计算机系统结构实验报告

—　操作系统的移植

学　　院　电子与信息工程学院

专　　业　计算机科学与技术

授课老师　郭玉臣

成　员 1　1852329 赵孟石

成　员 2　1853790 庄镇华

成　员 3　1852024 李兵磊

完成日期　2021.06.30

目录

一、 实验概要

1.1 实验内容

本次实验的要求是在 Windows10 操作系统下，利用 Vivado 集成环境，基于 XILINX 公司的 NEXYS4 DDR 板进行操作系统移植。

上一次实验完成了 89 条 MIPS 指令 CPU 的改造，并进行了实验结果正确性的验证。然而，虽然这一 CPU 已正确实现了最基本的 MIPS 指令，但对于移植操作系统来说，仍然缺少内存管理单元等移植操作系统必需的部件，无法真正实现操作系统的移植和兼容。

因此，本次实验中选用可以嵌入到 FPGA 中的 RISC 处理软核 MicroBlaze 嵌入式软核来进行操作系统的移植工作。移植的操作系统则选用了μC/OS -III 操作系统，这一操作系统架构简单，且可以支持 MicroBlaze 软核。

1.2 系统介绍

1.2.1 Micro Blaze 嵌入式软核

Micro Blaze CPU 是嵌入式、可修改预置 32 位/64 位 RISC 微处理器配置系列。系统设计者可利用 2019.2 中的 Vitis Core 开发套件或使用 2019.1 或更早版本中基于 Eclipse 的 Xilinx 软件开发套件 (SDK)，通过所选的评估套件启动 MicroBlaze 处理器的开发。MicroBlaze 处理器符合大量不同应用的需求，这些应用包括工业、医疗、汽车、消费类以及通信市场等。

其主要功能包括：

- ✧ 32 位指令集和通用寄存器
- ✧ 32 位地址总线，可扩展至 64 位
- ✧ 锁步和 TMR 功能
- ✧ 可选浮点单元
- ✧ 睡眠、休眠和暂停模式 / 指令

官方文档可参见 https://china.xilinx.com/products/design-tools/microblaze.html

1.2.2 μC/OS 操作系统

μC/OS -III 是源码公开的商用嵌入式实时操作系统内核，由著名的μC/OS -II 发展而来。μC/OS -III 针对以 ARM Cortex 为代表的新一代 CPU，面向带有可用于优先级查表的硬件指令(如前导零计算指令)的 CPU 的嵌入式应用。μC/OS -III 允许利用这类高端 CPU 的特殊硬件指令来实现高效的任务调度算法，而无须使用μC/OS -II 的软件任务调度算法，而且μC/OS -II 支持时间片轮转调度算法。从核心任务调度算法的改变来看，μC/OS -III 已经是一个全新的嵌入式 RTOS 内核。从μC/OS算起，该内核已有 20 余年应用史，在诸多应用领域得到了广泛的认可。

1.3 实验环境

1.3.1 硬件环境

Xilinx Nexys 4 Artix-7 开发板

1.3.2 软件环境

操作系统：Windows 10 (64-bit)

实验环境：Vivado v2019 (64-bit)

移植的操作系统：$\mu C/OS$ -III

CPU 软核 IP：Micro Blaze

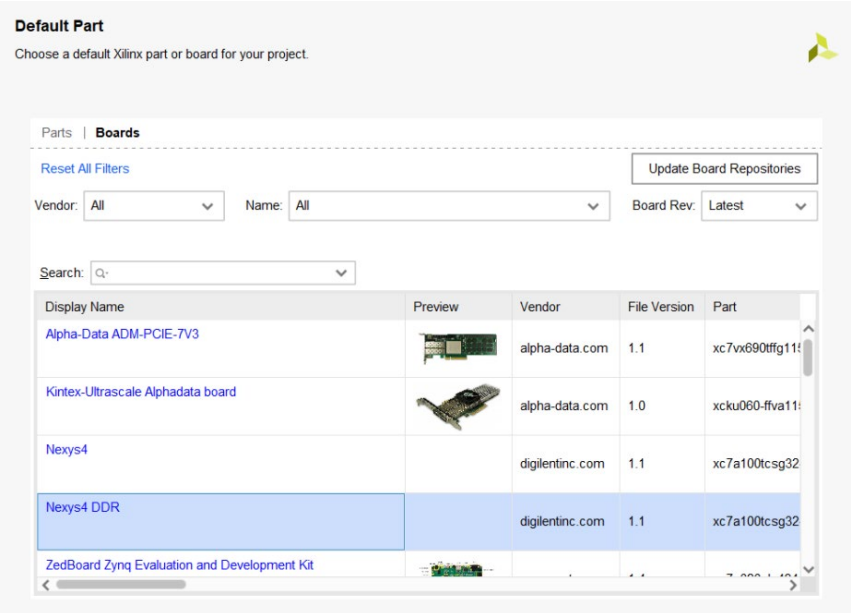
二、实验过程

2.1 安装板级开发包

传统的操作系统移植方法是利用交叉编译工具链进行交叉编译，最终挂载到相应的开发部件。本次综合实验不采取这种方式，而是采用 Vivado + Xilinx SDK + Micrium Xilinx BSP 的方式进行移植。

板级开发包是构建嵌入式操作系统所需的引导程序(Bootload)、内核(Kernel)、根文件系统(Rootfs)和工具链(Toolchain) 提供完整的软件资源包。它是介于主板硬件和操作系统中驱动层程序之间的一层，主要实现对操作系统的支持，为上层的驱动程序提供访问硬件设备寄存器的函数包，使之能够更好的运行于硬件主板。

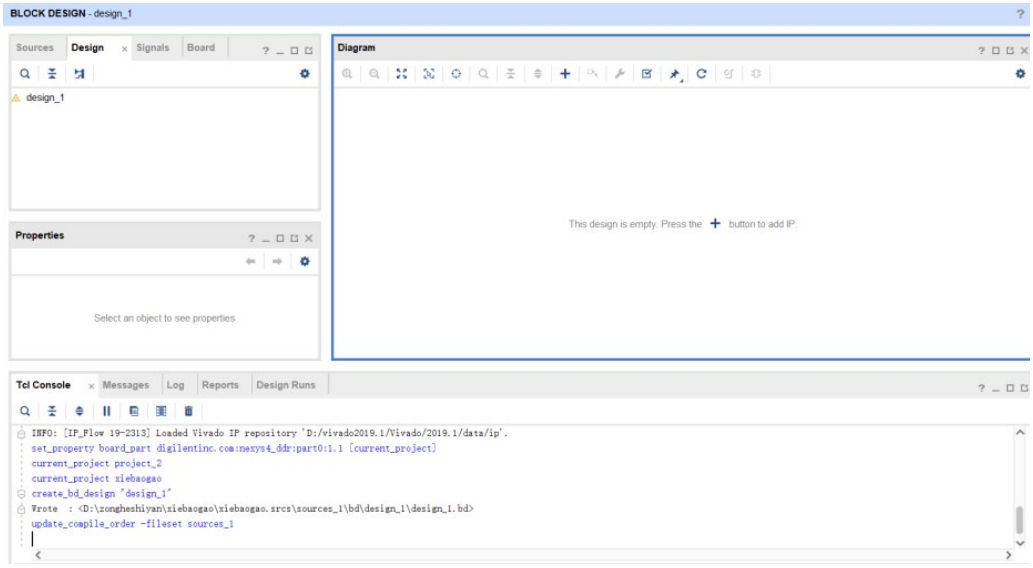
首先新建项目，类型选择 Boards，其中的 Nexys4 DDR 的板级开发包需要从官网 https://github.com/Digilent/vivado-boards 下载安装。



