Foreword

RT-Thread was created in 2006, and now more ten years has passed. Several years ago I knew Mr. Fire on the internet, I remember clearly at that moment Fire was not so popular as now(he just started making developing boards) and RT-Thread was a niche RTOS(Real Time Operation System).I haven’t meet Fire until the middle of 2018 when Fire came to the city of Shanghai, we said “**You are so young that looks unlike a strength pie.**” to each other.

In the end of 2017 when I heard Fire was planning to write a book about RT-Thread, I was quite exciting and full of appreciation. It would be the first book about RT-Thread and would play an important role in the spread of RT-Thread’s basic ecological chain. Moreover, as a RT-Thread’s official partner, Fire has planted a host of demos to all of his STM32 developing boards and a hand by hand course was provided to each demo.

There are two parts of this book written by Fire. The first part tells you how to construct RT-Thread from 0 to 1, from assemble language step by step to a RT-Thread kernel. This part unveiled how to define tasks, how to switch between tasks, and also described how to implement a delay in task, how to support multi priority, how to implement software timers and how to implement round-robin scheduling and the like. All of these are the basic and core knowledge of RT-Thread OS. The second part showed how to develop applications based on RT-Thread, and this will make readers do not have difficulty when learning and using RT-Thread.

The content of this book goes from shallow to deep, from easy to difficult step by step. It is exactly same as the beginners’ path, and there’s no better book than this one to learn RT-Thread. In addition, the whole book also takes into account the depth of the operating system, and are also worth to reading for readers who want to know the theory of RTOS kernel.

Father of RT-Thread 

Both the electronic version of this book and demo codes can be downloaded at the following Wechat official accounts.

Fire electronic RT-Thread Home of Chips



Ch1 Preface

1.1 About This Book

This book is the RT-Thread version of the wildfire embedded teaching series "RTOS Kernel Implementation and Application Development Practice Guide". *The FreeRTOS* Kernel *Implementation and Application Development Practice Guide Based on STM32* published by the mechanical industry press is the companion volume of this book. Both books are written for RTOS, but they are aimed at different RTOS objects, so there will be many similarities in the theoretical part of the book that involves the abstract layer of RTOS. The specific differences are reflected in the implementation principle and code level of these two RTOS.

This book is divided into two parts. The first part is "*Teach you to write RT-Thread kernel from 0 to 1*". The chapter name of this part is basically similar to the chapter name of its companion volume "*FreeRTOS Kernel Implementation and application development practice guide - based on STM32*". It arranges the chapter number and name in a step-by-step and layer by layer manner, but the specific content is different because they are two different RTOS. The two chapters "*New RT-Thread Project Software Simulation*" and "*Bare Machine System and Multitask System*" belong to the abstract layer of theory, they are not specific to any RTOS, so they are basically the same.

The second part of this book is "RT-Thread Kernel Application Development", which mainly explains the source code implementation of RT-Thread kernel components and the use of API. It is also similar to its companion volume "*FreeRTOS Kernel Implementation and application development practice guide based on STM32*" in chapter naming and sorting. The theoretical explanation of the common abstract layer of RTOS, such as what is task, blocking delay and semaphore application are basically similar, but when it comes to the principle implementation and code explanation of these two RTOS, it is totally different, and the knowledge of this part is the content that readers should really care about.

I classified these two books as companion volume because the writing style, content framework, chapter naming and sorting of the two books are basically the same, the language description is similar, and the theoretical part related to the RTOS abstraction layer is the same, but the implementation principle, kernel source code explanation and user APIs use of RTOS are different. These contents are the book's key part and the core of readers' concern and learning. The two books can't be combined into one due to these differences, so they are divided into two books.

If you have studied one of the books, and now study another one. The theoretical part related to the RTOS abstraction layer can be skipped, just focus on the implementation of RTOS kernel and the application of source APIs. The existing RTOSs are basically same at the theoretical level, but have their own characteristics in the specific code implementation, so the two books can be used as complementary learning. As long as you mastered the knowledge of one book and then learn the other one, it will be a piece of cake for you and seems God is helping you.

1.2 How to Learn This Book

This book is the first English book on RT-Thread system, which is divided into two parts. The first part "from 0 to 1, teach you to write RT-Thread kernel", that is, focus on the principle of RT-Thread’s implementation, start from 0, iterate continuously, teach you how to write RT-Thread kernel, let you thoroughly learn how to define thread, how to schedule the system (including the assemble language codes’ explanation), how to implement multi priority and so on. All of these are the deepest knowledge of operation system. When you get this book and start to learn, you will be surprised that learning RTOS is not so complicated, but rather so interesting. It turns out that you can write RTOS as well, and your sense of achievement will immediately bursts.

