



## **Hi3861 V100 / Hi3861L V100 SDK Development Environment Setup**

# **User 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 the SDK development environment (including SDK build and application development) of Hi3861 V100 and Hi3861L V100, facilitating the build of executable files for secondary development.

## Related Versions

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

Product Name	Version
Hi3861	V100
Hi3861L	V100



## Intended Audience

The document is intended for:



- Technical support engineers
- Software development engineers

## 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	<p>This issue is the first official release.</p> <ul style="list-style-type: none"><li>• In <a href="#">2.1 SDK Directory Structure</a>, the descriptions of the <b>Makefile</b>, <b>non_factory.mk</b>, and <b>factory.mk</b> directories are added to <a href="#">Table 2-1</a>, and <a href="#">Figure 2-1</a> is updated.</li><li>• In <a href="#">2.2.1 Build Method</a>, the description of the <b>factory</b> parameter is added to <a href="#">Table 2-2</a>, and <a href="#">Table 2-3</a> is updated.</li><li>• In <a href="#">2.2.2 Menuconfig</a>, <a href="#">Figure 2-2</a> and <a href="#">Table 2-5</a> are updated.</li><li>• In <a href="#">2.2.3 Precautions</a>, the description about the message indicating no permission when <b>./build.sh</b> is executed is updated.</li><li>• In <a href="#">2.3 Building the SDK (Makefile)</a>, the description of compiling the factory test program is added. The description of separately compiling library files is deleted.</li><li>• <a href="#">3.1 Creating a Directory Structure</a> is updated.</li><li>• In <a href="#">3.2 Developing Code</a>, the description of compiling <b>my_demo</b> for code debugging is added.</li><li>• <a href="#">3.3 app_main</a> is added.</li><li>• In <a href="#">4.2.2 Build Configuration</a>, <a href="#">Step 5</a> is updated.</li></ul>
00B08	2020-04-03	In <a href="#">Table 2-5</a> of <a href="#">2.2.2 Menuconfig</a> , the description of the OTA Settings menu is added.



Issue	Date	Change Description
00B07	2020-03-27	<ul style="list-style-type: none"><li>• In <a href="#">2.2 Building the SDK (Scons)</a>, the title is updated.</li><li>• <a href="#">2.3 Building the SDK (Makefile)</a> is added.</li></ul>
00B06	2020-03-06	<ul style="list-style-type: none"><li>• In <a href="#">1.2.2 Installing the Python Environment</a>, the description of using <b>python setup.py install</b> for installation and the description of using multiple pythons in the build environment are added.</li><li>• In <a href="#">2.2.3 Precautions</a>, the note that a package cannot be found during build is added.</li></ul>
00B05	2020-02-27	<ul style="list-style-type: none"><li>• <a href="#">Table 2-2</a> is added.</li><li>• In <a href="#">2.2.1 Build Method</a>, the path of the command execution script is added.</li><li>• In <a href="#">Table 2-3</a>, the description of <b>Hi3861_demo_ota_2.bin</b> is updated.</li><li>• <a href="#">Table 2-5</a> is updated.</li><li>• In <a href="#">3.4.2 Build Configuration Files, Step 3</a> is updated.</li><li>• <a href="#">4 SDK Component Customization</a> is added.</li></ul>
00B04	2020-02-12	<ul style="list-style-type: none"><li>• In <a href="#">1.2 Setting Up the Linux Development Environment</a>, the description of the cross compiler is updated.</li><li>• In <a href="#">1.2.1 Installing the Cross Compiler</a>, the description of the compiler installation package is updated in <a href="#">Step 1</a>, and the description of the compiler version number is added in <a href="#">Step 3</a>.</li><li>• In <a href="#">1.2.3 Installing SCons</a>, the recommended SCons version and the directory running command in <a href="#">Step 1</a> are updated; <a href="#">Figure 1-4</a> is updated; FAQs for SCons installation are added.</li><li>• In <a href="#">2.1 SDK Directory Structure</a>, the description of the <b>boot</b> directory in <a href="#">Table 2-1</a> is added, and <a href="#">Figure 2-1</a> is updated.</li><li>• In <a href="#">2.2.1 Build Method</a>, <a href="#">Table 2-3</a> is updated.</li><li>• In <a href="#">2.2.2 Menuconfig</a>, <a href="#">Table 2-5</a> is updated.</li></ul>



Issue	Date	Change Description
00B03	2020-01-15	<ul style="list-style-type: none"><li>• <a href="#">Figure 1-1</a> is updated.</li><li>• In <a href="#">1.2.2 Installing the Python Environment</a>, the description of installing Six and ECDSA in <a href="#">Step 5</a> is added.</li><li>• In <a href="#">Table 2-1</a>, the description of the <b>app</b> and <b>tools</b> directories is updated.</li><li>• <a href="#">Table 2-3</a> is updated.</li><li>• In <a href="#">2.2.2 Menuconfig</a>, the description of the menuconfig operations, menuconfig command help bar, and menuconfig configuration items is added.</li><li>• In <a href="#">Table 3-1</a>, the file name of the SDK linkage script and the description of <b>app/my_demo/SConscript</b> and <b>app/my_demo/src/SConscript</b> are updated.</li><li>• In <a href="#">Table 3-2</a>, the description of LD_DIRS is added.</li></ul>
00B02	2019-12-19	<ul style="list-style-type: none"><li>• In <a href="#">1.2.2 Installing the Python Environment</a>, the procedure for installing PyCryptodome is added.</li><li>• In step 1 of <a href="#">1.2.3 Installing SCons</a>, the recommendation of SCons 3.0.1+ is added.</li><li>• <a href="#">2.2.2 Menuconfig</a> is updated.</li><li>• In <a href="#">3.1 Creating a Directory Structure</a>, the procedure for creating the <b>app/my_demo/nv</b> directory is deleted.</li><li>• <a href="#">3.2 Developing Code</a> is updated.</li><li>• In <a href="#">Table 3-1</a>, the description of <b>build/scripts/common_env.py</b> is updated and the description of <b>app/my_demo/app.json</b> is added.</li><li>• <a href="#">3.4.2 Build Configuration Files</a> is updated.</li></ul>
00B01	2019-11-15	This issue is the first draft release.



