【独家】我就要用MDK来开发树莓Pico，怎么地吧！

树莓派 Pico是一个小巧、“迅速”且多功能的开发板，基于独家定制的RP2040芯片打造，是在英国的树莓派团队设计的全新微控制器。

以**30RMB左右的价格来看**，Pico作为一个开发板具有非常吸引人的特性：

- **搭载了设计最大频率为133MHz的双核Cortex-M0+**
  - 实际可以轻松超频到250MHz，甚至是400MHz
- **256K + 8K 的SRAM**
  - 由多个SRAM总线从机接口构成，从而保证了多总线主机访问时不易出现冲突的问题——双向八车道的高速，几乎不会堵车——吞吐量杠杠的
- 大量充满奇思妙想的外设（这里就不做赘述）

在开发环境上，Pico身为单片机，却有着Linux般豪华的富贵病——这么说吧，你要是没玩过cmake、gcc、没用过命令行、没搞过OpenOCD，你都不好意思说你是Pico的C玩家。

在Pico官方论坛上，曾经有一个带节奏的帖子叫做《**Pi Pico - the most user un-friendly MCU?**》（中文：树莓派Pico——对用户最不友好的MCU）？如果你可以看懂英文，建议去观摩下这个13页的热帖。其中你可以看到：

- Pico-SDK团队开发者~~卡场撕逼~~亲切的与~~各种暴躁老哥~~用户交流使用经验
- Pico-SDK开发者谈Pico开发环境的设计思路，总结如下：
  - **不会玩cmake的请学习cmake**，用不了你多少时间
  - **Windows我们也支持啊，你装个Linux模拟环境……**
  - **我们推荐所有用户都应该用树莓派4的Linux环境来开发Pico这个MCU**
  - 别人都玩得好好的，你玩不好一定是你不熟悉cmake
  - makefile玩家、IAR玩家、MDK玩家请自寻出路（“**on your own**”）
  - **我们团队庙小，4美元的开发板你还要啥自行车?**
  - ……

诸如此类，非常下饭。该贴的连接如下：

https://www.raspberrypi.org/forums/viewtopic.php?f=144&t=315750&start=75

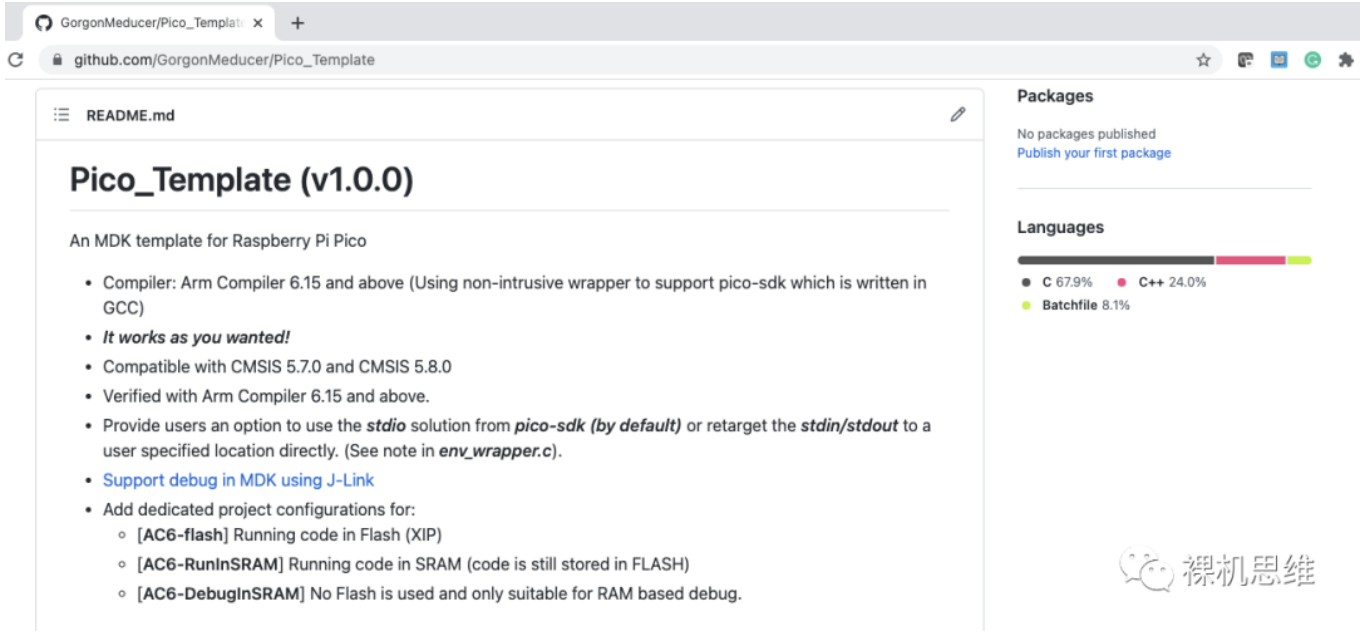
然而，MCU的开发并不同于MPU的开发。我的观察中，树莓派这类能跑Linux的系统，基本上使用的是Linux生态，在这一生态下，很多工具比如cmake、命令行、GDB或者OpenOCD之类都是如空气和水一样自然的东西。然而，树莓派团队在处理Pico这类MCU时可能多少有点“屁股决定脑袋”了，仍然按照自己的习惯照搬了Linux的那套开发习惯到MCU环境中。

> 我经常说，抛开正态分布的中央主极大、用两端的个案来举反例，就是耍流氓。

对MCU开发环境来说，虽然也有不少人使用gcc、cmake之类的工具，但主体的大多数人还是以IDE等“一站式”开发工具为主体的。RP2040无论多么优秀，它本质上就是个装了两个Cortex-M0+的大号MCU，凭什么非要上Linux环境才能开发?

MDK虽然老旧、不支持多级工程管理、偶尔闪退、语法提示经常出错、被破解的爹妈都不认识……被人骂了那么多，但**Cortex-M用MDK开发仍然是主流**。但无奈，人家的孩子人家说了算，**官方明确态度说暂时不支持Arm Compiler 6，也不支持用MDK这样的不带cmake支持的IDE，你也没办法啊**。

好在**Pico-SDK**是一个基于**BSD 3-Clause**协议的开源项目；RP2040的数据手册写的也很清晰。官方说不支持，我们就自己来呗？于是就有了这个**MDK**专属的**Pico-Template**开源项目。



实际上：

- **Pico-Template 是目前世界上第一个用MDK配合Arm Compiler 6开发Pico的模板；**
- **使用该模板你可以使用Pico-SDK来访问全部的外设**

实际使用中 Pico-Template 具有以下特点：

- **支持Arm Compiler 6**
- **可以使用RTE和Pack-Installer获得各类中间件软件包**
- **告别纯汇编编写的startup文件，使用纯C语言进行开发**
- **配置栈和堆的大小更为简单**
- **支持使用JLINK进行调试**
- **默认搭载了perf_counter服务**
- **一键切换不同的地址空间布局**
  - **在外部Flash里执行代码**
  - **在SRAM里执行代码（代码仍保存在外部Flash里）**
  - **在SRAM里调试**
- **【裸机思维】对该开源项目提供持续的维护和更新**

【Pico-Template的部署】

一个合格的工程模板，应该做到只要成功的下载到了本地，就能够立即使用——Pico-Template也是这样。因此，所谓的Pico-Template的部署，实际上有三种方式：它们主要围绕着如何处理Pico-Template所依赖的第三方仓库而有所区别。

