

# *Lab8: 板载温度计测试实验*

*基于 Nexys 4 FPGA 平台*

## Lab 8: 板载温度计测试实验

---

### 实验简介

---

本实验旨在使读者进一步熟悉 Xilinx 的 XPS 和 SDK 工具的使用，并初步掌握对板载温度计 IP 核的添加方法，最终完成一个通过串口输入数据来查看板载温度计数据的简单程序。

---

### 实验目标

---

在完成本实验后，您将学会：

- 如何在 XPS 工具中添加并根据设计需求调整板载温度计的 IP 核。
- 如何通过 C 语言实现对板载温度计的控制，并通过串口接收温度计数据。

---

### 实验过程

---

本实验旨在指导读者使用 Xilinx 的 XPS 工具，调用板载温度计的 IP 核，并将导入到 SDK，通过在串口输入数据来选择相应的功能，查看板载温度计的实时数据，然后在 Nexys 4 平台上进行测试验证。

实验由以下步骤组成：

1. 在 XPS 中建立工程
2. 添加 IP 核并调整相关设置
3. 进行端口的互连
4. 将工程导入到 SDK
5. 在 SDK 中添加 c 语言源程序
6. 在 Nexys 4 上进行测试验证

---

### 实验环境

---

- ◆ 硬件环境
  1. PC 机
  2. Nexys 4 FPGA 平台
- ◆ 软件环境
  - Xilinx ISE Design Suite 14.3 (FPGA 开发工具)

## 第一步 创建工程

1-1. 运行 **Xilinx Platform Studio**, 创建一个空的新工程, 基于 **xc6slx45csg484-3** 芯片和 **VHDL** 语言.

1-1-1. 选择 开始菜单 > 所有程序 > **Xilinx Design Tools** > **ISE Design Suite 14.3** > **EDK** > **Xilinx Platform Studio**。点击运行 **Xilinx Platform Studio(XPS)** (Xilinx Platform Studio 是 ISE 嵌入式版本 Design Suite 的关键组件, 可帮助硬件设计人员方便地构建、连接和配置嵌入式处理器系统, 能充分满足从简单状态机到成熟的 32 位 RISC 微处理器系统的需求), 如图 1 所示。

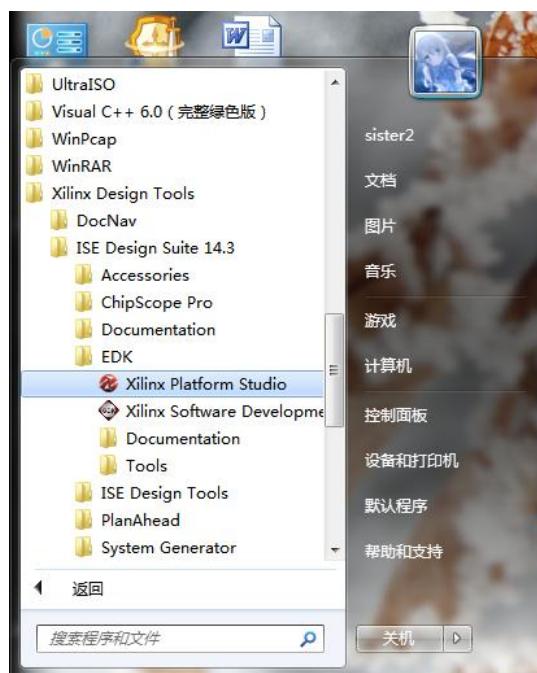


图 1: XPS 软件打开位置

1-1-2. 点击 **Create New Project Using Base System Builder** 来打开新工程建立向导。会出现一个 **Create New XPS Project Using BSB Wizard** 对话框, 如图 2。



图 2：新工程建立界面

- 1-1-3. 如图 3，在新工程建立向导对话框的 **Project File** 栏选择工程建立后存放的路径，笔者自定义的路径是 I:\labz\wendu，大家可以建立任意路径，只要路径名称不包含中文及空格即可。建立的工程的名我们使用 wendu。点击 **OK**。

