

HT32 MCU RT-Thread Application Example

D/N: AN0697EN

Introduction

The RT-Thread (Real Time-Thread) is an open source real-time operating system for embedded systems, with excellent tailorability and extensibility. It is designed to provide an efficient and small size real-time kernel for embedded devices, which is suitable for embedded systems with different architectures from microcontroller to Arm® Cortex®-M/R, MIPS, X86, RISC-V and so on. The RT-Thread follows the Apache 2.0 open source license agreement, which means that the RT-Thread kernel and components can be used in commercial products free of charge and do not require the release of private code to the public. Users only need to state “Based on the RT-Thread system” or “Powered by the RT-Thread” in the product documents or instructions.

This document will introduce the RT-Thread environment preparation, RT-Thread system tailoring, Env tool dependency scripts, writing method of the on-chip peripheral drivers and the migration application of the HT32 MCU on RT-Thread.

Environment and Code Preparation

Before using the RT-Thread, users need to prepare the relevant development tool, build a project compilation environment and obtain the RT-Thread source code package. This document takes the Env development tool provided by the RT-Thread as an example.

Env Tool Download and Installation

There are two parts to building an RT-Thread development environment.

1. Project generation and auto-configuration, using the native Env tool provided by the RT-Thread.
2. Third-party IDEs for developing projects, including Keil and IAR.

The Env tool is an official software toolkit provided by the RT-Thread for developing and compiling its system source code, which can be downloaded directly from the RT-Thread official website.

Chinese version download link: <https://www.rt-thread.org/download.html>

English version download link: <https://www.rt-thread.io/download.html>

The installation of the Env tool is divided into online installation and offline installation. Taking the Env v2.0.0 version as an example, the env-windows-v2.0.0.7z file is the online installation

version, and the env-windows-v2.0.0-venv.7z file is the offline installation version, which can be downloaded according to the developer usage. Note that if users select 'Website Download' in the Chinese version download link, the compressed file obtained is the online version.

Following are the two installation methods:

1. Online installation

Download and decompress the env-windows-v2.0.0.7z file to any system directory, double-click the env.exe to enter the Env environment, then initialise the environment for the first time.

Note: When using the Env tool for the first time, a pip dependency should be installed online, wait for the dependency installation to complete. If the installation fails, manually delete the .venv directory in the env-windows directory that is the decompression directory of the software, and then open the env.exe again to install the dependency again until the installation is successful.

2. Offline installation

Download and decompress the env-windows-v2.0.0-venv.7z file to the root directory in C disk, the directory structure is C:\env-windows. Double-click the env.exe to enter the Env environment, then initialise the environment for the first time.

Note: It must be decompressed to the root directory in C disk, the directory structure is C:\env-windows\env.exe.

After the installation is complete, developers can add the Env tool to the right-click menu to facilitate the use of the Env tool. Enter the Env directory, run env.exe or env.bat in the directory, the system will open a command window, right-click the title bar or menu bar, and then follow the steps in Figure 1 to add the Env tool to the right-click menu.

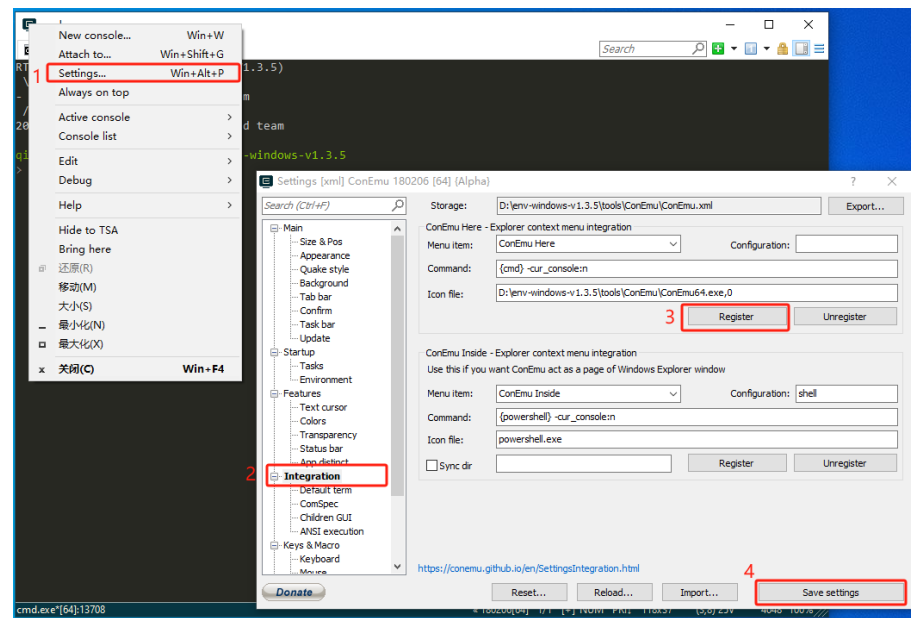


Figure 1. Add the Env Tool to the Right-Click Menu

After successfully adding the Env tool to the right-click menu, right clicking in any folder can find the “ConEmu Here” option, as shown in Figure 2, click it to open the Env tool.

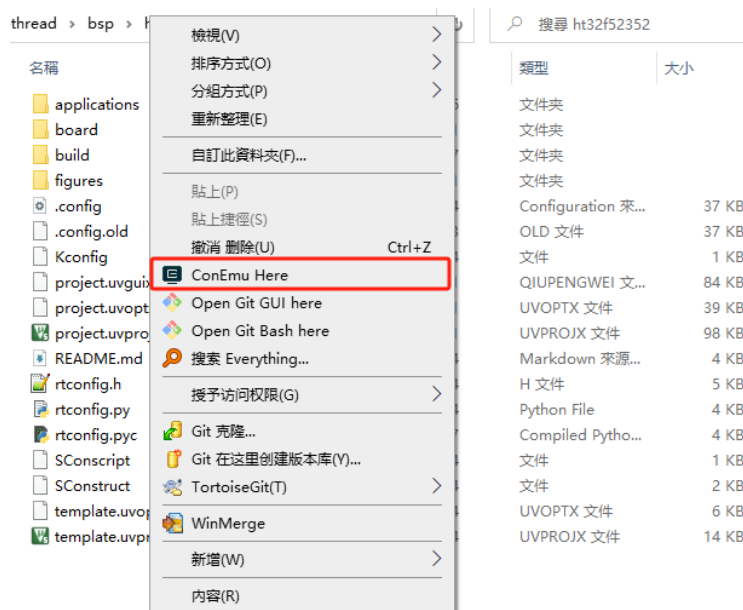


Figure 2. Right-Click to Use Env Tool

Regarding the installation and use of the third-party IDE, this document will not introduce more details, install it according to personal habits.

Obtain RT-Thread Source Code Package

A download link to the RT-Thread source code is also available on the same download page as the Env tool. Three download methods are provided, GitHub, Gitee and Baidu Network Disk. The download links are listed below:

Gitee download: <https://gitee.com/rththread/rt-thread>

GitHub download: <https://github.com/RT-Thread/rt-thread>

Baidu Network Disk download: https://pan.baidu.com/s/10_50j70OVGYrEWmm2X1qvQ
password: t22b

Taking the Gitee download as an example, go to the RT-Thread open source library, then to the master branch as shown in Figure 3. Check that everything is correct, then click “Clone/Download” to download. At this time, a download page will pop up, developers can use the git tool to clone it or download the ZIP-compressed file directly. As shown in Figure 4, click “Download ZIP”, the browser will automatically download the ZIP-compressed RT-Thread source code package. After the download is complete, the RT-Thread source code package can be obtained after decompression, as shown in Figure 5.



