

32 位 RSIC 架构的微控制器

应用注意事项

V1.0

目录

1. 问题反馈及解决方法.....	1
2. 注意事项.....	1
2.1 ADC 硬件数据校准功能	1
2.2 PC6/PC7 使用注意事项	1
2.2.1 PC6/PC7 作为通用 IO 的配置步骤.....	1
2.2.2 重新找回 SWD	2
2.2.3 注意事项	2
2.3 查看 NVR 配置区域数据的步骤	3
2.4 用户数据区的操作步骤及注意事项.....	4
2.4.1 简要介绍	4
2.4.2 操作步骤	5
2.4.3 注意事项	5
2.5 UART 单 PIN 升级 IO 用作其它功能	5
2.6 数码管/LED 驱动注意事项.....	6
2.7 TOUCH-KEY 使用注意事项	6
2.8 电源外挂电容注意事项	6
2.9 芯片连接 J-LINK 注意事项.....	7
2.10 用户程序和 BOOT 的 J-LINK 烧写注意事项	7
2.10.1 用户程序烧写	7
2.10.2 BOOT 烧写.....	9
2.11 用户 MAIN 区域 EFLASH 保存注意事项	11
2.12 定时器精确定时的使用方式	12
2.13 TK DEMO 程序说明	13
2.14 TK 与 LED 程序配置说明	13
2.15 芯片配置定义	16
2.16 不使用 BOOT 的方法.....	16
2.17 关于 ADC 的精度问题	16
2.18 关于修改 APP_EFLASH.HEX 和 APP_EFLASH.BIN 的文件名	17
2.19 关于 PC6 被用作普通 IO 及其他功能时的注意事项	17

1. 问题反馈及解决方法

序号	问题反馈	解决方法
1	使用 28PIN 芯片时没有使能 PB remap，导致 PB 口输出功能不对应	修改 SDK，初始化时默认打开 PB、PC remap，用户无需另外配置
2	使用 28PIN 芯片 SDK 把 MCLR 功能使能，导致不停复位，程序无法运行	修改批处理文件，用户只需要使用最新的批处理文件，无需另外处理
3	PC6、PC7 外挂 LED 灯导致 SWD 连接和程序烧写受影响	参考 2.2 《PC6/PC7 的使用注意事项》
4	ADC 采样数值与理论计算值偏差较大	参考 2.1 《ADC 硬件数据校准功能的说明》

2. 注意事项

2.1 ADC 硬件数据校准功能

硬件数据校准功能是为了让 ADC 实际采样值更接近理论值，关闭该功能后 ADC 采样值和理论值误差在 10mv 以内，开启该功能后误差可控制在 5mv 以内。

2.2 PC6/PC7 使用注意事项

2.2.1 PC6/PC7 作为通用 IO 的配置步骤

- 1) 在用户程序中失能 SWD;

```
SYSCTRL_REG_OPT (  
    SYSCTRL->SYS_CON1 &= ~LL_SYSCTRL_CON1_SWD_EN; //disable SWD  
);
```

- 2) 在用户配置文件中失能 SWD

文件路径: jsh300x_sdk\Project\JSH300x_Project_Template\LL\KEIL-ARM\Eflash_Proj

\makecode.ini (将图中的 SYS_SWD_EN =1 改为 SYS_SWD_EN =0);

注意: 需要把 SWD_Remaping_en 需要配置为 0;

```
[EFLASH_NVR]
CRC32_EN=1

[EFLASH]
CHIP_ID=00000000
EXE_ADDR=00000000
CODE_COPIES=1
CODE_LEN=7E00
CRC32_EN=1
NVR0_EW_EN=1
NVR1_EW_EN=1
NVR2_EW_EN=1
MAIN_EW_EN_BITMAP=FFFFFFFF
EXTERNAL_KEY_DIS=1
EXTERNAL_KEY=0
USERDATA_AREA_CNT=3
UART_BOOT_EN=0
NVR_CODE_BAK_EN=1
NVR_CODE_BAK_SECTOR_ADDR=3C
UART_BOOT_PIN_SEL=E
SWD_Remapping_en=0
SYS_MCLR_EN=0
SYS_SWD_EN=1
SYS_SWD_IO_PU_EN=1
MAIN_CODE_CRC32_HWCHECK_EN=1
```

3) 下载程序到芯片，SWD 功能被关闭

准备代码，检查代码并编译链接代码；使用 J-flash 工具下载生成的 app_eflash.hex，按照上面的步骤操作，代码下载完成后 SWD 功能被关闭，IO 功能使用正常；

注意：PC6\PC7 的 IO 使用，需要用户按照通用 IO 去配置 PC6\PC7；

4) PC6 默认电平

jsh300x_sdk\Project\JSH300x_Project_Template\LL\KEIL-ARM\Eflash_Proj
\makecode.ini 中 SYS_SWD_IO_PU_EN =1 表示芯片 PC6 开机默认上拉。