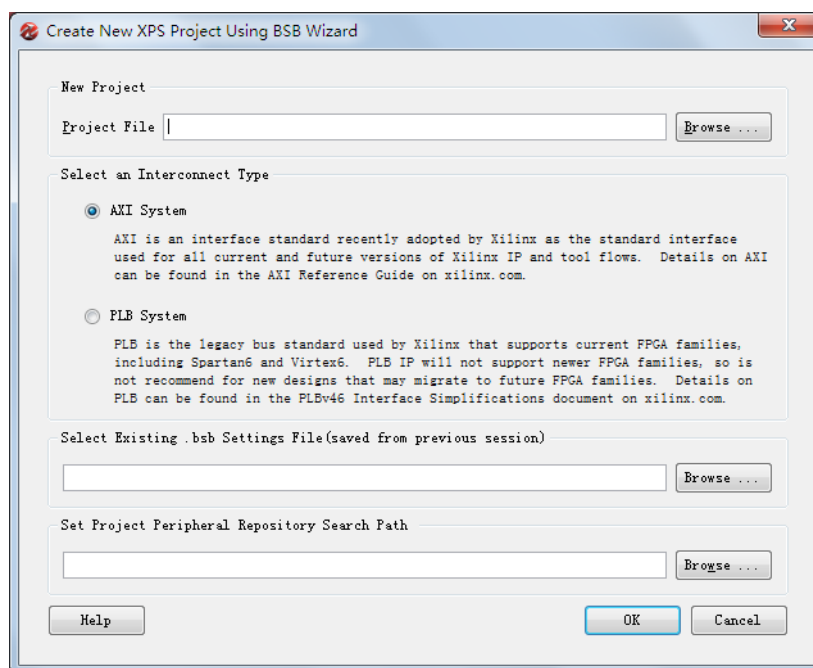


图 3：新工程建立向导

- 1-1-4. 新出现的是关于工程的一些参数设置的对话框，设置如下的参数后，点击 **Next**，如图 4。

**architecture: artix 7**

**Device: XC7a1007**

**Package: CSG324**

**Speed: -3**

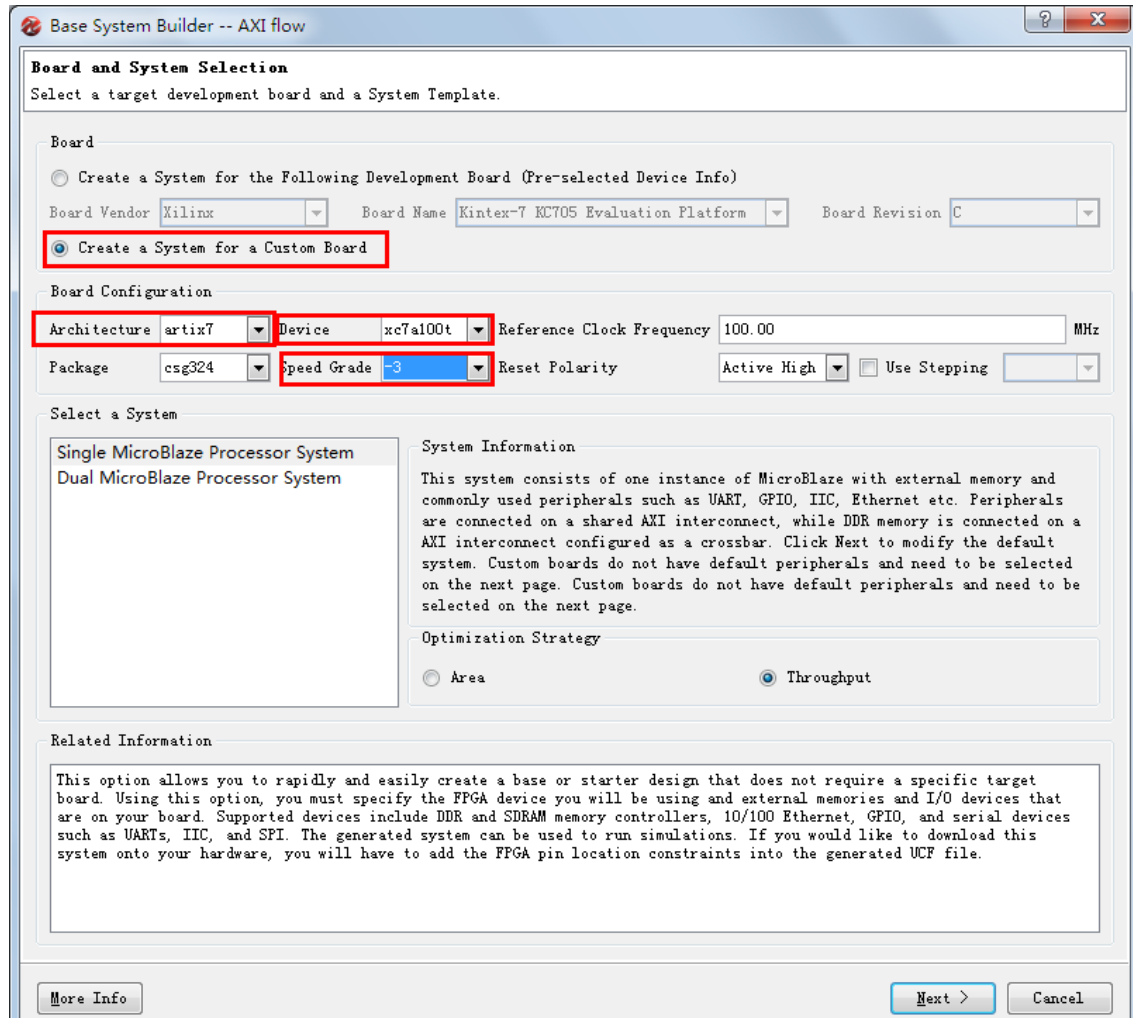


图 4：新工程参数设置

1-1-5. 在接下来出现的页面中选择要添加的 IP 核，并设置 IP 核的参数：

单击 **Select and configure Peripherals** 下的 **Add Device...**

出现图 5 中的蓝色对话框。

在 **IO Interface Type** 中的下拉菜单中选择 **UART**。

在 **Device** 的下拉菜单中选择 **RS232**。

单击 OK，即可添加 UART 的 IP 核。

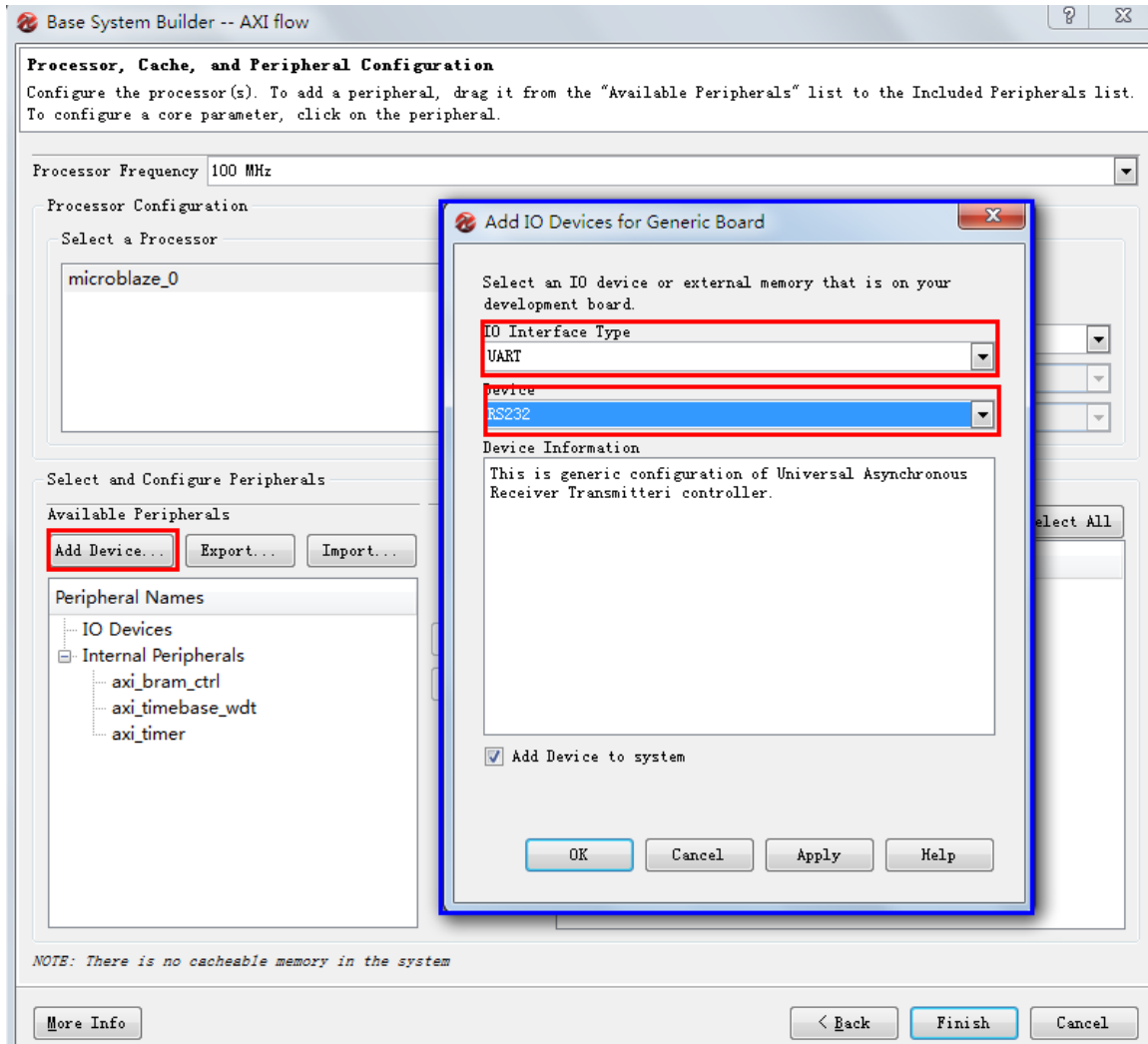


图 5：添加串口的 IP

在 **IO Interface Type** 中的下拉菜单中选择 **IIC**。在 **Device** 的下拉菜单中选择 **Generic\_IIC\_Bus**，因为我们要使用的板载温度计是基于 IIC 协议连接的，如图 6 所示。单击 **OK**，即可添加可以控制板载温度计的 IIC 的 IP 核。

