

Lite DCDC System Basis Chip Shield with TLE9471-3ES for Arduino

Lite SBC Family

About this document

Scope and purpose

This document describes hardware features and the usage of the reference code examples of the Infineon Lite DCDC System Basis Chip Shield with TLE9471-3ES for Arduino.

Intended audience

This document is intended for software and hardware engineers integrating an Infineon Lite SBC Family device into their application but also for makers and hobbyists.

Schematic and Layout of the shield can be found on Github:

<https://github.com/Infineon/SBC-for-Arduino>

Table of contents

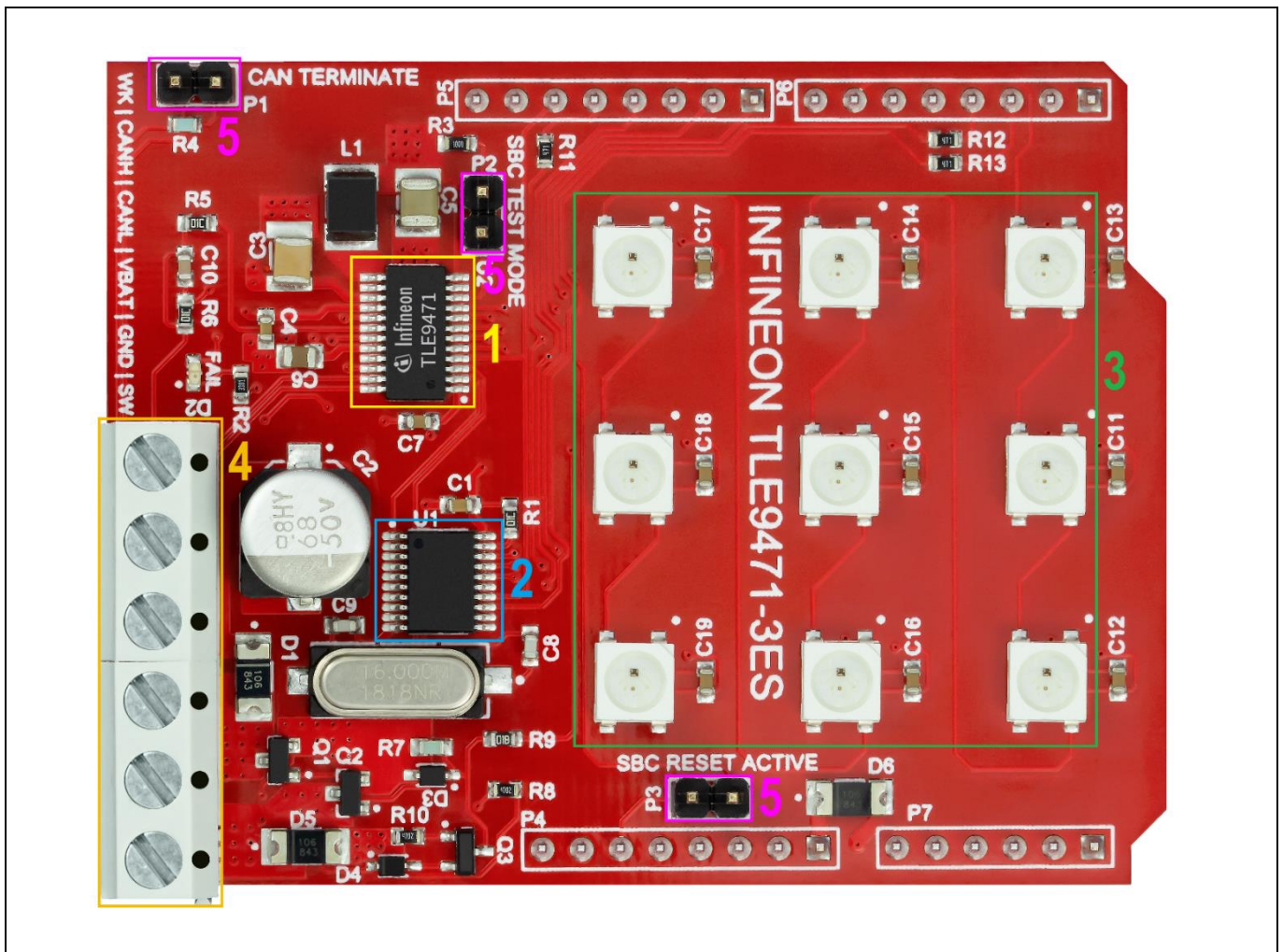
About this document	1
Table of contents	1
1 Board overview	2
1.1 TLE9471	3
1.2 MCP2515	3
1.3 LEDs	3
1.4 Terminals	3
1.5 Jumper configuration	4
2 Code examples	5
2.1 General example	5
2.2 CAN-PN example	6
3 Lite SBC library	7
4 Additional information	11
Revision history	12