2.2.2 重新找回 SWD

1) 关闭开发板的电源（电源包括 J-Link）；

2) 进入超级擦写模式找到 SWD

① jsh300x_sdk\Project\JSH300x_Project_Template\LL\KEIL-ARM\Eflash_Proj

\makecode.ini 配置文件中 SYS_SWD_IO_PU_EN=1

在断电的情况下将 PC6 接到 GND,PC7 和 PA3 接到 5V 电源上；

jsh300x_sdk\Project\JSH300x_Project_Template\LL\KEIL-ARM\Eflash_Proj

\makecode.ini 配置文件中 SYS_SWD_IO_PU_EN=1

在断电的情况下将 PC6, PC7 和 PA3 接到 5V 电源上；

② 上电，取下所有的连线（PC6\PC7 和 PA3），将 PC6\PC7 接到 J-Link 上寻找 SWD

3) 找到 SWD 后使用 J-Link_main_erase.bat 文件擦除 main 区域，使失能 SWD 的配置恢复到默认值；

4) 擦除成功后断电上电进入正常模式；

2.2.3 注意事项

1) PC7 和 PC6 之间不能放置二极管；

2) 使用超级模式前必须先断电再接线，不然很容易烧毁芯片；

3) SWD_Remapping_en 需配置为 0

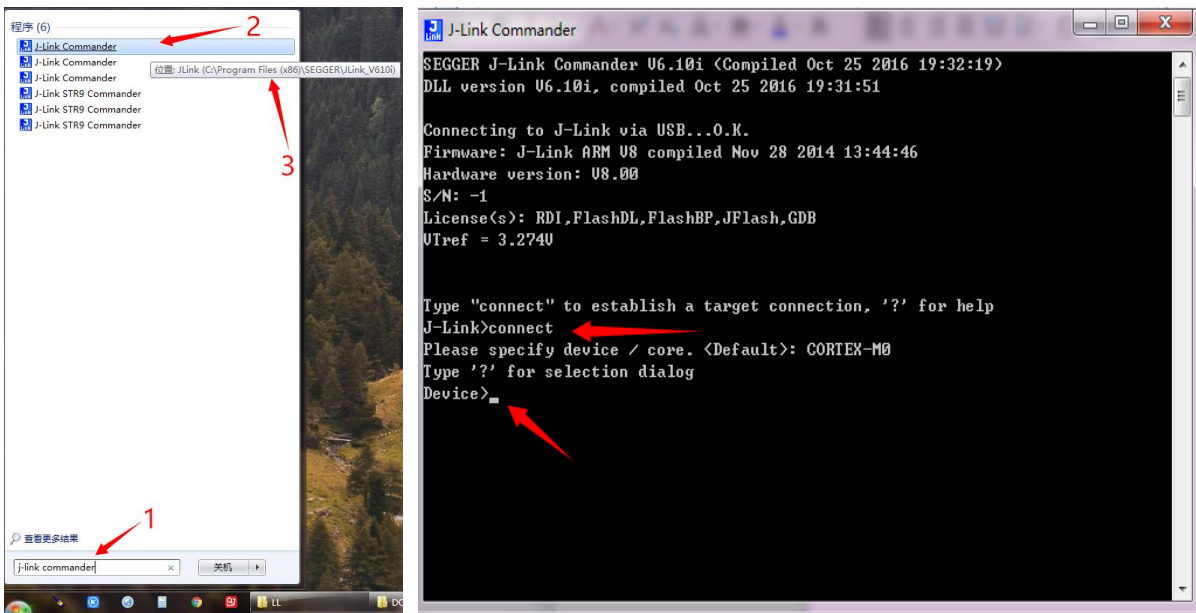
2.3 查看 NVR 配置区域数据的步骤

1) 打开开发板的电源并连接好 J-Link，保证找到 SWD；

2) 从 windows 开始界面搜索 J-Link 工具；

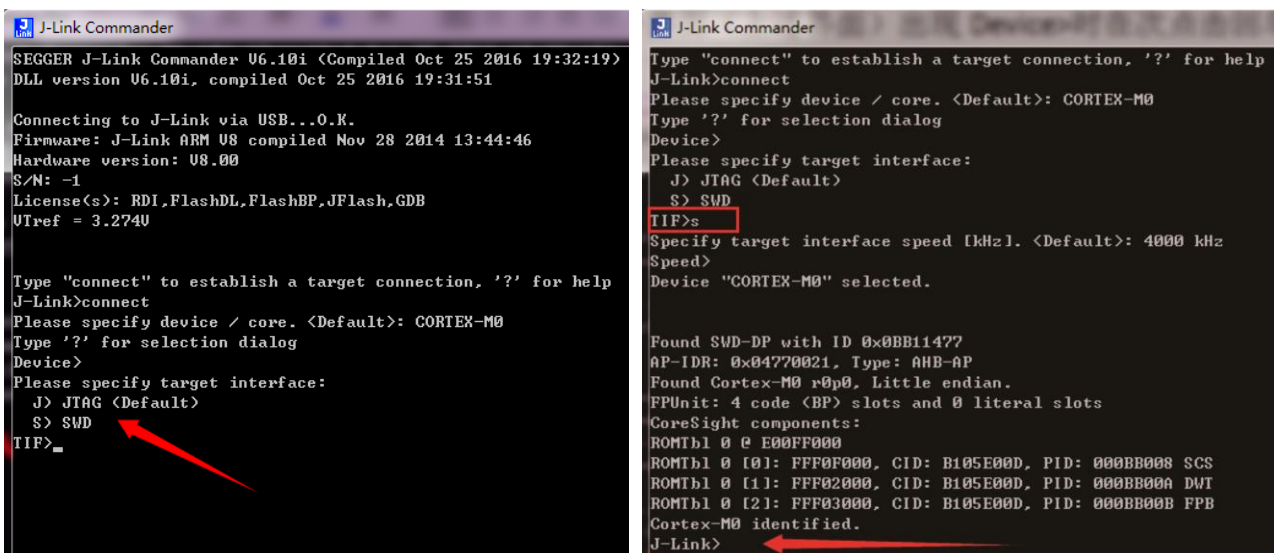
3) 找到（J-Link_v610i）驱动对应的 commander 工具；

4) 打开 J-Link commander 工具，输入命令 connect 点击回车键确认；



5) 当命令界面出现 Device>时，再次点击回车键；

6) 手动输入 s（不区分大小写），点击两次回车键；



7) 输入命令 mem32[空格][add][num]。如：mem32 0x1ff0620 2 再点击回车键，则可以读取 add 地址两个字的数据，最后截图发至我司对接人员；

```

J-Link Commander
Please specify device / core. <Default>: CORTEX-M0
Type '?' for selection dialog
Device>
Please specify target interface:
  J) JTAG <Default>
  S) SWD
TIP>s
Specify target interface speed [kHz]. <Default>: 4000 kHz
Speed>
Device "CORTEX-M0" selected.

Found SWD-DP with ID 0x0BB1477
AP-IDR: 0x04770021, Type: AHB-AP
Found Cortex-M0 r0p0, Little endian.
FPUnit: 4 code <BP> slots and 0 literal slots
CoreSight components:
ROMTbl 0 @ E00FF000
ROMTbl 0 [0]: FFF0F000, CID: B105E00D, PID: 000BB008 SCS
ROMTbl 0 [1]: FFF02000, CID: B105E00D, PID: 000BB00A DWT
ROMTbl 0 [2]: FFF03000, CID: B105E00D, PID: 000BB00B FPB
Cortex-M0 identified.
J-Link>mem32 0xff00620 2
1FF00620 = 00F00026 3FFF00F8
J-Link>

```

2.4 用户数据区的操作步骤及注意事项

2.4.1 简要介绍

- 1) 先了解文件 `makecode.ini`；该文件在 SDK 的位置为：
`jsh300x_sdk\Project\JSH300x_Project_Template\LL\KEIL-ARM\Eflash_Proj`，这是芯片后 64Byte 的配置文件；
- 2) 配置用户数据区大小（用记事本打开 `makecode.ini` 文件）；图中的 `USERDATA_AREA_CNT=3` 代表我们用户数据区域大小，即用作写数据的地方；

```

makecode.ini - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
Flag=5A69
Version=0x0100
CodeLoadToSramAddr=20000000
CodeExecAddr=20000000
CodeAddrInSPI=200
CodeLoadLen=2000
SPI_SIZE=100000
SPI_CLK_KHZ=3E8
PLL_SRC_MHZ=1A
PLL_EN=0
DebugInfoEn=0
AesEnable=0

[EFLASH_NVR]
CRC32_EN=1

[EFLASH]
CHIP_ID=00000000
EXE_ADDR=00000000
CODE_COPIES=1
CODE_LEN=7E00
CRC32_EN=1
NVR0_EN=1
NVR1_EN=1
NVR2_EN=1
MAIN_EN_BITMAP=FFFFFFFF
EXTERNAL_KEY_DIS=1
EXTERNAL_KEY=0
USERDATA_AREA_CNT=3
UART_BOOT_EN=0
NVR_CODE_BAK_EN=1
NVR_CODE_BAK_SECTOR_ADDR=3C
UART_BOOT_PIN_SEL=E
SWD_Remapping_err=0
SYS_MCLR_EN=0
SYS_SWD_EN=1
SYS_SWD_IO_PU_EN=1
MAIN_CODE_CRC32_HWCHECK_EN=1

```

- 3) 用户手册中描述: `USERDATA_AREA_CNT=3` 代表用户数据区有 4Kbyte-64byte(用户配置区)，实际上芯片可用的数据区只有 4Kbyte-512byte（一个 sector），因为擦除时不能擦除最后一个 sector；

5.2.5.3 用户配置 2 (0x7FC8)				
Width	Name	Reset	Property	Description
31:12	reserved	20' hFFFFFF	R0	保留
11:8	Read Region Size	4' hF	R0	DATA RAM 区域的大小 EFLASH 为 32Kbyte 时: 3' d0: 1 Kbyte 3' d1: 2 Kbyte 3' d2: 3 Kbyte 3' d3: 4 Kbyte 3' d4: 5 Kbyte 3' d5: 6 Kbyte 3' d6: 7 Kbyte 3' d7: 8 Kbyte Others: 无效

4) 用户数据区分配

需先看芯片代码大小，然后再去合理的分配芯片的数据区域；

2.4.2 操作步骤

- 1) 按照事先分配的大小改写配置文件 `makecode.ini` 中的 `USERDATA_AREA_CNT` 的值；
- 2) 在程序中按地址去读写；
- 3) 使用 keil 编译后，使用 J-flash 工具去下载编译生成的 `app_eflash.hex` 文件；

2.4.3 注意事项

使用 J-flash 工具下载我们的 hex 文件（即改写过用户配置区的 `app_eflash.hex`）后，整个芯片受保护不允许读写，但是芯片的用户数据区可允许读取，这不仅做到了保护代码安全，又实现了故障信息的读取。

2.5 UART 单 PIN 升级 IO 用作其它功能

- 1) 从用户手册上查找同时支持 `uart0_rx` 和模拟输入的 pin 脚（务必要输入 pin），此 pin 脚可用于 uart 单 pin 升级复用模拟输入功能。
- 2) 用于 uart 单 pin 升级时，电路设计需要能把此 pin 脚独立接在升级串口上，不能有外部干扰，pin 脚上不能加滤波电容和限流电阻（可考虑预留滤波电容和限流电阻的位置，焊接 0 欧电阻，但不焊接电容）。升级用到电脑 usb 转串口单 pin 小板（单 pin 小板的单 pin 电平要和被升级设备单 pin 上的工作电平一致。usb 转串口单 pin 小板和主板共地，被烧写设备主板上芯片供电电压和 usb 转串口单 pin 小板的 vcc 一致），单 pin 小板的升级单 pin 脚接在被升级 pin 脚上，pc 端用升级工具进行升级。升级时会把 main 区域全部擦掉重新升级成最新代码。Sdk 编译工具链会在...\\...\\Eflash_Proj 目录下生成一个 `app_eflash.bin` 的文件为升级工具使用。
- 3) 用 uart 单 pin 升级功能前需要先烧写 boot 程序和用户应用程序。Sdk 开发包里...\\...\\Eflash_Proj 目录下会有一个 `makecode.ini` 文件需要先配置：`UART_BOOT_EN=1` 和 `UART_BOOT_PIN_SEL=0xD`（此处 0xD 代表 PC1，是 16 进制的数，代表是哪个引脚支持 uart port mapping。用户手册中 用户配置 目录中有 uart mapping 引脚的说明）

7:0	UART mapping	7'hFF	RO	UART Port mapping 8' d0: PA11 8' d1: PB0 8' d2: PB2 8' d3: PB4 8' d4: PB6 8' d5: PB8 8' d6: PB10 8' d7: PB12 8' d8: PB14 8' d9: PC7 8' d10: PC6 8' d11: PC3 8' d12: PC2 8' d13: PC1 8' d14: PC0 8' d15: PA0
-----	--------------	-------	----	---

