25A, No.1369,Dongfang Road, Shanghai
Tel:+86-021-58893151 Fax: +86-021-58894598

Lab2: GPIO--Switch&Button

——添加 GPIO 外设

*基于 **Nexys 4 FPGA** 平台*

Lab 2: GPIO—Switch&Button

实验简介

本实验旨在指导读者添加 GPIO IP 核，并完成在串口显示当前开关和按钮状态的简单例程。

实验目标

在完成本实验后，您将学会：

- 从 IP Catalog 中添加 IP 核
- 串口显示拨码开关及按钮的数字电路的实现

实验过程

本实验在 lab1 的基础上，指导读者使用 Xilinx 的 XPS 工具，从 IP Catalog 添加 GPIO IP 核，并导入到 SDK，调用这个 IP 核，在串口上显示出开关和按钮的状态，然后在 Nexys 4 平台上进行测试验证。

实验由以下步骤组成：

1．打开 lab1 的 XPS 工程

2．添加 IP 核并调整相关设置

3．添加用户约束文件

4．将工程导入到 SDK

5．在 SDK 中修改 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）

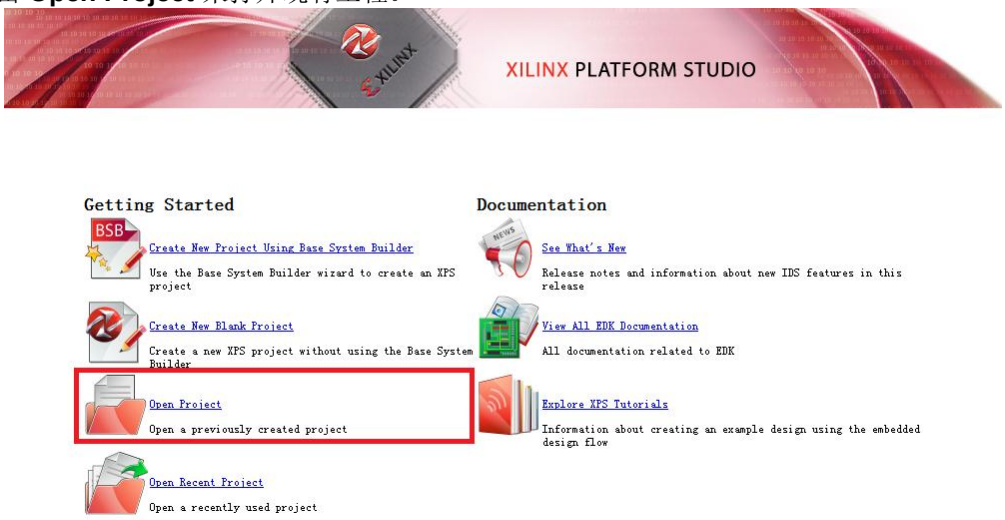
第一步 添加 GPIO IP 核

1-1. 运行 Xilinx Platform Studio,打开 lab1 工程.

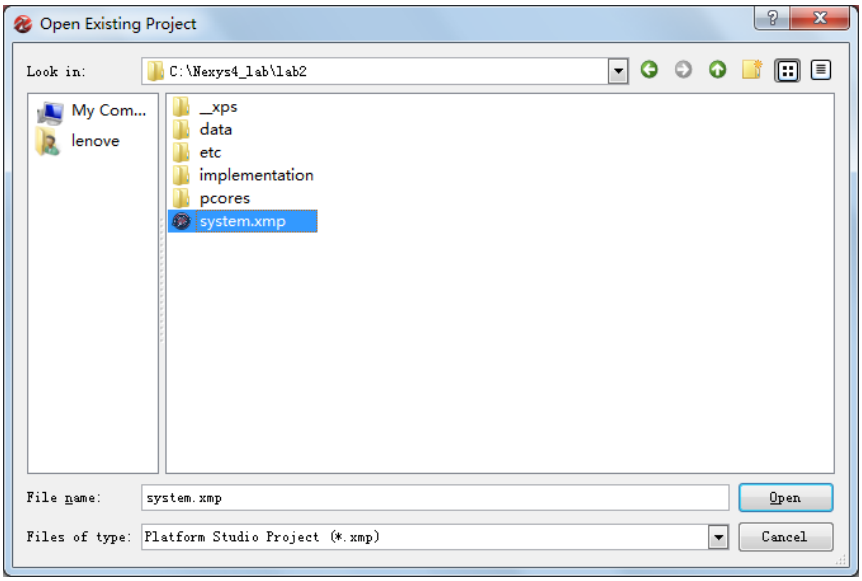
1-1-1. 打开工作目录（**C:\Nexys4_lab**），复制 **lab1** 文件夹，并重命名为 **lab2**