# Contents

<b>About This Document.....</b>	<b>i</b>
<b>1 Setting Up the Development Environment.....</b>	<b>1</b>
1.1 Overview.....	1
1.2 Setting Up the Linux Development Environment.....	2
1.2.1 Installing the Cross Compiler.....	2
1.2.2 Installing the Python Environment.....	3
1.2.3 Installing SCons.....	4
<b>2 Compiling the SDK.....</b>	<b>6</b>
2.1 SDK Directory Structure.....	6
2.2 Building the SDK (Scons).....	7
2.2.1 Build Method.....	7
2.2.2 Menuconfig.....	9
2.2.3 Precautions.....	12
2.3 Building the SDK (Makefile).....	12
<b>3 Creating an App.....</b>	<b>13</b>
3.1 Creating a Directory Structure.....	13
3.2 Developing Code.....	13
3.3 app_main.....	13
3.4 Configuring Build Attributes.....	14
3.4.1 Basic Build Scripts.....	14
3.4.2 Build Configuration Files.....	14
<b>4 SDK Component Customization.....</b>	<b>17</b>
4.1 Build System Overview.....	17
4.2 Component Creation.....	19
4.2.1 Build Output.....	19
4.2.2 Build Configuration.....	20
4.3 Menuconfig Customization.....	23



# 1 Setting Up the Development Environment

---

## 1.1 Overview

### 1.2 Setting Up the Linux Development Environment

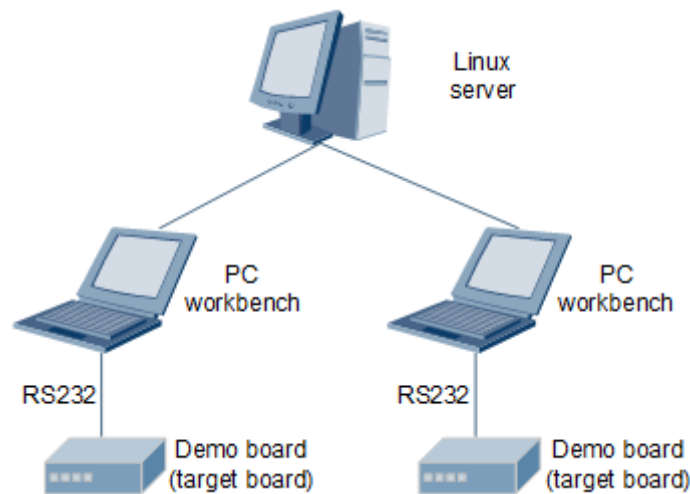
## 1.1 Overview

The typical SDK development environment includes:

- Linux server  
A Linux server is used to set up a cross compilation environment to compile executable code that can run on the target board.
- Workbench  
A workbench is used to burn and debug the target board. It connects to the target board through the serial port. Developers can burn the image and debug programs of the target board on the workbench. Terminal tools must be installed on the workbench to log in to the Linux server and target board and view the print information of the target board. Generally, the workbench runs the Windows or Linux operating system (OS). The terminal tools running on the workbench include SecureCRT, PuTTY, and miniCom. You need to download the software from their official websites.
- Target board  
This document takes the HiSilicon demo board as an example. The demo board is connected to the workbench through the USB-to-serial port. The workbench burns the demo board image generated after cross-compiling to the demo board through the serial port. See [Figure 1-1](#).



Figure 1-1 SDK development environment



## 1.2 Setting Up the Linux Development Environment

It is recommended that Ubuntu 16.04 or later be used for the Linux OS, Bash be used for the shell, and `hcc_riscv32` be used for the cross compiler. The build toolchain also contains Python and SCons.

### 1.2.1 Installing the Cross Compiler

- Step 1** Obtain the compiler installation package **`hcc_riscv32.tar.gz`**. (The installation package contains the version information. The document name is simple. The actual name is subject to the released compiler version.)
- Step 2** Install the compiler in the system directory and add it to the environment variable (requiring the **root** or **sudo** permission).

For example, decompress the compiler to the installation directory. The following uses **`/toolchain`** as an example:

1. Copy the compressed package to the **`/toolchain`** directory and decompress the package by running **`tar zxvf hcc_riscv32.tar.gz`**.
2. Modify the permission on the compiler installation directory recursively:  
**`chmod -R 755 /toolchain`**.
3. Set environment variable **`vim /etc/profile`** and add **`export PATH=/toolchain/hcc_riscv32/bin:$PATH`**.
4. Make the environment variable take effect: **`source /etc/profile`**.

- Step 3** Enter **`riscv32-unknown-elf-gcc -v`** in the shell command line. If the compiler version is displayed correctly, the compiler is installed successfully. [Figure 1-2](#) shows the installation success. The compiler version depends on the actual release.

