工作日志

时间：2023.03.02

学习潘多拉STM32L4系列

1. 串口下载

①还是不精，CH340驱动干嘛的，USB转R232，这里是USB转ST\_LINK!!第一次见，还把CH340的驱动安装半天。

ST\_LINK!!

②USB\_OTG：

USB OTG即USB On-The-Go的缩写，他目的是实现在没有Host的情况下，实现设备间的数据传送。例如数码相机直接连接到打印机上，通过OTG技术，连接两台设备间的USB口，将拍出的相片立即打印出来；手机读写U盘。

1. HAL库

我刚学的这个STM32L4系列的开发板用的就只有HAL库，我之前一直把HAL库和STM32CubeMX这个软件结合起来，现在看来好像有点偏差。

## 标准库：STM32有非常多的寄存器，而导致了开发困难，所以为此ST公司就为每款芯片都编写了一份库文件，也就是工程文件里stm32F1xx.....之类的。在这些.c .h文件中，包括一些常用量的宏定义，把一些外设也通过结构体变量封装起来，如GPIO口时钟等。所以我们只需要配置结构体变量成员就可以修改外设的配置寄存器，从而选择不同的功能。也是目前最多人使用的方式，也是学习STM32接触最多的一种开发方式，我也就不多阐述了。

## HAL库：是ST公司目前主力推的开发方式，全称就是Hardware Abstraction Layer（抽象印象层）。库如其名，很抽象，一眼看上去不太容易知道他的作用是什么。它的出现比标准库要晚，但其实和标准库一样，都是为了节省程序开发的时期，而且HAL库尤其的有效，如果说标准库把实现功能需要配置的寄存器集成了，那么HAL库的一些函数甚至可以做到某些特定功能的集成。也就是说，同样的功能，标准库可能要用几句话，HAL库只需用一句话就够了。并且HAL库也很好的解决了程序移植的问题，不同型号的stm32芯片它的标准库是不一样的，例如在F4上开发的程序移植到F3上是不能通用的，而使用HAL库，只要使用的是相通的外设，程序基本可以完全复制粘贴，注意是相通外设，意思也就是不能无中生有，例如F7比F3要多几个定时器，不能明明没有这个定时器却非要配置，但其实这种情况不多，绝大多数都可以直接复制粘贴。而且使用ST公司研发的STMcube软件，可以通过图形化的配置功能，直接生成整个使用HAL库的工程文件，可以说是方便至极，但是方便的同时也造成了它执行效率的低下，在各种论坛帖子真的是被吐槽的数不胜数。

1. 潘多拉STM32L系列与IOT物联网，为什么它俩连起来？

**潘多拉（Pandora）开发板也称为 IoT Board**，是一款由 **RT-Thread 和正点原子联合**推出的基于 ARM Cortex-M4 内核的物联网开发板，主控芯片是 STM32L475，最高主频为 80Mhz，拥有 96 KB RAM 和 512 KB Flash，并具有丰富的板载资源。

潘多拉 STM32L4 IOT 开发板资源丰富，并且**集成了ST LINK**。STM32L475 的绝大部分内部资源都可以在此开发板上验证，同时扩充丰富的接口和功能模块，适合**物联网开发者**与 **STM32L4系列初学者**的使用。

物联网是什么?

**物联网IOT（Internet of things）**，顾名思义就是物物相连，万物互联。物联网其实是互联网的一个延伸，互联网的终端是计算机（PC、服务器）。

如今，为人类服务的计算机呈现出各种形态，如穿戴设备、环境监控设备、虚拟现实设备等等。只要有硬件或产品连上网，发生数据交互，**就可以被称为物联网**。物联网的本质还是互联网，只不过终端不再是计算机，而是**嵌入式计算机系统及其配套的传感器**。

STM32以其强劲的优势，已经应用在物联网的各大产品中（如，穿戴设备、环境监控设备等）

1. 考核：程序移植！！

随之而来，就有一大堆的问题！为什么可以移植程序，为什么要移植程序?

时间：2023.03.06

不是学潘多拉物联网开发板吗

**和RT\_thread有什么关系？**

为什么要学？怎么学？怎么学？

一、代码里面的Kernel文件:

Kernel（操作系统内核）是指大多数操作系统的核心部分，它由操作系统中用于管理存储器、文件、外设和系统资源的那些部分组成。操作系统内核通常运行进程，并提供进程间的通信。Kernel的核心功能为：事件的调度和同步、进程间的通信（消息传递）、存储器管理、进程管理。在DOS操作系统，操作系统内核被认为是界于基本输入输出系统（BIOS）和应用软件之间的那部分（应用命令通过操作系统内核传递到BIOS，然后再传送到相关硬件）。

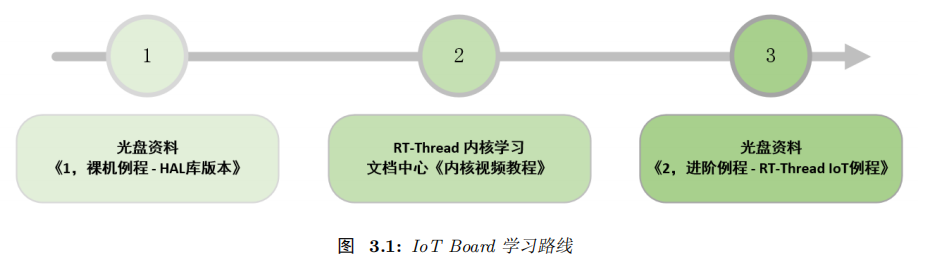
1. 潘多拉开发板有两个代码版本

一个是HAL库版本，另一个是RT\_thread操作系统版本的，它有进阶版本的，用来实现多任务多线程功能。

体会这两个版本有什么不同！为什么要分成这两个版本。

之前HAL也没学，RTT版本也不知道；底层什么样的，看了RTT的，入口参数都找不到！还得仔细认真研究代码。

三、如何学习？



第一步：把正点原子提供的裸机例程学习一遍，目的是学习各种外设的应用，懂得操作外设的一些步骤原理而不依赖于操作系统的平台；

第二步：学习 RT-Thread 的内核知识，资料在文档中心《内核视频教程》，目的是学习 RT-Thread内核，这是学习 RT-Thread 的基础也是学习 RT-Thread 最重要的一部分；

