ISSN 2522-6398 Vol. 5, Issue 17: 65-69, DOI: 10.25236/FER.2022.051712

The Teaching Analysis and Design of the Construction of C Language Multi-Document Project Based on Keil

Huiguo Cao*, Yanxiang Gong

School of Physics and Electr0nic Engineering, Taishan University, Taian City, Shandong Province, 271000, China
**Corresponding author: tscao@163.com*

Abstract: *The establishment of C language multi-file engineering is a difficult point to deepen the learning of C language, and is also the basis for learning STM32 programming. This article uses the project creation process of 8 LED lights as an example to establish the source program files and header files of each function and hardware. Guide students from the modularization of functions to the modularization of source files, teaching analysis and design of the establishment of multi-file engineering, so that more students can deeply understand and master the creation of multi-file engineering, laying a solid foundation for the deepening of embedded learning.*

Keywords: *C language, Multi file, Modularization, Teaching design*

1. Introduction

Liu Qingfeng, chairman of iFlytek, proposed at the 2016 China Is Forum that artificial intelligence should be elevated to the national strategic level like "Made in China 2025" [1]. Therefore, the development of artificial intelligence is related to the future of the country. Artificial intelligence can not be separated from C language. Building engineering with C language is the mainstream application of embedded control system [2].

At present, colleges and universities generally offer C language, 51 single-chip microcomputer, embedded microcontroller and other courses. Through learning, students can use Keil to establish a single file project. However, the establishment of multi-file project is generally confused and difficult to understand, so that most of the programs written by students after graduation only have one source file (main.c or Miam.cpp). When students open a STM32 project, they will feel confused and have no way to start when they see the dense C project files and header files. Obviously, this is far from enough for the in-depth research and application of embedded technology. Therefore, teaching analysis and design in simple terms through simple examples is the key to guide students to master and learn the establishment of multi document projects.

2. Instructional analysis and design of multi-document engineering establishment

Ordinary C language and SCM textbooks are single C source file projects. Single file projects can solve small and simple problems, and most students can basically understand and accept them. When encountering a slightly larger project, a single source program file will become very long, which may contain thousands of statements. It will be very difficult to maintain, transplant and even read the program. The establishment of multi-file project will completely solve the drawbacks of a single source program file[4], so that each file can only solve its own independent problem, and the reading, maintenance and transplantation of the program will become very easy and convenient. From the establishment of single source program file project to the establishment of multiple source program file project, students must be guided to firmly establish the concept of modular programming; Master the role of the header file and the underlying driver, it is natural to master the establishment of a multi-file project. The following is a teaching analysis and design of the establishment of the multi file project, taking the establishment of the flashing project of eight led lamps as an example.

ISSN 2522-6398 Vol. 5, Issue 17: 65-69, DOI: 10.25236/FER.2022.051712

2.1. Guide students to realize the transformation from functional modular programming to file modular programming, learn to write independent source program files for each function and hardware application

The idea of modular programming is mentioned in every programming language textbook, but in general, modular programming basically emphasizes that the main function calls each module function to solve problems. If the project needs to solve complex problems, there may be hundreds of functions involved. The declaration and establishment of all functions must be written in a source program file, which will inevitably make the source program file complex and lengthy. The emphasis here on modularity is to create each function as its own separate source file, a separate ***.c file. For example, in order to establish the flickering project of 8 LED lights, an independent delay file delay.c, LED flickering LED. c and main function source file main.c are established, and the main function source file is used to call other source files to complete the program function and realize the call from function call to source file. The specific file module is as follows.

2.1.1. Delay function source file delay.c

```
void delay ()
{
    unsigned int m,n;
    for(m=1000;m>0;m--)
    for(n=20;n>0;n--);
}
```

Since this file is only for implementation delay and does not involve hardware address, it does not include "reg52. h" and other header files.

2.1.2. Blinking light function source file led.c

```
#include <reg52.h>
#include "delay.h" //Notice that there is no semicolon
void led()
{
    P1=0x00;
    delay ();
    P1=0xff;
    delay ();
}
```

The flashing light source file led. c involves the definition of the register address of the hardware port and the call of the delay function, so it needs to include the header file "reg52.h" and the header file "delay.h". The function of delay.h is to declare the source file of delay.c. According to the specification of C language, the called function must be declared before calling the function.

2.1.3. Main function source file main.c

```
#include "led.h"
void main()
{
    led();
}
```

According to the functional requirements of the program, only 8 led lights are controlled to flash The c source file has realized its functions. Therefore, you can directly call the led () function in the main function file. Also, since LEd.c is called, "led.h" is introduced in the main function source file. The function of led. h is to declare the led. c source file.

