**[自制JLINK-V9（多图预警）](https://3ric.xin/wordpress/index.php/2017/12/10/jlink-v9/)**

* 发表于：[2017年12月10日](https://3ric.xin/wordpress/index.php/2017/12/10/jlink-v9/)
* 标签：[JLink](https://3ric.xin/wordpress/index.php/tag/jlink/), [嵌入式开发](https://3ric.xin/wordpress/index.php/tag/%e5%b5%8c%e5%85%a5%e5%bc%8f%e5%bc%80%e5%8f%91/), [开源](https://3ric.xin/wordpress/index.php/tag/%e5%bc%80%e6%ba%90/), [硬件](https://3ric.xin/wordpress/index.php/tag/%e7%a1%ac%e4%bb%b6/)

2017年12月26日22:12:35更新：添加了可自动升级固件以及其说明  
2018年2月6日17:42:54更新：添加了对于相关权限的设置和说明  
2018年3月30日09:45:57更新：更新了基本无Bug版JLink（仅SW下载模式）

———————————–原文————————————–

STM32是大家用的比较多的主控之一。我们可以通过串口或者下载器给32烧程序。但是串口的速度实在是。。。所以大家基本都用下载器下载。鉴于队里软件组的同学没有下载器，并且下载器都是20PIN-JTAG接口，不方便使用，我决定自制一个下载器。

市场上常见的下载器有**ST-Link**、**JLink**、**ULink**等等。ST-Link使用STM32F103C8T6作为主控，网上资料较全，易于自制，但是软件组同学觉得下载速度太慢了。。。呃。。。没办法，选择JLink吧。