1-1-2. 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。）。

1-1-3. 点击 **Open Project** 来打开现有工程。

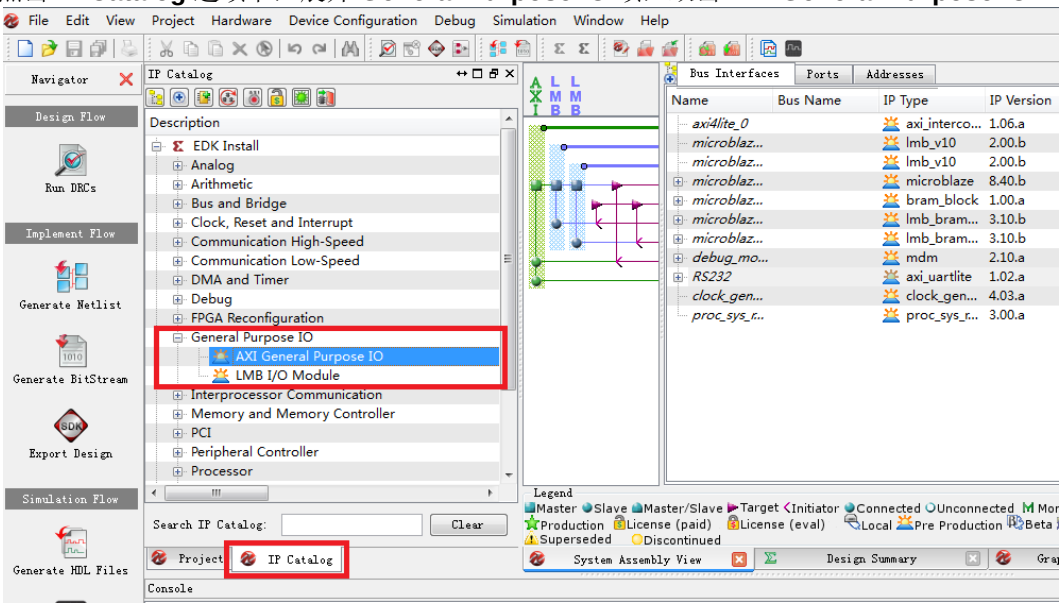


1-1-4. 在弹出 ***Open Existing Project*** 的对话框中选择 lab2 工程（**C:\Nexys4_lab\lab2\system.xmp**）。点击 **Open**。

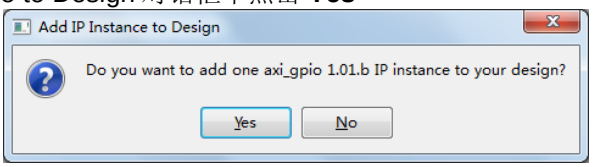


1-2. 添加 GPIO IP 核

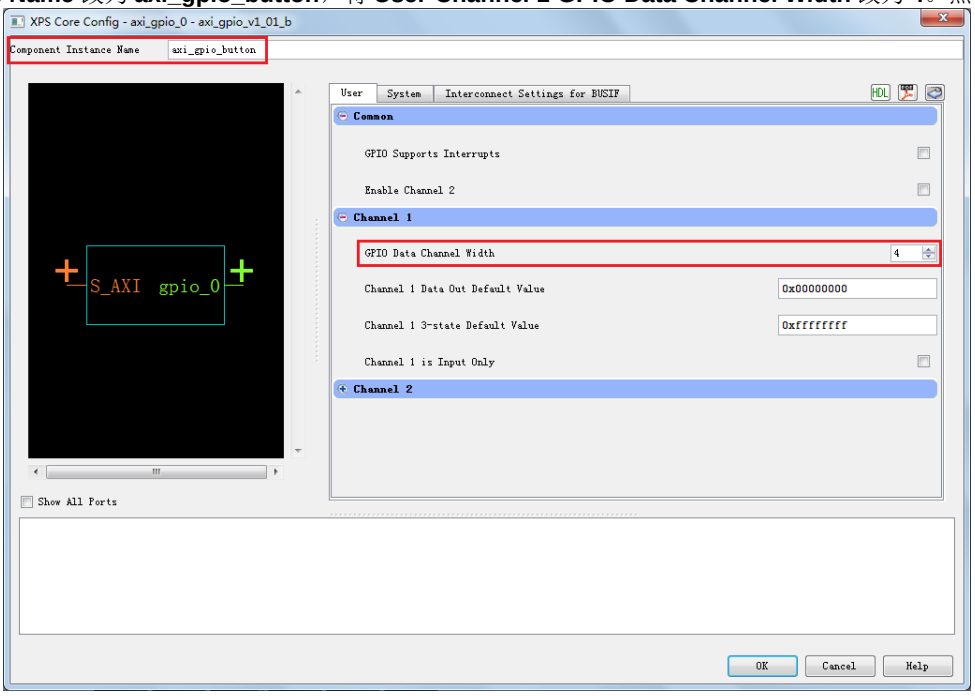
1-2-1. 点击 **IP Catalog** 选项卡，展开 **General Purpose IO** 项，双击 **AXI General Purpose IO**