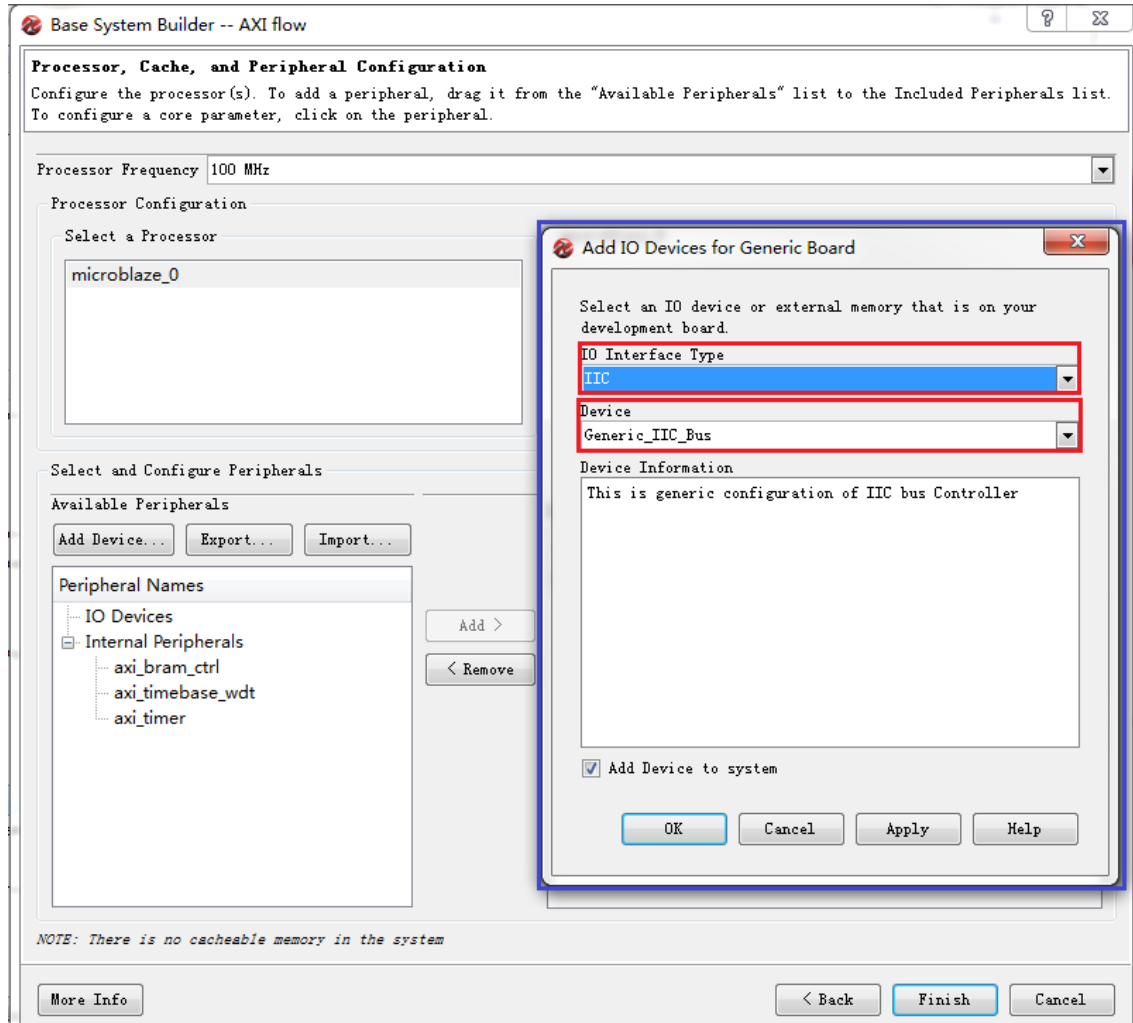


图 6：添加 IIC 的 IP

1-1-6. 注意，串口的默认波特率设置为 9600。在 Lab8 中，我们需要统一修改到 115200，以便提高数据传输速度，SDK 工程中的 Terminal 的波特率以及串口的其他设置必须与之保持一致。

