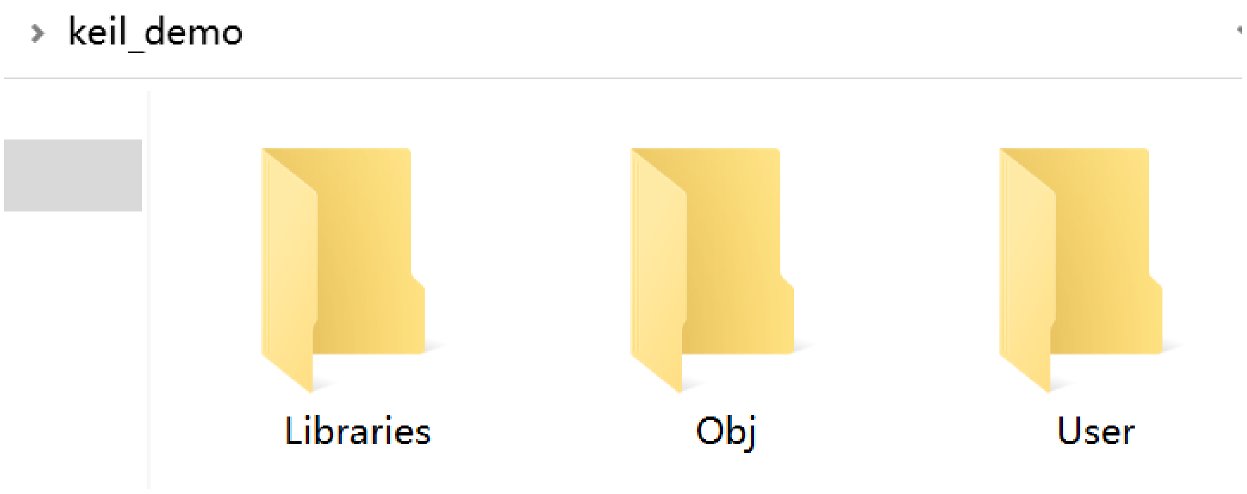
2.3 Development Software User Guide

1.Acquisition of the firmware library

To create a library function project template, you first need to have a firmware library package. There are many versions of the firmware library. Here is the V3.5 version. If you don't mind the trouble, you can also download it from ST's official website, or you can search and download it through Baidu, but you must download the STM32F1 firmware library.

2. Create a library function project

After the firmware library package is obtained, we will officially enter the creation of the project template. We create a folder anywhere on the computer, name it "keil_demo", and then create 3 new folders under it, as follows: (The name of the folder can be arbitrary, we named according to the file type here)

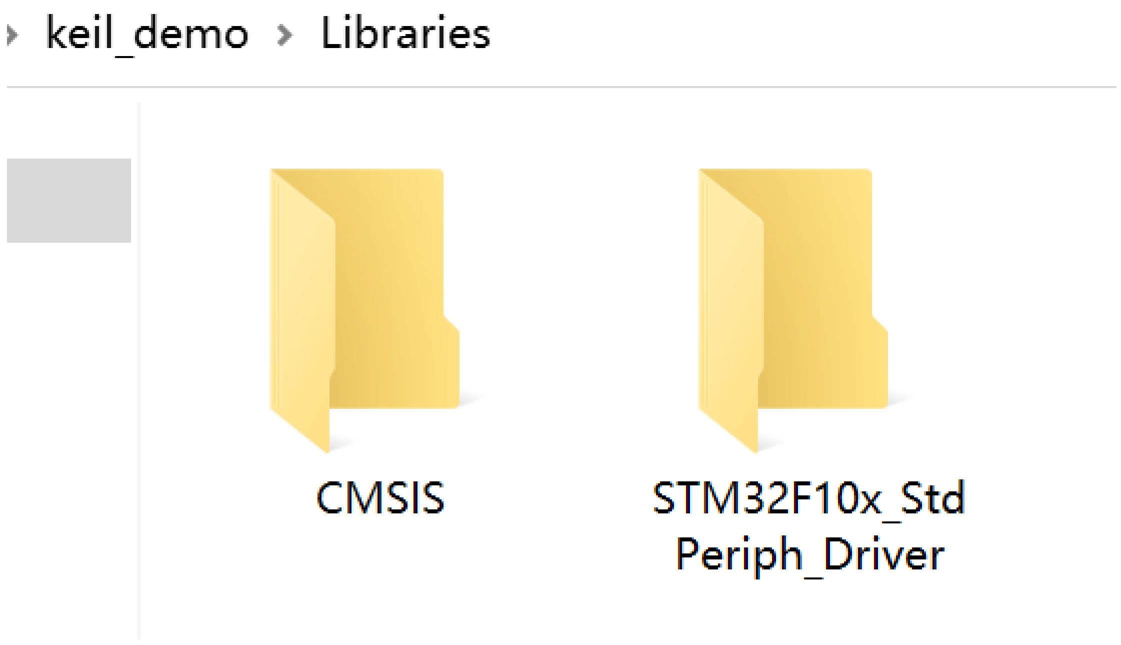


Obj folder: used to store c/assembly/link lists generated by compilation, debugging information, hex files, preview information, packaging libraries and other files.

User folder: used to store user-written main.c, stm32f10x.h header file, stm32f10x_conf.h configuration file, stm32f10x_it.c and stm32f10x_it.h interrupt function files

Libraries folder: used to store CMSIS standard and STM32 peripheral driver files.

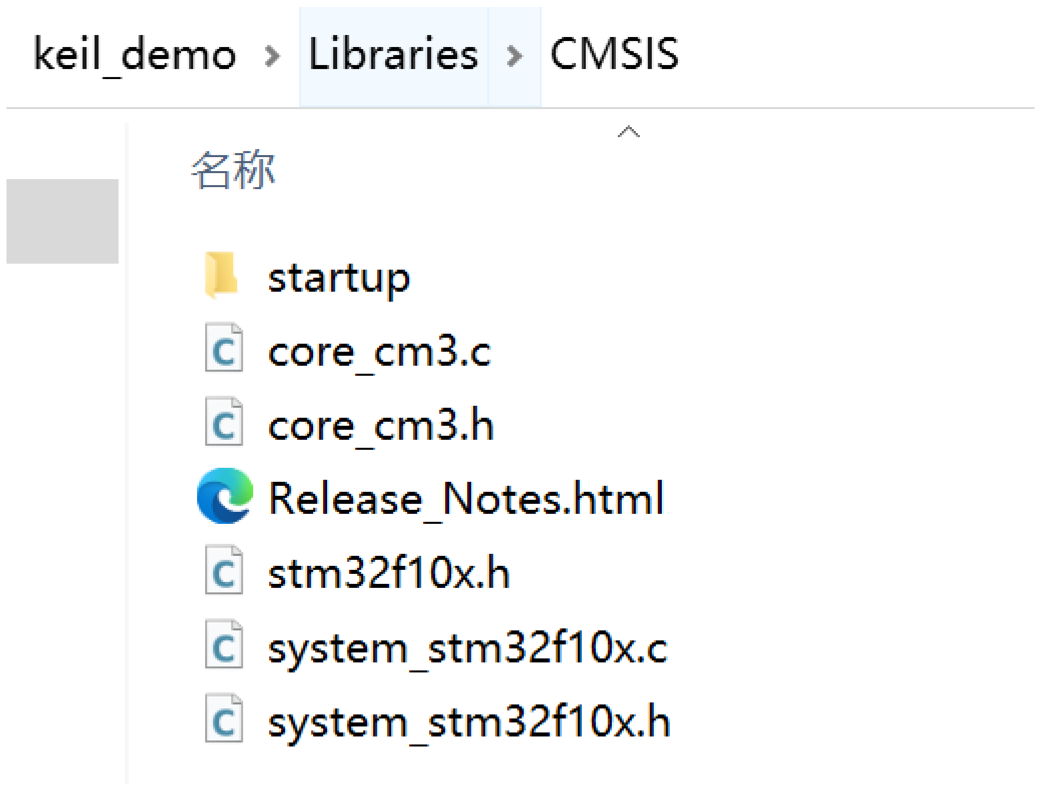
Create two new folders under this folder and name them CMSIS and STM32F10x_StdPeriph_Driver. In fact, these folder names are directly copied from the corresponding folder names of the firmware library, as follows



The CMSIS folder is used to store some CMSIS standard files and startup files; the STM32F10x_StdPeriph_Driver folder is used to store STM32 peripheral driver files.

