# 第一章 Hello World 实验

“Hello World!”是各种编程语言中最简单，同时也是最经典的入门实验。因此，我们将串口打印“Hello World”作为 MPSOC 嵌入式系统的开篇实验，这也是我们步入 MPSOC 的 PS 部分的始发点。通过本次实验我们将了解 MPSOC 嵌入式系统的开发流程，熟悉 MPSOC 嵌入式最小系统的搭建。

本章包括以下几个部分：

1.1 简介

1.2 实验任务

1.3 硬件设计

1.4 软件设计

1.5 下载验证

## 1.1 简介

首先我们来了解一下 MPSOC 嵌入式系统的开发流程。

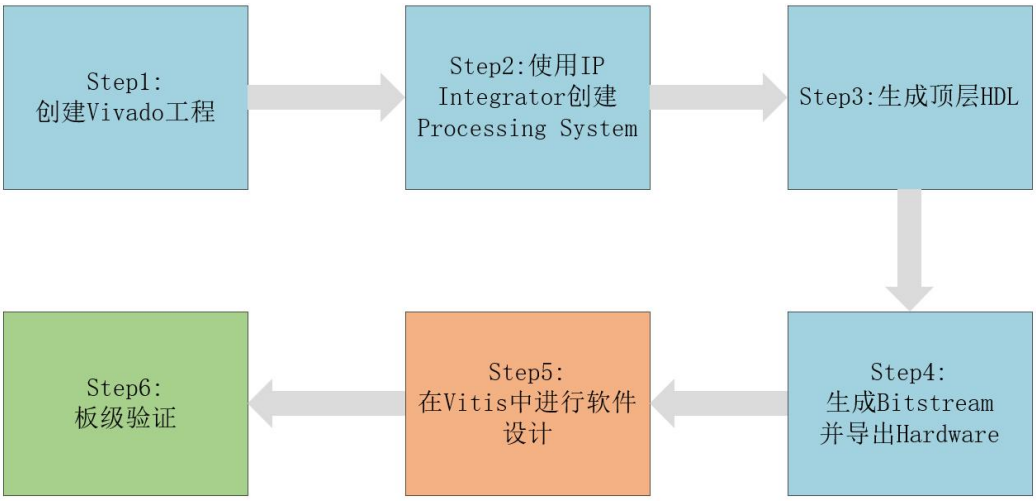


图 1.1.1 MPSOC 嵌入式系统开发流程

如上图所示，开发流程大体可以分为 6 步。其中 step1 至 step4 为硬件设计部分，在 Vivado 软件中实现；step5 为软件设计部分，在 Vitis 软件中实现；step6 为功能的验证。复杂的程序还涉及 Debug，这个也是在 Vitis 软件中实施。具体每一步的操作我们会在后面详细介绍。

在简单了解嵌入式系统的开发流程后，接下来我们来看一下什么是嵌入式最小系统。嵌入式最小系统的概念包括以下两个方面：一、它是使系统正常工作的最小条件；二、它是其他系统建立的基础。

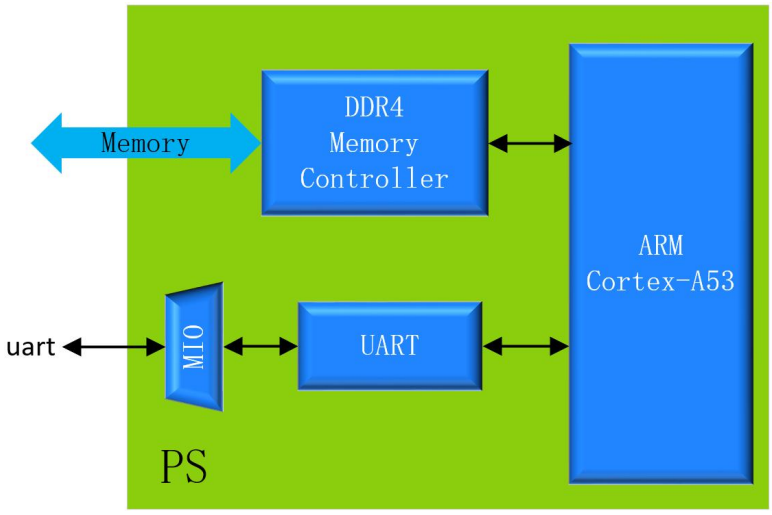


图 1.1.2 MPSOC 嵌入式最小系统

如图 1.1.2 所示，以 ARM Cortex 为核心、DDR 为内存，加上传输信息使用的 UART 串口就构成了 MPSOC 嵌入式最小系统。可以看到，这个最小系统只包括了 MPSOC 中的 PS 部分。

下面我们将按照 MPSOC 嵌入式系统开发流程，一步步的搭建上图所示的最小系统。

## 1.2 实验任务

本章的实验任务是在 MPSOC 开发板上搭建 MPSOC 嵌入式最小系统，并使用串口打印“Hello World”信息。

## 1.3 硬件设计

在图 1.1.1 中，我们将 step1 至 step4 划分为硬件设计部分。

**step1：创建 Vivado 工程**

1-1 打开 Vivado，进入 Vivado 界面后，点击“Quick Start”栏的 “Create Project”。然后在弹出的创建 Vivado 工程向导界面，点击“Next”。如下图所示：

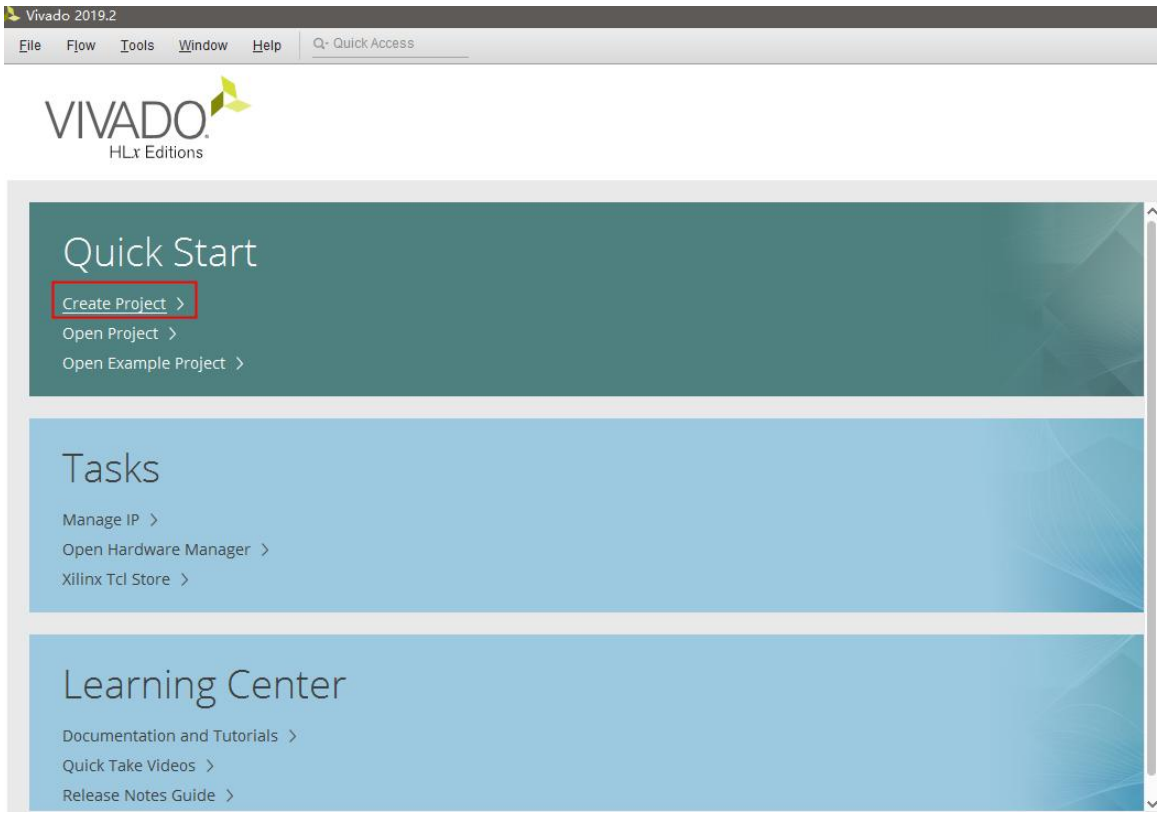


图 1.3.1 点击创建工程

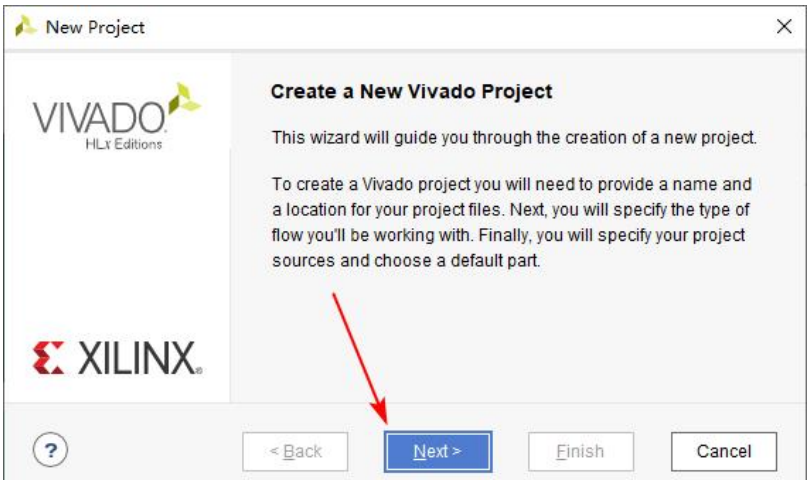


图 1.3.2 创建工程向导

1-2 进入工程命名界面。设置工程名为“hello_world”，工程路径可使用任意路径，本章我们将该工程放在 F:\ZYNQ\Embedded_System 文件夹下。注意，**工程名和路径只能由英文字母、数字和下划线组成，不能包含中文、空格以及特殊字符！**

确认已经勾选“Create project subdirectory”,点击“Next”，如下图所示：

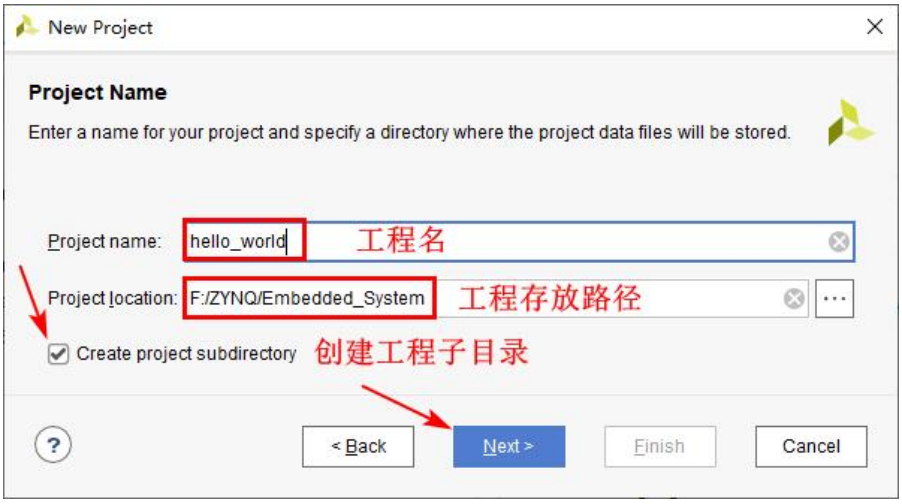


图 1.3.3 设置工程信息

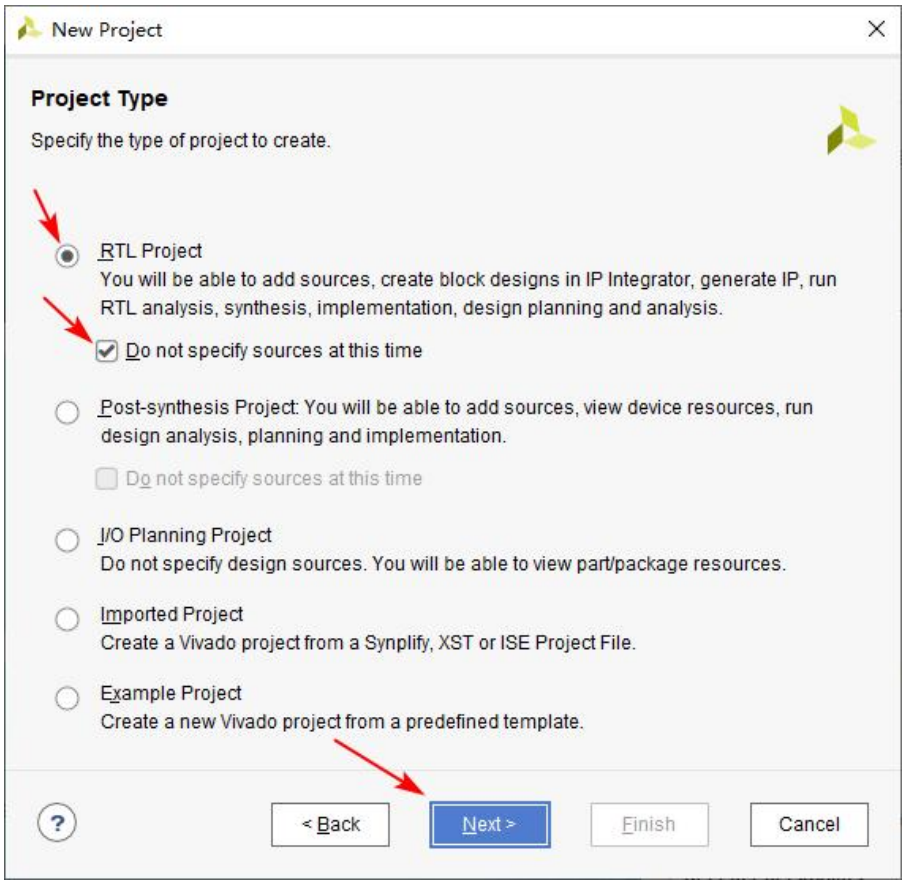


图 1.3.4 选择工程类型

1-3 进入图 1.3.4 所示的界面，在此界面设置工程类型。此处我们选择 “RTL Project”。本次实验不需要添加源文件和约束文件，所以勾选“Do not specify sources at this time”。勾选之后会省略后面添加源文件和约束文件的步骤，点击“Next”直接跳到器件选型界面。

1-4 器件选型界面。所选择的器件型号一定要跟开发板上的 MPSOC 芯片型号保持一致。开发板上的 MPSOC 芯片有两种型号，**XCZU2EG** 和 **XCZU4EV**。大家可以通过查看开发板上 MPSOC 芯片的丝印来确认所使用的芯片型号。其中 **XCZU2EG** 的 speed 等级为“-2”，**XCZU4EV** 的 speed 等级为“-1”，这在器件选型的时候需要注意。此处以 **XCZU4EV** 为例。

选择器件型号的方式有两种，一种是根据 Parts，另一种是根据 Boards，此处我们使用

Parts 选择器件。在 Family 栏里选择“ZYNQ Ultrascale+MPSOCs”, Speed 栏选择“-1”，需要注意的是，在 Package 栏选择“sfvc784”。然后根据开发板上的 MPSOC 芯片型号，在下面的器件列表中选择“xczu4ev-sfvc784-1-i”，如下图所示。若是 XCZU2EG，相对应的器件为“xczu2eg-sfvc784-2-i”。