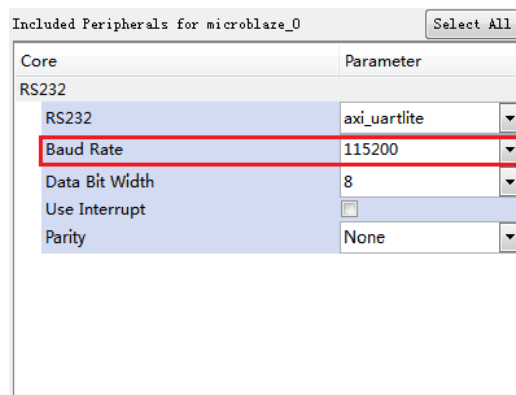


图 7：串口的设置

## 第二步 进行端口的互连

### 2-1. 调整 GPIO 设置

#### 2-1-1. Port 选项卡（展开 External Port），如图 8。

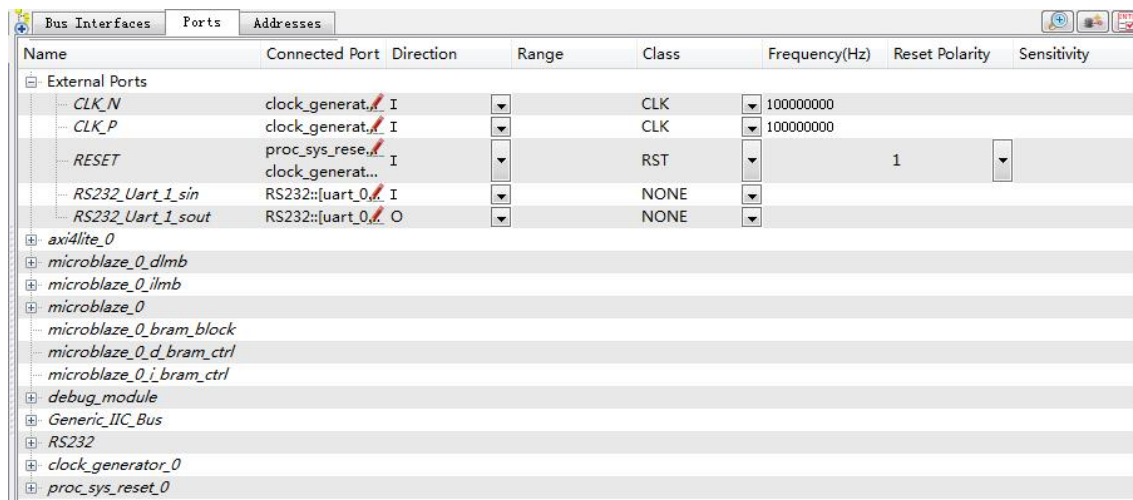


图 8：PORT 选项卡的初始状况

#### 2-1-2. 点击打开 Generic\_IIC\_Bus 选项。

找到 (IO\_IF) iic\_0 的 Sda（数据线）与 Scl（时钟线），分别对它们进行右键选中操作，并在菜单中点击 **Make external**。

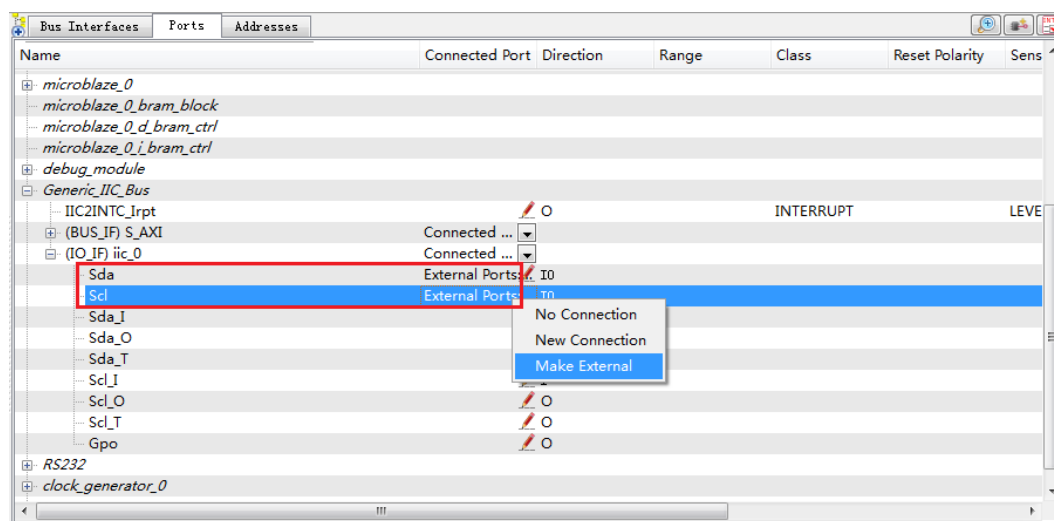


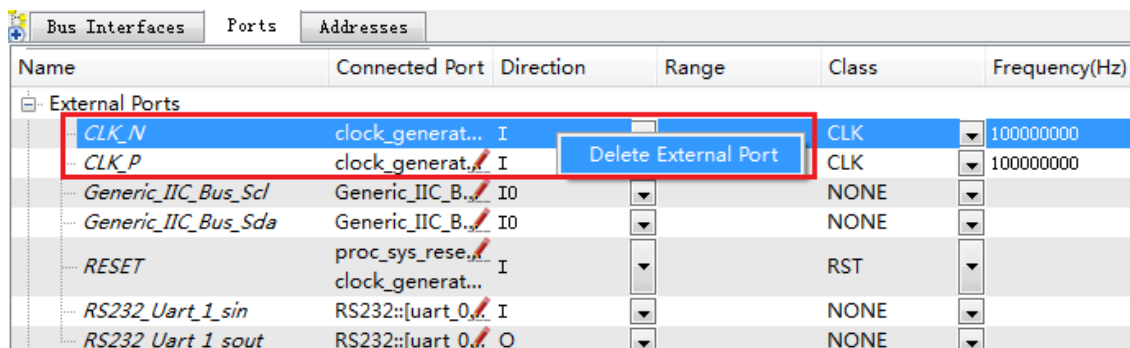
图 9：调整 GPIO 的设置



## 2-2. 在 PORT 选项卡中修改时钟的相关设置

### 2-2-1. 将 External Port 中的 CLK\_N 和 CLK\_P 都去掉。

右键选中该端口，然后点击 **Delete External Port**，如图 10。

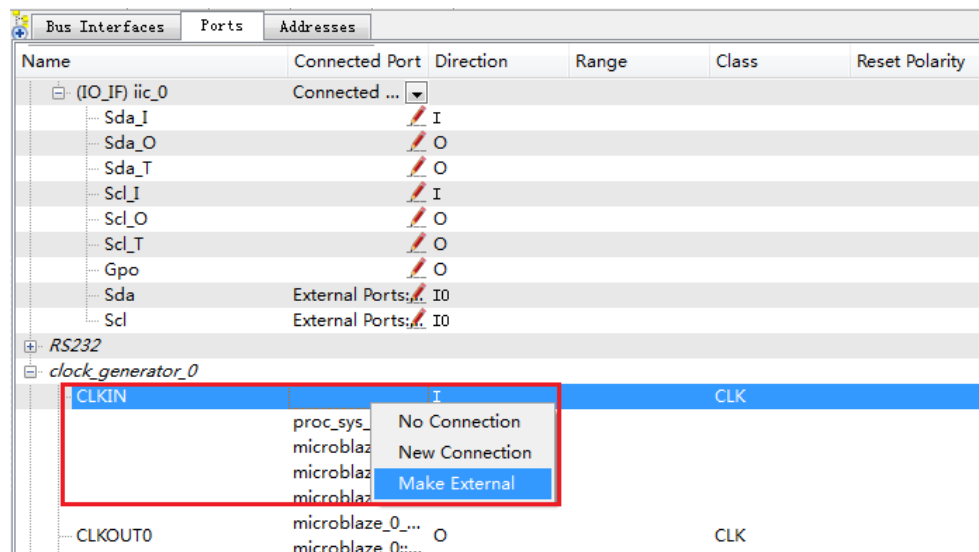


Name	Connected Port	Direction	Range	Class	Frequency(Hz)
<b>External Ports</b>					
CLK_N	clock_generat...	I		CLK	100000000
CLK_P	clock_generat...	I		CLK	100000000
Generic_IIC_Bus_Scl	Generic_IIC_B...	I/O		NONE	
Generic_IIC_Bus_Sda	Generic_IIC_B...	I/O		NONE	
RESET	proc_sys_rese...	I		RST	
RS232_Uart_1_sin	RS232::[uart_0...	I		NONE	
RS232_Uart_1_sout	RS232::[uart_0...	O		NONE	

图 10: 删除多余时钟

### 2-2-2. 将 Clock\_generator\_0 作为新的时钟，加入外部端口。

找到 **Clock\_generator\_0** 中的 **CLKIN**，右键选中，在菜单中点击 **Make external**。



Name	Connected Port	Direction	Range	Class	Reset Polarity
<b>(IO_IF) iic_0</b>					
Sda_I		I			
Sda_O		O			
Sda_T		O			
Scl_I		I			
Scl_O		O			
Scl_T		O			
Gpo		O			
Sda	External Ports::	I/O			
Scl	External Ports::	I/O			
<b>RS232</b>					
<b>clock_generator_0</b>					
CLKIN		I		CLK	
CLKOUT0	microblaze_0...	O		CLK	

图 11: Clock\_generator\_0 中的 CLKIN

注意 External 中的 Name 一项，这是我们添加用户约束文件（UCF）的依据。

Bus Interfaces		Ports	Addresses			
Name	Connected Port	Direction	Range	Class	Reset Polarity	
External Ports						
Generic_IIC_Bus_Scl	Generic_IIC_B...	I/O		NONE		
Generic_IIC_Bus_Sda	Generic_IIC_B...	I/O		NONE		
RESET	proc_sys_rese...	I		RST	1	
RS232_Uart_1_sin	RS232::[uart_0...	I		NONE		
RS232_Uart_1_sout	RS232::[uart_0...	O		NONE		
clock_generator_0_CLKIN_pin	clock_generat...	I		CLK		

图 12: 修改后的 External PORT

## 2-3. 出错处理

2-3-1. 出错处理: 为了防止后面运行程序的时候出现数据溢出的错误, 我们要在 Address 窗口将各个寄存器的 Size 一项均设置成 64K, 如下图所示:

Bus Interfaces		Ports	Addresses				
Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Bus Name	Lock
microblaze_0's Address Map							
microblaze_0_d_bram_ctrl	C_BASEADDR	0x00000000	0x0000FFFF	64K	SLMB	microblaze_0_...	
microblaze_0_i_bram_ctrl	C_BASEADDR	0x00000000	0x0000FFFF	64K	SLMB	microblaze_0_...	
RS232	C_BASEADDR	0x40600000	0x4060FFFF	64K	S_AXI	axi4lite_0	
Generic_IIC_Bus	C_BASEADDR	0x40800000	0x4080FFFF	64K	S_AXI	axi4lite_0	
debug module	C_BASEADDR	0x41400000	0x4140FFFF	64K	S_AXI	axi4lite_0	

图 13: 设置完成后的寄存器大小

## 第三步 添加用户约束文件

### 3-1. 打开初始 UCF 文件，根据需求进行修改

3-1-1. 在页面偏左找到 IP catalogue / Project 选项卡，双击 UCF File: data\wendu.ucf，ucf 文件在右侧打开。

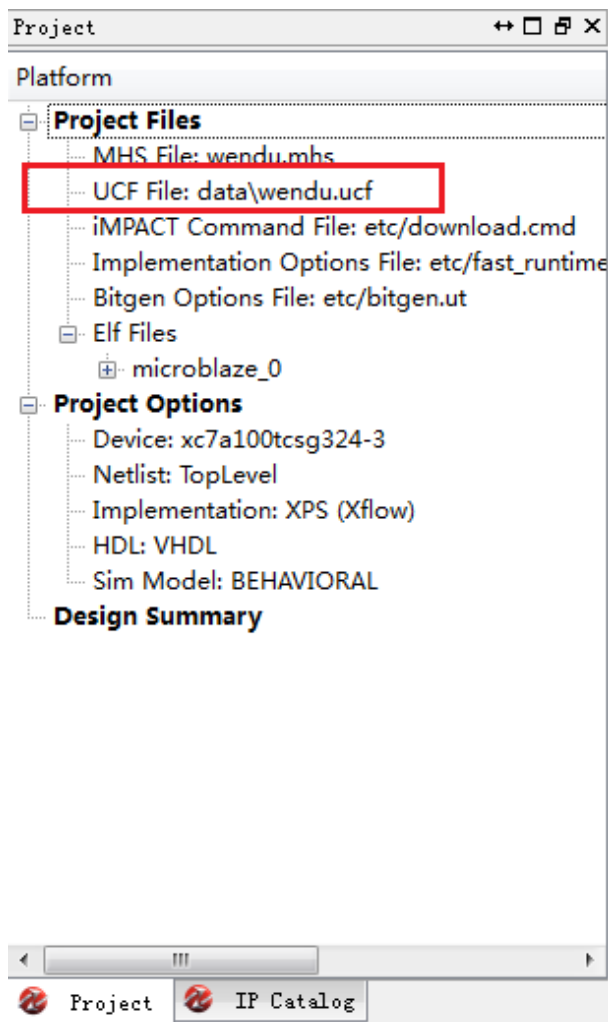
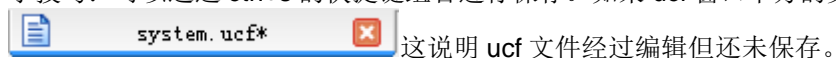


