



**Hi3861 V100 / Hi3861L V100 Third-Party Software**

## **Porting Guide**

<b>Issue</b>	<b>01</b>
<b>Date</b>	<b>2020-04-30</b>

**Copyright © HiSilicon (Shanghai) Technologies Co., Ltd. 2020. All rights reserved.**

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of HiSilicon (Shanghai) Technologies Co., Ltd.

## **Trademarks and Permissions**



**HISILICON**, and other HiSilicon icons are trademarks of HiSilicon Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

## **Notice**

The purchased products, services and features are stipulated by the contract made between HiSilicon and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided “AS IS” without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

## **HiSilicon (Shanghai) Technologies Co., Ltd.**

Address: New R&D Center, 49 Wuhe Road,  
Bantian, Longgang District,  
Shenzhen 518129 P. R. China

Website: <https://www.hisilicon.com/en/>

Email: [support@hisilicon.com](mailto:support@hisilicon.com)



# About This Document

## Purpose

This document describes how to port third-party software to the software development kit (SDK) and provides frequently asked questions (FAQs) and troubleshooting methods.

## Related Versions

The following table lists the product versions related to this document.

Product Name	Version
Hi3861	V100
Hi3861L	V100



## Intended Audience

This document is intended for software developers, who are expected to have necessary skills in:




- Build tools and syntax (including SCons and Makefile)
- Environment setup

## Symbol Conventions

The symbols that may be found in this document are defined as follows.

Symbol	Description
	Indicates a hazard with a high level of risk which, if not avoided, will result in death or serious injury.
	Indicates a hazard with a medium level of risk which, if not avoided, could result in death or serious injury.



Symbol	Description
 CAUTION	Indicates a hazard with a low level of risk which, if not avoided, could result in minor or moderate injury.
 NOTICE	Indicates a potentially hazardous situation which, if not avoided, could result in equipment damage, data loss, performance deterioration, or unanticipated results. NOTICE is used to address practices not related to personal injury.
 NOTE	Supplements the important information in the main text. NOTE is used to address information not related to personal injury, equipment damage, and environment deterioration.

## Change History

Issue	Date	Change Description
01	2020-04-30	This issue is the first official release. <ul style="list-style-type: none"><li>In <a href="#">1.2 SCons</a>, the porting procedure is updated.</li><li>In <a href="#">1.3.1 Independent Build of Third-Party Software</a>, the porting procedure is updated.</li></ul>
00B01	2020-01-15	This issue is the first draft release.



---

# Contents

---

<b>About This Document.....</b>	<b>i</b>
<b>1 Porting Guidance.....</b>	<b>1</b>
1.1 Overview.....	1
1.2 SCons.....	1
1.3 Make.....	4
1.3.1 Independent Build of Third-Party Software.....	4
<b>2 FAQs.....</b>	<b>5</b>



# 1 Porting Guidance

## 1.1 Overview

### 1.2 SCons

### 1.3 Make

## 1.1 Overview

In the SDKs of Hi3861 V100 and Hi3861L V100, SCons is used as the build tool. Therefore, you are advised to use SCons to port third-party libraries to ensure compilation integrity and continuity. Some libraries use the Make utility as the build tool, and the cost of using SCons to replace the porting is high. Therefore, this document provides the porting method and idea of using Make. Developers can select a porting solution based on the actual development situation.

## 1.2 SCons

### NOTE

The recommended porting method is to use SCons for build.

The following describes how to port **mbdttls-x.x**:

- Step 1** Place the third-party software directory in the **third\_party** directory, for example, **third\_party/mbdttls-x.x**.
- Step 2** Add the first-level SConscript in the **mbdttls-x.x** directory. You are advised to copy the package from the third-party software directory in the SDK. For example, copy **third\_party/cjson/SConscript** to **third\_party/mbdttls-x.x**. The SConscript queries the source code directory based on the configuration and generates .a library files.
- Step 3** Add the second-level SConscript in the **mbdttls-x.x/library** directory. You are advised to copy SConscript in the second-level directory from the third-party software directory in the SDK. For example, copy **cjson/cjson\_utils/SConscript** to **mbdttls-x.x/library/SConscript**. If the third-party software has multiple source code directories, you need to copy a second-level SConscript for each source code directory and execute [Step 5](#). If the source code directory structure is complex, see [Step 4](#).



