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

Lab5: 基于 GPIO 的七段数码管实验

*基于 **Nexys 4 FPGA** 平台*

Lab 5: 基于 GPIO 的七段数码管实验

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 GPIO 与 UART两个 IP 核的添加方法，最终完成一个数码管显示的秒表的简单程序。

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 GPIO 与 UART 这两个 IP 核。
- 如何通过 C 语言实现秒表效果。

实验过程

本实验旨在指导读者使用 Xilinx 的 XPS 工具，调用 GPIO 与 UART 的 IP 核，并将导入到 SDK，调用它们，通过在串口进行对程序的调试，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。**

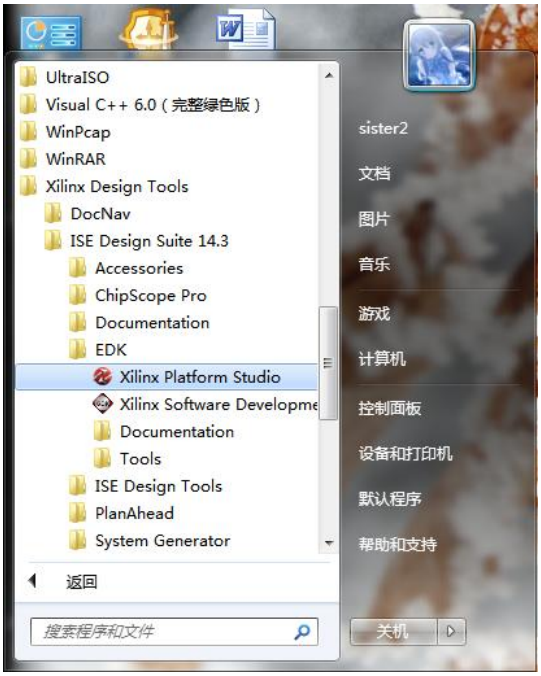


图 1：XPS 软件打开位置

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\testz，**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。**建立的工程的名称我们使用默认的 system.xmp 即可。点击 **OK**。

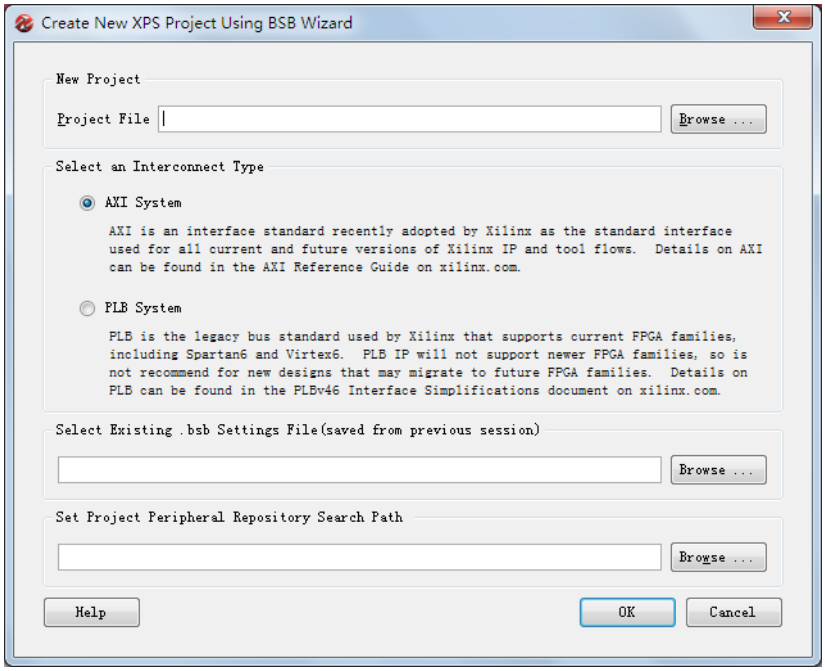


图 3：新工程建立向导

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: artrix 7
Device: XC7a1007
Package: CSG324
Speed: -3