**Figure 1-2 Successful compiler installation**

```
wn-elf/7.3.0/lto-wrapper
Target: riscv32-unknown-elf
Configured with: /usr1/heterogeneous_compiler_codesign/build/hcc_riscv32/../../o
pen_source/hcc_riscv32_build_src/gcc-7.3.0/configure --build=x86_64-pc-linux-gnu
--host=x86_64-pc-linux-gnu --target=riscv32-unknown-elf --with-arch=rv32imc --w
ith-abi=ilp32 --disable-__cxa_atexit --disable-libgomp --disable-libmudflap --en
able-libssp --disable-libstdc++-pch --disable-nls --disable-shared --disable-thr
eads --disable-multilib --enable-poison-system-directories --enable-languages=c,
c++ --with-headers=/usr1/heterogeneous_compiler_codesign/build/hcc_riscv32/riscv
32_elf_build_dir/hcc_riscv32/riscv32-unknown-elf/include --with-gnu-as --with-gn
u-ld --with-newlib --with-pkgversion='Heterogeneous Compiler&Codesign V100R003C0
0SPC200B023' --with-build-sysroot=/usr1/heterogeneous_compiler_codesign/build/hc
c_riscv32/riscv32_elf_build_dir/hcc_riscv32/riscv32-unknown-elf --with-gmp=/usr1
/heterogeneous_compiler_codesign/build/hcc_riscv32/riscv32_elf_build_dir/obj/hos
t-libc/usr --with-mpfr=/usr1/heterogeneous_compiler_codesign/build/hcc_riscv32/r
iscv32_elf_build_dir/obj/host-libc/usr --with-mpc=/usr1/heterogeneous_compiler_c
odesign/build/hcc_riscv32/riscv32_elf_build_dir/obj/host-libc/usr --with-isl=/us
r1/heterogeneous_compiler_codesign/build/hcc_riscv32/riscv32_elf_build_dir/obj/h
ost-libc/usr --with-build-time-tools=/usr1/heterogeneous_compiler_codesign/build
/hcc_riscv32/riscv32_elf_build_dir/hcc_riscv32/riscv32-unknown-elf/bin --with-sy
stem-zlib
Thread model: single
gcc version 7.3.0 (Heterogeneous Compiler&Codesign V100R003C00SPC200B023)
```

----End

## 1.2.2 Installing the Python Environment

- Step 1** On the Linux terminal, run the **python3 -V** command to check the Python version. Python 3.7 or later is recommended.
- Step 2** If the Python version is too early, run the **sudo apt-get update** command to update the system to the latest version, or run the **sudo apt-get install python3 -y** command to install Python3 (requiring the **root** or **sudo** permission). After the installation, check the Python version again.

If the version requirements are still not met, download the source code package of the required version from <https://www.python.org/downloads/source/>. For details about how to download and install the source code package, see <https://wiki.python.org/moin/BeginnersGuide/Download> and **README** in the source code package.

- Step 3** Run the **sudo apt-get install python3-setuptools python3-pip -y** command to install the Python package management tool (requiring the **root** or **sudo** permission).
- Step 4** Run the **sudo pip3 install kconfiglib** command (requiring the **root** or **sudo** permission) to install Kconfiglib 13.2.0 or download the .whl file (for example, **kconfiglib-13.2.0-py2.py3-none-any.whl**) from <https://pypi.org/project/kconfiglib>, and run the **pip3 install kconfiglib-xxx.whl** command to install the kconfiglib (requiring the **root** or **sudo** permission). Alternatively, download the source code package to the local PC, decompress it, and run the **python setup.py install** command to install the kconfiglib as (requiring the **root** or **sudo** permission). **Figure 1-3** shows the installation completion GUI.

**Figure 1-3** Example of Kconfiglib installation completion

```
# pip3 install kconfiglib-13.2.0-py2.py3-none-any.whl
ll kconfiglib-13.2.0-py2.py3-none-any.whl
Processing ./kconfiglib-13.2.0-py2.py3-none-any.whl
Installing collected packages: kconfiglib
Successfully installed kconfiglib-13.2.0
tools/menuconfig#
```

**Step 5** Install the Python component packages on which the upgrade file signing depends.

- **PyCryptodome**  
Download the .whl file (for example, **pycryptodome-3.7.3-cp37-cp37m-manylinux1\_x86\_64.whl**) from <https://pypi.org/project/pycryptodome/#files> and run the **pip3 install pycryptodome-xxx.whl** command to install the file (requiring the **root** or **sudo** permission). Alternatively, download the source code package to the local PC, decompress it, run the **python setup.py install** command to install the file, and perform the installation (requiring the **root** or **sudo** permission). After the installation is complete, the message "Successfully installed pycryptodome-3.7.3" is displayed.
- **Six**  
Download the .whl file (for example, **six-1.12.0-py2.py3-none-any.whl**) from <https://pypi.org/project/six/> and run the **pip3 install six-xxx.whl** command to install the file (requiring the **root** or **sudo** permission). Alternatively, download the source code package to the local PC, decompress it, run the **python setup.py install** command to install the file, and perform the installation (requiring the **root** or **sudo** permission). After the installation is complete, the message "Successfully installed six-xxx" is displayed.
- **ECDSA**  
Download the .whl file (for example, **ecdsa-0.14.1-py2.py3-none-any.whl**) from <https://pypi.org/project/ecdsa/> and run the **pip3 install ecdsa-0.14.1-py2.py3-none-any.whl** command to install the file (requiring the **root** or **sudo** permission). Alternatively, download the source code package to the local PC, decompress it, run the **python setup.py install** command to install the file, and perform the installation (requiring the **root** or **sudo** permission). After the installation is complete, the message "Successfully installed ecdsa-0.14.1" is displayed.  
  
Note: The installation of the ECDSA depends on Six. You need to install Six before installing the ECDSA.