- Step 4** (This step requires that you be familiar with the compilation of the SCons script.) If the source code directory is deep or contains many directories, you need to compile the SCons script independently. To reduce the porting difficulty, you are advised to place all source code and source code directories in the new **src** directory. In this way, you do not need to modify the first-level SConscript. You only need to modify the second-level SConscript to access each source code directory in traversal mode, and compile all .c or .s files.

The following example code is a script for the second-level SConscript to traverse all directories and generate .o files for developer's reference.

```
#!/usr/bin/env python3
# coding=utf-8

import os
Import('env')

env = env.Clone()

src_path = []
for root, dirs, files in os.walk('.'):
    src_path.append(root)

objs = []
for src in src_path:
    objs += env.Object(Glob(os.path.join(src, '*.c')))
    objs += env.Object(Glob(os.path.join(src, '*.S')))

Return('objs')
```

- Step 5** Modify the configuration file **build/script/common\_env.py**.

1. (Mandatory) Add the name of the new third-party software to **compile\_module**.

```
compile_module = ['drv', 'sys', 'os', 'wpa', 'lwip', 'at', 'mbedtls']
```

2. (Mandatory) Add the path of the new third-party software to **module\_dir**. Type the relative path of the project root directory. The path contains the first-level SConscript.

```
module_dir = {
    'drv': os.path.join('platform', 'drivers'),
    'mbedtls': os.path.join('third_party', 'mbedtls-x.x'), # Type the relative path of
the project root directory. The path contains the first-level SConscript.
    ...
}
```

3. (Mandatory) Add the library file name and source code file directory of the new third-party software to **proj\_lib\_cfg**. The format is module name:{*name of the generated library file*:[*path of the source code directory*]}. The path of the source code directory is the relative path of the first-level SConscript. The source code directory contains the second-level SConscript.

```
proj_lib_cfg = {
    #os modules
    'os':{
        'res_cfg': [
            os.path.join('kernel', 'redirect')
        ]
    },
    #third parties and components
    'mbedtls':{ 'mbedtls': ['library'] }, #Format 'module name':{ 'name of the generated library
file': [ 'path of the source code directory' ]}. The path of the source code directory is the
relative path of the first-level SConscript. The source code directory contains the second-
```



```
level SConscript.
```

```
} ...
```

4. (Optional) Add the compilation macro definition that needs to be specified in the compilation phase to **proj\_environment['defines']**. If the compilation macro definition does not exist, skip this step.

```
'defines':{
    'common':[ ('PRODUCT_CFG_SOFT_VER_STR', r\"%s\"'%product_soft_ver_str),
                ('PRODUCT_CFG_BUILD_DATE',r\"%s\"'%cur_date),
                ('PRODUCT_CFG_BUILD_TIME',r\"%s\"'%cur_time),
                ('PRODUCT_CFG_BUILD_TIME_YEAR',r\"%s\"'%cur_time_year),
                ('PRODUCT_CFG_BUILD_TIME_MONTH',r\"%s\"'%cur_time_month),
                ('PRODUCT_CFG_BUILD_TIME_DAY',r\"%s\"'%cur_time_day),
                'CYGPKG_POSIX_SIGNALS',
                '__ECOS__',
                '__RTOS__',
                'PRODUCT_CFG_HAVE_FEATURE_SYS_ERR_INFO',
                '__LITEOS__',
                'LIB_CONFIGURABLE',
                'LOSCFG_SHELL',
                'CONFIG_DRIVER_HI1131',
                'HISI_CODE_CROP',
                'LOSCFG_CACHE_STATICS',#This option is used to control whether Cache hit
ratio statistics are supported.
                '#LOG_PRINT_SZ',#This option is used to control whether print on shell
                'CUSTOM_AT_COMMAND',
                'LOS_COMPILE_LDM'
            ],
    'mbedtls':[], # The list can be left empty.
    ...
}
```

