

Rockchip RK2118 HiFi4 快速入门

文件标识: RK-JC-YF-594

发布版本: V1.1.0

日期: 2024-04-12

文件密级: 绝密 秘密 内部资料 公开

免责声明

本文档按“现状”提供，瑞芯微电子股份有限公司（“本公司”，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

版权所有 © 2024 瑞芯微电子股份有限公司

超越合理使用范畴，非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

介绍HiFi4工程导入、编译和打包

产品版本

芯片名称	内核版本
RK2118	RT-Thread 4.1.x

读者对象

本文档（本指南）主要适用于以下工程师：

HiFi4 DSP软件开发工程师

修订记录

版本号	作者	修改日期	修改说明
V1.0.0	tzb	2024-02-26	初始版本
V1.1.0	tzb	2024-04-12	添加内存布局配置说明

目录

Rockchip RK2118 HiFi4 快速入门

1. 开发环境搭建
 - 1.1 Xtensa开发工具安装
 - 1.2 Python安装
 - 1.2.1 Windows系统
 - 1.2.2 Linux系统
2. 内存空间
3. hifi4目录结构
4. 示例代码
5. 工程编译配置
 - 5.1 导入工程到Xplorer
 - 5.2 工程目录
 - 5.2.1 示例源码目录结构
 - 5.2.2 helloworld_demo工程示例源码目录结构
 - 5.2.3 选择项目
 - 5.2.4 选择DSP硬件配置
 - 5.2.5 选择编译目标
 - 5.2.6 配置HiFi4工程
 - 5.2.7 配置内存大小
 - 5.2.7.1 配置XIP内存布局
 - 5.2.7.2 配置SRAM内存布局
 - 5.2.7.3 配置DDR内存布局
 - 5.2.8 编译构建
 - 5.3 创建工程
 - 5.3.1 拷贝模板工程
 - 5.4 添加源码
 - 5.5 添加单个dsp私有的源码
 - 5.6 添加所有dsp共享的源码

1. 开发环境搭建

1.1 Xtensa开发工具安装

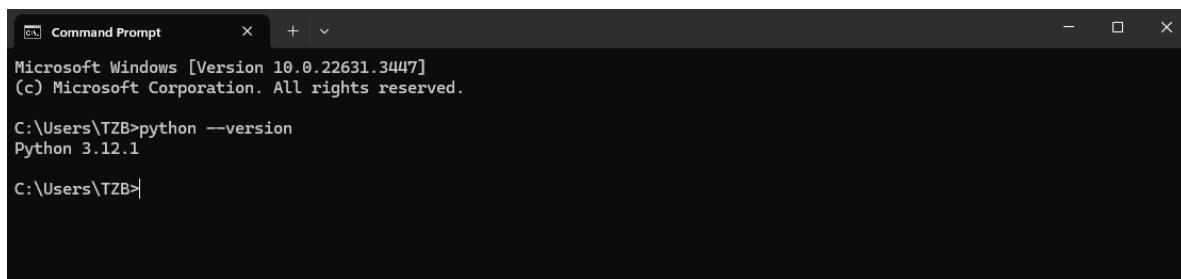
HIFI4的工程的开发编译使用的是Xplorer IDE，具体的安装说明请参考文档《Rockchip_Instruction_Xtensa_Development_Tools_Installation_CN》。

1.2 Python安装

HIFI4的工程编译有用到python的脚本，需要安装python（确保版本号大于3.8.0）。

1.2.1 Windows系统

1. 访问Python官方网站下载页面: [Python Downloads](#)
2. 选择适合Windows的Python版本并下载。
3. 运行下载的安装程序。在安装过程中，请确保选择了“Add Python 3.x to PATH”选项，以便将Python添加到系统环境变量中。
4. 完成安装后，打开命令提示符，输入 `python --version` 确认安装成功且版本正确。



```
Command Prompt
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

C:\Users\TZB>python --version
Python 3.12.1

C:\Users\TZB>
```

1.2.2 Linux系统

对于基于Debian的系统（如Ubuntu），使用以下命令安装Python:

```
sudo apt update
sudo apt install python3
```

安装完成后，使用`python3 --version`命令检查Python版本。

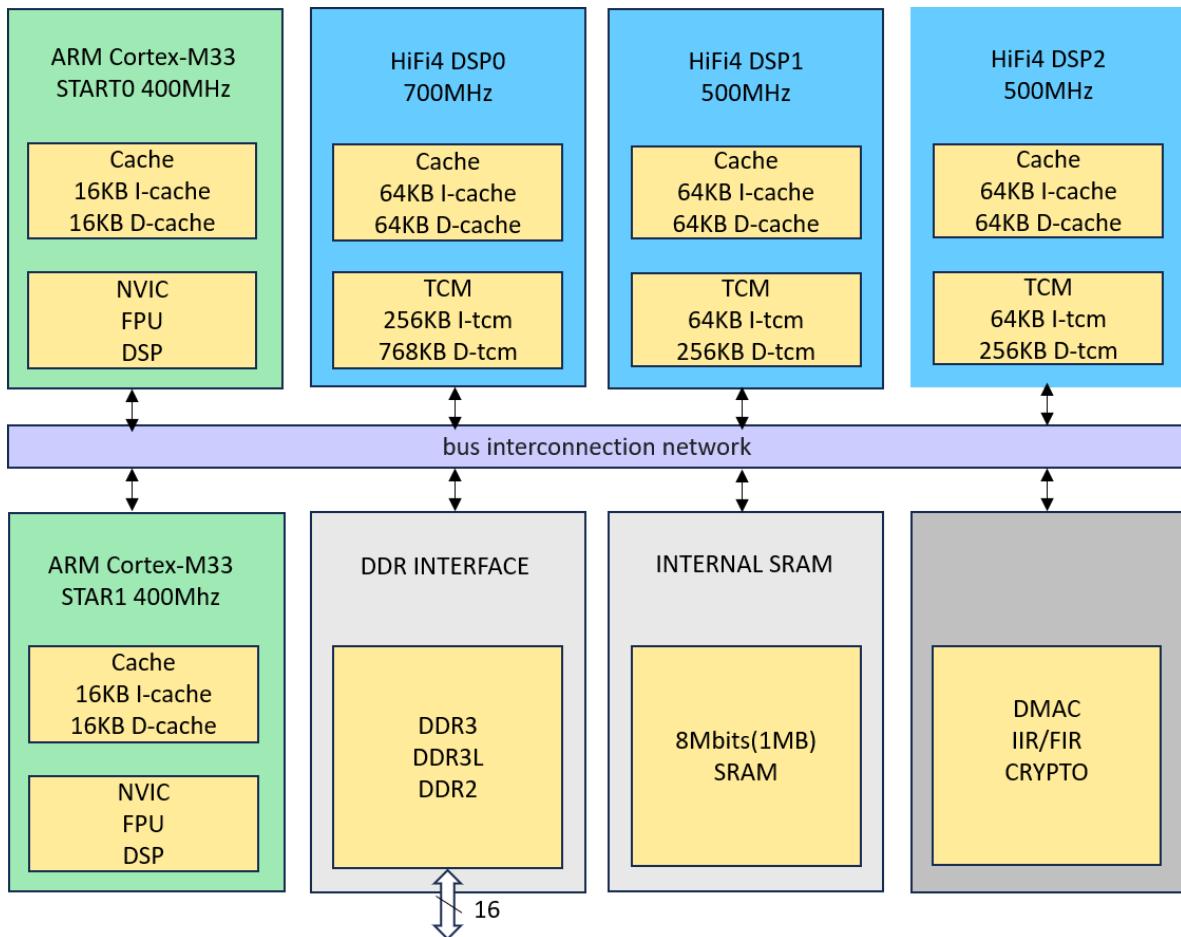
```
tzb@171server:~$ python3 --version
Python 3.8.10
```

