社区工程师专题系列第一期

作者：杨永胜（论坛 id:iysheng）

从事嵌入式 Linux 以及 RTOS 的软件开发,熟悉嵌入式 U-boot, Linux BSP 驱动开发以及内存管理。

内容简介：

正点原子阿波罗 STM32F767 开发板试用体验报告

阿波罗 STM32F767IGT 开发板由底板+核心板构成，底板完全兼容 STM32F429 和 STM32F7 两款核心板。STM32F767 核心板的板载资源丰富，可以满足各种应用的需求，可以独立使用，也可以联合使用。

电子发烧友社区合作：liuyong@huaqiu.com
开发板试用平台：https://bbs.elecfans.com/try.html

目录

【阿波罗 STM32F767 试用体验】第一篇 让我来展览一番阿波罗

非常感谢发烧友论坛让我获得这次试用机会，我知道这次机会真的来之不易，所以我就迫不及待的给大家伙展览一下神秘的阿波罗...

快递&包装：

上面这两个盒子其中一个 KIT 里面装的是 7 寸液晶屏幕，请看大图：

另一个盒子里自然而然就是阿波罗了：

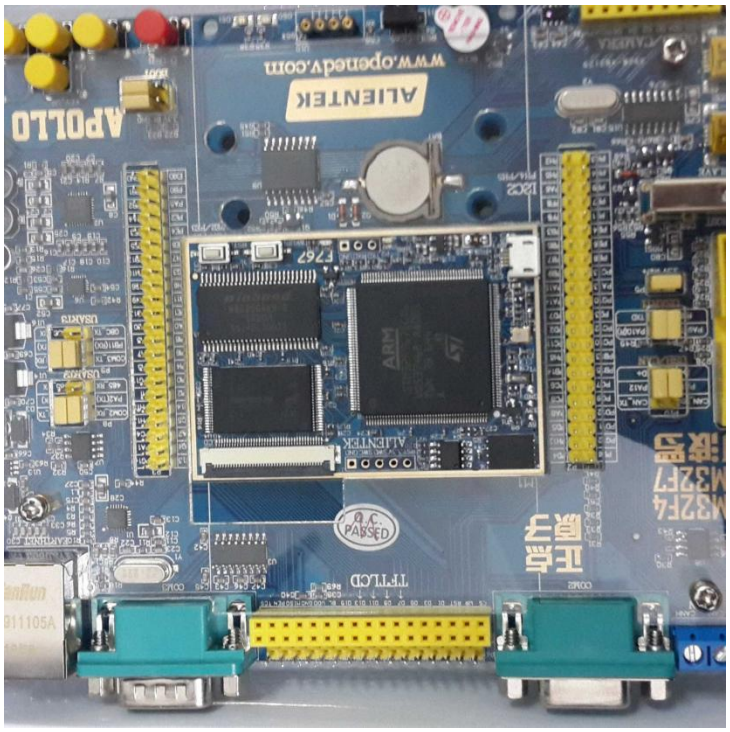
是不是很有科技感！！！

还有呢，盒子里的资料光盘也是这样极富科技感：

里面的内容我看了，就是百度云盘里面的资料 A 盘，不信的话请看我对光驱的截图：

还没看到阿波罗吧，马上就出来了，见下图：

特写一张：



接下来就是以下连接线了，主要是连接 st-link 的 u***线，电源线，等

下面看看盒子里的第二层，也是一些连接线：

至此，阿波罗的展览也就告一段落了。我自己就做一些总结了：整体包装很有档次，特别的外包装的设计，极富科技感，我很喜欢，至于开发板的性能，就需要后续的帖子来描述了，下次见。

【阿波罗 STM32F767 试用体验】第二篇 IAR 下闪烁 LED 灯

考虑到 IAR 的高效性，这次我选择 IAR 来变异阿波罗的工程，并且我会把代码都托管到 github 上。

https://github.com/iysheng/apollo.git

由于，我会不时地更新代码，所以试验效果可能会有出入，敬请见谅。

现在的代码现象是

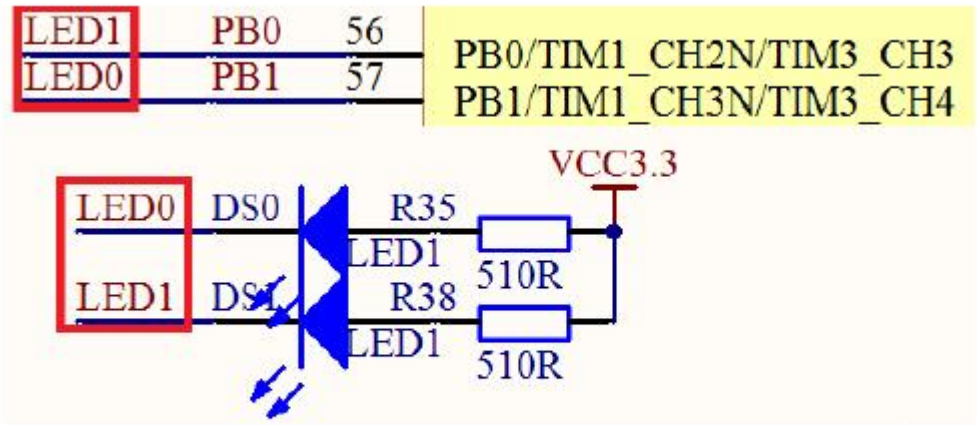
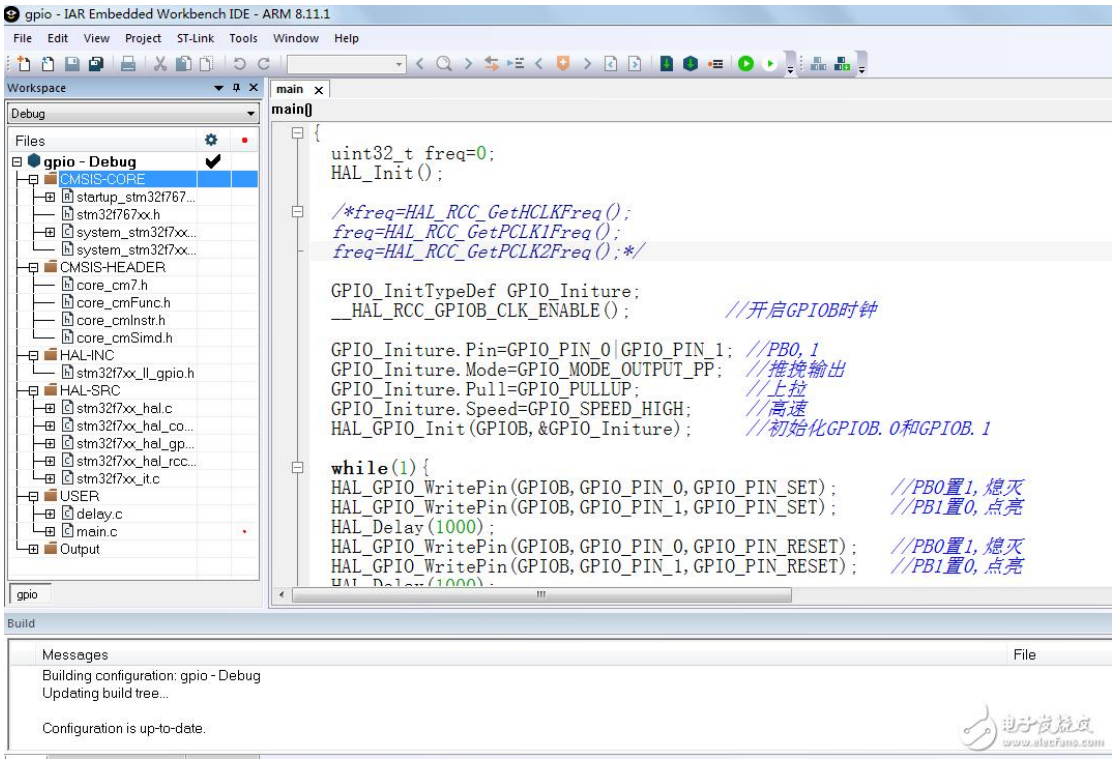


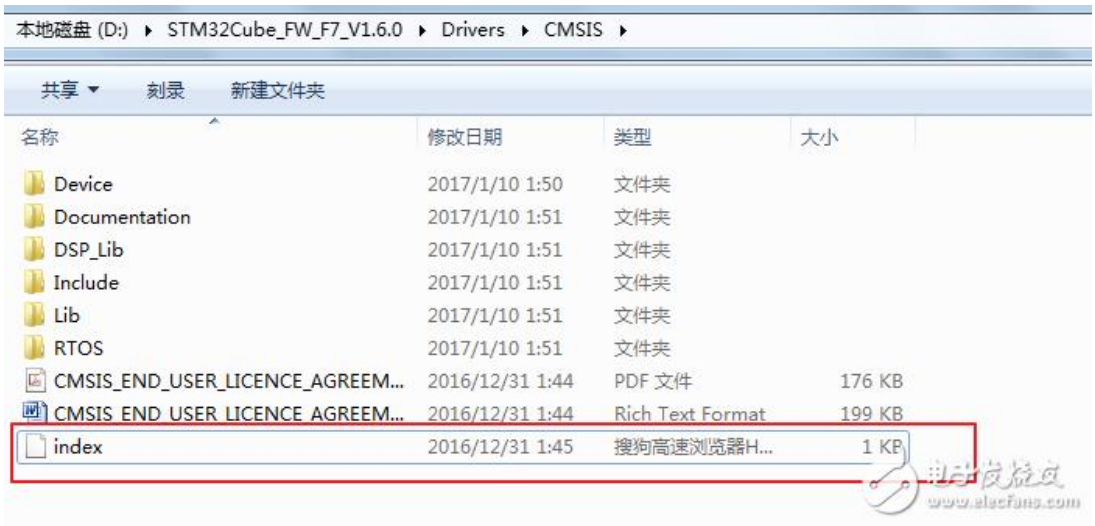
图 6.2.1 LED 与 STM32F767 连接原理图

LED0、LED1 每隔 1s 闪烁。

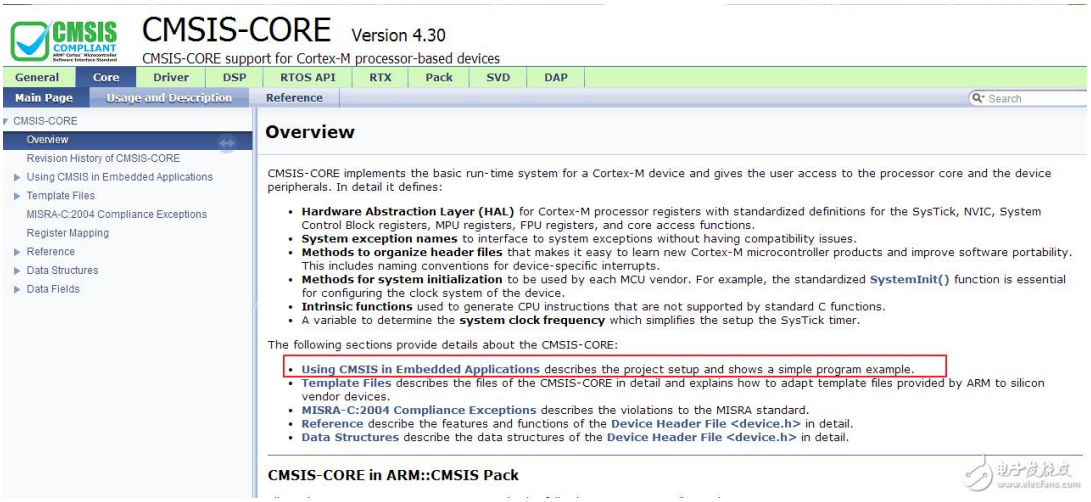
具体的工程我都上传了，看下简单的工程结构把：

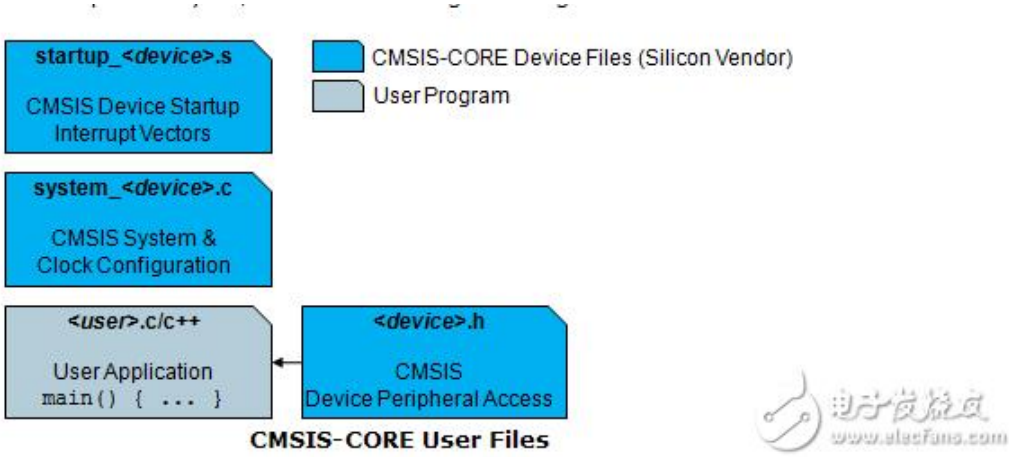


授之以鱼不如授之以渔，至于为什么会用到这些文件，我主要是参考：

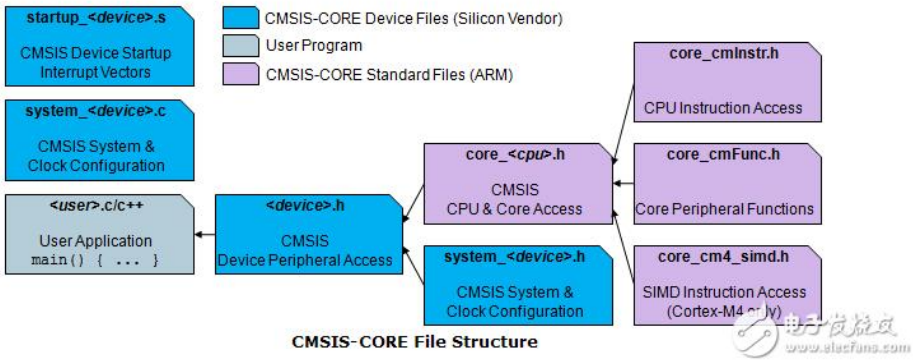


顺便截取一些主要的部分：

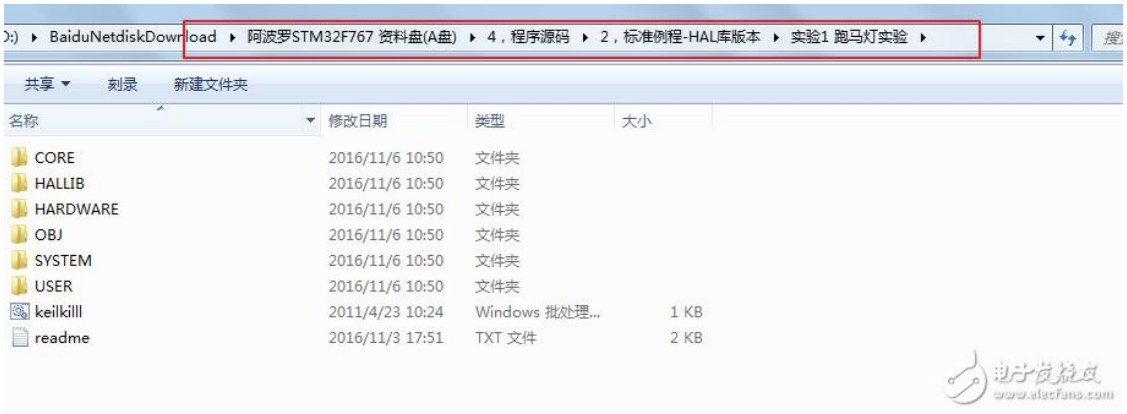




The detailed file structure of the CMSIS-CORE is shown in the following picture.

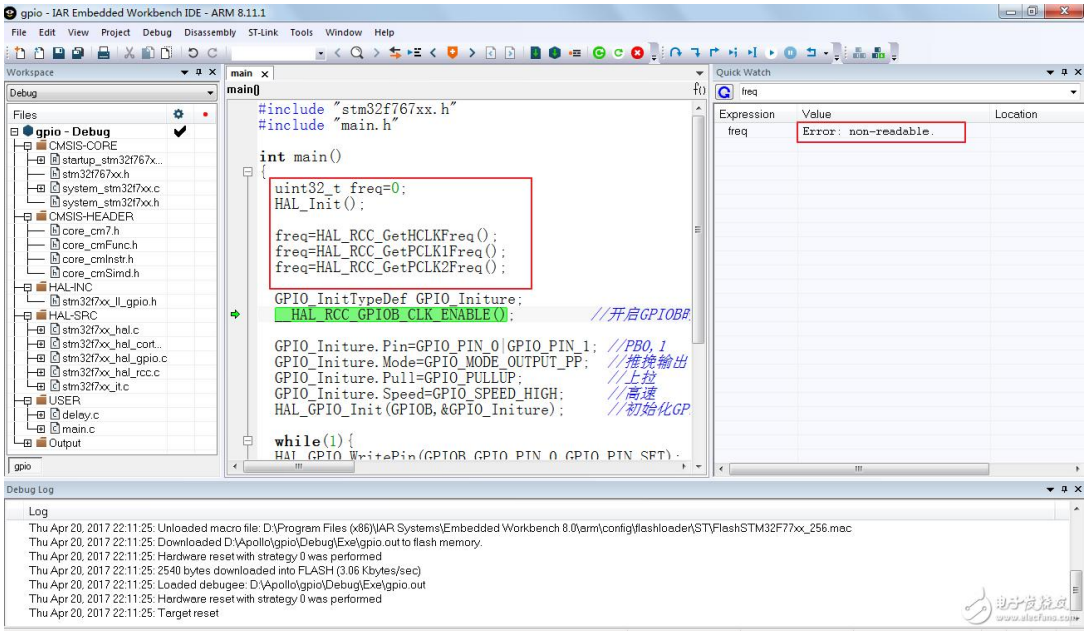


还有正点原子官方的流水灯的源代码，感谢正点原子。



至此，就可以实现工程的编译和下载了。

但是我遇到一个问题，截图如下：



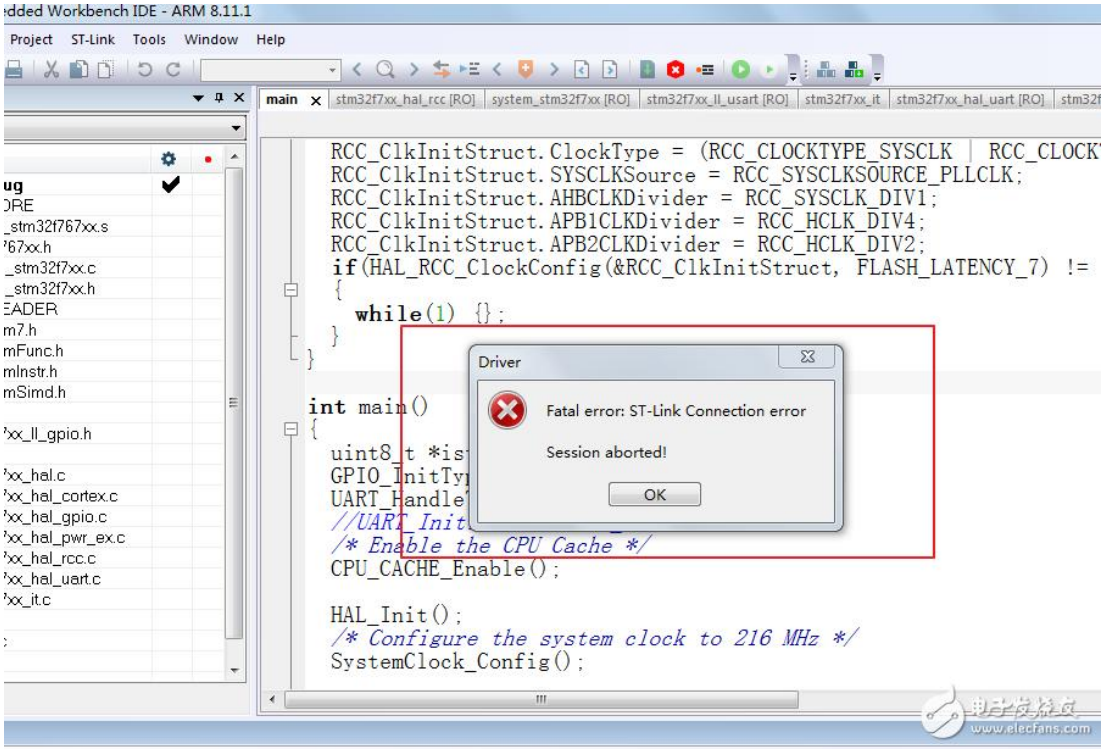
我想查看 freq 的值，但是执行这几条语句之后，都是提示 Error non-readable。

不知道失什么原因？

至于现象就很简单了，不班门弄斧的掩饰了。

【阿波罗 STM32F767 试用体验】第三篇 我以为阿波罗挂了呢

今天,在调试程序的时候,下载了一个"病毒程序",导致出现了如下问题:



所以,再我尝试重启电脑.以及重新安装 stlink 驱动都不行之后，我以为阿波罗挂了，那个伤心啊，，，

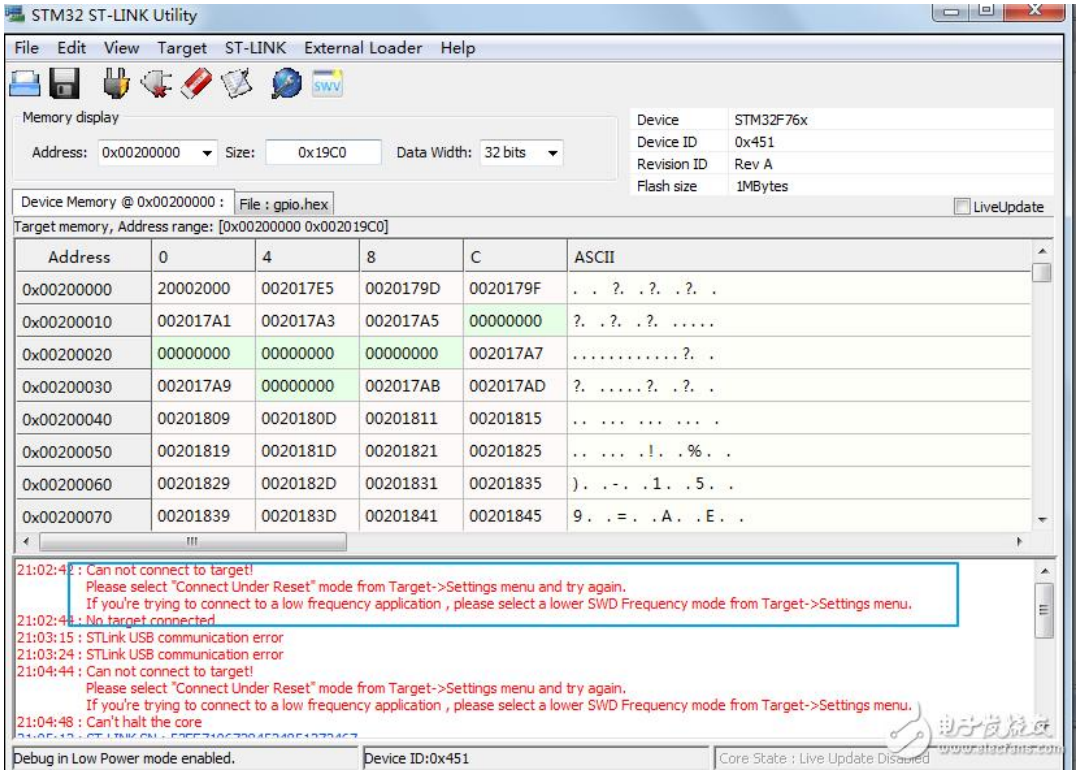
后来联系了技术支持，得知正常下载程序是不会烧坏芯片的，并且核心板的电源不亮是正常的，所以我稍微安心了点，就继续搜索解决办法，在一个网址上发现了点门路，

So I've started thinking that the problem was in the bootloader and I've spent a lot of time in internet searching in order to find a solution.