----End

#### NOTE

If the build environment contains multiple Pythons, especially Pythons of the same version, and you cannot identify the version in use, you are advised to use the source code of the component package when installing the Python component package.

## 1.2.3 Installing SCons

**Step 1** Open the Linux terminal and run the **sudo apt-get install scons -y** command (requiring the **root** or **sudo** permission).

If the installation package cannot be found, download the source code package from <https://scons.org/pages/download.html>. Decompress the source code



package to any directory, go to the directory, and run the **sudo python3 setup.py install** command (requiring the **root** or **sudo** permission). Wait until the installation is complete. The recommended SCons version is 3.0.4 or later.

**Step 2** Run the **scons -v** command to check whether the installation is successful. [Figure 1-4](#) shows the installation success result.

**Figure 1-4** Successful SCons installation

```
SCons by Steven Knight et al.:  
    script: v3.0.4.3a41ed6b288cee8d085373ad7fa02894  
kufra  
    engine: v3.0.4.3a41ed6b288cee8d085373ad7fa02894  
kufra  
    engine path: ['/usr/local/lib/scons/SCons']  
Copyright (c) 2001 - 2019 The SCons Foundation
```

----End

FAQs for SCons installation:

- If the terminal displays "SCons 3.0.4 or greater required, but you have SCons 3.0.x" when SDK build starts, it indicates that the SCons version of the build host is too early, and version upgrade is required.
- If the terminal displays "SCons version x.x.x does not run under Python version 3.x.x. Python 3 is not yet supported. " when SDK build starts, it indicates that the SCons version of the build host is too early to support the Python3 syntax, and version upgrade is required.
- If the terminal displays "parallel builds are unsupported by this version of Python;" when SDK build starts, it indicates that the threadless module is removed from Python 3.7. As a result, versions earlier than SCons 3.0.1 do not support multi-thread build. You can upgrade the SCons to solve this problem. Alternatively, you can continue to use single-thread build, which does not affect the build result.



# 2 Compiling the SDK

[2.1 SDK Directory Structure](#)

[2.2 Building the SDK \(Scons\)](#)

[2.3 Building the SDK \(Makefile\)](#)

## 2.1 SDK Directory Structure

**Table 2-1** shows the SDK directory structure.

**Table 2-1** SDK root directory

Directory	Description
app	Application layer code (including the demo program)
boot	Flash bootloader code
build	Library files, link files, and configuration files required for SDK build
components	SDK components
config	SDK system configuration files
documents	Documents (including SDK description documents)
include	API header files
output	Target files and intermediate files generated during build (including library files, printed logs, and binary files)
platform	Files related to the SDK platform, including images and kernel driver modules
third_party	Third-party software



Directory	Description
tools	Tools provided by the SDK for the Linux and Windows OSs, including the NV make tool, signing tool, and menuconfig
SConstruct	SCons build script
build.sh	Build script for startup, which can be customized using <b>sh build.sh menuconfig</b>
Makefile	Makefile. You can run the <b>make</b> or <b>make all</b> command to start building.
non_factory.mk	Build script of the non-factory-test version
factory.mk	Build script of the factory-test version

**Figure 2-1** shows the root directory after the SDK is decompressed.

**Figure 2-1** SDK decompression

```
.  
..  
app  
boot  
build  
build.sh  
components  
config  
documents  
factory.mk  
include  
Makefile  
non_factory.mk  
platform  
SConstruct  
third_party  
tools
```

## 2.2 Building the SDK (Scons)

### 2.2.1 Build Method

Run the **./build.sh** command in the root directory to build the SDK. [Table 2-2](#) describes the build commands.



**Table 2-2** List of build.sh parameters

Parameter	Sample	Description
None	./build.sh	Starts incremental build. By default, the <b>demo</b> project is built.
all	./build.sh all	Starts full build. By default, the <b>demo</b> project is built.
Name of the app project directory	./build.sh demo	Inputs the app project directory name to start incremental build. The default project is <b>demo</b> .
clean	./build.sh clean	Cleans up the intermediate files and burning files generated during build.
menuconfig	./build.sh menuconfig	Starts the menuconfig GUI.

The build result is stored in the **output\bin** directory, as shown in [Table 2-3](#).

**Table 2-3** Demo build result (compression upgrade is used as an example)

Directory	Description
Hi3861_boot_signed.bin	Bootloader file which has been signed
Hi3861_boot_signed_B.bin	Bootloader backup file which has been signed
Hi3861_demo.asm	Kernel ASM file
Hi3861_demo.map	Kernel MAP file
Hi3861_demo.out	Kernel output file
Hi3861_demo_allinone.bin	Burning file for the production equipment test (including the independent burning program and loader program)
Hi3861_demo_burn.bin	Burning file. You are advised to use <b>Hi3861_demo_allinone.bin</b> as the burning program.
Hi3861_demo_flash_boot_ota.bin	FlashBoot upgrade file
Hi3861_demo_ota.bin	Kernel upgrade file
Hi3861_demo_vercfg.bin	Kernel and boot version number file



