

KF32 Tool Chain Guide for Windows System

Contents

1. TOOL INTRODUCTION	3
1.1 RELEASE FORM	3
1.2 FILE STRUCTURE	3
1.3 MAIN TOOLS AND INSTRUCTIONS	4
1.3.1 <i>gmake.exe</i>	4
1.3.2 <i>kf32-g++.exe</i>	4
1.3.3 <i>kf32-gcc.exe</i>	4
1.3.4 <i>kf32-as.exe</i>	5
1.3.5 <i>kf32-ld.exe</i>	5
1.3.6 <i>kf32-objdump.exe</i>	6
1.3.7 <i>kf32-objcopy.exe</i>	6
1.3.8 <i>kf32-readelf.exe</i>	6
1.3.9 <i>kf32-size.exe</i>	6
1.3.10 <i>kf32-ar.exe</i>	7
1.3.11 <i>kf32-gdb.exe</i>	7
2. APPLICATION BASED ON KF32 IDE	9
2.1 MAKEFILE:	9
2.2 OBJECTS.MK	11
2.3 SOURCES.MK	11
2.4 SUBDIR.MK	12
2.5 USB-FS-DEVICE_DRIVER /SUBDIR.MK	13
2.6 DIFFERENCE OF DEBUG AND RELEASE OPTIONS	14
2.7 INSTRUCTIONS OF KUNGFU32 DIFFERENTIAL PARAMETERS	14
2.7.1 <i>Compiler kf32-gcc:</i>	14
2.7.2 <i>Compiler kf32-as:</i>	15
2.7.3 <i>Linker kf32-ld:</i>	15
2.7.4 <i>Library management kf32-ar:</i>	17

1. Tool Introduction

1.1 Release form

It's updated with ChipON IDE KF32 software and located at `ChiponCC32` folder under the installation directory, Or delivered based on Zip compressed files. This document is based on Windows version, and it's applicable for Linux version (path format difference).

1.2 File structure

<code>ccr1_issue</code>	Current release version, the number is r1
<code>ccr1_issue\lib</code>	For chip algorithm library and system-related library, such as <code>libmath.a</code> or Based on the standard <code>newlibs</code> and <code>libstd c++- v3</code> libraries <code>libc libm libstd c++libsupc++</code> .
<code>ccr1_issue\scripting</code>	For linker script files of this version
<code>ccr1_issue\bin</code>	Basic tool path, such as <code>kf32-gcc.exe</code> and <code>kf32-ar.exe</code>
<code>ccr1_issue\kf32\bin</code>	Tools of the tool chain (assembler, linker, etc.), such as <code>kf32-gcc.exe</code> and <code>kf32-ar.exe</code>
<code>chipregister</code>	Set of files that describe chip register, used for IDE software
<code>include</code>	Header files of chip register operations written in C or assembly language
<code>include\Sys</code>	Header files related to minimum adaptive system, such as <code>math</code> , <code>malloc</code> , etc.
<code>include\Std_Sys</code>	C header files based on the standard <code>newlibs</code> organization
<code>include\Std_Sys\c++\4.7.0</code>	C++header files based on standard <code>libstd c++- v3</code>
<code>common</code>	General tool path, such as <code>gmake</code> , <code>rm</code> , etc.
Provide standard and runtime libraries based on minimum requirements in <code>include\Sys</code> and <code>ccr1_issue\lib</code>	
Provide embedded adaptation standard C and C++library support based on <code>newlibs</code> and <code>libstd c++-v3</code> in <code>include\Std_Sys</code> and <code>ccr1_issue\lib</code> .	
Minimum library printing is a lightweight printing based on <code>usart</code> mapping. The standard library supports system level functionality and C++container class support. Printing is based	

on file system adaptation and requires the use of malloc method support.

1.3 Main tools and instructions

For more options of tools, please see gnu tool description. Refer to gcc 4.7.0 for gcc, binutils 2.24 for binutils and gdb 7.8 for gdb.

1.3.1 gmake.exe

The tool based on makefile for managing compilation. It's actually a rename of make. The current version is 3.81.

