SP49 Development-Kit

User guide

About this document

Scope and purpose

This document introduces the SP49 development-kit, its features and how to use them.

Intended audience

Intended audience are all engineers working with the SP49 development-kit, e.g. for developing and debugging code.

Table of contents

1 Introduction & Setup

The SP49 development-kit contains the SP49 development-board and shall enable the user to develop, debug and test his code.

1.1 System requirements

For using the SP49 development-kit the following hardware and software is required

- Computer running under Microsoft windows 10 or 11
- The SP49 development-kit itself (ordered from Infineon)
- Micro-USB Cable (comes with the kit)
- MDK-ARM Version 5.29 or above (Keil uVision 4 Version 5, can be purchased and downloaded from www.keil.com)
- SP49 Keil device-pack (can be downloaded from MyInfineon, https://myicp.infineon.com/sites/tpms-sp49/. Registration required.)
- SP49 development-board GUI (can be installed via Infineon Developer Center, https://www.infineon.com/toolbox)
- *Optional*: TPMS programming tool (can be installed via Infineon Developer Center)

1.2 Environment setup

This chapter explains how to setup the environment for SP49 step by step.

1.2.1 Setup MDK-ARM (Keil uVision 4 Version 5)

Please go to www.keil.com and download the MDK-ARM. For small projects (limited to 32kB, so enough for SP49) and for product evaluation the MDK-Lite version, which is for free, is enough. For productive code and professional code development it is recommended to purchase one of the payed versions.

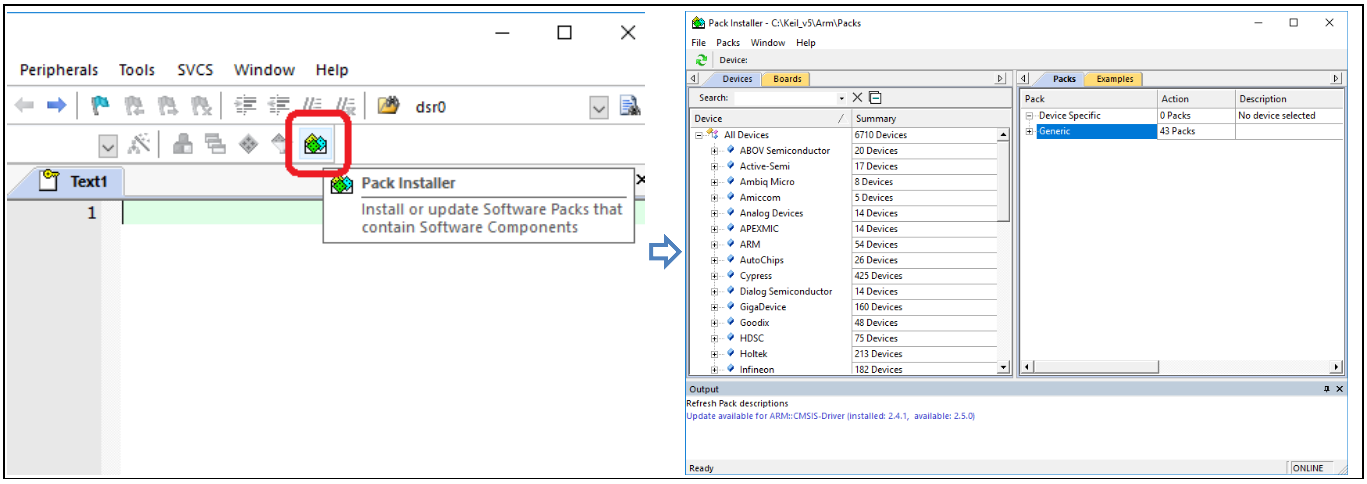
After downloading the installer, start the installation. By default Keil will be installed to C:\Keil_v5\ and it is recommended not to change this default location. At the end it can happen that a window “Pack Installer” opens, which can be closed at that point after it finished its initial actions – installation of the SP49 pack will be done later.

If a license shall be added after installation has finished, Keil must be started once with administrator privileges. Then, in order to enter a license-code (LIC) go to the top-menu “File” -> “License Management”. By clicking on “Get LIC via Internet…” a LIC can be generated from a product serial number (PSN). In the end add the obtained LIC in the bottom textbox and click on “Add LIC”. If no license shall be used, nothing has to be done at this step. Keil is now ready to use.

1.2.2 Install SP49 Keil Pack

The SP49 Keil Pack provides an easy way to add the necessary device information to Keil. Moreover, it contains sample codes that illustrate the usage of the different blocks of SP49 and can be used as a basis for code development.

The SP49 Keil Pack can be downloaded from the same Infineon site as this User Guide after logging into MyInfineon. After download completed, open Keil and go to the “Pack Installer”. The “Pack Installer” is opened by a click on the symbol in Keil as shown in the next figure.



Pack Installer
Install or update Software Packs that contain Software Components

Device	Summary
All Devices	6710 Devices
ABOV Semiconductor	20 Devices
Active-Semi	17 Devices
Ambiq Micro	8 Devices
Amiccom	5 Devices
Analog Devices	14 Devices
APEXMIC	14 Devices
ARM	54 Devices
AutoChips	26 Devices
Cypress	425 Devices
Dialog Semiconductor	14 Devices
GigaDevice	160 Devices
Goodix	48 Devices
HDSC	75 Devices
Holtek	213 Devices
Infineon	182 Devices

Pack	Action	Description
Device Specific	0 Packs	No device selected
Generic	43 Packs	

Refresh Pack descriptions
Update available for ARM::CMSIS-Driver (installed: 2.4.1, available: 2.5.0)

Figure 1 Open Keils “Pack Installer”

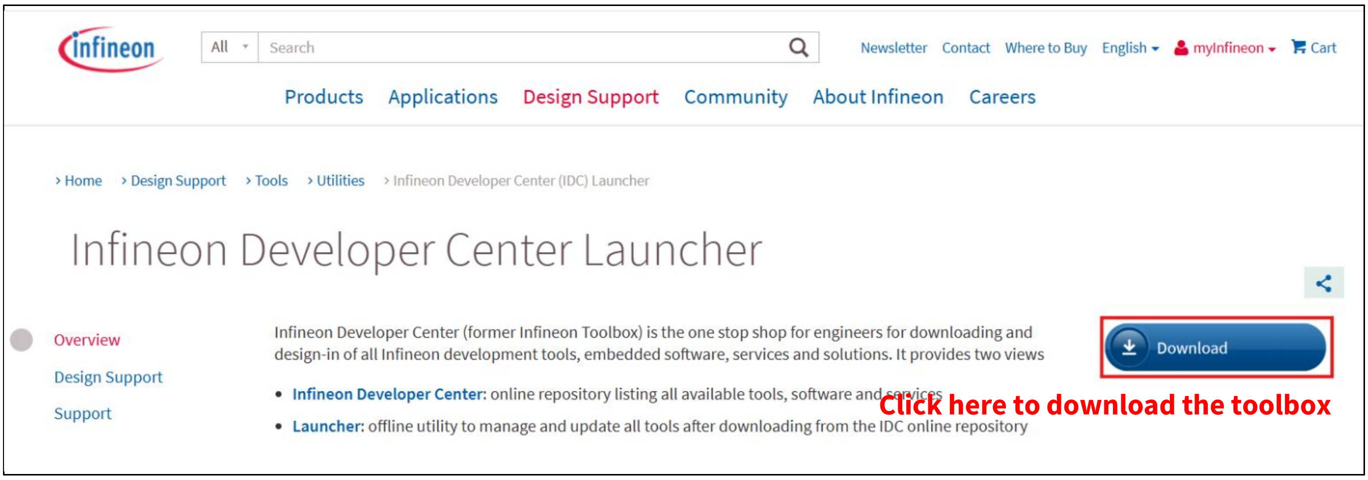
Now go to “File” -> “Import…” and navigate to the previously download pack file for SP49. When the installation is finished the pack installer windows can be closed. Keil will then ask for a reload of the packs, which has to be confirmed with “Yes”. Keil is now ready to work with SP49.

1.2.3 Install SP49 development-board GUI

The GUI for the SP49 development-board is available via the Infineon Development Center. To install it, please follow the following steps.

a) Download and install the Infineon Development Center

Go to https://www.infineon.com/toolbox and download the toolbox. After downloading, install it to your computer.


All
Search
Newsletter Contact Where to Buy English myInfineon Cart
Products Applications Design Support Community About Infineon Careers
› Home › Design Support › Tools › Utilities › Infineon Developer Center (IDC) Launcher
Overview
Design Support
Support
Download
Click here to download the toolbox

Infineon Developer Center Launcher

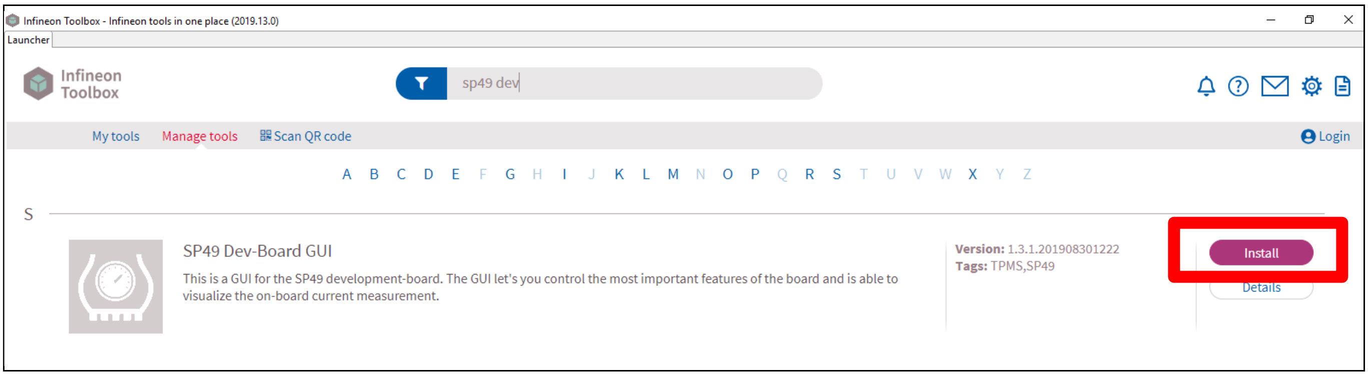
Infineon Developer Center (former Infineon Toolbox) is the one stop shop for engineers for downloading and design-in of all Infineon development tools, embedded software, services and solutions. It provides two views

- **Infineon Developer Center:** online repository listing all available tools, software and services
- **Launcher:** offline utility to manage and update all tools after downloading from the IDC online repository

Figure 2 Download Infineon Developer Center

b) Run Infineon Developer Center and install SP49 development-board GUI

After installation of the Infineon Developer Center, the Infineon Developer must be started. Afterwards open the tab “Manage tools” and enter “SP49 dev” into the search-box on the top of this tab. The tool will then be found. Install the tool by a click on “Install”.



Infineon Toolbox - Infineon tools in one place (2019.13.0)

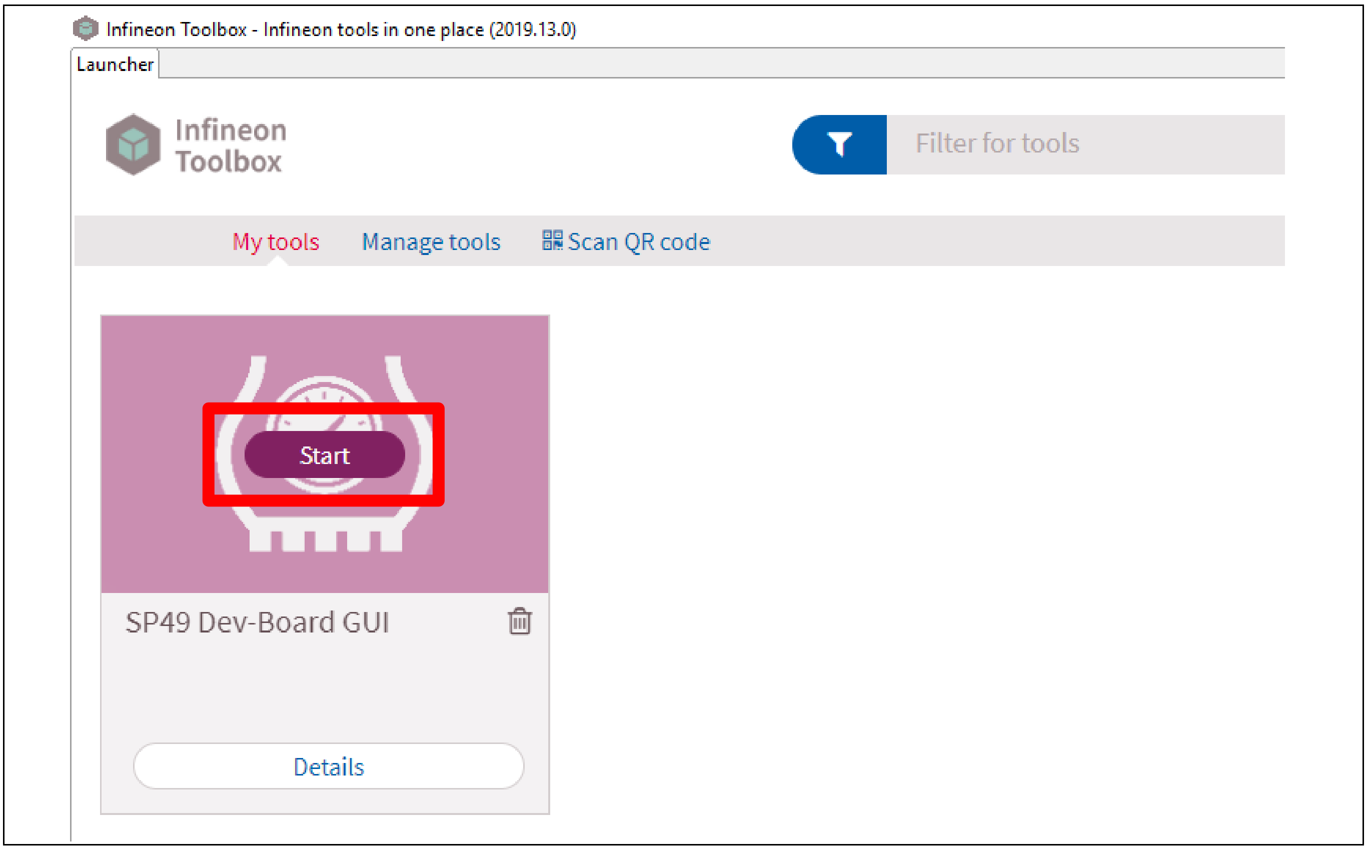
SP49 Dev-Board GUI

This is a GUI for the SP49 development-board. The GUI let's you control the most important features of the board and is able to visualize the on-board current measurement.

Version: 1.3.1.201908301222
Tags: TPMS,SP49

Figure 3 Install SP49 development-board GUI from Infineon toolbox

After the license agreement was accepted, the GUI will be installed and the toolbox must be restarted. Finally the SP49 development-board GUI can be started on the tab "My tools" within the Infineon Develper Center.



Infineon Toolbox - Infineon tools in one place (2019.13.0)

Figure 4 Starting tool from Infineon toolbox

c) Updating the GUI

Whenever there is an update of the GUI available, the Infineon Developer Center will show this and an "Update"-button will appear. Please make sure to always use the latest version of this GUI.

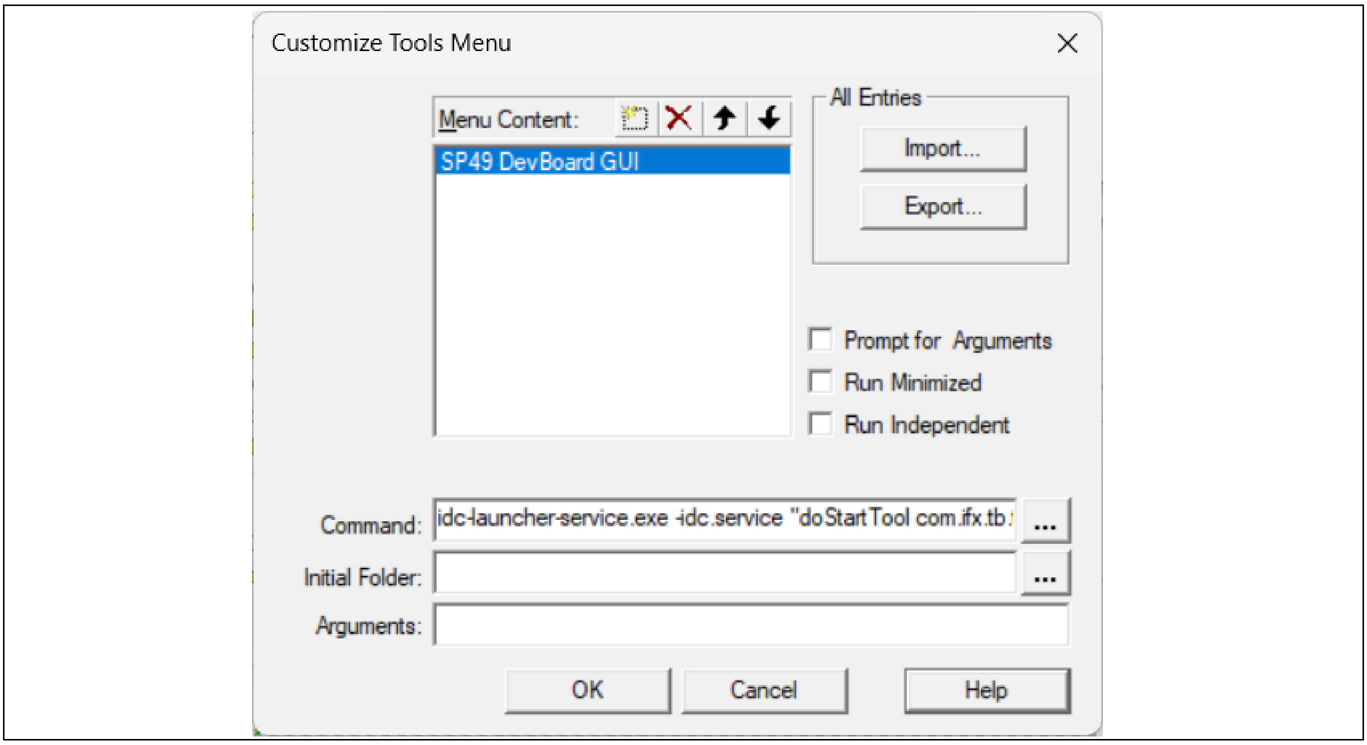
1.2.4 Making SP49 development-board GUI startable from Keil

Keil's menu can be customized such that it is possible to start the SP49 development-board GUI directly from Keil. To configure this feature, open Keil, then from the top menu open "Tools" -> "Customize Tools Menu…". Then click on the button "New" next to "Menu Content" so that a new entry will be created. Now enter a name for the entry, for example “SP49 DevBoard GUI”. Now activate the checkbox “Run Independent” and insert into “Command” the following text:

```
idc-launcher-service.exe -idc.service "doStartTool
com.ifx.tb.tool.sp49developmentboardgui FromKeil"
```

Note: *The folder that contains the “idc-launcher-service.exe” must be in the PATH environment variable. Alternatively, the full path to the “idc-launcher-service.exe” can be used*

The configured entry should now look like shown in the next figure.



SP49 DevBoard GUI

□ Prompt for Arguments
□ Run Minimized
□ Run Independent

Command: idc-launcher-service.exe -idc.service "doStartTool com.ifx.tb.
Initial Folder:
Arguments:

Figure 5 **Configuring tools menu in Keil for SP49 development-board GUI**

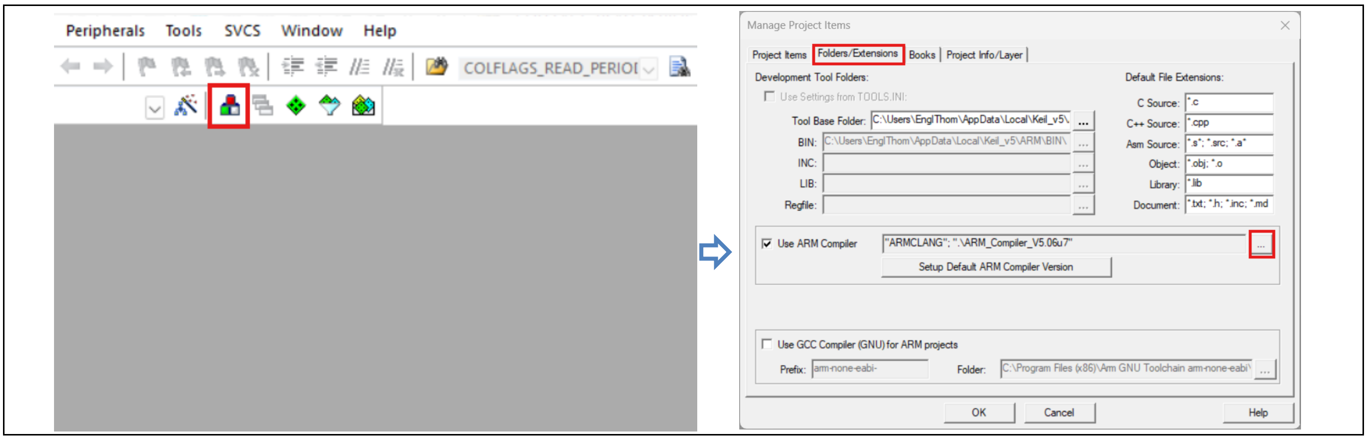
When you now click “OK” you will find a new entry in the top menu under “Tools”, e.g. “SP49 DevBoard GUI”.

1.2.5 Recommended: Install ARM Compiler version 5

Bot the firmware of SP49 and the sample codes provided as part of the Keil pack described in section 1.2.2 are developed using the Arm Compiler version 5. Particularly the sample codes are not compatible with the Arm Compiler version 6.

Currently, the MDK-ARM is provided with Arm compiler version 6. For compatibility, it is recommended to install Arm Compiler version 5. The Arm Compiler version 5 can be installed in parallel to the compiler version 6. To install Arm Compiler version 5, please go to https://developer.arm.com/documentation/ka005198/latest/, download one of the Arm Compiler version 5 from there and start the downloaded installer. Please note down the installation location of the Arm Compiler version 5, because it will be needed in the following steps.

After successful installation, the Arm Compiler version 5 must be added to the Keil installation. To do so, open the project item manager and navigate to the Folders/Extensions tab. In this tab, click on the button with “…” marked in the following figure:



Manage Project Items

Project Items | Folders/Extensions | Books | Project Info/Layer

Development Tool Folders:

□ Use Settings from TOOLS.INI:

Tool Base Folder: C:\Users\EnglThom\AppData\Local\Keil_v5\

BIN: C:\Users\EnglThom\AppData\Local\Keil_v5\ARM\BIN\

INC:

LIB:

Regfile:

Default File Extensions:

C Source:	*.c
C++ Source:	*.cpp
Asm Source:	*.s*; *.src; *.a*
Object:	*.obj; *.o
Library:	*.lib
Document:	*.txt; *.h; *.inc; *.md

☑ Use ARM Compiler "ARMCLANG"; ".\ARM_Compiler_V5.06u7" ...