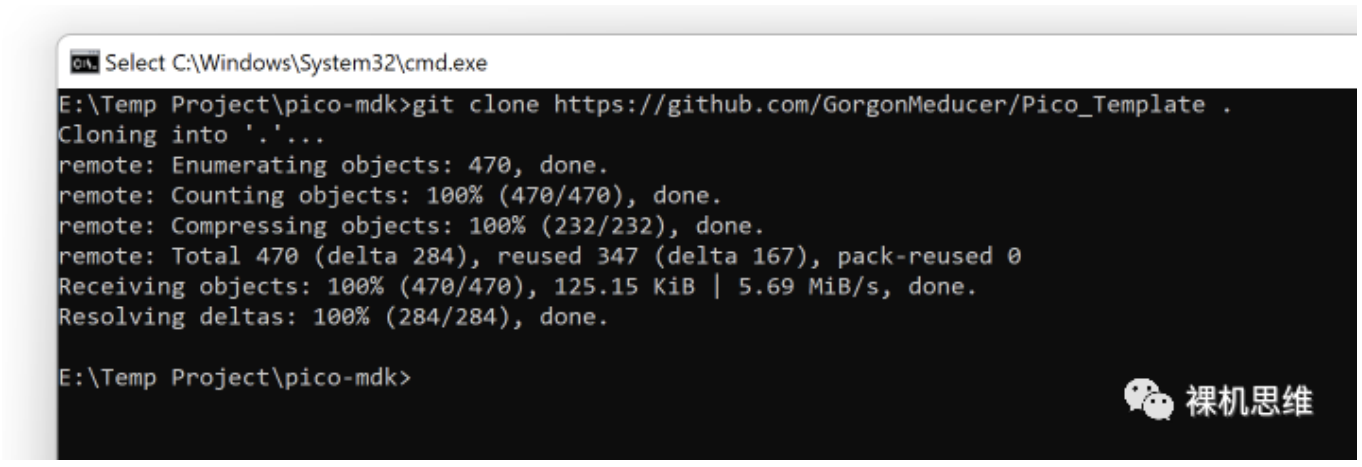
第一种方式：使用git工具进行下载

1、新建一个目录，比如叫做 pico-mdk来保存模板，并进入该目录

```
mkdir pico-mdk
cd pico-mdk
```

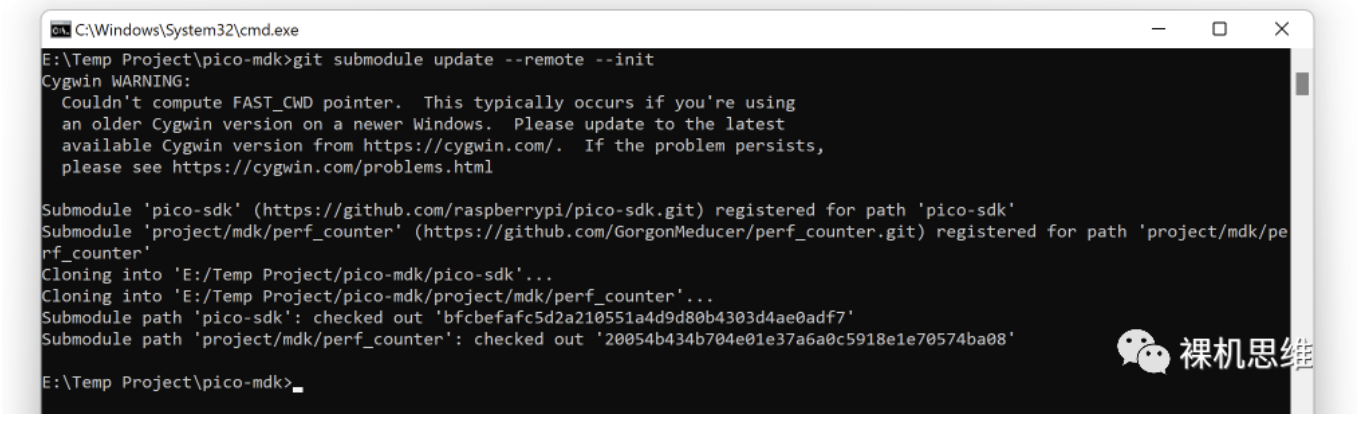
2、使用git工具clone模板到本地：

```
git clone https://github.com/GorgonMeducer/Pico_Template .
```



3、将Pico-Template所依赖的其它仓库以submodule的形式更新到本地：

```
git submodule update --remote --init
```