2.1.Reuse firmware library files

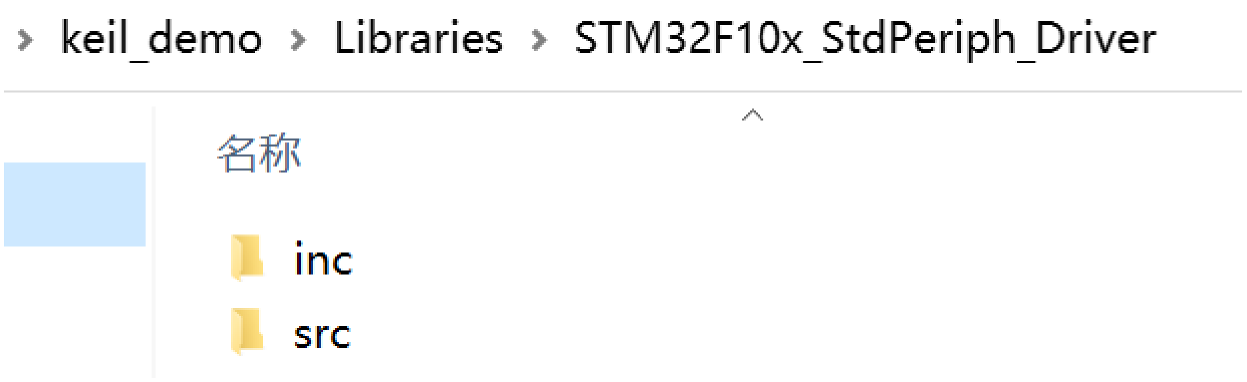
Copy the files in STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\CMSIS\CM3\CoreSupport to CMSIS, and then copy all the files in STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x to CMSIS. The final file in CMSIS is shown in the figure



startup
core_cm3.c
core_cm3.h
Release_Notes.html
stm32f10x.h
system_stm32f10x.c
system_stm32f10x.h

Copy the inc and src files directly from the firmware library "STM32 latest firmware library v3.5\Libraries\STM32F10x_StdPeriph_Driver" directory to the STM32F10x_StdPeriph_Driver folder. It stores the STM32 standard peripheral driver files, the src directory stores the source files of the peripheral drivers, and the inc directory stores the corresponding header files.

The files in the STM32F10x_StdPeriph_Driver folder are as follows:

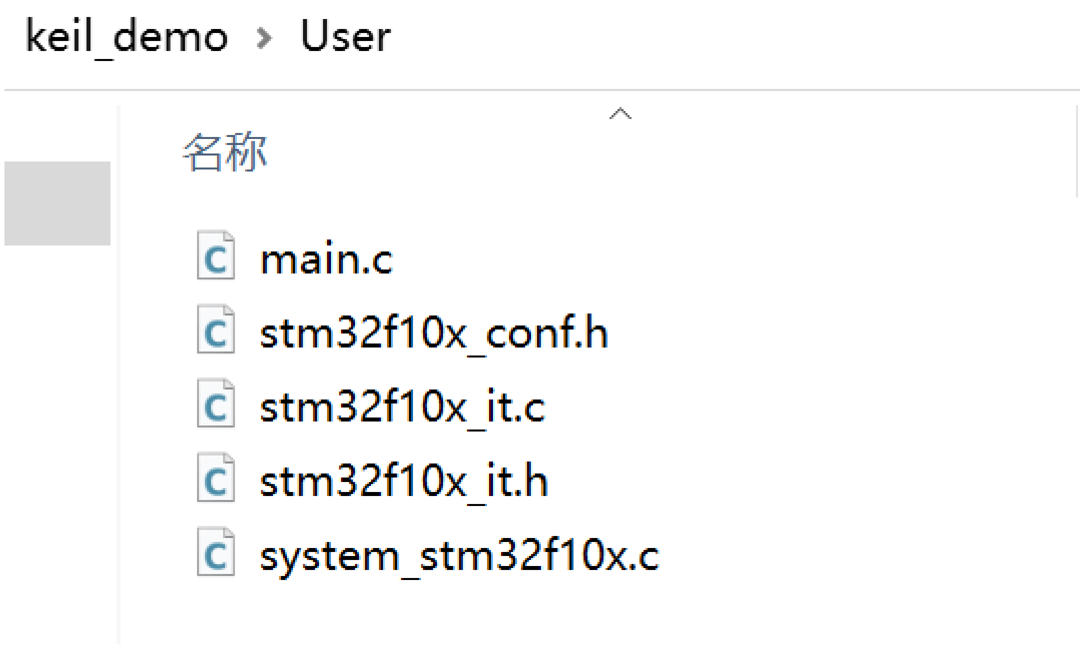


› keil_demo › Libraries › STM32F10x_StdPeriph_Driver

名称

inc
src

Copy the c and h files in STM32F10x_StdPeriph_Lib_V3.5.0\Project\STM32F10x_StdPeriph_Template into USER, as shown in the figure below:



keil_demo › User

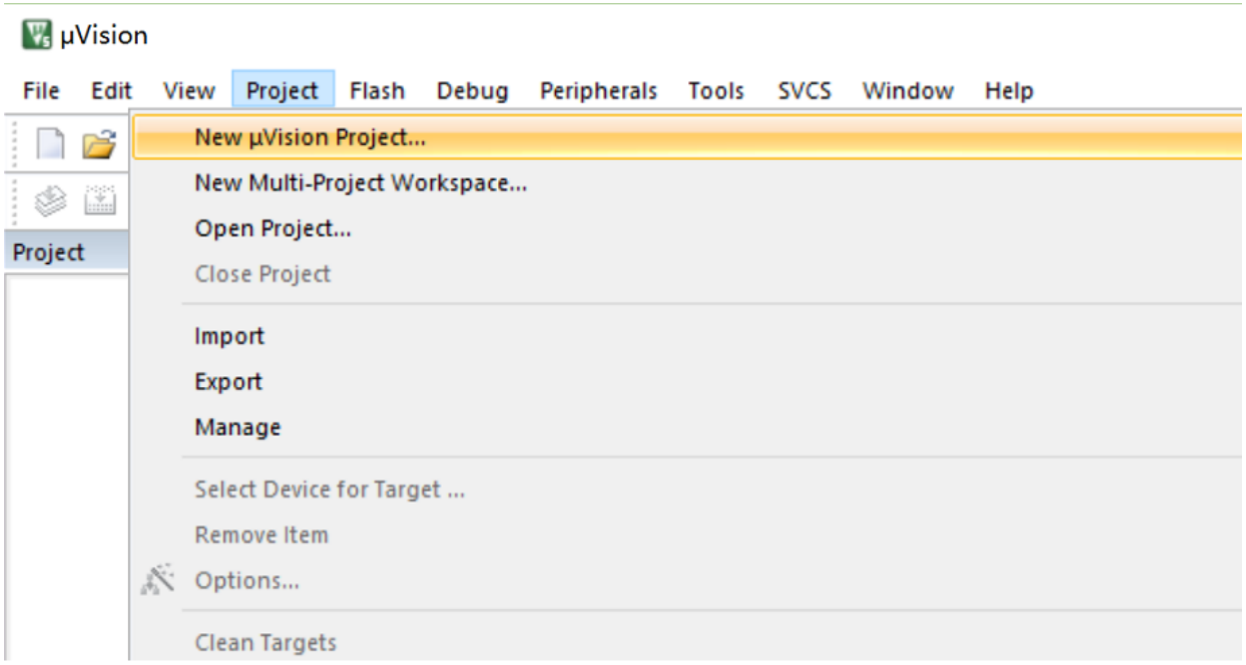
名称

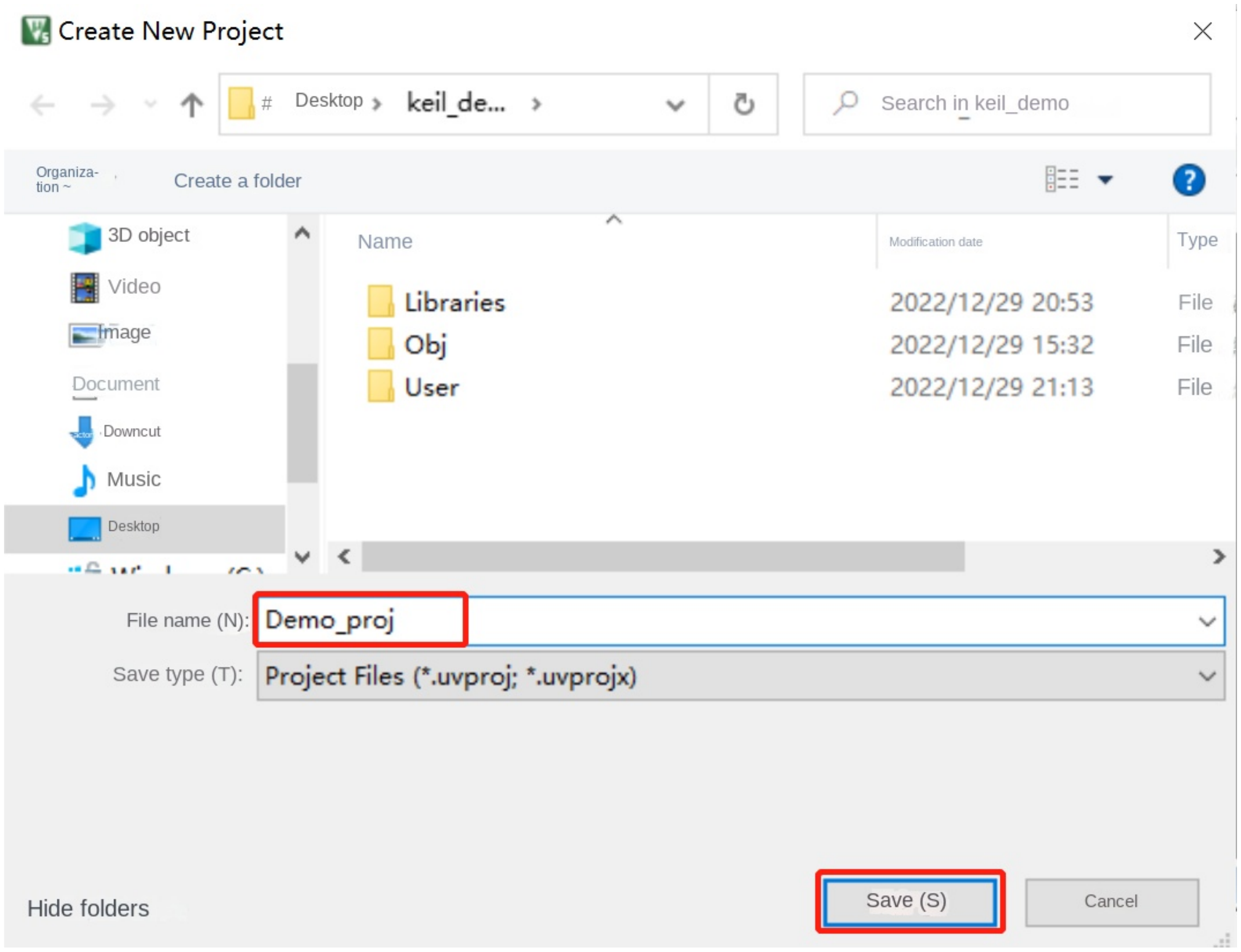
main.c
stm32f10x_conf.h
stm32f10x_it.c
stm32f10x_it.h
system_stm32f10x.c

At this point, the firmware library file required to create the library function template has been copied.

2.2. create a project

Open the KEIL5 software and create a new project. The project name should be named according to personal preference, but be careful to use English. If you use the Chinese name, there may be some strange errors. Here we name it Demo_proj and save it directly in the "keil_demo" file created at the beginning Clip down. Specific steps are as follows:



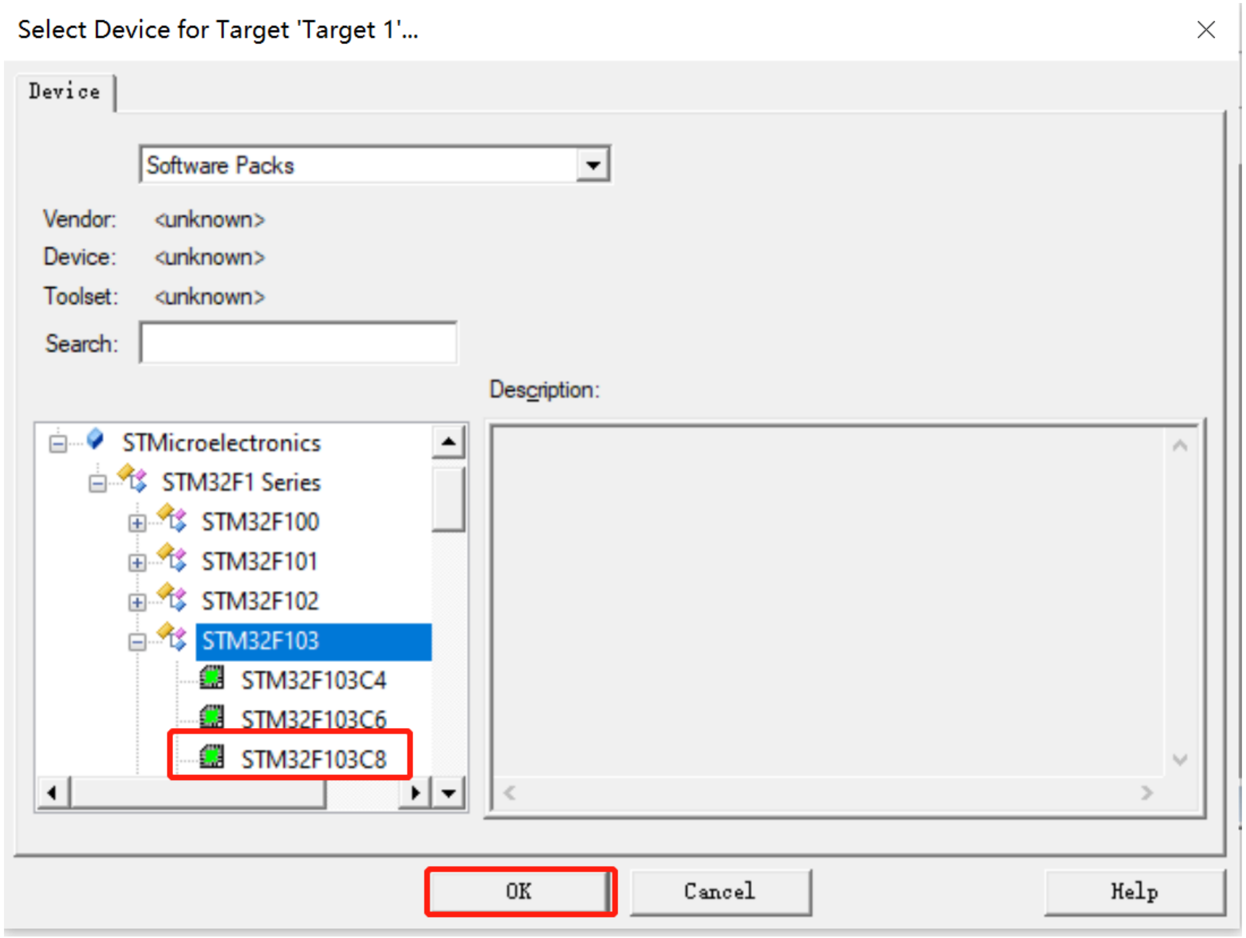


Name	Modification date	Type
Libraries	2022/12/29 20:53	File
Obj	2022/12/29 15:32	File
User	2022/12/29 21:13	File

File name (N): Demo_proj

Save type (T): Project Files (*.uvproj; *.uvprojx)

Chip model selection STM32F103C8


Select Device for Target 'Target 1'...

Device

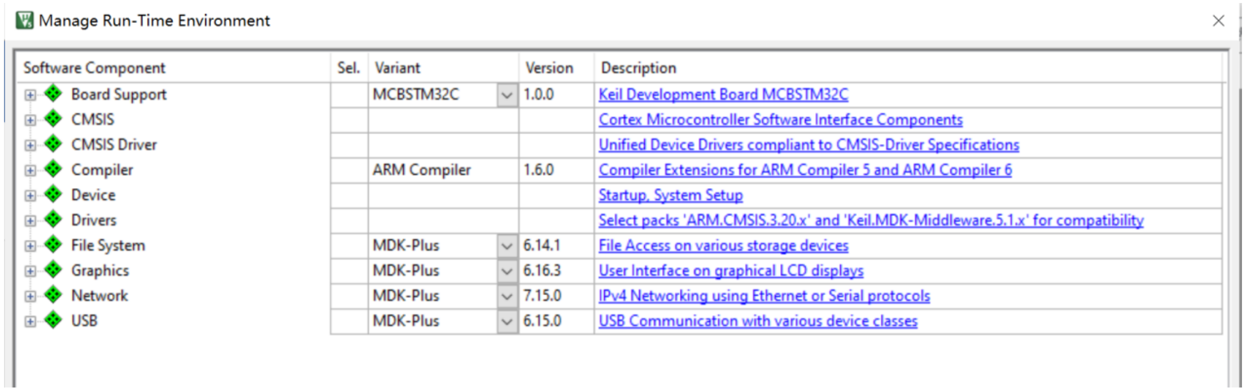
Software Packs

Vendor: <unknown>
Device: <unknown>
Toolset: <unknown>
Search:

Description:

STMicroelectronics
STM32F1 Series
STM32F100
STM32F101
STM32F102
STM32F103
STM32F103C4
STM32F103C6
STM32F103C8

After selecting the CPU and clicking OK, the interface for adding firmware library files online will pop up, as follows:

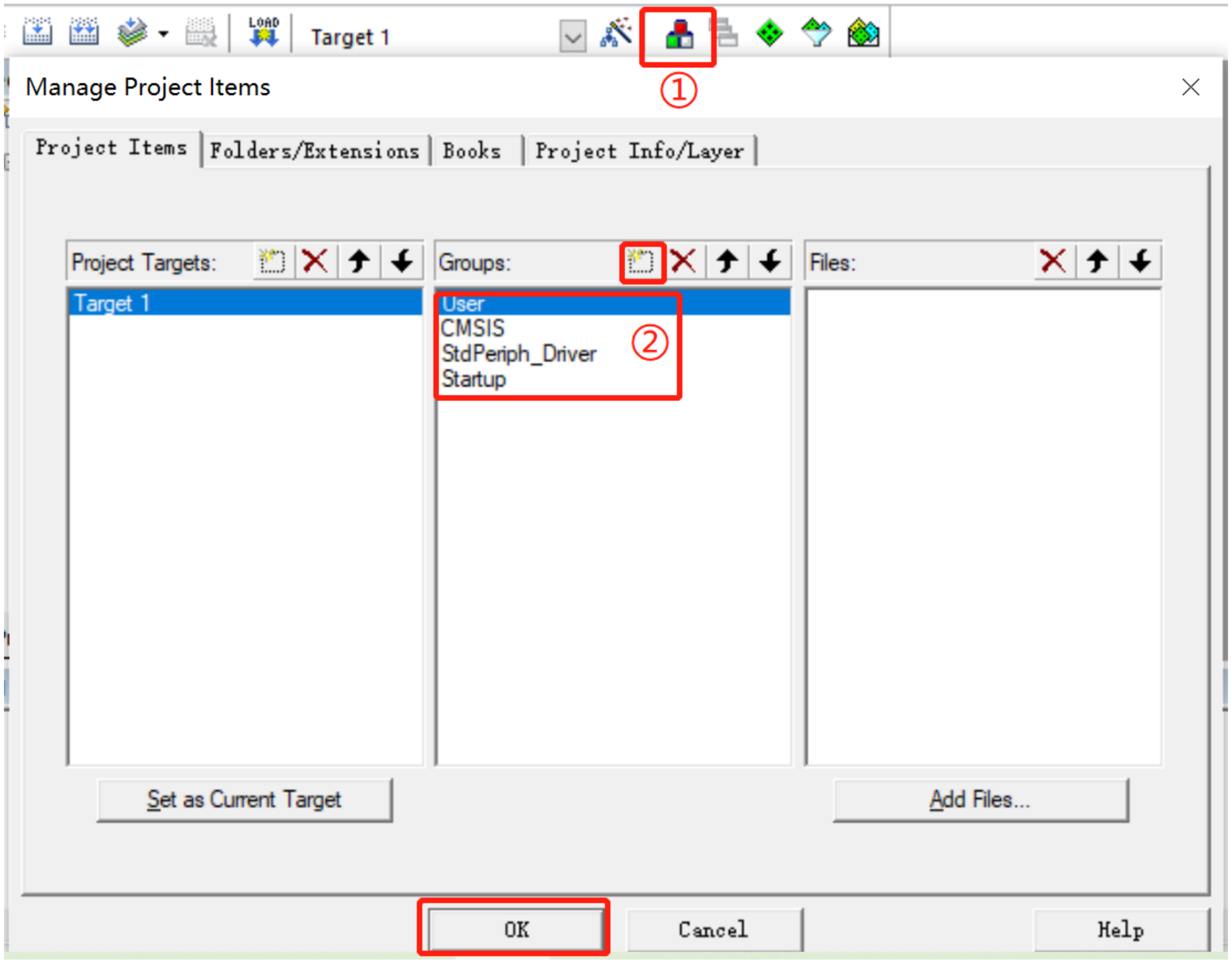

Manage Run-Time Environment

Software Component	Sel.	Variant	Version	Description
Board Support		MCBSTM32C	1.0.0	Keil Development Board MCBSTM32C
CMSIS				Cortex Microcontroller Software Interface Components
CMSIS Driver				Unified Device Drivers compliant to CMSIS-Driver Specifications
Compiler		ARM Compiler	1.6.0	Compiler Extensions for ARM Compiler 5 and ARM Compiler 6
Device				Startup, System Setup
Drivers				Select packs 'ARM.CMSIS.3.20.x' and 'Keil.MDK-Middleware.5.1.x' for compatibility
File System		MDK-Plus	6.14.1	File Access on various storage devices
Graphics		MDK-Plus	6.16.3	User Interface on graphical LCD displays
Network		MDK-Plus	7.15.0	IPv4 Networking using Ethernet or Serial protocols
USB		MDK-Plus	6.15.0	USB Communication with various device classes

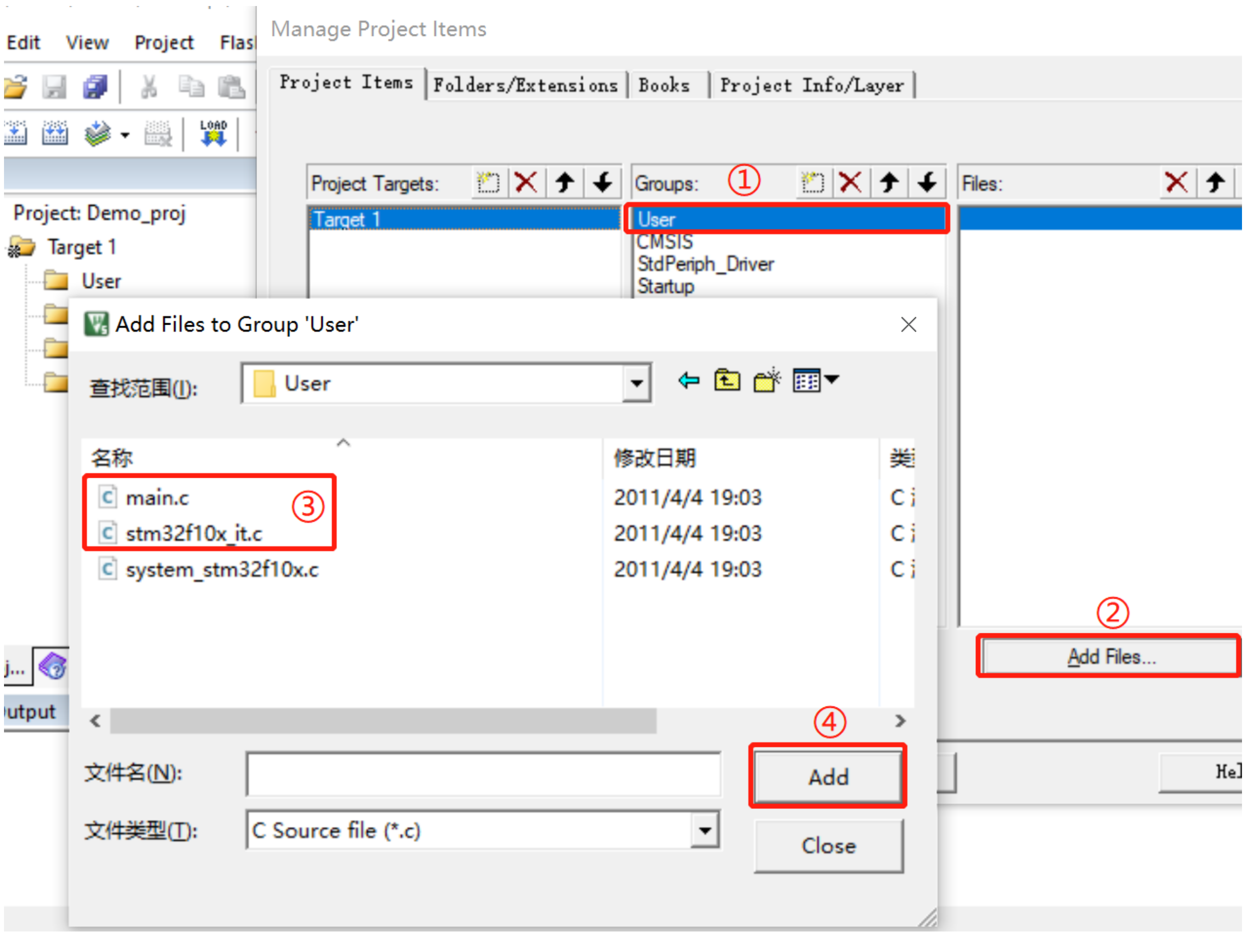
2.3. Add files to the project

Add files to the newly created project. The files are obtained from the "keil_demo" folder. Double-click the Group folder to display the path to add files, and then select the file. If we add all the files in the "keil_demo" directory to the default group of Group, it is obviously very messy, and it is extremely inconvenient for us to find project files and project maintenance. So here we need to build a new project group according to the file type. The operation steps are as follows:

Click the "Manage Project Button" to create CMSIS, User, Startup, StdPeriph_Driver files in turn