ISSN 2522-6398 Vol. 5, Issue 17: 65-69, DOI: 10.25236/FER.2022.051712

Can see through the above analysis, in addition to the main function is the main place of source files, for each function or hardware wrote a separate source file, the source files make up the engineering of each module, through the main function calls corresponding function module source file and the source file module call each other between the source file, all the functions to complete the project. Since each source file only implements its own small function, each independent source file content is not long, which is conducive to reading and understanding. At the same time, for each source program module, its own header file, namely * * * h files, and their respective header files are mainly used to declare the functions corresponding to their source files. In this way, each source file has independent functions, and each corresponding header file has independent declarations. It is easy to implement whether it is called or included, which is conducive to porting each source file to other suitable projects. Especially with the accumulation of projects, students can accumulate the functions, hardware files and header files that they usually use, so that they can use them in future work and practice. This programming idea also lays a solid foundation for students to learn and understand STM32 programs in the future. The source file of each function or hardware is equivalent to the BSP board level support package.

2.2. Guide students to thoroughly understand the role of header files, and learn to write the header files of individual hardware or functions by themselves

The function of the header file, namely ***. h file, has always been difficult for students to understand and accept. Most students even do not know that they can write header files until graduation, let alone write header files independently. The header file plays a significant role in the modular programming idea of the upper source file. It is because of the function and hardware independent header file that when calling the respective source file, it is equivalent to calling the respective function, must be declared first to call the source file function[3], so the function declaration is a main function of the header file. In addition, the header file can define many macro definitions, including hardware address and global static variable definitions. The hardware can only be used because the macro definition defines the hardware address. In this sense, the header file is equivalent to the underlying driver file; Macro definitions define global static variables. As long as you modify the contents of the header file, you can change everything without having to search and modify in the tedious code. Modular programming of source files requires writing header files to all source files except the main function. Therefore, students must be guided to learn how to write header files independently. Here, the header file that the led flashing lamp project needs to write itself is the header file of the function declaration. The writing method of the header file has strict specifications, and the syntax is consistent with that of the source file. The specific contents of the led. h and delay. h header files are shown below.

2.2.1. Blinking lamp holder file led.h

```
#ifndef _LED_H_
#define _LED_H_
void led();//Lights flashing
#endif
```

2.2.2. The delay function header file delay.h

```
#ifndef _DELAY_H_
#define _DELAY_H_
void delay();//Delay Functions
#endif
```

It can be seen from the above two examples that the writing of the header file is very simple. The macro definition identifier #ifndef and #defineare followed by the header file name in uppercase plus two underscores, and the end ends with #endif, the middle is defined as the specific contents of the header file, mainly the declaration of functions, global static variables or hardware registers and port addresses.

2.3. C language multi - file project based on Keil specific establishment process

2.3.1. Use Keil uVision2 software to establish a New Project

Open Keil uVision2 software, click the "Project" option in the menu bar, and select from the drop-down menu

ISSN 2522-6398 Vol. 5, Issue 17: 65-69, DOI: 10.25236/FER.2022.051712

"New Project" creates a project and names it according to the project function. For example, if it is named LingShuiDeng, the system will automatically save it as a LingShuiDeng.Uv2 project file. Select the CPU as prompted. The Copy Sandard 8051 Startup Code to Project Folder and Add File to Project dialog box is displayed. Select No. At this point, the Keil uVision2 interface looks like Figure 1.

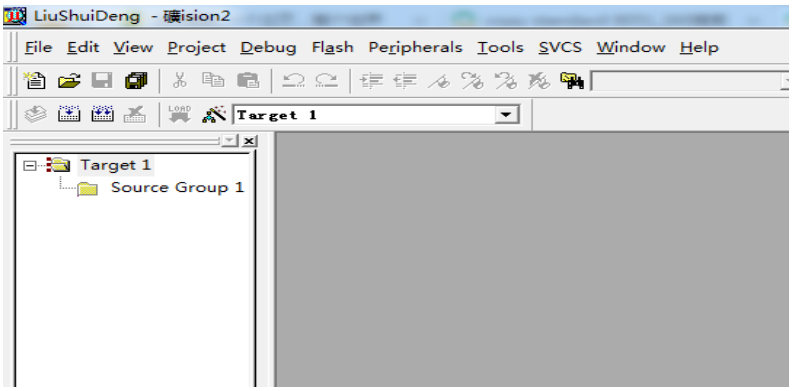

Figure 1: Keil uVision2 project interface

2.3.2. Create, edit and save C program source files and header files

Click "File" option on the menu bar to select or click "New File" button on the toolbar to establish the above-mentioned three source files main.c, LED. c and delay.c respectively. Note that suffix.c must be added when saving, so as to save as C File. Also create two header files, led. h and delay. h. When saving the header file, the suffix must be h。 Right click the "Source Group1" in the project window on the left, and select "Add File to 'Source Group1'" from the menu that appears. Add all the used source files just like adding a single file. Here are three source files. The project interface after adding is shown in Figure 2.

