

Rockchip RTOS DSP Developer Guide

ID: RK-KF-YF-302

Release Version: V2.2.0

Release Date: 2021-03-30

Security Level: ☐Top-Secret ☐Secret ☐Internal ☒Public

DISCLAIMER

THIS DOCUMENT IS PROVIDED "AS IS". ROCKCHIP ELECTRONICS CO., LTD. ("ROCKCHIP") DOES NOT PROVIDE ANY WARRANTY OF ANY KIND, EXPRESSED, IMPLIED OR OTHERWISE, WITH RESPECT TO THE ACCURACY, RELIABILITY, COMPLETENESS, MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE OR NON-INFRINGEMENT OF ANY REPRESENTATION, INFORMATION AND CONTENT IN THIS DOCUMENT. THIS DOCUMENT IS FOR REFERENCE ONLY. THIS DOCUMENT MAY BE UPDATED OR CHANGED WITHOUT ANY NOTICE AT ANY TIME DUE TO THE UPGRADES OF THE PRODUCT OR ANY OTHER REASONS.

Trademark Statement

"Rockchip", "瑞芯微", "瑞芯" shall be Rockchip's registered trademarks and owned by Rockchip. All the other trademarks or registered trademarks mentioned in this document shall be owned by their respective owners.

All rights reserved. ©2021. Rockchip Electronics Co., Ltd.

Beyond the scope of fair use, neither any entity nor individual shall extract, copy, or distribute this document in any form in whole or in part without the written approval of Rockchip.

Rockchip Electronics Co., Ltd.

No.18 Building, A District, No.89, software Boulevard Fuzhou, Fujian, PRC

Website: www.rock-chips.com

Customer service Tel: +86-4007-700-590

Customer service Fax: +86-591-83951833

Customer service e-Mail: fae@rock-chips.com

Preface

Overview

This document presents the basic develop method of Rockchip DSP.

Product Version

Chipset	Kernel Version
RK2108	RT-Thread
PISCES	RT-Thread
RK2206	RKOS

Intended Audience

This document (this guide) is mainly intended for:

Technical support engineers

Software development engineers

Revision History

Date	Version	Author	Revision History
2019-06-24	V1.0.0	Huaping Liao	Initial version
2019-08-02	V1.1.0	Cody Xie	Add Floating License Server installation instructions
2019-09-03	V1.2.0	Huaping Liao	Add firmware packaging instructions
2019-10-10	V1.3.0	Huaping Liao	Add instructions for RKOS
2019-10-16	V1.4.0	Huaping Liao	Add installing instructions under Ubuntu environment
2020-03-10	V1.5.0	Huaping Liao	Add picture for configuration installing
2020-05-22	V1.6.0	Chris Zhong	Modify compilation tools source code path
2020-06-18	V1.7.0	Jair Wu	Update packaging tools instructions
2020-08-06	V1.8.0	Jair Wu	Add Vendor Key verification instructions
2020-09-18	V1.9.0	Jair Wu	Update method of Map Configure modification and XIP mode instructions
2021-02-03	V2.1.0	Jair Wu	Update software environment installation instructions, add static library usage instructions
2021-03-30	V2.2.0	Jair Wu	Add Cache related instructions

Contents

Rockchip RTOS DSP Developer Guide

1. Rockchip DSP Introduction
2. Build HiFi3 Software Environment
 - 2.1 Install Xplorer
 - 2.1.1 Windows
 - 2.1.2 Ubuntu
 - 2.2 Install License
 - 2.3 Import Project and Compile
 - 2.3.1 Import Project
 - 2.3.2 Compile Options
 - 2.3.3 Compile
 - 2.4 Generate DSP Firmware
 - 2.5 Firmware Packaging Configure
 - 2.6 Firmware Conversion Configuration
 - 2.7 Map Configuration
 - 2.7.1 Modify Map Configuration
 - 2.7.2 Modify Link Script
3. RT-THREAD
 - 3.1 Code Path
 - 3.2 Menuconfig
 - 3.3 Driver Interface
 - 3.4 Test Case
 - 3.5 Vendor Key Verification Test
4. RKOS
 - 4.1 Code Path
 - 4.2 Menuconfig
 - 4.3 Driver Interface
 - 4.4 Test Case
5. Static Library Create And Usage
 - 5.1 Create Project
 - 5.2 Build Library
 - 5.3 Import Library
 - 5.4 FAQ
6. XIP Mode
 - 6.1 Configuration
 - 6.2 Map Modify
 - 6.3 Firmware Packaging
 - 6.4 Firmware Flashing
7. Communication Protocol
 - 7.1 Communication Protocol Instruction
 - 7.2 Cache Instruction
 - 7.3 Enable Cache

1. Rockchip DSP Introduction

DSP(Digital Signal Processor) is used as a digital signal processor to convert analog signals into digital signals for high-speed real-time processing by dedicated processors. It has high-speed, flexible, programmable, low-power interface functions, and plays an increasingly important role in communication fields such as graphics and image processing, voice processing, and signal processing. The following is an introduction to Cadence® Tensilica® HiFi3 DSP.

- HiFi3 DSP is a ISA, support 2-way SIMD processing .
- HiFi3 DSP supports simultaneous processing of two 32x32 bits, 24x32 bits data, or four 24x24 bits, 16x32 bits, 16x16 bits data.
- HiFi3 DSP supports simultaneous processing of two IEEE-754 floating point data.

Currently, the description of the DSP integrated on Rockchip SoC is as follows:

- RK2108, RK2206 integrate HIFI3 DSP.

2. Build HiFi3 Software Environment

2.1 Install Xplorer

2.1.1 Windows

The full name of HiFi3 develop tool is "Xtensa Xplorer 8.0.8", customers have to contact Cadence to get the license. At present, the tool version we used is "xplorer-8.0.8-windows"- installer.exe , the configurations is "HiFi3Dev181203_win32.tgz", the configurations is base on "XtensaTools_RG_2018_9_win32.tgz". All these packages you can contact our developer to obtain.

First time to install Xplorer-8.0.8-windows-installer.exe, please follow the prompts to complete the installation.

After Xplorer is installed, you should install the Xtensa Tools. In the Xplorer, click "File"-->"New" --> "Xtensa Configuration", found this page and select "Install a new build from downloaded bundle as the new configuration", then click Next:

Create a Xtensa Configuration

Select how the new configuration will be created

- ☐ Create new configuration with a new core ISA (requires XPG access)
- ☐ Create new configuration by importing from file.
- ☐ Create new configuration based on an already installed build
- ☒ Install a new build from downloaded bundle as the new configuration
- ☐ Install a new build from XPG as the new configuration (requires XPG access)

Firstly, click "Manege Xtensa Tools", then click "Install from Distribution" in the new window, browse the path of "XtensaTools_RG_2018_9_win32.tgz", complete the installation by follow prompts.