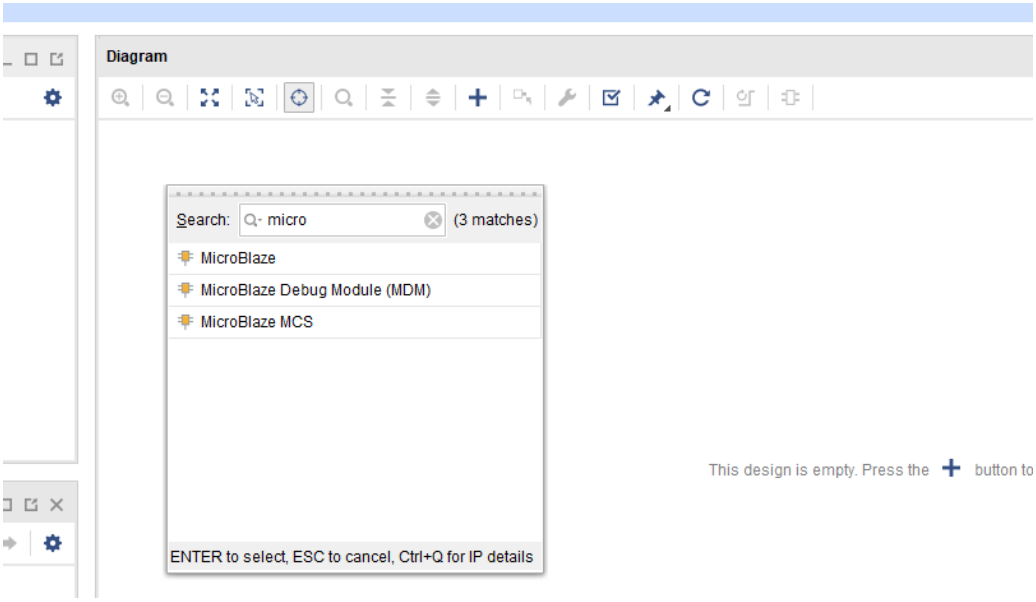
2.2 进行 Block Design

2.2.1 添加 CPU 部件 Micro Blaze

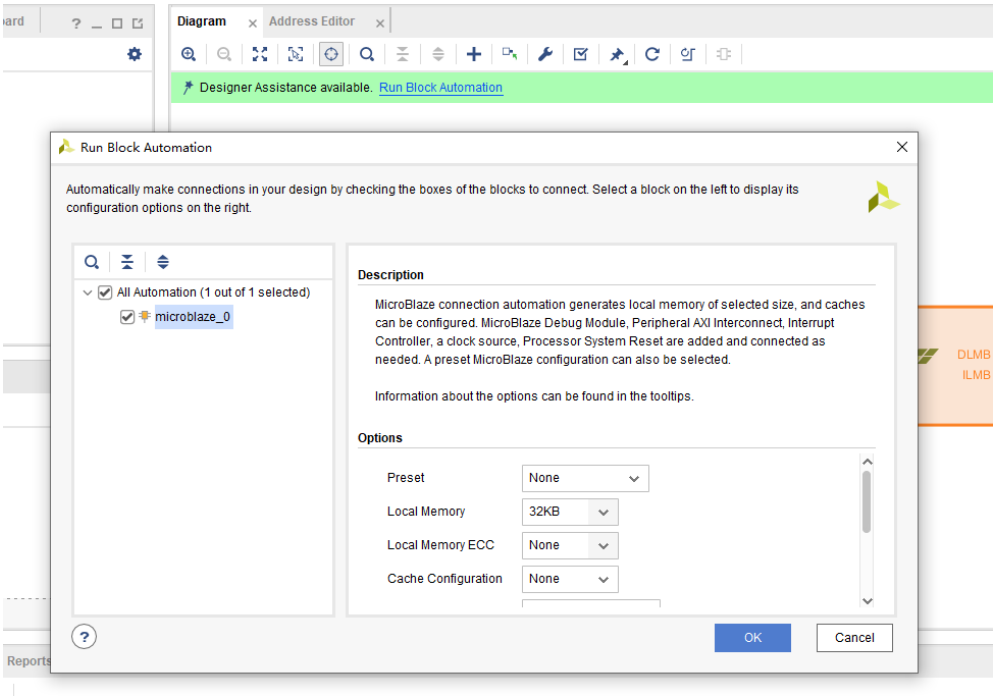
首先创建 Block Design，创建完成后，即可在下图所示界面进行部件的选择：



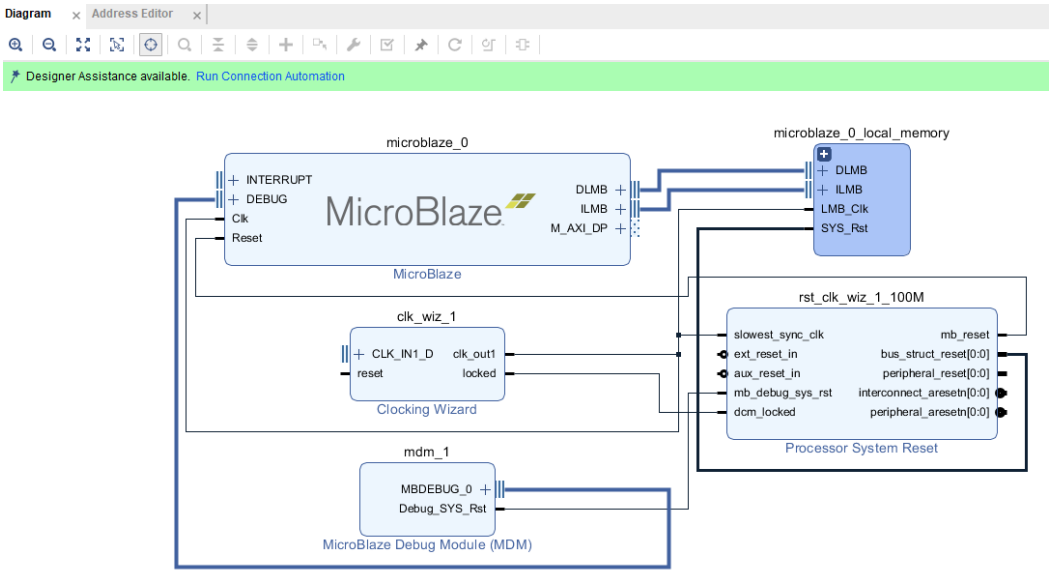
再先添加 CPU 部件 Micro Blaze：



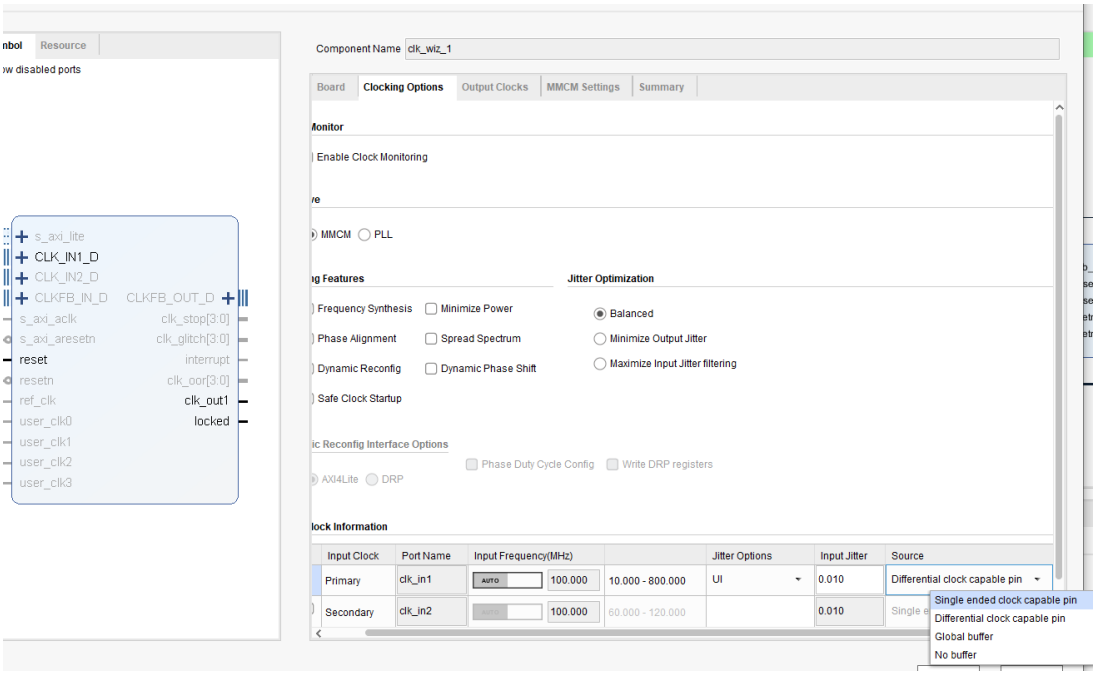
然后利用 Vivado 自动化生成功能，生成 Micro Blaze 运行需要的基本部件，即点击 Run Block Automation 按钮。这里暂时先不使用缓存设备（例如 DDR 等），而是直接利用片内 Memory 存储代码，故将 Local Memory 容量增大，设置为 32KB。