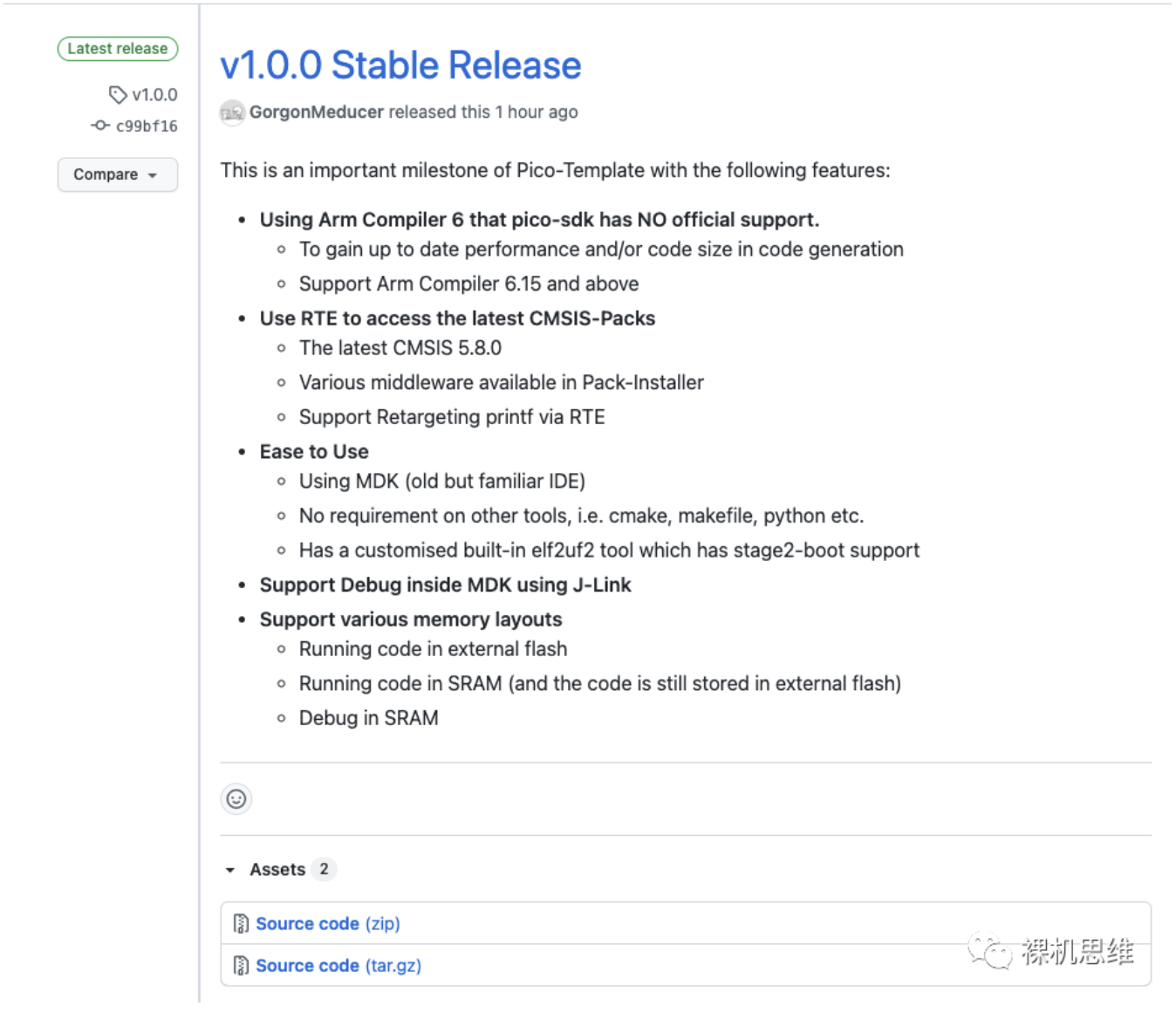


至此，我们已经成功的将Pico-Template同步到了本地一个叫做pico-mdk的目录下。

第二种方式：手工下载压缩包

1、打开Pico-Template在Github上的Release页面，下载最新版本的压缩包。

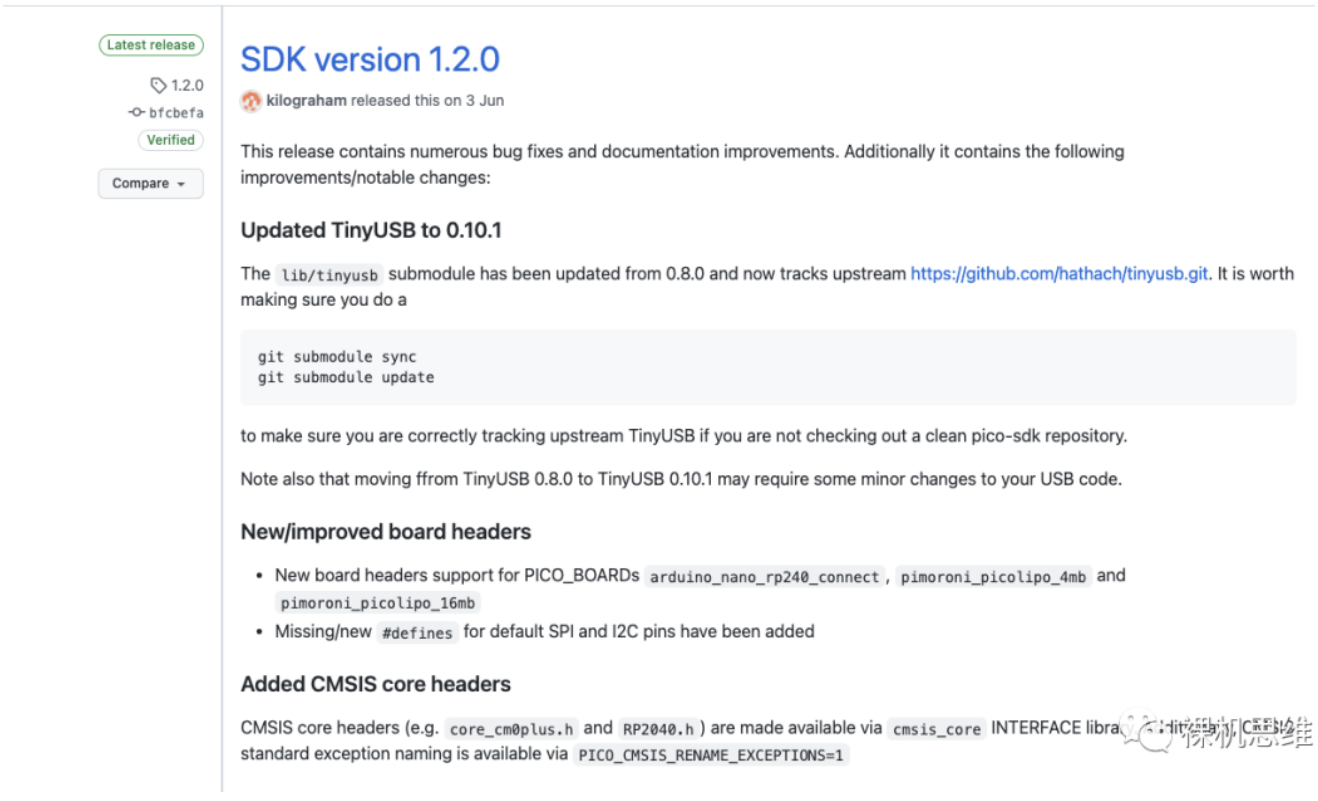
https://github.com/GorgonMeducer/Pico_Template/releases



完成下载后，解压缩到本地。

2、打开Pico-SDK在Github上的Release页面，下载最新的压缩包：

https://github.com/raspberrypi/pico-sdk/releases



Latest release

1.2.0

bfcbefa

Verified

Compare

SDK version 1.2.0

kilograham released this on 3 Jun

This release contains numerous bug fixes and documentation improvements. Additionally it contains the following improvements/notable changes:

Updated TinyUSB to 0.10.1

The `lib/tinyusb` submodule has been updated from 0.8.0 and now tracks upstream https://github.com/hathach/tinyusb.git. It is worth making sure you do a

```
git submodule sync
git submodule update
```

to make sure you are correctly tracking upstream TinyUSB if you are not checking out a clean pico-sdk repository.

Note also that moving ffrom TinyUSB 0.8.0 to TinyUSB 0.10.1 may require some minor changes to your USB code.

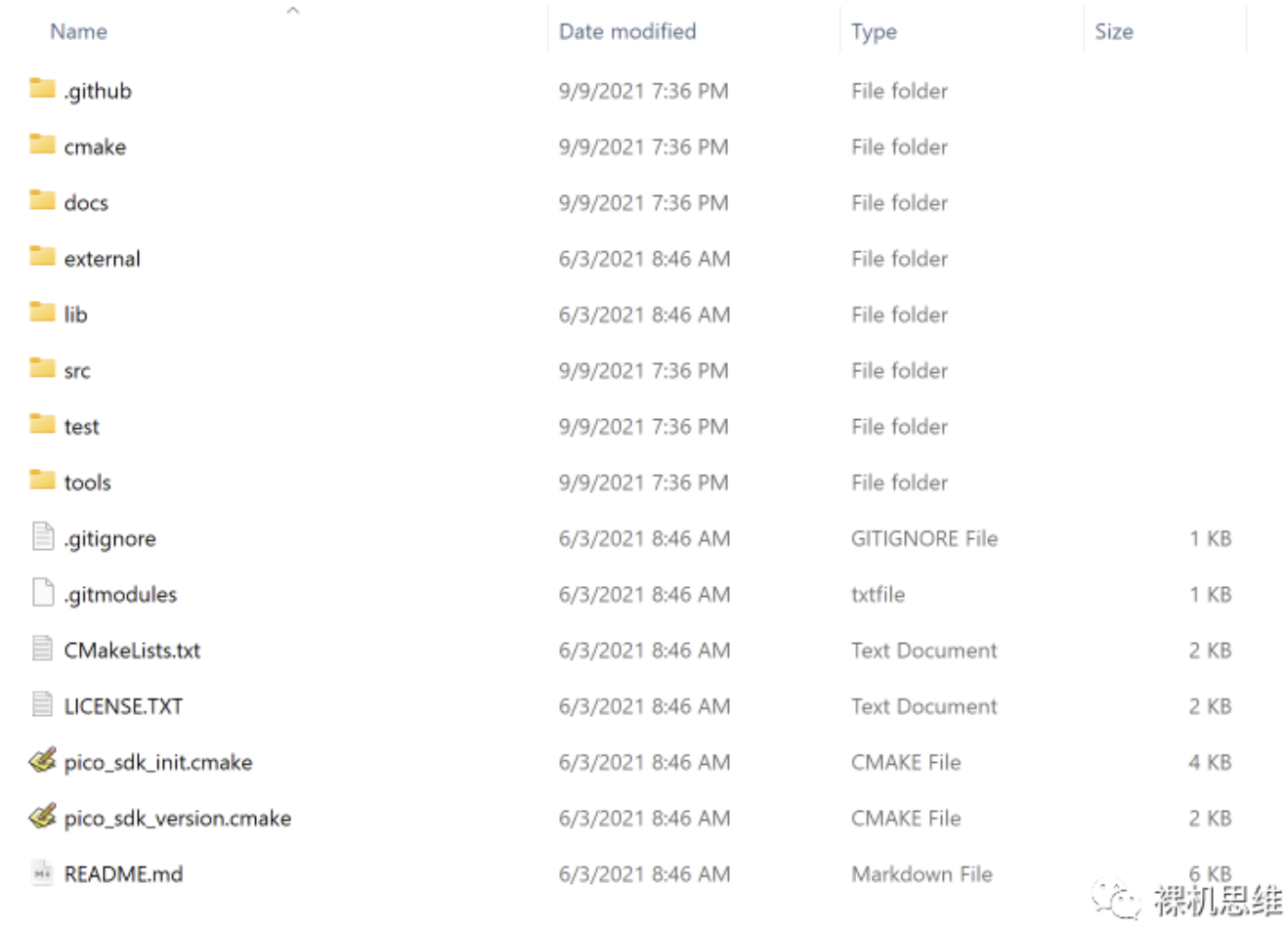
New/improved board headers

- New board headers support for PICO_BOARDs `arduino_nano_rp240_connect`, `pimoroni_picolipo_4mb` and `pimoroni_picolipo_16mb`
- Missing/new `#defines` for default SPI and I2C pins have been added