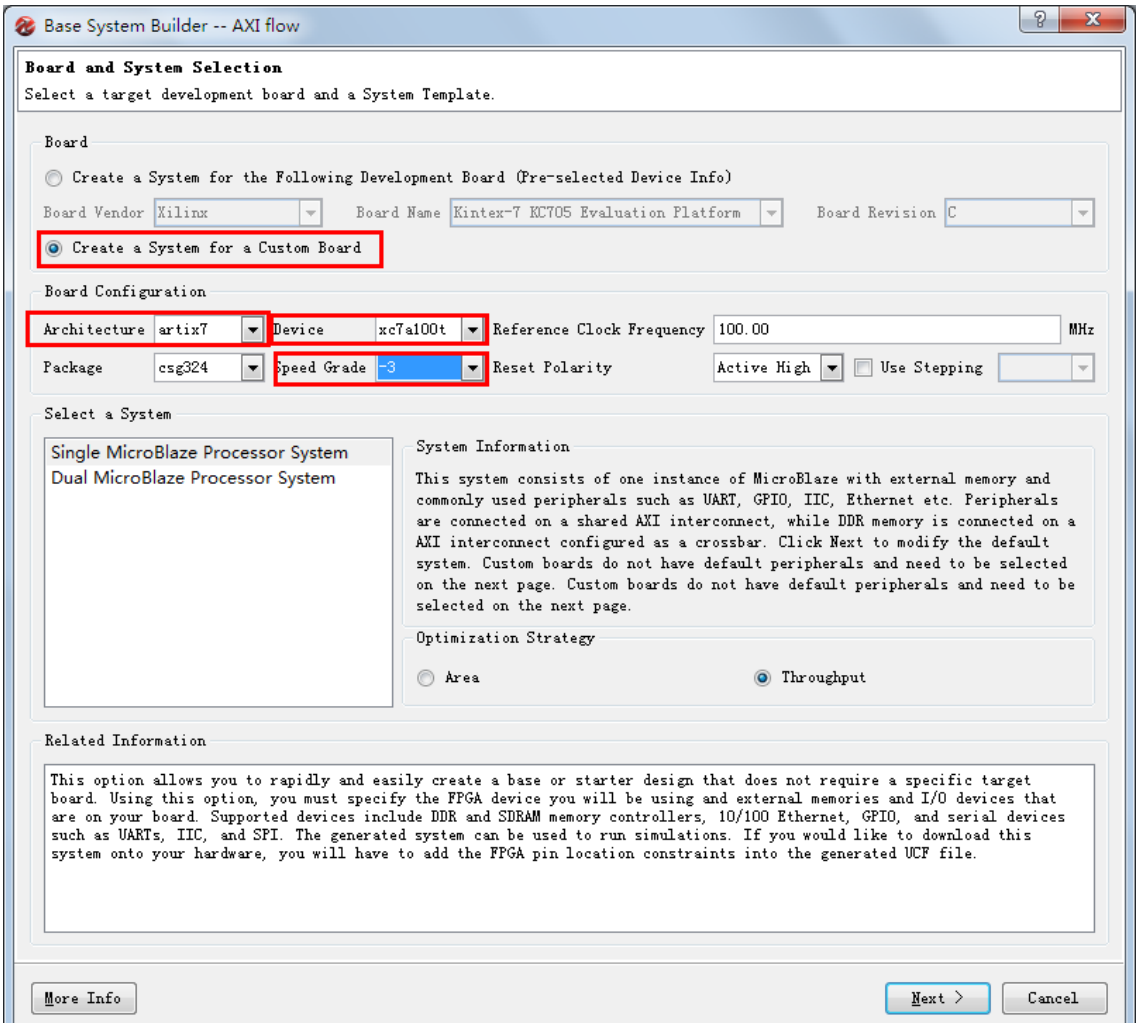


图 4：新工程参数设置

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。

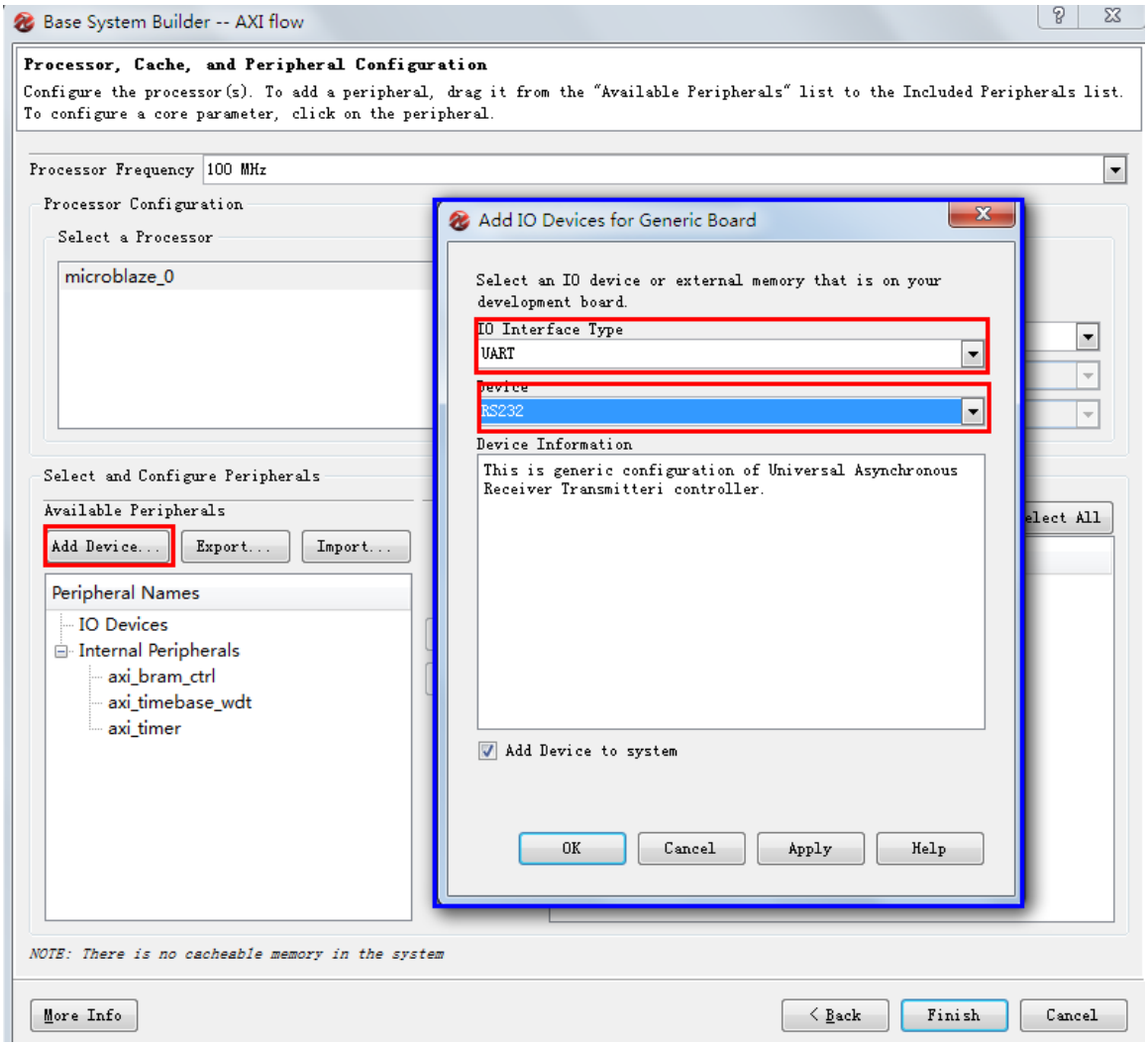


图 5：添加串口的 IP

在 IO Interface Type 中的下拉菜单中选择 GPIO。

在 Device 的下拉菜单中选择 LED_7segments，因为我们要通过 7 段数码管灯来展现秒表的效果，如图 6 所示。

单击 OK，即可添加可以控制 7 段数码管显示内容的 IP 核。

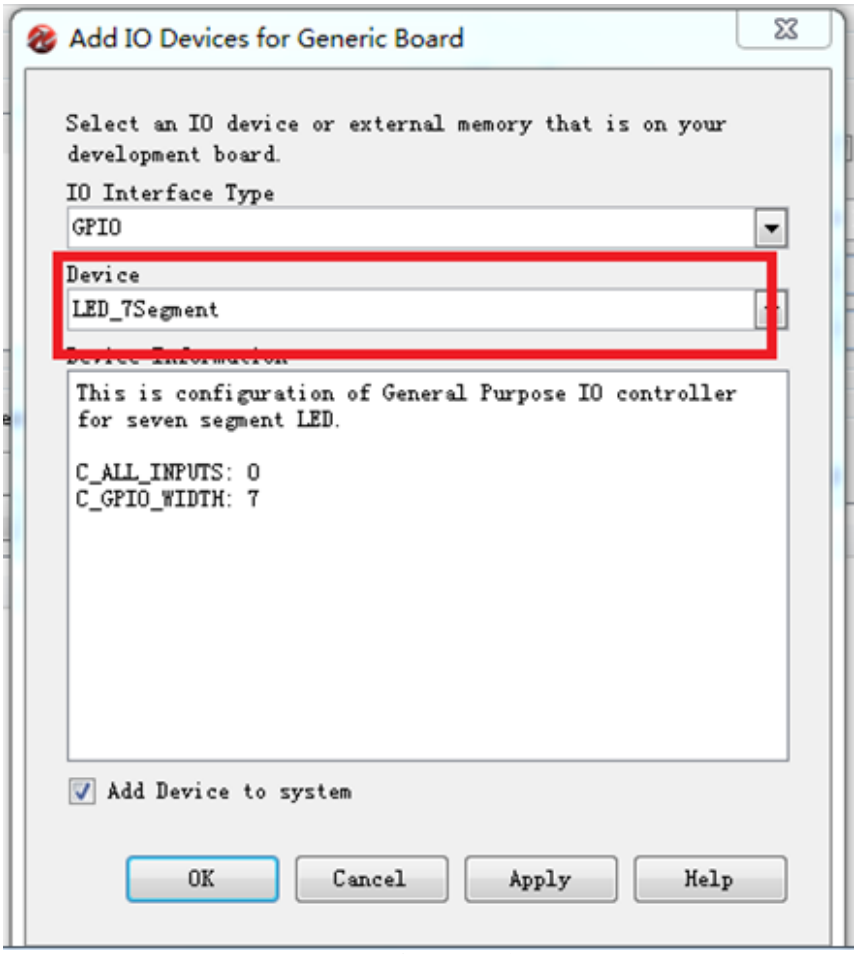


图 6：添加 GPIO 的 IP

1-1-6. 注意,串口的默认波特率设置为 9600。在 Lab2 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。

