DSP
环境开发指南

版本号：1.0
发布日期：2021.04.19

版本历史

版本号	日期	制/修订人	内容描述
1.0	2021.04.19	AWA1730	建立初版

版权所有 © 珠海全志科技股份有限公司。保留一切权利

目 录

1 前言 1
1.1 文档简介 1
1.2 目标读者 1
1.3 适用范围 1

2 DSP 介绍 2
2.1 功能介绍 2
2.2 相关术语介绍 2

3 资料准备 3

4 Linux 环境 4
4.1 SDK 框架 4
4.1.1 arch 目录结构 4
4.1.2 components 目录结构 4
4.1.3 drivers 目录结构 4
4.1.4 include 目录结构 4
4.1.5 kernel 目录结构 4
4.1.6 projects 目录结构 5
4.1.7 XtDevTools 5
4.2 环境安装 5
4.2.1 XCC 安装 5
4.2.2 package 包安装 6
4.2.3 环境变量 6
4.3 编译代码 6
4.4 下载固件 7
4.5 代码集成 7
4.5.1 集成源码 7
4.5.2 集成静态库 8

5 Windows 环境 9
5.1 工具安装 9
5.1.1 Xplorer 安装 9
5.1.2 xt-ocd 安装 12
5.1.3 JLink 驱动安装 15
5.1.4 License 配置 16
5.1.5 安装 package 包 17
5.2 工具使用 21
5.2.1 HelloWorld 工程 22
5.2.2 Jtag 调试 26
5.2.2.1 配置 26
5.2.2.2 探测 29

ALLWINNER®

5.2.2.3 选择 LSP 文件 31
5.2.2.4 elf 文件替换 33
5.2.2.5 单核调试 33
5.2.2.6 双核调试 35
5.2.2.7 常用调试窗口 35
5.2.2.8 设置源文件路径 36

6 系统 **41**
6.1 DSP 系统 41
6.1.1 LSP 文件 41
6.1.1.1 LSP 作用 41
6.1.1.2 LSP 生成 41
6.1.2 memmap 文件 42
6.1.3 启动 43
6.1.3.1 DSP 复位入口 43
6.1.3.2 _ResetHandler 复位函数 43
6.1.3.3 _start 函数 44
6.1.4 cache 控制器 44
6.1.5 中断 44
6.1.5.1 中断等级 44
6.2 FreeRTOS 系统 45
6.2.1 FreeRTOSConfig.h 配置 45
6.2.1.1 调度器 46
6.2.1.2 系统节拍 46
6.2.1.3 任务优先级 46
6.2.1.4 任务堆 47
6.2.1.5 中断管理 47
6.2.2 内存管理 47

7 调试组件 **49**
7.1 串口调试 49

插　图

5-1 安装步骤 1 9
5-2 安装步骤 2 10
5-3 安装步骤 3 10
5-4 安装步骤 4 11
5-5 安装步骤 5 11
5-6 安装步骤 6 12
5-7 xt-ocd 安装步骤 13
5-8 缺少 ftd2xx.dll 13
5-9 获取 ftd2xx.dll 路径 14
5-10 解决了 ftd2xx.dll 缺失问题 14
5-11 JLink 驱动安装步骤 15
5-12 JLink 成功识别 15
5-13 配置 License 步骤 1 16
5-14 配置 License 步骤 2 17
5-15 配置 License 步骤 3 17
5-16 检查 package 18
5-17 安装 package 步骤 1 18
5-18 安装 package 步骤 2 19
5-19 安装 package 步骤 3 20
5-20 安装 package 步骤 4 21
5-21 选择 workspace 22
5-22 配置 HelloWorld 工程步骤 1 23
5-23 配置 HelloWorld 工程步骤 2 23
5-24 配置 HelloWorld 工程步骤 3 24
5-25 编译 HelloWorld 工程 25
5-26 生成 HelloWorld 程序 26
5-27 配置 Debug 选项 27
5-28 Debug Configuration 27
5-29 J-Link Commander 28
5-30 core0 config 28
5-31 two cores config 29
5-32 Diagnose 29
5-33 Diagnose one core 30
5-34 Diagnose two cores 31
5-35 选择 debug_lsp 步骤 1 32
5-36 选择 debug_lsp 步骤 2 32
5-37 选择 debug_lsp 步骤 3 33
5-38 开始 Debug 34
5-39 Download Binary Image 34
5-40 Core info 35
5-41 Two Cores info 35

ALLWINNER®

5-42 Debug Windows 36
5-43 Source Lookup 36
5-44 Show Navigator 37
5-45 Create Folder 38
5-46 Create Folder 39
5-47 Src file 40
6-1 Interrupt processing 45
6-2 heap 选择 48
7-1 dsp_uart 输出 49

1 前言

1.1 文档简介

本文档是让开发者了解 DSP 平台，能够在 DSP 平台上开发新的算法方案。

1.2 目标读者

DSP 系统开发人员。

1.3 适用范围

表 1-1: 适用产品列表

产品名称	DSP 类型
R329	HIFI4
T113	HIFI4
D1	HIFI4

2 DSP 介绍

2.1 功能介绍

基于 Cadence Xtensa HIFI4 DSP 进行开发，该 DSP 具有:

- 以 72 位元累加器支援每循环 4 个 32x32 位元乘数累加器 (multiplier-accumulators，MACs)
- 在特定条件下，支援每循环 8 个 32x16 位元 MACs
- 4 个超长指令集 (VLIW) 插槽架构，能够每循环发出 2 个 64 位元负载
- 备有向量浮点运算单元可供选购，提供高达每循环 4 个单精密度 IEEE 浮点运算 MAC

2.2 相关术语介绍

术语	解释说明
sunxi	指 Allwinner 的一系列 SOC 硬件平台
DSP	Digital Signal Processor 数字信号处理器
FreeRTOS	一种开源的实时操作系统
XEA2	Xtensa Exception Architecture
Xplorer	Cadence 提供的一款基于 eclipse 的类 DS-5 的调试环境

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3 资料准备

要编译和仿真 DSP，需要以下资料:

1、Allwinner 发布的 D1 开发板中 DSP 核 SDK，SDK 需要包含 DSP 编译源码，Linux 版本 XCC 工具链，Linux 版本 DSP 的 package 包

2、Cadence Xtensa 的 Windows IDE 工具 (Xplorer-8.0.13 版本), Windows 版本 DSP 的 package 包

3、Cadence Xtensa 的 License，用于服务器代码编译和 Xplorer 仿真使用

获取以上资料请与 Allwinner 联系!

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4 Linux 环境

4.1 SDK 框架

整个 SDK 主要包括 arch（架构相关）、components（组件）、drivers（驱动）、kernel（内核）、projects（工程）、XtDevTools（DSP 核心 package 配置和 Linux XCC 工具链）。

4.1.1 arch 目录结构

arch 目录主要放置跟 SoC 架构相关的内容，每个 SoC 单独目录管理，目前 SDK 中有 D1，主要包括跟 SoC 相关的 init、lsp 等的实现。

4.1.2 components 目录结构

components 目录包含 allwinner 和第三方的组件。

4.1.3 drivers 目录结构

drivers 目录包含 DSP 所需的外设驱动，而板级相关的驱动则位于 board 目录。

4.1.4 include 目录结构

include 目录统一管理各模块提供的数据结构定义及函数声明。

4.1.5 kernel 目录结构

kernel 目录主要包含 FreeRTOS 的 kernel 源码、Xtensa 的 portable 源码以及我们实现的系统功能相关代码。

其中 FreeRTOS 的源码是根据 Cadence Support 的推荐从 Github 上拉取：

```
https://github.com/foss-xtensa/amazon-freertos （tag：v1.7-xtensa）
```

注意：使用 v1.7-xtensa 版本。

4.1.6 projects 目录结构

projects 目录下的每一个子目录代表一个 project，实现 main 入口，选择不同的 project 编译出来的 bin 具有不同功能。每个 project 有独立的 FreeRTOSConfig 配置。

4.1.7 XtDevTools

Linux XCC 工具链存放路径为:

```
<root>/XtDevTools/install/RI-2020.4-linux/XtensaTools
```

DSP 核心 package 配置存放路径为:

```
<root>/XtDevTools/downloads/aw_axi_cfg0_linux.tgz
```

Allwinner 的 DSP 核心配置包由数字设计提供，一般为一个 tgz 压缩包，首次使用时需开发者解压安装至 SDK 的指定位置，后续会进行说明。

4.2 环境安装

4.2.1 XCC 安装

把 XCC 工具链压缩包放在目录下

```
<root>/XtDevTools/install/
```

进行解压，解压后工具链应该存放在

```
<root>/XtDevTools/install/RI-2020.4-linux/XtensaTools
```

4.2.2 package 包安装

AW 的 dsp 核配置包由数字设计提供，一般为一个 tgz 压缩包，首次使用时需开发者解压安装至 SDK 的指定位置，步骤如下：

```
(1) cd <root>/XtDevTools
(2) tar zxvf downloads/aw_axi_cfg0_linux.tgz
(3) cd RI-2020.4-linux/aw_axi_cfg0
(4) ./install
    ---- 进入交互式安装
    Are you ready to proceed? [y] y
    Enter the path to the Xtensa Tools directory: <root>/XtDevTools/install/RI-2020.4-linux
    /XtensaTools
    Do you want to continue? [y] y
    What registry would you like to use? [default] ../config
    This Xtensa registry directory does not exist.
    Do you want to create it? [y] y
    Do you want to make "aw_axi_cfg0" the default Xtensa core? [y] n
    ----完成
```

注意：如果安装过程中断，需要重新解压 tgz，按上述流程重新进行安装。

4.2.3 环境变量

env.sh 环境变量主要宏如下：

```
1. PATH  添加编译工具链bin/路径

2. LM_LICENSE_FILE  license服务器地址

3. XTENSA_SYSTEM  DSP核心配置包安装后的路径

4. XTENSA_CORE  DSP核心配置包名称

5. XTENSA_TOOLS_DIR  XCC工具链位置
```

注意:DSP license 需要请与 Allwinner 联系！

当想自定义编译环境或者排除编译环境问题，可以通过检查以上宏是否设置正确。

4.3 编译代码

```
source env.sh    ---- 配置环境变量

pick the version of XtensaTools:
     1. RI-2020.4-linux
```

```
Which would you like?1

pick a core config for Xtensa:
     1. aw_axi_cfg0

Which would you like?1
select=1...

pick a project:
     1. d1

Which would you like?1
select=1...

make clean

make menuconfig  ---- 编译选项，生成 .config

make -j32
```

4.4 下载固件

1. 把 SDK 目录下的 dsp.bin 拷贝到 Linux 源码./device/config/chips/d1/bin/ 目录下，并且重新命名为 dsp0.bin
2. 执行./build.sh 和./build.sh pack，重新打包 IMG 固件下载即可。

4.5 代码集成

4.5.1 集成源码

往 SDK 中集成代码主要需要修改对应的 Kconfig 和 Makefile，使用方法与 linux kernel 的类似。

以下以 components/thirdparty/algo_sample 为例进行说明：

1. 新建目录 `components/thirdparty/algo_sample`，在 components/thirdparty 的 Kconfig 中加入 config 选项 `config COMPONENTS_ALGO_SAMPLE`，Makefile 中加入 `obj-$(CONFIG_COMPONENTS_ALGO_SAMPLE) += algo_sample/`。
2. 编写 algo_sample 的 Makefile，加入需要编译的.o 文件，例如 `obj-$(CONFIG_COMPONENTS_ALGO_SAMPLE) += algo_sample.o` 表示当 `CONFIG_COMPONENTS_ALGO_SAMPLE` 为 `y` 时，会使用同名的 `*.c/*.cpp/*.S` 文件生成 `*.o`，此处会将 `algo_sample.c` 编译为 `algo_sample.o`。

如此生成的.o 文件最后会链接起来生成 dsp bin 文件。

4.5.2 集成静态库

SDK 也支持使用 `obj-$(CONFIG_XXX) += xxx.o` 的语法集成静态库。例如存在一个静态库 foobar.a，可以在 Makefile 中加入：

```
makefile
obj-$(CONFIG_COMPONENTS_FOOBAR) += foobar.o
```

