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

Lite SBC Family

Z8F65294286

Getting Started

Rev 1.0, April 2019

Automotive Power

Revision History: 1

Previous Version: none

Page	**Subjects (major changes since last revision)**
1.0	Initial Release, All.

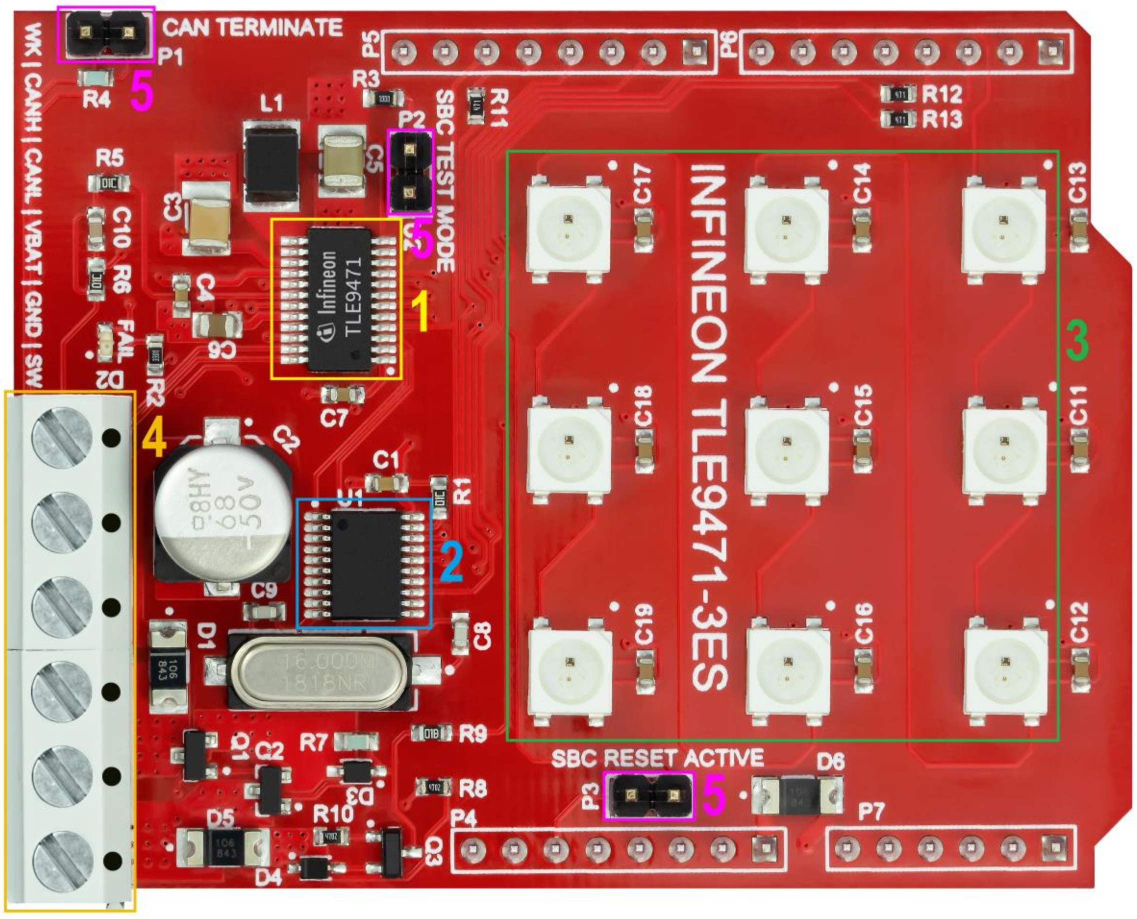
Table of contents

Schematic and layout of the shield can be found on Github:
https://github.com/Infineon/SBC-for-Arduino/

1. Board Overview

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

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



2. Terminals

The screw terminal contains 6 connections:

- **WK**: High voltage wake input. A 10 kOhm pull-down is populated on the board. A signal between 4V 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 (9-12V)
- **GND**: Main ground connection
- **SW**: Output of the high-side mosfets which are driven by the integrated charge-pump of the TLE9471. An external load up to 2.5A 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

3. LEDs

The board contains 9 pcs WS2812B LEDs 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.

4. 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 connected / disconnected 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 Ohms CAN termination
- Powerful 5V/500mA 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 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/

5. 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

6. Jumper Configuration

- **CAN TERMINATE**: A CAN bus must be properly terminated to ensure communication. This jumper inserts a 120 Ohms 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 Ohms = 60 Ohms.