2. 内存空间

RK2118使用的内存主要类型有Cache、TCM（包括ITCM和DTCM）、SRAM和DDR。每个CPU和DSP都有自己的Cache。在DSP中，每个核也有分配的TCM，其中ITCM专用于代码存储，DTCM专用于数据存储。SRAM的总容量为1MB，既可以存储代码也可以存储数据。

不同型号的RK2118对DDR支持不同。例如，RK2118M不支持DDR。可选配置包括合封的64MB DDR颗粒和外挂DDR颗粒。

内存空间分配如下所示：



3. hifi4目录结构

HiFi4的代码位于SDK的根目录下的 `components/hifi4`。以下是主要目录的详细说明：

<code>hifi4/</code>	
└── <code>dsp</code>	# DSP端代码
├── <code>core</code>	# 启动初始化，内存分配和地址转换
├── <code>drivers</code>	# 音频接口相关驱动
├── <code>asrc</code>	# ASRC驱动
├── <code>dma</code>	# DMA驱动
├── <code>facc</code>	# FACC驱动
├── <code>sai</code>	# SAI驱动
└── <code>spdifrx</code>	# SPDIFRX驱动
├── <code>rkstudio</code>	# RKStudio DSP端解析实现
└── <code>rockit</code>	# 多媒体框架
└── <code>rtt</code>	# RT-Thread CPU端运行代码
├── <code>codecs</code>	# Codecs驱动
└── <code>dsp_drp</code>	# DSP模块远程处理模块，负责clk/power/reset控制和固件下载

```

|   └── dsp_fw                         # DSP固件
|       ├── dsp0.bin                     # DSP0 bin文件, 可以烧入flash
|       ├── dsp1.bin                     # DSP1 bin文件, 可以烧入flash
|       ├── dsp2.bin                     # DSP2 bin文件, 可以烧入flash
|   └── rkstudio_parser                 # rkstudio mcu端解析实现
|   └── rockit                          # CPU端多媒体框架
└── samples                           # 示例代码
    └── rk2118                         # rk2118平台示例
        └── helloworld_demo             # helloworld_demo示例目录, 包含3个dsp的工程
            ├── helloworld_demo_dsp0    # dsp0 helloworld工程
            |   ├── inc                  # include目录
            |   |   └── hal_conf.h        # dsp0 配置文件
            |   └── src                  # source目录
            |       └── main.c           # main函数入口
            ├── helloworld_demo_dsp1    # dsp1 helloworld工程
            |   ├── inc                  # include目录
            |   |   └── hal_conf.h        # dsp1 配置文件
            |   └── src                  # source目录
            |       └── main.c           # main函数入口
            └── helloworld_demo_dsp2    # dsp2 helloworld工程
                ├── inc                  # include目录
                |   └── hal_conf.h        # dsp2 配置文件
                └── src                  # source目录
                    └── main.c           # main函数入口
            └── lsp
                ├── dsp0
                ├── dsp1
                └── dsp2
            └── mem_layout.h           # CPU和DSP内存配置文件, 包括XIP、SRAM和DDR的
内存布局配置
|   └── Kconfig                         # DSP HiFi4项目配置, 每个示例代码都要添加到这
边
└── shared                            # CPU/DSP 公共代码
    └── rk2118                         # rk2118平台
        ├── core                          # core id实现
        ├── ipc                           # 多核通信实现
        └── rkstudio                      # rkstudio cpu/dsp端公共代码
└── tools                             # 工具 (需要python 3.8以上版本)
    └── mkfw.py                        # dsp固件生成工具, elf2bin的转换工具
        ├── mkld.py                      # dsp工程Xplorer编译用到的链接脚本生成工具
        └── mkprojects.py                 # dsp工程创建工具

```

4. 示例代码

RK2118平台的示例代码位于 `samples/rk2118` 目录下, 涵盖多个用例, 具体包括:

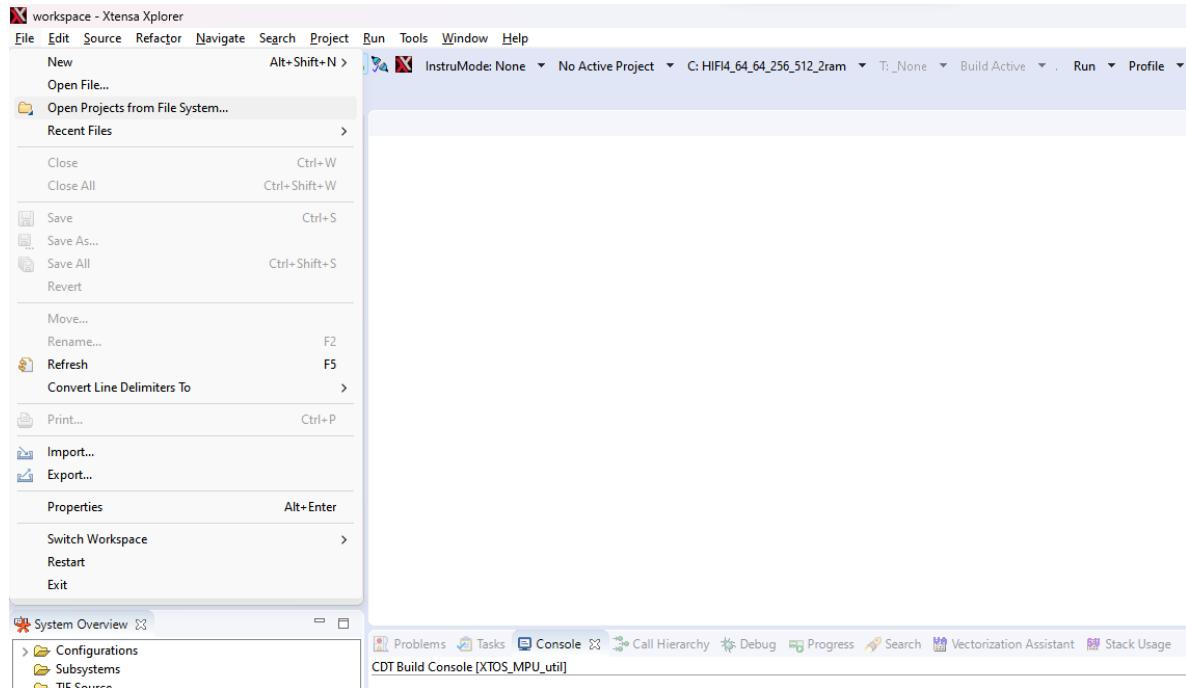
- `helloworld_demo` - 最基础的示例, 主要功能是在DSP上打印"HelloWorld", 用于验证基本的功能和DSP的启动。
- `adsp_demo` - 专为车载系统设计的示例, 包含车载相关的接口模块功能, 展示如何在车载环境中使用。
- `facc_demo` - 展示IIR和FIR滤波器的实现, 用于音频信号处理, 这个示例帮助开发者理解如何在DSP上使用FACC。

- `vocal_separate_demo` - 人声分离示例，从flash中读取wav文件，并处理以分离人声和背景音乐。
- `vocal_separate_spdifrx_demo` - 同样是人声分离示例，不过这个版本通过SPDIFRX音频接口读取实时音频数据，用于更实际的应用场景如实时音频处理。