Figure 3. RT-Thread Source Code Download

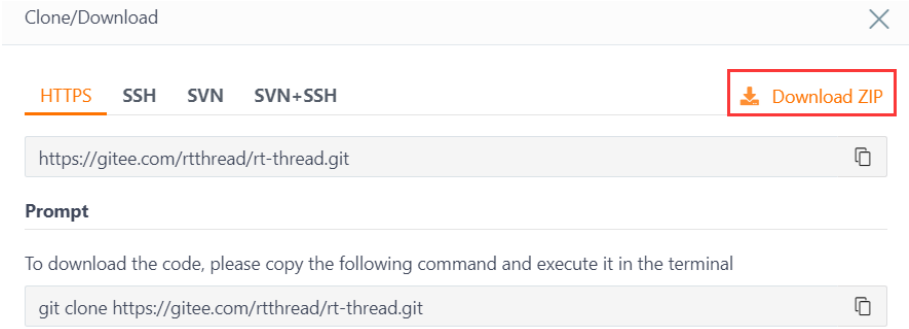


Figure 4. ZIP Download

名稱	修改日期	類型	大小
.devcontainer	2024/9/2 9:34	文件夾	
.gitee	2024/9/2 9:34	文件夾	
.github	2024/9/2 9:34	文件夾	
.hooks	2024/9/2 9:34	文件夾	
bsp	2024/9/2 9:40	文件夾	
components	2024/9/2 9:41	文件夾	
documentation	2024/9/2 9:41	文件夾	
examples	2024/9/2 9:41	文件夾	
include	2024/9/2 9:41	文件夾	
libcpu	2024/9/2 9:41	文件夾	
src	2024/9/2 9:41	文件夾	
tools	2024/9/2 9:41	文件夾	
.clang-format	2024/8/13 16:19	CLANG-FORMAT...	7 KB
.gitattributes	2024/8/13 16:19	文本文檔	1 KB
.gitignore	2024/8/13 16:19	文本文檔	1 KB
ChangeLog.md	2024/8/13 16:19	Markdown 來源...	136 KB
Kconfig	2024/8/13 16:19	文件	1 KB
LICENSE	2024/8/13 16:19	文件	12 KB
README.md	2024/8/13 16:19	Markdown 來源...	12 KB
README_de.md	2024/8/13 16:19	Markdown 來源...	12 KB
README_es.md	2024/8/13 16:19	Markdown 來源...	12 KB
README_zh.md	2024/8/13 16:19	Markdown 來源...	11 KB

Figure 5. RT-Thread Source Code Package

RT-Thread Instructions

After the Env tool has been installed and the RT-Thread source code package has been prepared, the RT-Thread can be used. The ht32f52352 is used as an example to introduce how to use the Env tool to tailor and configure the RT-Thread, then generate and verify the project. The ht32f52352 project is located in the path “rt-master\bsp\ht32\ht32f52352”.

System Tailoring and Configuration

In the “rt-thread-master\bsp\ht32\ht32f52352” directory, right-click to select “ConEmu Here” to open the Env tool. Then input “menuconfig.exe” and press Enter, the Env tool will enter the RT-Thread configuration page, as shown in Figure 6.

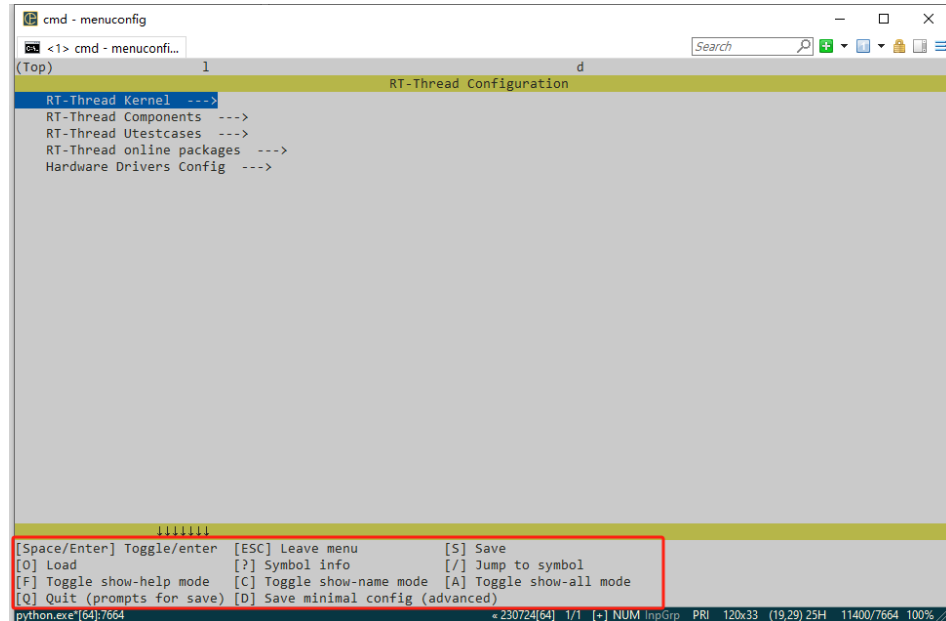


Figure 6. RT-Thread Tailoring and Configuration Page

Users can use the arrow keys and Enter key to enter different configuration menus according to their own requirements, and use the space bar to select or cancel the corresponding options, so as to implement system tailoring and configuration. If there are no special requirements, maintain the default configuration. After the configuration is complete, remember to save the configuration information before exiting, otherwise the configuration information will not take effect.

Create a Project

After tailoring and configuring the RT-Thread, users need to regenerate a new project according to the configured files, otherwise the tailoring and configuration information cannot be reflected in the project. The Env tool provides commands for generating different IDE projects such as MDK, IAR, and GCC, as shown in Figure 7.

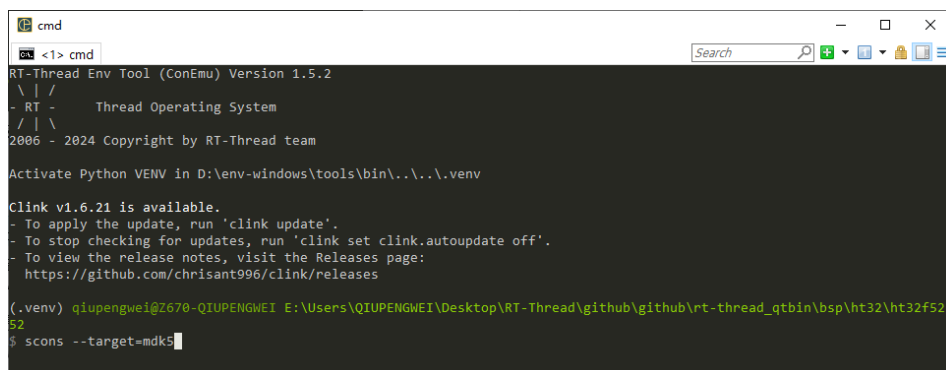
```

scons --target=mdk5
scons --target=mdk4
scons --target=iar
scons --target=gcc
  
```

Figure 7. Commands for Generating Project

For example, generate an MDK 5 project of the ht32f52352. Input “scons --target=mdk5” in the Env tool, as shown in Figure 8, then press Enter. The Env tool will regenerate a project named project based on the previous tailoring and configuration on the menuconfig page, using the template project as a template. A prerequisite for creating a new project is that there must be a template project named template in the project directory. For example, the directory of the

ht32f52352 project is rt-thread-master\bsp\ht32\ht32f52352. For the template project, refer to the “Template Project Introduction” section. After the creation process is complete, the MDK 5 project file will appear in the current directory.



```
cmd
RT-Thread Env Tool (ConEmu) Version 1.5.2
- RT - Thread Operating System
- / \ 5.2.0 build Apr 28 2024 09:36:10
2006 - 2024 Copyright by RT-Thread team

Activate Python VENV in D:\env-windows\tools\bin\...\venv

Clink v1.6.21 is available.
- To apply the update, run 'clink update'.
- To stop checking for updates, run 'clink set clink.autoupdate off'.
- To view the release notes, visit the Releases page:
  https://github.com/chrisant996/clink/releases

(.venv) qiupengwei@2670-QIUPENGWEI E:\Users\QIUPENGWEI\Desktop\RT-Thread\github\github\rt-thread_qtbin\bsp\ht32\ht32f52352
> scons --target=mdk5
```

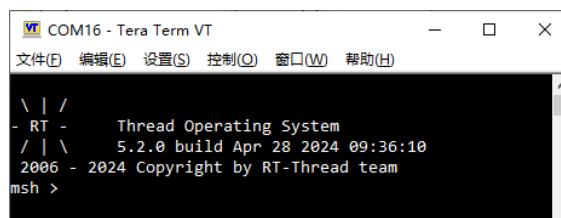
Figure 8. Command for generating an MDK 5 Project

Compile, Download and Verify

Projects created using the Env tool are named project. Double click the project file, it will be opened in Keil. As the code requires the V5 compiler and C99 standard support, users need to check the compiler version and C99 standard support before compiling, refer to “Keil_5 compiler version configuration” and “C99 mode selection” in the “Create and Configure Template Project” section.

After the check is complete, click “Compile” to compile the project. Taking the ht32f52352 as an example, use its development board, the ESK32-30501, as a verification board. After the board is connected to the computer, click “Download” in Keil to write the compiled project to the verification board. Once the project has been downloaded successfully, the system will run automatically, at this time the LED1 and LED2 on the verification board will flash alternately.