31

				8' d16: PA1 8' d17: PA2 8' d18: PA3 8' d19: PA4 8' d20: PA5 8' d21: PA6 8' d22: PA7 8' d23: PA8 8' d24: PA9 8' d25: PA10 Others: Disable
--	--	--	--	--

4) 软件设计上根据实际情况可考虑三种做法:

- ① 上电开机时等待 1s 左右的时间, 再初始化 uart 单 pin 升级脚为模拟输入, 在这个开机这个时间里, 芯片可以检测到升级序列, 达到 uart 单 pin 升级的目的。
 - ② 上电到开机很快把 uart 单 pin 升级脚初始化为模拟输入, 那么需要在程序中检测时就配成模拟输入功能, 检测完成后立即配成输入功能, 配成 uart 功能的时长最好大于 1s, 以便芯片检测升级序列。
 - ③ 开机后程序检测特殊按键组合, 检测到改组合就把单 pin 升级线功能配成输入功能, 并做 5 秒左右的超时退出, 在这段时间内可以对被烧写设备进行升级。
- 5) sdk 有一个 PC1 做 uart 单 pin 升级和 adc 采样的 demo (48pin 封装芯片的方案应用板), 由于 PC1 正好可以配成 uart 输入, 所以 demo 里可以按 uart 功能配置。支持单 pin 升级的 pin 脚直接配成输入即可实现单 pin 升级。
- 6) 升级过程中不能断电。

2.6 数码管/LED 驱动注意事项

- 1) 使用硬件驱动 LED 时, 只能选择 PB 作为 COM 口, PA 作为 SEG 口。
- 2) 使用软件驱动 LED 时, 只能选择 PB 或者 PC6、PC7 作为 COM 口。

2.7 Touch-KEY 使用注意事项

- 1) 优先使用 PA 口做触摸按键, 其次用 PB, PC 不能作为触摸按键。
- 2) TK 如果不需要和 SEG 口复用 IO, 需要把该 GPIO 配置成输出模式, 并输出 0。
- 3) 如果 TK 和 SEG 口复用 IO, 而 SEG 口用软件驱动, 需要注意在驱动完 SEG 口后把 SEG 口驱动输出 0。
- 4) Touch-KEY 采用电荷转移方式时, 需要在 PB4 (48Pin 的在 PB8 引脚) 外挂陶瓷电容, 必须使用 10% 高精度的 NPO 或 X7R 材质的电容, 容值为 1nF~10nF 之间, 根据不同方案进行调节。

2.8 电源外挂电容注意事项

电源 VCC 至少需要挂 1 个 $\geq 0.1\mu\text{f}$ 的电容, 最好是挂 2 个电容 (1 个 $\geq 2.2\mu\text{f}$, 另外 1 个 $\geq 0.1\mu\text{f}$)。

2.9 芯片连接 J-Link 注意事项

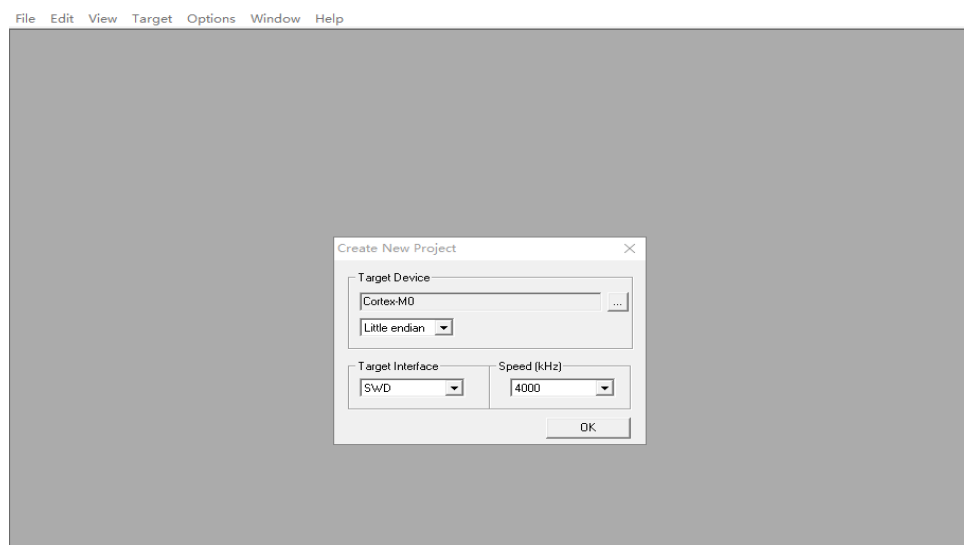
当 J-Link 连接芯片寻找 SWD 时，需要做到同电源同地，即芯片的电源要和 J-Link 的参考电源一致且要同地。

2.10 用户程序和 BOOT 的 J-LINK 烧写注意事项

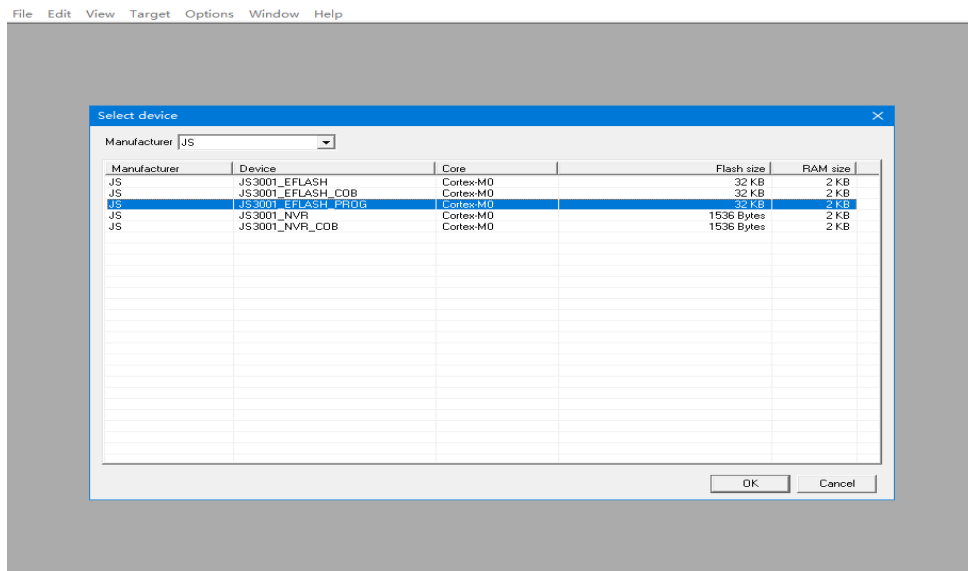
- 1) 需先烧写用户程序，再烧写 boot 程序
- 2) 若不使用单 pin 升级，使用 J-Link 烧写，需要先擦除 main，再进行烧写
- 3) 安装烧写工具链：
 - ① 用文本编辑工具打开 XXX_J-Link_eflash_support_v3\J-LINK_Support 下 setup.bat 文件。
注意：set keil_path=C:\Keil_v5\ARM\Flash 和 set J-Link_path=C:\Program Files (x86)\SEGGER\J-Link_V610i 的路径，需要和实际安装的工具软件路径保持一致。
 - ② 运行 setup.bat，进行烧写工具链的安装。
- 4) 若程序有改动，编译完成后，用 J-Link 烧写程序时需要先关闭已打开的需烧写的文件，然后再重新打开（J-Link 已打开的烧写文件若有改动则不会更新）。

2.10.1 用户程序烧写

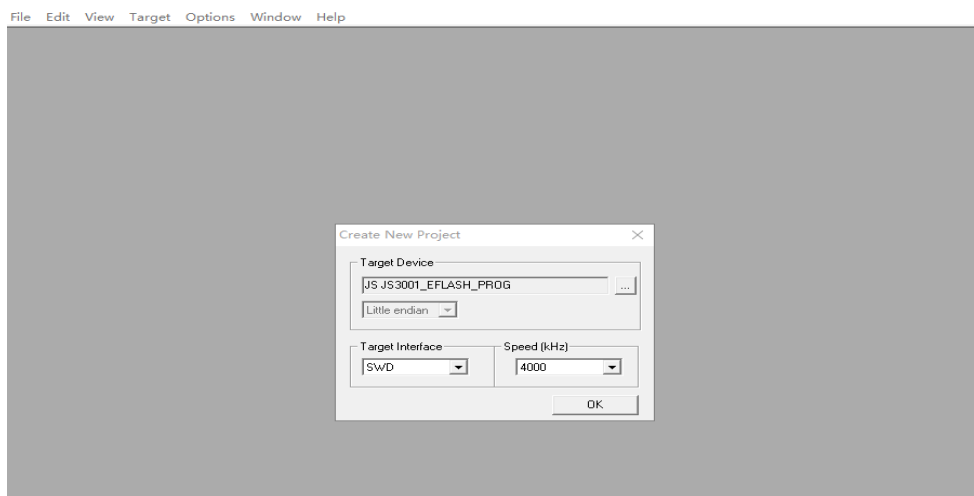
- 1) 打开...\...\Eflash_Proj\makecode.ini 编辑内容
修改：
 - ① UART_BOOT_EN=1
 - ② UART_BOOT_PIN_SEL=17（16 进制的数，代表是哪个引脚支持 uart port mapping。用户手册中——用户配置目录里有 uart mapping 引脚的说明，此处对应 PA8。切记不能有电容，因容易滤掉信号），用做单 pin 升级。
- 2) 编译程序
 - ① 打开 J-Flash，选择 file-> New project（若已配置并保存了 project，可直接选择 Open project）；



