

Lab12: Audio

——音频接口实验

基于 *Nexys 4 FPGA* 平台

Lab 12: Audio

实验简介

本实验旨在指导读者添加自定义音频 IP 核，并完成敲击键盘按键进行简单演奏的例程。

实验目标

在完成本实验后，您将学会：

- 添加自定义 IP 核
- 敲击键盘上不同按键发出不同音符的数字电路的实现

实验过程

本实验指导读者使用 Xilinx 的 XPS 工具，添加自定义的音频 IP 核，并导出到 SDK，调用这个 IP 核，使得敲击键盘上不同的按键发出不同的音符，然后在 Nexys 4 平台上进行测试验证。

实验由以下步骤组成：

1. 新建 XPS 工程
2. 添加自定义 IP 核
3. 添加用户约束文件
4. 将工程导出到 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 创建工程

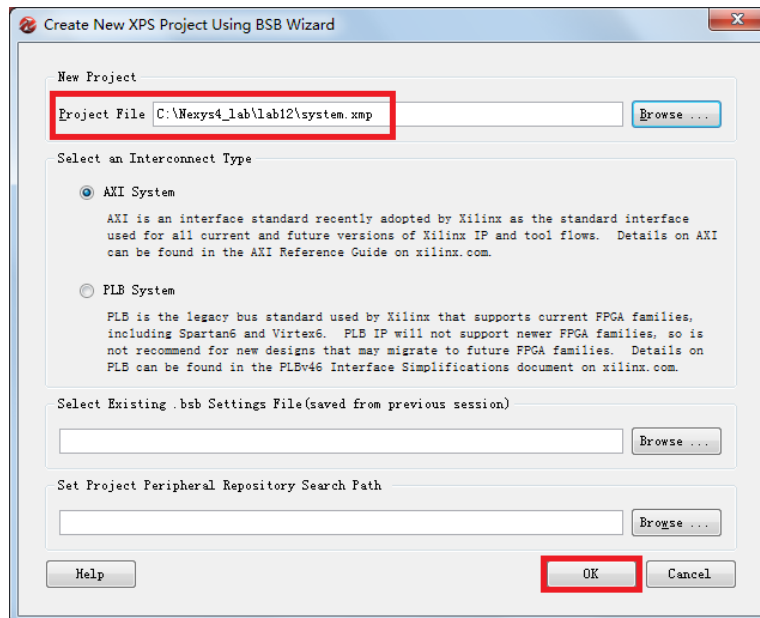
1-1. 运行 **Xilinx Platform Studio**, 创建一个空的新工程, 基于 **XC7A100TCSG324-3** 芯片和 **VHDL** 语言。

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。)。

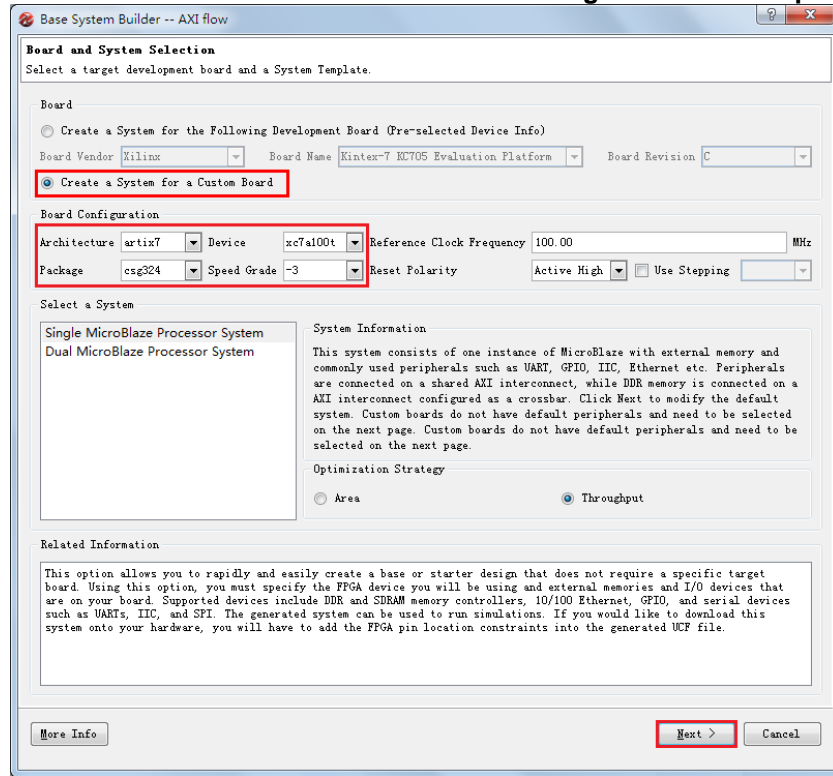
1-1-2. 点击 **Create New Project Using Base System Builder** 来打开新工程建立向导。



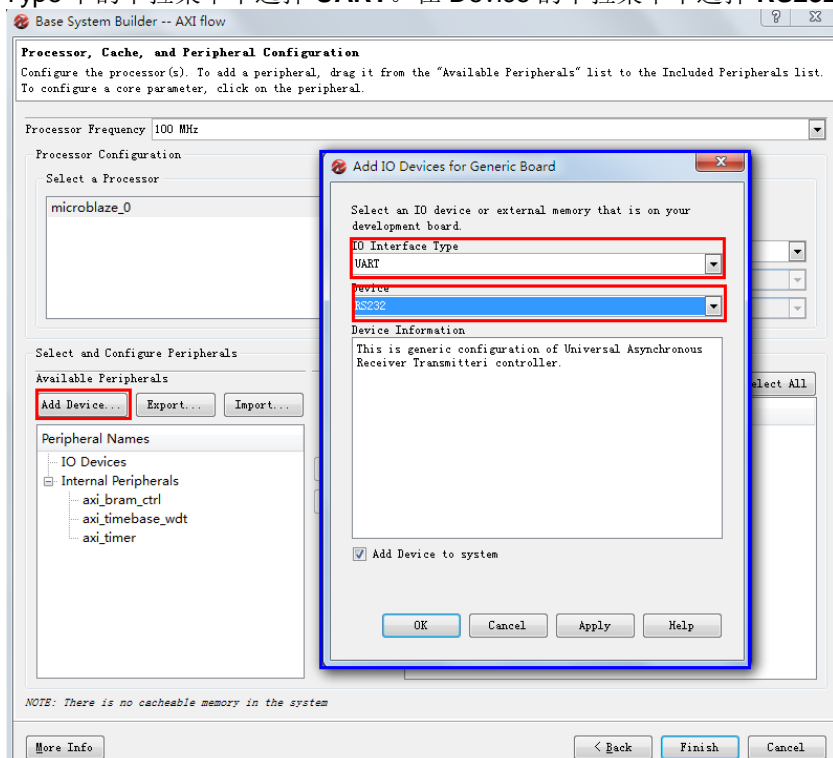
1-1-3. 在弹出的 **Create New XPS Project Using BSB Wizard** 对话框的 **Project File** 栏选择工程建立后存放的路径, 这里可以选择 **C:\Nexys4_lab\lab12\system.xmp**, (注意路径中不要有中文和空格), 点击 **OK**。



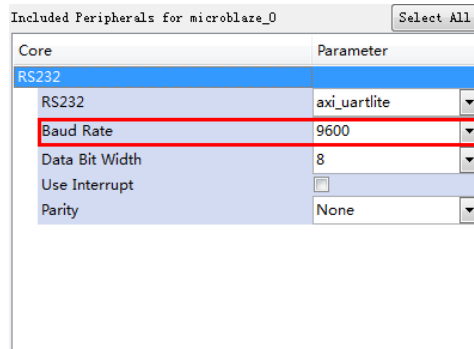
- 1-1-4. 新出现的是关于工程的一些参数设置的对话框，设置如下的参数后，点击 **Next**，如下图。
Architecture: ARTIX 7 Device: XC7A100T Package: CSG324 Speed: -3



