

KF32 系列

编译工具使用说明 V1.0

2024 年 07 月

目录

目录	2
修改记录	3
1. 工具的介绍	4
1.1 发布形式与系统说明	4
1.2 文件结构	4
1.3 工具使用简介	5
1.4 主要工具与举例说明	5
1.4.1 <i>kf32-g++</i>	5
1.4.2 <i>kf32-gcc</i>	6
1.4.3 <i>kf32-as</i>	6
1.4.4 <i>kf32-ld</i>	6
1.4.5 <i>kf32-objdump</i>	7
1.4.6 <i>kf32-objcopy</i>	7
1.4.7 <i>kf32-readelf</i>	7
1.4.8 <i>kf32-size</i>	7
1.4.9 <i>kf32-ar</i>	8
1.4.10 <i>kf32-gdb</i>	8
2. 基于 GMAKE 环境的应用举例	9
2.1 RELEASE/MAKEFILE	10
2.2 RELEASE/OBJECTS.MK	11
2.3 RELEASE/SOURCES.MK	12
2.4 RELEASE/SUBDIR.MK	12
2.5 RELEASE/CONFIG/SUBDIR.MK	13
2.6 RELEASE/SRC/SUBDIR.MK	14
2.7 KUNGFU32 应用参数说明	15
2.7.1 编译器 <i>compiler</i>	15
2.7.2 汇编器 <i>assemble</i>	16
2.7.3 链接器 <i>ld</i>	16
2.7.4 库管理 <i>ar</i>	17

修改记录

序号	日期	版本	变更及说明	其他
1	2024-07-04	V1.0	初稿	
2	2026-06-11	V1.1	工具链组织关系更新	
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				

1. 工具的介绍

1.1 发布形式与系统说明

工具发布，资源包名称为 ChiponCC32-**xxxx**.zip。

该资源包为独立的工具链二进制包。

该工具基于 CentOS-6.3-x86_64 的 Linux 64 位系统编译发布

1.2 文件结构

ChipONCC32 工具根目录

ccr l _issue	精简内核工具链，发布型号版本，示例编号为 r1
bin	基本工具路径，如 kf32-gcc kf32-ar
kf32	
bin	编译器驱动调用工具的实际功能程序，如 ar ld
lib	版本下的芯片实现库与最小系统相关库，如 libmath.a 以及基于标准 newlibs 和 libstdc++-v3 的库
gcc	
kf32	
4.7.0	框架编译器，驱动库路径，如 ccl libgcc.a。
scripting	对于版本下默认链接器脚本文件和调试时的描述文件
chipregister	服务 IDE 软件的芯片寄存器格式描述文件集合
include	芯片 C 语言和汇编语言编写的服务寄存器声明头文件
include\Sys	最小系统相关的服务方法头文件，如 math、malloc 等
include\Std_Sys	基于标准 newlibs 组织的 c 头文件
include\Std_Sys\c++\4.7.0	基于标准 libstdc++-v3 的 c++头文件
common	服务构建的三方依赖文件或其他功能工具
在 include\Sys 和 ccr l _issue\lib	提供基于最小需要的标准库与运行库
在 include\Std_Sys 和 ccr l _issue\lib	提供基于 newlibs 与 libstdc++-v3 的嵌入式

适配标准 c 库和 c++库支持。

最小库打印基于 usart 映射的轻量级打印，标准库支持系统级的功能和 c++容器类支持，打印基于文件系统适配，同时需要使用 malloc 方法支持。

1.3 工具使用简介

- 解压工具包到本地目录下，并根据需要调整资源包的路径名称，如 \$HOME/ChipONCC32
- 可选手动 shell 命令配置下调用 make 进行源码项目的编译与链接【工具生成 makefile 文件】。
cd 到当前项目执行输出目录，如/home/gjz/USART_Async2/Release, 执行：

```
PATH=./:%HOME/ChipONCC32/ccr1_issue/bin:$PATH:$HOME/bin
export PATH
make all
```

