

BHI360 BHI380

Ultra-low power, high performance, programmable
Smart Sensor with integrated IMU

BHI360/BHI380 SDK Quick Start Guide

Document revision	1.5
Document release date	April 18 th , 2023
Document number	BST-BHI360_BHI380-AN000-05
Technical reference code(s)	0 273 141 367 0 273 141 392
Notes	Data and descriptions in this document are subject to change without notice. Product photos and pictures are for illustration purposes only and may differ from the real product appearance.

Table of Contents

1	Introduction to the SDK.....	6
2	Setup in Windows	6
2.1	Installing the compiler and support tools	6
2.2	Installing the SDK	6
2.3	Importing the SDK into Eclipse	7
3	Setup in Linux	10
3.1	Installing the ARC GNU toolchain and support tools	10
3.2	Installing the SDK in Linux	11
4	Building the SDK and loading firmware into the BHI360 (BHI380)	12
5	Adding a BSX based new custom virtual driver	13
5.1	Driver directory structure	13
5.2	Writing driver code	14
5.3	Selecting a driver ID	15
5.4	Driver CMakeLists.txt File.....	15
5.5	Brief introduction to the board configuration file	16
5.6	Modifying the board configuration file	17
5.7	Brief introduction to the SDK configuration file.....	18
5.8	Modifying the SDK configuration file	19
5.9	Building the custom firmware	19
5.10	Lean orientation example.....	19
6	Adding a non-Bosch Sensortec Fusion Library related new custom virtual driver	20
6.1	Driver directory structure	20
6.2	Writing driver code	20
6.3	Selecting a driver ID	22
6.4	Driver CMakeLists.txt file.....	22
6.5	Modifying the board configuration file	23
6.6	Modifying the SDK configuration file	23
6.7	Build the custom firmware.....	23
6.8	Altitude example	24
7	Integrating a library and applying it to the custom sensor driver	25
7.1	Library directory structure.....	25
7.2	Implementing a sensor driver that uses the library	25
7.3	Modifying the board configuration file	26
7.4	Modifying the SDK configuration file	26
7.5	Build the custom firmware.....	27
8	Glossary.....	28
8.1	Virtual Sensor	28
8.2	Driver ID	28
8.3	Sensor ID	28

9 Legal disclaimer	29
10 Document history	30

List of Figures

Figure 1: BHI360 SDK installer	7
Figure 2: Installation destination directory	7
Figure 3: Eclipse workspace prompt.....	8
Figure 5: Configuring the build trigger.....	Error! Bookmark not defined.
Figure 5: Configuring the build trigger.....	9
Figure 6: Configuring the build trigger's arguments.....	9
Figure 7: GNU toolchain releases download page	10
Figure 8: Architecture of physical and virtual drivers	13
Figure 9: Driver descriptor overview	14
Figure 10: Driver CMakeLists.txt.....	15
Figure 11: Board configuration file overview.....	16
Figure 12: Modifying the board configuration file.....	17
Figure 13: Overview of the SDK configuration file	18
Figure 14: Modifying the SDK configuration file.....	19
Figure 15: Driver descriptor overview	21
Figure 16: Driver CMakeLists.txt.....	22
Figure 17: Modifying the board configuration file.....	23
Figure 18: Modifying the SDK configuration file.....	23
Figure 19: Altitude output data structure.....	24
Figure 20: CMakeLists.txt	25
Figure 21: Modifying the board configuration file.....	26
Figure 22: Modifying the SDK configuration file.....	26

List of Tables

Table 1: Pre-build SDK directory structure in Windows.....	10
Table 2: Pre-build SDK directory structure in Linux.....	11
Table 3: Driver directory content.....	13
Table 4: sif and bus options	17
Table 5: Driver directory content.....	20
Table 6: Library directory content	25
Table 7: Driver directory content.....	25

Abbreviations

BST	Bosch Sensortec
BSX	Bosch Sensortec Fusion Library
FIFO	First In First Out
GCC	GNU Compiler Collection
RAM	Random Access Memory
SDK	Software Development Kit
USB	Universal Serial Bus
TRNG	True Random Number Generator
RDRAND	Read Random

1 Introduction to the SDK

This document briefly describes the process of developing firmware for the BHI360 (BHI380). It provides instructions on how to

- ▶ set up the development environment
- ▶ build the SDK
- ▶ get started with custom configuration files.

For more details about hardware configuration, refer to *BHI360-BHI380 Datasheet*.

For more details about developing new drivers, refer to the following manual and user guide:

- ▶ BHI360-BHI380 Programmer's Manual
- ▶ BHI360-BHI380 Evaluation Setup Guide

The BHI360 (BHI380) SDK can be used to develop a custom firmware image. The customization includes

- ▶ modifying the board configuration
- ▶ changing the mapping of pins
- ▶ changing the device orientation
- ▶ allocating memory to the FIFO
- ▶ creating custom drivers which can run algorithms or other tasks
- ▶ data injection for processor in the loop verification.

The firmware built by using the SDK can be downloaded to the BHI360's RAM.

For more details, refer to *BHI360-BHI380 Programmer's Manual*.

2 Setup in Windows

This chapter describes how to install the required tools in Windows. The BHI360 (BHI380) SDK supports two toolchains: ARC GNU toolchain and Synopsys Metaware. This guide focuses on how to build the SDK with the ARC GNU toolchain. Since the SDK generates signed firmware images and the signing tool requires the True Random Number Generator feature of the CPU to generate a valid signature, the CPU used to build the SDK must support the RDRAND instruction.

2.1 Installing the compiler and support tools

The GNU Toolchain for ARC Processors can be downloaded from the [Synopsys Github Website](#). Download the file "*arc_gnu_2022.09_ide_win_install.exe*" or newer and run this setup installer executable. This will primarily install the Eclipse IDE and the ARC GNU Compiler.

Please download the latest ninja tool from [Ninja Release Website](#), extract *ninja.exe* from *ninja* Windows package and replace the old one at *win64/bin* under BHI360 (BHI380) SDK root path.

2.2 Installing the SDK

For Windows system, an SDK installer is provided. To install the SDK, do the following:

1. Execute the *BHI360_SDK_V1.0.6_Install.exe* or newer, accept the license agreement and click *Next*, as shown in Figure 1.