去[JLink官网](https://www.segger.com/)看了看，这价格跟55块的ST-Link比起来大概可能一般没人愿意买吧

再看看淘宝呢

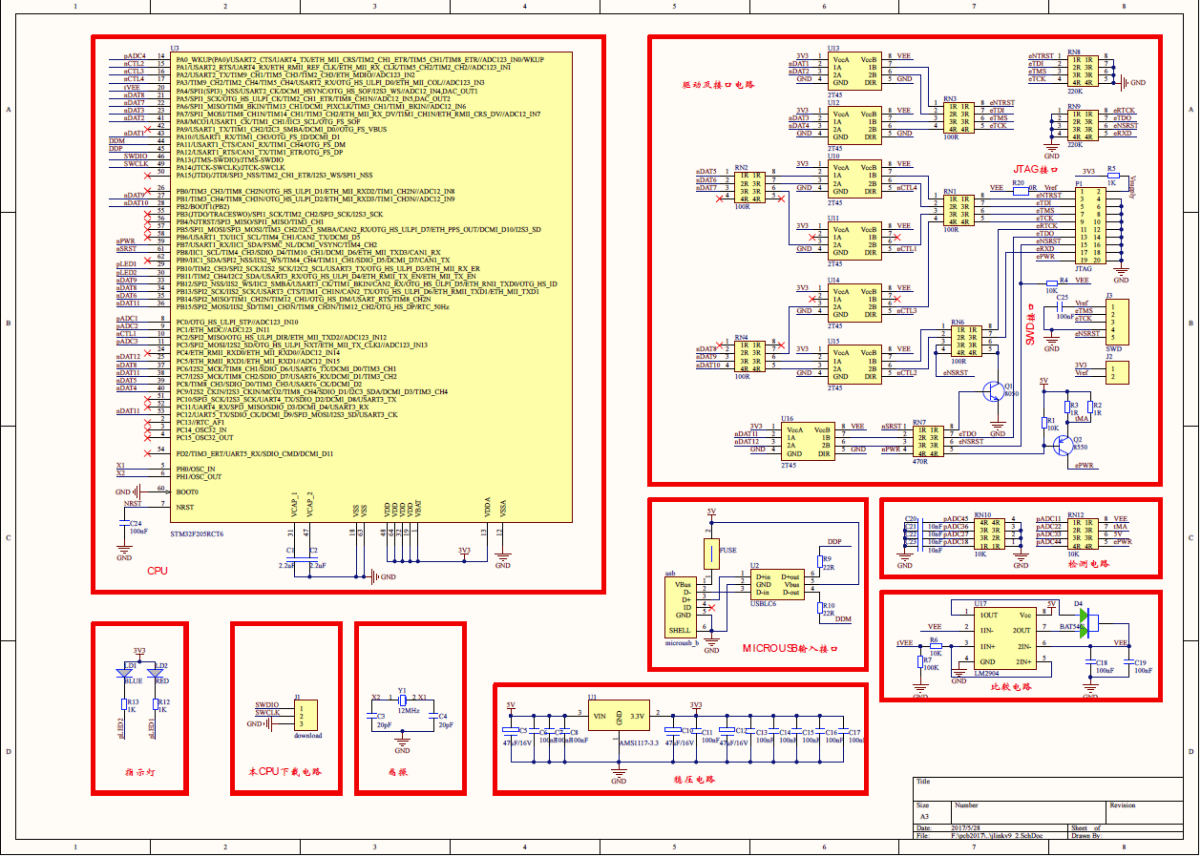
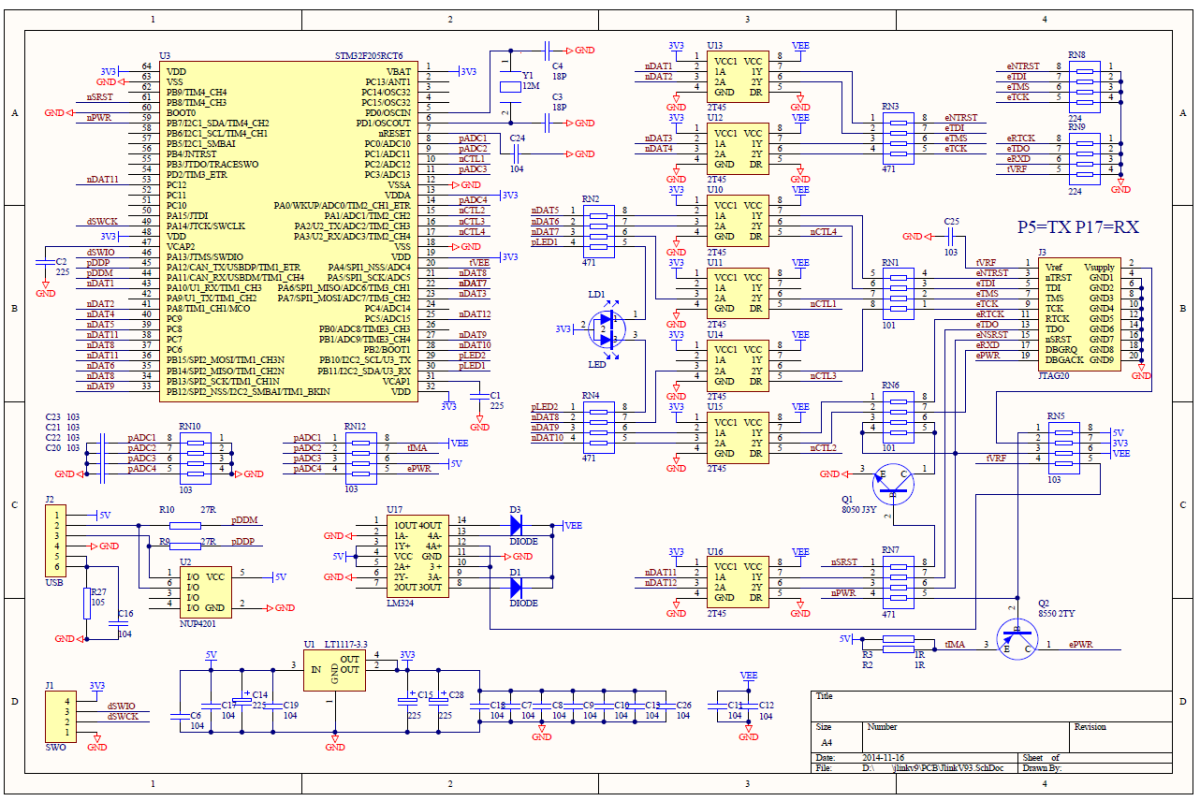
虽然跟正版比起来已经便宜很多了，但是自己做一个更有成就感吧。