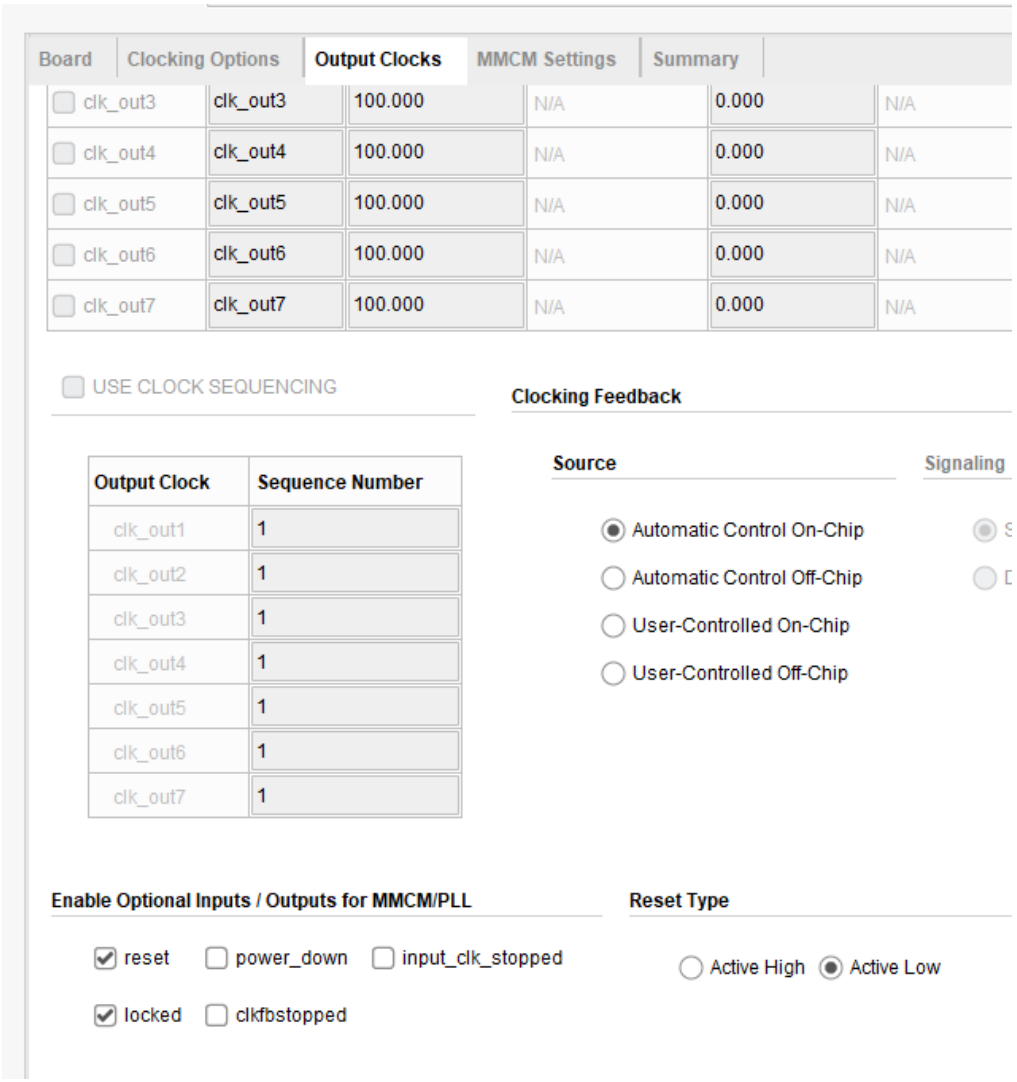


生成，得到下图所示结果。



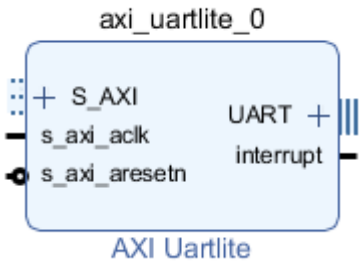
注意到，时钟默认为差分时钟，与 N4 板上时钟类型不符，故修改差分时钟为单端时钟，频率为 100Mhz，并设置复位信号低电平有效。