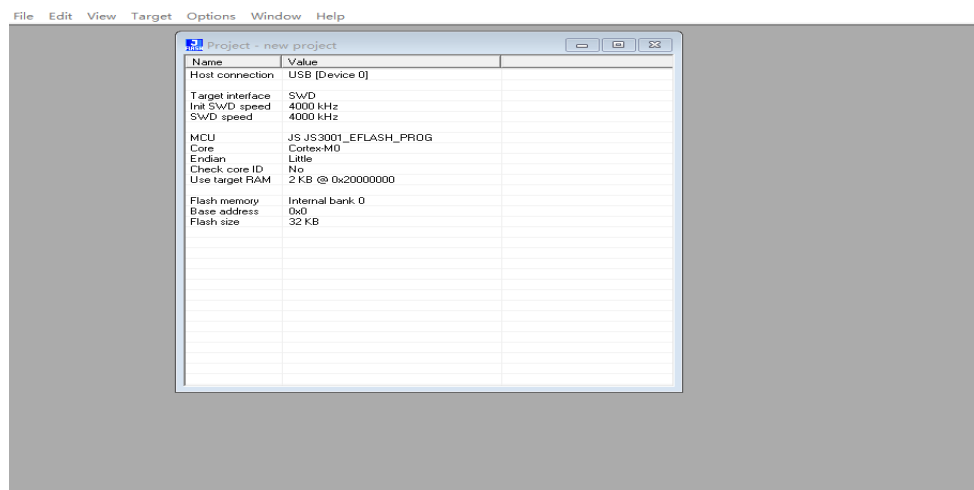
- ② 点击 Target Device 的选择按钮，输入 JS 后，选择 JS3001_EFLASH_PROG，点击 OK 按钮；



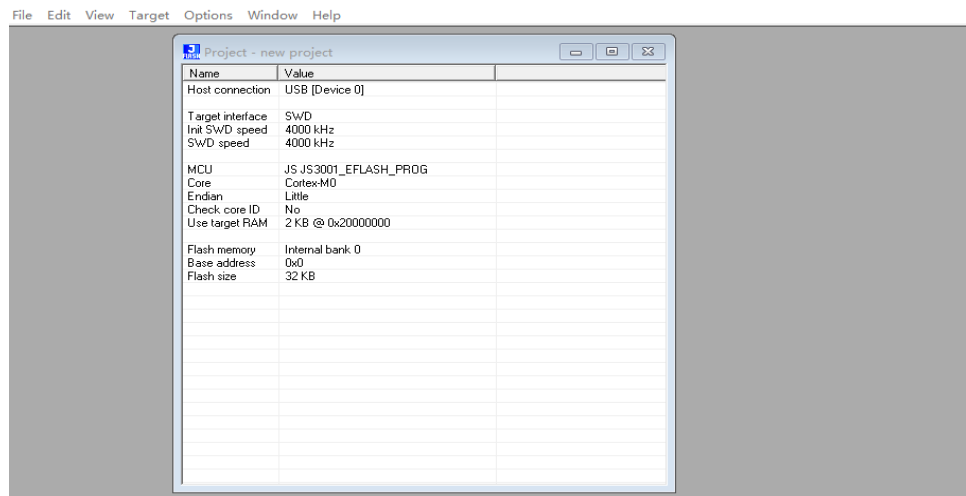
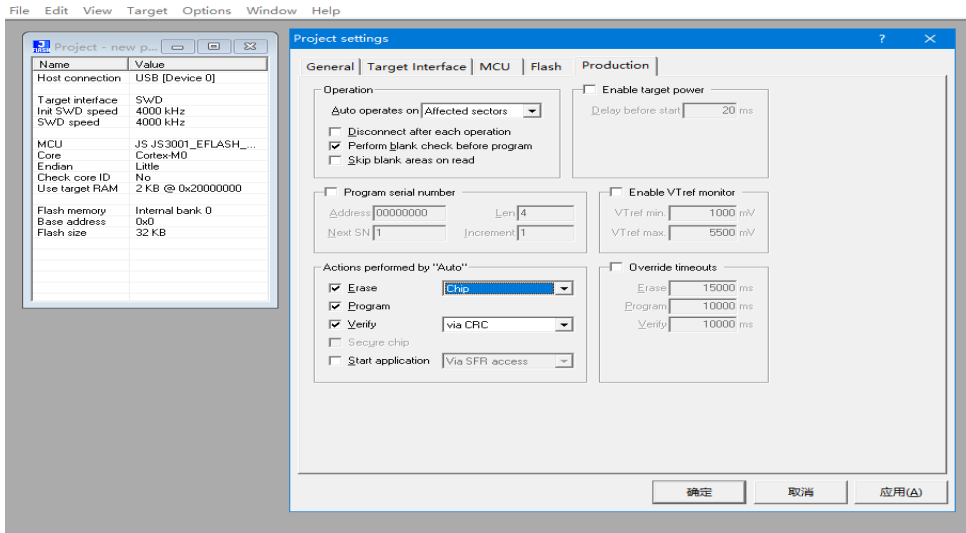
③ 点击 OK 按钮；



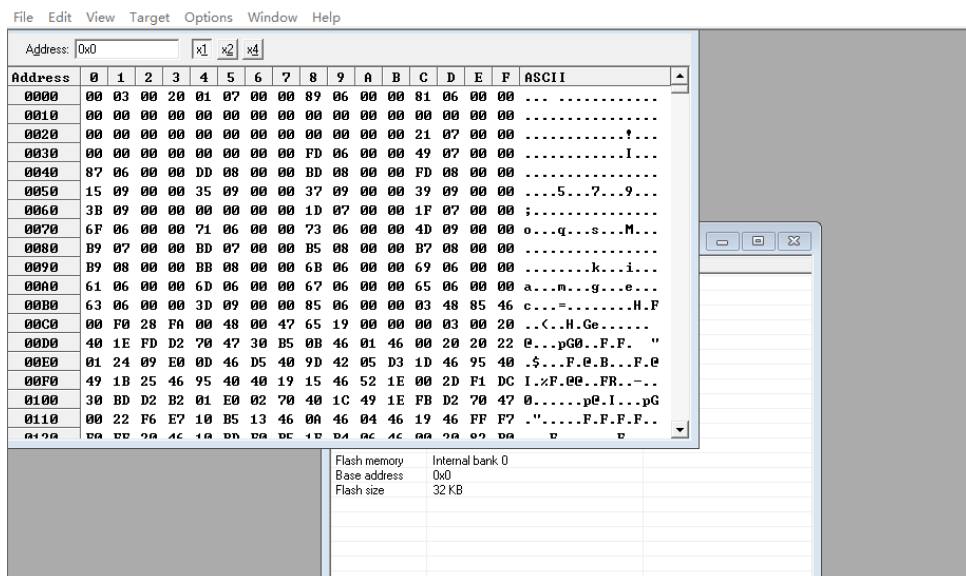
④ 出现是否需要保存改变的提示，请选择【否】；



⑤ 选择 Options->Project settings->Production 界面中把 Actions performed by"Auto"栏目中 Erase 类型选择为 Chip，点击确定按钮。



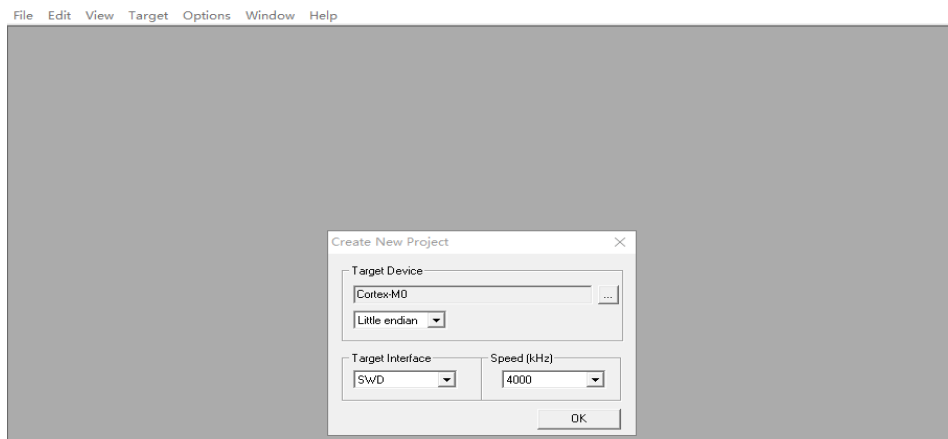
- ⑥ 选择 File->Open data file，从打开的界面中找到...\\.\Eflash_Proj\app_eflash.hex 后，点击打开按钮（烧写文件每次有更新，装载好的烧写文件不会自动更新，需要关闭再重新打开此文件）；