第三步：学习本教程，就是在 RT-Thread 上的应用。

那就慢慢的按照上面的来学！！

**四、HAL库和STM32CubeMX（即STM32Cube）有什么关系呢？**

学习HAL库，这个肯定最开始要明白的！！也是我一直没搞清楚的！！这个软件和这个库啥关系！学习HAL库一定需要Cube MX软件吗？因为之前两个都不熟悉，所以这两个关系一直不明白？

首先我们先了解一下STM32CubeMX，其实STM32CubeMX是目前**ST推荐的一款图形化开发工具**，它允许我们一步一步的来在**软件中配置相关参数最终一键生成C语言项目**，**这个项目支持很多IDE**，例如主流的MDK、IAR等等。**而STM32CubeMX软件生成的C语言项目所使用的正是开头所说的HAL库。**

有了STM32CubeMX这个软件，我们不必像之前使用标准外设库的时候那样，**一开始**就得着力于**初始化很多外设**，例如GPIO、NVIC、TIMER、ADC、DMA等等，**这些配置各种外设或者功能的步骤都可以在STM32CubeMX中采用图形化的方式来配置**（具体的配置步骤以后我会更新一些文章来说明），配置好之后就可以生成对应的Keil MDK的工程（当然别的开发环境也支持）。这样一来，**我们就可以把注意力集中在具体的算法实现或者程序逻辑上，而不用花费大量的时间配置一个一个的结构体，也算是提高了开发的效率吧**。

这款软件还有一个好处就是可以从整体的角度来分配IO的用途。一款MCU有很多IO脚，有的用于IIC，用的用于SPI等等。我们有时候会考虑，一个IO比如PA2使用了一个功能以后再有其他需要也使用这个IO的时候我们如何处理。这款软件就可以在整体上看到我们对不同的IO做了什么配置，这样在进行产品开发的时候，哪些IO被使用了，哪些IO没有被使用就一目了然。

1. 那后面这几天学习HAL库和STM32CubeMX软件

安装Cube MX软件。下班还有两个小时，应该可以搞定。

安装CubeMx一共需要三个：

JRE （Java Runtime Environment） Java运行环境 运行JAVA程序所必须的环境的集合

STM32CubeMX：去官网下载的话，还需要注册啥的，但是能获得最新版本的软件，下载别人下载好的，是之前的不是最新版本。还有一个问题这个软件是和KEIL5软件一起使用的，版本啥的怎么去兼容？

HAL库 STM32 HAL固件，ST官方推出的另一套库

如何查看keil uvision的版本型号？

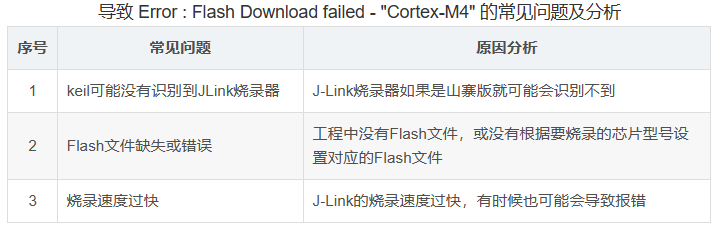
点击“Help”→“About μVision”。

1. 烧录错误问题 : Error:Flash Download failed - “Cortex-M4“

花了半个小时解决！！

**Error : Flash Download failed - "Cortex-M4"** 。这个错误原因大概是说"Cortex-M4"的Flash文件下载失败或者说无法下载的意思。

排除掉硬件损坏的情况后，通常可能是下面几个问题导致出现 Error : Flash Download failed - "Cortex-M4" 的错误：



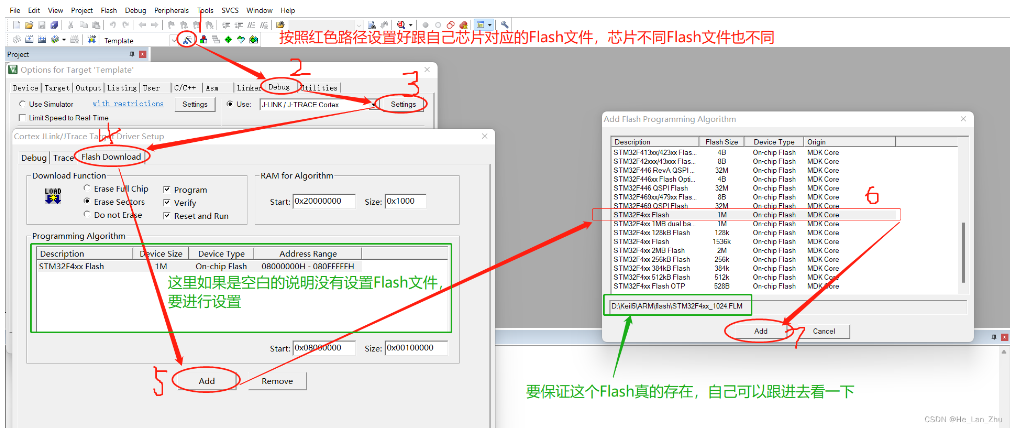
我的是问题2：没有Flash文件。

原因：就是因为一些重新安装的一些软件或者新电脑的时候会发生的一些问题。

检查有没有正确配置好对应芯片的Flash文件。

自己选择的这个Flash文件一定要真的存在Keil安装路径下的flash文件夹中。

最好是跟进Keil安装路径，看看有没有。



时间：2023.03.07

计划：①装一些必要的软件②接着昨天的计划学Cube MX软件③有了一定的基础之后在学操作系统

软件的话，安装学习Markdown、Git、Vs code等等。安装软件安装最新的，学会去官网上下载安装包等等！！在微信公众号和网盘下载确实方便，但不随最新版本的，而且，学不到！！

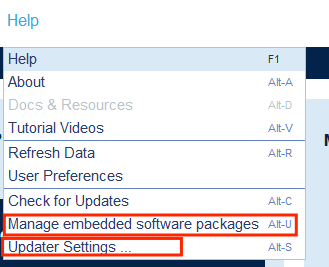
体会：

1. 有时候还是得看看视频，光看书的话学的不精。以我的意识发现不了什么高深知识点。

刚看视频，又学了一招，让C51和arm共存，这个其实我在早八百年就做到了，别人给我的C51的安装包！！

**值得注意的一点是：①安装arm之后，在安装C的时候，会出现一个窗口，绝对不能点Replace，否则，之前安装的arm的东西全部会被C51的替换点，必须点Skip。确实，我之前也这么做的，但是不知所以然，瞎点的也有可能。共存的话选Skip。②破解的时候：必须要以管理员身份打开！！之前忘了，没成功！！**