1.3.2 kf32-g++.exe

- C++ compiler

Format: `${COMMAND} ${FLAGS} ${INPUTS} ${OUTPUT_FLAG} ${OUTPUT}.`

For example:

```
kf32g++ -I"C:\Program Files (x86)\ChipON
IDE\KungFu32\ChiponCC32\include" -I"C:\Program Files (x86)\ChipON
IDE\KungFu32\ChiponCC32\include\Sys" -save-temps -fno-builtin-printf
-c -funsigned-char -Wa,--kf32-arch=kf32r,
-I"C:\Program Files (x86)\ChipON IDE\KungFu32\ChiponCC32\include"
-ffunction-sections -fdata-sections -D"KF32F130GQS" -std=gnu++11
-O2 ../main.cpp -o "main.o"
```

1.3.3 kf32-gcc.exe

- Driving C compiler, can also be used for assembling and linking the output target files

Format: `${COMMAND} ${FLAGS} ${INPUTS} ${OUTPUT_FLAG} ${OUTPUT}`

For example:

```
kf32cc -I"C:\Program Files (x86)\ChipON
IDE\KungFu32\ChiponCC32\include" -I"C:\Program Files (x86)\ChipON
IDE\KungFu32\ChiponCC32\include\Sys"-save-temps -fno-builtin-printf
-c -funsigned-char -Wa,--kf32-arch=kf32r,
-I"C:\Program Files (x86)\ChipON IDE\KungFu32\ChiponCC32\include"
```

```
-ffunction-sections -fdata-sections -D"KF32F130GQS" -std=gnu++11  
-O2 ../main.c -o "main.o"
```

1.3.4 kf32-as.exe

- Assembler

Format: `${COMMAND} ${FLAGS} ${INPUTS} ${OUTPUT_FLAG} ${OUTPUT}`

For example:

```
kf32-as -MD ./main.d --kf32-arch=kf32r -I"d:/workspace32/demo"  
-I"C:\Program Files (x86)\ChipON IDE\KungFu32\ChiponCC32\include"  
--defsym KF32A250GQS=1 "../main.asm" -o "main.o"
```

1.3.5 kf32-ld.exe

- Linker

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

For example:

```
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"
```

Note: The output files of linker are target files with elf format. `--kf32-autoihex` option achieves the automatic calling of hex file that is output from `objcopy`. This file can be programmed into MCU directly. Library files should be added after the compiler outputs o file. If other implementation of library is adopted, the library should be added first.

If `kf32-gcc` is used by linker, it will pass the library by default. The path of these libraries (`libgcc.a crt0.o crti.o crtn.o crtbegin.o crtend.o`) is `\lib\gcc\kf32\4.7.0`. Please add the option `-nostdlib` and pass the library of `-lmath -lio -lstring -lstdlib -lctype` instead.

When you use kf32-gcc for linker, you need to give the link option with

-Wl, option **OR** -Xlinker.

The minimum system dependency library is distributed in -lm -lio -lstring -lstlib -lctype -lcrtvx.

The system level supported libraries are distributed in -lgcc -lc -lm -lstdc++ -lsupc++.

Library communication has dependency relationships, and it is recommended to use group building methods such as --start-group -lgcc -lc -lm -lstdc++ -lsupc++ --end-group.

1.3.6 kf32-objdump.exe

- elf file dissembler

For example:

```
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
```

1.3.7 kf32-objcopy.exe

elf file output can execute the program with downloading format.

For example:

```
kf32-objcopy -O ihex project.elf project.hex
```

Supports output format ihex, binary and srec (S19). The linker can achieve the output of hex file from elf file by automatically calling parameters.

1.3.8 kf32-readelf.exe

- Information viewer of elf files or o files

1.3.9 kf32-size.exe

- Record the usage of RAM and FLASH in the project, please see software adaptation output for more details.

For example:

```
kf32-size XXX.elf
```

Output:

text	data	bss	eep	dec	hex	filename
8684	0	0	4096	12780	31ec	XXX.elf

1.3.10 kf32-ar.exe

- Management program for archived files, namely management program for library programs

<code>kf32-ar rcs libA.a libA.o [*o ...]</code>	Build or add to library based on object file set
<code>kf32-ar d libA.a libA.o [*o ...]</code>	Delete object file from library
<code>kf32-ar t libA.a</code>	View the content of library

Mainly used for the library publication management of the object files (o files) which are generated by compiling assemble program. The library files that end with a and start with lib can be simplified in the linker. For example, both `-lFunc` or `-llibFunc.a` can be used for `libFunc.a`.

1.3.11 kf32-gdb.exe

- Hardware simulation debugger

1. Start `gdb.exe XX.elf`

2. Configure the options

```
set confirm off
set pagination off
set step-mode off
set print elements 1024
set print repeats unlimited
```

3. Start up the hardware simulation target machine

--kft-code=[234] corresponds to the internal number of kungfu32 chip,
 --kfd-code=[0123] represents the method to start debugging, 0 means reset and run till main function; 1 means reset and run till startup; 2 means that switch off the watchdog, pause and debug the current program; 3 reserved.
 The effective port number range of the programmer is COM1-COM256.