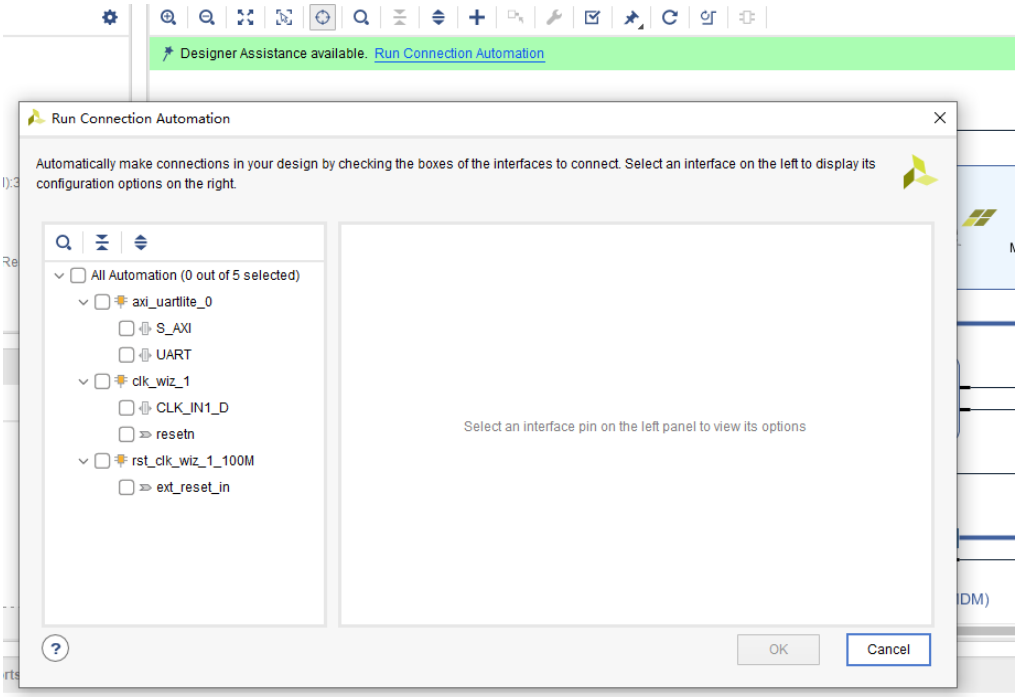


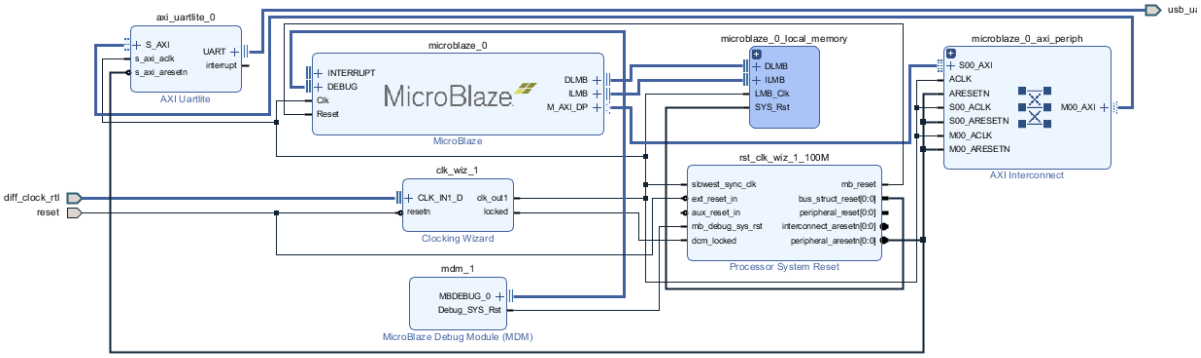
2.2.2 添加 AXI Uartlite IP 核

添加完 CPU 部件，还需要有串口模块提供字符串输出硬件支持，添加 AXI Uartlite IP 核：



进行自动化连接，之后带有串口打印功能的系统就搭载完成。

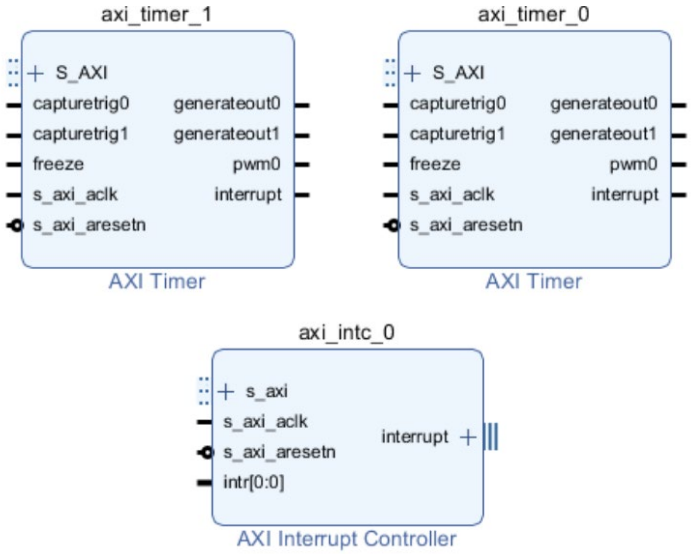




2.2.3 添加定时器 AXI Timer 和中断控制器 AXI Interrupt Controller

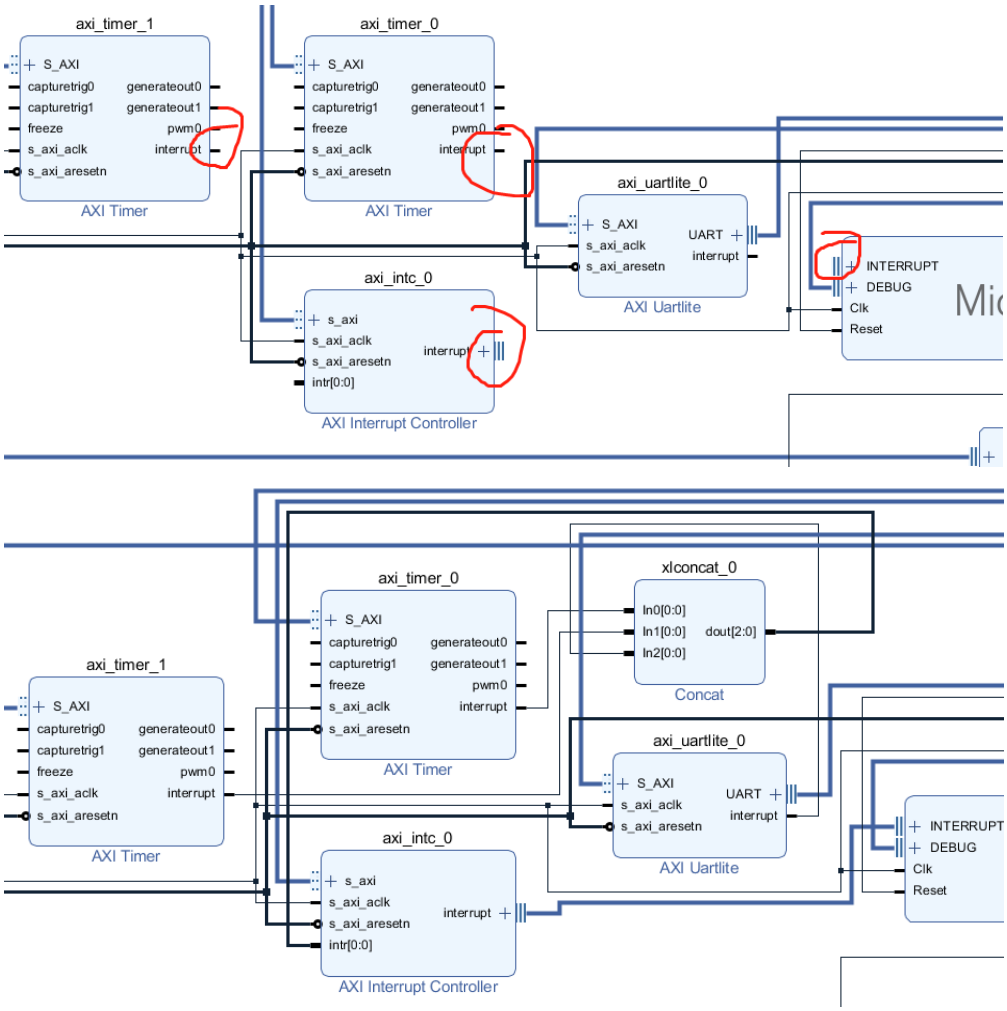
由于μC/OS的 BSP 中要求系统支持时钟中断以及定时器，故还需要添加这两个部件，让操作系统的时钟中断能够正常工作。

添加两个定时器 AXI Timer 和中断控制器 AXI Interrupt Controller：



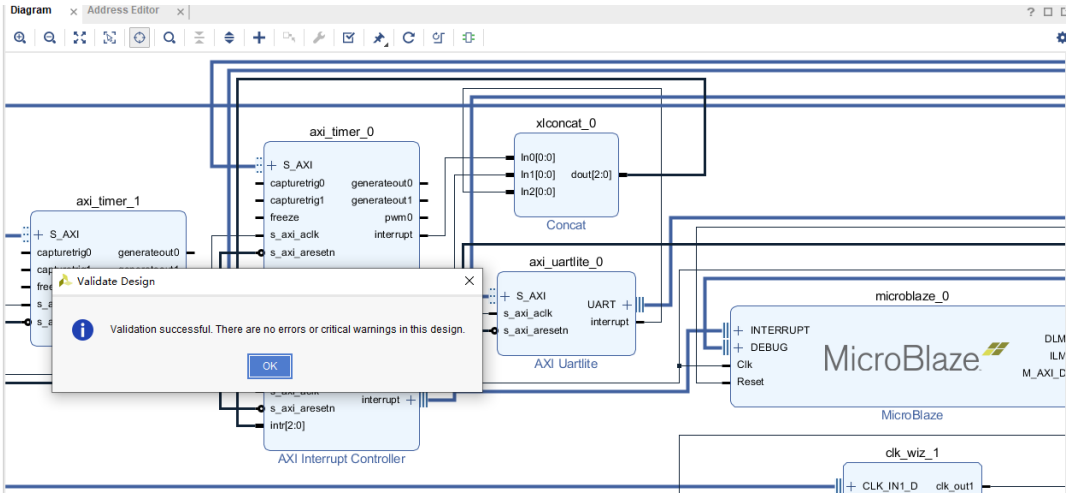
2.2.4 连接中断接口与中断控制器

进行 Auto Connection，手动连接 CPU 的中断接口和中断控制器；创建一个集线连接部件，连接所有 Interrupt 接口（包括两个时钟的中断口以及串口的中断口）后送回中断控制器。



2.2.5 验证布局

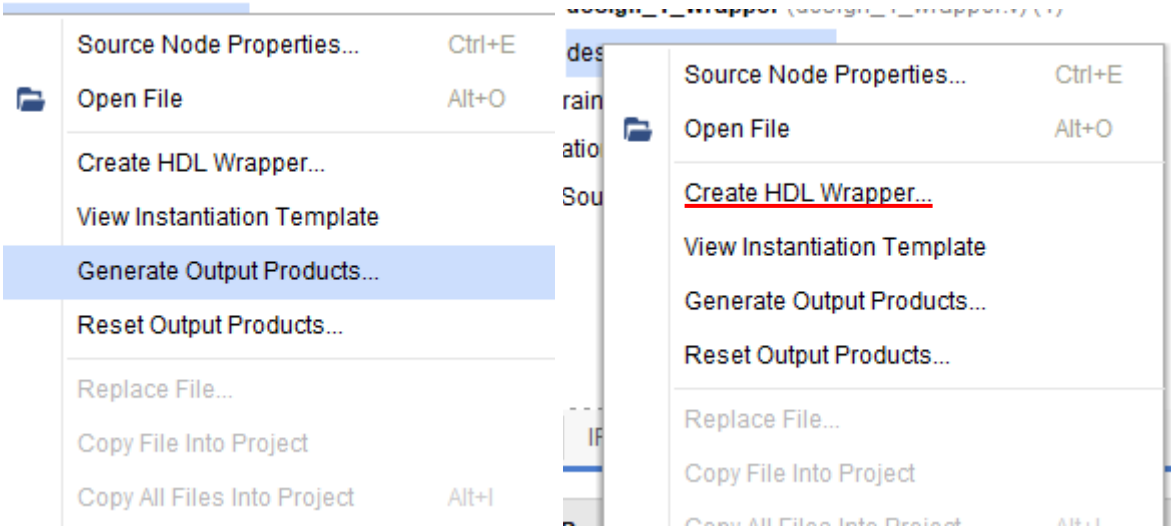
验证布局，block design 结束。



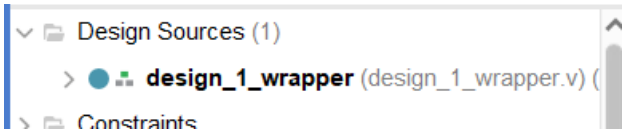
2.3 生成比特流

2.3.1 生成顶层模块

将布局设计输出为实际的 ip 核文件，并生成顶层模块：



顶层模块生成完毕，得到 design_1_wrapper.v 文件。

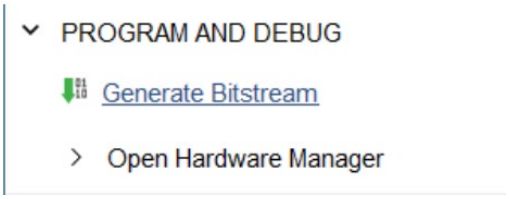


2.3.2 编写约束代码

进行接口约束， E3 为 N4 板上时钟信号源，J15 控制复位信号。C4 和 D4 是 N4 板串口接口。并且 C4 为串口输入信号，D4 为串口输出信号。

```
set_property IOST ANDARD LVCMOS33 [get_ports reset]
set_property IOSTANIDARD LVCMOS33 [get_ports usb_uart_rxd]
set_property IOSTANDARD LVCMOS33 [get_ports usb_uart_txd]
set_property IOST ANDARD LVCMOS33 [get_ports sys_clock]
set_property PACKAGE_PIN E3 [get_ports sys_clock]
set_property PACKAGE_PIN J15 [getports reset]
set_property PACKAGE_PIN C4 [get_ports usb_uart_rxd]
set_property PACKAGE_PIN D4 [get_ports usb_uart_txd]
```