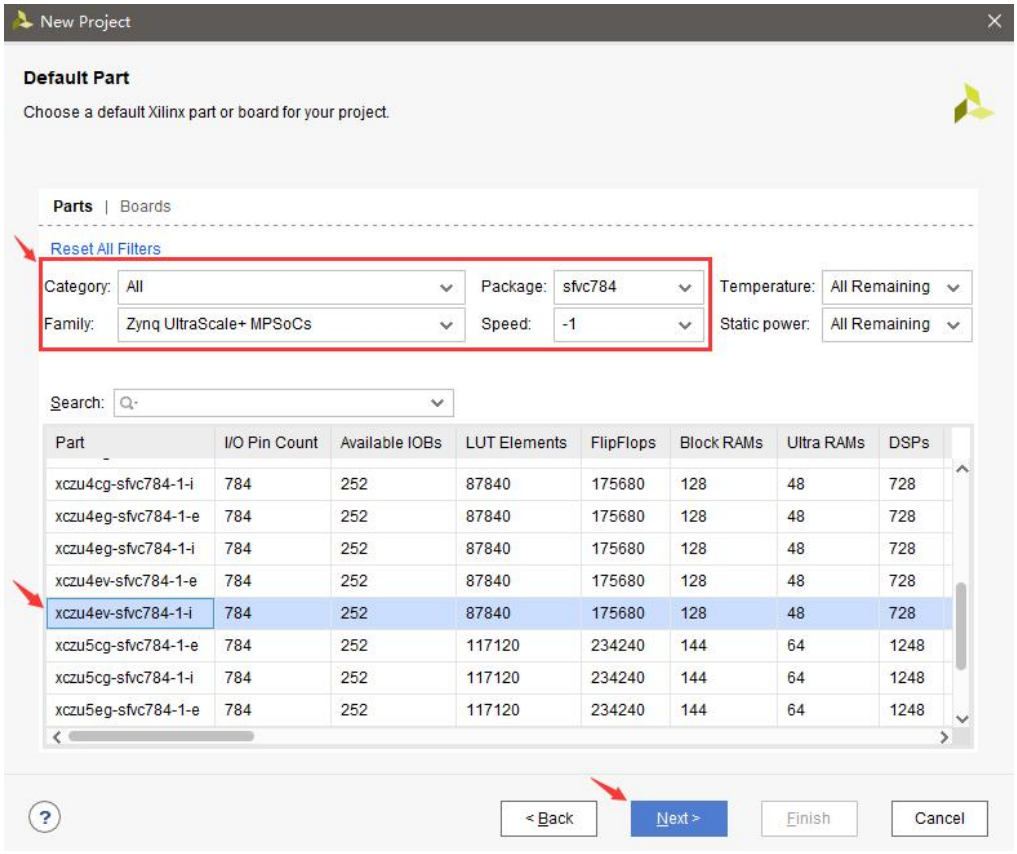


图 1.3.5 器件选型界面

选中之后，点击“Next”。

1-5 工程摘要界面。这是创建工程的最后一步，显示工程摘要信息，如图 1.3.6 所示。在此界面检查前面所设置的工程名称、所选择的器件型号等信息。如果发现工程设置有误，则可以通过 Back 按钮返回前面的步骤，重新设置。检查无误后点击“Finish”，完成工程创建。

工程创建完成后的 Vivado 界面如图 1.3.7 所示。

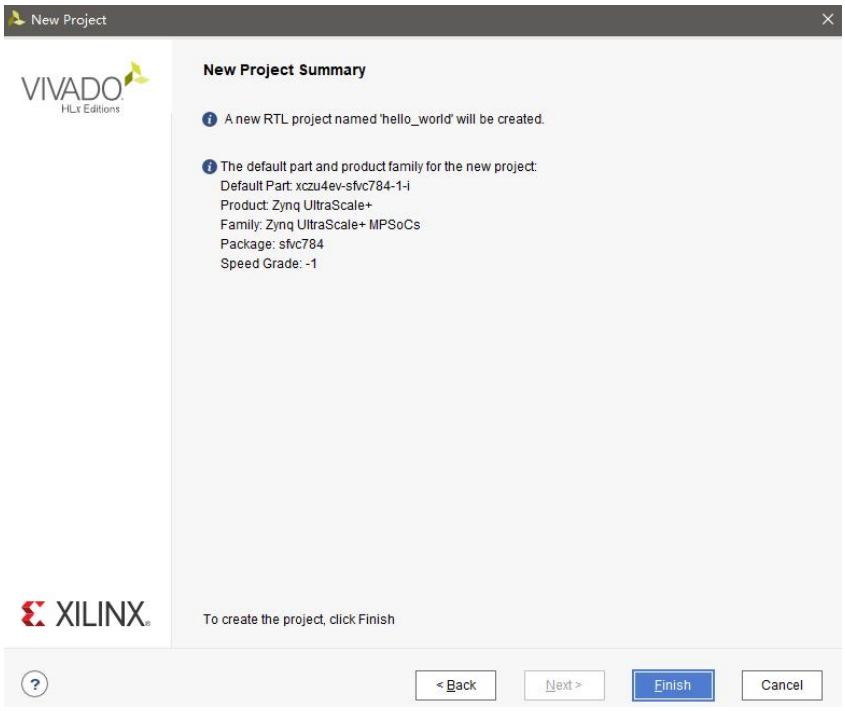


图 1.3.6 工程摘要界面

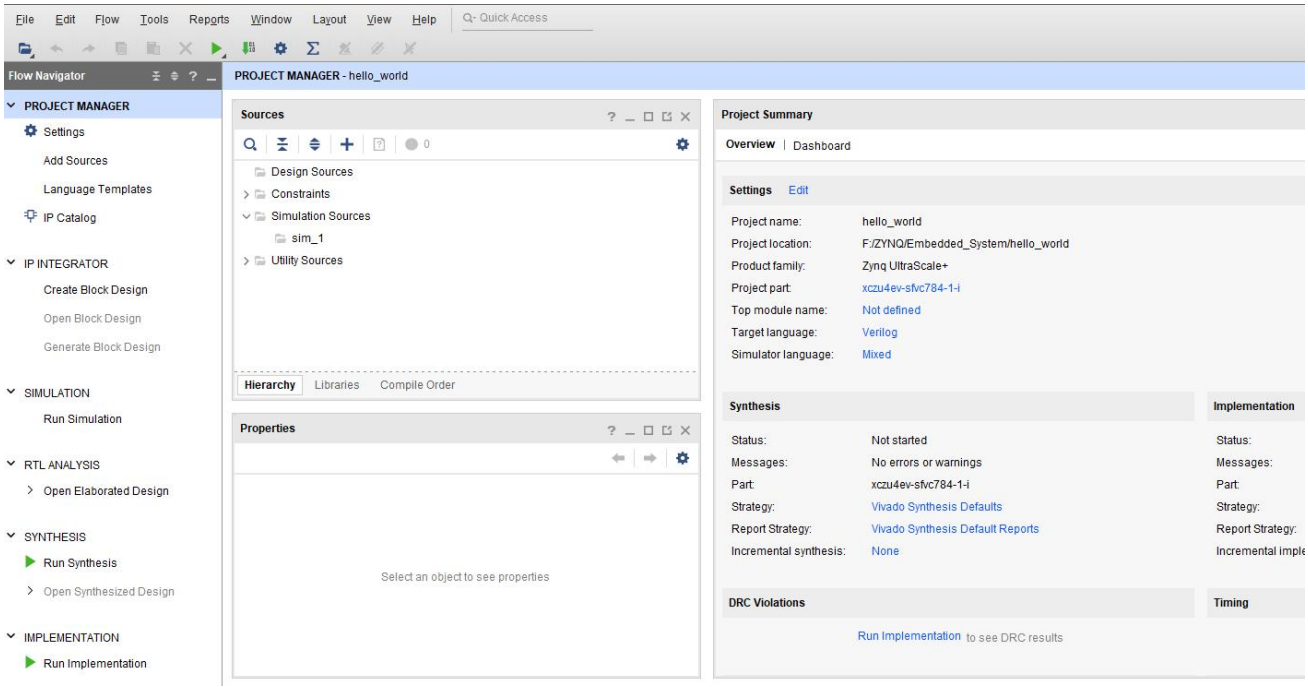


图 1.3.7 工程创建完成后的 Vivado 界面

**step2：使用 IP Integrator 创建 Processing System**

Vivado 开发套件中提供了一个图形化的设计开发工具——IP 集成器（IP Integrator），在 IP 集成器中可以非常方便的插入各种功能模块（IP）。它支持关键 IP 接口的智能自动连接、一键式 IP 子系统生成、实时 DRC 等功能，能够帮助我们快速组装复杂系统，加速设计流程。

接下来我们将在 IP 集成器中完成 MPSOC 嵌入式系统的搭建。

2-1 在左侧导航栏（Flow Navigator）中，单击 IP Integrator 下的 Create Block Design。然后在弹出的对话框中指定所创建的 Block Design 的名称，这里使用默认的“design_1”。如下图所示：

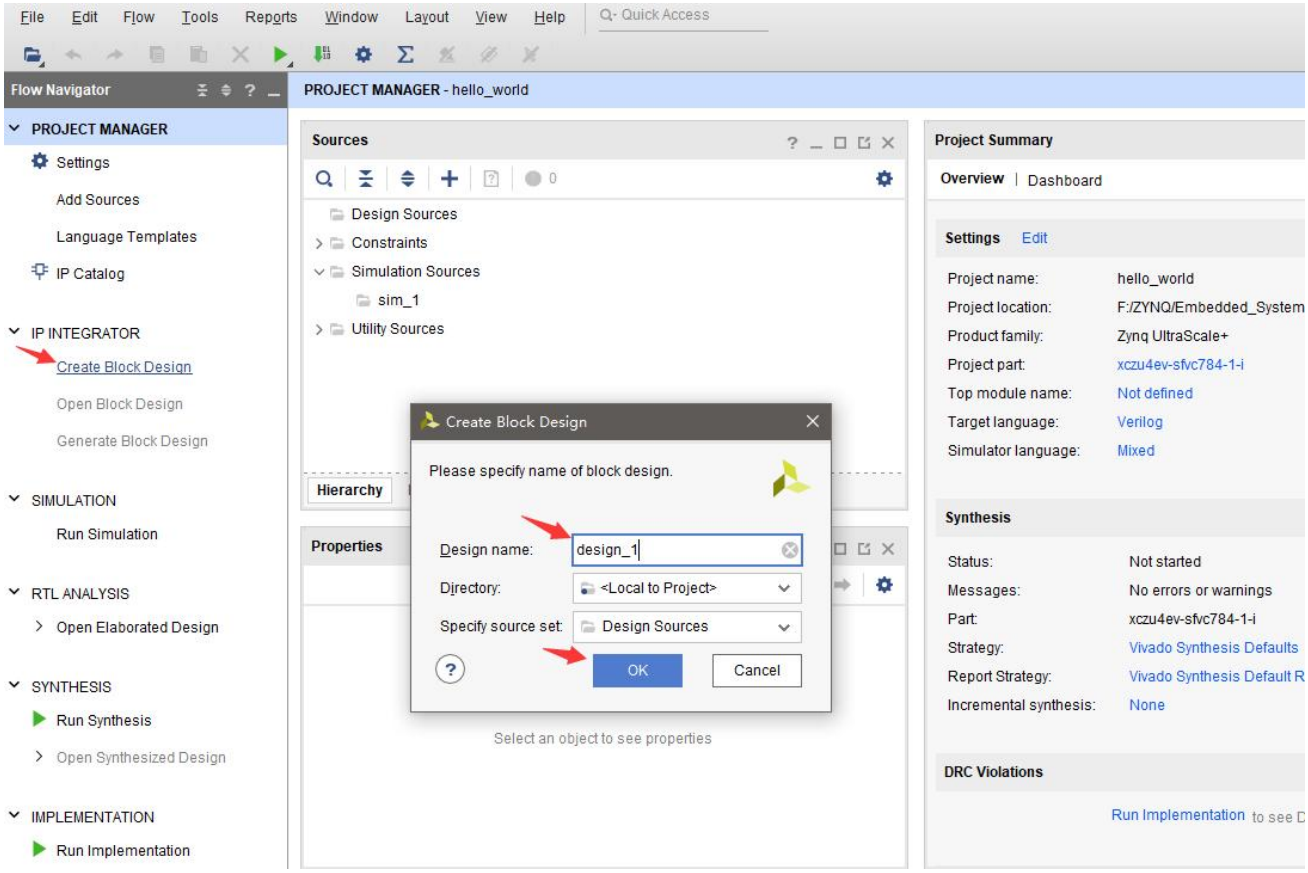


图 1.3.8 创建 Block Design

2-2 点击“OK”按钮，此时 Vivado 界面如下图所示。注意右侧的 Diagram 窗口，我们将在该窗口中以图形化的方式完成设计。

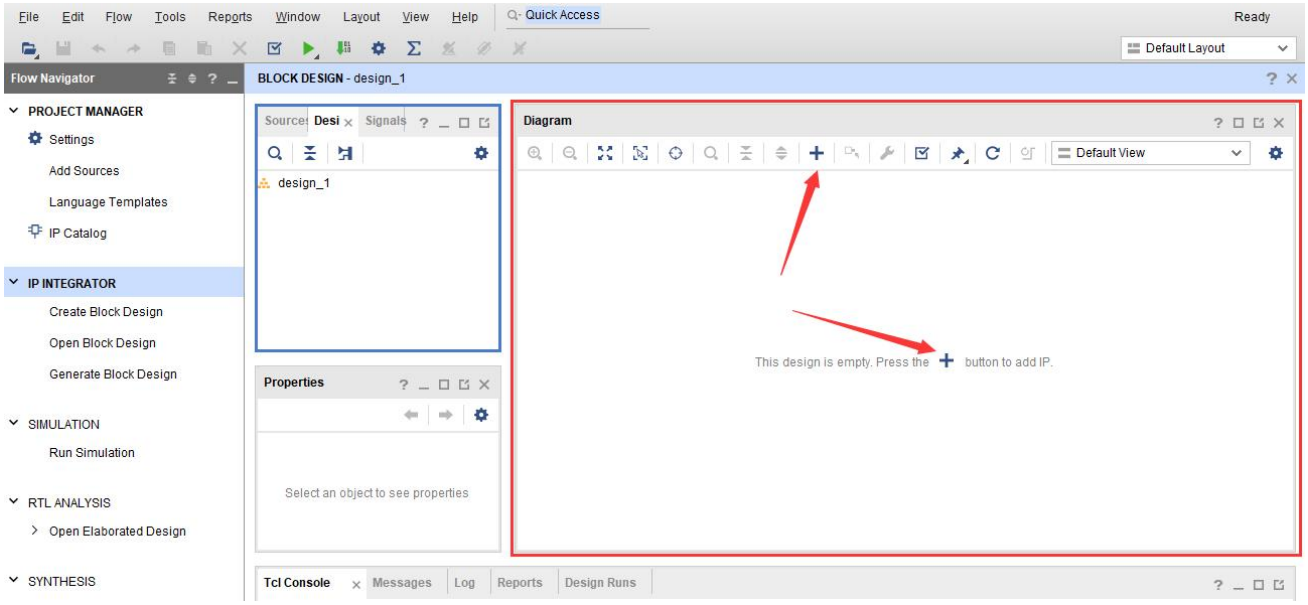


图 1.3.9 Block Design 界面

2-3 接下来在 Diagram 窗口中给设计添加 IP。点击上图中箭头所指示的加号(两个任选一个)“+”，会打开 IP 目录（IP Catalog）。也可以通过快捷键 Ctrl + I，或者右键点击 Diagram 工作区中的空白位置，然后选择“ADD IP”。