JLink有这些常见的版本：

* **JLink-V9** 这是目前比较新的版本（貌似V10也出了，但是不好破解），网上资源较多，但是每张原理图都有细微的差别。采用STM32F205RCT6作为主控。
* **JLink-V8及以下**V8网上也有很多资源，不过要做就做新版。。。采用AT91SAM7S64作为主控。
* **JLink-OB**怎么说呢，这玩意可以做到很袖珍，不过新版的JLink驱动会报错，不喜欢。。。采用STM32F103V8T6作为主控。

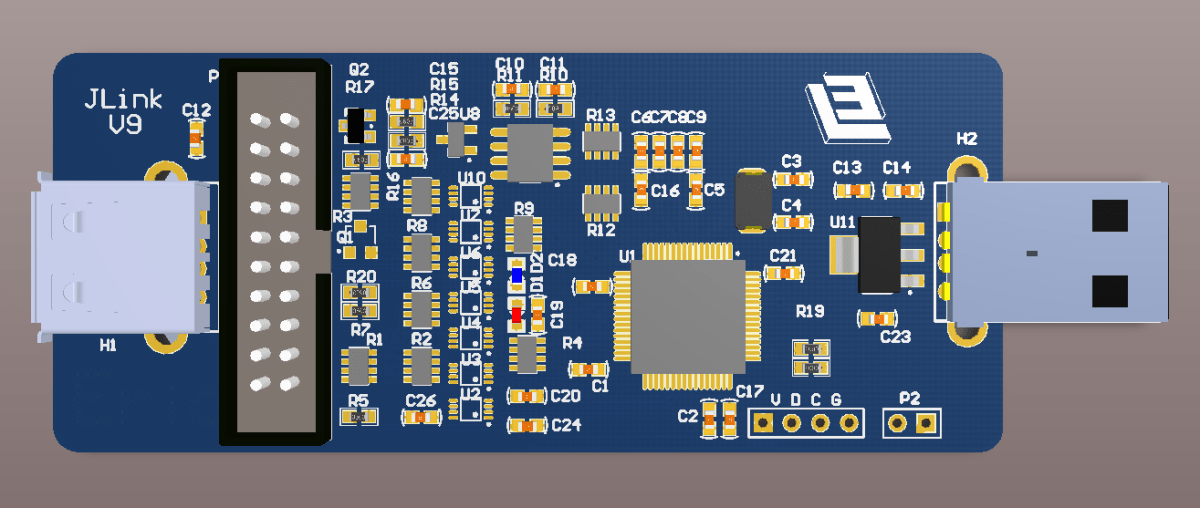
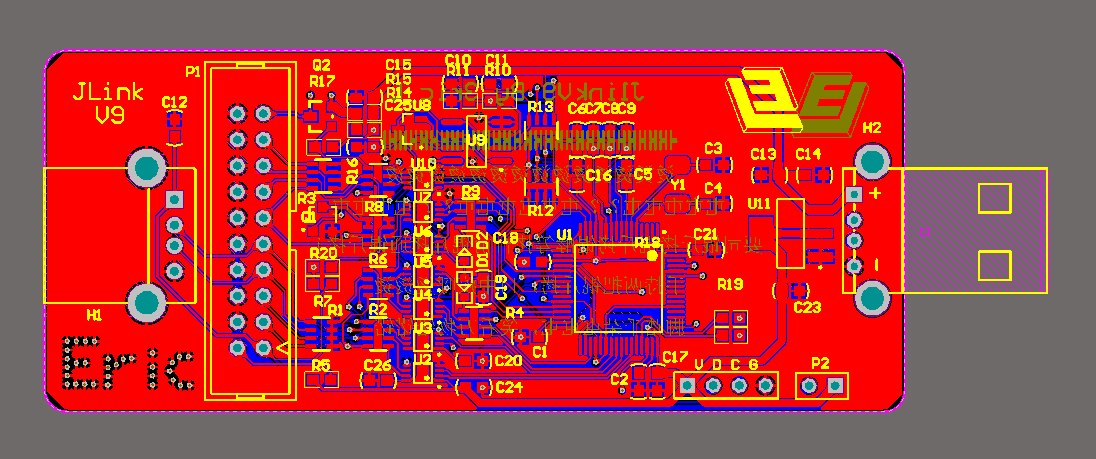
在查阅无数资料后，我选择自制JLink-V9.？（我也不知道具体是什么版本。。。）