Open the serial port tool to enable the serial port emulated through the e-Link32 Lite. Click “Setup” → “Serial port” to configure the serial port. The baud rate is 115200, with 8 data bits, 1 stop bit and no parity bit. After the serial port has been configured, reset the verification board, then the RT-Thread kernel information can be saw in the serial port tool, as shown in Figure 9. At this point, the basic functions of the RT-Thread have been verified. Note that the kernel information here is not fixed, the kernel information may change depending on the kernel version and compilation time, etc.



```
COM16 - Tera Term VT
文件(F) 编辑(E) 设置(S) 控制(Q) 窗口(W) 帮助(H)

- RT - Thread Operating System
- / \ 5.2.0 build Apr 28 2024 09:36:10
2006 - 2024 Copyright by RT-Thread team
msh >
```

Figure 9. RT-Thread Kernel Information

Env Tool Introduction

Env tool main functions:

1. Build a project: The Env tool automatically adds the source code to the project.
2. Resolve dependencies: The Env tool automatically adds the paths of header files to the project.
3. System configuration: The Env tool automatically generates the corresponding macro definitions into the project based on user options.
4. Generate an independent project: The Env tool copies the used source code file to a specified folder, generating an independent project.

The implementation of the above functions mainly relies on script files and configuration files. The “Build a project”, “Resolve dependencies” and “Generate an independent project” functions are mainly implemented by the SCons script, and the “system configuration” function is mainly implemented by the Kconfig script. Figure 10 shows the script dependency framework of the Env tool.

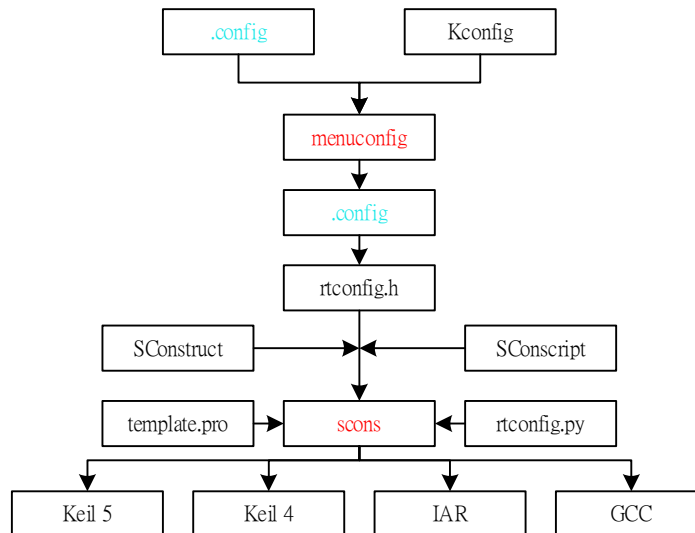


Figure 10. Script Dependency Framework of Env Tool

As shown in the figure above, the Env tool uses the menuconfig command to obtain the project configuration information and configuration menu from the .config and Kconfig files. After configuration, the .config and rtconfig.h files are generated. The Env tool uses the scons relevant commands to call the SConstruct and SConscript files in the SCons script, then creates projects for Keil_v5, Keil_v4, IAR and GCC compiler development according to the project template, template.pro, and the compiler relevant configuration information, rtconfig.py.

The main functions of the above files in the Env tool are as follows:

1. The “.config” file is used to save the RT-Thread configuration information.
2. The “rtconfig.h” file is used to save the macros generated by the configuration, this file will be added to the project.
3. The “Kconfig” file records the menu at the time of configuration. The menu used for configuration is read from this file.

4. The “SConstruct” file is the entry file for the scons script. This file guides the scons script to obtain the SConscript file from which paths.
5. The “SConscript” file is the management file of the header file and source file. This file guides the scons script which files should be added to the project. Usually, this file will be in the root directory of various source files.
6. The “rtconfig.py” file records information about the compiler and the chip kernel used in the Board Support Package (BSP).

HT32 Directory Structure Introduction

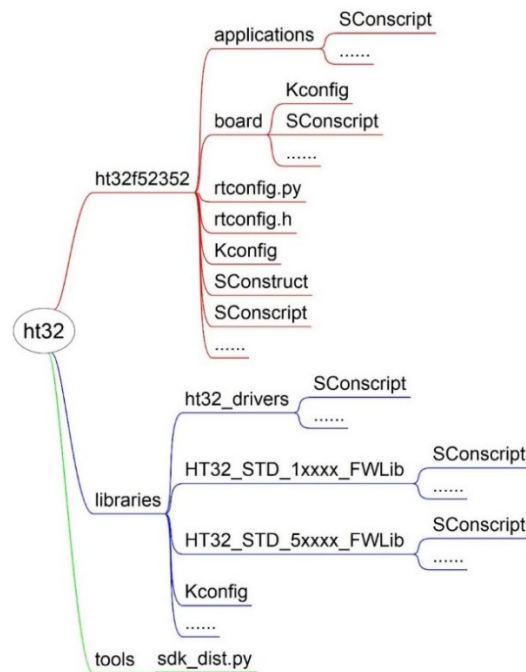


Figure 11. Script File Location

Taking the ht32 as an example, Figure 11 shows the locations of various script files in the ht32 directory, the ht32 folder is stored in the path “rt-thread-master\bsp”. The ht32 in the figure is a directory dedicatedly used to store the ht32 chip-related files. There are three important parts under this directory.

- The ht32fxxxxx is a unique file storage area for the specified chip, in this example, it is the ht32f52352.
- The “libraries” directory contains the ht32 generic libraries and RT-Thread drivers.
- The “tools” directory contains the Python scripts which are used to generate a project.

rtconfig.py File Introduction

The `rtconfig.py` script file is mainly used to select the processor kernel and configure compiler-related settings. For more details, refer to the `rtconfig.py` file in the path “rt-thread-master\bsp\ht32\ht32f52352”, the file can be opened via a text editor. This file records the chip kernel used by the board support package, the compilation operations for different kernels and the compiler path installed by the individual. The following two points should be noted if this file needs to be modified.

1. The file usually contains a CPU variable that specifies the processor kernel of the chip being used. For example, the HT32F52352 uses a Cortex[®]-M0 kernel, the configuration in the file is `CPU='cortex-m0'`. During the migration process, the kernel configuration, i.e. the value of the CPU, must be changed to the kernel corresponding to the target chip.
2. The `EXEC_PATH` parameter in the file refers to the compiler path installed by the individual, if a compiler installed by the developer is used, this parameter needs to be configured according to the actual software installation path.

SCons Script Introduction

The SCons script is mainly used to build a project, resolve dependencies and generate an independent project. Figure 12 shows the specific framework and the running flow of the SCons script.

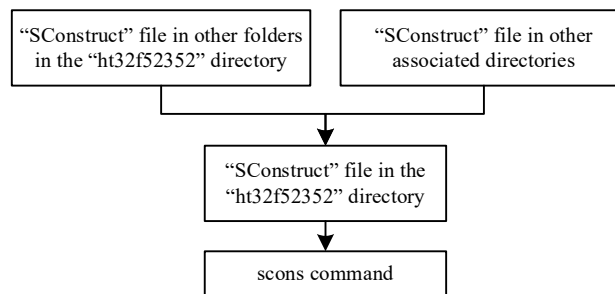


Figure 12. SCons Script Framework

The SCons script runs as follows:

- Use the scons relevant command in the BSP directory, in this example, the BSP directory refers to the ht32f52352.
- The SCons script calls the SConstruct file in the directory to complete the basic environment parameter configuration.
- The SConstruct file downloads and executes the associated SConscript file.
- The SConscript file executes specific building tasks.

SConstruct File

For an individual board support package, there is only one SConstruct file, which is the start of file relationship processing and the entry for the scons command. The following three points should be noted if this file needs to be modified.

1. Regarding the `libraries_path_prefix` parameter setting, the parameter records the path of the generic library. If the path is changed, the parameter setting should be modified.

2. Regarding the `ht32_library` parameter setting, the parameter records the path of the peripheral library for a specific chip. If the path is changed, the parameter setting should be modified.
3. Regarding the path to save drivers for RT-Thread, which is located at the “`#include drivers`” comment in the file, the folder name used in the example is “`ht32_drivers`”. If the path is changed, the corresponding parameter settings should be modified.

For more details, refer to the SConstruct file in the path “`rt-thread-master\bsp\ht32\ht32f52352`”, the file can be opened via a text editor.