Setup Default ARM Compiler Version

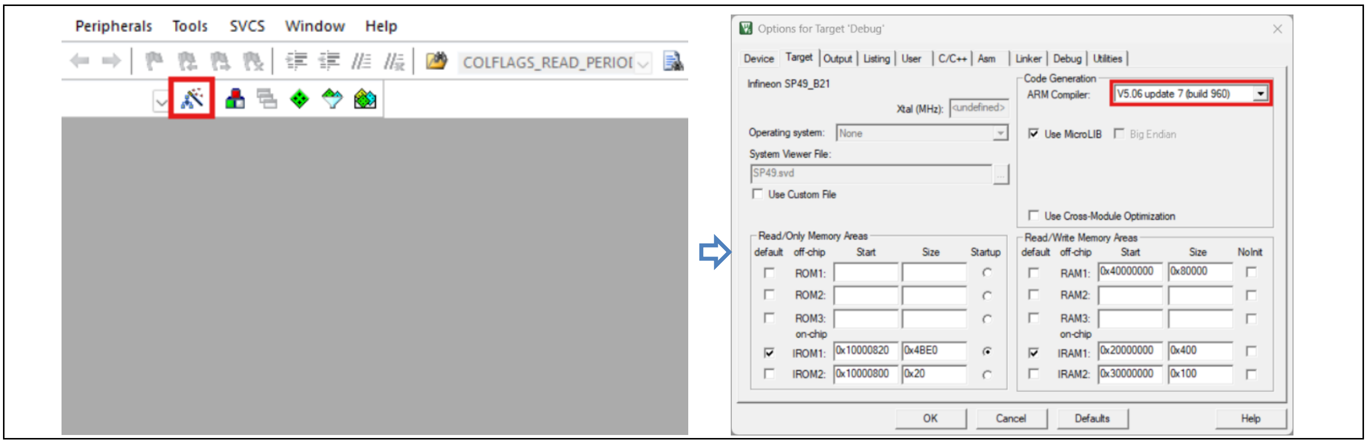
□ Use GCC Compiler (GNU) for ARM projects

Prefix: arm-none-eabi- Folder: C:\Program Files (x86)\Arm GNU Toolchain arm-none-eabi\

OK Cancel Help

This will open a new window, in which the Arm Compiler Version 5 can be added by clicking on the button “Add another ARM ompiler Version to List…”

Note that it may be necessary to explicitly select the Arm Compiler version 5 for each project separately. For this, open the Project Target options and select the Arm Complier version 5 in the “Target” tab as shown in the following figure:



Options for Target 'Debug'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

Infineon SP49_B21

Xtal (MHz): <undefined>

Operating system: None

System Viewer File: SP49.svd

□ Use Custom File

Code Generation

ARM Compiler: V5.06 update 7 (build 960)

☑ Use MicroLIB □ Big Endian

□ Use Cross-Module Optimization

Read/Only Memory Areas

default	off-chip	Start	Size	Startup
□	ROM1:			○
□	ROM2:			○
□	ROM3:			○
	on-chip			
☑	IROM1:	0x10000820	0x4BE0	●
□	IROM2:	0x10000800	0x20	○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
□	RAM1:	0x40000000	0x80000	□
□	RAM2:			□
□	RAM3:			□
	on-chip			
☑	IRAM1:	0x20000000	0x400	□
□	IRAM2:	0x30000000	0x100	□

OK Cancel Defaults Help

1.2.6 Optional: Install TPMS programming tool

For use cases where the DUT attached to the SP49 programming-board shall just be programmed without the use of Keil uVision, a standalone programming tool is also provided. This tool is also available in the Infineon Developer Center. For installation steps refer to chapter 1.2.3 but search for the tool “TPMS programming”.

Note: *The TPMS programming tool uses the programming mode via I2C. Therefore, when using this tool, the board jumpers “CON_SCL/Tx” and “CON_SDA/Rx” need to be set accordingly. Jumper “CON_SCL/TX” needs to be set to “SCL>PP0” and jumper “CON_SDA/RX” needs to be set to “SDA>PP1”, see also chapter 2.1.6.*

2 SP49 development environment description

This chapter gives an overview of development environment for SP49 and how to use it.

2.1 SP49 development-board

The SP49 development-board and the most important features are shown in the next figure.

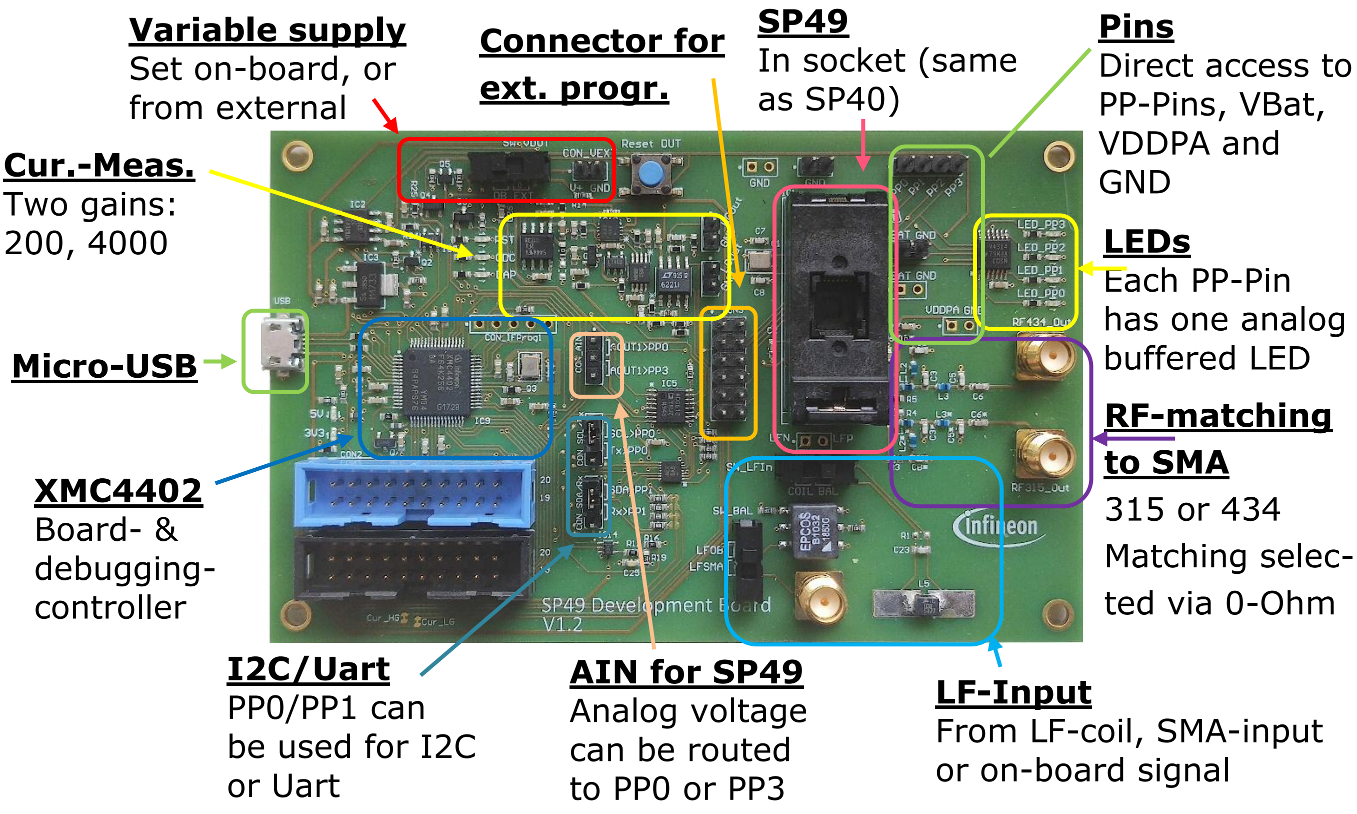

Variable supply
Set on-board, or from external
Connector for ext. progr.
SP49
In socket (same as SP40)
Pins
Direct access to PP-Pins, VBat, VDDPA and GND
Cur.-Meas.
Two gains: 200, 4000
LEDs
Each PP-Pin has one analog buffered LED
Micro-USB
RF-matching to SMA
315 or 434
Matching selected via 0-Ohm
XMC4402
Board- & debugging-controller
I2C/Uart
PP0/PP1 can be used for I2C or Uart
AIN for SP49
Analog voltage can be routed to PP0 or PP3
LF-Input
From LF-coil, SMA-input or on-board signal
SP49 Development Board V1.2

Figure 6 **Overview of SP49 development-board and its main features**

Note: Although the RF- and LF-matching give a good idea of the behavior of the flash application, this board is not intended for quantitate measurements.

For using the board, simply connect it to a PC via micro-USB cable. The board supports several use-cases:

- Code debugging: A SP49 sample must be placed in the socket, at least the jumpers CON3.1 and CON3.[4 … 6] must be set (see Figure 8). AOUT1 of the board must not be routed to PP3 (see chapter 2.1.7). Code can be debugged via Keil uVision.
- LF-receiver testing: A LF-signal can be either generated by the board, or can be applied externally either via SMA-connector or LF-coil (see chapter 2.1.2 and chapter 2.3.5).
- RF-transmitter testing: The output of the RF-transmitter can be measured in a 50-Ohm system at either the RF315_Out SMA-connector or the RF434_Out SMA-connector. The desired path must be selected via two 0-Ohm resistors in each frequency branch.
- Measurement of signals at PP-pins: Voltage-levels can be observed at the LEDs, or directly at the Pins.
- Current-measurement: The dynamic current-consumption of the chip can be measured and displayed directly in the SP49 development-board GUI, or via an external oscilloscope by measuring the output-voltage of the current-measurement circuit directly.

In general the board is split into two parts, the interface- / controller-part and the DUT-part. In the middle there is the connector CON3. When all jumpers from CON3 are removed, the whole DUT-part behaves as not present and an external board can be attached. If the jumpers on CON3 are placed, the DUT-part is active and the DUT in the socket is used.

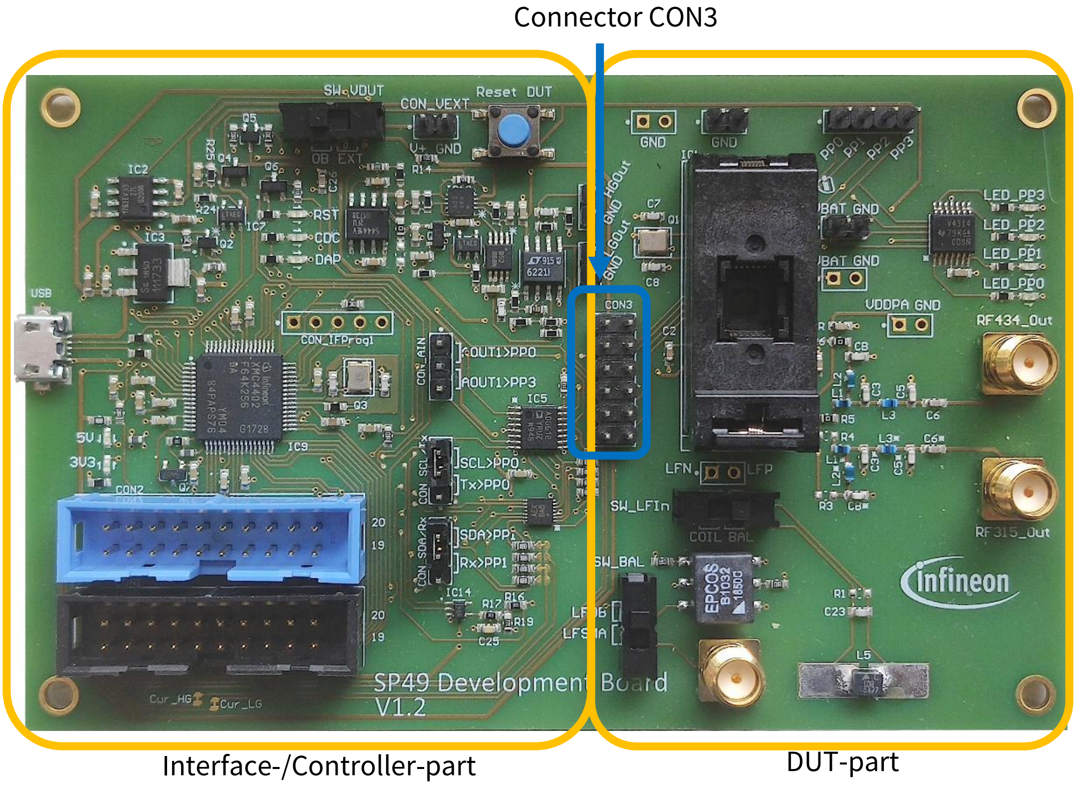
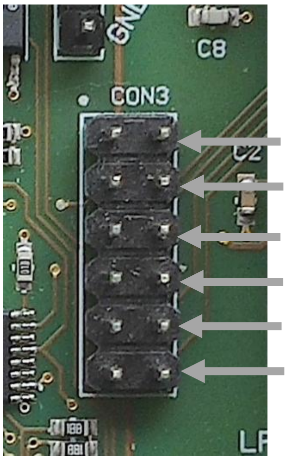


Figure 7 Split of SP49-develoment board into interface- / controller-part and DUT-part

The assignment of the signals in CON3 is shown in the next figure.

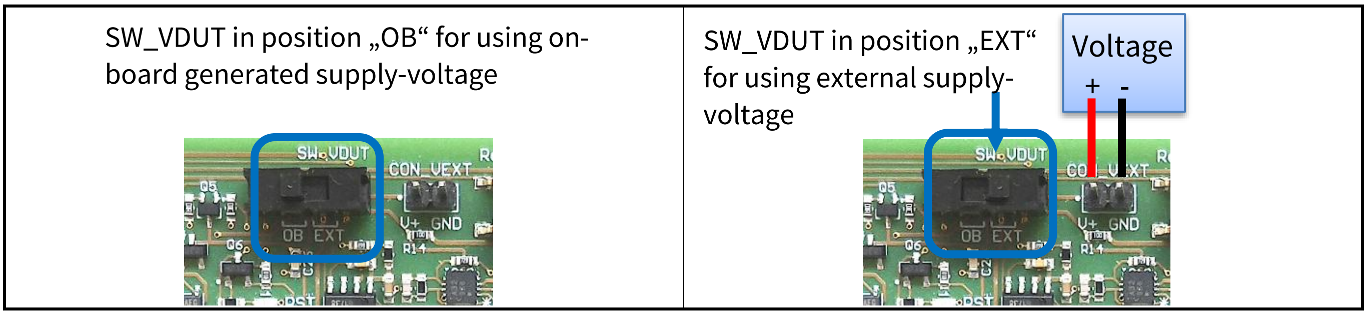


Pin	Signal
CON3.1	VBat
CON3.2	PP0
CON3.3	PP1
CON3.4	GND
CON3.5	PP2
CON3.6	PP3

Figure 8 Pin-assignments of CON3

2.1.1 Variable supply of the DUT

As VBat for the DUT the user has two options, either using the on-board generated voltage, or applying an external voltage. For using the on-board voltage, the switch SW_VDUT must be placed into position “OB”. For using an external supply this switch must be placed into position “EXT”, as shown in the next figure.



SW_VDUT in position „OB“ for using on-board generated supply-voltage

SW_VDUT in position „EXT“ for using external supply-voltage

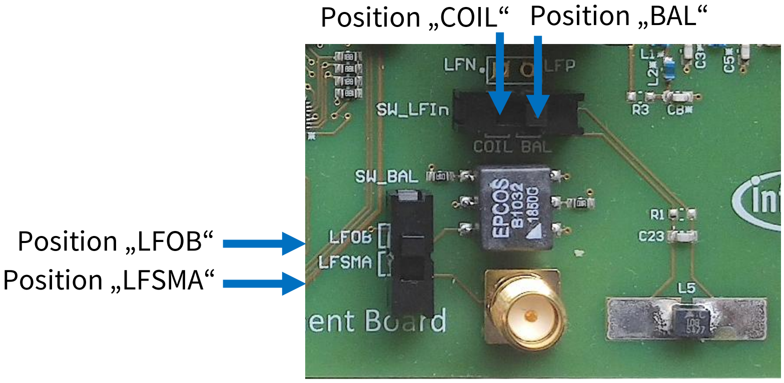
Figure 9 Variable supply of the DUT

Note: *Between the selection of the voltage-supply and the DUT there is the current-measurement circuit. In this circuit a 10-Ohm +/- 0.1% shunt resistor is added in series. This will cause a slight voltage-drop at the DUT at higher currents.*

Attention: ***The external voltage must not exceed [0V ... 3.6V]!***

2.1.2 Various LF-sources as input for LF-receiver

For testing the LF-receiver there are three possibilities how to apply a LF-signal, either from an on-board source, via 50-Ohm SMA-connector or via a LF-coil. The selection of the source happens via two switches SW_LFIn and SW_BAL. SWLFIn chooses between the LF-coil and the Balun. If the Balun is chosen, either the on-board generated LF-signal can be chosen or the one coming via SMA-connector. The selection possibilities are shown in the next figure.



Position „COIL“ Position „BAL“

Position „LFOB“

Position „LFSMA“

Position of SW_LFIn	Position of SW_BAL	Source of LF-signal
COIL	*Don’t care*	LF-coil
BAL	LFOB	On-board generated
BAL	LFSMA	SMA-connector

Figure 10 Various LF-input sources

Note: *The amplitude of the on-board generated LF-signal is fixed to ~20mVpp and cannot be modified. It is chosen such high that it should always be detected by SP49 regardless the configure sensitivity.*

2.1.3 Current-measurement

The SP49 development-board also offers the functionality to measure the dynamic current consumption of the connected device. Therefore, between the voltage-supply and the DUT, a 10-Ohm +/- 0.1% shunt resistor is placed. The voltage-drop across this resistor is then amplified with two gains. The amplified voltage is routed to the XMC on the board, which can convert it and stream it via USB to the PC, where the measured current can be displayed in the SP49 development-board GUI. Alternatively the amplified voltage can be directly measured by an external oscilloscope at the pins "LGOut" (low-gain out) and "HGOut" (high-gain out). The concept of this circuit is shown in the next figure.

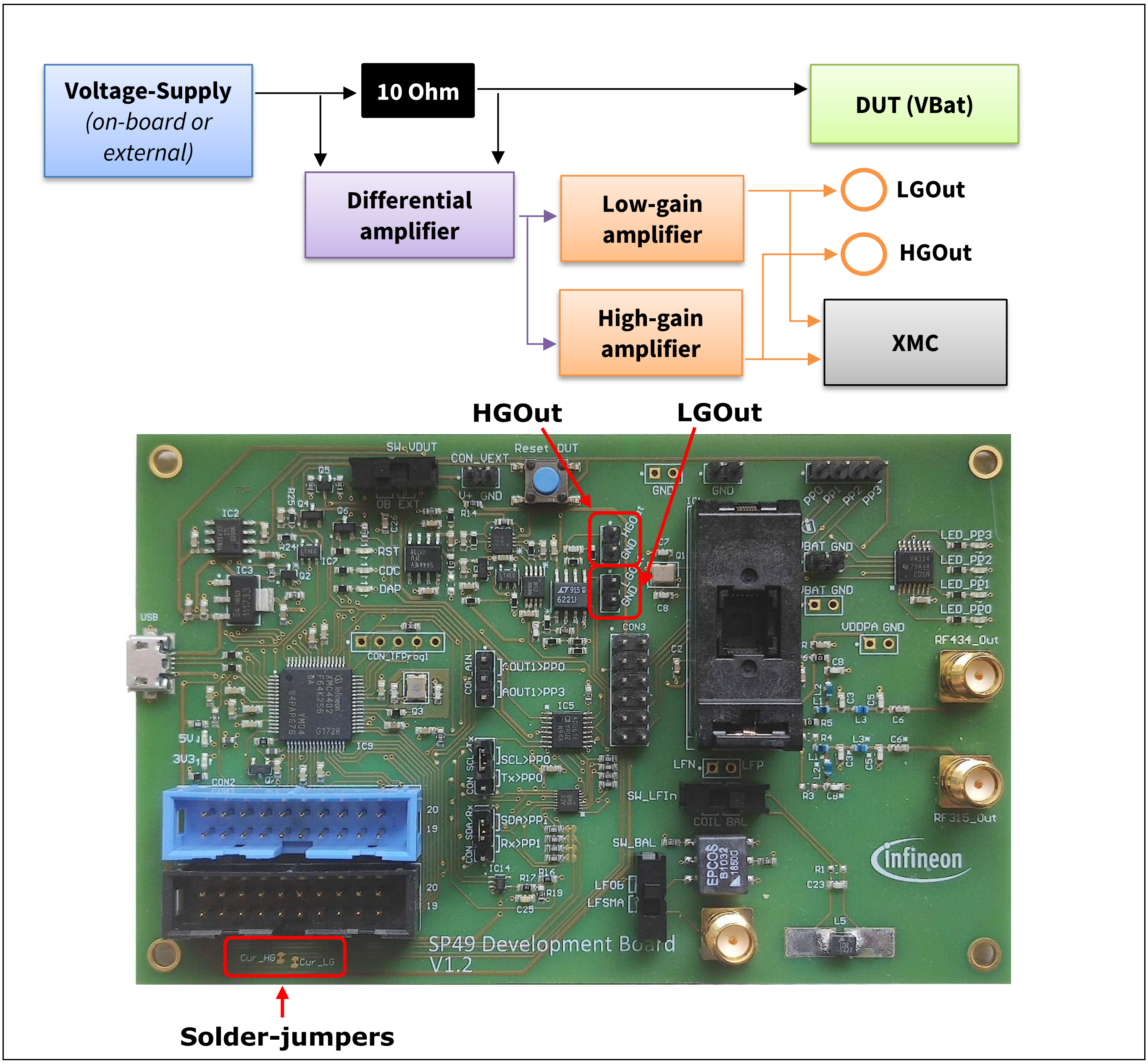


Figure 11 Concept of current-measurement on the SP49 development-board

Note: If current shall only be measured via external oscilloscope, it should be considered to cut the two solder-jumpers on the bottom-left of the board, in order that the sample-and-hold circuit of the XMC's analog input does not influence the measurement. Then the current-measurement feature of the SP49 development-board will not work anymore.

Attention: Using external oscilloscope and XMC for current-measurement in parallel is not possible, because they will be influenced by each other.

To use the current-measurement circuit via an external oscilloscope, the following characteristics of the current-measurement circuit are relevant to know:

Table 1 Characteristics of current-measurement circuit, room-temperature

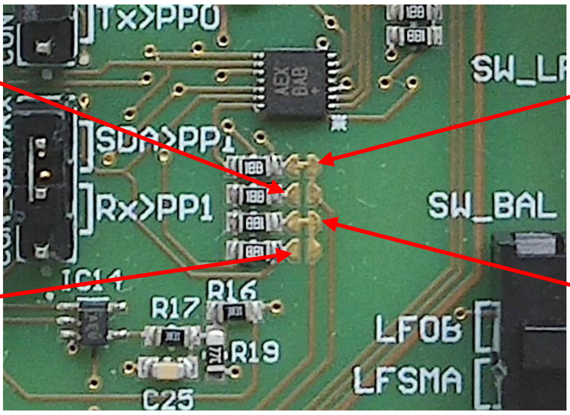
Parameter	Values			Unit	Note
	Min.	Typ.	Max.		
Bandwidth LGOut		200		kHz	
Bandwidth HGOut		120		kHz	
Low-Gain	214	217	220	V/A	Including 10-Ohm shunt-resistor. The XMC is calibrated during board production with the exact value[1].
High-Gain	4320	4420	4520	V/A	Including 10-Ohm shunt-resistor. The XMC is calibrated during board production with the exact value[1].
LGOut Offset	-8	2	12	mV	The XMC is calibrated during board production with the exact value[1].
HGOut Offset	50	82	180	mV	The XMC is calibrated during board production with the exact value[1].
Sampling frequency of XMC in mode "Auto"			32	kHz	Sampling frequency is limited by USB connection. Sampling frequency is automatically chosen by SP49 development board GUI.
Sampling frequency of XMC in mode "Normal"			200	kHz	Sampling frequency is limited by USB connection. Sampling frequency is automatically chosen by SP49 development board GUI.