感谢前人无私的奉献，我在网上找到了自制JLink必需的两样东西：**原理图**、**固件。**原理图我下载了很多，最终选择了看起来比较靠谱的两张：



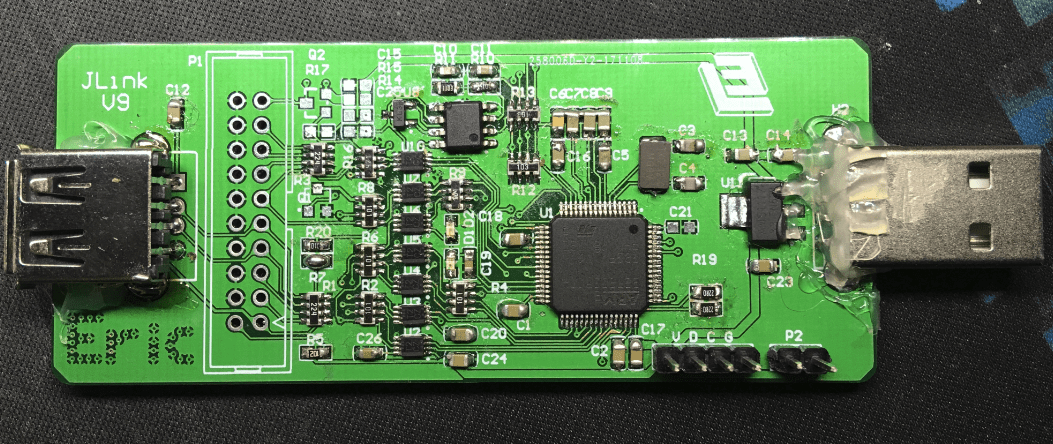
以下图为主，上图为辅，互相对照参考。画图布线过程就不再赘述了。还是想吐槽一下，AD2017真的很不稳，也可能是我电脑太渣了，画着画着就崩溃，重新打开，前功尽弃。。。于是乎我画的原理图就这么不见了。还好AD自动备份了我的PCB图，要不然。。。

画完过后大概是这样：



输入端直接插入电脑，输出使用一根MicroUSB线连接至板子。

打样焊接过程也不做赘述，最终效果如下：



可以看到有些元件我没有焊上去，后面再解释。

最后一步是刷固件。使用网上大神制作的固件生成工具，得到JLinkV9的固件，用一个现有的JLink，使用JFlash通过预留的SW下载口烧录至主控。

激动人心的时刻到了。如果原理图、焊接、固件都没有问题，那么我们把JLink插入电脑，则会在设备管理器中看到JLink设备，打开JCommand，可以看到相关信息，打开Keil，下载程序，一切正常。