New Xtensa Configuration

Install Xtensa Build
Add Xtensa Processor Builds to Your Workspace

Download Build from the Xtensa Processor Generator
XPG release: FindProcessors

Locate Previously Downloaded Build
 Browse...
Add Build

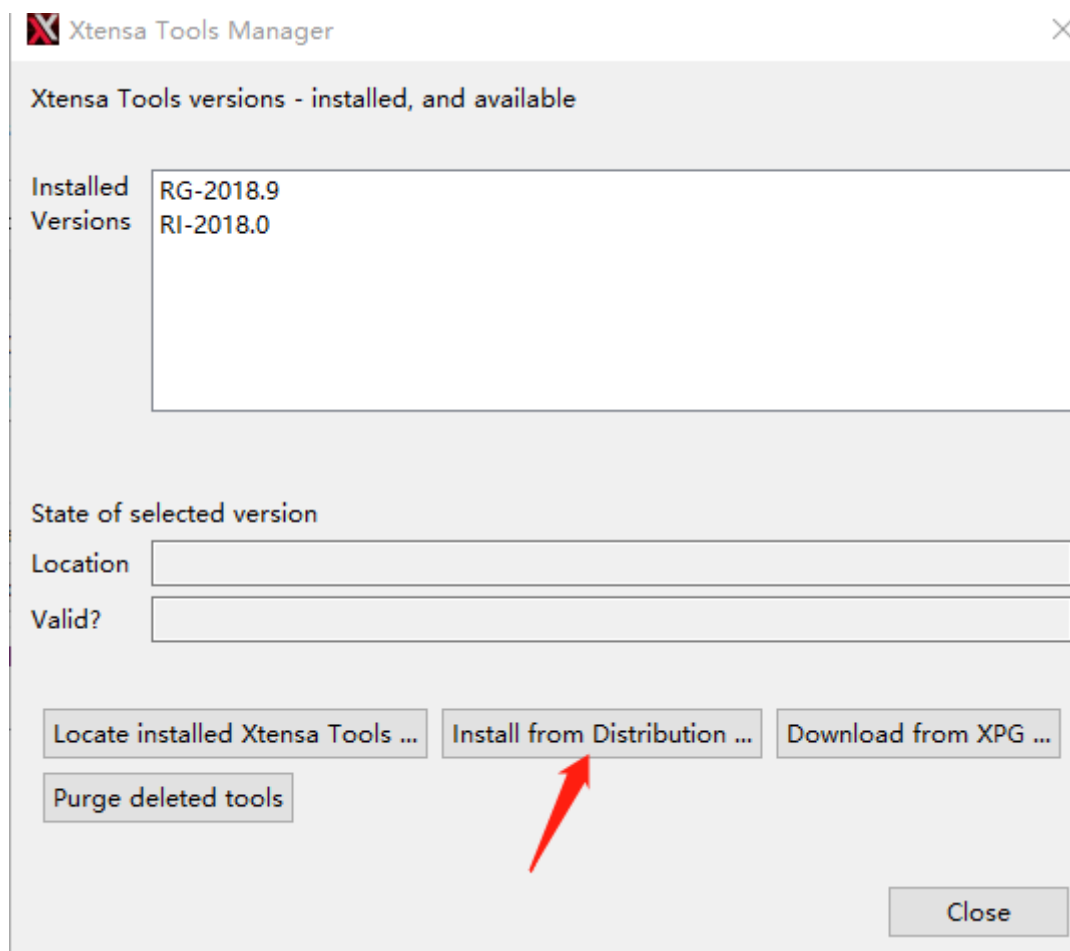
Xtensa Tools Manager
Manage Xtensa Tools

Add Processor Builds to Workspace

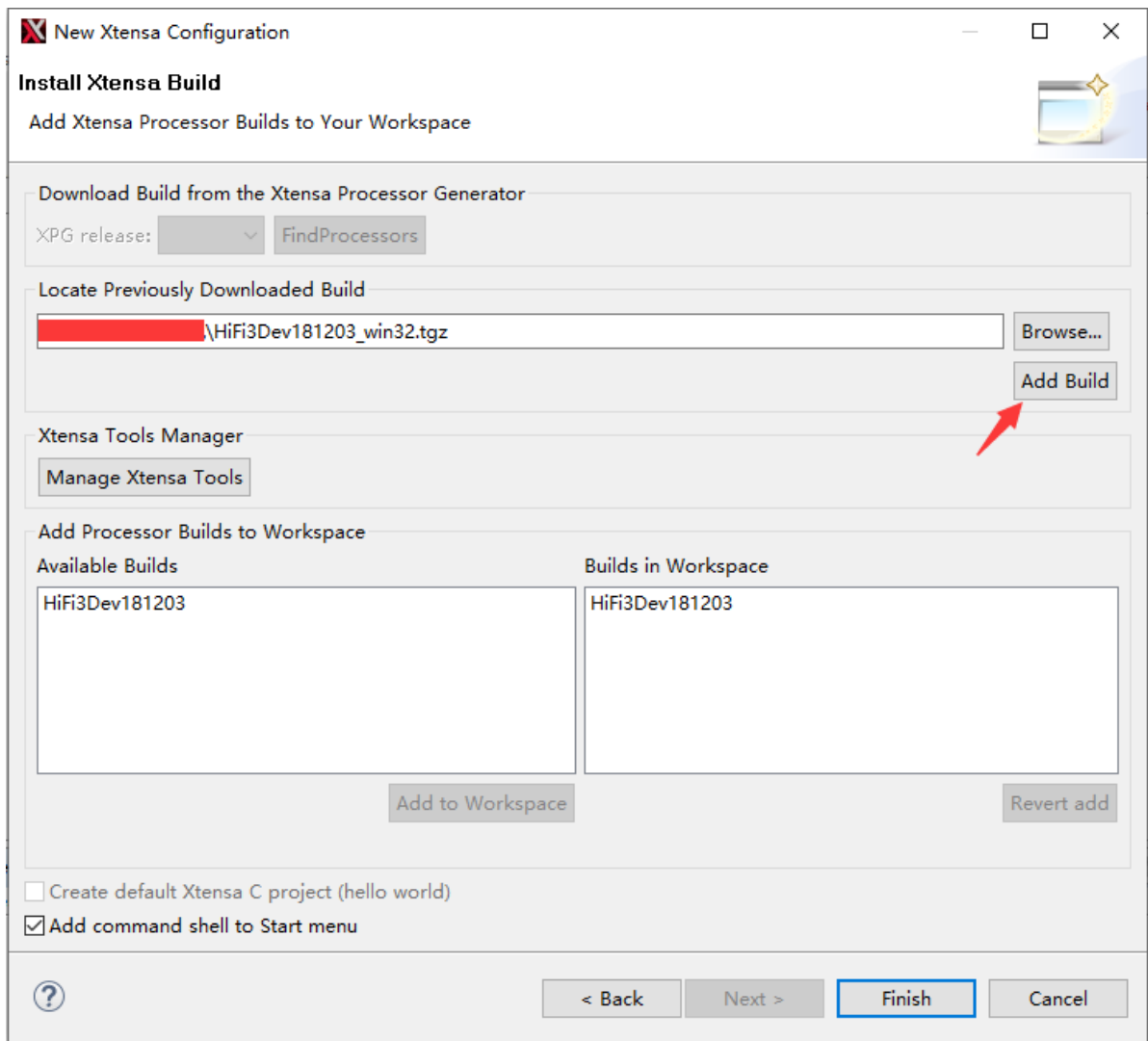
Available Builds	Builds in Workspace
<div></div>	<div></div>
Add to Workspace	Revert add