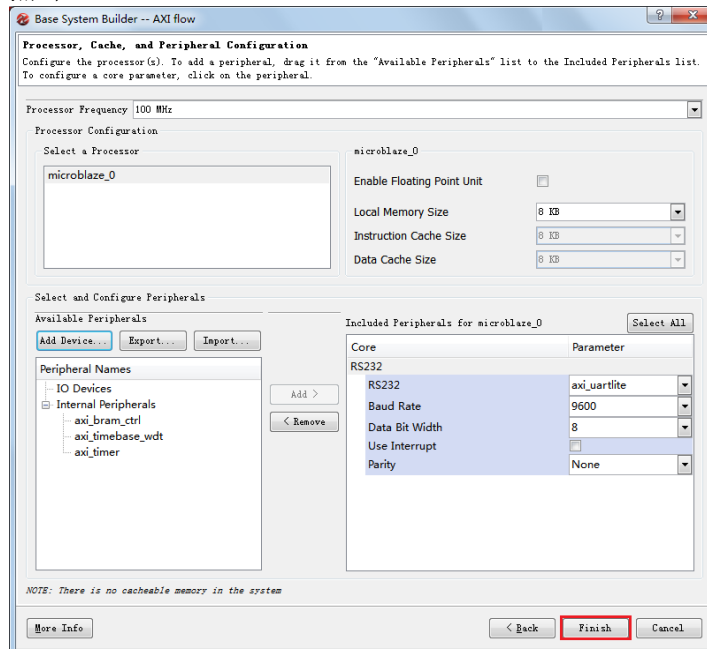
- 1-1-5. 在接下来出现的页面中选择要添加的 IP 核，并设置 IP 核的参数：
 单击 **Select and configure Peripherals** 下的 **Add Device...**，出现下图中蓝色的对话框。在 **IO Interface Type** 中的下拉菜单中选择 **UART**。在 **Device** 的下拉菜单中选择 **RS232**。单击 **OK**



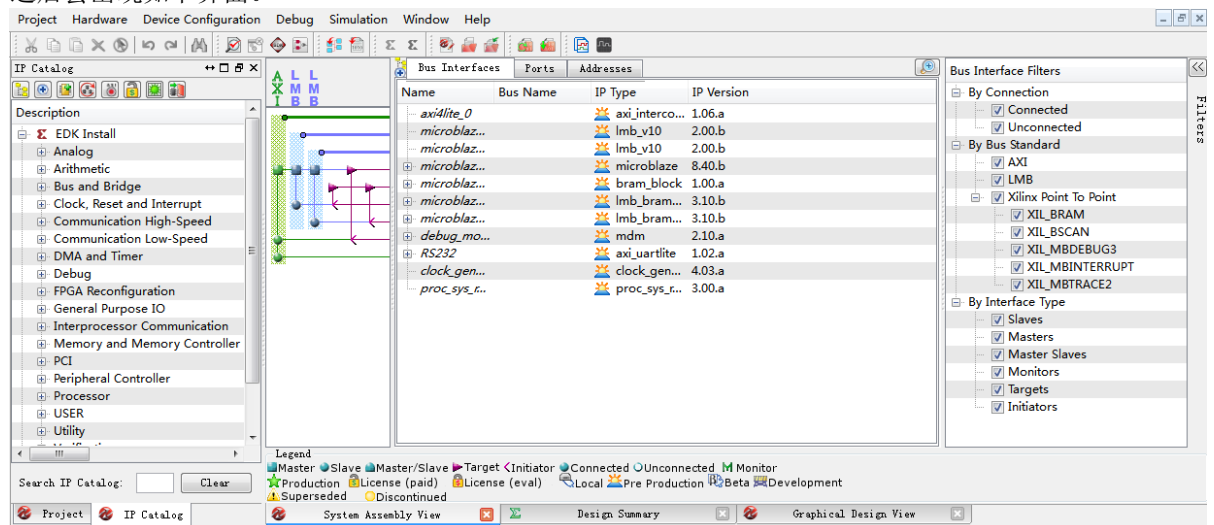
1-1-6. 注意,串口的默认波特率设置为 9600。在 Lab12 中,不做修改。以后的设计中根据需要进行调整。但是为了确保串口的正常通讯, SDK 工程中的 Terminal 的波特率以及串口的其他设置必须与之保持一致。