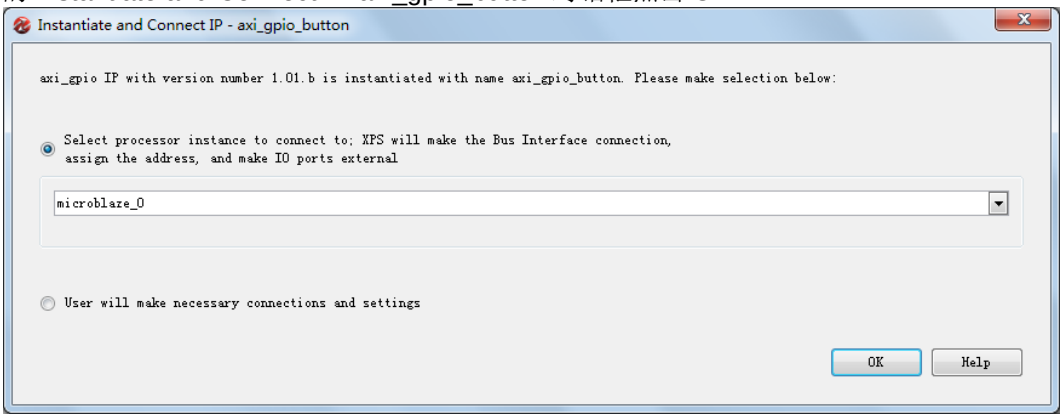
在弹出的 Add IP Instance to Design 对话框中点击 **Yes**



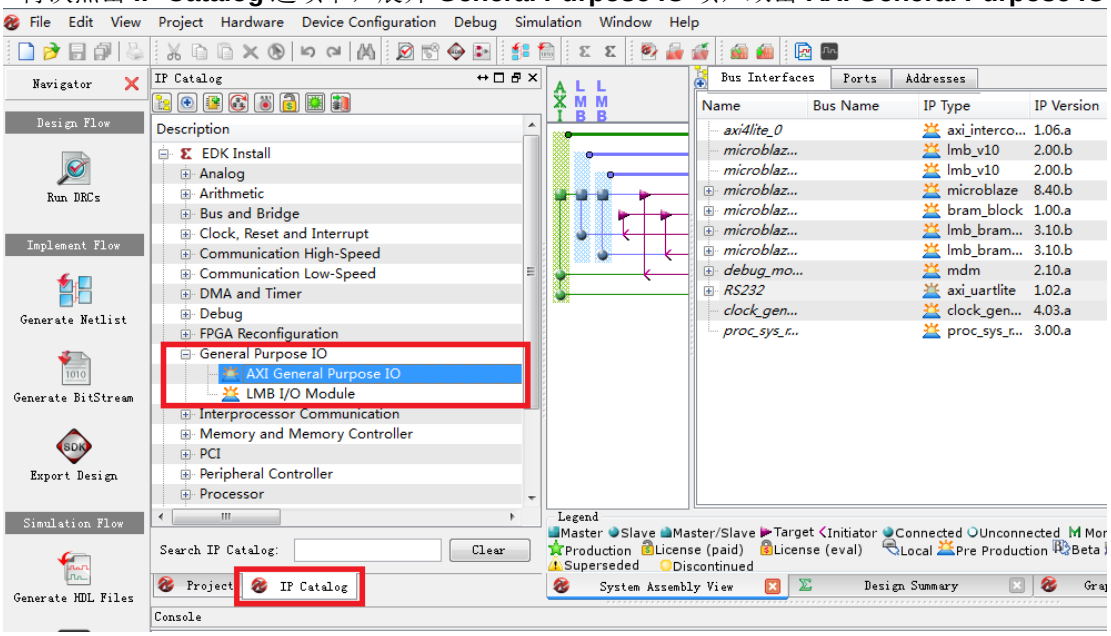
1-2-2. 在弹出的 XPS Core Config – axi_gpio_0 - axi_gpio_v1_01_b 页面中进行参数配置：将 **Component Instance Name** 改为 **axi_gpio_button**，将 **User-Channel 1-GPIO Data Channel Width** 改为 **4**。点击 **OK**。



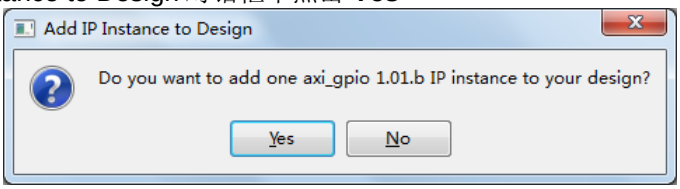
1-2-3. 弹出的 Instantiate and Connect IP-axi_gpio_button 对话框点击 **OK**



