/\*

操作系统：Ubuntu16.04

硬件平台：原子Stm32F767+7‘RGB屏幕

其他操作系统与开发板搭建环境基本差不多，注意的地方我会提到的。

工程[Github](https://github.com/near2see/stm32f767)

\*/

### 一、软件安装

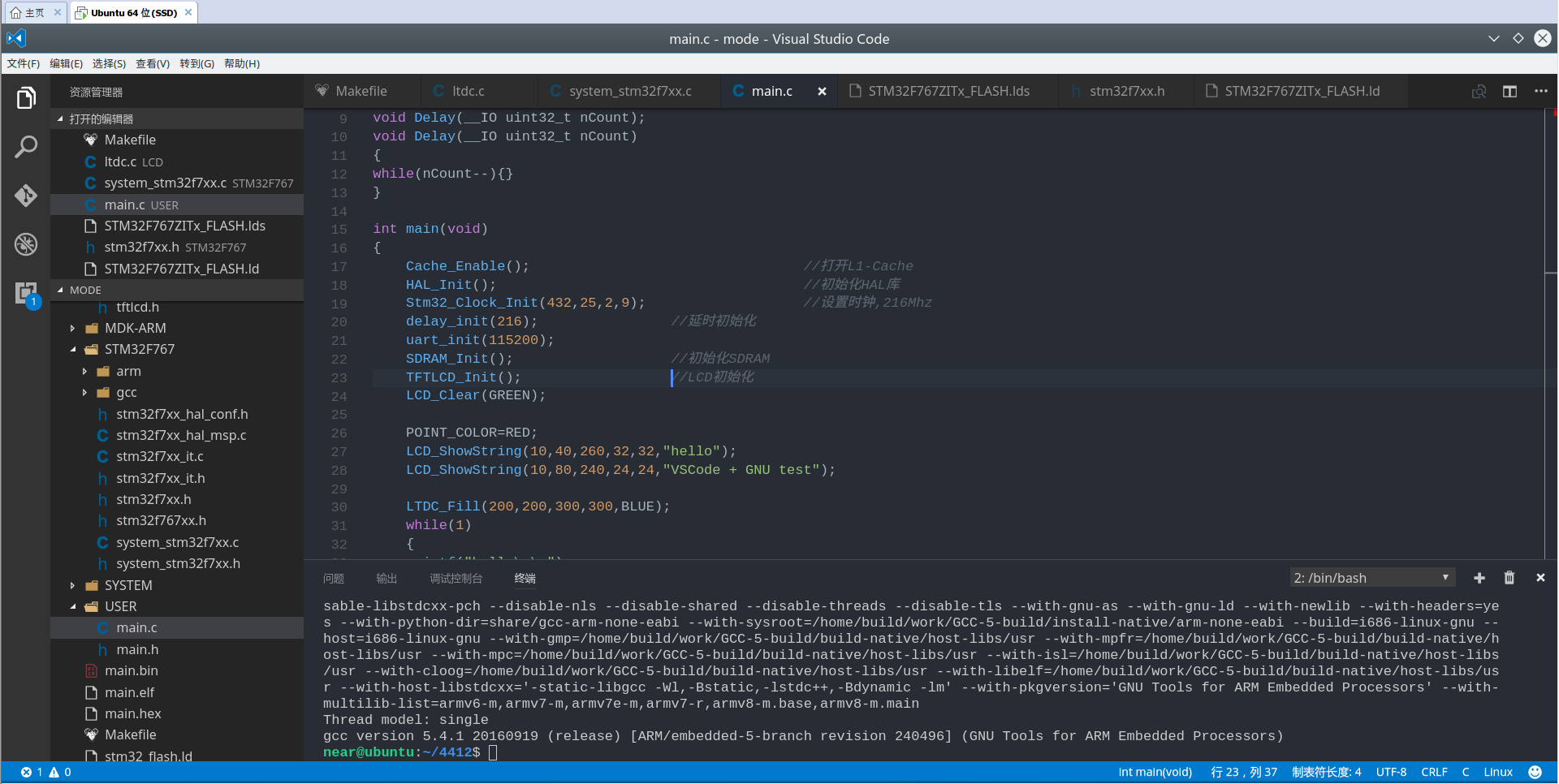
* **VSCode**

为什么用VSCode，相信大家都知道，没用过的同学下载下来用用。（推荐）

这个软件不管是什么系统都有，安装比较简单，不再赘述。

详情和下载请点击：<https://code.visualstudio.com/>

贴一张VSCode的图



资源管理器+和vs studio一样的代码高亮+集成终端

* **Arm-none-eabi-gcc**

下载链接：<https://launchpad.net/gcc-arm-embedded/+download>

Windows下安装傻瓜式的就不说了。

下面说下Ubuntu下安装方法：

1. 解压**gcc-arm-none-eabi-5\_4-2016q3-20160926-linux.tar.bz2**

|  |
| --- |
| $ tar -xvf gcc-arm-none-eabi-5\_4-2016q3-20160926-linux.tar.bz2 |

1. **将解压得得文件夹复制到/usr/bin**

|  |
| --- |
| $ sudo cp -rgcc-arm-none-eabi-5\_4-2016q3 /usr/bin |

1. 添加环境变量

|  |
| --- |
| $ vim /etc/profile  添加：export PATH=/usr/bin/gcc-arm-none-eabi-5\_4-2016q3/bin:$PATH |