2-4 打开 IP 目录后，在搜索栏中键入“ZYNQ”，找到并双击“Zynq UltraScale+MPSOC”，

将 Zynq UltraScale+MPSOC　IP 添加到设计中。

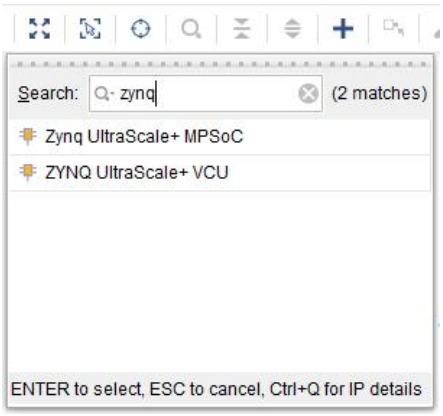


图 1.3.10 添加 Zynq UltraScale+MPSOC IP

2-5 添加完成后，Zynq UltraScale+MPSOC 模块出现在 Diagram 中，如下图所示：

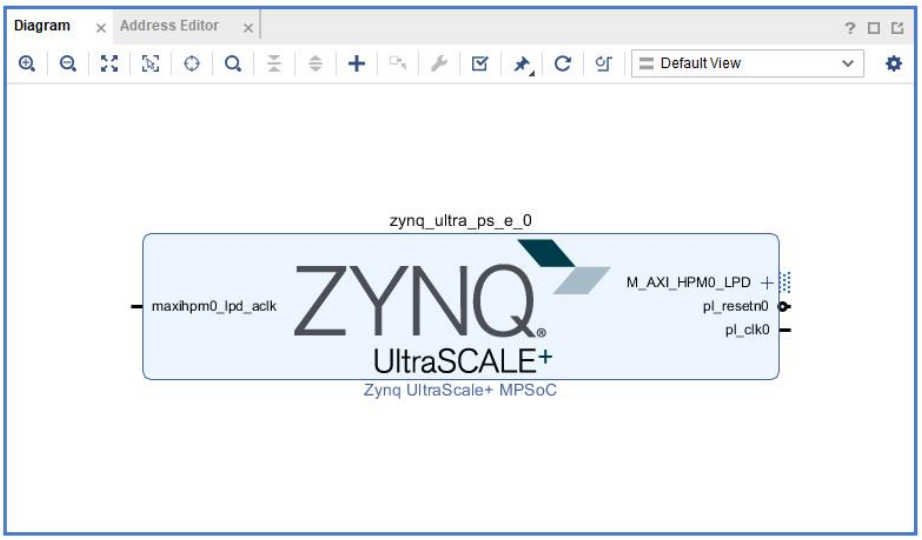


图 1.3.11 Zynq UltraScale+MPSOC IP

2-6 双击所添加的 Zynq UltraScale+MPSOC 模块，进入处理系统的配置界面。界面左侧为页面导航面板，右侧为配置信息面板。如下图所示：

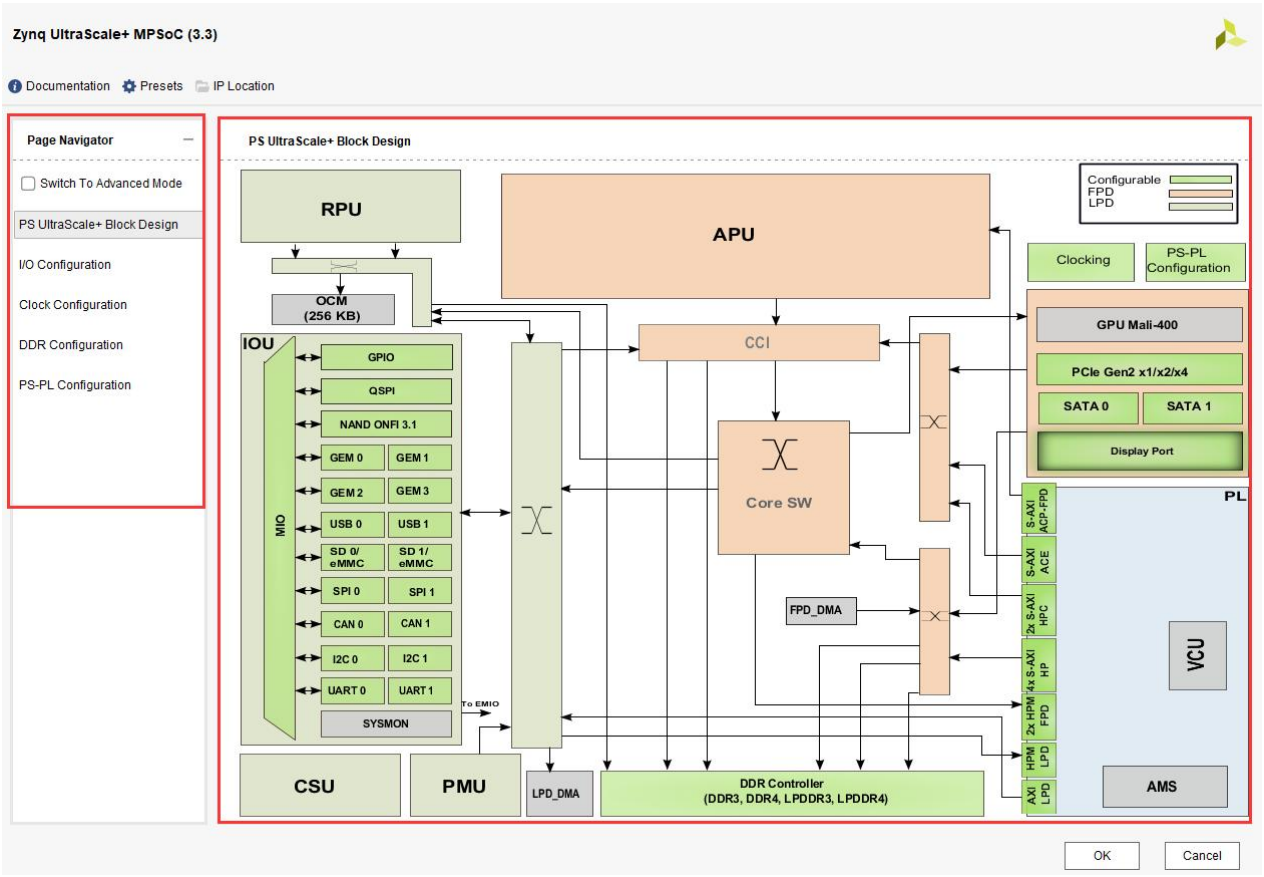


图 1.3.12 Zynq UltraScale+MPSOC 配置界面

下面我们简要地介绍一下页面导航面板中各个页面的作用。

PS UltraScale+ Block Design 页面显示了 zynq 硬核的整体架构图，其中绿色部分是可配置模块，可以点击进入相应的编辑界面进行配置，当然也可以在左侧导航栏选择相应的编辑界面。

I/O Configuration 页面可以选择不同的 I/O 外设并进行相应的配置。

Clock Configuration 页面分为 Input Clocks 和 Output Clocks 两个标签页，用来配置 PS 输入时钟、外设时钟，以及 DDR 和 CPU 时钟等。

DDR Configuration 页面用于设置 DDR 控制器配置信息。

PS-PL Configuration 页面用于 PS 和 PL 交互的相关配置，包括常用的中断、复位信号和数据接口。

2-7 配置 PS 的 UART。点击导航面板中 I/O Configuration，出现以下页面：

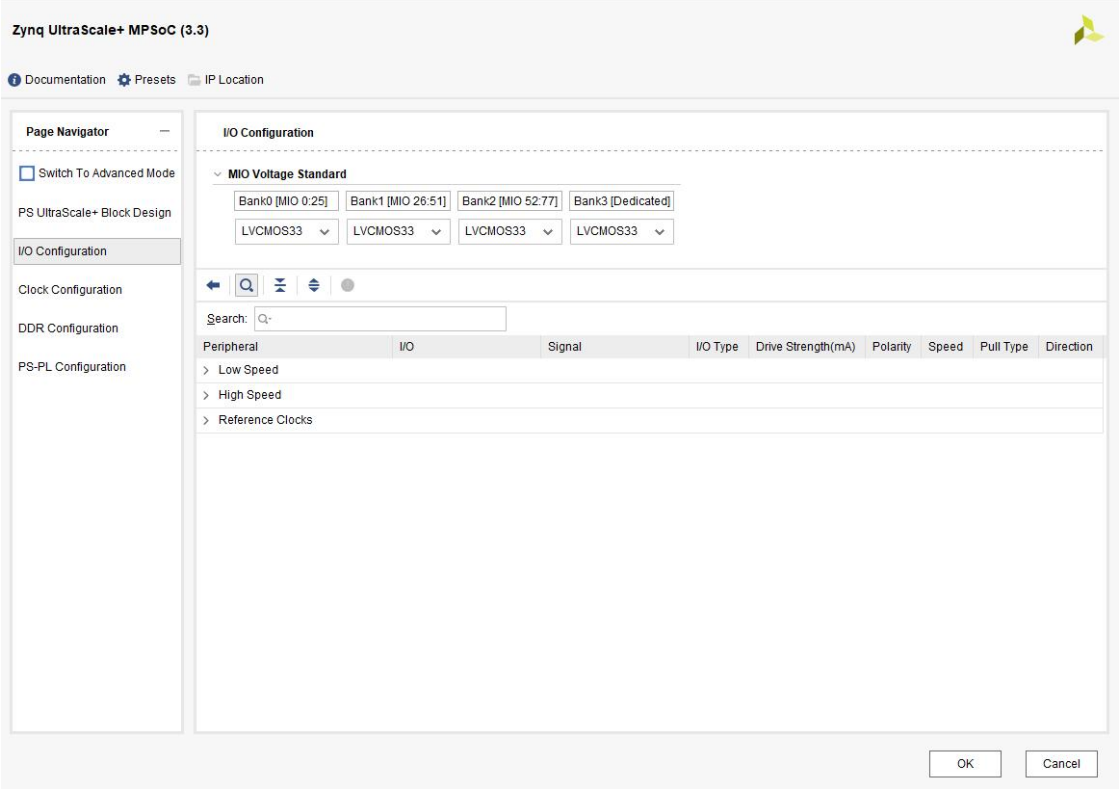


图 1.3.13 配置 MPSOC 的 UART

PS 和外部设备之间的连接主要是通过复用的输入/输出（Multiplexed Input/Output，MIO）来实现的。PS 的 78 个 MIO 引脚可以用于连接不同的外设接口。 UART 引脚接口的选择需要与开发板对应，我们的 MPSOC 开发板 PS 端 UART 接口如下图所示：

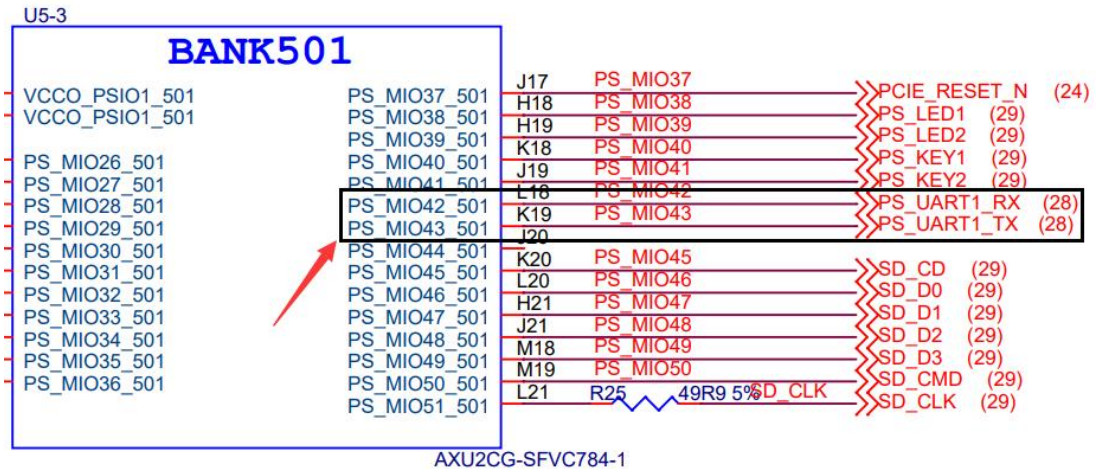


图 1.3.14 开发板原理图

图 1.3.14 是 MPSOC 开发板原理图的一部分。从图中我们可以看到，BANK501 中的 MIO42 和 MIO43 被用作 UART 串口通信的引脚，并最终与开发板上的 USB 转串口芯片 CH340 连接。因此，为了实现串口通信的功能，我们需要在 PS 中将 MIO42 和 MIO43 配置成 UART0 模块的接口引脚。

如下图所示，我们依次展开 Low speed -> I/O Peripherals -> UART：

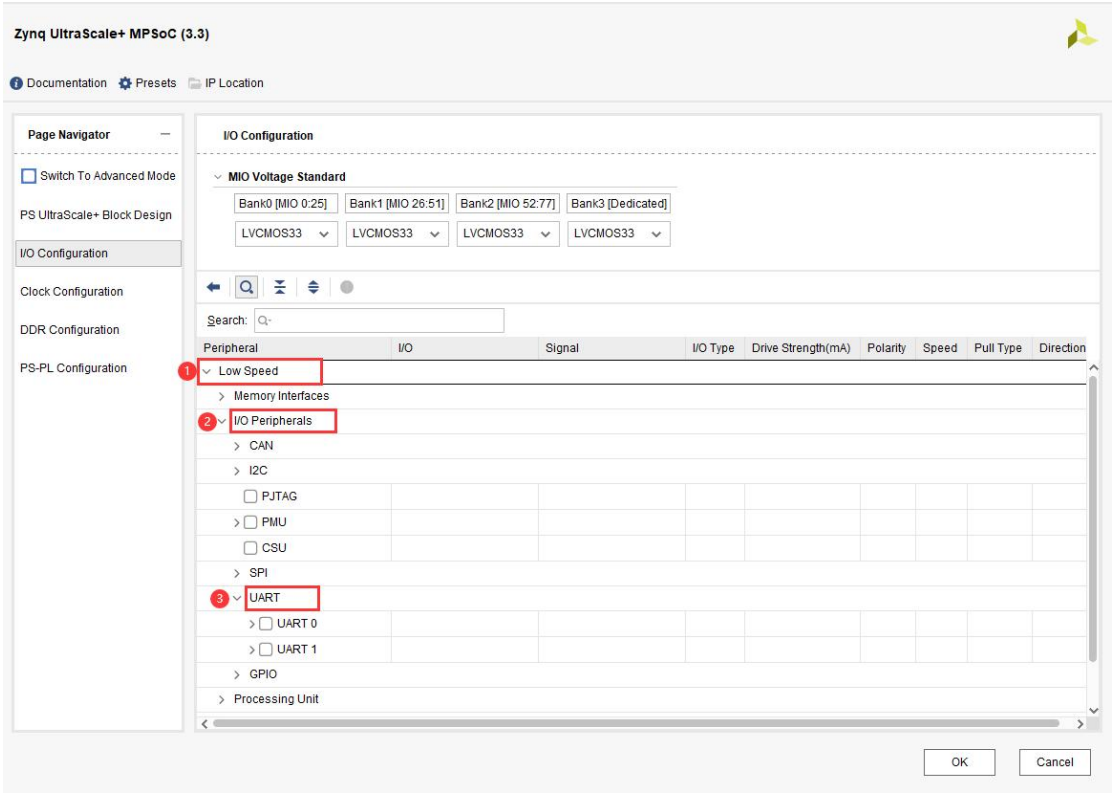


图 1.3.15 展开 UART

勾选 UART0,并在后面的 IO 栏中选择 MIO42..43，如下图所示：

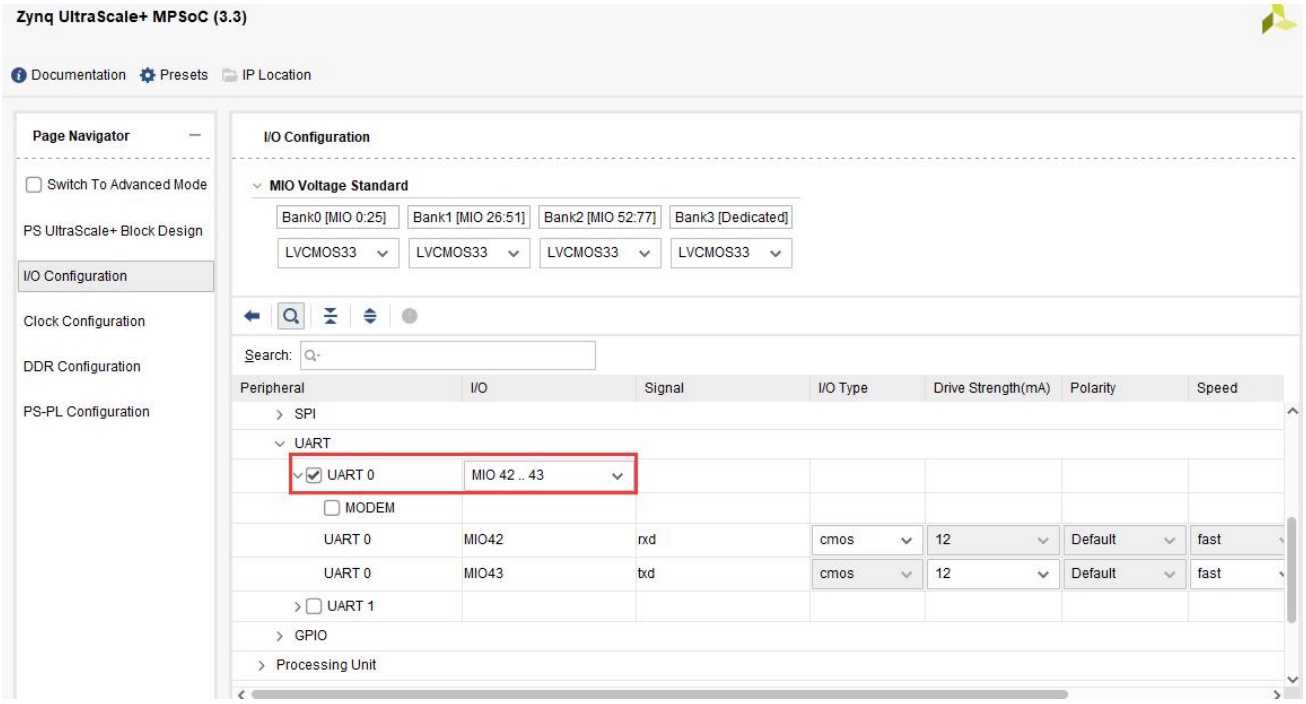


图 1.3.16 UART0 的具体引脚配置

我们的 MPSOC 开发板 bank500 和 bank502 连接的是 1.8V 电压，bank501 和 bank503 连接的是 3.3V 电压，所以将 bank0 和 bank2 电压设置为 1.8V，bank1 和 bank3 电压保持 3.3V 不变，如下图红圈 1 和 2 中所示。