Added CMSIS core headers

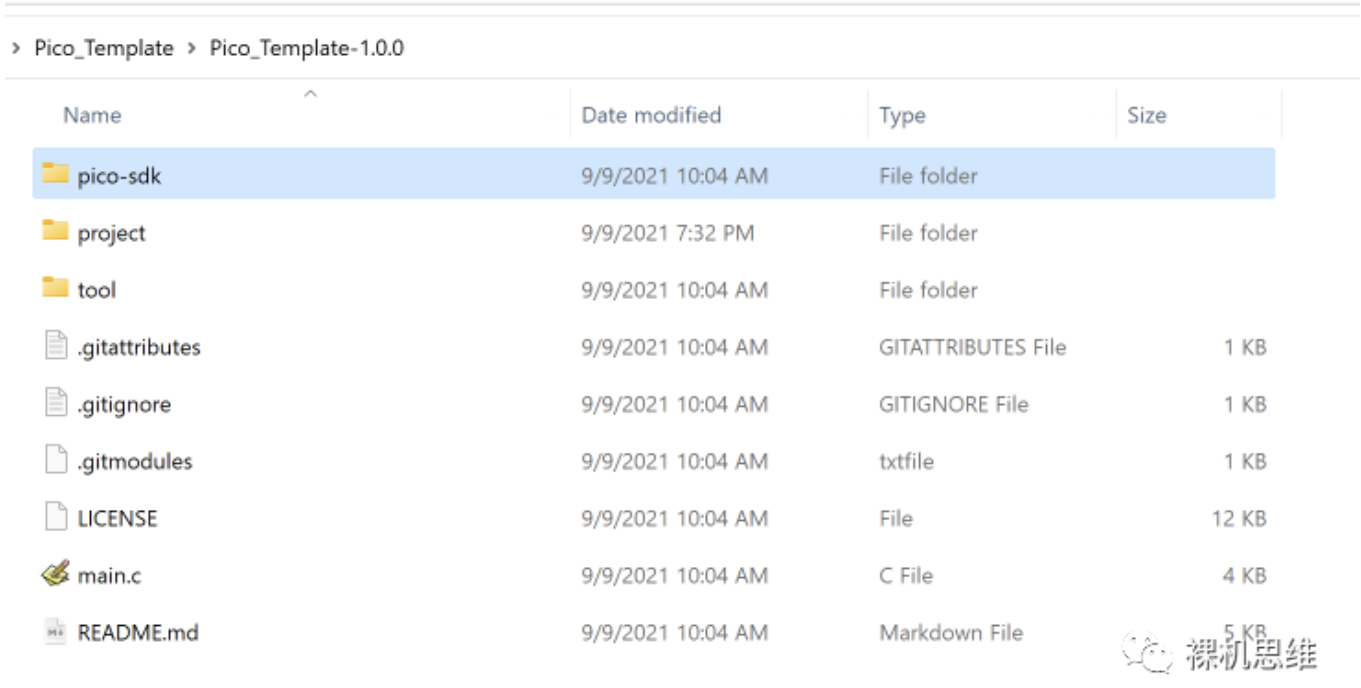
CMSIS core headers (e.g. `core_cm0plus.h` and `RP2040.h`) are made available via `cmsis_core` INTERFACE libra[illegible] standard exception naming is available via `PICO_CMSIS_RENAME_EXCEPTIONS=1`

完成下载后，解压缩到本地。打开解压后的目录，应该看到类似下图的内容：



Name	Date modified	Type	Size
.github	9/9/2021 7:36 PM	File folder	
cmake	9/9/2021 7:36 PM	File folder	
docs	9/9/2021 7:36 PM	File folder	
external	6/3/2021 8:46 AM	File folder	
lib	6/3/2021 8:46 AM	File folder	
src	9/9/2021 7:36 PM	File folder	
test	9/9/2021 7:36 PM	File folder	
tools	9/9/2021 7:36 PM	File folder	
.gitignore	6/3/2021 8:46 AM	GITIGNORE File	1 KB
.gitmodules	6/3/2021 8:46 AM	txtfile	1 KB
CMakeLists.txt	6/3/2021 8:46 AM	Text Document	2 KB
LICENSE.TXT	6/3/2021 8:46 AM	Text Document	2 KB
pico_sdk_init.cmake	6/3/2021 8:46 AM	CMAKE File	4 KB
pico_sdk_version.cmake	6/3/2021 8:46 AM	CMAKE File	2 KB
README.md	6/3/2021 8:46 AM	Markdown File	6 KB

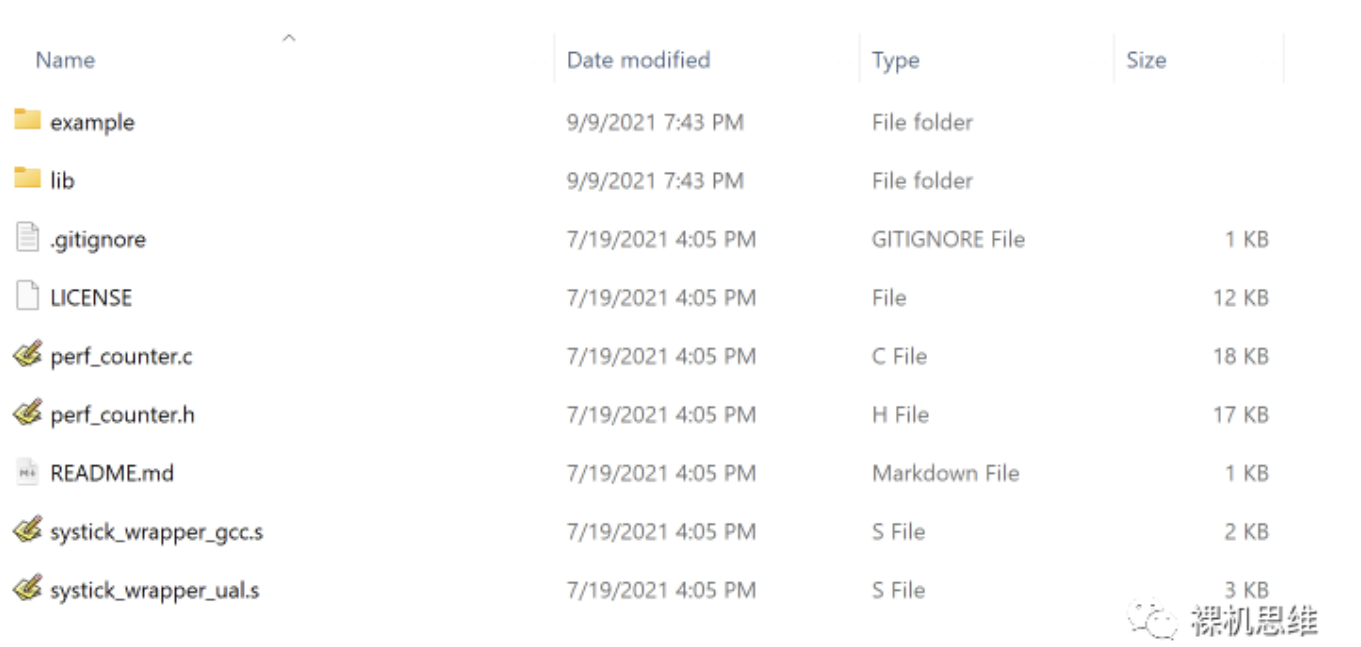
全选上述目录列表中的内容后，将它们拷贝到**Pico-Template**的**pico-sdk**目录内



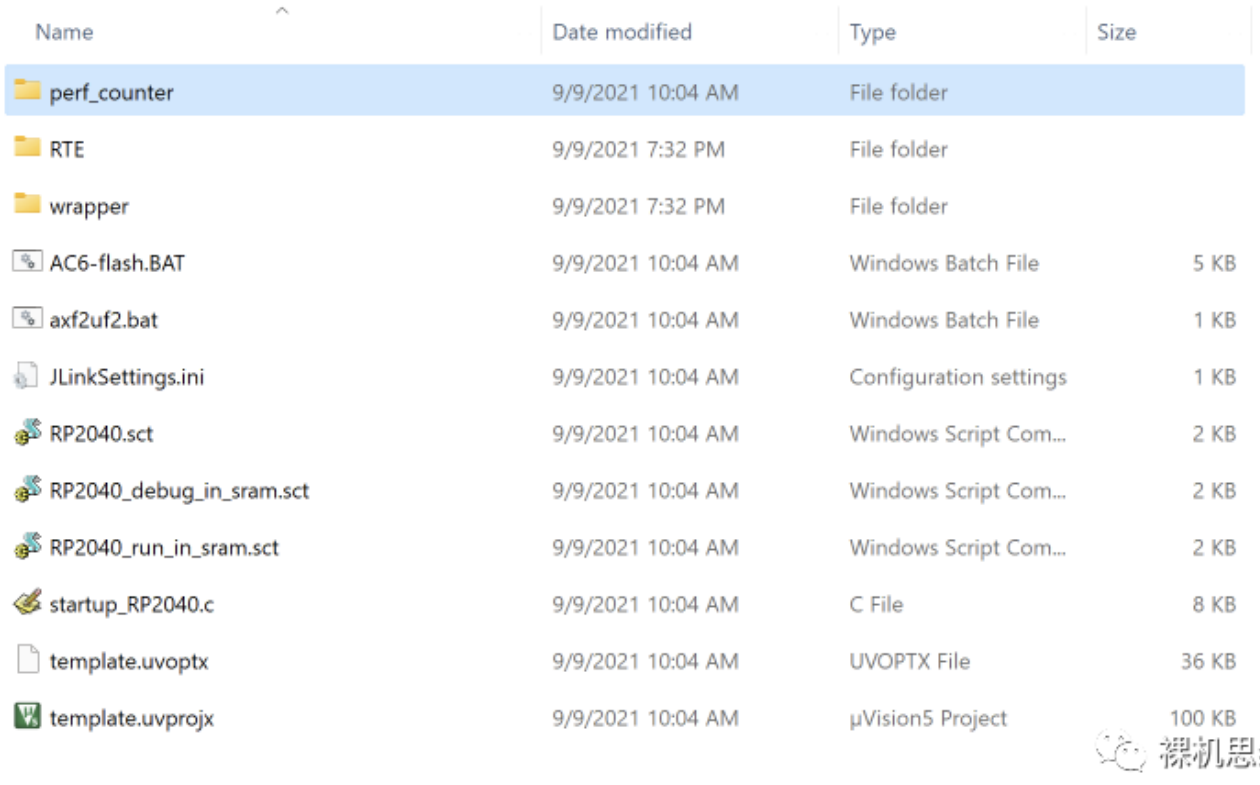
3、打开perf_counter在github上的Release页面，下载最新的压缩包：

