Idle, Tx Low Power, Idle). The device shall stay in each power state for the longest possible amount of time in order to enable current measurements.

Open the associate project "*SP4x_Powerdown.uvproj*" and follow the instructions as presented in the example 1.

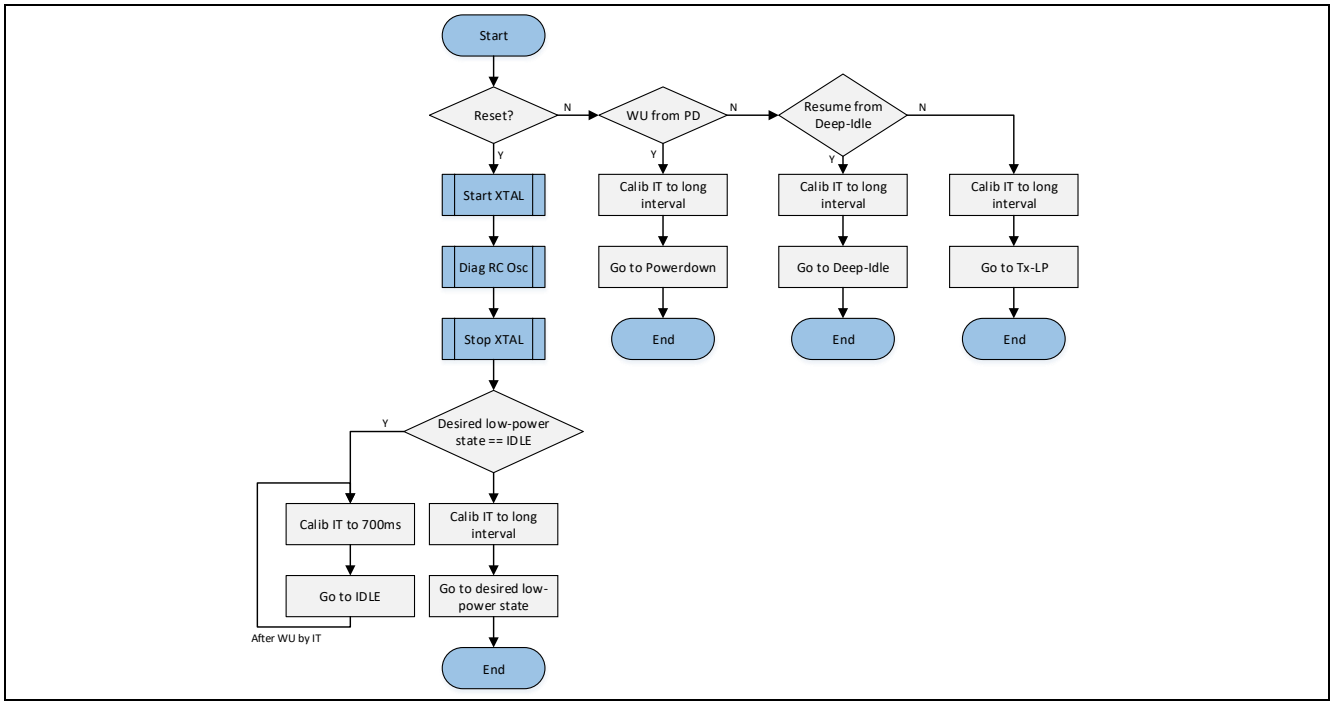

Figure 16 Flowchart of Example 2 (LowPowerStates)

For measuring the current consumed by the SP4x a shunt resistor may be selected between VDD and VBat via JP2. This will result in a voltage across this resistor while current is flowing through this resistor into the SP4x. Additionally the application should be run "with isolation" (see Figure 7) in order to avoid any current flowing out/into those pins from external. An exemplarily setup of the jumpers is given in the next figure.

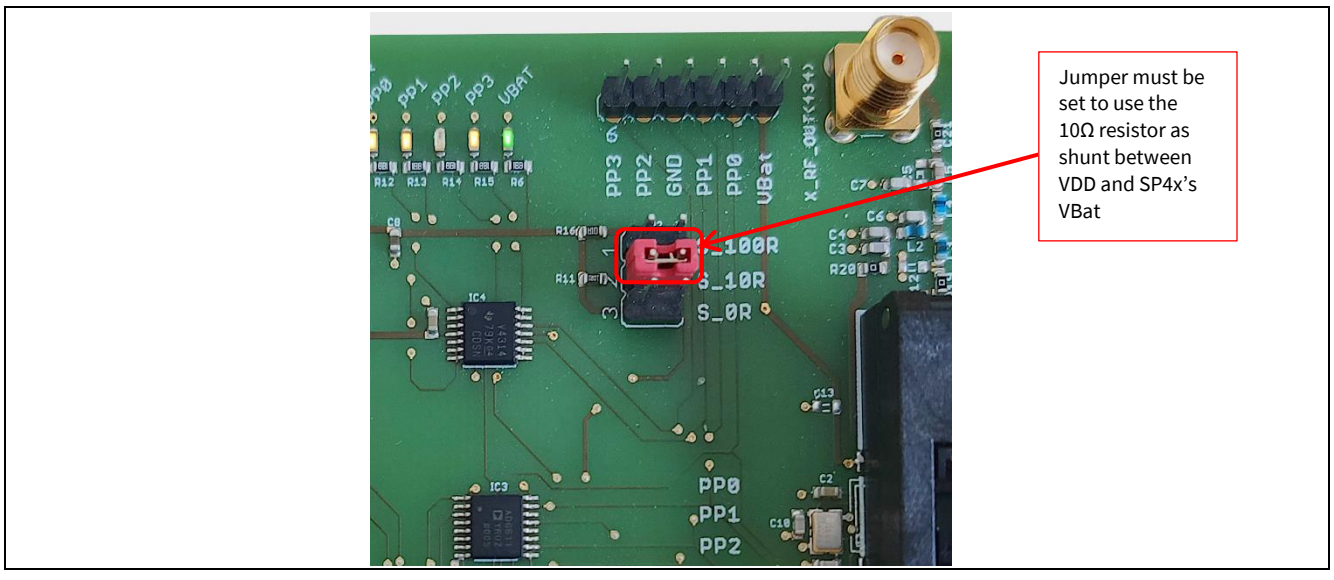

Figure 17 Jumper-setup for measurement of SP4x supply-current

For finally measuring this voltage, it is recommended to use an amplifier in combination with an oscilloscope in order to observe the current flowing into the SP4x's VBat pin. An example for such a setup is given in the next figure.

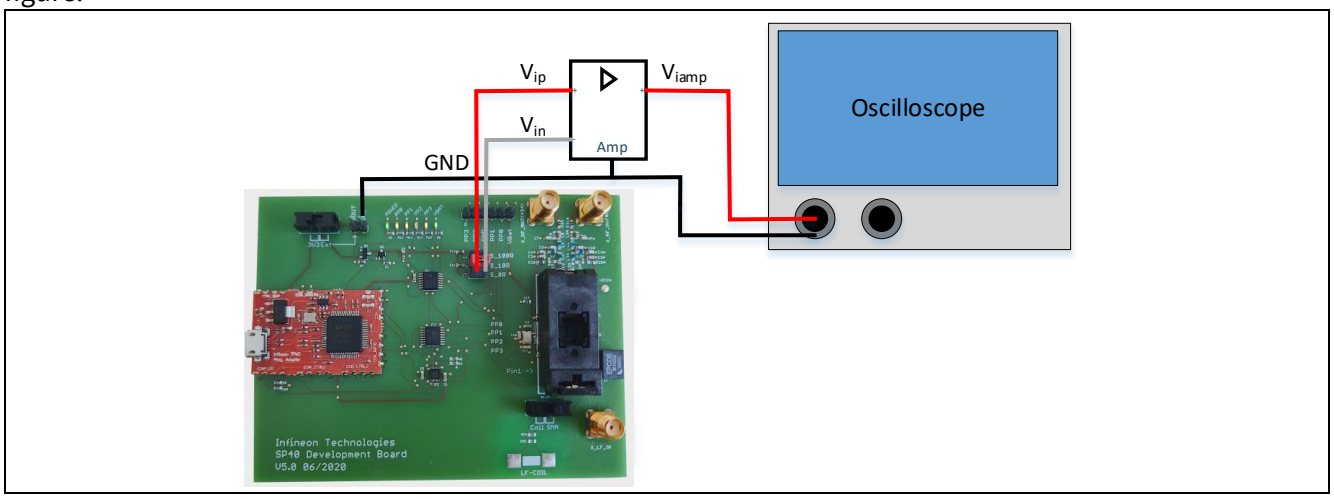

Figure 18 General measurement setup for current measurements

Alternatively to the UWLink's supply voltage, any other external voltage may be used to power to SP4x. Note that in this case, the switch "SW_VDUT" must be set to position "EXT". The setup using an external voltage for current measurements is given in the next figure.