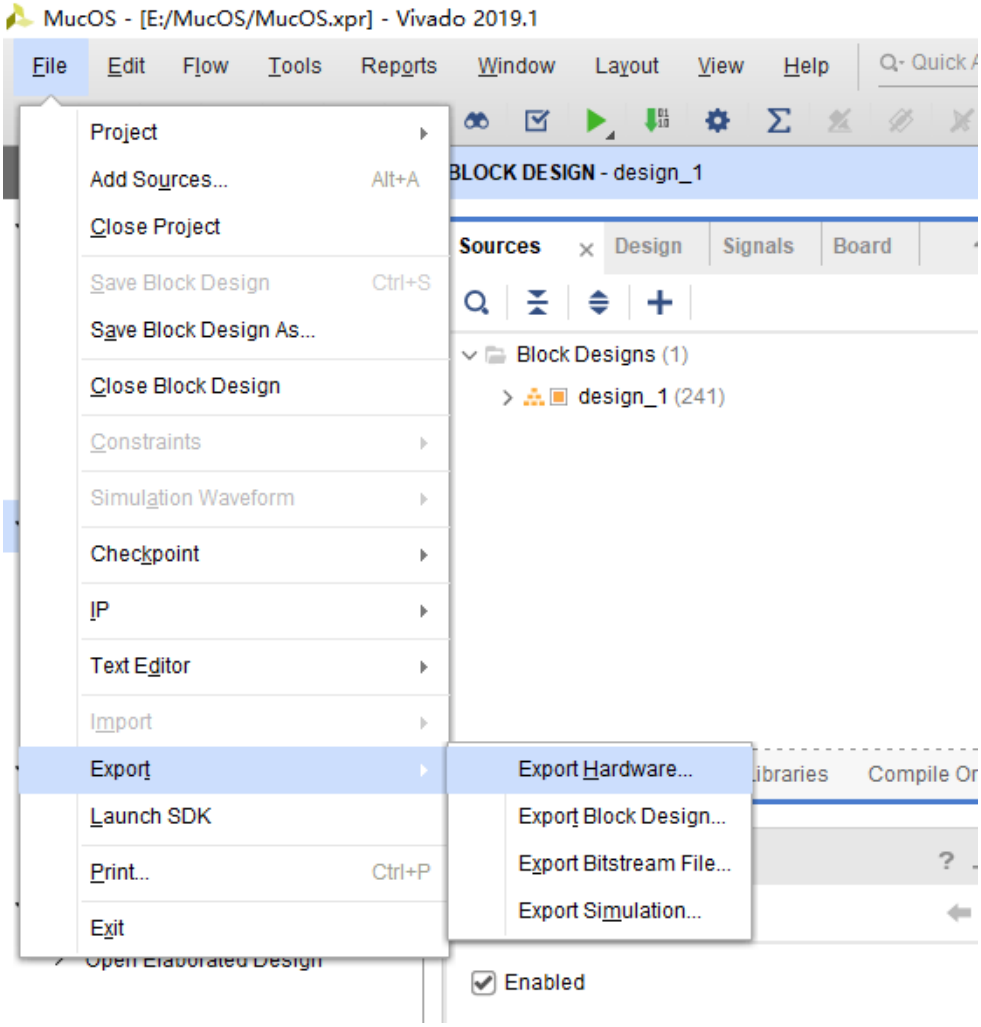
2.3.3 生成比特流



2.4 进行操作系统的移植

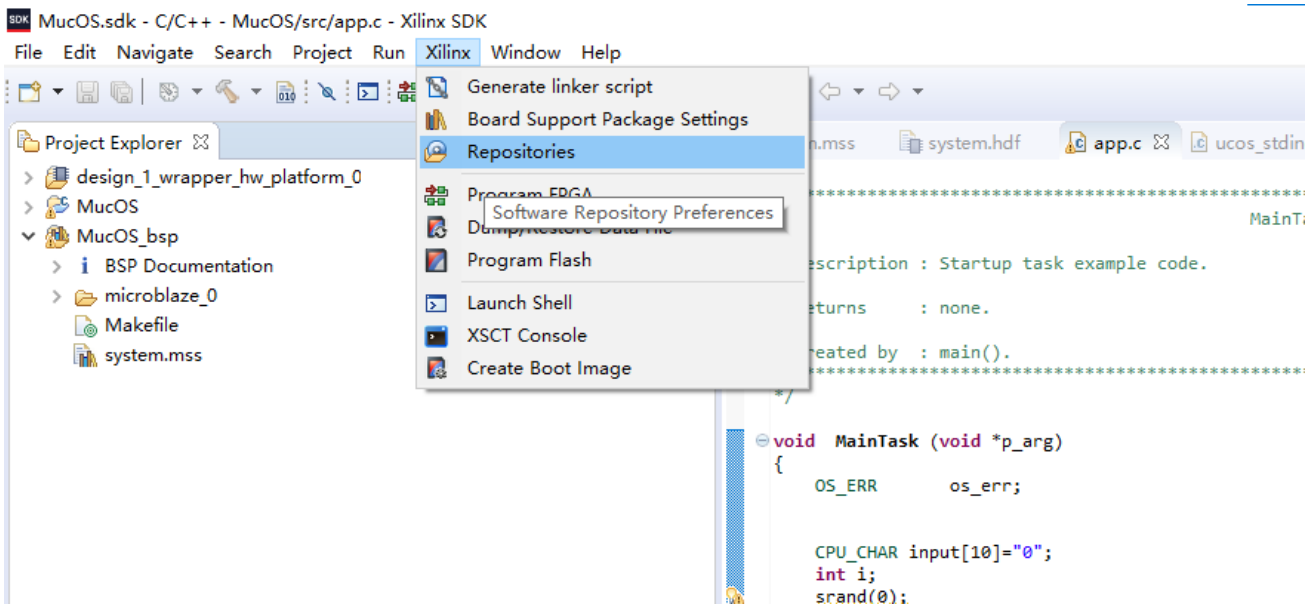
2.4.1 导出硬件

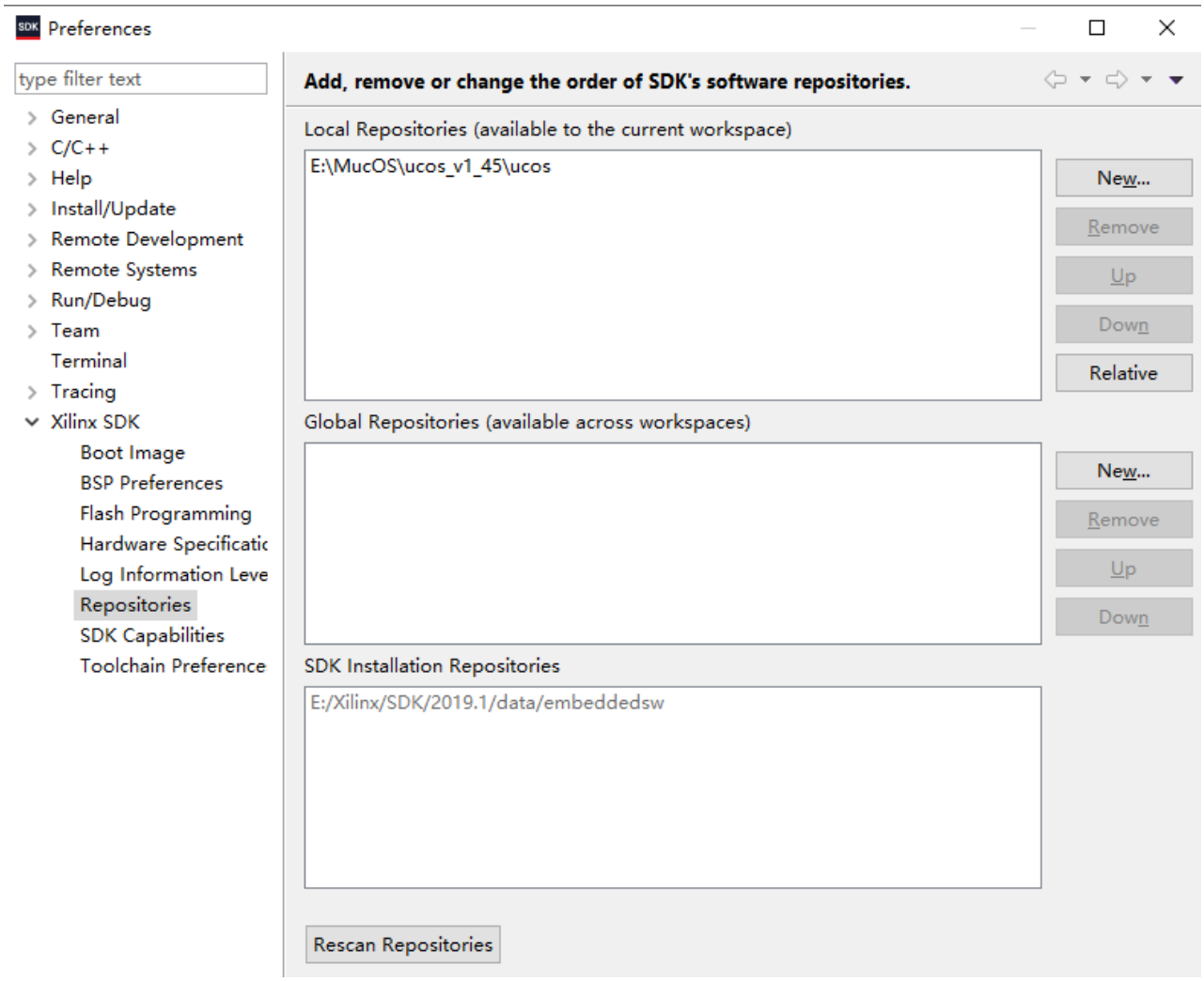
将硬件设计导出，便于 SDK 识别，注意需要包含比特流。



2.4.2 新建工程

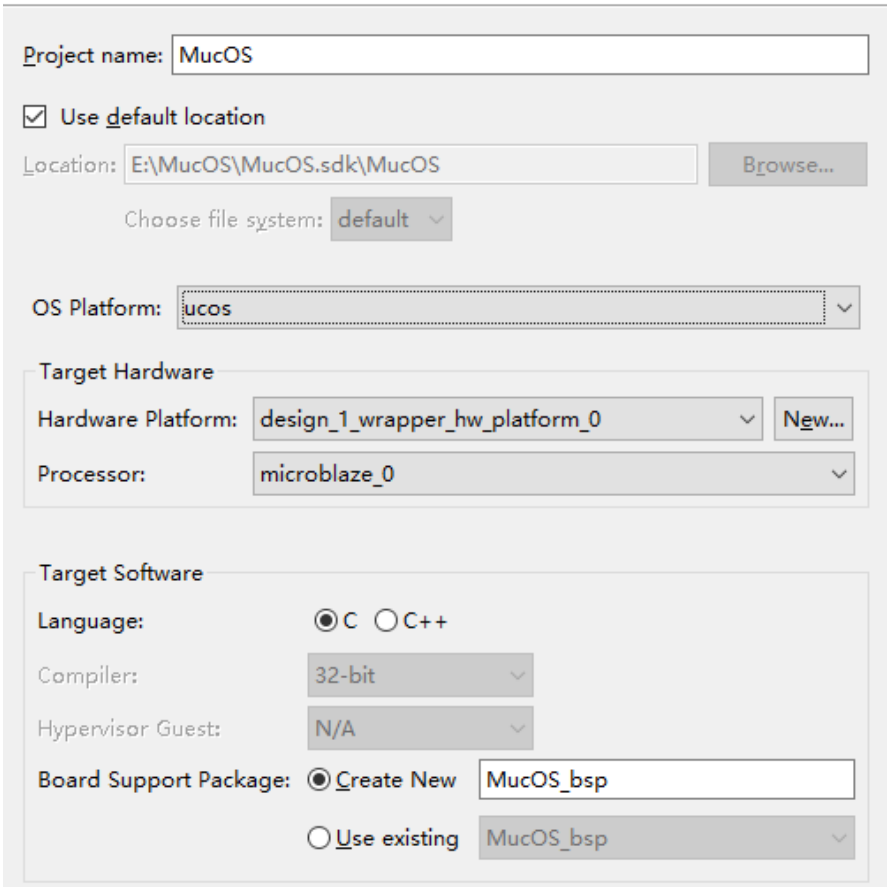
打开 SDK，加入 BSP 的目录地址，以便新建项目的时候可使用相关的μC/OS操作系统封装模块：





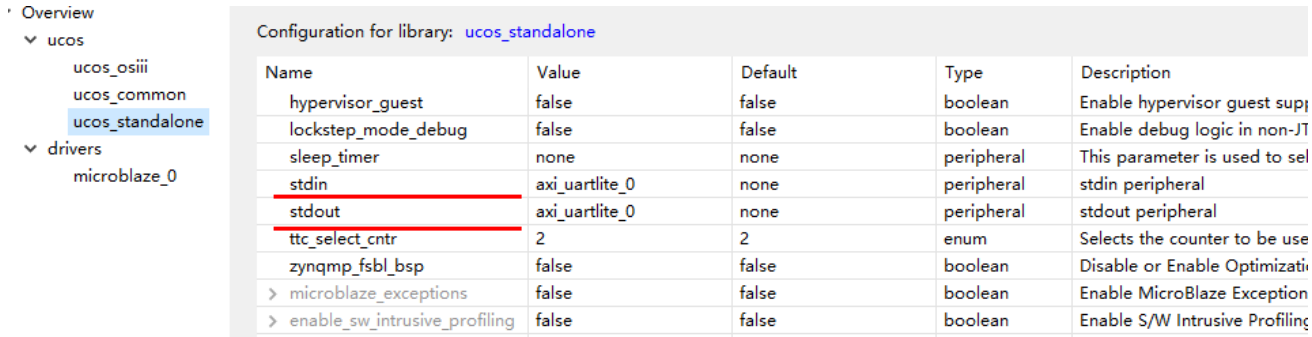
新建 application project，如果 BSP 目录配置正确，可以看到 OS Platform 中出现μC/OS的选择项。如果硬件设计配置正确，可以看到 Hardware platform 以及对应的 micro blaze 处理器。