1-1-7. 点击 Finish



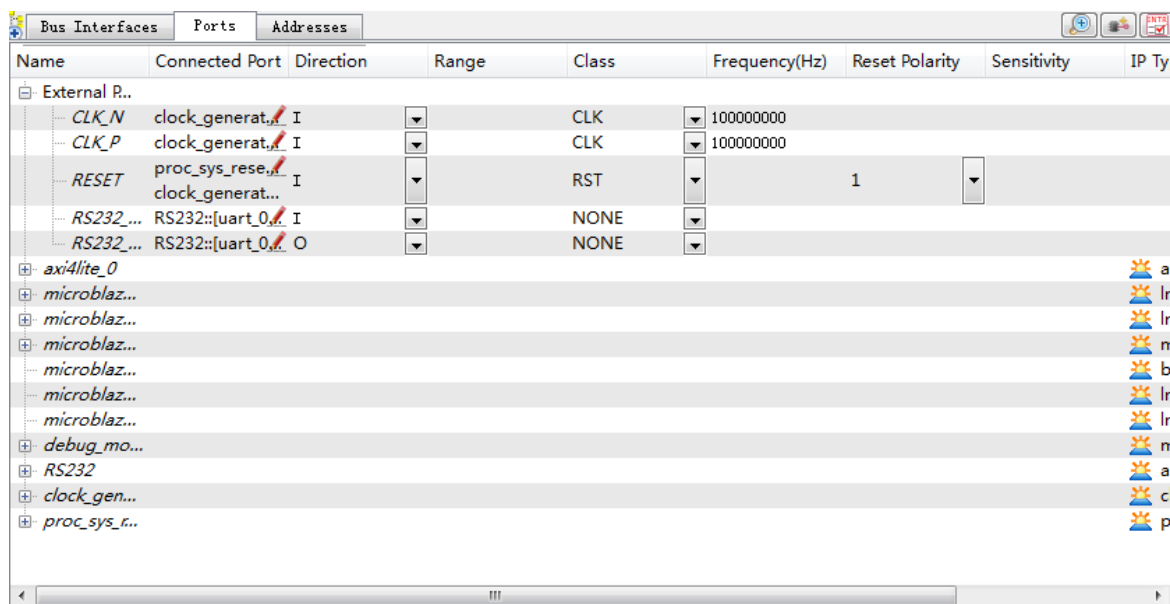
之后会出现如下界面。



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

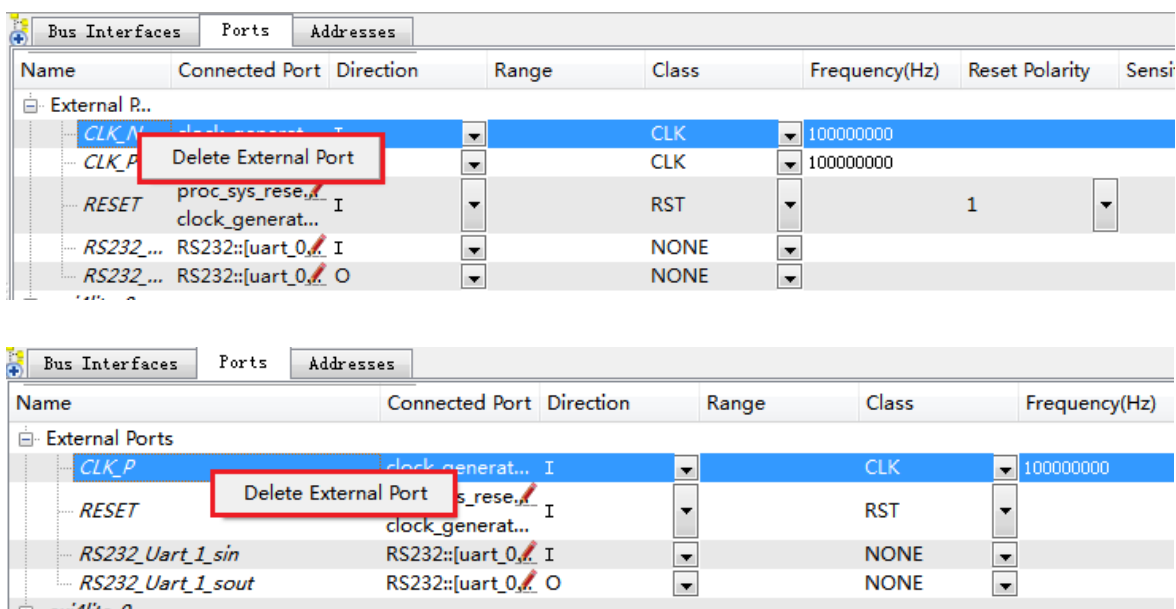
2-1-1. Port 选项卡（展开 External Port），如下图：



Name	Connected Port	Direction	Range	Class	Frequency(Hz)	Reset Polarity	Sensitivity	IP Ty
External P...								
CLK_N	clock_generat...	I		CLK	100000000			
CLK_P	clock_generat...	I		CLK	100000000			
RESET	proc_sys_rese...	I		RST		1		
RS232_...	RS232::[uart_0...	I		NONE				
RS232_...	RS232::[uart_0...	O		NONE				
axi4lite_0								a
microblaz...								lr
microblaz...								lr
microblaz...								n
microblaz...								b
microblaz...								lr
microblaz...								lr
debug_mo...								n
RS232								a
clock_gen...								c
proc_sys_...								p

2-1-2. 删去 External Port 中的 CLK_N 和 CLK_P。

右键选中该端口，然后点击 Delete External Port，如下图

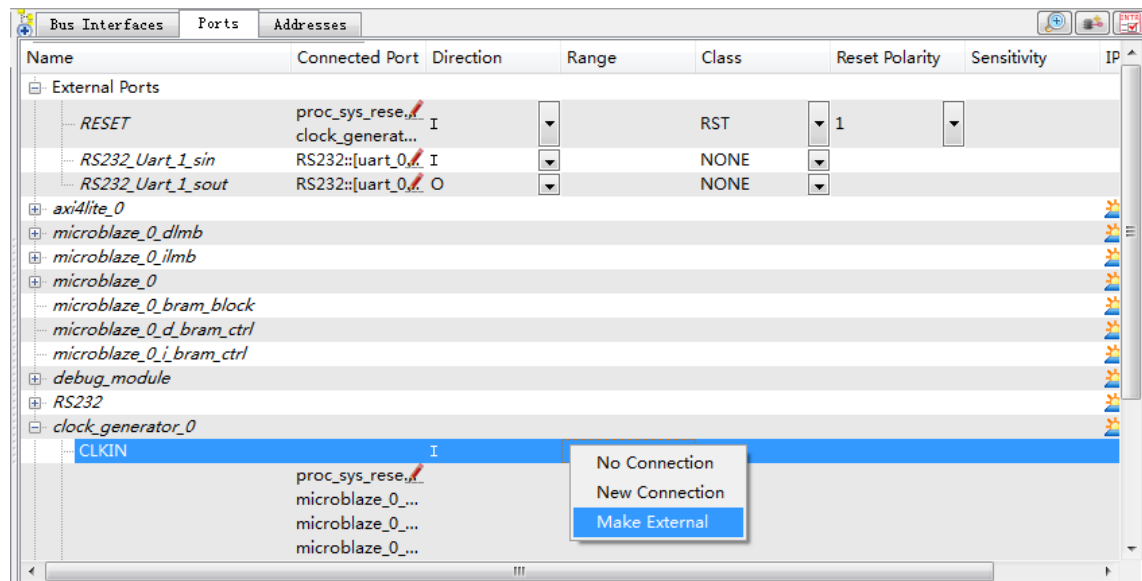


Name	Connected Port	Direction	Range	Class	Frequency(Hz)	Reset Polarity	Sensi
External P...							
CLK_N	clock_generat...	I		CLK	100000000		
CLK_P	clock_generat...	I		CLK	100000000		
RESET	proc_sys_rese...	I		RST		1	
RS232_...	RS232::[uart_0...	I		NONE			
RS232_...	RS232::[uart_0...	O		NONE			

Name	Connected Port	Direction	Range	Class	Frequency(Hz)
External Ports					
CLK_P	clock_generat...	I		CLK	100000000
RESET	proc_sys_rese...	I		RST	
RS232_Uart_1_sin	RS232::[uart_0...	I		NONE	
RS232_Uart_1_sout	RS232::[uart_0...	O		NONE	

2-1-3. 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**



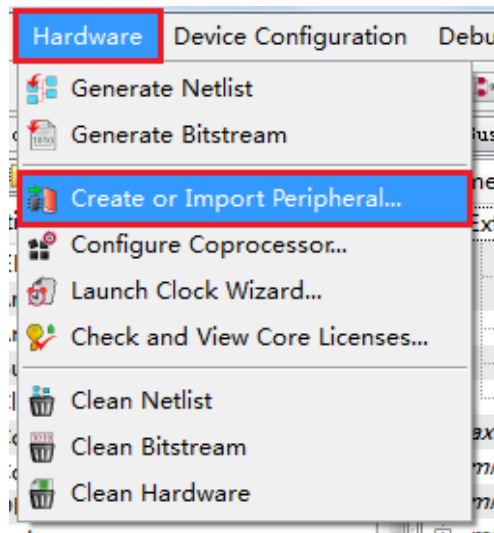
注意 **External** 中的 **Name** 一项，这是我们添加用户约束文件（UCF）的依据。

Name	Connected Port	Direction	Range	Class	Reset Polarity	Se
External Ports						
RESET	proc_sys_res...	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		