☐ Create default Xtensa C project (hello world)
☒ Add command shell to Start menu

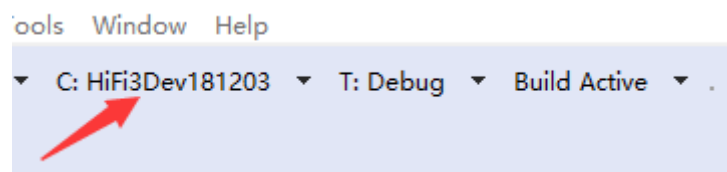
? < Back Next > Finish Cancel



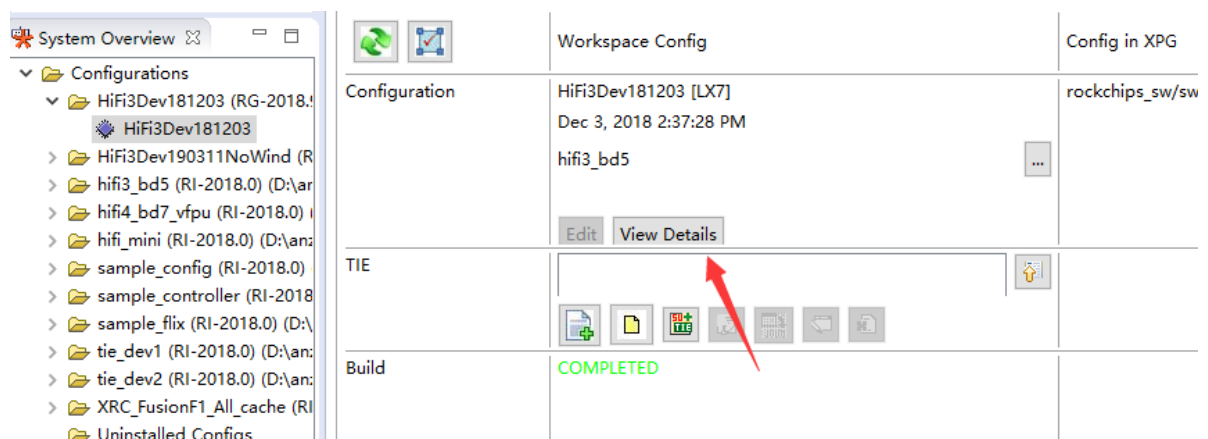
When finish the Xtensa Tools installation, back to this window, click Browse to select the path of "HiFi3Dev181203_win32.tgz", then click "Add Build" to add this configuration, after the configuration name is show out, click Finish.



After installation, you will find "C:(Active configuration)" in toolbar, click the triangle to display a drop-down box, select HiFi3Dev181203:



The System Overview in the lower left corner of the software will show the configuration file of HiFi3Dev181304, click on the file, and you will see the configuration information of the current Core. You can found the information of ITCM, DTCM, interrupt number, etc. The interrupt connected to the external INTC is Interrupt0.



2.1.2 Ubuntu

Since the development tool has unknown UI adaptation problems and usage problems in the Ubuntu environment, we recommend developing under windows as much as possible.

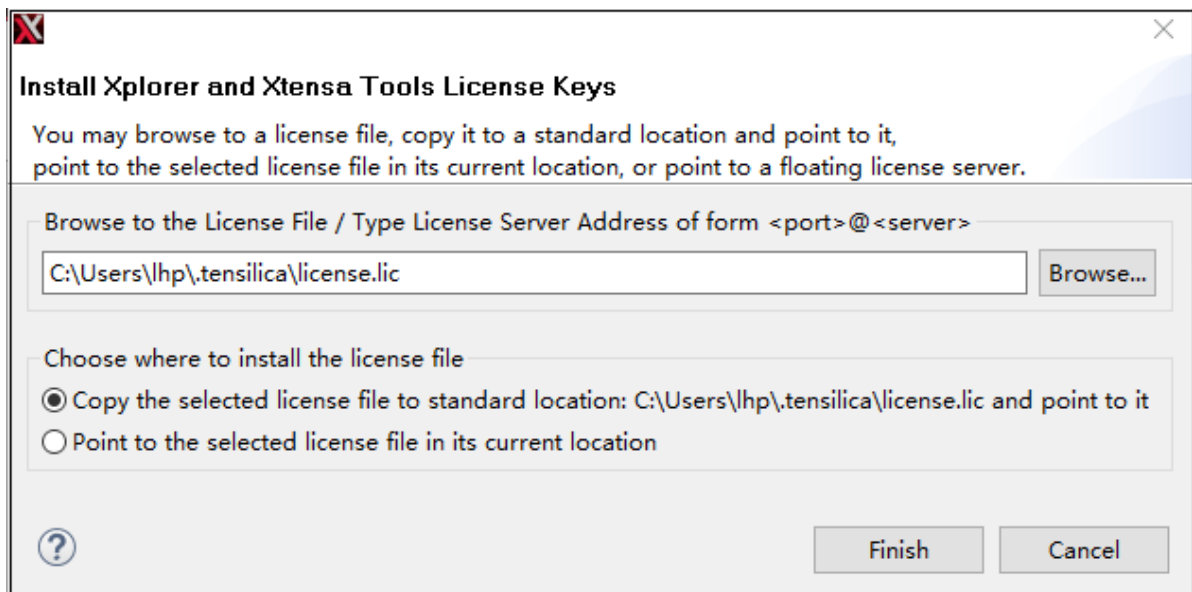
For Ubuntu 64bit version system, the recommended tool installation package is "Xplorer-8.0.8-linux-x64-installer.bin". For Ubuntu 32bit version system, the recommended tool installation package is "Xplorer-7.0.9-linux-installer". The configuration packages are "HiFi3Dev181203_linux.tgz" and "XtensaTools_RG_2018_9_linux.tgz". The installation process is the same as Windows.

Because the configuration package is 32bit, in order to be compatible with 64bit systems, you need to execute the following commands:

```
sudo apt-get install libgtk2.0-0:i386 gtk2-engines:i386 libc6:i386 libcanberra-gtk3-0:i386 libxtst6:i386 libncurses5:i386
sudo dpkg --add-architecture i386
sudo apt-get update
sudo apt-get install libc6:i386 libstdc++6:i386
sudo apt-get update -y
sudo apt-get update kernel* -y
sudo apt-get update kernel-headers kernel-devel -y
sudo apt-get install kernel-headers kernel-devel -y
sudo apt-get install compat-libstdc++-33.i686 -y
sudo apt-get install libstdc++.i686 -y
sudo apt-get install gtk2.i686 -y
sudo apt-get install libcanberra-gtk2.i686 -y
sudo apt-get install PackageKit-gtk3-module.i686 -y
sudo apt-get install libXtst.i686 -y
sudo apt-get install ncurses-libs.i686 -y
sudo apt-get install redhat-lsb.i686 -y
```

2.2 Install License

Startup Xplorer, click on "Help" --> "Xplorer License Keys", click "Install License Keys", input the path of license file, then click Finish. You can check license status by click "License Options" or "Check Xtensa Tools Keys". Note that the MAC address must be consistent with the host id in license file.