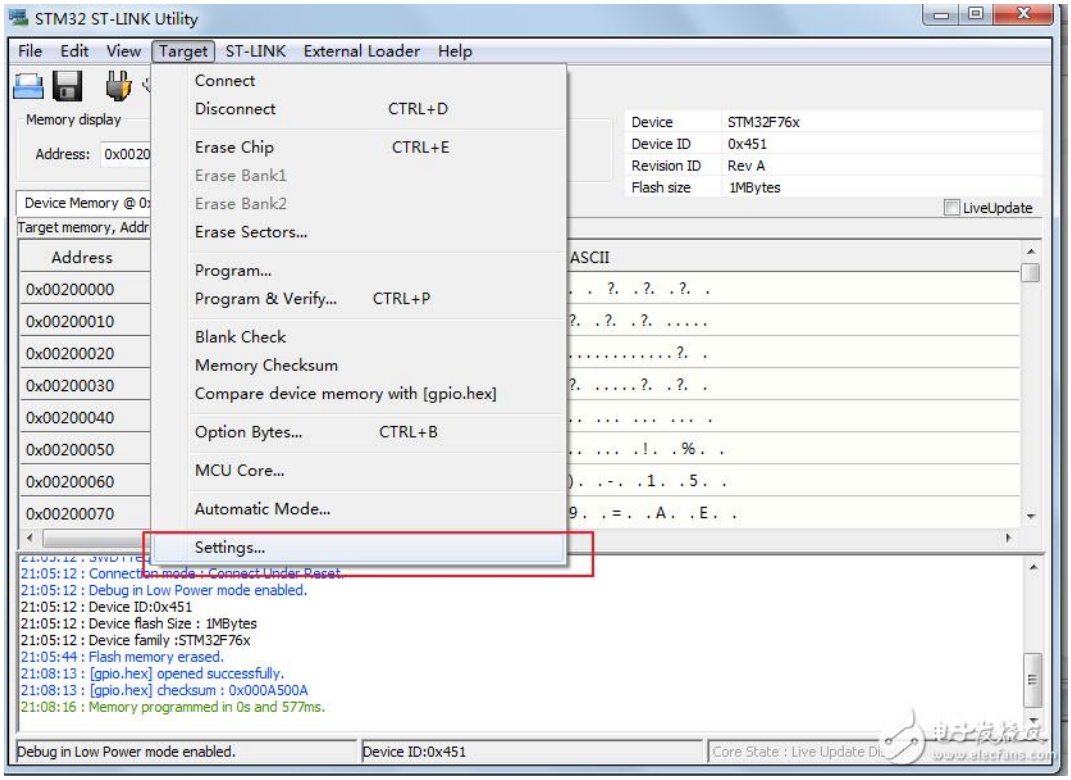
And et voilà, here is a topic on this problem: The Dreaded "Cannot connect to ST-LINK!" Error Message

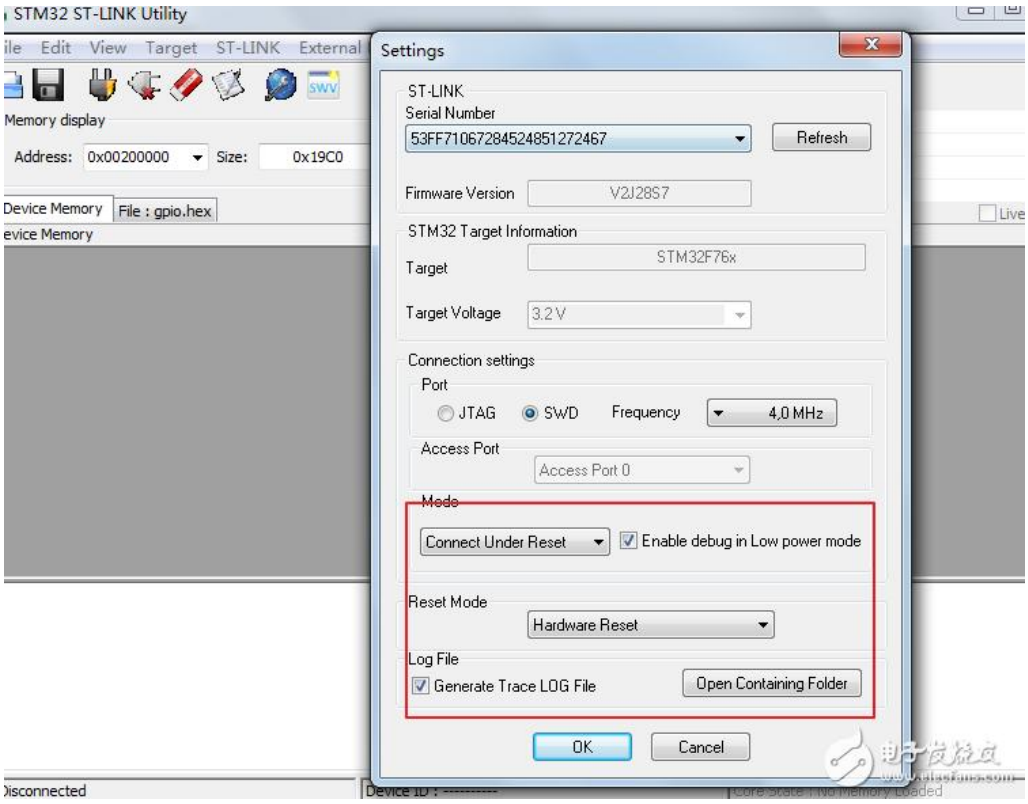
然后我就试着解决了这个问题，很开心。特此记录下来，需要做的就是一点，清除芯片内部的代码（病毒代码）。

需要的工具是 ST-LINK utility，下载安装之后，第一次尝试 connect chip 出现错误

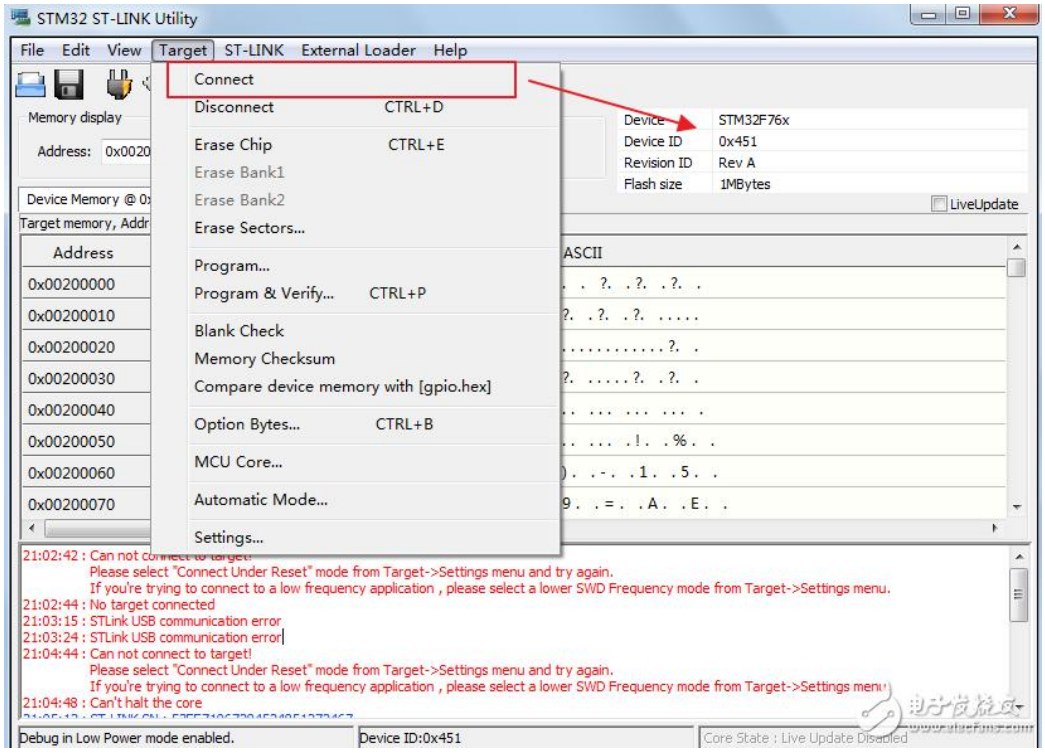


按照软件的提示进行修改配置

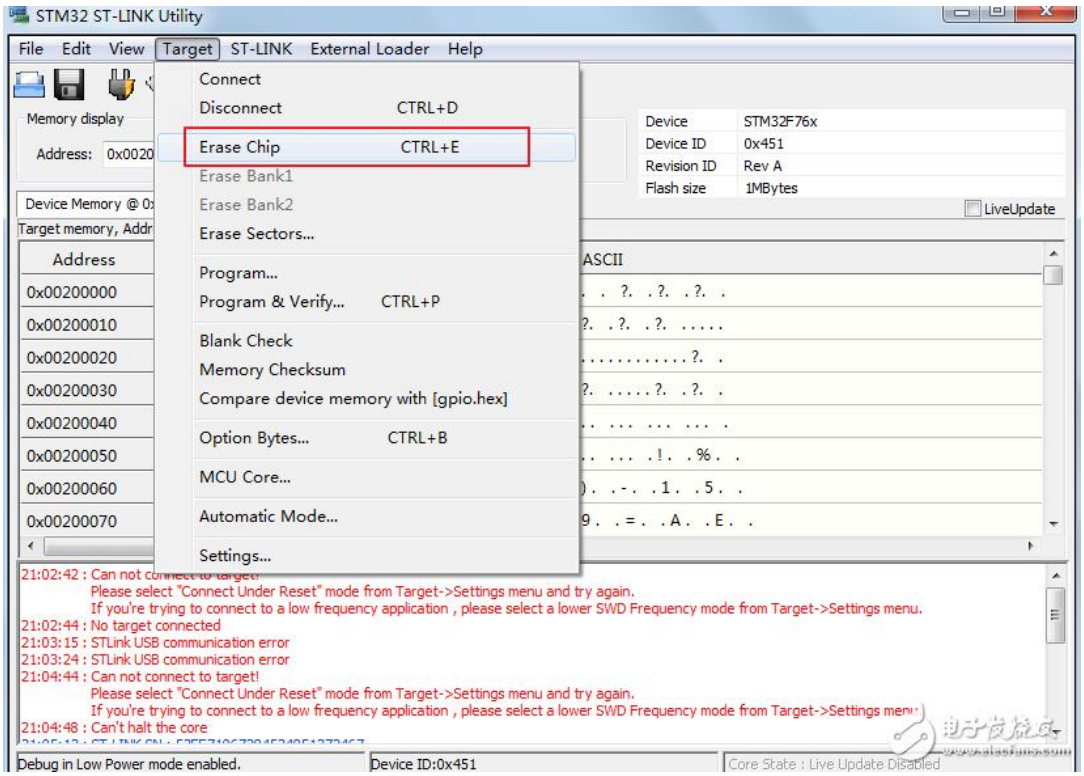




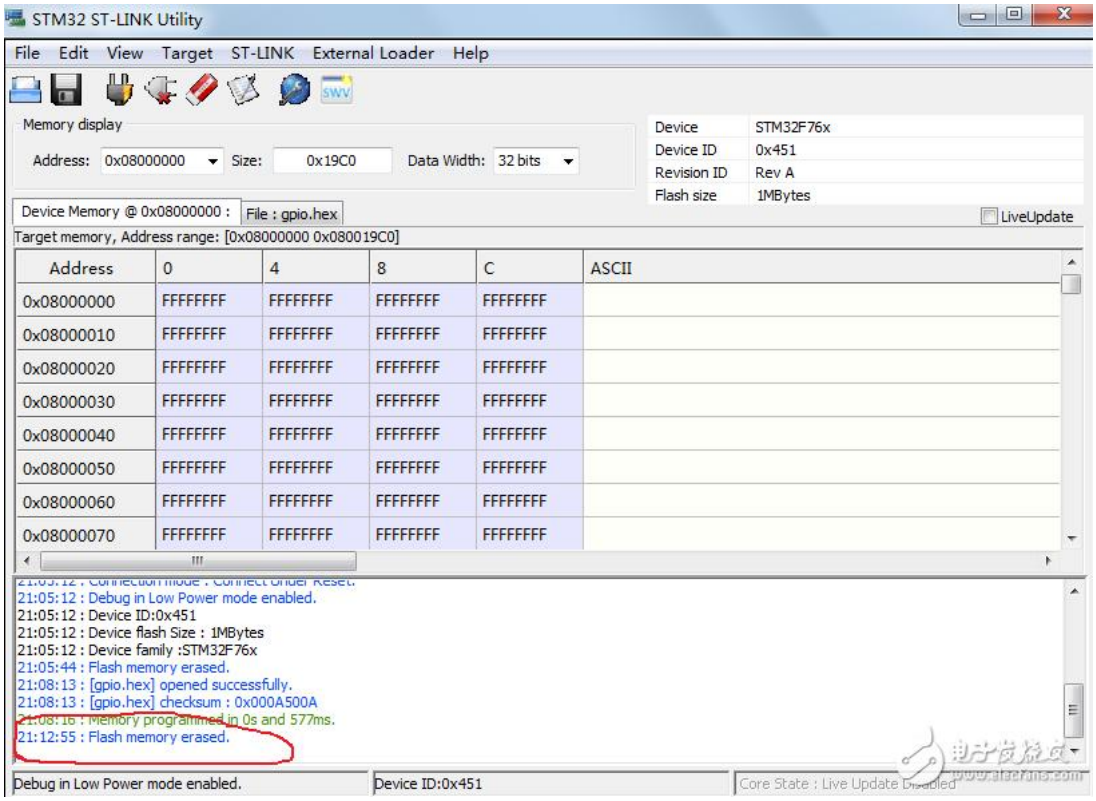
然后成功连接


STM32 ST-LINK Utility
File Edit View Target ST-LINK External Loader Help
Connect
Disconnect CTRL+D
Erase Chip CTRL+E
Erase Bank1
Erase Bank2
Erase Sectors...
Program...
Program & Verify... CTRL+P
Blank Check
Memory Checksum
Compare device memory with [gpio.hex]
Option Bytes... CTRL+B
MCU Core...
Automatic Mode...
Settings...
Memory display
Address: 0x0020
Device Memory @ 0x
Target memory, Addr
Address
0x00200000
0x00200010
0x00200020
0x00200030
0x00200040
0x00200050
0x00200060
0x00200070
Device STM32F76x
Device ID 0x451
Revision ID Rev A
Flash size 1MBytes
LiveUpdate
ASCII
21:02:42 : Can not connect to target!
Please select "Connect Under Reset" mode from Target->Settings menu and try again.
If you're trying to connect to a low frequency application , please select a lower SWD Frequency mode from Target->Settings menu.
21:02:44 : No target connected
21:03:15 : STLink USB communication error
21:03:24 : STLink USB communication error
21:04:44 : Can not connect to target!
Please select "Connect Under Reset" mode from Target->Settings menu and try again.
If you're trying to connect to a low frequency application , please select a lower SWD Frequency mode from Target->Settings menu
21:04:48 : Can't halt the core
Debug in Low Power mode enabled.
Device ID:0x451
Core State : Live Update Disabled

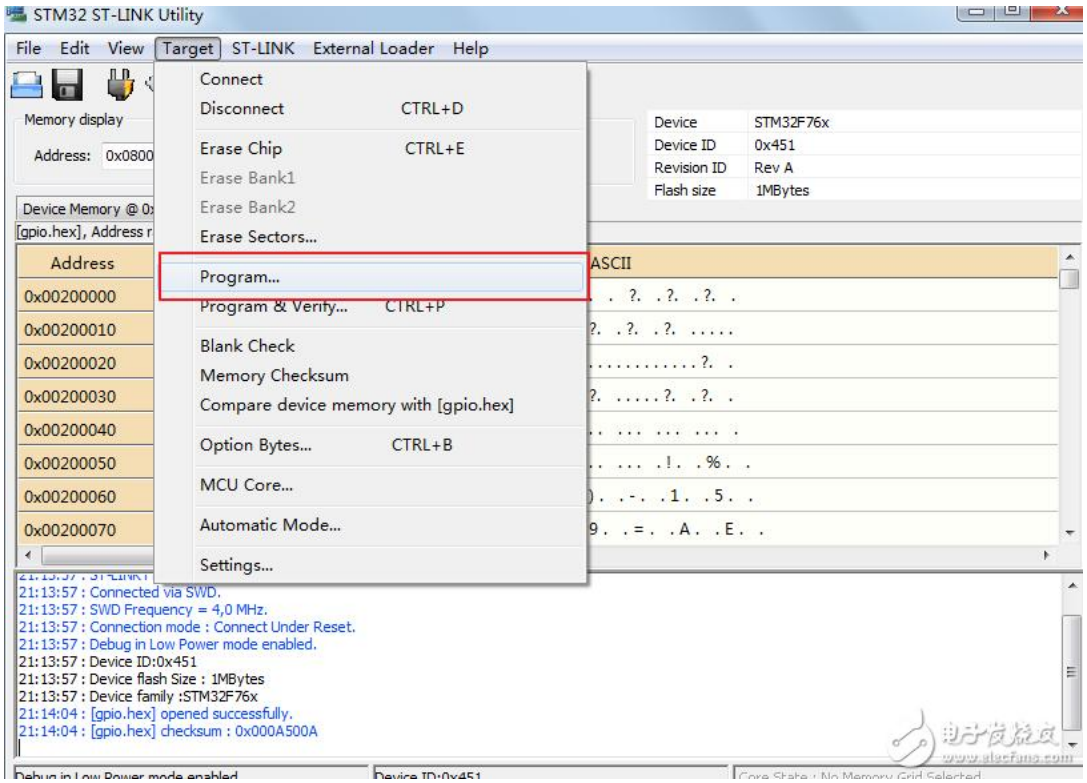
开始清除芯片内的程序

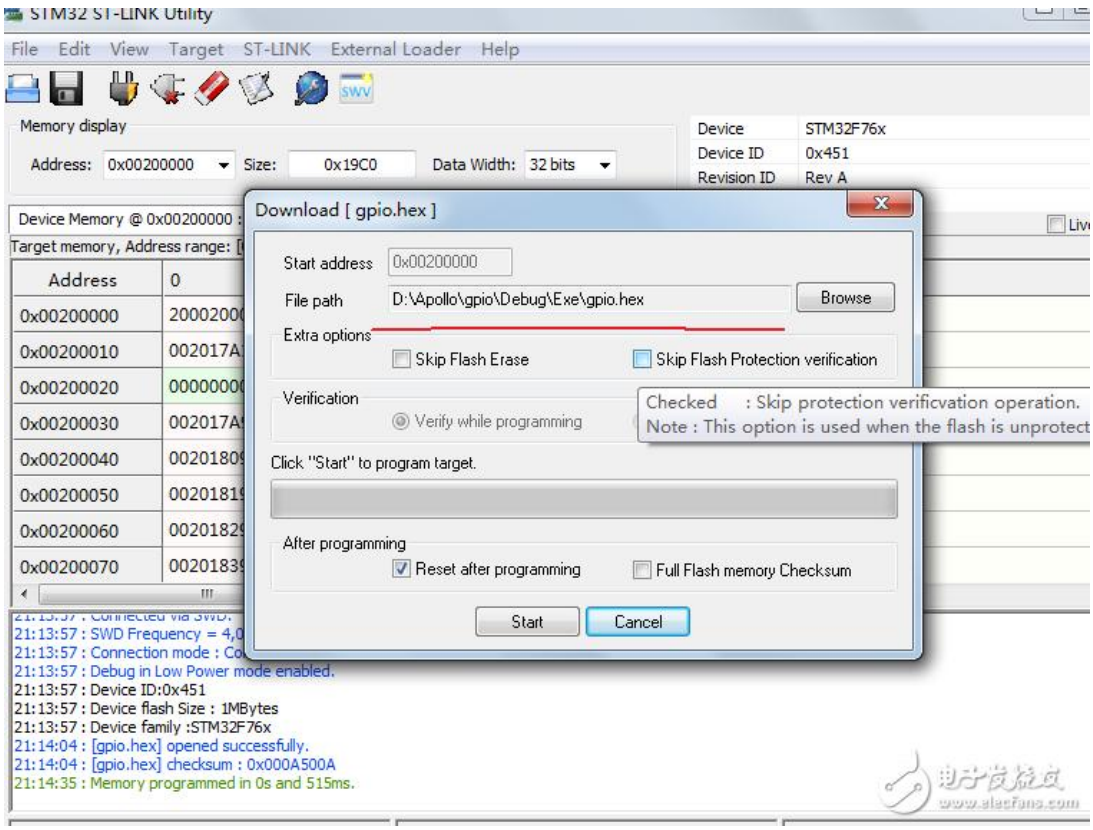

STM32 ST-LINK Utility
File Edit View Target ST-LINK External Loader Help
Connect
Disconnect CTRL+D
Erase Chip CTRL+E
Erase Bank1
Erase Bank2
Erase Sectors...
Program...
Program & Verify... CTRL+P
Blank Check
Memory Checksum
Compare device memory with [gpio.hex]
Option Bytes... CTRL+B
MCU Core...
Automatic Mode...
Settings...
Memory display
Address: 0x0020
Device STM32F76x
Device ID 0x451
Revision ID Rev A
Flash size 1MBytes
LiveUpdate
Address
ASCII
0x00200000
0x00200010
0x00200020
0x00200030
0x00200040
0x00200050
0x00200060
0x00200070
21:02:42 : Can not connect to target!
Please select "Connect Under Reset" mode from Target->Settings menu and try again.
If you're trying to connect to a low frequency application , please select a lower SWD Frequency mode from Target->Settings menu.
21:02:44 : No target connected
21:03:15 : STLink USB communication error
21:03:24 : STLink USB communication error
21:04:44 : Can not connect to target!
Please select "Connect Under Reset" mode from Target->Settings menu and try again.
If you're trying to connect to a low frequency application , please select a lower SWD Frequency mode from Target->Settings menu.
21:04:48 : Can't halt the core
Debug in Low Power mode enabled.
Device ID:0x451
Core State : Live Update Disabled