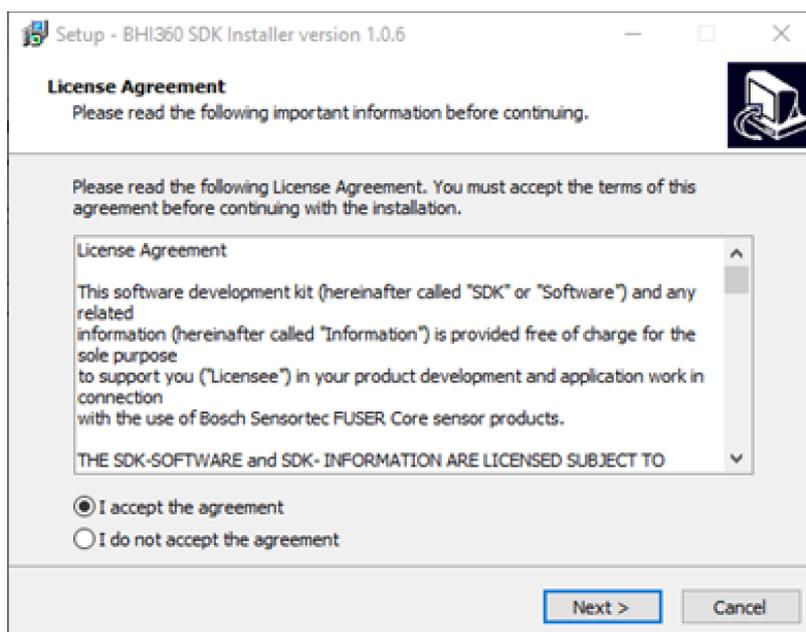


Figure 1: BHI360 SDK installer

2. Select the destination location for the SDK.

Then in the SDK destination directory, “*BHI360_SDK_VX.Y.Z*” is created.

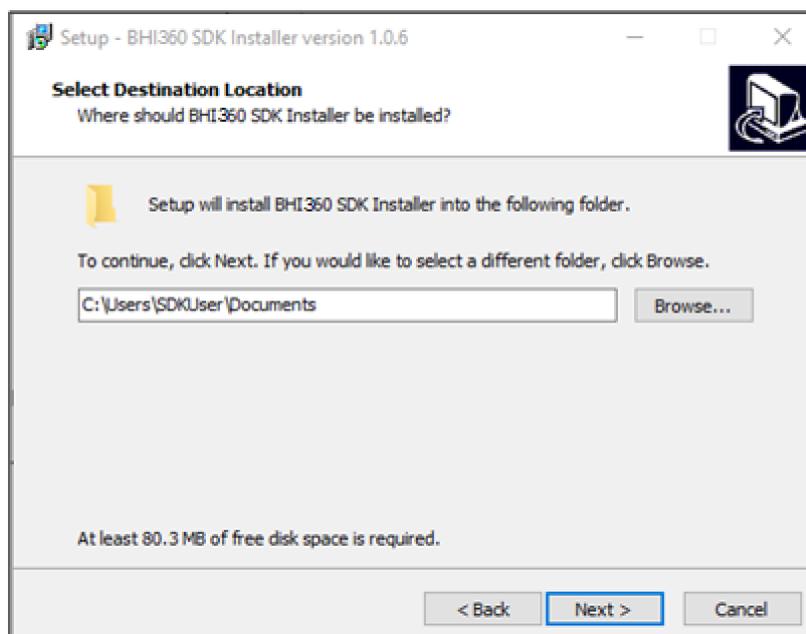


Figure 2: Installation destination directory

2.3 Importing the SDK into Eclipse

1. Set up Eclipse.
 - a. Run the Eclipse IDE by clicking on its shortcut on the Desktop, which should be generically named “*ARC GNU IDE YYYY.MM(-rcN) Eclipse*”. For example, “*ARC GNU IDE 2021.09 Eclipse*”.
 - b. During the first launch, you will be prompted to select a workspace. The default directory is an empty directory that stores multiple projects. You can select your preferred workspace directory.

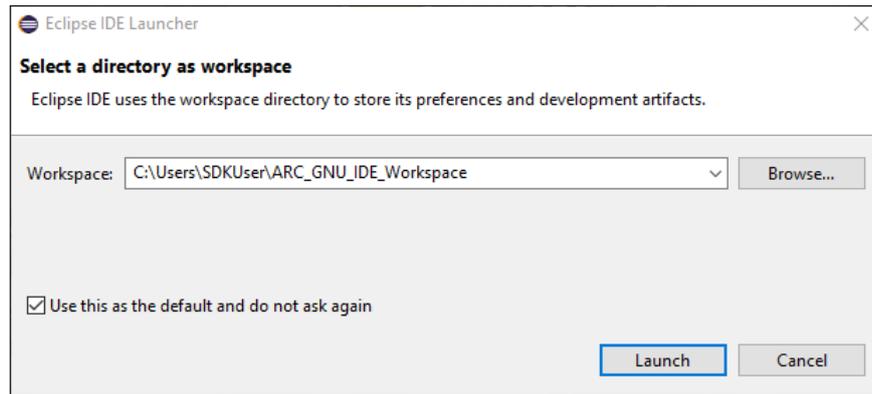
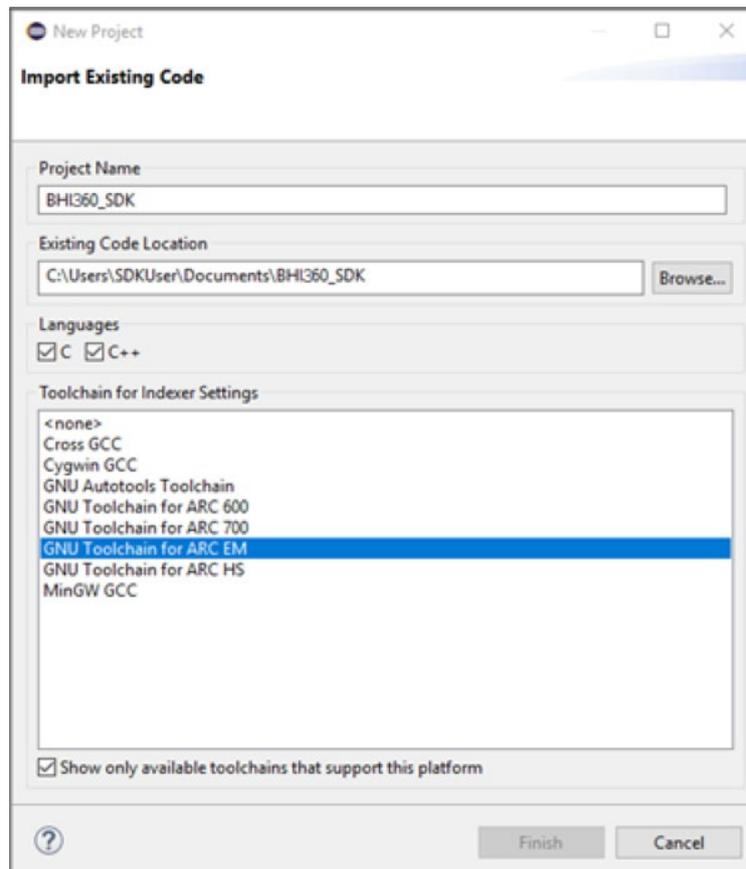


Figure 3: Eclipse workspace prompt

2. Import the BHI360 SDK as a project.
 - a. In the Eclipse IDE, go to *File > New > Makefile Project with Existing Code*
 - b. In the prompt, type a project name, for example, as shown in the figure below. Select the SDK directory and then click *Finish*.



- c. If the Welcome tab is open, close it to reveal the *Project Explorer*.
3. Link the project build to the batch script that builds the firmware.
 - a. In Windows, the building of the firmware is managed by a batch script named *build.bat* which can be found in the root of the SDK directory.
 - b. Right-click on the *BHI360_SDK* project and select *Properties*.