目前实现功能：**SW下载调试**

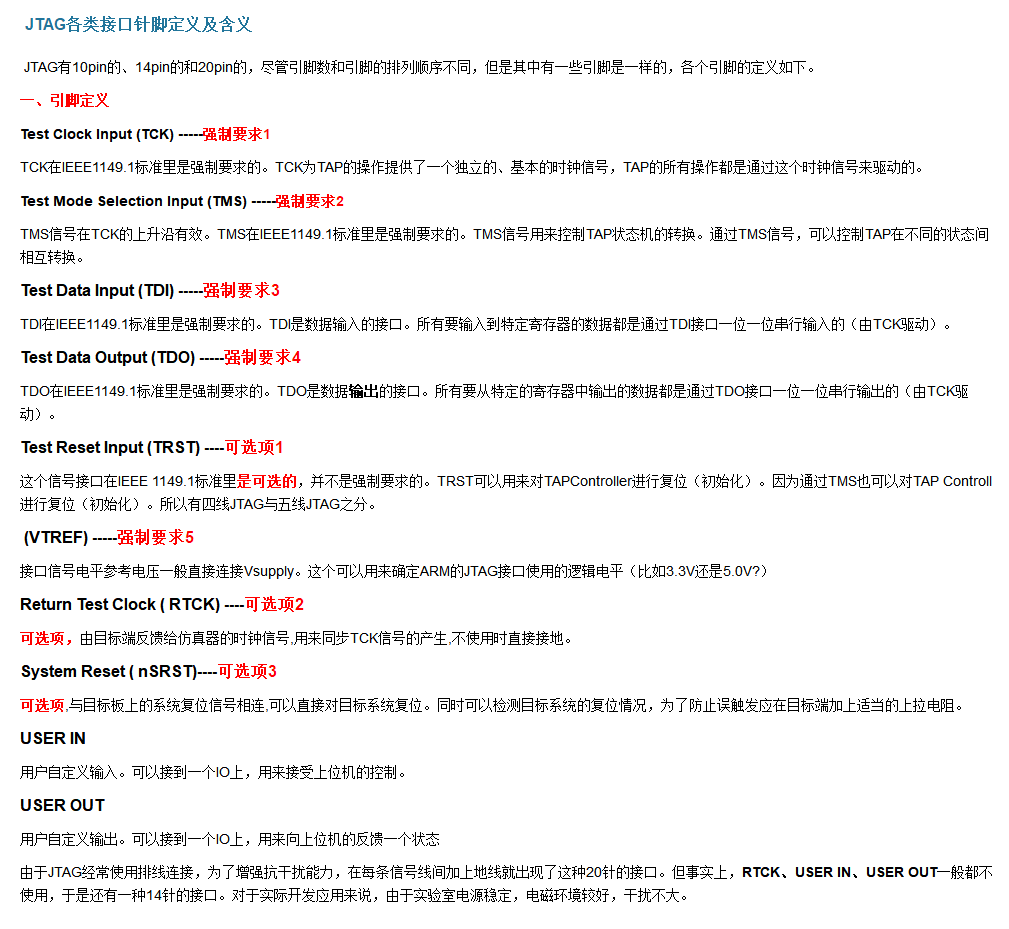
未测试功能：**JTAG下载调试**

因为板子上都是留的SW下载口，所以我就没焊接JTAG口以及相关外围元件了。能不能用需测试。

已知BUG：只能选择3.3V输出供电，需改原理图达到5V输出；板子上外部电池供电时，电压会通过MicroUSB口供至JLink上。 可能是Vref引脚那里跳线的问题；**无法自动升级固件**，因为没有BootLoader。

改进意见：JTAG用不到的话，可以去掉JTAG口以及相关外围元件，把板子变得更小一些。

找到一篇[讲JTAG口的文章](https://www.cnblogs.com/jeakon/archive/2012/10/07/2813683.html)，感觉很不错：



成本算下来，大概50左右吧，比淘宝便宜，但是手工费无价啊，身体要紧（雾）~

已上传资料 链接: <https://pan.baidu.com/s/1geMH7Y7> 密码: n77v

资料仅供研究学习，请勿商业化（毕竟是盗版人家东西）

转载请注明出处

———————————–更新————————————–

不能自动升级，对于我这个追求完美的人来说是一个致命的缺陷。百度了一波资料过后，发现amoBBS居然有牛人公布了[读取JLink Bootloader的程序](https://www.amobbs.com/thread-5653964-1-1.html)。真的机智，真的，真的！！！！！！巧妙利用了JLink驱动自带的命令读出了Bootloader，真的机智啊！！！！！