若要清理依赖目标 hex 和 o 文件，可执行 `make clean`
可通过用户目录下配置文件 `“.bash_profile”` (centos) 或 `“.profile”` (ubuntu) 尾部添加的方式提前添加工具集路径。
- 工具输出 elf 的链接文件，当使能 `-gstabs+` 或 `-gdwarf-3` (推荐) 选项下的 elf 文件用于启动调试器。链接器 ld 默认输出 map，但不输出 lst、hex 文件，可调用工具链 objcopy 输出相应的 bin 或 hex 格式文件。也可以给链接器添加选项 `--kf32-autoihex` 集成完成输出 hex 文件并根据选项输出反汇编的 lst 文件。
- 当源码为 C 语言时，参数 `-save-temps` 实现过程源文件同名的预处理 i 文件和 s 汇编文件。可根据需要参数控制输出文件的路径，如 `-save-temps=obj`。
- 需要传递选项函数功能独立成段，数据独立成段，参数 `-ffunction-sections` `-fdata-sections`

可以根据需要合理选择项目构建管理工具，如 make\cmake\scon。

1.4 主要工具与举例说明

更多工具的选项可见 gnu 通用工具使用描述。gcc 实现基于 gcc4.7.0，汇编链接等工具实现基于 binutils 2.24，gdb 实现基于 gdb7.8。常规与定制参数使用见环境应用举例。

1.4.1 kf32-g++

C++编译器，格式：\${COMMAND} \${FLAGS} \$< -o \${OUTPUT}

如

`kf32-g++`

`-I"/home/gjz/workspace32/4444"`

`-I"/home/gjz/ChipONCC32/include"`

`-I"/home/gjz/ChipONCC32/include/Sys"`

`-save-temps -fno-builtin-printf`

```
-c
-funsigned-char
-Wa,--kf32-arch=kf32r,-I"/home/gjz/ChipONCC32/include"
-ffunction-sections -fdata-sections
-D"KF32F130GQS"
-std=gnu++11 -O2 ../main.cpp -o "main.o"
```

1.4.2 kf32-gcc

C 编译器，格式：\${COMMAND} \${FLAGS} \$< -o \${OUTPUT}

如

```
kf32-gcc -MMD
-I"/home/gjz/workspace32/22222"
-I"/home/gjz/ChipONCC32/include"
-I"/home/gjz/ChipONCC32/include/Sys"
-save-temps
-fno-builtin-printf -fno-builtin-fprintf -fno-builtin-fputs
-c
-funsigned-char -fsigned-bitfields
-Wa,--kf32-arch=kf32r,-I"/home/gjz/ChipONCC32/include"
-ffunction-sections -fdata-sections
-D"KF32A251IQT" -Wno-packed-bitfield-compat
-std=gnu99 -O2 ../main.c -o "main.o"
```

1.4.3 kf32-as

汇编器，格式：\${COMMAND} \${FLAGS} \${INPUTS} \${OUTPUT_FLAG} \${OUTPUT}

如

```
kf32-as -MD ../main.d
--kf32-arch=kf32r -I"/home/gjz/workspace32/33333"
-I"/home/gjz/ChipONCC32/include"
-I"/home/gjz/ChipONCC32/include/Sys"
--defsym KF32A250GQS=1 "../main.asm" -o "main.o"
```

1.4.4 kf32-ld

链接器，格式：\${COMMAND} \${INPUTS} \${FLAGS} -o "\${ProjName}.elf" -Map "\${ProjName}.map"

如

```
kf32-ld ../_config/startup.o ../_config/vector.o ../kf_it.o ../main.o
-L"/home/gjz/ChipONCC32/ccr1_issue/lib"
-L"/home/gjz/workspace32/22222"
-lIQmath-R1 -lmath -lio -lstring -lstdlib -lctype -lcrtv1
-T"../AAAAAAA.ld"
```

```
--kf32-autoihex --kf32-arch=kf32r --kf32-z  
--gc-sections -o "22222.elf" -Map "22222.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 crti.o crtn.o crtbegin.o crtend.o，若不需要可添加选项 -nostdlib（推荐）。使用 kf32-gcc 下链接器专有选项请使用 -Wl,option 或 -Xlinker option，

最小系统依赖库分布在 -lmath -lio -lstring -lstdlib -lctype -lcrtvx 中。

系统级支持的库分布在 -lgcc -lc -lm -lstdc++ -lsupc++ 中。

库传达具有依赖关系，推荐使用如 --start-group -lgcc -lc -lm -lstdc++ -lsupc++ --end-group 的建立组。

1.4.5 kf32-objdump

elf 文件反汇编程序。

如：