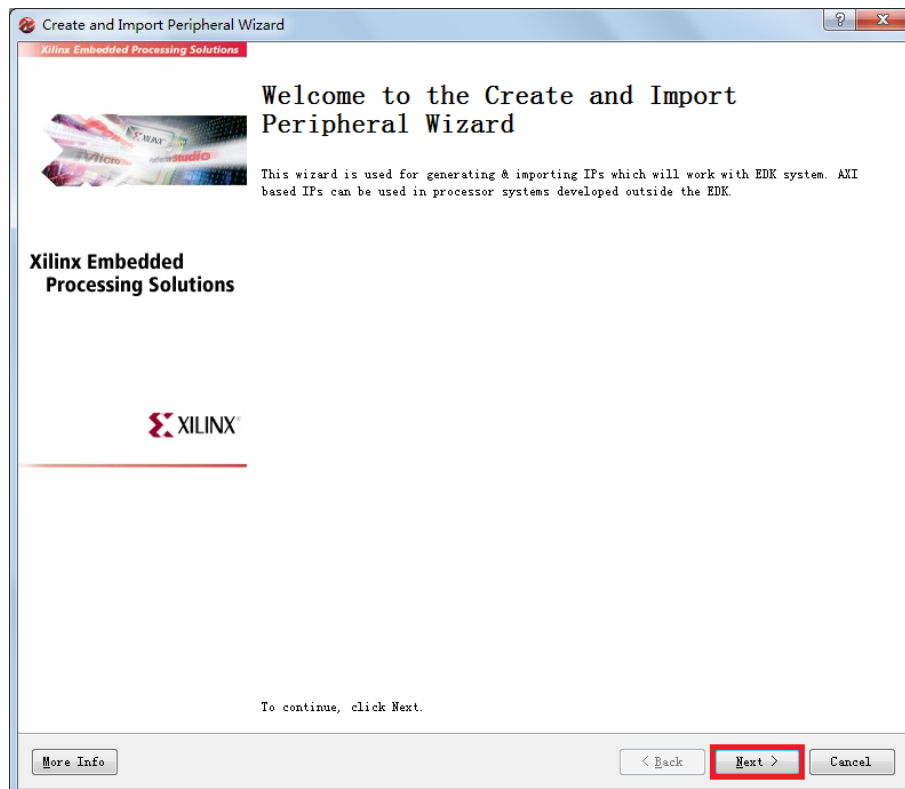
第三步 创建并添加用户自定义 IP 核

3-1. 创建用户自定义 IP 核

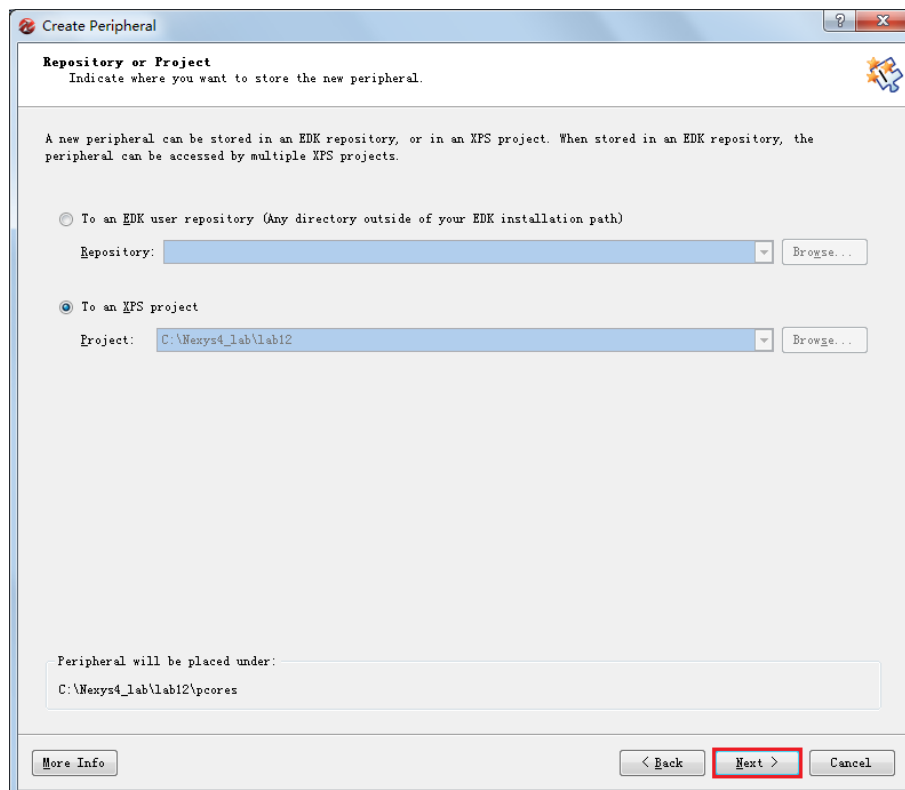
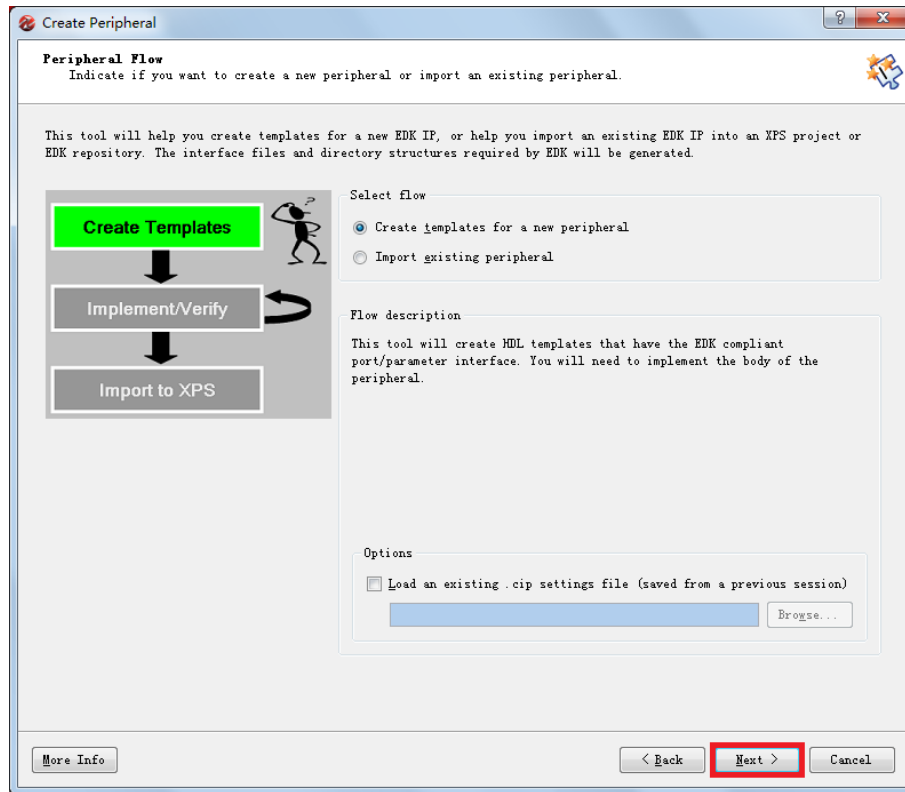
3-1-1. 选择 Hardware→Create or Import Peripheral...



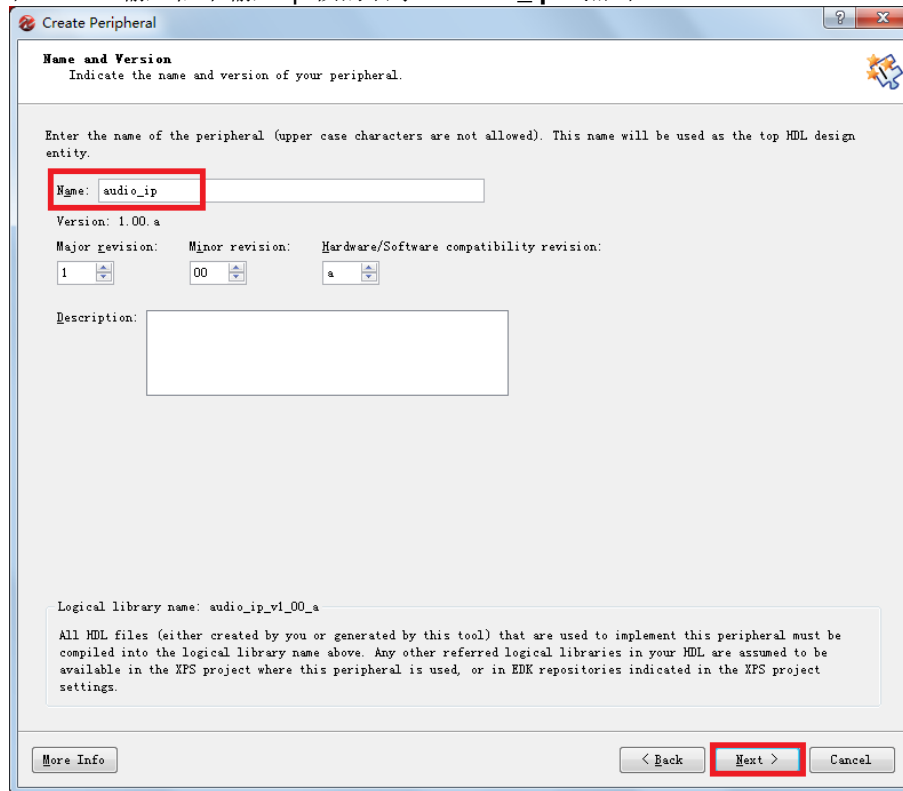
3-1-2. 弹出向导对话框，点击 Next



3-1-3. 一下两步均保持默认选项，点击 **Next**



3-1-4. 在 Name 输入框中输入 ip 核的名字: **audio_ip**, 点击 **Next**



Create Peripheral

Name and Version
Indicate the name and version of your peripheral.