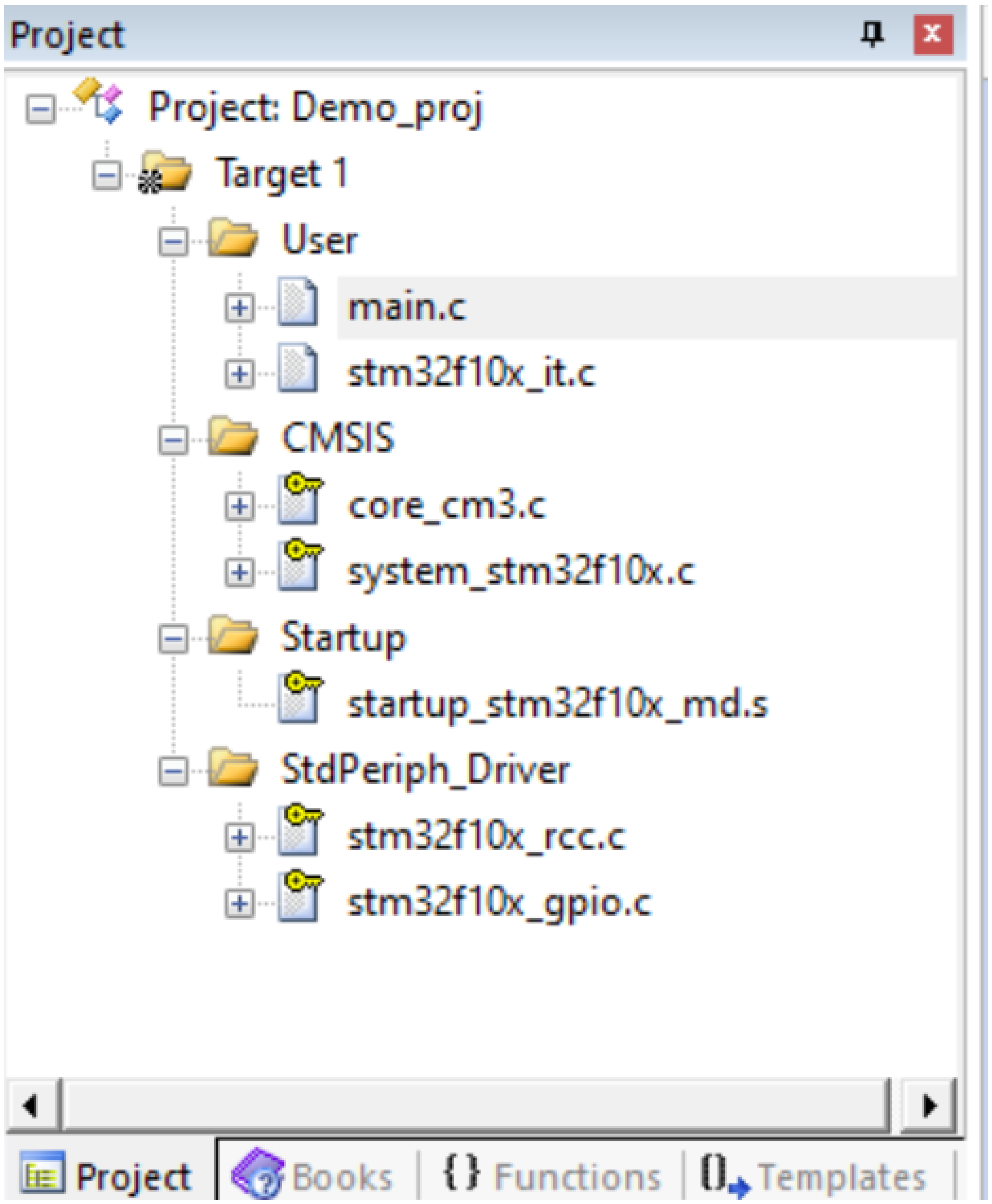


Then add the corresponding project file to the folder



CMSIS selects .c files (it will be added automatically when .h is compiled), USER selects all .c files, STARTUP selects startup_stm32f10x_hd.s in CMSIS\startup\arm, and adds gpio.c and rcc in src to StdPeriph_Driver The .c file is enough, click OK after adding.

After adding all files to the corresponding project group, as shown below:

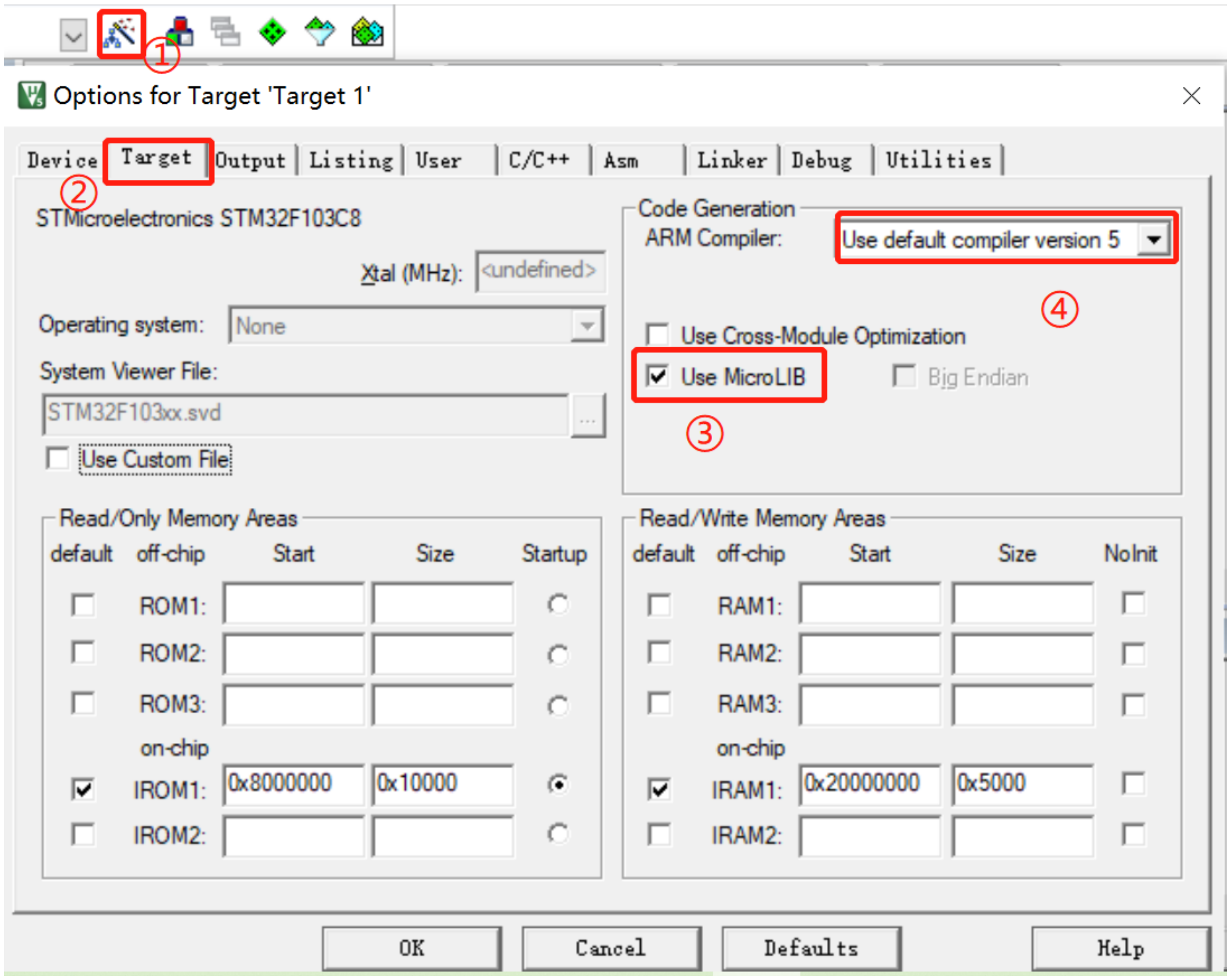


In the StdPeriph_Driver project group, we only added 2 source files. For STM32 program development, these 2 files are usually required. Other peripheral source files are added according to whether the peripherals are used. If you put all the source files It is no problem to add all of them, but the project will be slower when compiling, so our principle is to add the source file of which peripheral is used.

2.4 Configure Magic Wand Tab

The configuration work in this step is very important. Many people found that they could not find the HEX file after compiling the program, and some people could not print the information when they did the printf experiment later.

