

KF32 系列

链接工具校验和使用说明 V1.0

2021 年 08 月

目录

目录	2
修改记录	3
1. 工具的调用	4
2. 基本功能参数:	4
2.1 添加库路径-L	4
2.2 添加库-L	4
2.3 指定脚本	5
2.4 死段优化选项	5
2.5 指定输出	5
2.6 辅助选项	5
3. CHECKSUM 功能参数:	5
3.1 配置方法	6
3.2 内容模型--WITH-CHECKSUM-FILL=MODENUMBER	6
3.3 算法指定--WITH-CHECKSUMX=WHAT	8
3.3.1 典型 CRC-32	8
3.3.2 典型 CRC-32/MPEG-2	8
3.3.3 典型 SUM32	9
3.3.4 典型 SIG-CODE	9
3.4 起始地址--WITH-CHECKSUM-ADDRESSX=WHERE1	9
3.5 预计算长度--WITH-CHECKSUM-SIZEX=BYTES	9
3.6 目标地址--WITH-CHECKSUM-OUT-ADDRESSX=WHERE2	9

修改记录

序号	日期	版本	变更及说明	其他
1	2022-05-07	V1.0	初稿	
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				

1. 工具的调用

格式: `{COMMAND} {INPUTS} {FLAGS} -o "${ProjName}.elf" -Map "${ProjName}.map"`

举例: `kf32-ld ./_config/startup.o ./_config/vector.o ./kf_it.o ./main.o
-L"C:\Program Files (x86)\ChipON IDE\KungFu32\ChipONCC32\ccr1_issue\lib"
-L"d:\workspace32\demo" -lIQmath-R1 -lmath -lio -lstring -lstdlib -lctype -lcrtv1
-T"C:\Program Files (x86)\ChipON IDE\KungFu32\ChipONCC32\ccr1_issue\scripting\KF32F350MQV.ld"
--kf32-autoihex --kf32-arch=kf32r --kf32-z --gc-sections -o "demo.elf" -Map "demo.map"`

注: 链接器的输出文件为 elf 格式的目标机文件, --kf32-autoihex 选项实现自动调用 objcopy 的输出可烧录 hex 文件。库文件应在编译汇编输出 o 文件后面添加, 若库使用其他库方法, 该库应优先添加。

若链接使用 kf2-gcc, 默认会传递库, 这些库在路径\lib\gcc\kf32\4.7.0 下的 libgcc.a ccr0.o crt1.o crtn.o crtbegin.o crtend.o, 若不需要添加选项 -nostdlib。使用 kf32-gcc 下链接器专有选项请使用 -Wl,option 或 -Xlinker option, 另外库路径不使用 C:\Program Files (x86)\ChipON IDE\KungFu32\ChipONCC32\ccr1_issue\lib, 因目录存在 gcc 文件夹, 缩写 libgcc.a 的 gcc 冲突。可以使用\kf32\lib 作为库的所在路径, 此时 -lmath -lio -lstring -lstdlib -lctype 对于库可复制到这里, 也可以使用 libc.a libm.a 的通用合并后的库。

当程序的起始添加@时, 调用时将不使用 echo 进行输出到控制台, 否则将输出。

2. 基本功能参数:

2.1 添加库路径-L

示例默认系统库路径

`-L"C:/Program Files (x86)/ChipON IDE/KungFu32/ChiponCC32/lib/ccr1_issue"`

示例当前项目下主目录下库路径

`-L"D:\workspace32\KungFu32_ProDebug_WinUSB"`

2.2 添加库-l

默认库如下

<code>-lIQmath-R1</code>	定点浮点库
<code>-lSeriesDIServices</code>	串口交换协议接口
<code>-lmath -lio -lstring -lstdlib -lctype</code>	系统相关的实现库
<code>-lcrtv1</code>	型号相关的必须库

若库的命名规则为 libXXX.a，可以使用简化的-lXXX. 否则需要完整的传输，如 XXXX.a 使用 -lXXX.a.