Enter the name of the peripheral (upper case characters are not allowed). This name will be used as the top HDL design entity.

Name:

Version: 1.00.a

Major revision: Minor revision: Hardware/Software compatibility revision:

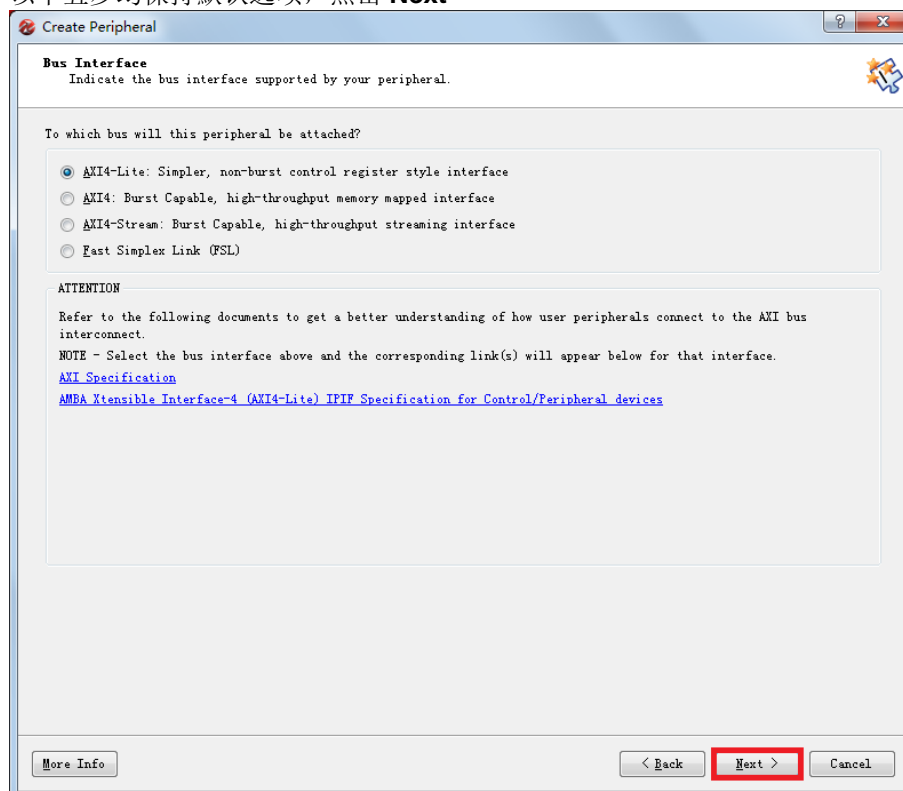
Description:

Logical library name: audio_ip_v1_00_a

All HDL files (either created by you or generated by this tool) that are used to implement this peripheral must be compiled into the logical library name above. Any other referred logical libraries in your HDL are assumed to be available in the XPS project where this peripheral is used, or in EDK repositories indicated in the XPS project settings.

[More Info](#) [< Back](#) [Next >](#) [Cancel](#)

3-1-5. 以下五步均保持默认选项, 点击 **Next**



Create Peripheral

Bus Interface
Indicate the bus interface supported by your peripheral.

To which bus will this peripheral be attached?

☒ AXI4-Lite: Simpler, non-burst control register style interface
☐ AXI4: Burst Capable, high-throughput memory mapped interface
☐ AXI4-Stream: Burst Capable, high-throughput streaming interface
☐ Fast Simplex Link (FSL)

ATTENTION
Refer to the following documents to get a better understanding of how user peripherals connect to the AXI bus interconnect.
NOTE - Select the bus interface above and the corresponding link(s) will appear below for that interface.
[AXI Specification](#)
[AMBA Xtensible Interface-4 \(AXI4-Lite\) IPIF Specification for Control/Peripheral devices](#)

[More Info](#) [< Back](#) [Next >](#) [Cancel](#)

Create Peripheral

IPIF (IP Interface) Services
Indicate the IPIF services required by your peripheral.

Your peripheral will be connected to the AXI4 interconnect through corresponding AXI IP Interface (IPIF) modules, which provide you with a quick way to implement the interface between the AXI4 interconnect and the user logic. Besides the standard functions like address decoding provided by the slave IPIF module, the wizard tool also offers other commonly used services and configurations to simplify the implementation of the design.

Master service and configuration

Typically required by complex peripherals like Ethernet and PCI for commanding data transfers between regions . Includes AXI4LITE master interface and AXI4LITE slave interface.

☐ User logic master support

Slave service and configuration

Typically required by most peripherals for operations like logic control, status report, data buffering, multiple memory/address space access, and etc. (AXI slave interface will always be included).

☐ Software reset ☒ User logic software register

☒ Include data phase timer

[More Info](#) < Back **Next >** Cancel

Create Peripheral

User S/W Register
Configure the software accessible registers in your peripheral.

The user specific software accessible registers will be implemented in the user-logic module of your peripheral. Such registers are typically provided for software programs to control and to monitor the status of your user logic. These registers are addressable on the byte, half-word, word, double word or quad word boundaries depending on your design. An example logic for register read/write will be included in the user-logic module generated by the wizard tool for your reference.

User logic software registers may take full advantage of the slave IPIF address-decoding service to generate CE decodes for all of the individual register of interest. The diagram on the left shows the simplest set of IPIF slave signals to read/write the registers.

Number of software accessible registers: (1 to 32)

[More Info](#) < Back **Next >** Cancel

Create Peripheral

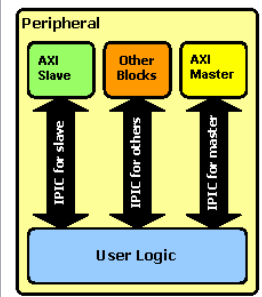
IP Interconnect (IPIC)
Select the interface between the logic to be implemented in your peripheral and the IPIF.

Your peripheral is connected to the bus through a suitable IPIF module. Your peripheral interfaces to the IPIF through a set of signals called the IP interconnect (IPIC) interface. Some of the ports are always present. You can choose to include the others based on the functionality required by your peripheral.

Note: all IPIC ports are active high.

Port description

Peripheral



☒ Bus2IP_Clk
☒ Bus2IP_Resetn
☐ Bus2IP_Addr
☐ Bus2IP_CS
☐ Bus2IP_RNW
☒ Bus2IP_Data
☒ Bus2IP_BE
☒ Bus2IP_RdCE
☒ Bus2IP_WrCE
☒ IP2Bus_Data
☒ IP2Bus_RdAck
☒ IP2Bus_WrAck
☒ IP2Bus_Error

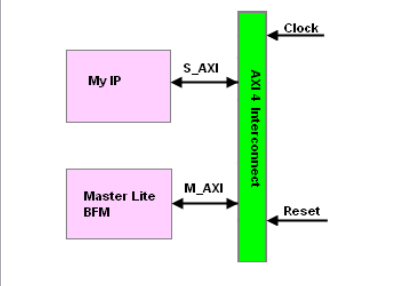
Restore Defaults

More Info < Back **Next >** Cancel

Create Peripheral

(OPTIONAL) Peripheral Simulation Support
Generate optional files for simulation using Bus Functional Models (BFM).

The EDK provides a BFM simulation platform to help you simulate your peripheral. Indicate if you want this tool to generate the appropriate HDL and stimulus file for the target bus.