成功清除


STM32 ST-LINK Utility
File Edit View Target ST-LINK External Loader Help
Memory display
Address: 0x08000000 Size: 0x19C0 Data Width: 32 bits
Device STM32F76x
Device ID 0x451
Revision ID Rev A
Flash size 1MBytes
Device Memory @ 0x08000000 :
File : gpio.hex
LiveUpdate
Target memory, Address range: [0x08000000 0x080019C0]
Address 0 4 8 C ASCII
0x08000000 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x08000010 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x08000020 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x08000030 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x08000040 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x08000050 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x08000060 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x08000070 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
21:05:12 : Debug in Low Power mode enabled.
21:05:12 : Device ID:0x451
21:05:12 : Device flash Size : 1MBytes
21:05:12 : Device family :STM32F76x
21:05:44 : Flash memory erased.
21:08:13 : [gpio.hex] opened successfully.
21:08:13 : [gpio.hex] checksum : 0x000A500A
21:08:16 : Memory programmed in 0s and 577ms.
21:12:55 : Flash memory erased.
Debug in Low Power mode enabled.
Device ID:0x451
Core State : Live Update Disabled

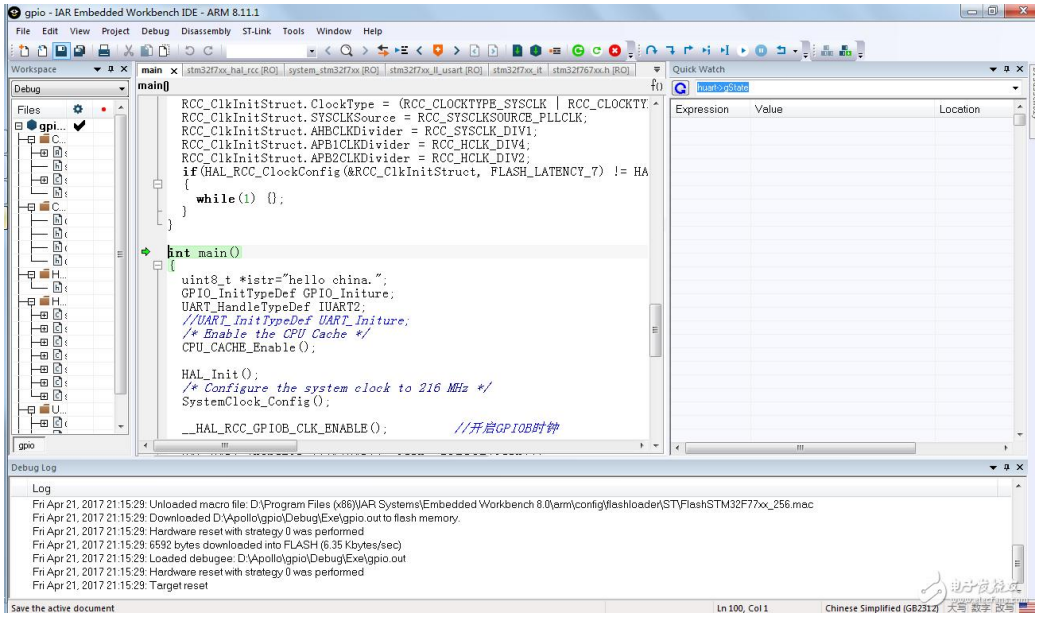
然后就可以了，我们可以通过 stlink 直接下载程序文件，如下所示





成功下载，说明在清除了片上的程序之后， 已经可以连接阿波罗了。

然后在 IAR 下查看可以进入调试界面，说明此时已经成功解决了阿波罗连不上电脑的问题。



【阿波罗 STM32F767 试用体验】第四篇 细说阿波罗的时钟以及通用定时器测试

老实说，用过很多的芯片，但是都没有太仔细的研究过芯片的细节，只要做的工作还是停留在调用函数的层面，但是要想更深入的学习一款芯片不了解细节是不行的。于是，我就拿阿波罗来班门弄斧了。粗略看了阿波罗的时钟体系，做一些笔记，记录下来（如果说的有错误，希望大家积极指正，我将不胜感激）：从时钟源的角度，分为两类外部时钟（E）和内部时钟（I）。

从时钟速率的角度，分为两类高速时钟(HS)和低速时钟(LS)。

而把它们组合起来就有四种时钟：HSE、HIS、LSE、LSI。至于为什么会有这么复杂的时钟配置，主要是考虑到系统的性能和功耗两个方面的因素吧。单一时钟的话可能会导致性能过剩并且功耗过高。多个时钟的话可以平衡功耗和性能之间的平衡。

特此说明一下，系统复位后，默认初始化的是 HIS 时钟提供 sysclock。也就是 16MHZ。为了提示系统性能，我们需要使能外部时钟晶振（板载 25MHz）。使能后也明显可以看出来芯片的温度升高了（使用内部的温度传感器测试，在后续有图片为证）。

这四类时钟在芯片内部通过配置，完成对各个外设的驱动。到了芯片内部，对应到那么多的外设，时钟的分类就更多了，但是主要考虑到桥的存在，分为五类：AHB3、AHB2、AHB1、APB2、APB1。芯片内的所有外设都分别挂载在这五个

总线上，至于哪个外设挂歪在哪个总线上，我们就需要查看芯片的 RM0410 Reference manual（Page74）了。

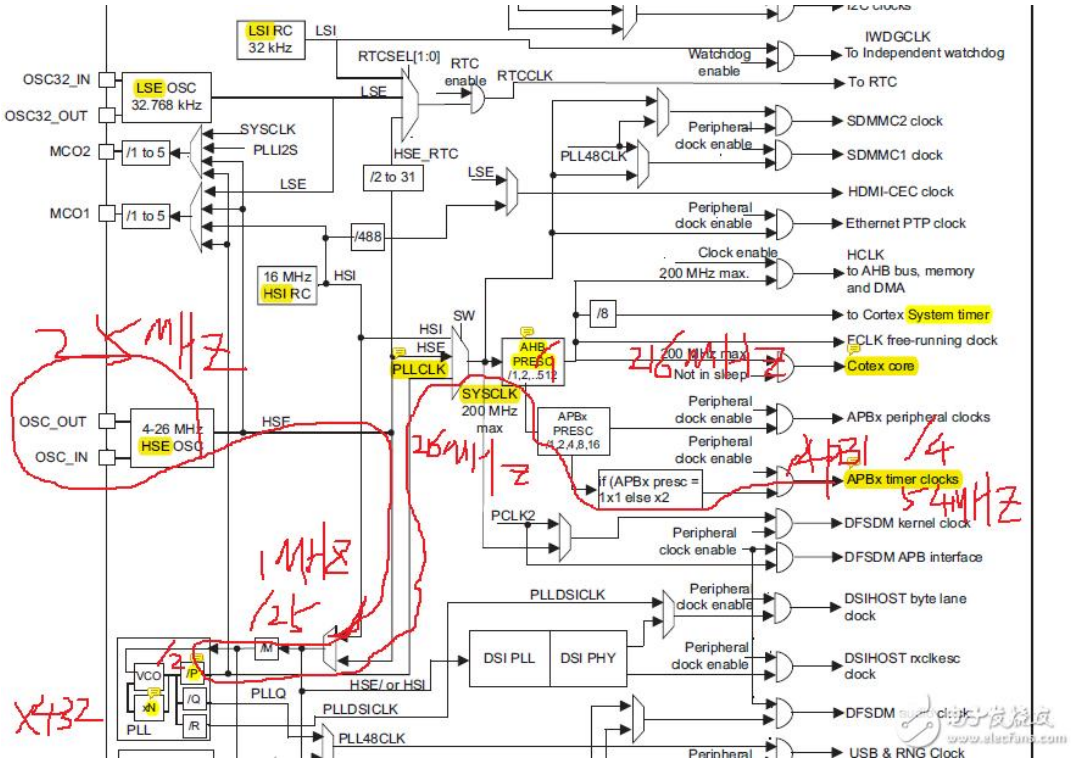
今天为了测试时钟的配置，我就用通用定时器 tiM2 做下示范操作：

首先查看 TIM2 挂载在哪条总线：

0x4000 4400 - 0x4000 47FF	USART2		
0x4000 4000 - 0x4000 43FF	SPDIFRX	APB1	Section 37.5.10: SPDIFRX interface register map on page 1409
0x4000 3C00 - 0x4000 3FFF	SPI3 / I2S3		Section 35.9.10: SPI/I2S register map on page 1328
0x4000 3800 - 0x4000 3BFF	SPI2 / I2S2		
0x4000 3400 - 0x4000 37FF	CAN3		Section 40.9.5: bxCAN register map on page 1524
0x4000 3000 - 0x4000 33FF	IWDG		Section 30.4.6: IWDG register map on page 1085
0x4000 2C00 - 0x4000 2FFF	WWDG		Section 31.4.4: WWDG register map on page 1092
0x4000 2800 - 0x4000 2BFF	RTC & BKP Registers		Section 32.6.21: RTC register map on page 1136
0x4000 2400 - 0x4000 27FF	LPTIM1		Section 29.6.9: LPTIM register map on page 1076
0x4000 2000 - 0x4000 23FF	TIM14		Section 27.5.12: TIM10/TIM11/TIM13/TIM14 register map on page 1040
0x4000 1C00 - 0x4000 1FFF	TIM13		
0x4000 1800 - 0x4000 1BFF	TIM12		Section 27.4.13: TIM9/TIM12 register map on page 1030
0x4000 1400 - 0x4000 17FF	TIM7		Section 28.4.9: TIM6/TIM7 register map on page 1055
0x4000 1000 - 0x4000 13FF	TIM6		
0x4000 0C00 - 0x4000 0FFF	TIM5		Section 26.4.21: TIMx register map on page 991
0x4000 0800 - 0x4000 0BFF	TIM4		
0x4000 0400 - 0x4000 07FF	TIM3		
0x4000 0000 - 0x4000 03FF	TIM2		

可知挂载在 APB1 总线上。

接着就可以查看 TIM 的时钟回路：



从上述截图中可以看出，最后 TIM 时钟的周期是 54MHZ。具体配置关键代码如下：

```
1. void SystemClock_Config(void)
2. {
3.  RCC_ClkInitTypeDef RCC_ClkInitStruct;
4.  RCC_OscInitTypeDef RCC_OscInitStruct;
5.
6.  /* Enable HSE Oscillator and activate PLL with HSE as source */
7.  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
8.  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
9.  RCC_OscInitStruct.HSIState = RCC_HSI_OFF;
```

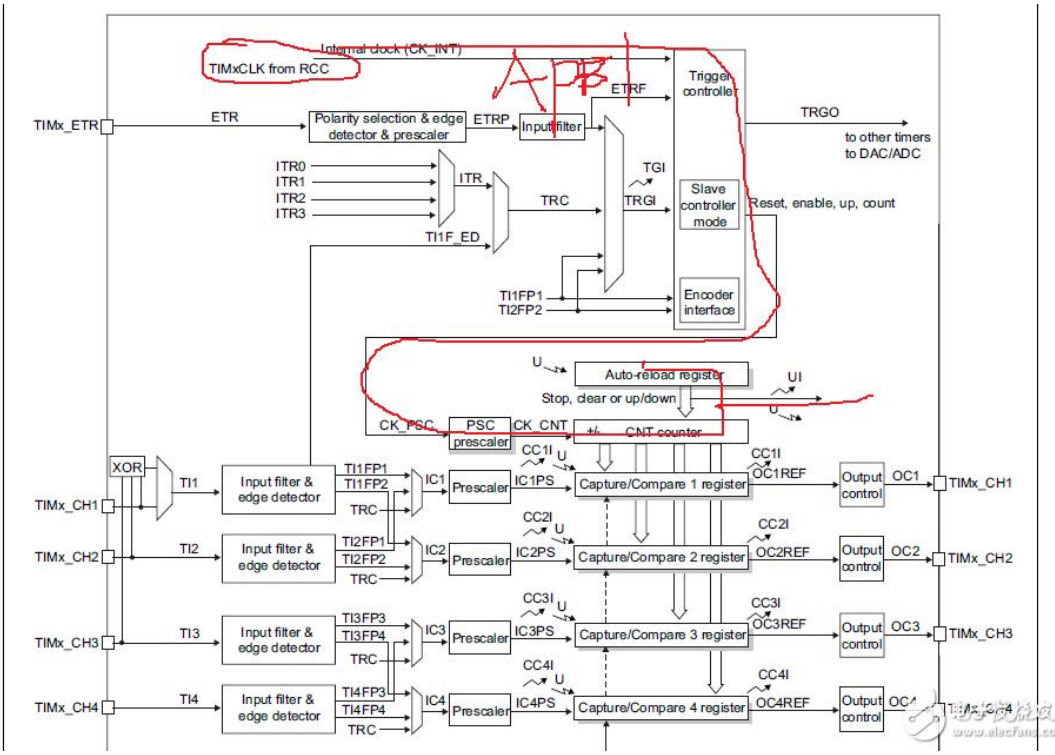
```
10.  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
11.  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
12.  RCC_OscInitStruct.PLL.PLLM = 25;
13.  RCC_OscInitStruct.PLL.PLLN = 432;
14.  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
15.  RCC_OscInitStruct.PLL.PLLQ = 9;
16.  RCC_OscInitStruct.PLL.PLLR = 7;
17.  if(HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
18.  {
19.    while(1) {};
20.  }
21.
22.  /* Activate the OverDrive to reach the 216 Mhz Frequency */
23.  if(HAL_PWREx_EnableOverDrive() != HAL_OK)
24.  {
25.    while(1) {};
26.  }
27.
28.  /* Select PLL as system clock source and configure the HCLK,
     PCLK1 and PCLK2
29.    clocks dividers */
```

```
30.  RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK |
     RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_PCLK1 |
     RCC_CLOCKTYPE_PCLK2);
31.  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
32.  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
33.  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
34.  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
35.  if(HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_7) !=
     HAL_OK)
36.  {
37.      while(1) {};
38.  }
39. }
40.
```

复制代码

考虑到 ST 的代码很清晰明了，对照着我作的标记大家应该可以看明白吧。

接着就进行配置 TIM2 的时钟分频，以及计数周期。由于 TIM2 的时钟可以由多个来源，既然我们用的是 APB1 时钟，那么就输入内部时钟也就是：

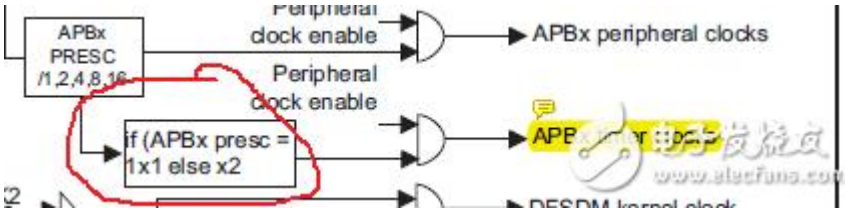


贴出对应的关键代码

```
1.  void TIM_init(TIM_HandleTypeDef *ITIMX)
2.  {
3.   ITIMX->Instance=TIM3;
4.   ITIMX->Init.Period=999;
5.   ITIMX->Init.Prescaler=53999;
6.   ITIMX->Init.CounterMode=TIM_COUNTERMODE_UP;
7.   HAL_TIM_Base_Init(ITIMX);
8.   HAL_TIM_Base_Start_IT(ITIMX);
9.  }
10.
```

复制代码

539999 对应上图中的 PSC Prescaler，分频后时钟频率为 54MHz/（53999+1）=1KHz。**接下来就要注意一点了，**在 TIM2 采样时钟之前，有



如果 APB 的分频比是 1，那么频率不变，否则频率 乘以 2，这样一来可以确定 TIM2 的采样频率是 2kHz 了。

接下来配置计数周期为 999。也就是对应 0.5s。在中断函数中又如下代码：

```
1.  void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
2.  {
3.   led_flag++;
4.   if(led_flag%2)
5.     LED_on();
6.   else
7.     LED_off();
8.  }
9.
```

复制代码

这样一来，就可以实现 DS0 和 DS1 交替 1s 闪烁，通过计时测试，成功了。

由此，使用 TIM2 演示时钟配置的任务已经完成了。

声明一下，此次阿波罗的试用，我会把我各个阶段的完整的代码托管到 github 上，如需查看欢迎 fork。

https://github.com/iysheng/apollo

【阿波罗 STM32F767 试用体验】第五篇 阿波罗的 RGB 屏幕显示图片还是会挺清晰的

在了解了正点原子的屏幕代码之后，我发现为了驱动某一特定的屏幕（RGB 或者 MCU 屏），官方的代吗有很多冗余的地方，于是我就把需要的函数提取出来移植到了 IAR 的 IDE 下。完整的代码在 github 上。

https://github.com/iysheng/apollo

具体的正点原子的代码，我就不在这里班门弄斧的分析了，主要说一些官方代码里面没有的部分。我发现官方代码里面有直接显示 ascii 码的函数。但是没有直接显示图片的函数，于是我就添加了这么一个函数。主要代码如下：

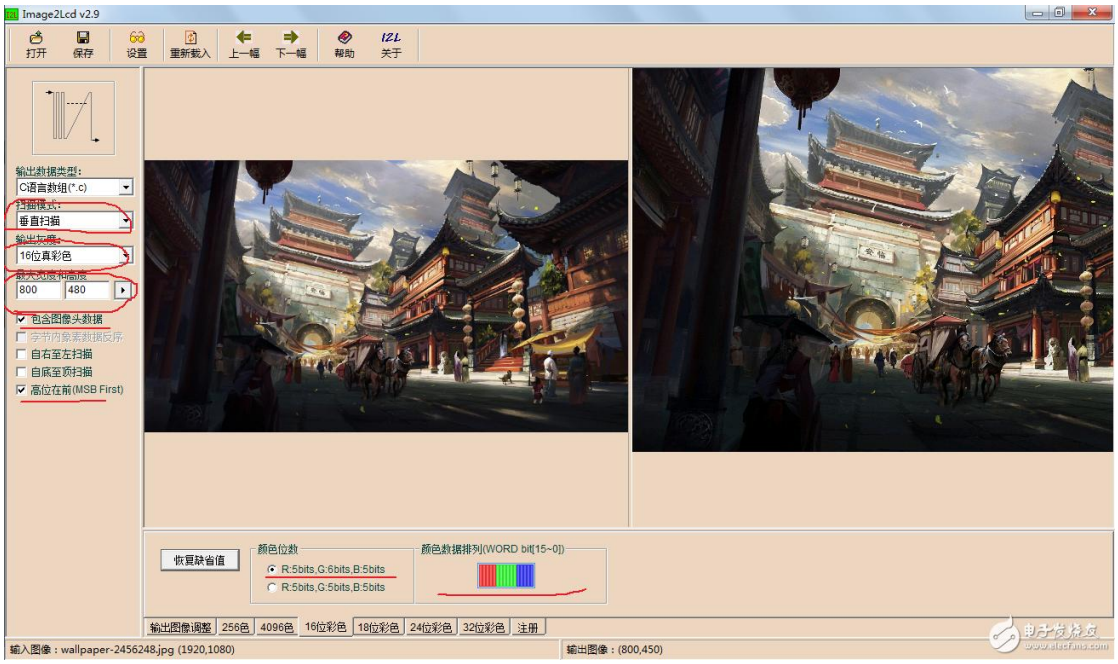
```
/*
*added by iysheng@163.com
*posx,posy:图像的起始点坐标
*uint8_t* 图像数组收地址
*/
void APPOLO_RGB(uint16_t posx,uint16_t posy,uint8_t *pic)
{
uint32_t width,high,piex,uitemp[2];
if(pic[0]&0x01)
{
   width=(pic[2]<<8)|pic[3];
   high=(pic[4]<<8)|pic[5];
   for(uitemp[0]=0;uitemp[0]<width;uitemp[0]++)
    {
      for(uitemp[1]=0;uitemp[1]<high*2;)
      {
```

```
17.         piex=((pic[(high<<1)*uitemp[0]+uitemp[1]])<<8)|(pic
    [(high<<1)*uitemp[0]+uitemp[1]+1]);
18.         LTDC_Draw_Point(uitemp[0]+posx,posy+uitemp[1]/2,(ui
    nt32_t)(piex));
19.       uitemp[1]+=2;
20.         }
21.     }
22. }
23. }
```

复制代码

上面的代码（默认横屏显示），可以起到自动判断图像大小（800*480 之内，因为屏幕的分辨率限制着呢）。还有很多可以优化的地方：比如说自动判断横竖屏显示等等，我就简单的希望可以起到抛砖引玉的作用吧。