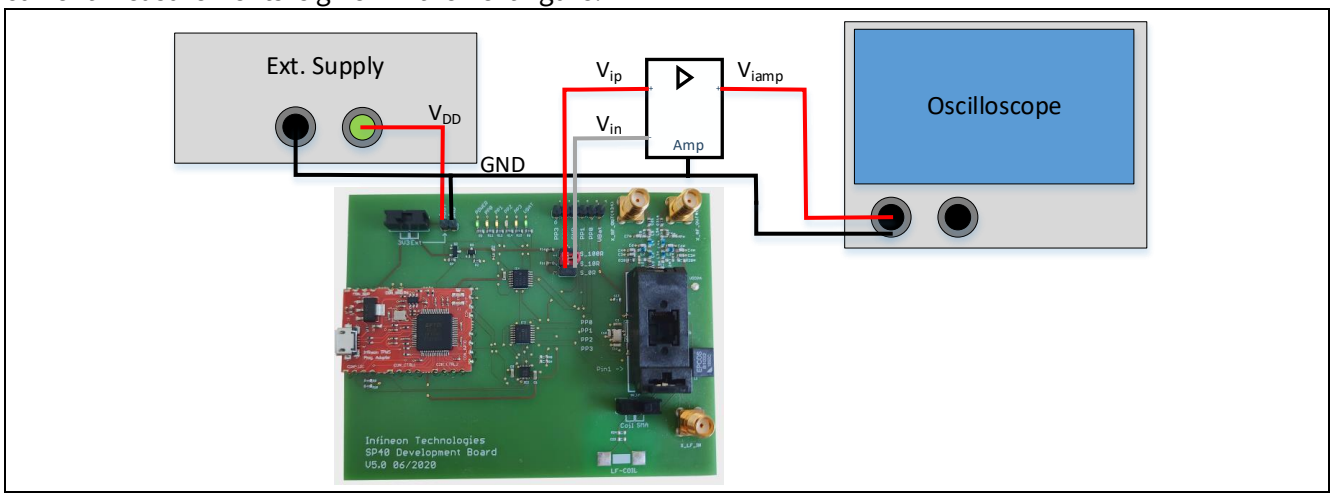

Figure 19 General measurement setup for current measurements with ext. supply

5.5 Example 3a: SensorMeasurements

The purpose of this program is to measure the sensor functionalities (pressure, acceleration, temperature and battery voltage) of the SP4x. The next figure shows how the sensor measurements example is implemented.

Open the associate project “SP4x_Sensor_Meas.uvproj” and follow the instructions as presented in the example 1.

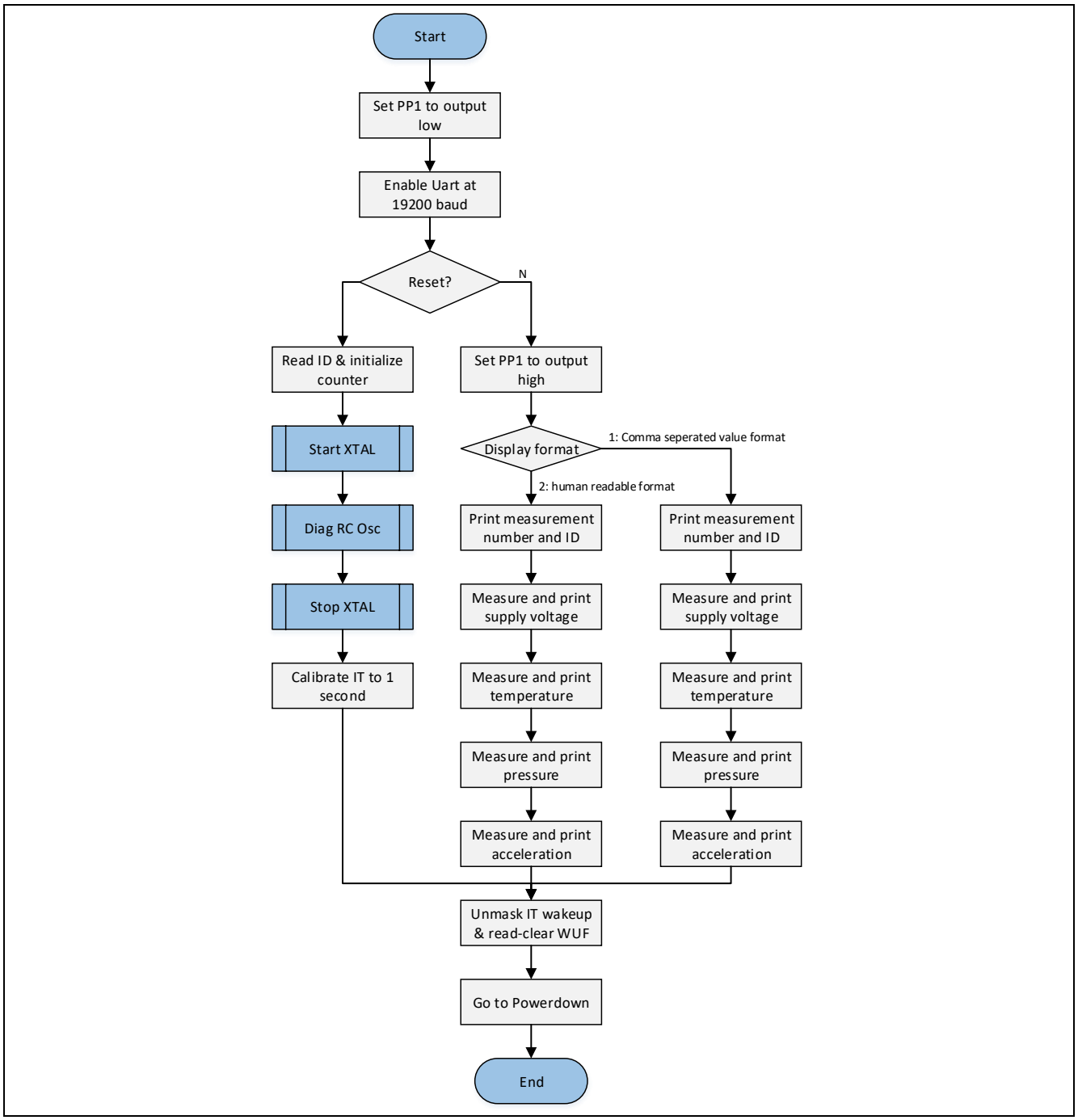

Figure 20 Flowchart of Example 3a (SensorMeasurements)

5.5.1 Usage of the serial connectors

The SP4x development board will generate a virtual COM-port on the target PC. Pre-requisite is that the FTDI-drivers are installed. The Uart-to-USB chip is an FTDI FT2232RL and the drivers are normally installed automatically after connecting to a PC. If not, they can be obtained from the FTDI homepage (http://www.ftdichip.com/Drivers/D2XX.htm). After drivers are installed properly, the generated COM-port may be found out by either observing the installation window of the drivers (see Figure 21) or by looking it up in windows' control panel -> hardware and sound -> device manager (as seen in Figure 22).

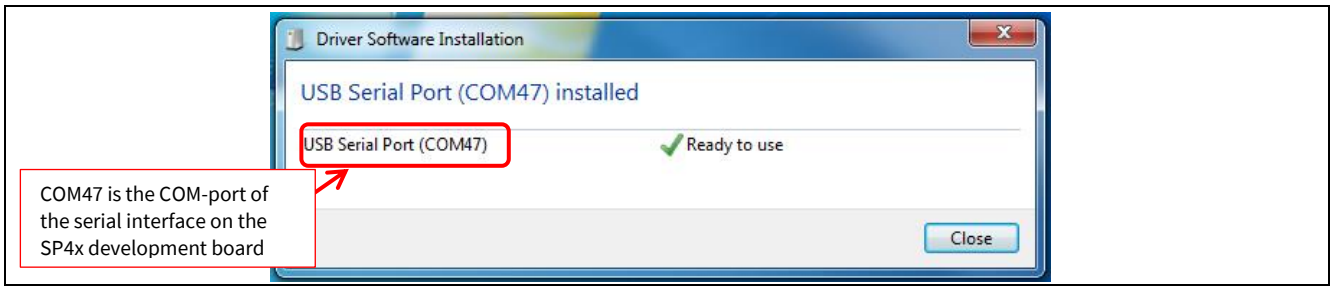

Figure 21 Installation window of the FTDI drivers

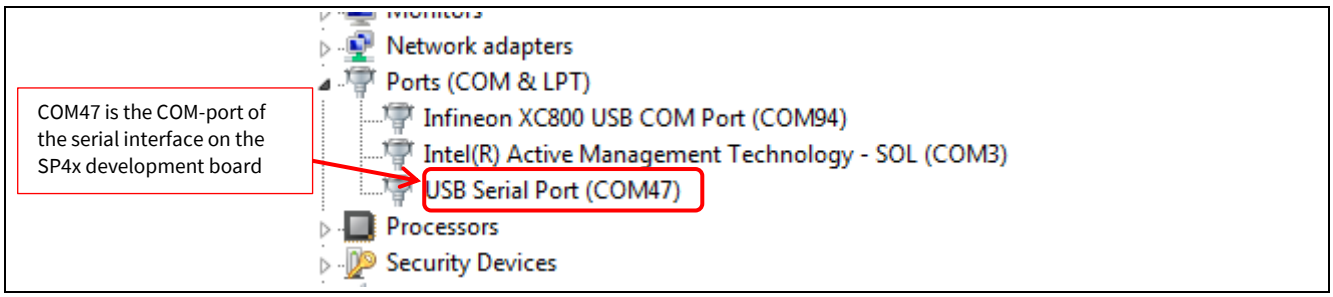

Figure 22 Locating the virtual COM-port number in windows' device manager

After the correct COM-port was evaluated, any terminal window (such as HyperTerminal or similar) may be used in order to connect to the port and send/receive data. In this example a freeware tool called "HTerm" was used to receive data when using the example 3 (SensorMasurements). First thing to be done is applying the correct settings, meaning specifying the correct COM-port, Baudrate, databit-, stopbit- and parity-settings, as shown in the next figure.