[1] The individually calibrated values can be readout from the board via a dll written for the board in .NET

In case very low currents shall be measured, an additional switch was added to the board which can disconnect the connection between any PP-pin and the rest of the board, in order that no currents through the PP-Pins can influence the result. Consequently the current consumed by the DUT-part can be precisely measured by the measurement-circuit. This switch can be enabled via the SP49 development-board GUI (see chapter 2.3.3) and consumes in open-state less than 1nA.

2.1.4 Streaming logical levels of two pins

Since Board-FW version 1.3 it is possible that also logical levels of two pins (called Pin0 and Pin1) can be streamed to the GUI, together with each current measurement. Those levels will be displayed in the SP49 development-board GUI as separate signals. Since HW version 1.2 the user can select on the board which pins from SP49 shall be streamed. By default PP2 and PP3 levels are streamed, but via solder-jumpers it is possible to select PP0 instead of PP2 and/or PP1 instead of PP3. The solder-jumpers are shown in the next figure.



Shortcut to enable PP1 streamed as Pin1

Shortcut to enable PP3 streamed as Pin1 *(selected as default)*

Shortcut to enable PP0 streamed as Pin0

Shortcut to enable PP2 streamed as Pin0 *(selected as default)*

!! Either PP2 or PP0 are allowed to be shorted to Pin0 at the same time !!

!! Either PP3 or PP1 are allowed to be shorted to Pin1 at the same time !!

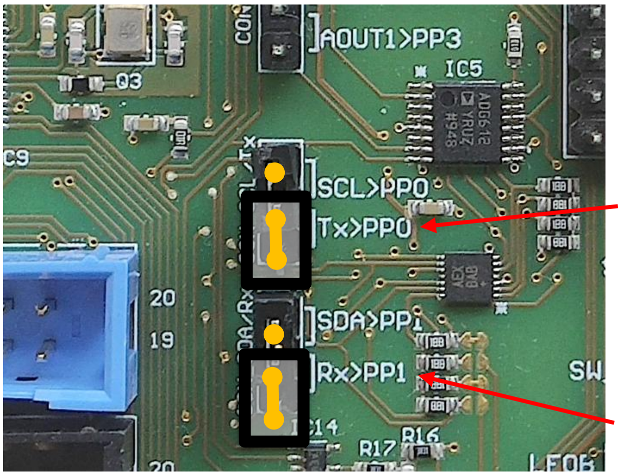
Figure 12 Jumper configuration for using Uart on PP0 and PP1

It is important to consider that only one of the SP49's pins PP0 or PP2 can be shorted at the same time to Pin0. Same is valid for PP1 and PP3. In the picture above this means that either the topmost solder-bridge can be shorted, or the one below, but never both. Similarly, either the lowest solder-bridge can be shorted, or the one above, but never both.

2.1.5 Using Uart on PP0 and PP1

The pins PP0 and PP1 of the SP49 are shared in use, either for the HW-I2C (master or slave) or the SW-Uart. Therefore only one interface can be used at the same time. The I2C-interface is mainly used for programming the SP49 in programming mode. Since this can also be done in debug-mode via SWD using pins PP2 and PP3, it's possible to program the device while using Uart without the need to change any jumpers.

The Uart will be routed to the XMC's Uart-interface, which will forward the received data via a virtual COM-Port to the PC. In order to use the Uart on pins PP0 and PP1 (see also the SP49's software examples), the jumpers must be set as shown in the next figure.



CON_SCL/Tx in position „Tx>PP0"

CON_SDA/Rx in position „Rx>PP1"

Figure 13 Jumper configuration for using Uart on PP0 and PP1

When the SP49 development-board is plugged via USB to a PC, a virtual COM-port will be created. Use any terminal window to connect to this COM-port in order to write and receive data via Uart.

2.1.6 Using I2C on pins PP0 and PP1

The pins PP0 and PP1 of the SP49 are shared in use, either for the HW-I2C (master or slave) or the SW-Uart. Therefore only one interface can be used at the same time. The I2C-interface is mainly used for programming the SP49 in programming mode. Since this can also be done in debug-mode via SWD using pins PP2 and PP3, it's possible to program the device while using Uart without the need to change any jumpers.

The I2C interface between XMC and DUT is only used for programming, where the XMC is the I2C-master and the DUT is the I2C-slave. For programming the DUT in programming-mode then, use the TPMS programming tool and set the jumpers as shown in the next figure.

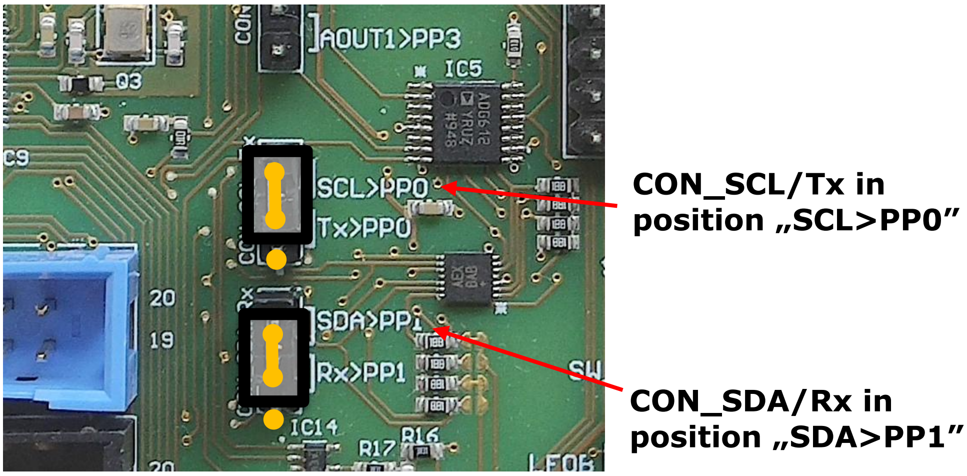

AOUT1>PP3
IC5
Q3
C9
SCL>PP0
Tx>PP0
20
19
SDA>PP1
Rx>PP1
IC14
R17
R16
SW
CON_SCL/Tx in position „SCL>PP0"
CON_SDA/Rx in position „SDA>PP1"

Figure 14 Jumper configuration for using I2C on PP0 and PP1

When the SP49 development-board is plugged via USB to a PC, the TPMS programming tool will communicate via the XMC over I2C with the DUT.

2.1.7 Using PP0 or PP3 for AIN-feature

The SP49 offers a feature called AIN, which can measure an analog voltage applied to either PP0 or PP3. The SP49 development-board offers the possibility to route an analog voltage generated by the XMC towards the SP49, either PP0 or PP3. For routing the analog voltage to one of those pins, refer to the next figure.

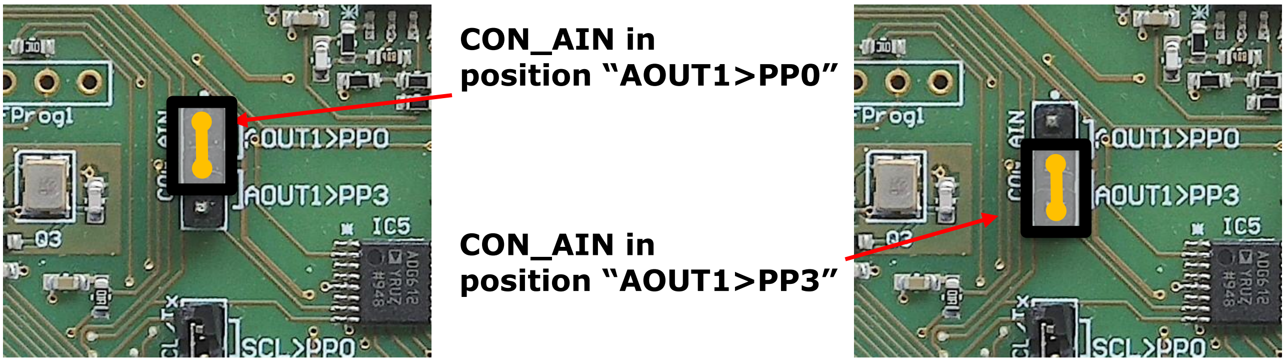

Prog1
CON_AIN
AOUT1>PP0
AOUT1>PP3
IC5
Q3
SCL>PP0
CON_AIN in position "AOUT1>PP0"
CON_AIN in position "AOUT1>PP3"

Figure 15 Jumper configuration for using AIN on PP0 or PP3

Note: When the analog voltage is routed to PP0 or PP3, no other feature must be used on that pin. In detail, if AIN on PP0 is used, no I2C and Uart can be used. If AIN on PP3 is used, no debugging is possible.

2.1.8 Possible combinations of pin-assignments for SP49

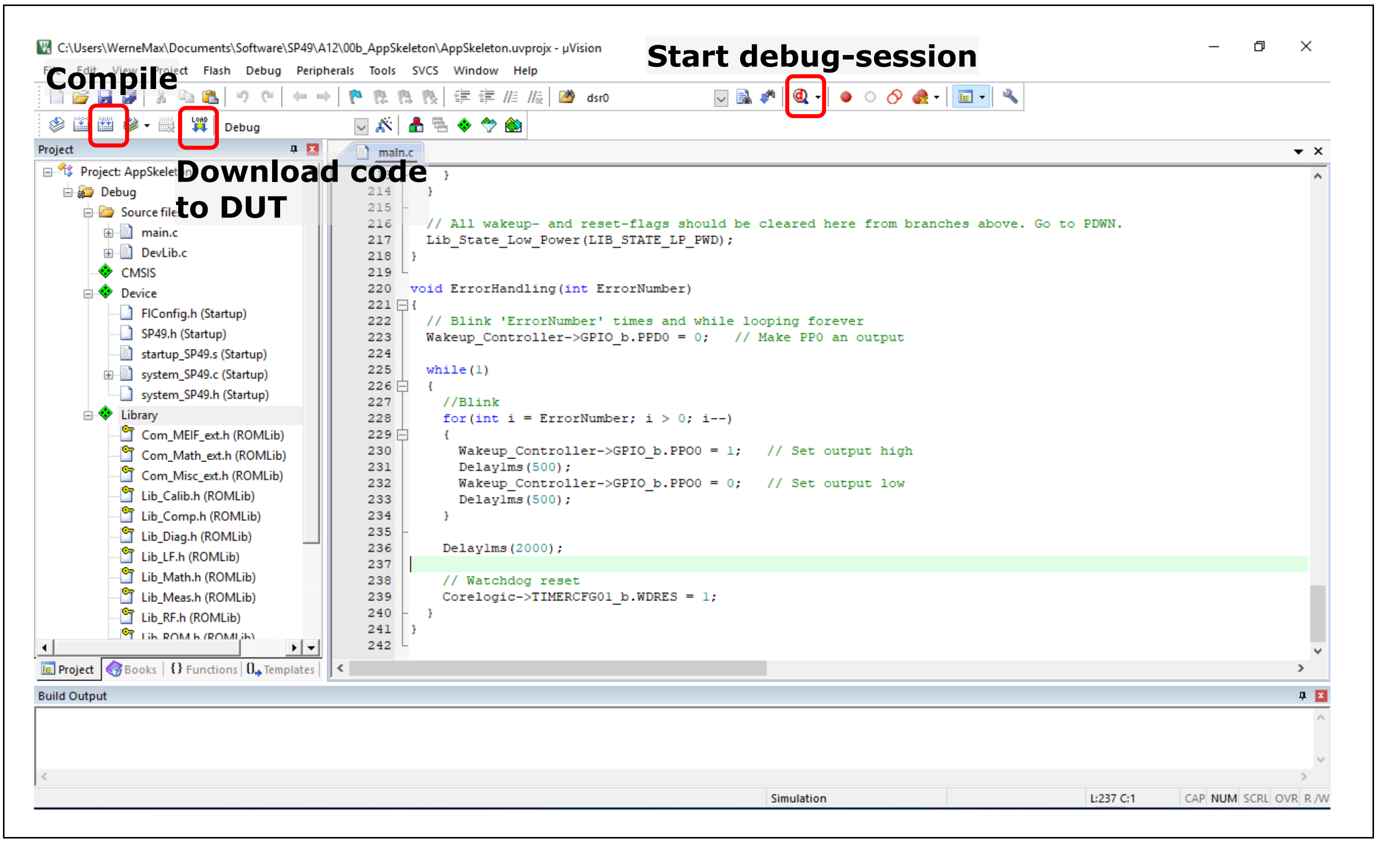
Since the SP49 has only four GPIOs, those pins are shared in their functionality and not every feature can be used at the same time. The following table shall give an idea of possible pin assignment combinations.

Table 2 Possible combinations of pin assignment scenarios for SP49

PP0	PP1	PP2	PP3	Jumpers
I2C (-master or –slave)		Debugging via SWD		CON_SCL/Tx: SCL>PP0 CON_SDA/Rx: SDA>PP1 CON_Ain: *no jumper set*
Uart		Debugging via SWD		CON_SCL/Tx: Tx>PP0 CON_SDA/Rx: Rx>PP1 CON_Ain: *no jumper set*
AIN	Uart (SP49 Tx only)	Debugging via SWD		CON_SCL/Tx: *no jumper set* CON_SDA/Rx: Rx>PP1 CON_Ain: AOUT1>PP0
I2C (-master or –slave)		*Free*	AIN	CON_SCL/Tx: SCL>PP0 CON_SDA/Rx: SDA>PP1 CON_Ain: AOUT1>PP3
Uart		*Free*	AIN	CON_SCL/Tx: Tx>PP0 CON_SDA/Rx: Rx>PP1 CON_Ain: AOUT1>PP3

2.2 Debugging with Keil environment

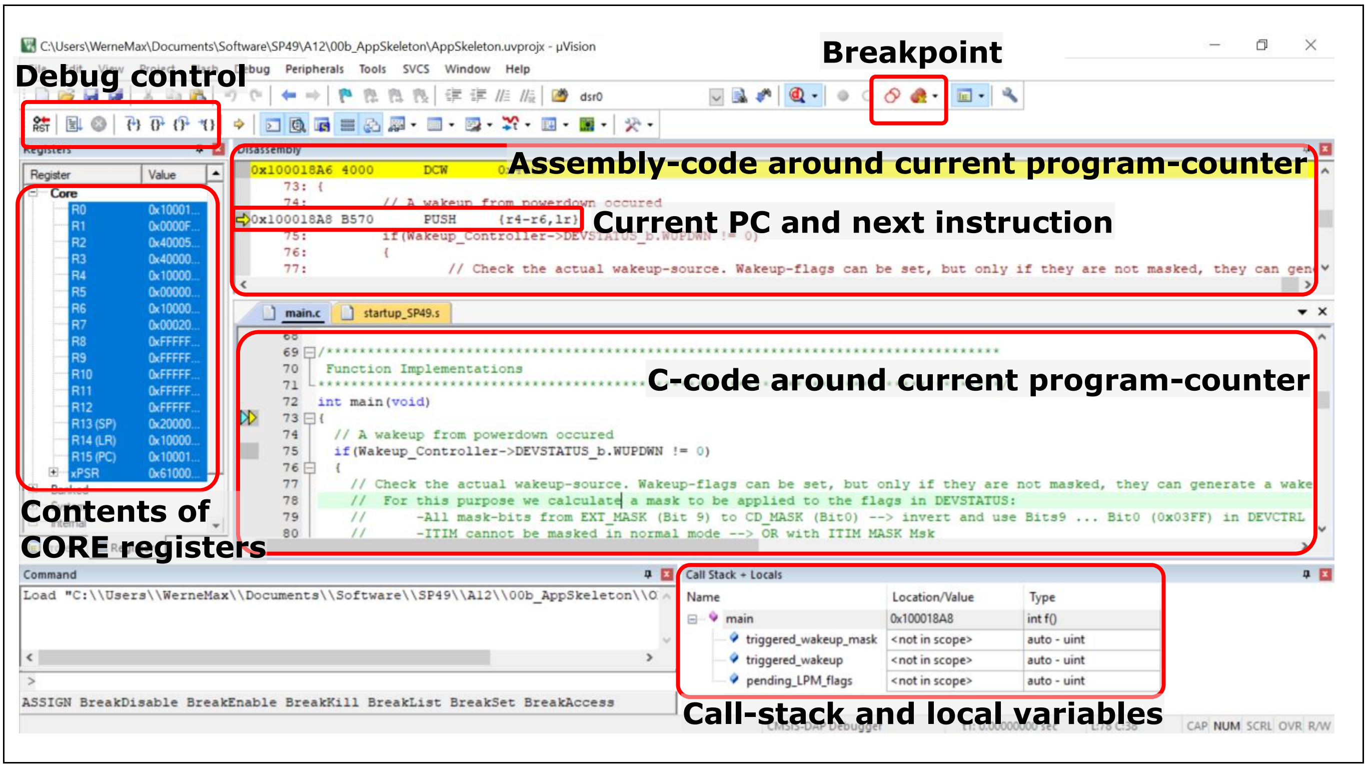
When Keil was started and a project for SP49 was loaded (for how to, see chapter 3.1), the window should look like this (main.c file opened):



```
  }
}

  // All wakeup- and reset-flags should be cleared here from branches above. Go to PDWN.
  Lib_State_Low_Power(LIB_STATE_LP_PWD);
}

void ErrorHandling(int ErrorNumber)
{
  // Blink 'ErrorNumber' times and while looping forever
  Wakeup_Controller->GPIO_b.PPD0 = 0;   // Make PP0 an output

  while(1)
  {
    //Blink
    for(int i = ErrorNumber; i > 0; i--)
    {
      Wakeup_Controller->GPIO_b.PPO0 = 1;   // Set output high
      Delay1ms(500);
      Wakeup_Controller->GPIO_b.PPO0 = 0;   // Set output low
      Delay1ms(500);
    }

    Delay1ms(2000);

    // Watchdog reset
    Corelogic->TIMERCFG01_b.WDRES = 1;
  }
}
```

Figure 16 Keil window with an open project in development view

When writing code is finished and the compilation was successful, a debug-session can be started. Therefore just click on the start debug-session button. After a short initialization, and maybe downloading the updated code, the Keil window is now in debug view and should look like the following figure.


C:\Users\WerneMax\Documents\Software\SP49\A12\00b_AppSkeleton\AppSkeleton.uvprojx - µVision
Debug control
Breakpoint
Debug
Peripherals
Tools
SVCS
Window
Help
dsr0
Registers
Disassembly
Assembly-code around current program-counter
Current PC and next instruction
main.c
startup_SP49.s
C-code around current program-counter
Contents of CORE registers
Command
Call Stack + Locals
Call-stack and local variables
CMSIS-DAP Debugger
CAP NUM SCRL OVR R/W

Register	Value
Core	
R0	0x10001...
R1	0x0000F...
R2	0x40005...
R3	0x40000...
R4	0x10000...
R5	0x00000...
R6	0x10000...
R7	0x00020...
R8	0xFFFFF...
R9	0xFFFFF...
R10	0xFFFFF...
R11	0xFFFFF...
R12	0xFFFFF...
R13 (SP)	0x20000...
R14 (LR)	0x10000...
R15 (PC)	0x10001...
xPSR	0x61000...

```
0x100018A6 4000      DCW      0x
    73: {
    74:         // A wakeup from powerdown occured
0x100018A8 B570      PUSH     {r4-r6,lr}
    75:         if(Wakeup_Controller->DEVSTATUS_b.WUPDWN != 0)
    76:         {
    77:             // Check the actual wakeup-source. Wakeup-flags can be set, but only if they are not masked, they can gen
```

```
/*******************************************************************************
 Function Implementations
********************************************************************************
int main(void)
{
  // A wakeup from powerdown occured
  if(Wakeup_Controller->DEVSTATUS_b.WUPDWN != 0)
  {
    // Check the actual wakeup-source. Wakeup-flags can be set, but only if they are not masked, they can generate a wake
    //  For this purpose we calculate a mask to be applied to the flags in DEVSTATUS:
    //      -All mask-bits from EXT_MASK (Bit 9) to CD_MASK (Bit0) --> invert and use Bits9 ... Bit0 (0x03FF) in DEVCTRL
    //      -ITIM cannot be masked in normal mode --> OR with ITIM MASK Msk
```

```
Load "C:\\Users\\WerneMax\\Documents\\Software\\SP49\\A12\\00b_AppSkeleton\\O
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess
```

Name	Location/Value	Type
main	0x100018A8	int f()
triggered_wakeup_mask	<not in scope>	auto - uint
triggered_wakeup	<not in scope>	auto - uint
pending_LPM_flags	<not in scope>	auto - uint

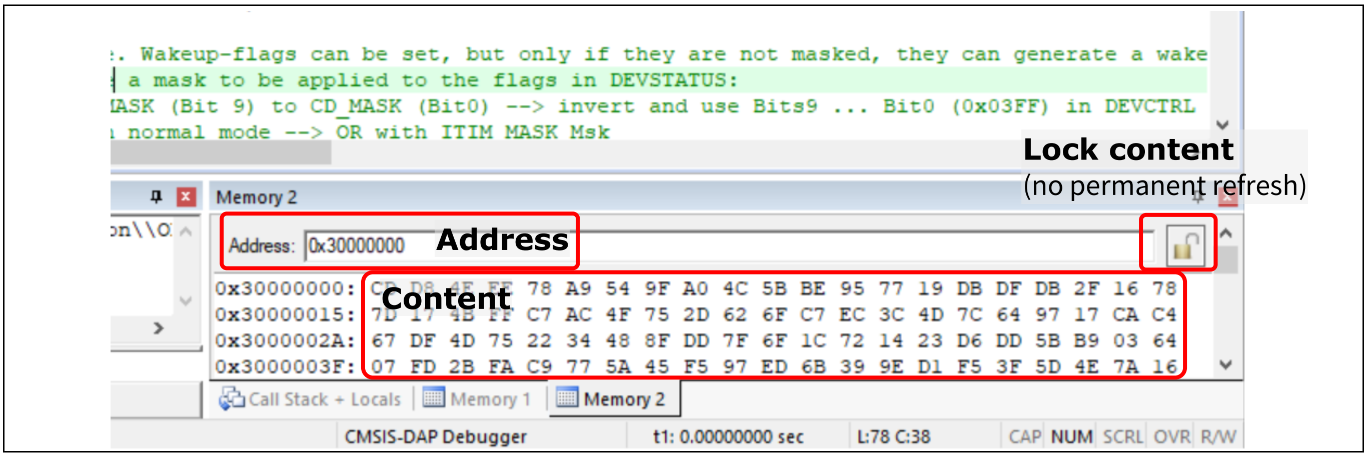
Via the buttons in debug control step-in, -over and –out can be done. A click on “Run” let’s the core run until a breakpoint or watchpoint is hit. With Reset a power-on-reset can be performed and the debug-session restarts.