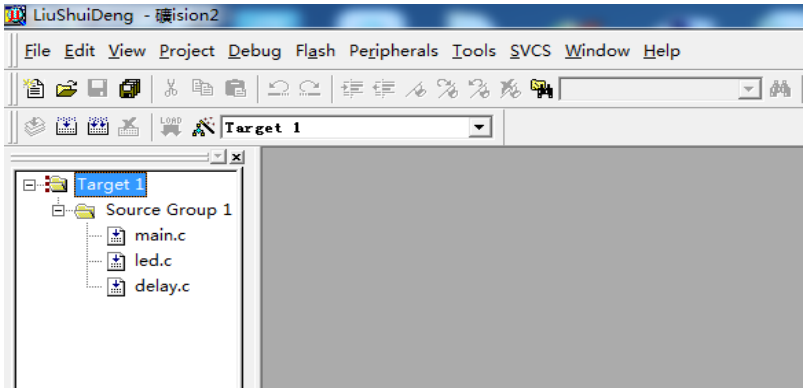

Figure 2: Keil uVision2 project interface after adding multiple source files

2.3.3. Compilation project

Click the "Build target" button in the toolbar to compile the project. After compiling, the header files contained in each source file will be displayed, as shown in Figure 3.

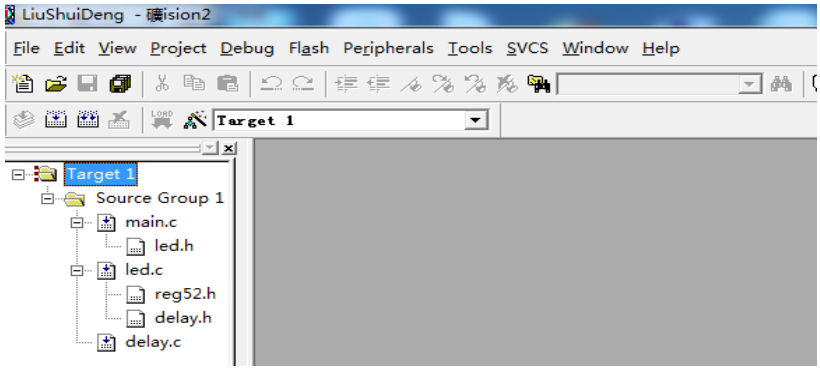

Figure 3: Keil uVision2 interface compiled from multiple source file projects

ISSN 2522-6398 Vol. 5, Issue 17: 65-69, DOI: 10.25236/FER.2022.051712

2.3.4. Generating HEX files

After compiling, if there is no syntax error, click the "options for Target" button on the toolbar, select the "Output" option in the pop-up window, and select the "Create HEX File" option, that is, check the box in front of it. Recompile to generate the HEX file. The interface is shown in Figure 4.

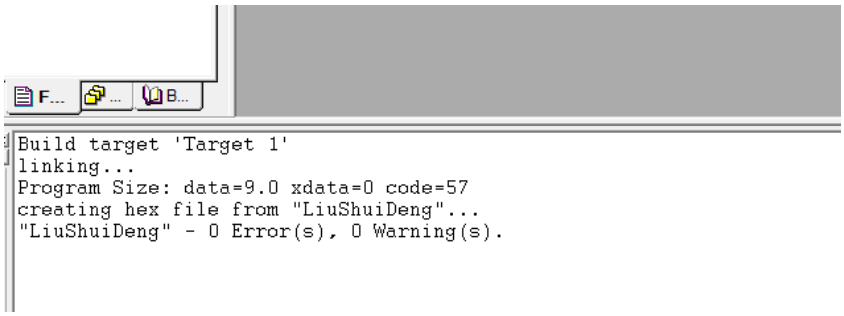

Figure 4: Keil uVision2 interface compiled to generate HEX file

At this point, the whole project is established, edited, compiled.

3. Conclusion

Compared with the establishment of a single source file project, the establishment of multiple source file projects is to modularize the functions and hardware used in the whole project into independent source files and header files, and use header files to declare the function files called, global static variables and hardware port register addresses. Use the main function to call other source program files and other source program files mutual call to complete the function of the whole project. This multi file project establishment mode can greatly simplify the content of the source file where the main function is located and other source files, facilitate the file migration, and be more conducive to the deepening and application of embedded learning in the future.

References

[1] Chinese government website http://www.gov.cn/hudong/2016-12/09/content_5145963.htm
[2] Zhou Jinzhi, Yang Ming, Tong Haiyan, Tang Shuping, Yu Wenjing, Research on the Intelligent Teaching Reform of C Language Programming [J]., Journal of Liupanshui Normal University, 2022, 34 (2): 88-96
[3] Bian Qian, Wang Zhenduo. Research on Flipped Classroom Teaching Model of "C Programming Language" Course based on MOOC [J]. Microcomputer Applications, 2018, 34 (03): 35-37
[4] Zhang Zhihui. Research on the Design Architecture of C Language Embedded System Programming Software [J]. SCM and Embedded System Applications, 2018 (1): 3-5, 10.