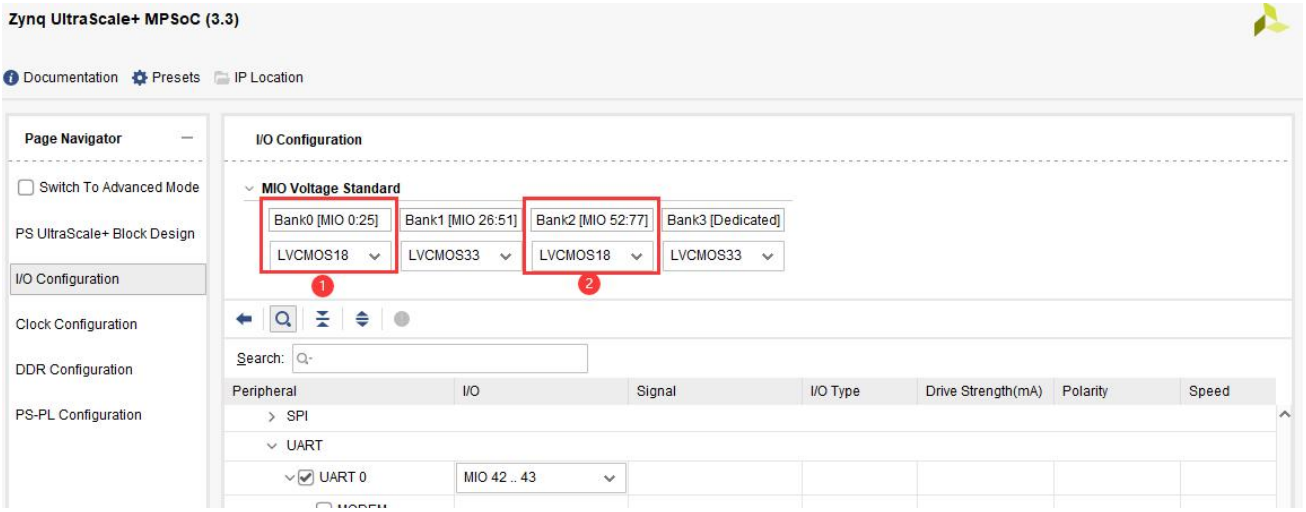


图 1.3.17 bank 电压设置

2-8 配置 PS 的 DDR4 控制器。

我们 MPSOC 开发板上 PS 端使用了 4 片 Micron(镁光)DDR4 芯片，型号是 MT40A256M16GE-083E，容量为每片 4Gbit(256M×16bit)，总位宽为 64bit(16bit×4)，最高运行速度可达 1200Mhz(数据速率 2400Mbps)。

点击导航面板中的 DDR Configuration 选项，打开 DDR 配置页面。在 Load DDR Presets 选项中选择 DDR4_MICRON_MT40A256MGE_083E，在弹出的 Select an option 对话框中，点击“Yes”确定；接着在 DDR Controller Options 一栏中，将 Effective DRAM Bus Width 设置为 64bit；然后在 DDR Memory Options 一栏中，分别按下图红圈中所示设置：

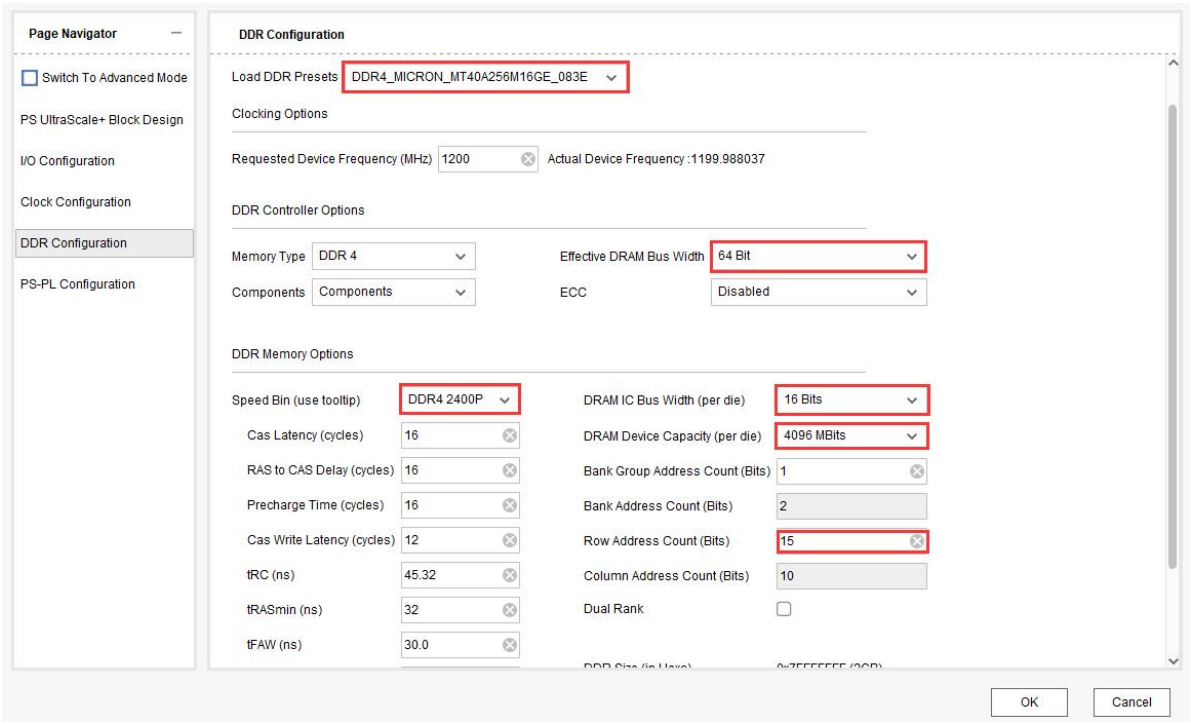


图 1.3.18 配置 PS 的 DDR 控制器

2-9 配置 PS 的时钟。

点击左侧 Clock Configuration 打开时钟配置页面，该界面主要是配置 MPSOC PS 中的时钟频率。比如输入时钟默认是 33.33333Mhz，这与我们开发板上的 PS 端输入时钟频率相同。对于 CPU 的时钟、DDR 的时钟以及其它外设的时钟，我们直接保持默认设置即可。如下图所示：

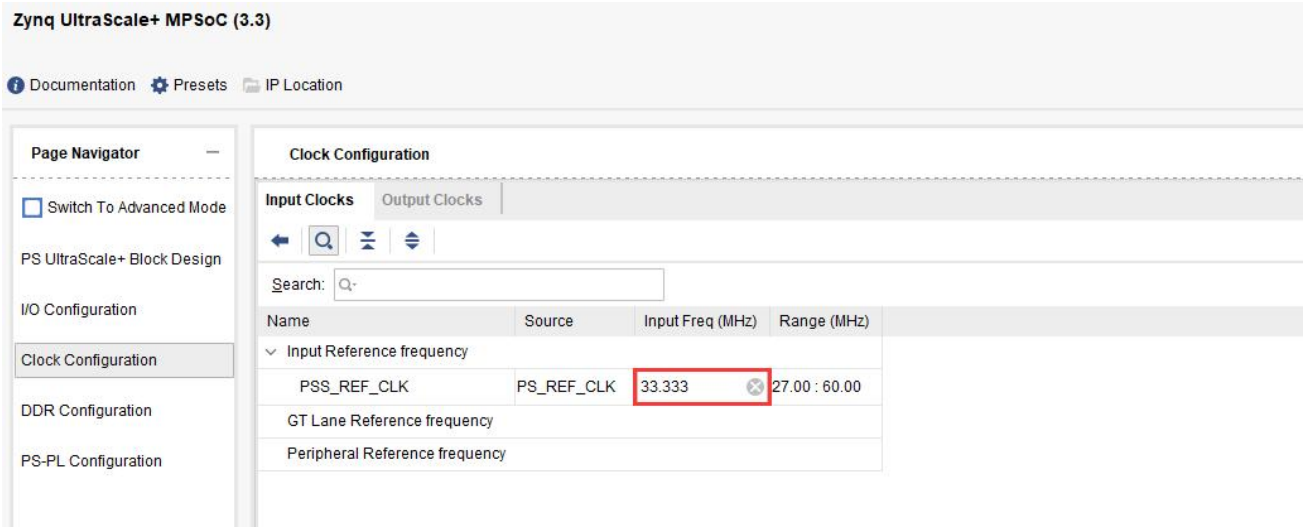


图 1.3.19 配置 PS 的时钟

2-10 因为本实验是搭建 MPSOC 的嵌入式最小系统，只需要使用 MPSOC 中的 PS 端。因此我们将 PS 中与 PL 端交互的接口信号移除。

同样是在 Clock Configuration 页面，点击 Output Clocks 标签页，展开 PL Fabric Clocks，取消勾选 PL0，如下图所示：

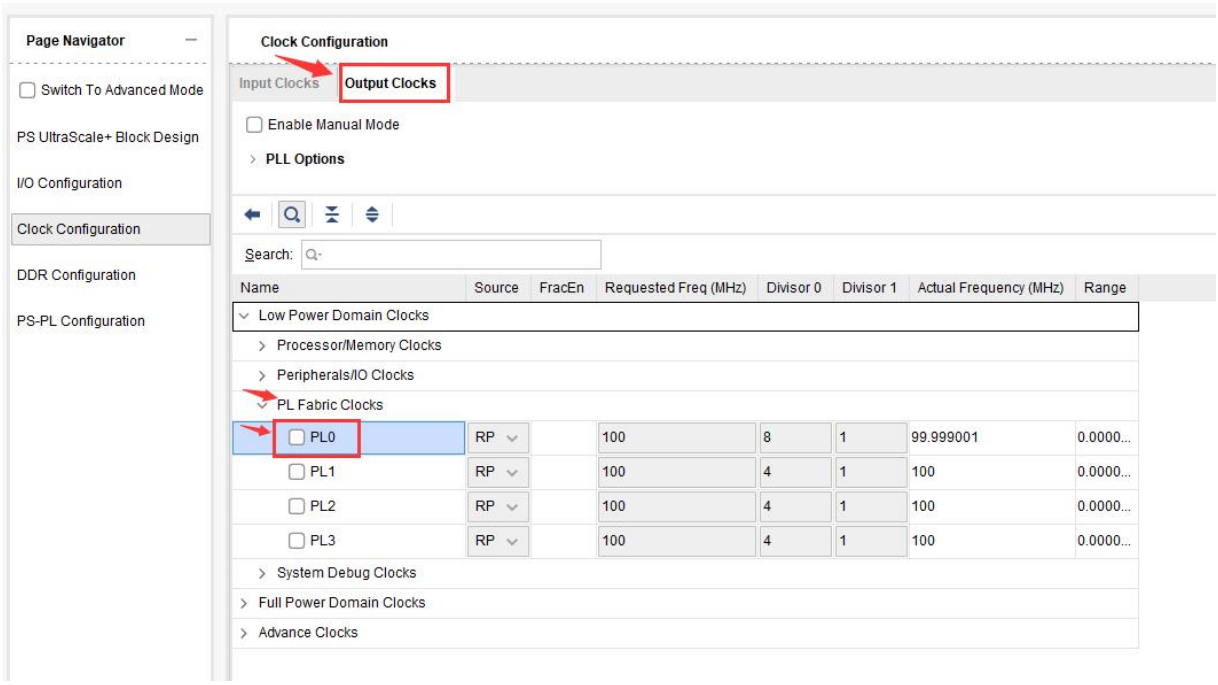


图 1.3.20 去掉勾选 PL0

点击左侧的 PS-PL Configuration 页面，然后在右侧展开 General，取消勾选 Fabric Reset Enable。

另外在当前页面中展开 PS-PL interfaces，可以看到 Master interface 和 Slave interface 两项，展开 Master interface，取消勾选其中的 AXI HPM0 LPD。如下图所示：

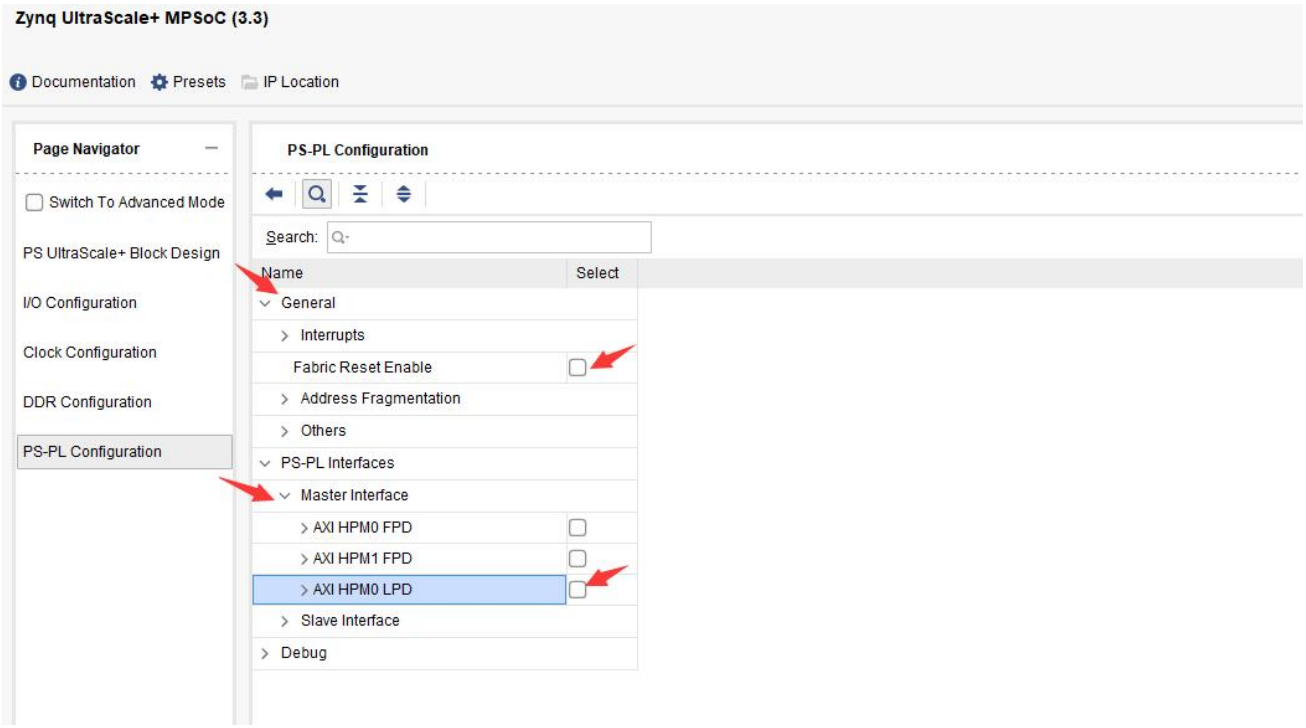


图 1.3.21 取消勾选 Fabric Reset Enable 和 AXI HPM0 LPD

2-11 配置 ZYNQ UltraScale+ MPSOC 完成，点击“OK”。

返回到 Vivado 界面后，在 Diagram 中可以看到 ZYNQ UltraScale+ MPSOC IP 模块发生了变化，如图 1.3.22 所示。 我们将其与图 1.3.11 作对比可以发现，该模块少了四组接口，