https://github.com/GorgonMeducer/perf_counter/releases

完成下载后，解压缩到本地。打开解压后的目录，应该看到类似下图的内容：



全选上述目录列表中的内容后，将它们拷贝到 **Pico-Template/project/mdk/perf_counter**目录内：



至此，我们成功的完成了Pico-Template的合体工作。恭喜恭喜！

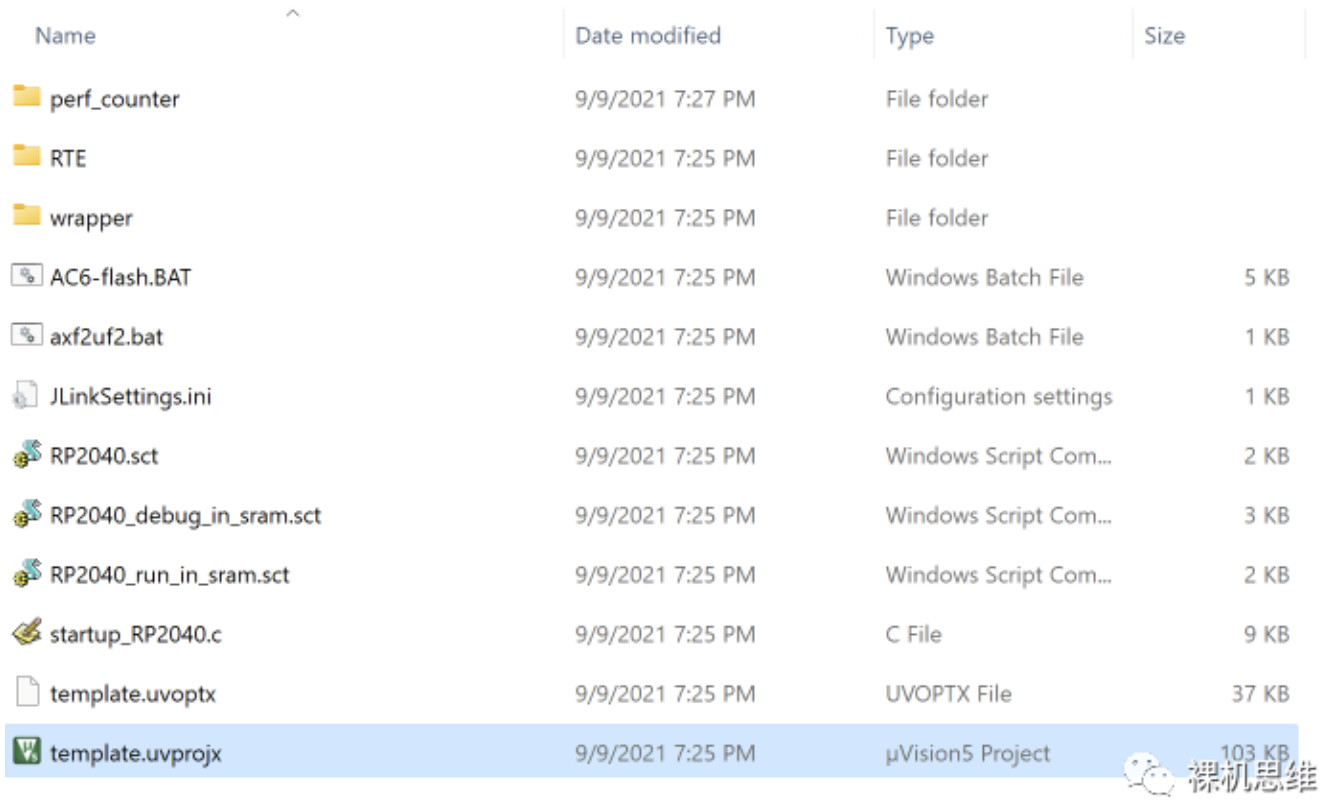
第三种方式：网盘见

如果你觉得上述方法都挺麻烦的，尤其是你无法稳定的访问Github，那么可以在订阅【裸机思维】公众号后发送关键字 "Pico"来获取网盘链接。下载成功后立即可以使用。