SConscript File

The SConstruct file configures the environment parameters and associated file directories, while the SConscript file is the actual script file that records the file dependencies. The SConscript file has a detailed description of how the relevant source code files are imported into the project and their dependencies, and records the header file paths that need to be included. The entire source code file management is implemented by several similar SConscript files distributed in different directories. Each SConscript file only manages the files in the current directory and can be searched and executed by file queries and path associations.

The following describes some common syntax and rules of the SConscript file, as this file uses the Python language, it needs to be written in accordance with the relevant syntax of Python language. The SConscript file is mainly divided into the following parts.

1. Import modules and variables from other modules, mainly to use functions or variables from other modules.
2. Obtain the path to the current script. It is especially important to obtain the file paths since the SCons script mainly obtains various files in the form of file path.
3. Obtain the paths to the required C files and form a list, this operation aims to build the dependencies of the C files.
4. Obtain the paths to the required header files and form a list, this operation aims to build the dependencies of the header files.
5. Set the global macro definition.
6. Create a Group, with the C file list, header file list, and global macro definition as its objects.
7. Return to the created group for use by the `scons` command.

Figure 13 shows the commonly used modules and variables from other modules in the SConscript file.

```

import('RTT_ROOT')
import('SDK_LIB')
import('rtconfig')
import os
import rtconfig
from building import *

```

Figure 13. Included Modules and Imported Variables

- `RTT_ROOT` is the path of the RT-Thread source code package.
- `SDK_LIB` is the path of the chip generic library.

- rtconfig is the rtconfig.h file.
- os is a module in Python that provides a method for processing various files and directories.
- The rtconfig module is the rtconfig.py file, call this module to obtain the functions in the rtconfig.py file.
- The building module is an extension function of the SCons script, which is provided by the RT-Thread source code package.

In the building module, there is a function to obtain the current script path, as shown in Figure 14. The corresponding path information will be stored in the cwd variable.

```
cwd = GetCurrentDir()
```

Figure 14. Obtain Current Script Path

There are three methods to create a C file list, as shown below.

```
src = Glob('src/*.c')

src = Split("""
HT32F5xxxx_Driver/src/ht32f5xxxx_rstcu.c
HT32F5xxxx_Driver/src/ht32f5xxxx_ckcu.c
""")

if GetDepend(['RT_USING_PIN']):
    src += ['drv_gpio.c']
```

Figure 15. Create a C File List

- The Glob function is used to obtain the files that match its arguments. “src = Glob('src/*.c')” means to obtain all C file paths in the src folder in the current directory, then form a list and save it to the src variable.
- The Split function is used to split the listed paths into a list and save it to the src variable.
- The GetDepend function is used to implement file import with conditional dependencies. As shown in the figure above, if the RT_USING_PIN macro is defined, the path of the drv_gpio.c file will be added to the src list, then whether the drv_gpio.c file is added to the project will be determined by the RT_USING_PIN macro.

This method of creating a C file list is not limited to adding C files. The SCons script adds files based on the file paths in the list and does not limit the file type, therefore, the SCons script can add any file to the project as long as the corresponding file path has been added to the list. For example, if the startup file of the chip needs to be added into the project, the adding process is shown in Figure 16.

```
startup_path_prefix = SDK_LIB
if rtconfig.CROSS_TOOL == 'keil':
    src += [startup_path_prefix +
            '/ht32f52352_library/Device/Holtek/HT32F5xxxx/Source/ARM/startup_ht32f5xxxx_01.s']
```

Figure 16. Add Startup File

It should be noted that the file path must be clearly recorded to ensure that the corresponding file can be accurately found through the file path.

After the C files are imported into the project, to meet the compiling environment requirements, the corresponding header file paths that the C files depend on should also be imported into the project. There are two methods to create the header file path, as shown in Figure 17.

```
path = [
    cwd + '/HT32F5xxxxx_Driver/inc',
    cwd + '/CMSIS/Include',
    cwd + '/Device/Holtek/HT32F5xxxxx/Include'
]

path = [cwd]
path += [cwd + '/config']
```

Figure 17. Create a Header File Path List

The first is to create a list and list the header file paths. The second is to create a list and then add the header file paths to the list one by one. The paths to the header files listed in the list will be saved in the path variable and then imported into the project. Taking Keil as an example, the header file paths will be finally added to the “Options for Target...” → C/C++ → Include Paths”.

The following method can be used to define the global macros in the SConscript file, as shown in Figure 18. List the macros that need to be defined into the CPPDEFINES variable, which will be imported into the project during the project building process.

```
CPPDEFINES = ['USE_HT32_DRIVER', 'USE_HT32F52352_SK', 'USE_MEM_HT32F52352']
```

Figure 18. Global Macro Definition

Once the files to be imported into the project, their dependencies and the macros to be defined has been prepared, a group can be created to integrate the information that needs to be imported to the project, so that it can be easily called and imported into the project during the building process. Figure 19 shows the method for creating a group.

```
group = DefineGroup('Drivers', src, depend = [], CPPPATH = path, CPPDEFINES =
CPPDEFINES)

Return('group')
```

Figure 19. Create a Group

- The DefineGroup function is used to create a group, which is provided by the building module.
- ‘Drivers’ is the group name. In the Keil, it is the name of the Groups.
- src is the file to be imported to the group.
- depend is a dependent macro for importing this group. If this macro is not created, the group will not be imported into the project. As shown in the figure above, the import of the group does not depend on any macros.
- CPPPATH is the header file path dependency.
- CPPDEFINES is the global macro that needs to be defined.
- The created group information is stored in the group variable, the Return function returns the group information to the upper file, which is convenient for calling during the creation of the project.

The above describes the overall structure of the SConscript file and the methods used to create a SConscript file. For more details, refer to the SConscript file in the path “rt-thread-master\bsp\ht32\ht32f52352\board”, the file can be opened via a text editor. Note that the contents of the SConscript file in different directories may be different, as they need to adapt to different building requirements and configurations.

Kconfig Script Introduction

The Env tool provides a system configuration function, which is implemented by using the menuconfig command to call the Kconfig script in the source code. To implement this function, the RT-Thread source code package provides a complete Kconfig script framework and offers different compilation options for different macros in the corresponding source files. If the source code of the driver and application written by the user needs to be added to the RT-Thread Kconfig script framework, the Kconfig file corresponding to the source code should be written.

The Kconfig script is mainly used to generate corresponding macro definitions according to various configuration options, then according to different macros, tailoring and file dependency configuration can be implemented for source code files. Before writing a Kconfig script, users need to learn some basic knowledge about the script, refer to the official RT-Thread document.

File address: <https://www.rt-thread.org/document/site/#/development-tools/build-config-system/Kconfig>

Kconfig Files

In the ht32f52352 example, there are three Kconfig files and their paths are as follows:

1. rt-thread-master\bsp\ht32\ht32f52352
2. rt-thread-master\bsp\ht32\libraries
3. rt-thread-master\bsp\ht32\ht32f52352\board

The first is the Kconfig file in the path “rt-thread-master\bsp\ht32\ht32f52352”, there are three macro definitions in the file, the BSP_DIR, RTT_DIR and PKGS_DIR. These macros record the relative paths to the root directory of the BSP project, the root directory of the RT-Thread source code and the board support package, so the paths here must be configured according to the actual situation. Refer to the Kconfig file in the path “rt-thread-master\bsp\ht32\ht32f52352”, the file can be opened via a text editor. This Kconfig file is the entry file for the menuconfig command, through which the Kconfig files in different directories are called to be parsed and executed.

The second is the Kconfig file in the path “rt-thread-master\bsp\ht32\libraries”, which is related to the main controller chip configuration, as shown in Figure 20. This file configures the chip series and the chip kernel definition.

```
config SOC_FAMILY_HT32
    bool

config SOC_SERIES_HT32F5
    bool
    select ARCH_ARM_CORTEX_M0
    select SOC_FAMILY_HT32

config SOC_SERIES_HT32F1
    bool
    select ARCH_ARM_CORTEX_M3
    select SOC_FAMILY_HT32
```

Figure 20. Chip Series and Chip Kernel Definition

The final is the Kconfig file in the path “rt-thread-master\bsp\ht32\ht32f52352\board”, which is related to the peripheral drivers of the development board. Different development boards may have different main controllers and peripherals, so the contents of the Kconfig file will be different. The

actual content should be written according to the specific peripherals and adaptations of the development board. For more details, refer to the Kconfig file in the path “rt-thread-master\bsp\ht32\ht32f52352\board”, the file can be opened via a text editor.