When you have mastered the knowledge of the first part thoroughly, it is easy to learn other RTOS. Looking at several popular RTOSs on the market, the implementation of their kernel is basically similar, only one of them needs to be studied in depth and it is unnecessary to learn all of them deeply. If you have time, it doesn't matter. The second part is "*RT-Thread Kernel Application Development*", focusing on RT-Thread porting, application of each component of the kernel, which is easier to master than the first part.

The content of the whole book is gradual and iterative. The former chapter is the basis of the latter chapter, which must be read from the beginning instead of jumping. When learning, the two things are must to be done: first, we should combine code with books to learn rather than read books blindly, Reading and debugging at the same time. Reading is very simple, but how to debug the code? That is, to execute every program step by step, to see whether the execution process and effect of the program are the same as what you designed in your brain; secondly, after finished each chapter, you must rewrite the demo codes (typing letter by letter, you can copy but should not paste even a semicolon), to make sure you have really understood. When you write by yourself, you will surely make a hundred mistakes. At this time, you should cherish these mistakes and debug them until they are addressed. This is the best opportunity for you to improve your programming ability. Remember that the program is not written in one time, but debug step by step.

1.3 Reference

1.RT-Thread official source code

2.*RT-Thread\_manual.zh*.pdf (electronic version)

3. *STM32 KuKaiFaShiZhanZhiNan* (electronic version)

1.4 Writing Style of This Book

The first part of this book is based on the official source code of RT-Thread nano 3.0.3. It is a continuous iteration to teach you how to write the RT-Thread kernel from 0. The data types, variable names, function names, file names, and file storage locations involved in the book are all implemented in the official way of RT-Thread. After you finished the book, you can seamlessly switch to make use of the raw RT-Thread. It should be noted that in the process of implementation, some parameters of functions and redundant codes are removed, and only the core functions are retained, but this will not affect learning.

The second part of this book mainly talks about the porting of RT-Thread and the using of kernel components. The source code will not be explained in depth but focus on application. If you are not interested in the first part, you can skip the first part and directly enter the second part. There is no inevitable relationship between them.

1.5 Supporting Hardware of This Book

This book supports all serial of STM32 development boards, the detail model is listed in table 1-1, and their photos are Fig1-1 to Fig1-9. There are three modes in Fire Challenger (TiaoZhanZhe)serial named F429, F767 and H743. They have common base board but different core boards. When learning, if we do experiments with these hardware platforms, we will get twice the result with half the effort. All kinds of problems will be avoided when porting BSP from one hardware to another. Actually, the process of porting is also the process of learning although it is painful.

Table 1-1 All the STM32 development boards of Fire

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Difference | | | |
| - | core | Pin Number | Ram size | Rom Size |
| MINI | Cortex-M3 | 64 | 48KB | 256KB |
| ZhiNanZhe | Cortex-M3 | 100 | 64KB | 512KB |
| BaDao | Cortex-M3 | 144 | 64KB | 512KB |
| BaTianHu | Cortex-M4 | 144 | 192KB | 1MB |
| TiaoZhanZheF429 | Cortex-M4 | 176 | 256KB | 1MB |
| TiaoZhanZheF767 | Cortex-M7 | 176 | 512KB | 1MB |
| TiaoZhanZheH7 | Cortex-M7 | 176 | 1MB | 2MB |