- **SBC TEST MODE**: This jumper enables the test mode for the TLE9471-3ES so 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 takes a quite long time until the Arduino sketch 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 200ms 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 take control over the reset of the Arduino. This means after a power-on, the reset is toggled and the firmware is started. 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 Arduino-IDE not being able to flash the software to the Arduino. Therefore the jumper should be open.

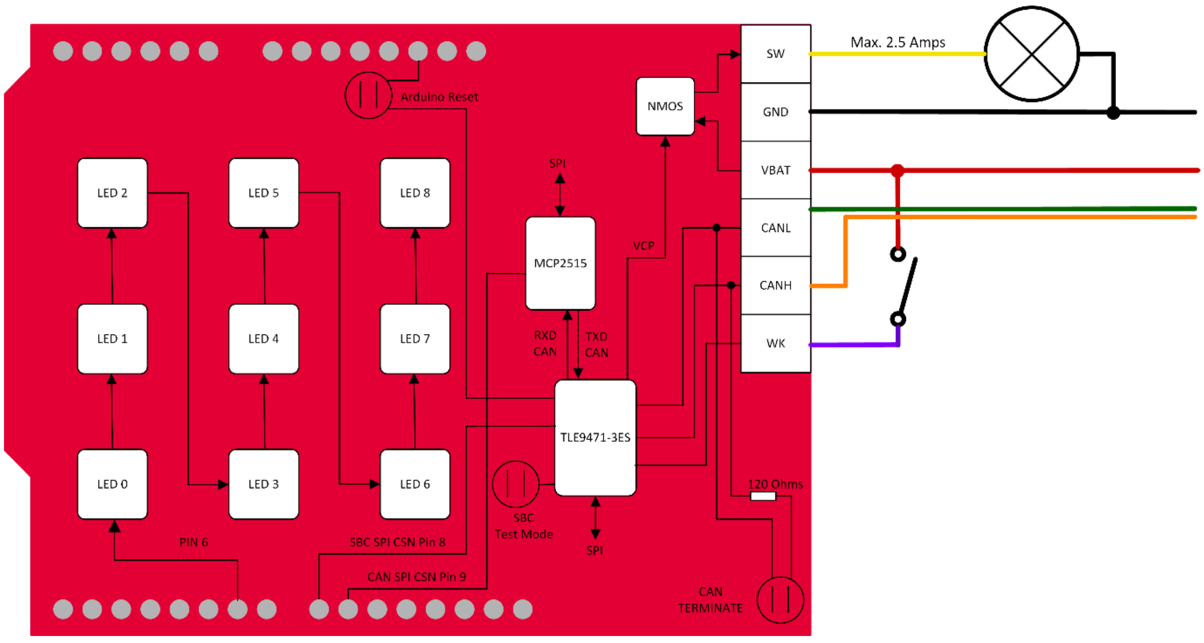
7. Code examples

There are code examples available for the TLE9471-3ES shield on Github.
See https://github.com/Infineon/SBC-for-Arduino/

Following examples are available:

- **General example**: This is an example to demonstrate how all the features like CAN communication, LEDs, charge-pump driven high-side mosfet, wake inputs can be used in a useful manner together.

  Please connect an external CAN communication node to the shield and make sure that the CAN bus is correctly terminated. The ground of the external CAN node and the ground of the shield should be connected together. Attach an external load to the “SW” pin (e.g. a light bulb) and connect a switch to the “WK” pin which is connected to main-supply as shown in the picture.



  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 by sending following CAN frame to the shield:
    - CAN-ID: 0x100 (not extended ID)
    - 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)

- 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

- **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 simply 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 mentioned CAN-PN configuration is written to the SBC and the LEDs will light up in green. The SBC is now waiting 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 the SBC sleep-mode directly and the LEDs will turn off.

As soon as 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.