库的传递不应附加路径信息，路径需要通过-L 选项传递。

2.3 指定脚本

-T“../KF32F341IQS_9KWinUSB.ld” 项目主目录的自定义脚本

或

-T“C:/Program Files (x86)/ChipON

IDE/KungFu32/ChiponCC32/scripting/ccr1_issue/KF32F130GQT.ld 工具默认项目型号脚本

2.4 死段优化选项

--gc-sections 不保留与输出为使用的段落，包括变量与函数，除非脚本中使用 KEEP 修饰。

2.5 指定输出

-o XXX.elf 机器可执行程序格式文件

-Map XXX.map 编译段与符号信息

2.6 辅助选项

--kf32-nodisassemble 不执行链接后反汇编动作

--kf32-z 反汇编连续的 0 也仍然输出

--kf32-autoihex 集成调用 kf32-objcopy 输出 ihex 格式文件

--kf32-nodisassemble 取消默认集成调用 kf32-dump 输出反汇编 lst 文件

3. CheckSum 功能参数:

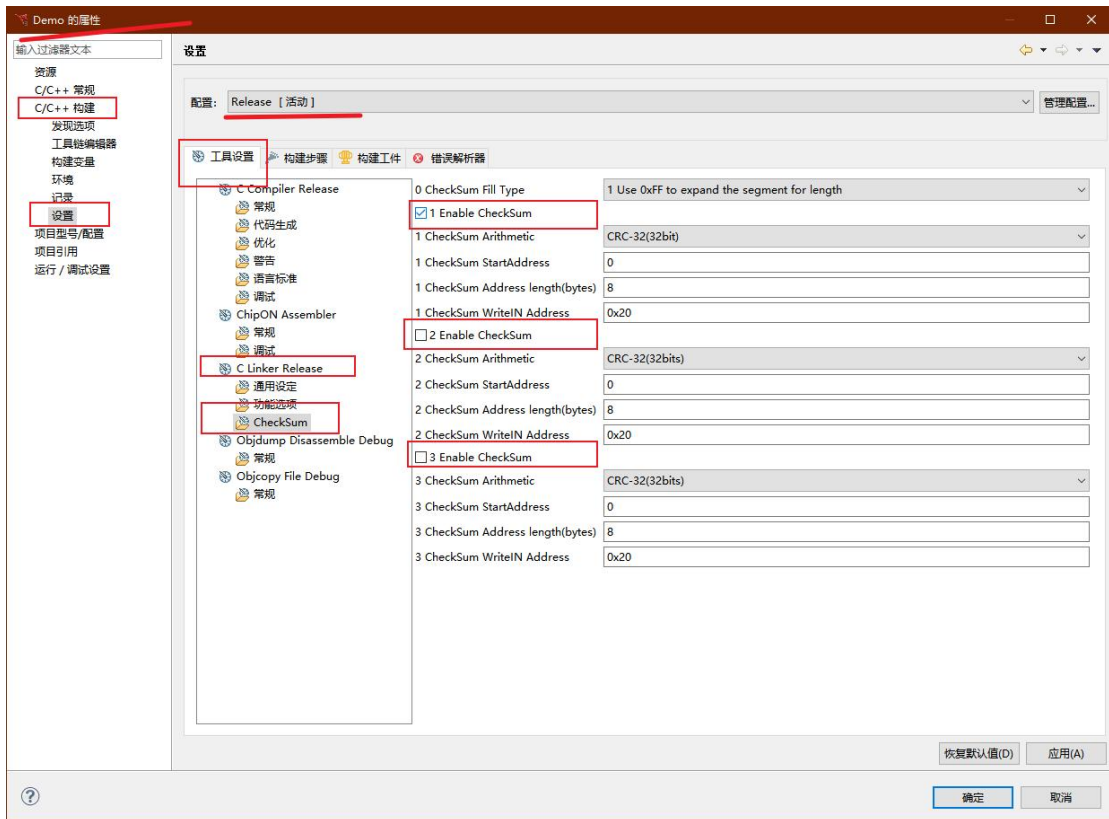
该功能为安全启动的辅助功能，即预计算段内容并保存特征结果到特定地址。程序运行启动过程时调用芯片硬件资源或软件资源重新对段内容进行计算，并与目标预期内容进行比较，若一致认为内容正确，若不一致认为内容异常(失效)，从而程序分支给与异常信息的后续处理。