Some helpful tricks shall be explained more in detail:

- How to see / modify content in RAM and retention RAM
- How to see / modify contents of SFRs
- How to use breakpoints and watchpoints
- How to watch variables / structures / arrays

2.2.1 How to see / modify content in RAM and retention RAM

In order to see / modify the content in RAM and retention RAM, the memory window in Keil will be used. In Keil’s top menu click on “View” -> “Memory Windows” -> “Memory 1”. This will add a tab on the bottom right. In address enter the address of RAM, e.g. 0x20000200. A second window “Memory 2” will be added the same way, but here we enter the address of retention RAM, e.g. 0x30000000. The Keil window displaying the content of retention RAM is shown in the next figure.



```
:. Wakeup-flags can be set, but only if they are not masked, they can generate a wake
 a mask to be applied to the flags in DEVSTATUS:
IASK (Bit 9) to CD_MASK (Bit0) --> invert and use Bits9 ... Bit0 (0x03FF) in DEVCTRL
ı normal mode --> OR with ITIM MASK Msk
```

Address: 0x30000000

```
0x30000000: CD D8 4E FE 78 A9 54 9F A0 4C 5B BE 95 77 19 DB DF DB 2F 16 78
0x30000015: 7D 17 4B FF C7 AC 4F 75 2D 62 6F C7 EC 3C 4D 7C 64 97 17 CA C4
0x3000002A: 67 DF 4D 75 22 34 48 8F DD 7F 6F 1C 72 14 23 D6 DD 5B B9 03 64
0x3000003F: 07 FD 2B FA C9 77 5A 45 F5 97 ED 6B 39 9E D1 F5 3F 5D 4E 7A 16
```

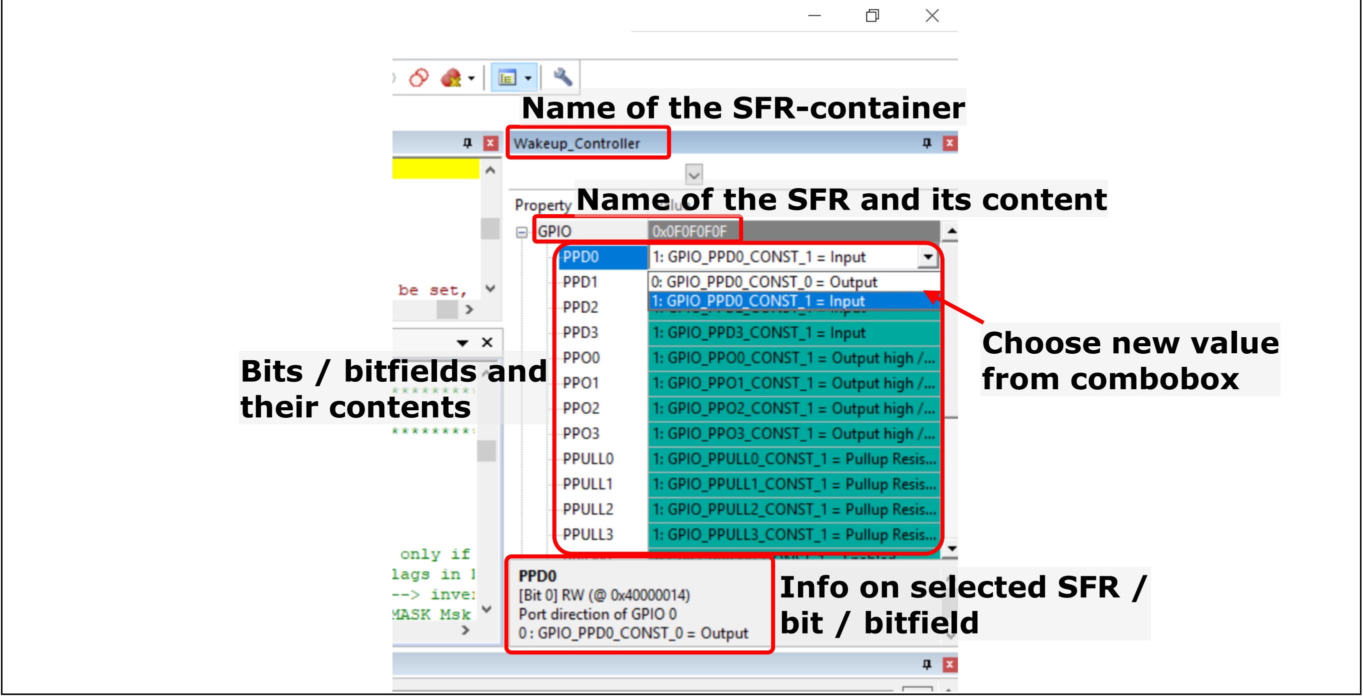
CMSIS-DAP Debugger t1: 0.00000000 sec L:78 C:38

Figure 17 Memory window in Keil displaying retention RAM content

The display format by default is 8-Bit hex and can be changed with a right-click into the content and then selecting “Decimal”. 8-, 16- or 32-Bit signed or unsigned formats can be chosen in addition. To modify a location simply double-click on the value and enter the new one.

2.2.2 How to see / modify contents of SFRs

All SFRs are known by Keil through the SP49 Keil Pack and the displaying window can be opened from the top menu in “Peripherals” -> “System Viewer”. Here each logical block of SP49 is listed. As example, click on “Wakeup_Controller” will open a new window on the right displaying all SFRs of the wakeup-controller and their contents. Each SFR, called “Property” here, can be expanded to display even its bits / bitfields and their values. By a click on the content it is possible to enter a new desired value. As example for the wakeup-controller’s SFR called GPIO, this is shown in the next figure.


Name of the SFR-container
Wakeup_Controller
Property
Name of the SFR and its content
Bits / bitfields and their contents
Choose new value from combobox
Info on selected SFR / bit / bitfield

GPIO	0x0F0F0F0F
PPD0	1: GPIO_PPD0_CONST_1 = Input
PPD1	0: GPIO_PPD0_CONST_0 = Output
PPD2	1: GPIO_PPD0_CONST_1 = Input
PPD3	1: GPIO_PPD3_CONST_1 = Input
PPO0	1: GPIO_PPO0_CONST_1 = Output high /...
PPO1	1: GPIO_PPO1_CONST_1 = Output high /...
PPO2	1: GPIO_PPO2_CONST_1 = Output high /...
PPO3	1: GPIO_PPO3_CONST_1 = Output high /...
PPULL0	1: GPIO_PPULL0_CONST_1 = Pullup Resis...
PPULL1	1: GPIO_PPULL1_CONST_1 = Pullup Resis...
PPULL2	1: GPIO_PPULL2_CONST_1 = Pullup Resis...
PPULL3	1: GPIO_PPULL3_CONST_1 = Pullup Resis...

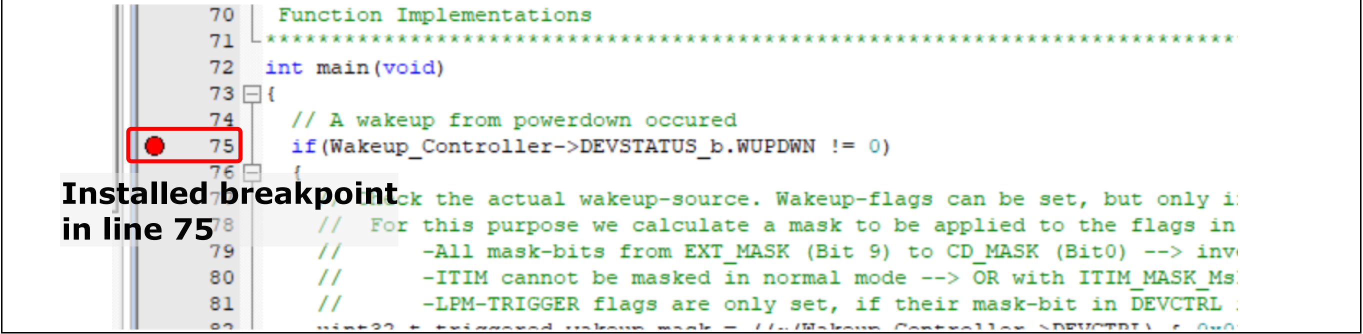
PPD0
[Bit 0] RW (@ 0x40000014)
Port direction of GPIO 0
0 : GPIO_PPD0_CONST_0 = Output

Figure 18 Display and modification of SFR contents

2.2.3 How to use breakpoints and watchpoints

Breakpoints are places in code where execution can be halted, while watchpoints observe accesses to a certain address and can lead to a halt when a certain access to that address is executed by the CPU. The SP49 supports 4 breakpoints and 2 watchpoints.

To set a breakpoint, go to the desired line in your assembly- or c-code-file and make a single left-mouse click on the left of the line-number in order to set a breakpoint here. When the breakpoint is set, this should look as shown in the next figure.

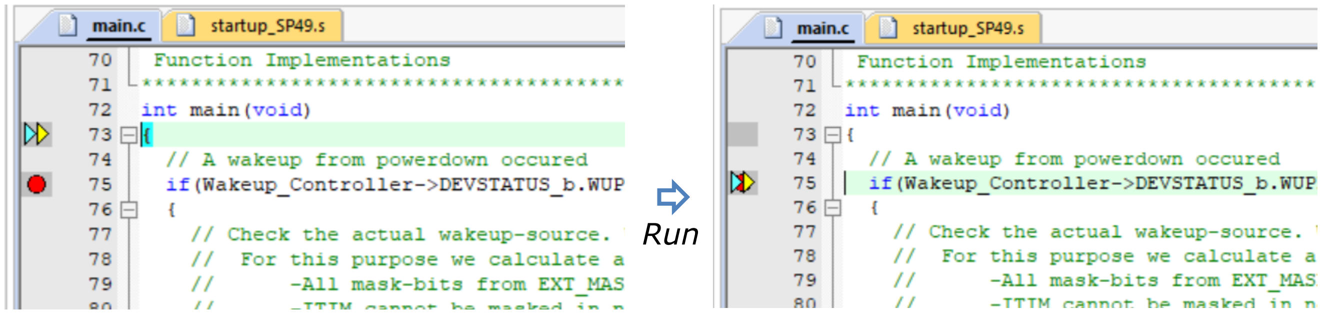


```
70  Function Implementations
71 *********************************************************************************
72 int main(void)
73 {
74   // A wakeup from powerdown occured
75   if(Wakeup_Controller->DEVSTATUS_b.WUPDWN != 0)
76   {
77         ck the actual wakeup-source. Wakeup-flags can be set, but only i
78     //  For this purpose we calculate a mask to be applied to the flags in
79     //      -All mask-bits from EXT_MASK (Bit 9) to CD_MASK (Bit0) --> inv
80     //      -ITIM cannot be masked in normal mode --> OR with ITIM_MASK_Ms
81     //      -LPM-TRIGGER flags are only set, if their mask-bit in DEVCTRL
```

Figure 19 Breakpoint set in a c-code file

Note: *A breakpoint can only work if set to code, which can be executed. Setting a breakpoint inside a comment, for example, will be ignored by Keil during debugging.*

After starting a debug-session, the CPU will be halted at the beginning of the code, as shown below on the left. When the breakpoint is installed you can click on “Run” and the CPU will run until it reaches the breakpoint, as shown below on the right.



main.c startup_SP49.s

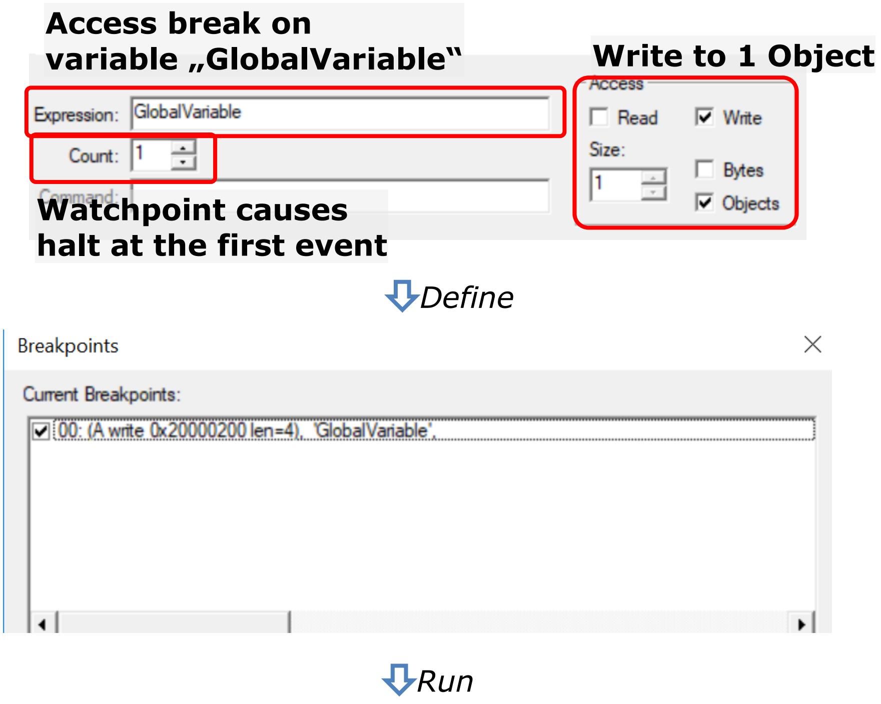
```
  Function Implementations
****************************************
int main(void)
{
  // A wakeup from powerdown occured
  if(Wakeup_Controller->DEVSTATUS_b.WUP
  {
    // Check the actual wakeup-source.
    //  For this purpose we calculate a
    //       -All mask-bits from EXT_MAS
    //       -ITIM cannot be masked in n
```

1) CPU halted after reset

2) CPU halted at breakpoint

A watchpoint can be inserted in the top menu in “Debug” -> “Breakpoints…”. This window shows all currently installed break- and watchpoints. To install a new watchpoint, enter an expression and choose the access-type and -size. A watchpoint can cause one of three breaks: access break, conditional break or execution break. After a click on “Define” the watchpoint is set and active.

- An access break simply halts the CPU whenever an access (can be read, write or both) happens to an address or global variable.
  - Example 1: Expression=”0x20000200”, Read, Write, 4 bytes. This leads to a halt when one of the bytes within [0x20000200 … 0x20000203] are read or written.
  - Example 2: Expression=”GlobalVariable”, Write, 1 Object. This leads to a halt when the variable named “GlobalVariable” is written. This is shown in the next figure.



Access break on variable „GlobalVariable“

Write to 1 Object

Expression: GlobalVariable

Count: 1

Command:

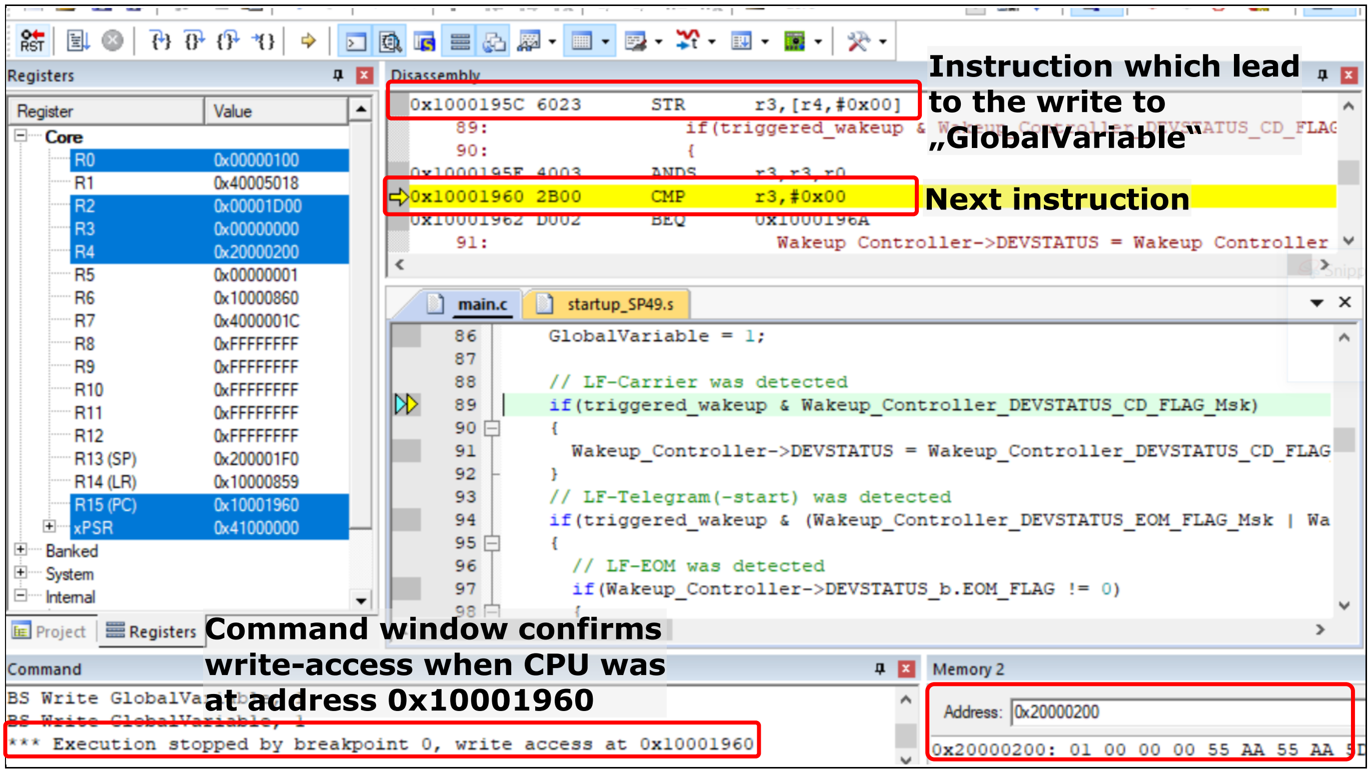
Access
☐ Read ☑ Write
Size: 1
☐ Bytes
☑ Objects

Watchpoint causes halt at the first event

Breakpoints

Current Breakpoints:

☑ 00: (A write 0x20000200 len=4), 'GlobalVariable',



Register	Value
Core	
R0	0x00000100
R1	0x40005018
R2	0x00001D00
R3	0x00000000
R4	0x20000200
R5	0x00000001
R6	0x10000860
R7	0x4000001C
R8	0xFFFFFFFF
R9	0xFFFFFFFF
R10	0xFFFFFFFF
R11	0xFFFFFFFF
R12	0xFFFFFFFF
R13 (SP)	0x200001F0
R14 (LR)	0x10000859
R15 (PC)	0x10001960
xPSR	0x41000000
Banked	
System	
Internal	

```
0x1000195C 6023      STR      r3,[r4,#0x00]
    89:                  if(triggered_wakeup & Wakeup_Controller_DEVSTATUS_CD_FLAG
    90:                  {
0x1000195E 4003      ANDS     r3,r3,r0
0x10001960 2B00      CMP      r3,#0x00
0x10001962 D002      BEQ      0x1000196A
    91:                    Wakeup Controller->DEVSTATUS = Wakeup Controller
```

Instruction which lead to the write to „GlobalVariable“

Next instruction

```
    GlobalVariable = 1;

    // LF-Carrier was detected
    if(triggered_wakeup & Wakeup_Controller_DEVSTATUS_CD_FLAG_Msk)
    {
      Wakeup_Controller->DEVSTATUS = Wakeup_Controller_DEVSTATUS_CD_FLAG
    }
    // LF-Telegram(-start) was detected
    if(triggered_wakeup & (Wakeup_Controller_DEVSTATUS_EOM_FLAG_Msk | Wa
    {
      // LF-EOM was detected
      if(Wakeup_Controller->DEVSTATUS_b.EOM_FLAG != 0)
      {
```

Command window confirms write-access when CPU was at address 0x10001960

```
BS Write GlobalVariable
BS Write GlobalVariable, 1
*** Execution stopped by breakpoint 0, write access at 0x10001960
```

```
Address: 0x20000200
0x20000200: 01 00 00 00 55 AA 55 AA 5D
```

Figure 20 Access watchpoint on a global variable example

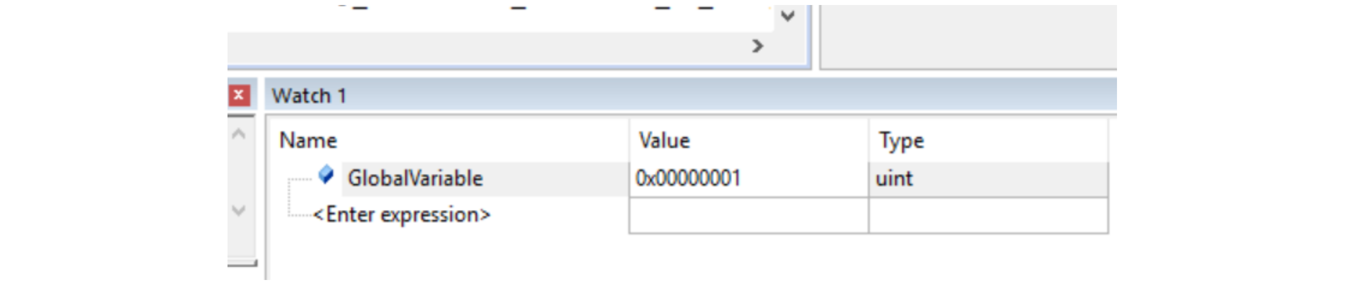
Note: *The halt of the CPU happens few instructions after the actual access-instruction to the target-address. The reason is the mechanism how the Cortex-Core buffers memory accesses and cannot be avoided.*

- A conditional break causes the CPU to halt when a condition is fulfilled, for example when the content at an address or of a variable holds a certain value.
  - Example: Expression=”GlobalVariable==1”, Read, Write, 1 Object
- The execution break causes the CPU to halt when a certain function is executed.
  - Example: Expression=”main”

Further information can be found in the uVision User’s Guide the Keil homepage:
http://www.keil.com/support/man/docs/uv4/uv4_db_dbg_breakpnts.htm

2.2.4 How to watch variables / structures / arrays

The content of local and global variables can be checked in Keil when a debug session is running. For single variables it is possible to hover over it with the mouse. Keil will then display the content of this variable. Furthermore it is possible to add variables to a watch-window. Therefore right-click on a variable and click on “Add ‘*VariableName*’ to …” -> “Watch 1”. This will open a new window “Watch 1”, where the variable can be always observed. This is shown in the next figure as example with a variable called “GlobalVariable”.



Watch 1

Name	Value	Type
GlobalVariable	0x00000001	uint
<Enter expression>		

Figure 21 **Watch window in Keil showing a global variable**

Note: *Local variables sometimes may not be displayed. Reason is that the compiler might not assign static addresses to it, but just temporarily uses one of the Core-registers. The local variable will only be visible within its scope then.*

Structures and arrays can also be added to a watch window. Those datatypes can be extended by a click on “+”, so all their members and contents will be shown then.

2.3 SP49 development-board GUI

This chapter explains the features of the SP49 development-board GUI and how to use them.

2.3.1 Getting started and Board-FW-update

For using this GUI, it is required that a SP49 development-board is connected to this PC via USB, according to the following figure.