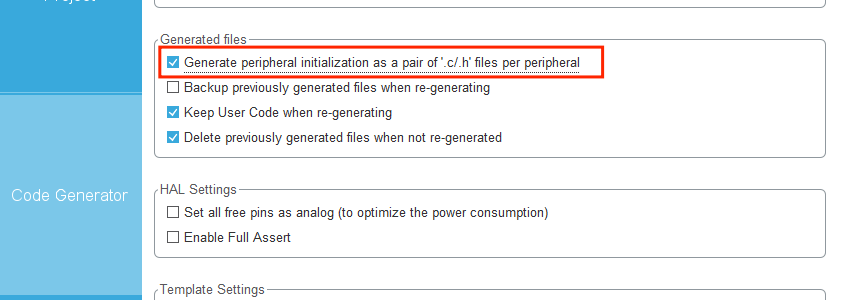
二、安装Cube MX软件需要注意的是，安装软件和安装库包的时候都不能有中文路径！！！



下面一个修改安装包路径，不然默认在C盘；上面一个安装库包！！！

在HELP里面换了路径了，还需要重新在线下载安装库包！！

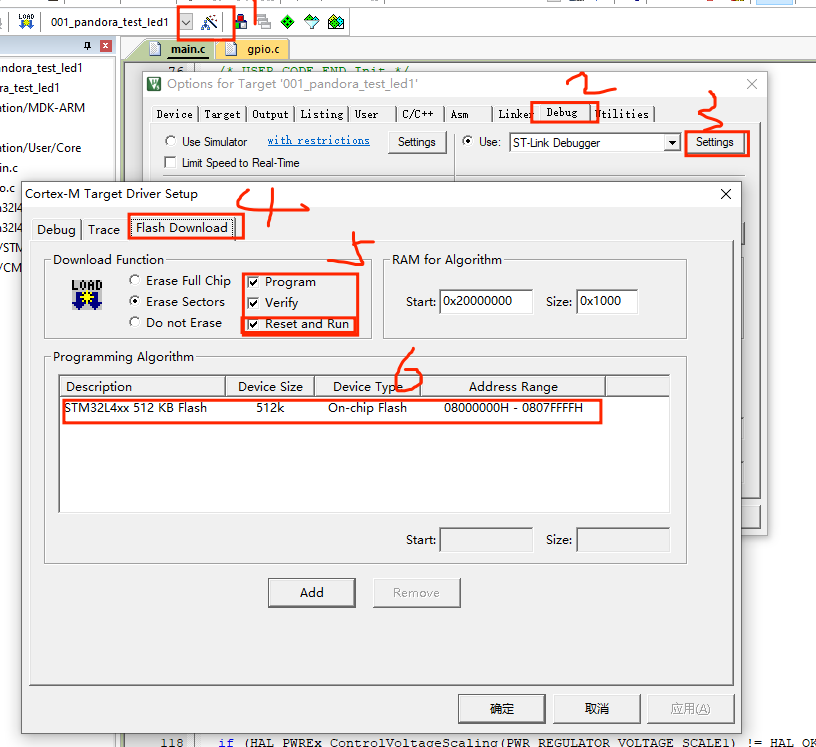
三、生成代码时，下面的这个要打上勾！！！



**不然这里然Cube MX生成的代码就会在main函数里面！！打上勾了就会自己生成一个.c文件！！**

1. 这个就是昨天的问题六，然后再加一点小东西！！

其实这些小东西都隐隐约约有点小印象！！应该是N年前看KEIL5用法时学的，KEIL魔术棒里面的东西还真不少需要了解的！



①上面按顺序操作呗！！出现Error:Flash Download failed - “Cortex-M4”，最关键的就是看一下**这里的6处，有没有添加Flash！！**