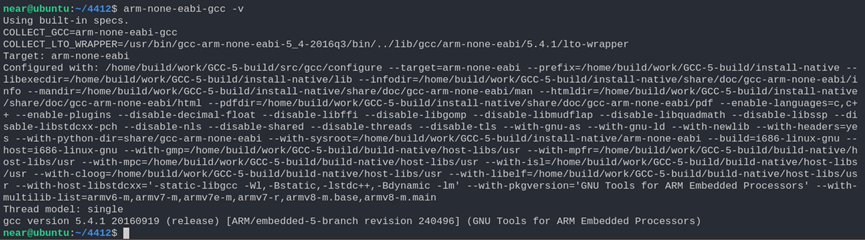
1. 使环境变量生效

|  |
| --- |
| $ source /etc/profile  $ reboot |

5>测试

|  |
| --- |
| $ arm-none-eabi-gcc -v |

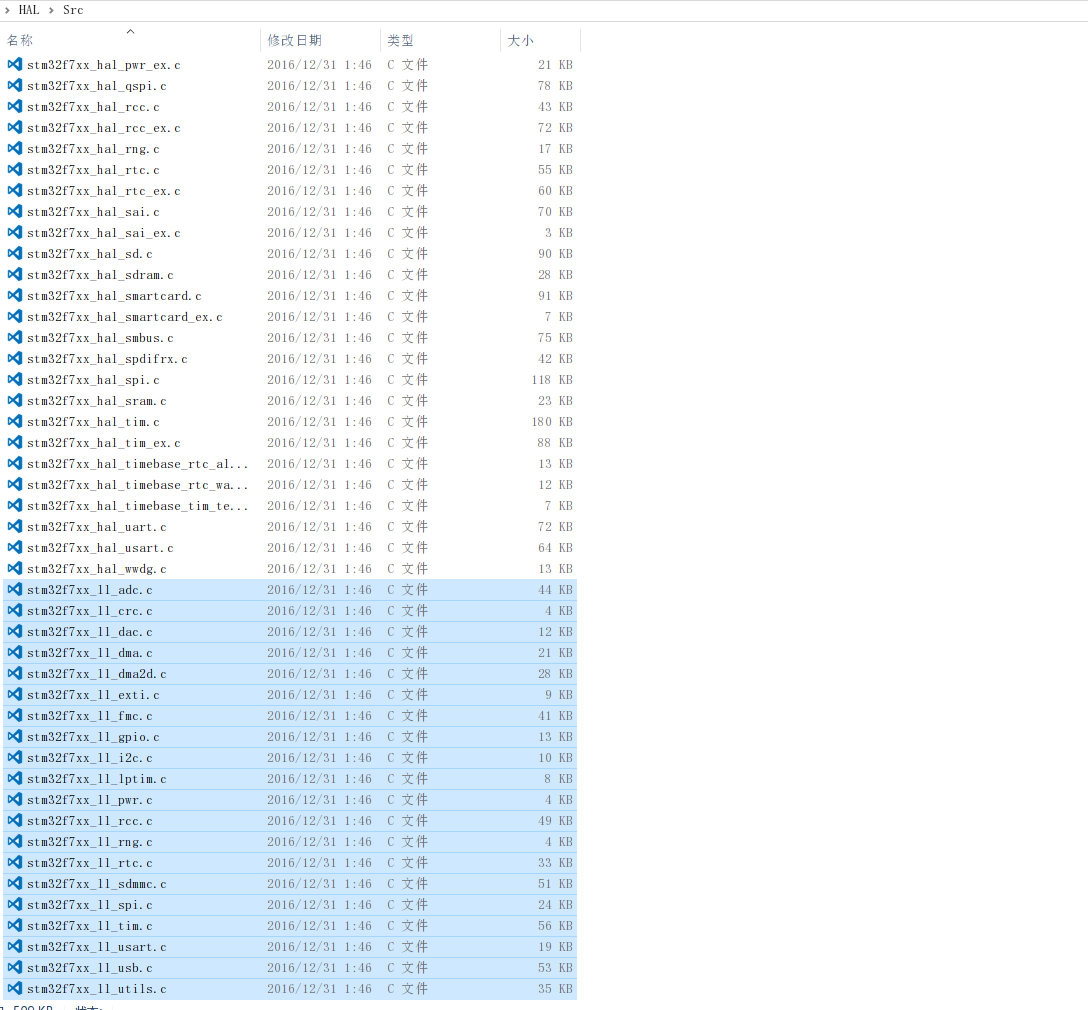
出现如下信息说明成功

****

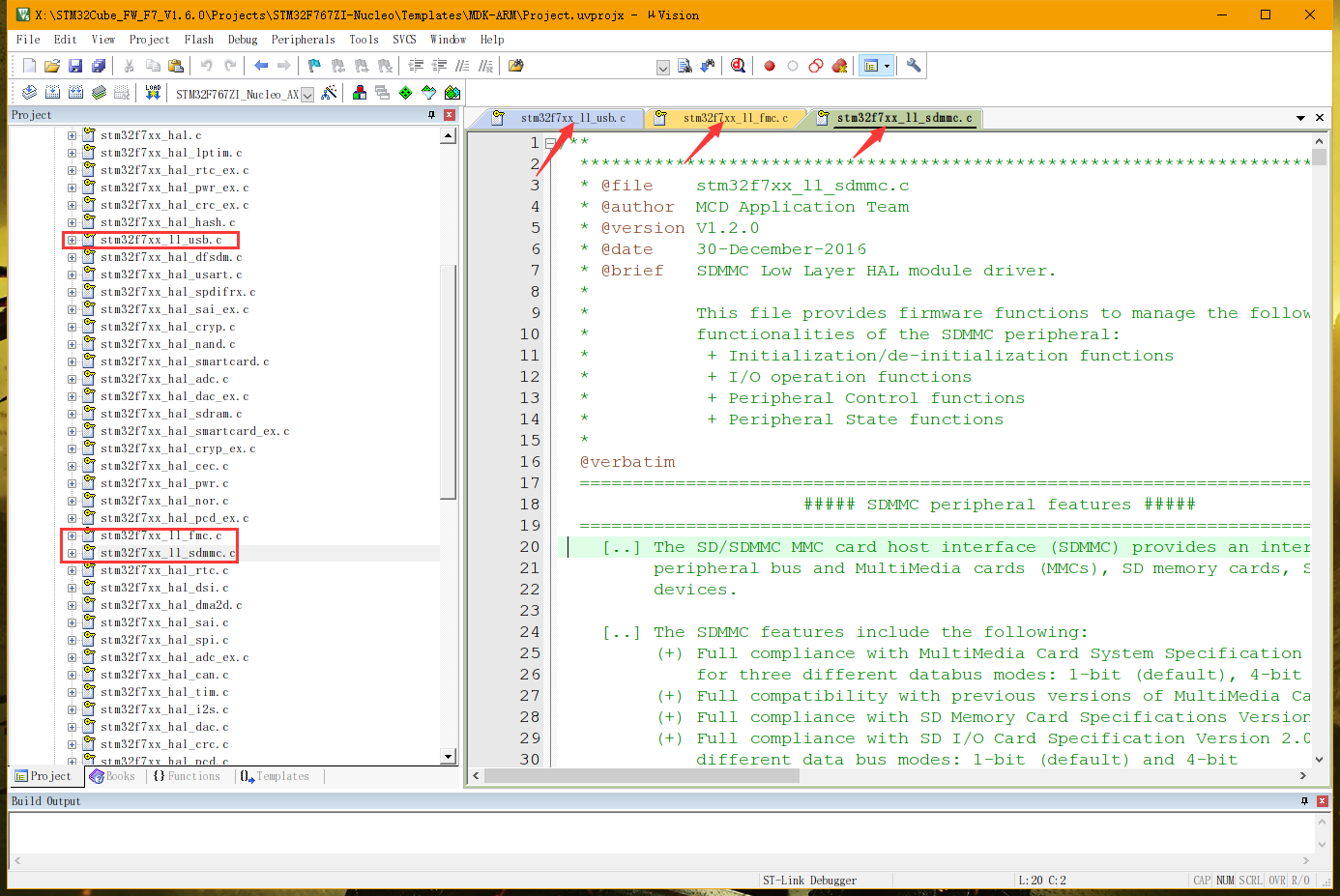
### 二、建立工程

就拿我的F7来说，我用的是最新的V1.6的HAL库

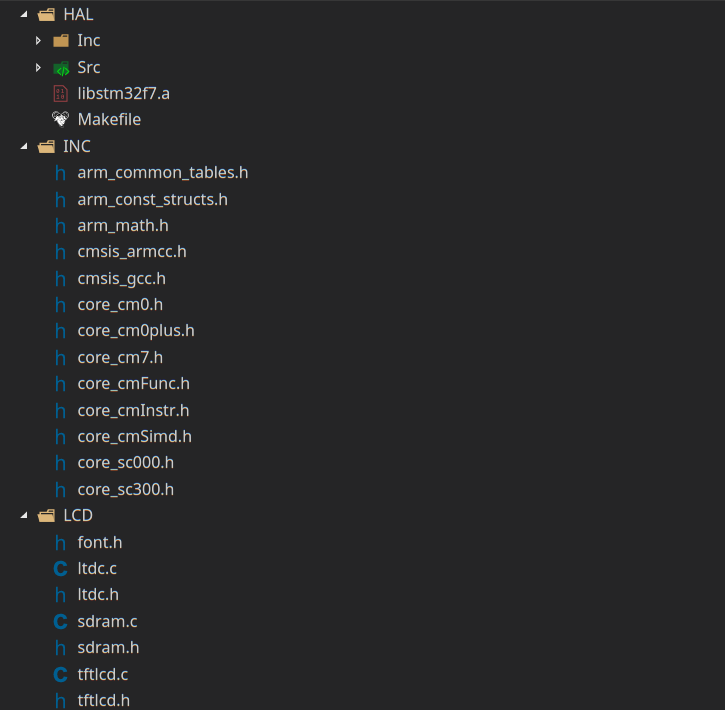
大体跟之前差不多，添加了些LL库的文件

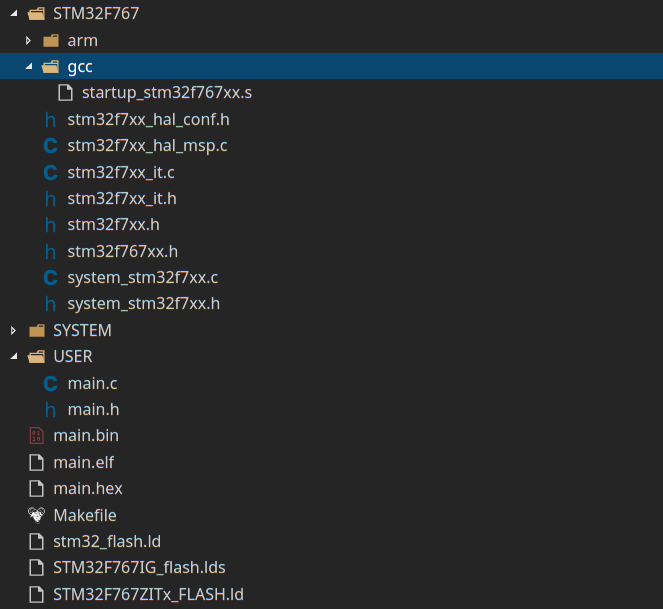