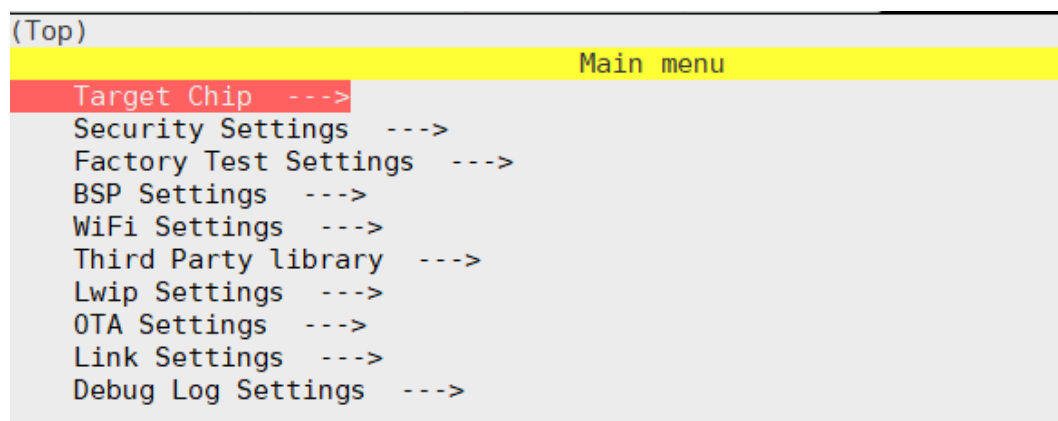
Directory	Description
Hi3861_loader_signed.bin	Load file used by the burning tool. You are advised to use <b>Hi3861_demo_allinone.bin</b> as the burning program.

Note: **Hi3861\_demo\_allinone.bin** is the program to be burnt to the board.

## 2.2.2 Menuconfig

Run the **sh build.sh menuconfig** script to start the menuconfig program. You can use the menuconfig program to configure build and system functions. The SDK integrates the default configurations. However, you are advised to perform the configurations when the SDK runs for the first time to reduce problems. You can run the **sh build.sh menuconfig** command to modify the configuration at any time.

Figure 2-2 Menuconfig GUI



Note: The actual GUI may be different.

**Table 2-4** describes the operations on the menuconfig window. You can press shortcut keys on the menuconfig window to perform the operations.

**Table 2-4** Common menuconfig commands

Shortcut Key	Description
Space and Enter	Selects or deselects.
ESC	Returns to the upper-level menu and exit the page.
Q	Exits the page.
S	Saves the configuration.





Shortcut Key	Description
F	Displays the help menu.

For details about all commands, see the official description in the lower part of the menuconfig window, as shown in [Figure 2-3](#).

**Figure 2-3** Help bar of menuconfig commands

[Space/Enter] Toggle/enter	[ESC] Leave menu	[S] Save
[O] Load	[?] Symbol info	[/] Jump to symbol
[F] Toggle show-help mode	[C] Toggle show-name mode	[A] Toggle show-all mode
[Q] Quit (prompts for save)	[D] Save minimal config (advanced)	

[Table 2-5](#) describes the menuconfig configuration items.

**Table 2-5** Menuconfig configuration items

Menu	Configuration Item	Description
Target Chip	Hi3861	Specifies the chip type.
	Hi3861L	
Security Settings	Signature Algorithm for bootloader and upgrade file	Signs the boot and upgrade files. RSA V15, RSA PSS, ECC, and SHA256 signing modes are supported.
	boot ver(value form 0 to 16)	Sets the boot version.
	kernel ver(value form 0 to 48)	Sets the kernel version.
	TEE HUKS support	Supports the TEE HUKS interface.
	FLASH ENCRPT suppot	Supports flash encryption.
Factory Test Settings	factory test enable	Enables factory tests.
BSP Settings	i2c driver support	Supports the I <sup>2</sup> C driver.
	i2s driver support	Supports the I <sup>2</sup> S driver.
	SPI driver support	Supports the SPI driver.
	DMA driver support	Supports the DMA driver.
	SDIO driver support	Supports the SDIO driver.



Menu	Configuration Item	Description
	SPI support DMA	The SPI driver supports the DMA transfer mode.
	UART support DMA	The UART driver supports the DMA transfer mode.
	PWM driver support	Supports the PWM driver.
	PWM hold after reboot	Maintains the PWM status after soft reset.
	Enable AT command	Supports AT commands.
	Enable file system	Supports the file system.
	Enable uart0 IO mux config	Supports UART0.
	Enable uart1 IO mux config	Supports UART1.
	Enable uart2 IO mux config	Supports UART2.
WiFi Settings	Enable WPS	WPA supports Wi-Fi Protected Setup (WPS). Supports connection to the network using the PIN and PBC.
	Authentication Option of Radio Parameters	Selects an authentication mode. The CE version and FCC version configurations can be authenticated.
	Enable MESH	Supports the mesh function.
Third Party library	cJson support	Supports CJSON library build.
	COAP support	Supports libcoap library build.
	MQTT support	Supports MQTT library build.
	iperf support	Supports iPerf library build.
Lwip Settings	Enable Option Router (Option3)	sets the gateway address option.
	Enable DHCP Hostname (Option12)	Sets or obtains the host name of the netif interface.
	Enable DHCP Vendorname (Option60)	Sets or obtains the vendor class identifier.
	Lwip Support Lowpower Mode	Sets lwIP support for the low-power mode. (The low-power mode needs to be configured.)
OTA Settings	compression ota support	Supports compression upgrade.



Menu	Configuration Item	Description
	dual-partition ota support	Supports dual-partition upgrade.
Link Settings	Hilink support	Supports HiLink connection.
Debug Log Settings	Enable debug log	Supports debug logs.

## 2.2.3 Precautions

- If the system displays a message indicating that you do not have the execute permission when you run the **./build.sh** command, run the **chmod +x build.sh** command to add the execute permission or run **sh ./build.sh**.
- During build, if an error message is displayed, indicating that a package cannot be found, check whether the required component has been installed in Python. If the build environment contains multiple Pythons, especially Pythons of the same version, and you cannot identify the version in use, you are advised to use the source code of the component package when installing the Python component package.
- The system preferentially uses the configured **menuconfig**. If **menuconfig** is not configured, the system uses the default configuration for build.
- For details about how to burn programs, see the *Hi3861 V100/Hi3861L V100 HiBurn User Guide*.