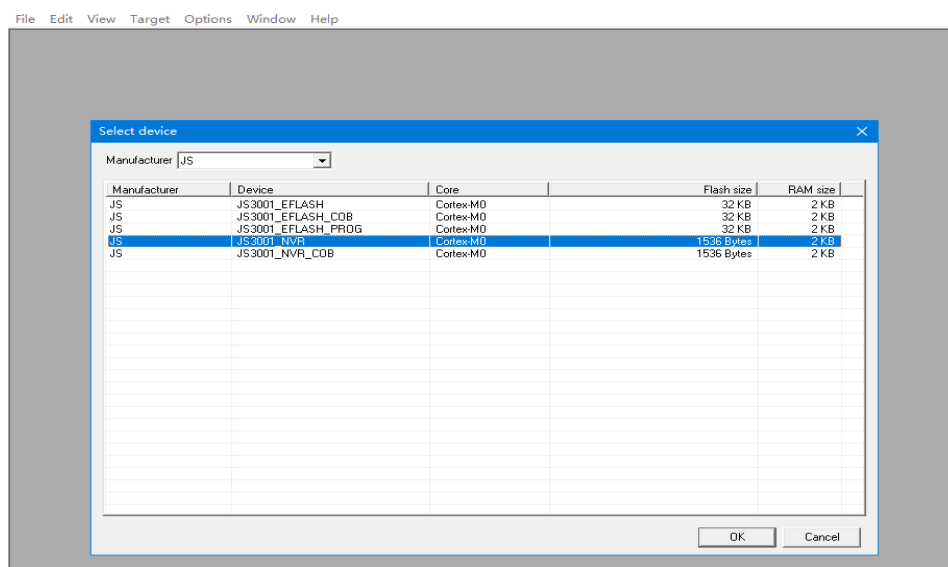
- ⑦ 选择 Target->Auto 进行烧写程序，成功后会有 success 的提示。

2.10.2 BOOT 烧写

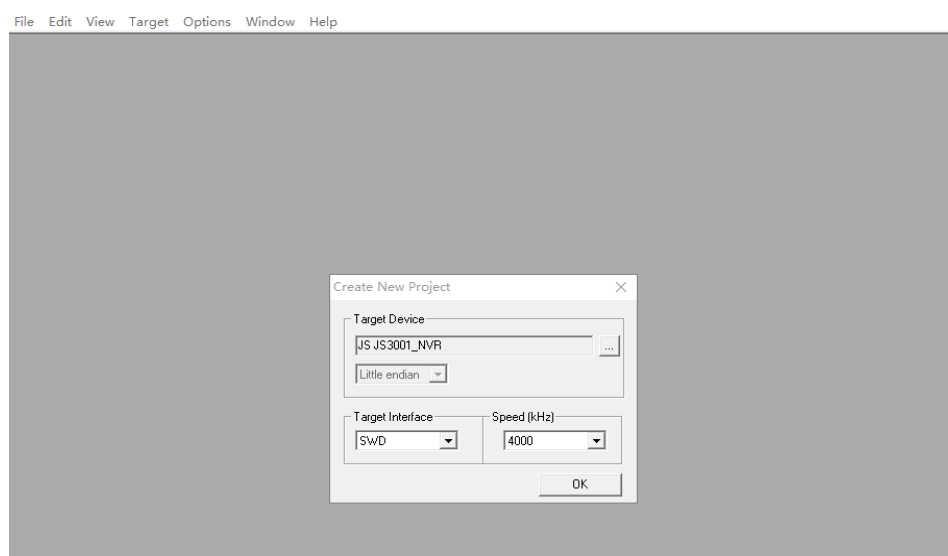
- ① 打开 J-Flash，选择 file-> New project（若已配置并保存了 project，也可直接选择 Open project）；



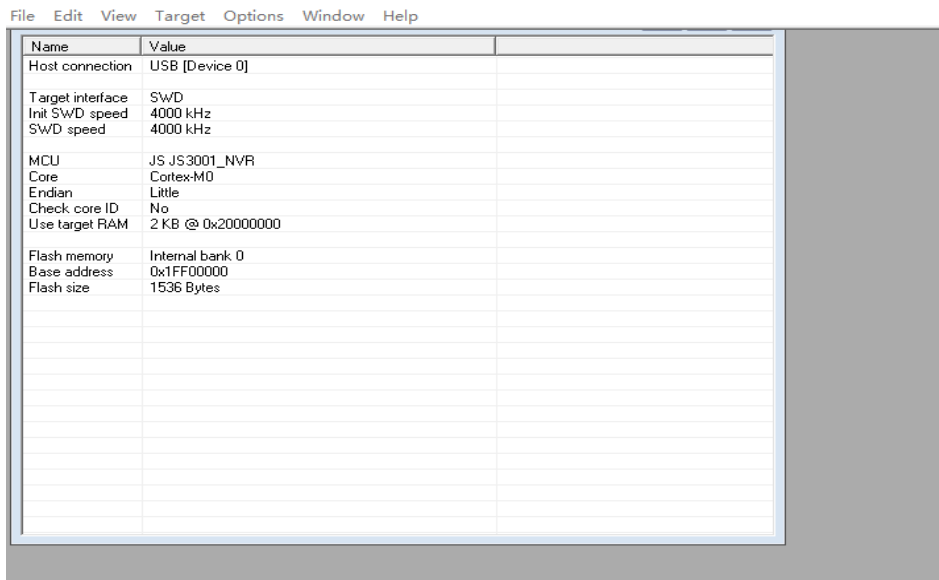
② 点击 Target Device 的选择按钮，输入 JS，然后选择 JS3001_NVR，再点击 OK 按钮；



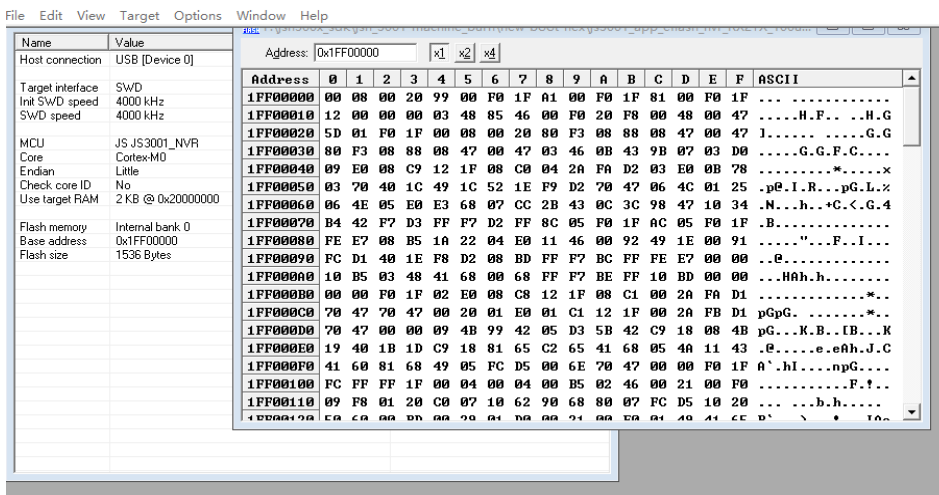
③ 点击 OK 按钮；



④ 出现是否需要保存改变的提示，可以选择【否】然后显示如下：



- ⑤ 选择 File->Open data file，从打开的界面中找到需要下载的 boot 文件（HEX 文件）后，点击打开按钮（烧写文件每次有更新，装载好的烧写文件不会自动更新，需关闭再重新打开此文件）；



- ⑥ 选择 Target->Auto 进行烧写程序，成功后会有 success 的提示。

2.11 用户 main 区域 EFLASH 保存注意事项

若用户使用了 makecode.ini 生成的 hex 文件，而且程序是有 crc 校验，那么用户使用 eflash 保存参数导致重新开机时，芯片检测到 crc 校验不对则不能开机。

故在保存参数时，需把保存参数的 eflash 地址挪到不统计 crc 校验的位置。

举例如下：

- ① 打开 Eflash_Proj\ makecode.ini 修改为 CODE_LEN=7C00。
- ② 打开 Eflash_Proj\ project.sct 修改为 ER_FLASH 0x00000000 0x00007BFC