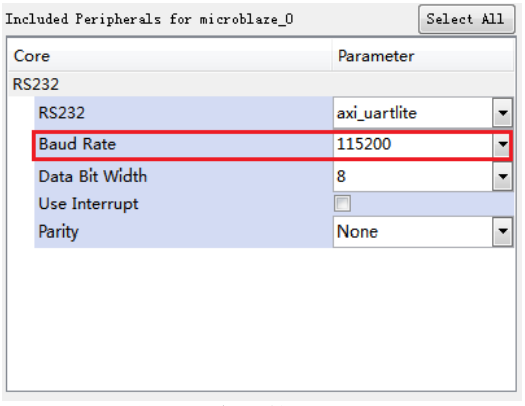


图 7：串口的设置

1-1-7. 下面我们要添加 7 段数码管的选通信号的 IP 核，在一个时刻只可能有一个数码管被选通，它的功能是给出一个 8 位的信号（低有效），控制具体哪个数码管要显示数据。只需添加一个普通的 GPIO IP 核即可。

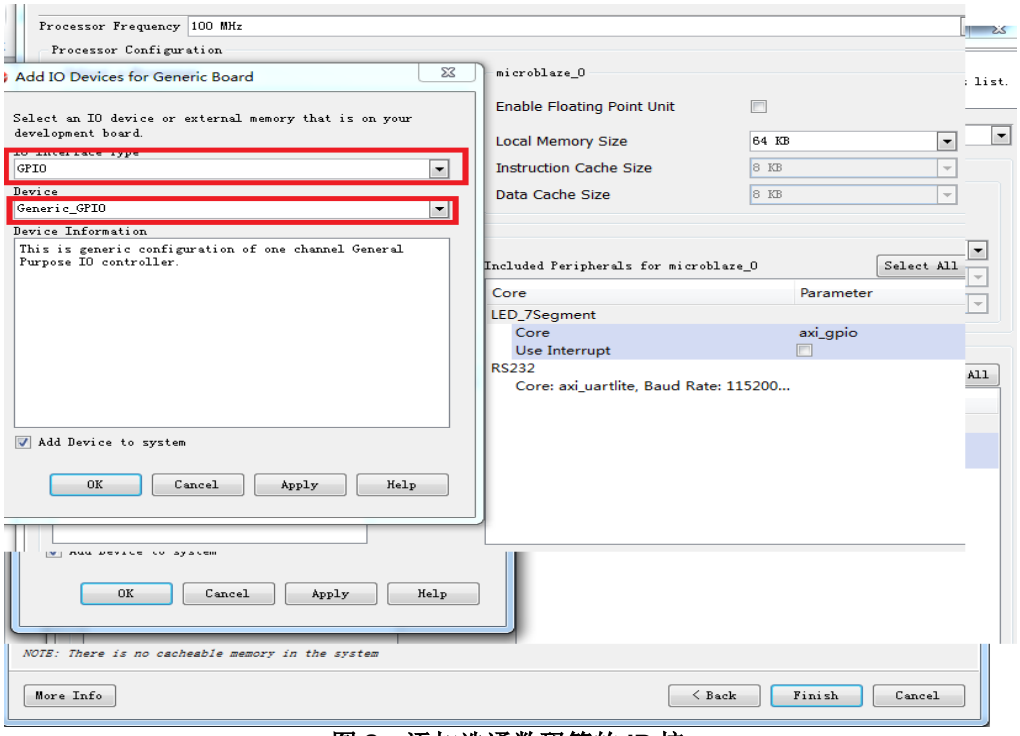


图 8：添加选通数码管的 IP 核

1-1-8. 下面我们要更改 FPGA 内部存储器 BRAM 的大小，将它的大小改为 64K，以防止存储空间太小无法存储软件代码部分。

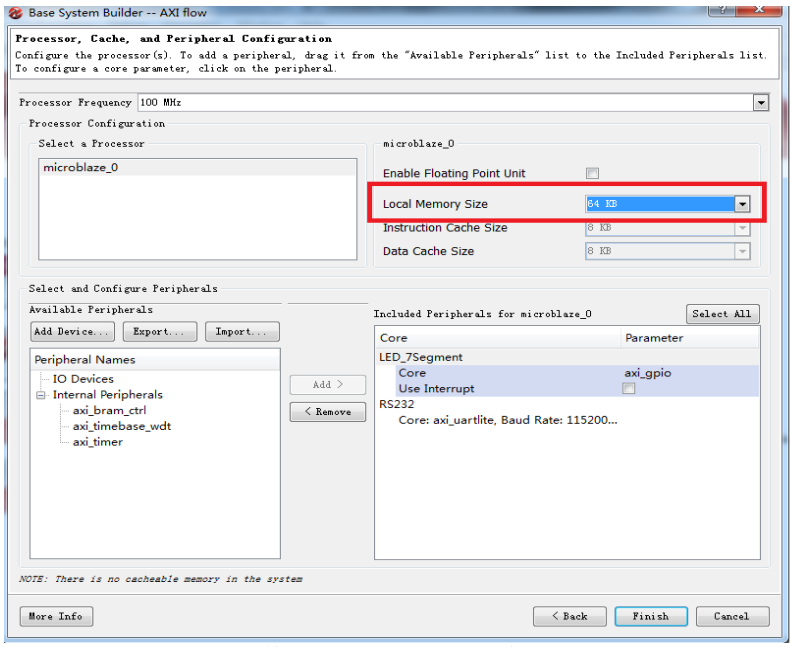


图 9：修改 local memory 大小

第二步 进行端口的互连

2-1. 在 PORT 选项卡中修改时钟的相关设置

2-1-1. Port 选项卡（展开 External Port），如图 10.