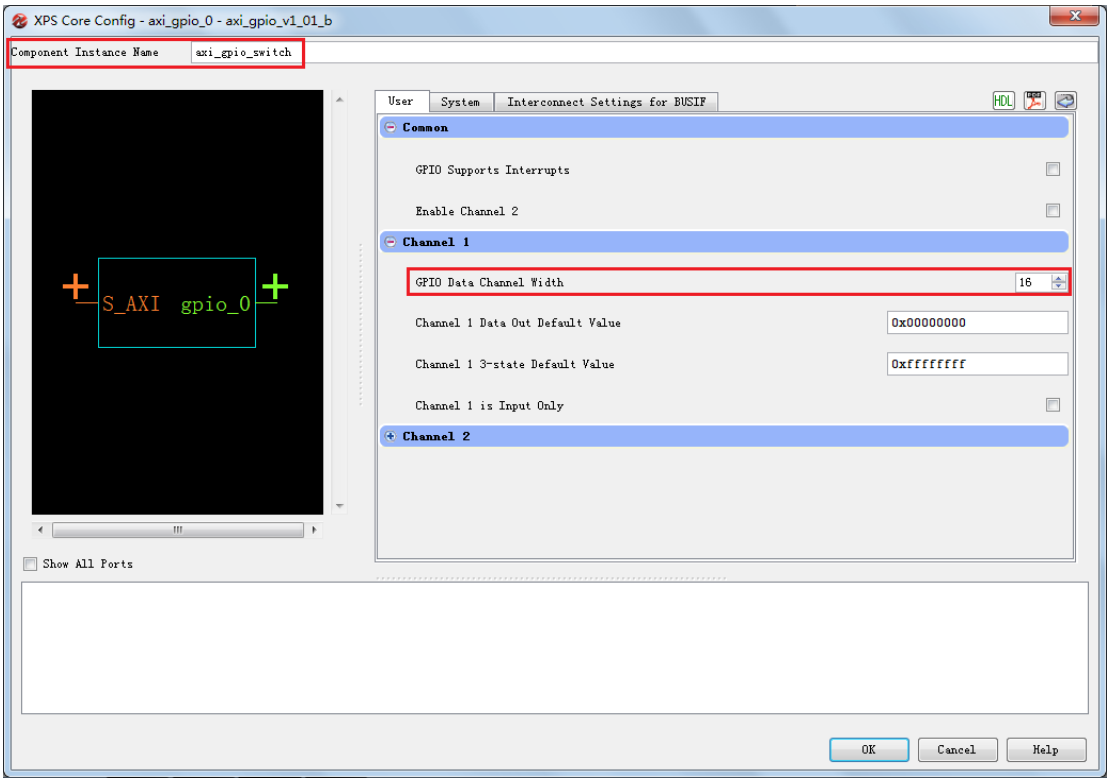
1-2-4. 再次点击 **IP Catalog** 选项卡，展开 **General Purpose IO** 项，双击 **AXI General Purpose IO**



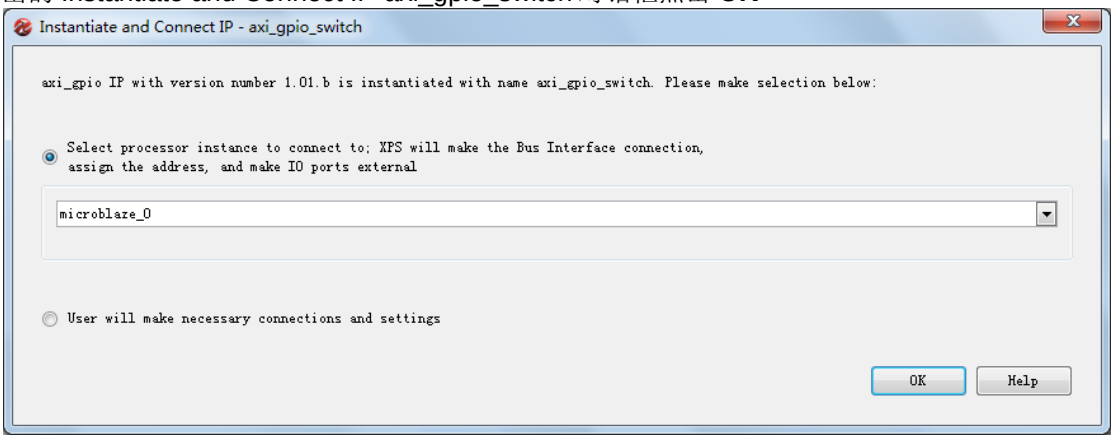
在弹出的 Add IP Instance to Design 对话框中点击 **Yes**



1-2-5. 在弹出的 XPS Core Config – axi_gpio_0 - axi_gpio_v1_01_b 页面中进行参数配置：将 **Component Instance Name** 改为 **axi_gpio_switch**，将 **User-Channel 1-GPIO Data Channel Width** 改为 **16**。点击 **OK**。



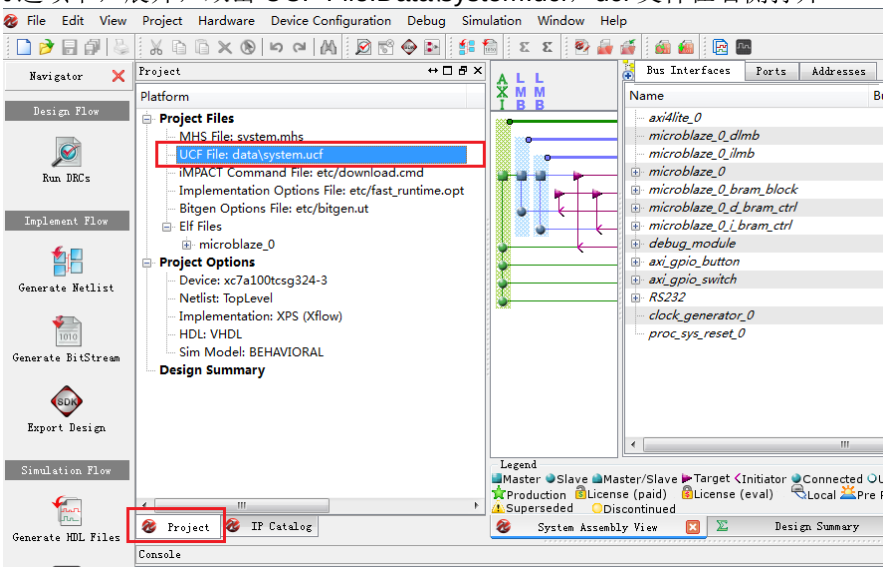
1-2-6. 弹出的 Instantiate and Connect IP-axi_gpio_switch 对话框点击 **OK**