(1) Select the microlibrary "Use MicroLib" in Target, select 5 for the compiler version, and keep other settings as default. The configuration is as follows:



STMicroelectronics STM32F103C8

Xtal (MHz): <undefined>

Operating system: None

System Viewer File: STM32F103xx.svd

☐ Use Custom File

Code Generation

ARM Compiler: Use default compiler version 5

☐ Use Cross-Module Optimization

☑ Use MicroLIB ☐ Big Endian

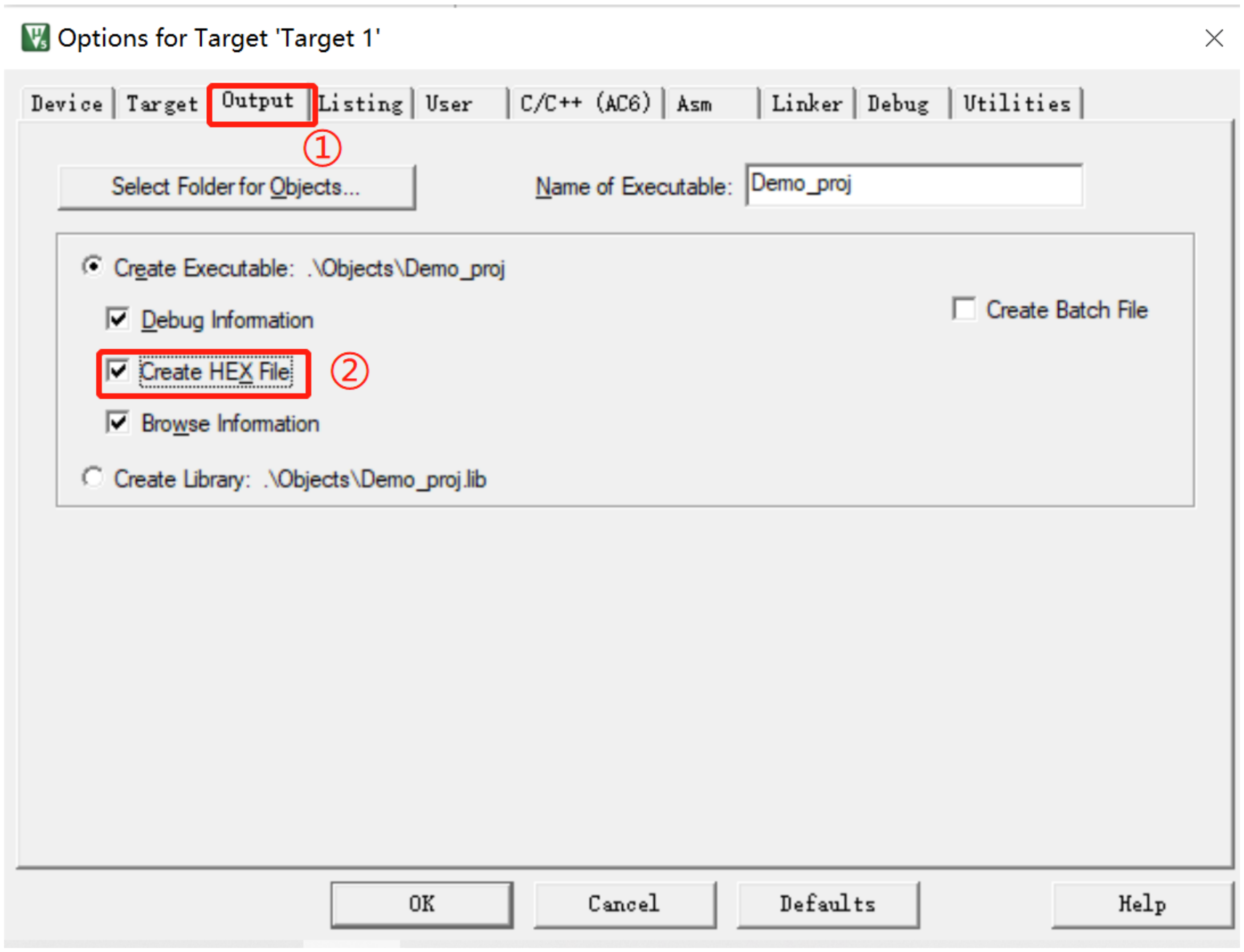
Read/Only Memory Areas

default	off-chip	Start	Size	Startup
☐	ROM1:			○
☐	ROM2:			○
☐	ROM3:			○
	on-chip			
☑	IROM1:	0x8000000	0x10000	●
☐	IROM2:			○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
☐	RAM1:			☐
☐	RAM2:			☐
☐	RAM3:			☐
	on-chip			
☑	IRAM1:	0x20000000	0x5000	☐
☐	IRAM2:			☐

(2) In the Output tab, locate the output folder to the Obj folder in our project directory. If you want to generate a hex file during compilation, then check the Create HEX File option. The configuration is as follows


Options for Target 'Target 1'

Device | Target | Output | Listing | User | C/C++ (AC6) | Asm | Linker | Debug | Utilities

Select Folder for Objects...

Name of Executable: Demo_proj

(●) Create Executable: .\Objects\Demo_proj

[✓] Debug Information

[✓] Create HEX File

[✓] Browse Information

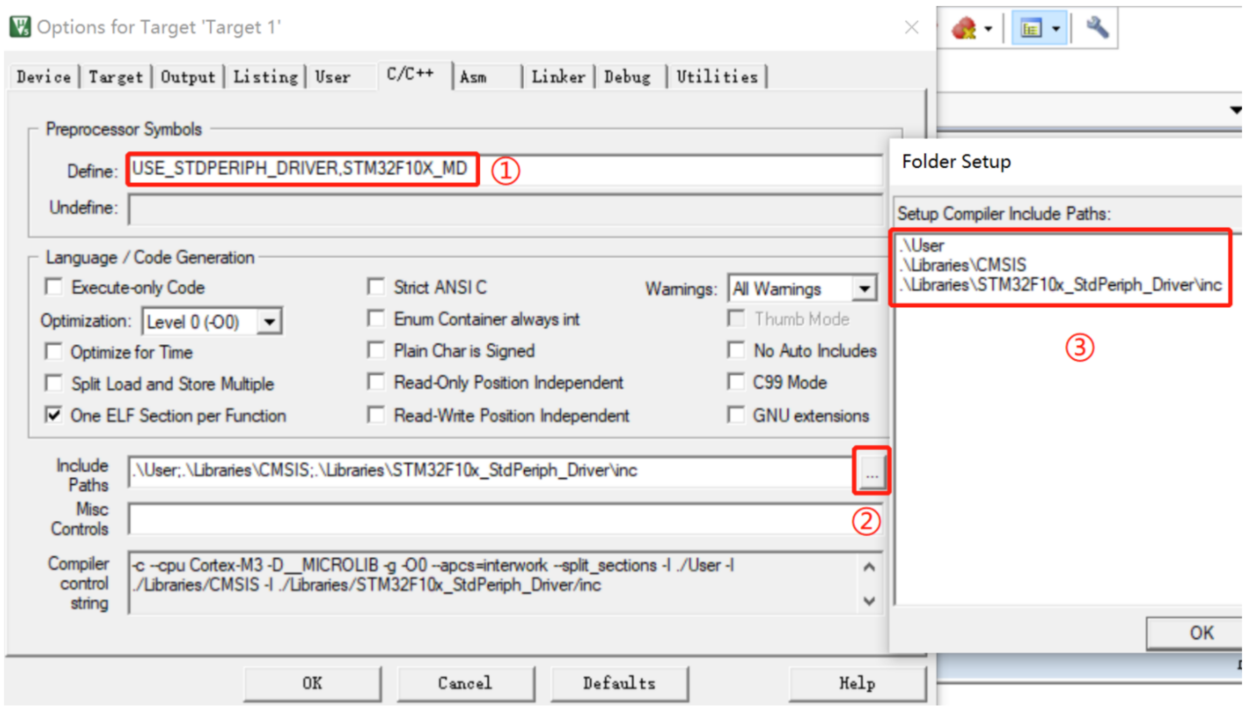
( ) Create Library: .\Objects\Demo_proj.lib

[ ] Create Batch File

OK | Cancel | Defaults | Help

(3) C/C++ tab configuration

Because the library function template is created, it is necessary to define the macro of the processor type and library. Copy these two macros in the Define column: USE_STDPERIPH_DRIVER, STM32F10X_MD. After setting the macro, we need to add it to the project The file path in the group is included, also in the C/C++ tab, the specific steps are as follows:


Options for Target 'Target 1'

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

Preprocessor Symbols

Define: USE_STDPERIPH_DRIVER,STM32F10X_MD ①

Undefine:

Language / Code Generation

- [ ] Execute-only Code
- Optimization: Level 0 (-O0)
- [ ] Optimize for Time
- [ ] Split Load and Store Multiple
- [✓] One ELF Section per Function
- [ ] Strict ANSI C
- [ ] Enum Container always int
- [ ] Plain Char is Signed
- [ ] Read-Only Position Independent
- [ ] Read-Write Position Independent
- Warnings: All Warnings
- [ ] Thumb Mode
- [ ] No Auto Includes
- [ ] C99 Mode
- [ ] GNU extensions

Include Paths: .\User;.\Libraries\CMSIS;.\Libraries\STM32F10x_StdPeriph_Driver\inc ②

Misc Controls:

Compiler control string: -c --cpu Cortex-M3 -D__MICROLIB -g -O0 --apcs=interwork --split_sections -I ./User -I ./Libraries/CMSIS -I ./Libraries/STM32F10x_StdPeriph_Driver/inc