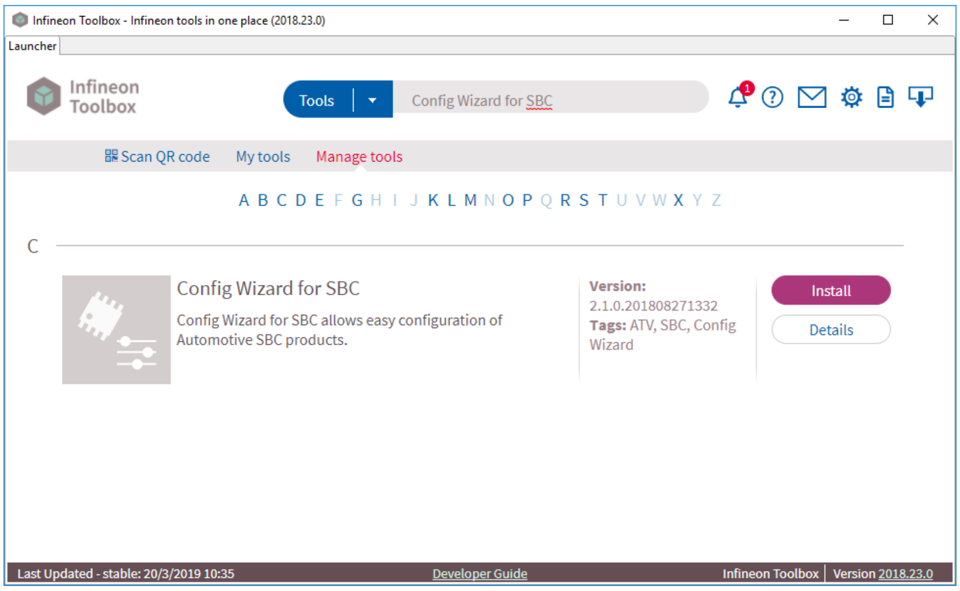
Bare in mind that the CAN-PN feature does not use the MCP2515 on the board. The CAN frames on the board must be acknowledged by an external node. The CAN frame is decoded by an integrated protocol handler which is exclusively used by the CAN-PN feature and not for standard CAN communication.

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.

8. Lite SBC Library

The code examples are based on the Lite SBC microcontroller library as part of the Config Wizard for SBC. All settings of the TLE9471 can be configured graphically. In order to use it, the Infineon Toolbox must be downloaded first (click here).

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



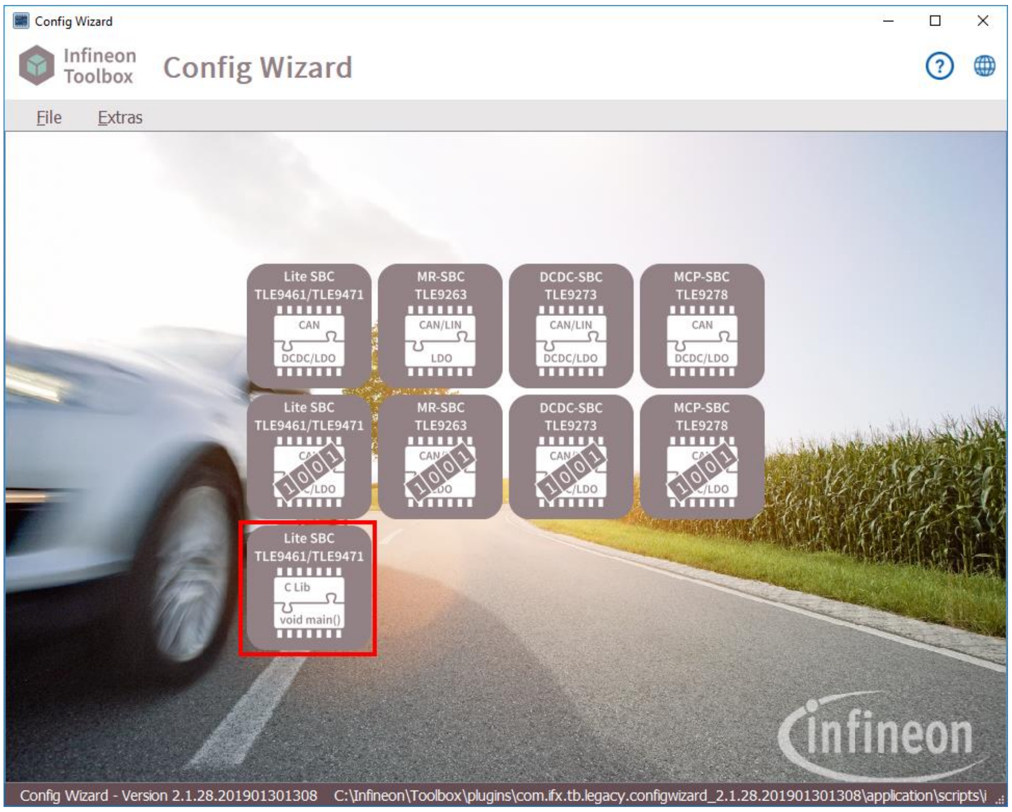
Config Wizard for SBC

Config Wizard for SBC allows easy configuration of Automotive SBC products.