这正是因为我们在配置该 IP 核的过程中移除了与 PL 相关的接口信号。

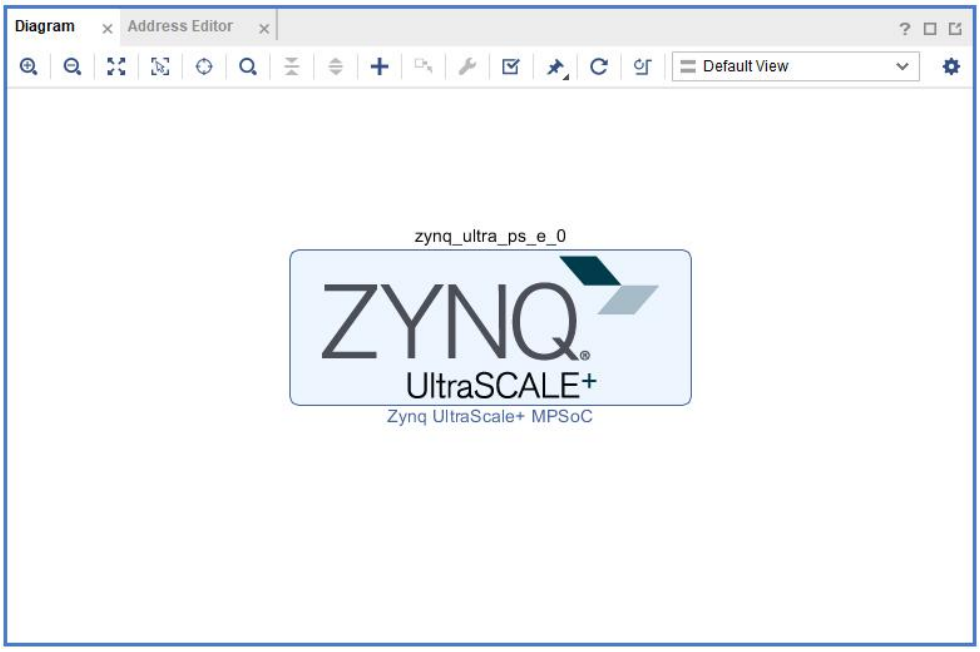


图 1.3.22 配置完成后的 ZYNQ UltraScale+ MPSOC

2-12 我们点击下图中箭头所指示的按钮“validate design”，对我们配置的 IP 核进行验证，如下图所示：

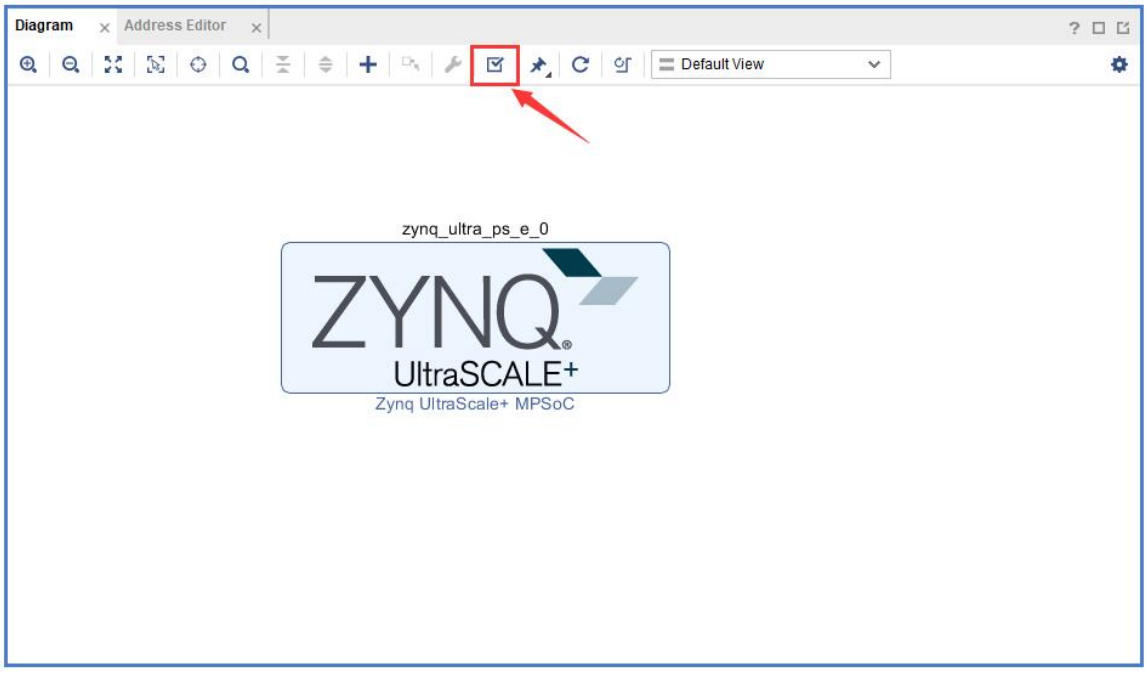


图 1.3.23 点击 validate design 检查 IP 核的配置

检查完成后没有错误，点击 OK，按 Ctrl+S 保存，如下图所示：

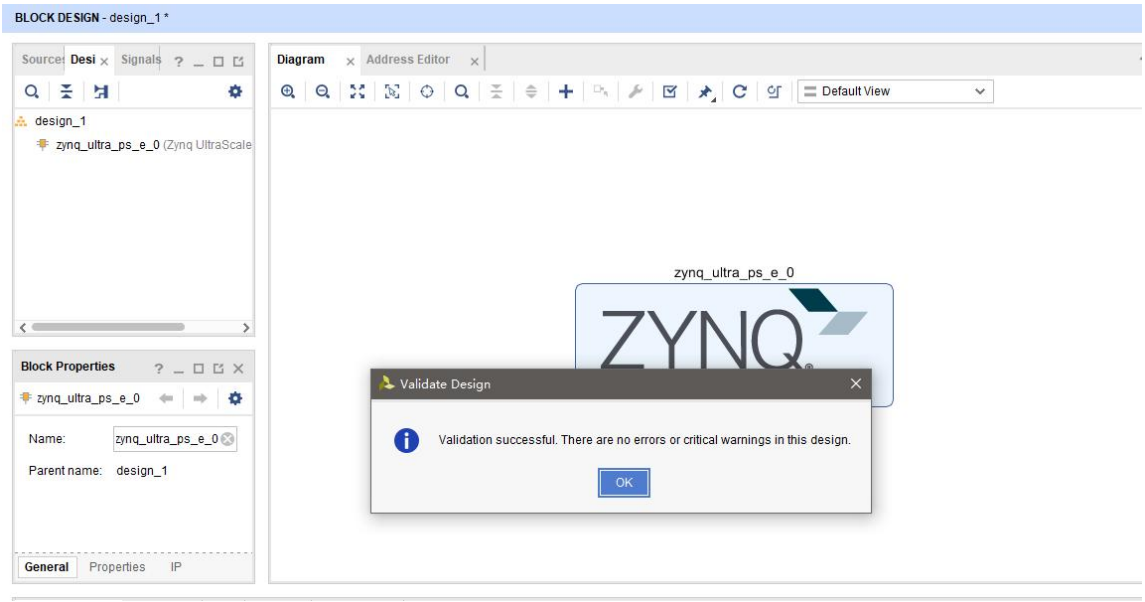


图 1.3.24 检查完成

**step3：生成顶层 HDL 模块**

3-1 在 Sources 窗口中，选中 Design Sources 下的 sysetm.bd，这就是我们刚刚完成的 Block Design 设计。右键点击 sysetm.bd，在弹出的菜单栏中选择“Generate Output Products”，如下图所示：

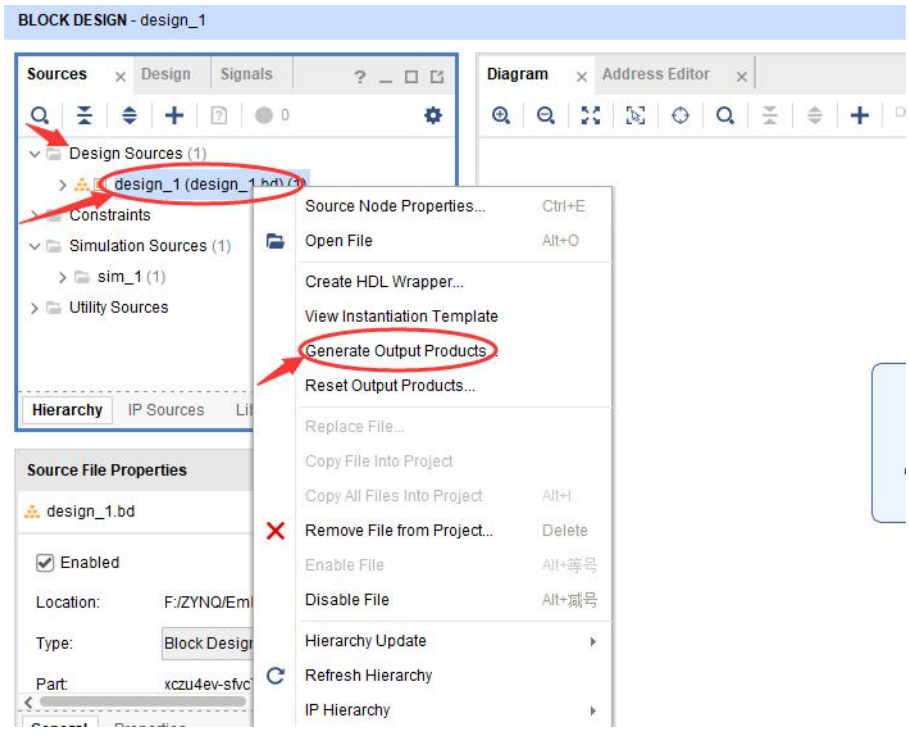


图 1.3.25 选择 Generate Output Products

3-2 弹出“Generate Output Products”对话框，如下图所示：

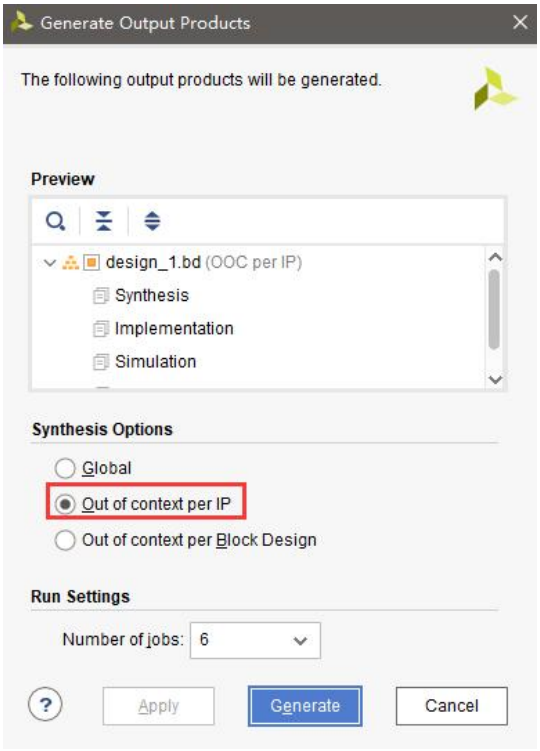


图 1.3.26 设置 Generate 选项

在对话框中，Synthesis Options 选择 Out of context per IP，这里我们保持默认；Run Setings 用于设置生成过程中要使用的处理器的线程数，进行多线程处理，保持默认或设置为个人电脑处理器最大可使用线程数都可以，一般选择最大可使用线程数。然后点击“Generate”来生成设计的综合、实现和仿真文件。

在“Generate”过程中会为设计生成所有需要的输出结果。比如 Vivado 工具会自动生成处理系统的 XDC 约束文件，因此我们不需要手动对 MPSOC PS 引出的接口（DDR 和 FIXED_IO）进行管脚分配。

Generate 完成后，在弹出的对话框中点击“OK”。

在 Sources 窗口中，点击“IP Source”标签页，可以看到 Generate 过程生成的输出结果。

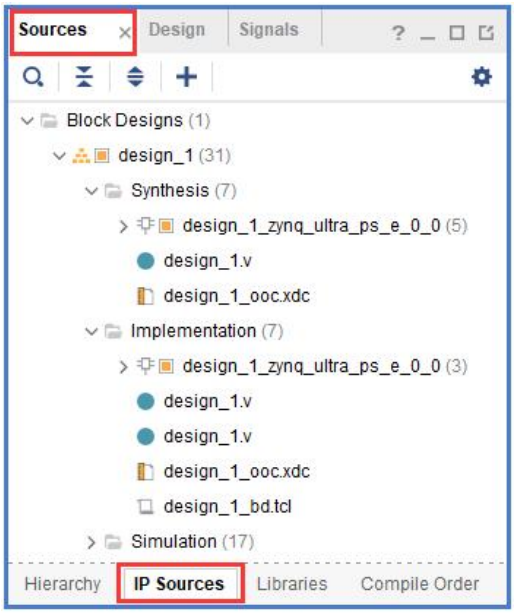


图 1.3.27 block design 生成的结果

3-3 在“Hierarchy”标签页再次右键点击 system.bd，然后选择“Create HDL Wrapper”。

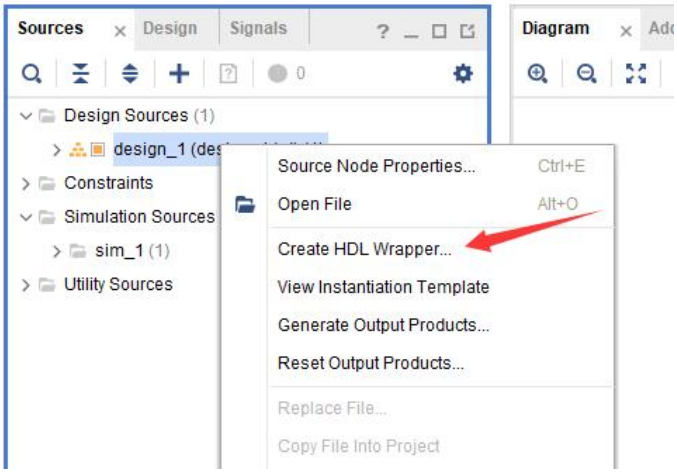


图 1.3.28 生成顶层模块

在弹出的对话框中确认勾选“Let Vivado manage wrapper and auto-update”，然后点击“OK”。

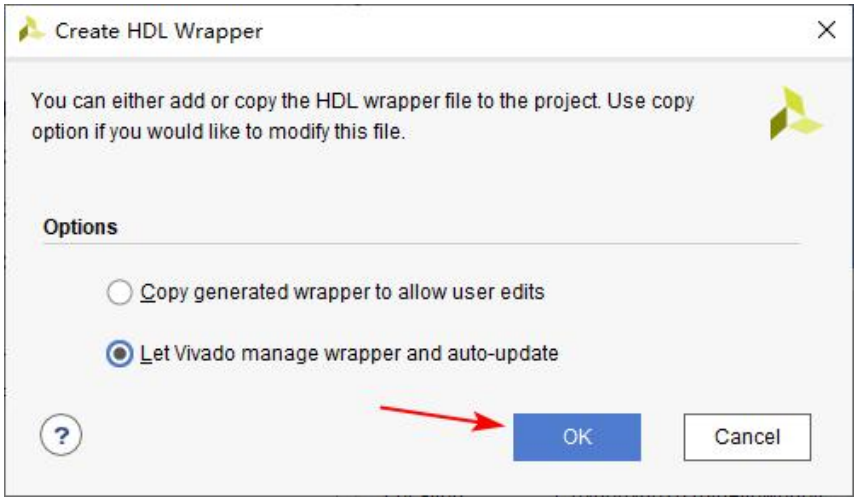


图 1.3.29 创建顶层 HDL 封装

创建完成后，Design Sources 结构如下图所示：

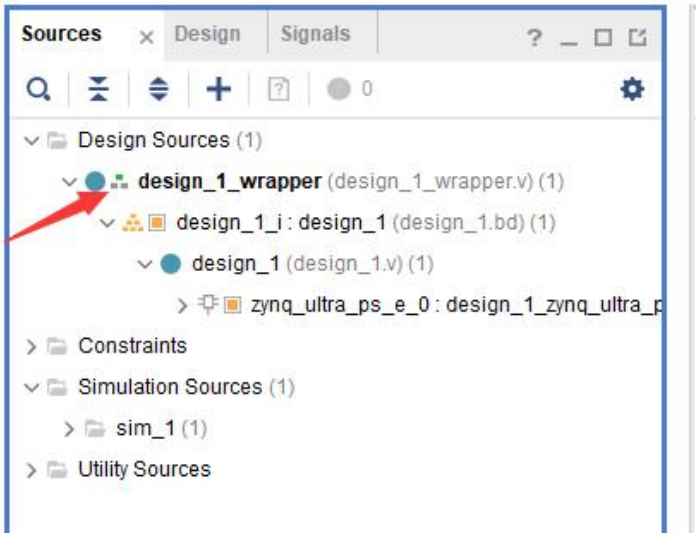


图 1.3.30 生成 design_1_wrapper.v 顶层文件

design_1_wrapper.v 为创建的 Verilog 文件，箭头所指的“品”字形图标指示当前模块为顶层模块。该模块使用 Verilog HDL 对设计进行封装，主要完成了对 block design 的例化，大家也可以双击打开该文件查看其中的内容。

另外我们在图 1.3.29 中勾选了“Let Vivado manage wrapper and auto-update”，这样我们在修改了 Block Design 之后就不需要再重新生成顶层模块，Vivado 工具会自动更新该文件。

**step4：生成 Bitstream 文件并导出 Hardware**

如果设计中使用了 PL 的资源，则需要添加引脚约束并对该设计进行综合、实现并生成 Bitstream 文件。由于本次实验未用到 PL 部分，所以无需生成 Bitstream 文件，只需将硬件导出即可。