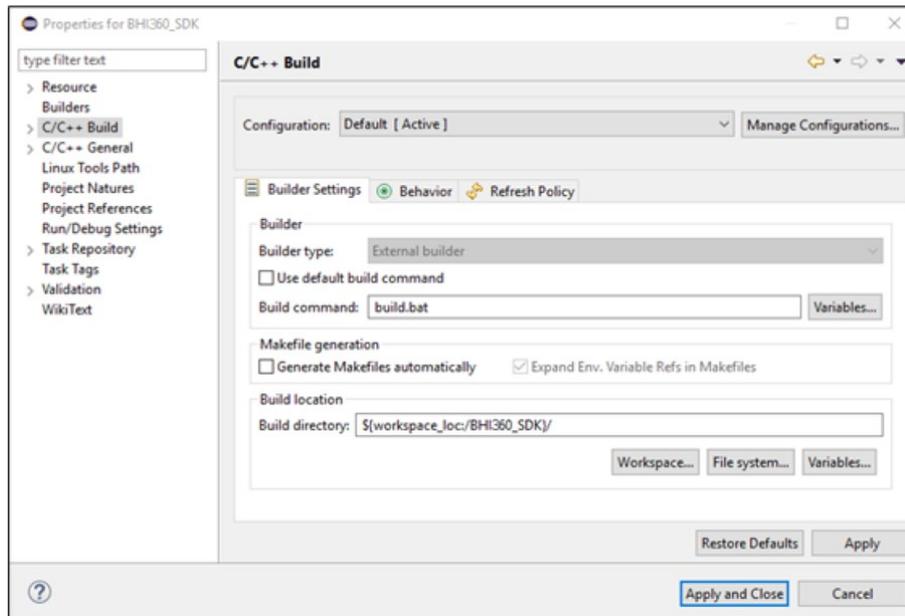


Figure 4: Configuring the build trigger

- c. Under *C/C++ Build / Builder settings*, deselect *Use default build command* and refer the image for selecting the build trigger. Click *Apply*.
- d. Under the *C/C++ Build / Behavior*, deselect *Clean* and remove the command *all* from the *Build behavior*, as shown in the figure below. Click *Apply and Close*.

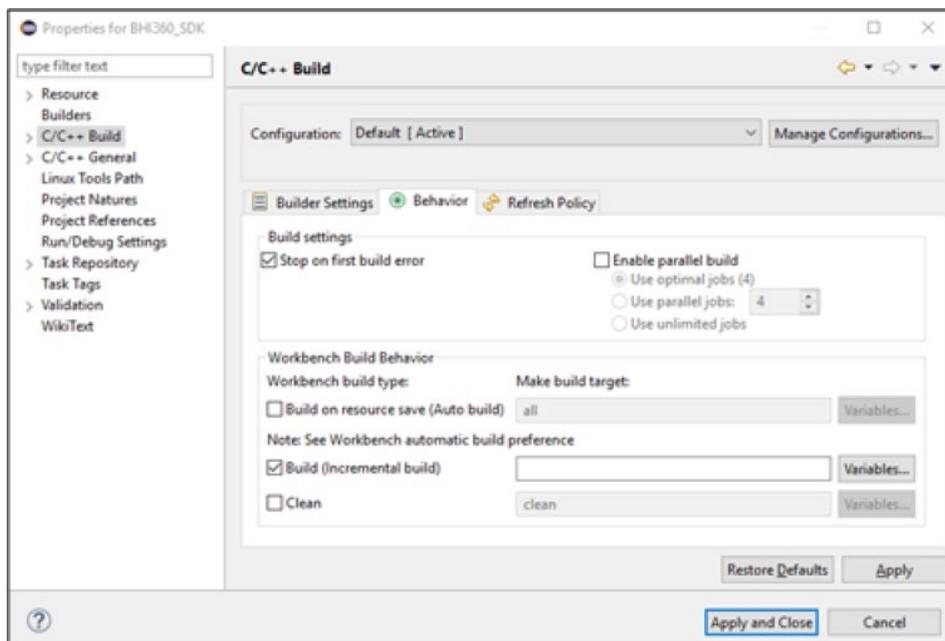


Figure 5: Configuring the build trigger's arguments

- e. Click on the build icon . This will run *build.bat* and the progress is visible in the console located at the bottom.
4. Locate the built firmware.
 - a. The firmware build can be found under *release/gccfw* in the root directory of the SDK. If the firmware is built by Metaware rather than ARC GNU toolchain, it can be found under *release/fw* instead.

Table 1: Pre-build SDK directory structure in Windows

SDK File/Directory	Description
apps	Directory which contains the source code for applications running outside the sensor framework
boards	Configuration files for the supported development boards and sensors
cmake	CMake files used to build the SDK
common	Source code for initialization code and reference header files
docs	SDK documentation
drivers	Source codes of sensor drivers from Bosch Sensortec
drivers_custom	Source codes of additional custom drivers
gdb	Support files for using gdb
kernel	Exported symbols for kernel-mode firmware
libs	Linkable binary image and header files for API libraries
user	Entry code for user-mode firmware, source code for custom user-mode RAM patches
win64	Executable image manipulation utilities, command line interface
build.bat	Shell script used to set up build directory and build the specified target

3 Setup in Linux

3.1 Installing the ARC GNU toolchain and support tools

To get started, the following system requirements must be met:

- ▶ 64-bit Linux operating system (Ubuntu 14.04 LTS or later)
- ▶ At least 1.1 GB of free disk space

Before the SDK can be used, ARC GNU toolchain, CMake, and other necessary dependencies must be installed. The operations in this guide have been verified on Ubuntu 14.04 LTS and 16.04 LTS.

1. Download the ARC GNU toolchain.

The ARC GNU toolchain releases are available on the [Synopsys Github Website](#). A pre-built toolchain that supports elf32 little-endian hosts is required.

In this example, the 2022.09 release is used. This release can be downloaded from the same download page as the previous releases.