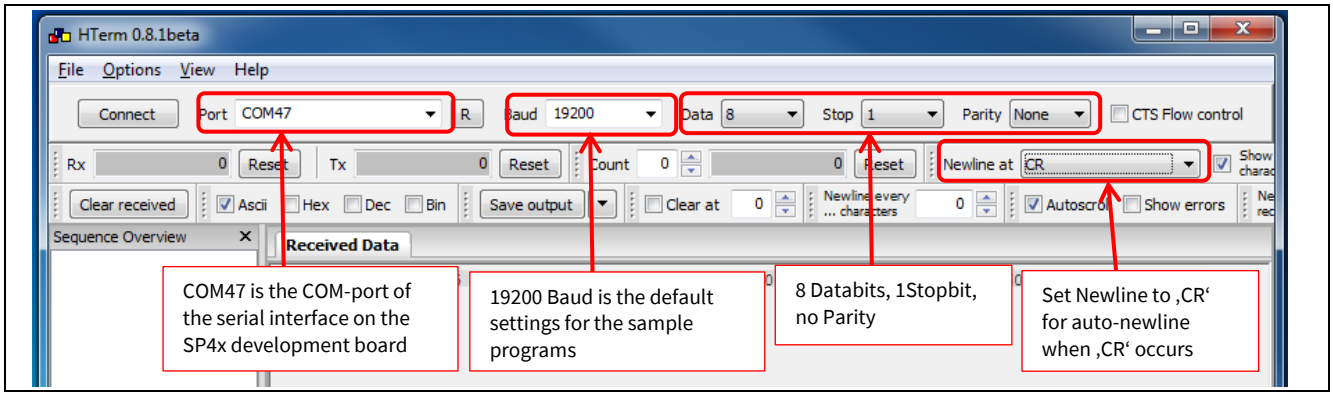

Figure 23 Configuration of the terminal window

After clicking "Connect" and loading the sample program to the SP4x, the output may be observed inside the window as shown in the next figure.

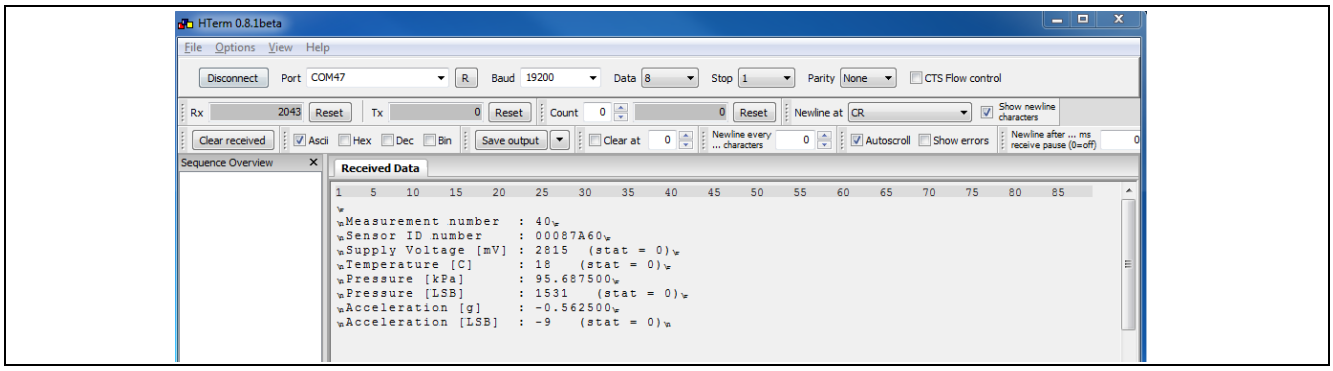

Figure 24 Output of example 3 (SensorMeasurements) in human readable format

If the output is configured as 'comma separated format' (via a pre-compiler option in the main.c file), the output will look as shown in the next figure.

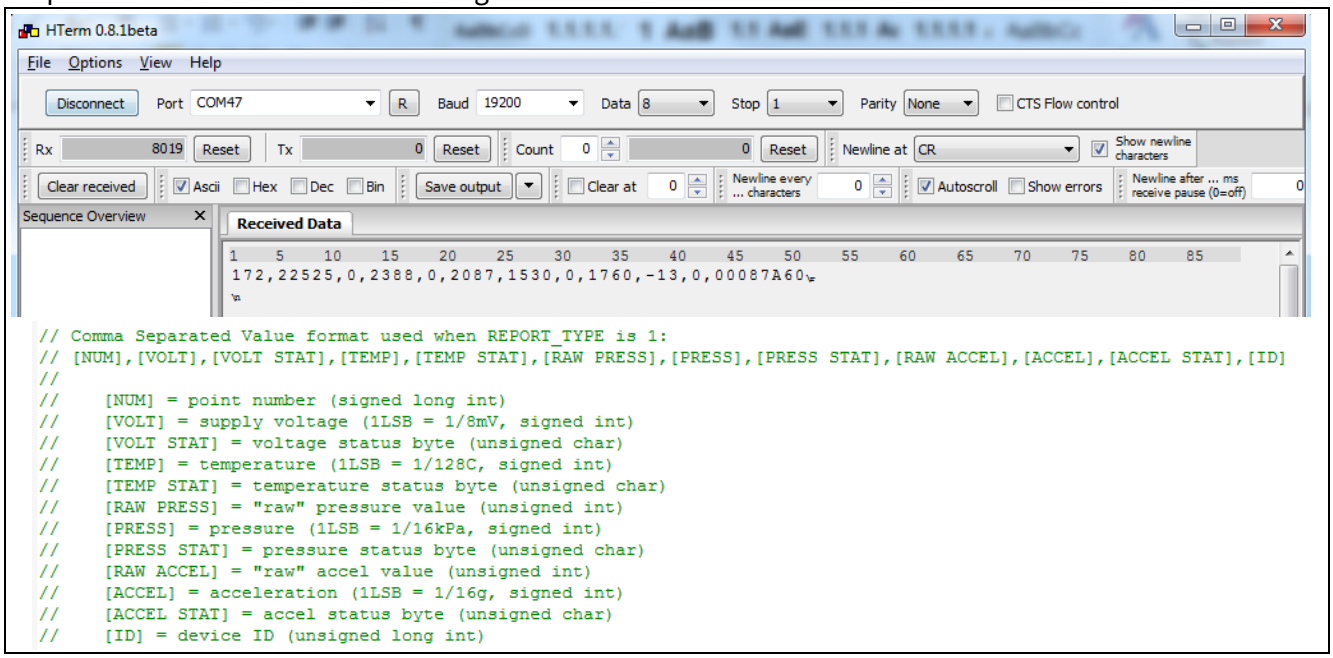

Figure 25 Output of example 3 (SensorMeasurements) in comma separated format

5.6 Example 3b: AccOffsetCompensation

This program shall demonstrate the usage of the Lib_Comp_Auto_Acc_Offset() function. The sample program does 160 cycles of measurement, where the first 50 measurements' RAW-values are manipulated in order to simulate an offset drift. The last 110 measurements are not manipulated. Due to the Auto-Acceleration-Offset-Removal it can be observed via Uart, whenever the library function updated the offset information. Furthermore the result of the compensation will show that the offset disappeared after the update of the offset information. The software flow is given in the next figure.