For example:

```
target kf32remote
COM11
--kfvoltage=3.0 --kfd-speed=2 --kft-code=2 --kfd-code=0
--kfinspect=kf32r --kfprint
--arch="C:\Program Files (x86)\ChipON IDE\KungFu32\ChipONCC32\ccr1
_issue\scripting\KF32F350MQV.def"
```

Software simulation does not support peripheral functions, it only debugs the code logic. The command for software simulation is:

```
target kf32sim --arch="C:\Program Files (x86)\ChipON IDE\KungFu32\Chip
ONCC32\ccr1_issue\scripting\KF32F350MQV.def" --kfinspect=kf32r --kf
print
```

Note: Differential configuration of structure files in initializing is not required at this time --arch= to any empty file is fine.

4. Load the target machine

load

Auxiliary tool gets the pid code of the program, which can be used for sending pause, such as `kill.exe -2 8892`. That is, Ctrl+C means pause. In addition, “taskkill /f /t /im kf32-gdb.exe” can be used to exit the debugger by force.

6. Breakpoints break main.

7. Run to the breakpoint: `r` , [Use command `c` to continue running, where `c` stands for continue]

8. Other commands for debugging

Enable/Disable of debugging interrupt: `intctl on intctl off`,
Integrated in single step and run function.

View general registers: `info registers info registers r0 r1`

Modify general registers: `set var $r0=0x1234567`

View the content of memory with specified address (range): `x /9w 0x500`

View the start address of a variable: `x /9w arr`

Modify the value of specified address:

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

View the flash space of a function: `x /16w main`

View global variables: `print variable`

View local variables: `info locals`

Modify local variables: `set x=0x1`

Consult function arguments: `info args`

```

Set breakpoints: b main.c:144
Consult breakpoints: info breakpoint
Delete breakpoints: delete delete 2
GDB identifies pause commands by getting ctrl+C signal.
Continue running: continue
Step into: stepi step
Step over: nexti next
Step return: finish
Reset: kfreset
Exit: quit

```

2. Application based on KF32 IDE

Take the project `KungFu32_ProDebug_WinUSB` as an example. The tool software is located at `C:/Program Files (x86)/ChipON IDE/KungFu32`, under the directory of `ChiponCC32`.

The structure of folders is as follows:

```

User, User\inc,
StdPeriph_Driver, StdPeriph_Driver\inc,
_config,
USB-FS-Device_Driver, USB-FS-Device_Driver\inc.

```

`gmake` generated based on IDE requires the following dependence and parameter selection. See the actual project output for details.

Note 1: If the file structure is not adjusted any more, `makefile` and relevant files can be pre-output by IDE. These files can then be called by console.

Note 2: the `makefile` of the command line mechanism has constraints on the length of parameters. The length of `cmd` command line is 8192 characters. The length of `make` is limited to 32767 characters via `creatprocess`, while 2048 characters via `shellexecute`.

2.1 Makefile:

```

#####
# Files generated automatically or manually maintained
#####

```

```
-include ../makefile.init

RM := rm -rf

# All of the sources participating in the build are defined here
-include sources.mk
-include _config/subdir.mk
-include User/subdir.mk
-include USB-FS-Device_Driver/subdir.mk
-include StdPeriph_Driver/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 targets
all: KungFu32_ProDebug_WinUSB.hex

# Invoking the tool
KungFu32_ProDebug_WinUSB.hex: $(OBJS) $(USER_RELS)
    @echo 'Building target: $@'
    @echo 'Invoking: C Linker Release'
    kf32-ld $(OBJS) $(USER_RELS) $(LIBS) -L"C:/Program Files (x86)/ChipON
IDE/KungFu32/ChiponCC32/lib/ccr1_issue"
-L"D:\workspace32\KungFu32_ProDebug_WinUSB" -lIQmath-R1
-lSeriesDIServices -lmath -lio -lstring -lstdlib -lctype -lcrtv1
-T"../KF32F341IQS_9KWinUSB.ld" --gc-sections -o
"KungFu32_ProDebug_WinUSB.elf" -Map "KungFu32_ProDebug_WinUSB.map"
    @echo 'Finished building target: $@'
    @echo ' '

# Other targets
clean:
    -$(RM) $(OBJS) $(EXECUTABLES) $(C_DEPS) KungFu32_ProDebug_WinUSB.hex
    -@echo ' '
```

```
.PHONY: all clean dependents
.SECONDARY:

    -include ../makefile.targets
```