5. (Optional) Add a compilation option to **proj\_environment['opts']**. If the compilation option does not exist, skip this step.
6. (Optional) Add the header file of Huawei LiteOS referenced by the new third-party software to **proj\_environment['liteos\_inc\_path']**. If the header file does not exist, skip this step.
7. (Mandatory) Add the header file of a non-Huawei LiteOS system referenced by the new third-party software to **proj\_environment['common\_inc\_path']** and pay attention to the format ('#').

```
'common_inc_path':{
    'common':[
        os.path.join('#', 'include'),
        os.path.join('#', 'platform', 'include'),
        os.path.join('#', 'platform', 'system', 'include'),
        os.path.join('#', 'config'),
        os.path.join('#', 'config', 'nv'),
    ],
    'mbedtls':[os.path.join('#', 'third_party', 'mbedtls-x.x', 'include', 'mbedtls')],
    ...
}
```

----End





## 1.3 Make

### NOTE

If the third-party software is built based on the Make utility and the porting cost is high, you can port the software by referring to this section.

### 1.3.1 Independent Build of Third-Party Software

The original makefile of the third-party software is used for independent build. The **make** command needs to be manually executed to compile the third-party library file, without depending on the SCons compilation script. The following uses porting **mbdttls-x.x** as an example:

- Step 1** Place the third-party software directory in the **third\_party** directory, for example, **third\_party/mbdttls-x.x**.
- Step 2** Run the **make** command in the **mbdttls-x.x** directory.
- Step 3** Copy the library file (for example, **libmbdttls.a**) generated by running the **make** command to the **build/libs** directory or the **app** project directory, and modify the **app.json** file to further reduce the dependency on the SDK. The build system automatically links all library files in **build/libs** as required.
- Step 4** If the source code project contains header files that depend on third-party libraries, add the header file search directory to the referenced module and modify the configuration file **build/scripts/common\_env.py**.

For example, if the source code in **components/at** depends on the header file **third\_party/mbdttls-x.x/include/mbdttls/certs.h**, add the search path of the header file of the **at** module to **build/scripts/common\_env.py**.

```
...
'common_inc_path':{
    ...
    'at':[
        os.path.join('#', 'third_party', 'mbdttls-x.x', 'include', 'mbdttls'),
        ...
    ],
    ...
}
```

----End



## 2 FAQs

- When SCons is used for build, an error message is displayed during compilation indicating that the header file cannot be found.  
Solution: Add the path of the header file to **proj\_environment['common\_inc\_path']** in **build/script/common\_env.py**. For details, see [Step 5.7](#).
- When SCons is used for build, no new third-party software library file is generated during compilation.  
Solution:
  - If the source code of the third-party software is not compiled, ensure that the configuration in [Step 5.1](#) is complete.
  - If only the source code of some third-party software is compiled, ensure that the configuration in [Step 5.3](#) is complete.
- When SCons is used for build, only some .c files need to be compiled in the source code directory of the new third-party software.  
The following three methods are provided:
  - Modify the second-level SConscript to traverse all .c files and change them to the .c files to be compiled (similar to the compilation of a makefile). The following sample code is for reference only.

```
wps_srcs = ['wps.c']
utils_srcs = ['base64.c']
all_srcs = []
all_srcs.extend(wps_srcs)
all_srcs.extend(utils_srcs)
objs = env.Object(all_srcs)
Return('objs')
```
  - Save the .c files to be compiled to another directory, place the second-level SConscript to the new directory, and change the source code path in **build/script/comm\_env.py**.
  - Delete unnecessary source code files from the directory.
- When SCons is used for build, there are multiple levels between the source code file and the source code directory in the new third-party software directory structure. For example, in **example/app/good/src/example.c**, the first-level SConscript is in **example/app**, while the second-level SConscript is in **example/app/good/src**.



Solution: Add the source code path (['good/src']) to **proj\_lib\_cfg** in **build/script/common\_env.py**.