第二步 导出工程到 SDK

2-1. 添加用户约束文件

2-1-1. 点击 **Project** 选项卡，展开，双击 UCF File:Data\system.ucf，ucf 文件在右侧打开

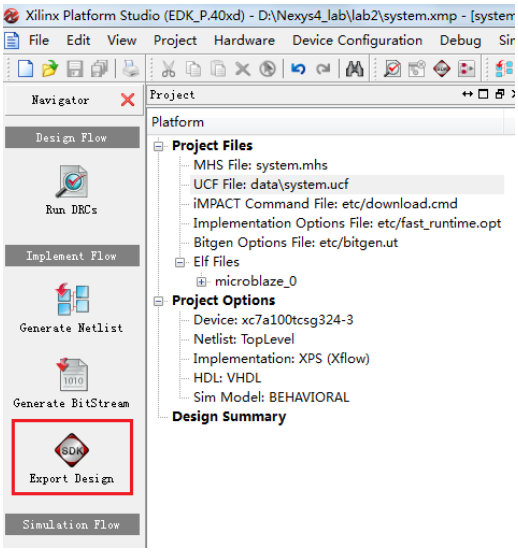


2-1-2. 编写 ucf 文件如下，点击**保存**。

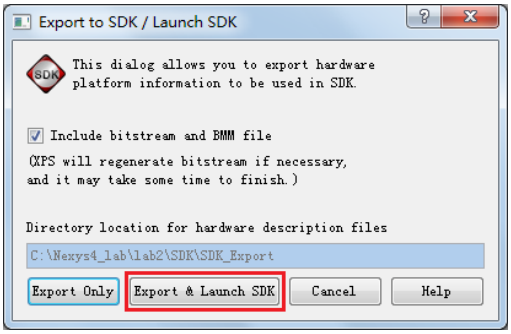
```
# Clock signal
NET "clock_generator_0_CLKIN_pin"    LOC = "E3"   | IOSTANDARD = "LVCMOS33";
NET "clock_generator_0_CLKIN_pin" TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 100 MHz HIGH 50%;

# USB-RS232 Interface
NET "RS232_Uart_1_sin"                LOC = "C4"   | IOSTANDARD = "LVCMOS33";
NET "RS232_Uart_1_sout"               LOC = "D4"   | IOSTANDARD = "LVCMOS33";

# Switches
NET "axi_gpio_switch_GPIO_IO_pin<0>"    LOC = "U9"   | IOSTANDARD = "LVCMOS33";
NET "axi_gpio_switch_GPIO_IO_pin<1>"    LOC = "U8"   | IOSTANDARD = "LVCMOS33";
NET "axi_gpio_switch_GPIO_IO_pin<2>"    LOC = "R7"   | IOSTANDARD = "LVCMOS33";
NET "axi_gpio_switch_GPIO_IO_pin<3>"    LOC = "R6"   | IOSTANDARD = "LVCMOS33";
NET "axi_gpio_switch_GPIO_IO_pin<4>"    LOC = "R5"   | IOSTANDARD = "LVCMOS33";
NET "axi_gpio_switch_GPIO_IO_pin<5>"    LOC = "V7"   | IOSTANDARD = "LVCMOS33";
NET "axi_gpio_switch_GPIO_IO_pin<6>"    LOC = "V6"   | IOSTANDARD = "LVCMOS33";
NET "axi_gpio_switch_GPIO_IO_pin<7>"    LOC = "V5"   | IOSTANDARD = "LVCMOS33";
NET "axi_gpio_switch_GPIO_IO_pin<8>"    LOC = "U4"   | IOSTANDARD = "LVCMOS33";
NET "axi_gpio_switch_GPIO_IO_pin<9>"    LOC = "V2"   | IOSTANDARD = "LVCMOS33";
NET "axi_gpio_switch_GPIO_IO_pin<10>"   LOC = "U2"   | IOSTANDARD = "LVCMOS33";
NET "axi_gpio_switch_GPIO_IO_pin<11>"   LOC = "T3"   | IOSTANDARD = "LVCMOS33";
NET "axi_gpio_switch_GPIO_IO_pin<12>"   LOC = "T1"   | IOSTANDARD = "LVCMOS33";
NET "axi_gpio_switch_GPIO_IO_pin<13>"   LOC = "R3"   | IOSTANDARD = "LVCMOS33";
NET "axi_gpio_switch_GPIO_IO_pin<14>"   LOC = "P3"   | IOSTANDARD = "LVCMOS33";
NET "axi_gpio_switch_GPIO_IO_pin<15>"   LOC = "P4"   | IOSTANDARD = "LVCMOS33";

# Buttons
NET "RESET"                           LOC = "E16"  | IOSTANDARD = "LVCMOS33";
NET "axi_gpio_button_GPIO_IO_pin<0>"    LOC = "F15"  | IOSTANDARD = "LVCMOS33";
NET "axi_gpio_button_GPIO_IO_pin<1>"    LOC = "T16"  | IOSTANDARD = "LVCMOS33";
NET "axi_gpio_button_GPIO_IO_pin<2>"    LOC = "R10"  | IOSTANDARD = "LVCMOS33";
NET "axi_gpio_button_GPIO_IO_pin<3>"    LOC = "V10"  | IOSTANDARD = "LVCMOS33";
```