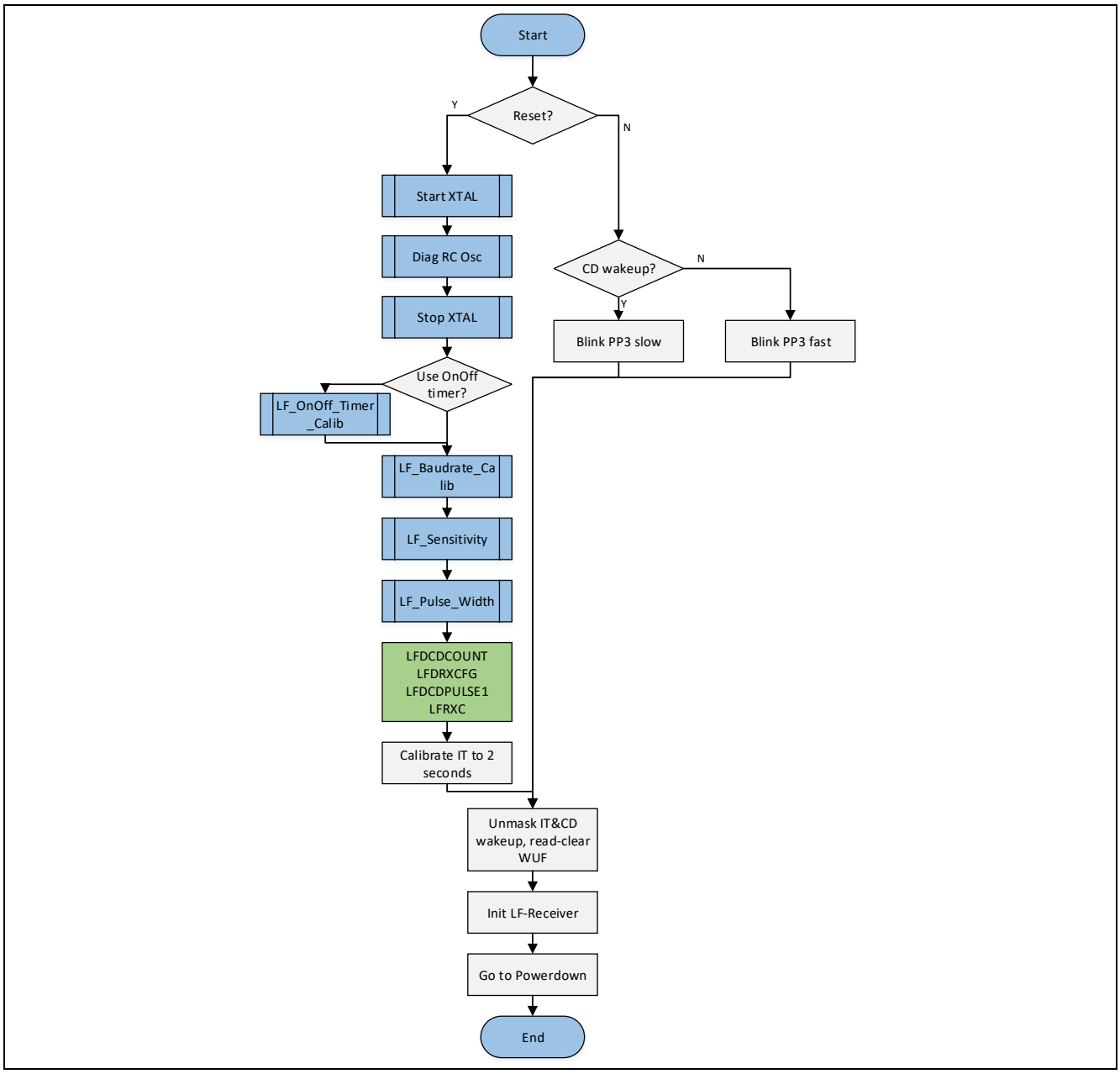

Figure 26 Flowchart of Example 3b (AccOffsetCompensation)

After setting up the project, loading it to the device and connecting to the serial port the output may be observed as shown in the following figures.

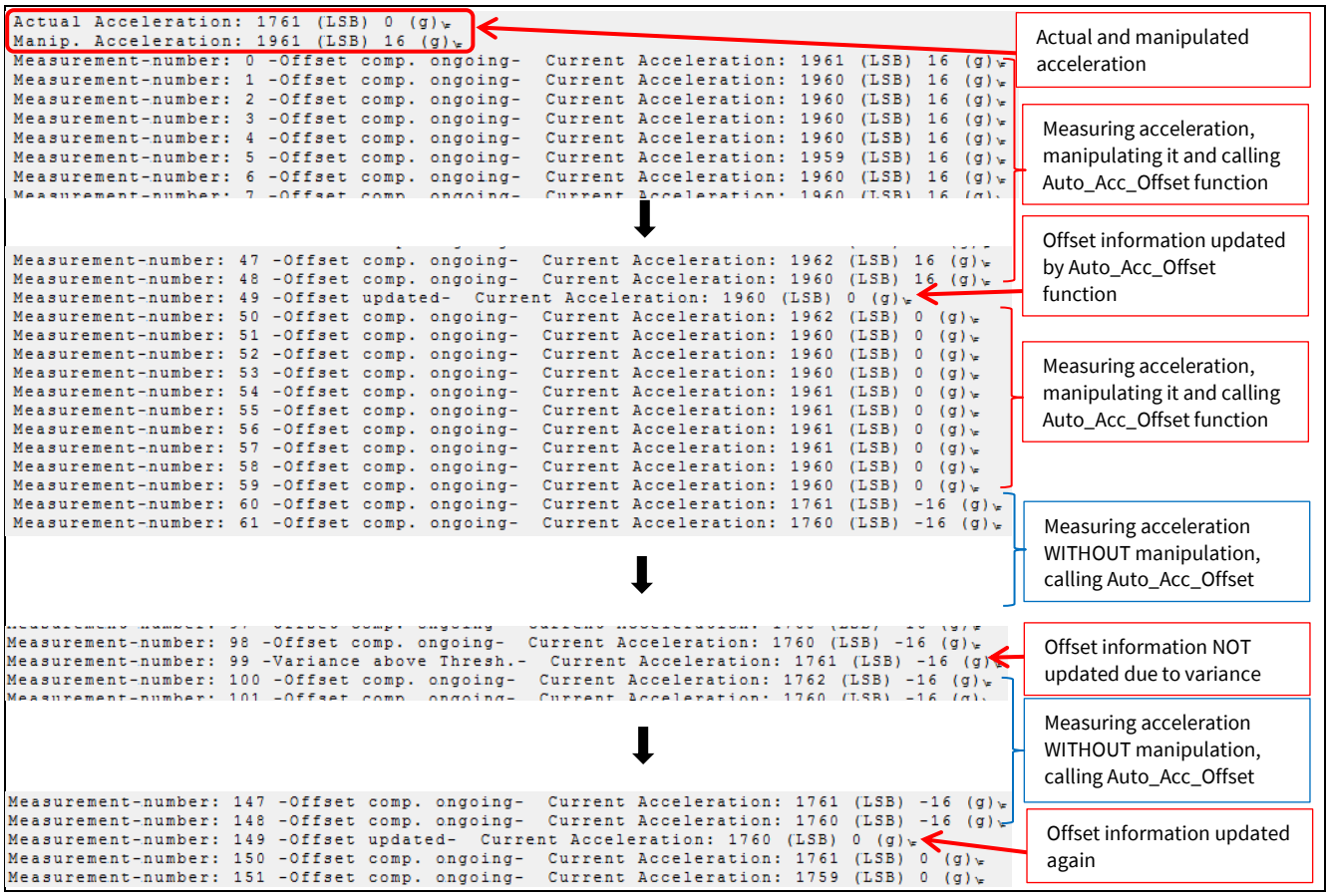

Figure 27 Observation of the output of example 3b while using Auto_Acc_Offset

5.7 Example 4a: LFCWwithPulseFilter

The device will enter POWERDOWN state after LF receiver configuration is complete. An LF CW meeting the necessary pulse width shall cause an LF CW wakeup. An LF CW wakeup shall cause the LED on PP3 to flash once slow. An Interval Timer wakeup shall cause the LED on PP3 to flash once fast. After the wakeup is processed, the device shall re-enter POWERDOWN state.

To wakeup the SP4x via LF-carrier either the SMA-connector or a LF-coil can be used (the switch “”SW_LF” must be on the respective position). A simple setup could like as shown in the next figure.

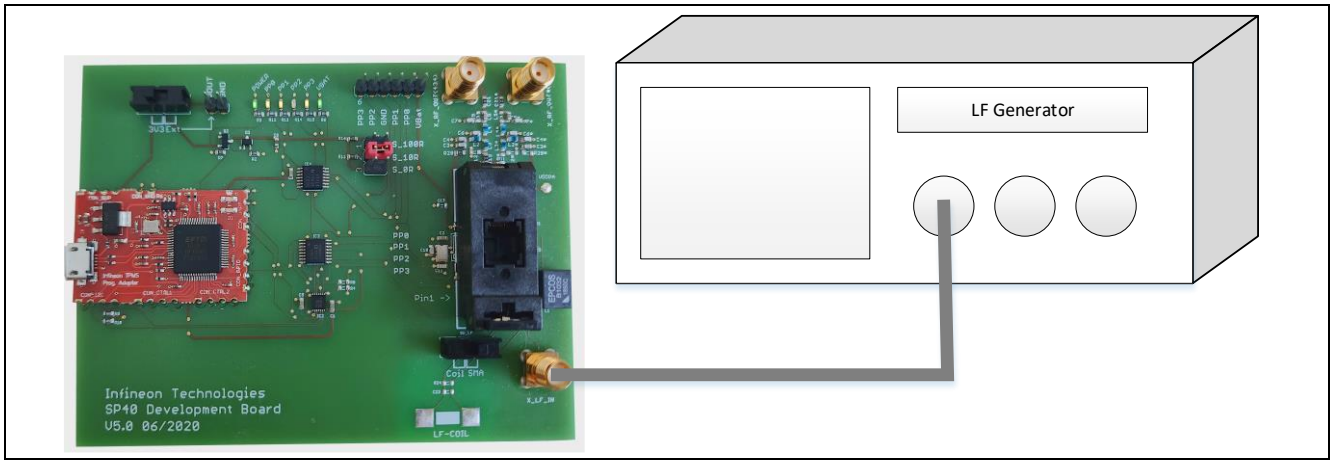

Figure 28 Basic setup for appliying LF-CW to the SP4x

The application flow is given in **Error! Reference source not found.**.