2.3 Import Project and Compile

2.3.1 Import Project

The projects path is hifi3/rkdsp/projects, store the configuration files and project files of different projects.

Click "File" --> "Import" --> "General" --> "Existing Projects into Workspace", click Browse, select the projects path, import the project code as needed. Different projects correspond to different project names. For example, the corresponding project name of RK2108 is RK2108, and the corresponding project name of RK2206 is RK2206.

After importing the project, you can click the triangle on the right at the "No Active Project" tab in the toolbar, and drop down to select the target project, such as P:RK2108.

2.3.2 Compile Options

Select the compiled Target in the toolbar, such as Debug, Release and ReleaseSize. Different Targets set different optimization levels and optimize the code to different degrees. The specific optimization content can be viewed and modified in the Target modification interface, or you can build your own Target as needed. Each configuration can be switched by clicking the triangle on the right side of "T:Debug" in the toolbar. Click Modify to enter the Target modification interface.

The commonly used tabs of the Target modification interface are:

Symbols: Add your macro definition.

Optimization: Compile options for optimization.

Linker: Select Linker support package, like linker script or simulator.

Libraries: Libraries manager.

Addl Linker: additional linker options.

You can check the options in the "All Options".

2.3.3 Compile

After select Project, Configuration, Target, click the "Build Active" to start compiling, the compiled file is stored under the bin directory.

2.4 Generate DSP Firmware

The executable file generated by the tool can only be used for tool simulation and cannot be run directly on the device. Run the CMD console, find the firmware generation script file "generate_dsp_fw.bat", enter the directory where the file is located, and execute the script. The usage is as follows:

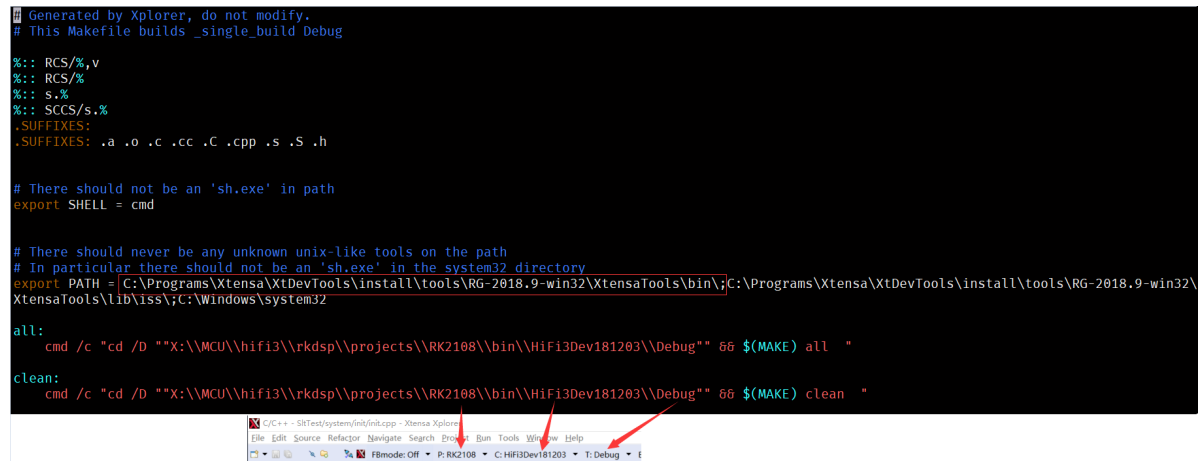
```
generate_dsp_fw.bat <project name> [config file]
```

<project name> is a required option, used to specify the project name, such as RK2108

[config file] is optional, used to specify the packaging configuration file, such as FwConfig.xml (the default is FwConfig.xml)

Example: generate_dsp_fw.bat RK2108 FwConfig.xml

Note: If the error "Cannot find the XXX file" occurs, please confirm whether the Project in the current Xplorer is selected correctly, the generate_dsp_fw.bat which is based on the Makefile in the project directory, is used to find the tool path, Target, Configuration, etc. automatically. If the currently selected project is incorrect, then the Makefile will not be generated. The correctly generated Makefile is shown in the figure below:



```
# Generated by Xplorer, do not modify.
# This Makefile builds _single_build Debug

%: RCS/%,v
%: RCS/%
%: S.%
%: SCCS/s.%
.SUFFIXES:
.SUFFIXES: .a .o .c .cc .C .cpp .s .S .h

# There should not be an 'sh.exe' in path
export SHELL = cmd

# There should never be any unknown unix-like tools on the path
# In particular there should not be an 'sh.exe' in the system32 directory
export PATH = C:\Programs\Xtensa\XtDevTools\install\tools\RG-2018.9-win32\XtensaTools\bin\;C:\Programs\Xtensa\XtDevTools\install\tools\RG-2018.9-win32\XtensaTools\lib\iss\;C:\Windows\system32

all:
cmd /c "cd /D ""X:\MCU\hifi3\rkdsp\projects\RK2108\bin\HiFi3Dev181203\Debug"" && $(MAKE) all "

clean:
cmd /c "cd /D ""X:\MCU\hifi3\rkdsp\projects\RK2108\bin\HiFi3Dev181203\Debug"" && $(MAKE) clean "
```

The generate_dsp_fw.bat will copy FwConfig.xml, Bin2Array.xml, firmware to the tool directory, and execute the program "HifiFirmwareGenerator.exe" to package firmware, the generated firmware is located in:

tools/HifiFirmwareGenerator/output/rkdsp.bin.

The source code of HifiFirmwareGenerator.exe is stored in:

<SDK>/components/hifi3/rkdsp/tools/source_code/HifiFirmwareGenerator

And the script will execute the program "FirmwareArrayGenerator.exe" to convert rkdsp.bin into the corresponding C array file according to the Bin2Array.xml configuration. The source code of FirmwareArrayGenerator.exe is stored in:

<SDK>/components/hifi3/rkdsp/tools/source_code/FirmwareArrayGenerator

For how to use different type firmware files, please refer to the description in section 3.2.

2.5 Firmware Packaging Configure

There is a FwConfig.xml in every project, this file use XML format to configure. HifiFirmwareGenerator.exe will parse the FwConfig.xml of the current project, here are the meanings of several key fields:

- CoreName: Core name like "HiFiDev181203", the script will replace the tag_corename by reading Makefile, or you can modify manually.
- ToolsPath: The path of Xplorer tools, the script will replace the tag_toolspath by reading Makefile, or you can modify manually.
- ExecutableFile: Name of source Firmware.
- SourceCodeMemStart: Start address of memory space for DSP.
- SourceCodeMemEnd: End address of memory space for DSP.