2-2. 将工程导出到 SDK

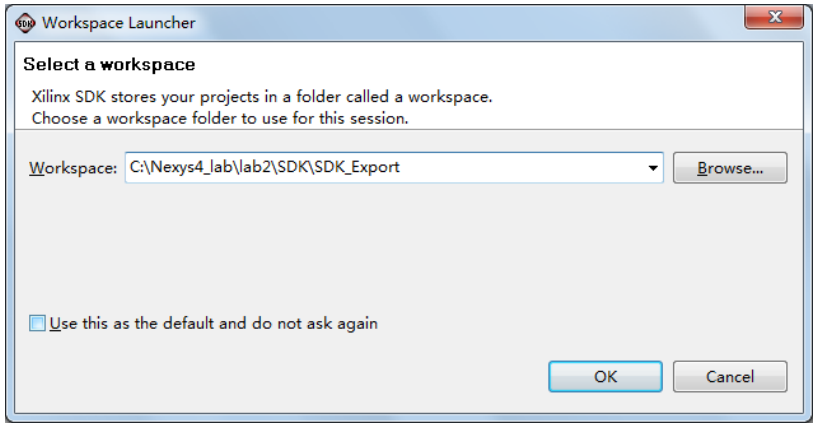
2-2-1. 点击页面左侧的 **Export Design** 按钮。



弹出的对话框点击 **Export & Launch SDK**。



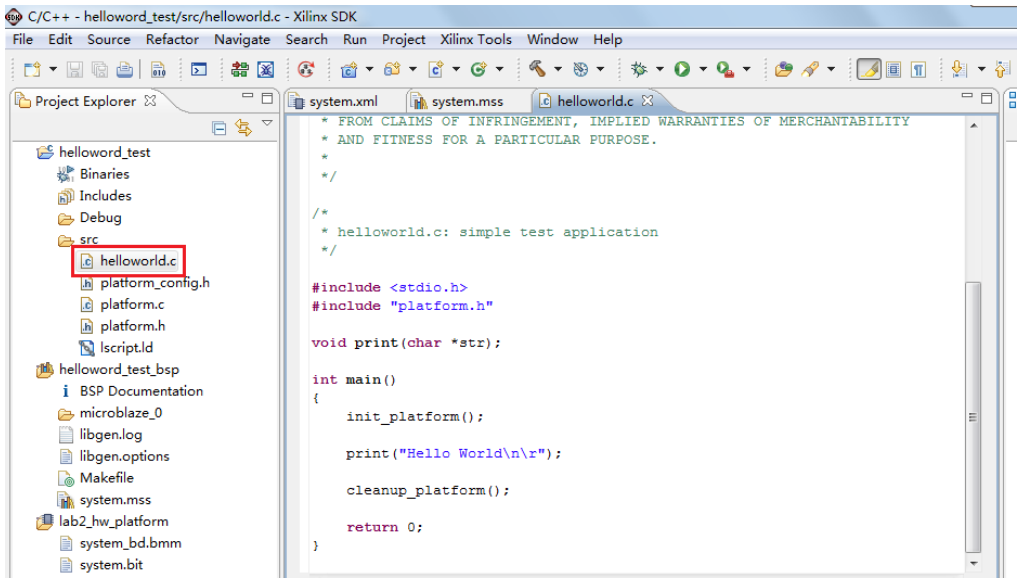
2-2-2. 当弹出如下对话框时，点击 **Browse** 选择导出路径(**注意要具体到..\SDK\SDK_Export**)，点击 **OK**



第三步 修改 C 语言程序

3-1. 打开现有工程，修改 C 语言程序

3-1-1. 打开 helloworld.c 文件（因为 lab2 是复制了 lab1 的工程后，在 lab1 的基础上添加功能的，所以打开 SDK 后并不是空的工程，而显示出了在 lab1 中建立的 helloworld_test 工程）