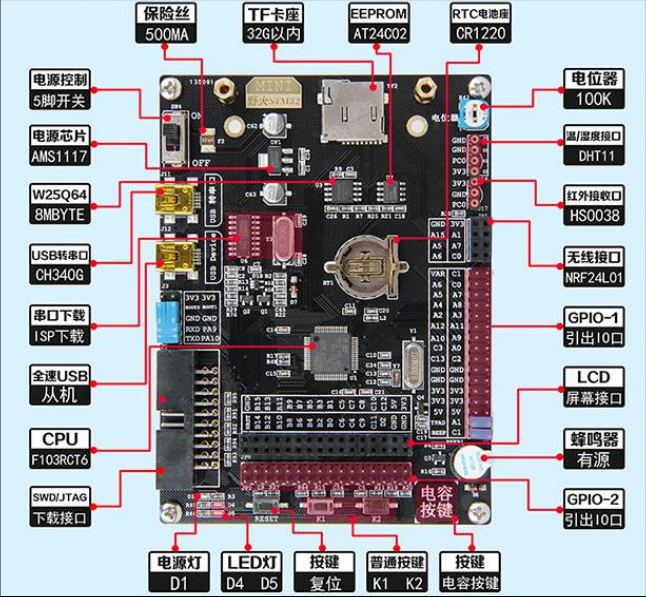


Fig 1-1 Fire[MINI] STM32F103RCT6 development board



Fig 1-2 Fire[MINI] STM32F103RCT6 development board with LCD

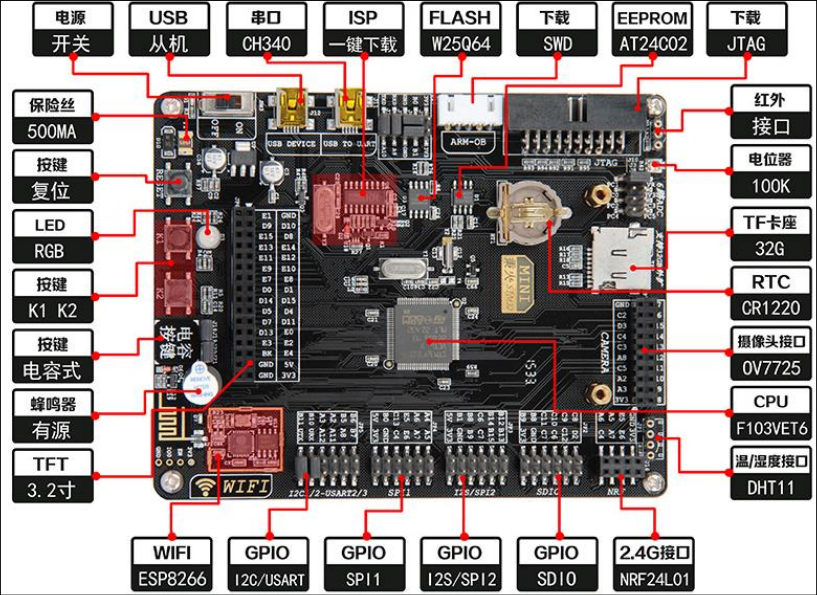


Fig 1-3 Fire[ZhiNanZhe] STM32F103VET6 development board



Fig 1-4 Fire[ZhiNanZhe] STM32F103VET6 development board with LCD

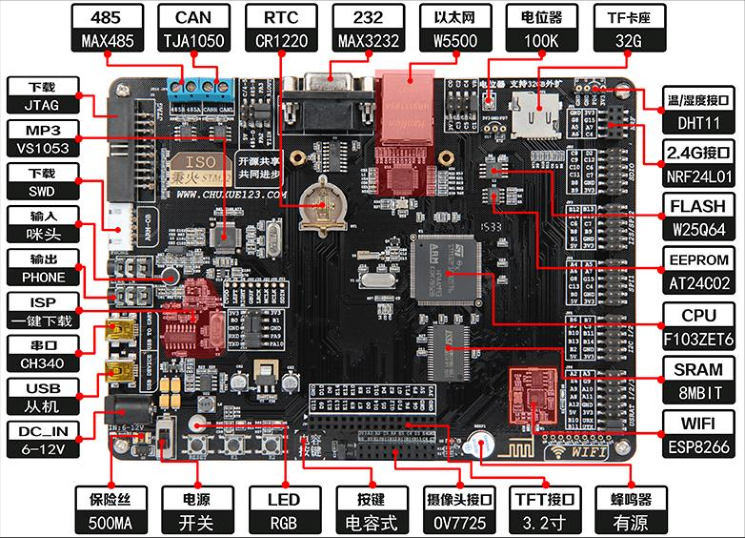


Fig 1-5 Fire[BaDao] STM32F103ZET6 development board



Fig 1-6 Fire[BaDao] STM32F103ZET6 development board with LCD



Fig 1-7 Fire[BaTianHu] STM32F407ZGT6 development board



Fig 1-8 Fire[BaTianHu] STM32F407ZGT6 development board with LCD

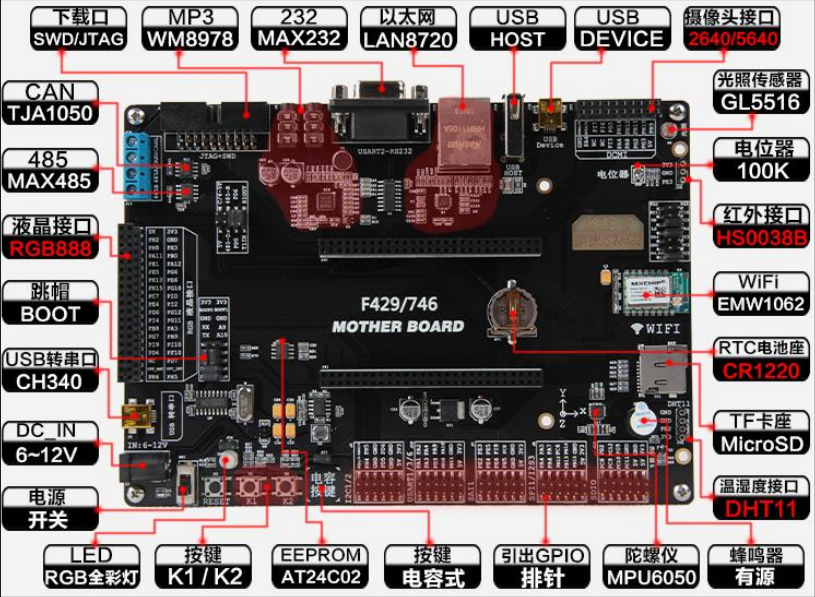


Fig 1-9 Fire[TiaoZhanZhe] F429/F767/F743development board



Fig 1-10 Fire[TiaoZhanZhe] F429/F767/F743development board with LCD

1.6 Technology Forum (Developer Community)

If you encounter problems in the learning process, you can go to wildfire electronic forum: www.firebbs.cn to post and exchange. Open source codes, share, and make progress together.

Considering the limited level, this book is bound to be flawed. Enthusiastic readers can also send the survey to the forum by mistake so that we can make better improvements. I wish you a happy study experience. In the RT-Thread world, wildfire will work together with you.

Ch2 How to Learn RTOS

2.1 Why Learn RTOS

When step into embedded field, the very first course we learn is MCU programming at most time, and in MCU programming, 51-core MCU is chosen as the first chip. The MCU programming we talking about here is bare machine programming, which means developing a program without any RTOS. The frequently used RTOS are μC/OS、RTX, RT-Thread、Huawei LiteOS and AliOS-Things. FreeRTOS got the highest market share due to its open source and free to use in commercial and non-commercial, but now Chinese RTOS RT-Thread has developed more than one decade, its market share is booming, and has become the NO.1 in RTOSs which are made in China.

In bare-machine system, almost all codes are coded by ourselves and all operations are executing in an infinite big loop. Many medium to small sized electronical products in our life are bare-machine system and they meet our need well. So why do we need learning programing with RTOS, bare-machine is good why a RTOS is to be added stubbornly? First reason is projects demand, as the number of functions of a given product is growing, bare-machine system can hardly address our problem perfectly, instead it may make our code more complicate. After introducing RTOS, we can use