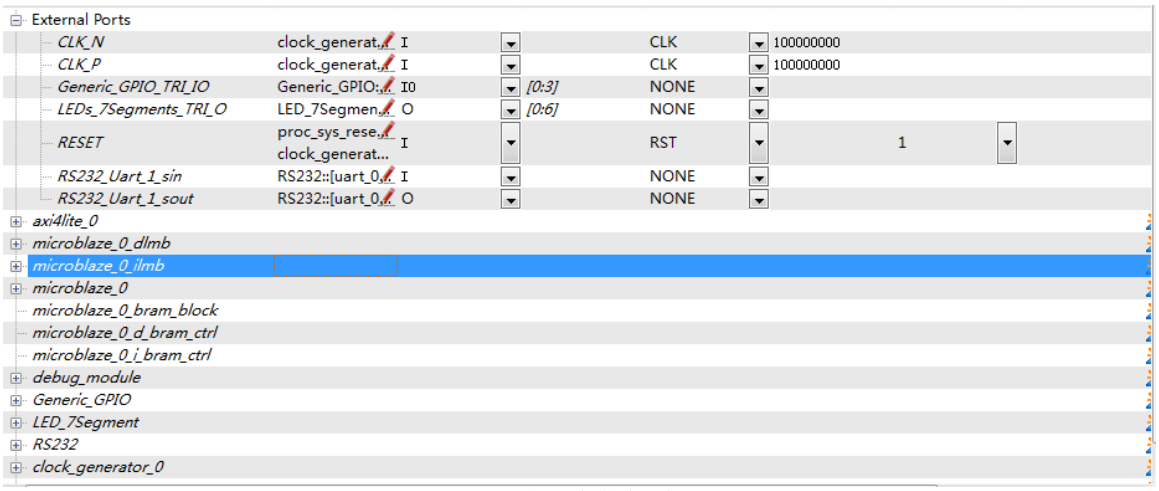


图 10：PORT 选项卡的初始状况

2-1-2. 点击打开 LED_7Segment 选项。

找到（**IO_IF**）**gpio_0 的 GPIO_IO_O，右键选中，在菜单中点击 Make external**。

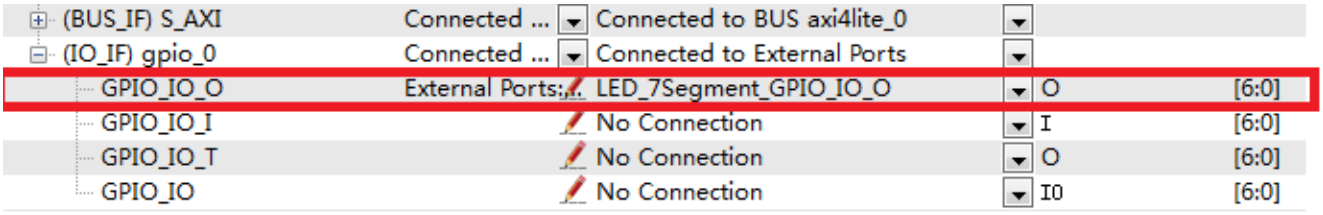


图 11：调整 GPIO 的设置

2-1-3. 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 12。

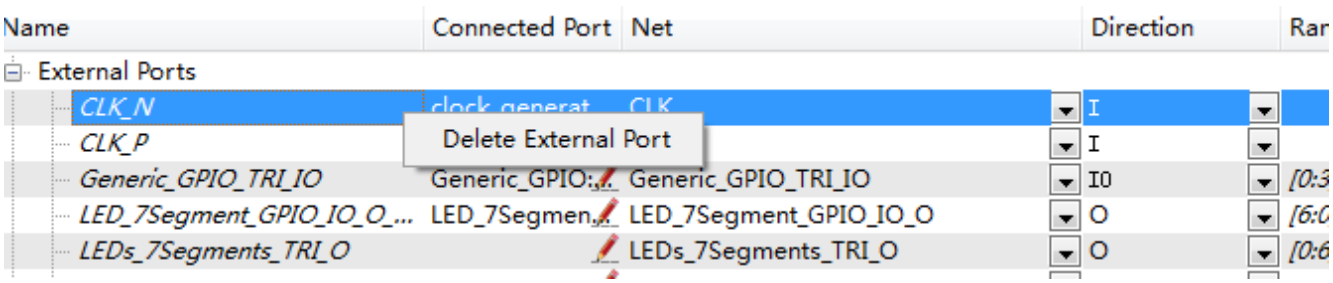


图 12：删除外部端口

2-1-4. 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**

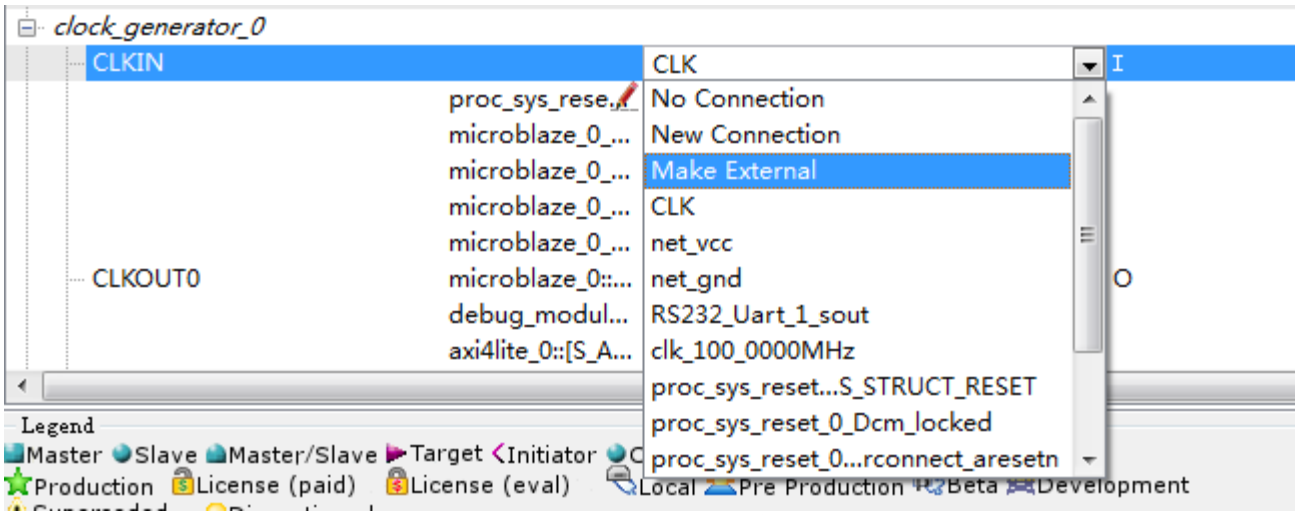


图 13： Clock_generator_0 中的 CLKIN

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

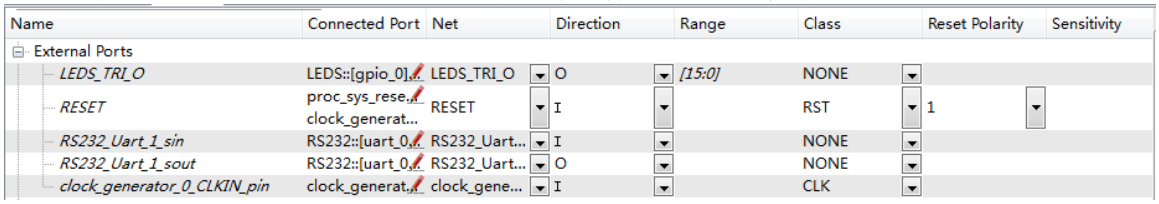


图 14：修改后的 External PORT

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\system.ucf**，ucf 文件在右侧打开

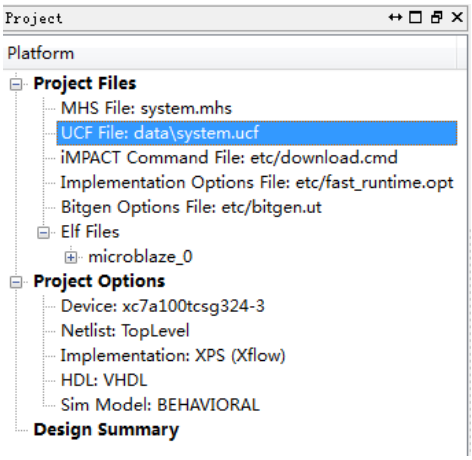
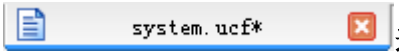