Template Project Introduction

The template project file configured here is mainly used as a project template when building a project using commands, with the various configuration information in the script being imported into the project according to this project template. Since the template project is generated by the IDE that the user usually uses, the new project generated from the template project will be more suitable for the IDE version that the user uses. In addition, the project generated from the template can be opened directly by the IDE tool for development and debugging, eliminating the need to configure the project again.

Create and Configure Template Project

The following is an example of how to create a template project in the Keil_v5 environment. First, the template project name must be template, because the Env tool will identify the template project according to the name. The project generated from the template is called project. Taking the ht32f52352 as an example, create a new project named template, as shown in Figure 21.

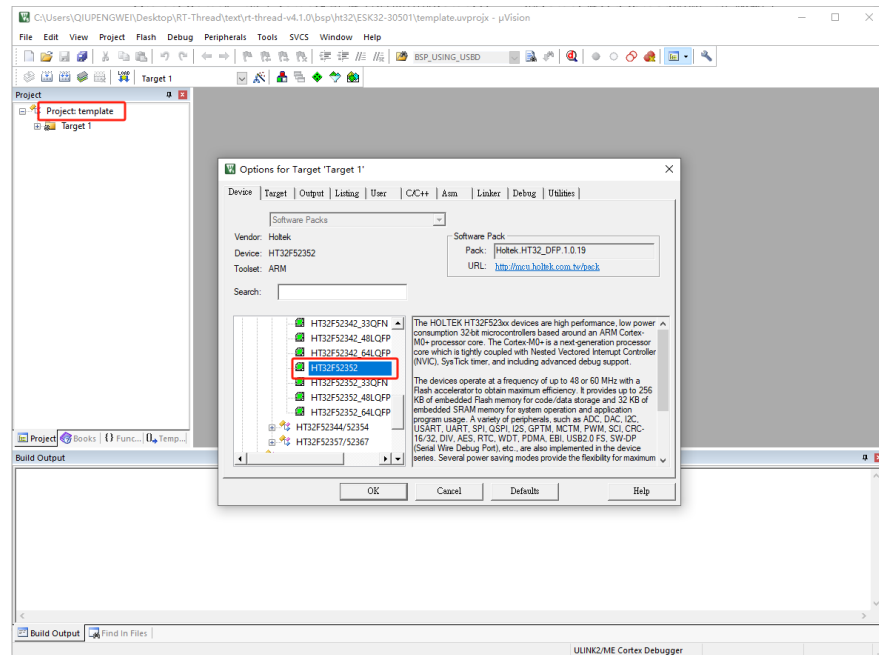


Figure 21. Create a Template Project

Start configuring the template project.

First, rename “Target”, the name can be modified to the name expected by the developer or not, the name used in the example is rt-thread.

Then, configure the Keil_5 compiler version, currently the V5 version is supported. As shown in Figure 22, click “Options for Target...” → “Target” → “Code Generation” → “ARM Compiler” to select the V5 compiler. Note that the compiler name is not fixed, users need to choose according to

the actual situation. The reference name is “Use default compiler version 5” or “V5.06 update 6 (build 750)”. Here choose “V5.06 update 4(build 422)”.

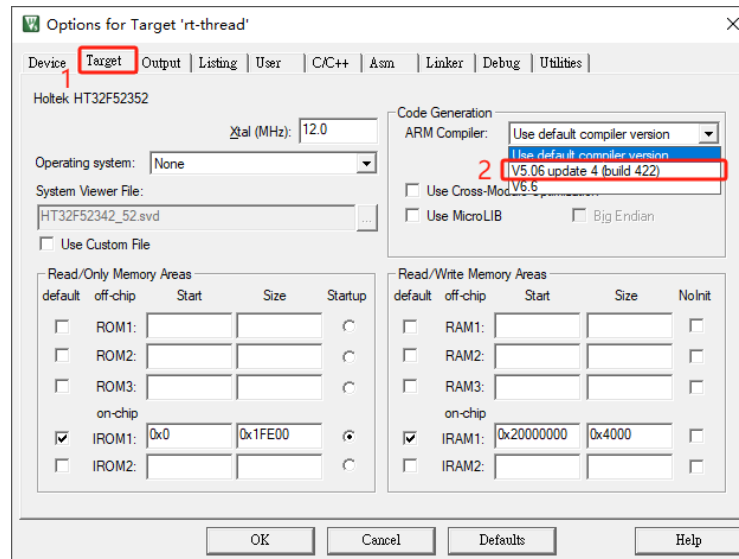


Figure 22. Keil_5 Compiler Version Selection

C99 mode selection: Follow the steps in Figure 23 for configuration. Click “Options for Target...” → “C/C++” → “Language/Code Generation” to check “C99 Mode”.

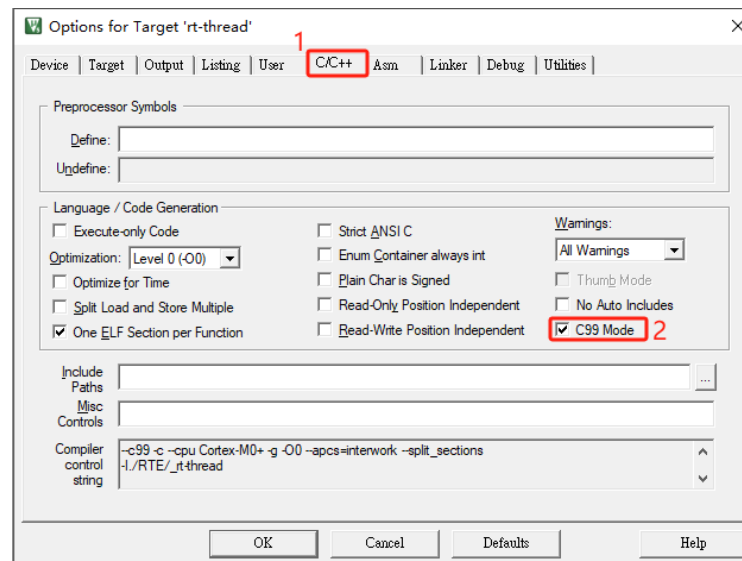


Figure 23. C99 Mode Selection

Assembly macro definition configuration: Follow the steps in Figure 24 for configuration. Click “Options for Target...” → “Asm” → “Define”. Input the assembly macro to be defined in the “Define” edit box, if there is no need to define assembly macro, this step can be omitted. The assembly macro definition may be different for different chips, users must configure it according to the specific assembly macro definition. For the specific assembly macro definition, refer to “Global macro definition modification” in the “Replace startup File” section of the “HT32 MCU Project Mutual Migration Method” application note.

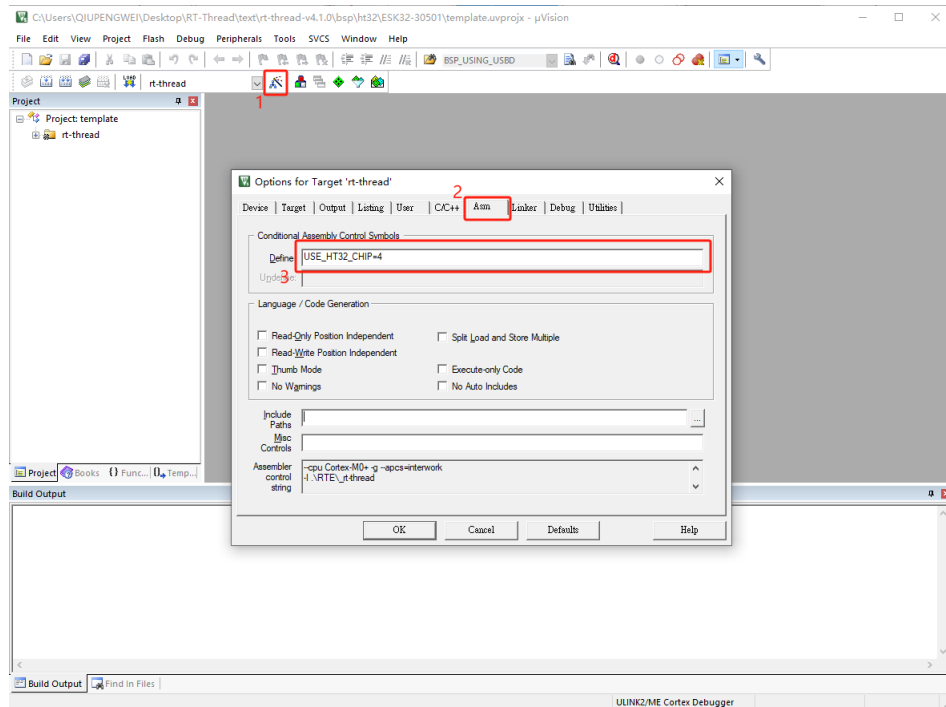


Figure 24. Add an Assembly Macro Definition

Link file configuration: First, create the link file for the Keil environment, create a link file with a suffix of .sct in the path “board\linker_scripts”, such as, the link.sct file in the ht32f52352 example, and modify the content as shown in Figure 25.