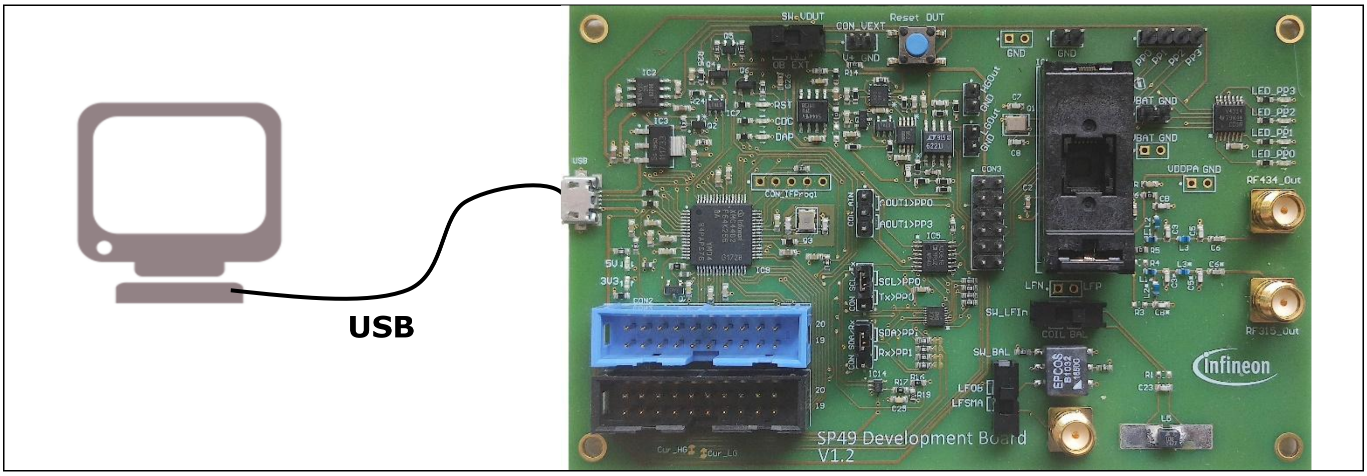
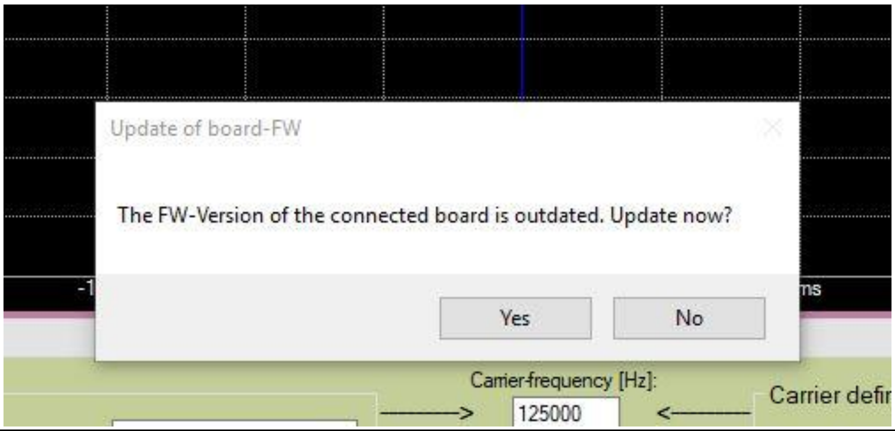


Figure 22 SP49 development-board setup

When the GUI starts up, it should automatically detect the board and connect to it. In case the firmware of the board is outdated, the GUI will ask whether it shall be updated. This should be confirmed with “Yes” as shown below.

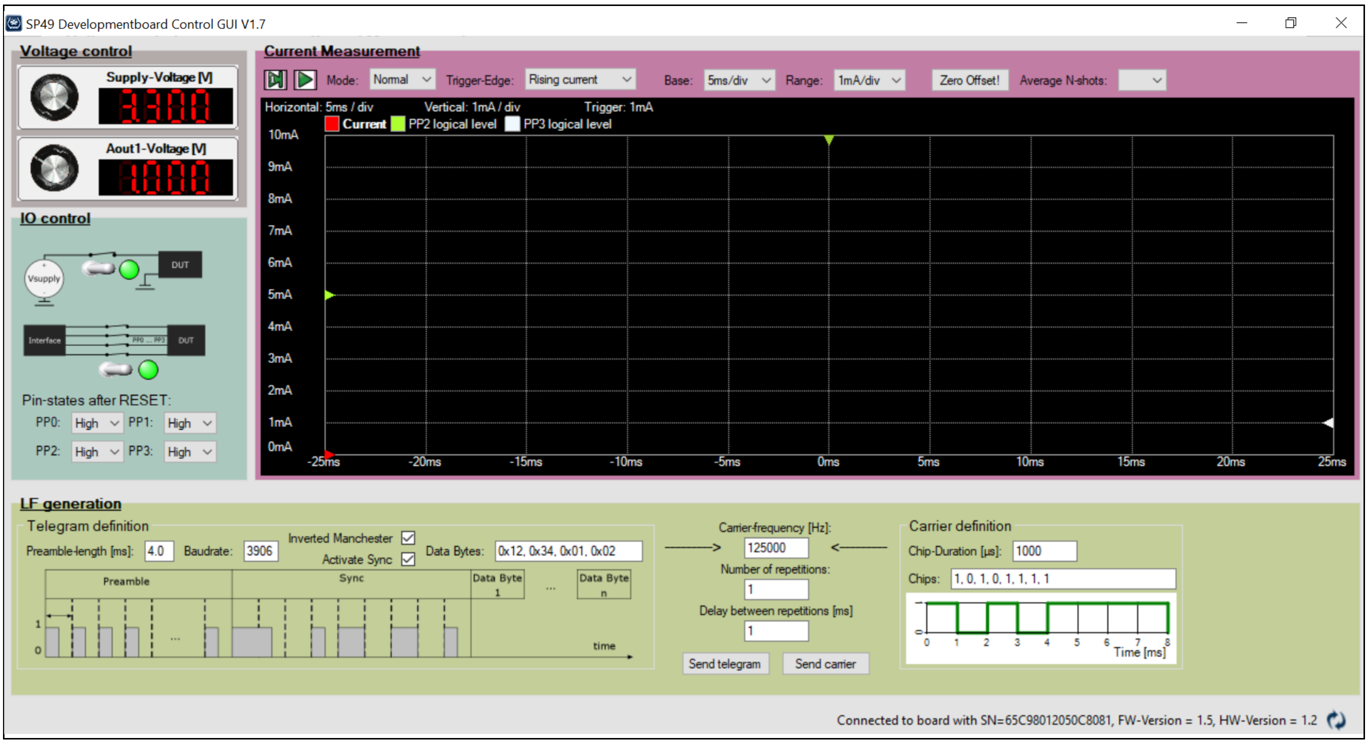


Update of board-FW

The FW-Version of the connected board is outdated. Update now?

Figure 23 Update of board-FW

After a click on “Yes” the updated board FW will be downloaded to the board. Afterwards the GUI will show up as follows.



SP49 Developmentboard Control GUI V1.7

Voltage control

Supply-Voltage [V]: 3.300

Aout1-Voltage [V]: 1.000

IO control

Pin-states after RESET:

PP0: High PP1: High

PP2: High PP3: High

Current Measurement

Mode: Normal Trigger-Edge: Rising current Base: 5ms/div Range: 1mA/div Average N-shots:

Horizontal: 5ms / div Vertical: 1mA / div Trigger: 1mA

LF generation

Telegram definition

Preamble-length [ms]: 4.0 Baudrate: 3906 Inverted Manchester ☑ Activate Sync ☑ Data Bytes: 0x12, 0x34, 0x01, 0x02

Carrier-frequency [Hz]: 125000

Number of repetitions: 1

Delay between repetitions [ms] 1

Carrier definition

Chip-Duration [µs]: 1000

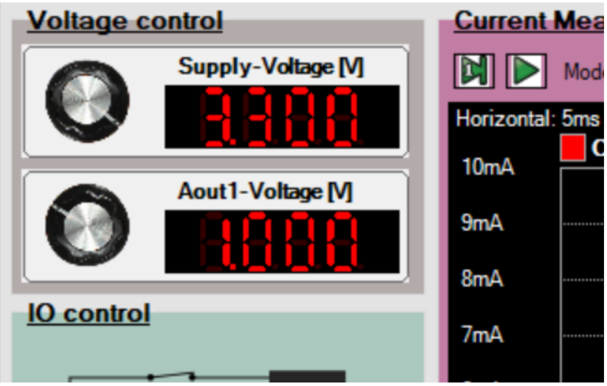
Chips: 1, 0, 1, 0, 1, 1, 1, 1

Connected to board with SN=65C98012050C8081, FW-Version = 1.5, HW-Version = 1.2

Figure 24 SP49 development-board GUI complete window

2.3.2 Using the voltage controls

The GUI provides two voltage controls. One for setting the supply-voltage of the DUT on the boards and another for setting an analog voltage towards the DUT, which can be routed to PP0 or PP3 of the DUT via an on-board jumper. The voltage controls are shown below.



Voltage control

Supply-Voltage [V]: 3.300

Aout1-Voltage [V]: 1.000

Figure 25 Voltage controls of the SP49 development-board GUI

Supply-voltage is the on-board generated voltage, see chapter 2.1.1. Aout1-voltage is the analog-voltage which can be routed to SP49's PP0 or PP3, see chapter 2.1.7. To set one of the voltages, either:

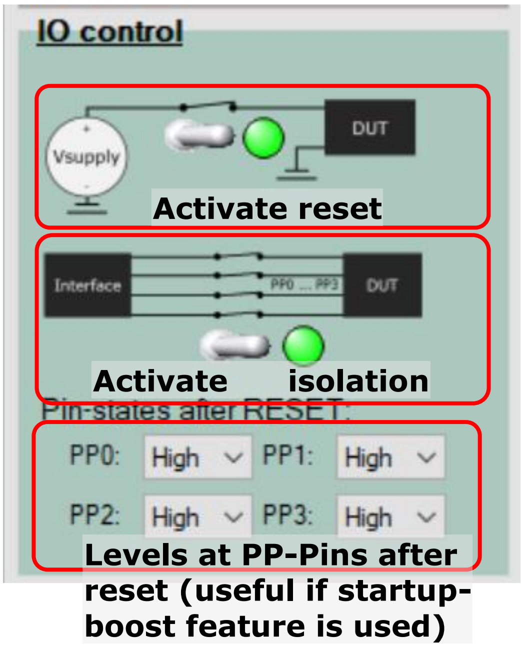
- Move your mouse over one of the controls' knob and use the mouse-wheel to turn the value up or down
- Or move your mouse over one of the knobs, hold down the left mouse-button and move the knob clock- or counter-clockwise

2.3.3 Using the IO controls

There are three features under "IO controls", which can be used:

- The first one is to activate a reset of the DUT. This is achieved by disconnecting the supply voltage.
- The second one is to activate isolation on all digital pins between the DUT and the rest of the board. This can be used to ensure that no load is connected on the DUT's PP-pins such that precise current-measurements can be performed.
- The third feature is to select the logical levels which will be applied to the four PP-Pins of the DUT after reset-release. Those levels will be hold after reset-release until the DUT has finished the boot process in order to ensure that the mode-selection is not skipped (see "start-up boost" feature of SP49).

The IO controls are shown in the next figure.



PP0: High PP1: High
PP2: High PP3: High

Figure 26 **IO controls of the SP49 development-board GUI**

2.3.4 Using the current measurement

The SP49 development-board supports onboard current-measurement and streaming the results via USB to the development-board GUI. The sampling-frequency is limited mainly by the USB connection, see also Table 1. There the data can be visualized on the right side, as shown in the following figure.

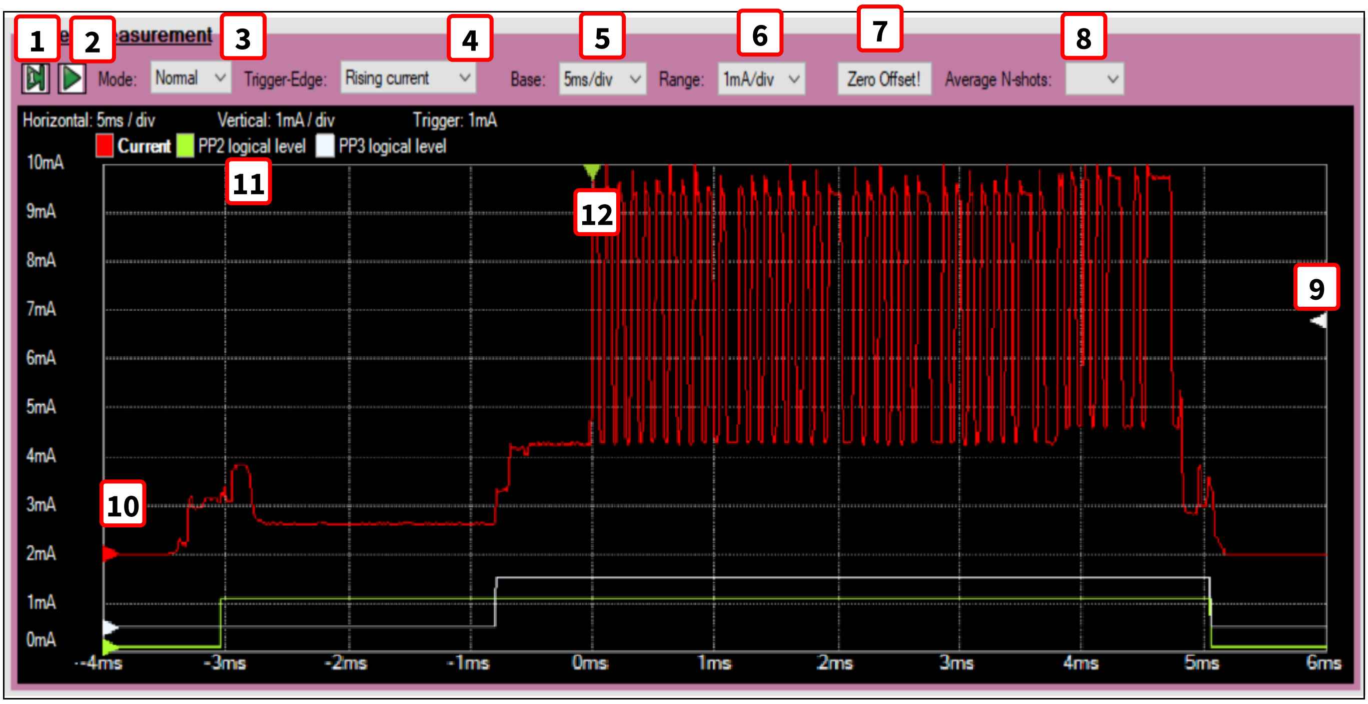


Figure 27 **Current measurement plot of the SP49 development-board GUI**

The following settings can be made for current-measurement (see numbers in figure above):

1. Single-shot: Make a single-shot measurement in the currently configured acquisition-mode (see 3).
2. Run: Run measurement in continuous mode in the currently configured acquisition-mode (see 3). When the time base is high enough a continuous stream will be displayed, where a vertical blue line shows the actual time of acquisition.
3. Acquisition-mode: Configure to either "None" or "Normal". "None" means that no trigger is used, therefore current is continuously measured and streamed to the PC. The maximum sampling-frequency of the tool is 33 kHz. "Normal" means that a trigger is used. The trigger can be set to a certain voltage-level (see 8) and an edge (rising or falling or both) can be selected (see 4).
4. Trigger-Edge: If acquisition-mode "Normal" is selected, then it is possible to choose here the edge, either rising or falling.
5. Base: Set here the time-base, meaning the amount of time per horizontal division.
6. Range: Set here the range, meaning the amount of measurement unit per vertical division.
7. Zero Offset: This button can be used to set the offset of the current measurement to zero. Note that for this any load on the board, especially the DUT, must be removed first.
8. Average N-shots: This option can be changed when Mode “Normal” is active. In this case, multiple shots, or triggers, will be averaged and the averaged results will be displayed. This can be usefull to get a more noise-free result for measurement of very low currents.
9. Trigger-level: This is the levels where in "Normal" acquisition-mode a trigger will be generated, depending on the selected trigger-edge (see 4). You can use the left mouse-button or the mouse-wheel (mouse positioned on the right of the plot) to move the level.
10. Vertical 0-level: This place where the 0-level is plotted can be moved. Use the left mouse-button or the mouse-wheel (mouse positioned on the left of the plot) to move the 0-level. By dragging the arrow, the respective trace (current or PPx-level) can be directly moved. When moving up/down on the y-axis scale on the left, only the currently active trace (indicated in bold on top) is moved.

11. Activation and selection of trace: Here the current-trace and the PPx-level-traces are shown. The one plotted in bold is the currently selected one and will be moved vertically, see point 9. Select a trace by a single left-mouse-click on it. Also it is possible to activate or deactivate each trace by a left-mouse-click inside the rectangle. This rectangle when filled with color, its trace will be shown in the plot below. If none colored, the respective trace is also not shown in the trace below.
12. Horizontal 0-level: Indicates t=0. Can be shifted horizontally by holding down the left mouse button inside the plot and moving the mouse left or right.

The current-measurement plot also support to use cursors for evaluation of measurement results. A first cursor can be set by double-click with the left mouse-button anywhere inside the plot. A first cursor will be added, which can be freely moved horizontally in the plot, as shown below. While the cursors is moved, the current x- and y-value are plotted.

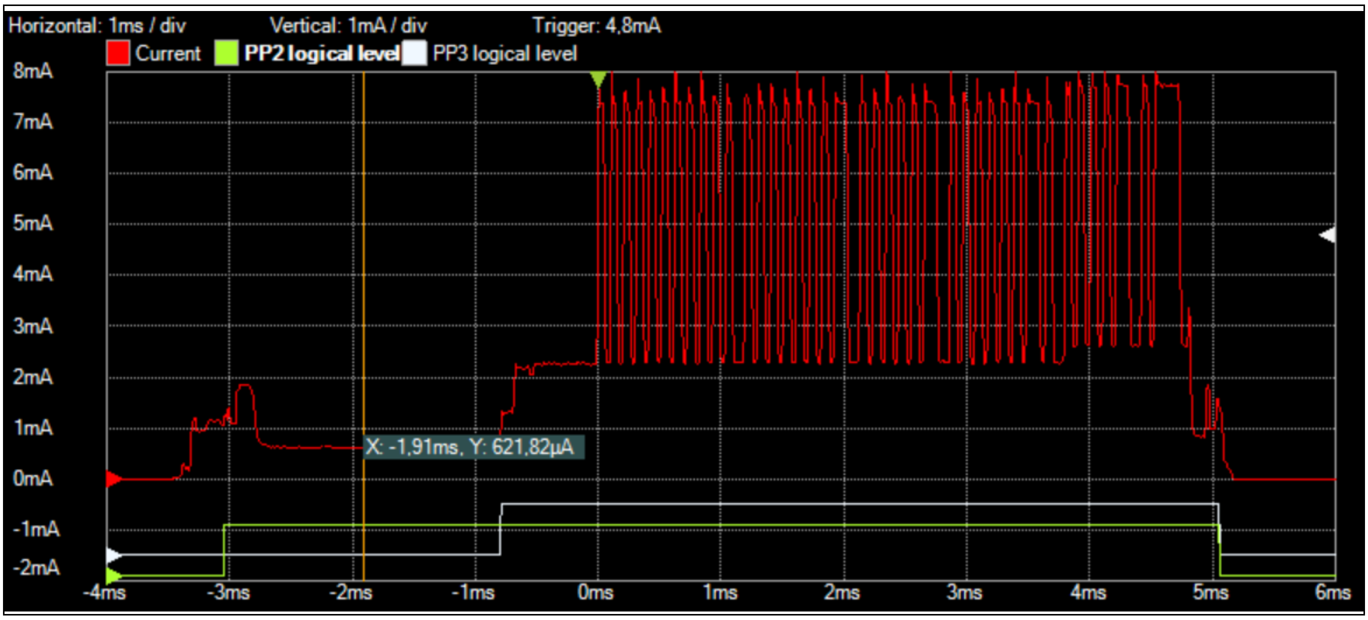


Figure 28 One cursor added in current measurement plot of the SP49 development-board GUI

When the desired position for the first cursor is found, it can be fixed by another single left mouse-button click. Consequently a second cursor will appear. It is now possible to freely move the second cursor horizontally by moving the mouse within the plot. During movement of the mouse as well the current x- and y-values at the second cursor are plotted, as also the difference between the first and second curst. Also a charge between cursor 1 and cursor 2 are calculated and plotted. The second cursor is shown in the following figure.

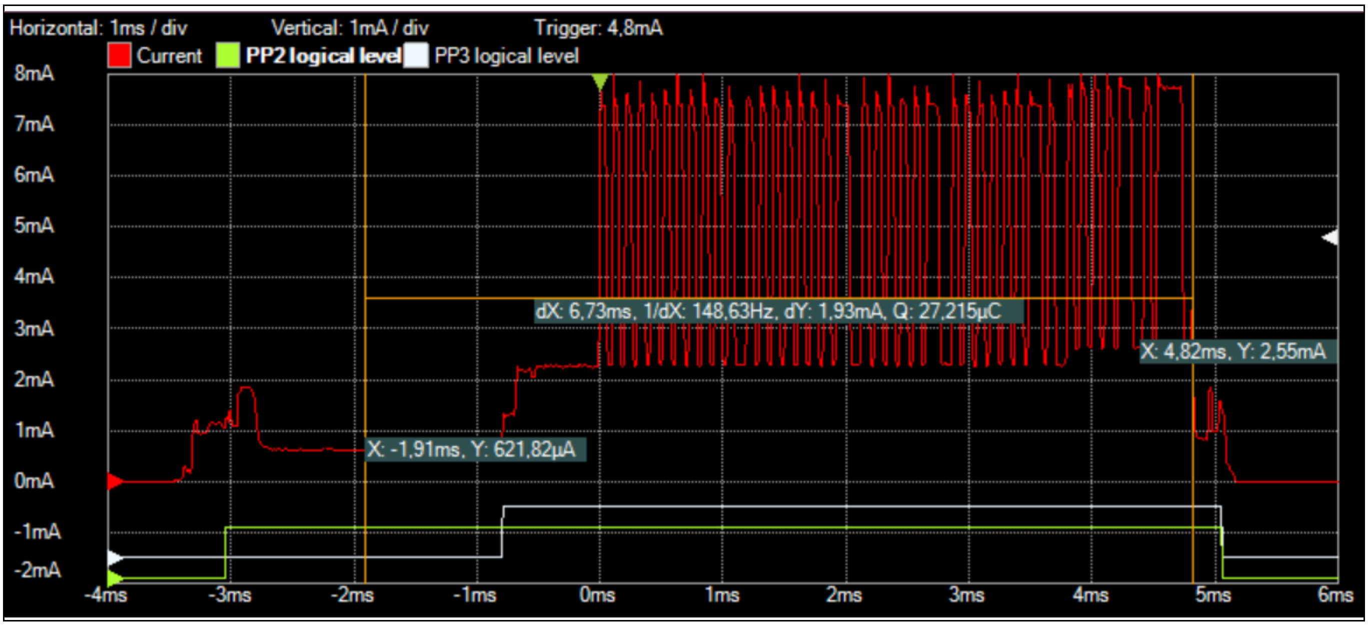
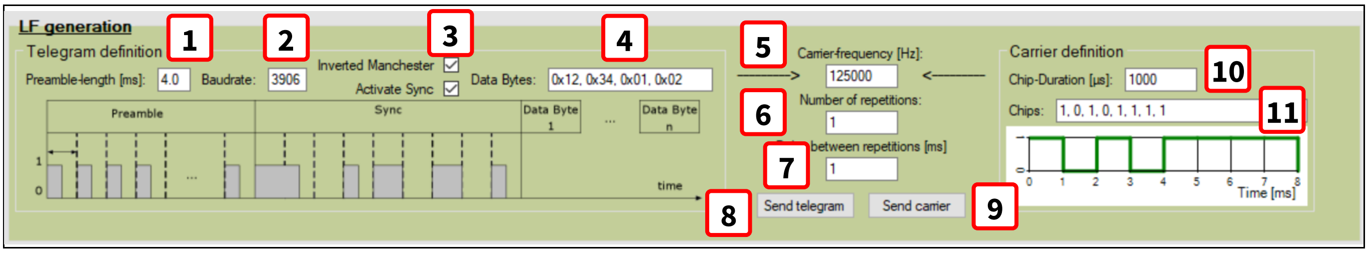


Figure 29 Second cursor added in current measurement plot of the SP49 development-board GUI

Also the second cursor can be fixed by another left mouse-button click. The mouse can then be freely moved without changing the values at first and second cursors and the calculated differences between them.

2.3.5 Using the LF generator

The SP49 development-board also supports generation of LF-carrier and telegram. The GUI can be used to send one or the other and the control for this is shown below.