The right installation package to download is “arc_gnu_2022.09_prebuilt_elf32_le_linux_install.tar.gz”.

arc_gnu_2021.09_ide_linux_install.tar.gz	1.49 GB
arc_gnu_2021.09_ide_macos_install.tar.gz	1.26 GB
arc_gnu_2021.09_ide_plugins.zip	865 KB
arc_gnu_2021.09_ide_win_install.exe	841 MB
arc_gnu_2021.09_prebuilt_elf32_be_linux_install.tar.gz	555 MB
arc_gnu_2021.09_prebuilt_elf32_be_macos_install.tar.gz	541 MB
arc_gnu_2021.09_prebuilt_elf32_le_linux_install.tar.gz	527 MB
arc_gnu_2021.09_prebuilt_elf32_le_macos_install.tar.gz	516 MB
arc_gnu_2021.09_prebuilt_glibc_be_archs_linux_install.tar.gz	119 MB
arc_gnu_2021.09_prebuilt_glibc_le_archs_linux_install.tar.gz	118 MB
arc_gnu_2021.09_prebuilt_glibc_le_archs_native_install.tar.gz	106 MB
arc_gnu_2021.09_prebuilt_uclibc_be_arch700_linux_install.tar.gz	72.5 MB
arc_gnu_2021.09_prebuilt_uclibc_be_archs_linux_install.tar.gz	88.8 MB
arc_gnu_2021.09_prebuilt_uclibc_le_arch700_linux_install.tar.gz	71.6 MB
arc_gnu_2021.09_prebuilt_uclibc_le_archs_linux_install.tar.gz	87.8 MB
Source code (zip)	
Source code (tar.gz)	

Figure 6: GNU toolchain releases download page

2. Install the GNU toolchain.

- a. Run the following commands to extract the GNU toolchain installation package:

```
$ tar -xvf arc_gnu_2022.09_prebuilt_elf32_le_linux_install.tar.gz
$ sudo mkdir -p /opt/arc_gcc
$ sudo mv arc_gnu_2022.09_prebuilt_elf32_le_linux_install /opt/arc_gcc
```

- b. Run the following commands to verify the GNU toolchain has been installed successfully:

```
$ cd /opt/arc_gcc/arc_gnu_2022.09_prebuilt_elf32_le_linux_install/bin/
$ ./arc-elf32-gcc -dumpversion
```

- c. Update the PATH variable to include “opt/arc_gcc/arc_gnu_2022.09_prebuilt_elf32_le_linux_install/bin”. This can be done by modifying the shell start-up script as appropriate. For example, edit “/etc/profile” with the following command.

```
$ sudo nano /etc/profile
```

- d. Add the path to the file by adding the following line.

```
export PATH=$PATH:/opt/arc_gcc/arc_gnu_2022.09_prebuilt_elf32_le_linux_install/bin
```

3. Install the CMake and other dependencies.

- a. To install the CMake, run the following commands:

```
$ sudo apt-get install cmake
$ cmake --version
```

- b. To install the other dependencies or tools if necessary, run the following commands.

```
$ sudo apt-get install libelf-dev
$ sudo apt-get install g++
$ sudo apt-get install lib32stdc++6
```

- c. It is highly recommended to install ninja to speed up the build process by parallel building.

```
$ sudo apt-get install ninja-build
```

3.2 Installing the SDK in Linux

The SDK is released as an installer “*BHI360_SDK_VX.Y.Z_Install.sh*”.

Take BHI360 SDK V1.0.6 for example, to make the installer executable, run the following command:

```
$/BHI360_SDK_V1.0.6_Install.sh
```

Bosch Sensortec License must be accepted by typing yes in the command line prompt. Then, the installer prompts to move to the preferred directory. The default installation directory is “**\$(HOME)/Bosch_Sensortec_Fuser2_SDK**”.

The SDK has the directory structure as shown in Table 2.

structure as shown

Table 2: Pre-build SDK directory structure in Linux

SDK File/Directory	Description
apps	Directory that contains the source code for applications running outside the sensor framework
boards	Configuration files for the supported development boards and sensors
cmake	CMake files used to build the SDK
common	Source code for initialization code and reference header files
docs	SDK documentation

SDK File/Directory	Description
drivers	Source codes of sensor drivers from Bosch Sensortec
drivers_custom	Source codes of additional custom drivers
gdb	Support files for using gdb
kernel	Exported symbols for kernel-mode firmware
libs	Linkable binary image and header files for API libraries
user	Entry code for user-mode firmware, source code for custom user-mode RAM patches
utils	Executable image manipulation utilities, command line interface
build.sh	Shell script used to set up a build directory and build the specified SDK

4 Building the SDK and loading firmware into the BHI360 (BHI380)

In Windows, clicking on the build icon  in the Eclipse IDE or executing the build.bat script will trigger the build process. In Linux, run the build script in its root:

```
$ ./build.sh
```

build and *release* directories are created after the build script is executed. If both the ARC GNU compiler and the Metaware compiler are available on the path, the Metaware compiler is used. To override this behavior and force the use of the ARC GNU compiler, add the option “USE_GCC” as an argument to the build script.

```
$ ./build.sh USE_GCC
```

```
$ ./build.bat USE_GCC
```

For the BHI360 (BHI380), one image for RAM is generated. With successive build triggers, all previously generated files under the *build* and *release* directories are removed and new firmware files are generated under the *release/gccfw* or *release/fw* folder.

The generated “*.fw” file can be verified by using the bhy2cli tool. The bhy2cli is a command line tool based on the COINES tool that interfaces with the BHI360 (BHI380) through Bosch Sensortec’s application board. The tool can be used to load and run standard and custom firmware images among other features.

For example, running

```
$ bhy2cli -b release\fw\Bosch_Shuttle3_BHI360_BMM150.fw -c 34:25
```

loads the firmware file Bosch_Shuttle3_BHI360_BMM150.fw for the board configuration Bosch_Shuttle3_BHI360_BMM150.cfg and switches on streaming of the [sensor ID](#) 34 at 25Hz to the terminal. Refer to *BHI360-BHI380 Evaluation Setup Guide* for more information on building the *bhy2cli* tool.

5 Adding a BSX based new custom virtual driver

In order to demonstrate how one can add a custom driver to the SDK, two drivers, *VirtBSXLeanDeviceOrientation* and *VirtBSXCustomAccelDataSource*, have already been included in the SDK as examples but have not been used in any of the firmware images.

Both *VirtBSXLeanDeviceOrientation* and *VirtBSXCustomAccelDataSource* are in the *drivers_custom* directory of the SDK. *VirtBSXCustomAccelDataSource* receives accelerometer data from the Bosch Sensortec Fusion Library but does not send it to the host. Instead, it triggers *VirtBSXLeanDeviceOrientation* which receives the data, processes it, and stores the processed data in the requested FIFO.