The file can also be generated by the project compiler by default, the default path is the Objects folder in the project directory. After compiling the project, double-click on the Objects folder to find a *.sct file. If the default *.sct file is used in the Demo project, the file will be generated in the path “build\keil\Obj” in the project directory.

Note: If the default *.sct file is used, make sure that the “Options for Target...” → “Linker” → “Use Memory Layout from Target Dialog” option is checked.

```

; *****
; *** Scatter-Loading Description File generated by uVision ***
; *****

LR_IROM1 0x00000000 0x0001FE00 {    ; load region size_region
ER_IROM1 0x00000000 0x0001FE00 {    ; load address = execution address
    *(RESET, +First)
    *(InRoot$$Sections)
    .ANY (+RO)
}
RW_IRAM1 0x20000000 0x00004000 {    ; RW data
    .ANY (+RW +ZI)
}
    
```

Figure 25. link.sct File Content

After the *.sct file has been created, follow the steps shown in Figure 26 for configuration. Click “Options for Target...” → “Linker” → uncheck “Use Memory Layout from Target Dialog” → click “...” and add the *.sct file created. Note that this step can be ignored if using the default .sct file generated when the project is compiled.

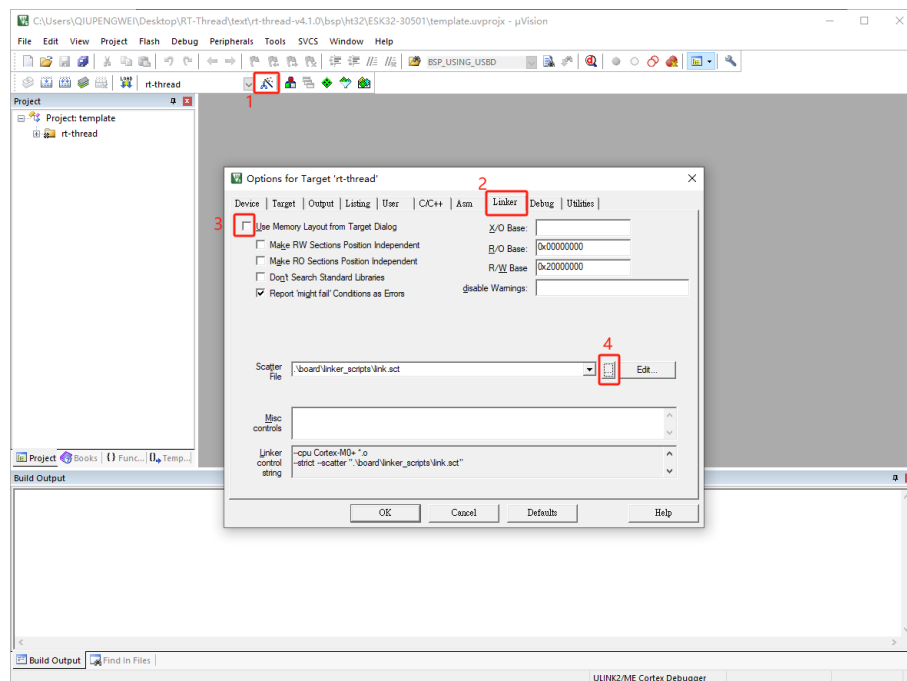


Figure 26. Configure Linker Information

Debugging interface and algorithm file configuration: Follow the steps in Figure 27 for configuration. Click “Options for Target...” → “Debug” → select “CMSIS-DAP Debugger” in the drop-down box → click “Settings” → “Flash Download” → check whether the algorithm files are configured in “Programming Algorithm” → if not, click “Add” → select the corresponding options → click “Add”.

At this time, the template project configuration has been completed. The Env tool will build a new project based on the template project.

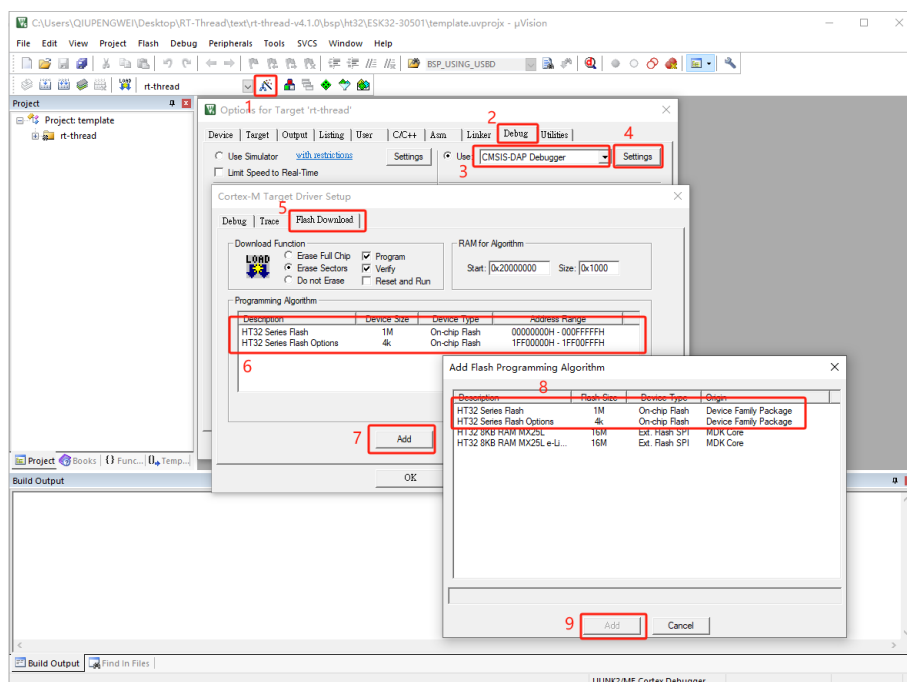


Figure 27. Configure Debugging Interface and Algorithm File

Env Tool Usage

The previous sections have provided a detailed introduction to the architecture of the Env tool and the contents of various files. The Env tool is mainly used to configure the system and build the project, which have already been introduced in the previous sections, refer to the “System Tailoring and Configuration” section and the “Build the Project” section in the “RT-Thread Instructions” chapter. For more details, refer to the official RT-Thread document.

Chinese version file address: <https://www.rt-thread.org/document/site/#/development-tools/env/env>
English version file address: <https://www.rt-thread.io/document/site/programming-manual/env/env>

Adapt to Other Chips Based on the Demo

If a required chip is not available in the BSP, users can migrate based on an adapted chip. Note that when using a Cortex®-M0 kernel chip, users must use the ht32f52352 as a template for migration; when using a Cortex®-M3 kernel chip, users must use the ht32f12366 as a template for migration. Taking the ht32f50343 chip as an example, the steps required for migration to a new chip project are described below.

Before starting, make sure that the Env tool has been installed and the latest RT-Thread source code package has been downloaded, refer to the “Environment and Code Preparation” section. In addition, read the relevant documents for the target IC, the HT32F50343, and its development board, the ESK32-30515, to understand the hardware resources of the chip and the use of the development board pins.

Preparation

First, prepare a new project for migration based on the Demo project and delete some irrelevant files. Since the ht32f50343 is a Cortex®-M0 kernel chip, use the ht32f52352 as the Demo project. Copy the Demo project and rename it to ht32f50343, as shown in Figure 28. Note that the copied project must be placed in the path “rt-thread-master\bsp\ht32\”, otherwise an error may occur.