RTOS to implement multi-thread manger to decrease the system complication, and this is the biggest edge in using RTOS. Second reason is learning demand. To implement a better career planning, to make preparation for go to peek of life that marry a girl who is white, rich and beautiful, we have to learning something more advance rather than persist in bare-machine programming. As a qualified embedded engineer study should not be stopped forever, and we should prepare for our future at any moments. As a Chinese proverb says: when using knowledge then find books you have read is not enough. I wish when opportunity comes you will not have this kind of feeling.

To help readers know how to program with RTOS, we will analysis the difference between bare-machine system and RTOS system in the chapter *Bare-machine System and Multi Thread System*. The difference is called as MingMen(means key) of learning RTOS. It will make you easy to learn RTOS. We will mainly talk about method and theory when describing the two programming ways, and will not involve in concrete codes but pseudo code.

2.2 How to learn RTOS

There is some difference between bare-machine programing and RTOS programming and many people say it’s difficult to learn RTOS, so someone will feel horrible when hearing RTOS programing, and then give it up.

So how to learn a RTOS? The easiest way is conferencing the RTOS API and calling these APIs to implement the functions you want on the system others has ported. In this way we care little about how to plant a RTOS, so it is the easiest and fastest way to get started RTOS. There are both advantages and disadvantages using this way. If you are developing a product, the benefit is you can implement functions in short time and put on the market, then win the opportunity, but when meeting bugs, and you are not knowing the RTOS enough, it will makes you difficult to debug, there nothing you can do. This is its disadvantage. It is not recommended only call APIs simply in learning RTOS. We should learn one RTOS deeply.

The kernel of all of the RTOSs in market have the similar implement, so we learn one of them is enough. And it is not so difficult to use anther RTOS after you learned one deeply. How to study a RTOS in depth? Here is the most efficient but most difficult way that reading the source codes of RTOS, researching how every component implemented. The progress is dull and painful, but it will bring you great harvest. To learn the essential of RTOS, if you do not go to the hell, who will?