- DestinationCodeMemStart: Start address of memory space for MCU. Because there may be different memory space mapping situations. For example, for the same physical memory address TCM, the address accessed by DSP is 0x30000000, and the address accessed by MCU is 0x20400000, which correspond to SourceCodeMemStart and DetinationCodeMemStart respectively. If the address mapping is the same, then just fill in accordingly.
- Image: Output firmware.
- Image/Name: Firmware name, used to distinguish firmware types. MAIN is the main firmware, and the generated firmware is named rkdsp.bin. The generated firmware contains section header information (address, size, etc.), which is used for MCU analysis and loading DSP firmware; EXT is additional firmware, and the generated firmware is named ext_rkdsp.bin , without Section header information, arranged in the order of Section addresses, and packaged into MCU firmware when used, and burned to the designated location of Flash. When DSP is running, the corresponding data is directly read without MCU loading.
- Image/AddrRange: Specify the section address range belonging to the firmware. Taking XIP as an example, the address range is 0x60000000~0x60800000, and Sections in this range will be packaged into the specified firmware (generally EXT firmware). Attention, during the firmware generation process, all code segments will be parsed into the Section array, and the code segments will be distributed in the order of the Image in FwConfig.xml. If a certain code segment is within the scope of the Image, it will be packaged into the Image and remove from the Section array. If the current Image does not specify AddrRange, all Sections in the current Section array are packaged into the current Image by default. Therefore, the Image with the specified AddrRange must be written before the Image without the AddrRange, as follows:

```
<Image>
  <Id>2056</Id>
  <Name>EXT</Name>
  <Type>Permanent</Type>
  <AddrRange>0x60000000:0x60800000</AddrRange>
</Image>
<Image>
  <Id>2046</Id>
  <Name>MAIN</Name>
  <Type>Permanent</Type>
</Image>
```

If MAIN is written before EXT, when MAIN is packed, all Sections in the Section array will be packed into MAIN, then when EXT is packed, there is no valid Section. Or add the AddrRange field restriction in MAIN, you can ignore the sort order. AddrRange can be specified multiple times in segments, as follows:

```
<Image>
  <Id>2056</Id>
  <Name>EXT</Name>
  <Type>Permanent</Type>
  <AddrRange>0x60000000:0x60800000</AddrRange>
</Image>
<Image>
  <Id>2046</Id>
  <Name>MAIN</Name>
  <Type>Permanent</Type>
  <AddrRange>0x30000000:0x30010000</AddrRange>
  <AddrRange>0x30200000:0x30280000</AddrRange>
</Image>
```

2.6 Firmware Conversion Configuration

The Bin2Array.xml file is in the project directory to specify the conversion template. The fields in the file are described as follows:

- Type: C array type.
- Name: C array name.
- Input: Input binary file.
- Output: Output C file.

2.7 Map Configuration

Xplorer needs to allocate the space of each data segment according to the Map configuration information during the link process. In "T:(active build target)" --> "Modify", select Linker tab, you can see the Standard option, and you can select the default Map configuration, Xplorer provides developers with min-rt, sim and other configurations. These configuration file directories are stored in " \explor8\XtDevTools\install\builds\RG-2018.9-win32\HiFi3Dev181203\xtensa-elflib". Please refer to the document " \XtDevTools\downloads\RI-2018.0\docs\lsp_rm.pdf" for configuration information. Or you can select Custom LSP Path to use custom package.

2.7.1 Modify Map Configuration

Map configuration file is `memmap.xmm`, take `map\min-rt\memmap.xmm` as example, there are three sections named `sram`, `iram0` and `dram0`, the `iram0` and `dram0` is TCM of DSP, with faster access speed, `sram` is system memory, which is accessible by DSP, but with lower access speed than TCM. The `sram` and `iram0` is executable and writable, `dram0` is writable. So, we always put text segment to `iram0`, put data segment to `dram0`, if `iram0` or `dram0` is out of space, then put text or data segment to `sram`.

There are two methods to put the text segment to `sram`, the first one is to add `__attribute__((section(".sram.text")))` before the functions. The other is to move the default text segment to `sram`, modify as follows:

```
@@ -51,7 +51,7 @@
BEGIN sram
0x20000000: sysram : sram : 0x100000 : executable, writable ;
- sram0 : C : 0x200C0000 - 0x200fffff : .sram.rodata .sram.literal .sram.text
.sram.data .sram.bss;
+ sram0 : C : 0x200C0000 - 0x200fffff : .sram.rodata .sram.literal .sram.text
.literal .text .sram.data .sram.bss;
END sram

BEGIN iram0
@@ -61,7 +61,7 @@ BEGIN iram0
iram0_2 : F : 0x3000057c - 0x3000059b : .DebugExceptionVector.text
.KernelExceptionVector.literal;
iram0_3 : F : 0x3000059c - 0x300005bb : .KernelExceptionVector.text
.UserExceptionVector.literal;
iram0_4 : F : 0x300005bc - 0x300005db : .UserExceptionVector.text
.DoubleExceptionVector.literal;
- iram0_5 : F : 0x300005dc - 0x3000ffff : .DoubleExceptionVector.text
.iram0.literal .literal .iram0.text .text;
```

```
+ iram0_5 : F : 0x300005dc - 0x3000ffff : .DoubleExceptionVector.text
.iram0.literal .iram0.text;
END iram0
```

After modification, you need to enter the map directory, and use the tool to regenerate the link script according to the configuration file:

```
cd rkdsp\projects\RK2108\map
<install path>\XtDevTools\install\tools\RG-2018.9-win32\XtensaTools\bin\xt-
genldscripts.exe -b <map directory> --xtensa-core=HiFi3Dev181203
```

<install path> is the installation path of tools, such as C:\usr\XtDevTools\..., <map directory> is the target map directory, such as min-rt.

The following prompt will be displayed when successful generation:

```
New linker scripts generated in min-rt/ldscripts
```

If the generation fails, the corresponding error will be printed, please adjust according to the error prompt.

Note: Sram is system memory, if you use sram, you need to negotiate the allocation of sram space with the MCU-side developer to avoid errors caused by accessing the same memory at the same time. If the starting address is specified in sram0 as 0x200C0000, then the previous space of the address is available for CPU, the space after this address is available for DSP. The MCU side also needs to be set accordingly.