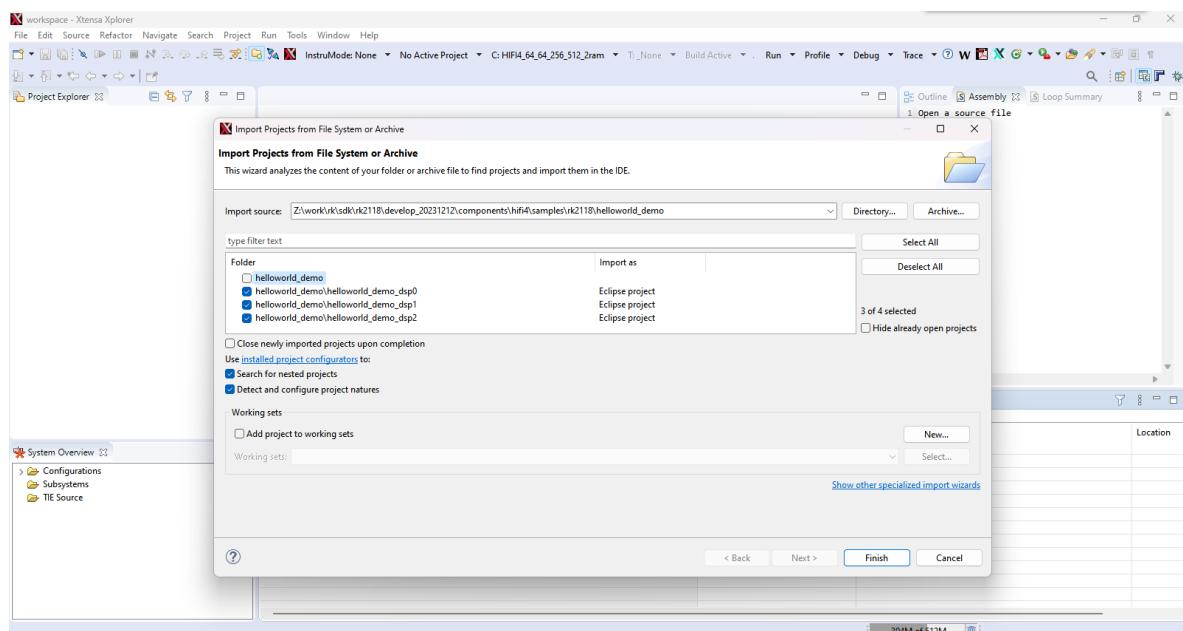
5. 工程编译配置

5.1 导入工程到Xplorer

打开Xtensa Xplorer，点击File->Import Projects from File System or Archive。

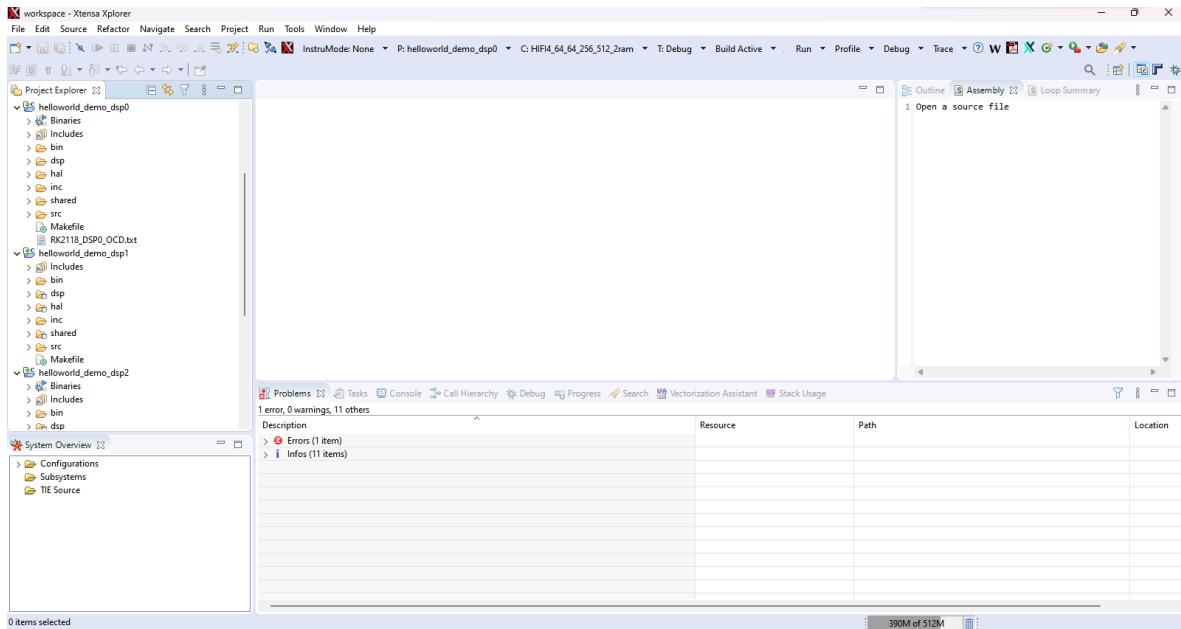


点击Directory，选择HiFi4目录下的helloworld_demo文件夹，可以看到Folder包含4个目录：



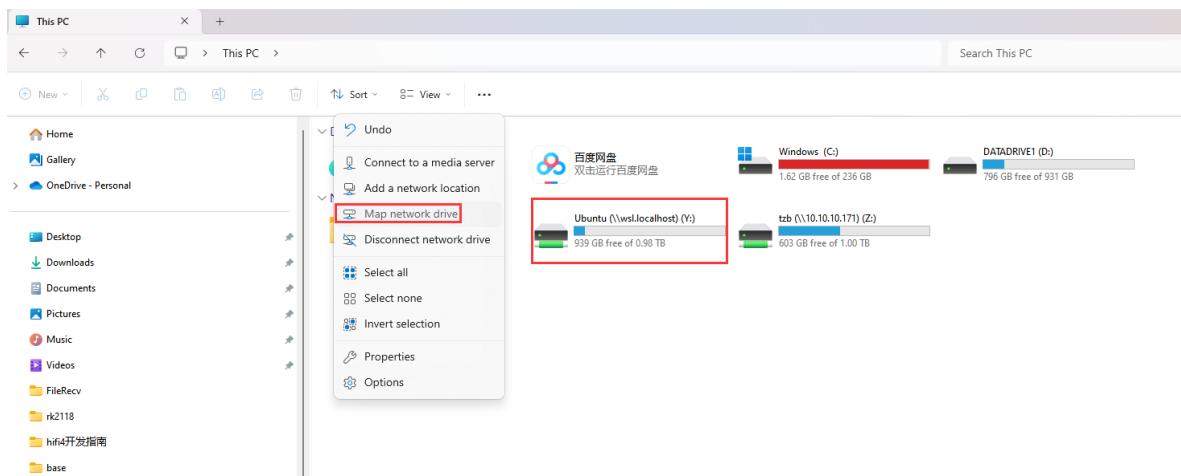
不要勾选helloworld_demo，其余3个目录分别对应同一个helloworld_demo工程下3个dsp的项目，点击Finish。

可以看到左侧Project Explorer窗口出现三个项目：



注意：使用wsl安装ubuntu作为开发环境的，需要把 \\wsl.localhost 映射到网络驱动器盘符。然后 explorer从这个驱动器导入工程。原因是explorer的pre build的钩子，是指定cmd来执行脚本，cmd在windows本身是不支持UNC路径（Universal Naming Convention）这种用于标识网络资源的路径格式。

例如映射 \\wsl.localhost 到Y盘，然后从Y盘开始导入工程：



5.2 工程目录

RK2118有3个dsp核，dsp0、dsp1和dsp2，一个完整的工程包含3个dsp项目。

例如helloworld_demo工程包括 helloworld_demo_DSP0、helloworld_demo_DSP1 和 helloworld_demo_DSP2 三个项目，分别对应3个dsp核。