```
kf32-objdump -G TestElf1.elf > TestElf1.lst  
kf32-objdump -S -l [-z ] [--kf32-arch=kf32r]  
--section=.text --section=.data --section=.bss  
project.elf > project.lst
```

1.4.6 kf32-objcopy

elf 文件输出可执行下载格式程序。

如：

```
kf32-objcopy -O ihex project.elf project.hex  
kf32-objcopy -O srec --srec-len 16 --srec-forceS3 project.  
elf project.s19
```

支持输出格式 ihex 和 s19。链接器可通过参数的自动调用该程序实现 elf 文件到 hex 文件的输出。工具同样支持 bin 格式输出（不推荐，不同资源段空间不连续）。

1.4.7 kf32-readelf

elf 文件或 o 文件的信息查看器。

```
kf32-readelf -h project.elf
```

1.4.8 kf32-size

统计项目的 flash 和 ram 的使用情况。

如：kf32-size XXX.elf 该输出具有更多定制信息，即围绕 .data 端具有 data 和 bss 不可区分，输出关键脚本变量用于调用者计算。

输出:

<u>text</u>	<u>data</u>	<u>bss</u>	<u>eeep</u>	<u>dec</u>	<u>hex</u>	<u>filename</u>
8684	0	0	4096	12780	31ec	XXX.elf

其他选项如数组显示进制格式, 统计格式, 如

--format={sysv|berkeley} Select output style (default is berkeley)

--radix={8|10|16} Display numbers in octal, decimal or hex

1.4.9 kf32-ar

归档文件管理程序, 即库程序管理程序。

kf32-ar rcs libA.a libA.o [*o ...] 基于对象文件集创建或添加到库

kf32-ar d libA.a libA.o [*o ...] 删除库中对象文件

kf32-ar t libA.a 查看库内容

更多参数见标准工具支持, 查看 kf32-ar --help

1.4.10 kf32-gdb

硬件仿真调试器, GDB 通过获取 ctrl+C 信号来识别暂停命令。示例调试器使用过程如下, 更多命令可参考 gdb 自身相关文档。

1、启动: gdb /.../XX.elf

2、示例配置选项

```
set confirm off
set pagination off
set step-mode off
set print elements unlimited
set print repeats unlimited
```

3、启动目标机:

target kf32remote /dev/ttyS2 --kfvoltage=3.0 --kfd-speed=6
--kft-code=2 --kfd-code=40 --kfinstruct=kf32r --kfprint --arch=
XXX.def (如/home/gjz/ChiponCC32/scripting/ccr1_issue/KF32F350MQ
V.def)

其中:

kf32remote 硬件仿真的基于本地串口模式的仿真模式
/dev/ttyS2 这里示例对应为串口设备 2。

--kfvoltage 实际支持 3.3V 或 5.0V 两个档位

--kfd-speed 支持数字 1-250 的空循环时钟参数, 典型参数 6

--kft-code 芯片类型编码, 如当前 **KF32A150 系列 2, KF32A156 系列 3, KF32A158 系列 4, KF32A138 系列 6。**

--kfd-code 行为差异控制, 如 0 进入调试并复位, 控制运行到 main;
如 1 进入调试并复位, 控制运行到 startup; 2 进入调试并暂停芯片, 3 进入调试

并暂停芯片，针对芯片类型编码 2 3 同步进行关闭看门狗功能。

--kfprint 配合 ChipON IDE 的命令交互信息输出

--kfinspect 指定芯片框架集，如 kf32r。

4、加载目标机 load 注：该命令更新状态，但不实现芯片程序的下载，下载需求其他工具下载后再启动调试。

5、设置启动运行断点 break main

6、运行到断点：r

7、其他参考调试命令

查看通用寄存器：info registers info registers r0 r1

修改通用寄存器：set var \$r0=0x1234567

查看指定内存地址：x /9w 0x500

查看变量起始地址的值：x /9w arr

修改指定地址的值：

set *0x10000000=0x12345678 set *(short *)0x10000000=0x1234

查看函数的 flash 空间：x /16w main

查看全局变量：print variable

查看局部变量：info locals

修改局部变量：set x=0x1

函数参数查询：info args

断点设定：b main.c:144

查询断点：info breakpoint

删除断点：delete delete 2

运行：continue

单步：stepi step

单步跳过：nexti next

单步返回：finish

复位：kfreset

退出：quit

当控制芯片处于运行状态时，

2. 基于 gmake 环境的应用举例

假定程序名称 USART_Async2，linux 用户 gjz，工具目录/home/gjz/ChipONCC32，样例程序路径/home/gjz/USART_Async2。

样例程序目录结构：

config	源码中断向量表与 ram 初始化引导程序
inc	外设库的头文件目录
src	外设库的 C 文件目录
KF32F350_Include.h	外设库的应用 include 声明
kf_it.c	默认硬件级中断入口程序样例

main.c	项目入口程序文件
sys.c sys.h	时钟初始化相关声明
Usart.c Usart.h	串口应用服务函数
Debug	调试模式编译执行与输出文件夹
Release	发布模式编译执行与输出文件夹
Release/runmake	样例注册工具路径与启动编译文件
Debug/runmake	样例注册工具路径与启动编译文件

基于 IDE 生成 makefile 或手动创建 makefile 文件集基于 Release 输出如下的部分示例文件与内容如下。

2.1 Release/Makefile