如此一来编译时会将 foobar.a 内部的.o 文件解包出来，再全部链接为 foobar.o。

具体的例子可参考 components/thirdparty/xtensa/hifi4_vfpu_library。

5 Windows 环境

windows 环境主要是安装和使用 Cadence 提供的 IDE 工具 Xplorer

5.1 工具安装

5.1.1 Xplorer 安装

目前使用 Xplorer-8.0.13 版本

```
Xplorer-8.0.13-windows-installer.exe
```

Windows 下直接双击安装文件进行安装。

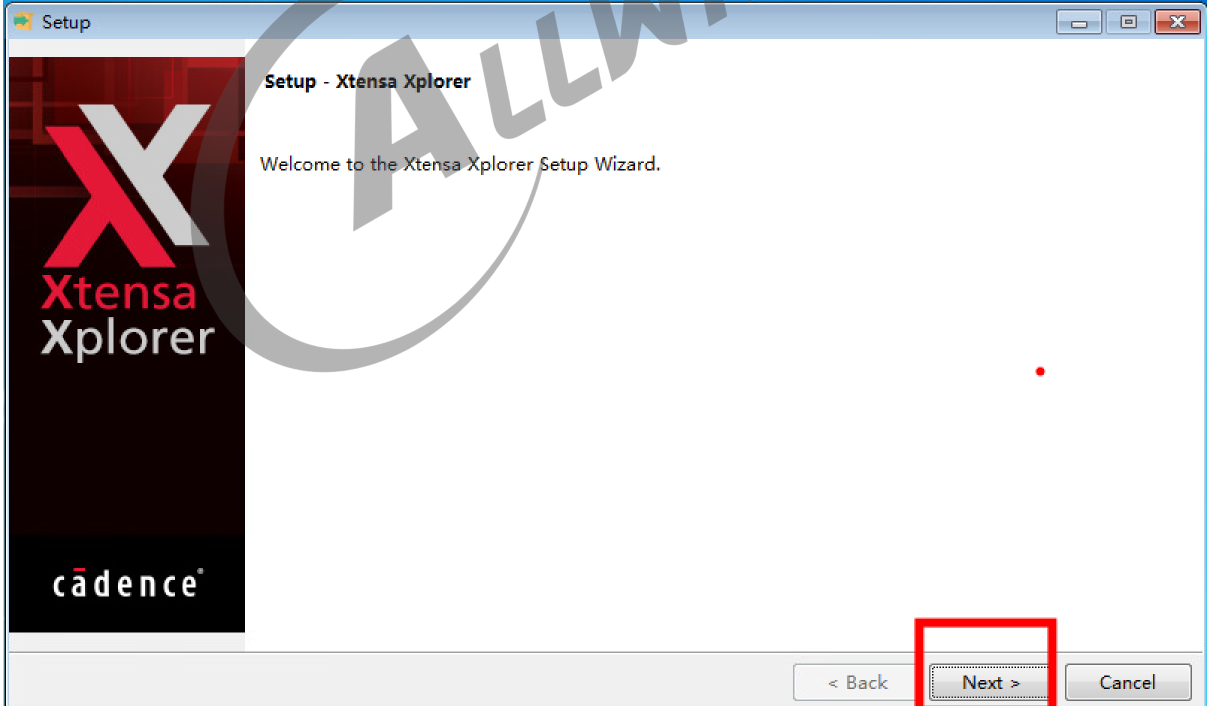

图 5-1：安装步骤 1

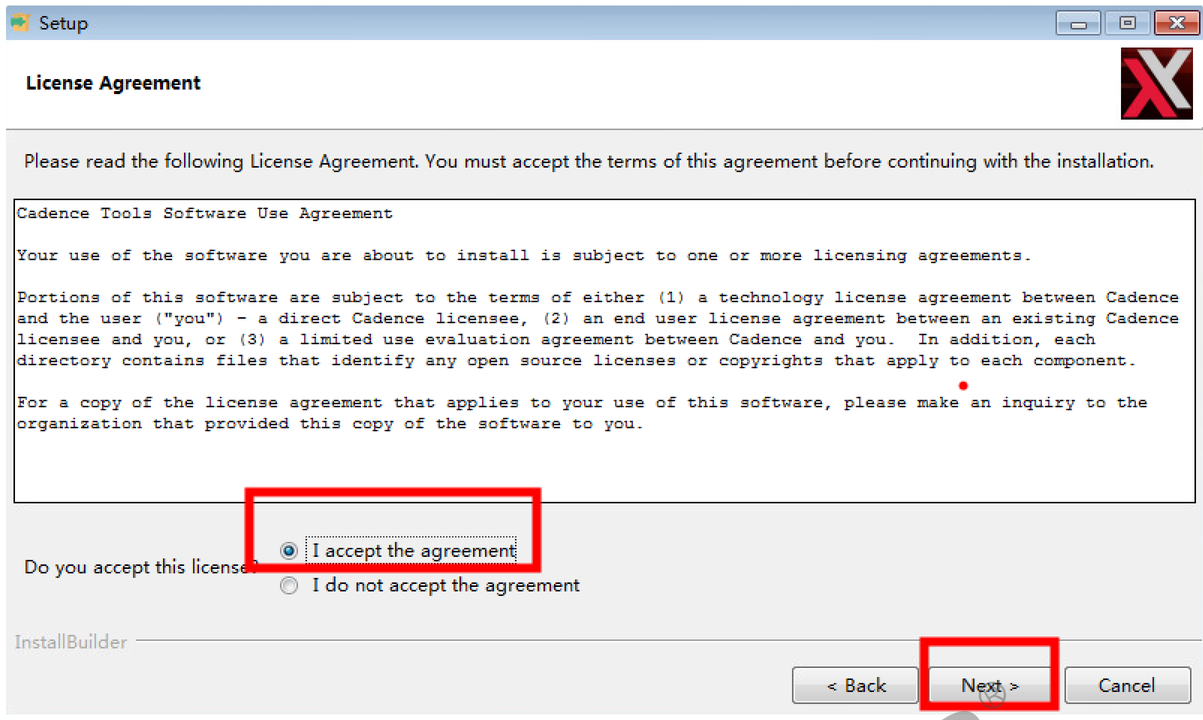

图 5-2: 安装步骤 2

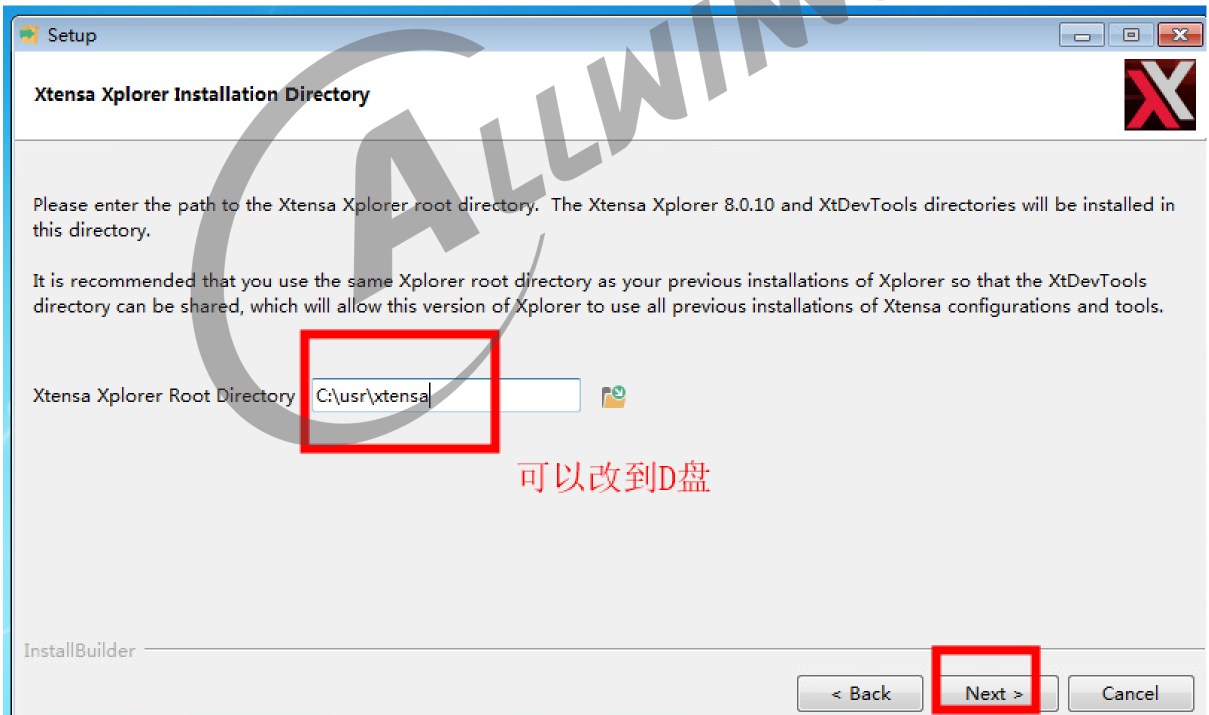

图 5-3: 安装步骤 3

如果本地之前没有安装过旧版的 Xplorer，选择 “否”，反之选择 “是”。

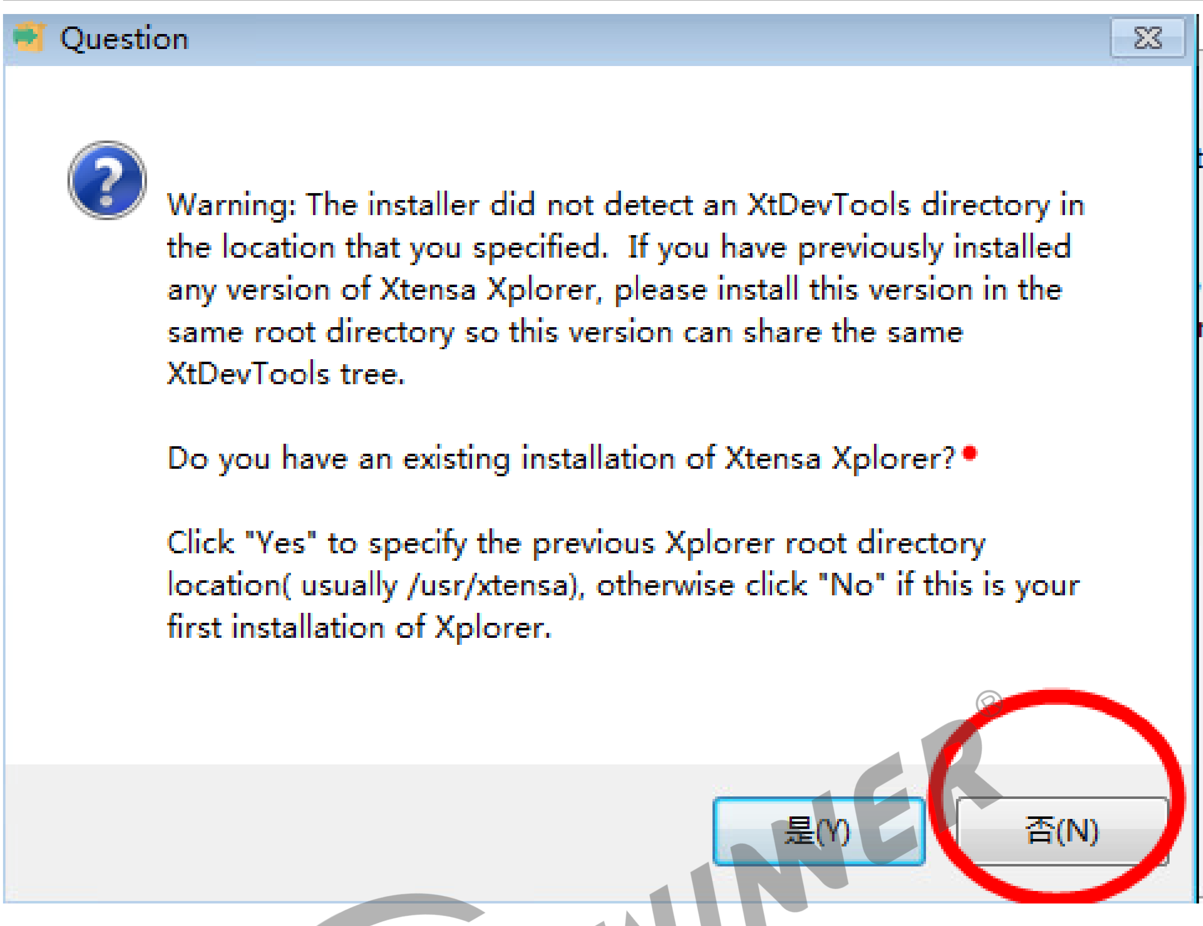

图 5-4: 安装步骤 4

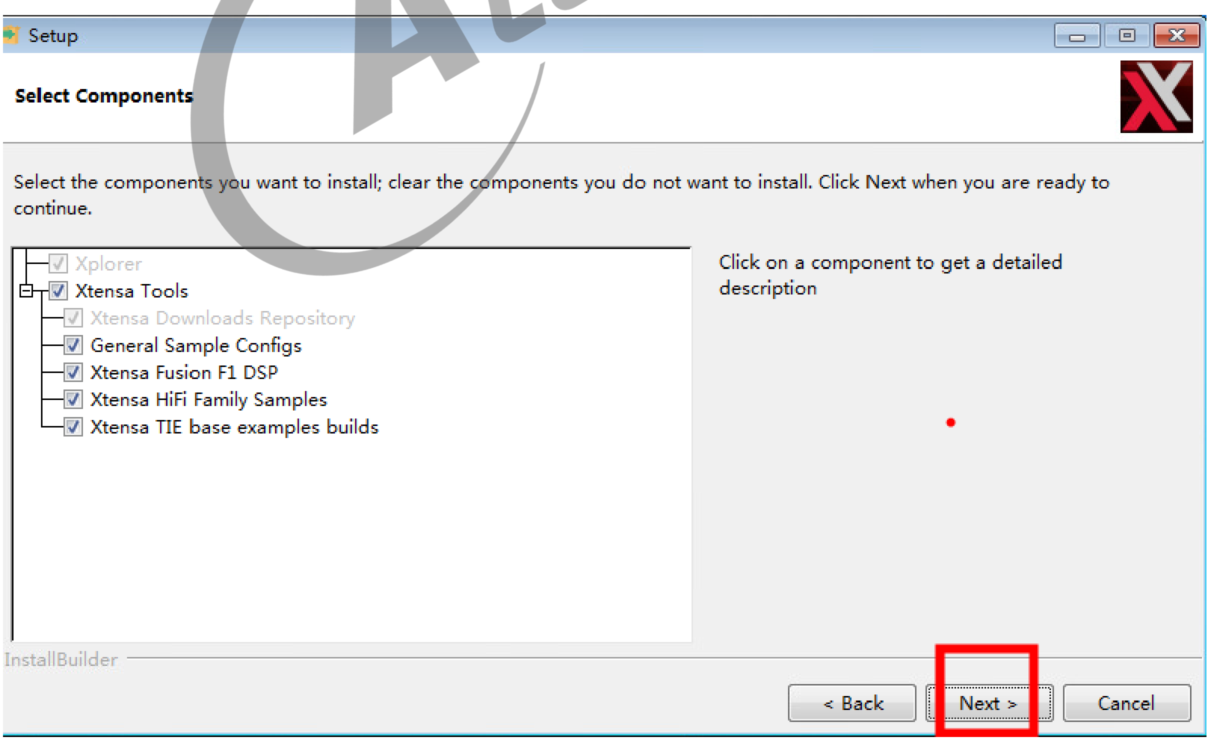

图 5-5: 安装步骤 5

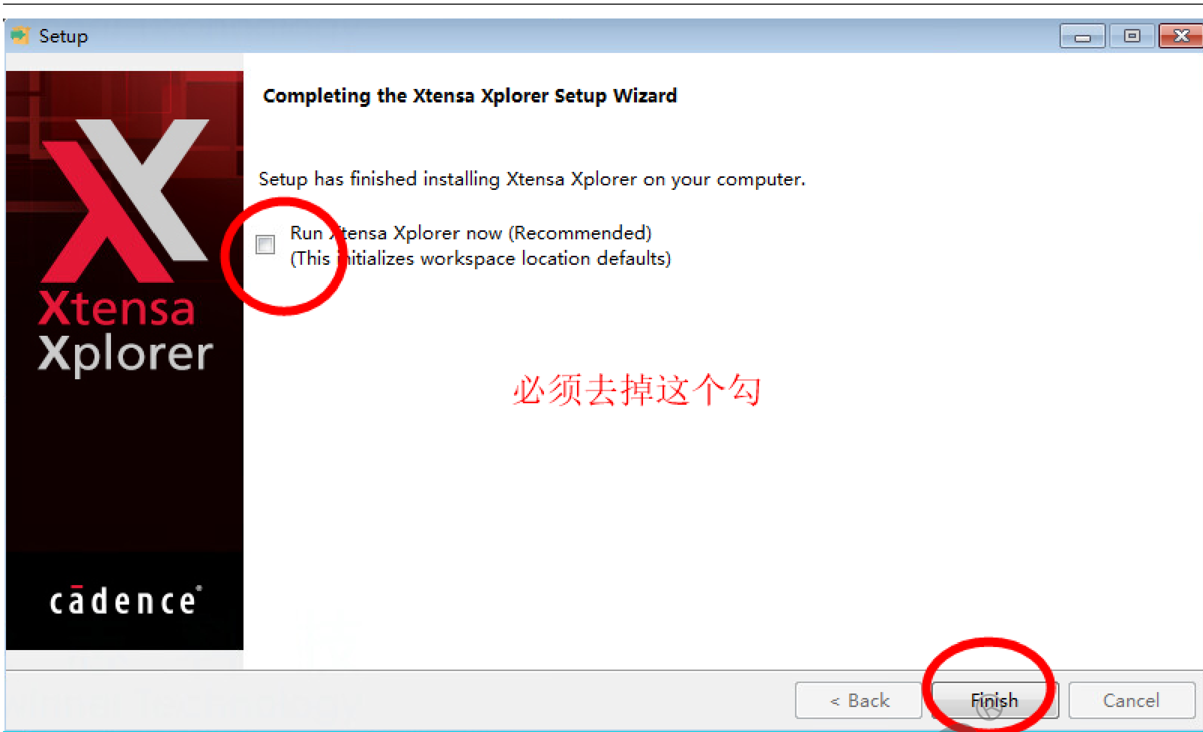

图 5-6: 安装步骤 6

5.1.2 xt-ocd 安装

安装完 Xplorer 后，还需要安装用于 Jtag 调试的 xt-ocd daemon，安装程序在 Xplorer 的安装路径下可以找到：

```
<install root>\XtDevTools\downloads\RI-2020.4\tools

xt-ocd-14.04-windows64-installer.exe
```

一直点击 Next 即可，安装路径建议跟 Xplorer 一致：

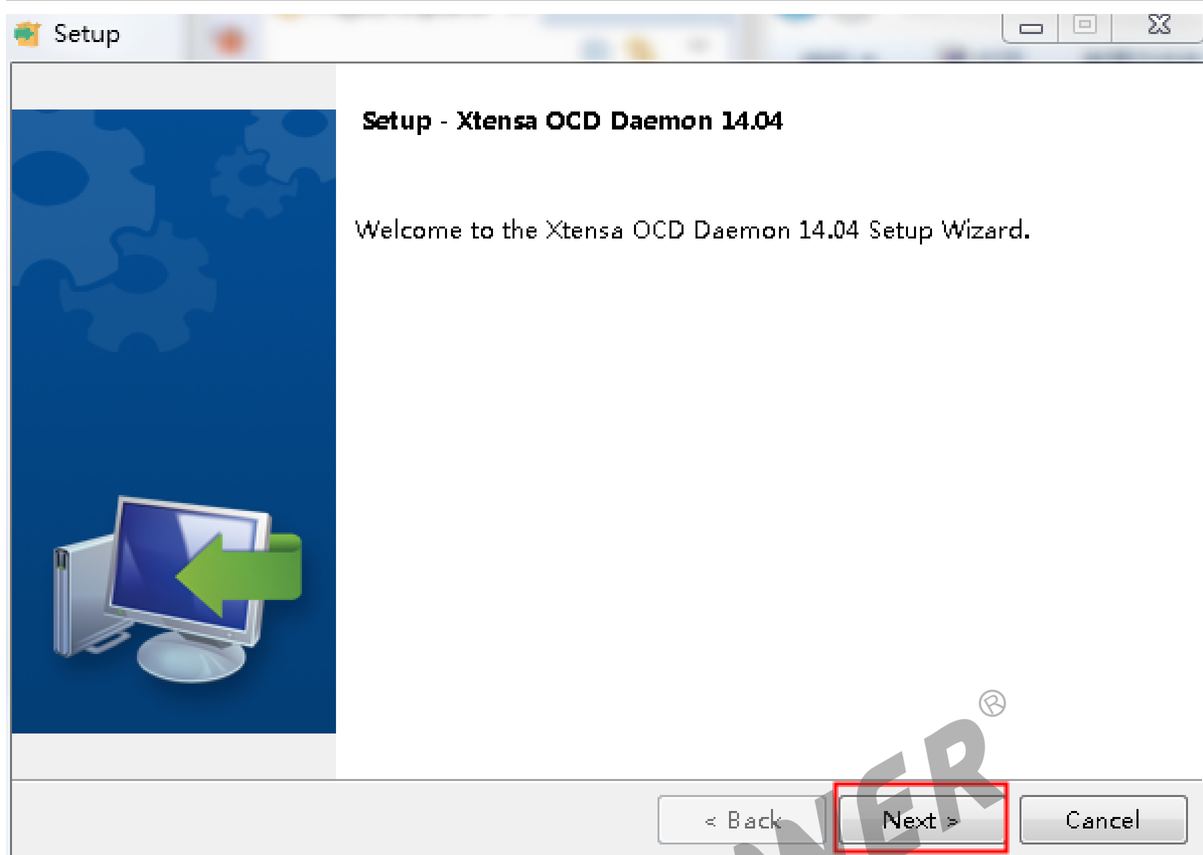

图 5-7: xt-ocd 安装步骤

安装完成后，我们检查 xt-ocd 能否正常启动，进到 xt-ocd 的安装目录，双击 xt-ocd.exe 运行，若出现如下错误，则说明计算机缺少 ftd2xx.dll：

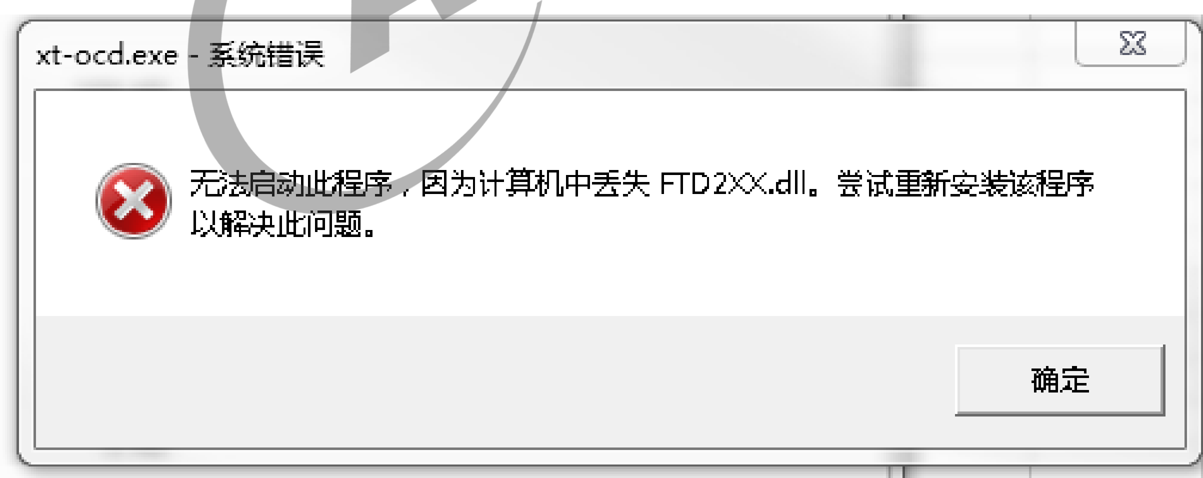

图 5-8: 缺少 ftd2xx.dll

而 ftd2xx.dll 在 xt-ocd 的安装路径下就能找到，如下：

```
<xt-ocd root>\modules\
```

```
ftd2xx64.dll
```