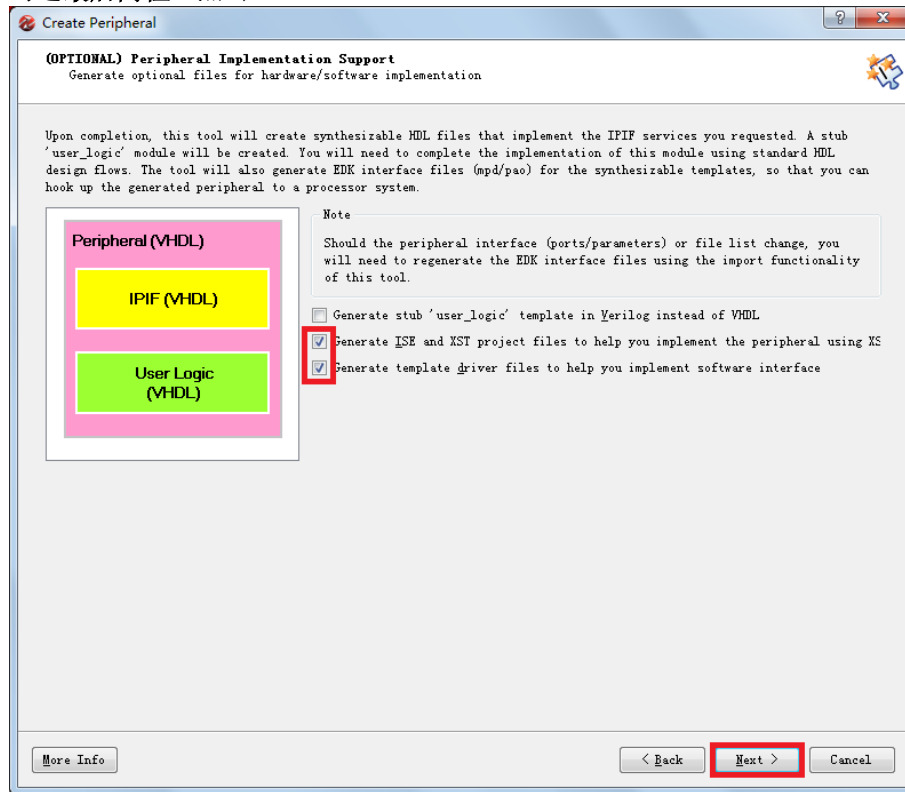
☐ Generate BFM simulation platform

- A testbench template will be generated on top of your peripheral.
- A test platform description file (bfm_system.mhs) consisting of the subsystem illustrated by the diagram will be generated as well.
- Stimulus for other non-CoreConnect bus I/Os of your peripheral can be defined in the testbench file.
- Please refer to the README file for BFM simulation instructions.

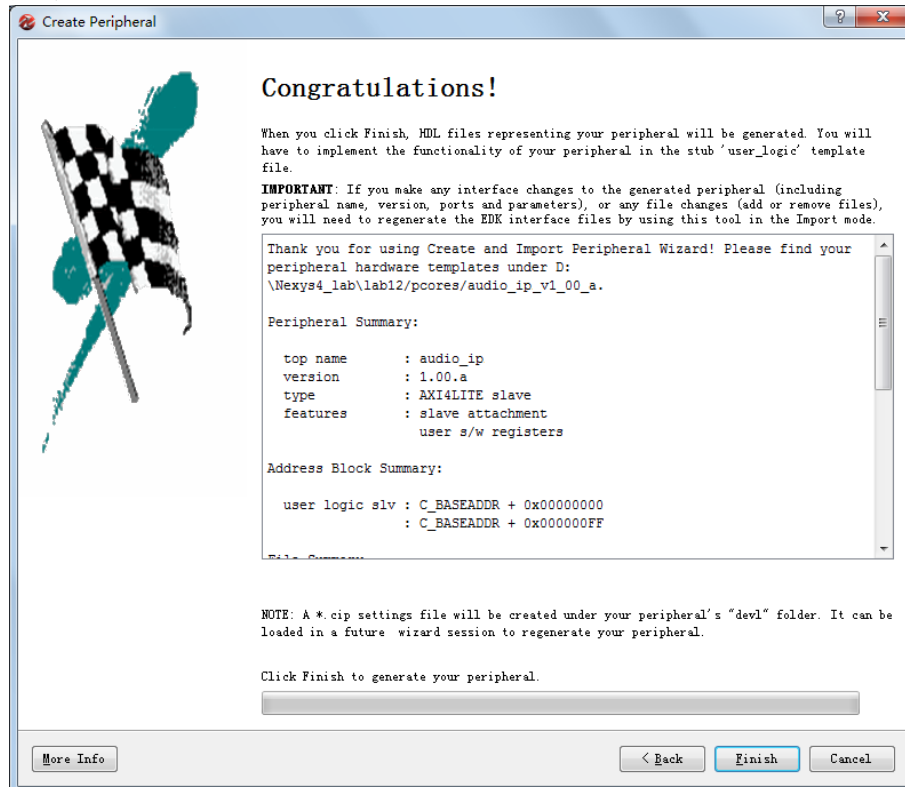
Note: License Required for BFM IPs in Simulation.

More Info < Back **Next >** Cancel

3-1-6. 勾选最后两栏，点击 Next

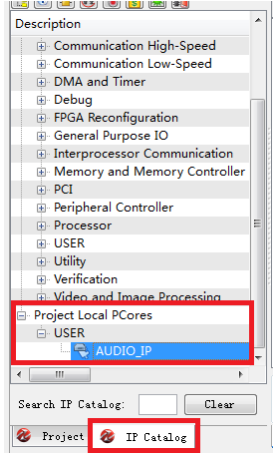


3-1-7. 点击 Finish

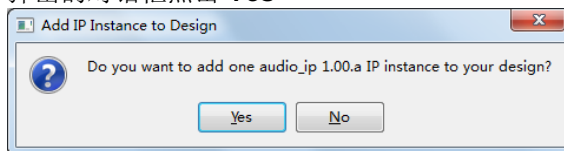


3-2. 添加自定义 IP 核

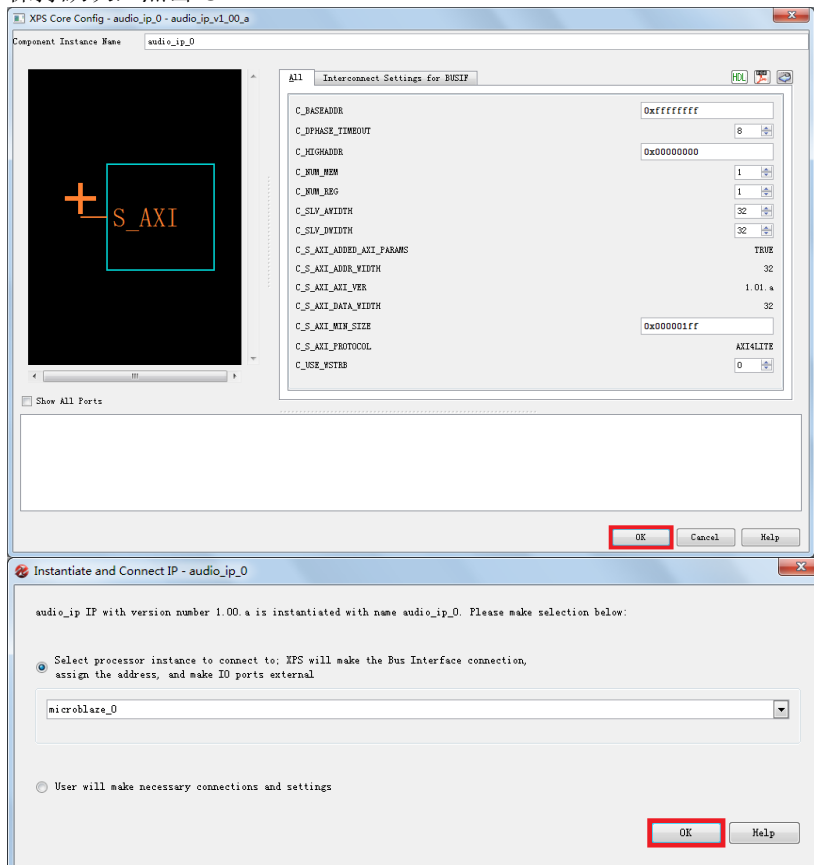
3-2-1. 点击 IP Catalog 选项卡，展开 Project Local PCores->USER，双击 AUDIO_IP