（虽然大家对站长的评价有点不好，但是论坛还是有不少干货。。。求一波注册码）

在众多的回复中，甚至有人直接贴出了完整可编译的程序，现摘录如下：

1. //gcc usb.c `pkg-config libusb-1.0 --libs --cflags` -o usb
2. #include <errno.h>
3. #include <signal.h>
4. #include <string.h>
5. #include <stdio.h>
6. #include <stdbool.h>
7. #include <stdlib.h>
8. #include <stdint.h>
9. #include <unistd.h>
10. #include <sys/select.h>
11. #include <termios.h>
13. #include <libusb.h>
15. #define EP\_DATA\_IN (0x3|LIBUSB\_ENDPOINT\_IN)
16. #define EP\_DATA\_OUT (0x3|LIBUSB\_ENDPOINT\_OUT)
18. **static** **struct** libusb\_device\_handle \*devh = NULL;
19. **static** **struct** libusb\_transfer \*recv\_bulk\_transfer = NULL;
21. **bool** jlinkSendCommand( **void** **const**\* commandBuffer, uint32\_t commandLength, **void**\* resultBuffer, uint32\_t resultHeaderLength)
22. {
23. **int** transferred, r;
24. r = libusb\_bulk\_transfer(devh, EP\_DATA\_OUT, commandBuffer, commandLength, &transferred, 200);
25. // if(!WriteFile( commandBuffer, commandLength, nullptr, nullptr))
26. // return false;
28. **if**(!resultHeaderLength)
29. **return** **true**;
31. // return !!ReadFile( resultBuffer, resultHeaderLength, nullptr, nullptr);
32. **return** 0==libusb\_bulk\_transfer(devh, EP\_DATA\_IN, resultBuffer, resultHeaderLength, &transferred, 200);
33. }
35. **bool** jlinkContinueReadResult(**void**\* resultBuffer, uint32\_t resultLength)
36. {
37. **int** transferred, r;
38. // return !!ReadFile( resultBuffer, resultLength, nullptr, nullptr);
39. **return** 0==libusb\_bulk\_transfer(devh, EP\_DATA\_IN, resultBuffer, resultLength, &transferred, 200);
40. }
41. **bool** jlinkCommandReadFirmwareVersion(**void**\* dataBuffer)
42. {
43. uint8\_t commandBuffer[1] = {0x01};
44. uint16\_t leftLength = 0;
45. **if**(!jlinkSendCommand( commandBuffer, **sizeof**(commandBuffer), &leftLength, **sizeof**(leftLength)))
46. **return** **false**;
48. **return** jlinkContinueReadResult( dataBuffer, leftLength);
49. }
50. **bool** jlinkCommandReadEmulatorMemory(uint32\_t address, uint32\_t length, **void**\* dataBuffer)
51. {
52. uint8\_t commandBuffer[9] =
53. {
54. 0xfe,
55. (uint8\_t)(address), (uint8\_t)(address >> 8), (uint8\_t)(address >> 16), (uint8\_t)(address >> 24),
56. (uint8\_t)(length), (uint8\_t)(length >> 8), (uint8\_t)(length >> 16), (uint8\_t)(length >> 24),
57. };
59. **return** jlinkSendCommand( commandBuffer, **sizeof**(commandBuffer), dataBuffer, length);
60. }
61. **int** main(**int** argc, **char** \*\*argv)
62. {
63. **int** r = 1;
65. r = libusb\_init(NULL);
66. **if** (r < 0) {
67. fprintf(stderr, "failed to initialise libusb\n");
68. exit(1);
69. }
71. devh = libusb\_open\_device\_with\_vid\_pid(NULL, 0x1366, 0x0105);
72. **if** (devh == NULL) {
73. fprintf(stderr, "Could not find/open device\n");
74. **goto** out;
75. }
77. r = libusb\_claim\_interface(devh, 2);
78. **if** (r < 0) {
79. fprintf(stderr, "usb\_claim\_interface error %d\n", r);
80. **goto** out;
81. }
82. printf("claimed interface\n");
84. **char** version[128];
85. **bool** ret = jlinkCommandReadFirmwareVersion(version);
86. printf("ver[%d] %s\n", ret, version);
88. uint8\_t bootloader[0xb700] = {0};
89. ret = jlinkCommandReadEmulatorMemory( 0x08000000, **sizeof**(bootloader), bootloader);
90. printf("read[%d]\n", ret);
92. **FILE**\* fBld = fopen("bootloader.bin", "wb");
93. fwrite(bootloader, 1, **sizeof**(bootloader), fBld);
94. fclose(fBld);
96. out\_release:
97. libusb\_release\_interface(devh, 2);
98. out:
99. libusb\_close(devh);
100. libusb\_exit(NULL);
101. **return** r >= 0 ? r : -r;
102. }

该代码需在Linux下编译运行，因为用到了一些奇奇怪怪的头文件，而且Linux也得装JLink驱动。需要一个某宝买的可自动升级固件的JLink（或者正版也行）。

已知问题：

*由于大家的高调，此命令已经需要用开关开启才可以使用，而且还加了扰码，以后将再无直接见到固件的机会了……  
当然，降版本大法还是有效的……——taoist*

大概意思就是高版本固件用到了一些加密，没办法直接读出Bootloader，解决办法呢：

1. 先使用CMD\_SET\_EMU\_OPTION写入一个KEY，KEY不为0的话就能通过CMD\_READ\_EMU\_MEM读出，读出的数据经过这个KEY作了极为简单的加密  
   具体细节还请自行看反汇编……——hackerboygn（我不太懂这个解密。。。）
2. 使用旧版本JLink。。。还好淘宝买的固件很旧，顺利读出了Bootloader。