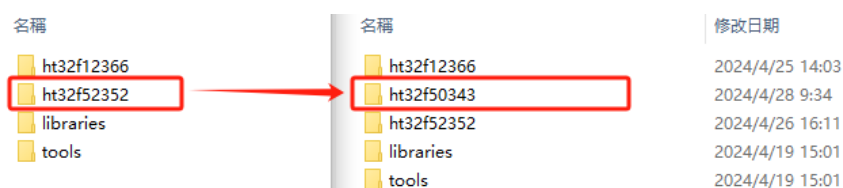


Figure 28. Copy and Rename the Project

After copying the Demo project, open the copied project and delete the files in the red boxes as shown in Figure 29. Only the files related to the Keil project will be deleted, not the files in other folders. A new project is ready for migration, then modify some files within the project and add the corresponding template project.

rt-thread-master > bsp > ht32 > ht32f50343 > 搜尋 ht32f50343















名稱	修改日期	類型	大小
 applications	2024/8/20 15:37	文件夾	
 board	2024/8/20 15:37	文件夾	
 figures	2024/8/20 15:37	文件夾	
 .config	2024/8/13 17:08	Configuration 來...	41 KB
 Kconfig	2024/7/10 10:08	文件	1 KB
 project.uvoptx	2024/8/15 16:10	UVOPTX 文件	44 KB
 project.uvprojx	2024/8/15 11:03	UVPROJX 文件	109 KB
 README.md	2024/7/10 10:08	Markdown 來源...	4 KB
 rtconfig.h	2024/8/13 17:08	H 文件	8 KB
 rtconfig.py	2024/7/10 10:08	Python File	5 KB
 SConscript	2024/7/10 10:08	文件	1 KB
 SConstruct	2024/8/13 17:08	文件	2 KB
 template.uvoptx	2024/8/14 17:36	UVOPTX 文件	6 KB
 template.uvprojx	2024/8/14 17:36	UVPROJX 文件	14 KB

Figure 29. Delete the Files in the Red Boxes

Project Modification

In order to adapt the project to the chip in use, two files in the project need to be modified.

The first is the SConscript file in the path “rt-thread-master\bsp\ht32\ht32f50343\board”. The startup file and the global macro definition used by the chip need to be modified in this file. Taking the example of building a Keil project, only the startup file of the Keil project needs to be modified. The startup file for the ht32f50343 chip is the startup_ht32f5xxx_06.s, the modified result is shown in Figure 30. For the corresponding startup file for the specific chip, refer to the section “startup Files and system Files” section in the “HT32 MCU Project Mutual Migration Method” application note.

```

startup_path_prefix = SDK_LIB
if rtconfig.CROSS_TOOL == 'gcc':
    src += [startup_path_prefix +
            '/HT32_STD_5xxx_FWLb/library/Device/Holtek/HT32F5xxx/Source/GCC/startup_ht32f5xxx_gcc_01.s']
elif rtconfig.CROSS_TOOL == 'keil':
    src += [startup_path_prefix +
            '/HT32_STD_5xxx_FWLb/library/Device/Holtek/HT32F5xxx/Source/ARM/startup_ht32f5xxx_06.s']
elif rtconfig.CROSS_TOOL == 'iar':
    src += [startup_path_prefix +
            '/HT32_STD_5xxx_FWLb/library/Device/Holtek/HT32F5xxx/Source/IAR/startup_ht32f5xxx_iar_01.s']

```

Figure 30. Modify the startup File

As the global macro definition used by each chip is different, modify it according to the macro definition of the specified chip, as shown in Figure 31. For the global macro definition used by the chip, refer to the “Global macro definition modification” in the “Replace system File” section of the “HT32 MCU Project Mutual Migration Method” application note.

```

CPPDEFINES = ['USE_HT32F52352_SK, USE_HT32F52342_52, USE_MEM_HT32F52352']
CPPDEFINES = ['USE_HT32F50343_SK, USE_HT32F50343, USE_MEM_HT32F50343']

```

Figure 31. Modify the Macro Definition

The second is the ht32_msp.h file in the path “rt-thread\bsp\ht32\ht32f50343\board\inc”. To adapt to the RT-Thread FinSH console, it is necessary to configure the corresponding serial port pins. Taking the ESK32-30515 development board as an example, use the UART1 connected to the e-Link32 Lite as the serial port to adapt to the FinSH console. The UART1 interface uses the PA4 and PA5 pins. Open the ht32_msp.h file, find the UART1, and then modify the corresponding pins

to PA4 and PA5, as shown in Figure 32. The pins of other interfaces can also be modified according to the actual situation.

Developers can check which UART is used according to the interface pins used by the FinSH console adaption by checking the “Pin Assignment” table in the chip datasheet, and then modify the ht32_msp.h file. If developers do not need to adapt to the FinSH console, this step can be ignored.

```
#define HTCFG_UART1_IPN                UART1

#define _HTCFG_UART1_TX_GPIOX          A
#define _HTCFG_UART1_TX_GPION          4
#define _HTCFG_UART1_RX_GPIOX          A
#define _HTCFG_UART1_RX_GPION          5
```

Figure 32. Modify UART1 Pins

After modifying the files, it is necessary to add the corresponding project template. Developers can create a new template project according to the “Template Project Introduction” section.

Project Generation and Verification

Before generating a project for development, users need to use the menuconfig command to configure the project to ensure that the macro in the rtconfig.h file is consistent with the migrated chip. Open the Env tool in the path “rt-thread-master\bsp\ht32f50343”, input the “menuconfig.exe” command to enter the configuration page.

First, select the chip to be used in the project, there are two options in the “Hardware Drivers Config” → “Chip Configuration”, which are the “Select the kernel” and “Select the chip you are using”. The “Select the kernel” option is used to select different kernels, CORTEX_M0 or CORTEX_M3. The “Select the chip you are using” option is used to select the specific chip, the optional content of this option lists the chips of the corresponding kernel according to the kernel selected by the “Select the kernel” option. Therefore, users need to select the corresponding kernel first, and then select the specific chip.

Taking the ht32f50343 as an example, select CORTEX_M0 first, then select the HT32F50343, as shown in Figure 33. For the specific chip kernel, refer to the datasheet. Note that if the corresponding chip is not listed in the above options, it means that the chip does not support RT-Thread currently.

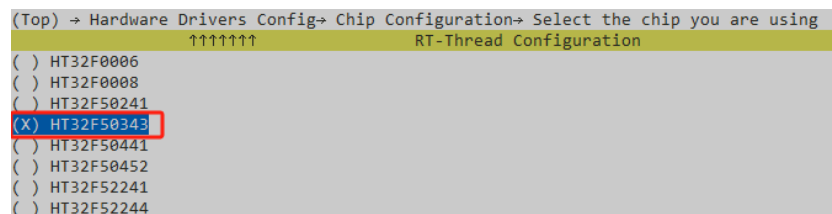


Figure 33. Chip Selection

Then configure the FinSH console, modify the corresponding parameter of “the device name for console” in the “RT-Thread Kernel” menu to “uart1”, as shown in Figure 34. This is because the RT-Thread FinSH console uses the uart1. If developers do not need to adapt to the FinSH console, this step can be ignored and the “Using console for rt_kprintf” option should be unchecked at the same time.

```
[ ] Using the scheduler thread context
[*] Using console for rt_kprintf
(128) the buffer size for console log printf
(uart1) the device name for console
[ ] Use atomic implemented in stdatomic.h
(32) Max number of backtrace level
```

Figure 34. Configure the Serial Port Used by the FinSH Console

Then, in the “Hardware Drivers Config” → “On-chip Peripheral Drivers” → “Enable UART” menu, select the “Enable UART1” option, as shown in Figure 35. After this, the UART1 can be used normally.

```
(Top) - Hardware Drivers Config -> On-chip Peripheral Drivers -> Enable UART g
↑↑↑↑↑↑↑ RT-Thread Configuration
[ ] Enable USART0
[ ] Enable USART1
[ ] Enable UART0
[*] Enable UART1
(uart1) uart1 bus name (NEW)
```

Figure 35. Enable UART1

After the above configurations have been completed, save the configurations and exit to the Env tool page, the Env tool will automatically update the macros in the rtconfig.h file. Refer to the “Build a Project” section, input the command “scons ---target=mdk5” to build a project for Keil_v5. As the LED pins in the HT32F50343 development board, the ESK32-30515, and the HT32F52352 development board, the ESK32-30501, are different, before compiling, open the new project and modify the LED pins in the main.c file to PC3 and PB6 as shown in Figure 36. Compile and verify the project after modification. For the verification effect, refer to the “Compile, Download and Verify” section. If the verification effect is consistent with configurations, it means that the new project migration is successful.

```
/* defined the led1 pin: pc3 */
#define LED1_PIN GET_PIN(C, 3)
/* defined the led2 pin: pb6 */
#define LED2_PIN GET_PIN(B, 6)
```

Figure 36. Modify LED Pins

Writing Method for On-Chip Peripheral Drivers

The RT-Thread operating system provides a unified calling interface and framework for various peripheral drivers. Therefore, the driver development must strictly follow the RT-Thread driver framework to implement the underlying hardware driver. Figure 37 is the RT-Thread related I/O device framework, which is divided into three layers. From top to bottom, they are I/O device management layer, device driver framework layer and device driver layer. The application obtains the correct device driver through the I/O device management interface, and then uses this device driver to perform data interaction with the underlying I/O hardware device.