Board overview

1 Board overview

The Lite DCDC System Basis Chip Shield with TLE9471-3ES for Arduino (in the following document just called “TLE9471-3ES shield”) features:

- Infineon TLE9471-3ES SBC (1)
- Microchip MCP2515 CAN-SPI controller (2)
- 9 pcs WS2812B LEDs (3)
- 6 pin screw terminal (4)
- 3 configuration jumpers (5)



Board overview

1.1 TLE9471

The shield features a TLE9471-3ES Infineon SBC with following functions:

- SPI interface (SCK: DIO-13, MISO: DIO-12, MOSI: DIO-11, CSN: DIO-8)
- System interrupt on INT1 (DIO-3) for signalization of e.g. level changes on wake-input
- System reset to Arduino reset pin – can be (dis)connected via jumper if not used
- Integrated charge-pump driving the reverse-polarity protected high-side power-mosfets for switching external loads up to 2.5 Amps
- Integrated CAN transceiver connected to MCP2515 CAN protocol handler with switchable 120 Ω CAN termination
- Powerful 5 V/500 mA buck converter driving the MCP2515 and the 9 pcs WS2812B LEDs
- Wake input with external pull-down for wake-up and high-level input voltage sensing
- Fail-output connected to LED D2 for status signalization
- SBC test mode available via jumper connection (no WD trigger via SPI needed to keep the SBC alive)

See more information related to all the features of the TLE9471-3ES on following web page:

<https://www.infineon.com/cms/en/product/automotive-system-ic/system-basis-chips-sbc/lite-sbc-family>

1.2 MCP2515

A MCP2515 CAN-SPI protocol handler IC is used to communicate to an external CAN bus via the CAN transceiver integrated in the TLE9471.

- SPI interface (SCK: DIO-13, MISO: DIO-12, MOSI: DIO-11, CSN: DIO-9)
- System interrupt on INT0 (DIO-2) for signalization of e.g. message-receive events
- CANTX / CANRX connected to TLE9471

1.3 LEDs

The board contains 9 pcs WS2812B LED's which are supplied by the powerful integrated buck converter of the TLE9471. The 9 LEDs are connected in series. The first LED is connected to DIO-6.

1.4 Terminals

The screw terminal contains 6 connections:

- **WK:**
High voltage wake input. A 10 k Ω pull-down is populated on the board. A signal between 4 V and VBAT voltage level can be applied here to either wake-up the TLE9471 from sleep-mode or alternatively as high-voltage input which can be read out via SPI interface.
- **CANH:**
CAN-HIGH terminal to external CAN bus
- **CANL:**
CAN-LOW terminal to external CAN bus
- **VBAT:**
Power supply input (9-12 V)
- **GND:**
Main ground connection

Board overview

- **SW:**
Output of the high-side mosfets which are driven by the integrated charge-pump of the TLE9471. An external load up to 2.5 A constant current can be connected here. Also a passive freewheeling diode is assembled on the board thus inductive loads can be switched off without any risk. The charge pump can be activated via SPI.

1.5 Jumper configuration

- **CAN TERMINATE:**
A CAN bus must be properly terminated to ensure communication. This jumper inserts a 120 Ω resistor differentially between CANH and CANL. If CAN bus is already externally terminated and the TLE9471-3ES shield is connected via a short stub to the bus, the jumper can be left open. If there is only one other node besides the TLE9471-3ES shield in the CAN network which is terminated, the jumper should be closed as a CAN network should be always terminated in sum with two times 120 Ω = 60 Ω .
- **SBC TEST MODE:**
This jumper enables the test mode for the TLE9471-3ES which means that the internal watchdog must not be triggered via SPI. In case, the jumper is not set and the watchdog is not triggered regularly depending on the configuration, the device will transit into fail-safe mode which will be indicated by the fail LED (D2). For normal use with Arduino it is recommended to keep the jumper closed, as Arduino has an integrated bootloader which needs quite long until the Arduino sketch itself is loaded (> 1 sec). As the SBC expects the first watchdog trigger at least after 200 ms after startup, a proper startup is not possible. If the shield is used with other microcontrollers which ensure triggering of the WD in the timeframe of 200 ms after startup, the jumper can be left open. See datasheet of TLE9471-3ES for more details.
- **SBC RESET ACTIVE:**
The SBC features a reset output pin to ensure a safe startup of the microcontroller. The reset is toggled low for approx. 2 ms after the internal buck regulator voltage is stable (as the microcontroller is usually supplied by the SBC). In case the jumper is set, the SBC will have control over the reset of the Arduino. This means after a power-on, the reset is toggled and the firmware is started. But also in case, the SBC is in sleep-mode, the reset will be kept low all the time to save current. For normal application use cases, the jumper should be closed. During debugging of the code this could lead to the problem, that the Arduino-IDE cannot flash the software to the Arduino. Therefore the jumper should be open.