5.2.1 示例源码目录结构

```

samples/rk2118/                               # rk2118示例代码
├── adsp_demo                                # adsp_demo示例
├── facc_demo                                 # facc_demo示例
├── helloworld_demo                          # helloworld_demo示例
├── Kconfig                                    # HiFi4 项目Kconfig
├── vocal_separate_demo                      # vocal_separate_demo示例
└── vocal_separate_spdifrx_demo             # vocal_separate_spdifrx_demo示例

```

rk2118示例源码目录结构中除了示例工程还包括一个 `Kconfig` 文件，这个 `Kconfig` 包含了目前选择的 HiFi4 项目配置：

```

menu "RK2118 HIFI4 Project"

choice
    prompt "HIFI4 Projects"
    default CONFIG_HIFI4_PROJECT_HELLOWORLD_DEMO

    config HIFI4_PROJECT_HELLOWORLD_DEMO
        bool "HelloWorld Demo"

    config HIFI4_PROJECT_ADSP_DEMO
        bool "ADSP Demo"

    config HIFI4_PROJECT_VOCAL_SEPARATE_DEMO
        bool "Vocal Separate Demo"

    config HIFI4_PROJECT_VOCAL_SEPARATE_SPDIFRX_DEMO
        bool "Vocal Separate SPDIFRX Demo"

    config HIFI4_PROJECT_FACC_DEMO
        bool "facc Demo"
endchoice

config HIFI4_PROJECT_MEM_LAYOUT_FILE
    string
    default "components/hifi4/samples/rk2118/helloworld_demo/mem_layout.h" if
HIFI4_PROJECT_HELLOWORLD_DEMO
    default "components/hifi4/samples/rk2118/adsp_demo/mem_layout.h" if
HIFI4_PROJECT_ADSP_DEMO
    default "components/hifi4/samples/rk2118/vocal_separate_demo/mem_layout.h"
if HIFI4_PROJECT_VOCAL_SEPARATE_DEMO
    default
"components/hifi4/samples/rk2118/vocal_separate_spdifrx_demo/mem_layout.h" if
HIFI4_PROJECT_VOCAL_SEPARATE_SPDIFRX_DEMO
    default "components/hifi4/samples/rk2118/facc_demo/mem_layout.h" if
HIFI4_PROJECT_FACC_DEMO
    help
        The selected hifi4 project to be built.
endmenu

```

CPU是运行RT-Thread系统，DSP是基于HAL开发的裸系统。对于共享的XIP Flash、SRAM和DDR先要预先配置内存大小，避免内存冲突。

内存相关的配置文件是 `mem_layout.h`，每个HiFi4的工程目录下都有一份 `mem_layout.h`。Xplorer会选择编译对应的工程目录下的 `mem_layout.h`。

RT-Thread通过 `Kconfig` 配置 `HIFI4_PROJECT_MEM_LAYOUT_FILE` 指定了 `mem_layout.h` 的路径。

注意：CPU配置的 `mem_layout.h` 路径必需和Xplorer配置的路径一致。如果不一致，Xplorer编译时会提示错误。

5.2.2 helloworld_demo工程示例源码目录结构

```
samples/rk2118/helloworld_demo/
├── helloworld_demo_dsp0
│   ├── inc
│   │   └── hal_conf.h
│   ├── .project
│   └── .settings
    格式
    ├── targets
    │   └── xtensa
    │       ├── CommonTarget.bts
    │       ├── Debug.bts
    │       ├── Release.bts
    │       └── ReleaseSize.bts
    ├── src
    │   └── main.c
    └── .xxproject
├── helloworld_demo_dsp1
│   ├── inc
│   │   └── hal_conf.h
│   ├── .project
│   ├── .settings
│   └── targets
    └── xtensa
        ├── CommonTarget.bts
        ├── Debug.bts
        ├── Release.bts
        └── ReleaseSize.bts
    ├── src
    │   └── main.c
    └── .xxproject
└── helloworld_demo_dsp2
    ├── inc
    │   └── hal_conf.h
    ├── .project
    ├── .settings
    └── targets
        └── xtensa
            ├── CommonTarget.bts
            ├── Debug.bts
            ├── Release.bts
            └── ReleaseSize.bts
    ├── src
    │   └── main.c
    └── .xxproject
└── lsp
    # 链接支持包，存放该工程3个dsp项目对应的链接
    相关的文件
    ├── dsp0
    │   ├── memmap.xmm.template
    │   └── specs
    基础库
```

helloworld_demo工程
helloworld_demo工程dsp0项目
dsp0项目inc目录
dsp0项目hal配置文件
dsp0项目xplorer项目文件
dsp0项目xplorer项目设置，.bts文件是xml

dsp0项目编译目标配置目录

公共的编译参数
Debug对应的编译参数
Release对应的编译参数
ReleaseSize对应的编译参数

dsp0项目src目录
dsp0主函数入口
dsp0项目xplorer项目文件
helloworld_demo工程dsp1项目

helloworld_demo工程dsp2项目

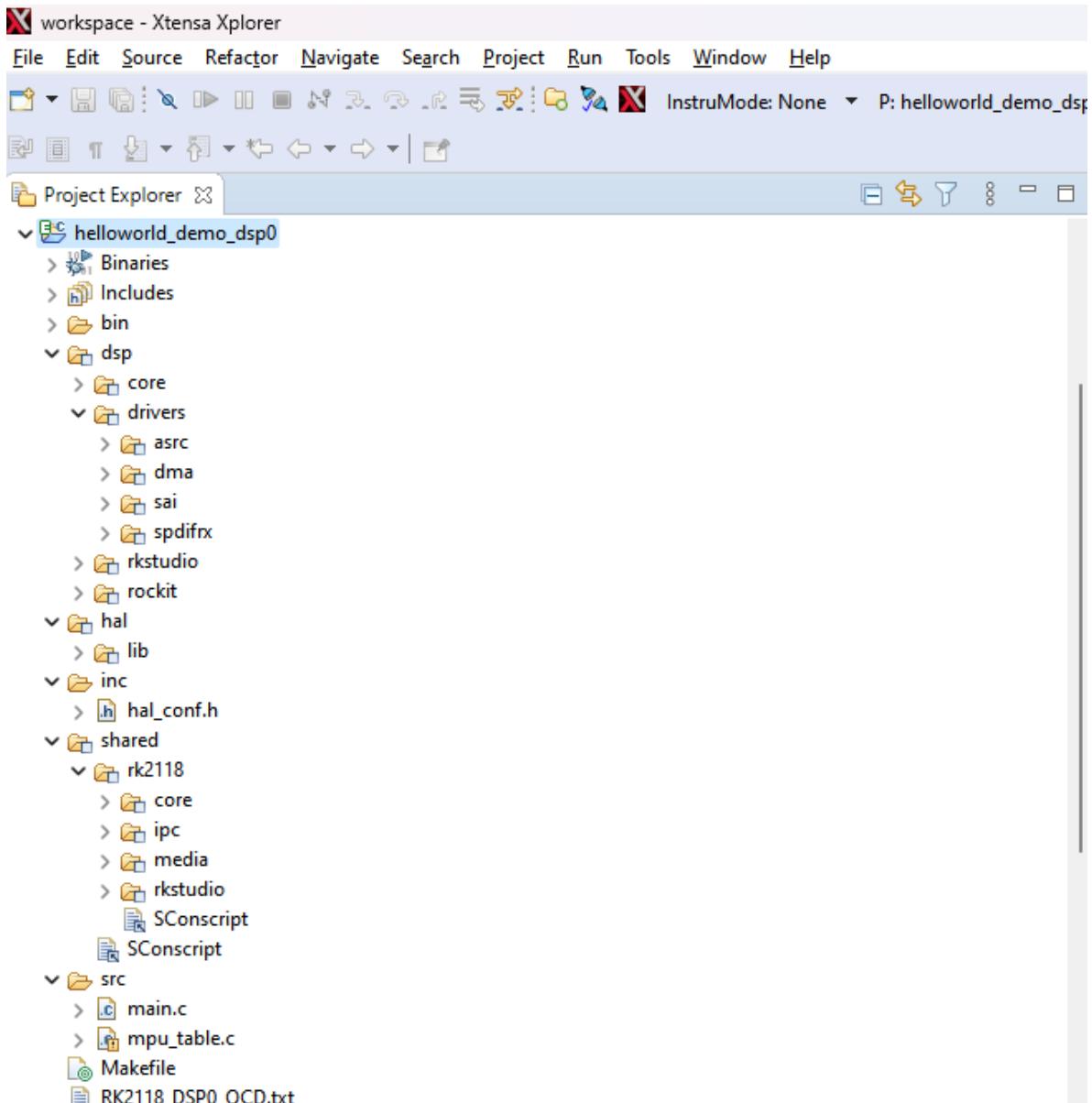
链接支持包，存放该工程3个dsp项目对应的链接
dsp0项目对应的链接脚本目录
dsp0项目的内存映射模板文件
dsp0项目使用的specs，配置链接的启动代码和

```

|   └── dsp1                                # dsp1项目对应的链接脚本目录
|       └── memmap.xmm.template
|
|   └── specs
|
└── dsp2                                    # dsp2项目对应的链接脚本目录
    └── memmap.xmm.template
    └── specs
|
└── mem_layout.h                            # helloworld_demo工程对应的mem_layout.h