其中 0x7BFC~0x7BFF 就是保存代码 crc 的。

显示如下：

```
LR_FLASH 0x00000000 0x00008000 {          ; 第一个加载域，名字是 LR_FLASH，起始地址
0x00000000 大小 0x00008000
```

```
ER_FLASH 0x00000000 0x00007BFC {          ; 第一个运行时域，名字是 ER_IROM1 起始地址
0x00000000 大小 0x00008000
```

```

*.o (RESET, +First)           ; IAP 第一阶段在 FLASH 中运行
*(InRoot$$Sections)          ; All library sections that must be in a root region
.ANY (+RO)                    ; .ANY 与*功能相似，用.ANY 可以把已经被指定的具

```

有 RW,ZI 属性的数据排除

```
}

```

- ③ 保存数据的 sector index: `u8 eflash_data_sec_index = LL_EF_MAIN_SECTOR_NUM - 2;` //这里 `eflash_data_sec_index = 62`,每个 sector 是 512 字节,起始地址正好是: $62 \times 512 = 0x7C00$ 。`0x7C00` 起始的 512 字节 (一个 sector) 用来保存参数,用户自行做参数的 crc 保存在此空间。
- ④ `0x7E00` 以后的 512 字节 (一个 sector) 内容用户请不要使用,因 chip 做了其它的配置信息。
- ⑤ 保存信息到 eflash 要注意: 芯片是 4 字节读写,所以用来读写的 ram 地址要 4 字节对齐。保存参数之前需要读取此参数值,若是和保存的一样就不要再保存。

请不要频繁保存参数,可以把保存参数的空间分成若干段,每一段都保存完成,然后需要保存就再次执行擦除操作,继续保存,避免频繁擦除芯片对应的 sector。

对同一个地址重复写数据需要执行擦除操作然后再写数据。

2.12 定时器精确定时的使用方式

该精准定时的功能是通过芯片内部的 trim 时钟值进行反向计算所实现,通过 `ll_timer_ms_cal` 计算反向值,主要修改定时器初始化函数即可:

下面介绍一个使用 timer0, 定时 100ms 的例子。

(1) 初始化定时器

```

138 TYPE_LL_TIMER_INIT p_init;
139 TYPE_LL_TIMER_CNT_CFG cnt_cfg;
140
141 memset(&p_init,0x00,sizeof(p_init));
142 memset(&cnt_cfg,0x00,sizeof(cnt_cfg));
143
144 p_init.prescaler = LL_TIMER_PSC_128;
145 p_init.timer_src_sel = LL_TIMER_SRC_SYS_RISING;
146 ll_timer_init(TIMER0, &p_init);
147
148 cnt_cfg.count_initial = 0;
149 cnt_cfg.count_period = ll_timer_ms_cal(TIMER0, LL_TIMER_SRC_SYS_RISING, LL_TIMER_PSC_128, 100); //100ms timer
150 printf("value = %d", cnt_cfg.count_period);
151 cnt_cfg.count_ie = ENABLE;
152 ll_timer_cnt_mode_config(TIMER_INDEX, &cnt_cfg);
153
154

```

通过以上初始化后,启动定时器 `ll_timer_start(TIMER0, LL_TIMER_MODE_SEL_COUNTER);`, 定时器将 100ms 进入一次定时器计数中断,下面介绍 `ll_timer_ms_cal` 函数的使用(调用该函数前提是已选择内部 52MHz 做系统时钟):

```

u32 ll_timer_ms_cal(TIMER_TypeDef *p_timer,TYPE_ENUM_LL_TIMER_SRC_SEL src_sel,
TYPE_ENUM_LL_TIMER_PSC psc, u16 delay_ms_data)

```

- (1) 第一个参数为定时器索引: TIMER0-4
- (2) 第二个参数,为时钟源选择,现在只支持 `LL_TIMER_SRC_SYS_RISING`
- (3) 第三个参数,为系统分频(可根据定时的时间选择对应分频)
- (4) 第四个参数,为毫秒级定时数,单位为 ms,比如 10 就是 10ms

对于 timer0-timer3 这四个定时器,毫秒定时数与分频的选择对照如下:

毫秒定时数	分频
1ms	psc=LL_TIMER_PSC_4
2-10ms	psc=LL_TIMER_PSC_16
11-30ms	psc=LL_TIMER_PSC_32
30-70ms	psc=LL_TIMER_PSC_64
71-100ms	psc=LL_TIMER_PSC_128

对于 timer4 这个定时器，毫秒定时数 与 分频 的选择对照如下：

毫秒定时数	分频
1ms-1000ms	psc=LL_TIMER_PSC_4

2.13 TK Demo 程序说明

SDK 中有张弛震荡和电荷转移这两种方式的 TK Demo 程序。

- ① ...\...\TK\TK_CC_MODE 中是电荷转移方式 demo 程序。
- ② ...\...\TK\TK_CC_MODE_INSERT_DEBUG 中是电荷转移方式调试 tk 的 demo 程序。

2.14 TK 与 LED 程序配置说明

TK demo 中张弛振荡 Demo 只是方便用户学习，实际应用须要使用电荷转移 Demo。

TK_CC_MODE（电荷转移） Demo 中通过 toucheky_led_init()进行 tk 和 led 的初始化操作，

```

384 L
385 void toucheky_led_init(void)
386 {
387
388     led_ctl_init();
389     tk_no_reuse_pins_init();
390     tk_ctl_init();
391
392 }

```

(1) led_ctl_init():进行 led 数码管 COM 与 SEG 口定义初始化及工作方式等配置。

(2) tk_no_reuse_pins_init():针对用作 TK 且不用作 LED SEG 口的 IO 做初始化，这些 IO 需要配置成 GPIO 输出模式，并且输出低电平，

(3)tk_ctl_init():针对 TK 进行配置函数，在电荷转移模式，针对 28Pin 引脚芯片需要在 PB4 外接一个 10nF 电容，而针对 48Pin 引脚芯片需要在 PB8 外接一个电容。

下面介绍如果对这些函数进行修改：

(1) led_ctl_init():

1、配置 SEG 口：

```

139 白 /* SEG: PA0~PA7
140      EX      IN
141      * seg1: PA11 SEG0
142      * seg2: PA10 SEG1
143      * seg3: PA9  SEG2
144      * seg4: PA8  SEG3
145      * seg5: PA7  SEG4
146      * seg6: PA6  SEG5
147      * seg7: PA5  SEG6
148      * seg8: PA4  SEG7
149      */
150      gpio_struct.gpio_pin = LL_GPIO_PIN_11 | LL_GPIO_PIN_10 | LL_GPIO_PIN_9 |
151                          LL_GPIO_PIN_8 | LL_GPIO_PIN_7 | LL_GPIO_PIN_6 |
152                          LL_GPIO_PIN_5 | LL_GPIO_PIN_4;
153      gpio_struct.gpio_mode = LL_GPIO_MODE_AF;
154      /* gpio_struct.gpio_type = LL_GPIO_TYPE_OUT_PP; */
155      #if USE_WHITE_LIGHT
156      gpio_struct.gpio_speed_mode = LL_GPIO_SPEED_MODE_V;
157      #endif
158      gpio_struct.gpio_speed_level = LL_GPIO_SPEED_LEVEL_7;
159      ll_gpio_init(GPIOA, &gpio_struct);
160
161      ll_gpio_pin_af_config(GPIOA, 4, LL_GPIO_AF_2);
162      ll_gpio_pin_af_config(GPIOA, 5, LL_GPIO_AF_2);
163      ll_gpio_pin_af_config(GPIOA, 6, LL_GPIO_AF_2);
164      ll_gpio_pin_af_config(GPIOA, 7, LL_GPIO_AF_2);
165      ll_gpio_pin_af_config(GPIOA, 8, LL_GPIO_AF_2);
166      ll_gpio_pin_af_config(GPIOA, 9, LL_GPIO_AF_2);
167      ll_gpio_pin_af_config(GPIOA, 10, LL_GPIO_AF_2);
168      ll_gpio_pin_af_config(GPIOA, 11, LL_GPIO_AF_2);
169