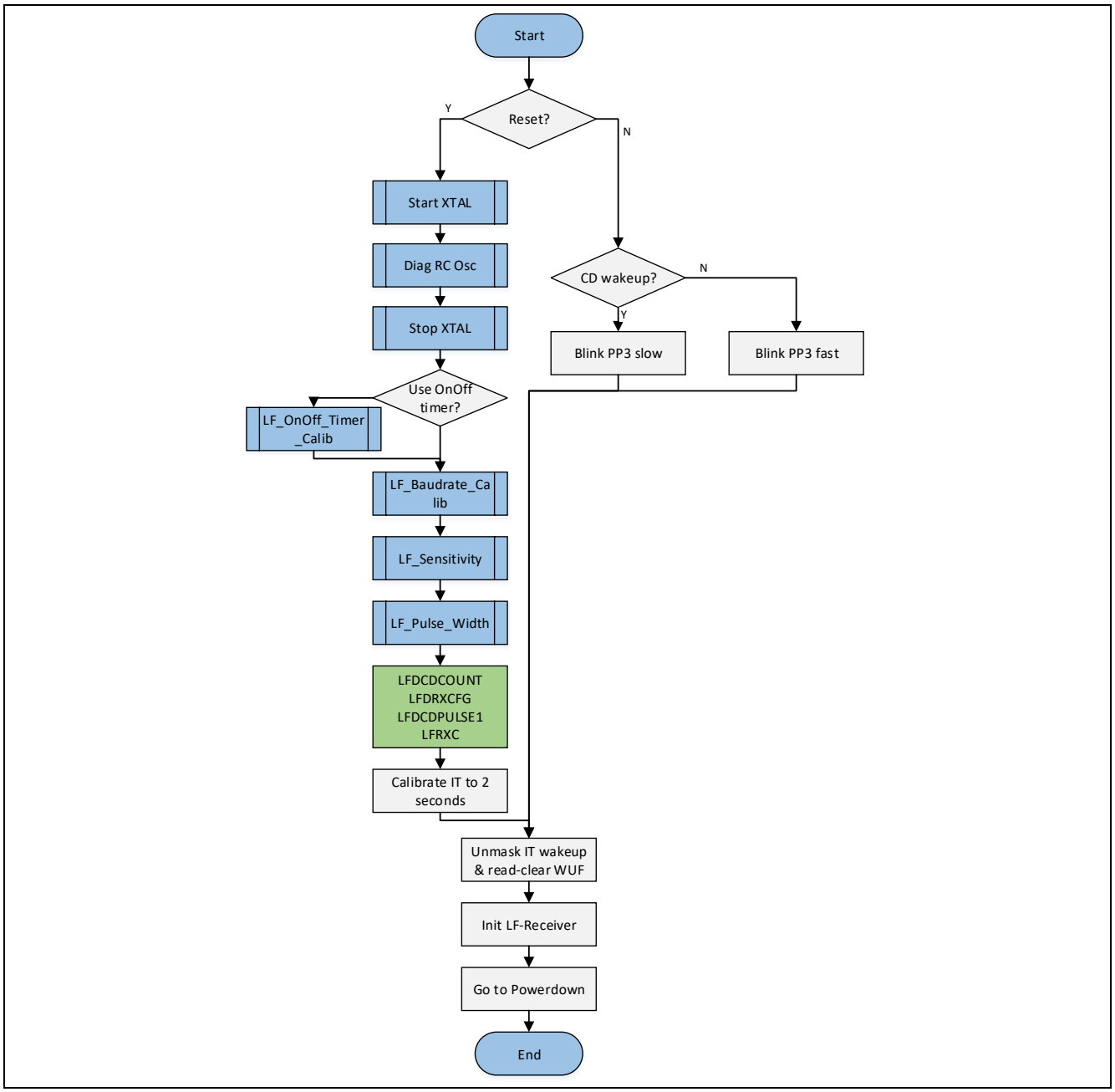

Figure 29 Flowchart of Example 4a (LFCWwithPulseFilter)

5.8 Example 4b: LFTelegramReception

As seen in the previous example, it is possible to wake up the device by LF Carrier Detection. Another common possibility to wakeup SP4x is via LF Telegram. This example is explaining how this is performed.

The purpose of this example is to enter in powerdown and to “listen” for LF Datagram (WU-pattern = 0x1234). When LF Datagram is detected, the datagram content is received and, depending on the payload, the LEDs connected to PP2 and PP3 will blink:

- After complete reception of one telegram: PP2 blinks according to the number of received bytes
- PP2 blinks fast if payload was ‘0xFF’
- PP2 links slow if payload was ‘0x1C’

- PP3 blinks fast otherwise

The **Error! Reference source not found.** shows how the telegram reception example is implemented.

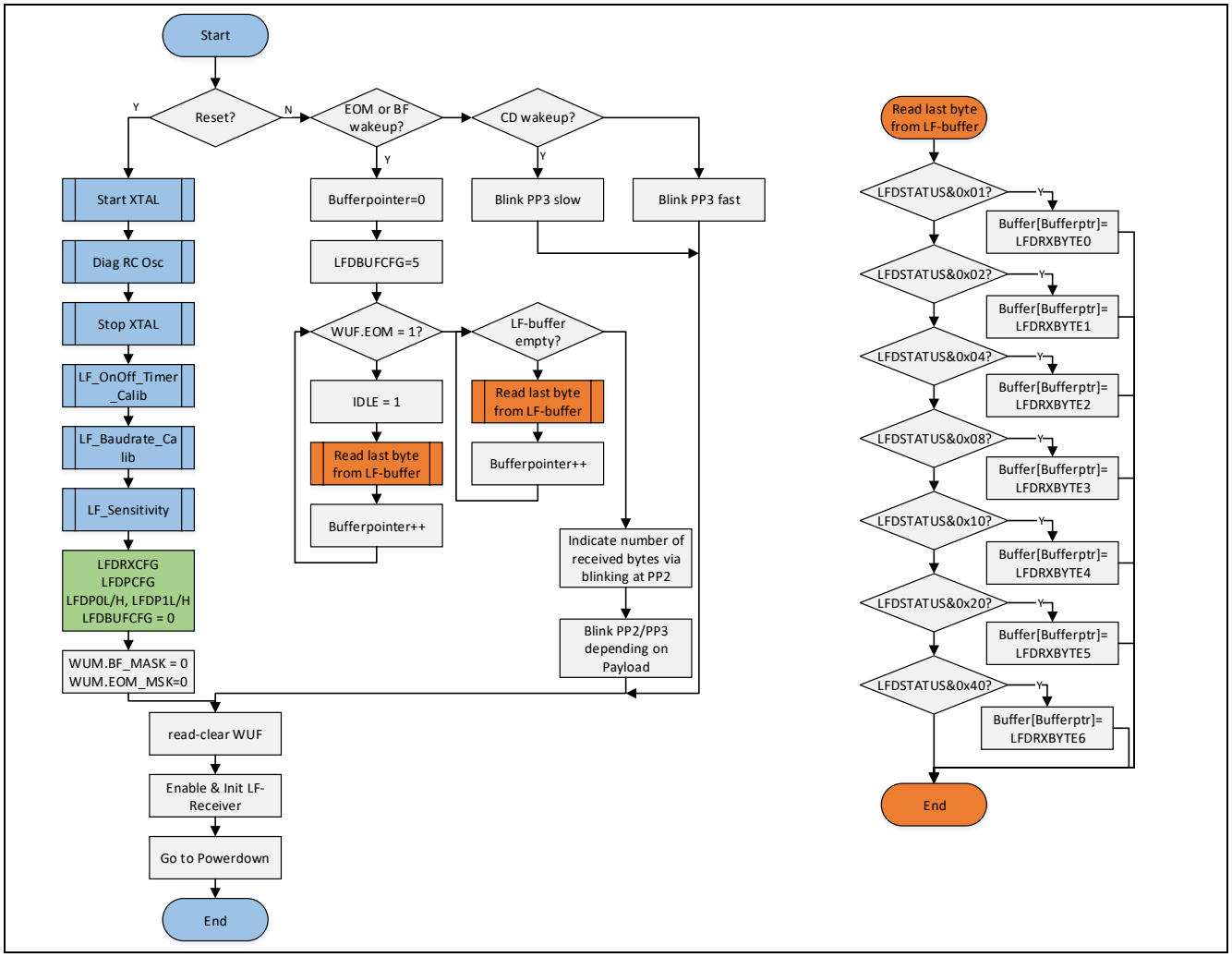

Figure 30 Flowchart of Example 4b (LFTelegramReception)

5.9 Example 5a: RfCW

Purpose of this project is to generate a RF-carrier-wave in order to enable output power measurements. The project may be configured to output either a 315MHz or a 434MHz carrier wave depending on a pre-compiler option shown in the next figure. The flow of the software is given in Figure 32.

```
33  //#define _434MHz
34  #define _315MHz
```