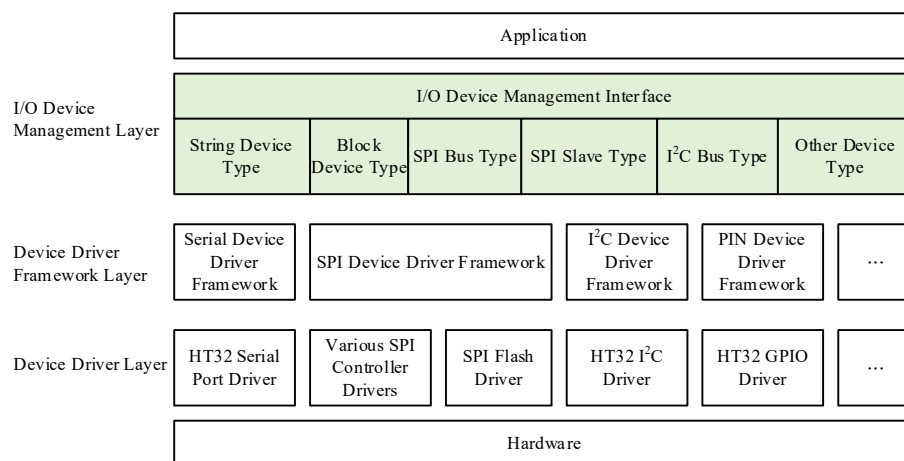


Figure 37. I/O Device Framework

Single Peripheral Driver Framework

By understanding the I/O device framework, users can clearly understand how the application layer of the RT-Thread operating system calls the underlying driver. Therefore, the following steps are required to implement a driver that can be called by the RT-Thread operating system.

1. Implement the underlying interface function provided by the RT-Thread according to the chip in use.
2. Initialise and register the driver.

The implementation of the above steps is mainly for the RT-Thread operating system to successfully find the driver device and interact with the underlying hardware device. The following takes the UART as an example.

The implementation framework of each peripheral driver in the RT-Thread is declared in the corresponding file in the path "rt-thread-master\components\drivers\include\drivers". Open the serial.h file in this path, which defines the macros and various structures for the UART initialisation and configuration in the upper-layer application. Note that the member variables and functions in the struct `rt_uart_ops` can directly correspond to the underlying hardware. The device driver framework layer of the serial port can operate on the underlying hardware through the interface defined by this structure.

The contents of the struct `rt_uart_ops` are shown in Figure 38. As can be seen from the struct `rt_uart_ops`, the member function pointers of the serial port are the "configure" function, the "control" function and the functions related to data sending and receiving. The underlying driver is written to implement the function of these functions accordingly.

```
/**
 * uart operators
 */
struct rt_uart_ops
{
    rt_err_t (*configure)(struct rt_serial_device *serial,
                          struct serial_configure *cfg);
    rt_err_t (*control)(struct rt_serial_device *serial,
                        int cmd, void *arg);

    int (*putc)(struct rt_serial_device *serial,
                char c);
    int (*getc)(struct rt_serial_device *serial);

    rt_ssize_t (*dma_transmit)(struct rt_serial_device *serial,
                               rt_uint8_t *buf,
                               rt_size_t size,
                               int direction);
};
```

Figure 38. struct rt_uart_ops Contents

Single Peripheral Driver Implementation

Drivers need to be written in conjunction with the use of hardware peripherals to implement the specified framework functions. The steps are as follows:

1. Functional implementation of interface functions

Before implementing these functions, analyse the function that each function needs to implement.

- configure function: Serial port configuration function, it is used to initialise and configure the serial port peripheral.
- control function: Serial port control function, it is used to enable or disable the serial port interrupt.
- putc function: The serial port sends a string data.
- getc function: The serial port receives a string data.
- dma_transmit function: Data transfer function for serial DMA mode.

The specific function implementation process is not described in detail, refer to the “drv_usart.c” file in the path “rt-thread-master\bsp\ht32\libraries\ht32_drivers”.

2. Assign interface functions to device structure members

After completing the functional implementation of interface functions, it is necessary to assign interface functions to the struct rt_uart_ops variables to facilitate subsequent registration to the driver link list, as shown in Figure 39.

```
static const struct rt_uart_ops ht32_usart_ops =
{
    .configure      = ht32_configure,
    .control        = ht32_control,
    .putc           = ht32_putc,
    .getc           = ht32_getc,
    .dma_transmit   = ht32_dma_transmit,
};
```

Figure 39. struct rt_uart_ops Variable Assignment

3. Register the device object in the system driver link list

After completing the above steps, it is necessary to assign the struct `rt_uart_ops` variables to the device object structure in the serial port driver initialisation function and call the system registration function to register the device object into the system so as to be called by the upper-layer application. This initialisation function should be called in the system initialisation process to complete the serial port object assignment and registration. The serial port driver initialisation function is shown in Figure 40.

```
int rt_hw_usart_init(void)
{
    rt_size_t obj_num;
    int index;

    obj_num = sizeof(usart_config) / sizeof(struct ht32_usart);
    struct serial_configure config = RT_SERIAL_CONFIG_DEFAULT;
    rt_err_t result = 0;

    for (index = 0; index < obj_num; index++)
    {
        usart_config[index].serial.ops = &ht32_usart_ops;
        usart_config[index].serial.config = config;

        /* register uart device */
        result = rt_hw_serial_register(&usart_config[index].serial,
                                      usart_config[index].name,
                                      RT_DEVICE_FLAG_RDWR |
                                      RT_DEVICE_FLAG_INT_RX |
                                      RT_DEVICE_FLAG_INT_TX,
                                      &usart_config[index]);

        RT_ASSERT(result == RT_EOK);
    }

    return result;
}
```

Figure 40. Serial Port Driver Initialisation Function

4. Add the peripheral driver initialisation function to the system startup execution function

Use the `INIT_BOARD_EXPORT` macro to add the `rt_hw_usart_init` function to the system startup execution function. After the `rt_hw_usart_init` function has been defined, it can be used as a parameter to the `INIT_BOARD_EXPORT` macro, as shown in Figure 41.

```
INIT_BOARD_EXPORT(rt_hw_usart_init);
```

Figure 41. Add `rt_hw_usart_init` Function to System Startup Execution Function

5. Add the peripheral driver to the system tailoring menu

At this point, the usart driver has been almost configured. To add the driver to the system tailoring menu, users can add some compilation macros for the usart driver, and then add the driver configuration function in the corresponding Kconfig file according to the Kconfig file writing rules.

Conclusion

This document has introduced the simple use of the RT-Thread, the Env tool framework, and the method of adapting more chips based on the Demo. Developers can learn some basics of RT-Thread building and usage. Hopefully, this document will be helpful for developers to develop their programs.

Reference Material

HT32 MCU Project Mutual Migration Method Description (AN0638EN).

For more information, refer to the Holtek official website: <https://www.holtek.com>.

Revision and Modification Information

Date	Author	Issue	Modification Information
2024.09.18	邱鵬偉	V1.00	First Version

Disclaimer

All information, trademarks, logos, graphics, videos, audio clips, links and other items appearing on this website ('Information') are for reference only and is subject to change at any time without prior notice and at the discretion of Holtek Semiconductor Inc. and its related companies (hereinafter 'Holtek', 'the company', 'us', 'we' or 'our'). Whilst Holtek endeavors to ensure the accuracy of the Information on this website, no express or implied warranty is given by Holtek to the accuracy of the Information. Holtek will bear no responsibility for any incorrectness or leakage.

Holtek will not be liable for any damages (including but not limited to computer virus, system problems or data loss) whatsoever arising in using or in connection with the use of this website by any party. There may be links in this area, which allow you to visit the websites of other companies. These websites are not controlled by Holtek. Holtek will bear no responsibility and no guarantee to whatsoever Information displayed at such sites. Hyperlinks to other websites are at your own risk.

Limitation of Liability

In no event shall Holtek Limited be liable to any other party for any loss or damage whatsoever or howsoever caused directly or indirectly in connection with your access to or use of this website, the content thereon or any goods, materials or services.

Governing Law

The Disclaimer contained in the website shall be governed by and interpreted in accordance with the laws of the Republic of China. Users will submit to the non-exclusive jurisdiction of the Republic of China courts.

Update of Disclaimer

Holtek reserves the right to update the Disclaimer at any time with or without prior notice, all changes are effective immediately upon posting to the website.