```

根据硬件连接的seg口管脚进行配置，并配置复用功能

2、配置 COM 口:

```

170 白 /* COM PB0-PB7 config
171      *EX      IN
172      *COM1: PB0 COM7
173      *COM2: PB1 COM6
174      *COM3: PB2 COM5
175      *COM4: PB3 COM4
176      *COM5: PB4 COM3
177      *COM6: PB5 COM2
178      *COM7: PB6 COM1
179      */
180      memset(&gpio_struct, 0x0, sizeof(gpio_struct));
181      gpio_struct.gpio_pin = LL_GPIO_PIN_0 | LL_GPIO_PIN_2 | LL_GPIO_PIN_4 | LL_GPIO_PIN_6 |
182                          LL_GPIO_PIN_9 | LL_GPIO_PIN_10 | LL_GPIO_PIN_12 | LL_GPIO_PIN_15;
183      gpio_struct.gpio_mode = LL_GPIO_MODE_AF;
184      gpio_struct.gpio_pupd = LL_GPIO_PUPD_UP;
185      gpio_struct.gpio_type = LL_GPIO_TYPE_OUT_OD;
186      gpio_struct.gpio_speed_level = LL_GPIO_SPEED_LEVEL_7;
187      ll_gpio_init(GPIOB, &gpio_struct);
188
189      ll_gpio_pin_af_config(GPIOB, 0, LL_GPIO_AF_2);
190      ll_gpio_pin_af_config(GPIOB, 2, LL_GPIO_AF_2);
191      ll_gpio_pin_af_config(GPIOB, 4, LL_GPIO_AF_2);
192      ll_gpio_pin_af_config(GPIOB, 6, LL_GPIO_AF_2);
193      ll_gpio_pin_af_config(GPIOB, 9, LL_GPIO_AF_2);
194      ll_gpio_pin_af_config(GPIOB, 10, LL_GPIO_AF_2);
195      ll_gpio_pin_af_config(GPIOB, 12, LL_GPIO_AF_2);
196      ll_gpio_pin_af_config(GPIOB, 15, LL_GPIO_AF_2);
197
198

```

根据硬件连接COM口的引脚进行配置，并配置相应的复用功能

3、led 功能结构体初始化:

```

199      led_struct.led_drv_en = DISABLE;
200      led_struct.com_divt_en = DISABLE;
201      led_struct.led_drv = 0x00;
202      led_struct.sweep_hold_time = 0xB7FE; /* every COM or SEG light up time (us) */
203      led_struct.scan_polling_time = 0xB800 << 4; /* Set the maximum refresh rate(ms) */
204      led_struct.auto_scan_en = ENABLE; /* auto scan */
205      led_struct.scan_way = LL_LED_SCAN_WAY_COM; /* scan way */
206      led_struct.led_polarity = LL_LED_PLY_COMM_CATHODE; /* comm cathode led display */
207      led_struct.led_en = ENABLE; /* LED enable */
208      led_struct.support_num = 8; /* support num, COM support most 8 */
209      led_struct.led_com_map = 0xFF; /* enable COM map */
210      led_struct.led_seg_map = 0xFF; /* enable SEG map */
211      led_struct.led_addr = (uint32_t)&led_base_addr[0];
212      ll_led_init(LED, &led_struct);
213

```

Sweep_hold_time:为每个 COM 扫描的时间，可根据实际情况修改。

Scan_polling_time:为第一轮第一个 COM 扫描至下一轮第一个 COM 扫描的时间，可根据实际情况修改。

Scan_way:为扫描方式选择，建议选择 COM 扫描。

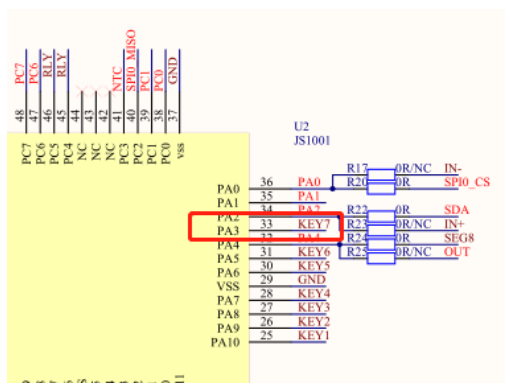
Led_polarity:为扫描极性选择，根据硬件是共阴还是共阳对应选择即可。

Suppot_num:为 COM 的个数。

Led_com_map:为 led 中 com 的 map 图，如上图配置为 0xFF，是因为芯片的 COM0-COM7 连接了数码管的 com 口。

Led_seg_map:为 seg 的 map 图。如上图配置为 0xFF，是因为芯片的 seg0-seg7 连接数码管的 seg 口。

(2) tk_no_reuse_pins_init()



根据上图方案板中只有 KEY7 没有复用 SEG 口，且 KEY7 对应的 PA3，所以只需要将 PA 配置为输出模式并输出低电平即可。如下述：

```

216 ②/**
217  * @brief TK relaxation oscillator led display init config
218  * @param None
219  * @retval None
220  */
221 void tk_no_reuse_pins_init(void)
222 ②{
223     /* ===== */
224     /* single TK IO output low voltage */
225     /* ===== */
226     /* tochekey PA3 --Key7 mode output*/
227     TYPE_LL_GPIO_INIT gpio_struct;
228     memset(&gpio_struct, 0x0, sizeof(gpio_struct));
229     gpio_struct.gpio_pin = LL_GPIO_PIN_3;
230     gpio_struct.gpio_mode = LL_GPIO_MODE_OUT;
231     gpio_struct.gpio_speed_mode = LL_GPIO_SPEED_MODE_V;
232     gpio_struct.gpio_speed_level = LL_GPIO_SPEED_LEVEL_7;
233     ll_gpio_init(GPIOA, &gpio_struct);
234     ll_gpio_reset_bits(GPIOA, LL_GPIO_PIN_3);
235 }
236

```

注意：如果所有的 TK 按键都被复用，可以不调用此函数，或者函数体不放置代码。

(3) tk_ctl_init():

该函数只需针对芯片类型配置相应的外接基准电容的引脚进行配置即可

```

/*cap charge mode tochekey PB8 connect Benchmark capacitance mode:analog*/
memset(&gpio_struct, 0x0, sizeof(gpio_struct));
gpio_struct.gpio_pin = LL_GPIO_PIN_8;
gpio_struct.gpio_mode = LL_GPIO_MODE_AN;
gpio_struct.gpio_speed_mode = LL_GPIO_SPEED_MODE_V;
gpio_struct.gpio_speed_level = LL_GPIO_SPEED_LEVEL_7;
ll_gpio_init(GPIOB, &gpio_struct);
ll_gpio_reset_bits(GPIOB, LL_GPIO_PIN_8);

```

以上测试由于使用的是 48Pin 引脚即需要在 PB8 外接 10nF 电容，而对于 28Pin 引脚需要外接在 PB4 外接 10nF 电容。

(4) 修改 jsh300x_tk_cfg.h 文件中阈值：

1、配置 TK 使能的 map 图及个数：

```

29 #define TK_BIT_ENABLE 0x0017f
30 #define TK_ENABLE_NUM 8
31

```

TK_BIT_ENABLE 为 tk 使能的按键，上图 0x0017f 为对应 bit0、bit1、bit2、bit3、bit4、bit5、bit6、bit8 为置 1，使能的按键即为 TK0、TK1、TK2、TK3、TK4、TK5、TK6、TK8 共 8 个按键，TK_ENABLE_NUM 为 8

2、调节软硬件阈值

```

55 白/*
56  * TK Offset Setting
57  */
58
59 #define TK_PA11_OFFSET_HW 18 //TK0
60 #define TK_PA10_OFFSET_HW 18 //TK1
61 #define TK_PA9_OFFSET_HW 15 //TK2
62 #define TK_PA8_OFFSET_HW 14 //TK3
63 #define TK_PA7_OFFSET_HW 17 //TK4
64 #define TK_PA6_OFFSET_HW 15 //TK5
65 #define TK_PA5_OFFSET_HW 16 //TK6
66 #define TK_PA4_OFFSET_HW 10 //TK7
67 #define TK_PA3_OFFSET_HW 30 //TK8
68 #define TK_PA2_OFFSET_HW 10 //TK9
69 #define TK_PA1_OFFSET_HW 10 //TK10
70 #define TK_PA0_OFFSET_HW 10 //TK11
71 #define TK_PB7_OFFSET_HW 10 //TK12
72 #define TK_PB6_OFFSET_HW 10 //TK13
73 #define TK_PB5_OFFSET_HW 10 //TK14
74 #define TK_PB4_OFFSET_HW 10 //TK15
75 #define TK_PB3_OFFSET_HW 10 //TK16
76 #define TK_PB2_OFFSET_HW 10 //TK17
77 #define TK_PB1_OFFSET_HW 10 //TK18
78 #define TK_PB0_OFFSET_HW 10 //TK19
79
80
81 #define TK_PA11_OFFSET_SW 28 //TK0
82 #define TK_PA10_OFFSET_SW 27 //TK1
83 #define TK_PA9_OFFSET_SW 20 //TK2
84 #define TK_PA8_OFFSET_SW 18 //TK3
85 #define TK_PA7_OFFSET_SW 25 //TK4
86 #define TK_PA6_OFFSET_SW 18 //TK5
87 #define TK_PA5_OFFSET_SW 24 //TK6
88 #define TK_PA4_OFFSET_SW 10 //TK7
89 #define TK_PA3_OFFSET_SW 40 //TK8
90 #define TK_PA2_OFFSET_SW 10 //TK9
91 #define TK_PA1_OFFSET_SW 10 //TK10
92 #define TK_PA0_OFFSET_SW 10 //TK11
93 #define TK_PB7_OFFSET_SW 10 //TK12
94 #define TK_PB6_OFFSET_SW 10 //TK13
95 #define TK_PB5_OFFSET_SW 10 //TK14
96 #define TK_PB4_OFFSET_SW 10 //TK15
97 #define TK_PB3_OFFSET_SW 10 //TK16
98 #define TK_PB2_OFFSET_SW 10 //TK17
99 #define TK_PB1_OFFSET_SW 10 //TK18
100 #define TK_PB0_OFFSET_SW 10 //TK19
101