Code examples

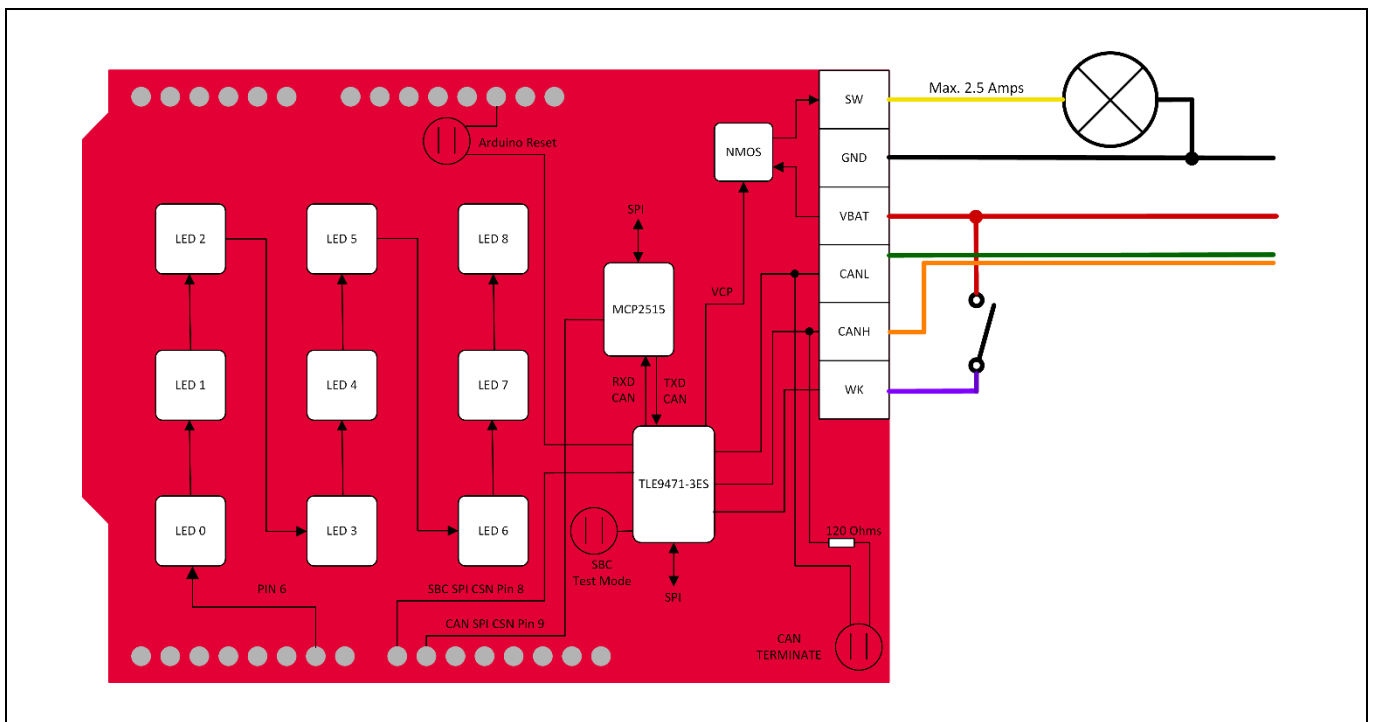
2 Code examples

There are code examples available for the TLE9471-3ES shield on Github.

See <https://github.com/Infineon/SBC-for-Arduino>

2.1 General example

This is an example to demonstrate how all the features like CAN communication, LEDs, charge-pump driven high-side mosfet, wake inputs are used in a meaningful manner together.