2.2 objects.mk

```
#####
# Files generated automatically or manually maintained
#####

USER_OBJS :=

LIBS :=
```

2.3 sources.mk

```
#####
#Files generated automatically or manually maintained
#####

O_SRCS :=
C_SRCS :=
OBJ_SRCS :=
C_DEPS :=
OBJS :=
EXECUTABLES :=

# Every subdirectory with source files must be described here
SUBDIRS := \
. \
_config \
User \
USB-FS-Device_Driver \
StdPeriph_Driver \
```

2.4 subdir.mk

```
#####
# Files generated automatically or manually maintained
#####

# Add inputs and outputs from these tool invocations to the build variables
C_SRCS += \
./flash.c \
./main.c \
./system_init.c

OBJS += \
./flash.o \
./main.o \
./system_init.o

C_DEPS += \
./flash.d \
./main.d \
./system_init.d

# Each subdirectory must supply rules for building sources it contributes
%.o: ../%.c
    @echo 'Building file: $<'
    @echo 'Invoking: C Compiler Release'
    kf32-gcc -MMD -I"D:\workspace32\KungFu32_ProDebug_WinUSB"
-I"D:\workspace32\KungFu32_ProDebug_WinUSB\USB-FS-Device_Driver\inc"
-I"D:\workspace32\KungFu32_ProDebug_WinUSB\StdPeriph_Driver\inc"
-I"D:\workspace32\KungFu32_ProDebug_WinUSB\User\inc" -I"C:/Program
Files (x86)/ChipON IDE/KungFu32/ChiponCC32/include" -I"C:/Program Files
(x86)/ChipON IDE/KungFu32/ChiponCC32/include/Sys" -save-temps
-fno-builtin-printf -fno-builtin-fprintf -fno-builtin-fputs -c
-funsigned-char -fsigned-bitfields -Wa,--kf32-arch=kf32r,-I"C:/Program
Files (x86)/ChipON IDE/KungFu32/ChiponCC32/include"
-ffunction-sections -fdata-sections -D"KF32F341IQS"
-Wno-packed-bitfield-compat -std=gnu99 -O2 $< -o "$@"
    @echo 'Finished building: $<'
```

2.5 USB-FS-Device_Driver /subdir.mk

```
#####
# Files generated automatically or manually maintained
#####

# Add inputs and outputs from these tool invocations to the build variables
C_SRCS += \
./USB-FS-Device_Driver/usb_core.c \
./USB-FS-Device_Driver/usb_init.c \
./USB-FS-Device_Driver/usb_int.c \
./USB-FS-Device_Driver/usb_mem.c \
./USB-FS-Device_Driver/usb_regs.c \
./USB-FS-Device_Driver/usb_sil.c

OBJS += \
./USB-FS-Device_Driver/usb_core.o \
./USB-FS-Device_Driver/usb_init.o \
./USB-FS-Device_Driver/usb_int.o \
./USB-FS-Device_Driver/usb_mem.o \
./USB-FS-Device_Driver/usb_regs.o \
./USB-FS-Device_Driver/usb_sil.o

C_DEPS += \
./USB-FS-Device_Driver/usb_core.d \
./USB-FS-Device_Driver/usb_init.d \
./USB-FS-Device_Driver/usb_int.d \
./USB-FS-Device_Driver/usb_mem.d \
./USB-FS-Device_Driver/usb_regs.d \
./USB-FS-Device_Driver/usb_sil.d

# Each subdirectory must supply rules for building sources it contributes
USB-FS-Device_Driver/%.o: ../USB-FS-Device_Driver/%.c
    @echo 'Building file: $<'
    @echo 'Invoking: C Compiler Release'
    kf32-gcc -MMD -I"D:\workspace32\KungFu32_ProDebug_WinUSB"
-I"D:\workspace32\KungFu32_ProDebug_WinUSB\USB-FS-Device_Driver\inc"
-I"D:\workspace32\KungFu32_ProDebug_WinUSB\StdPeriph_Driver\inc"
-I"D:\workspace32\KungFu32_ProDebug_WinUSB\User\inc" -I"C:/Program
```

```
Files (x86)/ChipON IDE/KungFu32/ChiponCC32/include" -I"C:/Program Files
(x86)/ChipON IDE/KungFu32/ChiponCC32/include/Sys" -save-temps
-fno-builtin-printf -fno-builtin-fprintf -fno-builtin-fputs -c
-funsigned-char -fsigned-bitfields -Wa,--kf32-arch=kf32r,-I"C:/Program
Files (x86)/ChipON IDE/KungFu32/ChiponCC32/include"
-ffunction-sections -fdata-sections -D"KF32F341IQS"
-Wno-packed-bitfield-compat -std=gnu99 -O2 $< -o "$@"
@echo 'Finished building: $<'
```

2.6 Difference of debug and release options

The difference reflects on compiler options, `-O0` for debug while `-O2` for release, `-gstabs+` is both passed, so it supports debugging in any of the two mode.