图 14: UCF 文件的位置

3-1-2. 这里我们手动输入 LOC（引脚位置）约束代码，如图 15。点击保存。

小技巧：可以通过 **ctrl+s** 的快捷键组合进行保存。如果 ucf 窗口下方的文件名旁边有\*，如图所示：



这说明 ucf 文件经过编辑但还未保存。

```

5
6 # Clock signal
7 NET "clock_generator_0_CLKIN_pin" LOC = "E3" | IOSTANDARD = "LVCMOS33";
8 NET "clock_generator_0_CLKIN_pin" TNM_NET = sys_clk_pin;
9 TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 100 MHz HIGH 50%;
10
11 # Switches
12 NET "RESET" LOC = "U9" | IOSTANDARD = "LVCMOS33";|
13
14 # Temperature Sensor
15 NET "Generic_IIC_Bus_Scl" LOC = "F16" | IOSTANDARD = "LVCMOS33";
16 NET "Generic_IIC_Bus_Sda" LOC = "G16" | IOSTANDARD = "LVCMOS33";
17
18 # USB-RS232 Interface
19 NET "RS232_Uart_1_sin" LOC = "C4" | IOSTANDARD = "LVCMOS33";
20 NET "RS232_Uart_1_sout" LOC = "D4" | IOSTANDARD = "LVCMOS33";
21