All CAN communication is done on 500 kBaud/s bus speed. After start-up, following features can be used:

- **Feature 1:** Control a single LED over CAN bus via sending following CAN frame into the shield:
 - CAN-ID: 0x100 (not extended)
 - MSG[0] = LED index (0-8)
 - MSG[1] = Red value (0-255)
 - MSG[2] = Green value (0-255)
 - MSG[3] = Blue value (0-255)
- **Feature 2:** Control all LEDs at once over CAN bus via sending following CAN frame into the shield:
 - CAN-ID: 0x101 (not extended)
 - MSG[0] = Red value (0-255)
 - MSG[1] = Green value (0-255)
 - MSG[2] = Blue value (0-255)

Code examples

- **Feature 3:** Control external high-side switch and FO-LED (D2) over CAN bus via sending following CAN frame into the shield:
 - CAN-ID: 0x102 (not extended)
 - MSG[0] = High-side switch / Charge-Pump on/off (0=off, 1=on)
 - MSG[1] = FO-LED (D2) on/off (0=off, 1=on)
- **Feature 4:** Red blinking of all LEDs by sending random CAN frames into the shield:
 - CAN-ID: any
 - MSG: any
- **Feature 5:** Shield will send out CAN frame in case of high-to-low or low-to-high event on the WK-input crossing approx. 3V threshold. Following CAN frame can be received by external CAN node:
 - CAN-ID: 0xAA
 - MSG[0]: 0xAA

2.2 CAN-PN example

This example is a demonstration how to use the CAN Partial-Networking feature. CAN Partial-Networking is used to wake-up selected automotive control units by just sending a “magic” wake-up frame (WUF) over the CAN bus. The SBC is in sleep-mode for current saving purposes and also Arduino is shut down by keeping the reset low at any time. In case, the “magic” CAN message is send over the bus, the SBC wakes up and releases the reset for the Arduino to start.

To use the example, please connect an external CAN communication node to the CANH/CANL of the shield. Please ensure that the grounds of the shield and the external CAN communication node are connected together and that the CAN bus is terminated correctly.

The SBC CAN-PN module is sensitive to following “magic-frame”:

- **Speed:** 250 kBaud/s
- **ID:** 0x2D (extended ID feature used)
- **Message:** (Byte 7) 0xDE 0xAD 0xC0 0xDE 0xC0 0xDE 0xBA 0x5E (Byte 0)

After startup, the above mentioned CAN-PN configuration is written to the SBC and the LEDs will light up in green. The SBC is waiting now for a first CAN frame on the bus so that the internal CAN protocol handler can get in sync. This can be any CAN message on 250 kBaud/s bus speed except the “magic-frame” above. Please ensure that the “magic-frame” is not sent at the beginning (as long as SBC protocol handler is not in sync).

If the SBC protocol-handler is in sync, the code-example initiates directly the SBC sleep-mode and the LEDs will turn off.

If the “magic-frame” mentioned above is sent, the SBC wakes up from sleep-mode and releases the reset of the Arduino. Now, the LEDs light up in green once again and the SBC protocol handler can get in sync again.