起始呢，这么多LL库的文件并不是所有都用得到，参考官方的模板例程，也就使用了3个如图：



参考官方模板我们建立自己的工程结构





大体是这样的一个结构，大家可以根据自己的想法构建。

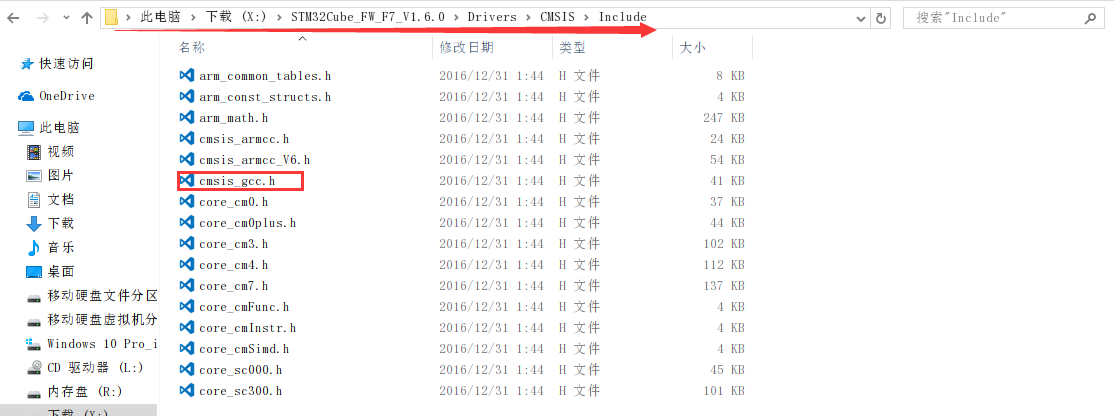
要注意的就是一下几点

启动文件：用GCC编译所需的启动文件跟再MDK编译的启动文件是不同的具体在Cube库的这里



里面就是常用开发环境用的启动文件。

然后就是就是一些CMSIS标准所需的头文件



根据自己板子的型号可以删除一些不必要的文件。

其中cmsis\_gcc.h是用gcc编译所需的头文件。

工程目录到这一步就差不多了。

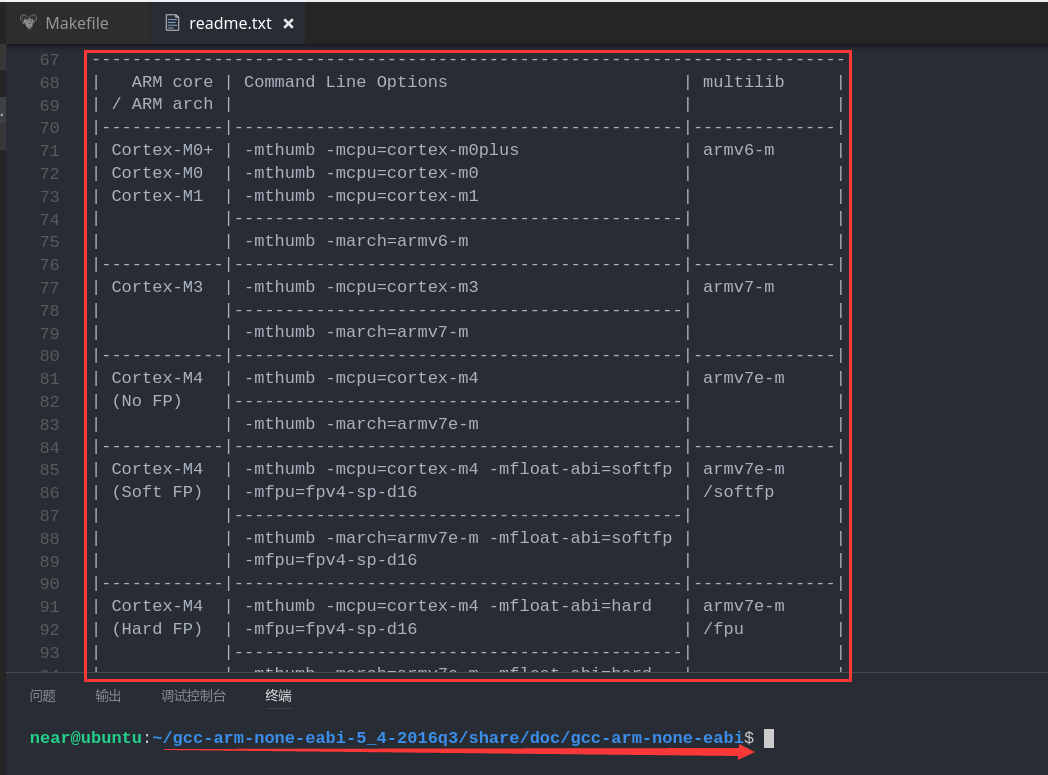
### 三、编写Makfile

首先在HAL库文件下新建一个Makfile，为的就是将HAL库编译成一个静态的lib.