OK | Cancel | Defaults | Help

Folder Setup

Setup Compiler Include Paths:

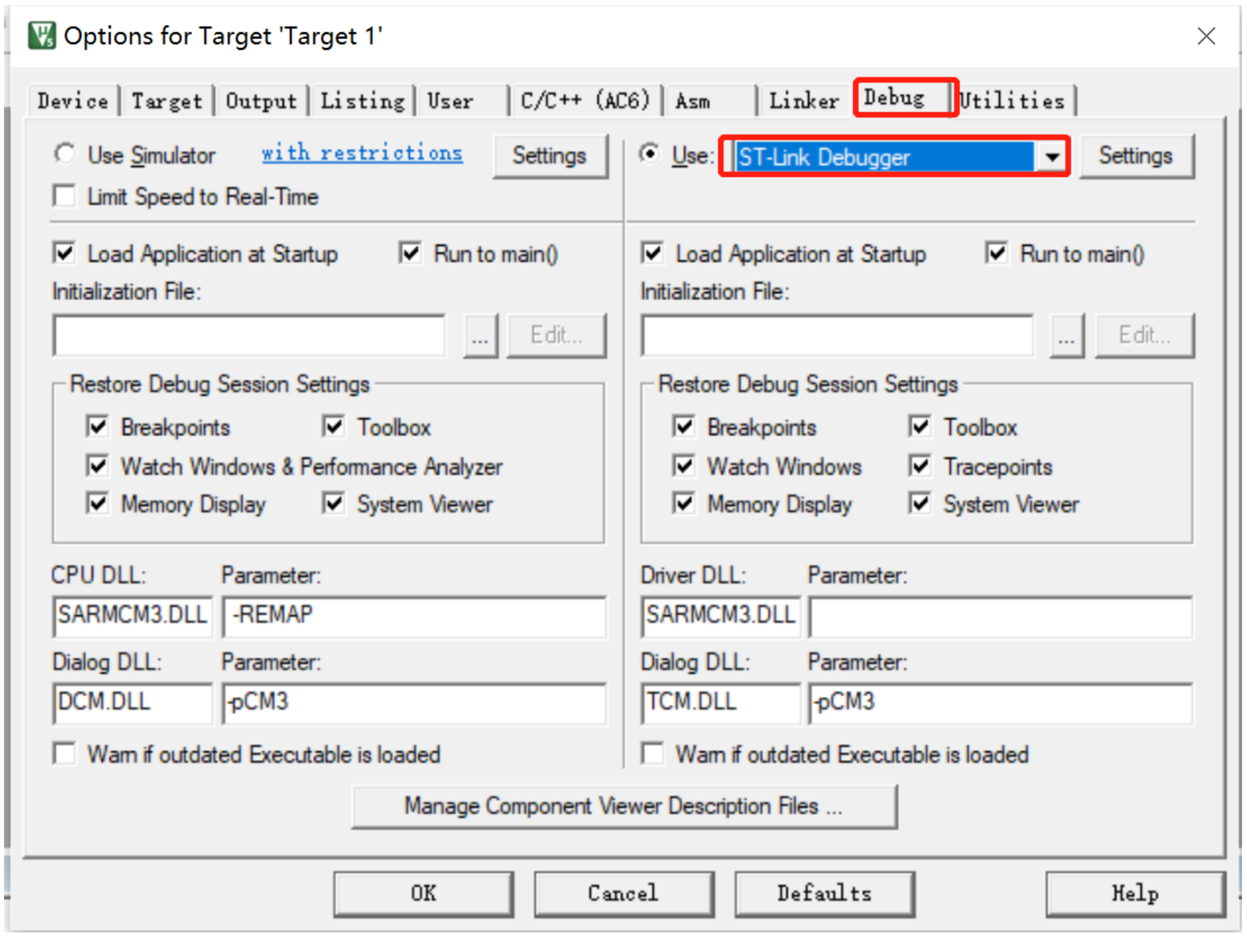
.\User
.\Libraries\CMSIS
.\Libraries\STM32F10x_StdPeriph_Driver\inc

③

OK

(4) ARM emulator configuration

When your emulator is installed with the driver and connected to the computer and the board, the development board can be powered on, and the program and simulation can be downloaded as long as the corresponding configuration is performed in the KEIL software. Before using it, you need to choose the corresponding tool. I use ST-LINK here, and you can choose according to the tools you use.


Options for Target 'Target 1'

Device | Target | Output | Listing | User | C/C++ (AC6) | Asm | Linker | Debug | Utilities

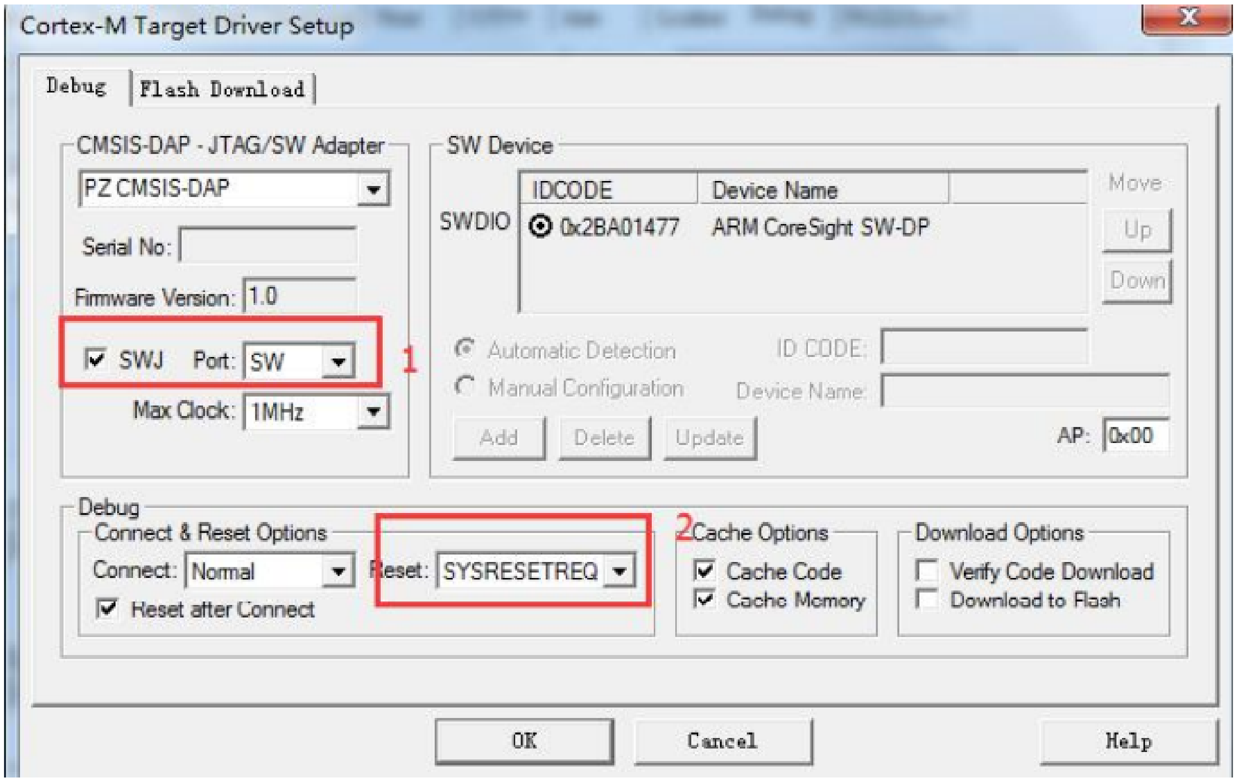
( ) Use Simulator with restrictions Settings
- [ ] Limit Speed to Real-Time
- [✓] Load Application at Startup
- [✓] Run to main()
- Initialization File:
- Restore Debug Session Settings: [✓] Breakpoints [✓] Toolbox [✓] Watch Windows & Performance Analyzer [✓] Memory Display [✓] System Viewer
- CPU DLL: SARMCM3.DLL Parameter: -REMAP
- Dialog DLL: DCM.DLL Parameter: -pCM3
- [ ] Warn if outdated Executable is loaded

(•) Use: ST-Link Debugger Settings
- [✓] Load Application at Startup
- [✓] Run to main()
- Initialization File:
- Restore Debug Session Settings: [✓] Breakpoints [✓] Toolbox [✓] Watch Windows [✓] Tracepoints [✓] Memory Display [✓] System Viewer
- Driver DLL: SARMCM3.DLL Parameter:
- Dialog DLL: TCM.DLL Parameter: -pCM3
- [ ] Warn if outdated Executable is loaded

Manage Component Viewer Description Files ...