```
#####
#
#####
-include ../makefile.init
RM := rm -rf
# All of the sources participating in the build are defined here
-include sources.mk
-include src/subdir.mk
-include config/subdir.mk
-include subdir.mk
-include objects.mk

-include $(C_DEPS)

-include ../makefile.defs

#Add inputs and outputs from these tool invocations to the build
variables

#
all: USART_Async2.elf secondary-outputs
#
USART_Async2.elf: $(OBS) $(USER_RELS)
    @echo 'Building target: $@'
    @echo 'Invoking: C Project Linker Release'
    kf32-ld $(OBS) $(USER_RELS) $(LIBS) -L"/home/gjz/
ChiponCC32/ccr1_issue/lib" -L"/home/gjz/USART_Async2" -lIQmath-R1
-lSeriesDIServices -lmath -lio -lstring -lstdlib -lctype -lcrtv1
-T"/home/gjz/ccr1_issue/scripting/KF32F350MQV.ld" --gc-sections -o
```

```
"USART_Async2.elf" -Map "USART_Async2.map"
@echo 'Finished building target: $@'
@echo ' '

USART_Async2.s19: $(OBJS) $(EXECUTABLES) $(USER_RELS) $(LD_SRCS)
# @echo 'Invoking: Objcopy File Debug'
@kf32-objcopy -O srec --srec-len 16 --srec-forceS3
"USART_Async2.elf" "USART_Async2.s19"
@echo 'Finished building: $@'
# @echo ' '

USART_Async2.lst: $(OBJS) $(EXECUTABLES) $(USER_RELS) $(LD_SRCS)
# @echo 'Invoking: Objdump Disassemble Debug'
@kf32-objdump -S -l --section=.text --section=.data --section=.bss
--kf32-arch=kf32r "USART_Async2.elf" > "USART_Async2.lst"
@echo 'Finished building: $@'
# @echo ' '

#
clean:
-$(RM) $(OBJS)$(EXECUTABLES)$(C_DEPS) USART_Async2.elf
USART_Async2.lst USART_Async2.s19
-@echo ' '

secondary-outputs: USART_Async2.lst USART_Async2.s19

.PHONY: all clean dependents
.SECONDARY:

-include ../makefile.targets
```

2.2 Release/objects.mk

```
#####
#
#####

USER_OBJS :=

LIBS :=
```

2.3 Release/sources.mk

```
#####  
#  
#####  
  
O_SRCS :=  
C_SRCS :=  
OBJ_SRCS :=  
C_DEPS :=  
OBJS :=  
EXECUTABLES :=  
  
# Every subdirectory with source files must be described here  
SUBDIRS := \  
.  
src \  
config \
```

2.4 Release/subdir.mk