Figure 31 Pre-compiler option in example 5a to switch frequency

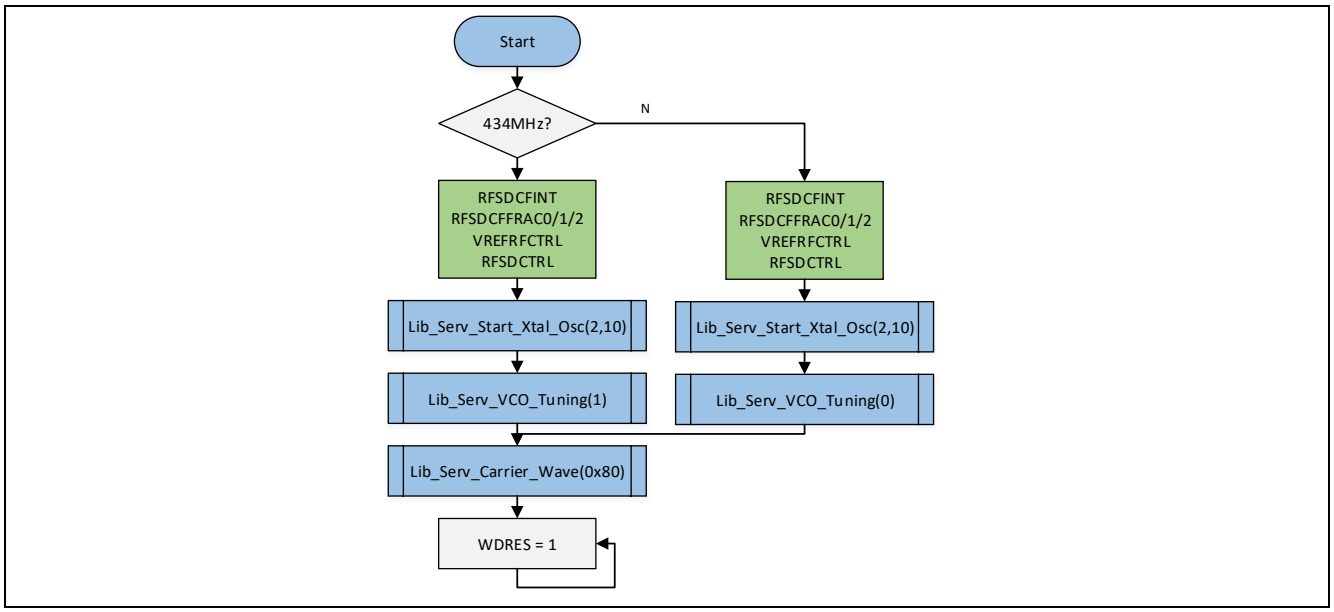

Figure 32 Flowchart of Example 5a (RfCW)

After loading the project to the SP4x and connecting a spectrum analyzer to the appropriate RF-output-SMA-connector, a peak can be observed at the target frequency, as for example in the following figure for 315MHz.

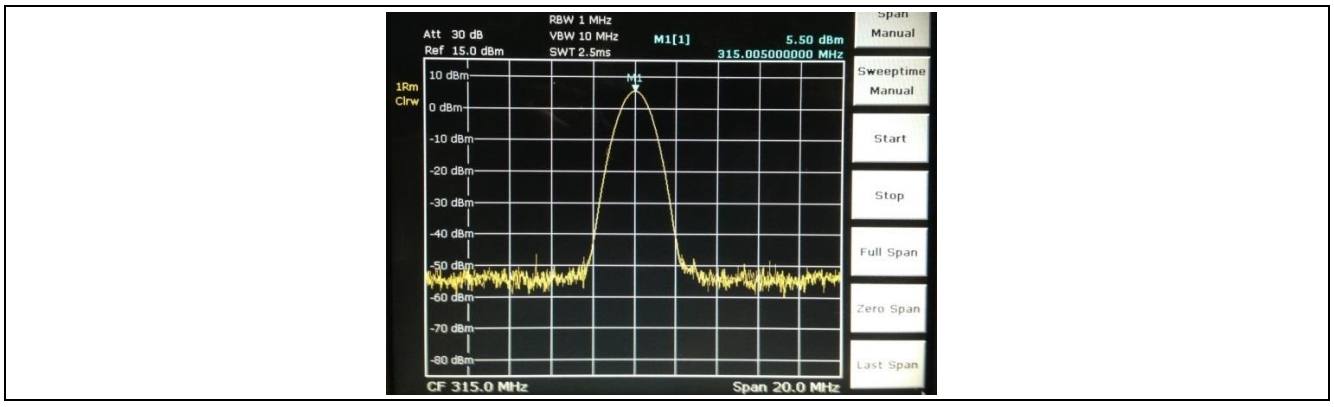

Figure 33 Measuring RfCW at 315MHz with a spectrum analyzer

Note: For having an idea of the output power, only one matching must be active on the board (refer to chapter 3.1). Furthermore the development board is not designed for perfect RF-performance (due to socket and layout), thus the output power will most likely not fully achieve the specification. For reliable RF-performance measurements another board (such as the datasheet reference board) should be used.

5.10 Example 5b: RFTelegramFSM

This project show how an RF-telegram is generated by using the TX state machine (FSM). In detail, the involved registers are shown and the complete configuration-flow according to the user manual is implemented for either FSK or ASK modulation (two subversions with *.1 or *.2 extension). The basic software flow is given in the next figure.

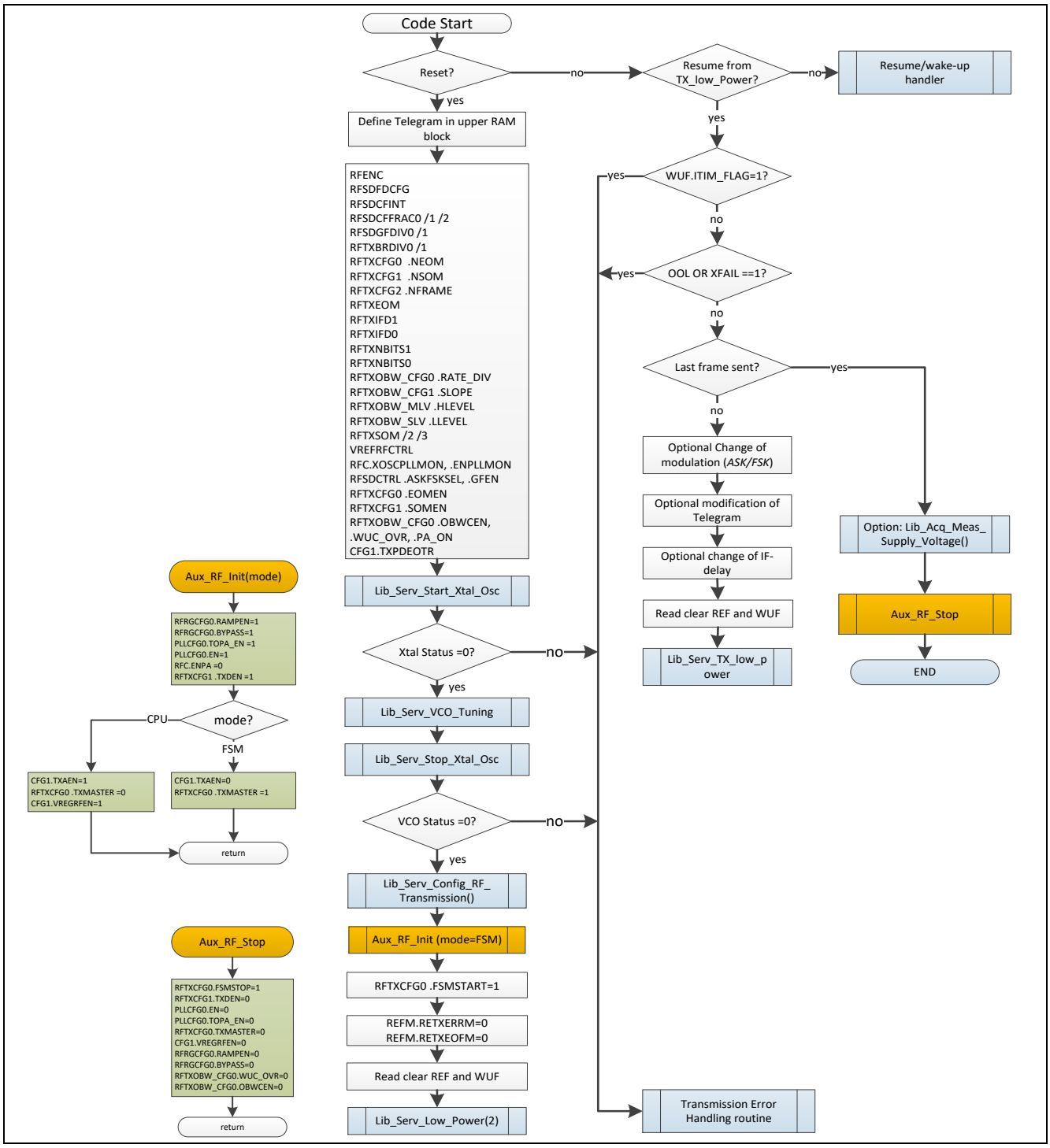

Figure 34 Flowchart of Example 5b (RFTelegramFSM)

As an addition, the LED connected to PP2 will blink once fast whenever the devices resumes from TX-Low-power state, indicating the transmission of one frame, where such a frame will be repeated ten times. After having transmitted all frames, the device will blink 4 times and wait for a watchdog reset afterwards.

The transmitted frames may either be observed at the correct RF-output-connector or by a feature called Data-to-Pin. For the seconds feature, the sample program must be modified a little bit by inserting two lines at a special position, as shown in the next figure.

```
128         REFM = ~REFM_RETXERRM_MASK;
129         REFM &= ~REFM_RETXEOFM_MASK;
130
131         P1DIR &= ~0x04;                 //Make PP2 as output
132         Lib_Serv_RF_Data_To_Pin(0);     //Enable RF-Endoder data at PP2
133
134         RFTXCFG0 |= RFTXCFG0_FSMSTART_MASK;                    //Start
135         store_WUF = WUF;                                       //alway
```

Add these two lines to enable RF-encoder data visualization at PP2

Figure 35 Modification of example 5b for observing RF-telegram at PP2

After these two lines where added and the sample program was loaded to the SP4x, RF-encoder data may be observed with an oscilloscope at PP2, as shown below.