图 15：UCF 文件的位置

3-1-2. 这里我们手动输入 LOC（引脚位置）约束代码，如**图 16。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。

```
1  ## This file is a general .ucf for the Nexys4 rev B board
2  ## To use it in a project:
3  ## - uncomment the lines corresponding to used pins
4  ## - rename the used signals according to the project
5
6  ## Clock signal
7  NET "clock_generator_0_CLKIN_pin"   LOC = "E3"  | IOSTANDARD = "LVCMOS33";             #Bank = 35, Pin name =
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";    #Bank = 34, Pin name = IO_L21P_T3_DQS_34,
13
14 ## 7 segment display
15 NET "LEDs_7Segments_TRI_O<0>"       LOC = "L3"  | IOSTANDARD = "LVCMOS33";    #Bank = 34, Pin name = IO_L2N_
16 NET "LEDs_7Segments_TRI_O<1>"       LOC = "N1"  | IOSTANDARD = "LVCMOS33";    #Bank = 34, Pin name = IO_L3N_
17 NET "LEDs_7Segments_TRI_O<2>"       LOC = "L5"  | IOSTANDARD = "LVCMOS33";    #Bank = 34, Pin name = IO_L6N_
18 NET "LEDs_7Segments_TRI_O<3>"       LOC = "L4"  | IOSTANDARD = "LVCMOS33";    #Bank = 34, Pin name = IO_L5N_
19 NET "LEDs_7Segments_TRI_O<4>"       LOC = "K3"  | IOSTANDARD = "LVCMOS33";    #Bank = 34, Pin name = IO_L2P_
20 NET "LEDs_7Segments_TRI_O<5>"       LOC = "M2"  | IOSTANDARD = "LVCMOS33";    #Bank = 34, Pin name = IO_L4N_
21 NET "LEDs_7Segments_TRI_O<6>"       LOC = "L6"  | IOSTANDARD = "LVCMOS33";    #Bank = 34, Pin name = IO_L6P_
22
23 #NET "dp"            LOC = "M4"  | IOSTANDARD = "LVCMOS33";    #Bank = 34, Pin name = IO_L16P_T2_34,
24
25 NET "axi_gpio_0_GPIO_IO_pin<0>"        LOC = "N6"  | IOSTANDARD = "LVCMOS33";    #Bank = 34, Pin name = IO_L
26 NET "axi_gpio_0_GPIO_IO_pin<1>"        LOC = "M6"  | IOSTANDARD = "LVCMOS33";    #Bank = 34, Pin name = IO_L
27 NET "axi_gpio_0_GPIO_IO_pin<2>"        LOC = "M3"  | IOSTANDARD = "LVCMOS33";    #Bank = 34, Pin name = IO_L
28 NET "axi_gpio_0_GPIO_IO_pin<3>"        LOC = "N5"  | IOSTANDARD = "LVCMOS33";    #Bank = 34, Pin name = IO_L
29 NET "axi_gpio_0_GPIO_IO_pin<4>"        LOC = "N2"  | IOSTANDARD = "LVCMOS33";    #Bank = 34, Pin name = IO_L
30 NET "axi_gpio_0_GPIO_IO_pin<5>"        LOC = "N4"  | IOSTANDARD = "LVCMOS33";    #Bank = 34, Pin name = IO_L
```