点击 next，然后选择μC/OS - III，至此新建工程完毕。

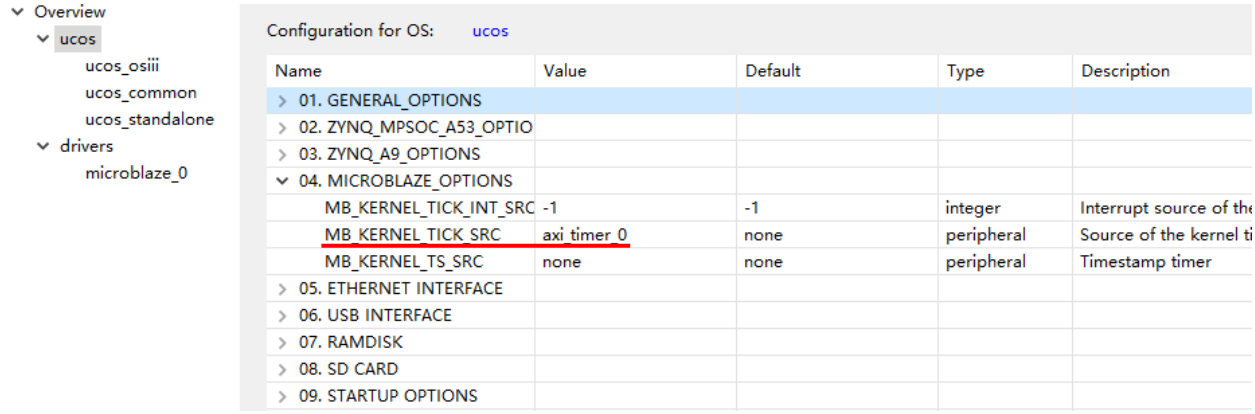


2.4.3 板级配置

打开板级配置菜单，指定标准输入输出的串口：



指定中断时钟信号：



点击 OK，重新生成 BSP。

之后的步骤主要为了验证移植的操作系统可以正常工作。

2.5 代码编写

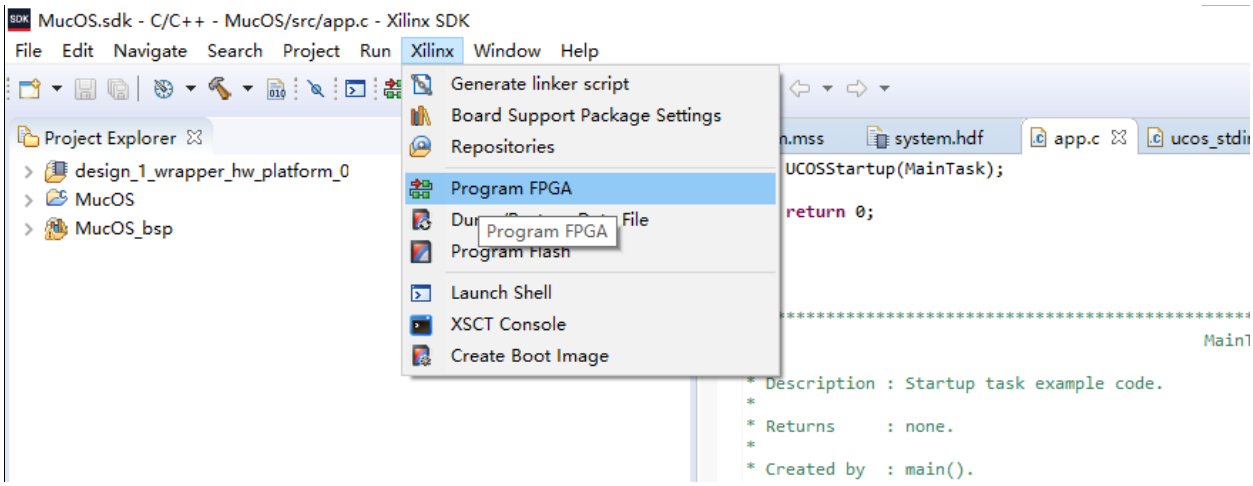
验证程序的开发在 app.c 文件中进行，以打印 helloworld 为例，编写 maintask，主要为了验证时钟中断、计时器部件以及串口部件的正确性：

```
void MainTask (void *p_ arg)
{
    while(1)
    {
        UCOS_Print ("Hello world from the main task\r\n");
        sleep(1);
    }
    while (DEF_TRUE) {
        OSTimeDlyHMSM(0, 0, 10, 0);
        UCOS_ Print("Periodic output every 10 seconds from the main task\r\n");
    }
}
```