```

从Xploerer看到的helloworld_demo_dsp0项目结构:



可以看到除了src和inc外，还导入了hal、dsp、share和mpu_table.c，这些文件夹和子文件夹都带有一个空心的矩形，空心的矩形表示是导入的源码是一个外部的链接，源码不在工程项目路径中。

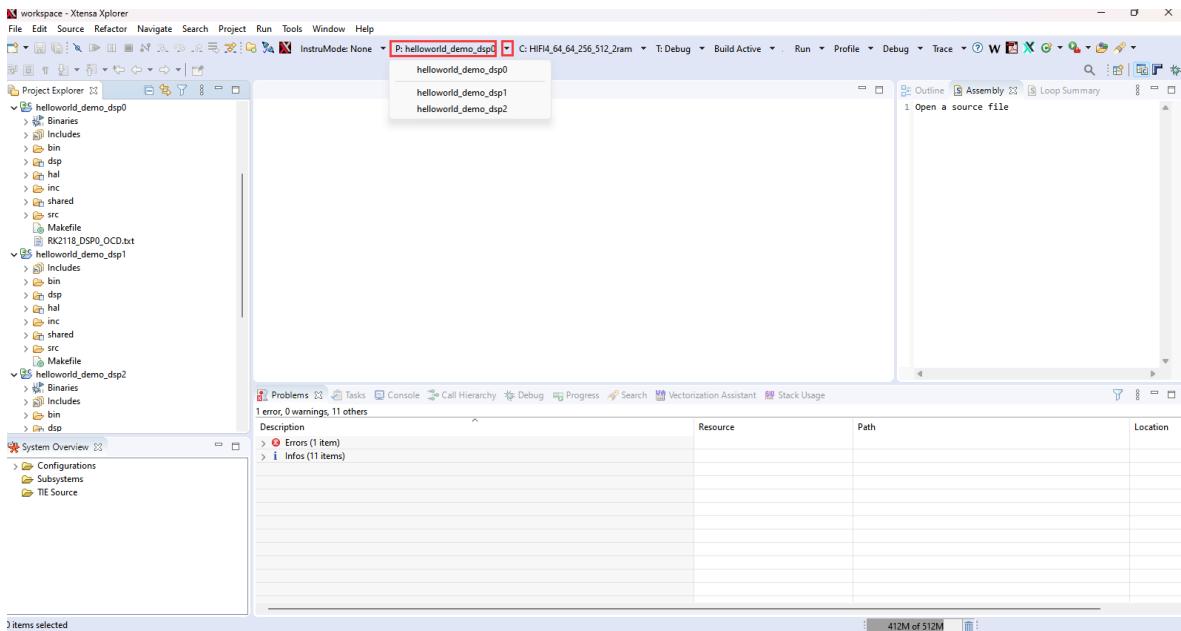
这些导入的外部的链接信息存在了项目路径下的.project文件。其中dsp和share对应的源码目录在hifi4目录结构介绍有提到，hal对应的源码目录是sdk下的bsp/rockchip/common/hal。

mpu_table.c（配置项目的mpu属性，内存区域的读写权限和cache情况）有一个黄颜色矩形的感叹号，是因为这个文件还不存在，在编译的时候才会自动生成。

5.2.3 选择项目

在顶部工具栏选择需要编译的项目 P:helloworld_demo_dsp0，可以通过倒三角切换不同的项目。

选择: helloworld_demo_dsp0

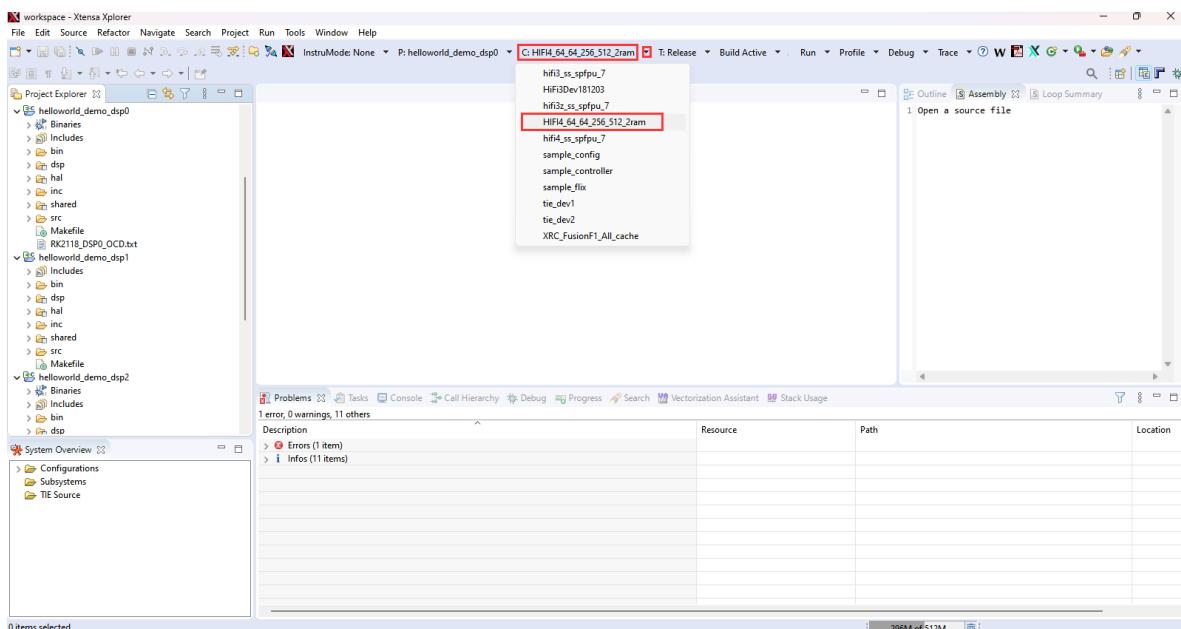


注意: 这边选择的项目, 编译时Xplorer会使用对应工程目录下的 mem_layout.h。例如选择 P:helloworld_demo_dsp0 , 使用的是 helloworld_demo/mem_layout.h 。