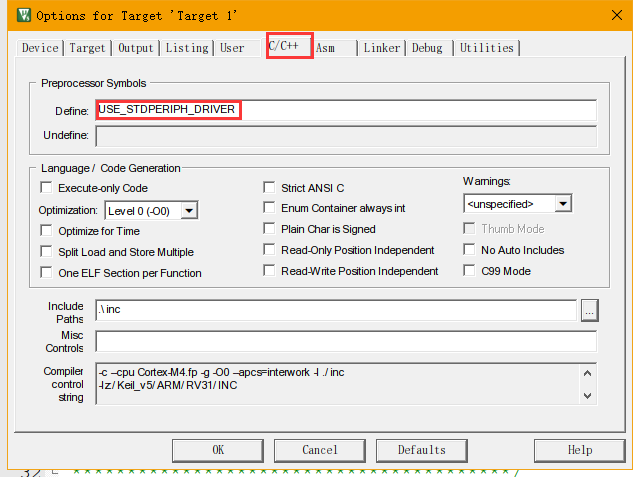
代码如下：

|  |
| --- |
| CC=arm-none-eabi-gcc  AR=arm-none-eabi-ar  ###########################################  vpath %.c Src  CFLAGS = -g -O2 -Wall  CFLAGS += -mthumb -mcpu=cortex-m7 -mfloat-abi=hard -mfpu=fpv5-d16  CFLAGS += -mlittle-endian -mthumb-interwork  CFLAGS += -ffreestanding -nostdlib  CFLAGS += -IInc -I../INC -IInc/Legacy -I../STM32F767  CFLAGS += -DSTM32F767xx -DUSE\_HAL\_DRIVER  SRCS = $(wildcard Src/\*.c)  OBJS = $(SRCS:.c=.o)  .PHONY: libstm32f7.a  all: libstm32f7.a  %.o : %.c  $(CC) $^ $(CFLAGS) -c -o $@  libstm32f7.a: $(OBJS)  $(AR) -r $@ $(OBJS)  clean:  rm -f $(OBJS) libstm32f7.a |

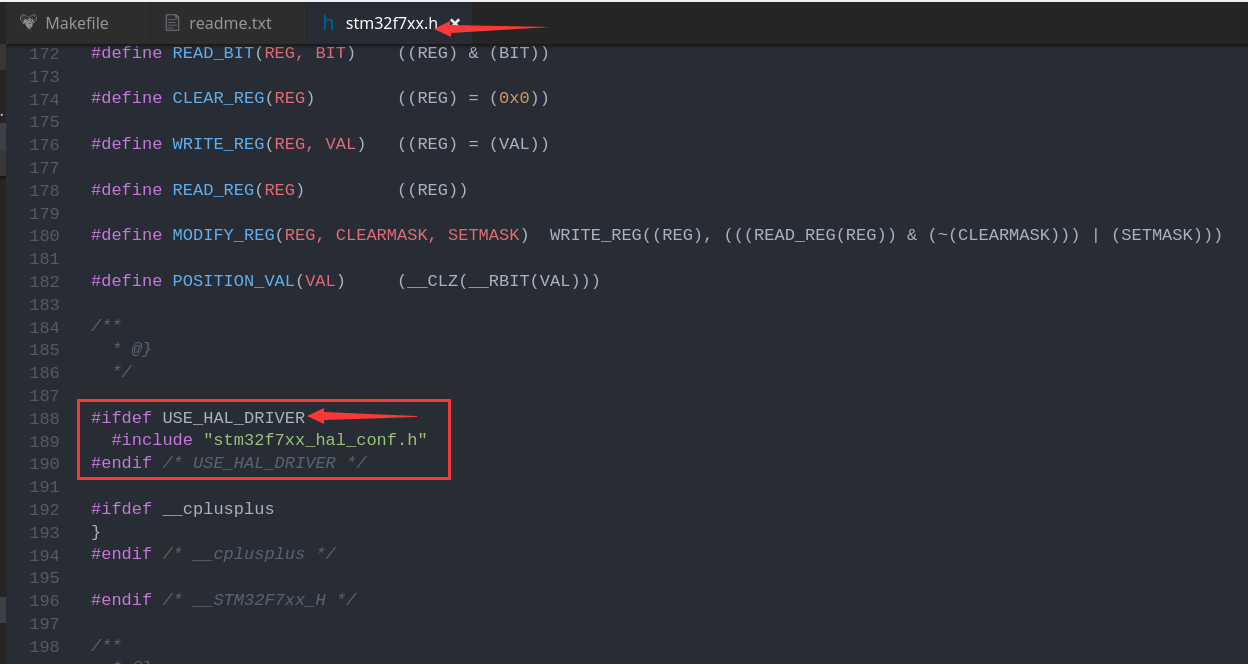
这个Makfile相对简单，值得注意的是CFLAGS中的几个编译选项，device有关的可以查看图中的readme.txt