3-2-2. 弹出的对话框点击 Yes

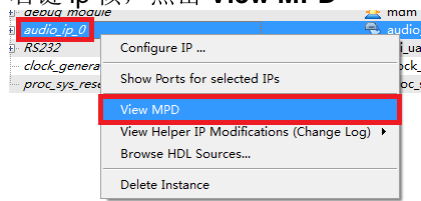


3-2-3. 保持默认，点击 OK

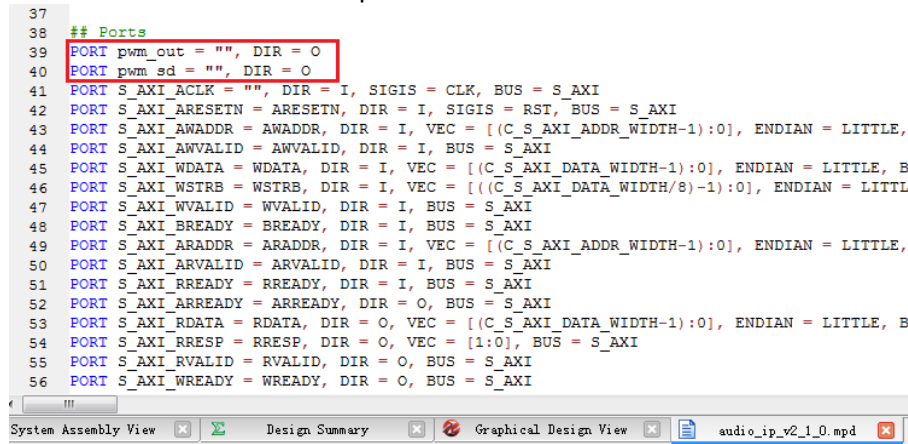


3-3. 修改 IP 核文件

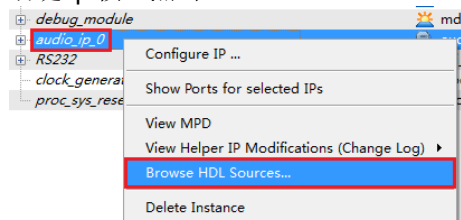
3-3-1. 右键 ip 核，点击 View MPD



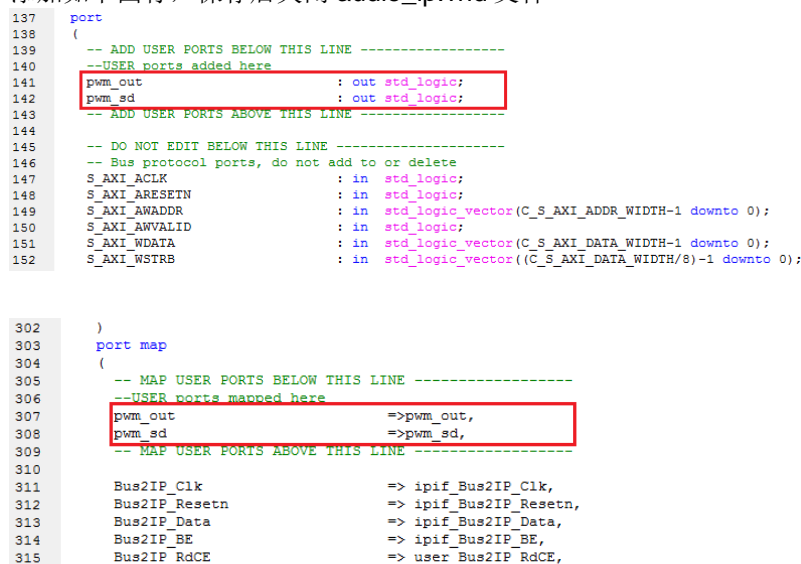
添加如下两行，保存后关闭 mpd 文件



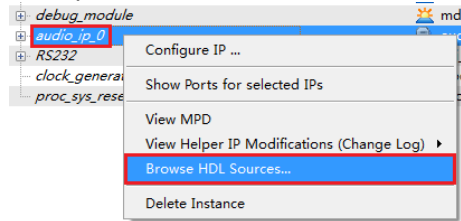
3-3-2. 右键 ip 核，点击 Browse HDL Sources..., 打开 audio_ip.vhd 文件



添加如下四行，保存后关闭 audio_ip.vhd 文件



3-3-3. 右键 ip 核，点击 **Browse HDL Sources...**，打开 **user_logic.vhd** 文件



添加如下代码，保存后关闭 **user_logic.vhd** 文件

```

97  port
98  (
99      -- ADD USER PORTS BELOW THIS LINE -----
100     --USER ports added here
101     pwm_out      : out std_logic;
102     pwm_sd       : out std_logic;
103     -- ADD USER PORTS ABOVE THIS LINE -----
104
105     -- DO NOT EDIT BELOW THIS LINE -----
106     -- Bus protocol ports, do not add to or delete
107     Bus2IP_Clk   : in  std_logic;
108     Bus2IP_Resetn : in  std_logic;
109     Bus2IP_Data  : in  std_logic_vector(C_SLV_DWIDTH-1 downto 0);
110     Bus2IP_BE    : in  std_logic_vector(C_SLV_DWIDTH/8-1 downto 0);
143  signal slv_read_ack      : std_logic;
144  signal slv_write_ack    : std_logic;
145
146  component top is port
147  (
148      clk : in  STD_LOGIC;
149      tone : in  STD_LOGIC_VECTOR (4 downto 0);
150      mode : in  STD_LOGIC_VECTOR (2 downto 0);
151      pwm_out : out STD_LOGIC;
152      pwm_sd : out STD_LOGIC
153  );
154  end component ;
155
156  begin
157
158     --USER logic implementation added here
159     u_top : component top
160     port map
161     (
162         clk => Bus2IP_Clk,
163         tone => slv_reg0(4 downto 0),
164         mode => slv_reg0(7 downto 5),
165         pwm_out => pwm_out,
166         pwm_sd => pwm_sd
167     );
168
169     -- Example code to read/write user logic slave mode

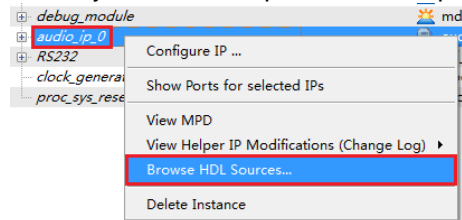
```

3-3-4. 将本实验 source 目录下的 top.vhd、SimpleSynth.vhd 和 PWM.vhd 拷贝到 C:\Nexys4_lab\lab12\pcores\audio_ip_v1_00_a\hdl\vhdl 目录下



3-3-5. 右键 ip 核，点击 **Browse HDL Sources...**，到

C:\Nexys4_lab\lab12\pcores\audio_ip_v1_00_a\data 目录下打开 **_audio_ip_xst.prj** 文件

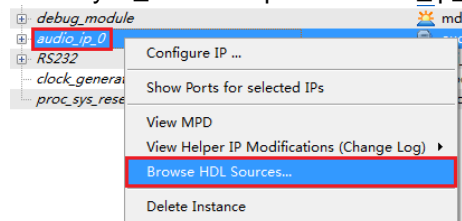


添加如下三行，保存后关闭 **prj** 文件

```
10 vhdl axi_lite_ipif_v1_01_a "D:\Xilinx\14.3\ISE_DS\EDK\hw\Xilin:
11 vhdl audio_ip_v1_00_a "../hdl/vhdl/user_logic.vhd"
12 vhdl audio_ip_v1_00_a "../hdl/vhdl/audio_ip.vhd"
13 vhdl audio_ip_v1_00_a "../hdl/vhdl/top.vhd"
14 vhdl audio_ip_v1_00_a "../hdl/vhdl/SimpleSynth.vhd"
15 vhdl audio_ip_v1_00_a "../hdl/vhdl/PWM.vhd"
```