There are some books in the market explained the source codes of RTOS, but if you are lack of basic knowledge and haven’t the RTOS before then reading its source codes is still a dull job, and you cannot master the structure and realization of the RTOS.

Fortunately, now we take a new way to teach you one RTOS. Neither explain plainly its APIs nor analysis its source codes sentence by sentence. What we do is writing the RTOS from nothing to everything, starting from 0, step by step, improving continuous. You will feel the joy of success in every learning stage. In the process of implement RTOS, the unique knowledge you should have is basic C language skill. Following this course written by Fire, achievement is on the horizon.

2.3 Which RTOS Should be Chosen

The RTOS used to teaching, because we will not develop a RTOS by ourselves and not create wheels again, is chosen as RT-Thread which is the most popular one in China and developed by Chinese engineer. Based on RT-thread, we rewrite it from 0 to 1 step by step. During the process, we will comply with the coding styles in RT-Thread, including data type, variable name, function name, file type etc., and we will not rename any functions and variables. So that after finished this course, you can make use of RT-Thread directly without any gaps.

CH3 First acquaintance with RT-Thread

3.1 Brief introduction to RT-Thread

3.1.1 Copyright

The copyright of RT-Thread is belonging to Shanghai RT-Thread Technology Co., Ltd. The first version was released in January 2006 and its version number is 0.1.0. after more than 10 years of development, the main version number has upgraded to 4.0, The total number of developers has reached millions, and the number of devices running RT-Thread in the various of life has exceeded 20 million. RT-Thread has been the highest market share RTOS in all RTOSs made by Chinese.

3.1.2 Charge policy

RT-Thread complying with GPLv2+ opensource license is an open source and free to use RTOS. The open source we talking about here means that you can get RT-Thread source code free of charge. When RT-Thread is used in your product and you haven’t modified the RT thread kernel source code, all the code of your product can be closed source and doesn’t need to open source. But when you modified the RT thread kernel source code, you must open source the modified part and feed them back to the community. But your application code doesn’t need to open source. Free means that no matter you are an individual or a company, you can use it for free without paying a cent.

3.2 Significance of RT-Thread

I don't know whether you have found that in the field of RTOS, the real-time operations we can access are basically not from China, which is rare to have Chinese manufacturers in the RTOS field. From the hottest μC/OS in the early years to the FreeRTOS which has the highest market share today, to the RTX with the most security verification (Keil's own), to the ThreadX with the most profitability, they are all not from China. But a little over 10 years ago, in China, there was a talented, stubborn geek named Xiong PuXiang, who has compiled the first generation of RT thread kernel, and worked with the geeks in China's open source community to continuously improve and innovate. After more than 10 years of development, today, has become the leader of Chinese RTOS, hundreds of thousands of developers are growing every year. In addition, AI, the Internet of things and other outlets make RT-Thread has the potential of unifying the Jianghu(means RTOS market). From we have completed a round of multi-million dollar financing this year, we can see that in the next five years, RT thread will be the best choice for you to learn and make products.

What is the meaning of RT thread? RT thread is from China, which makes us see that Chinese technology developers can write such excellent RTOS as well, not everything is good for foreign countries. Based on our experience of more than 10 years in electronic industry, we met many Chinese developers in the past, when they saw something great, they would ask whether that was foreign or Chinese. If that was Chinese, they would ask whether that has referred to foreign countries. Many people are born with this kind of chondropathy, which is very difficult to treat. It is a natural feeling. Now RT thread is a good medicine for this kind of osteomalacia, which is the biggest significance of RT-Thread in my opinion. Of course, RT-Thread, as a Chinese Internet of things operating system, is simple, easy to use, low power design, rich components and other features will also make RT-Thread shine. Wildfire, as a Chinese embedded education brand, is also our honor to contribute to the Chinese RTOS. I hope this book can help you quickly get started and master RT-Thread.

Part one Write RT-Thread kernel from 0 to 1

The first part of this book is based on RT thread nano. It teaches you how to write RT thread from 0 step by step. This part focuses on the process of how to implement RTOS. When you finish this part, you will feel easy to use RT-Thread or other RTOS, not only knowing what it is, but also knowing why. In the process of source code implementation, the data type, variable name, function name, file name and the directory where the files are stored will all be implemented according to RT-Thread. Some unnecessary codes may be removed, but it will not affect you to understand the whole OS function.