2.7.2 Modify Link Script

The way to modify the memmap.xmm file can only be modified in units of segments. In some cases, you need to assign a file to a certain segment space, and you can directly modify the link script to achieve:

```
--- a/rkdsp/projects/RK2108/map/min-rt/ldscripts/elf32xtensa.x
+++ b/rkdsp/projects/RK2108/map/min-rt/ldscripts/elf32xtensa.x
@@ -211,6 +211,8 @@ SECTIONS
{
    _iram0_text_start = ABSOLUTE(.);
    *(.iram0.literal .iram.literal .iram.text.literal .iram0.text .iram.text)
+   *file1.o(.literal .literal.* .text .text.*)
+   *file2.c.o(.literal .literal.* .text .text.*)
+   *libx.a:*.o(.literal .literal.* .text .text.*)

    . = ALIGN (4);
    _iram0_text_end = ABSOLUTE(.);
} >iram0_5_seg :iram0_5_phdr
```

Note: There are two type of object file, .o or .c.o. Normally, .c.o is compiled by Xplorer, and .o is a file in the standard library, you can see the .map file(like RK2108.map) to check whether it is .o or .c.o.

libx.a:.o(.literal .literal.* .text .text.*) assigns all objects file in the library to this section.

In addition, execute the commands in [Modify Map Configuration](#) to regenerate the link script will lose the manual partial modification, so if there is manual modification, please make a backup before executing the commands.

You can select the map in Xplorer->Target-> Linker tab, and then rebuild. If you select the "Generate linker map file" in Linker tab, the tool will generate `.map` file(like RK2108.map) in the bin directory, which is recorded the details of space allocation to check if these modification take effect.

3. RT-THREAD

3.1 Code Path

DSP Driver:

```
bsp/rockchip/common/drivers/drv_dsp.c
bsp/rockchip/common/drivers/drv_dsp.h
```

DSP Test Sample:

```
bsp/rockchip/common/drivers/drv_dsp.c
bsp/rockchip/common/drivers/drv_dsp.h
```

DSP driver call process:

```
bsp/rockchip/common/tests/dsp_test.c
```

3.2 Menuconfig

Open menuconfig, find DSP config by these path:

```
RT-Thread bsp drivers --->
  RT-Thread rockchip rk2108 drivers --->
    Enable DSP --->
      [*] Enable DSP
      [*] Enable firmware loader to dsp
          Dsp firmware path (Store firmware data in file) --->
            (/rkdsp.bin) Dsp firmware path
      [ ] Enable dsp send trace to cm4
      (-1) Config dsp debug uart port
```

"Enable firmware loader to dsp" means that when the DSP driver starts, it will download the DSP firmware.

"Dsp firmware path" has two options:

- "Store firmware data in file": Load firmware from filesystem, "Dsp firmware path" specify the file path.
- "Store firmware data in builtin": Compile the DSP firmware into MCU firmware, the compile tool will compile the `rkdsp_fw.c`¹ under `dsp_fw` directory Please refer to [Generate DSP Firmware](#) to generate `rkdsp_fw.c`. This way is easier than filesystem loading, but make sure the XIP is enabled(it is enabled by default), otherwise it will waste SRAM.

"Enable dsp send trace to cm4": Enable trace functions, DSP can send log to MCU by Mailbox.

"Config dsp debug uart port": The UART to print log, -1 means to use default UART(UART0).

3.3 Driver Interface

Please refer to "bsp/rockchip-common/tests/dsp_test.c" for driver calling.

```
struct rt_device *dsp_dev = rt_device_find("dsp0");
rt_device_open(dsp_dev, RT_DEVICE_OFLAG_RDWR);
rt_device_control(dsp_dev, RKDSP_CTL_QUEUE_WORK, work);
rt_device_control(dsp_dev, RKDSP_CTL_DEQUEUE_WORK, work);
rt_device_close(dsp_dev);
```

When calling `rt_device_open`, it will call the `rk_dsp_open` function in driver, which will startup DSP core, loading firmware and run DSP.

When calling `rt_device_control(dsp_dev, RKDSP_CTL_QUEUE_WORK, work)`, pass the work pointer, the driver will send this work to DSP by Mailbox, DSP parse the work and execute the corresponding algorithm, send the result back after work is done.

When calling `rt_device_control(dsp_dev, RKDSP_CTL_DEQUEUE_WORK, work)`, it can get the result of algorithm. If DSP is busy, this function will block until DSP work is done.

3.4 Test Case

Enable DSP TEST and AUDIO TEST:

```
RT-Thread bsp test case --->
  RT-Thread Common Test case --->
    [*] Enable BSP Common TEST
    [*]   Enable BSP Common AUDIO TEST
    [*]   Enable BSP Common DSP TEST
    [*]     Enable Dsp wakeup function
```

Compile and flash the firmware, after the shell boot up, input `dsp_vad_test`, you can see the log as follows:

```
msh />dsp_vad_test
dsp wakeup_test
Hmsh />ifl3: Hifi3 config done
Hifi3: kwsSetConfig ok
Hifi3: init uv_asr ok
ringbuf_addr:0x30260000, period_size:0x00000280
```

Input `audio_capture`, say "xiaoduxiaodu" to MIC, it will detect the wakeup words:

```
msh />audio_capture
audio_capture
vad buf: 0x30260000, size: 0x20000 bytes
vad periodsize: 0x280 kbytes
msh />Hifi3: xiaodu_wakeup-----xiaoduxiaodu-----
Hifi3: process return value = 1
work result:0x00000001
```

3.5 Vendor Key Verification Test

Take flash UUID as example, the basic process is as follows:

1. Use FlashKeyTool to read flash UUID
2. Use the flash UUID calculate a key for verification
3. Use FlashKeyTool write the key to vendor partition in flash
4. When testing, CPU read the key from vendor partition and send it to DSP
5. DSP read flash UUID, calculate a key and contrast with the key which CPU sent
6. DSP send the verification result to CPU
7. CPU and DSP do corresponding operation according to the verification result

Note: Steps 1~3 are done with PC in normal case, you can also use the relevant source code customization tool to complete the reading-calculating-flashing work in one tool.

On the side of CPU

Please refer to bsp/rockchip/common/tests/dsp_test.c for test sample .

Input dsp_vendor_test in shell to start the test, make sure that the verify test is supported in the DSP firmware .

On the side of DSP

The code path is rkdsp/application/RK2108/key_verify.cpp.

Define `DSP_VENDOR_VERIFY = 1` in Target, and dock your calculate algorithm to snor_key_verify functions in rkdsp/application/RK2108/key_verify.cpp. This function will input the key which is sent by CPU and return the result of verification.

4. RKOS

4.1 Code Path

DSP Driver:

```
src/driver/dsp/DspDevice.c
include/driver/DspDevice.h
```

DSP Test Sample:

```
src/subsys/shell/Shell_DspDevice.c
```

4.2 Menuconfig

Open menuconfig, find DSP config by these path:


```

BSP Driver --->
  Enable DSP --->
    [*] Enable DSP
    [*]   Enable firmware loader to dsp
           Dsp firmware path (Store firmware data in file) --->
    (/rkdsp.bin) Dsp firmware path
    [ ]   Enable dsp send trace to cm4
    (-1)  Config dsp debug uart port
    [ ]   Enable dsp jtag

```

Please refer to [Menuconfig](#) in [RT-THREAD](#) for instructions. There are two difference:

1. The directory to store rkdsp_fw.h is `include/dsp/dsp_fw/`.
2. "Enable dsp jtag": To enable DSP JTAG for debug.

4.3 Driver Interface

Please refer to "src/subsys/shell/Shell_DspDevice.c" for samples.

```

rkdev_create(DEV_CLASS_DSP, 0, NULL);
HDC dsp_dev = rkdev_open(DEV_CLASS_DSP, 0, NOT_CARE);
rk_dsp_open(dsp_dev, 0);
rk_dsp_control(dsp_dev, RKDSP_CTL_QUEUE_WORK, work);
rk_dsp_control(dsp_dev, RKDSP_CTL_DEQUEUE_WORK, work);
rk_dsp_close (dsp_dev);
rkdev_close(dsp_dev);
rkdev_delete(DEV_CLASS_DSP, 0, NULL);

```

Please refer to [Driver Interface](#) in [RT-THREAD](#) for instructions.