LF generation

Telegram definition

1 Preamble-length [ms]: 4.0

2 Baudrate: 3906

3 Inverted Manchester ☑

Activate Sync ☑

4 Data Bytes: 0x12, 0x34, 0x01, 0x02

5 Carrier-frequency [Hz]: 125000

6 Number of repetitions: 1

7 between repetitions [ms]: 1

Carrier definition

10 Chip-Duration [µs]: 1000

11 Chips: 1, 0, 1, 0, 1, 1, 1, 1

Figure 30 LF generator of the SP49 development-board GUI

From this part of the GUI, it is possible to either generate a LF-telegram or a LF-carrier. To generate a LF-telegram, the following steps have to be done:

1. Define desired length in [ms] of the preamble in the input marked with (1). The preamble will be 0-1-chips for maximum the given duration.
2. Configure desired baud rate of the data in [Bit/s] in the input marked with (2).
3. Choose whether Manchester- or inverted Manchester-coding shall be applied for the data in the checkbox below the mark (4). Check whether the (fixed) sync-pattern shall be sent before the data bytes in the checkbox below.
4. Enter the payload as hex-values or binary-value, or even a mix of them, each separated by ',' in the input marked with (4). For example: '0x01, 0x02, 0b1010, 0x01, 0b11'.

To define a (pulsed) LF-carrier, follow those steps:

1. Enter the desired duration in [us] of one chip into the input marked with (10)

2. Enter a chip or a sequence of chips, by either '1' (carrier active) or '0' (carrier inactive). Each chip is to be separated by ','. For example: '1, 0, 0, 1', which will generate a sequence of Carrier-Active, Carrier-Inactive, Carrier-Inactive, Carrier-Active (each chip is as long as defined by "Chip-Duration [us]").

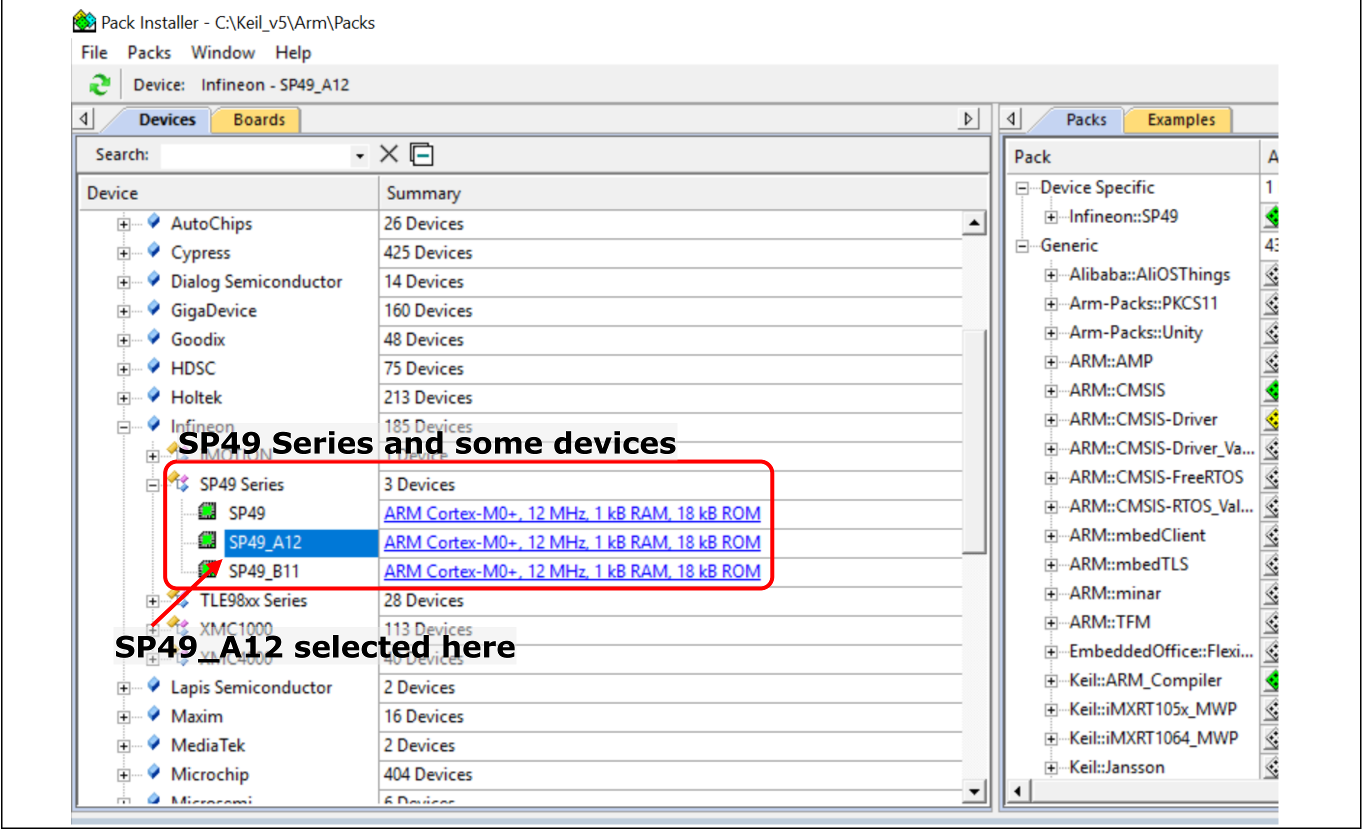
Finally either the LF-telegram or the LF-carrier (-sequence) can be sent on a definable carrier-frequency (given in [Hz] into the input marked with (5)). Also it can be repeated several times (defined by the input marked with (6)), each separated by a definable delay in [ms] (defined by the input marked with (7)).

3 Setup, Structure and usage of software-projects for SP49

This chapter explains how software projects for SP49 are setup and used. Infineon created a set of sample programs, which are included in the SP49 Keil Pack (for installation see chapter 1.2.2). This chapter is based on those sample programs from the SP49 Keil Pack.

3.1 Load projects from Keil Pack

To load one of the sample programs into Keil, open Keil, then open the Pack Installer. The newly opened window is split into two half. The left half shows “Devices” and “Boards”, the right half shows “Packs” and “Examples”. In the left half select “Devices”, then search for “Infineon” and expand it. Inside this group there is one entry “SP49 Series”. After expanding this group there should be listed some SP49 devices. After selecting the right one, for which the software shall be developed, the window should look like shown below.



Device	Summary
AutoChips	26 Devices
Cypress	425 Devices
Dialog Semiconductor	14 Devices
GigaDevice	160 Devices
Goodix	48 Devices
HDSC	75 Devices
Holtek	213 Devices
Infineon	185 Devices
iMOTION	1 Device
SP49 Series	3 Devices
SP49	ARM Cortex-M0+, 12 MHz, 1 kB RAM, 18 kB ROM
SP49_A12	ARM Cortex-M0+, 12 MHz, 1 kB RAM, 18 kB ROM
SP49_B11	ARM Cortex-M0+, 12 MHz, 1 kB RAM, 18 kB ROM
TLE98xx Series	28 Devices
XMC1000	113 Devices
XMC4000	40 Devices
Lapis Semiconductor	2 Devices
Maxim	16 Devices
MediaTek	2 Devices
Microchip	404 Devices
Microsemi	6 Devices

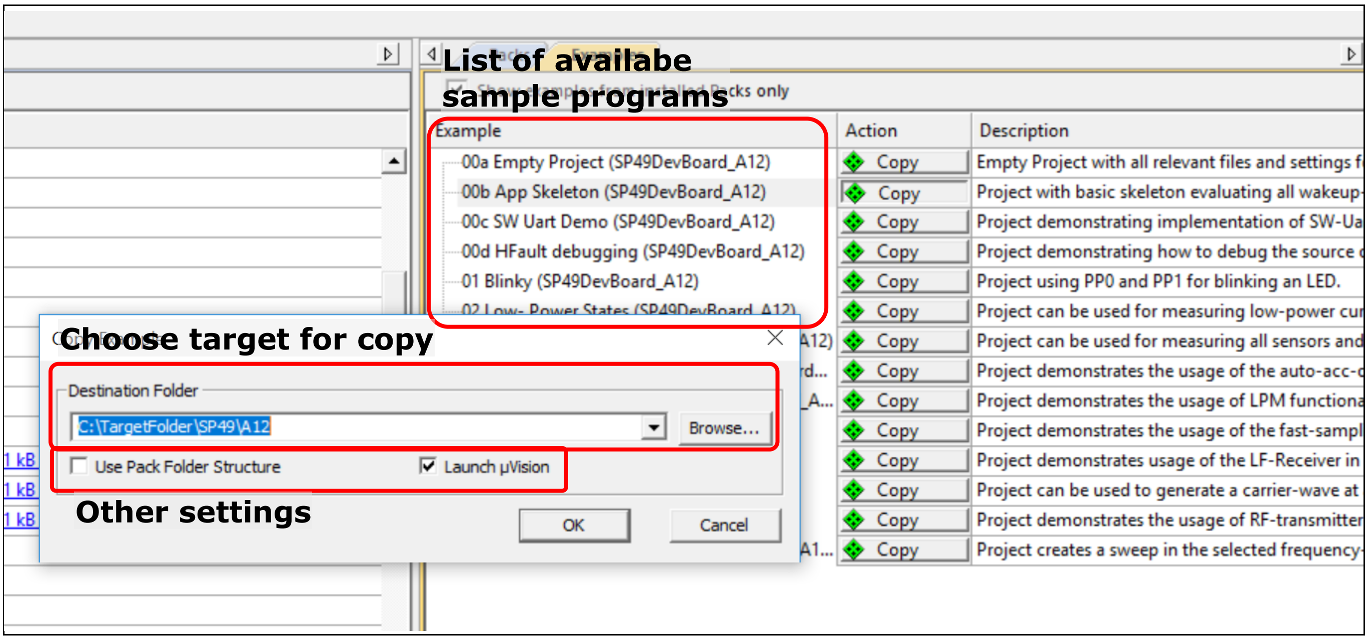
SP49 Series and some devices

SP49_A12 selected here

Figure 31 Pack Installer showing devices in “SP49 Series” group

Note: The list of devices here (SP49, SP49_A12, SP49_B11) might vary in future depending on the content of the installed Keil Pack.

Now go to the right half and select “Examples”. There will be listed several sample programs from which you can choose one to use. On the desired one click on “Copy”. This will open another dialog as shown in the next figure.



List of availabe sample programs

Show examples from installed Packs only

Example	Action	Description
00a Empty Project (SP49DevBoard_A12)	Copy	Empty Project with all relevant files and settings f
00b App Skeleton (SP49DevBoard_A12)	Copy	Project with basic skeleton evaluating all wakeup-
00c SW Uart Demo (SP49DevBoard_A12)	Copy	Project demonstrating implementation of SW-Ua
00d HFault debugging (SP49DevBoard_A12)	Copy	Project demonstrating how to debug the source c
01 Blinky (SP49DevBoard_A12)	Copy	Project using PP0 and PP1 for blinking an LED.
02 Low- Power States (SP49DevBoard_A12)	Copy	Project can be used for measuring low-power cur
A12)	Copy	Project can be used for measuring all sensors and
rd...	Copy	Project demonstrates the usage of the auto-acc-c
_A...	Copy	Project demonstrates the usage of LPM functiona
	Copy	Project demonstrates the usage of the fast-sampl
	Copy	Project demonstrates usage of the LF-Receiver in
	Copy	Project can be used to generate a carrier-wave at
	Copy	Project demonstrates the usage of RF-transmitter
A1...	Copy	Project creates a sweep in the selected frequency-

Choose target for copy

Destination Folder: C:\TargetFolder\SP49\A12

Browse...

☐ Use Pack Folder Structure ☑ Launch µVision

Other settings

OK Cancel

Figure 32 Copy sample program from Keil Pack Installer

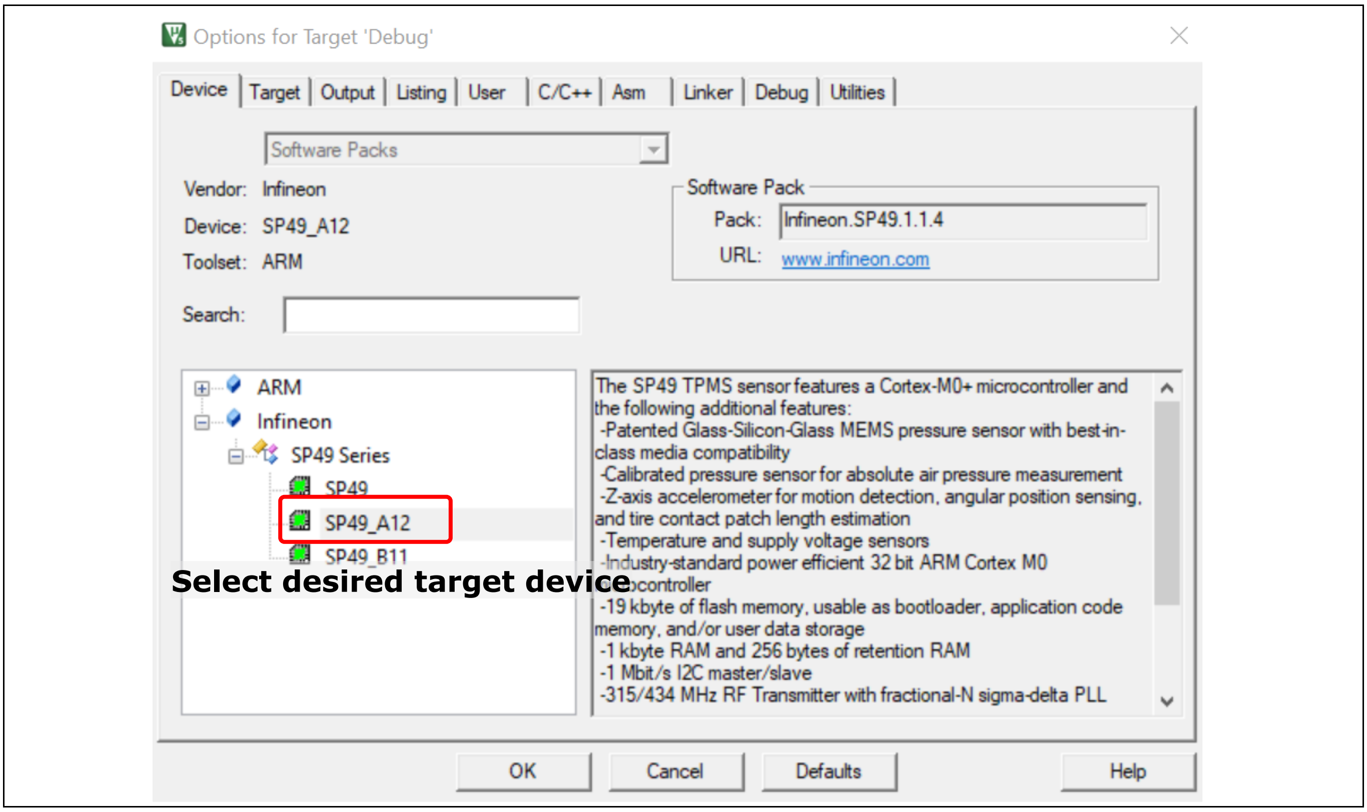
After a click on “OK” Keil will copy the whole selected project to the target folder and then start (if option “Launch uVision” is enabled) Keil with the copied project. Development based on this project can then start.

Note: *All relevant settings regarding debugger and code download are already done in all sample programs. Therefore those projects should run out of the box on the SP49 development-boards, as long as the jumpers are set correctly (see chapter 2.1.8).*

3.2 Required Keil settings for projects

Although the projects’ settings inside the Keil Pack are already preconfigured, it might happen that they might become corrupted once. This chapter shall give an overview of the essential settings in Keil for SP49 projects.

In the project overview on the left right-click on the topmost folder (for example “Debug” or “Release”) and select “Options for Target”. Alternatively search for the symbol 🔧 in the menu bar on top. This will open the target option window. Go to tab “Device” and select the desired device, for example “SP49_A12” as shown below.


Options for Target 'Debug'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities
Software Packs
OK
Cancel
Defaults
Help

Vendor: Infineon
Device: SP49_A12
Toolset: ARM
Search:

Software Pack
Pack: Infineon.SP49.1.1.4
URL: www.infineon.com

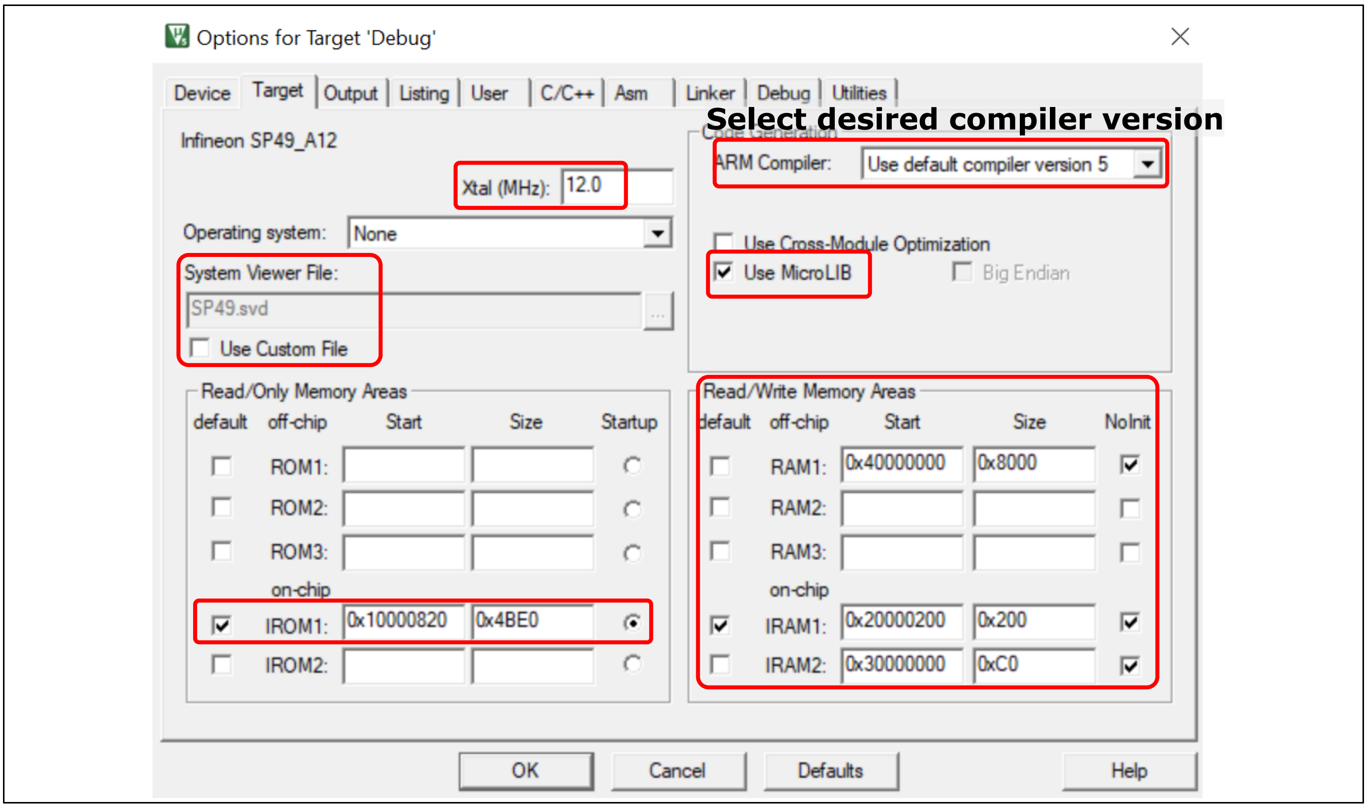
- ARM
- Infineon
  - SP49 Series
    - SP49
    - SP49_A12
    - SP49_B11

The SP49 TPMS sensor features a Cortex-M0+ microcontroller and the following additional features:
-Patented Glass-Silicon-Glass MEMS pressure sensor with best-in-class media compatibility
-Calibrated pressure sensor for absolute air pressure measurement
-Z-axis accelerometer for motion detection, angular position sensing, and tire contact patch length estimation
-Temperature and supply voltage sensors
-Industry-standard power efficient 32 bit ARM Cortex M0 ocontroller
-19 kbyte of flash memory, usable as bootloader, application code memory, and/or user data storage
-1 kbyte RAM and 256 bytes of retention RAM
-1 Mbit/s I2C master/slave
-315/434 MHz RF Transmitter with fractional-N sigma-delta PLL

Select desired target device

Figure 33 “Device” tab in target options

On Tab “Target”, make sure that all settings are as indicated below.


Options for Target 'Debug'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities
Code Generation
OK
Cancel
Defaults
Help

Infineon SP49_A12

Xtal (MHz): 12.0

Operating system: None

System Viewer File:
SP49.svd
□ Use Custom File

Select desired compiler version

ARM Compiler: Use default compiler version 5

□ Use Cross-Module Optimization
☑ Use MicroLIB □ Big Endian

Read/Only Memory Areas

default	off-chip	Start	Size	Startup
□	ROM1:			○
□	ROM2:			○
□	ROM3:			○
	on-chip			
☑	IROM1:	0x10000820	0x4BE0	●
□	IROM2:			○

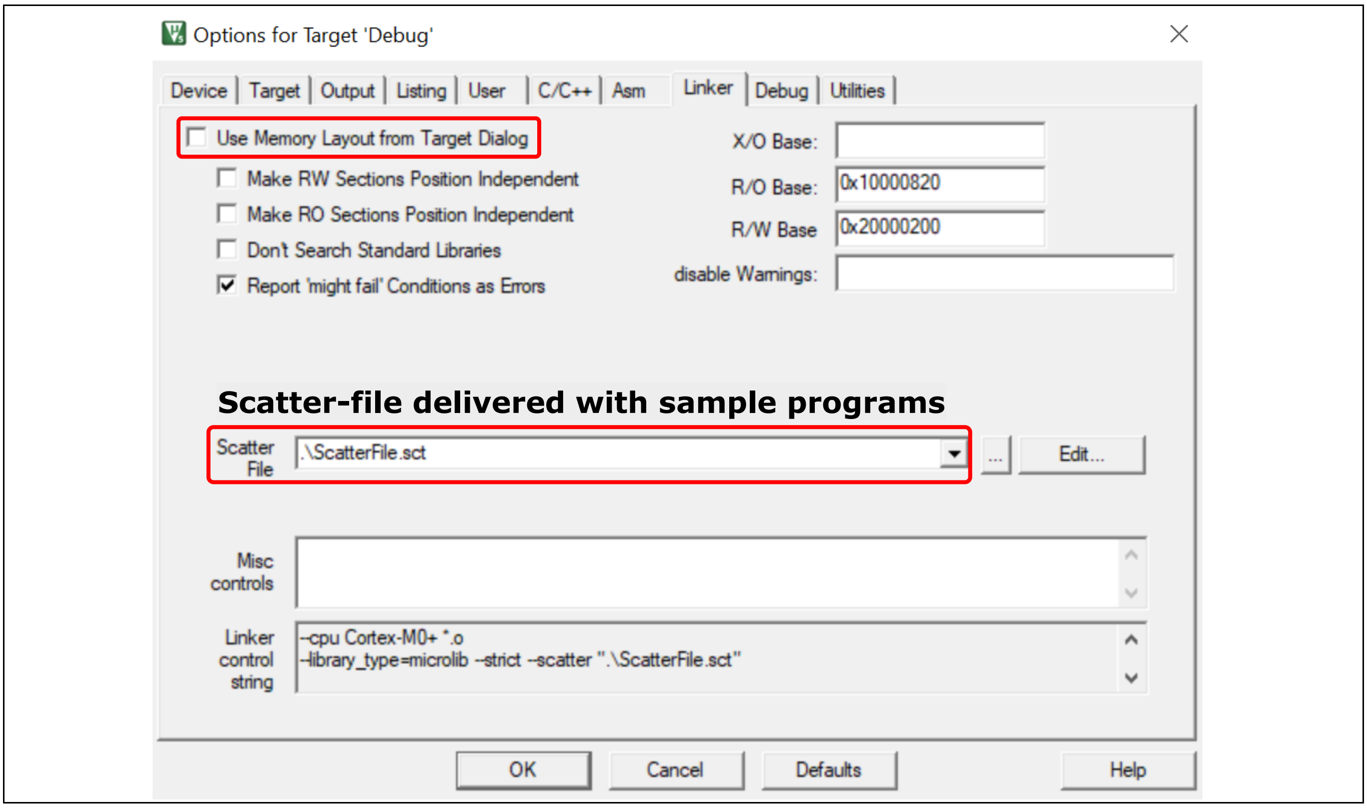
Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
□	RAM1:	0x40000000	0x8000	☑
□	RAM2:			□
□	RAM3:			□
	on-chip			
☑	IRAM1:	0x20000200	0x200	☑
□	IRAM2:	0x30000000	0xC0	☑

Figure 34 “Target” tab in target options and its settings

On the tabs “Output”, “Listing”, “User”, “C/C++” and “Asm” use the settings which are required for your project.