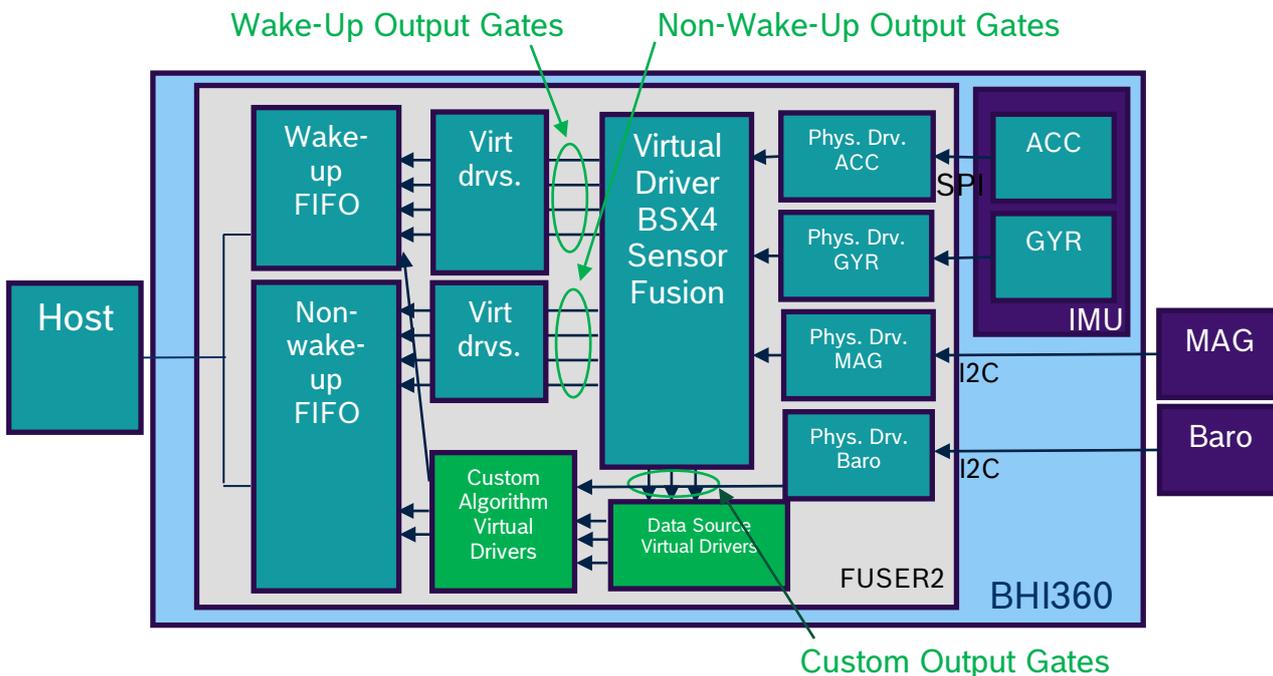


Figure 7: Architecture of physical and virtual drivers

For more information on how to develop a new physical sensor driver or [virtual sensor](#) driver in the SDK, refer to the BHI360-BHI380 Programmer's Manual.

5.1 Driver directory structure

The sensor driver code must be in its own directory under the *drivers_custom* directory of the SDK. The directory name should reflect the device name and driver type, for example, *VirtBSXLeanDeviceOrientation*.

Table 3: Driver directory content

File in Driver Directory	Description
CMakeLists.txt	Build description of the driver
VirtBSXLeanDeviceOrientation.c	Source code of the driver
Header file	Header file typically defining register locations and other constants for the driver if needed

5.2 Writing driver code

The dependency between the two [virtual sensors](#) is described below.

For more detailed and complete information on how to program sensor drivers, refer to the *BHI360-BHI380 Programmer's Manual*.

“hidden = TRUE” means the sensor is not visible to host, and it only provides data source.

VirtBSXCustomAccelDataSource is the trigger source of *VirtBSXLeanDeviceOrientation*.

```
VIRTUAL_SENSOR_DESCRIPTOR VirtualSensorDescriptor
descriptor_virt_bsx_custom_accel_data_source = {
    .physicalSource = {
        .sensor = {
            .type = {
                .value = BSX_INPUT_ID_ACCELERATION,
                .flags = DRIVER_TYPE_PHYSICAL_FLAG,
            },
        },
    },
    .info = {
        .id = DRIVER_ID,
        .version = DRIVER_REV,
    },
    .type = {
        .value =
        SENSOR_TYPE_BSX(BSX_CUSTOM_ID_ACCELERATION_CORRECTED),
        .flags = DRIVER_TYPE_VIRTUAL_FLAG,
        .wakeup_ap = FALSE,
        .hidden = TRUE,
    },
    .expansionData = {
        .f32 = OUTPUT_SCALING_FACTOR,
    },
    .maxRate = 800.0F,
    .minRate = 1.5625F,
    .outputPacketSize = sizeof(output_3axis_t),
    .priority = PRIORITY_2,
    .initialize = NULL,
    .handle_sensor_data =
    BSXSupport_trigger_custom_sensors,
};
```

```
VIRTUAL_SENSOR_DESCRIPTOR VirtualSensorDescriptor
descriptor_virt_bsx_lean_device_orientation = {
    .triggerSource = {
        .sensor = {
            .type = {
                .value =
                SENSOR_TYPE_BSX(BSX_CUSTOM_ID_ACCELERATION_CORRECTED)
            },
            .flags = DRIVER_TYPE_VIRTUAL_FLAG,
        },
    },
    .physicalSource = {
        .sensor = {
            .type = {
                .value = BSX_INPUT_ID_ACCELERATION,
                .flags = DRIVER_TYPE_PHYSICAL_FLAG,
            },
        },
    },
    .info = {
        .id = DRIVER_ID,
        .version = DRIVER_REV,
    },
    .type = {
        .value = SENSOR_TYPE_CUSTOMER_VISIBLE_START,
        .flags = DRIVER_TYPE_VIRTUAL_FLAG,
        .wakeup_ap = FALSE,
    },
    .maxRate = 800.0F,
    .minRate = 1.5625F,
    .outputPacketSize = sizeof(output_t),
    .priority = PRIORITY_2,
    .initialize = ldo_initialize,
    .handle_sensor_data = ldo_handle_sensor_data,
    .mode_changed = ldo_on_power_mode_changed,
};
```

The [Sensor ID](#) is made visible to the host.

Figure 8: Driver descriptor overview

5.3 Selecting a driver ID

To add a new [virtual sensor](#) driver, the first step is to select the available driver ID for compilation. Unless the driver to be developed is already included in the SDK, users may choose any unused 8-bit number. There is a python script under the root directory of the SDK. Running it will show the existing driver names and associated driver IDs. Using this script will need an existing installation of [Python](#).

```
$ python find_BHy3_driver_IDs.py
```

In this excerpt, we have selected the driver IDs **131** and **132** in the Driver CMakeLists.txt file (See section 5.4). Each driver has a unique driver ID.

5.4 Driver CMakeLists.txt File

The below mentioned *CMakeLists.txt* file automatically pulls in the sources from each driver. It is used by the build system at link time to associate the driver ID listed with a driver's object file. More driver IDs can be defined in the same way. Usually users do not need to modify it.

Take *drivers_custom/VirtBSXLeanDeviceOrientation/CMakeLists.txt* for example:

```
SET(DRIVER_ID 132)

get_filename_component( DRIVER_KEY ${CMAKE_CURRENT_LIST_DIR} NAME)

project(${DRIVER_KEY} C)

FILE(GLOB SOURCES "*.c")

include_directories(../../libs/BSXSupport/includes/
                    ../../libs/BSX/includes/)

ADD_ARC_DRIVER(${DRIVER_KEY} ${DRIVER_ID} ${SOURCES})
```

Figure 9: Driver CMakeLists.txt

Table 4: sif and bus options

sif	M1	M2	M3
0	SPI0	SPI1	I2C1
1	SPI0	I2C0	I2C1
2	I2C0	SPI1	I2C1

For more details about the sif configuration, refer the BHI360-BHI380 datasheet. For details about the board configuration file, refer the BHI360-BHI380 Programmer's manual.