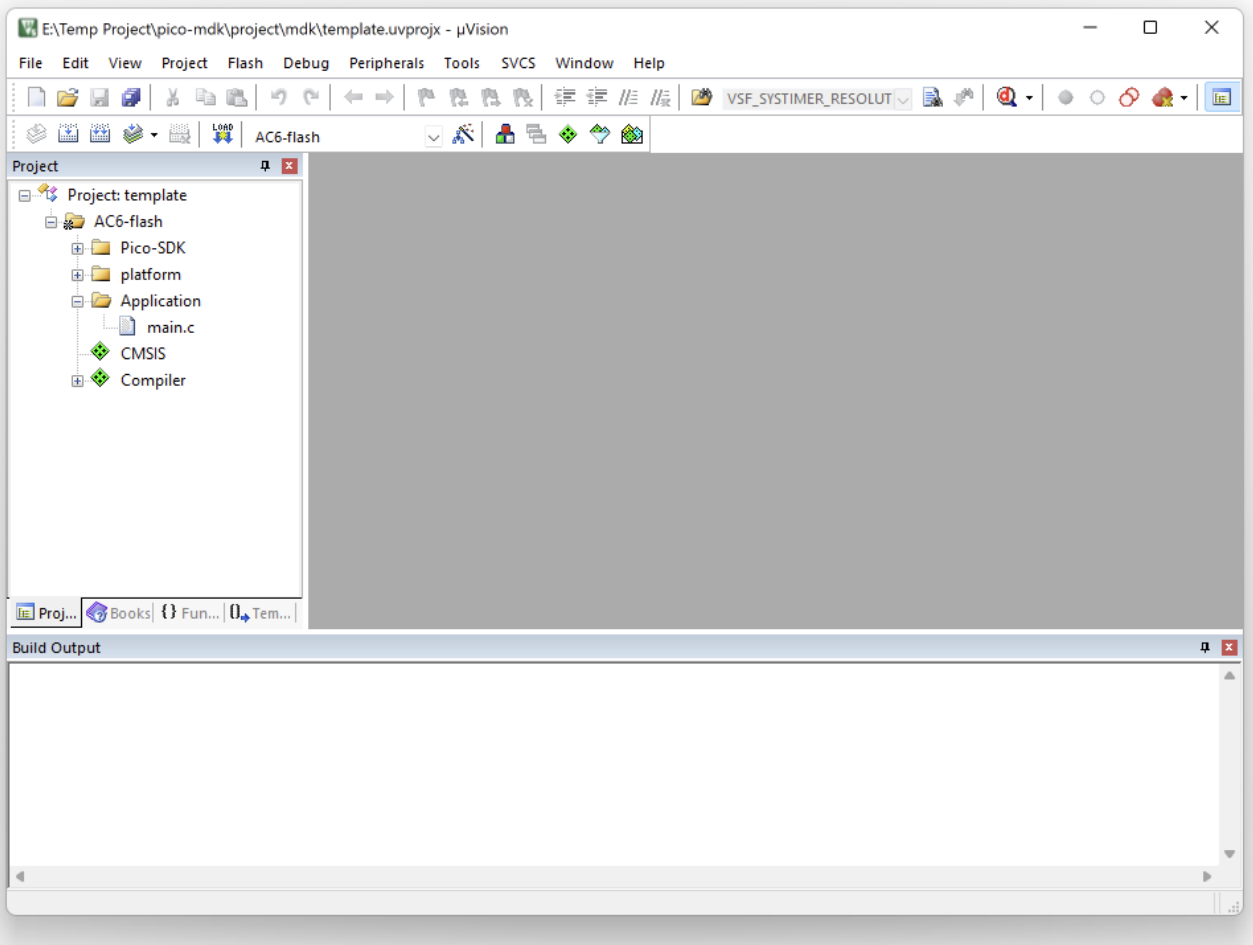
这一方法唯一的缺点是：我可能会忘记更新网盘上的压缩包。

【如何编译和下载】

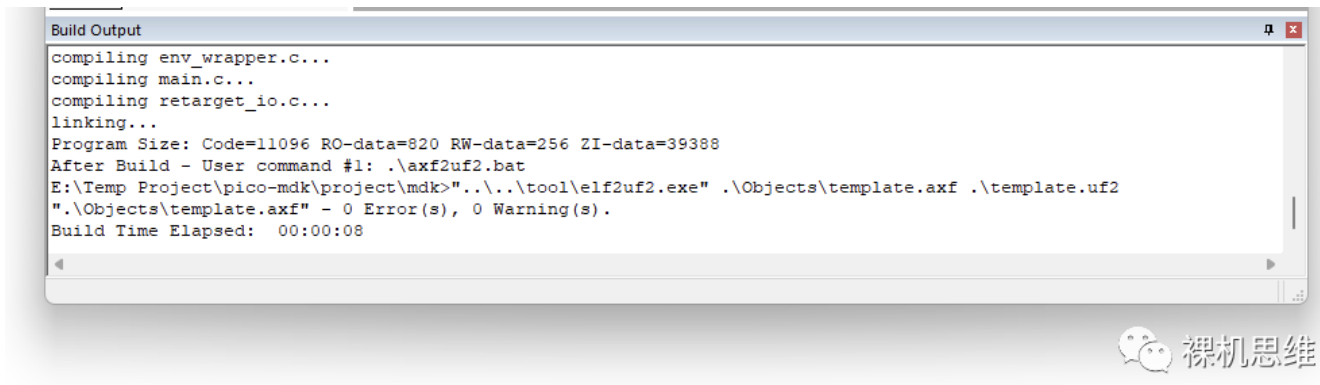
当你获得了Pico-Template后，可以通过路径project/mdk找到工程文件：



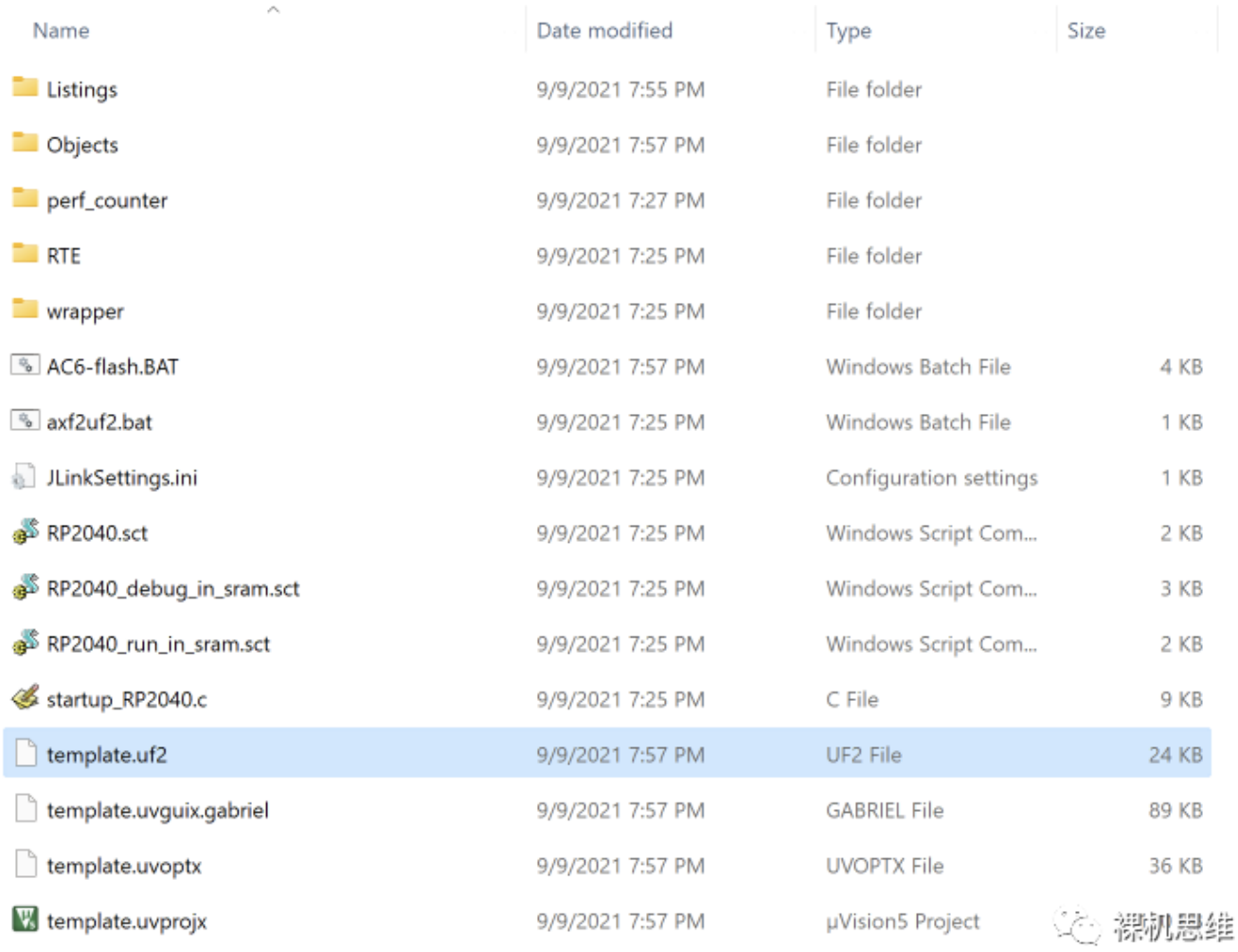
双击后，就可以见到我们熟悉的界面：



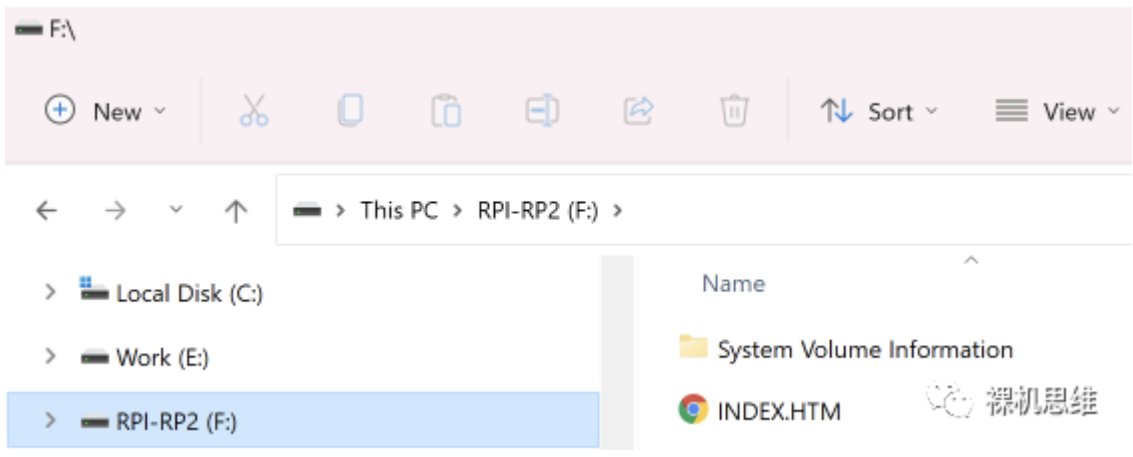
单击编译，应该可以顺利的看到类似如下的结果：



可以看到，在工程目录下（**project/mdk/**）生成了一个名为 **template.uf2** 的文件——这就是Pico专用的镜像文件：



此时，我们可以按住Pico上的白色按钮不放、将Pico的USB接口连接PC。当我们在文件管理器中发现一个新的叫做 PRI-RP2 的U盘时，说明Pico已经成功进入烧录准备状态。