4.4 Test Case

Enable the DSP TEST:

```

Components Config --->
  Command shell --->
    [*] Enable DSP shell command

```

Compile and flash the firmware, after the shell boot up, input dsp_test, you will see the log as follows:

```

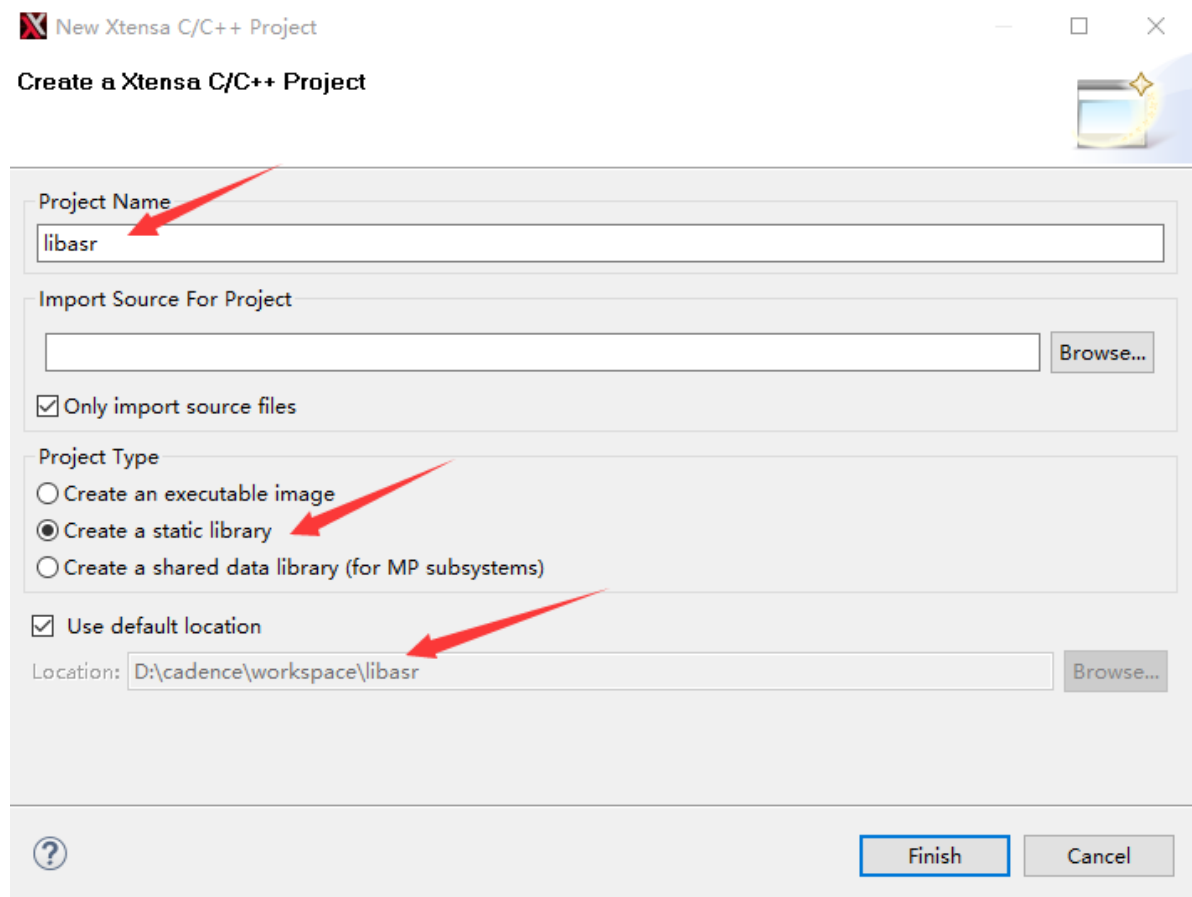
dsp_test
Hifi3: Hifi3 config done
Hifi3: kwsSetConfig ok
Hifi3: init uv_asr ok
config end
[A.DspTe][000024.61]workresult:0x00000000
[A.DspTe][000024.61]work result:0x00000000
[A.DspTe][000024.61]work result:0x00000000

```

5. Static Library Create And Usage

5.1 Create Project

Click File->New->Xtensa C/C++ Project, input the information of project. After click Finish, you can see libasr in Project Explorer.



You code can just be put into the directory of project, or right-click the project in Project Explorer, click File->Import->General->File System, Browse your code file or directory, and input into the folder. "Create links in workspace" mean create a soft link of file, without copying the file into the project. It makes it easy to maintain the code path structure. The project provided by our company also use this way.

File system

✖ Source directory is not valid or has not been specified.



From directory: Browse...

Filter Types... Select All Deselect All

Into folder: Browse...

Options

- ☒ Overwrite existing resources without warning
- ☒ Create top-level folder
- << Advanced
- ☐ Create links in workspace
- ☒ Create virtual folders
- ☒ Create link locations relative to: PROJECT_LOC

? < Back Next > Finish Cancel

5.2 Build Library

Please refer to [Compile Options](#), modify your compile options, then click Build Active to start compile. The libasr.a will be generated in project/bin/HiFi3Dev181203//. Target means Debug, Release, ReleaseSize etc. Now you can copy libasr and header file to applications project.

5.3 Import Library

Put the libasr.a into hifi3/rkdsp/library/libasr, and import to project by click File->Import->File System. Add "-lasr" to Addl linker tab in Target. Add library path "\${xt_project_loc}\\..\\..\\library\\libasr" to Librarys tab. And choose the right map in Linker tab.

Create new files libasr.cpp and libasr.h in libasr and import them to project, perform interface adaptation in these two files and call it in algorithm_handler.cpp.

5.4 FAQ

1. Cannot find library

Make sure the path added to Libraries tab is right, the library file is named right(must be libxxx.a, the xxx means the name of library, use -lxxx to link your library), and put it into the right path, import project successfully.

2. 'function' was not declared in this scope

Make sure the library import successfully. Declare functions with extern "C" like follows:

```
#ifndef __LIBXXX_H__
#define __LIBXXX_H__

#ifdef __cplusplus
extern "C" {
#endif

int libxxx_init(int arg);
int libxxx_deinit(int arg);

#ifdef __cplusplus
}
#endif

#endif // __LIBXXX_H__
```

Or rename your file to C file like rename libasr.cpp to libasr.c, then rebuild.