Version: 2.1.0.201808271332
Tags: ATV, SBC, Config Wizard

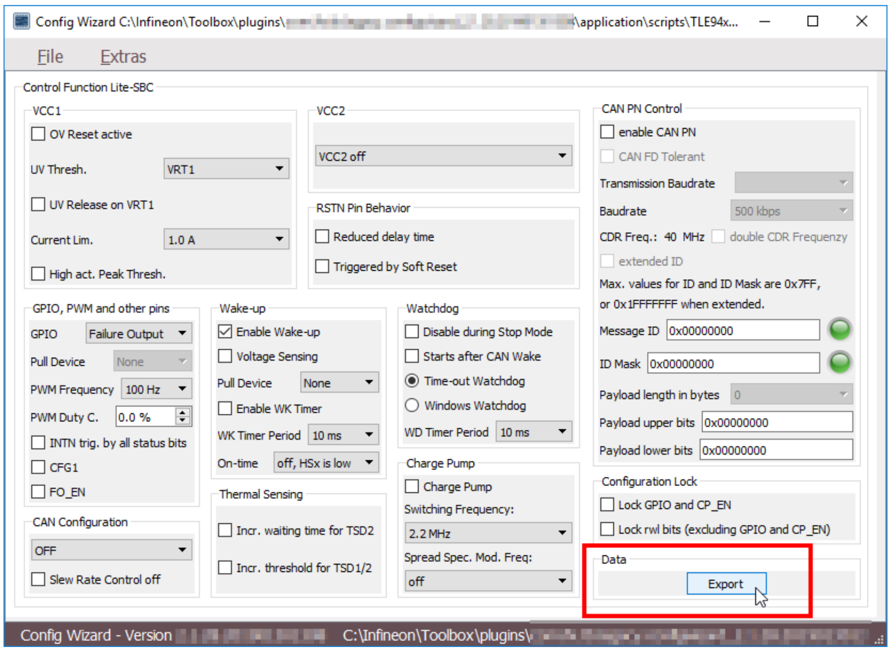
Last Updated - stable: 20/3/2019 10:35 | Infineon Toolbox | Version 2018.23.0

Open the “Config Wizard for SBC” application and click on the Lite SBC C-Lib configurator.



Config Wizard - Version 2.1.28.201901301308 C:\Infineon\Toolbox\plugins\com.ifx.tb.legacy.configwizard_2.1.28.201901301308\application\scripts\

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


Config Wizard C:\Infineon\Toolbox\plugins\[redacted]\application\scripts\TLE94x...

File Extras

Control Function Lite-SBC

VCC1
- ☐ OV Reset active
- UV Thresh. VRT1
- ☐ UV Release on VRT1
- Current Lim. 1.0 A
- ☐ High act. Peak Thresh.

VCC2
- VCC2 off

RSTN Pin Behavior
- ☐ Reduced delay time
- ☐ Triggered by Soft Reset

CAN PN Control
- ☐ enable CAN PN
- ☐ CAN FD Tolerant
- Transmission Baudrate
- Baudrate 500 kbps
- CDR Freq.: 40 MHz ☐ double CDR Frequenzy
- ☐ extended ID
- Max. values for ID and ID Mask are 0x7FF, or 0x1FFFFFFF when extended.
- Message ID 0x00000000
- ID Mask 0x00000000
- Payload length in bytes 0
- Payload upper bits 0x00000000
- Payload lower bits 0x00000000

GPIO, PWM and other pins
- GPIO Failure Output
- Pull Device None
- PWM Frequency 100 Hz
- PWM Duty C. 0.0 %
- ☐ INTN trig. by all status bits
- ☐ CFG1
- ☐ FO_EN

Wake-up
- ☑ Enable Wake-up
- ☐ Voltage Sensing
- Pull Device None
- ☐ Enable WK Timer
- WK Timer Period 10 ms
- On-time off, HSx is low

Watchdog
- ☐ Disable during Stop Mode
- ☐ Starts after CAN Wake
- ◉ Time-out Watchdog
- ○ Windows Watchdog
- WD Timer Period 10 ms

CAN Configuration
- OFF
- ☐ Slew Rate Control off

Thermal Sensing
- ☐ Incr. waiting time for TSD2
- ☐ Incr. threshold for TSD1/2

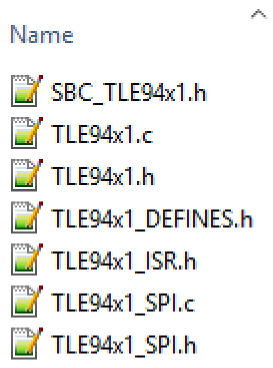
Charge Pump
- ☐ Charge Pump
- Switching Frequency: 2.2 MHz
- Spread Spec. Mod. Freq: off

Configuration Lock
- ☐ Lock GPIO and CP_EN
- ☐ Lock rwl bits (excluding GPIO and CP_EN)

Data
- Export

Config Wizard - Version [redacted] C:\Infineon\Toolbox\plugins\[redacted]

The export will output these 7 files.