该计算的内容采用小格式存储表达，如地址对齐内容 0x12345678, 其输入为: 78 56 34 12。

目标地址段必须设计脚本使其存在原始内容。

3.1 配置方法

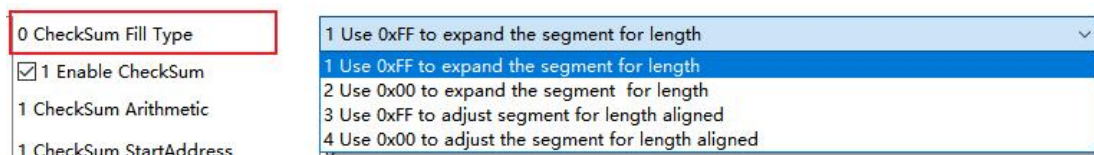
如图所示，需要在项目属性下的工具链配置页面，配置链接器的功能参数：



注 1：使用时应优先勾选使能，最大支持 3 组并顺序计算与输出。

注 2：仅配置 1 的结果地址支持输出到 0 地址，其他情况结果不支持输出到 0 地址。地址与长度的参数支持 16 进制的 0x 起始或 H 结尾，也支持 10 进制的直接输入。

3.2 内容模型--with-checksum-fill=ModeNumber



如图所示支持 4 中模型选 1，该模型针对 3 组统一生效。对应命令行参数分别为：

```
--with-checksum-fill=0xFF (默认)          --with-checksum-fill=0x00
--with-checksum-fill=0x55ff                --with-checksum-fill=0x5500
```

针对程序开发过程，如 bootload 的开发模型，若计算 bootload 的连续代码时，虽然脚本上划分了可用的 flash 空间，如 16K，实际情况代码往往占用空间小于该值，并且全局变量初始化或 ram 函数的内容会跟随附加在代码的后面。从而容量是与代码绑定的。

该参数值的低 8bits 为计算填充值（计算扩展无干预区域芯片实际值，不是有效片段脚本的填充间隙值）。因此根据芯片特性擦除后默认 bit 为 1，推荐模式 1 或 3。

当有效片段大小随代码量变化，模式 1 为绝对长度并无使用判断根据芯片擦除特性为 1 的补充最小字节内容 0xFF。如 bootload 设计区域 16K，计算 16K-4，并结果 4bytes 存放在最后的 4 个字节空间。该模型支持跨段落计算，如 16K 应用为 前 x bytes 指定段空间（间隙）中间 y bytes 指定段空间（间隙）尾 4 bytes 结果空间。该间隙为工具加载前的默认值，一般与芯片擦除结果时空一致。

若仅考虑有效代码的计算，推荐模式 3，该模型建立在每一个段的起始地址都是固定情况，如 16K 为整体待分配空间，或应用为 前 x bytes 指定段空间（间隙）中间 x bytes 指定段空间（间隙）尾 4 bytes 结果空间。传递时的预计算长度应大于可分配空间（至少大于有效使用空间，否则没有实际验证意义），如输入最大长度 16K (0x4000)。该模式将长度适配有效段空间，考虑结果为 32bit 或 128bit，因此使用尾字节扩充内容实现尾地址对齐。该模型一个计算只能计算一个片段，如整体 16K 假设，或分段假设的某一个片段。如一个段起始为 0x338，配置长度 0x400，实际代码量 0x331 字节，因此有效片段尾不包含地址尾 0x669，假定为 crc32 的算法，0x55FF 参数，补充内容 0x669 0x66A 0x66B 并内容为 0xFF 的进行校验和计算。

特别说明：针对通用单区域的程序时，起始固定，脚本变量 `__text_end__` 对应这程序片段的结尾地址，`__data_start__` 和 `__data_end__` 的差值对应着全局初始化和 ram 函数的大小，从而有效判断尾地址尾 `__text_end__ + __data_end__ - __data_start__`。否则模型 3、4 对应的待计算区域应脚本建立自定义命令变量供代码识别。如