```

图 15: UCF 文件

### 3-1-3. 保存之后将工程导入到 SDK

在页面左边，点击 **Export Design**。

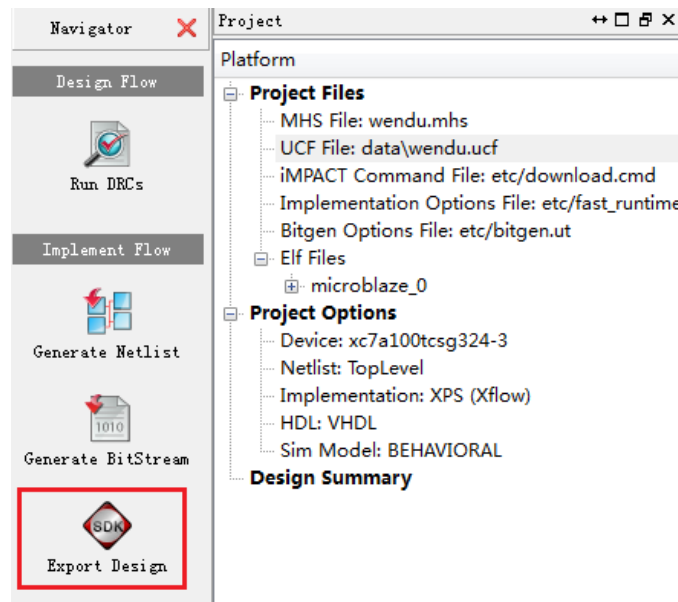


图 16: export design

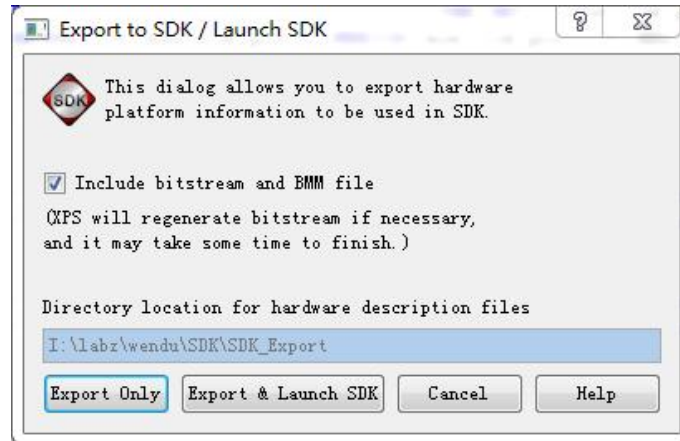


图 17：在弹出的对话框中选择 **Export & launch sdk**

### 3-1-4. 选择 SDK 导入路径

注意要具体到..\sdk\sdk\_export。

点击 ok。

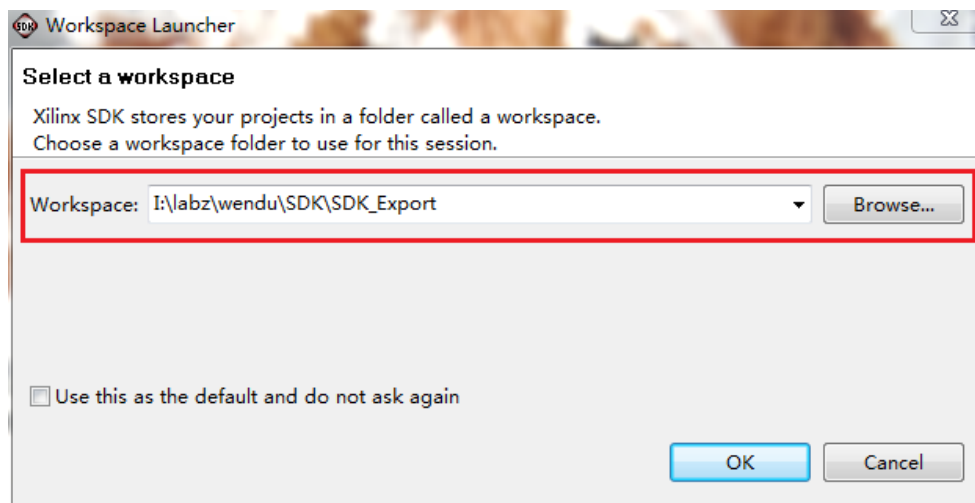


图 18：选择 SDK 导入路径

## 第四步 添加 app

4-1. 编译第一步：综合，生成网表文件。

4-1-1. 在 SDK 的用户界面中，选择 file—new—application project。

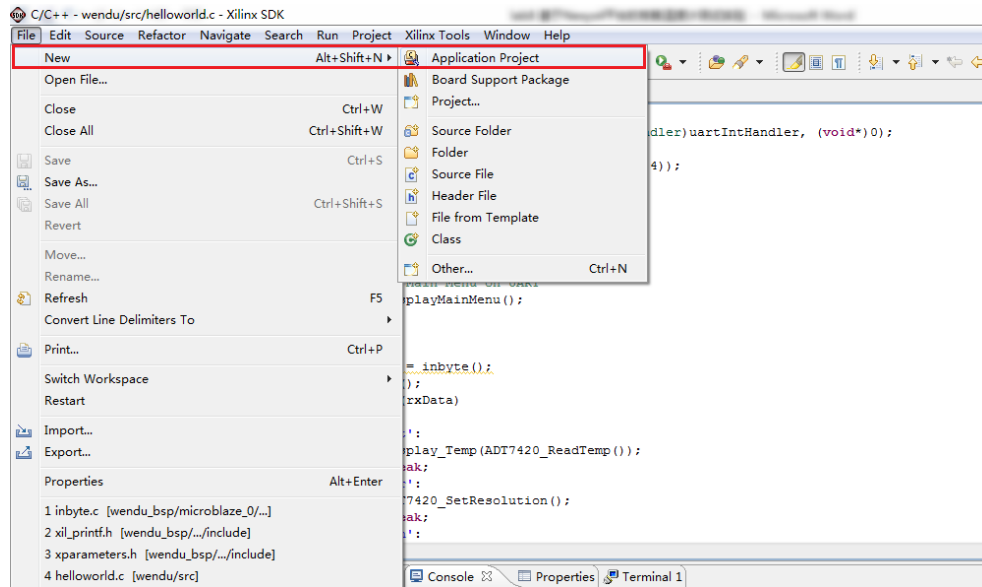


图 19：建立新的工程

4-1-2. 输入工程的名称，这里使用 **wendu**, 同样不要包含空格和中文，点击 **next**

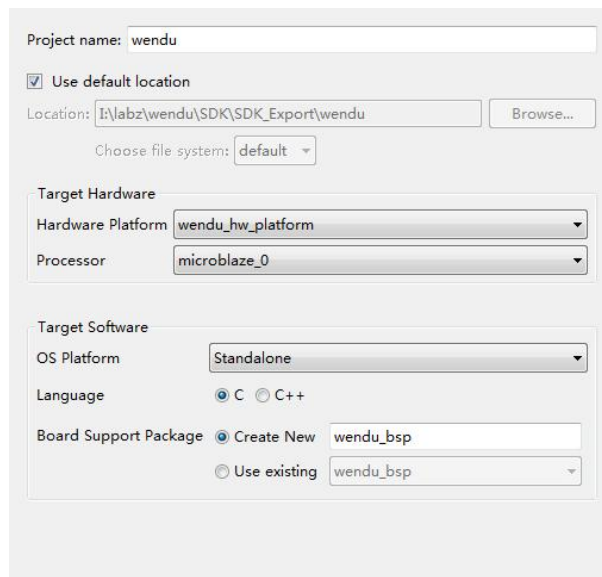


图 20：配置工程

4-1-3. 在下一步弹出的对话框中选择 **hello world**，然后点击 **finish**。

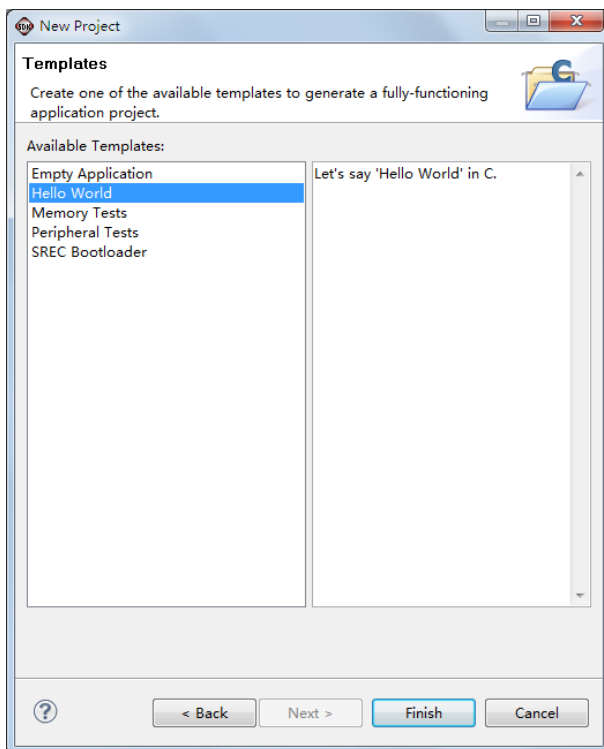


图 21：选择软件工程的模板

4-1-4. 添加完毕以后我们先不着急写代码，需要在路径 `labz\wendu\SDK\SDK_Export\wendu\src` 添加如下文件（文件内容请见附件）：

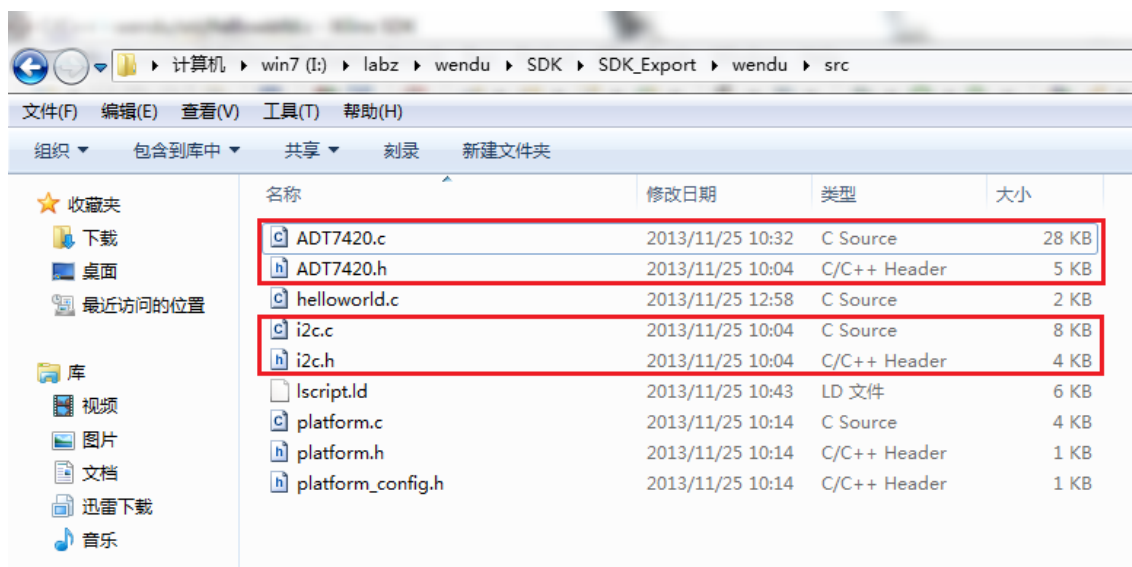


图 22：需要添加的库文件

#### 4-1-5. 接下来我们添加板载温度计测试代码:

```
/****** Include Files *****/
/******
#include <stdio.h>
#include "xparameters.h"
#include "xil_cache.h"
#include "xil_io.h"
#include "i2c.h"
#include "ADT7420.h"
#include "mb_interface.h"
#include "xil_printf.h"

void uartIntHandler(void);
char rxData = '0';

int main()
{
    Xil_ICacheEnable();
    Xil_DCacheEnable();

    // Set Interrupt Handler
    microblaze_register_handler((XInterruptHandler)uartIntHandler, (void*)0);
    // Enable UART Interrupts
    Xil_Out32(XPAR_RS232_BASEADDR+0x0C, (1 << 4));
    // Enable Microblaze Interrupts
    microblaze_enable_interrupts();
    // Initialize ADT7420 Device
    ADT7420_Init();

    // Display Main Menu on UART
    ADT7420_DisplayMainMenu();

    do{
        rxData = inbyte();
        inbyte();
        switch(rxData)
        {
            case 't':
                Display_Temp(ADT7420_ReadTemp());
                break;
            case 'r':
                ADT7420_SetResolution();
                break;
            case 'h':
                ADT7420_DisplaySetTHighMenu();
                break;
            case 'l':
                ADT7420_DisplaySetTLowMenu();
                break;
            case 'c':
                ADT7420_DisplaySetTCritMenu();
                break;
        }
    }
```



```
        case 'y':
            ADT7420_DisplaySetTHystMenu();
            break;
        case 'f':
            ADT7420_DisplaySetFaultQueueMenu();
            break;
        case 's':
            ADT7420_DisplaySettings();
            break;
        case 'm':
            ADT7420_DisplayMenu();
            break;
        case '0':
            break;
        default:
            xil_printf("\n\rWrong option! Please select one of the options below.\n\r");
            ADT7420_DisplayMenu();
            break;
    }
}while(rxData != 'q');

xil_printf("Exiting application\n\r");

exit(0);

Xil_DCacheDisable();
Xil_ICacheDisable();

return 0;
}

void uartIntHandler(void)
{
    if(Xil_In32(XPAR_RS232_BASEADDR + 0x08) & 0x01)
    {
        rxData = Xil_In32(XPAR_UARTLITE_0_BASEADDR);
    }
}
```