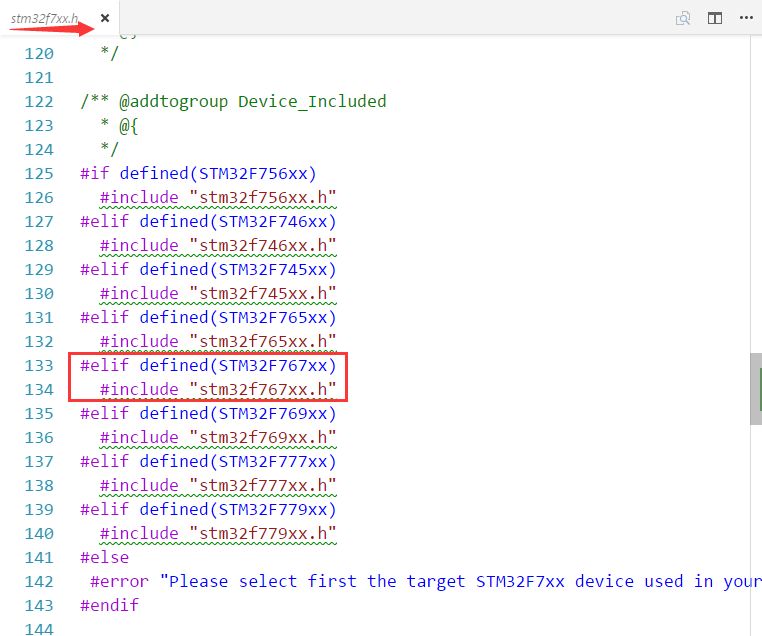


其中一个预处理选项即CFLAGS += -DSTM32F767xx -DUSE\_HAL\_DRIVER -D选项表示预定义，为什么添加这两个？



很熟悉不是？其实这些最根源的是由于





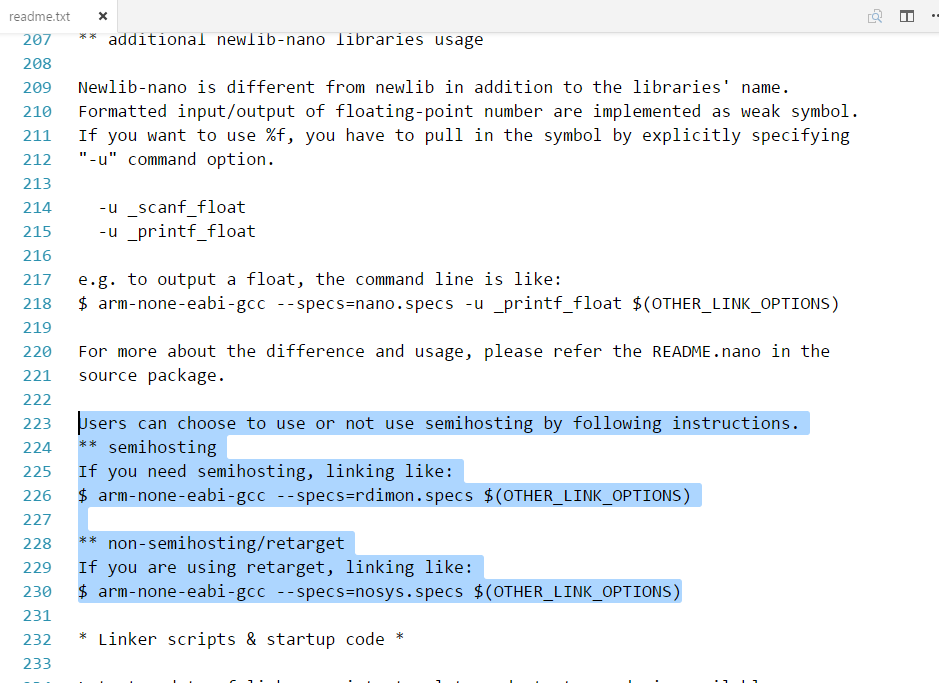
然后再工程根目录新建一个顶层的Makefile。代码如下：

|  |
| --- |
| PROJ\_NAME = main  CC=arm-none-eabi-gcc  OBJCOPY=arm-none-eabi-objcopy  CFLAGS = -g -O2 -Wall -TSTM32F767ZITx\_FLASH.ld  CFLAGS += -mlittle-endian -mthumb -mcpu=cortex-m7 -mthumb-interwork  CFLAGS += -mfloat-abi=hard -mfpu=fpv5-d16  CFLAGS += --specs=nosys.specs  LFLAGS += --specs=nosys.specs  SRCS = stm32f7xx\_it.c system\_stm32f7xx.c stm32f7xx\_hal\_msp.c sys.c usart.c delay.c ltdc.c sdram.c tftlcd.c main.c  vpath %.c USER SYSTEM/usart SYSTEM/delay SYSTEM/sys LCD STM32F767  vpath %.a HAL  ROOT=$(shell pwd)  CFLAGS += -ILCD -ISYSTEM/sys -ISYSTEM/usart -ISYSTEM/delay -IUSER  CFLAGS += -IHAL/Inc -IHAL/Inc/Legacy -IINC -ISTM32F767  CFLAGS += -DSTM32F767xx  SRCS += STM32F767/gcc/startup\_stm32f767xx.s  OBJS = $(SRCS:.c=.o)  .PHONY: lib proj  all: lib proj  lib:  $(MAKE) -C HAL  proj: $(PROJ\_NAME).elf  $(PROJ\_NAME).elf: $(SRCS)  $(CC) $(CFLAGS) $^ -o $@ -LHAL -lstm32f7  $(OBJCOPY) -O ihex -S $(PROJ\_NAME).elf $(PROJ\_NAME).hex  $(OBJCOPY) -O binary $(PROJ\_NAME).elf $(PROJ\_NAME).bin  clean:  $(MAKE) -C HAL clean  rm -f $(PROJ\_NAME).elf  rm -f $(PROJ\_NAME).hex  rm -f $(PROJ\_NAME).bin |

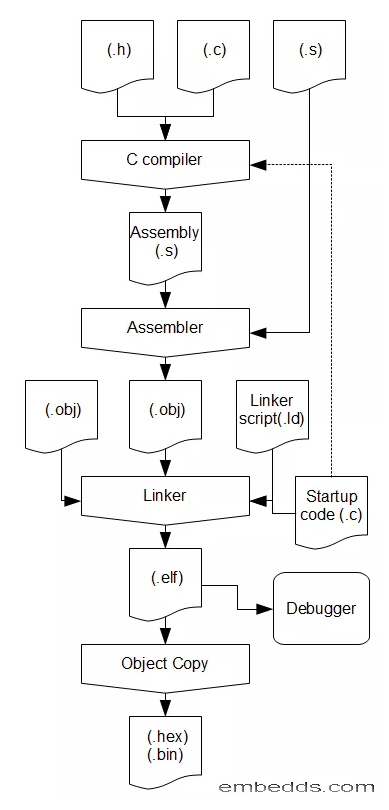
这里需要注意的几点是 -TSTM32F767ZITx\_FLASH.ld 这是来加载linkerscript文件的。

后面会说到Linkerscript文件怎么写。

还有就是关于这个选项--specs=nosys.specs，它是关于使用重定向要用的选项具体还是看之前那个readme.txt，在[Github](https://github.com/near2see/stm32f767)中的GNU\_DOC目录中