4-1 导出硬件。

在菜单栏选择 File > Export > Export hardware。

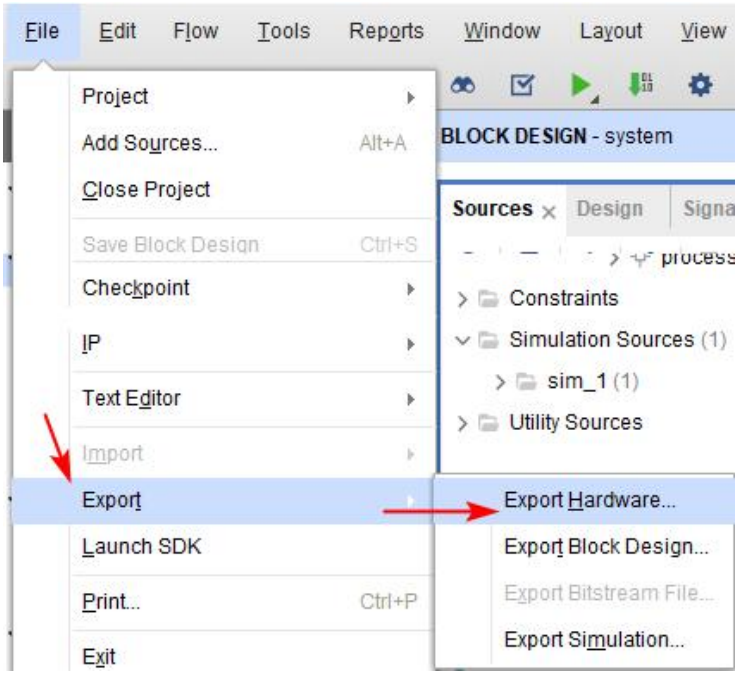


图 1.3.31 导出硬件

在弹出的对话框中，因为没有生成 bitstream 文件，所以无需勾选“Include bitstream”，直接点击“OK”按钮。

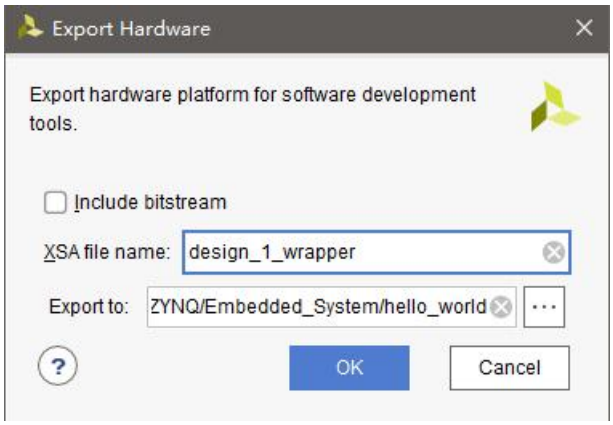


图 1.3.32 无需勾选“Include bitstream”

上图中，XSA file name 一栏是产生的硬件信息文件的文件名，这里我们保持默认。Export to 后面的路径是生成的包含硬件信息文件的路径，生成的文件如下所示：

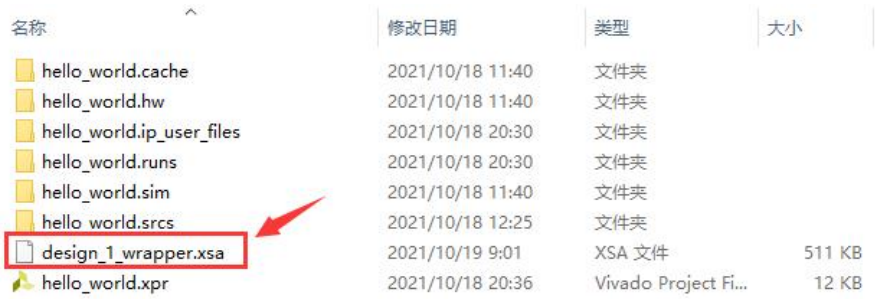


图 1.3.33 生成的 xsa 文件

4-2 硬件导出完成后，我们在工程路径下新建一个名为 vitis 的文件夹，用于存放 vitis 的工程，同时将 design_1_wrapper.xsa 文件拷贝进去，如下图所示：