为了正常显示图片，我们需要这样设置：

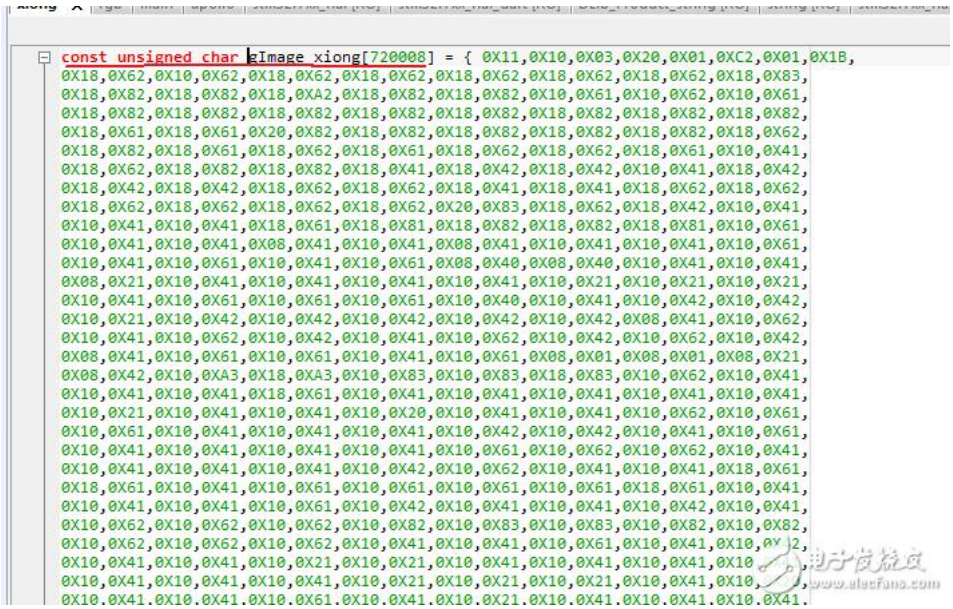


然后， 我们就可以显示这样图片了，调用方法如下：

```
1.  APPOLO_RGB(0,0,gImage_xiong);
```

复制代码

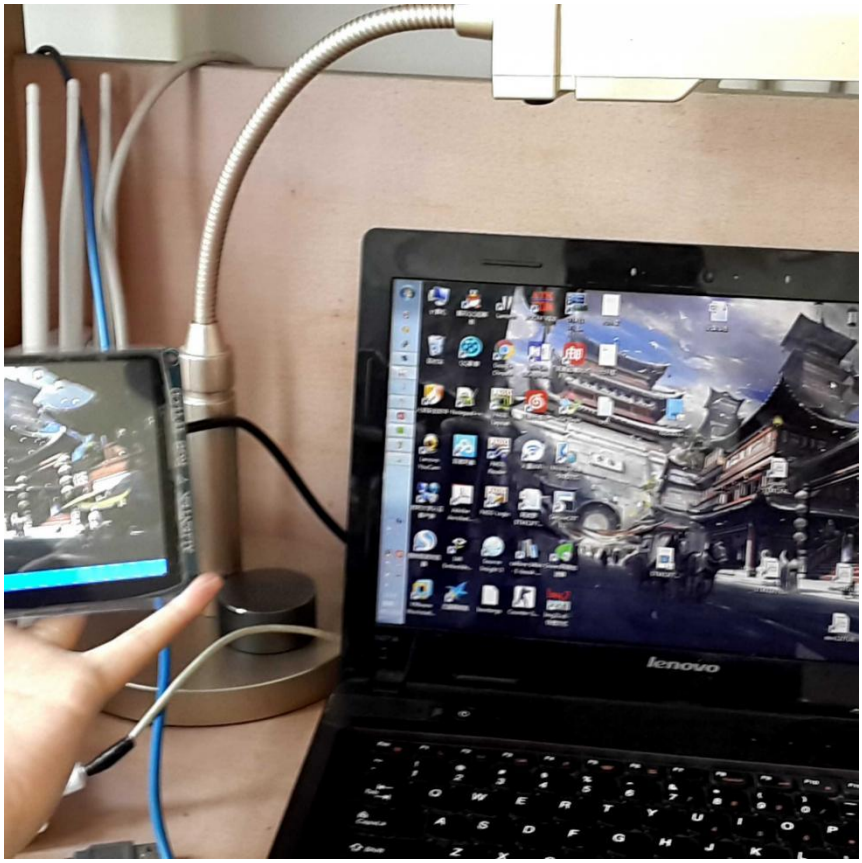
其中 gImage_xiong 定义如下：



```
const unsigned char gImage_xiong[720008] = { 0X11,0X10,0X03,0X20,0X01,0XC2,0X01,0X1B,
0X18,0X62,0X10,0X62,0X18,0X62,0X18,0X62,0X18,0X62,0X18,0X62,0X18,0X62,0X18,0X83,
0X18,0X82,0X18,0X82,0X18,0XA2,0X18,0X82,0X18,0X82,0X10,0X61,0X10,0X62,0X10,0X61,
0X18,0X82,0X18,0X82,0X18,0X82,0X18,0X82,0X18,0X82,0X18,0X82,0X18,0X82,0X18,0X82,
0X18,0X61,0X18,0X61,0X20,0X82,0X18,0X82,0X18,0X82,0X18,0X82,0X18,0X82,0X18,0X62,
0X18,0X82,0X18,0X61,0X18,0X62,0X18,0X61,0X18,0X62,0X18,0X62,0X18,0X61,0X10,0X41,
0X18,0X62,0X18,0X82,0X18,0X82,0X18,0X41,0X18,0X42,0X18,0X42,0X10,0X41,0X18,0X42,
0X18,0X42,0X18,0X42,0X18,0X62,0X18,0X62,0X18,0X41,0X18,0X41,0X18,0X62,0X18,0X62,
0X18,0X62,0X18,0X62,0X18,0X62,0X18,0X62,0X20,0X83,0X18,0X62,0X18,0X42,0X10,0X41,
0X10,0X41,0X10,0X41,0X18,0X61,0X18,0X81,0X18,0X82,0X18,0X82,0X18,0X81,0X10,0X61,
0X10,0X41,0X10,0X41,0X08,0X41,0X10,0X41,0X08,0X41,0X10,0X41,0X10,0X41,0X10,0X61,
0X10,0X41,0X10,0X61,0X10,0X41,0X10,0X61,0X08,0X40,0X08,0X40,0X10,0X41,0X10,0X41,
0X08,0X21,0X10,0X41,0X10,0X41,0X10,0X41,0X10,0X41,0X10,0X21,0X10,0X21,0X10,0X21,
0X10,0X41,0X10,0X61,0X10,0X61,0X10,0X61,0X10,0X40,0X10,0X41,0X10,0X42,0X10,0X42,
0X10,0X21,0X10,0X42,0X10,0X42,0X10,0X42,0X10,0X42,0X10,0X42,0X08,0X41,0X10,0X62,
0X10,0X41,0X10,0X62,0X10,0X42,0X10,0X41,0X10,0X62,0X10,0X42,0X10,0X62,0X10,0X42,
0X08,0X41,0X10,0X61,0X10,0X61,0X10,0X41,0X10,0X61,0X08,0X01,0X08,0X01,0X08,0X21,
0X08,0X42,0X10,0XA3,0X18,0XA3,0X10,0X83,0X10,0X83,0X18,0X83,0X10,0X62,0X10,0X41,
0X10,0X41,0X10,0X41,0X18,0X61,0X10,0X41,0X10,0X41,0X10,0X41,0X10,0X41,0X10,0X41,
0X10,0X21,0X10,0X41,0X10,0X41,0X10,0X20,0X10,0X41,0X10,0X41,0X10,0X62,0X10,0X61,
0X10,0X61,0X10,0X41,0X10,0X41,0X10,0X41,0X10,0X42,0X10,0X42,0X10,0X41,0X10,0X61,
0X10,0X41,0X10,0X41,0X10,0X41,0X10,0X41,0X10,0X61,0X10,0X62,0X10,0X62,0X10,0X41,
0X10,0X41,0X10,0X41,0X10,0X41,0X10,0X42,0X10,0X62,0X10,0X41,0X10,0X41,0X18,0X61,
0X18,0X61,0X10,0X41,0X10,0X61,0X10,0X61,0X10,0X61,0X10,0X61,0X18,0X61,0X10,0X41,
0X10,0X41,0X10,0X41,0X10,0X61,0X10,0X42,0X10,0X41,0X10,0X41,0X10,0X42,0X10,0X41,
0X10,0X62,0X10,0X62,0X10,0X62,0X10,0X82,0X10,0X83,0X10,0X83,0X10,0X82,0X10,0X82,
0X10,0X62,0X10,0X62,0X10,0X62,0X10,0X41,0X10,0X41,0X10,0X61,0X10,0X41,0X10,0X42,
0X10,0X41,0X10,0X41,0X10,0X21,0X10,0X21,0X10,0X41,0X10,0X41,0X10,0X41,0X10,0X42,
0X10,0X41,0X10,0X41,0X10,0X41,0X10,0X21,0X10,0X21,0X10,0X21,0X10,0X41,0X10,0X42,
0X10,0X41,0X10,0X41,0X10,0X61,0X10,0X41,0X10,0X21,0X10,0X41,0X10,0X41,0X10,0X41,
```

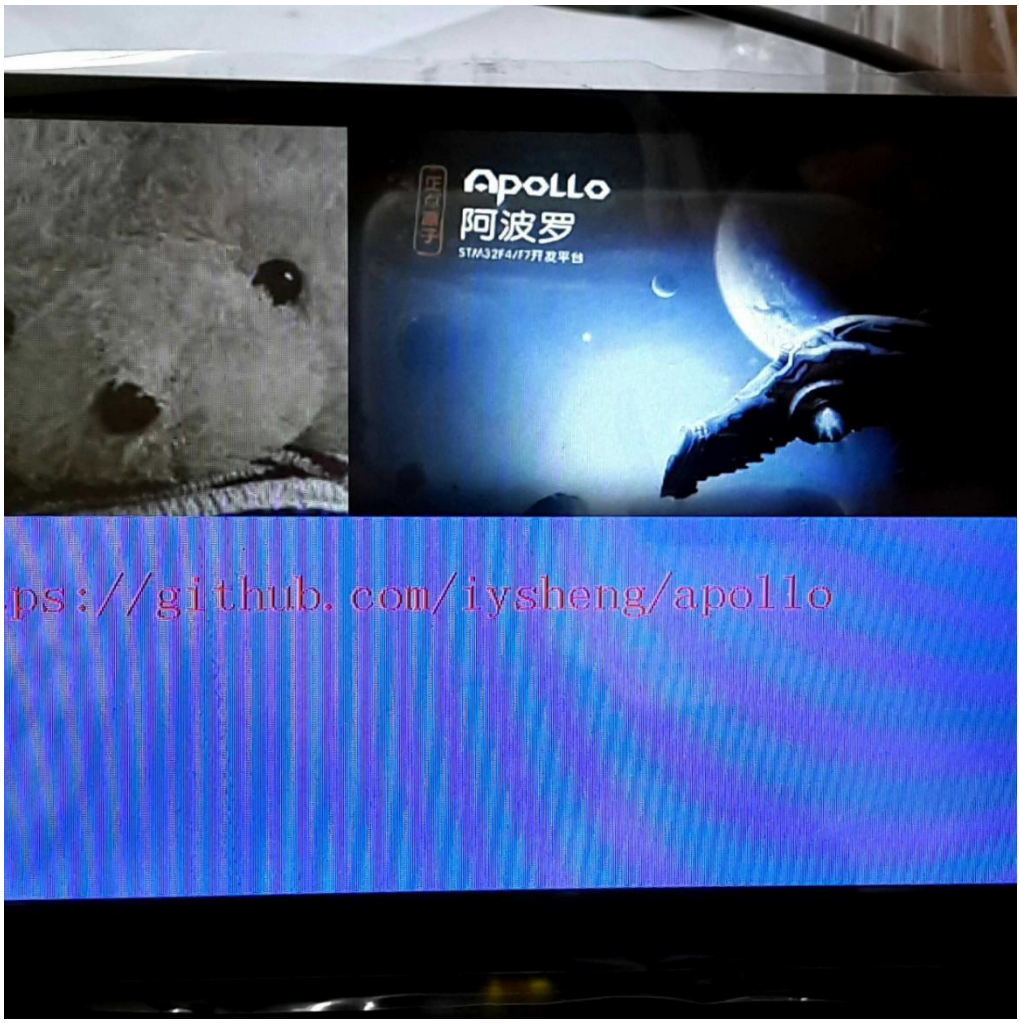
通过 Img2Lcd 软件导出来的。

看下显示效果吧：



特意和我的笔记本作了一个对比，借助了@3guoyangyang7 的创意，在此表示感谢。

再上传一张图片和字符串一起显示的图片：

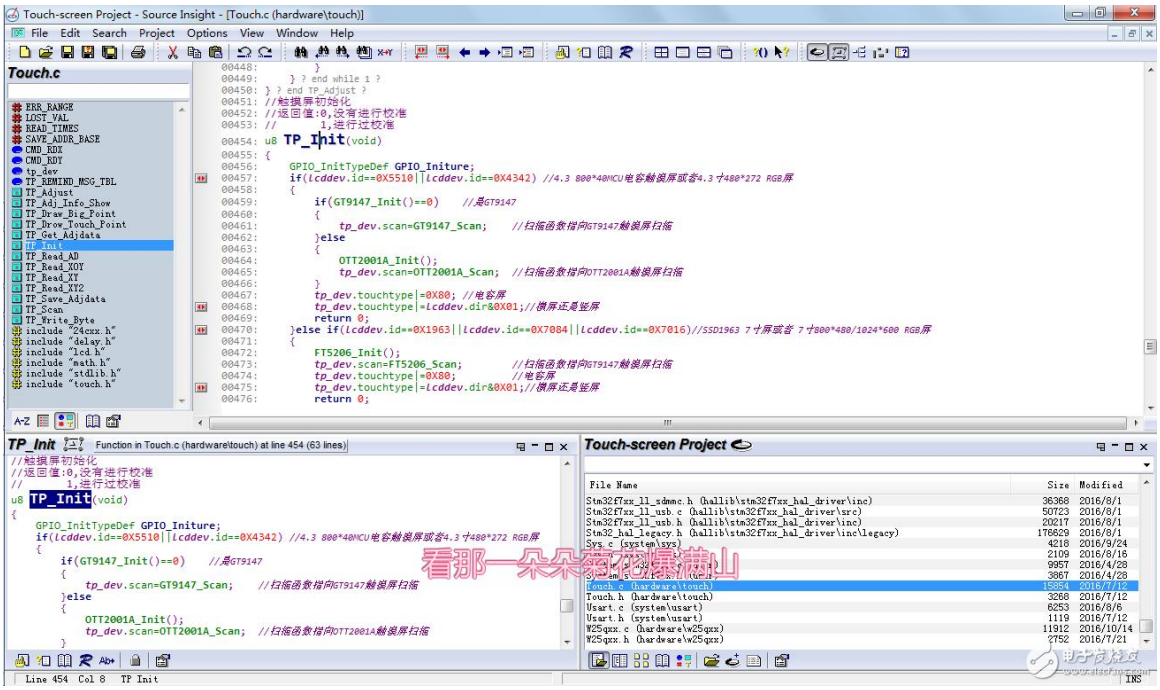


完整的代码在 github 上 https://github.com/iysheng/apollo，欢迎大家 fork。

【阿波罗 STM32F767 试用体验】第六篇 给阿波罗添加电容触摸屏支持

还好我之前研究过一段时间的电容触摸芯片，当时用的 GT 系列的（该系列的生产公司，大家可以去搜索，我是为了避免打广告的嫌疑，嘿嘿），这次阿波罗上的 RGB 屏幕用的电容触摸芯片是 FT 系列的（这是另一个公司的了），然后我就研究了正点原子的代码，研究之后还是比较顺畅的移植到了 IAR 的 IDE 下面，老实说，虽然我没有用 mdk，但是我觉得 IAR 的编译速度还是蛮快的，这点我很欣慰。当初我就是喜欢 IAR 的高速编译，才选择 IAR 的。

说道分析正点原子的代码，我用的工具 sourceinsight，这是一个很好的代码分析工具，在此做下推荐。截图如下，



移植到 IAR 后，我修改正点原子的查询方式读取按键为中断方式，当有触摸按下

时，中断引脚 PH7 会有上升沿中断：

1.3 Interrupt signal from CTPM to Host

As for standard CTPM, host need to use both interrupt control signal and serial data interface to get the touch data. There are two kind of method to use interrupt: interrupt trigger and interrupt query.

Here is the timing to get touch data.

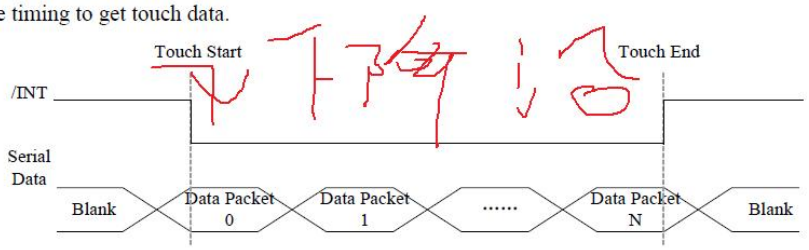
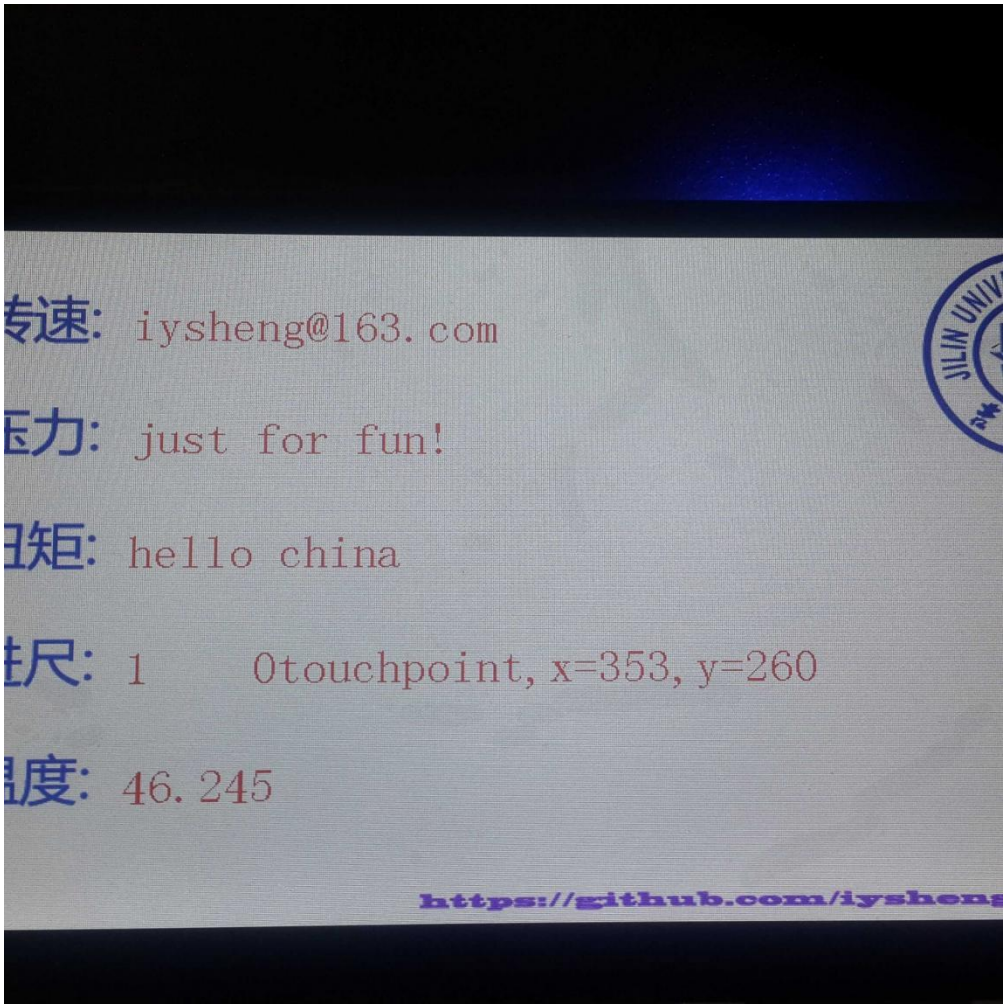


Figure 1-2 Interrupt query mode

这样就可以配置 PH7 上升沿中断，在中断函数中完成按键坐标的读取，该芯片 FT5426，最多支持 5 个点的触摸。我在此作了简单的测试，读取按键值，然后通过串口输出所有的键值。但是只是在屏幕输出最后的一个键值，具体的实验截图如下：

```
CTP ID:1
0touchpoint,x=454,y=322
0touchpoint,x=333,y=216
0touchpoint,x=565,y=337
1touchpoint,x=682,y=352
0touchpoint,x=508,y=0
0touchpoint,x=115,y=0
0touchpoint,x=773,y=463
0touchpoint,x=705,y=325
1touchpoint,x=4095,y=4095
0touchpoint,x=4083,y=259
1touchpoint,x=404,y=381
2touchpoint,x=4095,y=4095
0touchpoint,x=745,y=164
1touchpoint,x=287,y=333
2touchpoint,x=4095,y=4095
3touchpoint,x=4095,y=4095
0touchpoint,x=759,y=160
1touchpoint,x=596,y=63
0touchpoint,x=727,y=267
1touchpoint,x=556,y=104
0touchpoint,x=752,y=176
1touchpoint,x=576,y=74
2touchpoint,x=316,y=144
3touchpoint,x=4095,y=4095
0touchpoint,x=692,y=331
1touchpoint,x=500,y=148
2touchpoint,x=230,y=297
0touchpoint,x=289,y=160
1touchpoint,x=154,y=119
2touchpoint,x=651,y=188
3touchpoint,x=4095,y=4095
0touchpoint,x=491,y=413
1touchpoint,x=4095,y=255
0touchpoint,x=195,y=210
1touchpoint,x=4095,y=4095
0touchpoint,x=148,y=101
1touchpoint,x=266,y=208
2touchpoint,x=323,y=384
0touchpoint,x=656,y=229
1touchpoint,x=538,y=417
2touchpoint,x=4095,y=4095
3touchpoint,x=4095,y=4095
4touchpoint,x=4095,y=4095
0touchpoint,x=665,y=414
```