②**5处的Reset and Run最好要打勾！！**不然下载的程序需要复位一下才跑！！

1. 完美解决Keil 打开时报错 Loading PDSC Debug Description Failed

报错原因

首先明确，报错的原因是Keil版本与Package版本的不匹配

解决方案

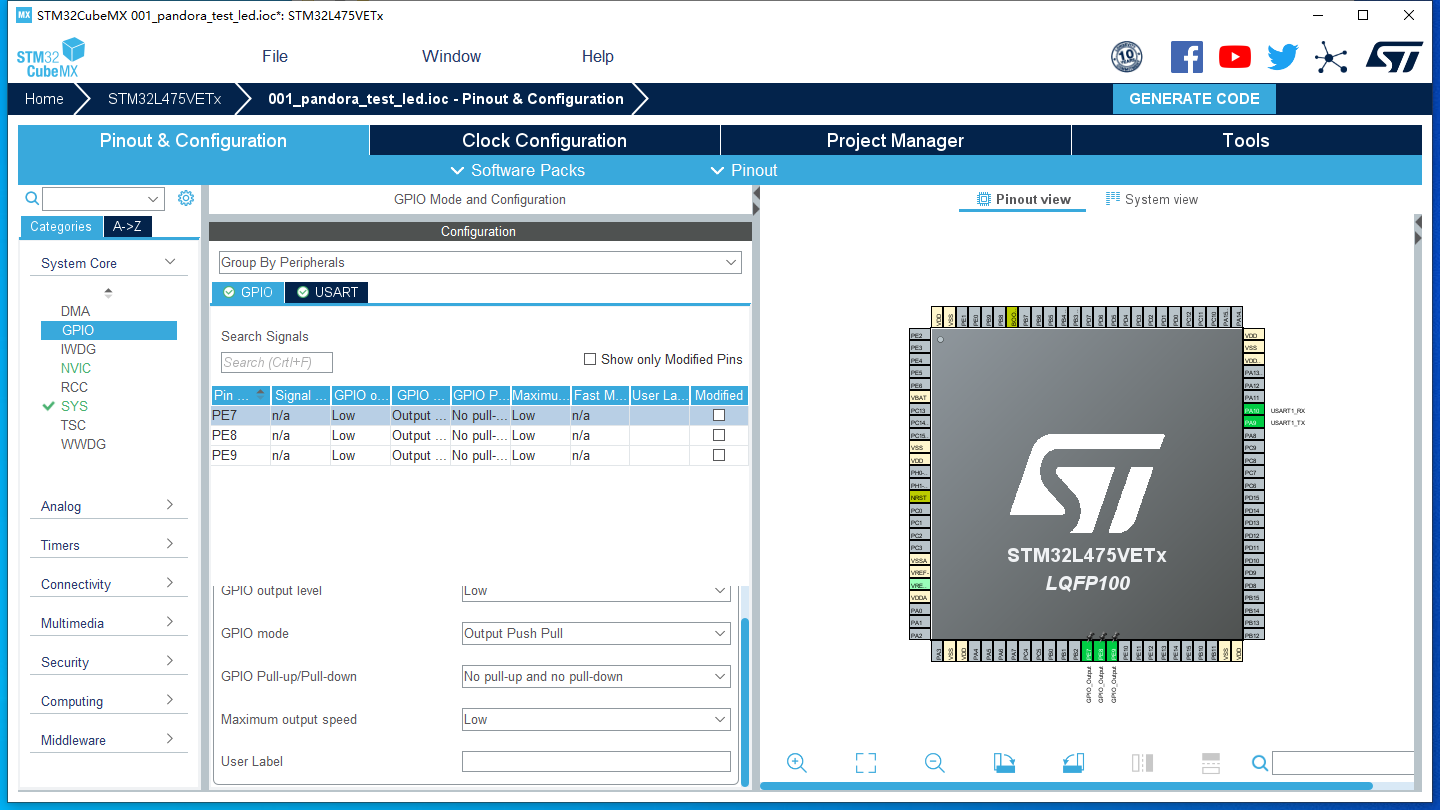
更新Keil版本，package版本

我的KEIL版本5.16a可能确实有点老了！！下载最新的用！！

1. STM32Cube MX软件使用

①用Cube MX做初始化的时候，假如一开始使用就只初始了一个引脚，就创建工程项目了，后面又需要初始化端口引脚了，肯定是可以的！

**在打开Cube MX工程进行修改，就行了！！再次生成工程，会覆盖上一次的初始化，主体函数不会改变。所以需要代码移植性强的话使用Users label标签，来增强代码移植性。**



这才是正常的，调出引脚修改的，配置引脚的地方！！

②做一个按键控制灯翻转的案例

任务：按键按下，点亮LED，再按下熄灭LED

分析：

按键按下：引脚GPIO Input—>上拉

**为什么？原理图知，按键接地，也就是说平时是高电平，用来检测低电平的，所以设置上拉！！**

点亮LED：GPIO Output设置为推挽输出。

熄灭LED：GPIO Output：推挽上拉＋初始值为高点平。

**RGB灯接的是高，引脚给高电平的灭，低电平的时候是亮；初始化的时候引脚设置高。**

对于输入引脚，可以配置的就是 GPIO Pull-up/Pull-down。

你的引脚平时处于高电平用于检测低电平，你就使用 Pull-up。如果你希望你的引脚平时处于低电平用于检测高电平，你就使用 Pull-down。

对于输出引脚，比输入多了更多的配置：

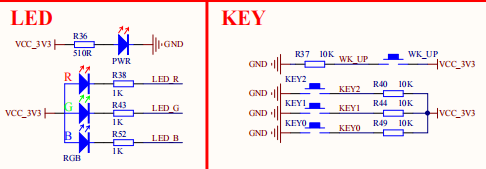
GPIO output level -> 初始化输出电平

GPIO mode -> 输出方式 -> 开漏或推挽输出

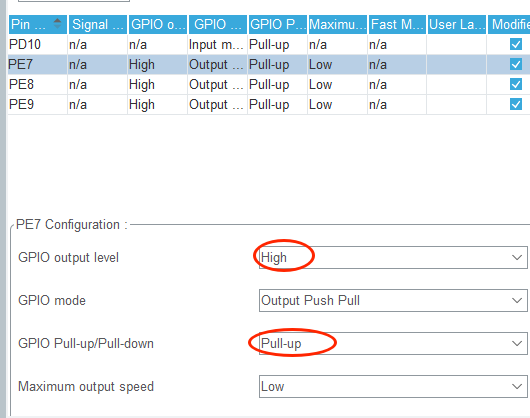
GPIO Pull-up/Pull-down -> 上拉或下拉输出

Maximum output speed 选中 GPIO 管脚的速率





按照上面的分析对引脚进行初始化配置！！！



很重要啊，圈的这两个也要设置！GPIO output level要设置高，一开始要灭的，所以给高！！不然一开始就亮了！！这个自己写标准库的时候也用到过啊！！

分析可得，还需要上拉！！

HAL\_GPIO\_WritePin(GPIOE, GPIO\_PIN\_7|GPIO\_PIN\_8|GPIO\_PIN\_9, GPIO\_PIN\_SET);

**GPIO\_InitStruct.Pull = GPIO\_PULLUP;这个写标准库的时候确实没有！！**只有PIN、SPEED、MODE三个！！

写小灯取反的时候，又了解了一个HAL库函数（积累！！）

**HAL\_GPIO\_TogglePin(GPIOE, GPIO\_PIN\_9);（取反函数）**

if(HAL\_GPIO\_ReadPin(GPIOD,GPIO\_PIN\_10) == 0 )//按键判断

{

while(HAL\_GPIO\_ReadPin(GPIOD,GPIO\_PIN\_10) == 0);//松手检测

//HAL\_GPIO\_WritePin(GPIOE, GPIO\_PIN\_9, GPIO\_PIN\_RESET);

HAL\_GPIO\_TogglePin(GPIOE, GPIO\_PIN\_9);

}

③STM32Cube MX软件中Users　label的使用！！！

提高代码的可读性！！

可以看到上面的代码，都是把GPIO\_PIN\_10引脚都直接写到代码里面，那如果小灯的IO口，现在需要做PWM或者ADC通道，需要把小灯换一个引脚，那么在CubeMX代码里面修改初始化，这个简单，后面的主函数任务里面的代码怎么办，就需要全部一个一个的修改，这样代码的可读性就极差！！

User Label

对于任意引脚，它都有这么一个选项。我想告诉你这个选项特别特别好用！这个选项简单的说就是它帮你在 main.h 中生成 define 语句。但是对于 HAL 库编程，main.h 会被用户的每个模块调用，也就是这些 define 语句的作用域几乎是全局。