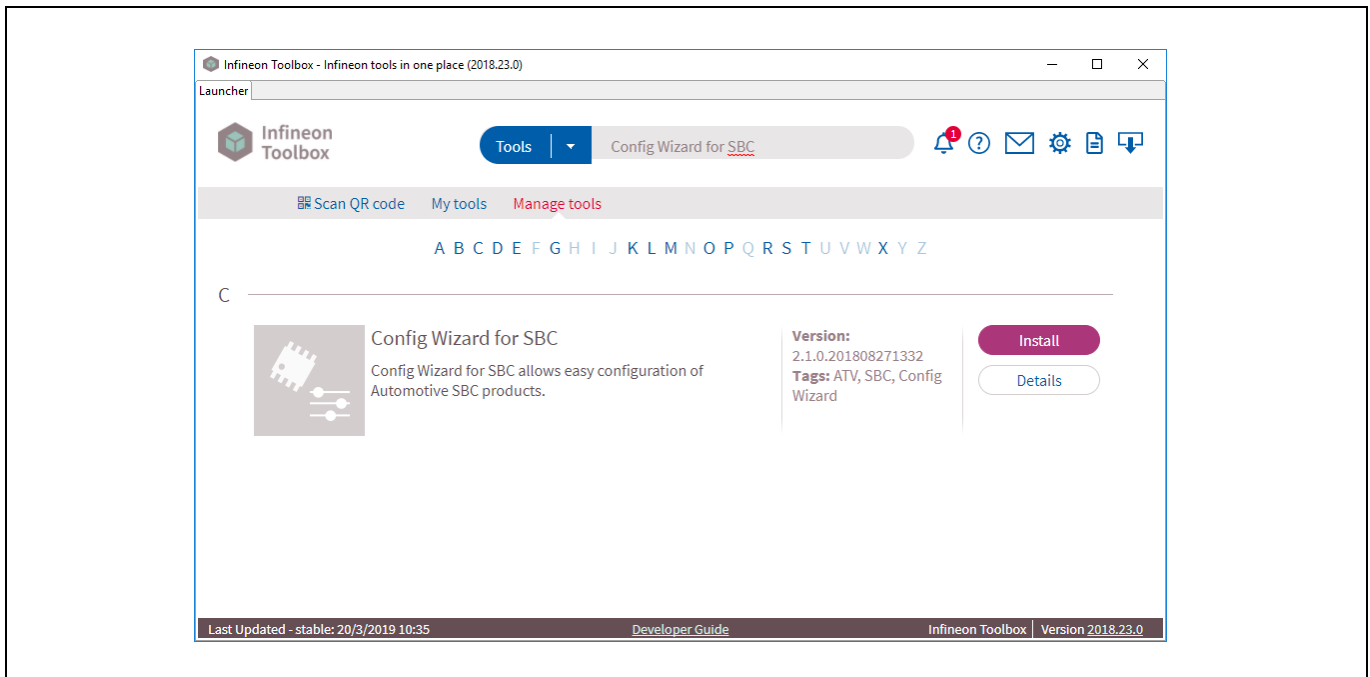
Please have in mind that the CAN-PN feature doesn’t use the MCP2515 on the board. So the CAN frames on the board must be acknowledged by an external node. The CAN frame is decoded by an integrated protocol handler which is only usable for the CAN-PN feature and not for standard CAN communication.

Note: Additional Info: In case, the “magic-frame” is sent before the transition to SBC sleep-mode, the CAN-PN configuration is