图 16：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**。

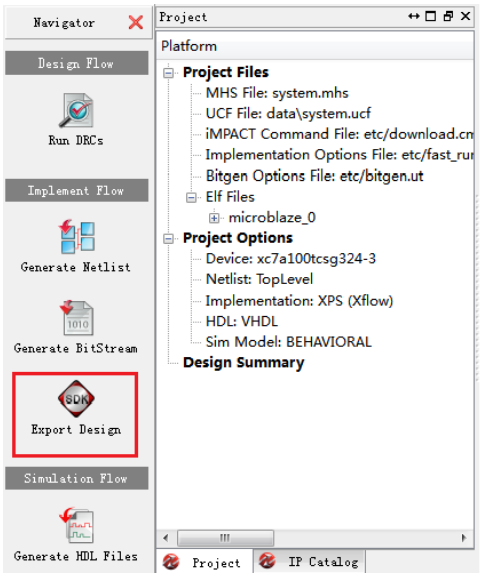


图 17：export design

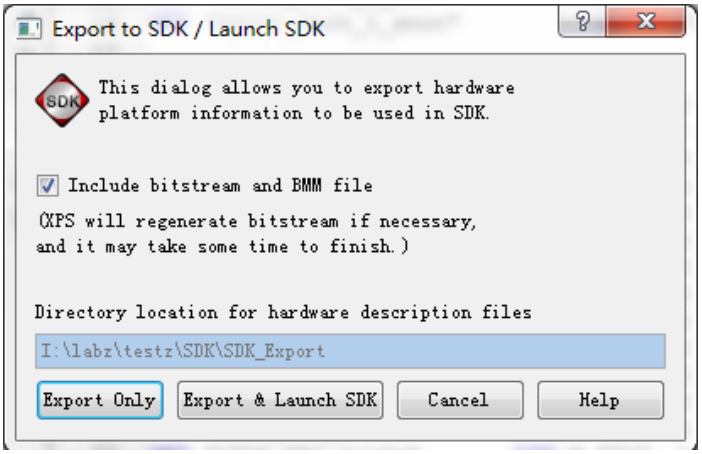


图 18：在弹出的对话框中选择 Export & launch sdk

3-1-4. 选择 SDK 导入路径

注意要具体到..\sdk\sdk_export

点击 ok

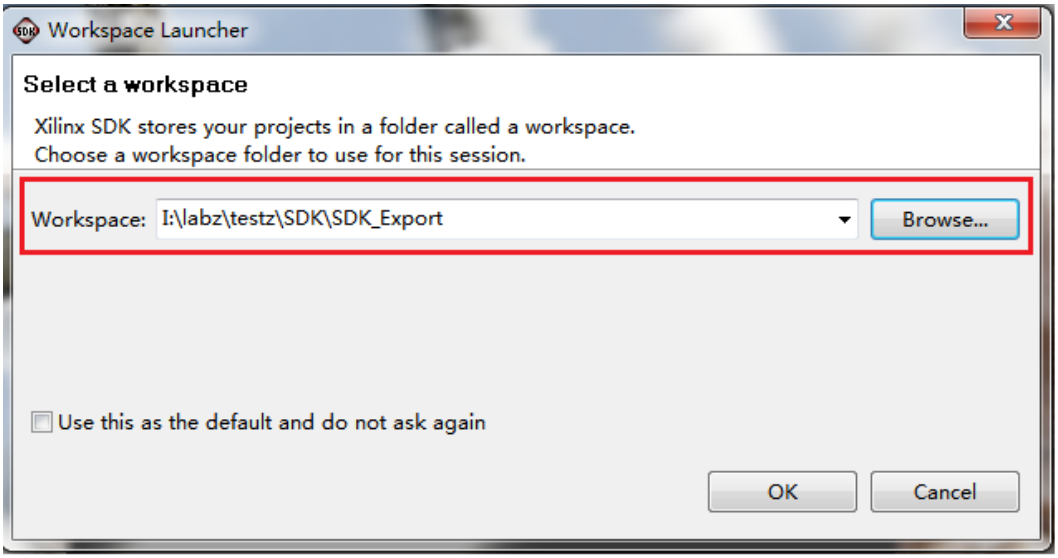
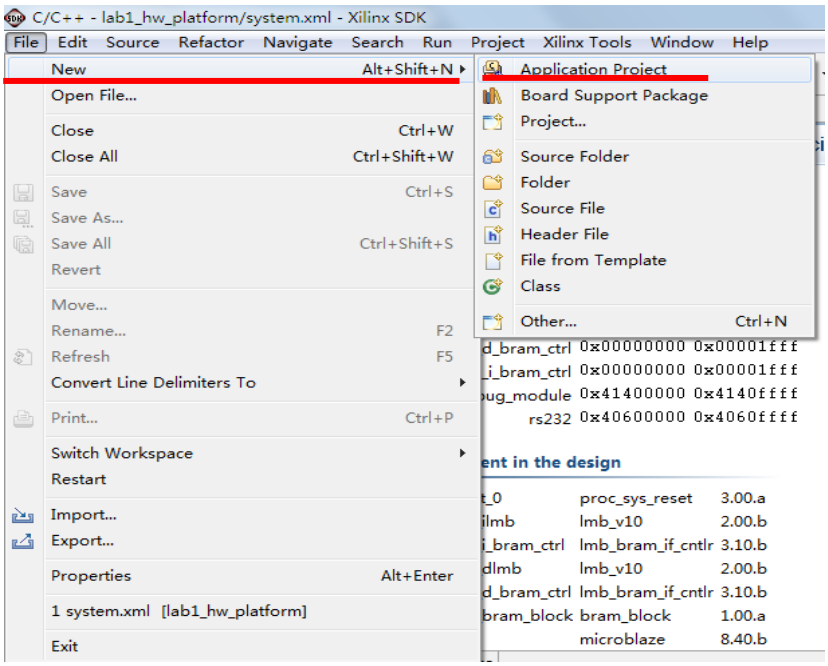