#define **LED\_B\_Pin** GPIO\_PIN\_9

#define **LED\_B\_GPIO\_Port** GPIOE

#define KEY\_0\_Pin GPIO\_PIN\_10

#define KEY\_0\_GPIO\_Port GPIOD

这里我就是把PE789设置了几个标签，然后main.h里面就有了这几个宏定义！！学会找到，并使用！！并不是你定义就是啥，稍作改变！！宏定义应该会用！

举个例子让你感受一下，在一次开发中，我使用 PA0 来作为输出引脚。如果随着开发的继续 PA0 被迫要用于其他功能，那么你该怎么办？那你必须使用另外一个引脚（假设是 PB1）来替代它。

**如果你没有配置 User Label 选项，那你的代码中可能大量的充斥着**

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_0, GPIO\_PIN\_RESET);//将PA0引脚状态改为低电平

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_0, GPIO\_PIN\_SET);//将PA0引脚状态改为高电平

然后你又需要用 PB1 来代替 PA0，那你就需要将整个代码中有关 PA0 的 GPIOA 改成 GPIOB，将 GPIO\_PIN\_0 改成 GPIO\_PIN\_1。这会导致巨大的工作量，并且容易出错。那么我们来看看使用了 User Label 会带来什么变化，使用 User Label 把他取名 R1。那你的代码中充斥着的不在是HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_0, GPIO\_PIN\_RESET)，而是 HAL\_GPIO\_WritePin(R1\_GPIO\_Port, R1\_Pin, GPIO\_PIN\_RESET)。**当遇到 PA0 被迫要用于其他功能，你只需要把 PB1 的 User Label 取名为 R1 后，代码不需要做丝毫改变**。

1. asynchronous：异步

同步通信，异步通信吗！听过的！现在大部分都是异步通信

用Cube MX进行相关引脚的初始化，这里初始化USART1串口1，很简单！

**现在是如何使用HAL库，里面有哪些函数！！入口参数怎么写！**

HAL\_StatusTypeDef HAL\_UART\_Transmit(UART\_HandleTypeDef \*huart, const uint8\_t \*pData, uint16\_t Size, uint32\_t Timeout)

HAL\_UART\_Transmit(&huart1,temp,10,20);

从上面的定义的函数体上面，可知道为什么这么写？

1. HAL库 stm32使用printf函数打印到串口

本项目使用到的软件

1. stm32cubeMX 2. keil 3. 串口助手

\*以下所有带代码均在keil中编写，项目的初始化和生成都由stm32cubeMX生成。

**缘由**

在stm32单片机中，printf函数是无法打印到控制台的。为了让这个函数能实现打印功能，我们必须要把这个函数重定向，让它重定向到串口中，把要输出的内容打印到串口中。为此需要以下几步。

在Main函数中添加！！！

方法一：

#include "stdio.h"

#ifdef \_\_GNUC\_\_

#define PUTCHAR\_PROTOTYPE int \_\_io\_putchar(int ch)

#else

#define PUTCHAR\_PROTOTYPE int fputc(int ch, FILE \*f)

#endif

PUTCHAR\_PROTOTYPE

{

HAL\_UART\_Transmit(&huart1,(uint8\_t \*)&ch,1,0xFFFF);//阻塞方式打印,串口1

return ch;

}

方法二：

在main中引入：

#include <stdio.h>

在 USER CODE BEGIN 0 添加：

int fputc(int ch, FILE \*f){

uint8\_t temp[1] = {ch};

HAL\_UART\_Transmit(&huart1, temp, 1, 2);//huart1需要根据你的配置修改

return ch;

}

然后你就可以在任意地方使用 printf 语句方便的输出你想要的内容。

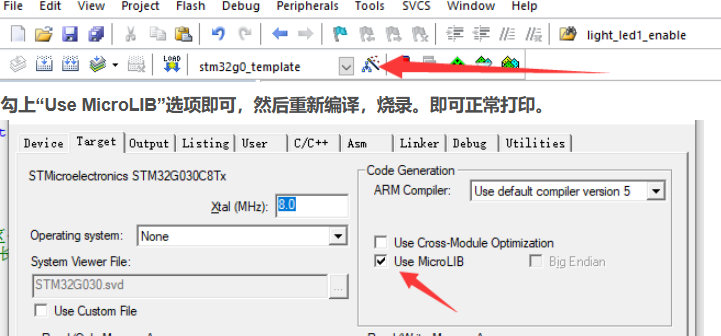
实际上一样的！！！

我按上面的操作结果不行！！我用它下面的坑解决了！！很强！！怎么找到问题所在的！！

**遇到的坑**

如果你们发现在正常运行中，打印不出来，可以这样判断一下：先进入debug模式中，运行，看printf函数有没有进入死循环，如果能往下执行，并且串口能打印出来。 那就说明你没有使用MicroLIB库。

解决方法：在keil中点击下如下按钮 ，进入项目参数设置界面



**关于Use MicroLIB是否勾选问题**

**Stm32芯片调试中经常用到串口来打印一些数据。**因此如果能使用类似windows的Console中的printf就很完美了~

MicroLib是一个针对用C编写的基于ARM的嵌入式应用程序的高度优化的库。与包含在ARM编译器工具链中的标准C库相比，MicroLib提供了许多嵌入式系统所需的代码大小的显著优势。

时间：2023.03.08

计划：先看看前面的一些总结和理解，软件安装（没事的时候装）；今天接着学Cube MX软件的使用，一些串口，定时器的案列；一些编写代码的技巧!!!

想到计划了！！可能有点难！

**不是在学Cube MX吗，有潘多拉的开发板，也有HAL库的例程，就根据这个HAL库的例程使用Cube MX来进行初始化，自己写代码实现HAL库功能！！要知道Cube MX就是一个帮你生成初始化的工具！！只是学会熟练使用Cube MX初始化你需要的引脚就OK！！不难！！**

1. 昨天学习的串口，这个UP主分享了一些编写代码的技巧，之前是闻所未闻啊，不过简单的，一点就通，但确实没用过。

串口格式：Log 信息格式

参考目前主流嵌入式、安卓等输出方式：

**[日志级别] 文件名 ： 日志信息**

//例：[info] main.c : init ok!