3-1-2. 修改 helloworld.c 文件如下，保存

```
#include <stdio.h>
#include "platform.h"
#include "xparameters.h"
#include "xgpio.h"
#include "xuartlite_l.h"

void print(char *str);

XGpio Gpio_button;
XGpio Gpio_switch;

int main()
{
    init_platform();
    int Status_button;
    int Status_switch;
    Status_button = XGpio_Initialize(&Gpio_button, XPAR_AXI_GPIO_BUTTON_DEVICE_ID);
    if (Status_button != XST_SUCCESS) {
        return XST_FAILURE;
    }
    Status_switch = XGpio_Initialize(&Gpio_switch, XPAR_AXI_GPIO_SWITCH_DEVICE_ID);
    if (Status_switch != XST_SUCCESS) {
        return XST_FAILURE;
    }

    XGpio_SetDataDirection(&Gpio_button, 1, 0xf);
    XGpio_SetDataDirection(&Gpio_switch, 1, 0xffff);

    int value_button;
    int value_switch;
    int temp_button;
    int temp_switch;

    temp_button = XGpio_DiscreteRead(&Gpio_button, 1);
    temp_switch = XGpio_DiscreteRead(&Gpio_switch, 1);
```

```
    while(1)
    {
        value_button = XGpio_DiscreteRead(&Gpio_button, 1);
        value_switch = XGpio_DiscreteRead(&Gpio_switch, 1);

        if(value_button != temp_button)
        {
            if((value_button & 1) == 1)
                print("BTNU is pushed!\r\n");

            if((value_button & 2) == 2)
                print("BTNL is pushed!\r\n");

            if((value_button & 4) == 4)
                print("BTNR is pushed!\r\n");

            if((value_button & 8) == 8)
                print("BTND is pushed!\r\n");

            temp_button = value_button;
        }

        if(value_switch != temp_switch)
        {
            if((value_switch & 1) == 1)
                print("SW0:ON  ");

            if((value_switch & 2) == 2)
                print("SW1:ON  ");

            if((value_switch & 4) == 4)
                print("SW2:ON  ");

            if((value_switch & 8) == 8)
                print("SW3:ON  ");

            if((value_switch & 16) == 16)
                print("SW4:ON  ");

            if((value_switch & 32) == 32)
                print("SW5:ON  ");

            if((value_switch & 64) == 64)
                print("SW6:ON  ");

            if((value_switch & 128) == 128)
                print("SW7:ON  ");

            if((value_switch & 256) == 256)
                print("SW8:ON  ");

            if((value_switch & 512) == 512)
                print("SW9:ON  ");

            if((value_switch & 1024) == 1024)
                print("SW10:ON  ");

            if((value_switch & 2048) == 2048)
                print("SW11:ON  ");

            if((value_switch & 4096) == 4096)
                print("SW12:ON  ");

            if((value_switch & 8192) == 8192)
                print("SW13:ON  ");

            if((value_switch & 16384) == 16384)
                print("SW14:ON  ");

            if((value_switch & 32768) == 32768)
                print("SW15:ON  ");

            if(value_switch != 0)
                print("\r\n");

            temp_switch = value_switch;
        }
    }
    cleanup_platform();
    return 0;
}
```

程序首先进行了一些变量的声明以及 GPIO 的初始化及方向设置，之后读取了当前按钮和开关的状态。每次循环都会读取当前按钮和开关的状态，当按钮或开关的值发生变化时，进行判断，如果按钮被按下或开关被打开，则 gpio 的相应位的值会变为 1，判断出哪些位的值为 1 后就可以在串口打印出哪个按钮被按下或哪个开关被打开了。

程序中用到了以下和 GPIO 相关的函数：

（1）初始化函数 XStatus XGpio_Initialize (XGpio *InstancePtr, Xuint16 DeviceId);

用来将唯一的设备 ID 和 Xgpio 结构体联系起来指定设备。

InstancePtr 是 Xgpio 结构体指针，存储器的指针参数必须被预先指定；例如，在程序的最开始定义了 XGpio Gpio_button 和 XGpio Gpio_switch。

DeviceId 是由 Xgpio 控制的唯一的设备 ID，可在 xparameter.h 文件中找到。

（2）数据方向设置函数 void XGpio_SetDataDirection (XGpio *InstancePtr, unsigned Channel, Xuint32 DirectionMask);

用来配置 GPIO 的数据传输方向。

InstancePtr 是 Xgpio 结构体指针；

Channel 为 GPIO 的通道数（每个 GPIO 模块有两个通道），可选值为 1 或 2；在本实验中只使用了 channel1；

DirectionMask 是输入输出的位标识，对应位的值为 0 表示输出，1 表示输入。例如本实验中的 XGpio_SetDataDirection(&Gpio_button, 1, 0xf);则是将四个数据传输方向都设置成了输入。

（3）读函数 Xuint32 XGpio_DiscreteRead (XGpio *InstancePtr, unsigned Channel);