2.7 Instructions of KungFu32 differential parameters

2.7.1 Compiler kf32-gcc:

Add `-MMD`, output dependent files based on the names of source files.

The content is like:

```
main.o: ../main.c ../system_init.h \
C:/Program\ Files\ (x86)/ChipON\
IDE/KungFu32/ChiponCC32/include/Sys/string.h \
C:/Program\ Files\ (x86)/ChipON\
IDE/KungFu32/ChiponCC32/include/Sys/stdint.h \
C:/Program\ Files\ (x86)/ChipON\
IDE/KungFu32/ChiponCC32/include/Sys/stddef.h \
```

```
D:\workspace32\KungFu32_ProDebug_WinUSB\StdPeriph_Driver\inc/KF32
F_BASIC.h ...
```

Add header file search path

`-I"D:\workspace32\KungFu32_ProDebug_WinUSB"`, points to the path of main project, the implementation of relative path access under sub path can be omitted.

`-I"C:/Program Files (x86)/ChipON`

`IDE/KungFu32/ChiponCC32/include"` Directory for header files of model

`-I"C:/Program Files (x86)/ChipON`

IDE/KungFu32/ChiponCC32/include/Sys" Directory for header files about system

Add-fno-builtin-printf -fno-builtin-fprintf -fno-builtin-fputs

That is, the printing does not map to function format of gcc, which means that the KF32 library method with explicit function name is adopted for implementation. For example, the "put" method is not called as the "fwrite" method.

Add-Wa, --kf32-arch=kf32r

Select the instruction set of corresponding chips. kf32r is selected currently.

-save-temps **Save process files**

i files and assembly files that have been preprocessed, such as main. s.

2.7.2 Compiler kf32-as:

Output dependent files

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

Specify the chip core

--kf32-arch=kf32r

Add the header file search path

-I"C:/Program Files (x86)/ChipON

IDE/KungFu32/ChiponCC32/include " Directory for header files of model

2.7.3 Linker kf32-ld:

Add library path

-L"C:/Program Files (x86)/ChipON

IDE/KungFu32/ChiponCC32/lib/ccr1_issue"

System library path

-L"D:\workspace32\KungFu32_ProDebug_WinUSB"

Library path under main directory of the current project

Basic KungFu Minimum system library

-lIQmath-R1

Library of fixed point or floating point

-lSeriesDIServices

Serial communication protocol

	interface
<code>-lmath -lio -lstring -lstdlib -lctype</code>	System related implementation library
<code>-lcrtv1</code>	Model related library (Necessary)
Specify Scripts	
<code>-T"../KF32F341IQS_9KWinUSB.ld"</code>	Custom script for project home directory
Or	
<code>-T"C:/Program Files (x86)/ChipON IDE/KungFu32/ChiponCC32/scripting/ccr1_issue/KF32F130GQT.ld"</code>	Default model script for tool catalog
Dead sections optimization option	
<code>--gc-sections</code>	
Specify output	
<code>-o XXX.elf</code>	Program format file that machine can execute
<code>-Map XXX.map</code>	Compiling block and symbol information
Auxiliary option	
<code>--kf32-nodisassemble</code>	Do not execute anti-assembling after linking
<code>--kf32-z</code>	Continuous 0 is also output in disassembling

Checksum function

Calculate based on address range and output checksum. Refer to relevant document for details.

```
--with-checksum-fill=0xFF
--with-checksum-fill=0x00
--with-checksum-fill=0x55ff
--with-checksum-fill=0x5500 //Select one pattern from four
Support 3 groups of configurations (1/2/3)
--with-checksumX=CRC-32
--with-checksumX=CRC-32/MPEG-2
--with-checksumX=SUM32
--with-checksumX=SIG-CODE //Supported Algorithm options
```



```
--with-checksum-addressX=Where1  
--with-checksum-sizeX=Bytes  
--with-checksum-out-addressX=Where2
```

2.7.4 Library management **kf32-ar**:

Build and add or only add library files: `kf32-ar.exe rcs libA.a libA.o` **【libB.o】**

Delete specified library files: `kf32-ar.exe d libA.a libA.o` **【libB.o】**

View more options: `kf32-ar.exe --h`