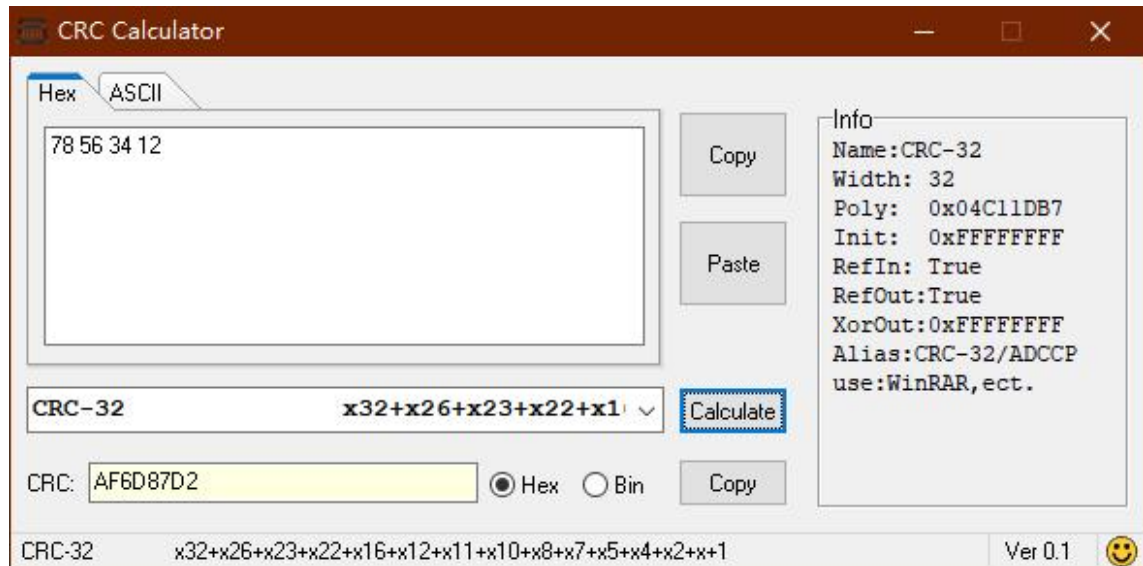
```
.exxflash :
{
    KEEP (*(.exxzoom*))
    . = ALIGN(4);
    __exxflash_End__ = .;
} > EW_mem = 0xFFFFFFFF //start form EW_mem,length=__exxflash_End__ - start
```

编写计算代码参数起始与长度时可以直接使用脚本变量，也可以先将脚本变量的值存放在一个数组中，脚本变量在代码中使用应通过 `&` 前缀的对应其地址值。如

```
extern unsigned int __text_end__;
extern unsigned int __data_start__;
extern unsigned int __data_end__;
unsigned int Arr[3]={
    (unsigned int)&__text_end__,
    (unsigned int)&__data_end__,
    (unsigned int)&__data_start__
};
void __attribute__((noinline)) Function(unsigned int start,unsigned int length){
    asm("NOP");
}
int main()
{
    Function(0x0,(unsigned int)&__text_end__ + (unsigned int)&__data_end__ -
    (unsigned int)&__data_start__);
    Function(0x0,Arr[0] + Arr[1] - Arr[2]);
    while(1) {
    }
}
```