//例：[debug] adc.c : adc\_getvalue -> 3.3v

**说到这里我还想向大家介绍一下条件编译。因为在进行单片机开发的过程中，会需要大量的 Log 信息，但是在开发结束时，你又不想它一直打印（这会拖慢单片机的速度）。**所以我提出我的办法：

①在头文件中添加：

#define Log 1 // 打印Log信息，不想打印时改为0即可

再把.c 文件中将所有的 printf 包裹上 #if Log 与 #endif:

#if Log

printf("[info]main.c:init!\r\n");

#endif

②**可变参数宏**

关于这个内容，是我在阅读国内某云物联网模块源码是发现并学习的。

我觉得这个解决方案比之前提到的条件编译强 100 倍，甚至让我感觉到以前的做法

多么的愚蠢。这种方法不仅达到了代码的格式化，同时也完成了条件编译。

**//#define USER\_MAIN\_DEBUG**

#ifdef USER\_MAIN\_DEBUG

#define user\_main\_printf(format, ...) printf( format "\r\n", ##\_\_VA\_ARGS\_\_)

#define user\_main\_info(format, ...) printf("[main]info:" format "\r\n", ##\_\_VA\_ARGS\_\_)

#define user\_main\_debug(format, ...) printf("[main]debug:" format "\r\n", ##\_\_VA\_ARGS\_\_)

#define user\_main\_error(format, ...) printf("[main]error:" format "\r\n",##\_\_VA\_ARGS\_\_)

#else

#define user\_main\_printf(format, ...)

#define user\_main\_info(format, ...)

#define user\_main\_debug(format, ...)

#define user\_main\_error(format, ...)

#endif

当我需要打印串口信息的时候，define 一个 USER\_MAIN\_DEBUG, 在我不需要时将其注释。

这代码

声明语法与可变参数函数类似：**使用三个完整句号“...”的序列表示必须传递一个或多个参数**。 在宏扩展期间，宏替换列表中特殊标识符\_\_VA\_ARGS\_\_的每次出现都被传递的参数替换。

必须这样写！！不然下面使用user\_main\_printf的会报错的！！

user\_main\_printf("stm32");

缺省号代表一个可以变化的参数表。使用保留名 \_\_VA\_ARGS\_\_ 把参数传递给宏。当宏的调用展开时，实际的参数就传递给 printf()了。

③小插曲：个性化输出

1、借助下面的网站设计自己的字符

<http://patorjk.com/software/taag/>

**//这个先复制下来，有错误的加一个转义字符\就行，这里看全乱套了，没关系的，串口上正常显示**

printf(" \_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_ \_ \r\n");

printf(" | \_\_ \\\_\_ \_\_||\_\_ \_\_| | | |\r\n");

printf(" | |\_\_) | | |\_\_\_\_\_\_| | | |\_\_ \_ \_\_ \_\_\_ \_\_ \_ \_\_| |\r\n");

printf(" | \_ / | |\_\_\_\_\_\_| | | '\_ \\| '\_\_/ \_ \\/ \_` |/ \_` |\r\n");

printf(" | | \\ \\ | | | | | | | | | | \_\_/ (\_| | (\_| |\r\n");

printf(" |\_| \\\_\\ |\_| |\_| |\_| |\_|\_| \\\_\_\_|\\\_\_,\_|\\\_\_,\_|\r\n");

printf(" \r\n");



1. 外部中断

今天在学一下，用Cube MX生成外部中断和定时器！！结合我的开发板进行实验！！

查原理图PE9可以定时器通道一。

外部中断生成PWM，调节PE9小灯。



不太对吧！之前学标准库，外部中断，是用按键产生外部中断，进入外部中断，做一些事情来体现进入了中断。

生成PWM波，是使用定时器来实现的吧！！这里咋回事！！下午再看一下！！

下午把外部中断实验用Cube MX实现，还有定时器中断、定时器输出PWM、定时器输入捕获等等，全部复现一下。

按键实现外部中断，仿照他的代码来，不算难！！但是我搞了一下午！原因，PD8910需要设置为下降沿！！我没太注意，设置了上升沿！！代码没反应

/\*\*

\* @brief 中断服务程序中需要做的事情,在HAL库中所有的外部中断服务函数都会调用此函数

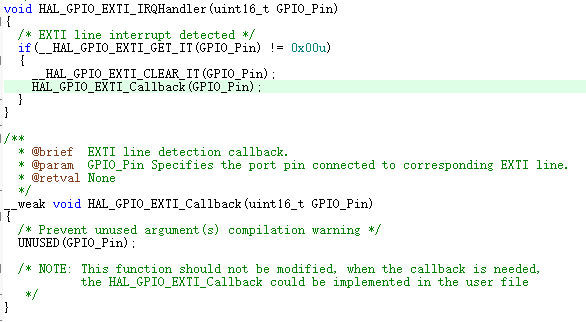
\* @param GPIO\_Pin 中断引脚号

\* @return void

\*/

void HAL\_GPIO\_EXTI\_Callback(uint16\_t GPIO\_Pin)

这个函数！！和标准库对比，好像并不太一样！！



刚用cube不久对HAL库不熟悉，对于callback这个函数仅仅看hal library无法知道他的道道。

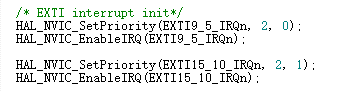
再工程里面一个名叫**stm32f1xx\_hal\_gpio.h的头文件中**有这么个函数定义。

原来cubemx会自动设置好相应的 HAL\_GPIO\_EXTI\_IRQHandler(uint16\_t GPIO\_Pin)，使用中断函数只需要在 HAL\_GPIO\_EXTI\_Callback(GPIO\_Pin)中写入你所需执行的逻辑即可。

一开始用这个函数的时候也很奇怪！这是怎么进入中断的？中断服务函数在哪？没有清楚中断吗？等等疑问

又仔细看了**stm32f1xx\_hal\_gpio.h**这个头文件，其实中断里面相关的东西大多都在这里面的！！看上面的函数void HAL\_GPIO\_EXTI\_IRQHandler(uint16\_t GPIO\_Pin)，这个不是中断的进入入口函数吗！里面就有清中断相关操作！！还有是IT还是RXNE中断（判断）！！里面才有这个函数void HAL\_GPIO\_EXTI\_Callback(uint16\_t GPIO\_Pin)，是一层一层的调用的！