到这儿Makefile差不多介绍完了，Makefile就是起统筹的作用。如图所示：



### 四、Linkerscript介绍

代码如下：

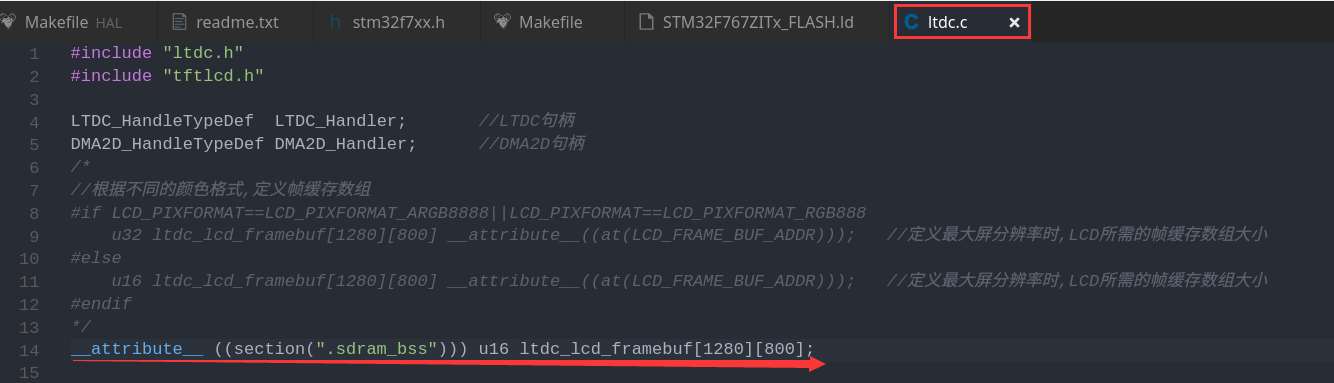
|  |
| --- |
| /\* Generated by LinkerScriptGenerator [http://visualgdb.com/tools/LinkerScriptGenerator]  \* Target: STM32F767IG  \* The file is provided under the BSD license.  \*/  ENTRY(Reset\_Handler)  MEMORY  {  FLASH (RX) : ORIGIN = 0x08000000, LENGTH = 1M  SRAM (RWX) : ORIGIN = 0x20000000, LENGTH = 512K  ITCMRAM (RWX) : ORIGIN = 0x00000000, LENGTH = 16K  /\* --- begin generated external memories -- \*/  SDRAM (RWX) : ORIGIN = 0xc0000000, LENGTH = 32M  /\* --- end generated external memories -- \*/  }  \_estack = 0x20080000;  SECTIONS  {  .isr\_vector :  {  . = ALIGN(4);  KEEP(\*(.isr\_vector))  . = ALIGN(4);  } > FLASH  .text :  {  . = ALIGN(4);  \_stext = .;  \*(.text)  \*(.text\*)  \*(.rodata)  \*(.rodata\*)  \*(.glue\_7)  \*(.glue\_7t)  KEEP(\*(.init))  KEEP(\*(.fini))  . = ALIGN(4);  \_etext = .;  } > FLASH  .ARM.extab :  {  . = ALIGN(4);  \*(.ARM.extab)  \*(.gnu.linkonce.armextab.\*)  . = ALIGN(4);  } > FLASH  .exidx :  {  . = ALIGN(4);  PROVIDE(\_\_exidx\_start = .);  \*(.ARM.exidx\*)  . = ALIGN(4);  PROVIDE(\_\_exidx\_end = .);  } > FLASH  .ARM.attributes :  {  \*(.ARM.attributes)  } > FLASH  .preinit\_array :  {  PROVIDE(\_\_preinit\_array\_start = .);  KEEP(\*(.preinit\_array\*))  PROVIDE(\_\_preinit\_array\_end = .);  } > FLASH  .init\_array :  {  PROVIDE(\_\_init\_array\_start = .);  KEEP(\*(SORT(.init\_array.\*)))  KEEP(\*(.init\_array\*))  PROVIDE(\_\_init\_array\_end = .);  } > FLASH  .fini\_array :  {  PROVIDE(\_\_fini\_array\_start = .);  KEEP(\*(.fini\_array\*))  KEEP(\*(SORT(.fini\_array.