屏幕的演示效果如下（比较粗糙，现在还处于调试期）：

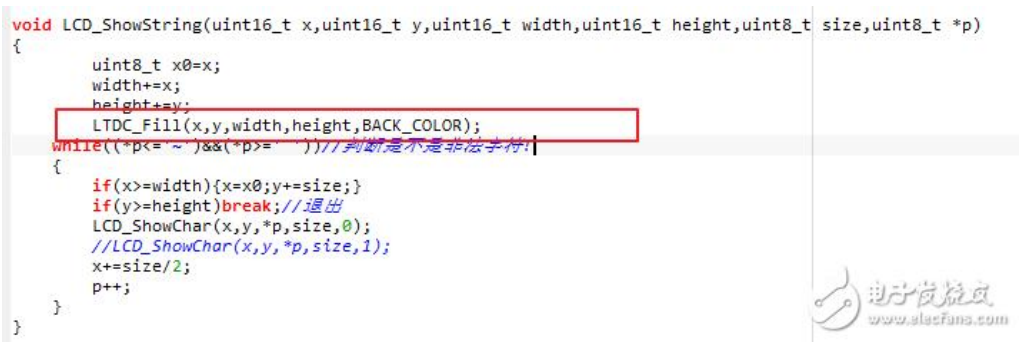


在正点原子的源代码中的显示字符串函数，我发现了一个 bug，就是会留下残影的问题。通过修改代码如下解决这个问题：

```
void LCD_ShowString(uint16_t x,uint16_t y,uint16_t width,uint16_t
height,uint8_t size,uint8_t *p)
{
```

```
        uint8_t x0=x;
        width+=x;
        height+=y;
      LTDC_Fill(x,y,width,height,BACK_COLOR);//显示前先清屏
    while((*p<='~')&&(*p>=' '))//判断是不是非法字符!
    {
      if(x>=width){x=x0;y+=size;}
      if(y>=height)break;//退出
      LCD_ShowChar(x,y,*p,size,0);
      x+=size/2;
      p++;
    }
}
```

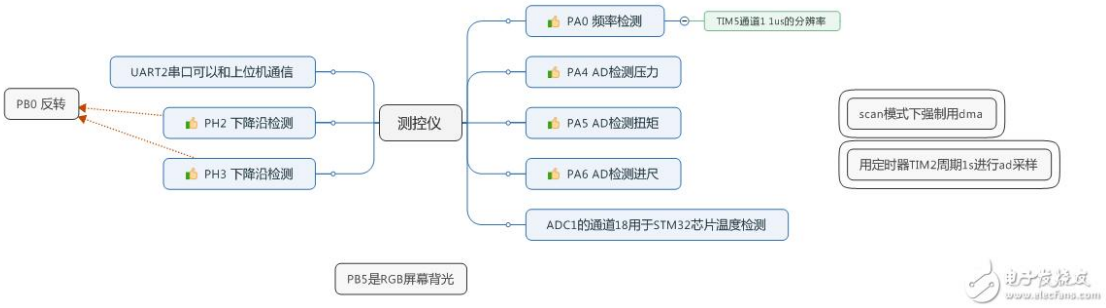
复制代码



【阿波罗 STM32F767 试用体验】第七篇 裸机下的测控仪开发搞一段落，但路还很长

根据自己的试用计划，现在裸机下面开发出测控仪，然后移植到 ucosiii 上，到目前为止，裸机下的开发可以说是高一段落了。

从硬件层面，主要涉及的外设及其框架结构如下所示：



上述外设 PH2、PH3 工作在外部中断模式。UART2 用于和上位机通信工作在非中断模式。PA0 使用定时器 5 的通道 1，PA0 引脚用于输入脉冲捕获，分辨率为 1us。PA4、PA5、PA6 分别工作在 ADC1 的通道 4、5、6，分配在 ADC1 的 rank1、2、3，内部温度 AD 分配在 ADC1 的 rank0 上。一共使用了 4 路 AD 工作在 DMAscan 模式下，借助定时器 2 周期 1s 的进行 ad 准换操作。

除了这些之外，RGB 电容触摸屏通过中断进行读取按键坐标，提升了效率。

```
/*PH7  上升沿  触摸屏*/
void EXti9_5_IRQHandler(void) {
  FT5206_Scan();
```

```
  __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_7);
}
```

复制代码

完整的代码，我都托管到了 github 上，需要的可以自行 fork，我使用的 IDE 是 IAR。

接下来，我会移植该工程到 ucosiii RTOS 上去。路还很长呢，，，

我测试的视频，由于是裸机下的初级阶段，还是比较粗糙的，后续我会不断完善的

【阿波罗 STM32F767 试用体验】第八篇

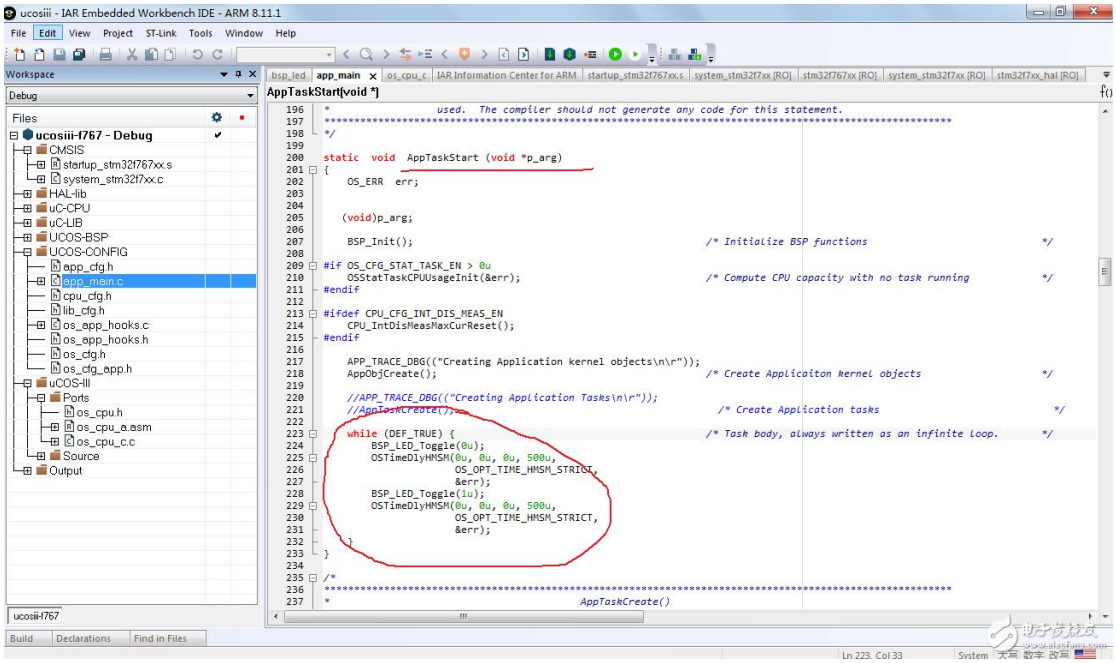
ucosiiiv3.06.00 应该移植成功了

老实说，今天移植 ucosiii，感觉自己也没有做太多的工作，本来是想按照正点原子的步骤来操作的，但是发现有点凌乱，我就自己开始捉摸着移植试一试了。特此记录下我的移植过程，

1。到 Micrium 官网下载源码（需要注册用户，我已经下载下来了，放在附件中了）：

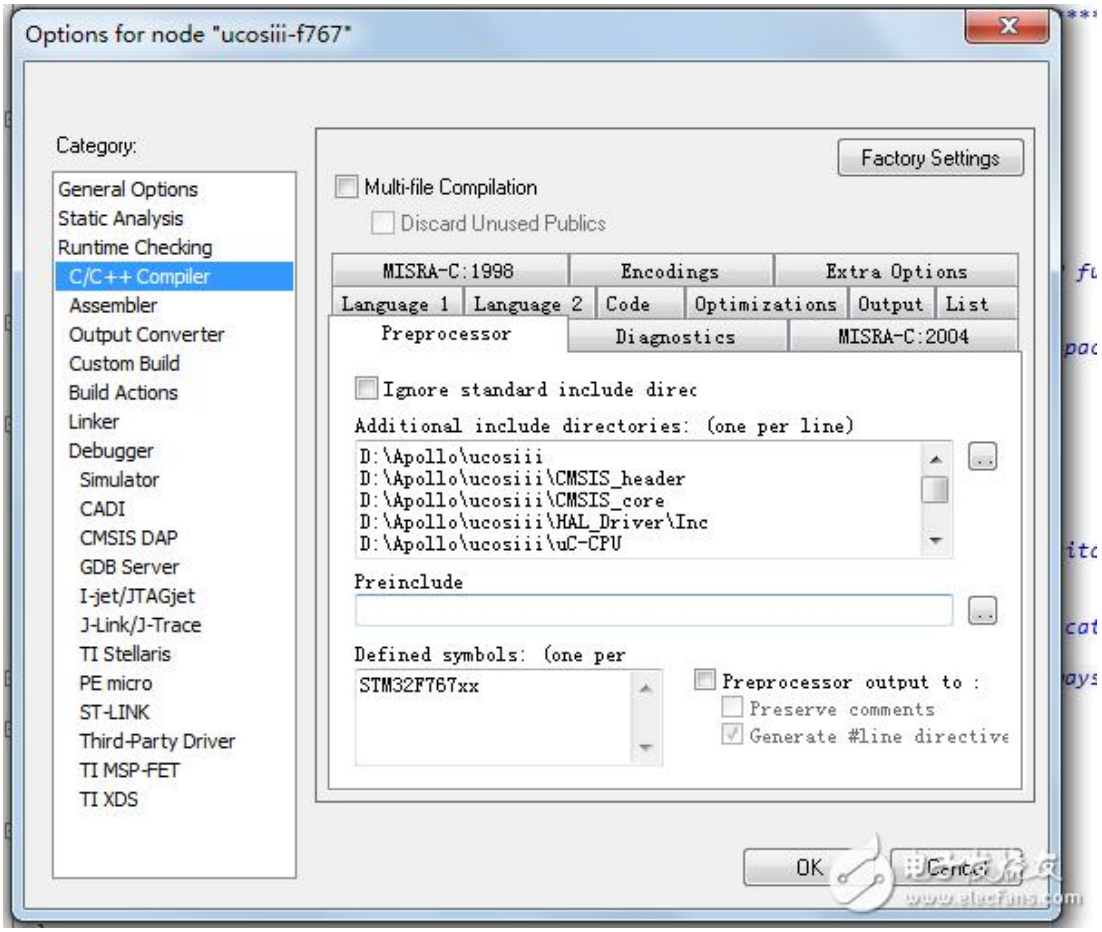
	STM32CubeF7 Library V1.0.1			
STMicroelectronics STM32F7xx	µC/FS SD/MMC storage µC/FS V4.07.03 µC/Clk V3.09.03 µC/OS-II V2.92.12 STM32CubeF7 Library V1.0.1	STM32F746XX-SK	IAR (EWARM) V7.x STM32CubeFx	2016/09/02
STMicroelectronics STM32F7xx	µC/TCP-IP Webserver µC/TCPIP V3.04.00 µC/DHCPc V2.10.00 µC/HTTPs V3.00.01 µC/OS-II V2.92.12 STM32CubeF7 Library V1.0.1	STM32F746XX-SK	IAR (EWARM) V7.x STM32CubeFx	2016/09/02
STMicroelectronics STM32F7xx	µC/OS-II µC/OS-II V2.92.12 STM32CubeF7 Library V1.0.1	STM32F746XX-SK	IAR (EWARM) V7.x STM32CubeFx	2016/09/02
STMicroelectronics STM32F7xx	µC/OS-III µC/OS-III V3.06.00 STM32CubeF7 Library V1.0.4	STM32746G-EVAL2	Atollic TrueSTUDIO V6.x IAR (EWARM) V7.x STM32CubeFx	2016/09/12
STMicroelectronics STM32F7xx	µC/OS-II µC/OS-II V2.92.12 STM32CubeF7 Library V1.0.4	STM32746G-EVAL2	Atollic TrueSTUDIO V6.x IAR (EWARM) V7.x STM32CubeFx	2016/09/12

2。下载下来之后解压缩，我使用的是 IAR 编译器，所以把需要的文件添加到工程里面（这些步骤可以参考正点原子的 STM32F767 UCOS 开发手册_V1.0），为了工程整洁并且有条理，创建一些分组，截图如下：

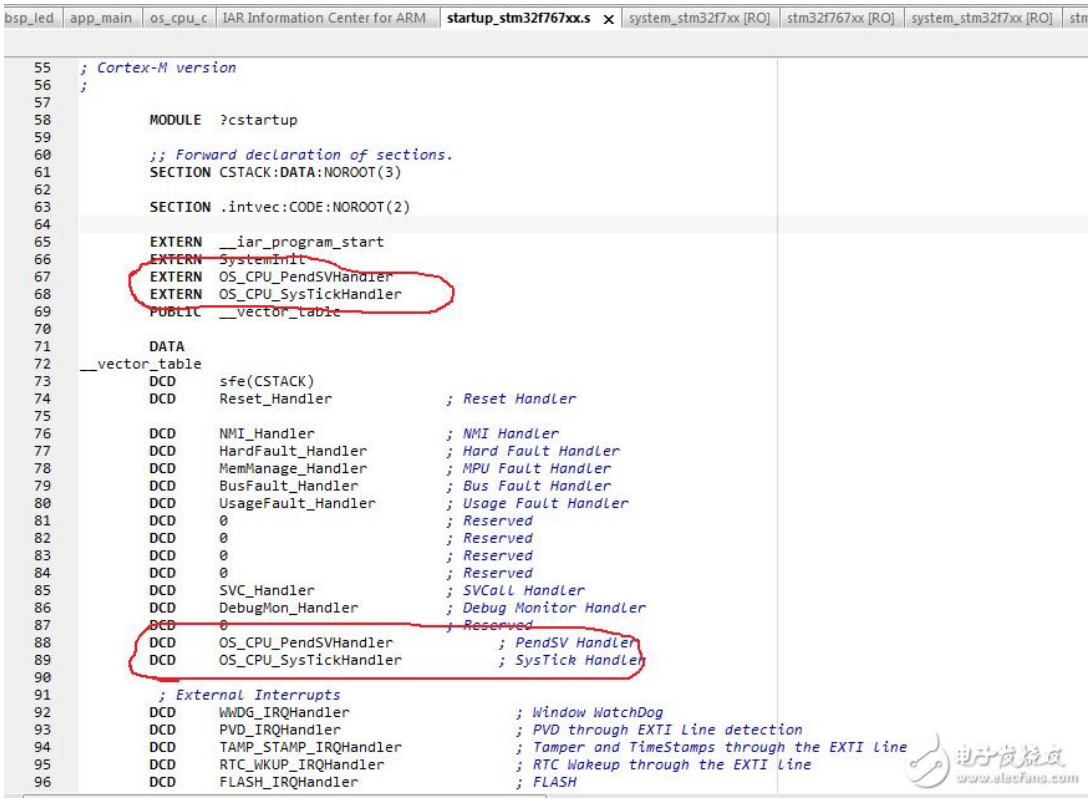


如上图所示，其中 CMSIS 组和 HAL-lib 组，是我添加的 stm32f767 HAL 库中的文件（和我开始裸机下的程序用的是一样的文件）。CMSIS 组是最基本的启动代码部分（有了这两个文件 stm32f767 的裸机工程就可以启动了），HAL-lib 组是 HAL 库文件部分（是为了后期进行开发需要的源程序）。

3。添加 include 路径支持以及 STM32F767 宏定义



4。修改 startup_stm32f767xx.s 文件，更改两个重要的中断处理函数的名字。



```
bsp_led | app_main | os_cpu_c | IAR Information Center for ARM | startup_stm32f767xx.s | system_stm32f7xx [RO] | stm32f767xx [RO] | system_stm32f7xx [RO] | stm

55  ; Cortex-M version
56  ;
57
58          MODULE  ?cstartup
59
60          ;; Forward declaration of sections.
61          SECTION CSTACK:DATA:NOROOT(3)
62
63          SECTION .intvec:CODE:NOROOT(2)
64
65          EXTERN  __iar_program_start
66          EXTERN  SystemInit
67          EXTERN  OS_CPU_PendSVHandler
68          EXTERN  OS_CPU_SysTickHandler
69          PUBLIC  __vector_table
70
71          DATA
72  __vector_table
73          DCD     sfe(CSTACK)
74          DCD     Reset_Handler             ; Reset Handler
75
76          DCD     NMI_Handler               ; NMI Handler
77          DCD     HardFault_Handler         ; Hard Fault Handler
78          DCD     MemManage_Handler         ; MPU Fault Handler
79          DCD     BusFault_Handler          ; Bus Fault Handler
80          DCD     UsageFault_Handler        ; Usage Fault Handler
81          DCD     0                         ; Reserved
82          DCD     0                         ; Reserved
83          DCD     0                         ; Reserved
84          DCD     0                         ; Reserved
85          DCD     SVC_Handler               ; SVCall Handler
86          DCD     DebugMon_Handler          ; Debug Monitor Handler
87          DCD     0                         ; Reserved
88          DCD     OS_CPU_PendSVHandler            ; PendSV Handler
89          DCD     OS_CPU_SysTickHandler           ; SysTick Handler
90
91           ; External Interrupts
92          DCD     WWDG_IRQHandler                 ; Window WatchDog
93          DCD     PVD_IRQHandler                  ; PVD through EXTI Line detection
94          DCD     TAMP_STAMP_IRQHandler           ; Tamper and TimeStamps through the EXTI line
95          DCD     RTC_WKUP_IRQHandler             ; RTC Wakeup through the EXTI line
96          DCD     FLASH_IRQHandler                ; FLASH
```

至此，ucosiii 的系统移植就已经完成了。

5。接下来就是修改用户代码，来使用 led 灯程序进行测试 ucosiii。由于阿波罗上面的 DS0 和 DS1 使用的是 PB1 和 PB0

。而原工程中使用的是 PF10。修改的关键代码示例：

```
void  BSP_LED_Toggle (CPU_INT08U  led)
{
    switch (led) {
        case 0u:HAL_GPIO_TogglePin(GPIOB, BSP_LED0_GPIOF_PIN);break;
        case 1u:HAL_GPIO_TogglePin(GPIOB, BSP_LED1_GPIOF_PIN);break;
        default:HAL_GPIO_TogglePin(GPIOF, BSP_LED1_GPIOF_PIN);break;
```

```
    }
}
```

复制代码

```
/*
*********************************************************************************************************
*************************
*                                            DEFINES
*********************************************************************************************************
*************************
*/

#define  BSP_LED0_GPIOF_PIN                     DEF_BIT_00
#define  BSP_LED1_GPIOF_PIN                     DEF_BIT_01
```

复制代码

上述修改的关于 led 的程序都是 bsp_led.c 文件中的内容。

整个完整的阿波罗 ucosiii 的代码，我上传到了一个新的仓库中：

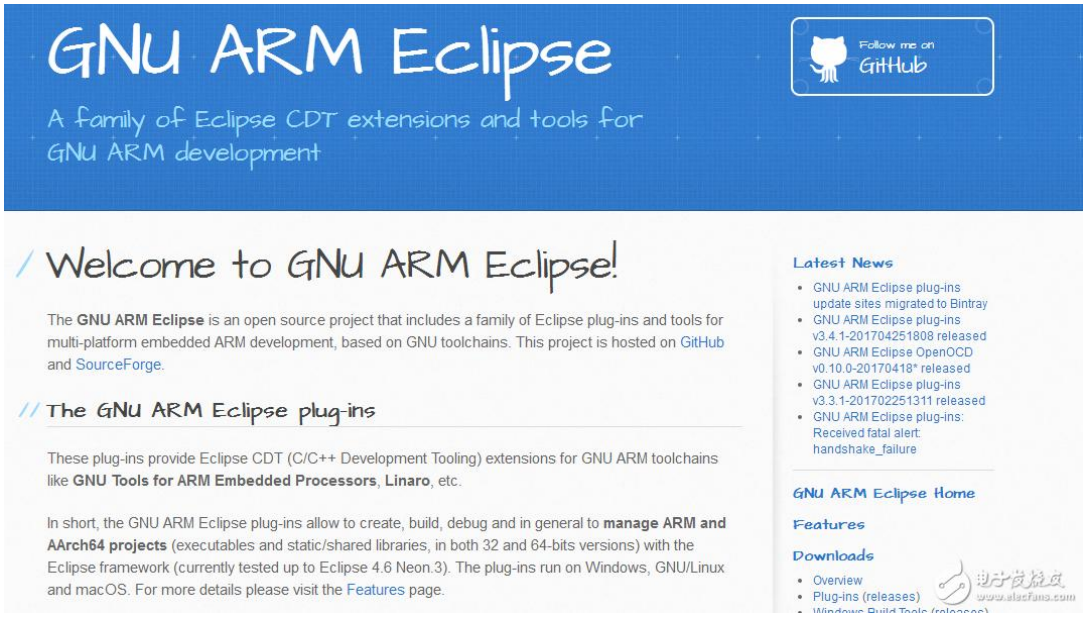
https://github.com/iysheng/ucosiii-f767

如果需要欢迎大家 fork。

【阿波罗 STM32F767 试用体验】第九篇 搭建 eclipse 的开发环境并完成 led 测试

一直想搭建比较流行的免费的开发环境,在了解 eclipse 很流行之后,观望了很久,于昨天晚上开始搭建环境,中间走了很多坑,功夫不负有心份,环境终于还是搭建好了,特此记录下来避免后来人再踩坑.搭建 eclipse 环境最好的手册就是官方指导网站,