6. XIP Mode

If TCM and SRAM are out of space, you can put text and rodata segment to XIP only if MCU enable XIP(Enabled by default).

XIP access speed is significantly slower than TCM or SRAM, so it is recommended to put the hot code in TCM and SRAM

6.1 Configuration

XIP mode need call0 Software ABI, the configuration is `HiFi3Prod200605_Call0_win32.tgz`, please contact our developer to obtain. And enable `Target->Optimization->Enable long calls` in your compilation.

6.2 Map Modify

Add following configs to memmap.xmm, please refer to [Modify Map Configuration](#) regenerate the link script:

```

BEGIN srom
0x60000000: sysrom : srom : 0x800000 : executable ;
    srom0 : C : 0x607C0000 - 0x607fffff : .srom.info.rodata .srom.rodata
.srom.literal .srom.text .rom.store;
END srom

```

Note: This section can only store read-only segments like `.literal .text .rodata`.

6.3 Firmware Packaging

Packaging in FwConfigXIP.xml for XIP mode:

```
generate_dsp_fw.bat RK2108 FwConfigXIP.xml
```

After successful packaging, a LOG will remind you whether it is in XIP mode currently. If it does not match your expectations, please check whether the address range of the EXT section is the same with the address range in FwConfigXIP.xml.

Note: If there is no FwConfigXIP.xml under your project, please refer to [Firmware Packaging Configure](#) add EXT image to your FwConfig.xml or contact our developer to obtain.

6.4 Firmware Flashing

The generate script will output two files: `rkdsp_fw.c` and `ext_rkdsp.bin` in XIP mode, the `rkdsp_fw.c` usage is the same. The `ext_rkdsp.bin` needs to be flashed into the right offset of FLASH. Take the address in [Map Modif](#) as example, the start address of `srom0` is `0x607C0000`, the offset of start address `0x60000000` of SROM is `0x7C0000`, the block size of FLASH is `0x200`, so $0x7C0000 / 0x200 = 0x3E00$. The `ext_rkdsp.bin` should be flashed to `0x3E00`.

7. Communication Protocol

7.1 Communication Protocol Instruction

MCU communicates with DSP by Mailbox, the Mailbox has 4 channels, each channel can transfer 32bits CMD and 32bits Data. The CMD is command code, like `DSP_CMD_WORK`, `DSP_CMD_READY`, `DSP_CMD_CONFIG`, generally correspond to algorithm interface, The Data generally is work or config pointer.

When DSP startup, DSP will perform its own initialization and other operations. After the initialization is completed, the DSP will send the `DSP_CMD_READY` command. After receiving it, MCU will call the "rk_dsp_config" function send `DSP_CMD_CONFIG` to configure DSP for trace and others. The DSP receives the `DSP_CMD_CONFIG` and operate the configuration, `DSP_CMD_CONFIG_DONE` will be sent by DSP, indicating that the configuration has been completed and the algorithm can be worked. These three message transmissions are equivalent to a handshake process, and the algorithm can be called after the handshake is completed.

7.2 Cache Instruction

In the process of data interaction between MCU and DSP, because the two sides have independent Caches (16k icache and 16k dcache on DSP), there is a Cache coherency problem, that is, when data is written to an address, it will be written to the Cache first, the memory is only written under certain conditions (such as cache block replacement, explicit call to write back, or cache strategy is write-back), which will cause the two parties to obtain the data is not the latest. Therefore, in order to avoid this problems, it is usually recommended to explicitly call invalidate and writeback interfaces in your code. The interfaces of each platform are as follows:

```
// RK2108:
rt_hw_cpu_dcache_ops(RT_HW_CACHE_FLUSH | RT_HW_CACHE_INVALIDATE, (void *)addr,
size);
rt_hw_cpu_dcache_ops(RT_HW_CACHE_FLUSH, (void *)addr, size);
rt_hw_cpu_dcache_ops(RT_HW_CACHE_INVALIDATE, (void *)addr, size);
// RK2206:
rk_dcache_ops(RK_HW_CACHE_CLEAN | RK_HW_CACHE_INVALIDATE, (void *)addr, size);
rk_dcache_ops(RK_HW_CACHE_CLEAN, (void *)addr, size);
rk_dcache_ops(RK_HW_CACHE_INVALIDATE, (void *)addr, size);
// HIFI3 DSP:
xthal_dcache_region_writeback_inv((void *)addr, size);
xthal_dcache_region_writeback((void *)addr, size);
xthal_dcache_region_invalidate((void *)addr, size);
```

Generally, before the MCU sends message to DSP, for the output of MCU (which means input of DSP), need to writeback in the MCU side, invalidate in the DSP side, for the output of DSP (which means input of MCU), need to invalidate in MCU side². After DSP processing, DSP needs to call writeback for the output before sends message back. Take the interfaces of RK2108 platform as example:

```
char *buf_to_dsp;    // write by MCU, read by DSP
int buf_len_to_dsp;
char *buf_to_mcu;    // write by DSP, read by MCU
int buf_len_to_mcu;

/* MCU side start */
rt_hw_cpu_dcache_ops(RT_HW_CACHE_FLUSH, buf_to_dsp, buf_len_to_dsp);
rt_hw_cpu_dcache_ops(RT_HW_CACHE_INVALIDATE, buf_to_mcu, buf_len_to_mcu);
rt_device_control(dsp_dev, RKDSP_CTL_QUEUE_WORK, work); // Send message to DSP
/* MCU side end */

/* DSP side start */
xthal_dcache_region_invalidate(buf_to_dsp, buf_len_to_dsp);
/* processing */
xthal_dcache_region_writeback(buf_to_mcu, buf_len_to_mcu);
/* DSP side end */

/* MCU side start */
rt_device_control(dsp_dev, RKDSP_CTL_DEQUEUE_WORK, work); // Wait message from DSP
/* MCU side end */
```

7.3 Enable Cache

Generally, the ITCM, DTCM and SRAM are cacheable, the XIP, PSRAM are uncacheable, you can use this interface to make they cacheable:

```
// XCHAL_CA_WRITEBACK is only one of the strategies, you can check other  
strategies in then definition  
xthal_set_region_attribute((void *)addr_base, size, XCHAL_CA_WRITEBACK, 0);
```

Note: Cacheable does not mean performance improvement, it is depends on the hit-rate, if hit-rate is low, the performance may be worse. There is no hit-rate statistical tool in DSP, it is recommend to analyze the code and the simulation result for improvement.

1. This file is specified by RT_DSPFW_FILE_NAME in menuconfig. [↵](#)

2. It is recommended to call the invalidate before MCU sending a message, because MCU is a multi-threaded environment, and if buffer applied for at this time may be stored in the Cache because other threads have used it before, and probably it is "dirty" data. After DSP processing and writeback, if block replacement occurs in the Cache on the MCU side, which may cause dirty data to be written back, thereby contaminating the correct data. [↵](#)