\*)))  PROVIDE(\_\_fini\_array\_end = .);  } > FLASH  . = ALIGN(4);  \_sidata = .;  .data : AT(\_sidata)  {  . = ALIGN(4);  \_sdata = .;  PROVIDE(\_\_data\_start\_\_ = \_sdata);  \*(.data)  \*(.data\*)  . = ALIGN(4);  \_edata = .;  PROVIDE(\_\_data\_end\_\_ = \_edata);  } > SRAM  .bss :  {  . = ALIGN(4);  \_sbss = .;  PROVIDE(\_\_bss\_start\_\_ = \_sbss);  \*(.bss)  \*(.bss\*)  \*(COMMON)  . = ALIGN(4);  \_ebss = .;  PROVIDE(\_\_bss\_end\_\_ = \_ebss);  } > SRAM  /\* --- begin generated external memory sections -- \*/  . = \_sidata + SIZEOF(.data);  . = ALIGN(4);  \_sisdram\_text = .;  .sdram\_text : AT(\_sisdram\_text)  {  . = ALIGN(4);  \_ssdram\_text = .;  PROVIDE(\_\_sdram\_text\_start = \_ssdram\_text);  \*(.sdram\_text)  \*(.sdram\_text\*)  . = ALIGN(4);  \_esdram\_text = .;  PROVIDE(\_\_sdram\_text\_end = \_esdram\_text);  } > SDRAM  . = \_sisdram\_text + SIZEOF(.sdram\_text);  . = ALIGN(4);  \_sisdram\_data = .;  .sdram\_data : AT(\_sisdram\_data)  {  . = ALIGN(4);  \_ssdram\_data = .;  PROVIDE(\_\_sdram\_data\_start = \_ssdram\_data);  \*(.sdram\_data)  \*(.sdram\_data\*)  . = ALIGN(4);  \_esdram\_data = .;  PROVIDE(\_\_sdram\_data\_end = \_esdram\_data);  } > SDRAM  .sdram\_bss (NOLOAD) :  {  . = ALIGN(4);  \_ssdram\_bss = .;  PROVIDE(\_\_sdram\_bss\_start = \_ssdram\_bss);  \*(.sdram\_bss)  \*(.sdram\_bss\*)  . = ALIGN(4);  \_esdram\_bss = .;  PROVIDE(\_\_sdram\_bss\_end = \_esdram\_bss);  } > SDRAM  /\* --- end generated external memory sections -- \*/  PROVIDE(end = .);  .heap (NOLOAD) :  {  . = ALIGN(4);  PROVIDE(\_\_heap\_start\_\_ = .);  KEEP(\*(.heap))  . = ALIGN(4);  PROVIDE(\_\_heap\_end\_\_ = .);  } > SRAM  .reserved\_for\_stack (NOLOAD) :  {  . = ALIGN(4);  PROVIDE(\_\_reserved\_for\_stack\_start\_\_ = .);  KEEP(\*(.reserved\_for\_stack))  . = ALIGN(4);  PROVIDE(\_\_reserved\_for\_stack\_end\_\_ = .);  } > SRAM  } |

关于官方的linkerscript在这里：



我这个Linker script只是在官方的基础添加对SDRAM的支持。

为什么呢，因为要使用RGB屏，



一下子定义了这么大的数组，显然RAM是放不下的，只有放在SDRAM中了，图中被注释的部分是在MDK Keil的用法。

当然在ubuntu下装上st-link驱动或其他的再加上debug server进行调试也是没有问题的，我还没有尝试，先就写到这儿了。