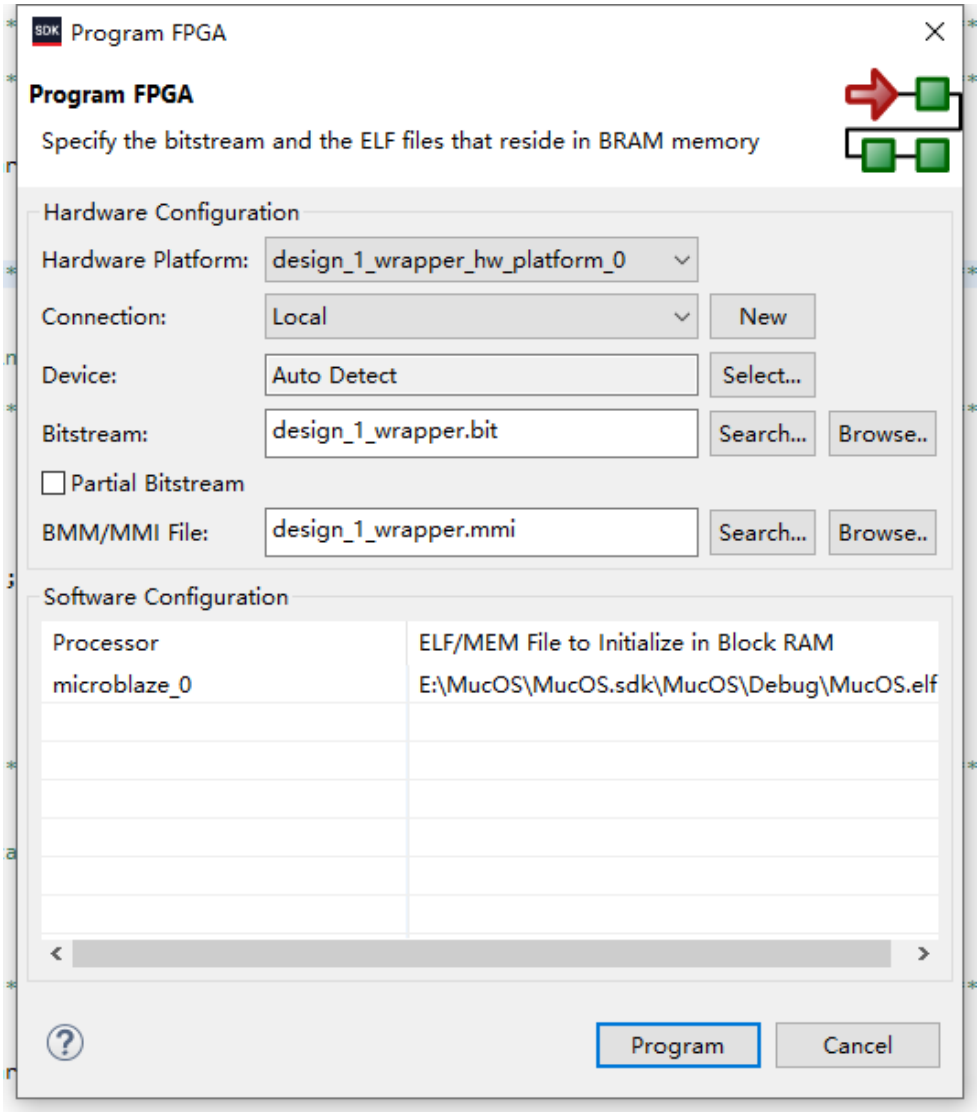
编译生成 Trans.elf.size 文件。

2.6 下板验证

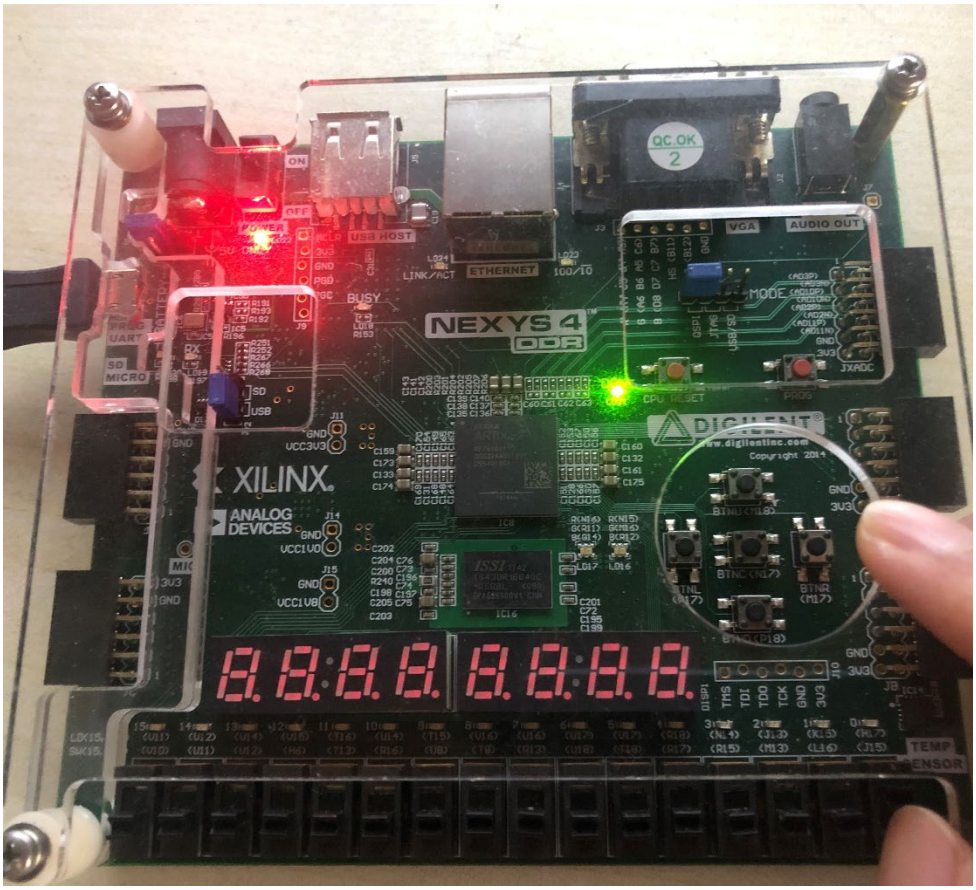
通过 USB 连接好 N4 板，利用 SDK 下板：

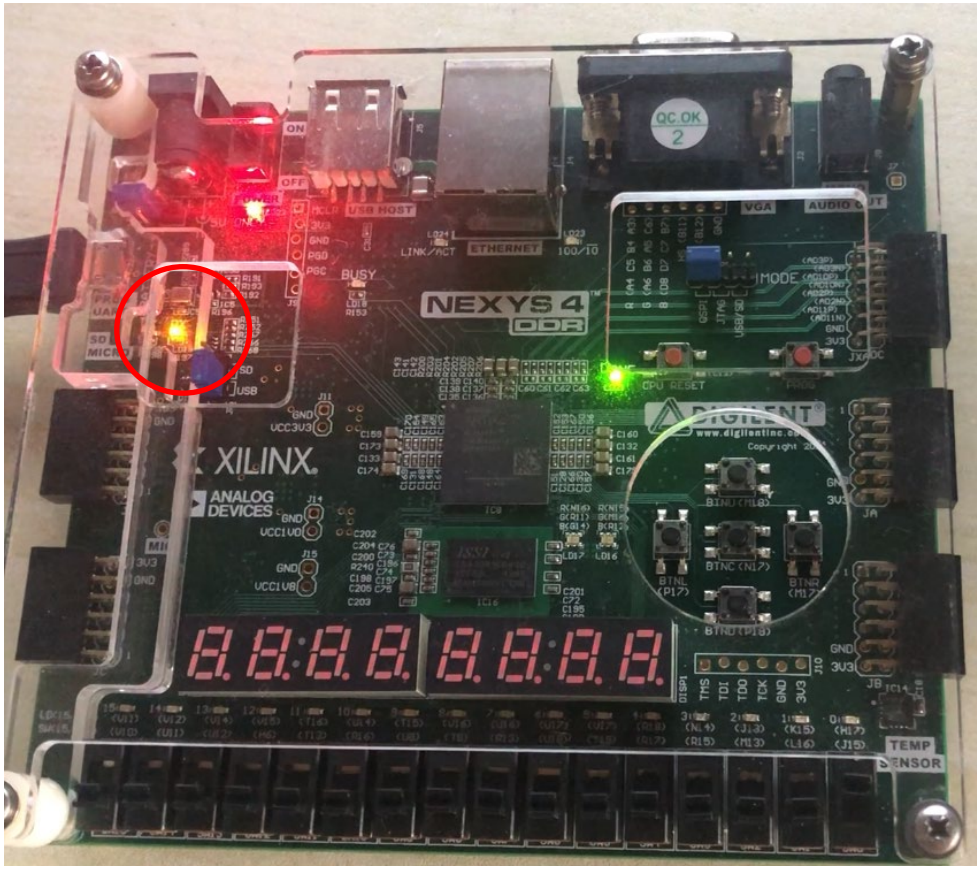


进行下板配置，Processor 的 software 选择刚刚编译完成的 Trans.elf.size 文件，点击 Program。



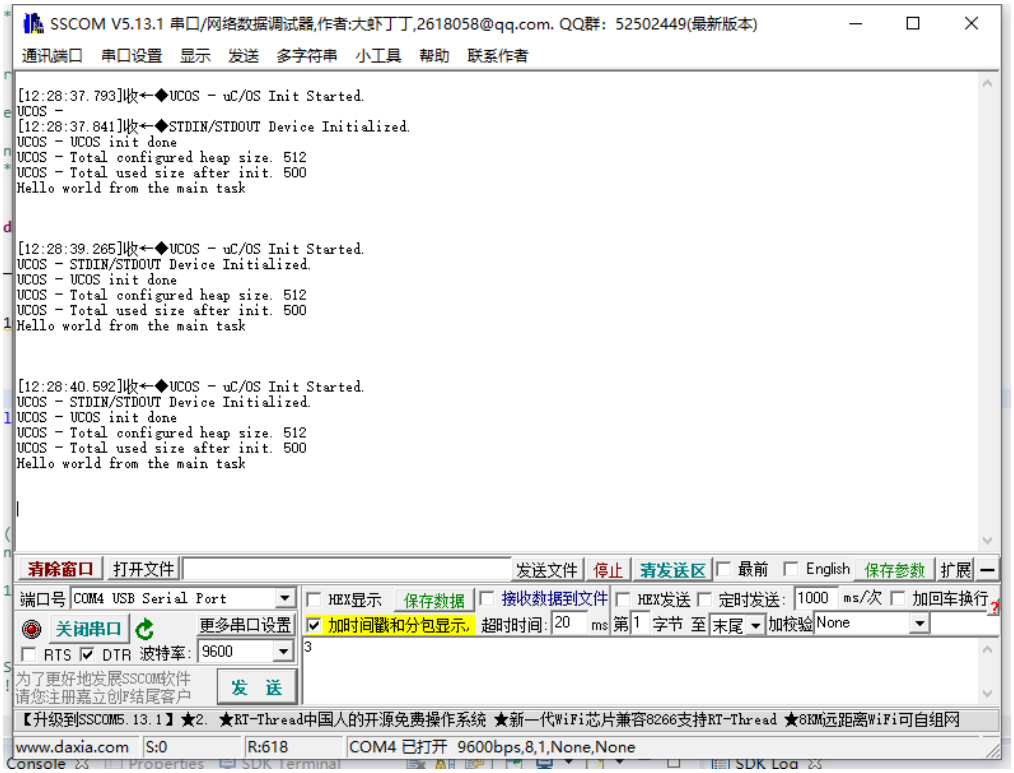
由于复位信号低电平有效，当 J15 拨杆上拨的时候，操作系统开始运行。**串口灯**一秒闪烁一次，说明字符串在循环打印，串口部件以及时钟中断正常工作：


NEXYS 4
DDR
XILINX
ANALOG DEVICES
DIGILENT
QC.OK
2
VGA
AUDIO OUT
ETHERNET
USB HOST
MODE
CPU RESET
TEMP SENSOR


NEXYS 4
DDR
XILINX
ANALOG DEVICES
DIGILENT
QC.OK
2
VGA
ETHERNET
USB HOST
MODE
CPU RESET
TEMP SENSOR

如图，串口灯亮

打开串口调试助手进行验证，波特率选择 9600，N4 板通过 USB 口连接到电脑，选择 COM7 USB 串口：



消息差不多每秒打印一次，时间上有一定的传输误差；拨动复位开关，重启操作系统后会出现初始化串口信息。μC/OS重启后，会初始化标准输入输出设备，指定了串口，然后分配对应的程序空间。

三、实验总结

3.1 总结与体会