5.6 Modifying the board configuration file

In order to add the *VirtBSXLeanDeviceOrientation* and *VirtBSXCustomAccelDataSource* virtual sensors into the *Bosch_Shuttle3_BHI360_BMM150_Cus.fw* one must add a new configuration file *boards/Bosch_Shuttle3_BHI360_BMM150_Cus.cfg* (take *Bosch_Shuttle3_BHI360_BMM150.cfg* as the reference) and add the virtual drivers to the virtual sensor list in the respective “*.cfg” file as shown below.

```

...
#Virtual Drivers,maxRate
(131, 800.000000 # VirtBSXCustomAccelDataSource: depends on a physical accelerometer.)
(132, 800.000000 # VirtBSXLeanDeviceOrientation: depends on a virtual BSX source. )
240, -1.000000 # VirtBSX: BSX depends on a programatic trigger source.
241, 400.000000 # VirtBSXAccel: accelerometer corrected data depends on VirtBSX.
209, 400.000000 # VirtBSXAccelOffset: accelerometer offset data depends on VirtBSX.
205, 400.000000 # VirtBSXAccelPassthrough: accelerometer passthrough data depends on VirtBSX.
...

```

Link *VirtBSXCustomAccelDataSource* (Driver ID: 131) and *VirtBSXLeanDeviceOrientation* (Driver ID: 132) into the *Bosch_Shuttle3_BHI360_BMM150_Cus.fw*

Figure 11: Modifying the board configuration file

5.7 Brief introduction to the SDK configuration file

In brief, all SDK generated firmware images include both the pre-built kernel image and user images. This configuration file includes board configuration files, enabled drivers, libraries, etc.

The SDK has one configuration file *common/config.7189_di03_rtos_bhi360.cmake*, which can be edited as needed.

<pre>... set(BOARDS Bosch_Shuttle3_BHI360 Bosch_Shuttle3_BHI360_turbo Bosch_Shuttle3_BHI360_BMM150 ...) ...</pre>	<p>The BOARDS variable describes which of the target boards' configurations are to be built. When the "build.sh" or "build.bat" script is executed, only the firmware for those specific boards are built.</p>
<pre>set(DRIVERS_NO_SOURCE BMM150Mag BHI360SigMotion ... VirtBME680Humidity ... VirtHangDetection)</pre>	<p>The DRIVERS_NO_SOURCE variable describes which drivers (including physical and virtual drivers) are already present as library files in the SDK.</p>
<pre>set(ENABLED_DRIVERS #Example Injection driver \${DRIVERS_NO_SOURCE})</pre>	<p>Drivers whose sources need to be built, like custom drivers, should be directly added to the ENABLED_DRIVERS variable.</p>

Figure 12: Overview of the SDK configuration file

5.8 Modifying the SDK configuration file

To build a firmware that contains the reference custom drivers for the target board configuration the `common/config.7189_di03_rtos_bhi360.cmake` needs to be modified as shown below.

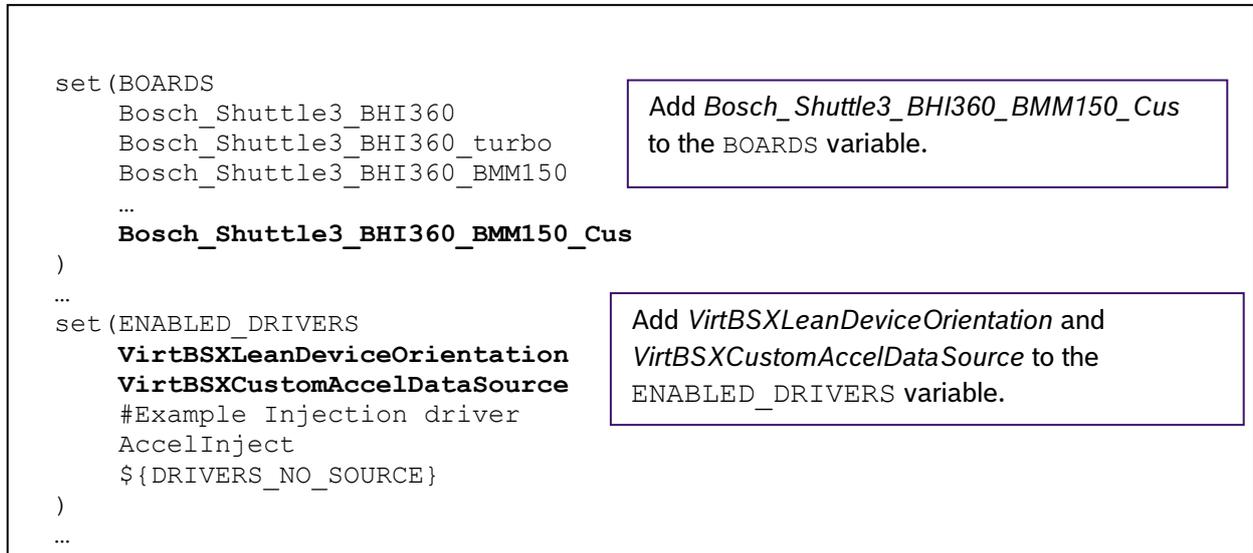


Figure 13: Modifying the SDK configuration file

5.9 Building the custom firmware

To build only a specific board configuration file, the name of the board can be passed as an argument. For example, in Linux this would look like,

```
$ ./build.sh Bosch_Shuttle3_BHI360_BMM150_Cus
```

If more than one board needs to be built in a similar way, a semicolon is required between board names like below,

```
$ ./build.sh "Bosch_Shuttle3_BHI360_BMM150_Cus;Bosch_Shuttle3_BHI360"
```

The build.bat file for Windows can accept similar arguments.

The custom firmware is now available under `release/gccfw` as `Bosch_Shuttle3_BHI360_BMM150_Cus.fw`. Like with the reference firmware, you can use the `bhy2cli` like below to load the firmware and view the lean orientation sensor's output using the generic type of handle.

```
$ bhy2cli -b release/fw/Bosch_Shuttle3_BHI360.fw -a 160:"Lean Orientation":2:c:c -c 160:1
```

5.10 Lean orientation example

With the new custom virtual drivers inside the firmware, the host should also be able to configure the sensor ID and parse sensor events from the FIFO.

The corresponding host side example of the virtual sensor `VirtBSXLeanDeviceOrientation` called "Lean Orientation" is provided separately. Refer to the BHI360-BHI380 Evaluation Setup Guide on how to verify and evaluate the aforementioned newly integrated virtual sensors.

6 Adding a non-Bosch Sensortec Fusion Library related new custom virtual driver

This chapter takes *VirtAltitude* for example to describe how to add a non-Bosch Sensortec Fusion Library related new custom virtual sensor driver. The steps are same as in Chapter 5, except that some details vary depending on the actual requirements.

VirtAltitude is in the *drivers_custom* directory. It creates custom altitude data and sends it to the respective FIFO.

For more information on how to develop a physical/virtual sensor driver in the SDK, refer to the BHI360-BHI380 Programmer's Manual.

6.1 Driver directory structure

The sensor driver code must be in its own directory in the *drivers_custom* directory of the SDK. The directory name should reflect the device name and driver type, for example *VirtAltitude*.