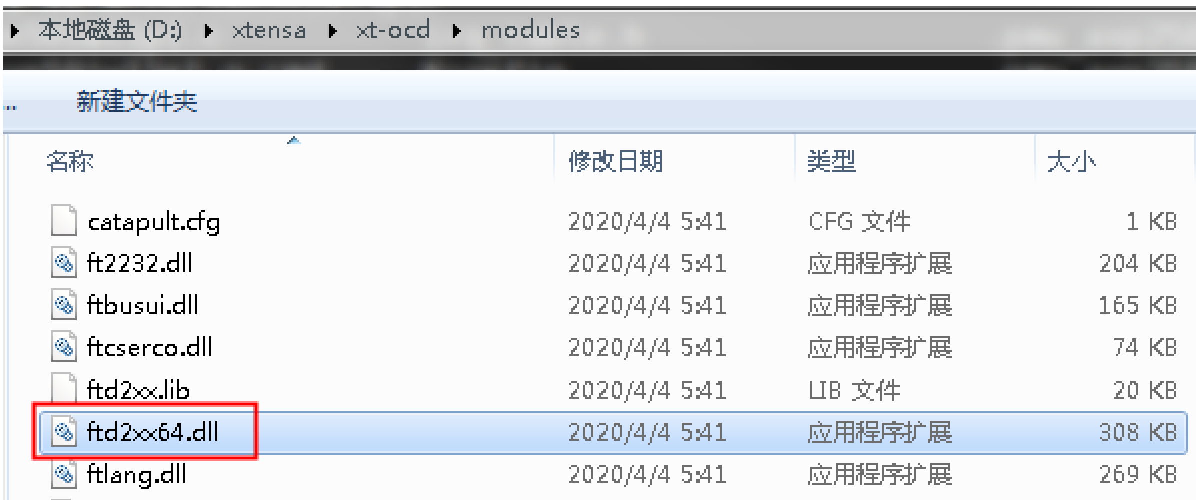

图 5-9: 获取 ftd2xx.dll 路径

只需将上述路径的 ftd2xx64.dll 拷贝到 xt-ocd 的安装根目录，重命名为 ftd2xx.dll 即可，重新双击 xt-ocd.exe 运行，之前的错误消失，如下（其他 error 提示不影响正常使用）：