原理如下：

*jlink的bootloader占用的是0x08000000到0x08010000之间的64k内容，其中从0x0800b700开始到0x0800c000的部分占据0x900个字节，jlink管他称为ots，大概是什么one time section的缩写？我们在读取bootloader的时候完全可以跳过ots和config部分，读取前面的0xb700个字节就可以了（实际上bootloader只有不到0x4000个字节）*

读出来一个二进制文件，然后用JFlash烧录进自制JLink就行了。插入只有Bootloader的JLink，打开JCommander，会提示“该JLink只有Bootloader”，然后就可以自动升级固件了~~~~

貌似S/N码会显示-1，因为算是刚出厂，得自己设置一下，输入命令 exec setsn=xxxxxxxx即可，可随意设置（八位数字）。

有趣的是，JLink的硬件版本是通过S/N计算出来的。。。emmmmmm。。。。。

*接下来我们来看看序列号，他位于bf00，这个序列号用的地方有两个，一个自然就是序列号，另外一个是硬件版本，是的你没有看错。  
硬件版本是用序列号来计算的，首先把序列号除以100000，得到的商再除以10，得到的余数如果大于等于8则取2，得到的就是子版本号。  
比如我这个盗版设备，sn=59101308，他的硬件版本显示为9.10  
sn先除以100000，商591，再除以10，余1，所以就是9.10  
当序列号这个位置4个字节都是ff的时候，我们可以使用exec setsn=xxx来设置一个序列号，如果序列号已经有值了，这个命令就不能用了。*

我设置的S/N是23333333，所以硬件版本是V9.3。。。有趣。。。

之前提到的bug都可以修改。。。不知道什么时候有时间呢。。。

Bootloader下载：[bootloader](http://120.79.34.16/wordpress/wp-content/uploads/2017/12/bootloader.zip)

（发现了个神奇的东西：[把STLink刷成JLinkOB](https://www.segger.com/downloads/jlink/" \l "STLink_Reflash)。。。Segger官方出品，最为致命，这是不给STLink活路啊）

资料仅供研究学习，请勿商业化（毕竟是盗版人家东西）

转载请注明出处

——2017年12月26日23:57:51 By 3ric

———————————–更新————————————–

最近批量生产了一波，由于用不到JTAG模式，索性就去掉了相关元件的焊接，然后就意外修复了逆向供电的bug。。。

准备有时间重画一版精简版，做的小一点。

用JFlash时发现了一个严重的bug：下载程序时提示没有license，所以没办法用JFlash（。。。然而Keil和VS以及JLinkCommander依旧可以正常使用，猜测他们用的是JLinkARM.dll的开放API）。插上某宝买的成品，在JLinkCommander下使用命令license show，发现果然需要自己手动添加license。

这是应该显示的license信息：

Built-in licenses: RDI,FlashBP,FlashDL,JFlash,GDB  
No installable licenses.