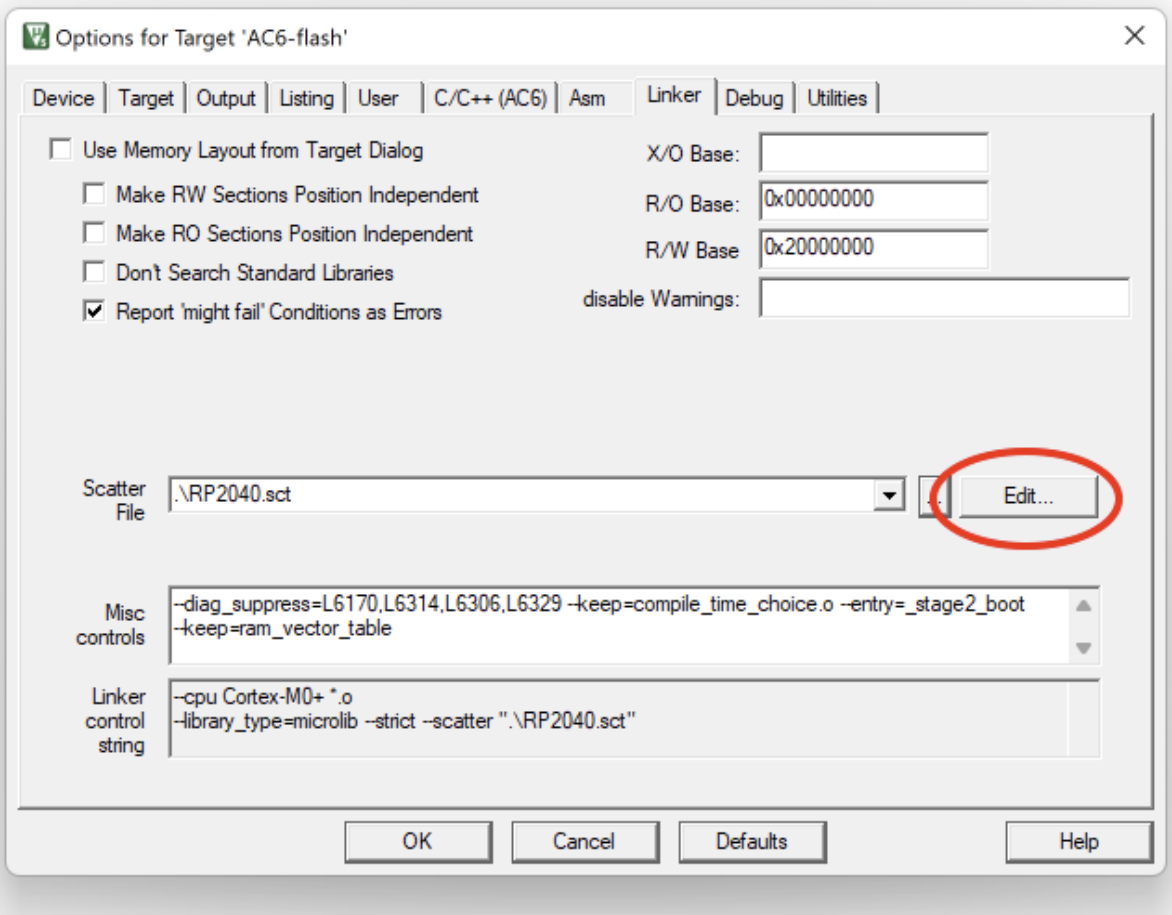
将**template.uf2**拖放到U盘中即可。

如果一切顺利，可以看到Pico上的LED以大约0.5Hz的频率进行呼吸。

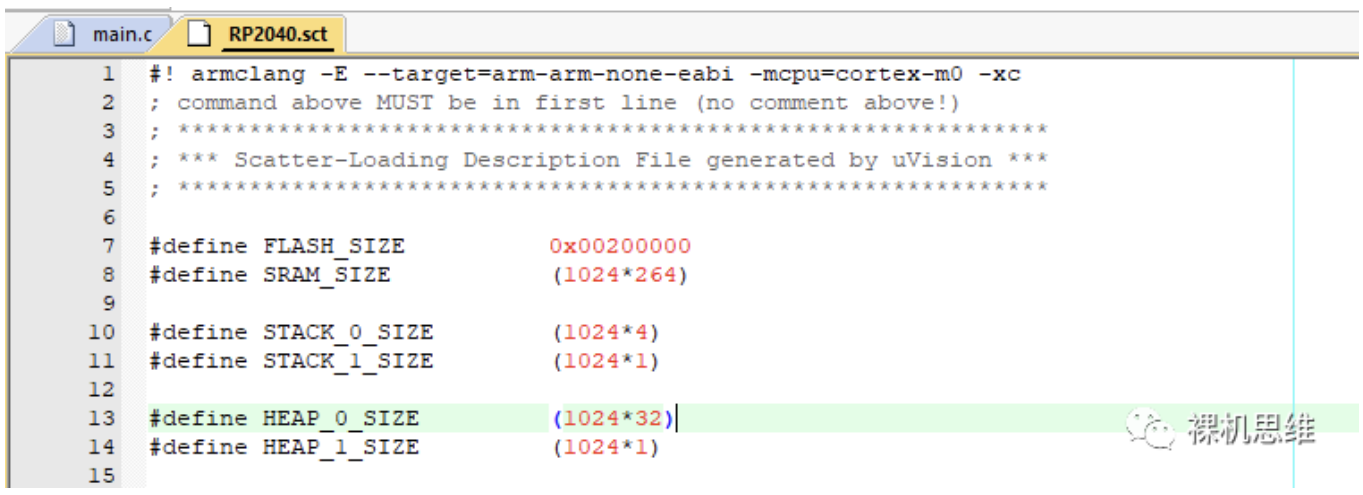
【如何配置栈和堆的大小】

一个实用的工程模板，最绕不开的问题之一就是：如何设置栈和堆的大小。Pico-Template提供了极其简单的方法。步骤如下：

1、打开Options for Target窗口，进入Linker选项卡：



单击图中红圈内选中的“Edit”按钮。



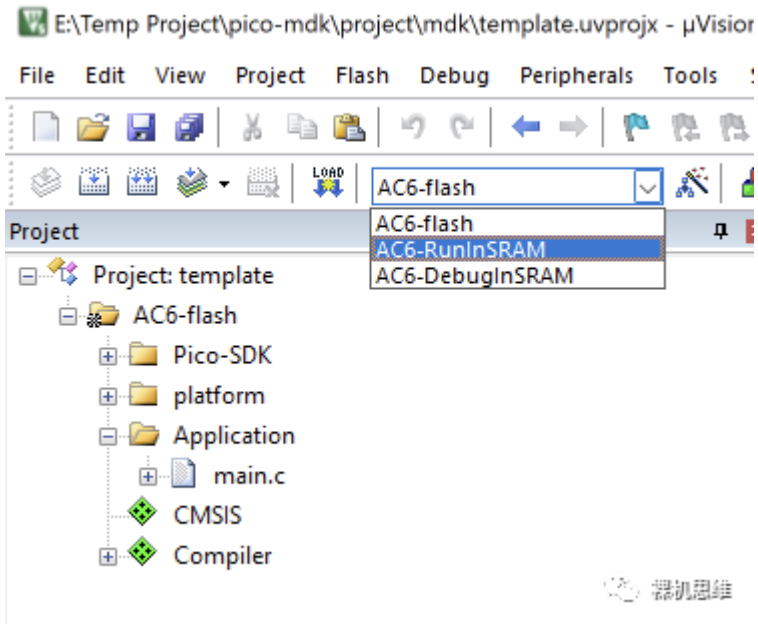
这里宏**STACK_0_SIZE**和**HEAP_0_SIZE**就是我们要配置的栈与堆的尺寸。**请暂时无视其它宏的内容，也不要修改它们。**