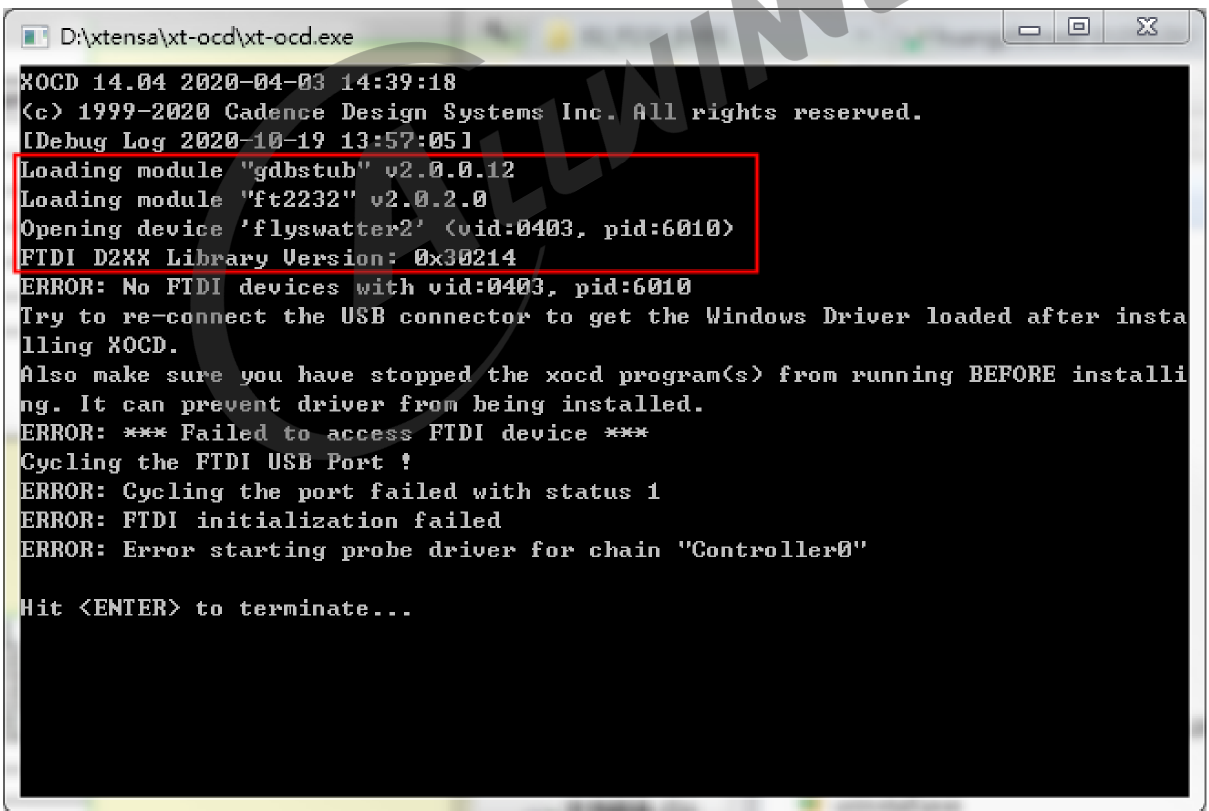

图 5-10: 解决了 ftd2xx.dll 缺失问题

5.1.3 JLink 驱动安装

JLink 是我们用于 Jtag 调试的调试器，需要安装驱动，安装包如下：

```
JLink_Windows_V654c.exe
```

一直点击 Next 即可：

图 5-11: JLink 驱动安装步骤

安装完成后，将 JLink 调试器接入 PC，即可成功安装驱动：

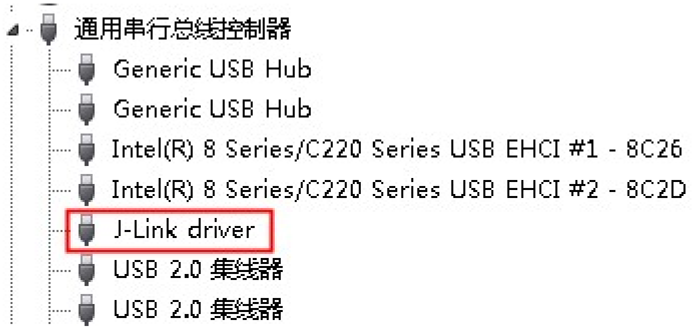

图 5-12: JLink 成功识别

5.1.4 License 配置

首次运行 Xplorer 需要进行 License 配置，与 Allwinner 联系！

以下举例说明配置 License 步骤：

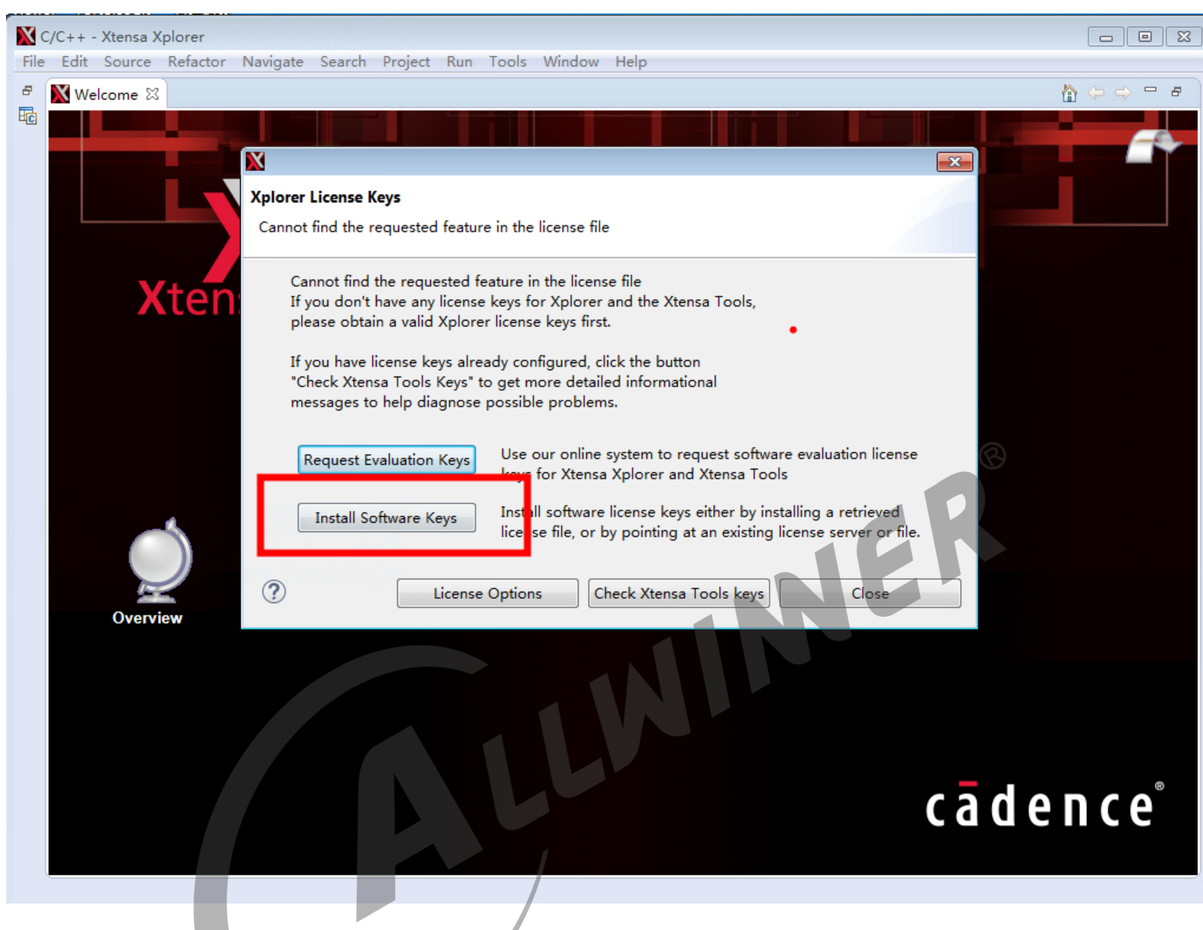

图 5-13: 配置 License 步骤 1

ALLWINNER

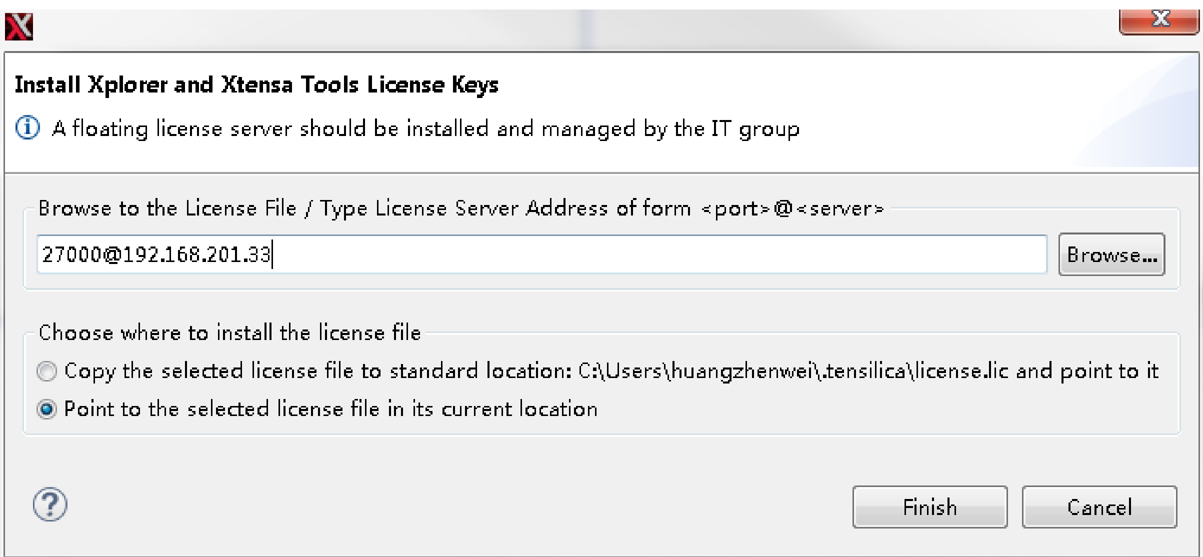

图 5-14: 配置 License 步骤 2

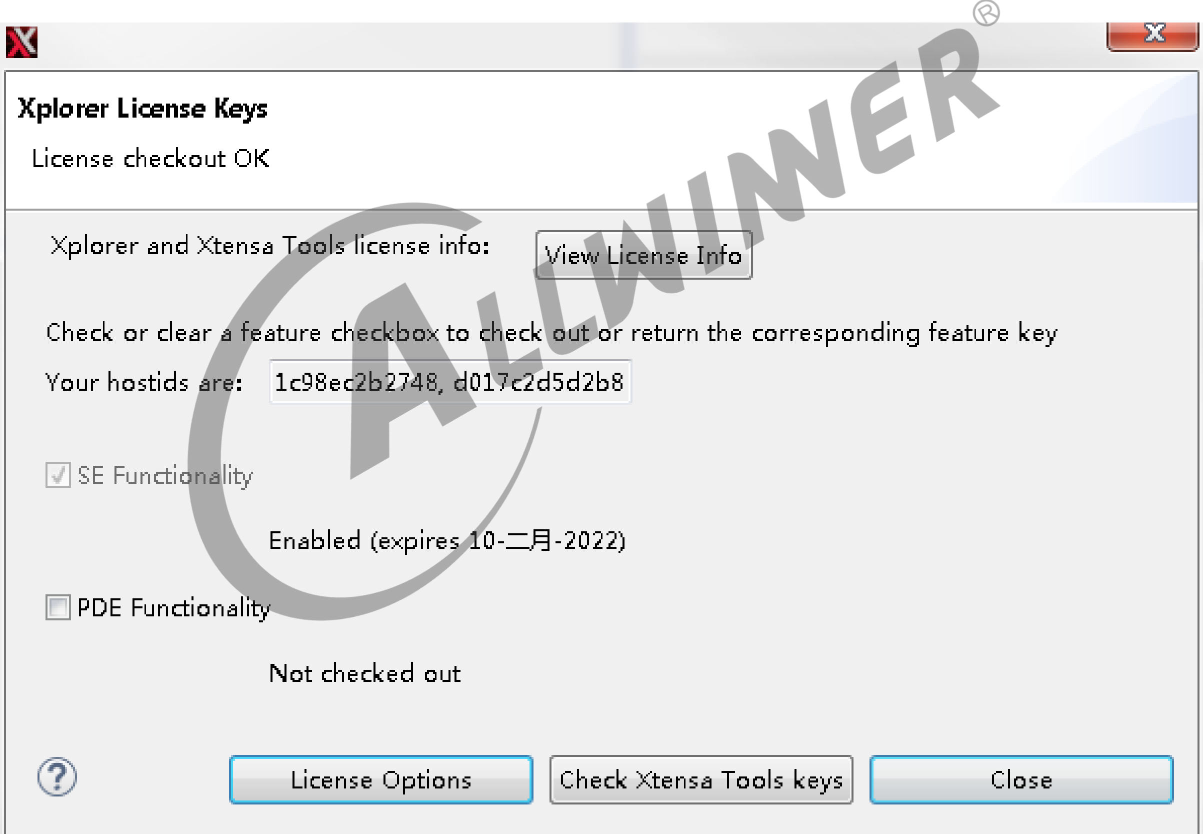

图 5-15: 配置 License 步骤 3

5.1.5 安装 package 包

首先检查是否正确安装好 package 包

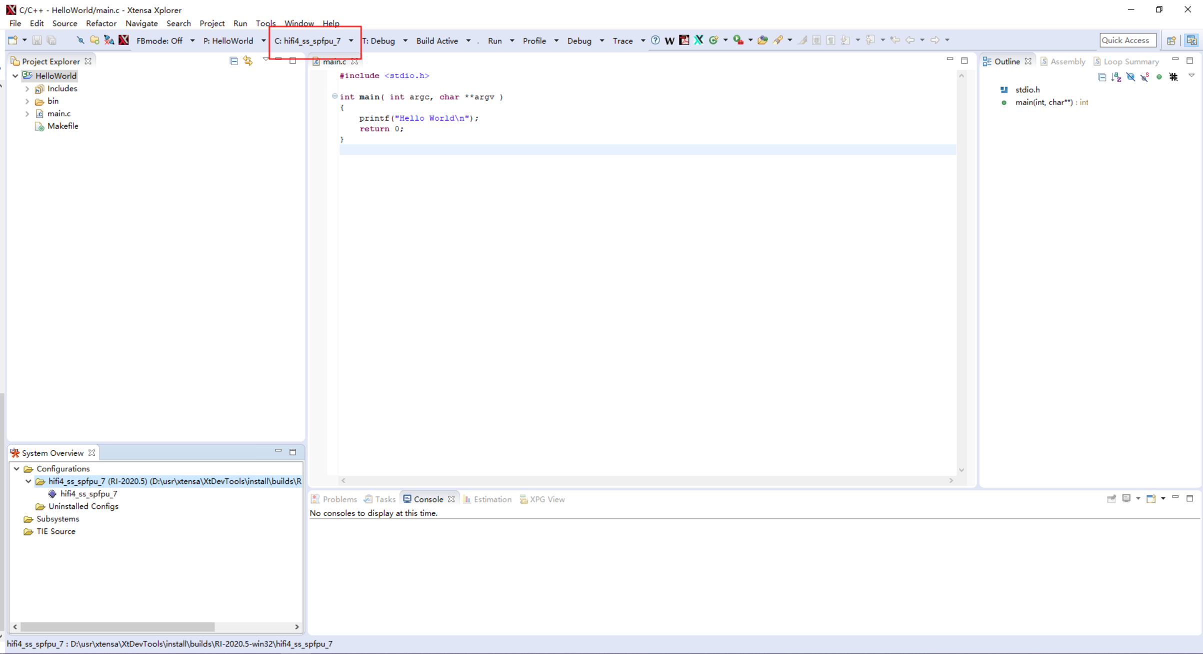

图 5-16: 检查 package

没有对应 package，我们要手动安装

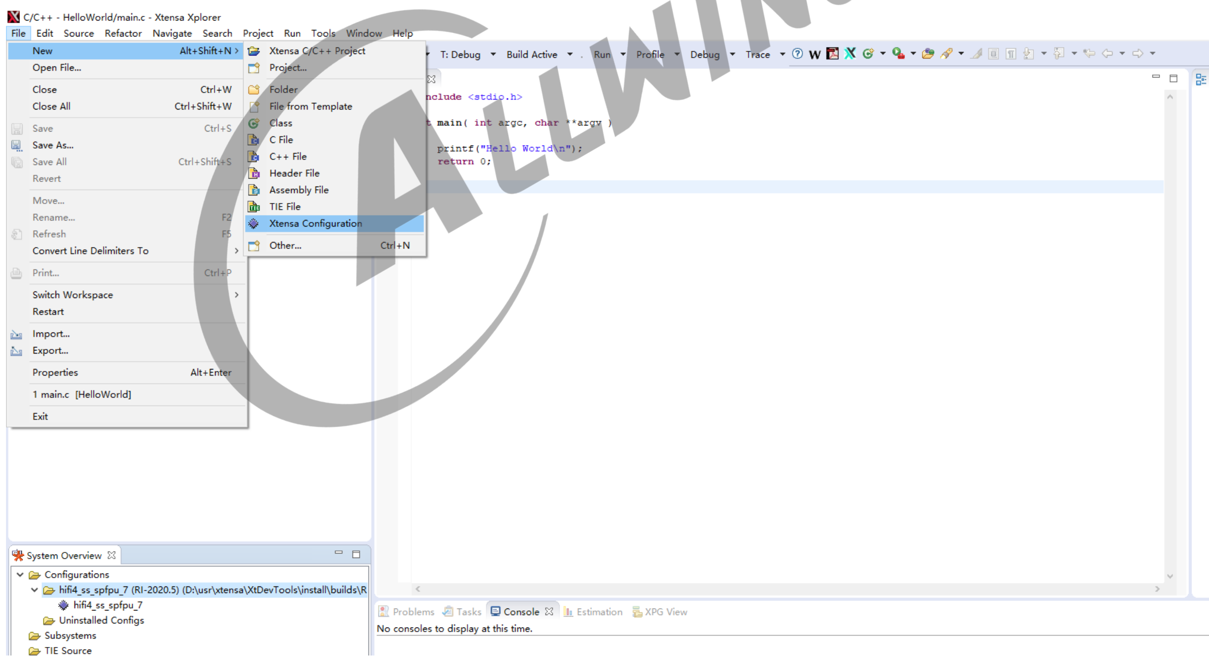

图 5-17: 安装 package 步骤 1

New Xtensa Configuration

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)

< Back | Next > | Finish | Cancel

图 5-18: 安装 package 步骤 2

版权所有 © 珠海全志科技股份有限公司。保留一切权利

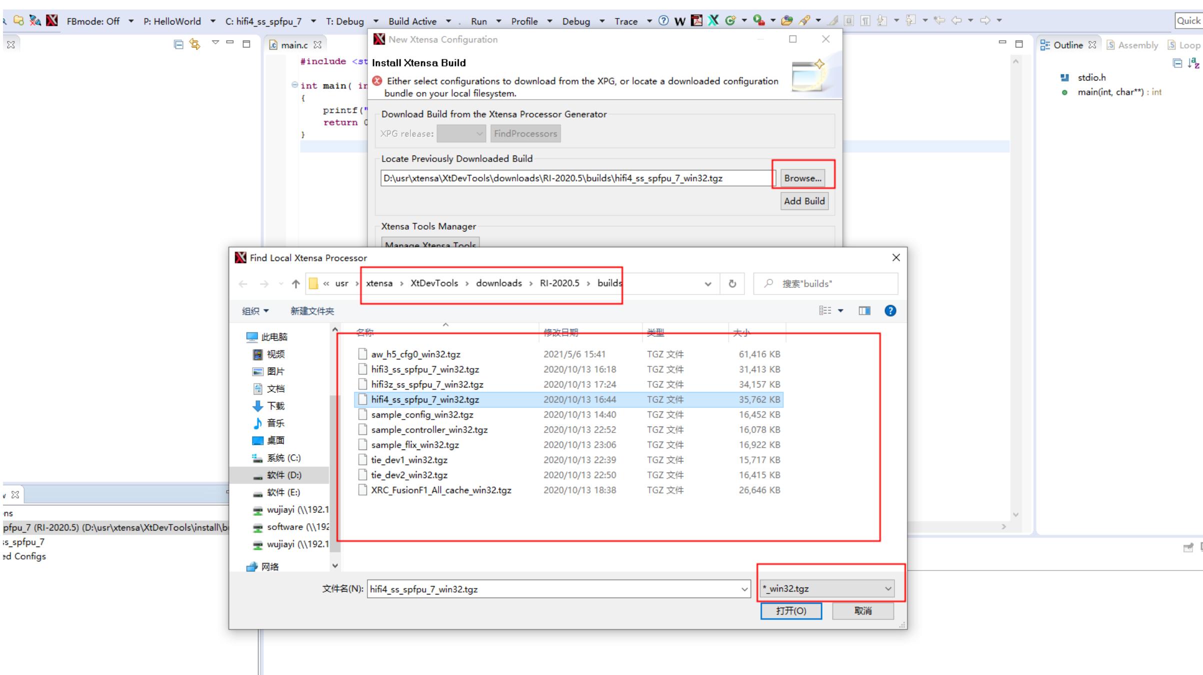

图 5-19: 安装 package 步骤 3

图 5-20: 安装 package 步骤 4

5.2 工具使用

Xplorer 是一款基于 eclipse 的调试环境，跟 ARM DS-5 类似，运行后首先要选择我们的 workspace 路径：

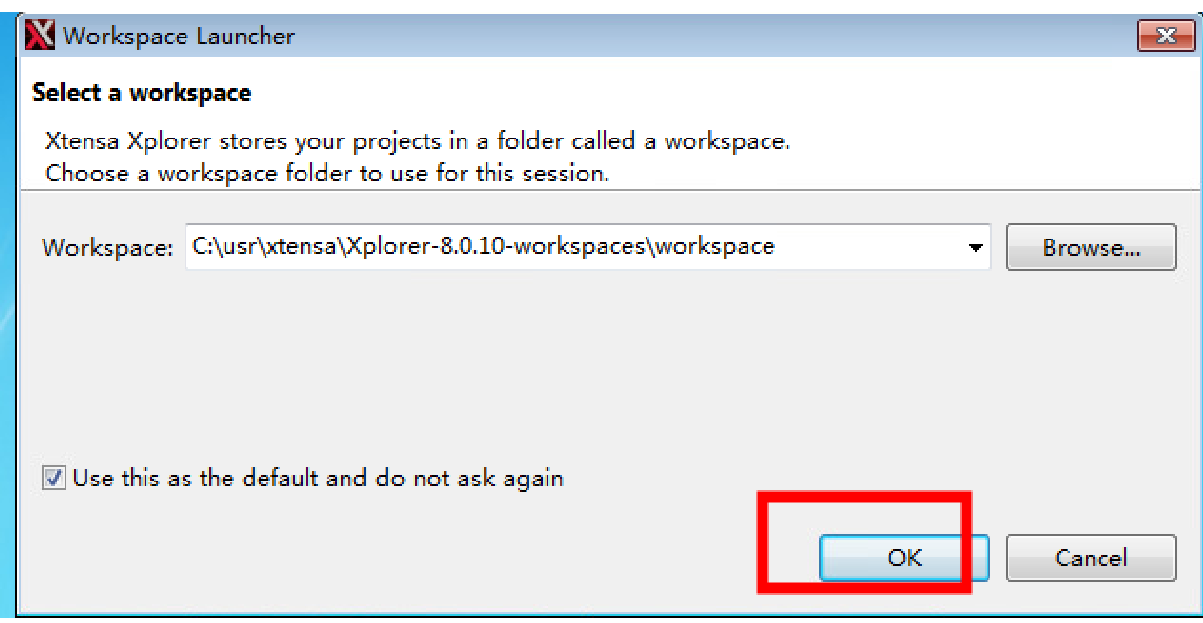

图 5-21：选择 workspace

5.2.1 HelloWorld 工程

Xplorer 本身具备编译功能，但我们的 DSP SDK 都是在 Linux 开发环境下编译的，多数情况下我们只用 Xplorer 进行 debug，因此我们需要新建一个工程环境用于调试。

这里我们选择 Xplorer 提供的 demo——HelloWorld 工程。步骤如下：

在 Help 菜单选择 “Welcome”：

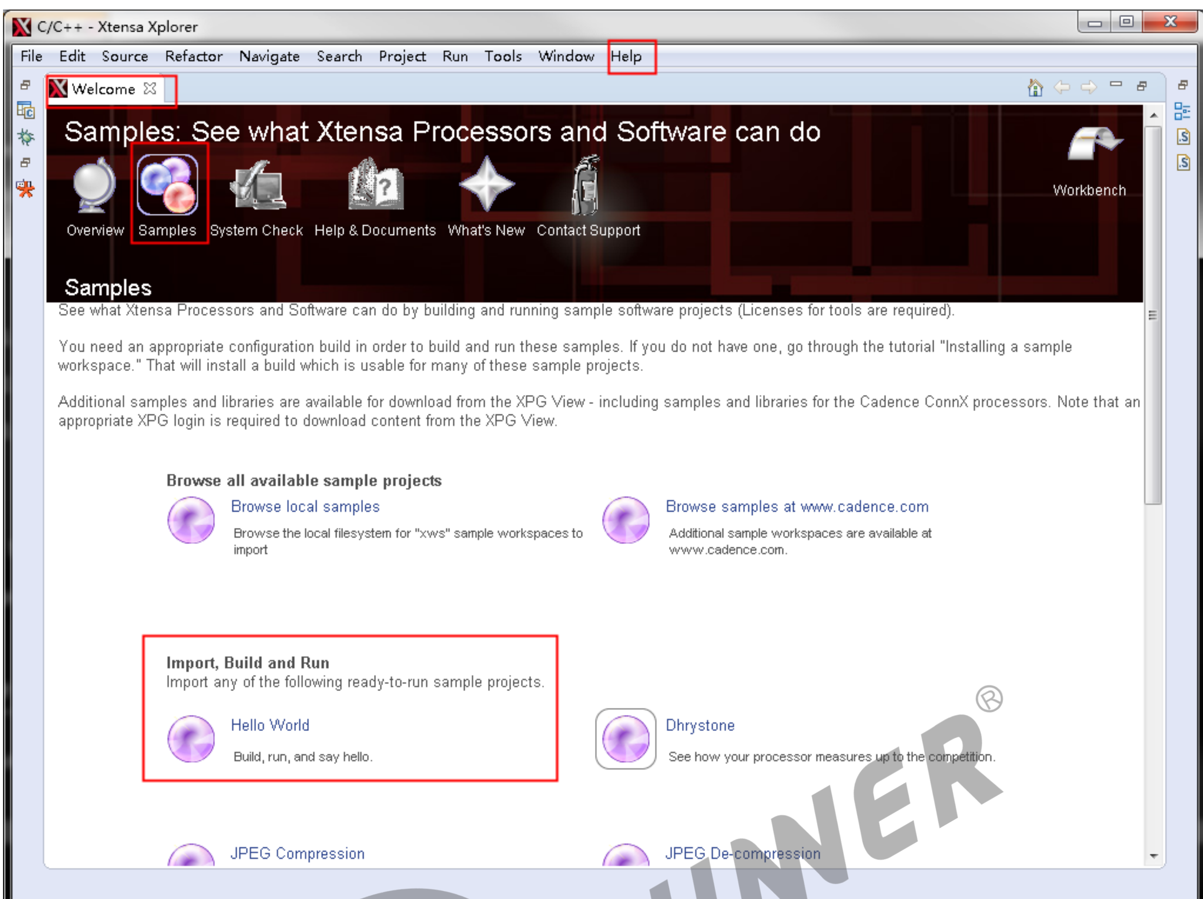

图 5-22: 配置 HelloWorld 工程步骤 1

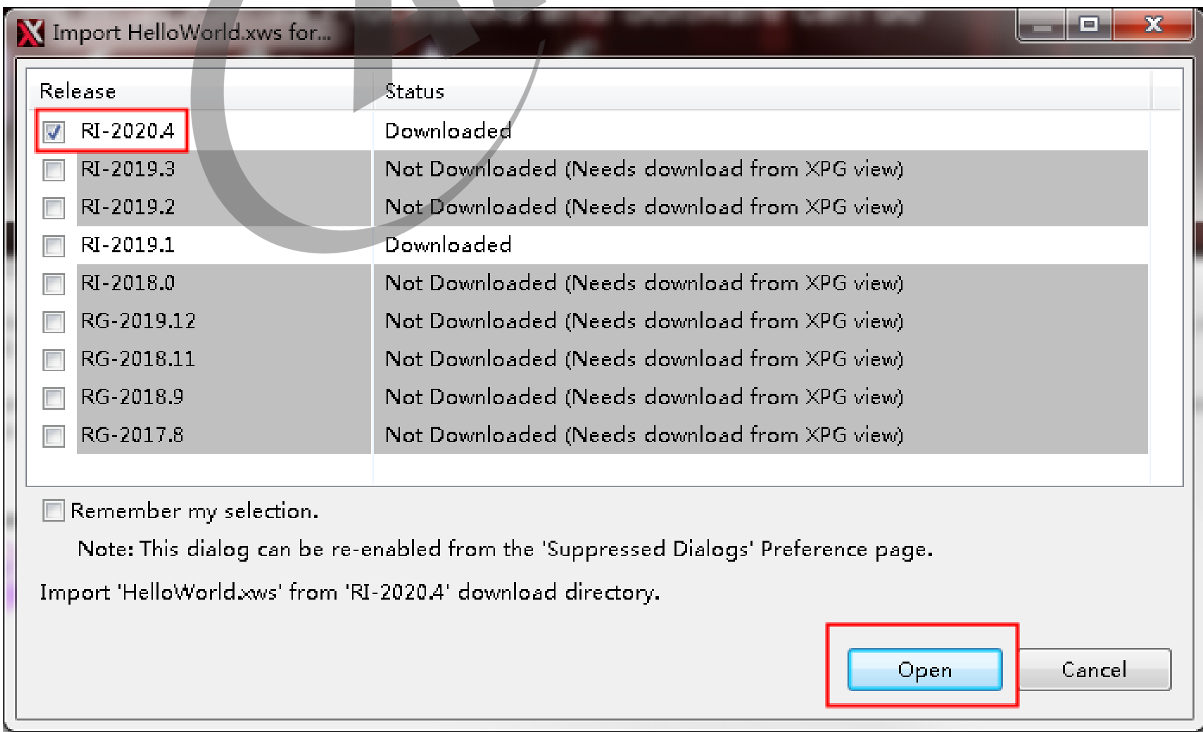

图 5-23: 配置 HelloWorld 工程步骤 2

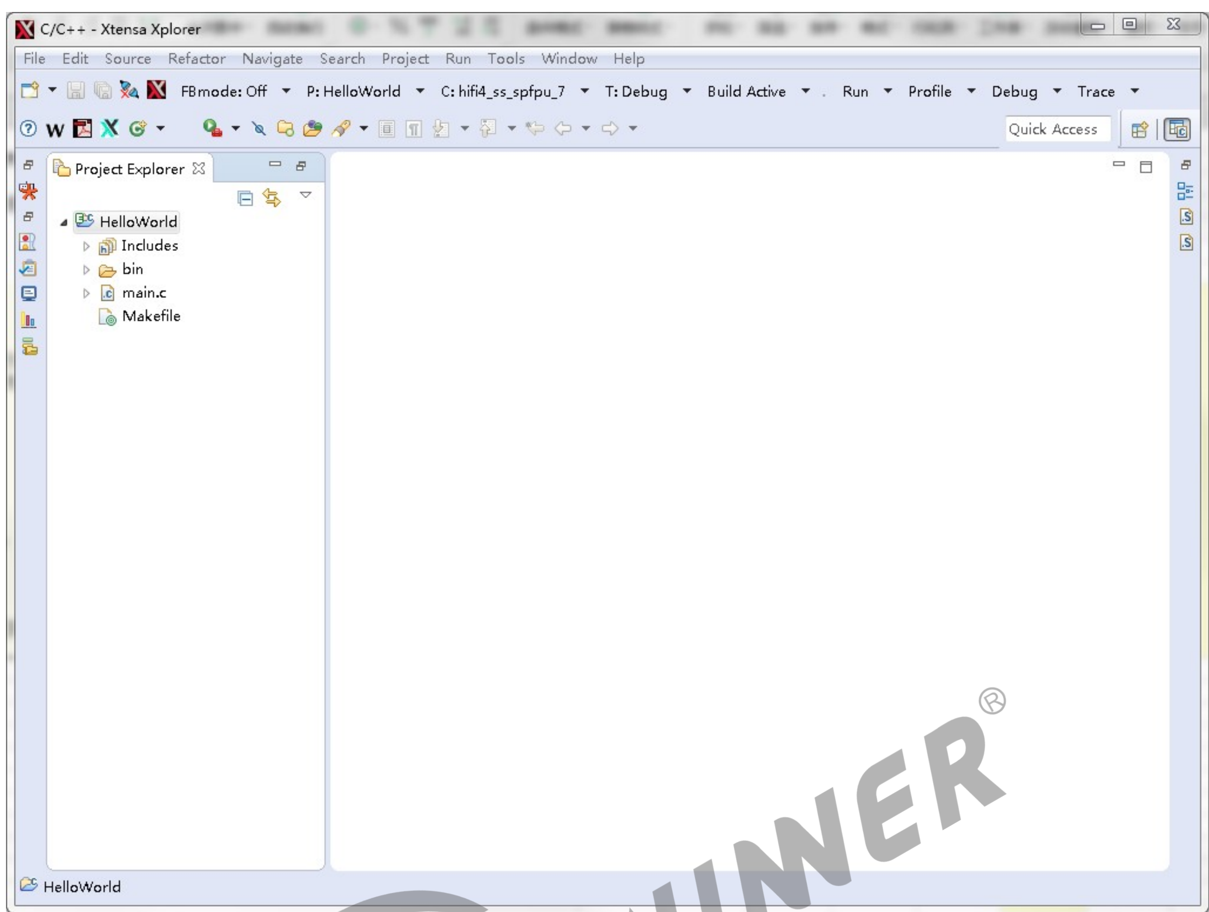

图 5-24: 配置 HelloWorld 工程步骤 3

这样就新建了一个 HelloWorld demo 工程，接下来编译工程，选择 active project 为 HelloWorld，选择 hifi4_ss_spfpu_7，然后点击 Build Active 进行编译：