```
#####  
#  
#####  
  
# Add inputs and outputs from these tool invocations to the build variables  
C_SRCS += \  
../Usart.c \  
../kf_it.c \  
../main.c \  
../sys.c  
  
OBJS += \  
./Usart.o \  
./kf_it.o \  
./main.o \  
./sys.o
```

```
C_DEPS += \  
./Usart.d \  
./kf_it.d \  
./main.d \  
./sys.d
```

Each subdirectory must supply rules for building sources it contributes

```
%.o: ../%.c  
    @echo 'Building file: $<'  
    @echo 'Invoking: C Compiler'  
    kf32-gcc -MMD -I"/home/gjz/USART_Async2"  
-I"/home/gjz/USART_Async2/inc" -I"/home/gjz/ChipONCC32/include"  
-I"/home/gjz/ChipONCC32/include/Sys" -save-temps -fno-builtin-printf  
-fno-builtin-fprintf -fno-builtin-fputs -c -funsigned-char  
-fsigned-bitfields -Wa,--kf32-arch=kf32r -ffunction-sections  
-fdata-sections -D"KF32F350MQV" -Wno-packed-bitfield-compat -std=gnu99  
-O2 -gdwarf-3 $< -o "$@"  
    @echo 'Finished building: $<'
```

2.5 Release/config/subdir.mk

```
#####  
#  
#####  
  
# Add inputs and outputs from these tool invocations to the build variables  
C_SRCS += \  
../config/startup.c \  
../config/vector.c  
  
OBJS += \  
./config/startup.o \  
./config/vector.o  
  
C_DEPS += \  
./config/startup.d \  
./config/vector.d
```

```
# Each subdirectory must supply rules for building sources it contributes
config/%.o: ../config/%.c
    @echo 'Building file: $<'
    @echo 'Invoking: C Compiler'
    kf32-gcc -MMD -I"/home/gjz/USART_Async2"
-I"/home/gjz/USART_Async2/inc" -I"/home/gjz/ChipONCC32/include"
-I"/home/gjz/ChipONCC32/include/Sys" -save-temps -fno-builtin-printf
-fno-builtin-fprintf -fno-builtin-fputs -c -funsigned-char
-fsigned-bitfields -Wa,--kf32-arch=kf32r -ffunction-sections
-fdata-sections -D"KF32F350MQV" -Wno-packed-bitfield-compat -std=gnu99
-O2 -gdwarf-3 $< -o "$@"
    @echo 'Finished building: $<'
```

2.6 Release/src/subdir.mk

```
#####
#
#####

# Add inputs and outputs from these tool invocations to the build variables
C_SRCS += \
../src/kf32f350_gpio.c \
../src/kf32f350_int.c \
../src/kf32f350_osc.c \
../src/kf32f350_pclk.c \
../src/kf32f350_rst.c \
../src/kf32f350_usart.c

OBJS += \
./src/kf32f350_gpio.o \
./src/kf32f350_int.o \
./src/kf32f350_osc.o \
./src/kf32f350_pclk.o \
./src/kf32f350_rst.o \
./src/kf32f350_usart.o

C_DEPS += \
./src/kf32f350_gpio.d \
./src/kf32f350_int.d \
./src/kf32f350_osc.d \
```

```
./src/kf32f350_pclk.d \  
./src/kf32f350_rst.d \  
./src/kf32f350_usart.d  
  
# Each subdirectory must supply rules for building sources it contributes  
src/%.o: ../src/%.c  
    @echo 'Building file: $<'  
    @echo 'Invoking: C Compiler'  
    kf32-gcc -MMD -I"/home/gjz/USART_Async2"  
-I"/home/gjz/USART_Async2/inc" -I"/home/gjz/ChipONCC32/include"  
-I"/home/gjz/ChipONCC32/include/Sys" -save-temps -fno-builtin-printf  
-fno-builtin-fprintf -fno-builtin-fputs -c -funsigned-char  
-fsigned-bitfields -Wa,--kf32-arch=kf32r -ffunction-sections  
-fdata-sections -D"KF32F350MQV" -Wno-packed-bitfield-compat -std=gnu99  
-O2 -gdwarf-3 $< -o "$@"  
    @echo 'Finished building: $<'
```

2.7 KungFu32 应用参数说明

2.7.1 编译器 compiler

添加-MMD基于源文件名输出依赖文件，如输出main.d。内容如：