图 19：选择 SDK 导入路径

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**



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

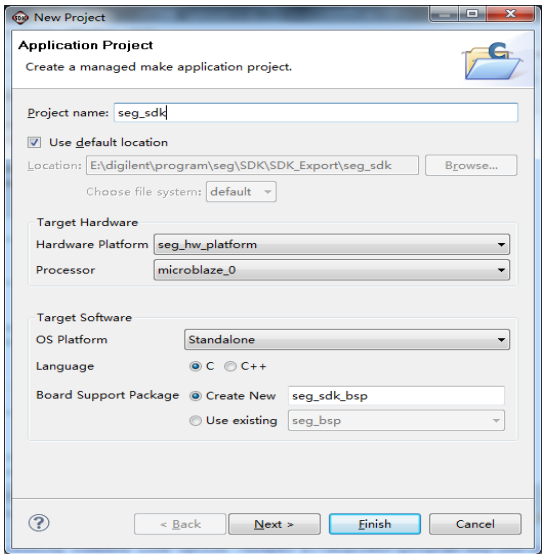


图 20：建立软件工程

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

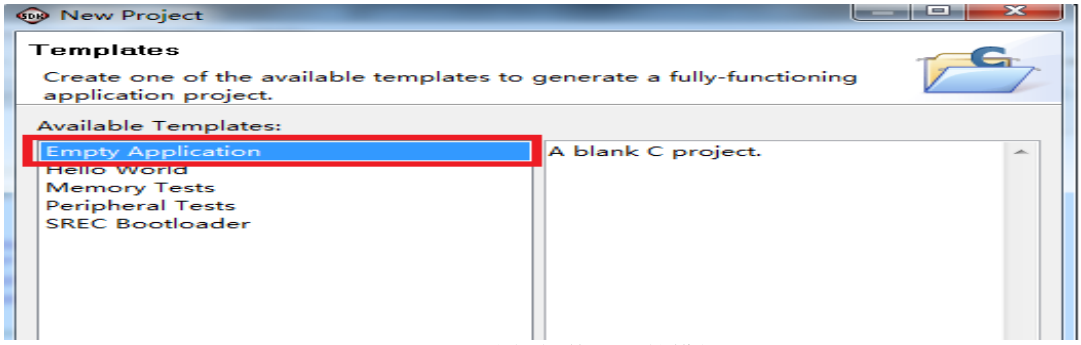


图 21：选择软件工程的模板

4-1-4. 因为开始时建立的是空工程，所以需要手动添加源文件，并输入文件名（注意后缀），如图 **22** 和 **23.**

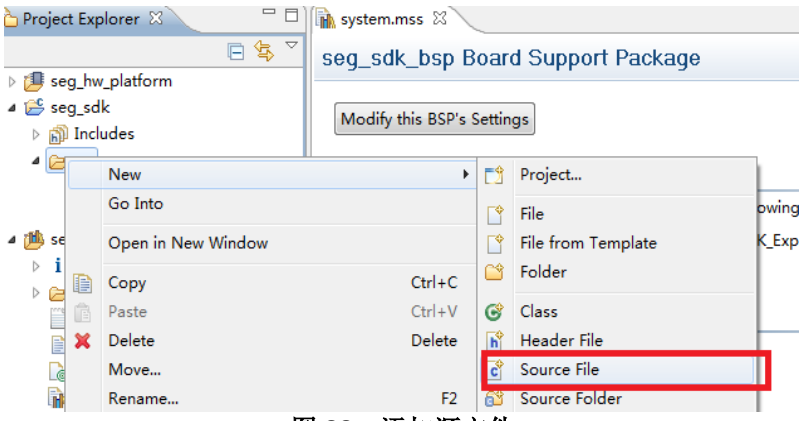


图 22：添加源文件

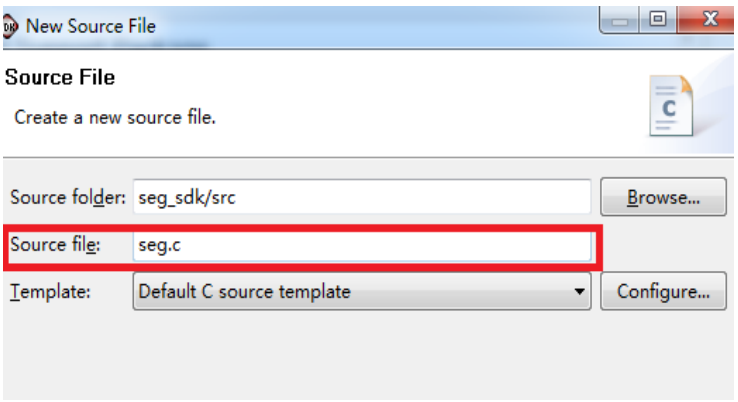


图 23：输入文件名

4-1-5. 添加完毕以后可以在左侧双击源文件，查看这段代码：

system.mss | seg.c

```
/*
 * seg.c
 *
 *  Created on: 2013-11-26
 *      Author: tp
 */

#include "xparameters.h"
#include "xgpio.h"
#include "xil_io.h"
#include "xil_printf.h"


XGpio Gpio,Gpio1;


int select(int m);


int main(void)
{

    int Status1,Status2,i,p,q,temp1,temp2;
```

```
    Status1 = XGpio_Initialize(&Gpio, XPAR_AXI_GPIO_0_DEVICE_ID);
    if (Status1 != XST_SUCCESS)
        return XST_FAILURE;
    }

    Status2 = XGpio_Initialize(&Gpio1, XPAR_LED_7SEGMENT_DEVICE_ID);
        if (Status2 != XST_SUCCESS) {
            return XST_FAILURE;
        }
    XGpio_SetDataDirection(&Gpio, 0x1, 0);

    while(1)
    {
        for(p=0;p<10;p++)
        {
            temp2=select(p);

        for(q=0;q<100;q++)
            {
                XGpio_DiscreteWrite(&Gpio1, 0x1, temp2);
                XGpio_DiscreteWrite(&Gpio, 0x1, 0xfd);
                for(i=0;i<60000;i++);
                temp1=select(q/10);

                XGpio_DiscreteWrite(&Gpio1, 0x1, temp1);
                XGpio_DiscreteWrite(&Gpio, 0x1, 0xfe);
                for(i=0;i<60000;i++);
```

对 GPIO1 初始化

对 GPIO2 初始化

数码管译码函数

```
            }
        }
        }

        return XST_SUCCESS;

}
int select(int m)
{
    int temp=0;
    switch(m)
    {
    case 0:
        temp=0b0000001;
        break;
    case 1:
        temp=0b1001111;
        break;
    case 2:
        temp=0b0010010;
        break;
    case 3:
        temp=0b0000110;
        break;
```

数码管译码函数

```
        case 4:
            temp=0b1001100;
            break;
        case 5:
            temp=0b0100100;
            break;
        case 6:
            temp=0b0100000;
            break;
        case 7:
            temp=0b0001111;
            break;
        case 8:
            temp=0b0000000;
            break;
        case 9:
            temp=0b0000100;
            break;
        default:
            temp=0b0000001;
            break;
        }
        return temp;

}
```

图 24：数码管完整代码

注意：本段代码是模拟一个计数器运行的状态，一直计数记到 99，然后归零。在主函数中首先对 XPS 里建立的两个 GPIO 进行初始化，，初始化完毕之后就开始对数码管的寄存器进行写操作，利用 select 译码函数，把想要显示的数字转换成相应的二进制数据存入数码管的寄存器之中。利用了两个 FOR 循环来控制计数的大小，用最里面空走的两个 FOR 循环作为数码管的刷新频率。