完成修改后，保存、重新编译即可。

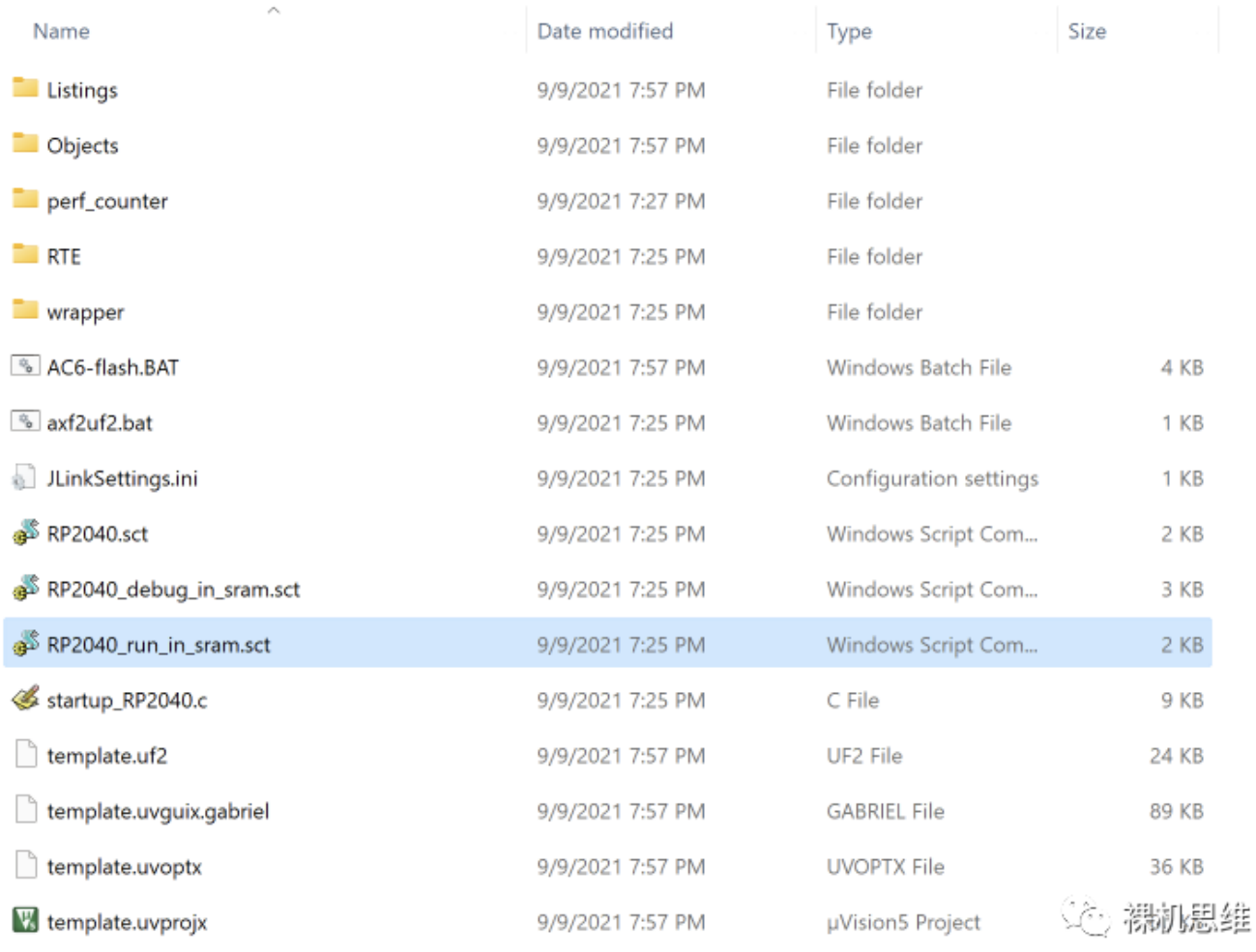
【如何在SRAM中执行代码】

由于RP2040芯片并没有片内Flash，因此通常会像Pico那样使用外部Flash来保存程序。由于RP2040的XIP已经将外部Flash的内容映射到了Cortex-M0+的地址空间中（从0x10000000开始），因此可以直接在外部Flash上执行代码。众所周知，外部Flash是通过SPI或者QSPI来连接的，其速度肯定无法媲美芯片内部的Flash，因此即便XIP有cache来提高速度，直接从0x1000-0000的地址上运行程序（或者是读取数据）显然存在性能上的瓶颈。

为了解决这一问题，在SRAM富余的情况下（RP2040带了264KB的SRAM）对一些小的应用来说，完全允许用户直接在SRAM中执行代码。为了提供这一功能，Pico-Template贴心的提供了对应的工程配置：我们可以在下拉列表中直接一键切换：



这一操作的本质实际上是更换了对应的scatter-script脚本。所有用到的链接脚本都保存在工程目录下：



有兴趣的小伙伴可以自行把玩。

【如何使用MDK进行调试】

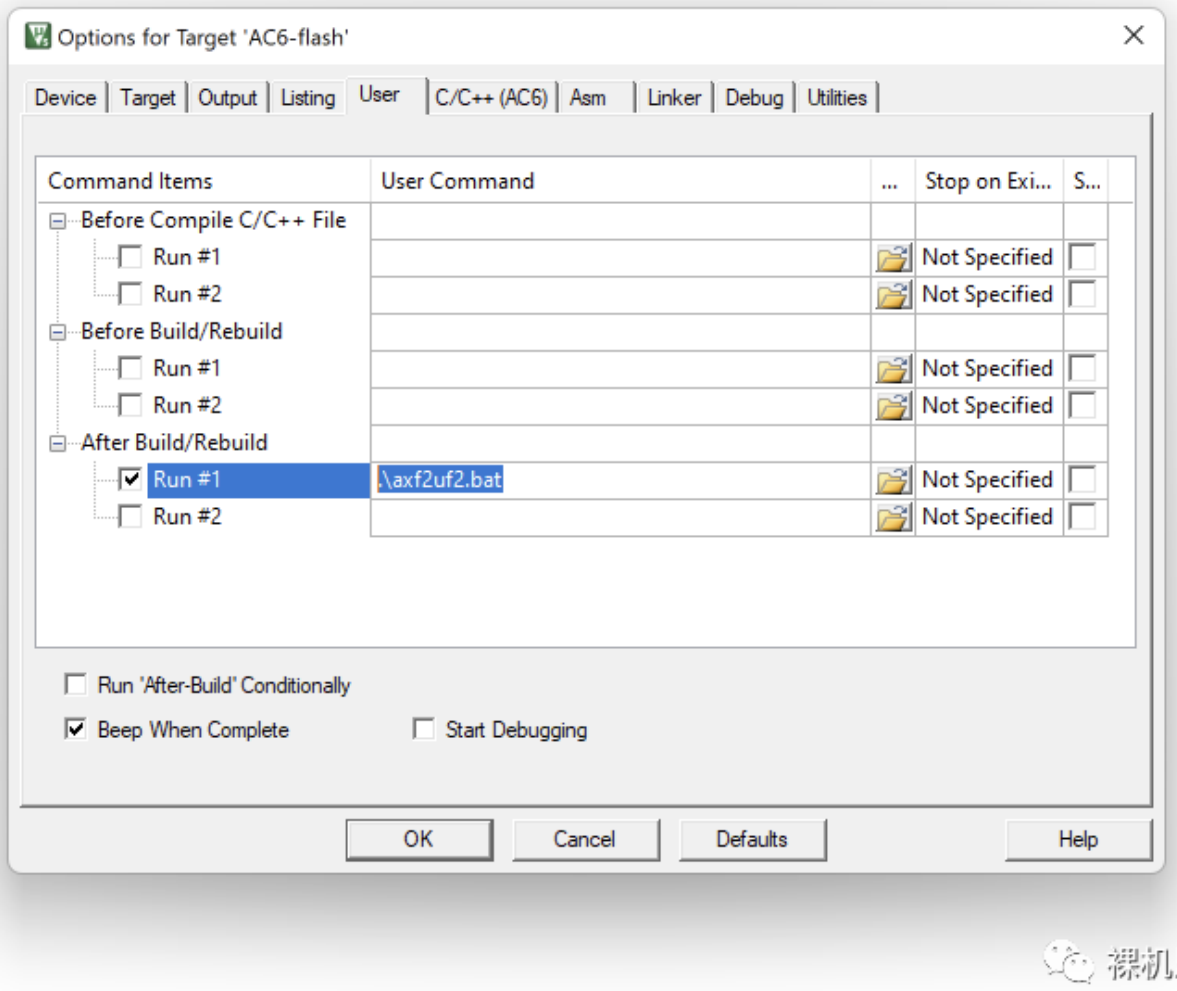
在众多的商业调试工具中，Segger 的J-Link很迅捷的就添加了对RP2040的调试支持，具体细节可以通过下面的网址来了解：

https://wiki.segger.com/Raspberry_Pi_Pico

Pico-Template默认已经选择J-Link作为调试工具。需要注意的是，并非所有的J-Link都能支持RP2040的调试，按照官方的说法，只有v9版本的J-Link硬件才有对应的功能。**如果你手头正好有符合要求的J-Link，恭喜你，获得了完整的MDK体验——基本告别了手动拖放uf2文件，调试全靠LED的生活。**

【elf2uf2转换工具】

最后，值得特别说明的是，在Pico_Template的tool目录下有一个我亲手定制过的elf2uf2.exe——增加了自动计算0x1000-0000地址开始的252个字节的CRC32校验码，并将校验结果追加其后的功能——如果不这么做，生成的uf2将无法通过stage2-boot的校验。



工程模板会在每次编译完成后执行 **axf2uf2.bat**，将生成的elf/axf文件转换成Pico可以直接使用的uf2文件，方便用户进行U盘拖放操作。

这个模板已经涵盖了除tinyUSB支持以外的几乎大部分功能，成功的将Pico以普通Cortex-M0+的身份拉回了国内大部分嵌入式工程师所熟悉的开发环境中。