用来读取寄存器的值。

InstancePtr 是 Xgpio 结构体指针；

Channel 为 GPIO 的通道数，可选值为 1 或 2。在本实验中只使用了 channel1。

第四步 上板验证

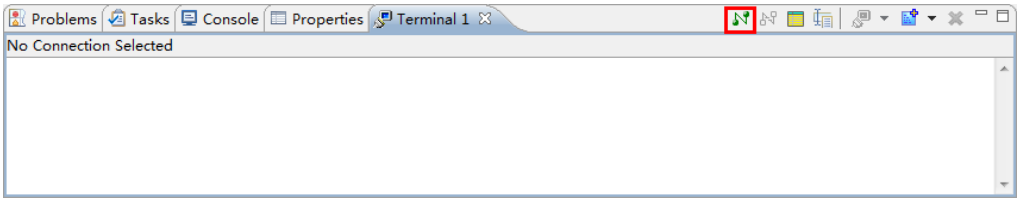
4-1. 将 Nexys4 与 PC 的 USB 接口连接

4-2. 查看端口号

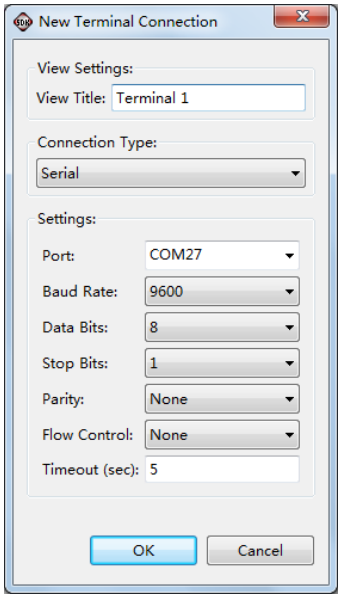
右键“我的电脑”—“属性”—在页面左侧选择“设备管理器”
发现与 com27 端口相连：（不同电脑可能有所区别，同一电脑每次连接也有可能有所区别）

4-3. 在 SDK 中打开串口：

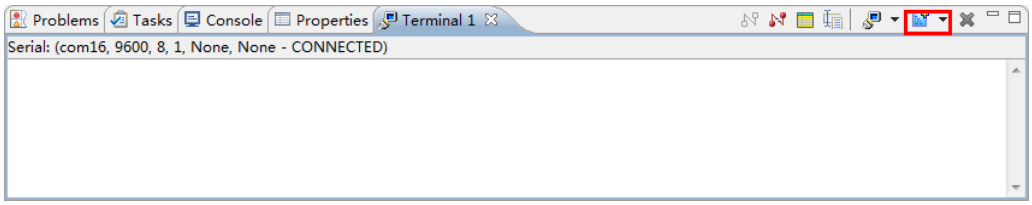
4-3-1.在下面的在页面下方找到 terminal 选项卡，然后点击绿色的连接按钮。



4-3-2.按照端口号和 XPS 中的波特率（baud rate）进行如下设置：

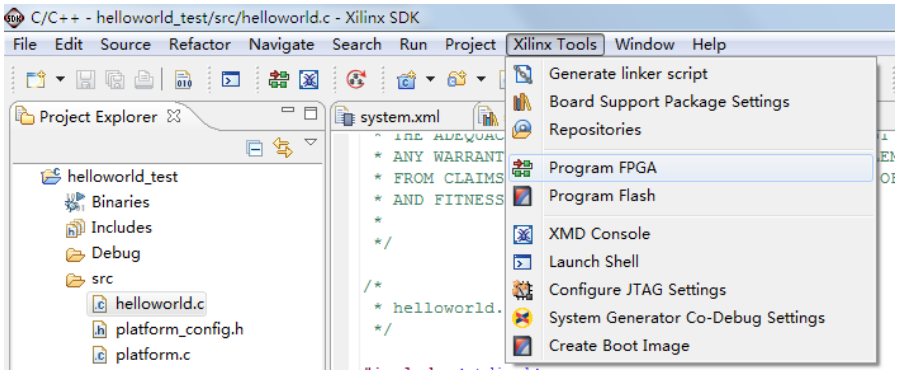


如果报了“no such port”的错误，可以通过新建串口，更改串口号：

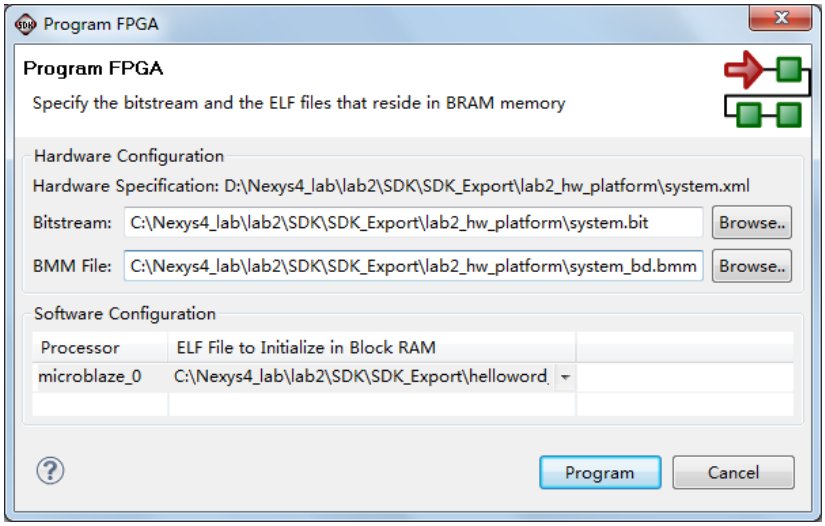


4-4. 将程序下载到板子上并运行

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



4-4-2.注意要选择正确的 elf 文件，点击 program



4-5. 按下 BTND 按钮和拨动 SW2、SW5 开关的运行结果：