可以看到，提供了相关软件的使用权限。

所以，我们得手动添加相关license进入我们的JLink。

使用命令 exec addfeature RDI, exec addfeature FLashBP…..一条一条添加即可。

P.S.没想到JFlash还可以逆向读出32内的程序。。。。。这为盗版提供了便捷的途径啊。。。

（可能是我没有进行保护？？？所以我该怎么保护32内的程序呢？？？）

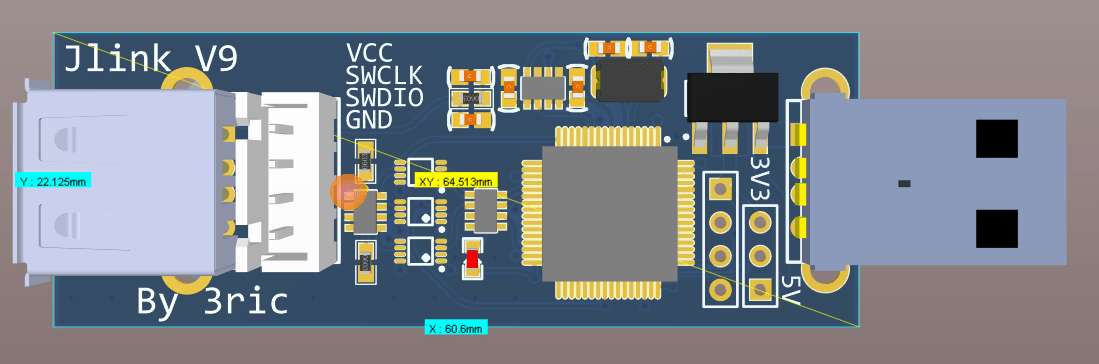
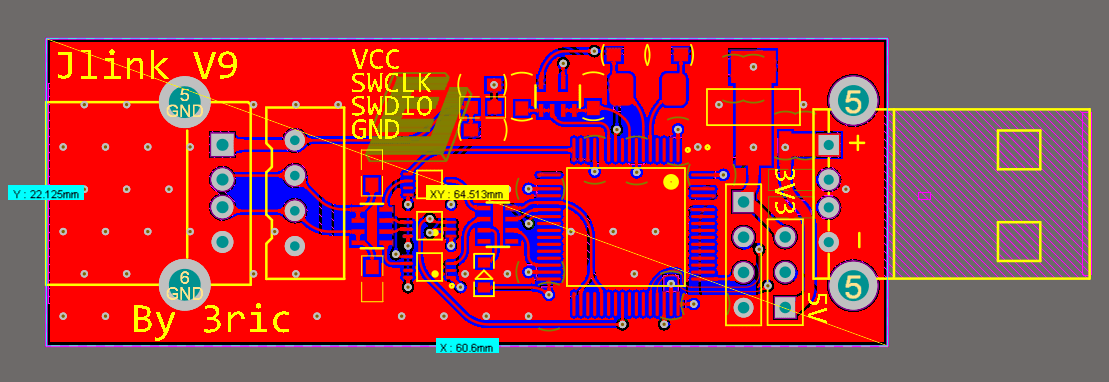
——2018年2月6日17:42:54 By 3ric

———————————–更新————————————–

最近有空修复了一波已知Bug，然后重新画了下板子，让它变得更小了（因为之前那样插在电脑上会把相邻的USB口挡住。。。）。

因为大多数情况都用不到JTAG口。所以干脆就把它去掉了，只用到了SW下载模式，所以板子可以变得很小，大概和JLink-OB一样大小。

上图：



只用到了三个2T45和三个排阻，元件数量大大下降。。。感觉f2的被浪费了好多IO脚。。。

成本呢，要是不拼板的话，大概是27.78（f2）+ 2.12\*3（SN74LVC2T45DCUR）+ 5（别的元件）+ 6.9（一块PCB）= 46.04¥

感觉之前成本算的50算少了。。。不过还是很划得来了。

P.s.

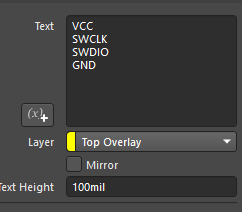
这次就不提供PCB文件了，自己动手，丰衣足食。这次布线是直接在原有PCB上删删删，然后改网络标号，然后布线。。。没有原理图也能搞(￣▽￣)／

在给f2下程序的时候遇到一个问题，那就是JFlash可以连接上32，但是一烧写就会报fail to download ramcode的错误。。。本来都打算放弃了，然后把JLink插到电脑上或者任何能供电的地方，程序就烧进去了。。。不知谁能解释一下这是为什么啊。。。难道是因为直接3.3V供电会导致32不稳定？所以得5V供电转成3.3V？之前也有板子遇到过这种情况。。。毒。。。

还有就是最近换成AD18了，确实要好用了一些，但是有些界面的位置变了，操作方式也变了，需要熟悉一下才行。然后就遇到了一个Bug：



emmmmmmmmm。。。。。本来我这四个是写在一个丝印里的，用回车换行，然后嘉立创那边把换行符给我导成了乱码。。。



然后我问他这咋办，他说你用低版本AD。。。emmmmmmmmmmm。。。无奈。。。

不过总的来说还是很开心的٩(๑>◡<๑)۶ ~~~

——2018年3月30日09:45:57 By 3ric