```

硬件阈值

软件阈值

软件阈值必须大于硬件阈值，硬件阈值为 TK 触发的阈值，而通过软件方式是否符合软件阈值进而判断是否触发。

2.15 芯片配置定义

在 SDK 中有用哪种封装芯片的宏定义：USE_CHIP_16_PIN， USE_CHIP_20_PIN， USE_CHIP_24_PIN， USE_CHIP_48_PIN，只能使能其中一个，主要是控制 io remap 以及宏 __CHIP_PB_NO_REMAP（adk 驱动 h 文件用到），用户根据自己的实际芯片配置。

2.16 不使用 Boot 的方法

将 Sdk 开发包里... \ Project\KEIL-ARM \目录下会有一个 makecode.ini 文件需要先配置：
UART_BOOT_EN=0，并且还需要设置：**UART_BOOT_PIN_SEL=FF**，这样才能在关闭掉 UART_BOOT 功能同时也让 PIN 脚不失效，根据这两个步骤设置后，用户可以不烧录 boot 文件。

2.17 关于 ADC 的精度问题

对于量产的芯片，如果 jsh300x_system.c 中的 USE_ADC_NON_MASS_PRODUCT 宏定义配置为 1 时，会关闭芯片自动校准功能，只能保证 8bit 的精度；如果将此宏定义配置为 0 时，这样程序不会关闭芯片自动校准功能，精度会比将该宏定义配置为 1 时高。

对于非量产芯片，jsh300x_system.c 中 USE_ADC_NON_MASS_PRODUCT 宏定义需配置为 1。

2.18 关于修改 app_elflash.hex 和 app_elflash.bin 的文件名

为了方便客户修改生成的目标文件名，将在新的 SDK（V1.01-20190712）及之后版本中增加此功能，用户可通过修改···\Project\KEIL-ARM\ 目录下的 make.bat 文件中的 set Target_Name=XXX, XXX 为新的文件名，以达到生成最终烧录的文件名。

2.19 关于 PC6 被用作普通 IO 及其他功能时的注意事项

当需要将 PC6 作为普通 IO 或其他功能脚时，若在上电时对 PC6 没控制好，会导致芯片误入超级模式，从而导致芯片无法正常运行。

先来分析一下，两种情况会进入超级模式：

（1）当配置 makecode.ini 中的信息如下配置时：

SYS_SWD_EN=0 ----->意思是：关闭 SWD

SYS_SWD_IO_PU_EN=1 ----->意思是：PC6 芯片内部上拉

当外部给予该引脚（PC6）的电压为低电平的时候，且此时 PA3 和 PC7 为高时，就会进入超级模式。

（2）当配置 makecode.ini 中的信息如下配置时：

SYS_SWD_EN=0 ----->意思是：关闭 SWD

SYS_SWD_IO_PU_EN=0 ----->意思是：PC6 芯片内部下拉

当外部给予该引脚（PC6）的电压为高电平的时候，且此时 PA3 和 PC7 为高时，就会进入超级模式。

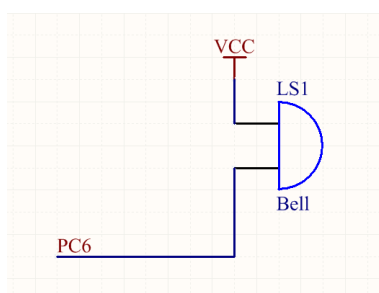
由上述所知，特别是当 PA3 和 PC7 为悬空时，高阻态即为电平不确定时，客户就会出现有时候芯片时而正常运行，时而不正常运行的情况，为防止进入超级模式的情况，需注意如下：

- a) 当外面电路给予电压为高时，配置 makecode.ini 中 SYS_SWD_IO_PU_EN=1;
- b) 当外面电路给予电压为低时，配置 makecode.ini 中 SYS_SWD_IO_PU_EN=0;
- c) 当外面不给予电压时，makecode.ini 中 SYS_SWD_IO_PU_EN=0 或 1 都可以，但前提不要影响外部电路工作。

下面通过几个常用例子进行分析：

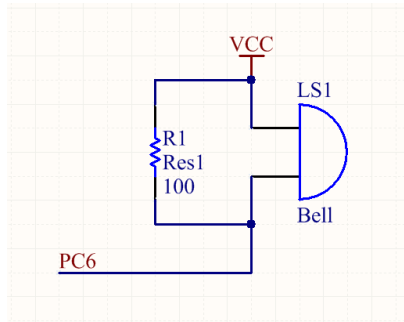
（1） PC6 引脚必须确保外面电路的电平是固定的，尽可能不要接 ADC 电路，因为 ADC 有可能处于无法固定的电平，所以 PC6 最好不要接外部 ADC 电路，若一定要接 ADC 必须保证芯片上电一瞬间，PA3 和 PC7 至少有一个需外部给予低电平。

（2） PC6 直推压电式无源蜂鸣器的情况：



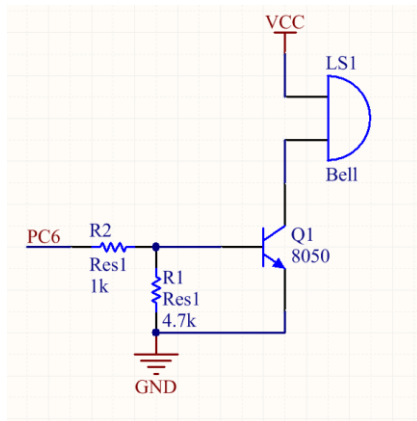
分析：如果 SYS_SWD_IO_PU_EN 配置成下拉，蜂鸣器上电会处于导通状态，所以应该把 SYS_SWD_IO_PU_EN 配成上拉。

(3) PC6 另外一种推压电式无源蜂鸣器的情况：



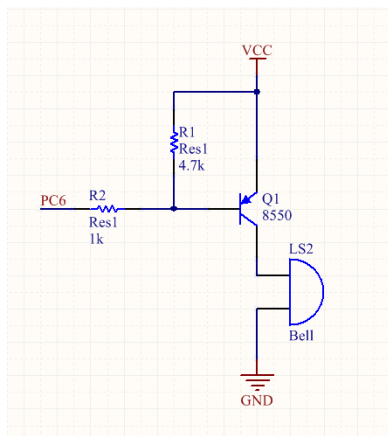
分析：PC6 外面经过上拉电阻 R1，即系统上电，外面给予 PC6 电平为高，如果 makecode.ini 中 SYS_SWD_IO_PU_EN 配置为下拉的情况下，且 PA3、PC7 都为高电平，就会进入超级模式，而且当配置为下拉，蜂鸣器也会响，所以为了避免这种情况，必须将 makecode.ini 中 SYS_SWD_IO_PU_EN 配置为上拉。

(4) PC6 三极管 (NPN) 推蜂鸣器的情况：



分析：PC6 外面经过 R2 和 R1 连接到地，所以为低电平，因此 makecode.ini 中 SYS_SWD_IO_PU_EN 配置为上拉的情况下，且 PA3、PC7 都为高电平，就会进入超级模式，所以为了避免这种情况，必须将 makecode.ini 中 SYS_SWD_IO_PU_EN 配置为下拉。

(5) PC6 三极管 (PNP) 推蜂鸣器的情况：



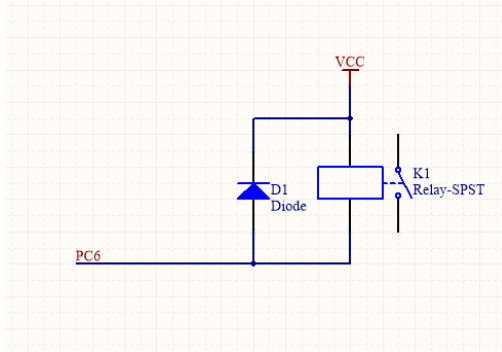
分析：PC6 外面经过 R2 和 R1 连接到 VCC，所以为高电平，因此 makecode.ini 中

SYS_SWD_IO_PU_EN 配置为下拉的情况下，且 PA3、PC7 都为高电平，就会进入超级模式，所以为了避免这种情况，必须将 makecode.ini 中 SYS_SWD_IO_PU_EN 配置为上拉。

(6) PC6 用作串口发送引脚的情况：

分析：作为串口输出脚的时候，最好将 makecode.ini 中 SYS_SWD_IO_PU_EN 配置为上拉。

(7) PC6 直推继电器的情况：



分析：作为直推继电器的情况，最好将 makecode.ini 中 SYS_SWD_IO_PU_EN 配置为上拉。

注意：PC6、PC7 直推继电器的时候请不要加限流电阻。