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）

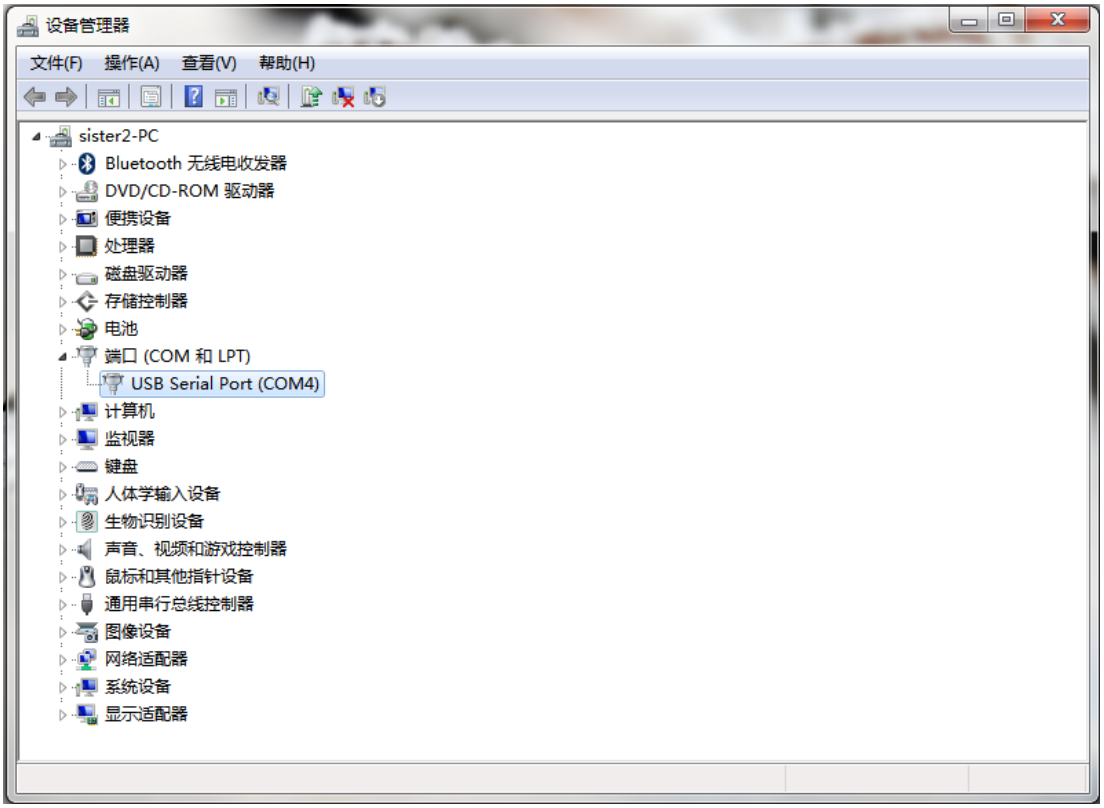


图 25：查看端口号

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。

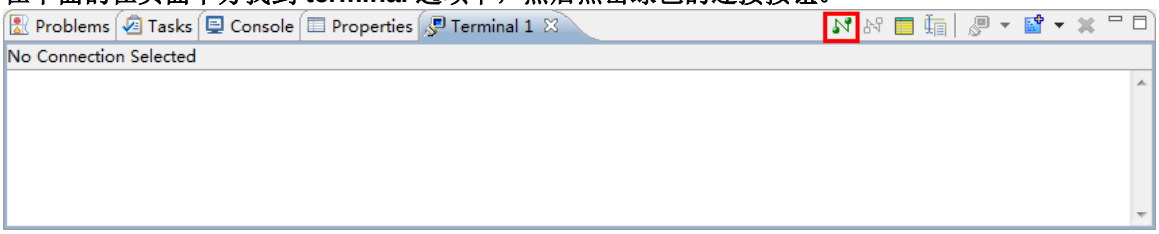


图 26：连接端口

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

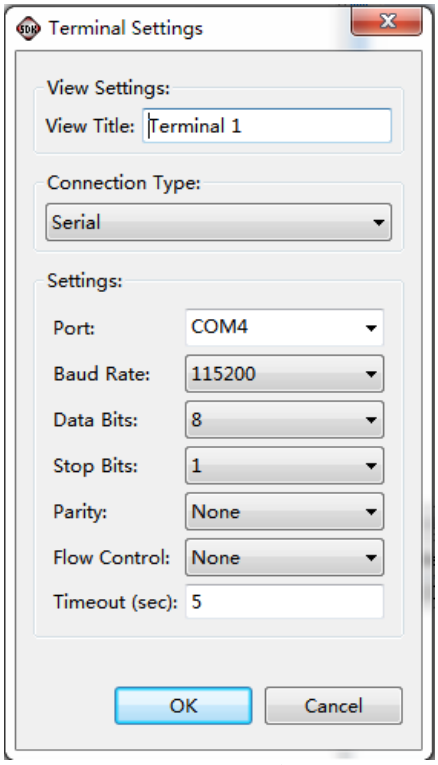
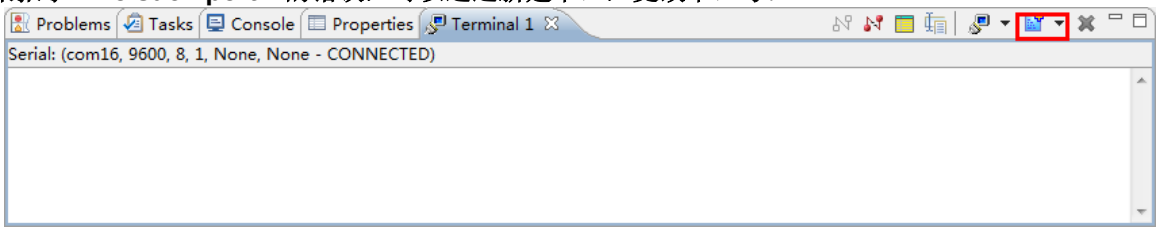


图 27：设置端口

如果报了“no such port”的错误，可以通过新建串口，更改串口号：



5-4. 将程序下载到板子上并运行

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

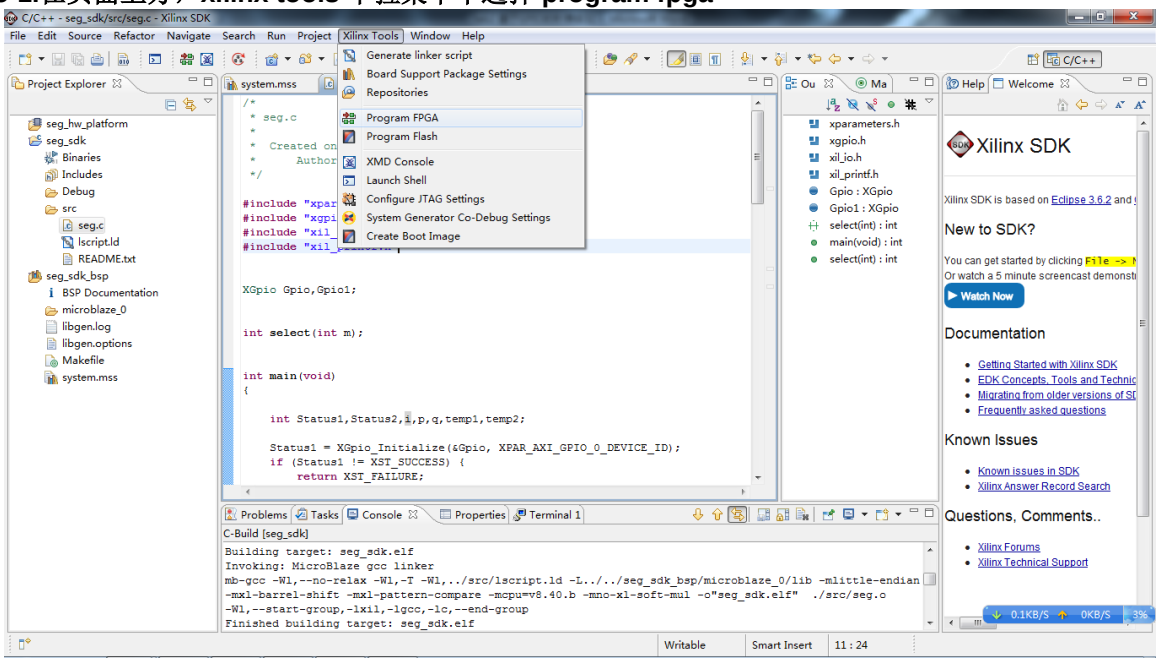


图 29：将 C 语言程序下载到开发板中

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

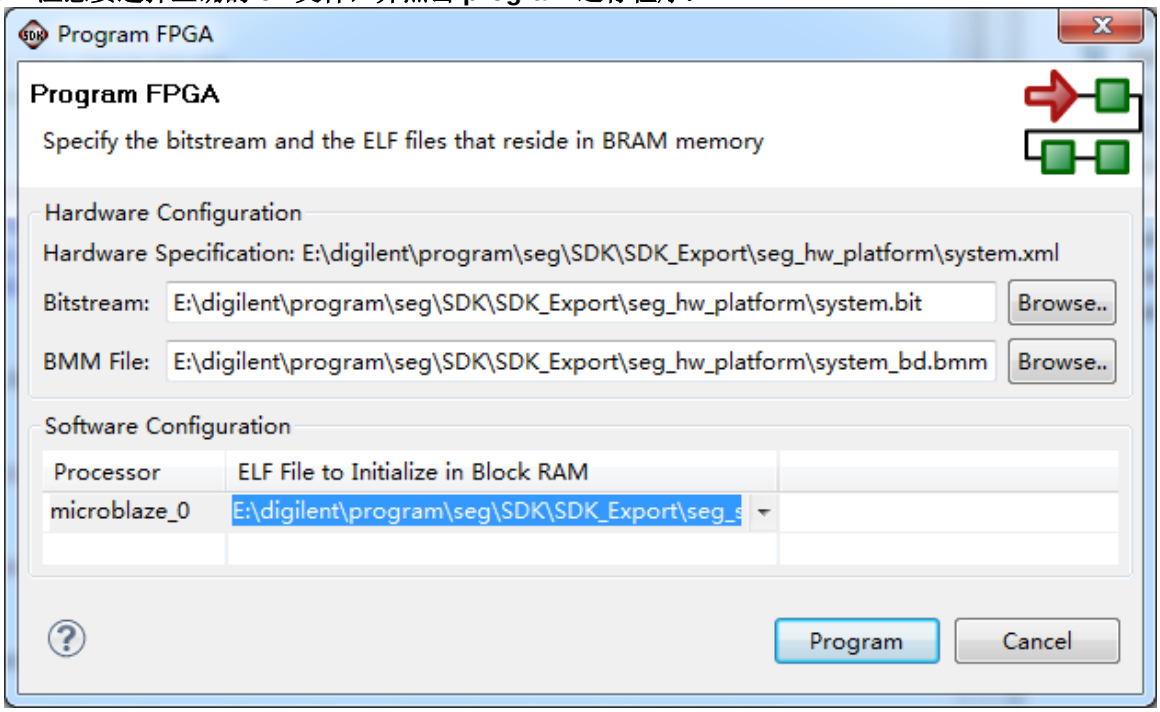


图 30：选择 elf 文件

5-5-3. 实验现象：这个代码只控制了 8 个 7 段数码管中的最右边 2 个数码管，这两个数码管会从 0 开始计数，一直计数记到 99 然后清零并循环下去。