On tab “Linker” make sure the settings are as shown below.



☐ Use Memory Layout from Target Dialog

☐ Make RW Sections Position Independent

☐ Make RO Sections Position Independent

☐ Don't Search Standard Libraries

☑ Report 'might fail' Conditions as Errors

X/O Base:

R/O Base: 0x10000820

R/W Base 0x20000200

disable Warnings:

Scatter File: .\ScatterFile.sct

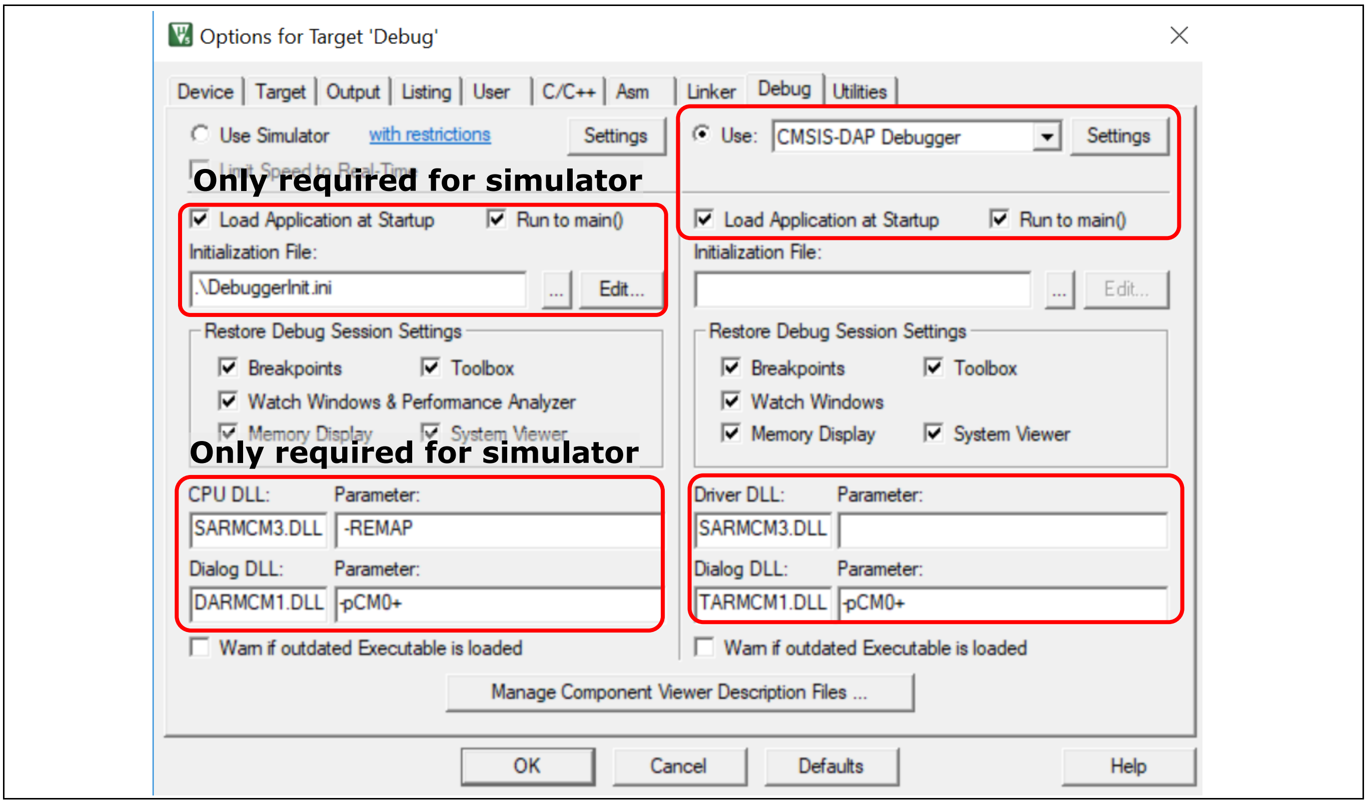
Misc controls:

Linker control string:

```
--cpu Cortex-M0+ *.o
--library_type=microlib --strict --scatter ".\ScatterFile.sct"
```

Figure 35 “Linker” tab in target options and its settings

On tab “Debug” make sure the settings are as shown below.



Options for Target 'Debug'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

○ Use Simulator with restrictions Settings

☐ Limit Speed to Real-Time

Only required for simulator

☑ Load Application at Startup ☑ Run to main()

Initialization File: .\DebuggerInit.ini ... Edit...

Restore Debug Session Settings
☑ Breakpoints ☑ Toolbox
☑ Watch Windows & Performance Analyzer
☑ Memory Display ☑ System Viewer

Only required for simulator

CPU DLL: SARMCM3.DLL Parameter: -REMAP

Dialog DLL: DARMCM1.DLL Parameter: -pCM0+

☐ Warn if outdated Executable is loaded

● Use: CMSIS-DAP Debugger Settings

☑ Load Application at Startup ☑ Run to main()

Initialization File: ... Edit...

Restore Debug Session Settings
☑ Breakpoints ☑ Toolbox
☑ Watch Windows
☑ Memory Display ☑ System Viewer

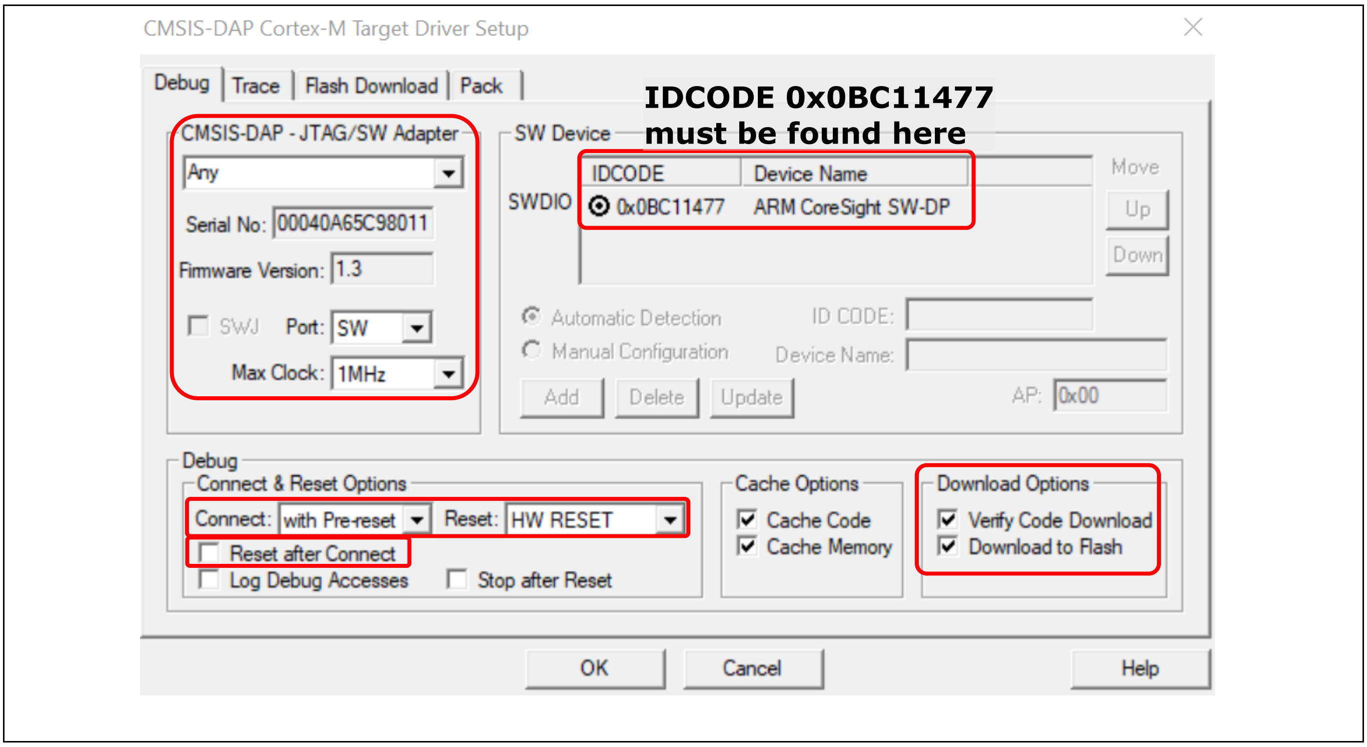
Driver DLL: SARMCM3.DLL Parameter:

Dialog DLL: TARMCM1.DLL Parameter: -pCM0+

☐ Warn if outdated Executable is loaded

Figure 36 “Debug” tab in target options and its settings

The debugger has to be configured in some points additionally. To apply those settings click on “Settings” right to the combo box where “CMSIS-DAP Debugger” is selected. Make sure all settings are set as shown below.



CMSIS-DAP Cortex-M Target Driver Setup

Debug | Trace | Flash Download | Pack

IDCODE 0x0BC11477 must be found here

CMSIS-DAP - JTAG/SW Adapter
Any
Serial No: 00040A65C98011
Firmware Version: 1.3
☐ SWJ Port: SW
Max Clock: 1MHz

SW Device

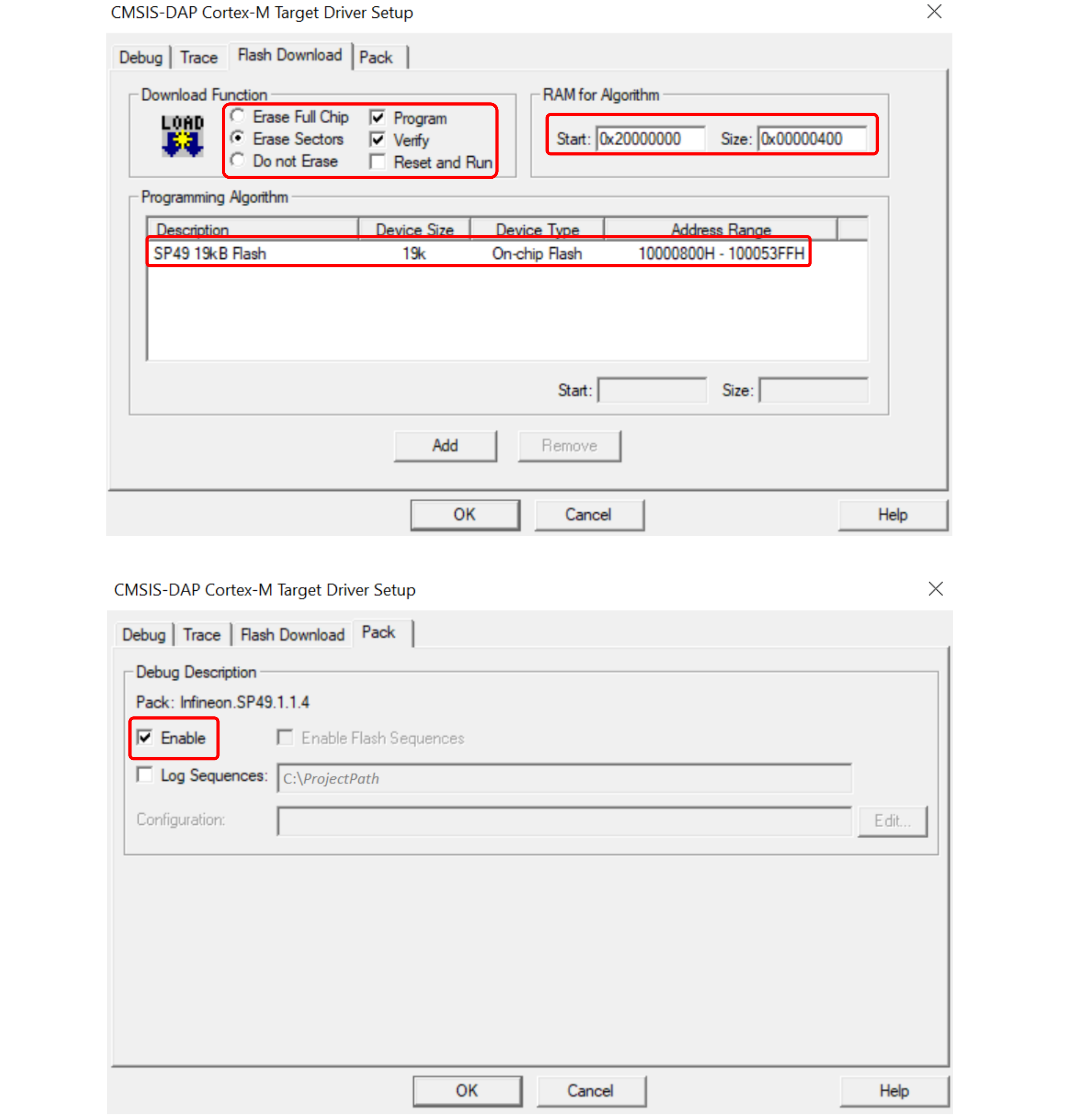
	IDCODE	Device Name
SWDIO	⊙ 0x0BC11477	ARM CoreSight SW-DP

● Automatic Detection ID CODE:
○ Manual Configuration Device Name:
AP: 0x00

Debug
Connect & Reset Options
Connect: with Pre-reset Reset: HW RESET
☐ Reset after Connect
☐ Log Debug Accesses ☐ Stop after Reset

Cache Options
☑ Cache Code
☑ Cache Memory

Download Options
☑ Verify Code Download
☑ Download to Flash



CMSIS-DAP Cortex-M Target Driver Setup

Download Function

○ Erase Full Chip ☑ Program
● Erase Sectors ☑ Verify
○ Do not Erase □ Reset and Run

RAM for Algorithm

Start: 0x20000000 Size: 0x00000400

Programming Algorithm

Description	Device Size	Device Type	Address Range
SP49 19kB Flash	19k	On-chip Flash	10000800H - 100053FFH

Start: Size:

CMSIS-DAP Cortex-M Target Driver Setup

Debug Description

Pack: Infineon.SP49.1.1.4

☑ Enable □ Enable Flash Sequences

□ Log Sequences: C:\ProjectPath

Configuration: Edit...

Figure 37 Settings of CMSIS-DAP Debugger for SP49

When all those settings are correct, flash download and debug should work.

3.3 Usage software projects

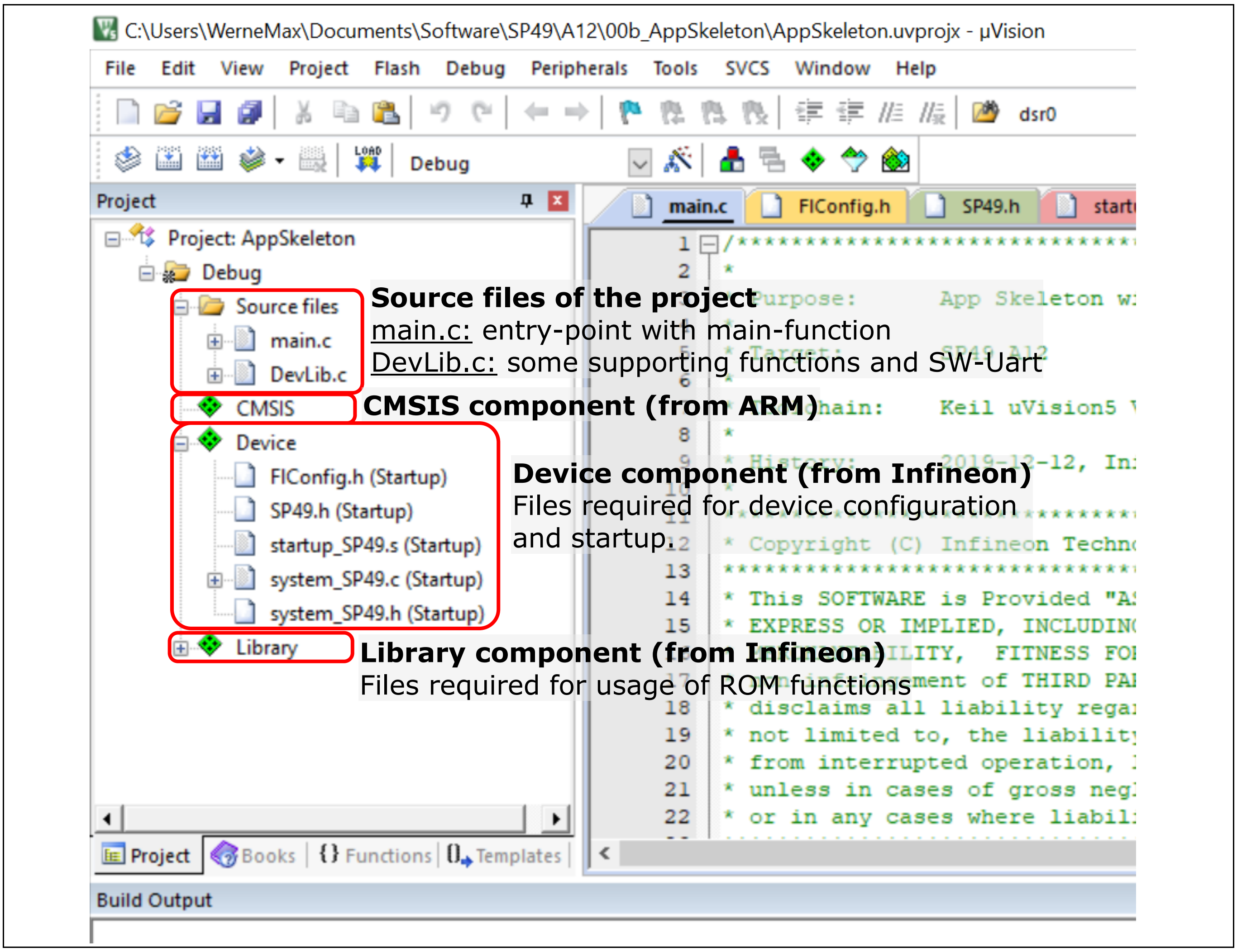
Each project for the SP49 requires a set of standard files. Firstly a folder “Source files”, where the user can place all his code-files (additional folders can be added also, if desired). Next is the component “Device”, which itself required the component “CMSIS”. Finally, for being able to use the Infineon provided ROM functions, a component “Library”.

Component “Device” contains all files required for starting up the application, the SFR-description file and a file for basic configuration of the device:

- FlConfig.h (to be modified by the user): In this header the flash configuration can be changed. SP49 offers a flexible way how the complete flash is configured and sub-divided into logical sectors (boot-, user-configuration- and code-sector). The first two *#define* let the user choose how many boot- and how many user-configuration-sectors shall be present on the device
- SP49.h (not to be modified): Contains the description of the SFRs and includes CMSIS-header-files
- startup_SP49.s (optionally to be modified by the user): Contains the reservation of RAM area for the stack and the entry point for the application. The stack can be adjusted in size here.
- system_SP49.c (to be modified by the user): In this c-file the data for device configuration, protection, lockbytes and CRC over sectors are reserved.
- system_SP49.h (not to be modified): Contains definitions required for system_SP49.c.

Component “Library” contains the pre-compiled flash-library “SP49_Library.lib” and the headers for the software modules, for example Lib_LF.h or Lib_State.h.

An overview of the files within a SP49 Keil project is given below.



```
/*****************************
 *
 * Purpose:     App Skeleton w
 * Target:      SP49 A12
 * Toolchain:   Keil uVision5 V
 *
 * History:     2019-12-12, In
 *****************************
 * Copyright (C) Infineon Techn
 *****************************
 * This SOFTWARE is Provided "A
 * EXPRESS OR IMPLIED, INCLUDIN
 *              ILITY,  FITNESS FO
 *              ment of THIRD PA
 * disclaims all liability rega
 * not limited to, the liabilit
 * from interrupted operation, 
 * unless in cases of gross neg
 * or in any cases where liabil
```

Source files of the project
main.c: entry-point with main-function
DevLib.c: some supporting functions and SW-Uart

CMSIS component (from ARM)

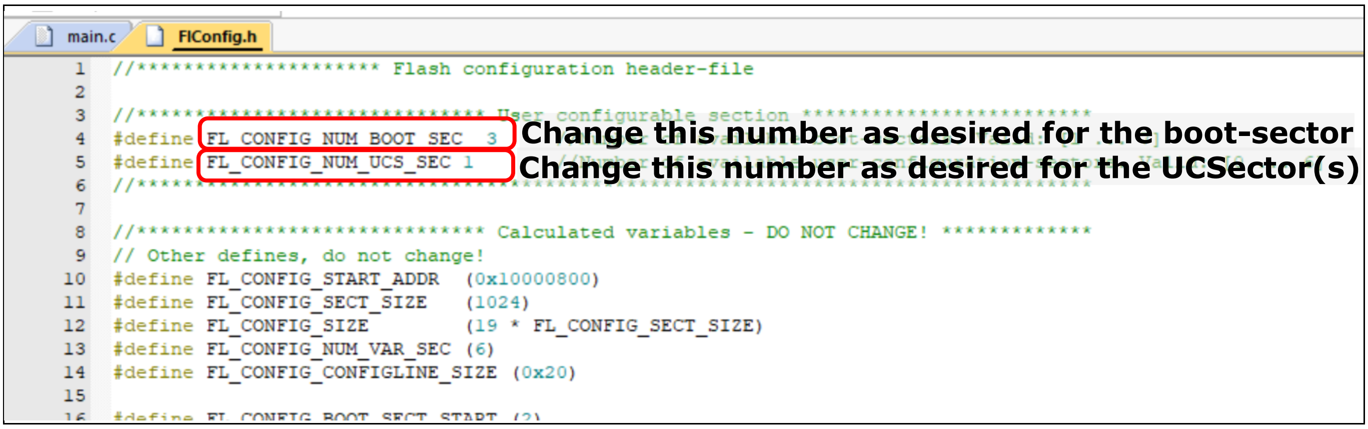
Device component (from Infineon)
Files required for device configuration and startup.