图 1.3.34 将 xsa 文件拷贝到 vitis 文件夹下

在菜单栏中选择 Tools > Launch Vitis，启动 Vitis 开发环境。如下图所示：

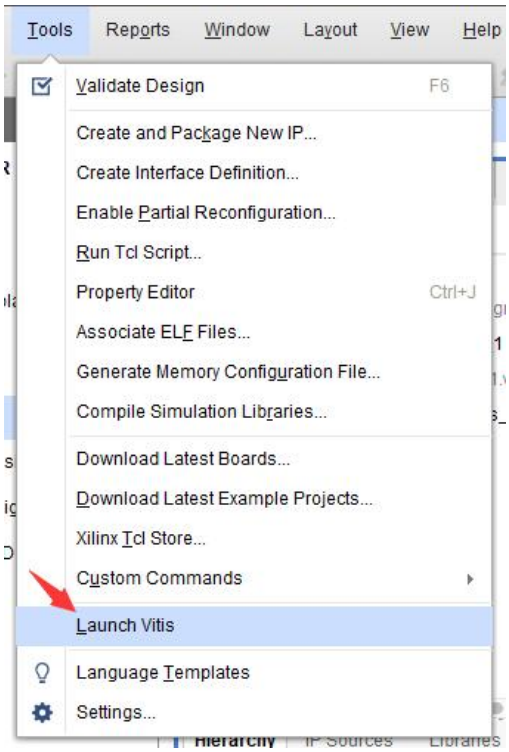


图 1.3.35 启动 Vitis 开发环境

在弹出的对话框中，我们将工程路径指定到新建的 vitis 文件夹下，如下图所示：

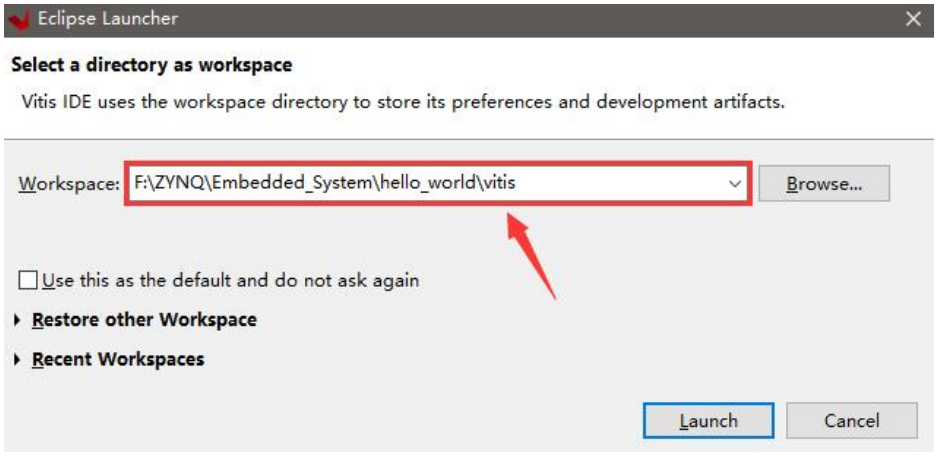


图 1.3.36 设置工作空间

到这里，我们已经完成了 MPSOC 嵌入式系统的硬件设计部分。接下来需要到 Vitis 软件中进行应用程序开发，也就是软件设计部分。

## 1.4 软件设计

在硬件设计的最后，我们启动了软件开发环境(Vitis)，如下图所示：

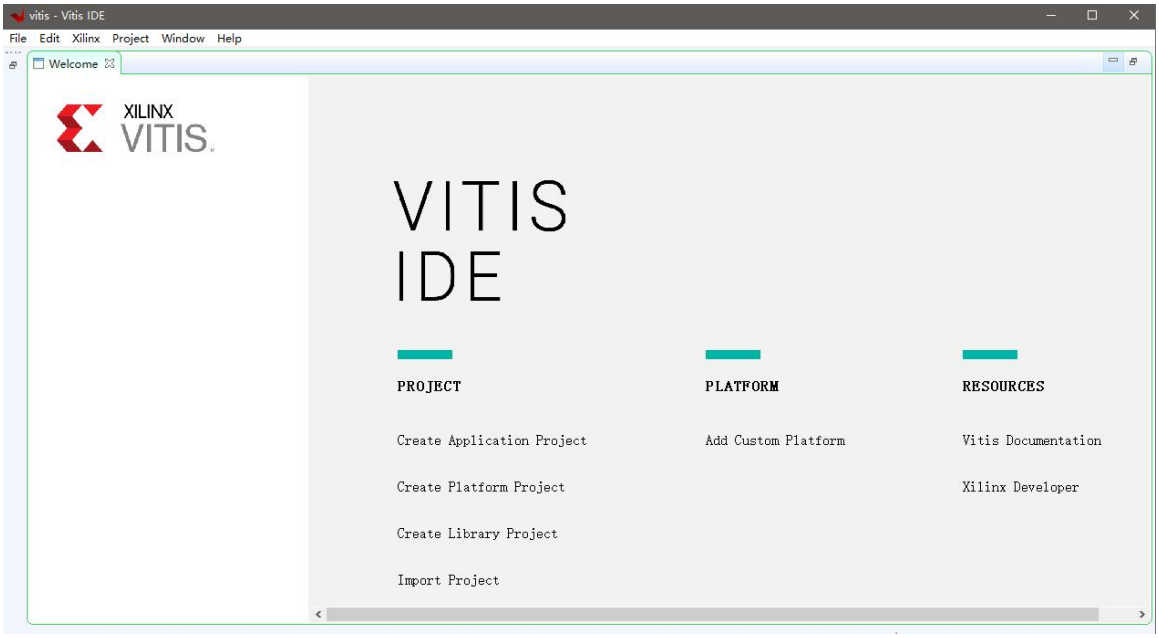


图 1.4.1 Vitis 开发环境界面

**step5：在 Vitis 中创建应用工程**

5-1 在菜单栏选择 File > New > Application Project，新建一个 vitis 应用工程，如下图所示：

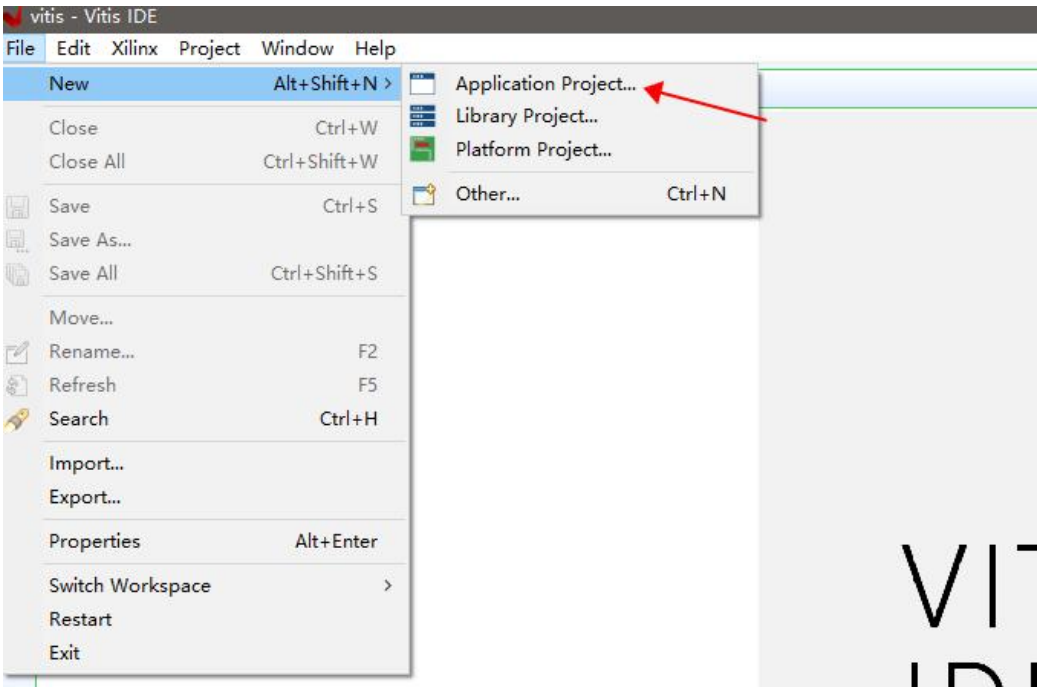


图 1.4.2 新建应用工程

5-2 在弹出的对话框中，输入工程名“hello_world”，其它选项保持默认即可，点击“Next”，如下图所示：

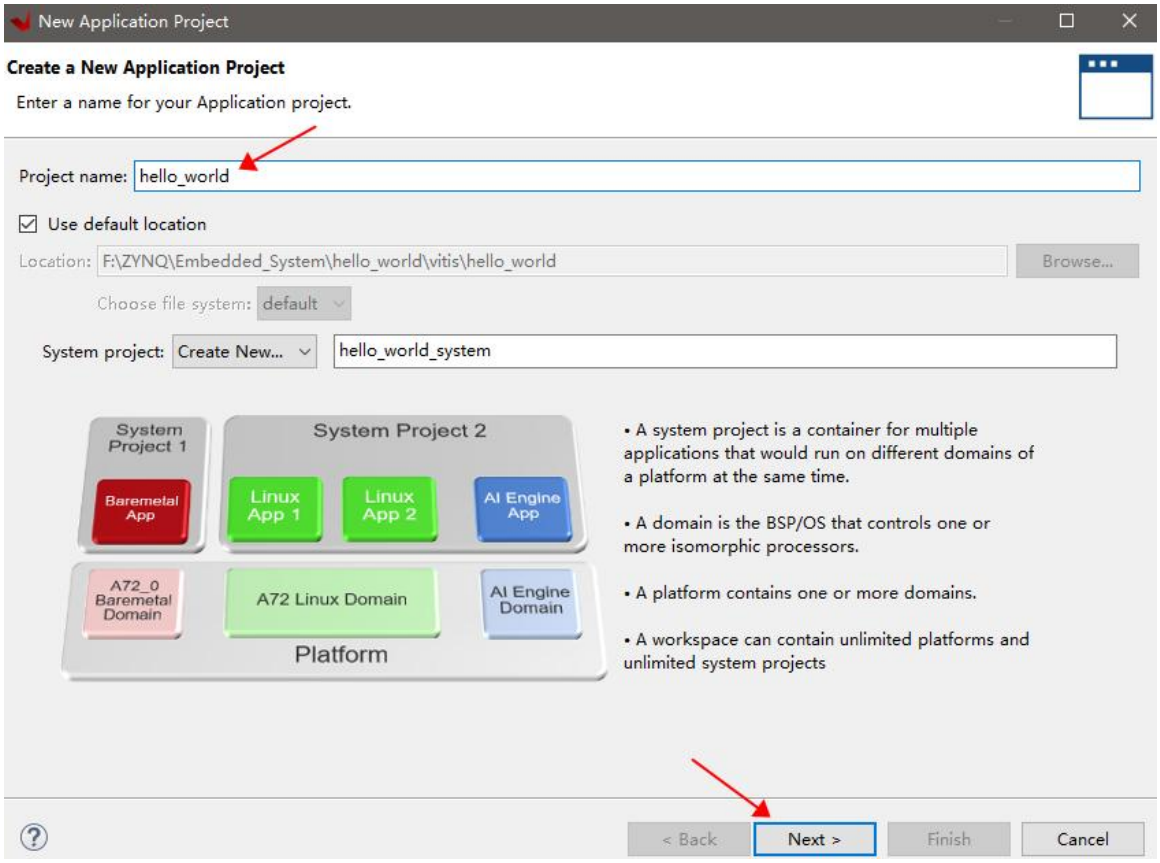


图 1.4.3 配置工程

5-3 打开 Create a new platform from hardware(XSA)标签页，点击“+”添加 xsa 文件，如下图所示：

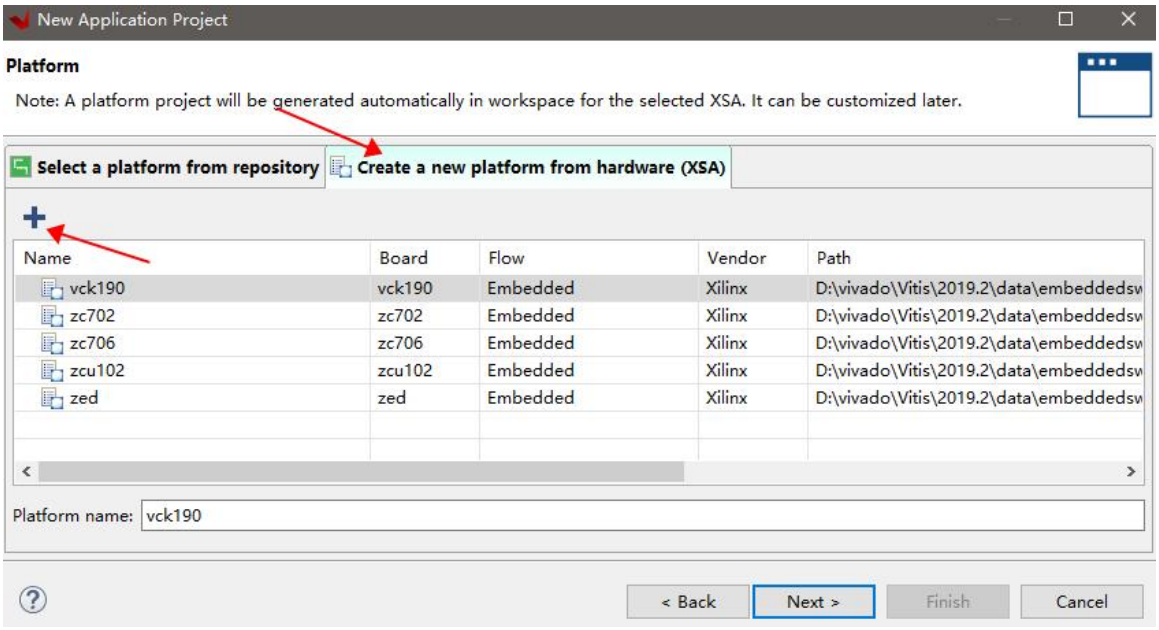


图 1.4.4 添加 xsa 文件

在弹出的窗口中选择 design_1_wrapper.xsa 文件，如下图所示：


图 1.4.5 选择 design_1_wrapper.xsa 文件

添加 xsa 文件后的页面如下图所示，点击 next：

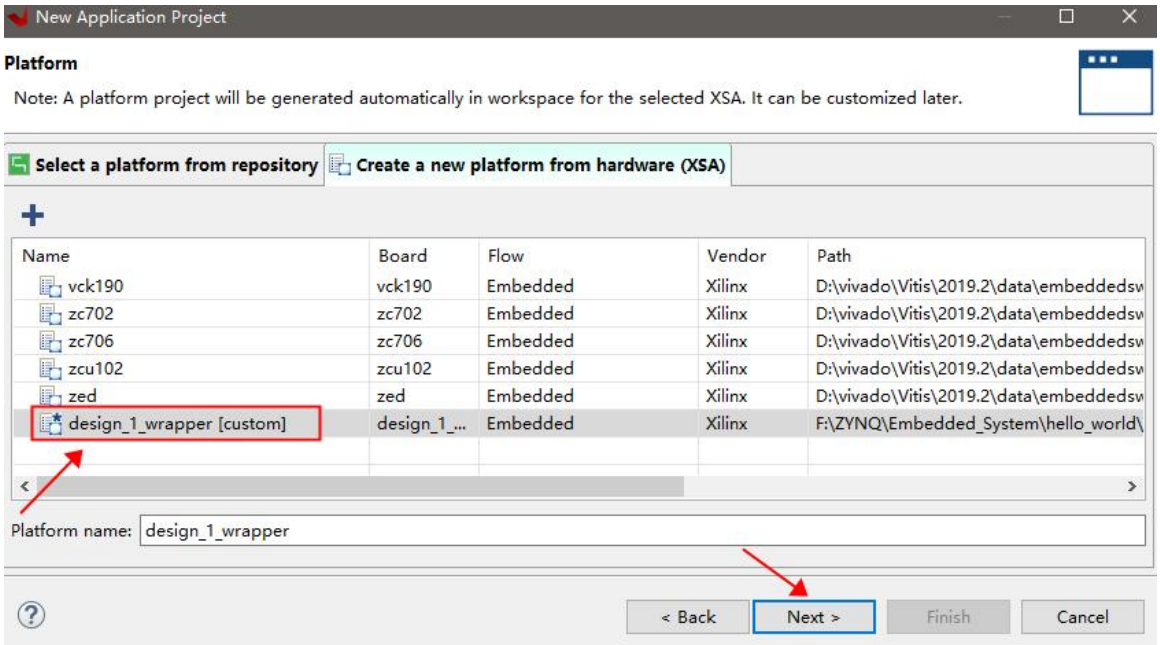


图 1.4.6 添加完成 xsa 文件

5-4 在弹出的页面中有一个 Generate boot components 选项，如果勾选，软件会自动生成

fsbl 工程，这里我们选择默认勾选，然后点击 next，如下图所示：

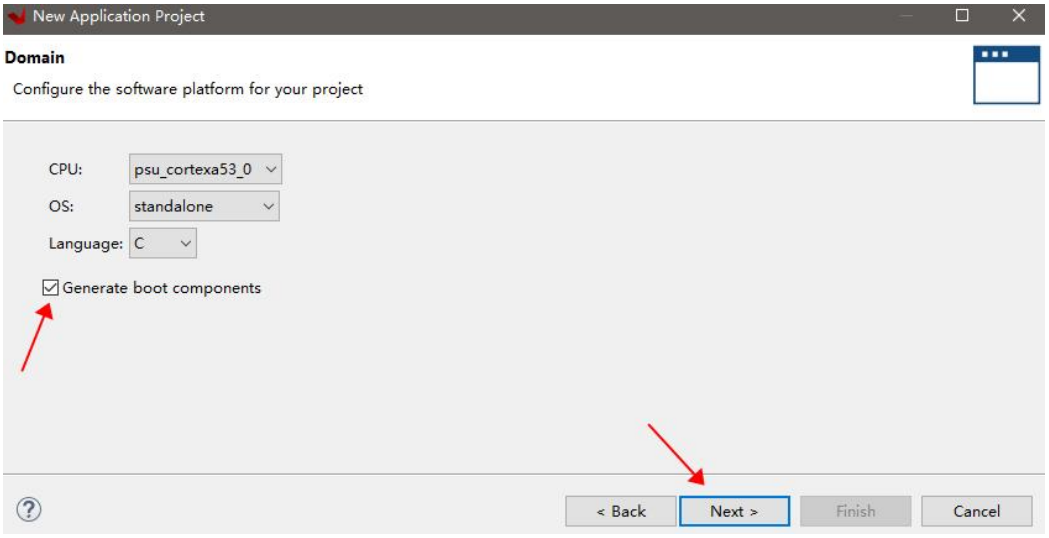


图 1.4.7 默认生成 fsbl 工程

5-5 在弹出的工程模板选择页面里，我们选择已有的 Hello World 模板，然后点击 Finish，如下图所示：

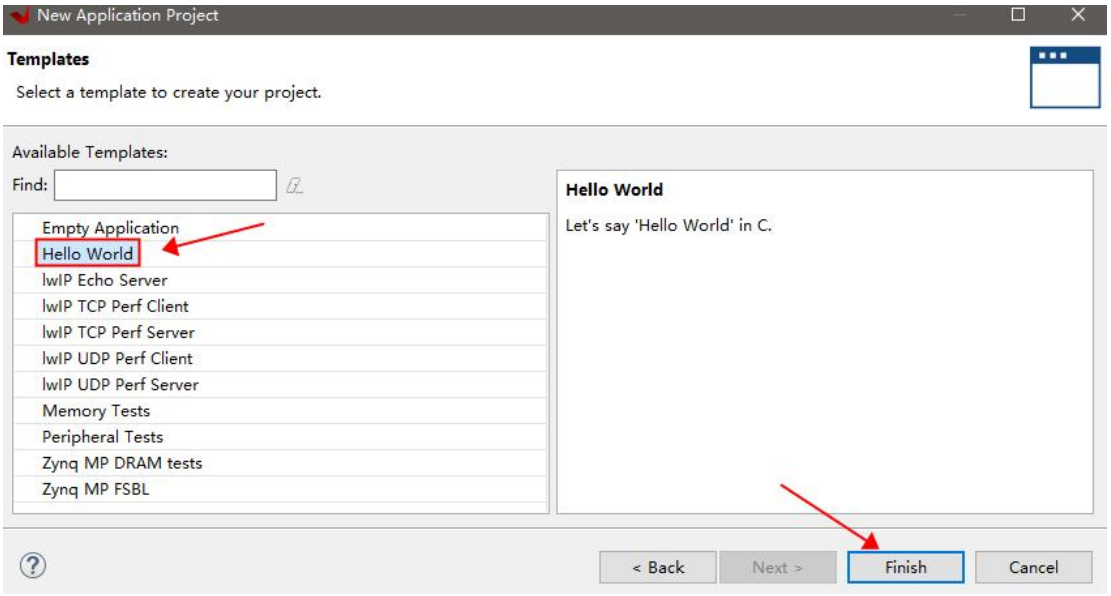


图 1.4.8 选择 Hello World 模板

工程建立完成后的页面如下图所示，我看可以看到生成了两个工程，一个是硬件平台工程，即 platform 工程，一个是应用工程。

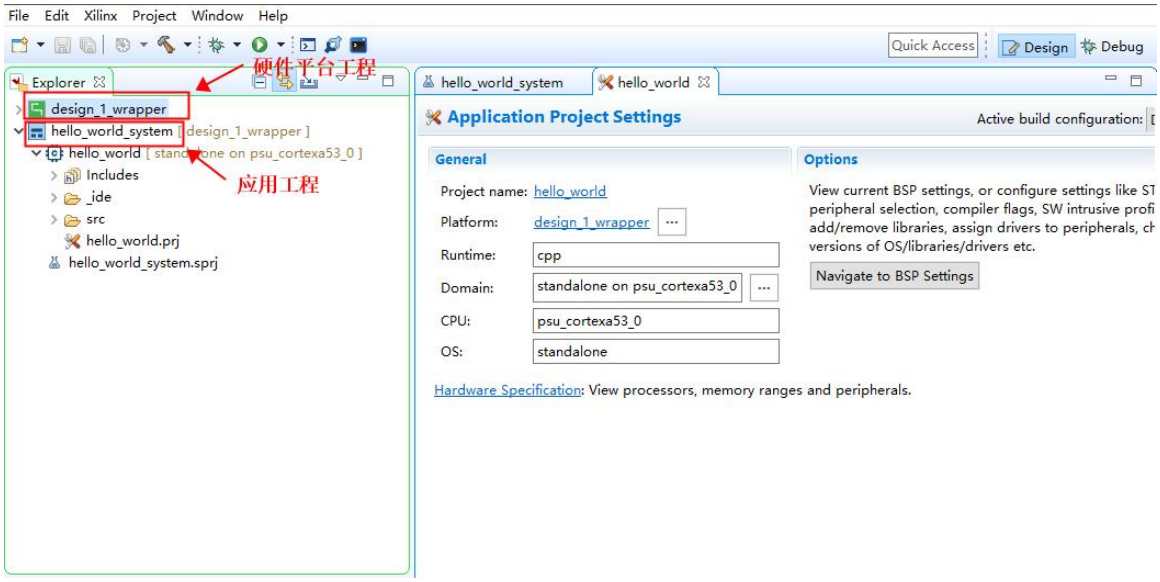


图 1.4.9 工程建立完成

5-6 双击打开 hello_world/src 工程目录下 helloworld.c 文件，可以看到源代码如下：

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"

int main()
{
    init_platform();

    print("Hello World\n\r");

    cleanup_platform();
    return 0;
}
```

可以看到程序中主函数调用了 3 个函数，分别是 init_platform()、cleanup_platform()和 print()函数。我们将鼠标停留在各个函数名上，vitis 就会显示该函数的声明。如果想查看函数的定义，可以按住 Ctrl 键不放，用鼠标点击相应的函数，就会跳转到其定义的地方。

init_platform 和 cleanup_platform 函数定义如下：

```
void
init_platform()
{
    /*
     * If you want to run this example outside of SDK,
     * uncomment one of the following two lines and also #include "ps7_init.h"
     * or #include "ps7_init.h" at the top, depending on the target.
     * Make sure that the ps7/psu_init.c and ps7/psu_init.h files are included
     * along with this example source files for compilation.
     */
    /* ps7_init();*/
    /* psu_init();*/
    enable_caches();
    init_uart();
}