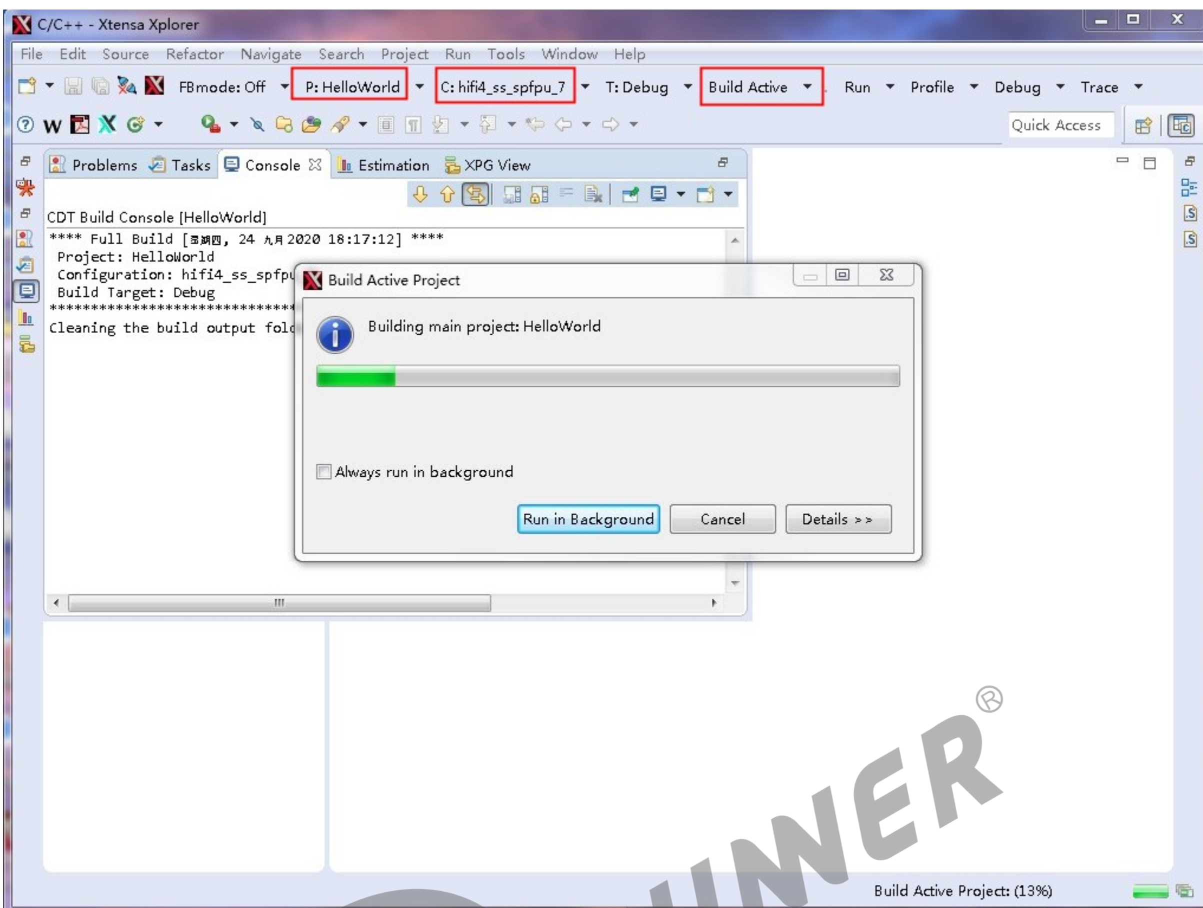

图 5-25: 编译 HelloWorld 工程

编译成功后就可以看到生成的 HelloWorld 程序：

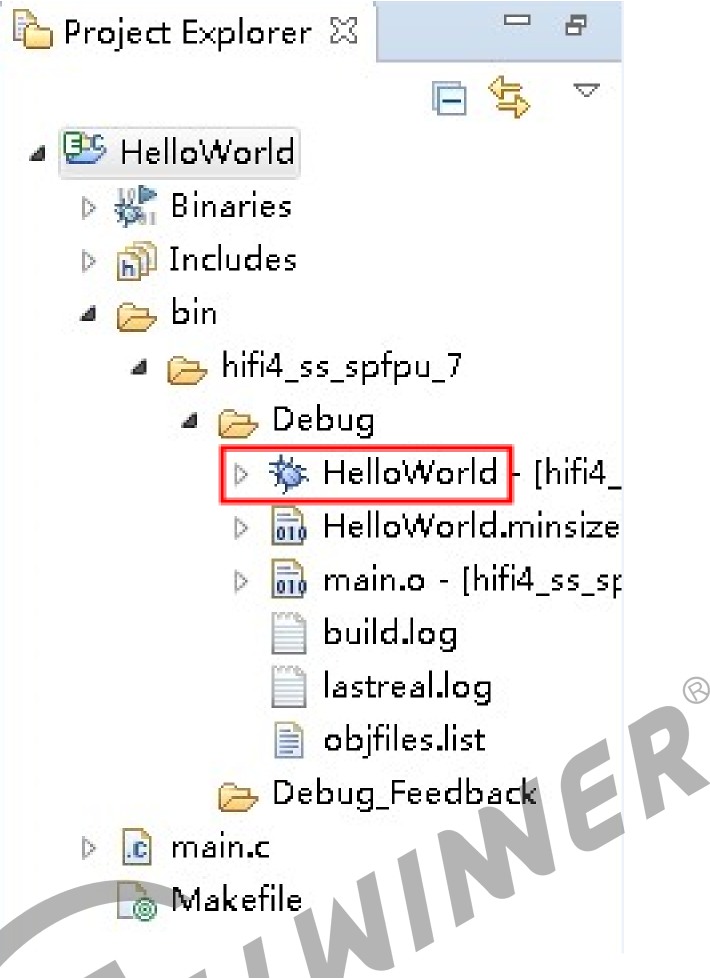

图 5-26: 生成 HelloWorld 程序

至此，我们的 HelloWorld 调试工程就搭建完成了，可以进行 JLink 调试。

5.2.2 Jtag 调试

调试器用的是 JLink，需要将调试器与板子的 Jtag 口连接，并将相应的 pin 的 pinmux 配置成 Jtag 功能。

5.2.2.1 配置

进行调试前，需要先进行调试选项的配置，点击 Debug 按钮的下拉，选择 Debug Configurations：

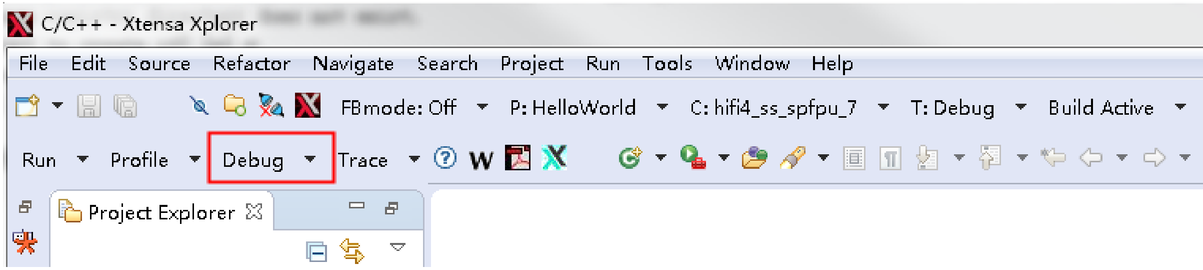

图 5-27: 配置 Debug 选项

在 “Xtensa On Chip Debug” 下新建 core0 配置，如下：

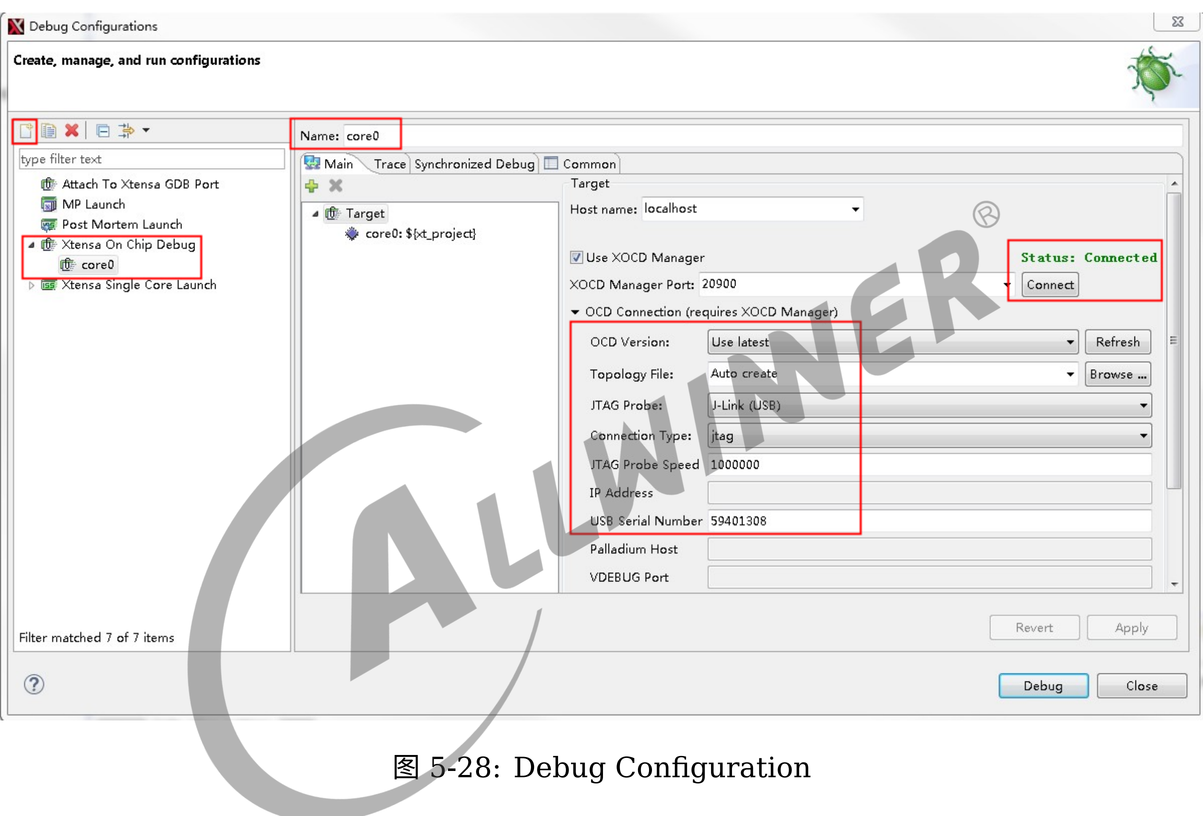

图 5-28: Debug Configuration

其中 JLink 的 USB 序列号可通过 J-Link Commander 获取：

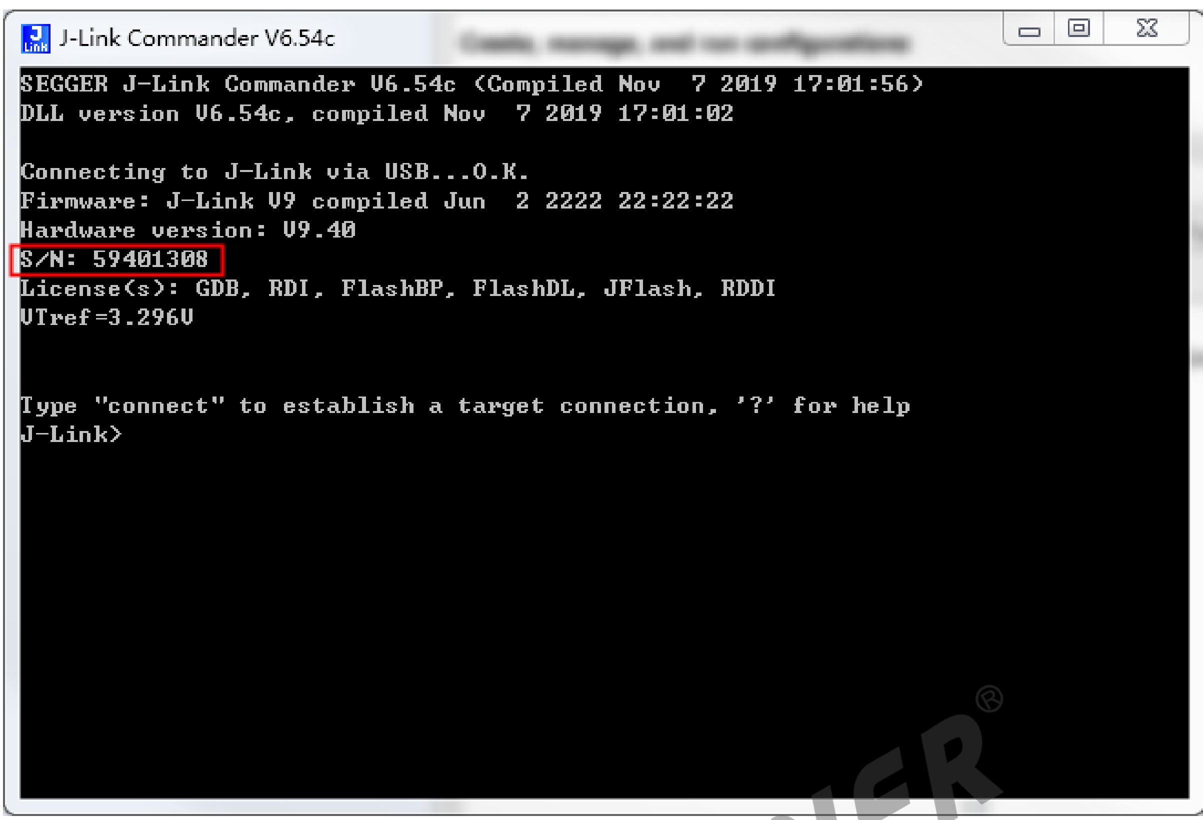

图 5-29: J-Link Commander

然后点击新建的 core0 config，进行进一步配置，这里基本上用默认配置即可：

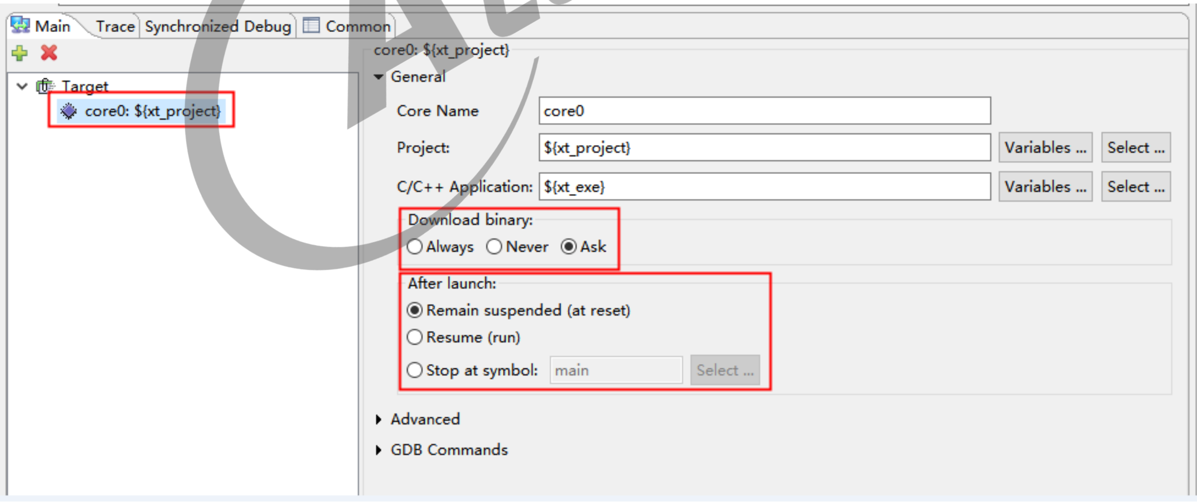

图 5-30: core0 config

某些平台是双核 DSP，如果需要双核同时 debug，则参照 core0，再新建一个 core1 config 即可：

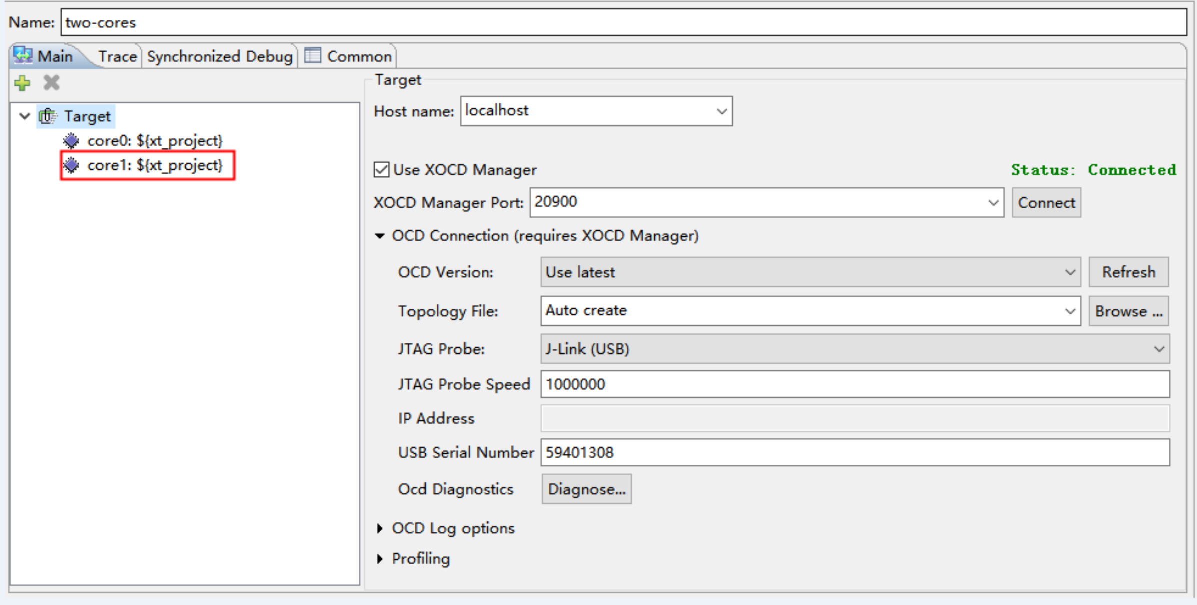

图 5-31：two cores config

5.2.2.2 探测

配置完成后，调试前我们先进行探测（Diagnose），如下：

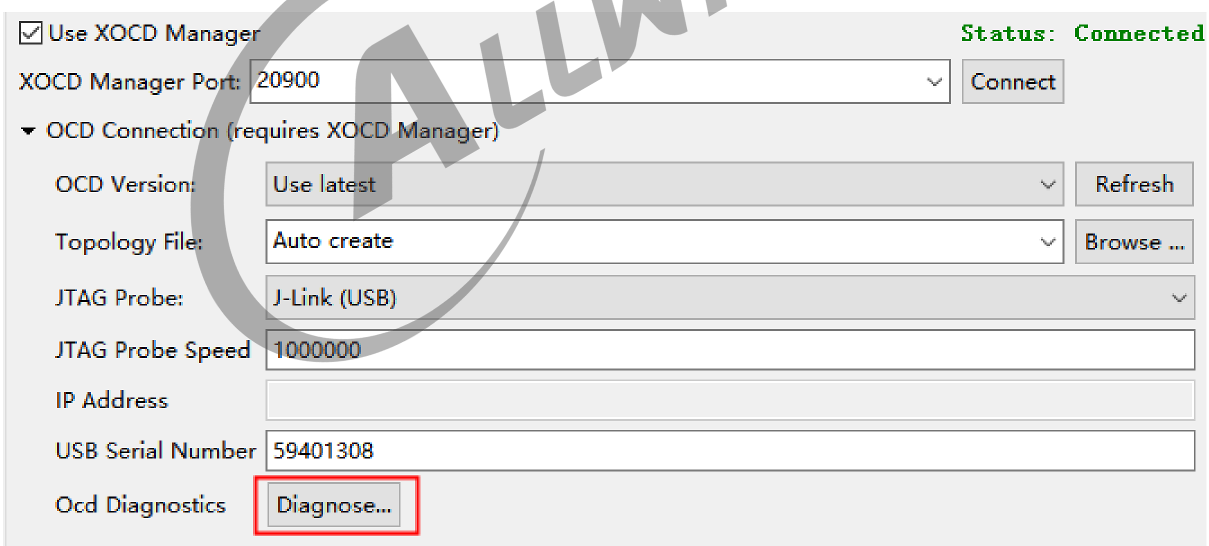

图 5-32：Diagnose

主要用于确认配置及 JLink 连接是否正常。若成功，则会返回当前活动的 DSP 核的情况，如：

ALLWINNER

OCD Diagnostic

OCD Diagnostic

Press 'Query' button for the diagnosis

☑ Query probes

☑ Query scanchain ☑ Advance diagnostic

Query results:

```
*
*
*************************************************************************


*************************************************************************
*                       Diagnosis of scanchain:
*
*                  --- Controller 0 ( 1 devices) ---
*
*         Device                 IR length                Idcode
*           0                        5                 0x120034e5
*
*
*************************************************************************
```

Query   Close

图 5-33: Diagnose one core

图 5-34: Diagnose two cores

5.2.2.3 选择 LSP 文件

在 Linux 中 DSP 工程，仿真和正常启动选用 LSP 文件是不一样的，需要手动配置

首先清楚编译环境，进入 menuconfig

```
make clean

make menuconfig
```

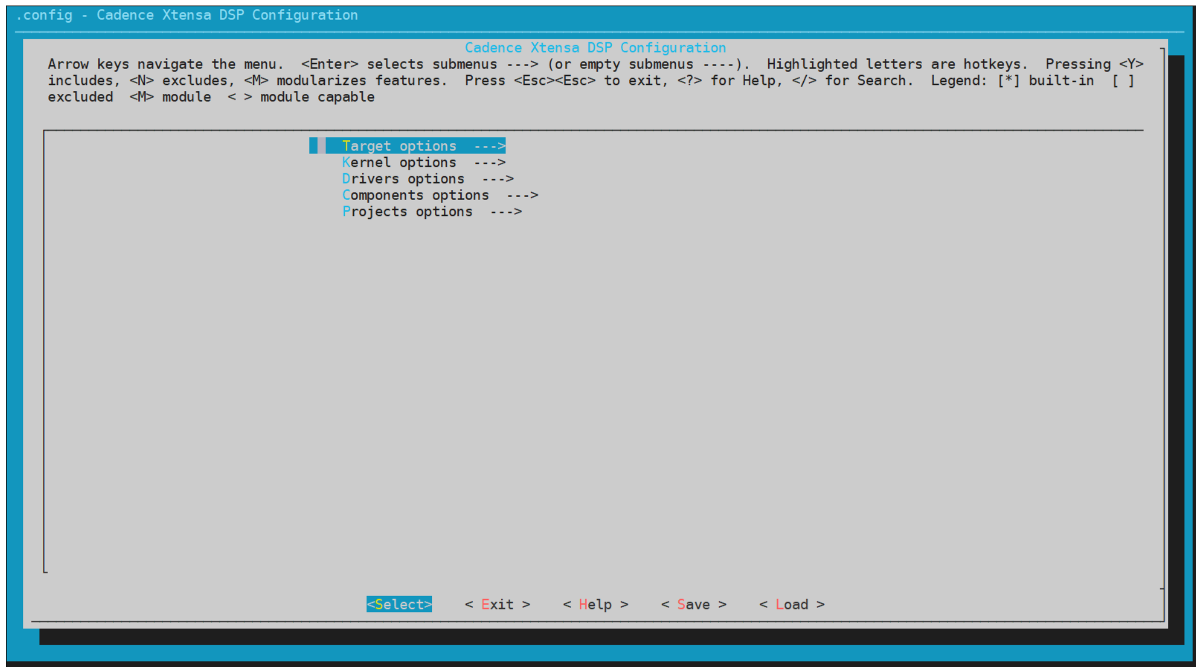

图 5-35: 选择 debug_lsp 步骤 1

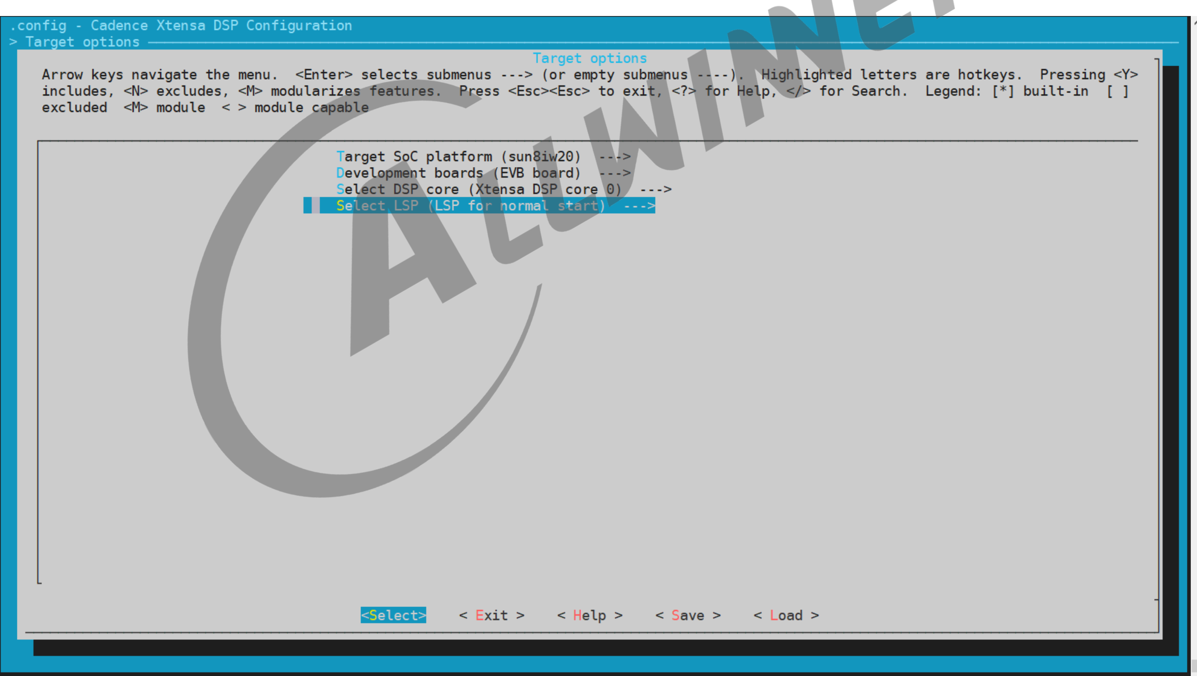

图 5-36: 选择 debug_lsp 步骤 2

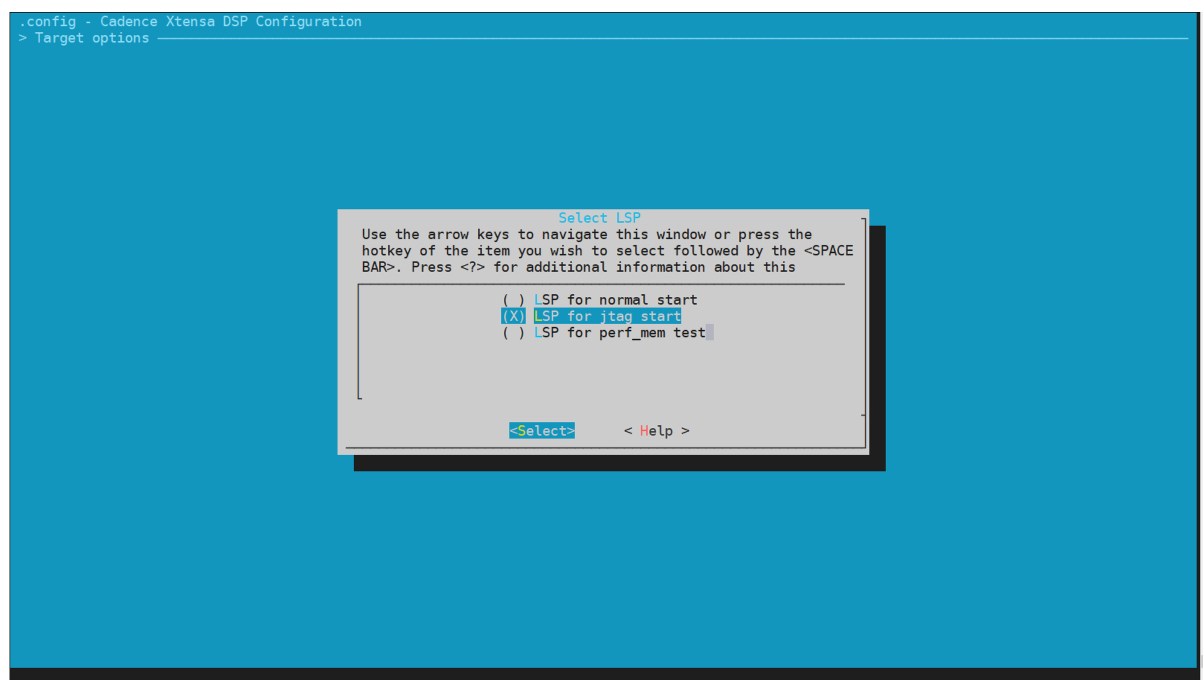

图 5-37: 选择 debug_lsp 步骤 3

最后重新编译，生产 dsp.elf

5.2.2.4 elf 文件替换

仿真时候，真正用到的 DSP 二进制 elf 是在 Linux 编译环境编译出来的 elf。我们需要把 Linux 中 DSP 工程编译出来 dsp.elf 替换掉 HelloWorld，再进行 debug

```
<workspace root>\Xplorer-8.0.13-workspaces\workspace\HelloWorld\bin\hifi4_ss_spfpu_7\Debug

HelloWorld ---- 替换成DSP的elf（注：binary名称需保留用“HelloWorld”）
```

5.2.2.5 单核调试

Diagnose 正常后，就可以点击 Debug 按钮进行 Jtag 调试：

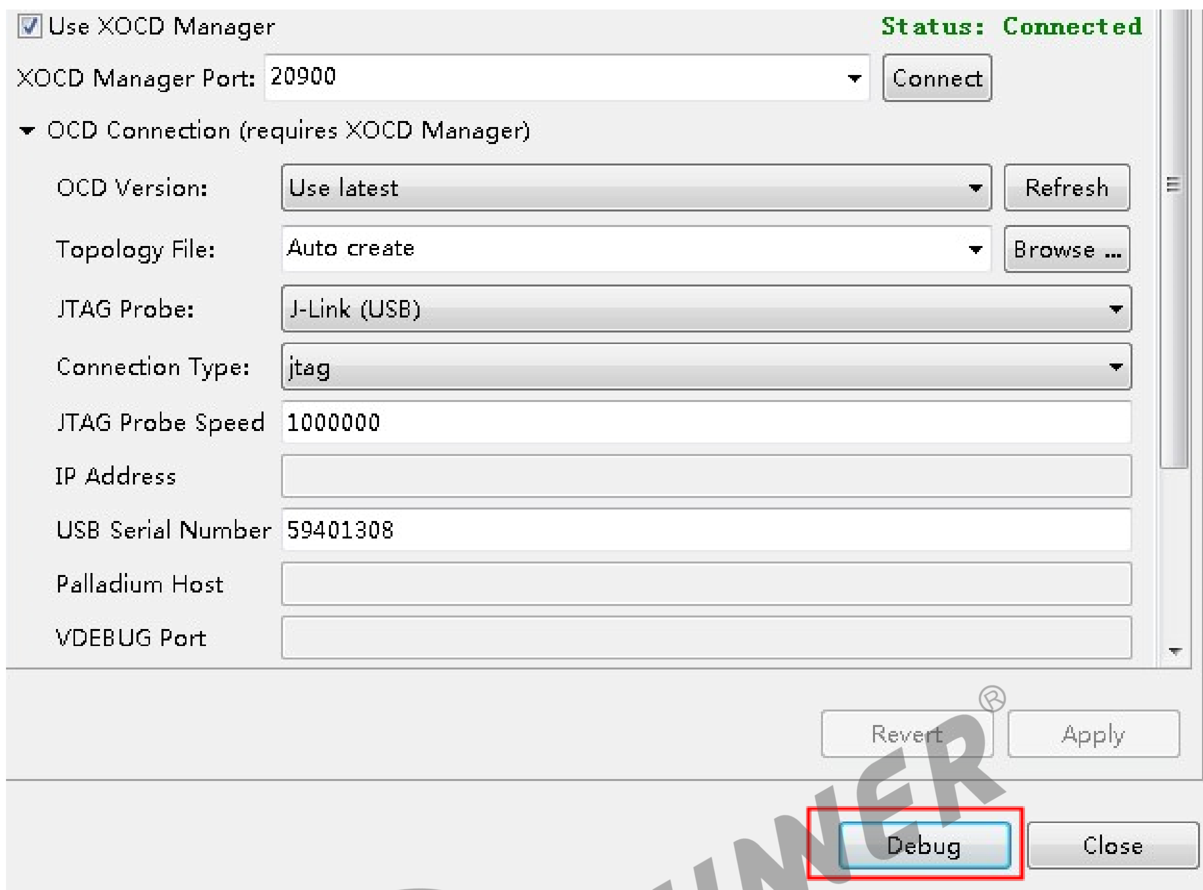

图 5-38: 开始 Debug

若上述的 core config 中的 “Download binary” 选择了 “Ask”，则会弹窗提示是否要下载二进制程序。

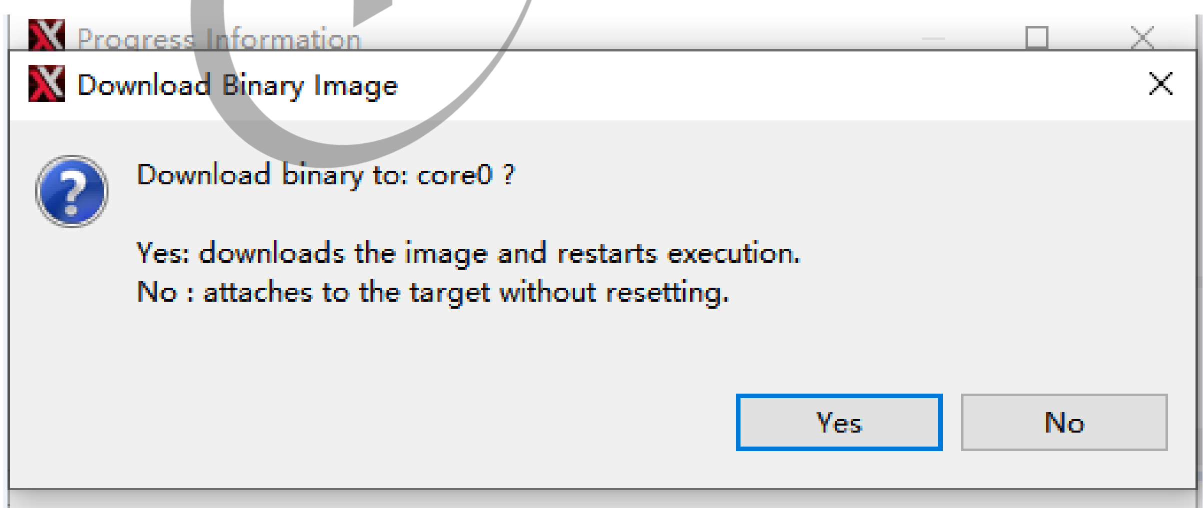

图 5-39: Download Binary Image

若选择 “No”，则仅进行 Jtag 连接；若选择 “Yes”，则先下载二进制程序，并复位 DSP 从 reset vector 重头开始跑起。

成功连接 Jtag 后，则会看到进程及堆栈信息（注：若仅进行 Jtag 连接而并未下载 elf，则需要导入符号表才能看到具体的函数名，跟 ARM DS-5 类似）：