还有，中断优先级分组在哪设置的呢？只看见的初始化分组！



就在GPIO里面！！就目前想到的都找到了！！

三、Cube MX我最近才开始使用。现在碰到这样一个问题：使用Cube MX生成了一个工程，并且已经完成了。但是需要增加新功能，外设比原先增加了。我重新修改了Cube MX文件，重新生成了工程，原有工程里面自己的代码有些被删除了，但是无法知道哪里被删除了。

有一些没有删除！！所以是代码不规范！！

**①如果不是自己单独建立.c和.h文件，直接采用Cube MX自动生成的文件，那么就把自己的程序写在**

/\* USER CODE BEGIN Init \*/

/\* USER CODE END Init \*/

之间，才不会被系统删除。

②**要养成良好的习惯**

使用这种自动生代码成工具，建议将自己的代码放在自己建立的文件里面，与自动生成的代码分隔开，只留下调用接口或者使注册回调函数的方法，尽量降低代码的耦合度；

等你的工程越来越大，需求改动越来越多的时候，强耦合的代码再加上自动生成工具会降低你的工作效率，苦不堪言。

1. 定时器

这个就要必须设置时钟了，前面的USART1和EXTI等，他们的时钟可能不需要设置，就只是内部时钟源就行！！这里就必须学习Cube MX的时钟设置了！！

没搞出来！！明天接着搞！！

时间：2023.3.9周四

计划：①复习之前的吧，随便看看，别忘了就行。周末做一周总结！！

②昨天搭建的Cube MX初始化，内容包含了入门篇的几乎所有的外设！！可以多搭建几次，熟悉，要能熟练应用为止。昨天的一个小小的错误，浪费了我下午一大半的时间，就这么算了！！吸取教训，就是因为昨天学的串口中断，用到按键输入做外部中断触发条件，结果把按键下降沿触发，不小心设置成上升沿触发了！！结果代码死活跑不通！查了半天错，才发现！！

③定时器啥的设置！！今个这个学完了，Cube MX入门篇算是学忘了！抓紧

体会：

1. 对使用Cube MX软件初始化后，需要做什么？

那肯定不同的要求，写不同的代码，需要添加那些HAL库函数，函数功能是什么！这些都是写代码时积累的！！

HAL\_TIM\_Base\_Start\_IT(&htim3);//启动定时器

定时器相关设置和操作：

**总结**，对于定时器的定时中断，相对比较简单，绝大多数功能图形化配置已经为我们配置好了，我们做的只需两步：

在main函数中，**开启TIM2中断**，**HAL\_TIM\_Base\_Start\_IT(&htim2);**

htim2现在知道含义了，大差不差的都是！！h:HAL tim:定时器 2；系统初始化的时候给你定义的一些关键参数，找到并应用。

重写 **HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)** 函数，然后在里面实现我们需要的功能。定时器中使用这个函数进行操作。

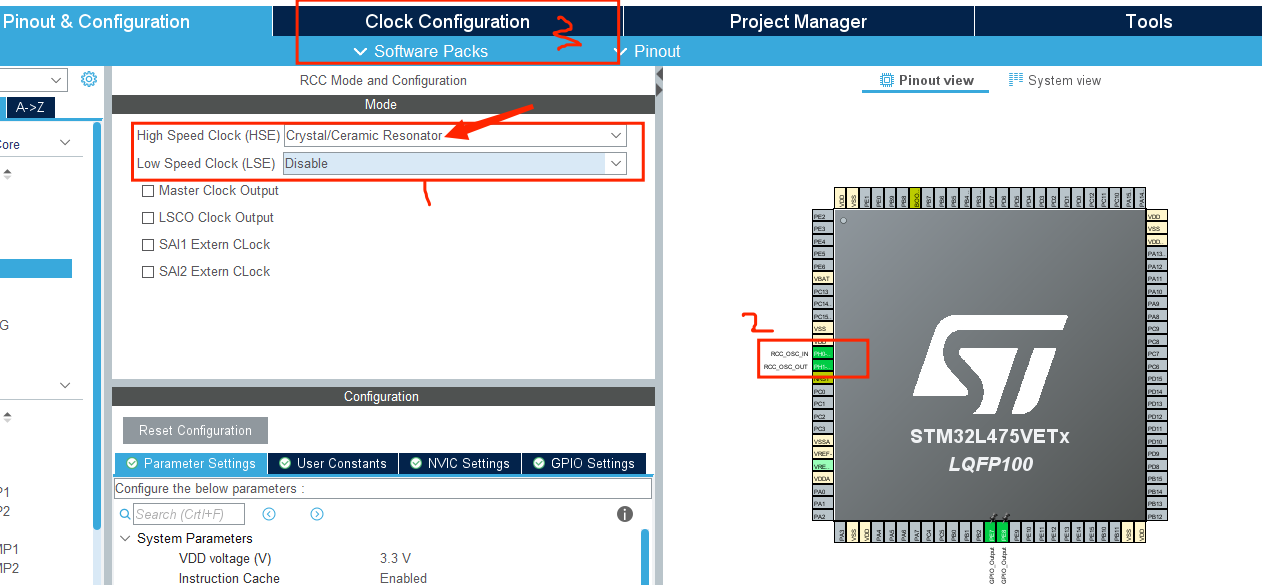
1. STM32CUBEMX使用外部晶振配置时钟

慢慢学，今天学配置外部晶振步骤！！

①要想使用外部晶振必须要先打开晶振接口。

我就说怎么这样的步骤呢？

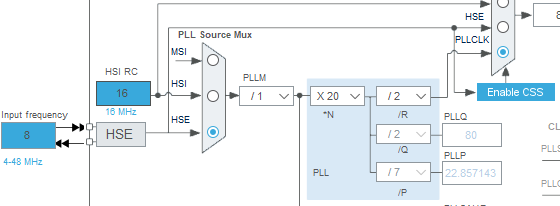
系统时钟源选择外部晶振时钟源RCC；设置时钟在进入clock configuration设置；