Library component (from Infineon)
Files required for usage of ROM functions

Figure 38 Exemplary SP49 Keil project and its components

3.3.1 Changing flash configuration

In order to change the SP49's flash configuration, just open the file "**FlConfig.h**" in the "Device" component. In this file the desired number of physical flash sectors assigned to the logical boot sector can be chosen, as well as the number of physical flash sector assigned to user-configuration-sector(s). The following example shows a configuration using 3kB for the boot sector and one 1kB sized user configuration sector.



```
//******************** Flash configuration header-file

//****************************** User configurable section *************************
#define FL_CONFIG_NUM_BOOT_SEC  3
#define FL_CONFIG_NUM_UCS_SEC 1
//*************************************************************************************

//****************************** Calculated variables - DO NOT CHANGE! *************
// Other defines, do not change!
#define FL_CONFIG_START_ADDR  (0x10000800)
#define FL_CONFIG_SECT_SIZE   (1024)
#define FL_CONFIG_SIZE        (19 * FL_CONFIG_SECT_SIZE)
#define FL_CONFIG_NUM_VAR_SEC (6)
#define FL_CONFIG_CONFIGLINE_SIZE (0x20)

#define FL_CONFIG_BOOT_SECT_START (2)
```

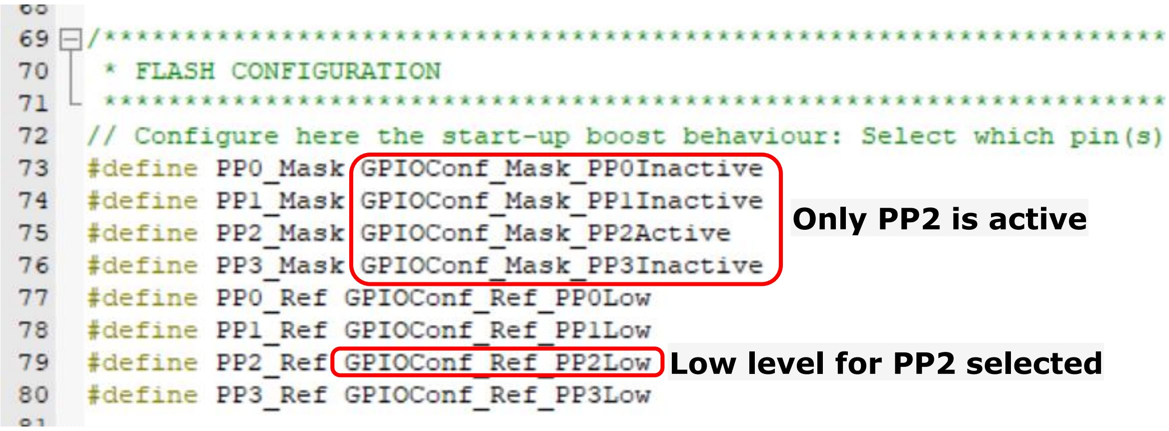
Change this number as desired for the boot-sector
Change this number as desired for the UCSector(s)

Figure 39 Example for flash configuration in SP49 Keil project

After compilation the flash-configuration line (at address 0x10000800) will contain the data appropriate to the chosen configuration.

3.3.2 Changing start-up boost configuration

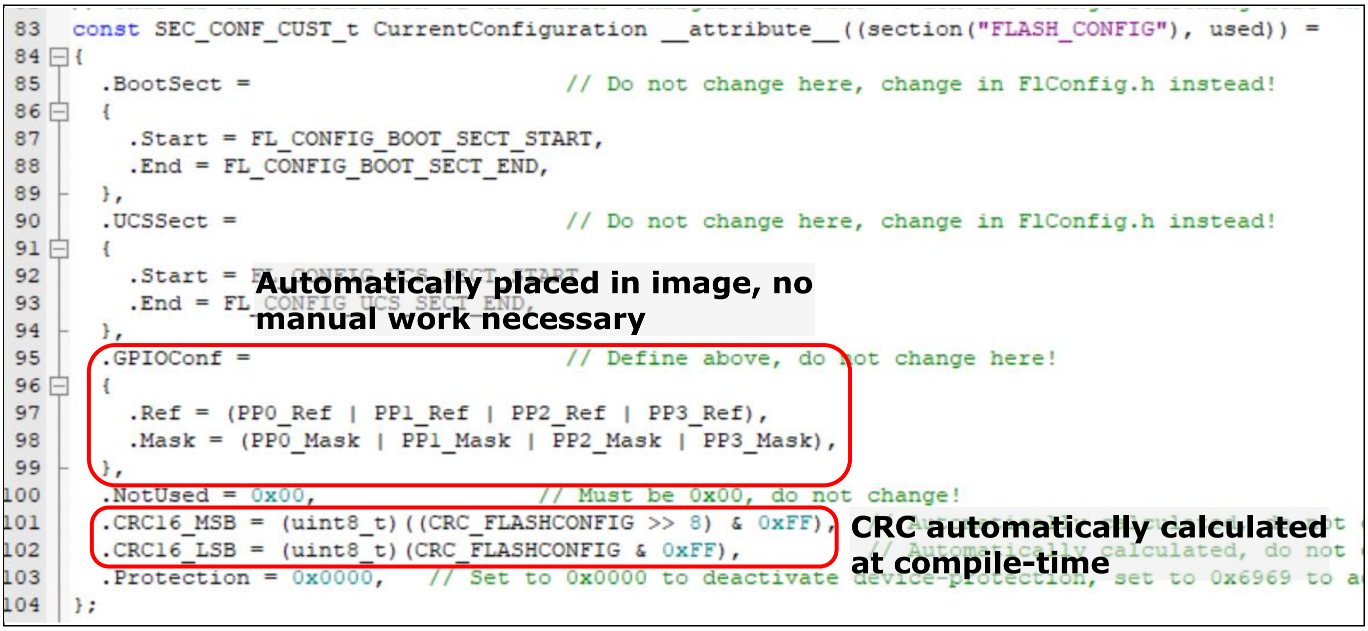
SP49 supports a feature to skip the mode-selection loop after any kind of reset. This feature is called start-up boost and is configured in the first line of user flash at address 0x10000800 and influenced by one or multiple levels at the digital IO pins PP0 to PP3. The user can enable one or multiple of the PPx pins, each individually. The user can also choose for each enabled PPx pin an individual logical level that must be set after reset, in order to skip the mode-selection loop. The Keil projects reserve the memory location for this configuration and includes it in the compiled output file. To change the selection and configuration of the pins for the start-up boost feature, open file "**system_SP49.c**" and change the lines as shown below. Here, as an example, only PP2 is activated for startup-boost and a logical level of 0 is configured.



```
/*****************************************************************
 * FLASH CONFIGURATION
 *****************************************************************
// Configure here the start-up boost behaviour: Select which pin(s)
#define PP0_Mask GPIOConf_Mask_PP0Inactive
#define PP1_Mask GPIOConf_Mask_PP1Inactive
#define PP2_Mask GPIOConf_Mask_PP2Active
#define PP3_Mask GPIOConf_Mask_PP3Inactive
#define PP0_Ref GPIOConf_Ref_PP0Low
#define PP1_Ref GPIOConf_Ref_PP1Low
#define PP2_Ref GPIOConf_Ref_PP2Low
#define PP3_Ref GPIOConf_Ref_PP3Low
```

Only PP2 is active
Low level for PP2 selected

… some line below …



```
const SEC_CONF_CUST_t CurrentConfiguration __attribute__((section("FLASH_CONFIG"), used)) =
{
  .BootSect =                    // Do not change here, change in FlConfig.h instead!
  {
    .Start = FL_CONFIG_BOOT_SECT_START,
    .End = FL_CONFIG_BOOT_SECT_END,
  },
  .UCSSect =                     // Do not change here, change in FlConfig.h instead!
  {
    .Start = FL_CONFIG_UCS_SECT_START,
    .End = FL_CONFIG_UCS_SECT_END,
  },
  .GPIOConf =                    // Define above, do not change here!
  {
    .Ref = (PP0_Ref | PP1_Ref | PP2_Ref | PP3_Ref),
    .Mask = (PP0_Mask | PP1_Mask | PP2_Mask | PP3_Mask),
  },
  .NotUsed = 0x00,              // Must be 0x00, do not change!
  .CRC16_MSB = (uint8_t)((CRC_FLASHCONFIG >> 8) & 0xFF),   // Automatically calculated, do not
  .CRC16_LSB = (uint8_t)(CRC_FLASHCONFIG & 0xFF),          // Automatically calculated, do not
  .Protection = 0x0000,     // Set to 0x0000 to deactivate device-protection, set to 0x6969 to a
};
```

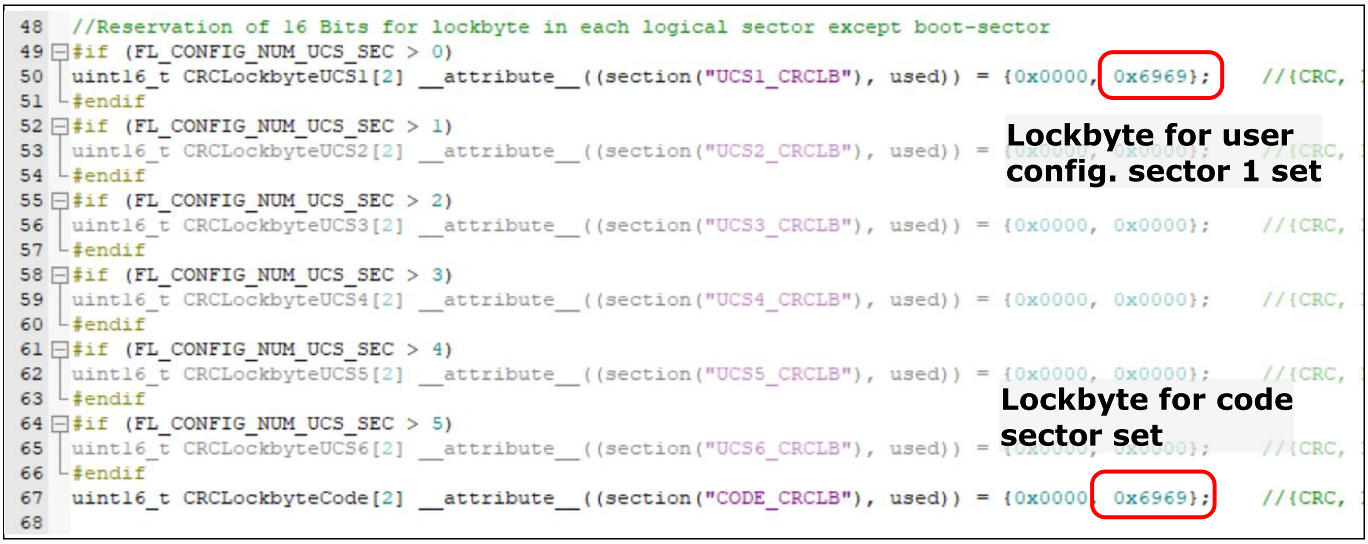
Automatically placed in image, no manual work necessary

CRC automatically calculated at compile-time

Figure 40 Example for start-up boost configuration in SP49 Keil project

3.3.3 Setting lockbytes for sectors in flash image

Most of the logical flash sectors of SP49 have a lockbyte, which prevents further writing it. Such lockbyte can also be already included in the compiled flash image. To add a lockbyte to a sector, open file "**system_SP49.c**". Below an example is shown with a lockbyte in the first user configuration sector and the code sector being set.



```
//Reservation of 16 Bits for lockbyte in each logical sector except boot-sector
#if (FL_CONFIG_NUM_UCS_SEC > 0)
uint16_t CRCLockbyteUCS1[2] __attribute__((section("UCS1_CRCLB"), used)) = {0x0000, 0x6969};    //{CRC,
#endif
#if (FL_CONFIG_NUM_UCS_SEC > 1)
uint16_t CRCLockbyteUCS2[2] __attribute__((section("UCS2_CRCLB"), used)) = {0x0000, 0x0000};    //{CRC,
#endif
#if (FL_CONFIG_NUM_UCS_SEC > 2)
uint16_t CRCLockbyteUCS3[2] __attribute__((section("UCS3_CRCLB"), used)) = {0x0000, 0x0000};    //{CRC,
#endif
#if (FL_CONFIG_NUM_UCS_SEC > 3)
uint16_t CRCLockbyteUCS4[2] __attribute__((section("UCS4_CRCLB"), used)) = {0x0000, 0x0000};    //{CRC,
#endif
#if (FL_CONFIG_NUM_UCS_SEC > 4)
uint16_t CRCLockbyteUCS5[2] __attribute__((section("UCS5_CRCLB"), used)) = {0x0000, 0x0000};    //{CRC,
#endif
#if (FL_CONFIG_NUM_UCS_SEC > 5)
uint16_t CRCLockbyteUCS6[2] __attribute__((section("UCS6_CRCLB"), used)) = {0x0000, 0x0000};    //{CRC,
#endif
uint16_t CRCLockbyteCode[2] __attribute__((section("CODE_CRCLB"), used)) = {0x0000, 0x6969};    //{CRC,
```

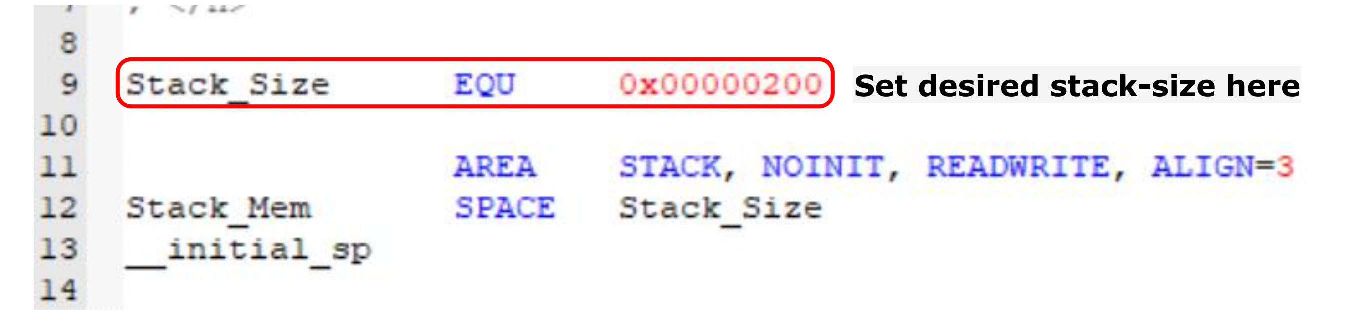
Lockbyte for user config. sector 1 set

Lockbyte for code sector set

Figure 41 Example for lockbyte configuration in SP49 Keil project

3.3.4 Adapting stack size

It is important that the stack is configured big enough. In SP49 the ROM functions don't have a statically reserved RAM area for their variables, they are all put on stack. Therefore the stack must be big enough for both, application and ROM functions called. By default the Keil projects provided by Infineon reserve 512 Bytes of RAM for the stack, which is considered to be big enough for most projects. However, it is important that the user verifies that this stack size is big enough before releasing his productive code. If the stack-size is too small or too big, there are basically two steps required in the Keil projects to adapt it. First step is in "**startup_SP49.s**". The example below shows a stack-size of 512 Bytes.

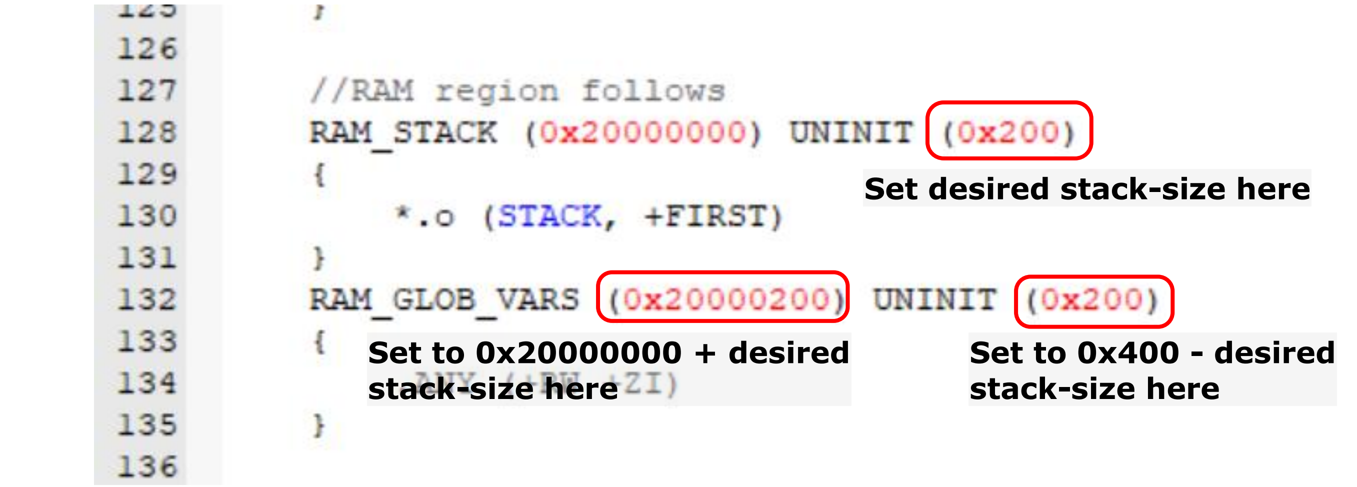


```
Stack_Size      EQU     0x00000200

                AREA    STACK, NOINIT, READWRITE, ALIGN=3
Stack_Mem       SPACE   Stack_Size
__initial_sp
```

Set desired stack-size here

Figure 42 Setting stack-size in startup_SP49.s

The second step needs to be done in the linker's scatter file "**ScatterFile.sct**" as shown below.



```
//RAM region follows
RAM_STACK (0x20000000) UNINIT (0x200)
{
   *.o (STACK, +FIRST)
}
RAM_GLOB_VARS (0x20000200) UNINIT (0x200)
{
   .ANY (+RW +ZI)
}
```

Set desired stack-size here

Set to 0x20000000 + desired stack-size here

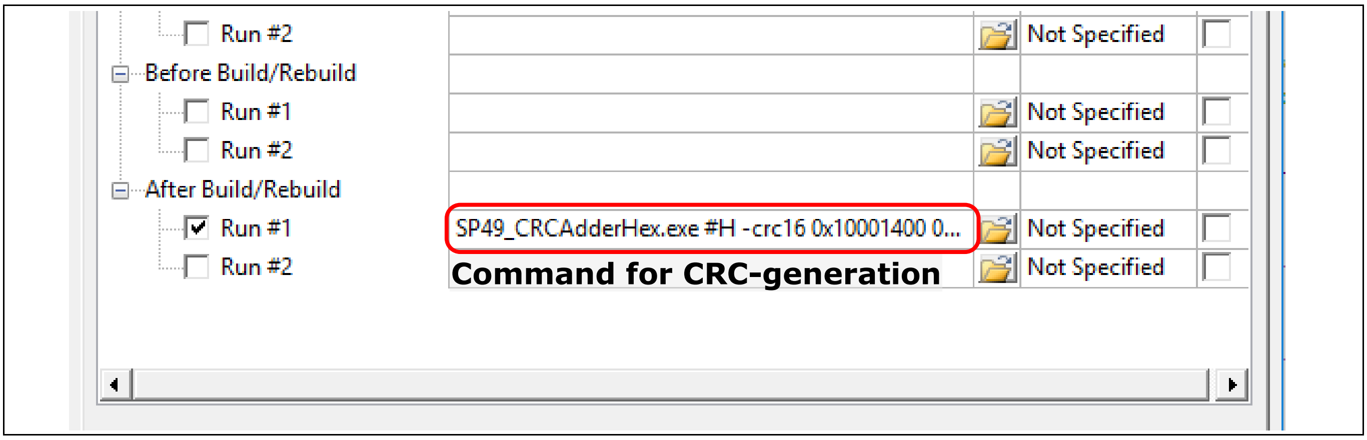
Set to 0x400 - desired stack-size here

Figure 43 Adapting scatter file for new stack size

3.4 Adding CRC(s) to compiled hex-file

Sometimes it is necessary that the generated hex-file includes one or multiple CRCs over defined areas. In SP49 always the CRC16-CCITT type of CRC is used. Each project contains in its root folder a file called "SP49_CRCAdderHex.exe", which can be called with arguments after compilation finished to automatically add those CRC(s) at configurable places.

First step is to configure Keil to properly call the executable after compilation. Therefore open the project settings and go to tab "User". In the line below "After Build/Rebuild" the command must be entered, as exemplary shown in the next figure.



□ Run #2 | Not Specified □
Before Build/Rebuild
□ Run #1 | Not Specified □
□ Run #2 | Not Specified □
After Build/Rebuild
☑ Run #1 | SP49_CRCAdderHex.exe #H -crc16 0x10001400 0... | Not Specified □
□ Run #2 | Not Specified □

Figure 44 **Command for automatic CRC-generation after build**

The tool “SP49_CRCAdderHex.exe” uses the following syntax for arguments:

```
SP49_CRCAdderHex.exe InputPath\InputFile.hex -crc16 DataStartAddress
CRC16StartAddress [-crc16 DataStartAddressN CRC16StartAddressN] -o
OutputPath\OutputFile.hex
```

As many –crc16 tags with DataStartAddress and CRC16StartAddress can be added as desired. The CRC16-CCITT will take into account the data between [DataStartAddress until CRC16StartAddress-1]. The calculated CRC will be stored big-endian at address [CRC16StartAddress until CRC16StartAddress+1].

Keil offers the usage of key codes to automatically detect output folder and output file name. The following table is a copied excerpt from keil.com (http://www.keil.com/support/man/docs/uv4/uv4_ut_keysequence.htm).

Table 3 **Key codes and file codes for user command in Keil**

Key codes		
Key	**Example**	**Description**
%	PROJECT1.UVPROJX	File name with extension
#	C:\MYPROJECT\PROJECT1.UVPROJX	Complete path and file name with extension
@	PROJECT1	File name without extension or path specification
$	C:\MYPROJECT\	Path name of a file extended with a backslash.
!	.\SRC\TEST.C	File name with extension and relative path specification to the current folder

File codes	
File	**Description**
H	Application HEX file name (PROJECT1.H86).
P	Current project file name.

Note: Enclose Key Sequences within quotes (" ") when using folder names that might contain special characters (space, ~, or #).

As an example, assume a generated hex-file called “Outputfile.hex”. In this example two CRCs shall be generated, one over the area [0x10001400 … 0x100017FB] and placed at [0x100017FC … 0x100017FD], another CRC over the area [0x10001800 … 0x1000153FB] and placed at [0x1000153FC … 0x1000153FC]. The Keil commands would look like

- When generated “Outputfile.hex” shall be replaced with the new one including the CRCs

```
SP49_CRCAdderHex.exe “#H” -crc16 0x10001400 0x100017FC -crc16 0x10001800
0x100053FC -o “#H”
```

- When generated “Outputfile.hex” shall not be replaced, but a new file “Outputfile_withCRC.hex” shall be generated

```
SP49_CRCAdderHex.exe “#H” -crc16 0x10001400 0x100017FC -crc16 0x10001800
0x100053FC -o “$H@H_withCRC.hex”
```

Revision history

Document version	Date of release	Description of changes
1.0	2020-04-24	Initial version
1.1	2025-04-17	First publicly available version
		Chapter “1.2.2 Install SP49 Keil Pack”: Information about where to find the Keil pack updated.
		Chapter “1.2.4 Making SP49 development-board GUI startable from Keil”: Information about launching SP49 Development GUI from Keil updated
		Chapter “1.2.5 Recommended: Install ARM Compiler version 5” added
		Chapter “2.1.4 Streaming logical levels of two pins” added
		Chapter “2.3.4 Using the current measurement” updated : New feature for averaging N-shots added
		Chapter “2.3.5 Using the LF generator” updated: Description for entering payload in binary format added

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

Edition 2025-04-17

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2025 Infineon Technologies AG.
All Rights Reserved.

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.