OK | Cancel | Defaults | Help

After selecting the CMSIS-DAP Debugger model, click Settings, and the following interface will pop up. If your model is not wrong, it will automatically recognize your ARM emulator ID, and then you can set the SW or JTAG mode and reset method. It is recommended You use SW mode, if you use JTAG mode, some of the following routines occupy part of the pins. So it is recommended to use SW mode to download and debug. The specific setting steps are as follows:


Cortex-M Target Driver Setup

Debug | Flash Download

CMSIS-DAP - JTAG/SW Adapter: PZ CMSIS-DAP
Serial No:
Firmware Version: 1.0
[✓] SWJ Port: SW
Max Clock: 1MHz

SW Device

	IDCODE	Device Name
SWDIO	0x2BA01477	ARM CoreSight SW-DP

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

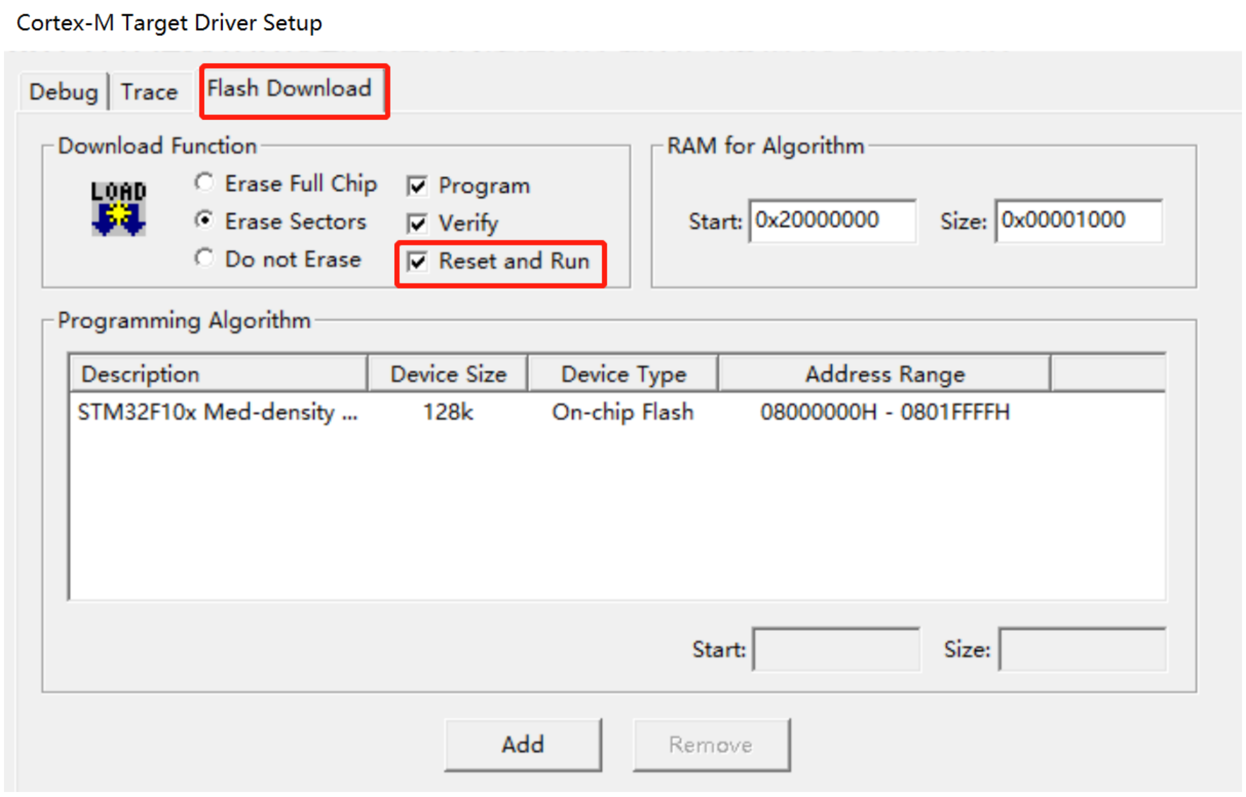
Debug
Connect & Reset Options
Connect: Normal
Reset: SYSRESETREQ
[✓] Reset after Connect

Cache Options
[✓] Cache Code
[✓] Cache Memory

Download Options
[ ] Verify Code Download
[ ] Download to Flash

(5) Chip model selection

Also set in the Debug tab in the previous step, click Settings, select the Flash Download tab, in the Reset and Run option in box 3, if it is checked, it will automatically reset and run after the program is downloaded, if it is not checked After the program is downloaded, you need to press the reset button on the development board to run it. Usually we choose to tick. The specific configuration is as follows:


Cortex-M Target Driver Setup

Debug | Trace | Flash Download

Download Function

- ( ) Erase Full Chip
- (•) Erase Sectors
- ( ) Do not Erase
- [✓] Program
- [✓] Verify
- [✓] Reset and Run

RAM for Algorithm

Start: 0x20000000 Size: 0x00001000

Programming Algorithm

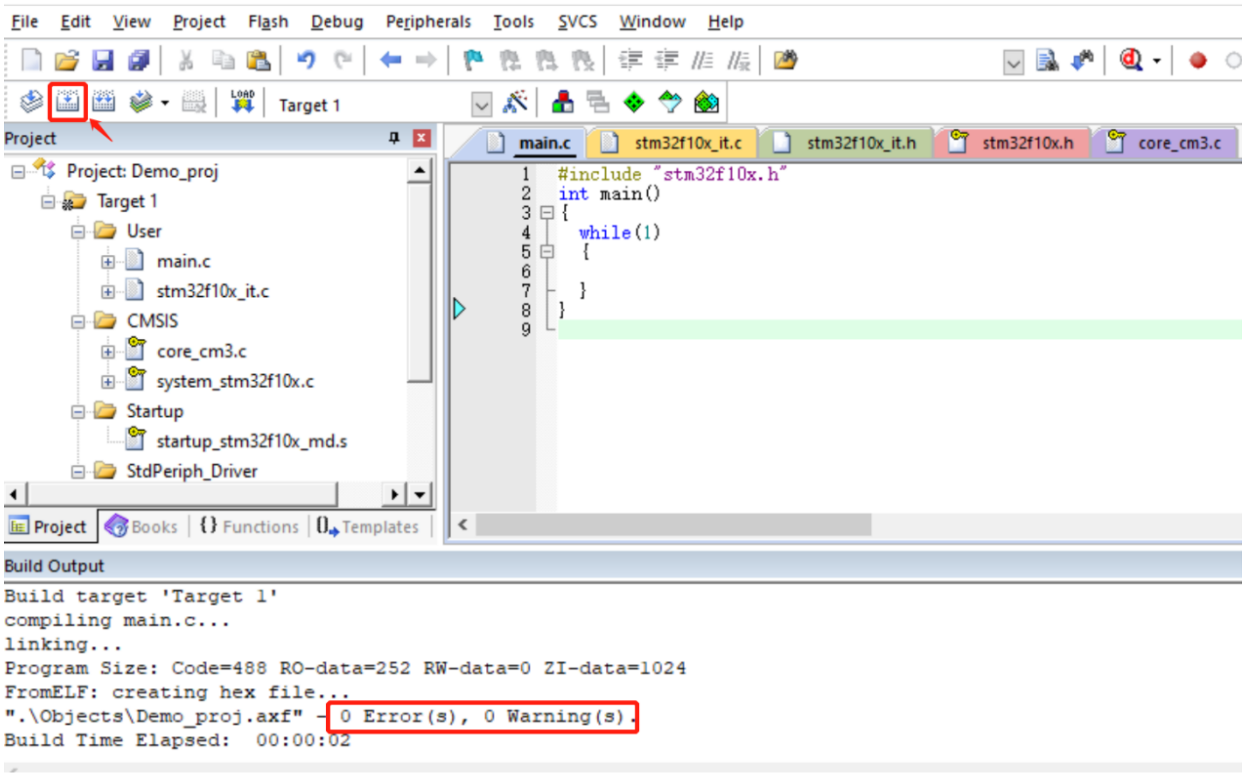
Description	Device Size	Device Type	Address Range
STM32F10x Med-density ...	128k	On-chip Flash	08000000H - 0801FFFFH

Start: Size:

Finally click the OK button. Then double-click the main.c file in the engineering group and you will find that there are a lot of codes in it. This is copied directly from the template provided by ST Company, so we delete all the contents in the main.c file and write the simplest The framework program, we enter the following content:

```
#include "stm32f10x.h"

int main()

{

while(1)

{



}

}
```

Then we compile the project, and the result after compilation is 0 errors and 0 warnings, indicating that the library function template we created is completely correct. as follows:



```
#include "stm32f10x.h"
int main()
{
  while(1)
  {

  }
}
```

```
Build target 'Target 1'
compiling main.c...
linking...
Program Size: Code=488 RO-data=252 RW-data=0 ZI-data=1024
FromELF: creating hex file...
".\Objects\Demo_proj.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:02
```

The first compilation in icon 1 is to compile a single file, the second compilation is to compile the modified files in the project, and the third compilation is to compile all files in the project. Usually we use the middle compilation, which is more efficient. At this point, our first project demo_proj has been successfully created!