5.2.4 选择DSP硬件配置

在顶部工具栏选择DSP的硬件配置 C:HIFI4_64_64_256_512_2ram , 可以通过倒三角切换不同的配置。

注意: DSP的硬件配置已经硬化到SoC, 只能是HIFI4_64_64_256_512_2ram。



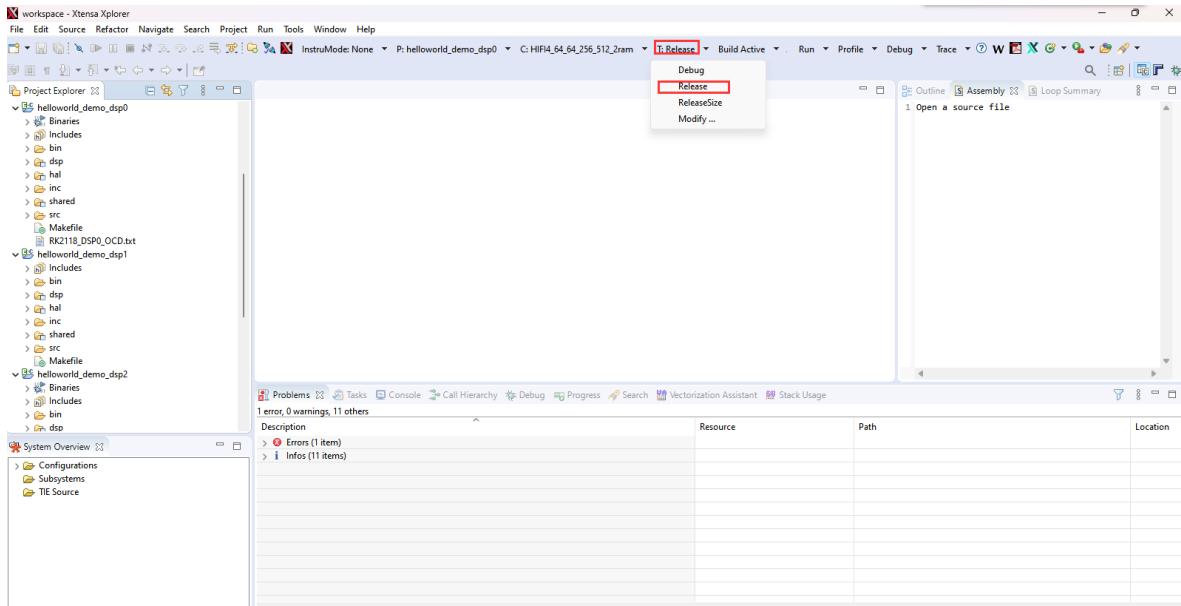
5.2.5 选择编译目标

在顶部工具栏选择编译目标 T:Release , 可以通过倒三角切换不同的目标。

编译目标可以选择 Debug 、 Release 和 ReleaseSize 。

`Debug` 一般用于调试，开启了HiFi4指令，不开启编译优化；`Release` 在 `Debug` 基础上开启了O2优化。`ReleaseSize` 在 `Release` 的基础上进一步优化空间。

建议开发阶段使用 `Release`，碰到需要单步跟踪指令分析异常的时候，可以选择 `Debug`。空间不足可以尝试 `ReleaseSize`。选择： `Release`。



5.2.6 配置HiFi4工程

CPU和DSP内存配置是通过mem_layout.h来配置，现在Xplorer已经选择了helloworld_demo工程，Xplorer会使用helloworld_demo工程下的mem_layout.h文件。

2个CPU是运行RT-Thread系统，同样需要指定同一份mem_layout.h文件，在bsp/rockchip/rk2118目录下执行命令：

```
scons --menuconfig
```

选择helloworld_demo：



退出保存后在.config有如下配置：

```

#
# RK2118 HIFI4 Project
#
CONFIG_HIFI4_PROJECT_HELLOWORLD_DEMO=y
# CONFIG_HIFI4_PROJECT_ADSP_DEMO is not set
# CONFIG_HIFI4_PROJECT_VOCAL_SEPARATE_DEMO is not set
# CONFIG_HIFI4_PROJECT_VOCAL_SEPARATE_SPDIFRX_DEMO is not set
# CONFIG_HIFI4_PROJECT_FACC_DEMO is not set
CONFIG_HIFI4_PROJECT_MEM_LAYOUT_FILE="components/hifi4/samples/rk2118/helloworld
_demo/mem_layout.h"

```

5.2.7 配置内存大小

TCM和Cache是独享，不需要配置。XIP、SRAM和DDR内存是共享的，需要静态划分区域大小，通过mem_layout.h文件配置。

打开helloworld_demo工程下的mem_layout.h文件，只能修改size对应的宏，内存前后布局不允许修改。

注意：如果大小不为0，允许的最小粒度是1KB，1KB以下粒度不支持

5.2.7.1 配置XIP内存布局

如果有XIP，例如nor flash，XIP默认配置如下：

```

/* ----- */
/* XIP Memory Layout */
/* ----- */

XIP_BASE      =      0x11000000;

/* Sizes - User-modifiable */
XIP_RKPARTITIONTABLE_SIZE  =  0x00010000; /* 64 KB */
XIP_IDBLOCK_SIZE          =  0x00020000; /* 128 KB */
XIP_CPU0_TFM_SIZE         =  0x00020000; /* 128 KB */
XIP_CPU1_LOADER_SIZE      =  0x00010000; /* 64 KB */
XIP_CPU0_RTT_SIZE         =  0x00040000; /* 256 KB */
XIP_DSP0_FIRMWARE_SIZE    =  0x00100000; /* 1 MB */
XIP_DSP1_FIRMWARE_SIZE    =  0x00100000; /* 1 MB */
XIP_DSP2_FIRMWARE_SIZE    =  0x00100000; /* 1 MB */
XIP_CPU1_RTT_SIZE         =  0x00040000; /* 256 KB */
XIP_USER_DATA_SIZE        =  0x00020000; /* 128 KB */

```

- XIP_RKPARTITIONTABLE_SIZE - 分区表大小，不允许修改
- XIP_IDBLOCK_SIZE - ID Block的大小，不允许修改
- XIP_CPU0_TFM_SIZE - CPU0 TF-M安全固件的大小，不允许修改
- XIP_CPU1_LOADER_SIZE - CPU1 loader固件大小，不允许修改
- XIP_CPU0_RTT_SIZE - CPU0 RT-Thread固件大小，可以修改
- XIP_DSP0_FIRMWARE_SIZE - DSP0固件大小，可以修改
- XIP_DSP1_FIRMWARE_SIZE - DSP1固件大小，可以修改
- XIP_DSP2_FIRMWARE_SIZE - DSP2固件大小，可以修改
- XIP_CPU1_RTT_SIZE - CPU1 RT-Thread固件大小，可以修改
- XIP_USER_DATA_SIZE - 剩余空闲大小，可以修改

注意：如果使用XIP Flash，在固件打包阶段，修改完XIP的内存布局，需要修改对应的setting.ini分区大小和mem_layout.h保持一致。

5.2.7.2 配置SRAM内存布局

1MB SRAM默认配置如下：

```
/*
 *-----*
 * SRAM Memory Layout
 *-----*
 */

SRAM_BASE          = 0x30200000;

/* User-modifiable sizes */
SRAM_CPU0_TFM_SIZE    = 0x00010000; /* 64 KB */
SRAM_CPU0_RTT_SIZE    = 0x00010000; /* 64 KB */
SRAM_DSP0_SIZE        = 0x00050000; /* 320 KB */
SRAM_DSP1_SIZE        = 0x00048000; /* 288 KB */
SRAM_DSP2_SIZE        = 0x00047000; /* 284 KB */
SRAM_SPI2APB_SIZE     = 0x00001000; /* 4 KB */
```