not valid anymore. Thus, the transition to SBC sleep-mode will fail. See datasheet for more information.

3 Lite SBC library

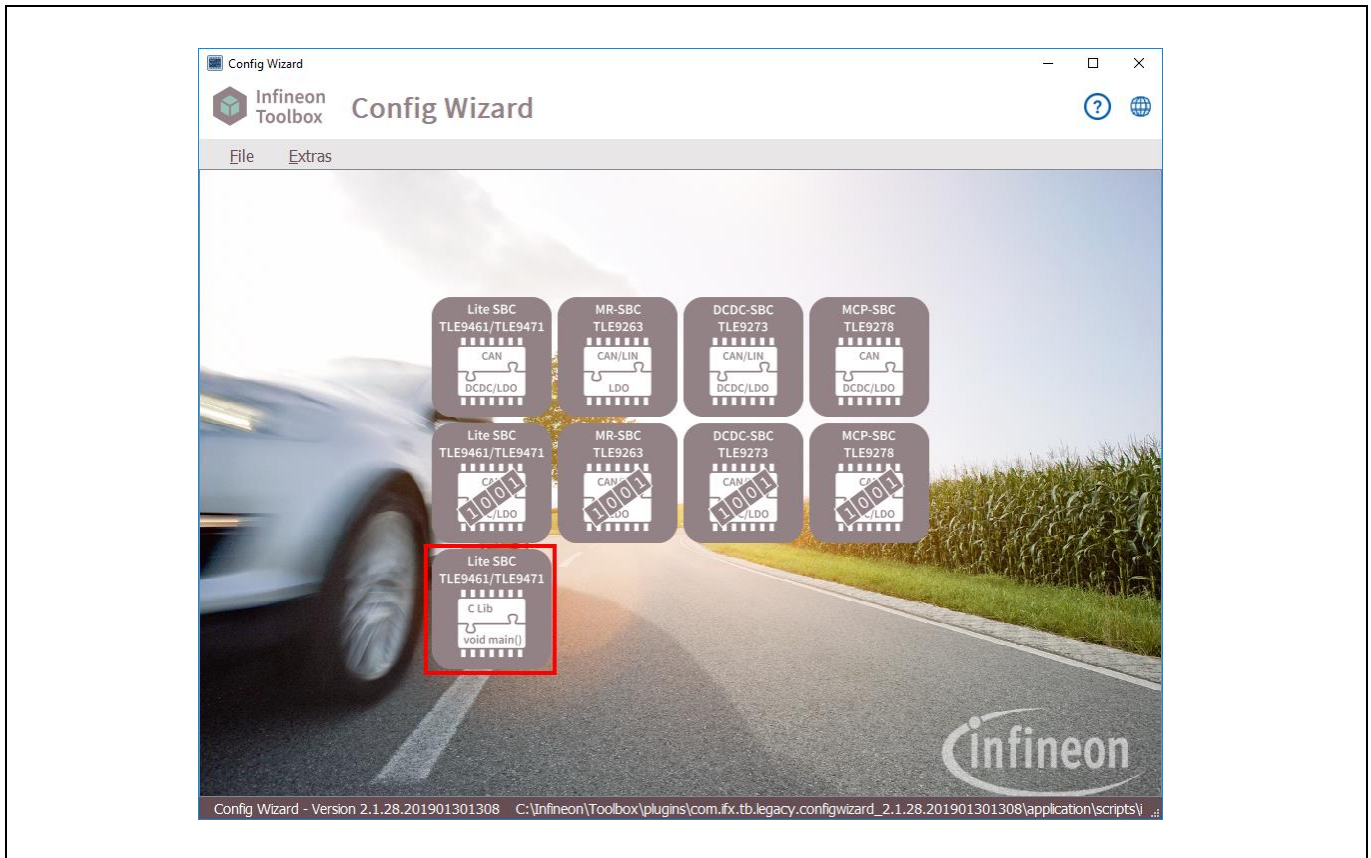
The code examples are based on the Lite SBC microcontroller library as part of the SBC Config Wizard. All settings of the TLE9471 can be configured graphically. In order to use it, the Infineon Toolbox must be downloaded first (<https://www.infineon.com/cms/en/tools/landing/infineontoolbox.html>).