图 23: 板载温度计完整测试代码

代码说明:

- 1、通过 **ADT7420\_Init()**等函数对传感器等部件进行了初始化;
- 2、程序通过调用 **ADT7420\_DisplayMainMenu()**函数来在串口显示选择菜单;
- 3、程序将用户输入串口的字符存入变量 **rxData** 中, 通过读取该变量的值来调用 **AD7420\_ReadTemp()**等关于板载温度传感器 **ADT7420** 的不同函数, 这些函数被调用后会自动通过 **xil\_printf()**函数向串口打印实时的显示数据。

## 第五步 上板验证

### 5-1. 将 Nexys4 与 PC 的 USB 接口连接

### 5-2. 查看端口号：

右键“我的电脑”-“属性”，在页面左侧选择“设备管理器”。

发现与 com4 端口相连：（不同电脑可能有所区别，同一电脑每次连接也有可能有所区别）。

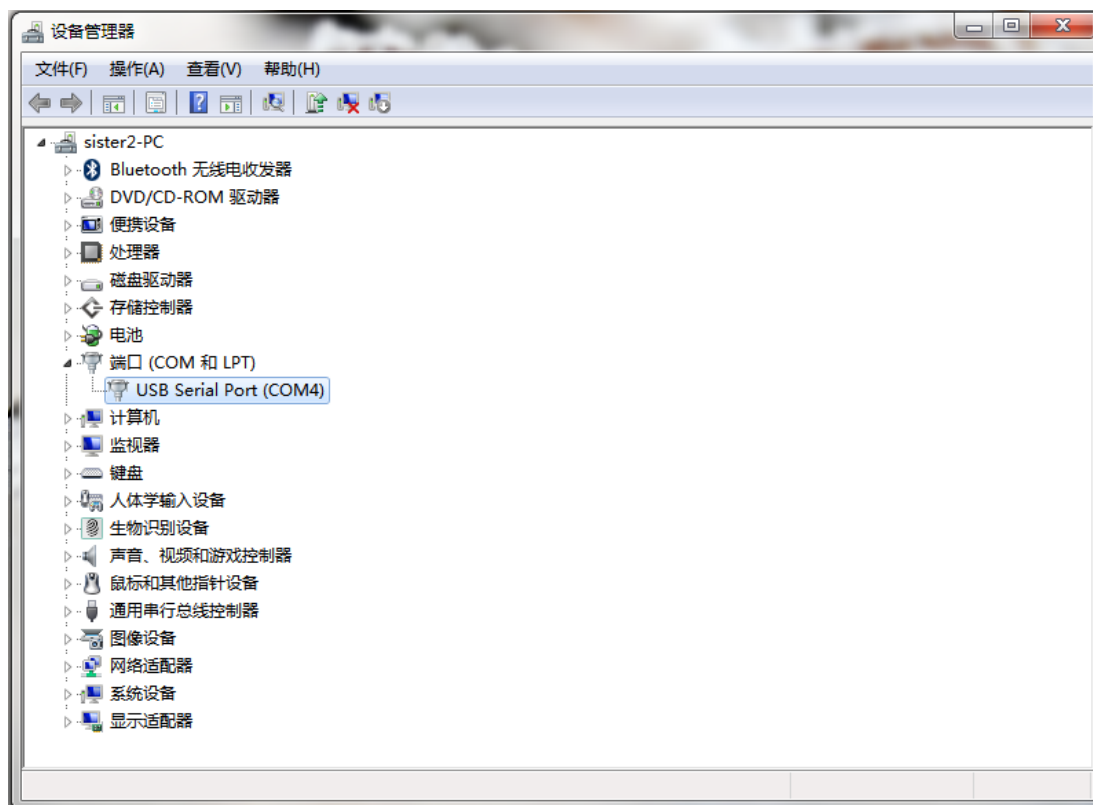


图 24：查看端口号

### 5-3. 在 SDK 中打开串口：

5-3-1. 在下面的在页面下方找到 **terminal** 选项卡，然后点击绿色的连接按钮。

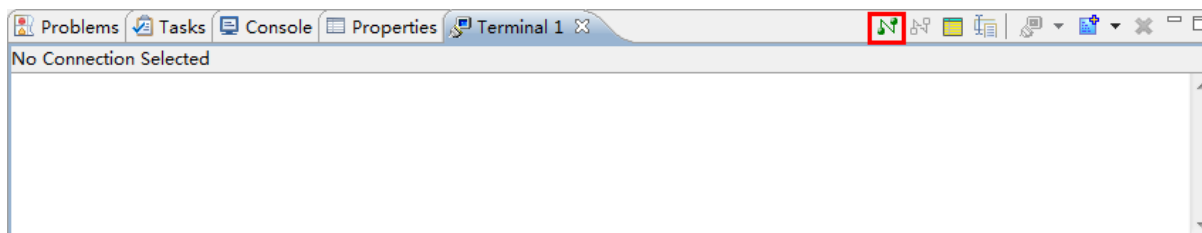


图 25：连接端口

5-3-2. 按照端口号和 XPS 中的波特率（**baud rate**）进行如下设置：

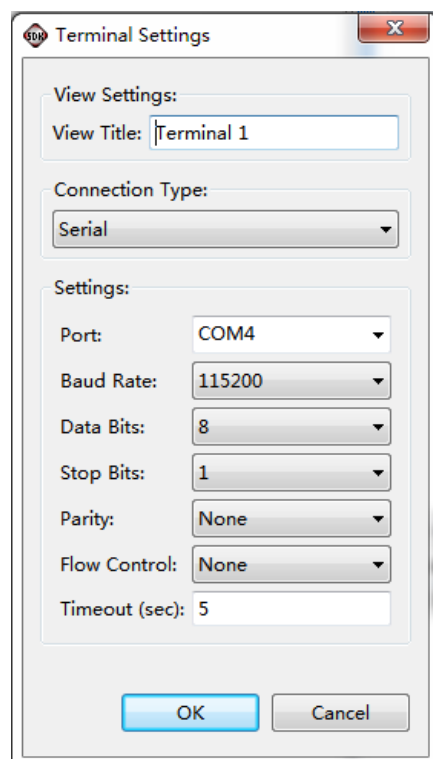


图 26：设置端口

如果报了“no such port”的错误，可以通过新建串口，更改串口号：

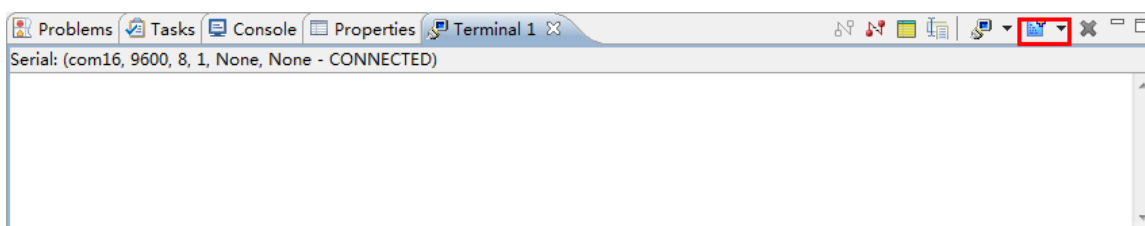