http://gnuarmeclipse.github.io/

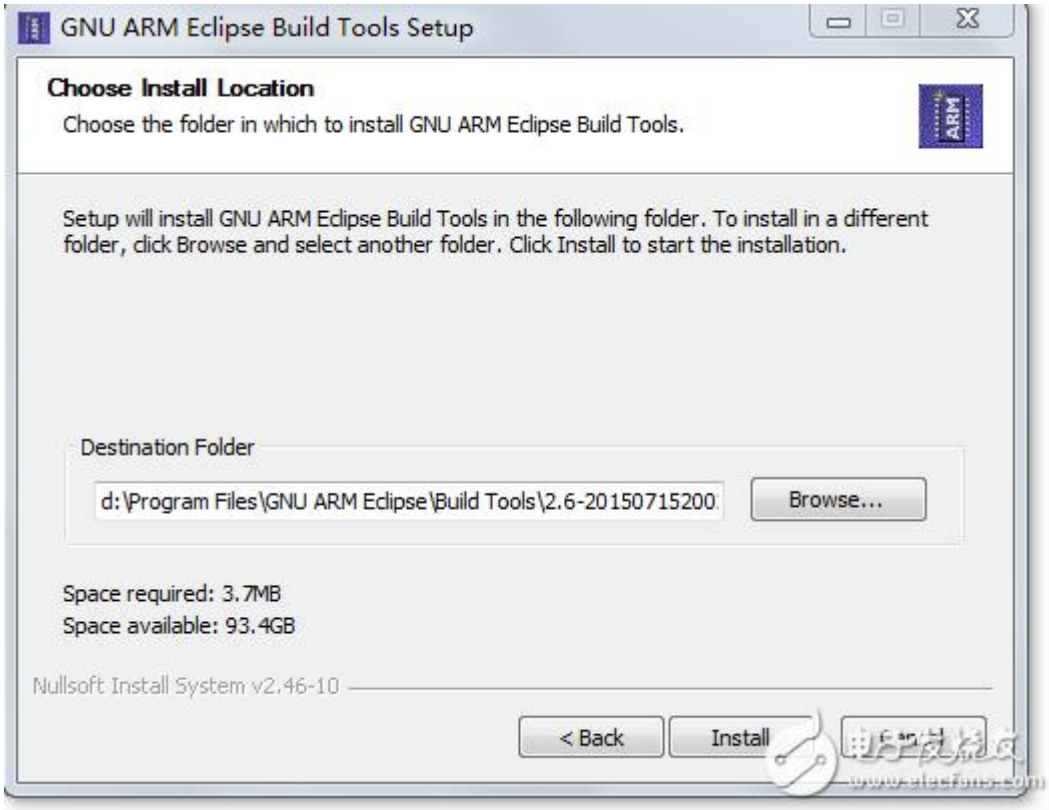


接下来讲述我的安装过程.首先,我列举一个提纲,搭建给予 eclipse 的开发环境需

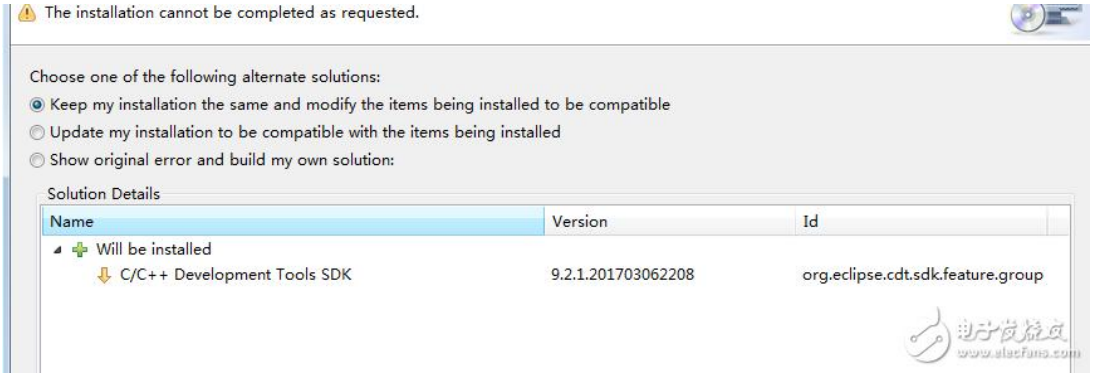
要安装的软件包：

然后我就截图记录一些安装的细节：

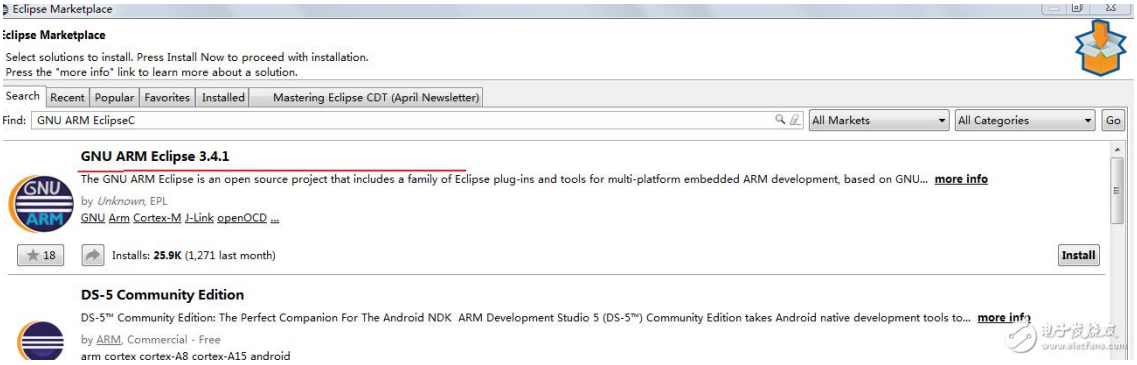
安装编译工具 make\echo 等小工具包



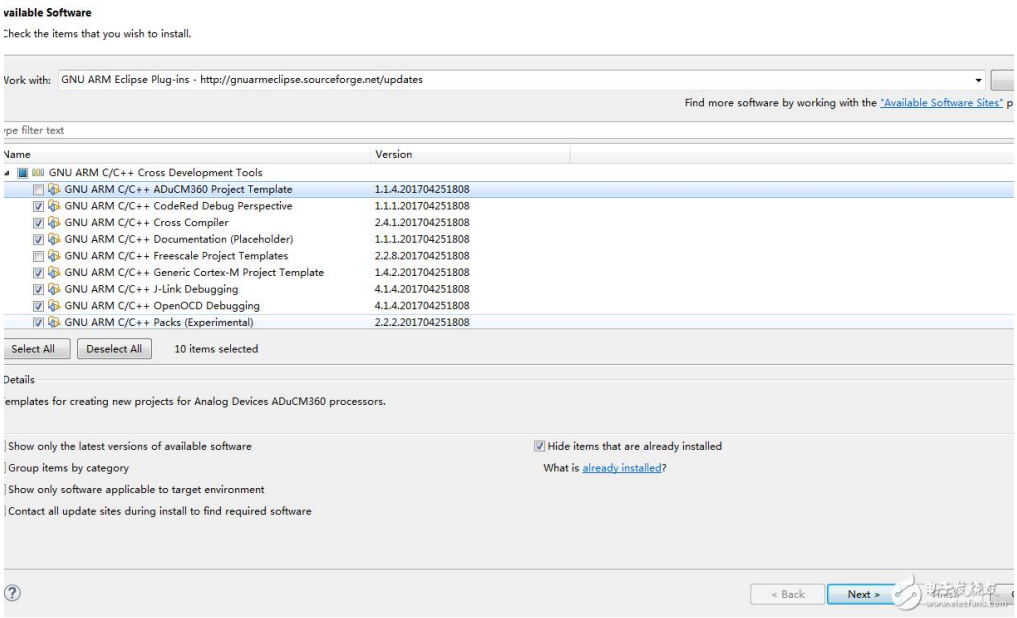
安装 CDT，选择 SDK。



安装 GNU ARM plug-ins for Eclipse



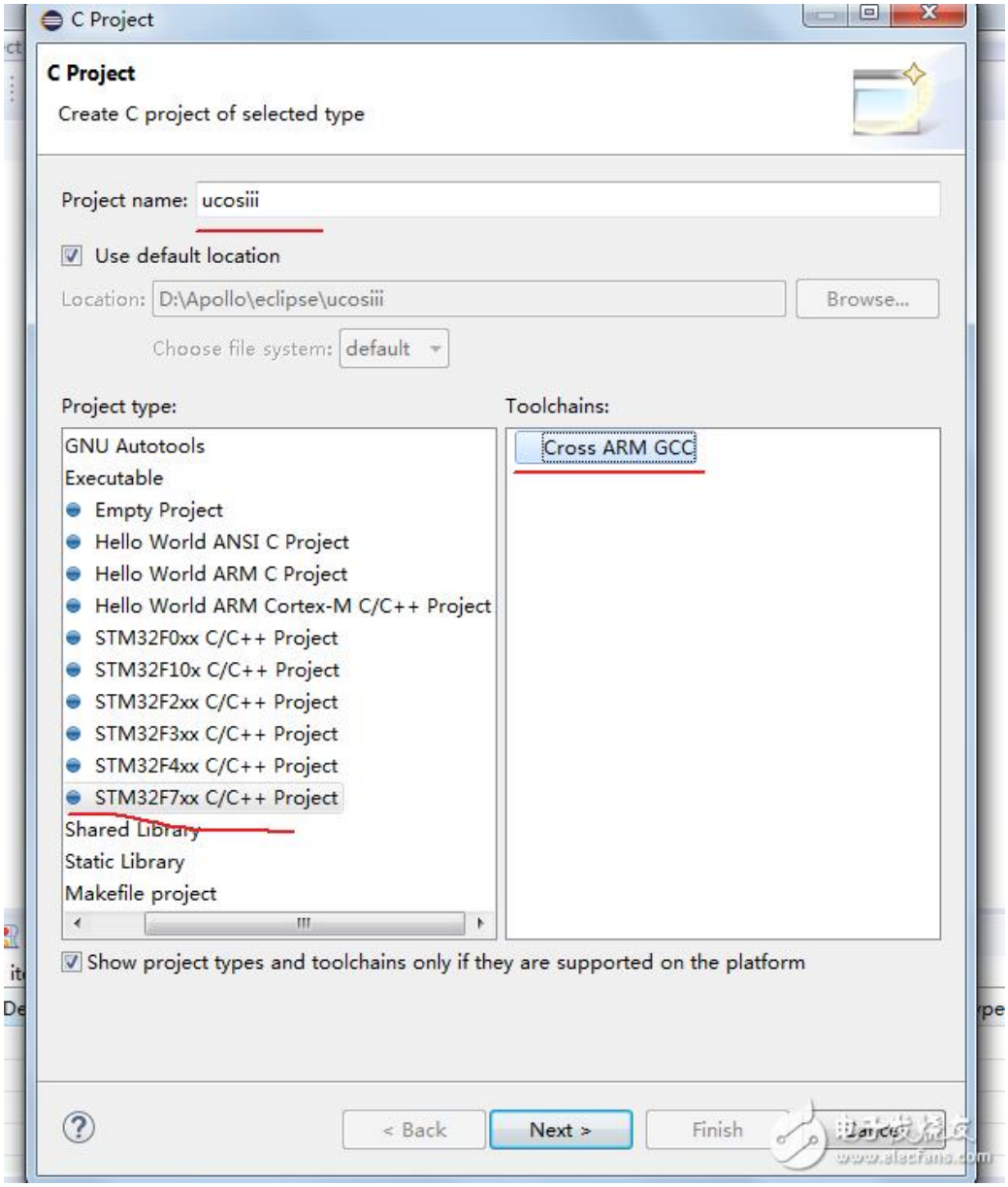
可以选择配置如下(默认的就好):



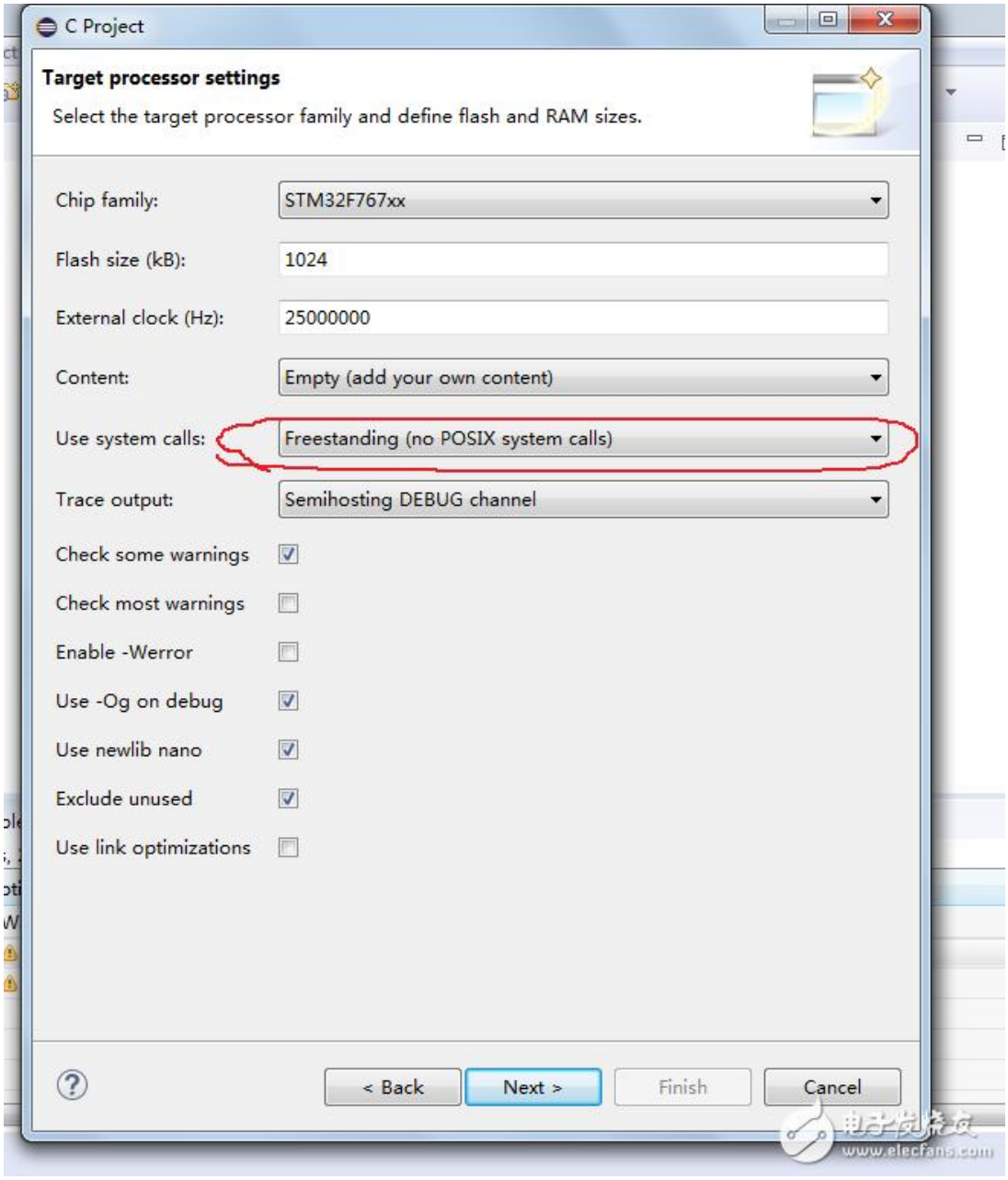
就简单的截图几张记录下吧,都是基本的安装软件,难度应该不大.

重要的是在创建工程之后,我们需要作一些函数上的修改.

首先创建一个基于 STM32f7x 系列的空工程

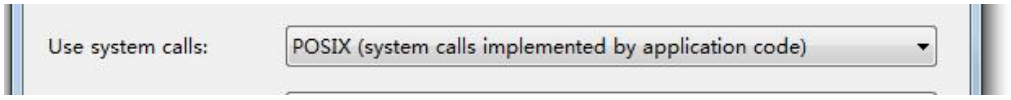


然后，选择

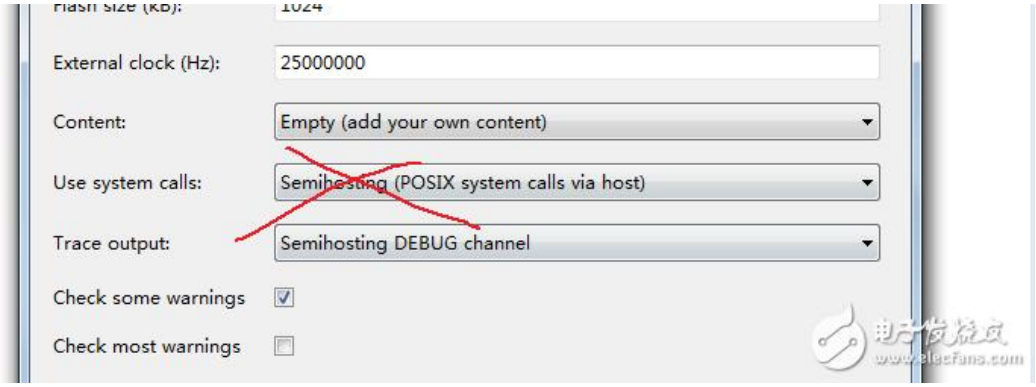

C Project
Target processor settings
Select the target processor family and define flash and RAM sizes.
Chip family:
STM32F767xx
Flash size (kB):
1024
External clock (Hz):
25000000
Content:
Empty (add your own content)
Use system calls:
Freestanding (no POSIX system calls)
Trace output:
Semihosting DEBUG channel
Check some warnings
Check most warnings
Enable -Werror
Use -Og on debug
Use newlib nano
Exclude unused
Use link optimizations
< Back
Next >
Finish
Cancel

或

者


Use system calls:
POSIX (system calls implemented by application code)

好像都可以。但是我尝试选择

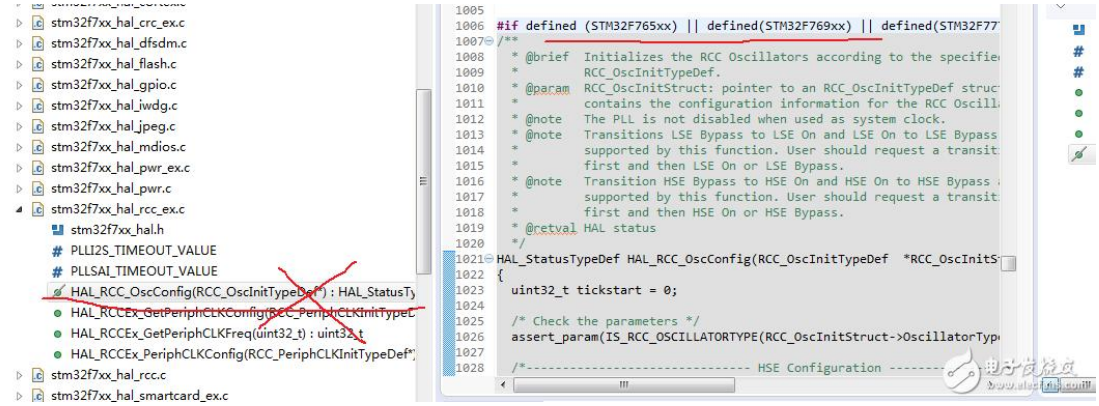


编译的程序无法运行。

然后就一直到 finish 了。

然后我们需要修改一个 RCC 模块的函数，否则系统无法启动。

主要原因是新创建的空函数模板中



文件 stm32f7xx_hal_rcc_ex.c

函数 HAL_StatusTypeDef

HAL_RCC_OscConfig(RCC_OscInitTypeDef *RCC_OscInitStruct)不能很好的

试配 stm32f767igt,修改如下内容

#if defined (STM32F765xx) || defined(STM32F769xx) ||

defined(STM32F777xx) || defined(STM32F779xx)

将该函数注释掉，因为我们定义了宏 STM32F767xx，将其从上述预处理语句中去掉就可以了，

接着为了使 systick 正常工作，我们需要修改

_initialize_hardware.c 文件中的函数

```
1.  // Disable when using RTOSes, since they have their own handler.
2.  #if 1
3.
4.  // This is a sample SysTick handler, use it if you need HAL timings.
5.  void __attribute__ ((section(".after_vectors")))
6.  SysTick_Handler(void)
7.   {
8.  #if defined(USE_HAL_DRIVER)
9.      HAL_IncTick();
10. #endif
11.  }
12.
13. #endif
```

复制代码

将原本的 if 0 改为 if 1。这样就可以测试 led 灯闪烁程序了。

关键的 main 代码为

```
1.  #include "stm32f7xx_hal.h"
```

复制代码

后续有空，我会看看在 eclipse 下关于在线调试硬件部分了。

【阿波罗 STM32F767 试用体验】第十篇 eclipse 下 egit 插件的使用简述

我从上周六开始尝试在 eclipse 下搭建 **STM32**f7 的开发环境，用了两天的时间，搭建完成，程序也可以编译下载了。并且简单的小程序也可以正常的调试了。当时我感觉很高兴，一个原因是 eclipse 是一个 free tool，并且它还可以安装各种各样的插件进行开发，来扩展 eclipse 的功能。并且在搭建开发环境的过程中，也锻炼了自己的对于程序编译、汇编、链接的了解。但是当我把自己在 IAR 下的整个完整的工程移植到 eclipse 下的时候，事情开始出现了意想不到的变化，出现了很多问题：第一，由于使用到了外部 SRAM，所以我需要把程序连接到 0xc0000000，外部 sdram 空间，在 IAR 下可以一句话解决这个问题：

```
1.  __root uint16_t ltdc_lcd_framebuf[1280][800] @ 0xc0000000;
```

复制代码

到了 gnu 下就不是这么简单了，这个解决办法是：

首先定义二维数组如下：

```
1.
2.  uint16_t                    __attribute__((section(".sdram_section")))
    ltdc_lcd_framebuf[1280][800];
```