```
main.o: ../main.c ../sys.h /home/gjz/  
ChiponCC32/include/Sys/string.h \  
    /home/gjz/ChipONCC32/include/Sys/stdint.h \  
/home/gjz/ChipONCC32/include/Sys/stddef.h ../KF32F350_Include.  
h \  
    /home/gjz/USART_Async2/inc/KF32F350.h \  
/home/gjz/USART_Async2/inc/kf32f350_osc.h \  
/home/gjz/USART_Async2/inc/KF32F350.h \  
/home/gjz/USART_Async2/inc/kf32f350_int.h \  
/home/gjz/USART_Async2/inc/kf32f350_gpio.h \  
/home/gjz/USART_Async2/inc/kf32f350_usart.h \  
/home/gjz/USART_Async2/inc/kf32f350_rst.h \  
/home/gjz/USART_Async2/inc/kf32f350_pclk.h ../Usart.h \  
/home/gjz/ChipONCC32/include/Sys/stdio.h \  

```

/home/gjz/ChipONCC32/include/Sys/stdarg.h

添加头文件搜索路径

-I"/home/gjz/USART_Async2", 指向主项目路径, 可以不引入子路径下相对路径访问实现

-I"/home/gjz/ChipONCC32/include" 型号头文件目录所在

-I"/home/gjz/ChipONCC32/include/Sys" 系统头文件目录所在

添加-fno-builtin-printf -fno-builtin-fprintf -fno-builtin-fputs

即对应打印函数不映射到 gcc 的函数格式, 意味着采用明面函数名的 KF32 库方法实现, 如代码 put 方法而调用 gcc 函数 fwrite。

添加-Wa, --kf32-arch=kf32r

即选择对应的芯片指令集, 当前选择 kf32r。

-save-temps 保存过程文件

即预处理后的 i 文件和编译的汇编文件, 如 main.s

2.7.2 汇编器 assemble

输出依赖文件

-MD \$(dir \$@)\$(basename \$(notdir \$@)).d

指定芯片内核

--kf32-arch=kf32r

添加头文件搜索路径

-I"/home/gjz/ChipONCC32/include" 型号头文件目录所在

2.7.3 链接器 ld

添加库路径

-L"/home/gjz/ChipONCC32/ccr1_issue/lib" 系统库路径

-L"/home/gjz/USART_Async2" 当前项目下主目录下库路径

基本 KungFu 库

-lIQmath-R1

定点浮点库

-lSeriesDIServices

串口交换协议接口

-lmath -lio -lstring -lstdlib -lctype

系统相关的实现库, 与标准数学库-lm 和标准 c 相关库

-lc 的使用具有差异

-lcrtvx

型号相关的必须库, 如-lcrtv1, -lcrtv2, -lcrtv3 对应不同的芯片系列 2/3/4(A150|A156|A158), 当不调用库打印和部分系列的库 FlashAPI 方法时, 任意一个存在库版本在当前编译器支

持的型号下工具是可行。

指定脚本

-T"/KF32F350MQV.ld" 项目主目录的定义脚本

或

-T"/home/gjz/chipONCC32/ccr1_issue/scripting/KF32F350MQV.ld"

工具目录发布的默认型号脚本

不传递该参数下的默认使用工具自带脚本。不具有 KF32 适配性。

死段优化选项

--gc-sections

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

辅助选项

--kf32-autoihex 集成自动调用实现输出 hex 可下载程序文件

--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-nocheck app 应用下重映射向量表入口下不输出

校验和选项，否则默认 0 地址的向量表，并地址 0x001C-0x001F 生成程序存在标记值。

2.7.4 库管理 ar

创建并添加或添加库文件：

kf32-ar rcs libA.a libA.o [libB.o]

删除指定库文件：

kf32-ar d libA.a libA.o [libB.o]

查看更多选项：kf32-ar --h