## 2.3 Building the SDK (Makefile)

The usage of **Makefile** at the top level of the Wi-Fi module is described as follows:

- Build the entire project.  
make / make all
- Build the factory test program.  
make factory
- Delete the build files of the entire project.  
make clean



# 3 Creating an App

---

[3.1 Creating a Directory Structure](#)

[3.2 Developing Code](#)

[3.3 app\\_main](#)

[3.4 Configuring Build Attributes](#)

## 3.1 Creating a Directory Structure

### NOTE

You can create an app in the same directory as **demo**. The following uses **my\_demo** as an example.

To create a directory structure, perform the following steps:

- Step 1** Copy **app/demo/SConscript** in the root directory of the project to **app/my\_demo/SConscript**.
- Step 2** Create a directory structure for **my\_demo** development (for example, **src**, **include**, and **init**) by referring to **app/demo**. Copy **app/demo/src/SConscript** to the new source code directory in **my\_demo**.

----End

## 3.2 Developing Code

After the directory structure is created, start the development code (for details, see **app/demo**). After the code is developed, run the **sh ./build.sh my\_demo** command to compile **my\_demo** for code debugging.

## 3.3 app\_main

The **app\_main** API is implemented in **app/demo/src/app\_main.c** (entrance of the entire app). In the **demo** sample, the **app\_main** API calls some program initialization APIs to initialize the chip, including the initialization of some



peripheral drivers (including the flash memory, UART, watchdog, I/O, DMA, I<sup>2</sup>C, I<sup>2</sup>S, and SPI) and some functional modules (including AT, DIAG, shell commands, Wi-Fi, and UPG). The demo implements some basic function examples. You can create an app task in **app\_main** by referring to **demo**.

## 3.4 Configuring Build Attributes

### 3.4.1 Basic Build Scripts

**Table 3-1** describes the basic build script files. (The paths are relative to the project path.)

**Table 3-1** Basic build scripts

Directory	Description
SConstruct	Entry for the SDK build script
build/scripts/link.ld.S build/scripts/system_config.ld.S	SDK link script
build/scripts/common_env.py	Public and system configurations for build
app/my_demo/SConscript	Build script of the my_demo module (top-level SConscript)
app/my_demo/src/SConscript	Source code build script of the my_demo module (second-level SConscript)
app/my_demo/app.json	Build configuration file of the my_demo module, which must comply with the JSON format

### 3.4.2 Build Configuration Files

- Step 1** Create the **app.json** file in **my\_demo** or copy the **app.json** file from **demo** to **my\_demo**.
- Step 2** Build and configure the **app.json** file by referring to **Table 3-2**, save the file, and exit. This file is a JSON file and must comply with the JSON format requirements.

**Table 3-2** Build items of app.json

Item	Description	Example
TARGET_LIB	.a library file named <b>TARGET_LIB</b>	"TARGET_LIB": "my_demo"



Item	Description	Example
APP_SRCS	Source code of SConscripts. The relative path of the directory must be input. For example, if the <b>my_demo</b> directory contains the <b>init</b> and <b>src</b> directories and the SConscripts file already exists in the directories, add <b>init</b> and <b>src</b> to <b>APP_SRCS</b> .	"APP_SRCS": ["init","src"]
INCLUDE	Header files. The relative path of the SDK root directory must be input. If no other header file directory is required, leave this parameter blank. For example, <b>my_demo</b> needs to reference the header files in <b>my_demo/include</b> . That is, add <b>app/my_demo/include</b> to <b>INCLUDE</b> .	"INCLUDE": [ "app/my_demo/include", "config/app" ]
CC_FLAGS	Compiler option configuration, which is similar to that of GCC. If this parameter is not required, leave it blank.	"CC_FLAGS": ["-Werror"]
DEFINES	GCC build macro. The configuration content is the <b>-D</b> option of GCC. Each element in the <b>defines</b> list is used as the <b>-D</b> option of GCC during build. If this parameter is not required, leave it blank.	"DEFINES": [ "_PRE_WLAN_FEATURE_CSI", "_PRE_WLAN_FEATURE_P2P", "LWIP_ENABLE_DIAG_CMD=0" ]
AR_FLAGS	Build option of the static library packaging command. If this parameter is not required, leave it blank.	"AR_FLAGS": ["-X32_64"]
LD_FLAGS	Linkage option. If this parameter is not required, leave it blank.	"LD_FLAGS": ["-A"]
AS_FLAGS	Build assembly file option. If this parameter is not required, leave it blank.	"AS_FLAGS": ["--warn"]
LD_DIRS	Linkage directory option. Place the required .a library files in this directory. The link is automatically added to the linkage option during linking. If this parameter is not required, leave it blank.	"LD_DIRS":["app/ my_demo/libs"]



Item	Description	Example
CLEAN	Files are automatically deleted when you run the <b>scons -c</b> command to clean the project. Generally, leave this parameter blank. This parameter can be set for files that require special processing, for example, temporary files or intermediate files. Set this parameter to the relative path of the SDK root directory.	"CLEAN": ["app/my_demo/test.txt"]

**Step 3** Go to the SDK root directory and run the **sh build.sh my\_demo** command to start build. You can also run the **scons app=my\_demo** command to start build. To change the default app project for build, modify the **build.sh** file and add the **scons app=my\_demo** parameter to the **scons** command.

----End



# 4 SDK Component Customization

---

## NOTE

This topic describes information about the SDK build system. Generally, you only need to focus on app development to complete project development. If the app development cannot meet specific requirements, you can add, modify, or delete components in the SDK to complete project development.