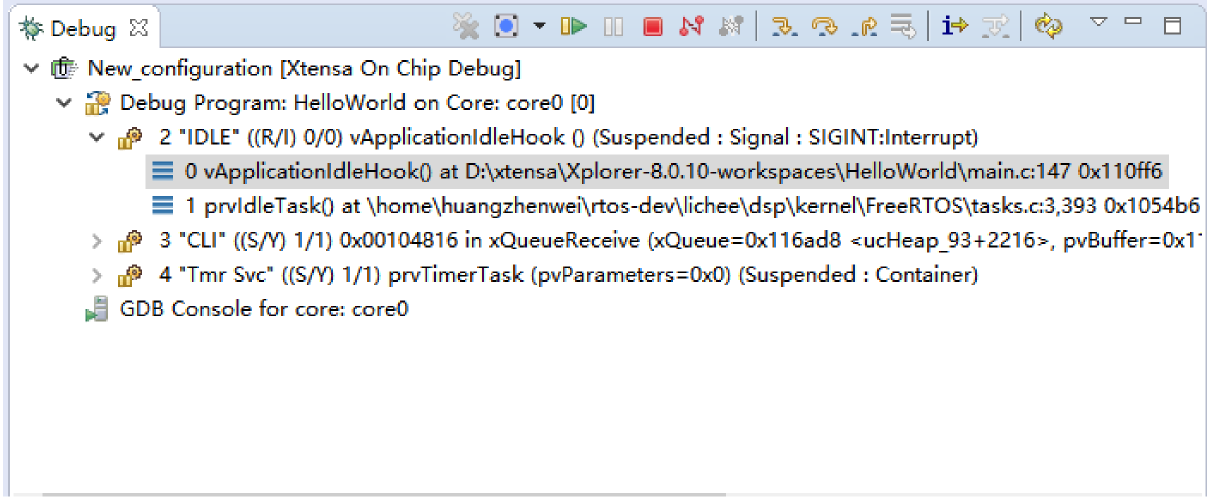

图 5-40: Core info

5.2.2.6 双核调试

双核调试跟单核调试操作类似，这里不重复叙述，成功连接后会显示两个核的信息：

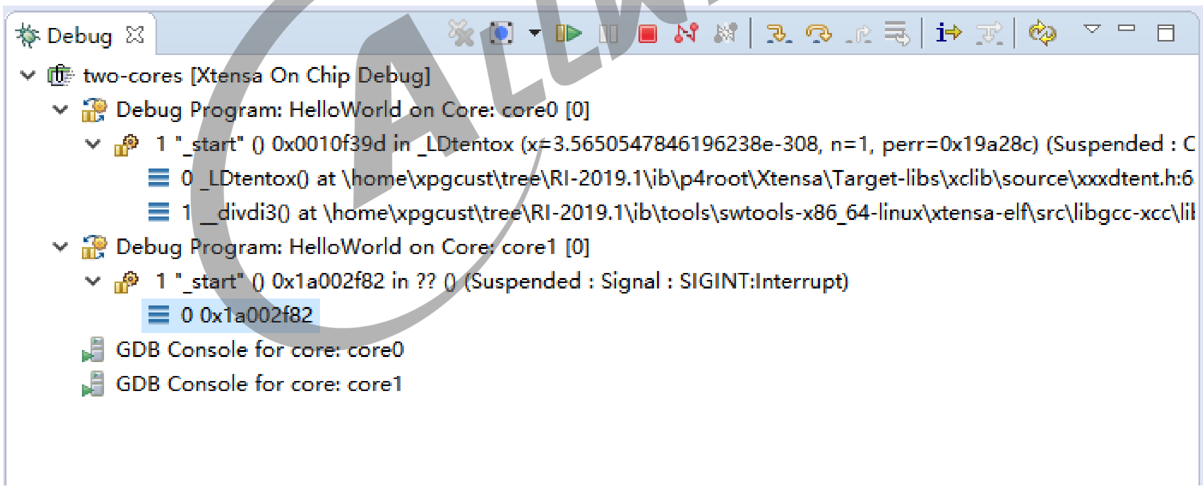

图 5-41: Two Cores info

5.2.2.7 常用调试窗口

连接成功后就可以进行常规的 debug，如设置断点、查看内存、查看寄存器等，都可以找到相应的调试窗口，这里不一一赘述，需要自行摸索：

版权所有 © 珠海全志科技股份有限公司。保留一切权利

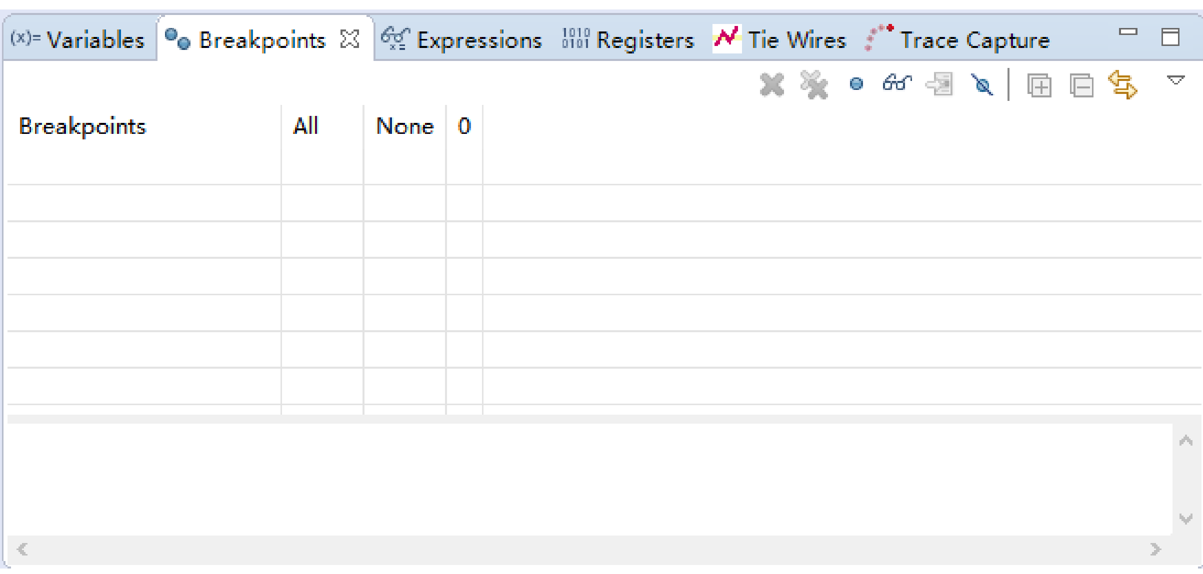

图 5-42: Debug Windows

5.2.2.8 设置源文件路径

当进行仿真时候，无法看到对应源文件，需要进行源文件路径设置 (特别注意的是 main.c 文件依然无法查找):

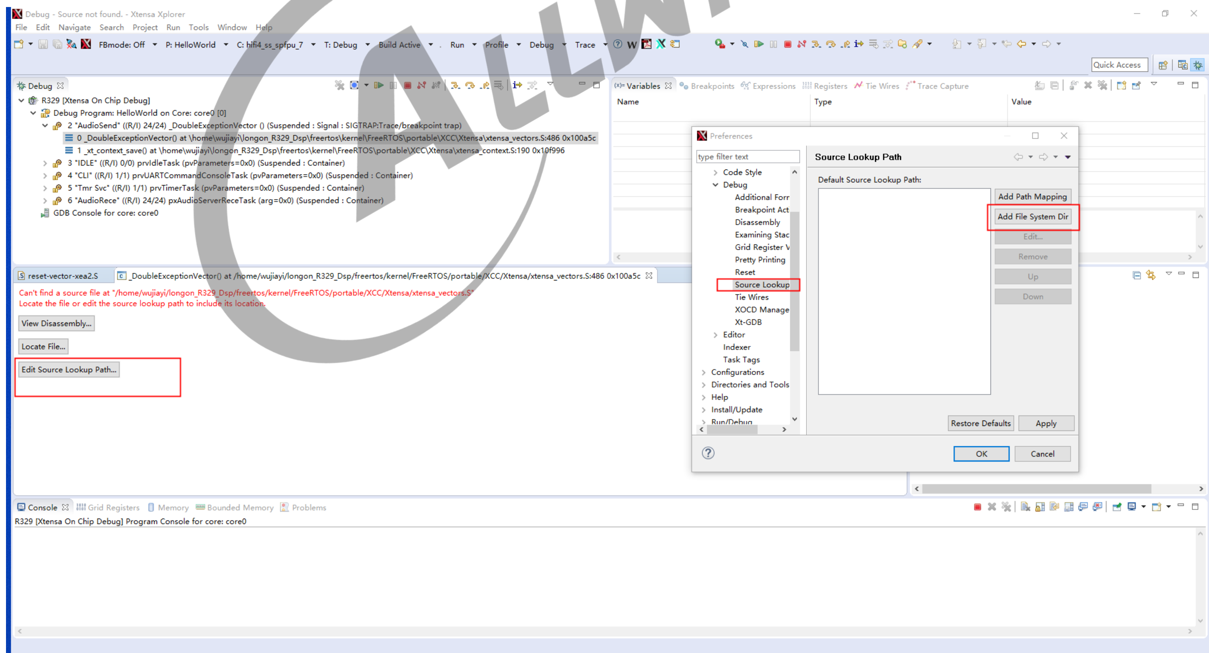

图 5-43: Source Lookup

设置好以后，查看 Navigator

ALLWINNER

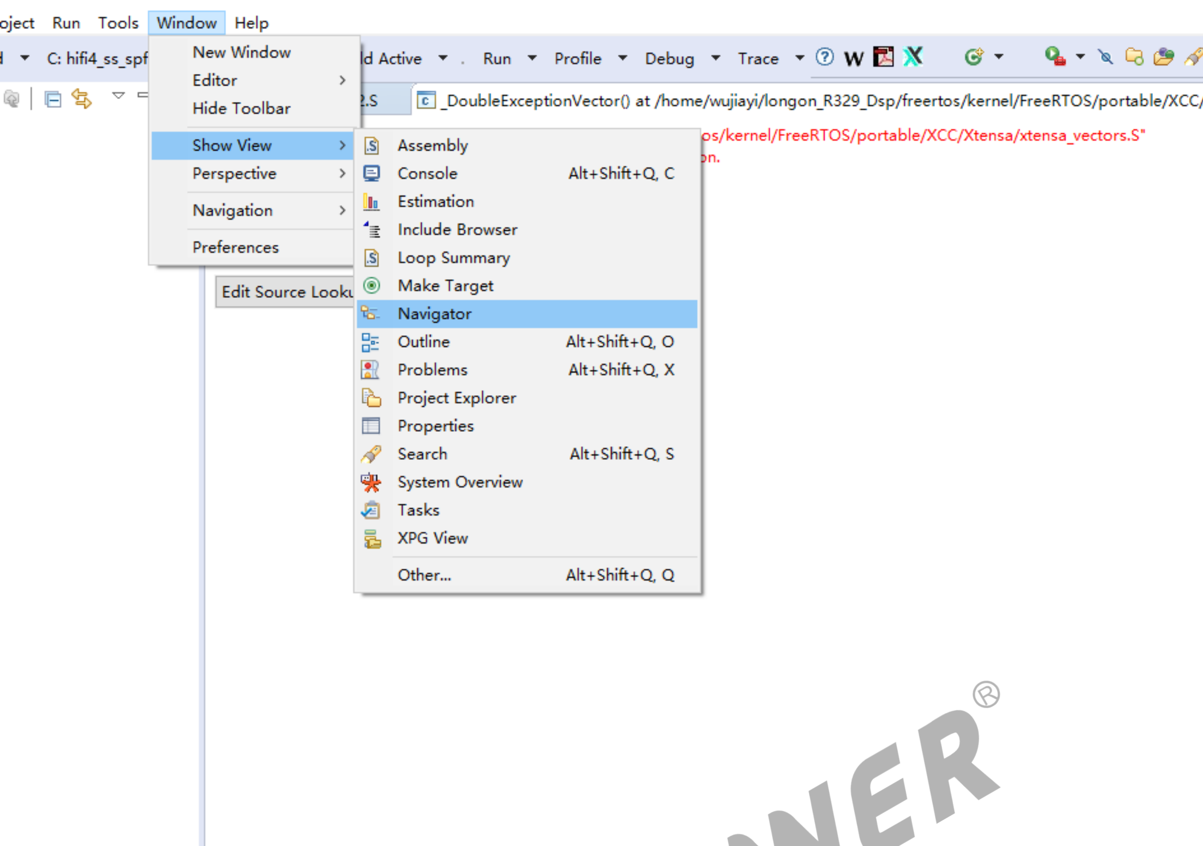

图 5-44: Show Navigator

发现会自动创建了一个 libc_debug_srclink 工程，用于 debug 时候源文件查找，同时可以在这个工程中添加所需源文件路径

ALLWINNER

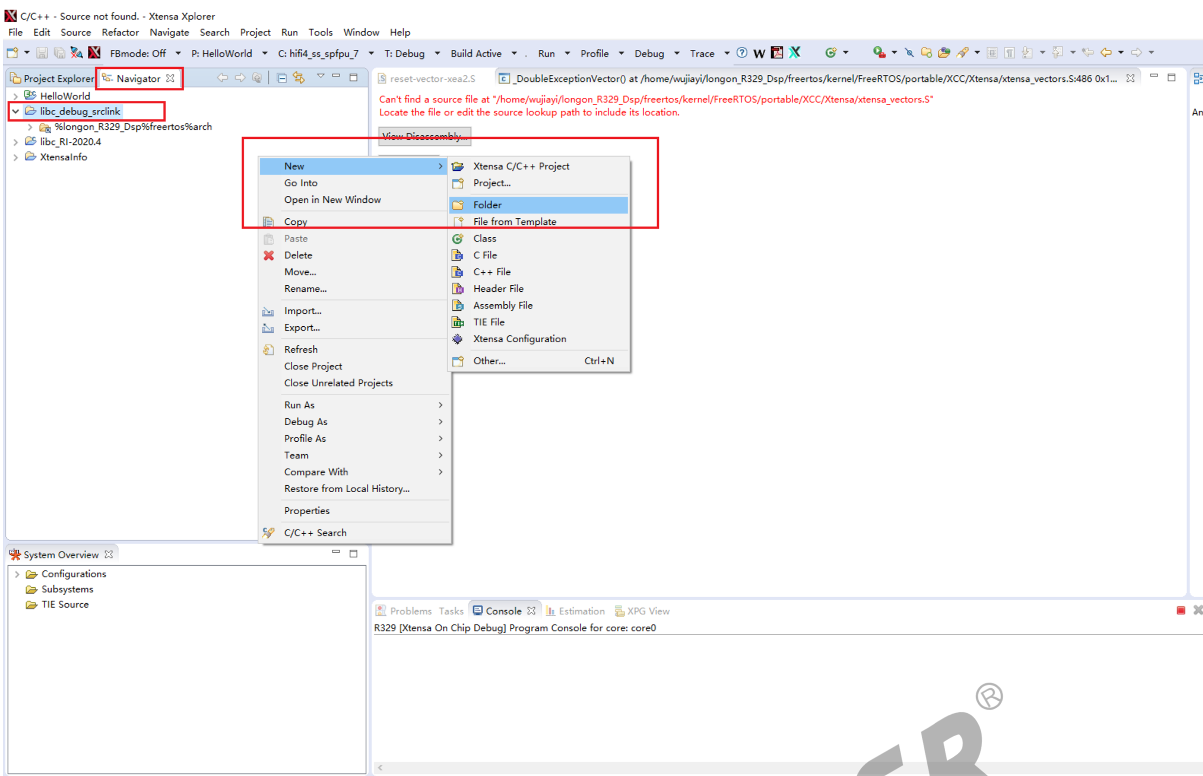

图 5-45: Create Folder

点击 Advanced ，选择 Linked Folder 方式打开源文件路径:

图 5-46: Create Folder

设置 OK 后, 就可以愉快的仿真，操作 (查看/打断点) 对应.c 文件

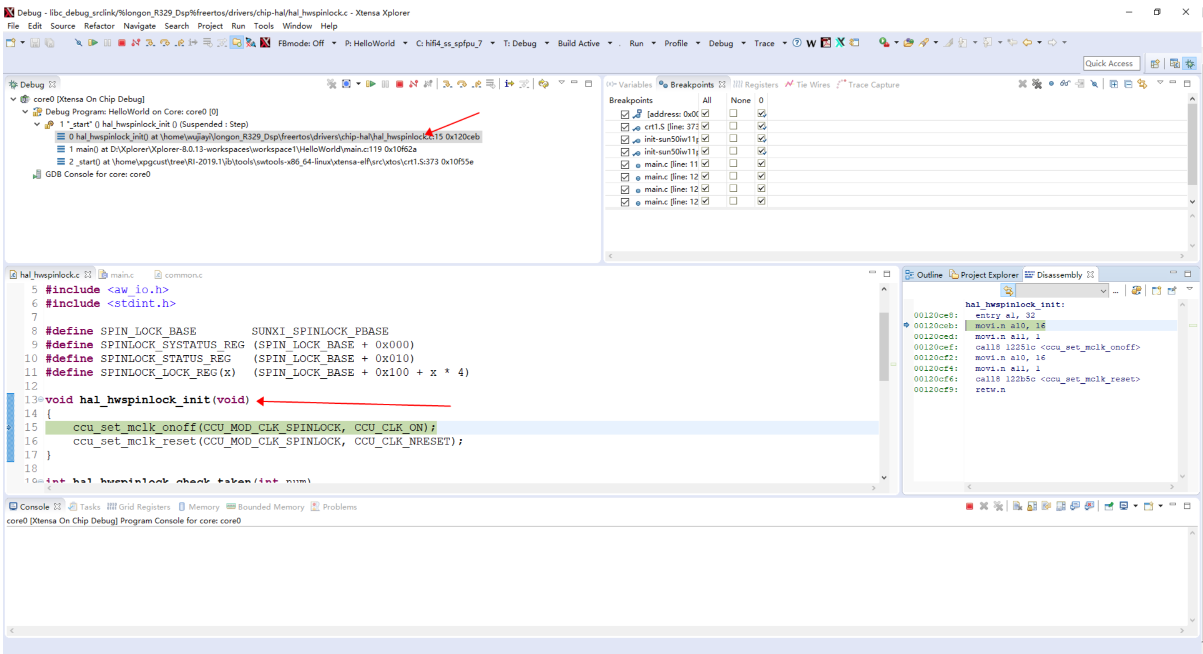

图 5-47: Src file

6 系统

6.1 DSP 系统

6.1.1 LSP 文件

6.1.1.1 LSP 作用

根据官方文档《lsp_rm.pdf》描述，LSP 文件本质上就是 Xtensa 定义的链接脚本，指明代码和数据链接加载位置，一般 LSP 文件存在 ldscripts 文件夹下。

例如在 SDK 目录下:

```
<root>/arch/sun8iw20/lsp/dsp0/ldscripts
```

存在 4 个 LSP 文件，这里 SDK 默认使用 elf32xtensa.x

6.1.1.2 LSP 生成

LSP 文件一般情况下由以下 3 个文件生成:

```
1. 内存映射规则（memmap.xmm文件）;
2. specs文件，描述所需要链接的目标文件和库文件;
3. 链接所需的目标文件和库文件;
```

执行如下命令，重新生产 LSP

```
xt-genldscripts -b <lspDir>
```

例如:

```
进入目录 <root>/arch/sun8iw20/lsp/dsp0/
执行命令 xt-genldscripts -b dsp0
```

重新生成会显示如下信息:

```
New linker scripts generated in dsp0/ldscripts
```

6.1.2 memmap 文件

这里以一个基本的内存映射规则 memmap.xmm 进行分析，如下所示：

```
/*该脚本只用到sram*/
BEGIN sram
0x100800: sysram : sram : 0x1cf800 : executable, writable ;
/* 0x100800 表示这片内存起始地址
*  sysram 表示这片内存是系统ram,还有其他属性例如 sysrom
*  sram 表示这片内存的名字
*  0x1cf800 表示这片内存的大小
*  executable, writable 表示这片内存的属性是代码可以执行，数据可以写。
*/

 sram0 : F : 0x100800 - 0x100bff : .ResetVector.text .ResetHandler.literal .ResetHandler.
    text;
/* sram0 表示sram内存里面的一小片内存的名称
*  F 表示sram0是不可更改区域，一般用于存放中断向量表或者异常向量
*  0x100800 - 0x100bff 表示这片内存的范围
*  .ResetVector.text .ResetHandler.literal .ResetHandler.text 表示这片内存存放的段
*/

 sram1 : F : 0x100c00 - 0x100d7b : .WindowVectors.text .Level2InterruptVector.literal;
 sram2 : F : 0x100d7c - 0x100d9b : .Level2InterruptVector.text .Level3InterruptVector.
    literal;
 sram3 : F : 0x100d9c - 0x100dbf : .Level3InterruptVector.text .DebugExceptionVector.
    literal;
 sram4 : F : 0x100dc0 - 0x100dfb : .DebugExceptionVector.text .NMIExceptionVector.literal;
 sram5 : F : 0x100dfc - 0x100e1b : .NMIExceptionVector.text .KernelExceptionVector.literal;
 sram6 : F : 0x100e1c - 0x100e3b : .KernelExceptionVector.text .UserExceptionVector.literal
    ;
 sram7 : F : 0x100e3c - 0x100e5b : .UserExceptionVector.text .DoubleExceptionVector.literal
    ;
 sram8 : F : 0x100e5c - 0x2cffff : STACK : HEAP : .DoubleExceptionVector.text .sram.rodata
    .rodata .sram.literal .literal .sram.text .text .sram.data .data .sram.bss .bss;
END sram
```

在 D1 工程，memmap.xmm 书写规则可以参考：

```
<root>/arch/sun8iw20/lsp/dsp0/memmap.xmm
```

6.1.3 启动

6.1.3.1 DSP 复位入口

根据内存映射规则 memmap.xmm，路径为:

```
<root>/arch/sun8iw20/lsp/dsp0/memmap.xmm
```

DSP 复位入口为 0x400660, 存放着段.ResetVector.text，中断向量表首地址 0x401000

```
VECRESET=0x400660
VECSELECT=0x1
VECBASE=0x401000

BEGIN iram0
    0x400000: instRam : iram0 : 0x10000 : executable, writable ;
    iram0_0 : F : 0x400000 - 0x40065f : .oemhead.text .oemhead.literal;
    iram0_1 : F : 0x400660 - 0x400fff : .ResetVector.text .ResetHandler.literal .
    ResetHandler.text;
    iram0_2 : F : 0x401000 - 0x40117b : .WindowVectors.text .Level2InterruptVector.literal;
    iram0_3 : F : 0x40117c - 0x40119b : .Level2InterruptVector.text .Level3InterruptVector.
    literal;
    iram0_4 : F : 0x40119c - 0x4011bb : .Level3InterruptVector.text .DebugExceptionVector.
    literal;
    iram0_5 : F : 0x4011bc - 0x4011db : .DebugExceptionVector.text .NMIExceptionVector.
    literal;
    iram0_6 : F : 0x4011dc - 0x4011fb : .NMIExceptionVector.text .KernelExceptionVector.
    literal;
    iram0_7 : F : 0x4011fc - 0x40121b : .KernelExceptionVector.text .UserExceptionVector.
    literal;
    iram0_8 : F : 0x40121c - 0x40123b : .UserExceptionVector.text .DoubleExceptionVector.
    literal;
    iram0_9 : F : 0x40123c - 0x40ffff : .DoubleExceptionVector.text .iram0.literal .iram0.
    text;
END iram0

BEGIN ddr1
0x32000000: sysram : ddr1 : 0x100000 : executable, writable ;
ddr1_0 : C : 0x32000000 - 0x320fffff : STACK : HEAP: .ddr1.rodata .rtos.rodata .rodata .
    clib.data .clib.percpu.data .rtos.percpu.data .rtos.    data .FSymTab .stubTab .ddr1.
    data .data .literal .rtos.literal .clib.literal .clib.text .rtos.text .text .clib.
    percpu.bss .rtos.percpu.bss .r    tos.bss .ddr1.bss .bss;
END ddr1
```

6.1.3.2 _ResetHandler 复位函数

_ResetHandler 复位函数链接到.ResetVector.text 段中, 对应文件 reset-vector-xea2.S，最终调用 _start 函数

版权所有 © 珠海全志科技股份有限公司。保留一切权利

6.1.3.3 _start 函数

_start 函数位于 crt1.S(配置好 C 运行环境, 例如配置 stack、清 bss 段、调用 board_init，调用 __clibrary_init 等等) , _start 函数先调用 board_init , 进行板级初始化, 接着调用 main 函数 (由开发者实现)。

我们需要关注的是 board_init 和 main。board_init 顾名思义就是板级初始化操作，这个需要由用户自行定义，针对 D1，位于：

```
<root>/arch/sun8iw20/init-sun8iw20.c:
```

main 则是用户程序的入口点，上述也提到，由每个 project 自行实现, 位于:

```
<root>/projects/d1/src/main.c
```

6.1.4 cache 控制器

关于 cache 操作，当我们使用的内存是支持 cache 属性的时候，原则上需要注意 cache 的 invalidate 和 writeback 操作：

```
xthal_icache_all_invalidate()      - Invalidate entire instruction cache
xthal_dcache_all_invalidate()      - Invalidate entire data cache (and L2 cache)
xthal_dcache_all_writeback()       - Writeback entire data cache (and L2 cache)
xthal_dcache_all_writeback_inv()   - Writeback and invalidate dcache (and L2 cache)
xthal_icache_region_invalidate()   - Invalidate range of addresses from instruction cache
xthal_dcache_region_invalidate()   - Invalidate range of addresses from data cache (and L2
    )
xthal_dcache_region_writeback()    - Writeback range of addr. from data cache (and L2)
xthal_dcache_region_writeback_inv() - Writeback and invalidate range of addr. from data
    cache (and L2)
```

6.1.5 中断

6.1.5.1 中断等级

1. Low-Level Interrupts 处理中断速度较慢，中断会跳转到 UserExceptionVector 或者 KernelExceptionVector
2. Mid-Level Interrupts 处理中断速度比 Low-Level Interrupts 要快，命名规则是 Level2InterruptVector, Level3InterruptVector
3. High-Level Interrupts 最快的中断（具有最低的延迟），用汇编代码书写

用户主要关注 Low-Priority Interrupts 和 Medium-Priority Interrupts，这两个优先级的中断处理函数可以用 C 实现。

以 Allwinner 的 Xtensa core configuration 为例，结合 freeRTOS Xtensa port 的实现，梳理出中断处理流程如下图：

图 6-1: Interrupt processing

6.2 FreeRTOS 系统

6.2.1 FreeRTOSConfig.h 配置

本章节描述一些常用和关键的宏定义

6.2.1.1 调度器

```
/* 置 1： RTOS 使用抢占式调度器; 置 0： RTOS 使用协作式调度器
 *
 * 在多任务管理机制上，操作系统可以分为抢占式和协作式两种。
 * 抢占式: 高优先级任务可以抢占低优先级任务
 * 协作式: 是任务主动释放 CPU 后，切换到下一个任务。
 */

#define configUSE_PREEMPTION            1

/* 置 1： 时间片轮询
 *
 * 处于就绪态的多个相同优先级任务将会以时间片切换的方式共享处理器
 */

#define configUSE_TIME_SLICING 1
```

6.2.1.2 系统节拍

```
/*
 * 原本是指,写入实际的 CPU 内核时钟频率，也就是 CPU 指令执行频率，通常称为 Fclk
 * Fclk 为供给 CPU 内核的时钟信号，我们所说的 cpu 主频为 XX MHz，
 * 就是指的这个时钟信号，相应的， 1/Fclk 即为 cpu 时钟周期;
 * 由于DSP使用timer0作为系统节拍时钟,这里是指timer0的主频
 */

#define configCPU_CLOCK_HZ              2000000

/* FreeRTOS 系统节拍中断的频率
 * 表示操作系统每 1 秒钟产生多少个tick， tick 即是操作系统节拍的时钟周期
 * 目前是1S有50tick,即任务调度20ms进行一次
 */

#define configTICK_RATE_HZ              ( 50 )
```

6.2.1.3 任务优先级

```
/* 某些运行 FreeRTOS 的硬件有两种方法选择下一个要执行的任务:
 * 通用方法和特定于硬件的方法（以下简称“特殊方法”）。
 *
 * 通用方法:
 * 1.configUSE_PORT_OPTIMISED_TASK_SELECTION 为 0 或者硬件不支持特殊方法。
 * 2.可以用于所有 FreeRTOS 支持的硬件
 * 3.完全用 C 实现(主要查表)，效率略低于特殊方法。
 * 4.不强制要求限制最大可用优先级数目
 *
 */
```

```
#define configUSE_PORT_OPTIMISED_TASK_SELECTION 0

/*
 * 任务优先级最大值为25，可以选择标号0-24，标号越大任务优先级越高
*/

#define configMAX_PRIORITIES            ( 25 )
```

6.2.1.4 任务堆

```
/*
 * 任务堆大小，要设当调整大小，该值过小，会导致创建任务失败
*/

#define configTOTAL_HEAP_SIZE           ( ( size_t ) (384 * 1024) )
```

6.2.1.5 中断管理

```
/*
 * 用于中断的屏蔽，假如中断标号比configMAX_SYSCALL_INTERRUPT_PRIORITY小,则这些中断不受freeRTOS管理,
    调用关中断API不可以屏蔽这些中断
*/

#define configMAX_SYSCALL_INTERRUPT_PRIORITY    XCHAL_EXCM_LEVEL
```

6.2.2 内存管理

当前内存分配有如下两种方式：

```
1. C库的malloc/free(当前使用的是默认的Xtensa的C库——xclib)，堆的起始跟结束地址由链接脚本指定;
2. freeRTOS MemMang heap_1/2/3/4/5的实现 pvPortMalloc/vPortFree，若采用heap_3，则是封装了C库的
    malloc/free，跟1描述一致；其余实现的堆则用静态分配方式分配;
```

heap_x 内存分配方式比较：

```
heap_1 - 最简单，不允许释放内存;
heap_2 - 允许释放内存，但不能合并相邻的空闲块;
heap_3 - 简单包装标准的malloc()和free()以确保线程安全;
heap_4 - 合并相邻的空闲块以避免碎片，包括绝对地址放置选项;
heap_5 - 按照heap_4，具有跨多个不相邻的内存区域扩展堆的能力;
```

目前默认选择 heap_4 内存管理模式。

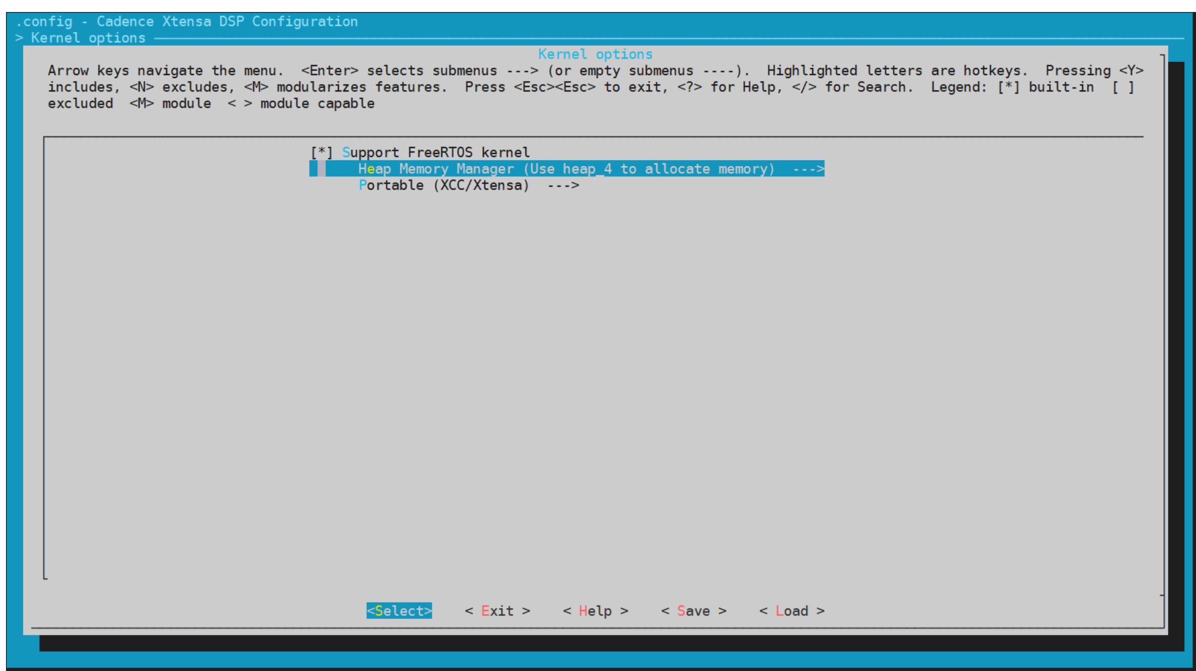

图 6-2: heap 选择

7 调试组件

7.1 串口调试

一般情况下，会预留一个串口给 DSP 作为调试用于，在命令终端输入 help，会输出可以用到命令行

```
------> pxAudioServerReceTask Start <------
FreeRTOS command server.
Type Help to view a list of registered commands.

>help
Lists all the registered commands

[       help-built-in]--------------Lists all the built-in registered commands

[          task-stats]--------------
task-stats:
 Displays a table showing the state of each FreeRTOS task


[   echo-3-parameters]--------------
echo-3-parameters <param1> <param2> <param3>:
 Expects three parameters, echos each in turn


[     echo-parameters]--------------
echo-parameters <...>:
 Take variable number of parameters, echos each in turn


[                  md]--------------
md <start_addr> <len> :
 Dump memory from <start_addr> to <start_addr + len - 1>


[                  mw]--------------
mw <addr> <value> :
 Write <value> to <addr>


[              aplay]--------------playback test

[            arecord]--------------record test

[            aduplex]--------------duplex test

[             amixer]--------------mixer test

[          jtag_init]--------------Init S_JTAG for DSP debugging

[          dsp1_power]--------------DSP1 power control

[          dumpregion]--------------"dump reg when suspend."

[             dspfreq]--------------"set/get dsp freq."

[            perf_mem]--------------Memory access performance test

[                help]--------------List all registered commands

[                free]--------------Free Memory in Heap

[                 asi]--------------algorithm sampel install

[          wakeupdemo]--------------algorithm wakeup demo


[Press ENTER to execute the previous command again]
>
```

图 7-1：dsp_uart 输出

版权所有 © 珠海全志科技股份有限公司。保留一切权利

同时我们可以新增自己需要命令，利用宏 FINSH_FUNCTION_EXPORT_CMD 将命令添加到列表中, 具体流程可以参考以下目录命令的实现

```
<root>components/aw/testbench/
```

著作权声明

版权所有 ©2021 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明

（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。