Name

SBC_TLE94x1.h
TLE94x1.c
TLE94x1.h
TLE94x1_DEFINES.h
TLE94x1_ISR.h
TLE94x1_SPI.c
TLE94x1_SPI.h

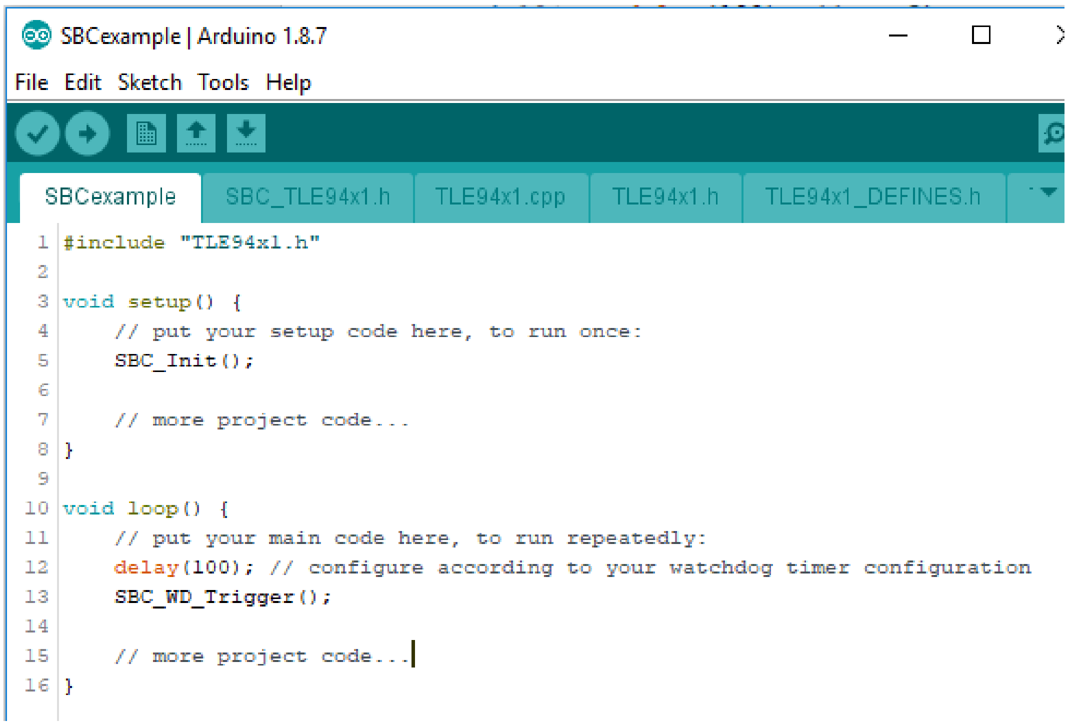
- **SBC_TLE94x1.h** Header file generated by the ConfigWizard which stores all the configuration made by the user as defined
- **TLE94x1.h / TLE93x1.c** Definition and Implementation of the main library functionalities
- **TLE94x1_DEFINES.h** Definition of all SPI register addresses, bitmasks, bitpositions and possible field-values
- **TLE94x1_ISR.h** Definition of all possible interrupt sources the SBC is able to generate. The user can 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, rename all .c files to .cpp as Arduino is based on C++ (and will not compile standard .c files) and modify the TLE94x1_SPI.cpp 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;
}
```

Include “TLE94x1.h” into your Arduino sketch via “`#include`” directive. Call *`SBC_Init()`* once at the beginning to initialize the SPI and to transmit all the configured values from the Config Wizard to 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.



```
#include "TLE94x1.h"

void setup() {
    // put your setup code here, to run once:
    SBC_Init();

    // more project code...
}

void loop() {
    // put your main code here, to run repeatedly:
    delay(100); // configure according to your watchdog timer configuration
    SBC_WD_Trigger();

    // more project code...
}
```

9. Additional Information

For further information please visit www.infineon.com/SBC or contact your regional FAE.

Edition 2019-05-04

Published by
Infineon Technologies AG
81726 Munich, Germany