- `SRAM_CPU0_TFM_SIZE` - CPU0 TF-M固件的SRAM大小，不允许修改
- `SRAM_CPU0_RTT_SIZE` - CPU0 RT-Thread的SRAM大小，可以修改
- `SRAM_DSP0_SIZE` - DSP0的SRAM大小，可以修改
- `SRAM_DSP1_SIZE` - DSP1的SRAM大小，可以修改
- `SRAM_DSP2_SIZE` - DSP2的SRAM大小，可以修改
- `SRAM_SPI2APB_SIZE` - SPI2APB的SRAM大小，不允许修改

5.2.7.3 配置DDR内存布局

如果有DDR，需要配置DDR内存，64MB的DDR默认配置如下：

```
/*
 *-----*
 * DRAM Memory Layout
 *-----*
 */

DRAM_BASE          = 0xA0000000;

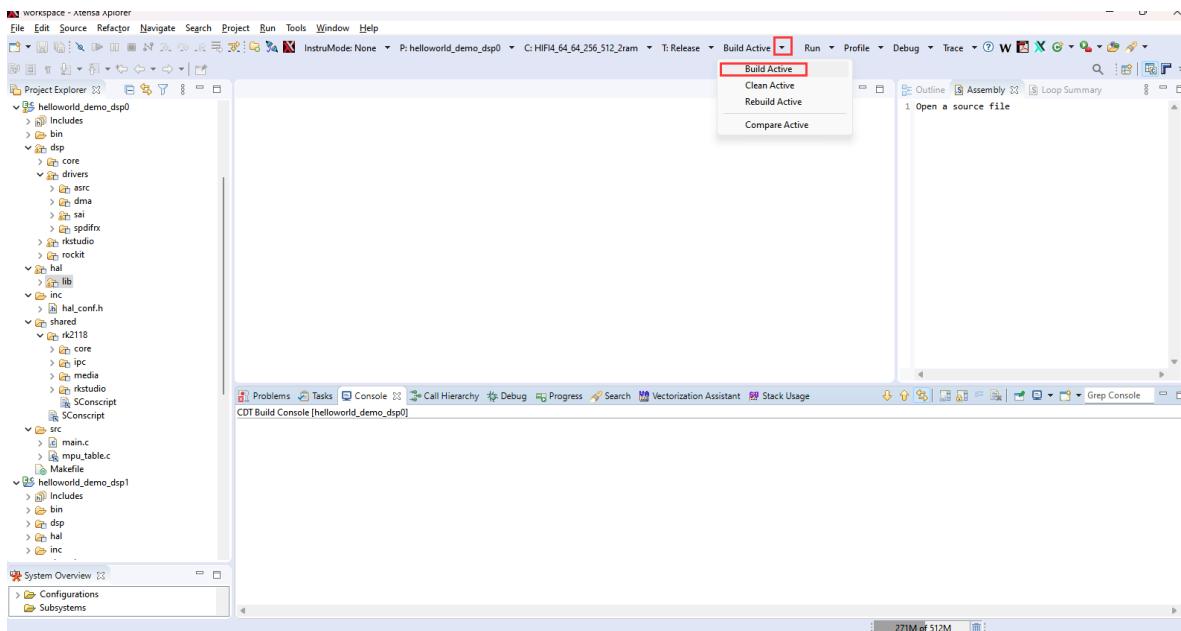
/* User-modifiable sizes */
DRAM_CPU0_TFM_SIZE    = 0x00100000; /* 1 MB */
DRAM_CPU0_RTT_SIZE    = 0x00100000; /* 1 MB */
DRAM_NPU_SIZE        = 0x00400000; /* 4 MB */
DRAM_DSP0_SIZE        = 0x01300000; /* 19 MB */
DRAM_DSP1_SIZE        = 0x01300000; /* 19 MB */
DRAM_DSP2_SIZE        = 0x01300000; /* 19 MB */
DRAM_CPU1_LOADER_SIZE = 0x00008000; /* 32 KB */
DRAM_CPU1_RTT_SIZE    = 0x000F8000; /* 992 KB */
```

- `DRAM_CPU0_TFM_SIZE` - CPU0 TF-M固件的DDR大小，不允许修改
- `DRAM_CPU0_RTT_SIZE` - CPU0 RT-Thread的DDR大小，可以修改
- `DRAM_NPU_SIZE` - NPU的DDR大小，可以修改
- `DRAM_DSP0_SIZE` - DSP0的DDR大小，可以修改
- `DRAM_DSP1_SIZE` - DSP1的DDR大小，可以修改
- `DRAM_DSP2_SIZE` - DSP2的DDR大小，可以修改

注意：默认DSP DDR相关的内存分配功能给关闭了，如果确认有DDR，需要在DSP对应项目中的 hal_config.h 配置 HAVE_DDR 宏。

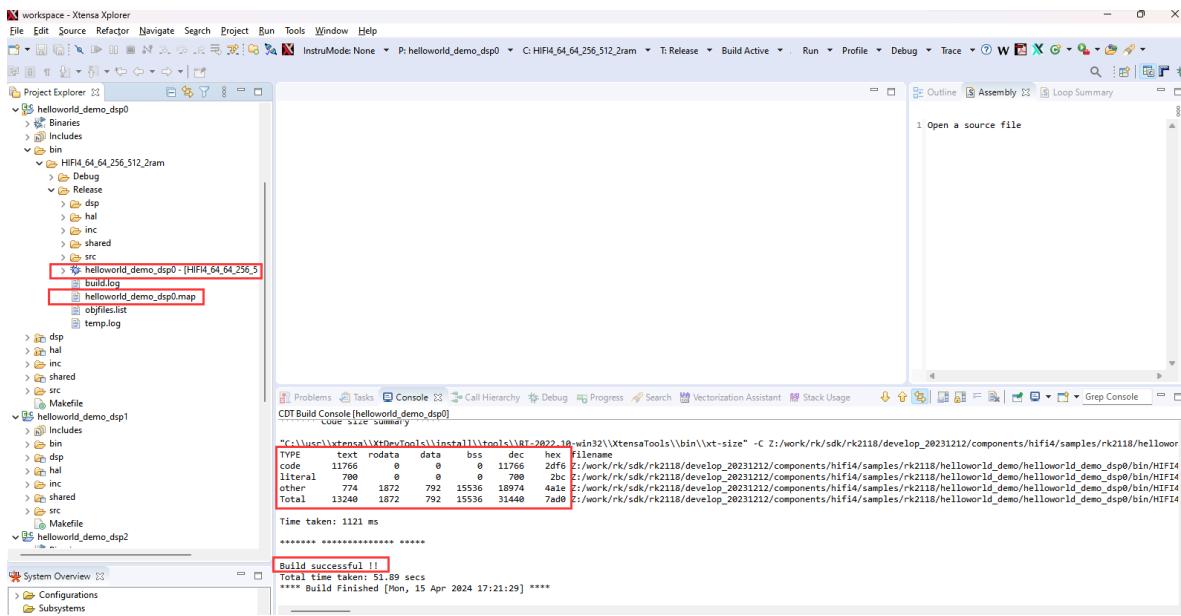
5.2.8 编译构建

在顶部工具栏选择DSP的配置(Build Active)：



Build Active是编译， Clean Active是清除编译内容， Rebuild Active是重新编译。

点击Build Active后，编译成功如下所示，在底部窗口会显示Build Successful !!和程序使用的静态内存空间，不包括动态内存分配空间。



左侧Project Explorer红框中就是编译出来的ELF文件 helloworld_demo_DSP0，可以看到 bin 目录底下的目录是和选择编译目标 Debug、Release 一一对应的。