After installation of the Infineon Toolbox, go to “Manage tools” and search for “Config Wizard for SBC” and press “Install”:

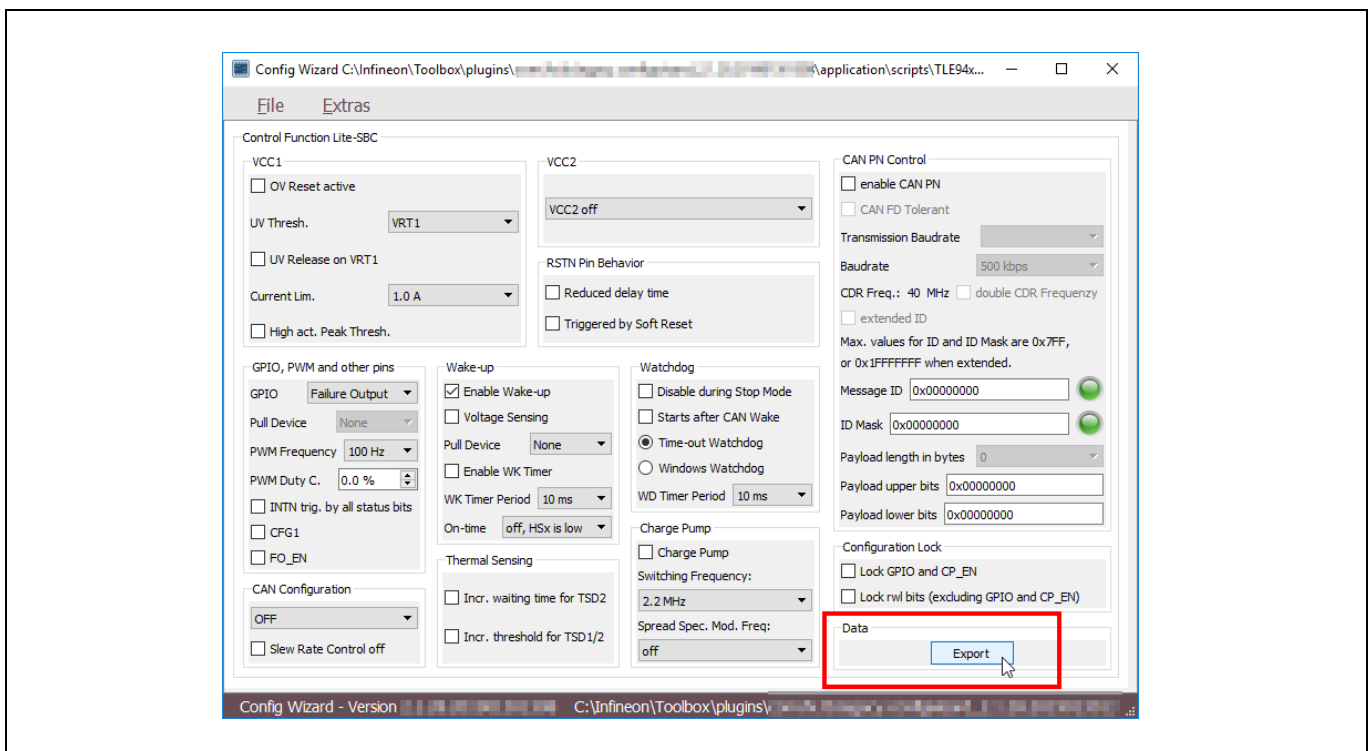


Lite SBC library

After installation, open the “Config Wizard for SBC” application and click on the Lite SBC C-Lib configurator:

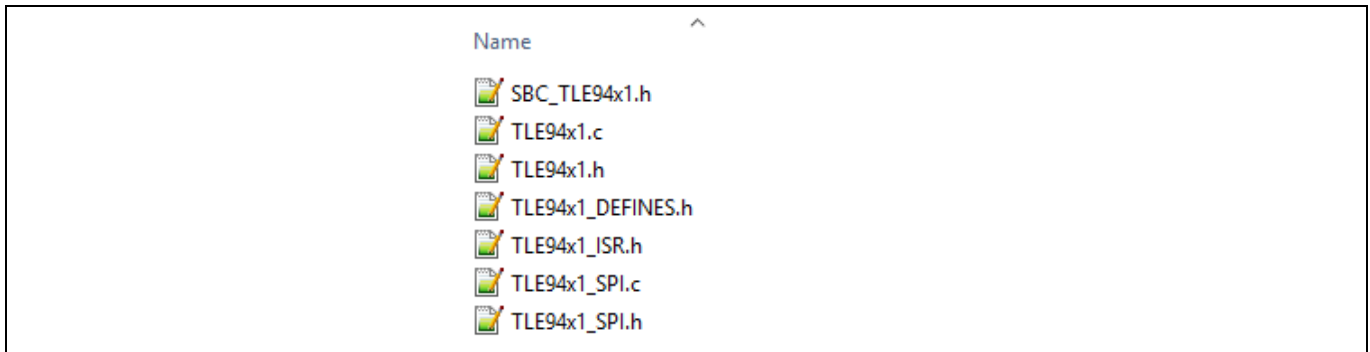


Afterwards, select your wished configuration of VCC2, internal PWM generator, CAN-PN configuration, etc. and press “Export”:



Lite SBC library

The export will output these 7 files:



- **SBC_TLE94x1.h**
Header file generated by the Config Wizard which stores all the configuration made by the user as defined
- **TLE94x1.h / TLE93x1.c**
Definition and Implementation of the main library functionalities
- **TLE94x1_DEFINES.h**
Definition of all SPI register addresses, bitmasks, bitpositions and possible field-values
- **TLE94x1_ISR.h**
Definition of all possible interrupt sources which can be generated by the SBC. The user can later on register self-defined functions which will be called back by the library in case a certain, registered event will occur
- **TLE94x1_SPI.h / TLE94x1_SPI.c**
Definition of SPI related functions (SPI initialization and SPI transmit functions). The SPI related functions have to be implemented by the user as they are dependent on the used microcontroller. Examples for Arduino Uno, Arduino Due and Infineon XMC are available.

Copy all 7 files to your Arduino project and modify the TLE94x1_SPI.c to work with Arduino UNO:

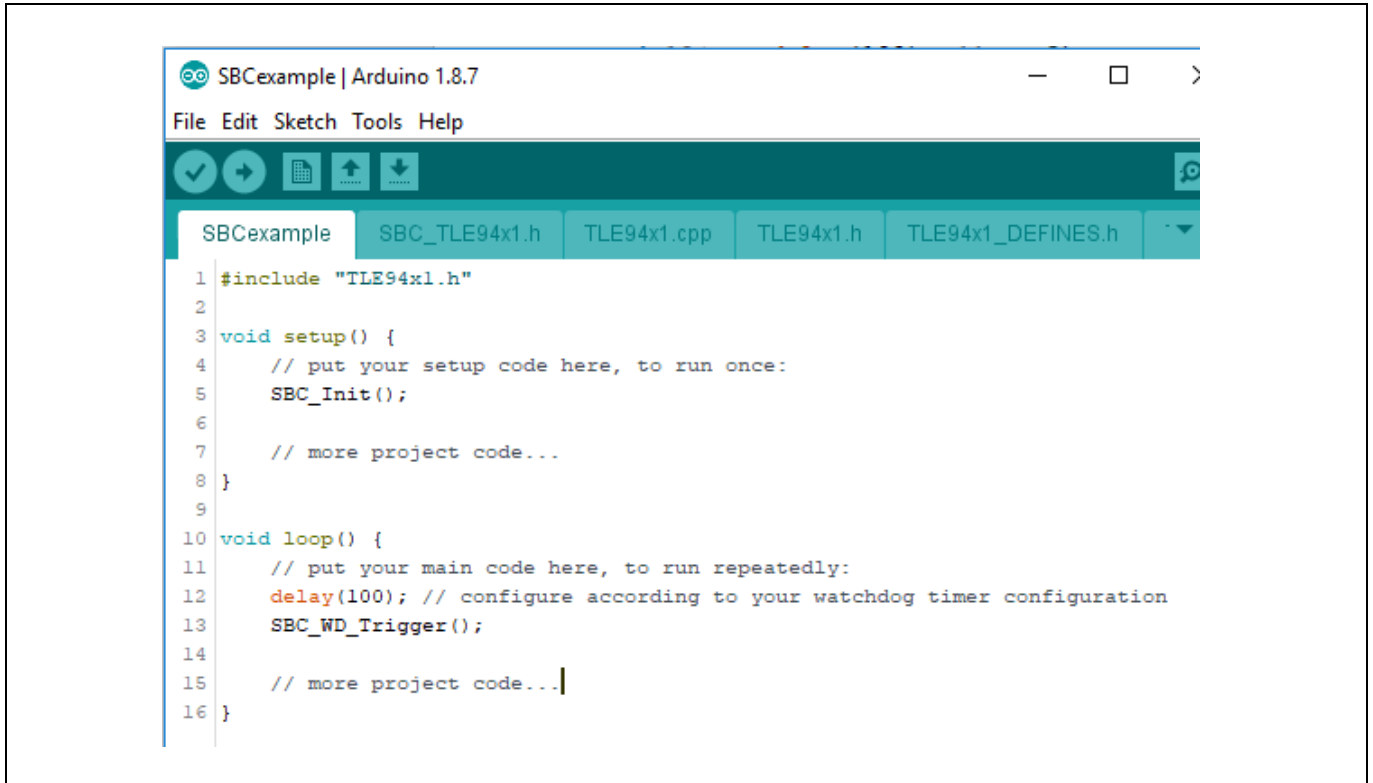
```
#include "TLE94x1_SPI.h"
#include <Arduino.h>
#include <SPI.h>

int8_t SBC_SPI_INIT(void) {
    pinMode(8, OUTPUT);
    digitalWrite(8, HIGH);
    SPI.begin();
    return 0;
}

uint16_t SBC_SPI_TRANSFER16(uint8_t Upper, uint8_t Lower) {
    uint16_t outdata = 0;
    SPI.beginTransaction(SPISettings(1000000, LSBFIRST, SPI_MODE1));
    digitalWrite(8, LOW);
    outdata = (SPI.transfer(Upper) << 8);
    outdata |= SPI.transfer(Lower);
    SPI.endTransaction();
    digitalWrite(8, HIGH);
    return outdata;
}
```

Lite SBC library

Include “TLE94x1.h” into your Arduino sketch via “#include”. Call *SBC_Init()* once at the beginning to initialize the SPI and to transmit all the configured values from the Config Wizard into the SBC. Inside your main *Loop()* routine you should call the *SBC_WD_Trigger()* to trigger the watchdog. When the SBC test-mode jumper is closed, the WD trigger can also be skipped.



```
SBCexample | Arduino 1.8.7
File Edit Sketch Tools Help
SBCexample SBC_TLE94x1.h TLE94x1.cpp TLE94x1.h TLE94x1_DEFINES.h
1 #include "TLE94x1.h"
2
3 void setup() {
4     // put your setup code here, to run once:
5     SBC_Init();
6
7     // more project code...
8 }
9
10 void loop() {
11     // put your main code here, to run repeatedly:
12     delay(100); // configure according to your watchdog timer configuration
13     SBC_WD_Trigger();
14
15     // more project code...
16 }
```

4 Additional information

For further information you may contact <http://www.infineon.com/SBC> or your regional FAE.

Revision history

Revision history

Document version	Date of release	Description of changes
1.1	2019-08-14	Editorial Changes
1.0	2019-05-04	Initial Release