[4.1 Build System Overview](#)

[4.2 Component Creation](#)

[4.3 Menuconfig Customization](#)

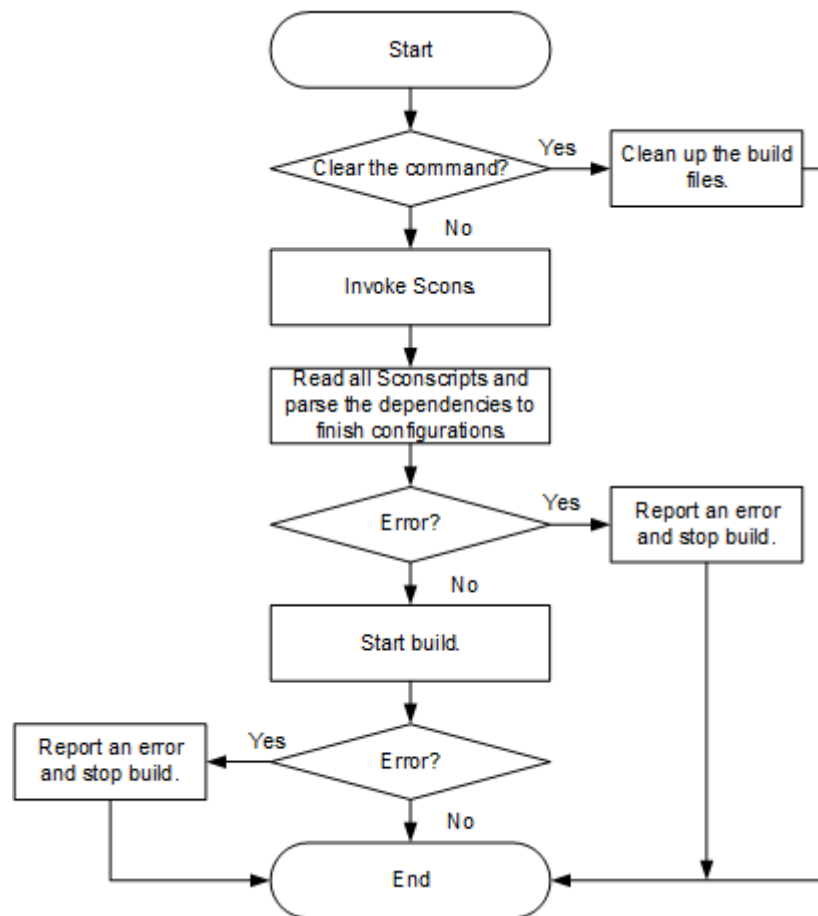
## 4.1 Build System Overview

In the SDKs of Hi3861 V100 and Hi3861L V100, SCons is used as the build tool. Therefore, you are advised to use SCons for development to ensure build integrity and continuity. SCons is developed based on Python and uses the Python syntax.

[Figure 4-1](#) shows the SCons build process.



**Figure 4-1** SCons build process



Common problem in debugging the SCons build system: Since SCons is configured with build information such as pre-reading scripts and dynamically generating build dependencies, the build starts after these operations are complete. However, in the process of pre-reading scripts, the python statement of non-SCons content is executed. For example, the print function of the Python statement is printed when the SCons pre-reads the script. During the build, no print statement can be executed.

For details about SCons, visit <http://www.scons.org/>.

**Table 4-1** describes the common files used in the SDK build system.

**Table 4-1** Common files used in the SDK build system

File Name and Path	Description
build.sh	Build startup script, in which the default app project can be edited



File Name and Path	Description
SConstruct	SCons startup entry, whose name must comply with industry standards. It is the main control implementation file of the SDK build process, including build and linkage. SCons follows the implementation of this file to generate the dependency of the library file on which the final file depends and the generation method.
SConscript	<p>Subordinate SCons script, whose name must comply with industry standards. The SDK is classified into two types. For example:</p> <ul style="list-style-type: none"><li>• <b>app/demo/SConscript</b> is used to configure the script for generating library files. SCons follows the implementation of this file to generate the dependency of the .o file on which the library file depends and the generation method (type-1 SConscript for short).</li><li>• <b>app/demo/src/SConscript</b> is used to configure the script for generating .o files. SCons follows the implementation of this file to generate the dependency between the .c and .s files on which the .o file depends and the generation method (type-2 SConscript for short).</li></ul> <p>Generally, type-1 SConscript and type-2 SConscript are in a tree structure physically, that is, type-1 SConscript belongs to a root node, and type-2 SConscript belongs to a leaf node.</p>
build/scripts/common_env.py	Configuration file of build options, including the header file directory on which a single module depends and build options of a single module

## 4.2 Component Creation

This section uses **my\_module** as an example to describe how to modify the build system and add **my\_module**. **my\_module** contains the **include** directory for storing header files and the **src** directory for storing source code files. The entire module is placed under **components**.

### 4.2.1 Build Output

The SDK contains build information. To view the detailed information, change the value of **log\_output\_flag** in **build/scripts/common\_env.py**.

- **True:** Detailed output is enabled.
- **False:** Detailed output is disabled.

#### NOTE

Detailed output may not be synchronized with the build process. For details, see [4.1 Build System Overview](#).



## 4.2.2 Build Configuration

To configure the build, do as follows:

**Step 1** Place the **my\_module** folder in the **components** directory, for example, **components/my\_module**.

**Step 2** Add the first-level SConscript in the **my\_module** directory.

You are advised to copy the script from the app directory in the SDK. For example, copy **app/demo/SConscript** to **components/my\_module**. The SConscript is a type-1 script. The source code directory will be queried based on the configuration and a library file (.a) will be generated.