同样，通过选择项目编译 helloworld_demo_DSP1 和 helloworld_demo_DSP2，这样3个dsp项目就编译完成了。

另外之前配置的 mem_layout.h 也会在编译过程中通过 mkld.py 解析，打印出使用的 mem_layout.h 文件和目前 DSP SRAM 和 DDR 的配置，生成对应的链接脚本：

```

Executing pre-build step...
cmd /c python Z:/work/rk/sdk/rk2118/develop/components/hifi4/tools/mkld.py

***** lsp *****

Project mem layout path is consistent with .config.

MEMORY LAYOUT FILE:
Z:\work\rk\ sdk\rk2118\develop_20231212\components\hifi4\samples\rk2118\helloworld_demo\mem_layout.h

SRAM_DSP0_BASE: 0x30220000, SRAM_DSP0_END: 0x3026ffff, SRAM_DSP0_SIZE: 320KB
SRAM_DSP1_BASE: 0x30270000, SRAM_DSP1_END: 0x302b7fff, SRAM_DSP1_SIZE: 288KB
SRAM_DSP2_BASE: 0x302b8000, SRAM_DSP2_END: 0x302fefff, SRAM_DSP2_SIZE: 284KB
DRAM_DSP0_BASE: 0xa0600000, DRAM_DSP0_END: 0xa18fffff, DRAM_DSP0_SIZE: 19456KB
DRAM_DSP1_BASE: 0xa1900000, DRAM_DSP1_END: 0xa2bfffff, DRAM_DSP1_SIZE: 19456KB
DRAM_DSP2_BASE: 0xa2c00000, DRAM_DSP2_END: 0xa3efffff, DRAM_DSP2_SIZE: 19456KB

```

编译完成后，在hifi4目录下，可以使用如下指令生成3个dsp的bin文件：

```

./tools/mkfw.py --chip RK2118 --dsp0
samples/rk2118/helloworld_demo/helloworld_demo_dsp0/bin/HIFI4_64_64_256_512_2ram
/Release/helloworld_demo_dsp0 --output-format bin -o rtt/dsp_fw/dsp0
./tools/mkfw.py --chip RK2118 --dsp1
samples/rk2118/helloworld_demo/helloworld_demo_dsp1/bin/HIFI4_64_64_256_512_2ram
/Release/helloworld_demo_dsp1 --output-format bin -o rtt/dsp_fw/dsp1
./tools/mkfw.py --chip RK2118 --dsp2
samples/rk2118/helloworld_demo/helloworld_demo_dsp2/bin/HIFI4_64_64_256_512_2ram
/Release/helloworld_demo_dsp2 --output-format bin -o rtt/dsp_fw/dsp2

```

这会在rtt/dsp_fw生成dsp0.bin、dsp1.bin和dsp2.bin文件，通过固件打包程序烧入到flash完成dsp固件更新（默认的setting配置的打包路径即是dsw_fw目录下的dsp0.bin、dsp1.bin和dsp2.bin）。

注意：发布测试最好保留好ELF文件，在出错的时候会打印相关的异常堆栈，可以通过ELF调试异常。

5.3 创建工程

5.3.1 拷贝模板工程

执行如下命令创建工程：

```
./tools/mkprojects.py --template samples/rk2118/helloworld_demo/ -o my_demo
```

在samples\demo\rk2118目录，基于helloworld_demo模板工程，创建新的my_demo工程。

然后把my_demo工程添加到samples/rk2118/Kconfig目录：

```

--- a/samples/rk2118/Kconfig
+++ b/samples/rk2118/Kconfig
@@ -19,6 +19,9 @@ config HIFI4_PROJECT_VOCAL_SEPARATE_SPDIFRX_DEMO

config HIFI4_PROJECT_FACC_DEMO
    bool "facc Demo"
+
+config HIFI4_PROJECT_MY_DEMO
+    bool "my Demo"

```

```

endchoice

config HIFI4_PROJECT_MEM_LAYOUT_FILE
@@ -28,6 +31,7 @@ config HIFI4_PROJECT_MEM_LAYOUT_FILE
    default "components/hifi4/samples/rk2118/vocal_separate_demo/mem_layout.h"
if HIFI4_PROJECT_VOCAL_SEPARATE_DEMO
    default
"components/hifi4/samples/rk2118/vocal_separate_spdifrx_demo/mem_layout.h" if
HIFI4_PROJECT_VOCAL_SEPARATE_SPDIFRX_DEMO
    default "components/hifi4/samples/rk2118/facc_demo/mem_layout.h" if
HIFI4_PROJECT_FACC_DEMO
+    default "components/hifi4/samples/rk2118/my_demo/mem_layout.h" if
HIFI4_PROJECT_MY_DEMO
    help
        The selected hifi4 project to be built.
endmenu

```

后续可以参考导入工程章节，开始导入工程、配置工程和编译工程。

注意：DSP的工程只能放到samples\demo\rk2118目录下，原因是工程涉及到一些相对路径。

5.4 添加源码

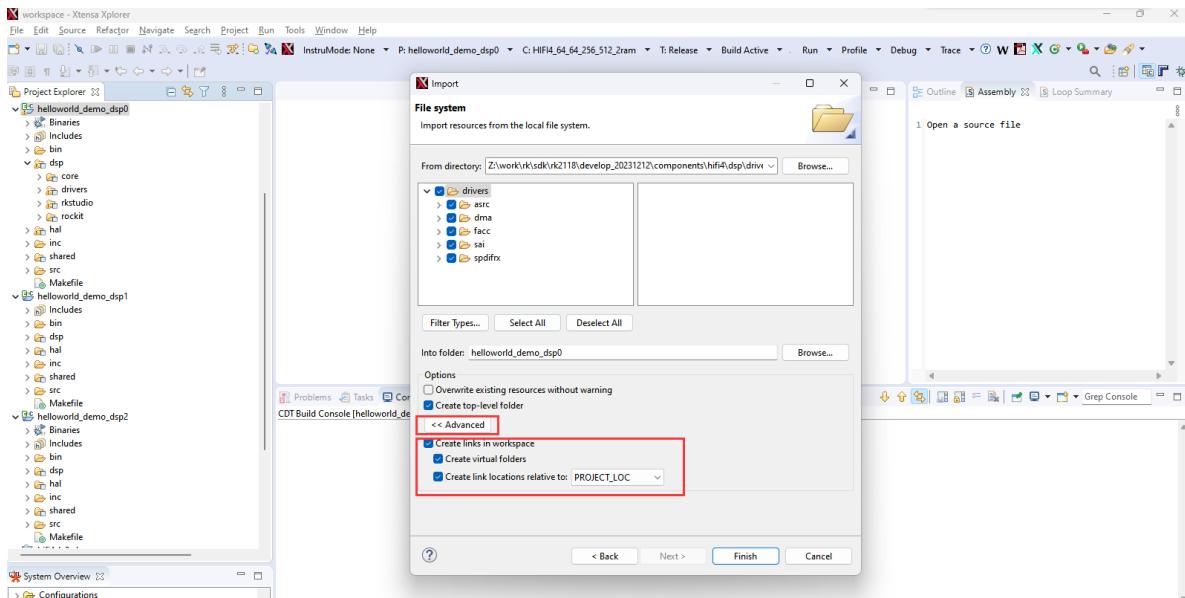
5.5 添加单个dsp私有的源码

在对应dsp项目中，选择new，可以新建文件夹，添加.c .h文件，和其它IDE没有差别。

5.6 添加所有dsp共享的源码

例如，hifi4/dsp目录是所有dsp共享的目录，helloworld_demo_dsp0项目导入这个共享目录源码。

右键选择项目，选择import，在import提示框中选择File System，然后点击Next:



点击Finish，完成导入。

注意：红框部分，一定要选择，这样就在dsp工程中导入了dsp共享的源码，创建了虚拟文件夹（有点类似linux中的软连接）。共享的源码会在左侧项目文件中显示，并且对应的文件夹带有一个空心的矩形标识。