图 27：串口配置的修改方法

## 5-4. 将程序下载到板子上并运行

5-4-1.在页面上方，**xilinx tools** 下拉菜单中选择 **program fpga**。

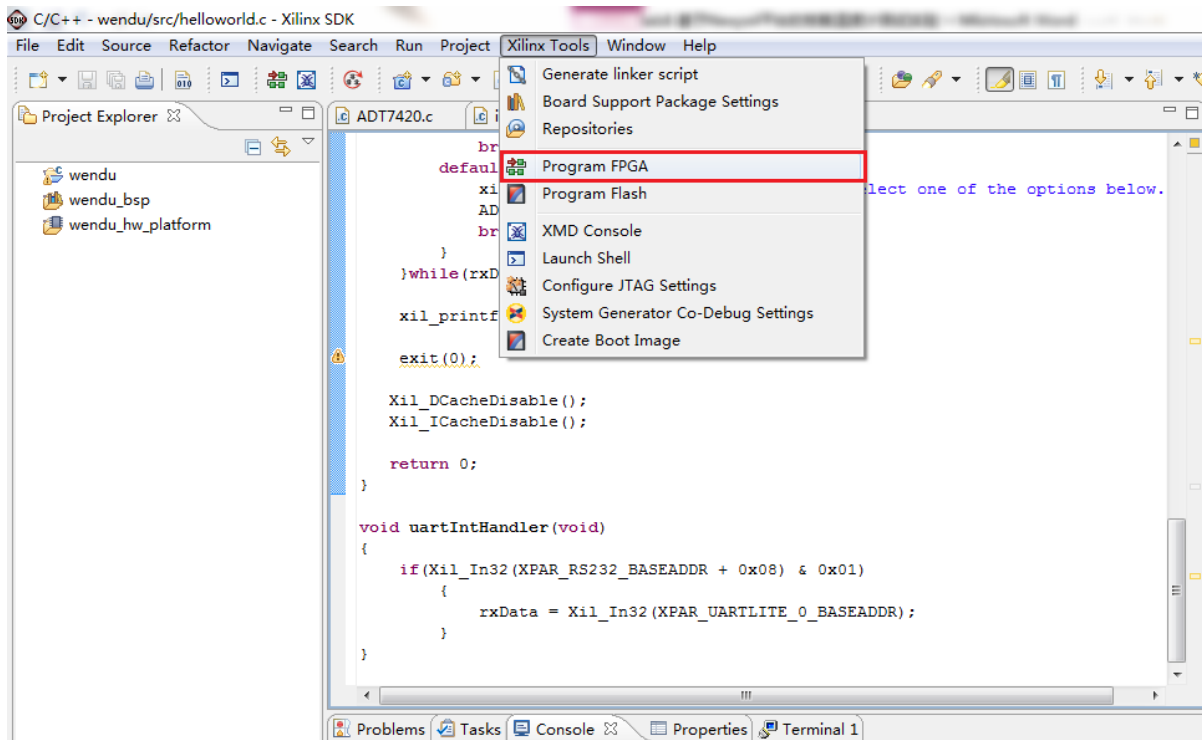


图 28: 将 C 语言程序下载到开发板中

5-4-2. 注意要选择正确的 elf 文件，并点击 **program** 运行程序：

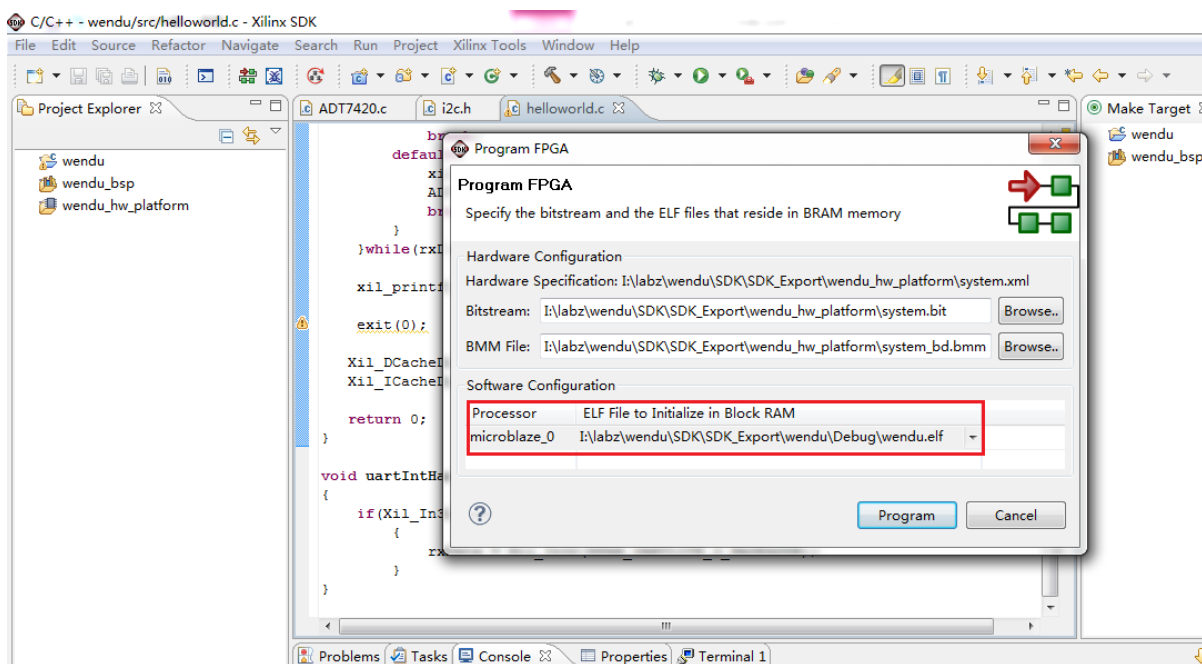
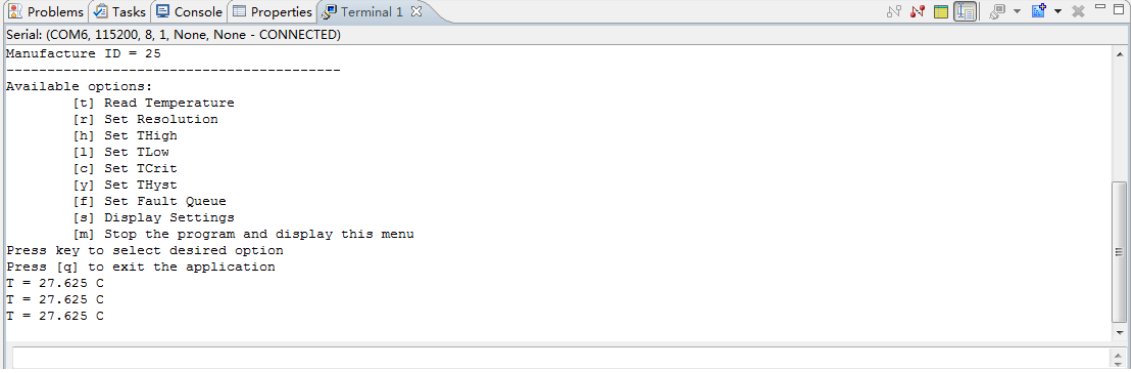


图 29: 选择 elf 文件

**5-5. 在串口中看到结果：（每次程序运行后串口都会提示输入数据）。**



```
Serial: (COM6, 115200, 8, 1, None, None - CONNECTED)
Manufacture ID = 25
-----
Available options:
    [t] Read Temperature
    [r] Set Resolution
    [h] Set THigh
    [l] Set TLow
    [c] Set TCrit
    [y] Set THyst
    [f] Set Fault Queue
    [s] Display Settings
    [m] Stop the program and display this menu
Press key to select desired option
Press [q] to exit the application
T = 27.625 C
T = 27.625 C
T = 27.625 C
```

**图 30：串口显示效果**