**上述分别为高速时钟源和低速时钟源，高速时钟一般是12M、8M、25M等以兆为单位的晶振，低速时钟往往是32.768k的频率。**

2处：此时mcu边上也会出现相关提示，表示这几个口需要与晶振相连接。

②下面来到时钟树配置界面，发现使用外部时钟源时，时钟输入端的频率可以修改：



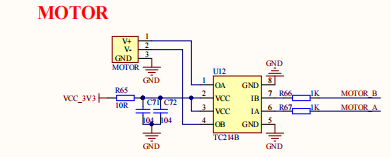
**这个数值需要与你实际使用的晶振频率对应**，比如我现在板子上的晶振是25M的，那么我在这里也需要填写25M。与此同时，需要把这前面一个节点的配置选成**HSE，即外部高速时钟，另一个选择PLLCLK，即锁相环倍频输出的时钟。**除此之外，部分mcu可能具有CCS功能，即在外部时钟失常的时候自动切换到内部时钟工作，我在这里已经使能了这个CCS功能，因此在**此显示CCS Enabled**。

③调整完毕后在此回到时钟树的配置。直接在最右边的CPU CLOCK里面输入480即可，系统会提示没有解决方案，问你要不要使用其他方法，点击确定即可，等待一会，配置完成，系统已经全速运行。

1. PWM输出实验

这次不是什么呼吸灯了，玩玩电机

预备知识：小灯、串口、按键啥的先不说了！！电机和定时器通道的选择（原理图分析）





观察原理图可得：在PA0和PA1口，可供选择的定时器TIM2\_CH12,TIM5\_ch12,等。

使用 STM32L475 的 TIM2\_CH1 和 TIM2\_CH2 来产生PWM，通过 TC214B 电机驱动芯片来驱动板载电机。

潘多拉 STM32 开发板上的电机是通过 **TC214B 来驱动**的，芯片的两路控制引脚分别连在PA0 和 PA1 上，这里我们使用 TIM2\_CH1 和 TIM2\_CH2 产生两路 PWM 来控制芯片 TC214B，从实现驱动电机。

一些中断频率的计算当然要会计算，并且熟悉:

1MHz=1000KHz

TIM2\_CHnHandler.Pulse = arr / 2; //设置比较值,此值用来确定占空比，默认比较值为自动重装载值的一半,即占空比为50%

就是Arr：Auto Reload Register - 32 bits value自动重装载值。

前面不是说根据定时器来输出PWM，控制电机加速，减速，正转，反转；

PA01两个引脚，不是要设置复用吗！使用Cube MX不用自己设置复用的！！自己设置好了！！

①然后就是给通道值来调节占空比！！

/\* @brief 设置TIM2通道2的占空比

\* @param compare 比较值

\* @return void

\*/

void TIM\_SetTIM2Compare2(uint32\_t compare)

{

**TIM2->CCR2 = compare;**

}

给值方式，对相关寄存器写值。

②//PWM和定时器中断一样，也需要启动函数

//HAL\_TIM\_Base\_Start\_IT(&htim2);

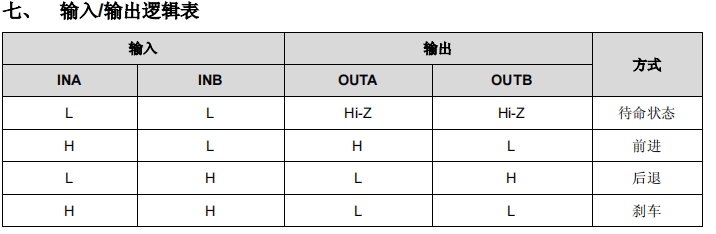
//在相关函数里面找找看，里面的参数在他的.c文件里面找！！这是我需要学的

HAL\_TIM\_PWM\_Start((TIM\_HandleTypeDef \*) &htim2, (uint32\_t) TIM\_CHANNEL\_1);

HAL\_TIM\_PWM\_Start((TIM\_HandleTypeDef \*) &htim2, (uint32\_t) TIM\_CHANNEL\_2);

和正点原子的HAL库有点点区别，有的使用Cube MX初始化的时候并不能面面俱到，需要注意和HAL库区别，函数的使用！！

③潘多拉开发板TC214B芯片驱动电机！！



1. USMART 调试组件实验

刚发现这个USMART这个还怪好玩！！有机会学一下！！

真六，RTC实验使用 TFTLCD 模块来显示日期和时间，实现一个简单的实时时钟，并可以设置闹铃。时间和日期不正确，可以利用**上一章介绍的 usmart 工具**，通过串口来设置，并且可以设置闹钟时间等。

用串口设置了时间后代码，代码里面的数据是实打实的跟改了哇！！有这么六吗，用串口去修改开发环境里面的代码数据！！

时间：2023.03.10

1. 接着昨天的USMART

可以不用管这个调试助手功能如何实现的，但需要会移植它，应用在你所需要的地方！！

这个代码好像不能并不能实际改变啊！那我昨天使用串口把代码修改了，今天在下载，代码的数据并没有实现改变！！

不过当时重新下载时，确实改变了！这就搞不太明白了！！

二、**FATFS文件系统（了解）**

FATFS是一个完全免费开源的FAT 文件系统模块，专门为小型的嵌入式系统而设计。完全用标准C 语言编写，所以具有良好的硬件平台独立性。可以移植到8051、PIC、AVR、SH、Z80、H8、ARM 等系列单片机上而只需做简单的修改。它支持FATl2、FATl6 和FAT32，支持多个存储媒介；有独立的缓冲区，可以对多个文件进行读／写，并特别对8 位单片机和16 位单片机做了优化。

这个和前面的USMART差不多，了解它的功能，学会移植它进行应用。

debug是排除故障，排除错误，调整程序，移去程序中的错误的意思。debug也是一种程序，一种调试工具，可以供程序员检查修改问题.

时间：2023.3.13

计划：上周学的使用Cube MX软件初始化搭建工程，搭建了一些STM32L475内核的东西LED，USART、TIMER、PWM、电机、BEEP等，TFT的花了一天做初始化，我的版本的和网上的不一样，还没搞出来。但有点心得，得出结论应该有两种方法：①简单的初始化TFT彩屏，（这个TFTLCD屏不是SPI驱动的吗，这个不需要单独设置引脚），再移植相关驱动，当然中间的移植文件代码规范不一样自己需要一点点修改。②做复杂初始化，SPI驱动接口都设置下。具体还不太清楚，毕竟还没搞出来。之后的任务！算一个

后面的话，开始学习RT-Thread了！！从零开始！！笔记的话，用Markdown记录了，正好翻篇！！