**Step 3** Add the second-level SConscript in the **my\_module/src** directory.

You are advised to copy the SConscript in the second-level directory from the app directory in the SDK. For example, copy **app/demo/src/SConscript** to **my\_module/src/SConscript**. The SConscript is a type-2 script. If **my\_module** has multiple source code directories, you need to copy a type-2 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 build of the SCons script.) If the source code directory is deep or contains many directories, you need to build the SCons script independently. To reduce the 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 type-1 SConscript. You only need to modify the type-2 SConscript to access each source code directory in traversal mode, and build all .c or .s files.

The following example code is a type-2 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**, including the library file name, source code path, and module build configuration.

1. (Mandatory) Add the new **my\_module** to **compile\_module**.  
`compile_module = ['drv', 'sys', 'os', 'at', 'mqtt', 'mbedtls', 'sigma', 'my_module']`

2. (Mandatory) Add the path of the new **my\_module** to **module\_dir**. Type the relative path of the project root directory. The path contains the type-1 SConscript.

```
module_dir = {
    'drv': os.path.join('platform', 'drivers'),
    'my_module': os.path.join('components', 'my_module'), #Enter the path relative
    to the project root directory, which contains type-1 SConscript.
    ...
}
```

3. (Mandatory) Add the library file name and source code file directory of the new **my\_module** 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 type-1 SConscript. The source code directory contains the type-2 SConscript.

```
proj_lib_cfg = {
    'drv':{'adc' : ['adc'],
            'flash' : ['flash'],
            'spi' : ['spi'],
            'uart' : ['uart'],
            'pwm' : ['pwm'],
            'i2c' : ['i2c']},
    'my_module':{'my_module':['src']}, #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 type-1 SConscript. The source code directory contains the type-2
    SConscript.
    ...
}
```

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

```
'defines':{'
    'common':[ ('PRODUCT_CFG_SOFT_VER_STR', r"\"%s\""%product_soft_ver_str),
                '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'
            ],
    'iperf':[], # The list can be left empty.
    ...
}
```

5. (Optional) Add build options to **proj\_environment['opts']**. If no build options exist, skip this step.
6. (Optional) Add the header file of Huawei LiteOS referenced by the new **my\_module** to **proj\_environment['liteos\_inc\_path']**. If no header file exists, skip this step.



7. (Mandatory) Add the header file of a non-Huawei LiteOS system referenced by the new **my\_module** 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'),
    ],
    'my_module':[os.path.join('#', 'components', 'my_module', 'include')],
    ...
}
```

----End

**build/scripts/common\_env.py** contains most build configurations. [Table 4-2](#) lists the common configuration items.

**Table 4-2** Common configuration items in common\_env.py

Configura tion Item	Description
compile_ module	Python list format, which defines the sub-module to be built
log_outpu t_flag	Output enable for the detailed build content
os_lib_pat h	Additional link directory
module_di r	Python dictionary type, which defines the sub-module configuration. It includes the module name (which must be the same as that in <b>compile_module</b> ) and module path. The module path is the path of type-1 SConscript.
proj_lib_cf g	Python diction type, which defines the configurations of the subdirectories in the submodule. The subdirectories contain .c and .s files, that is, all directories containing type-2 SConscript.



Configuration Item	Description
proj_environment	<p>Common build options:</p> <ul style="list-style-type: none"><li>• <b>ar_flags</b>: packaging parameter</li><li>• <b>link_flags</b>: link parameter</li><li>• <b>defines</b>: macro definition used for build. <b>common</b> indicates a global macro definition, which can be used to configure the macro definition of a single module.</li><li>• <b>opts</b>: build options supported by the compiler. <b>common</b> indicates a global option, which can be used to configure a submodule.</li><li>• <b>liteos_inc_path</b>: path of OS header files. <b>common</b> indicates the directory of the header files added globally, which can be used to configure a submodule. Set this parameter to the relative path of <b>platform/os/Huawei_LiteOS/</b>.</li><li>• <b>common_inc_path</b>: path of header files not related to OS. <b>common</b> indicates the directory of the header files added globally, which can be used to configure a submodule. Set this parameter to the relative path of the SDK root directory, which starts with a number sign (#).</li></ul>

## 4.3 Menuconfig Customization

**menuconfig** is a customization tool of the Linux kernel. It is used to configure the function switch, define a configuration value, and customize new content in the SDK.

The following uses **my\_module** as an example to describe how to customize **menuconfig**:

**Step 1** Add the kconfig file to **components/my\_module/** and build the configuration content.

For details about the syntax of kconfig, see the Linux Kernel document <https://www.kernel.org/doc/html/latest/kbuild/kconfig.html>.

**Step 2** Open the **tools/menuconfig/Kconfig** file, create a line at the end of the file, add **source components/my\_module/Kconfig** to the line, and save and close the file.

**Step 3** Run **sh build.sh menuconfig** to check whether the new configuration is displayed on the GUI.

**Step 4** Select **Save**.

**Step 5** Open the **build/config/usr\_config.mk** file and find the copied configuration item **CONFIG\_ENABLE\_MY\_MODULE**.

**Step 6** Edit the **build/scripts/common\_env.py** file, find the comment **Configurations**, and add the workflow for reading configuration items to the code. That is, when **CONFIG\_ENABLE\_MY\_MODULE** is set to **y**, the system builds **my\_module**. Save the configuration and exit.



```
1 # Configurations
2 if scon_usr_bool_option('CONFIG_ENABLE_MY_MODULE') == 'y':
3     compile_module.append("my_module")
```

**----End**