Table 5: Driver directory content

File in Driver Directory	Description
CMakeLists.txt	Build description of the driver
VirtAltitude.c	Source code of the driver
Header file	Header file typically defining register locations and other constants for the driver if needed

6.2 Writing driver code

Below is code snippets *VirtAltitude* driver in Figure 16, which explains its trigger source and how to exchange reference sea level values with the host through the parameter interface.

For more information on how to program sensor drivers, refer the BHI360-BHI380 Programmer's Manual.

```

...
#define DRIVER_REV          (4u)
...

#define PARAM_PAGE_OPTIONAL_SDK          (8)
#define OPTIONAL_SDK_PARAM_ALTITUDE_SEE_LEVEL  (0x00)

bool optional_sdk_page_write_handler(uint8_t param, uint16_t length, uint8_t buffer[])
...

bool optional_sdk_page_read_handler(uint8_t param, uint16_t length, uint8_t buffer[], uint16_t
*ret_length)
...

static SensorStatus virt_altitude_initialize_sensor(VirtualSensorDescriptor *self)
{
    (void) registerWriteParamHandler(PARAM_PAGE_OPTIONAL_SDK, optional_sdk_page_write_handler);
    (void) registerReadParamHandler(PARAM_PAGE_OPTIONAL_SDK, optional_sdk_page_read_handler);
    verbose("altitude initialize\n");
    return SensorOK;
}
...

VIRTUAL_SENSOR_DESCRIPTOR VirtualSensorDescriptor virt_altitude_descriptor =
{
    .triggerSource =
    {
        .sensor =
        {
            .type =
            {
                .value = BSX_INPUT_ID_PRESSURE,
                .flags = DRIVER_TYPE_PHYSICAL_FLAG,
            },
        },
    },

    .physicalSource =
    {
        .sensor =
        {
            .type =
            {
                .value = BSX_INPUT_ID_PRESSURE,
                .flags = DRIVER_TYPE_PHYSICAL_FLAG,
            },
        },
    },

    .info =
    {
        .id = DRIVER_ID,
        .version = DRIVER_REV,
    },

    .type =
    {
        .value = SENSOR_TYPE_ALTITUDE_SENSOR,
        .flags = DRIVER_TYPE_VIRTUAL_FLAG,
        .wakeup_ap = FALSE,
        .no_decimation = FALSE,
        .on_change = FALSE,
    },

    .outputPacketSize = sizeof(output_t),
    .priority = PRIORITY_6, /* Low priority */
    .initialize = virt_altitude_initialize_sensor,
    .handle_sensor_data = virt_altitude_handle_sensor_data,
}

```

The driver's version number is 4.

Use the Parameter page 0x08, index 0x00 to set and get the reference pressure at sea level. It is 4 bytes register to contain an unsigned 32 bit value. Host can set and get this reference pressure configuration in run-time to get accurate altitude estimation.

VirtAltitude's trigger source is physical pressure sensor.

VirtAltitude's sensor ID is a visible ID, which means that it is visible to the HOST.

Figure 14: Driver descriptor overview

6.3 Selecting a driver ID

To add a new virtual sensor driver, the first step is to select the available virtual driver ID for compilation. Unless the driver to be developed is already included in the SDK, users may choose any unused 8-bit number.

In this excerpt, we have selected the driver ID **123** in the `CMakeLists.txt` file. Each driver has a unique driver ID.

6.4 Driver `CMakeLists.txt` file

The `CMakeLists.txt` file pulls in all the sources from the root directory of each driver. It is used by the build system while linking to associate the driver ID listed with a driver's object file. Additional driver IDs can be defined in the same way. This generic file typically needs no modification.

Take `drivers_custom/VirtAltitude/CMakeLists.txt` for example:

```
SET(DRIVER_ID 123)
get_filename_component( DRIVER_KEY ${CMAKE_CURRENT_LIST_DIR} NAME)

project(${DRIVER_KEY} C)

FILE(GLOB SOURCES "*.c")

ADD_ARC_DRIVER(${DRIVER_KEY} ${DRIVER_ID} ${SOURCES})
```

Figure 15: Driver `CMakeLists.txt`

6.5 Modifying the board configuration file

The example below describes how to include *VirtAltitude* in *Bosch_Shuttle3_BHI360_BMM150_BME680.fw* by editing the existing configuration file “*\$SDK/boards/Bosch_Shuttle3_BHI360_BMM150_BME680.cfg*” as shown below.

```
...
#Virtual Drivers,maxRate
217, -1.000000 # VirtTemperature: temperature depends on a physical temp source.
123, 16.000000 # VirtAltitude: depends on a physical pressure source.
184, -1.000000 # VirtPressure: pressure depends on a physical pressure source.
219, -1.000000 # VirtHumidity: humidity depends on a physical humidity source.
216, -1.000000 # VirtGas: gas depends on a physical gas source.
183, -1.000000 # VirtWakeupTemperature: wakeup temperature depends on a physical temp source.
218, -1.000000 # VirtWakeupPressure: wakeup pressure depends on a physical pressure source.
185, -1.000000 # VirtWakeupHumidity: wakeup humidity depends on a physical humidity source.
```

Link *VirtAltitude* (Driver ID: 123) into the *Bosch_Shuttle3_BHI360_BMM150_BME680.fw*

Figure 16: Modifying the board configuration file

6.6 Modifying the SDK configuration file

In order to build a firmware that contains the *VirtAltitude* virtual driver for the target board configuration the *common/config.7189_di03_rtos_bhi360.cmake* needs to be modified as shown below.

```
set (BOARDS
    Bosch_Shuttle3_BHI360
    Bosch_Shuttle3_BHI360_turbo
    Bosch_Shuttle3_BHI360_BMM150
    ...
    Bosch_Shuttle3_BHI360_BMM150_BME680
)
...
set (ENABLED_DRIVERS
    VirtBSXLeanDeviceOrientation
    VirtBSXCustomAccelDataSource
    VirtAltitude
    #Example Injection driver
    AccelInject
    ${DRIVERS_NO_SOURCE}
)
```

Add
Bosch_Shuttle3_BHI360_BMM150_BME680 to
the **BOARDS** variable.

Add *VirtAltitude* to the **ENABLED_DRIVERS**
variable.

Figure 17: Modifying the SDK configuration file

6.7 Build the custom firmware

As described in Chapter 2 for Windows and Chapter 3 for Linux system, trigger the respective build.

The custom firmware is now available in *release/gccfw* as *Bosch_Shuttle3_BHI360_BMM150_BME680.fw*. Like with the reference firmware, you can run the *bhy2cli* like below to load the firmware and view the lean orientation sensor’s output using the generic type of handle.

```
$ bhy2cli -a 161:"Altitude":4:s32 -b release\fw\Bosch_Shuttle3_BHI360_BMM150_BME680.fw -c 161:1
```

6.8 Altitude example

With new virtual sensor drivers in the firmware, a new sensor data parser should also be added to the host. An example of the virtual sensor `VirtAltitude` is provided separately.