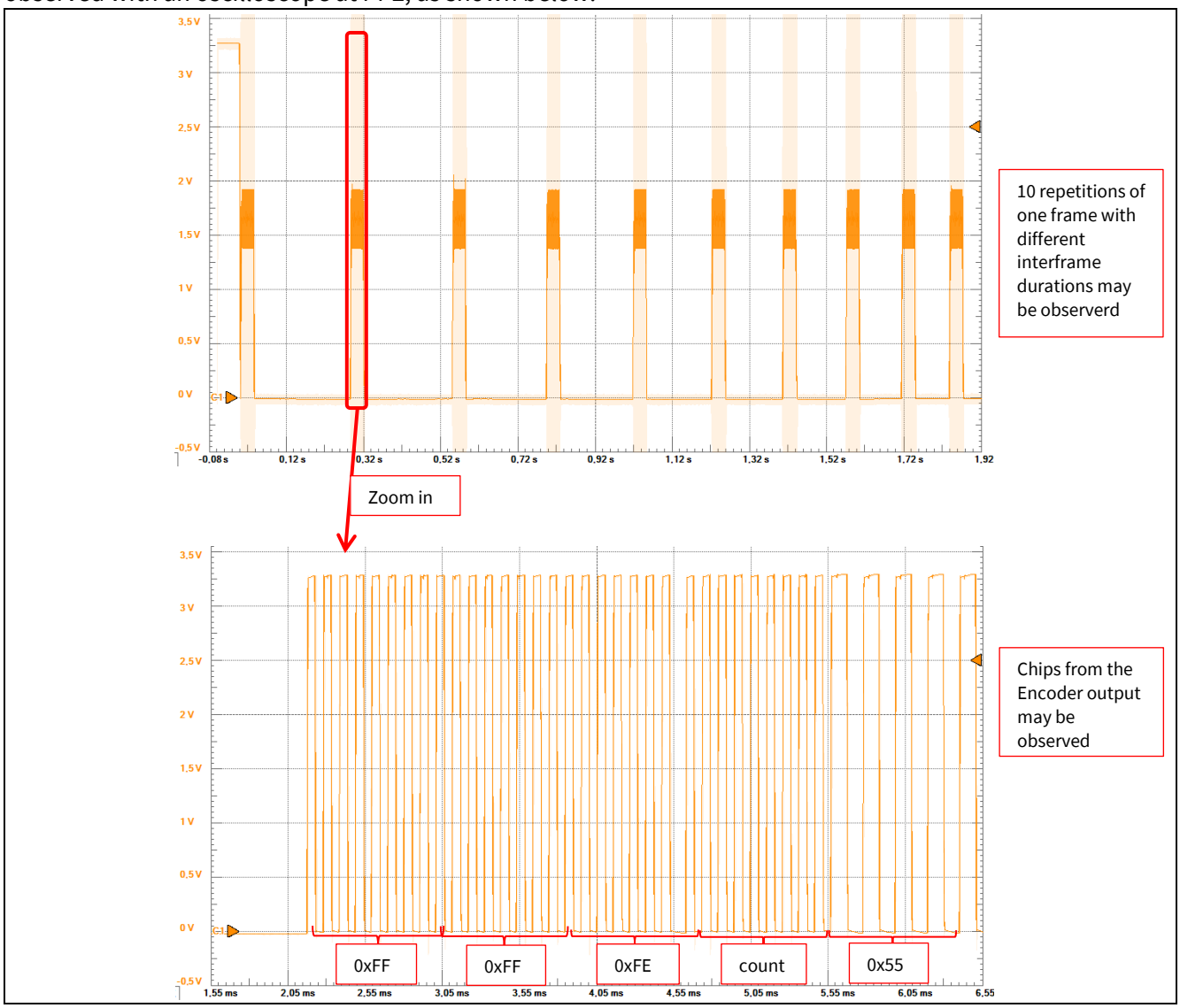

Figure 36 RF-Encoder output at PP2

5.11 Example 5c: RFTelegramCPU

This project show how an RF-telegram is generated in CPU mode, meaning without the usage of the TX state machine (FSM). In detail, the involved registers are shown and the complete configuration-flow according to the user manual is implemented for either FSK or ASK modulation (two subversions with *.1 or *.2 extension). The basic software flow is given in the next figure.

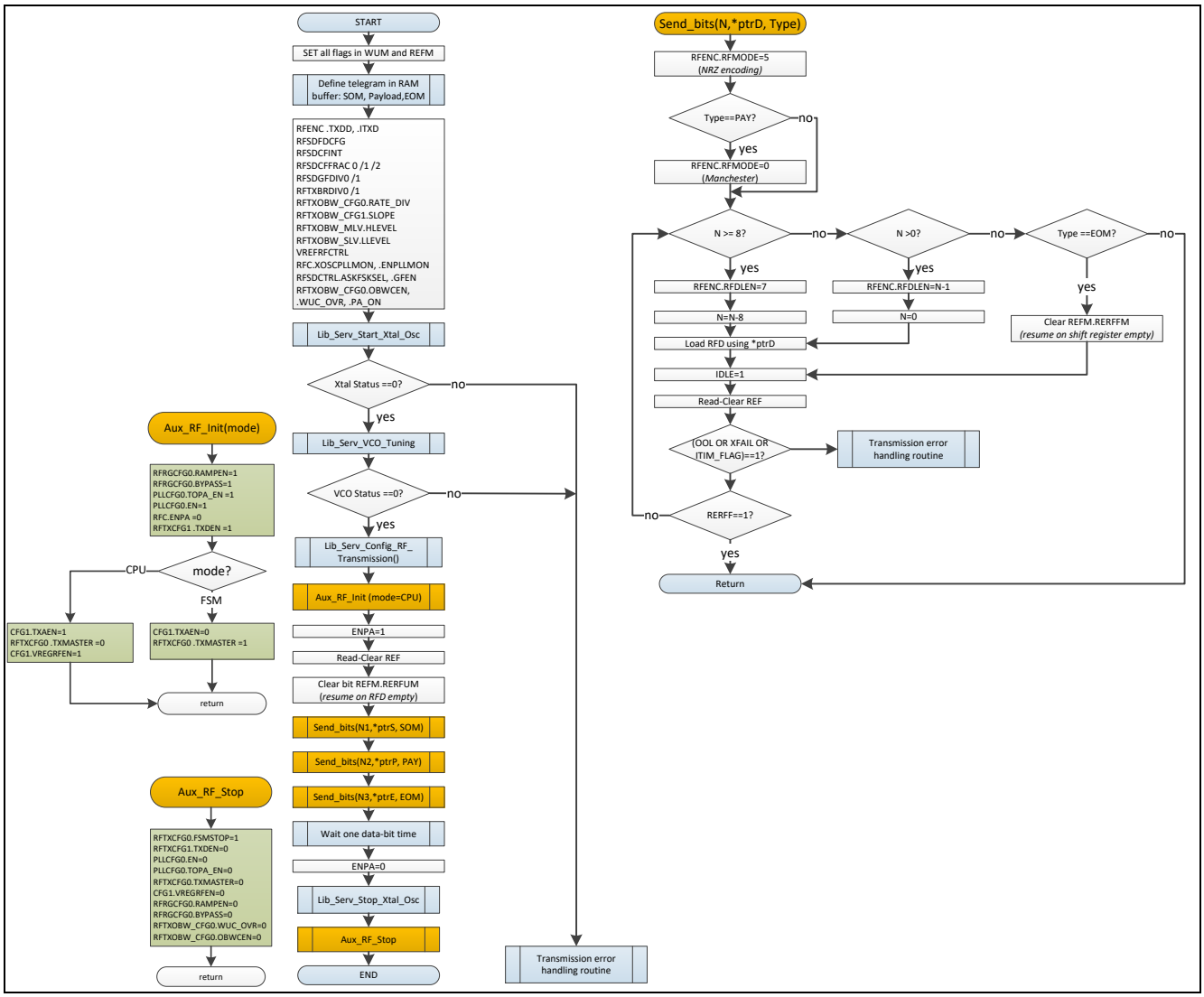

Figure 37 Flowchart of Example 5c (RFTelegramCPU)

Similar to the previous sample program, this program will send an RF-frame, but without repetitions this time. One frame will be sent out, afterwards the device will go to Powerdown. After wakeup, which occurs each second, the same frame is sent again and so on.

For observing RF-encoder output at PP2, a small modification is again necessary, as shown in below.

```
98    RFC |= RFC_ENPA_MASK;
99
100   P1DIR &= ~0x04;                    //Make PP2 to outp
101   Lib_Serv_RF_Data_To_Pin(0); //Enable RF-Encoder output at PP2
102
103   store REF = REF;
```

Add these two lines to enable RF-encoder data visualization at PP2

Figure 38 Modifying example 5c for observation of RF-encoder output at PP2

The output that can be observed is display in the next figure.