复制代码

然后再链接文件中显式声明该段：

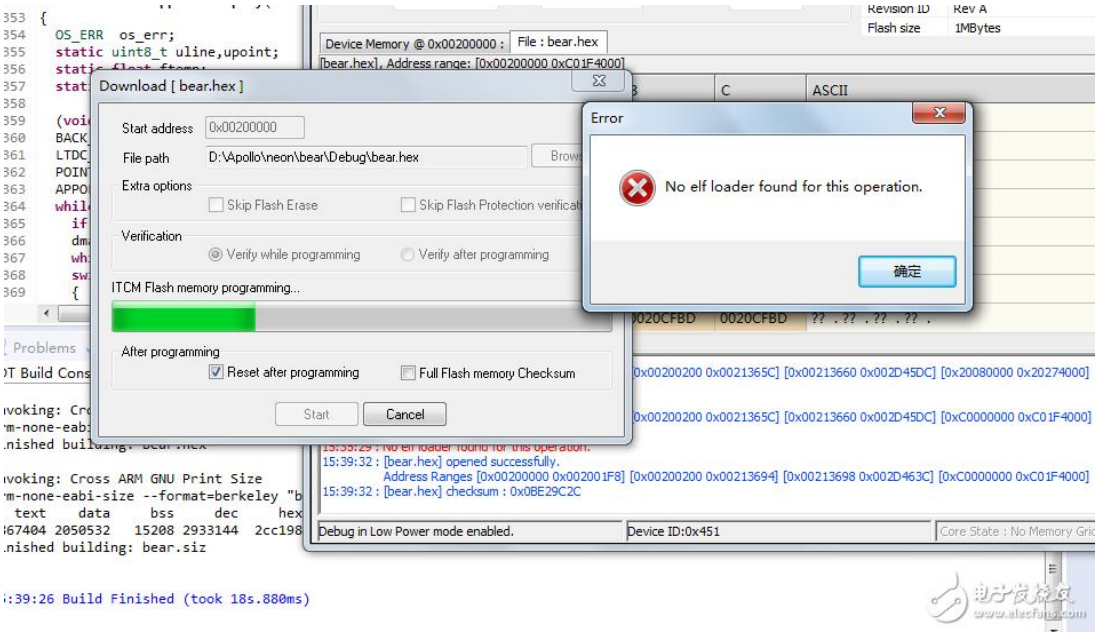
```
1.  MEMORY
2.  {
3.  FLASH (rx)      : ORIGIN = 0x00200000, LENGTH = 2048K
4.  RAM (xrw)       : ORIGIN = 0x20000000, LENGTH = 512K
5.  SDRAM (xrw)      : ORIGIN = 0xC0000000, LENGTH = 32M
6.  }
```

复制代码

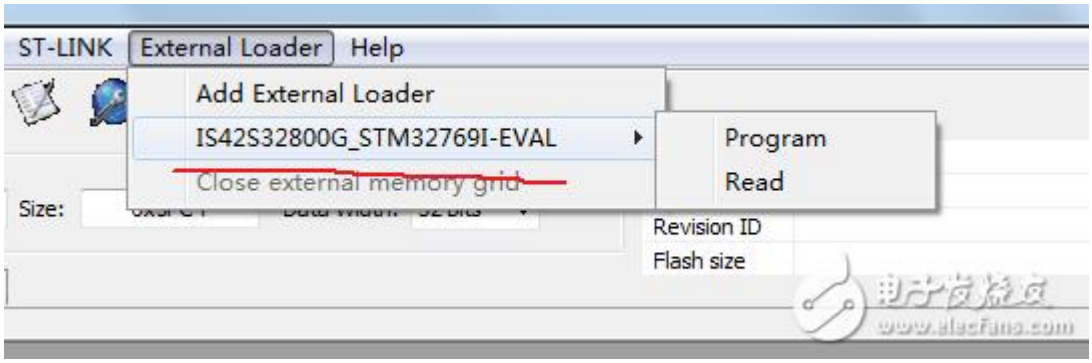
```
{
*(.sdram_section);
} > SDRAM
```

这个编译问题，解决之后，我就用 stlink 下载程序，然后错误又来

了：



然后，我尝试自己编译阿波罗的 stlink extern loader，但是自己不才，没有成功。后来发现可以使用如下的 extern loader,这是 stlink 自带的，阿波罗可以使用。



最后最大的问题来了，当我移植完整的程序到阿波罗之后，发现 debug 开始出现问题了，就是这个不能 debug 的问题困扰了我到现在，谁让我是一个强迫症患者阿，这个问题不解决怎么能行呢，我开始怀疑一些可能出现的问题，但是到

现在都没有找到，中途，我也尝试了一个 IDE TRUESTUDIO。这个是基于 eclipse 的，由 free 版本的，我看他的调试使用的是 stlink 接口，而不是 openocd 接口，我在 iar 下完整的程序移植到 truestudio 后，可以正常调试，虽然调试之前，要经过很长时间才能把程序下载到 flash，hex 文件大概 7M 左右。但是可以调试，我本来就像屈服与 struestududio 了。虽然 lite 版本每次都有 5s 的广告时间。但是有一个问题是 lite 版本很不灵活，sprintf 函数无法读入 float 型的变量，这让我很无奈，索性，还是回到 eclispe 把，我准备勉强暂时先接受 eclipse 无法调试的大型程序的问题了。于是目前我又回到解放前了，刚才 eclispe 下建立起阿波罗 ucosiii 的支持。

代码我托管到新的仓库了：

https://github.com/iysheng/pig.git

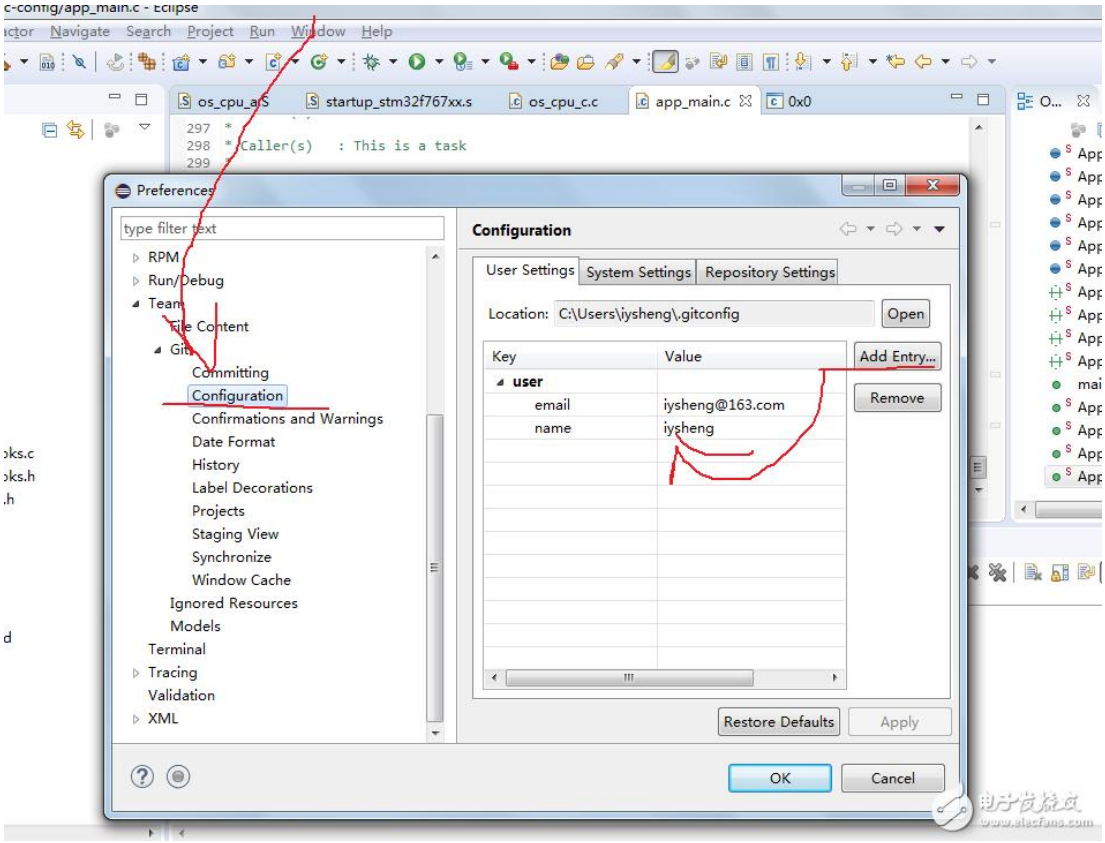
就使用 pig 以纪念我的愚蠢。

但是这个帖子只写这些有点太少内容了，所以，我就记录下我在 eclipse 使用 github 的经历吧。

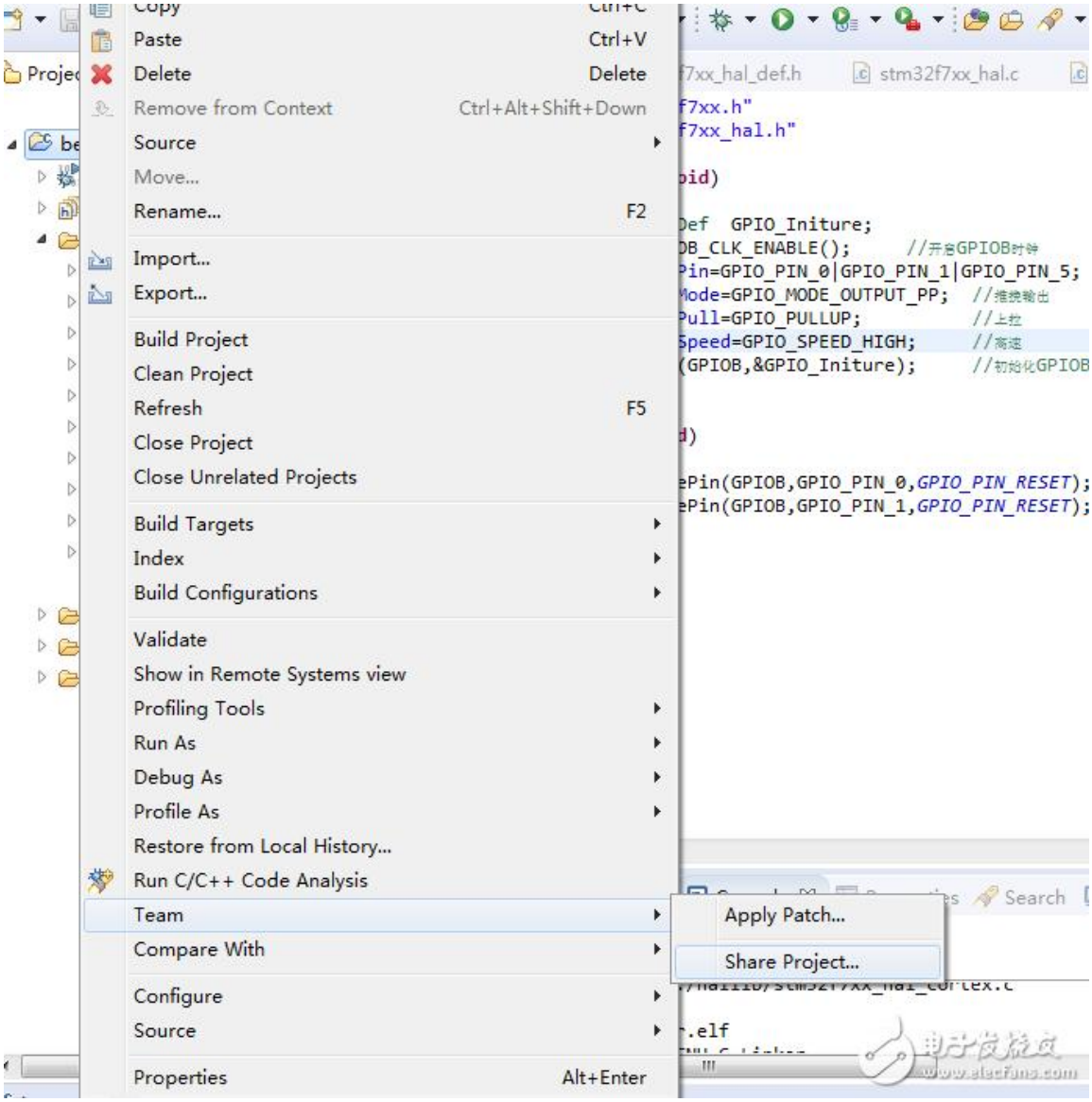
我使用的是 egit 插件，他的 wiki

https://wiki.eclipse.org/EGit/Us ... _to_version_control

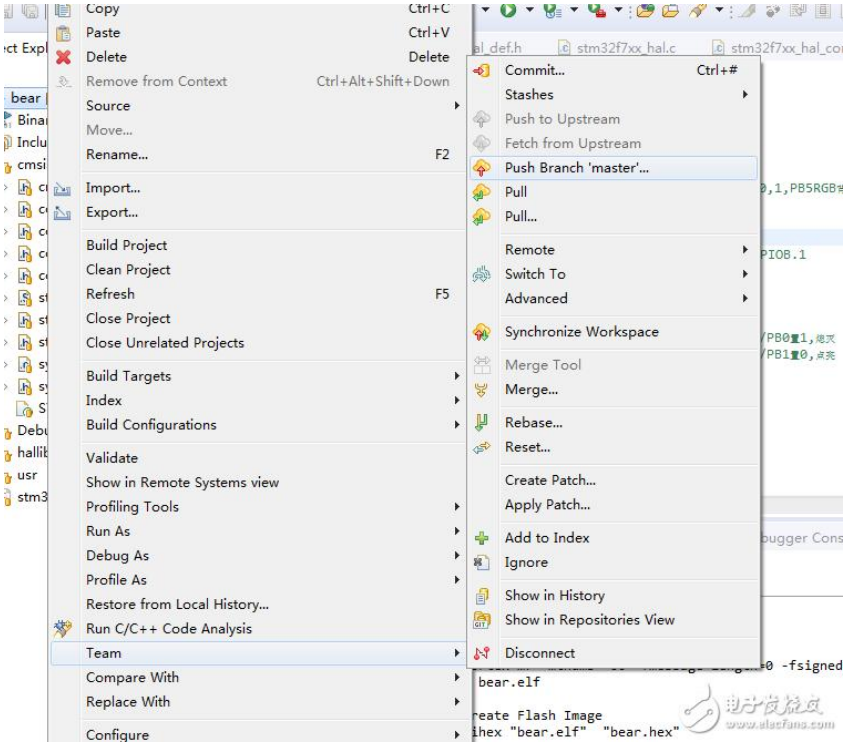
首先，创建一个帐号


Preferences
type filter text
RPM
Run/Debug
Team
File Content
Git
Committing
Configuration
Confirmations and Warnings
Date Format
History
Label Decorations
Projects
Staging View
Synchronize
Window Cache
Ignored Resources
Models
Terminal
Tracing
Validation
XML
Configuration
User Settings
System Settings
Repository Settings
Location: C:\Users\iysheng\.gitconfig
Open
Key
Value
Add Entry...
Remove
user
email
iysheng@163.com
name
iysheng
Restore Defaults
Apply
OK
Cancel

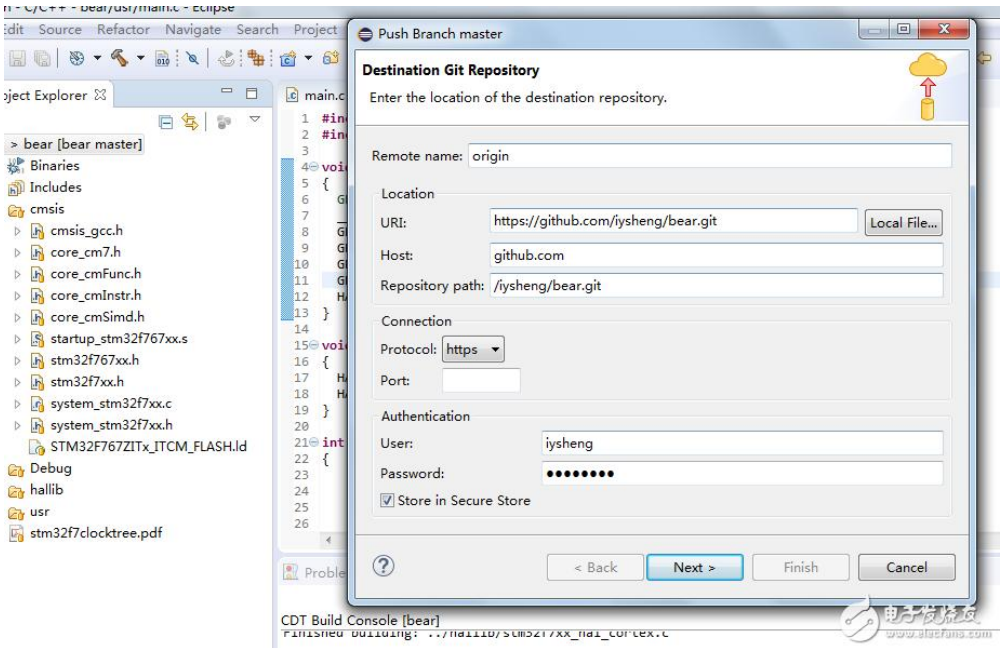
然后，选择自己的工程进行分享



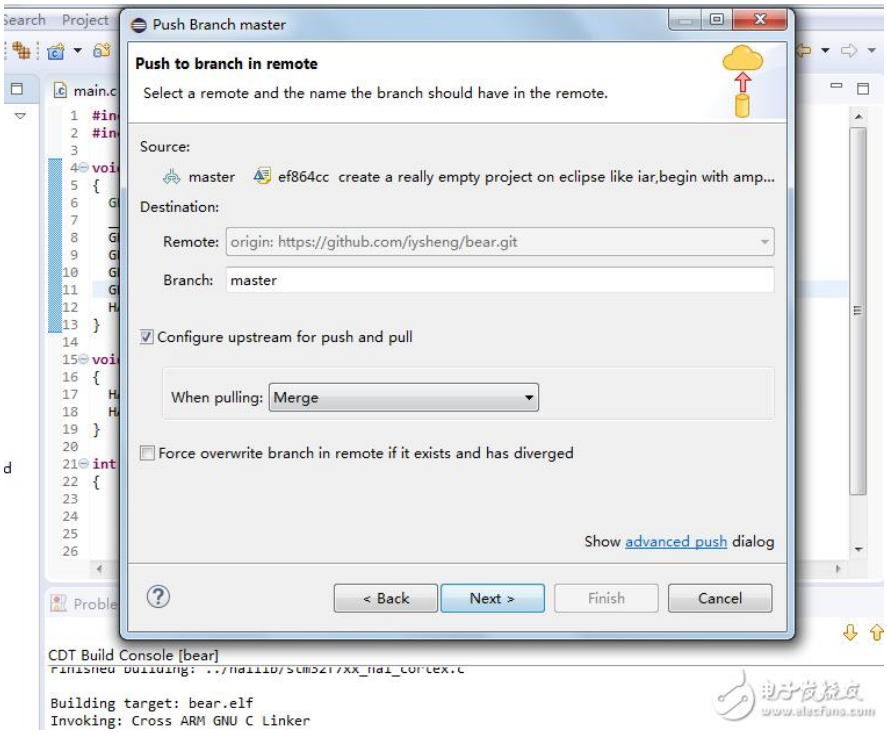
接着，就可以推送，commit 等操作了

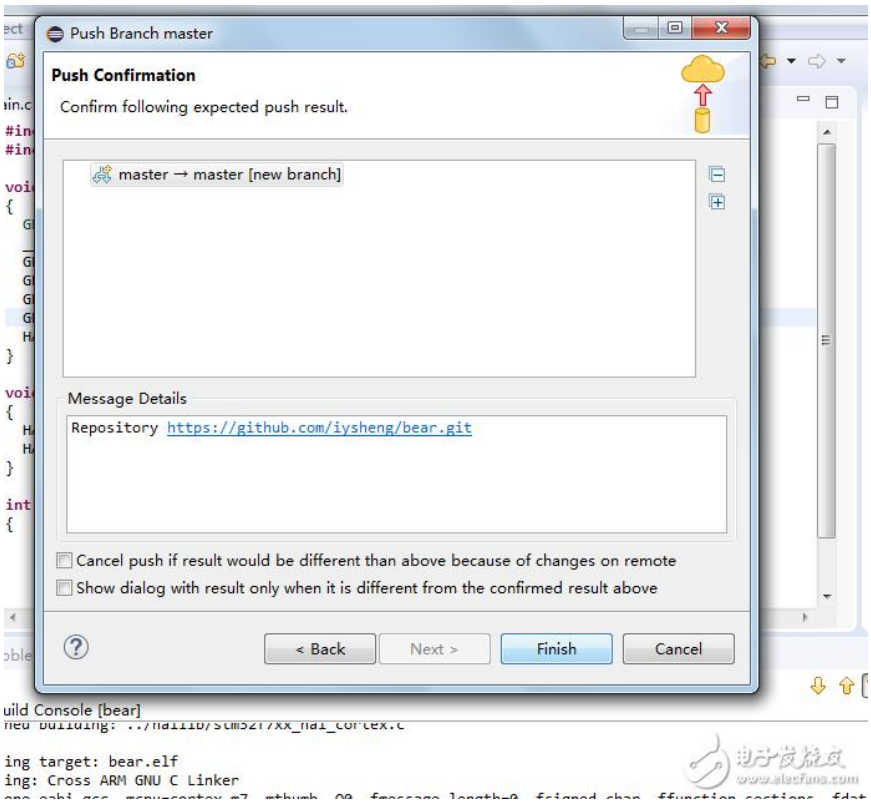


输入仓库地址

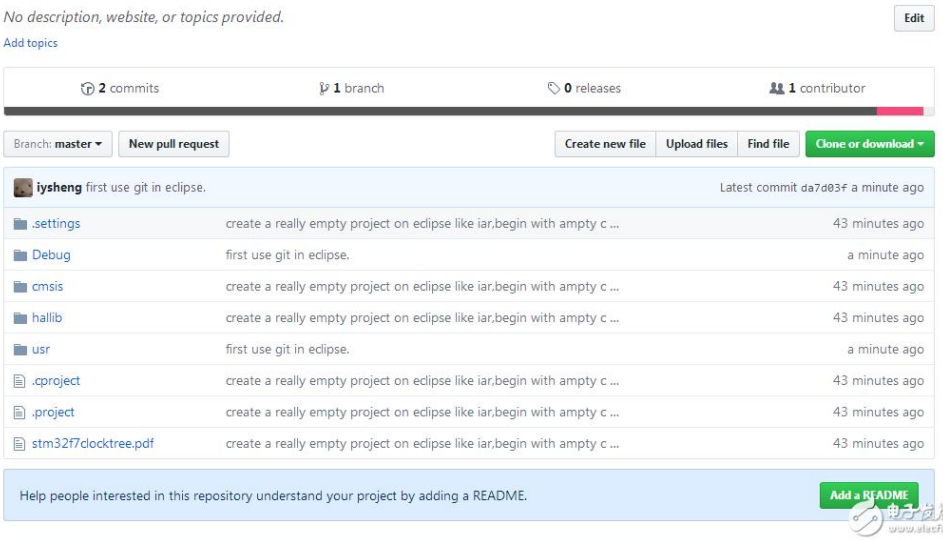


选择分支





然后看下 github



这样就不用每次，都在命令行下和 github 交互了，尽管我经历了这么的问题，但是我还是支持 eclipse 的。付出总会有回报的。

【阿波罗 STM32F767 试用体验】第十一篇 优化代码，在 ucosiii 下,实现同步信号亮.