本次综合实验的目的是移植操作系统到 N4 板上，能够编写相应的应用程序并最终下板。首先是操作系统的选择，主流的选择是 FreeRTOS 及$\mu C/OS$ 。FreeRTOS 任务间通讯只支持 Queque，Semaphores，Mutex。$\mu C/OS$除这些外，还支持 Flag，MailBox。应用程序的拓展上面，$\mu C/OS$还有大量外延，比如 FS，USB，GUI，CAN 的支持。因而我们最终选择了$\mu C/OS$。

关于合适 CPU 架构的选择，我们之前完成的流水线 CPU 功能并不完善，缺少众多部件，而且需要考虑到串口等输出调式部件的搭载，故不选择，最终选择 VIVADO 已经中封装成 IP 核的嵌入式、可修改预置 32 位 / 64 位 RISC 微处理器的 Micro Blaze。

关于开发工具链的选择，传统的操作系统移植方法是利用交叉编译工具链进行交叉编译，最终挂载到相应的开发部件。此过程需要针对不同的开发板进行驱动的删减，我们则经过调研之后，最终采用 VIVADO + Xilinx SDK + Micrium Xilinx BSP 的方式进行移植。

整个 CPU 的配置和操作系统的移植过程都非常需要细心，如硬件部分的连线、各个模块内部时钟的配置等等。在本次实验中，我更进一步地了解了 Micro Blaze 软核、$\mu C/OS$ 操作系统以及 VIVADO 上两者的移植和配置，同时为进一步进行应用程序的编写打下了基础。

四、参考文献

[1] https://blog.csdn.net/yundanfengqing_nuc/article/details/87872872 （micro blaze 软核处理器及其 ip 核调用）

[2] http://www.ucos-ii.com/（$\mu C/OS$ 官网）

[3] https://cloud.tencent.com/developer/article/1658607（走进$\mu C/OS$ - III 操作系统）

[3] https://china.xilinx.com/products/design-tools（XILINX 官网）