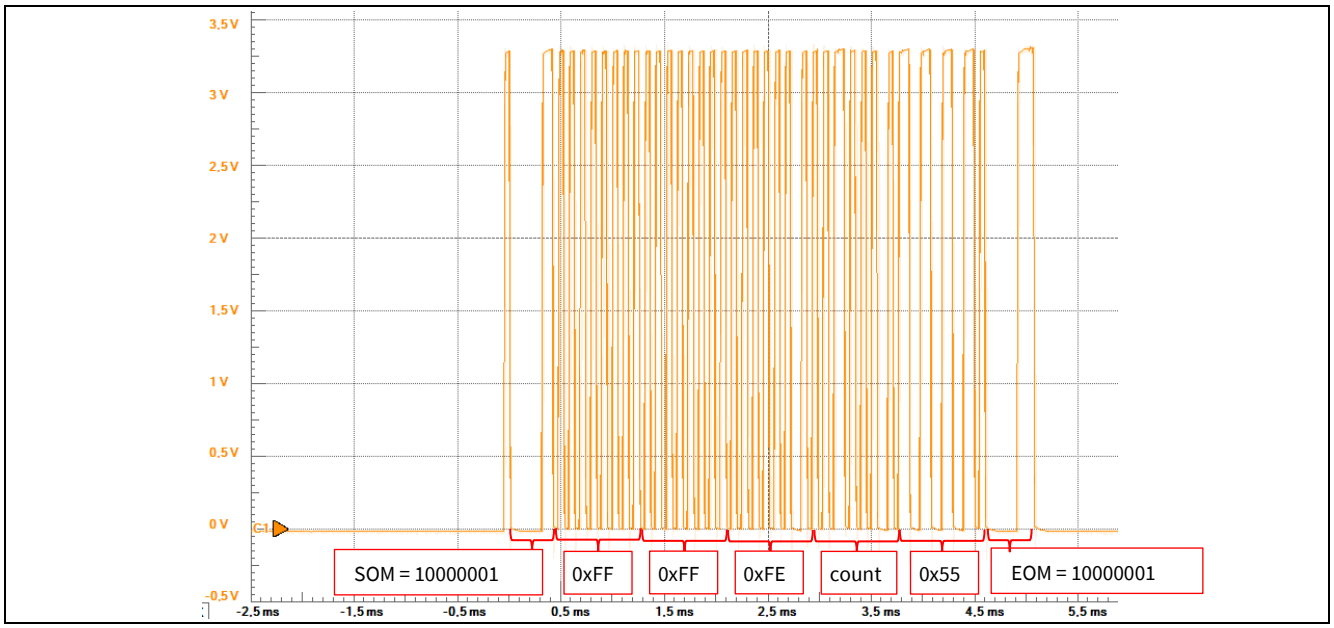

Figure 39 RF-encoder output of example 5c

5.12 Example 6: IFXApsDemo

This program shall demonstrate a basic APS application using all available IFX APS functions. The device is in powerdown and waits until an acceleration threshold is reached. Afterwards sampling is started using Lib_APS_Sample(), followed by the processing also done by library functions. The most important parameters may be defined in the application in order to adapt the application to different scenarios. During processing, robustness criteria 1-3 can optionally be enabled and then restart the application in case of poor data. After processing, either extrapolation- or time&phase-method may be used in order to transmit APS information to a receiver.

This example can be configured in many ways in order to adopt it flexibly to many different scenarios. All necessary parameters may be changed inside the file “Application.h” and they are grouped in two categories:

- Mode Selection:
    - Information mode: Either Extrapolation or Time-and-Phase
    - RC mode: Activate/Deactivate checking of robustness criteria
    - K1 adapt mode: The software is also capable of correction of the K1 parameter (according to wheel size) in order to support many different wheel sizes without the need of re-programming the APS application.
    - Mounting direction: positive or negative, must be according to the mounting of the module
    - DFT mode: use either software cordic (fast) or hardware cordic (flexible)
    - APS mode: activate fine frequency calculation of Lib_APS_Sin_Parameter_Estimate (needed for extrapolation)
- Adaptable Constants:
    - N0: Number of samples to be taken per period
    - DFT_Min/DFT_Max: limits for the hardware cordic
    - K1: Parameter which is proportional to the wheel size
    - LOG2_OSF: Defines oversampling factor for acceleration sampling
    - ABSDIF_TH: limit for robustness criteria 1 (AbsDif)

- MAXMIN_TH: limit for robustness criteria 2 (MaxMin)
- DFTPTA_TH: limit for robustness criteria 3 (DftPta)
- ACC_TH: threshold of acceleration in g for starting APS
- PHASE_TX: applies only in extrapolation mode and defines the desired angle where to transmit

Furthermore there is the possibility to blink the LED connected to PP2 or PP3 whenever the APS if finished. In extrapolation method this would mean after reaching the desired angle, thus the LED should ideally blink always at the same angle. The option to turn on/off the blinking of the LED is inside the file “Application.h” and is shown in the next figure.

```
152 BUILD OPTIONS
153 Set to (TRUE) or (FALSE) as desired; not all combinations are valid!
154 (Conflicts are detected and result in an error at compile time)
155 */
156 //#define PP2_OUT (FALSE)
157 #define PP2_OUT (TRUE)
158
159 #define PP3_OUT (FALSE)
160 //#define PP3_OUT (TRUE)
161
162 #endif
```

Figure 40 Configuration for turning LED blinking on/off in example 6

The flow of the application is given in the next figure.

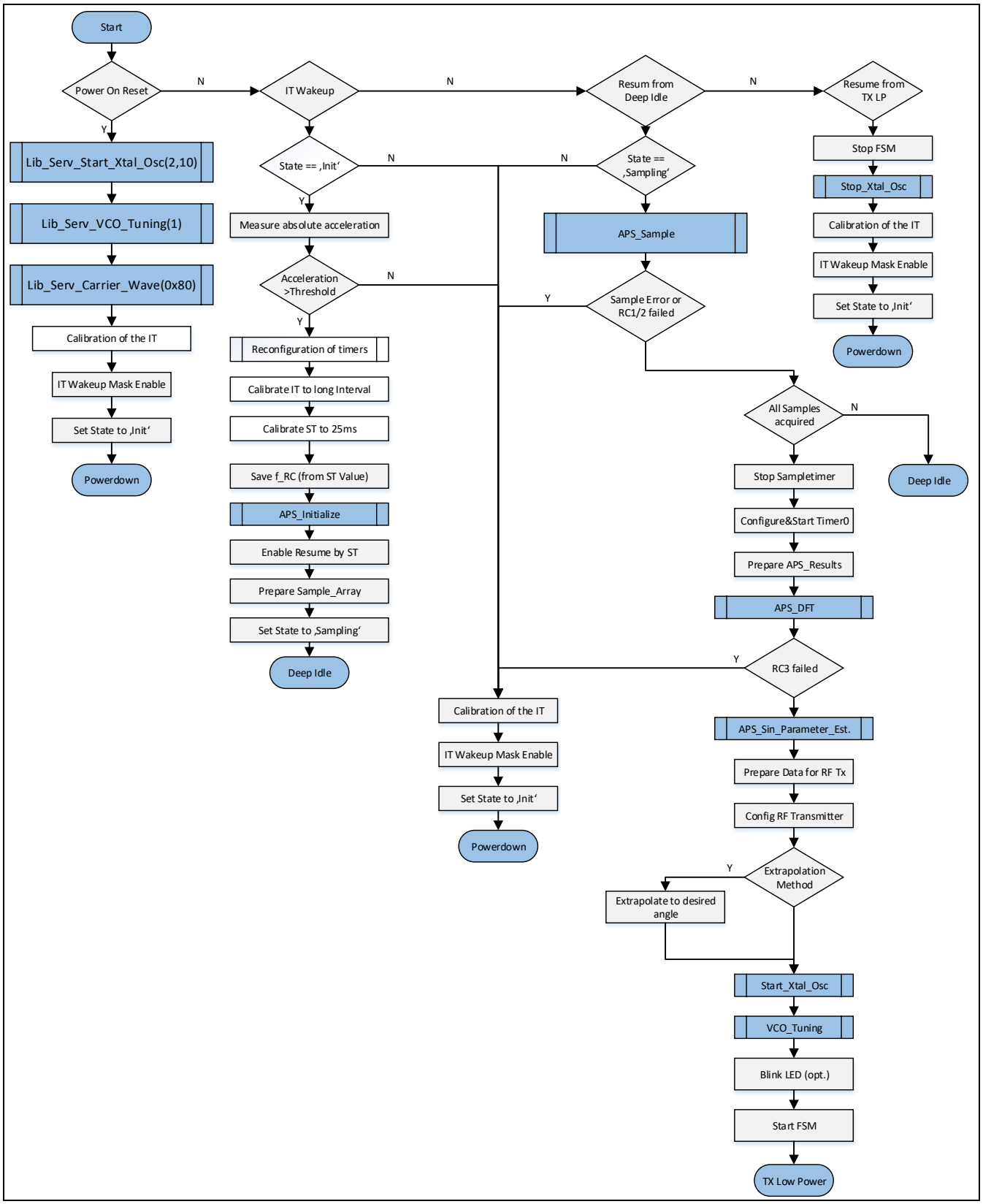

Figure 41 Flowchart of Example 6 (IFXApsDemo)

Revision history

Document version	Date of release	Description of changes
V 2.0	2021-01-20	Update for new development-board V5

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2021-01-20

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

IMPORTANT NOTICE

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.