考虑到 rtos 的特点,既然已经使用了 ucosiii,那么肯定要借助她的功能,为了实现,系统的多任务实时响应,今天完成了优化之前的多任务下的轮训操作,改为任务信号量同步.本次,主要有三个任务.

task0:dma 下 4 路 adc 检测,

task1:pwm 检测获取输入频率

task2:获取 rgb 屏幕按键中断

3 个任务都在等待 task 信号量,这里有三个中断分别给他们发送 task 信号量.

分别是

```
1.  void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
2.  {
3.      static OS_ERR err;
4.    if(hadc->Instance==ADC1)
5.    {
6.      OSTaskSemPost(&AppTaskObj0TCB,OS_OPT_POST_NONE,&err)
    ;
7.    }
```

```
}

/*输入捕获中断的回调函数*/
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
 static OS_ERR err;
 if(htim->Instance==TIM5)
 {
    switch(ic_state)
    {
                                                    case 0x00:ic_state=0x40;ic_value=0x00;__HAL_TIM_DISABLE(htim);__HAL_TIM_SET_COUNTER(htim,0);__HAL_TIM_ENABLE(htim);break;// 捕获第一个中断
                                                    case 0x40:ic_state|=0x80;ic_value=HAL_TIM_ReadCapturedValue(htim,TIM_CHANNEL_1);
     OSTaskSemPost(&AppTaskObj1TCB,OS_OPT_POST_NONE,&err);break;//捕获第二个中断，完成一次周期捕获
    default:break;
    }
 }
```

```
}

void EXTI9_5_IRQHandler(void) {
 static OS_ERR err;
 OSIntEnter();
 OSTaskSemPost(&AppTaskObj2TCB,OS_OPT_POST_NONE,&err);
 __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_7);
 OSIntExit();
}
```

复制代码

而三个任务等待信号量的操作如下：

```
while (DEF_TRUE) {
        BSP_LED_Off(0);
        OSTaskSemPend (
                100,
                        OS_OPT_PEND_BLOCKING,
                        NULL,
                        &os_err);
        if(os_err==OS_ERR_NONE)
```

```
9.          {
10.         while(uline<4){
11.         switch(uline)
12.         {
13.                                     case    0:{sprintf((char
    *)rstr,"PA4:%4dKg",raw_icekong[1]);break;}
14.                                     case    1:{sprintf((char
    *)rstr,"PA5:%4dN.m",raw_icekong[2]);break;}
15.                                     case    2:{sprintf((char
    *)rstr,"PA6:%4dcm",raw_icekong[3]);break;}
16.                                                 case
    3:{uitemp=raw_icekong[0];ftemp=((float)uitemp)/4095*3300;ftemp
    =((ftemp-760.0)/2.5)+25;
17.                 sprintf((char *)rstr,"%0.3f",ftemp);break;}
18.         default:break;
19.         }
20.          LCD_ShowString(120,130+uline*80,strlen(rstr)*16,32,32,(u
    int8_t *)rstr);
21.         //printf("%s\r\n",rstr);
22.         uline++;
23.         }
24.         uline=0;
```

```
25.         }
26.             APP_TRACE_INFO(("Object test task 0 running ....\r\n"));
27.     }
28.
29. while (DEF_TRUE) {
30.             OSTaskSemPend (
31.                             100,
32.                                     OS_OPT_PEND_BLOCKING,
33.                                     NULL,
34.                                     &os_err);
35.             if(os_err==OS_ERR_NONE)
36.             {
37.                         ic_state&=0x3f;
38.                         hole_ic_value=ic_state*(0xffffffff);
39.                         hole_ic_value+=ic_value;
40.                         ic_value=hole_ic_value/1000;
41.                                                     sprintf((char
    *)rstr,"PWM:%6dms...%9lldus",(int)ic_value,(long                long
    int)hole_ic_value);
42.                          LCD_ShowString(120,50,strlen(rstr)*16,32,32,(ui
    nt8_t *)rstr);
43.                         printf("%s\r\n",rstr);
```

```
                    ic_state=0x00;
            }
        APP_TRACE_INFO(("Object test task 1 running ....\r\n"));
    }

while (DEF_TRUE) {
        OSTaskSemPend (100,
                                    OS_OPT_PEND_BLOCKING,
                                    NULL,
                                    &os_err);
        if(os_err==OS_ERR_NONE)
        {
        BSP_LED_On(0);
        FT5206_Scan();
        if(tp_dev.sta!=0)
                        {
                        tp_dev.sta&=0x1f;
                        while(tp_dev.sta&0x01){
                          upoint++;
                          tp_dev.sta>>=1;
                        }
```

```
65.                         for(tp_dev.sta=0;tp_dev.sta<upoint;tp_dev.sta++)
66.                         {
67.                                                         sprintf((char*)rstr,"%dtouchpoint,x=%3d,y=%3d",tp_dev.sta,(uint16_t)tp_dev.x[tp_dev.sta],(uint16_t)tp_dev.y[tp_dev.sta]);
68.                         if((tp_dev.x[tp_dev.sta]<800)&&(tp_dev.y[tp_dev.sta]<480))
69.                         {
70.                           //printf("%s\r\n",rstr);
71.                             LCD_ShowString(300,50+3*80,32*16,32,32,(uint8_t *)rstr);
72.                         }
73.                         }
74.                         upoint=0;
75.                         tp_dev.sta=0;
76.                       }
77.         }
78.         else if(os_err==OS_ERR_TIMEOUT)
79.         {
80.                     printf("task3 time out.\r\n");
81.         }
```

```
82.         APP_TRACE_INFO(("Object test task 2 running ....\r\n"));
83.     }
```

复制代码

简而言之,主要适用了两个函数:

```
1.  OS_SEM_CTR  OSTaskSemPend (OS_TICK    timeout,
2.                             OS_OPT     opt,
3.                             CPU_TS    *p_ts,
4.                             OS_ERR    *p_err)//等待信号量
```

复制代码

```
1.
2.  OS_SEM_CTR  OSTaskSemPost (OS_TCB  *p_tcb,
3.                             OS_OPT    opt,
4.                             OS_ERR  *p_err)//发送信号量
```

复制代码

架构如下:

主任伍不断尝试获取信号量,而 3 路中断则发给对应的三个任务,

用插图显示如下:

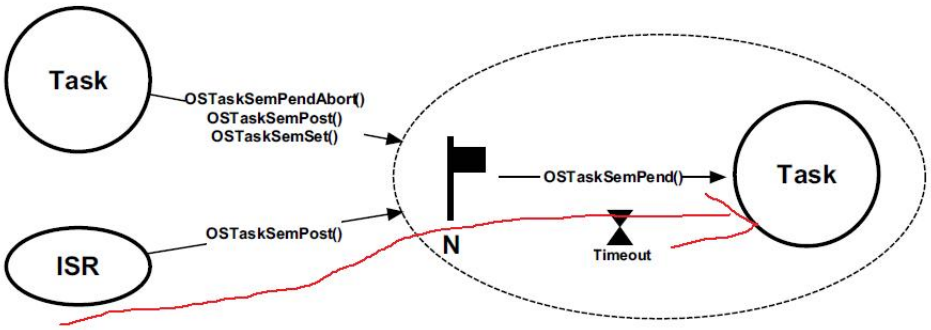


Figure 14-7 Semaphore built-into a Task

完整的代码,在 https://github.com/iysheng/chicken

【阿波罗 STM32F767 试用体验】第十二篇 在 ucosiii 下,给触摸屏和转速添加消息队列支持

这次主要记录消息队列的使用方法:1.把触摸屏的键值放到消息队列中去,这样可以在多个任务中读取键值,进行相应的操作,提高了效率.

2.把转速的值也放到消息队列中去

关键的代码:

初始化两个消息队列

```
1.  void  MessQueue_Init (void)
2.  {
3.      OS_ERR p_err;
4.      OSQCreate (&KEY_Q,
5.                 "Touch key messgae queue",
6.                 5,
7.                 &p_err);
8.      if(p_err!=OS_ERR_NONE)
```

```
9.                  APP_TRACE_INFO(("Create KEY_Q failed.\r\n"));
10.         OSQCreate (&RPM_Q,
11.                    "Round per minnet messgae queue",
12.                    5,
13.                    &p_err);
14.         if(p_err!=OS_ERR_NONE)
15.                  APP_TRACE_INFO(("Create RPM_Q failed.\r\n"));
16. }
17.
```

复制代码

发送键值的消息队列

```
1.  OSQPost (&KEY_Q,
2.                                                  (void
    *)(&(tp_dev.x[0])),
3.                                                  1,
4.                                                  OS_OP
    T_POST_FIFO + OS_OPT_POST_ALL,
```

```
5.                                                    &os_err);//发送消
    息队列
```

复制代码

读取键值

```
1.  key_value=(uint16_t  *)OSQPend (&KEY_Q,
2.                                  1000,
3.                                                  OS_OPT_PEND_BLOCK
    ING,
4.                                  &msg_size,
5.                                  NULL,
6.                                  &os_err);
```

复制代码

转速也是类似的,获取转速

```
1.  rpm_value=(uint32_t  *)OSQPend (&RPM_Q,
2.                                  1000,
```

```
3.                                          OS_OPT_PEND_BLOCKING,
4.                                  &msg_size,
5.                                  NULL,
6.                                  &os_err);
```

复制代码

发送消息队列

```
1.  OSQPost (&RPM_Q,
2.                          (void *)&ic_value[0],
3.                          1,
4.                                              OS_OPT_POST_FIFO +
    OS_OPT_POST_ALL,
5.                          &err);
```

复制代码

主要就是 OSQPost 和 OSQPend 函数的使用,操作系统的作用就是提供了很多 API 接口函数,我们可以使用并实现替换一些传统的裸机编成实现的效果.

完整的代码在 github 上

https://github.com/iysheng/chicken

【阿波罗 STM32F767 试用体验】第十三篇 结项贴,ucosiii 下完成数据测量,以及 YYFISH 的 version 0.1 版本

我的项目申请,主要分为两个部分:

1.根据正点原子的开发板,完成自己的毕业设计,主要就是在 ucosiii 下完成 ad 和频率数据测量,以及电机控制.系统框图如下所示:

设计到的硬件主要有,定时器\外部中断\RGB 屏\AD\DMA\温度检测\串口.整个软件在 ucosiii 操作系统下,涉及到任务信号量和消息队列支持.本来还想移植 stemwin,但是苦于时间有限,目前没有成功,以后有时间争取补上吧.在移植到 ucosiii 之前,基于裸机下的演示在帖子第七篇☞裸机下的测控仪开发搞一段落，但路

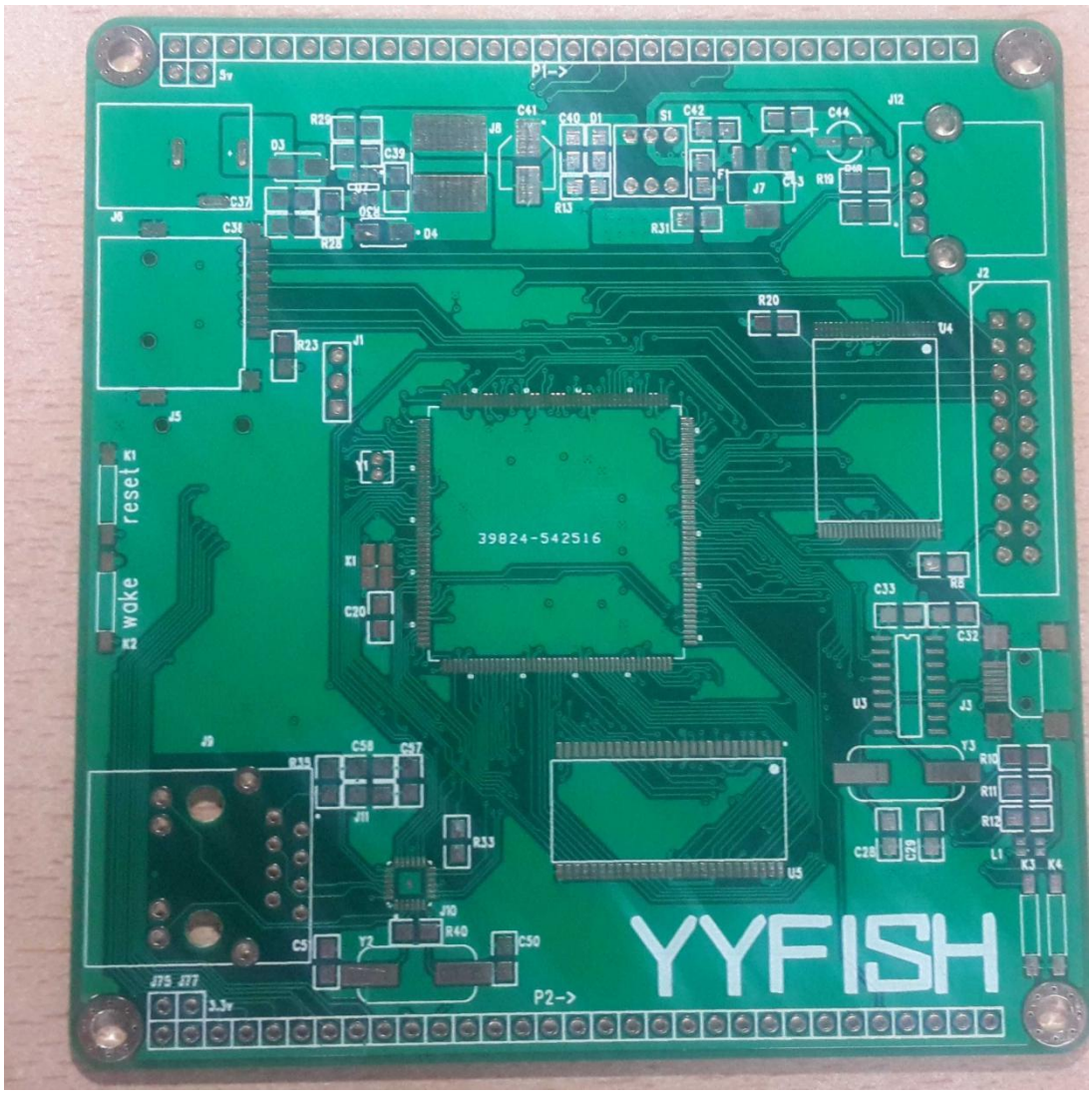
还很长好像那个视频播放有些问题.

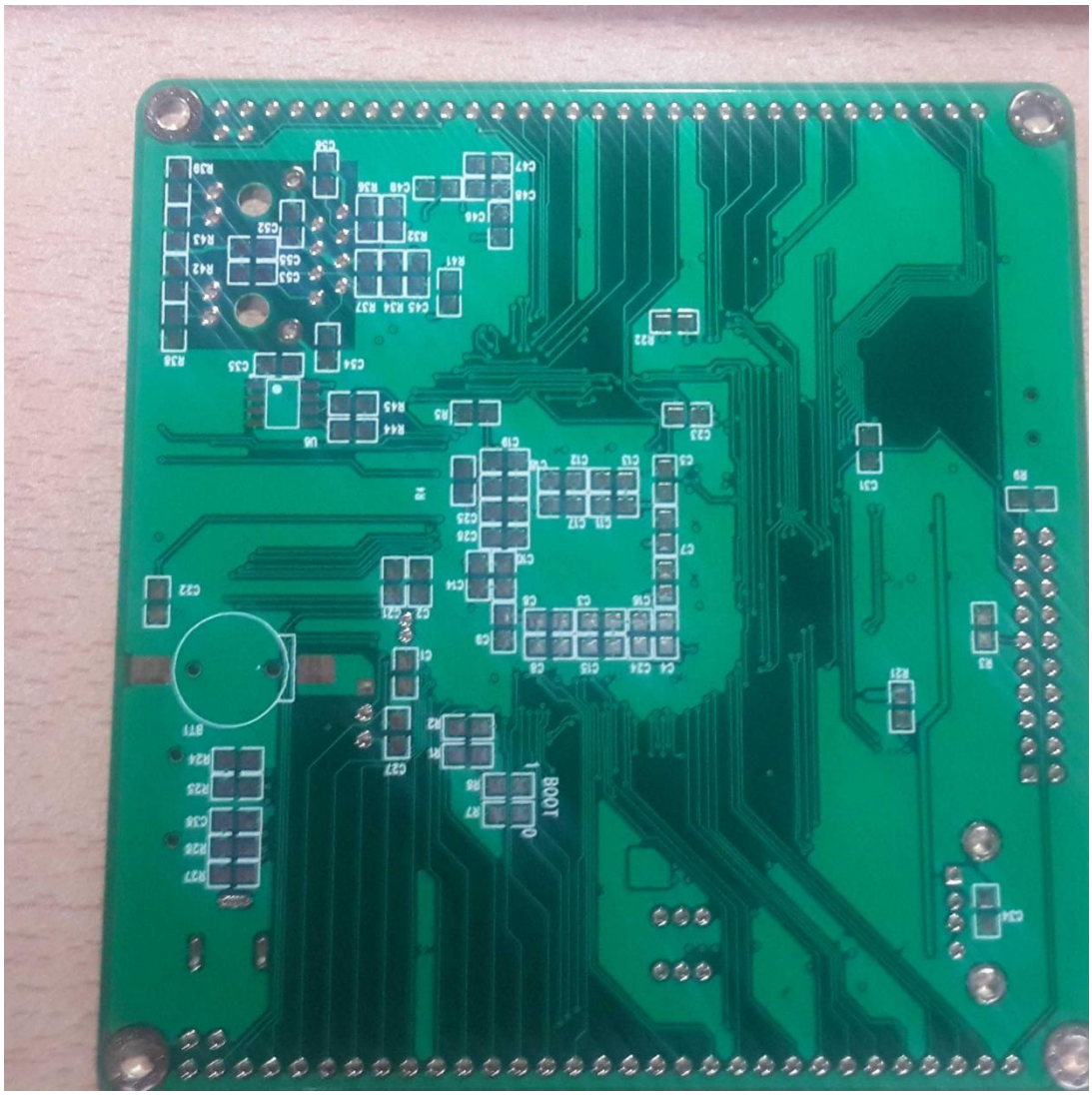
这次我就再录制下,在 ucosiii 下的效果(包括触摸屏的测试),

看一下简单录制的测试视频:

2.根据自己的需要,裁减阿波罗的硬件,自己做一个板子,然后众筹一下(**这个是本次试用的重点**).除去生产 PCB 用的时间,我用了大概 10 天的时间吧,完成了画板和器件焊接调试.

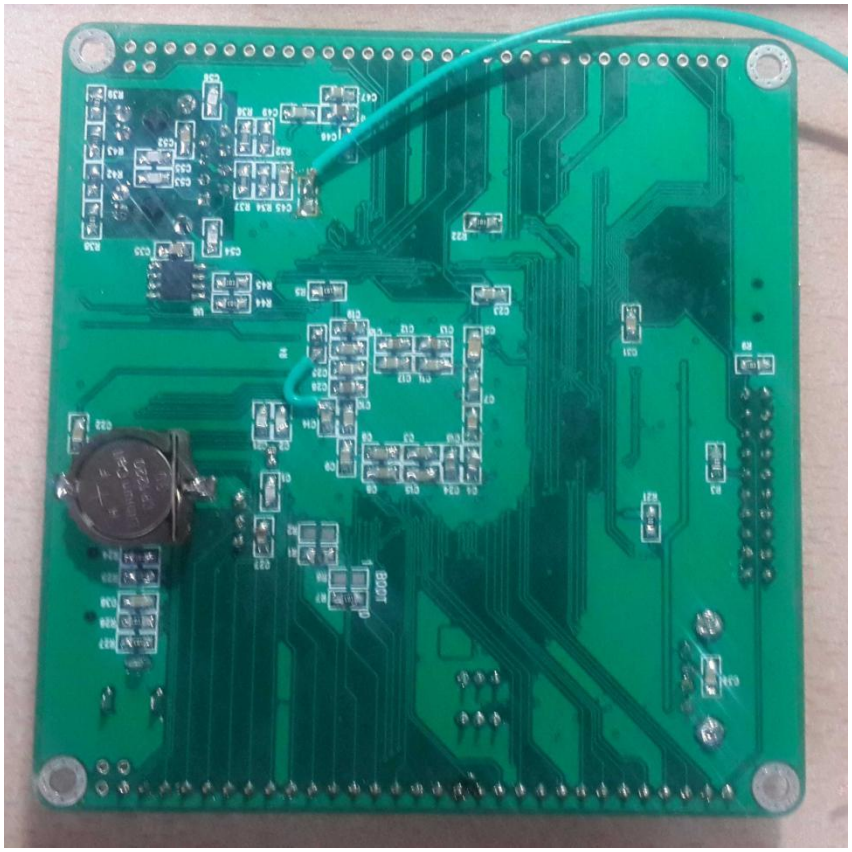
首先看下我做的板子(四层):





看下焊接好的效果,焊接过程可谓丑态百出,前两个都是把电源芯片焊接错了,导致电源供电错误,坏了两个,终于第三次成功了,焊接中,我们实验室还没有助焊剂,去同学实验室借的,特此向牛同学表示感谢.在焊接过程中,也学到了一些如何焊接密集引脚的芯片的经验。自己的感觉就是要有好的工具，才能焊好这些密集封

装的芯片。焊好的效果如下所示：



上图种那根绿色的线,是我在调试网口的时候焊接的。连接的是 lan8720 的（15 号）nrst 引脚,现在经过调试,只有网口不好用,怀疑是 lan8720 这个网卡芯片没有焊接好（QFN 的 GND 在芯片下面），并且网口的 rest 引脚没有连对。

经过测试其余的外设: 包括 SDRAM、NAND、RCT、SD 卡、USB、usart、eeprom 都正常可用。测试视频如下

YYFISH 本质是一个 **STM32**f767 的增强型核心板, 除去已经占用的 GPIO 引出了一共 60 个没有用到的 PIN。

现在还处于第一个版本，后续还需要继续的做下去，，，，

欢迎关注电子发烧友论坛公众号

获取更多工程师内容