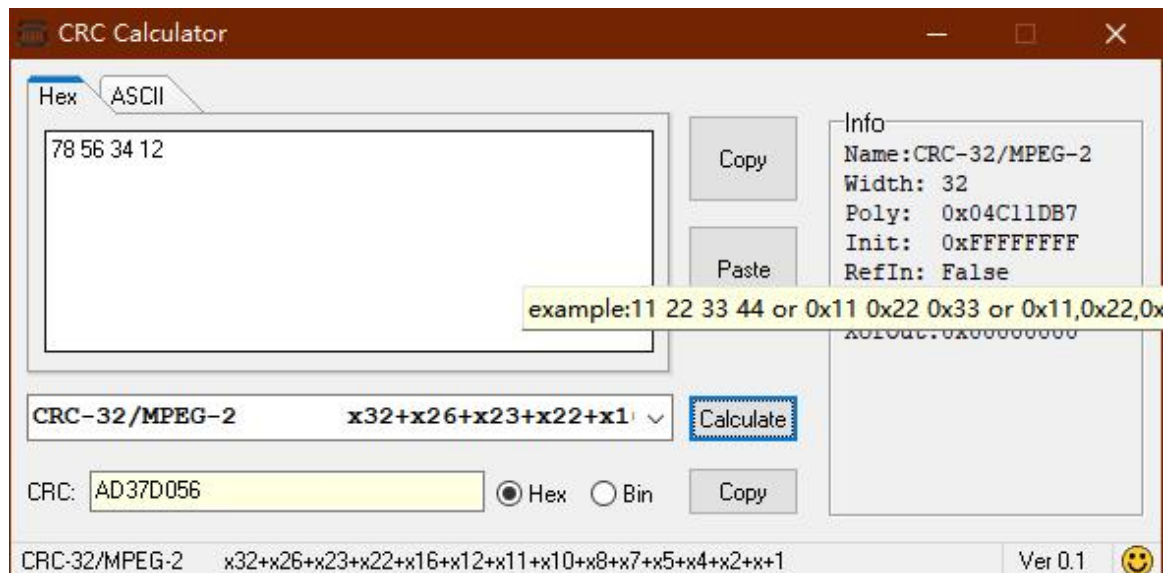
3.3 算法指定 `--with-checksumX=What`

3.3.1 典型 CRC-32



假定起始 0 地址内容 0x12345678 的小格式存储输入举例如上。

3.3.2 典型 CRC-32/MPEG-2



假定起始 0 地址内容 0x12345678 的小格式存储输入举例如上。

3.3.3 典型 SUM32

该对应输入地址 4 对齐，长度 4 对齐的整数内容的加和，如 78 56 34 12 78 56 34 12 对应 0x12345678+-x12345678;

3.3.4 典型 SIG-CODE

该算法为 16 对齐的芯片硬件算法，可以添加库函数声明头文件“ChipMessageApi.h”并调用：

```
//Return:CMD_SUCCESS                                0x00
//Return:PARAM_ERROR                                0x0C
//Parameter:startaddress aligned by 16bytes like 0 16 32 48 64
//Parameter:length aligned by 16bytes like 16 32 48 64
//Parameter:Checksum Resultin.size of result must more than 4*4 bytes and address aligned by 4bytes. like
0x1XXX XXX0 or 0x1XXX XXX4 0x1XXX XXX8 0x1XXX XXXC by defined with type of int
__attribute__((section(".indata"))) unsigned int
FLASH_CheckSum_With_128bits__(unsigned int startaddress,unsigned int
length,unsigned int result[]);
```

3.4 起始地址--with-checksum-addressX=Where1

固定的起始地址，模式 3、4 时应为真实的起始地址，模式为 1、2 时支持小于起始的被填充，如配置代码段起始地址为 4，但该参数为 0 的模式 1 对应补充 0xFFFFFFFF 的区域进行计算。

起始值支持综合的数据格式，即 10 进制、16 进制、8 进制、2 进制。

3.5 预计算长度--with-checksum-sizeX=Bytes

模式 1、2 的固定长度参与计算，该值应大于有效代码大小的包含。

模式 3、4 的乐观最大长度的参与传递。

长度值支持综合的数据格式，即 10 进制、16 进制、8 进制、2 进制。

3.6 目标地址--with-checksum-out-addressX=Where2

第一组参数可用于实验目标的输出参数到 0 地址 (实际不应该发生, 0 地址往往对应向量表的默认 msp 初始值)。

起始值支持综合的数据格式，即 10 进制、16 进制、8 进制、2 进制。

根据不同的算法结果的长度对应着固定的 4bytes 或 16bytes.

要求该地址必须在程序段分配下存在预留结果空间。即脚本段明确设计存在。结果的地址一般不应该在被计算的区间，即原始值参与并替换的不可验证。

当后续组的计算包含该目标地址时，因前面组计算并替换完成，结果校验值将参与后续组的校验值计算。