void
cleanup_platform()
{
    disable_caches();
}
```

图 1.4.10 init_platform 和 cleanup_platform 函数定义

可以看到 init_platform 函数的作用是使能 caches 和初始化 uart；cleanup_platform 函数的作用是取消使能 caches。实际上这两个函数在该工程中并没有启动任何作用，因为这两个函数是针对于特定平台如 Microblaze 的，对于我们使用的 MPSOC 平台而言是不起作用的，所以 main 函数中只需包含第 9 行的 print 语句就可以了，出于平台的通用性和可移植性，此处我们保留这两个函数。

另外需要注意程序中打印字符串“Hello World”使用的是 print()函数，而不是 C 语言里的 printf()函数。print()函数是 Xilinx 定义的一个用于打印字符串的函数，调用该函数需要包含头文件“xil_printf.h”。

5-7 选中应用工程，右键 Build Project 对工程进行编译。

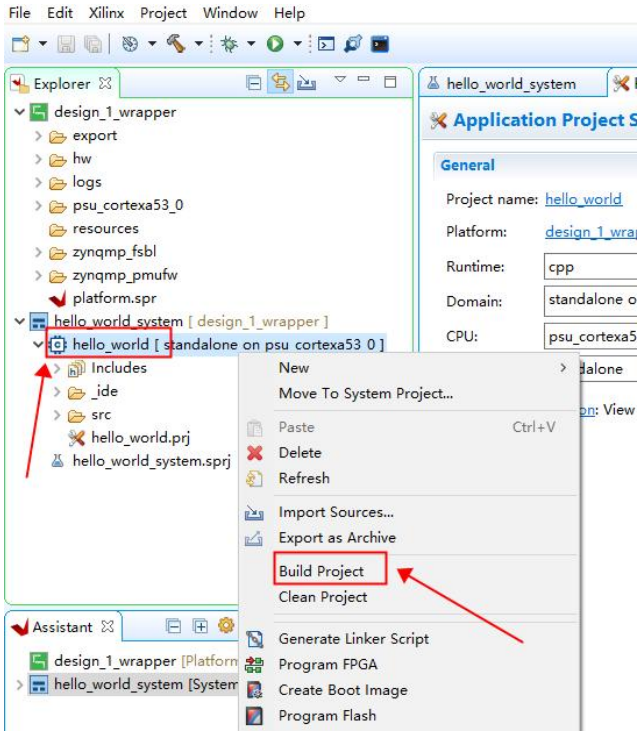


图 1.4.11 编译工程

编译进度可以在工具下方的控制台面板（Console）中进行查看，编译完成后显示“Finished building：hello_world.elf”，如下图所示：

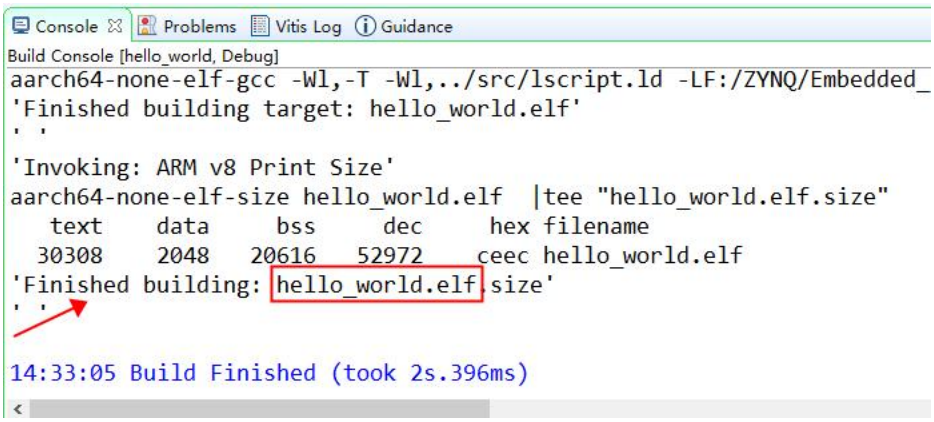


图 1.4.12 编译完成

到这里我们已经完成了本次实验的软件设计部分。

## 1.5 下载验证

首先我们将下载器与 MPSOC 开发板上的 JTAG 接口连接，下载器另外一端与电脑连接。然后使用 USB 连接线将开发板 USB_UART (PS_PORT)接口与电脑连接，用于串口通信。接下来将开发板上四个启动模式开关均置为 ON，即设置为 JTAG 模式。最后连接开发板电源给开发板上电。如下图所示：

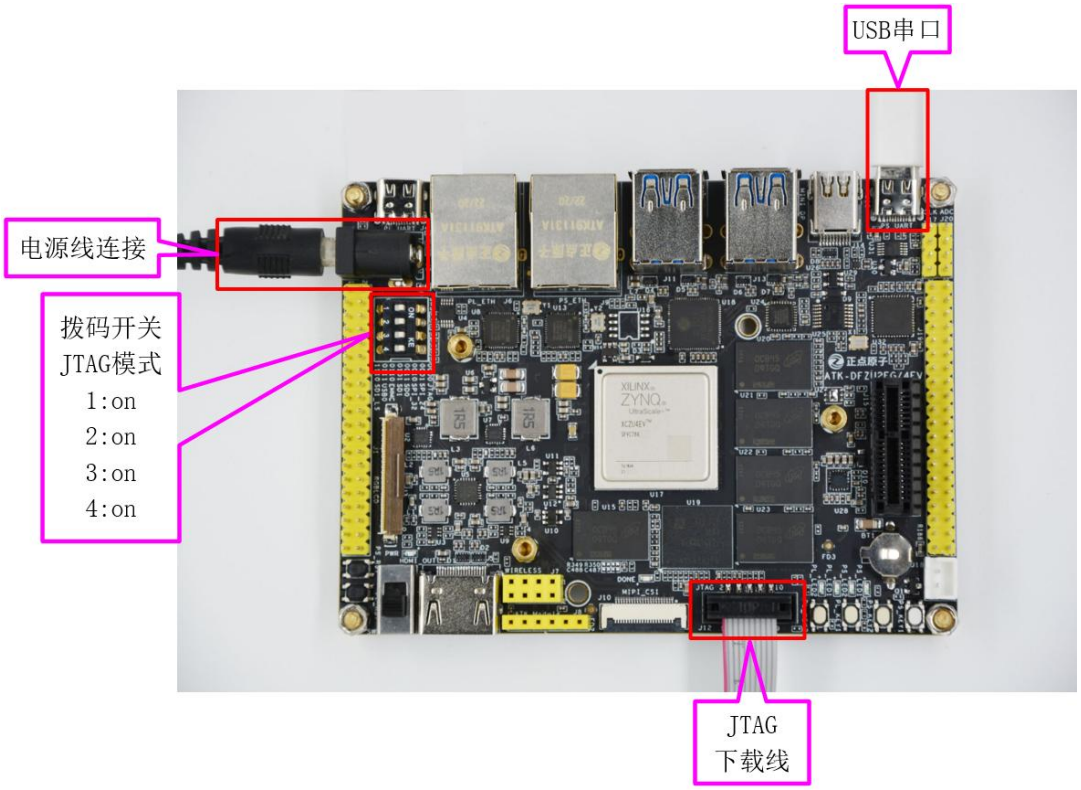


图 1.5.1 MPSOC 开发板实物图

注意第一次连接开发板 USB_UART 接口时，需要安装 USB 串口驱动。在开发板随附的资料中找到“6_软件资料/1_软件/CH340 驱动(USB 串口驱动)”文件夹，双击打开文件夹中的“SETUP.EXE”进行安装，驱动安装界面如下图所示。界面中提示 INF 文件为 CH341SER.INF，我们不需要理会（CH341，CH340 驱动是共用的），直接点安装即可。


图 1.5.2 安装 USB 串口驱动

**step6：板级验证**

6-1 在 Vitis 软件的下方，找到 Terminal 窗口。如果界面中没有找到该窗口，或者操作过程中把该窗口给关闭了，则可以通过在菜单栏中选择 Window > Show View > Other，在 Show View 窗口中搜索添加 Terminal。

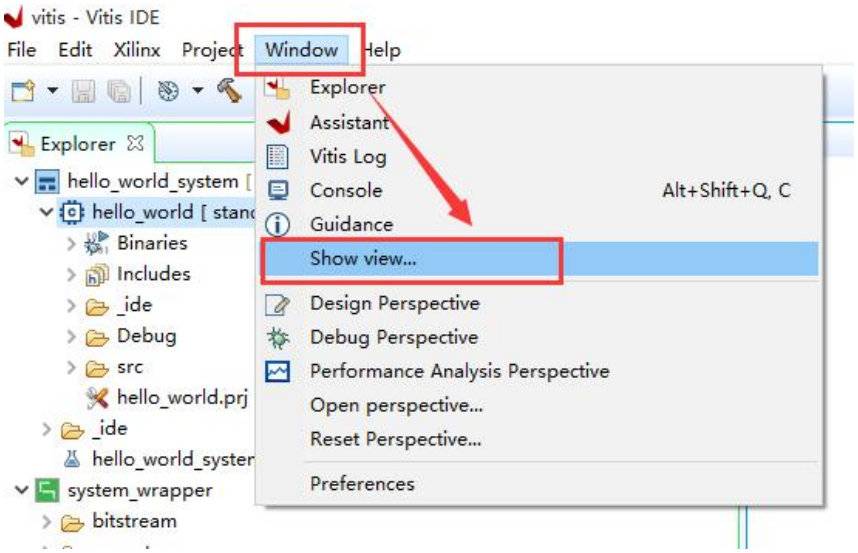


图 1.5.3 搜索添加 Terminal

点击 Open 添加 Terminal：

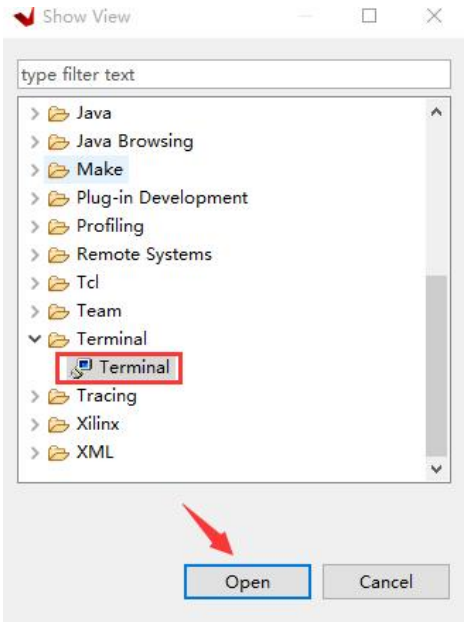


图 1.5.4 添加 Terminal

添加 Terminal 后如下图所示，点击箭头处的图标对串口进行设置。

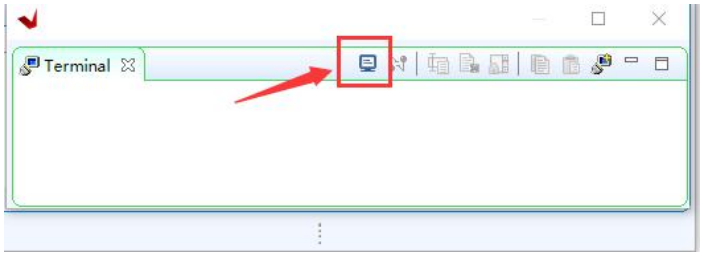


图 1.5.5 Vitis 自带的 Terminal 终端

在弹出的窗口中，Choose terminal 一栏中，下拉选择 Serial Terminal 串口终端，如下图所示：

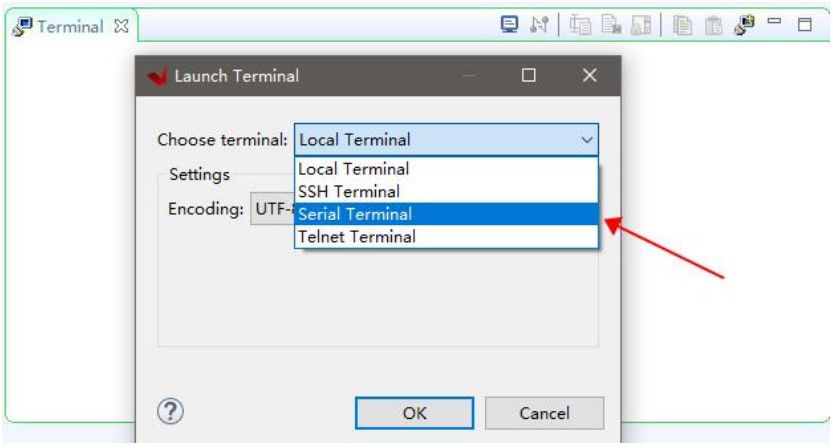


图 1.5.6 选择串口终端

选择串口终端后，接下来需要对串口设置。这里设置波特率为 “115200”，数据位为 8 位，停止位为 1 位，然后点击 OK，如下图所示：

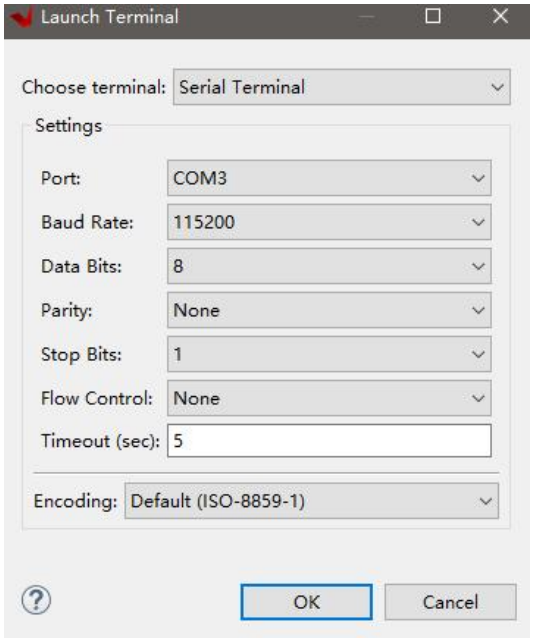


图 1.5.7 配置串口终端

配置完成，连接成功后如下图所示：

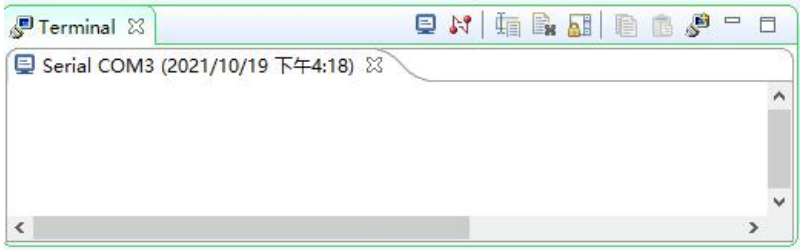


图 1.5.8 串口连接成功

需要注意的是，在设置串口端口（Port）时，在下拉列表中可能会看到多个可选端口。我们需要选择与开发板上的串口所连接的端口，具体的端口号可在计算机设备管理器中查看。因为开发板上使用的 USB 转串口芯片型号为 CH340，因此在设备管理器中找到 USB-SERIAL CH340 所对应的端口，在我这台电脑上该端口号为 COM3，如下图所示：

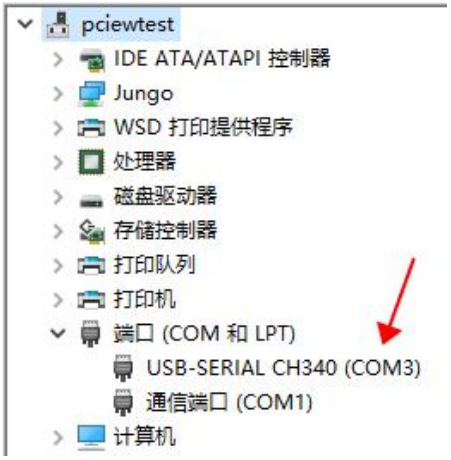


图 1.5.9 查看串口端口

6-2 下载程序。右键点击 hello_world 工程，选择“Run As”，然后选择最后一项“Run Confagurations…”，如下图所示：

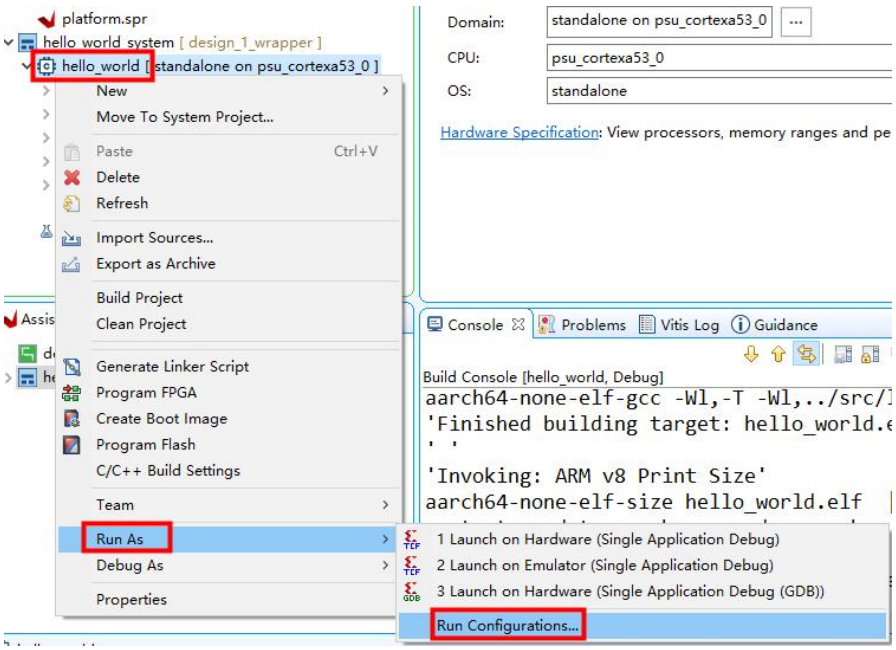


图 1.5.10 打开下载页面

在打开的下载页面中，没有出现下载选项，这时需要双击左侧列表中 Single Application Debug 一项，双击后，该项下面出现新的项 Debugger_hello_world-default，同时在右侧出现的页面中选择 Target Setup 标签页，勾选复位，然后点击 run 下载程序，如下图所示：

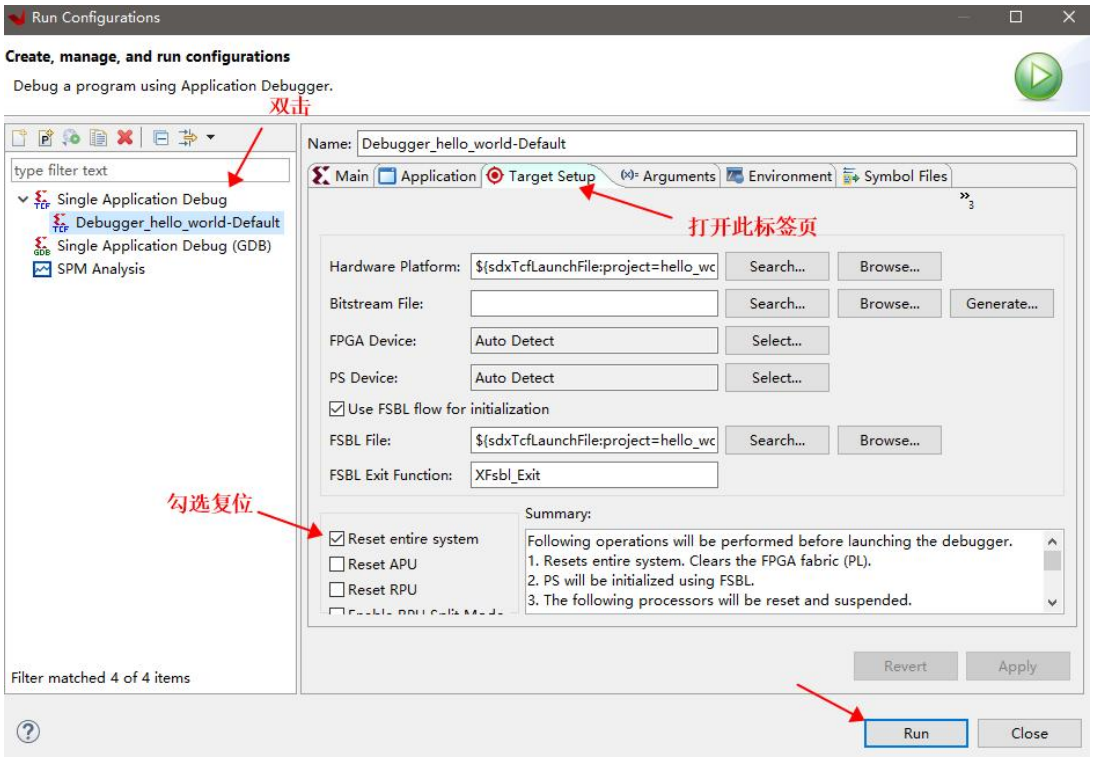


图 1.5.11 下载程序

6-3 下载完成后，应用程序会将字符串“Hello World”通过 MPSOC PS 端的串口模块发送出去。在 Terminal 窗口可以看到上位机接收到的字符串，如下图所示：

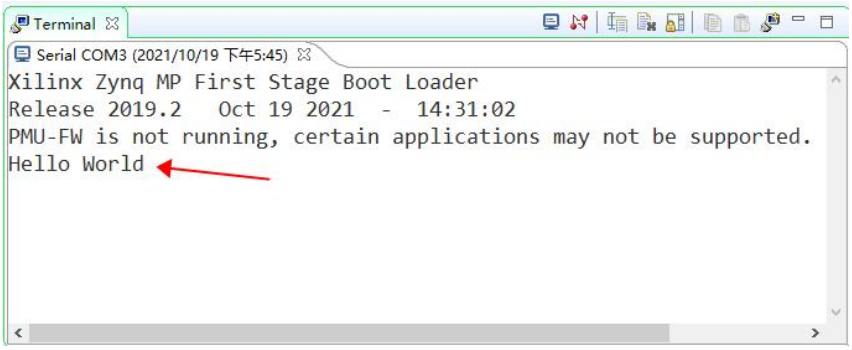


图 1.5.12 程序运行结果

程序成功打印出了“Hello World”字符串，说明本次实验在 MPSOC 开发板上面下载验证成功。