The virtual altitude sensor's output unit is in centimeters.

For information on how to verify and evaluate the BHI360's virtual sensors, refer to the BHI360-BHI380 Evaluation Setup Guide.

```
typedef struct {  
    Sint32  altitude;  
} __attribute__((packed)) output_t;
```

Here defines `VirtAltitude` output data as 4 bytes, so HOST side should parse altitude data into 4 bytes.

Figure 18: Altitude output data structure

7 Integrating a library and applying it to the custom sensor driver

This chapter describes how to integrate a library and use it in a driver. The library can be found under *libs/template* and the driver under *drivers_custom/VirtIntegrateLibTemplate*. For information on the development of a custom driver, refer to Chapter 5 and 6.

7.1 Library directory structure

The library directory and its files must be in the *libs* directory of the SDK. The directory name should indicate the library name, for example, *libs/template*. Table 6 lists the contents of the library directory.

Table 6: Library directory content

Source in Library Directory	Description
CMakeLists.txt	Build description of the library
template.sdk.cmake	CMake file of the library
libtemplate.a	Library file
includes	Header files directory of the library

7.2 Implementing a sensor driver that uses the library

For implementing a sensor driver, the contents of the driver directory are shown in Table 7.

Table 7: Driver directory content

Source in Library Directory	Description
CMakeLists.txt	Build description of the driver
VirtIntegrateLibTemplate.c	Driver file

Content of CMakeLists.txt. Note the includes required.

```
SET(DRIVER_ID 111)

get_filename_component( DRIVER_KEY ${CMAKE_CURRENT_LIST_DIR} NAME)

project(${DRIVER_KEY} C)

FILE(GLOB SOURCES "*.c")

include_directories(../../libs/template/includes/)

add_definitions("-DDESCRIPTOR_NAME=virt_${DRIVER_KEY}_desc")

ADD_ARC_DRIVER(${DRIVER_KEY} ${DRIVER_ID} ${SOURCES})
```

Figure 19: CMakeLists.txt

7.3 Modifying the board configuration file

To build a firmware that contains library and driver, the board configuration file needs modification.

Using the *boards/Bosch_Shuttle3_BHI360.cfg* as reference.

```
#Virtual Drivers,maxRate
111, -1.000000 #VirtIntegrateLibTemplate: an example for integrate library
240, -1.000000 # VirtBSX: BSX depends on a programatic trigger source.
241, 400.000000 # VirtBSXAccel: accelerometer corrected data depends on VirtBSX.
209, 400.000000 # VirtBSXAccelOffset: accelerometer offset data depends on VirtBSX.
```

Figure 20: Modifying the board configuration file

7.4 Modifying the SDK configuration file

To build a firmware that contains library and driver, the SDK configuration file needs modification.

SDK configuration file *common/config.7189_di03_rtos_bhi360.cmake*

```
.....
SET(BUILD_LIBS
  ${EXPORT_LIB_SOURCE}
  ${EXPORT_LIB_BINARIES}
  DMA
  SensorInterfaceInit
  SensorInterfaceRAM
  DMAUnitTests
  HIDUnitTests
  HostInterfaceStreamingRAM

  BSXSupport
  OscTrim
  SensorCalibration
  Outerloop
  template
)

.....
# libraries linked to standard board images
set(BOARDS_LIBS
  MetawareDouble
  MetawarePrintf
  template
)
...

set(ENABLED_DRIVERS
  ....
  VirtIntegrateLibTemplate

# Example Injection driver
  AccelInject

  ${DRIVERS_NO_SOURCE}
)
.....
```

Figure 21: Modifying the SDK configuration file

7.5 Build the custom firmware

As described in Chapter 2 for Windows and Chapter 3 for Linux system, trigger the respective build. Then you can test the driver in the same way as described in Chapter 6.

8 Glossary

8.1 Virtual Sensor

A Virtual Sensor is a term used to identify the output of one or more algorithms. This output is available in the FIFO and can be identified and referenced by the host of the BHI360(BHI380) using a Sensor ID.

8.2 Driver ID

A virtual sensor driver is responsible for implementing the interface between the Software Framework and the algorithm, among other tasks. Each driver has a unique ID in the SDK which is referred to as the Driver ID. This Driver ID is used to select which driver can be included into a firmware build.

8.3 Sensor ID

The Sensor ID is a unique identifier for a Virtual sensor. This Sensor ID is defined as an 8-bit unsigned integer value. A list of all Virtual sensors and their corresponding sensor IDs also known as a FIFO event IDs are described in the datasheet in the table Overview of FIFO Event IDs. The Sensor ID is defined as part of the driver's descriptor.

9 Legal disclaimer

i. Engineering samples

Engineering Samples are marked with an asterisk (*) or (e). Samples may vary from the valid technical specifications of the product series contained in this data sheet. They are therefore not intended or fit for resale to third parties or for use in end products. Their sole purpose is internal client testing. The testing of an engineering sample may in no way replace the testing of a product series. Bosch Sensortec assumes no liability for the use of engineering samples. The Purchaser shall indemnify Bosch Sensortec from all claims arising from the use of engineering samples.

ii. Product use

Bosch Sensortec products are developed for the consumer goods industry. They may only be used within the parameters of this product data sheet. They are not fit for use in life-sustaining or safety-critical systems. Safety-critical systems are those for which a malfunction is expected to lead to bodily harm, death or severe property damage. In addition, they shall not be used directly or indirectly for military purposes (including but not limited to nuclear, chemical or biological proliferation of weapons or development of missile technology), nuclear power, deep sea or space applications (including but not limited to satellite technology).

The resale and/or use of Bosch Sensortec products are at the purchaser's own risk and his own responsibility. The examination of fitness for the intended use is the sole responsibility of the purchaser.

The purchaser shall indemnify Bosch Sensortec from all third party claims arising from any product use not covered by the parameters of this product data sheet or not approved by Bosch Sensortec and reimburse Bosch Sensortec for all costs in connection with such claims.

The purchaser accepts the responsibility to monitor the market for the purchased products, particularly with regard to product safety, and to inform Bosch Sensortec without delay of all safety-critical incidents.

iii. Application examples and hints

With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the device, Bosch Sensortec hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights or copyrights of any third party. The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics. They are provided for illustrative purposes only and no evaluation regarding infringement of intellectual property rights or copyrights or regarding functionality, performance or error has been made.

10 Document history

Rev. No	Chapter	Description of modification/changes	Date
1.0	All	Main release	2020-02-04
1.1	All	Updated references	2020-02-24
1.2	All	Added Glossary	2020-04-24
1.3	All	Updated bhy2cli command formats	2020-05-18
1.4	5.9	Updated bhy2cli command to enable lean orientation	2021-10-20
1.5	2.1; 2.3; 3.1	Update compiler version number	2023-04-18



Bosch Sensortec GmbH
Gerhard-Kindler-Straße 9
72770 Reutlingen / Germany

www.bosch-sensortec.com

Modifications reserved
Preliminary - specifications subject to change without notice
Document number: BST-BHI360-BHI380-AN000-05
Revision_1.5