3-3-6. 右键 ip 核，点击 **Browse HDL Sources...**，到

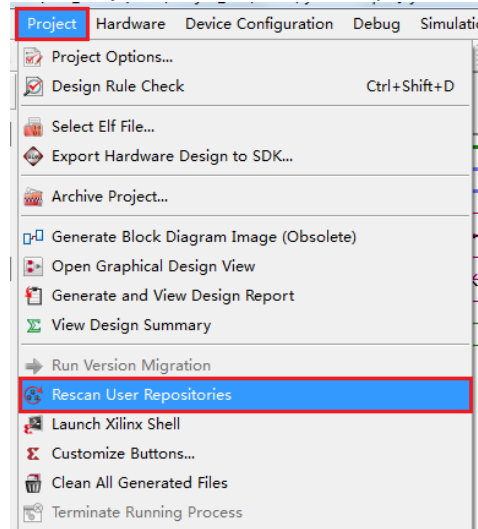
C:\Nexys4_lab\lab12\pcores\audio_ip_v1_00_a\data 目录下打开 **audio_ip_v2_1_0.pao** 文件



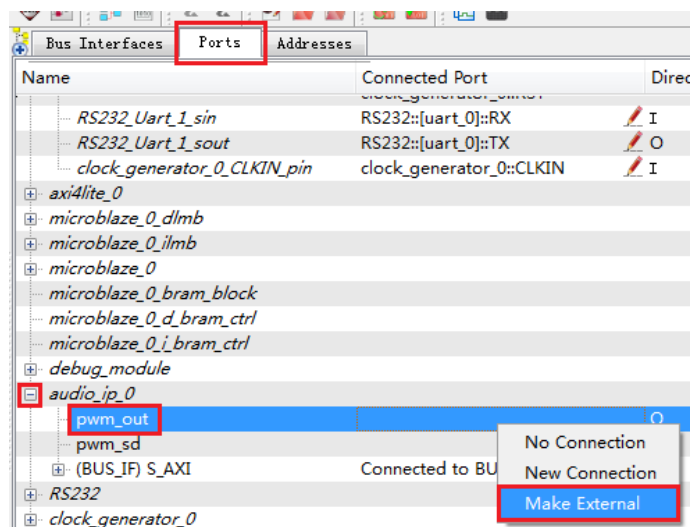
添加如下三行，保存后关闭 **pao** 文件

```
8 lib axi_lite_ipif_v1_01_a all
9 lib audio_ip_v1_00_a user_logic vhd
10 lib audio_ip_v1_00_a audio_ip vhd
11 lib audio_ip_v1_00_a top vhd
12 lib audio_ip_v1_00_a SimpleSynth vhd
13 lib audio_ip_v1_00_a PWM vhd
```

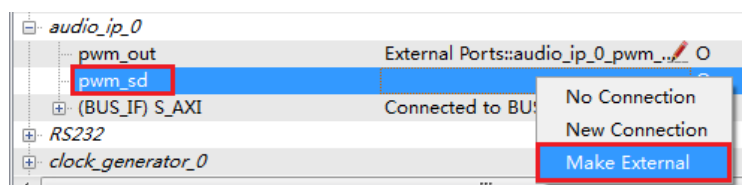
3-3-7. 点击 **Project->Rescan User Repositories**



3-3-8. 点击 Ports 选项卡，展开 ip 核，右键 pwm_out，点击 Make External



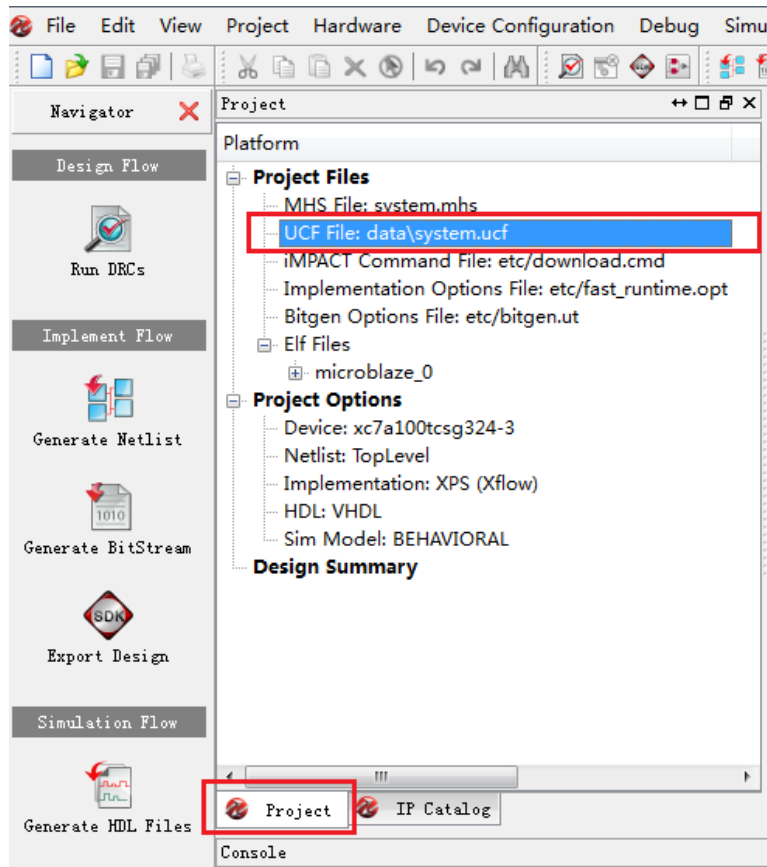
右键 pwm_sd，点击 Make External



第四步 导出工程到 SDK

4-1. 添加用户约束文件

4-1-1. 点击 **Project** 选项卡，展开，双击 UCF File:Data\system.ucf，ucf 文件在右侧打开



4-1-2. 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%;

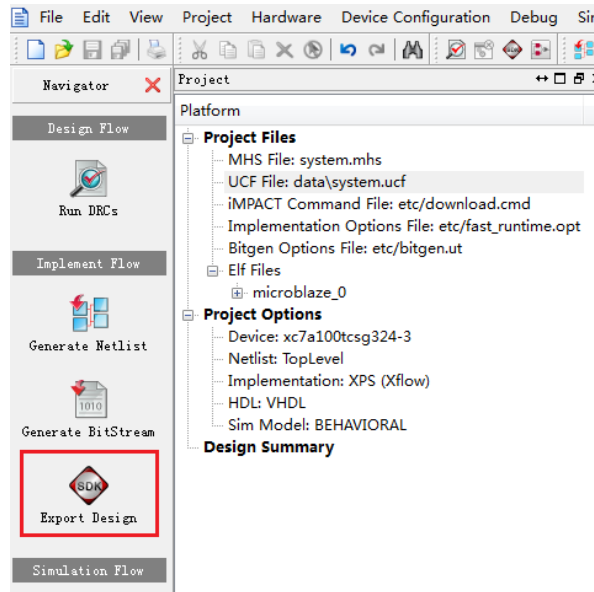
# Switches
NET "RESET" LOC = "U9" | IOSTANDARD = "LVCMOS33";

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

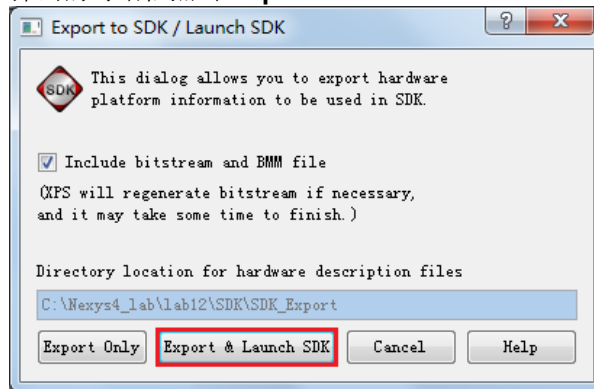
# PWM Audio Amplifier
NET "audio_ip_0_pwm_out_pin" LOC = "A11" | IOSTANDARD = "LVCMOS33";
NET "audio_ip_0_pwm_sd_pin" LOC = "D12" | IOSTANDARD = "LVCMOS33";
```

4-2. 将工程导出到 SDK

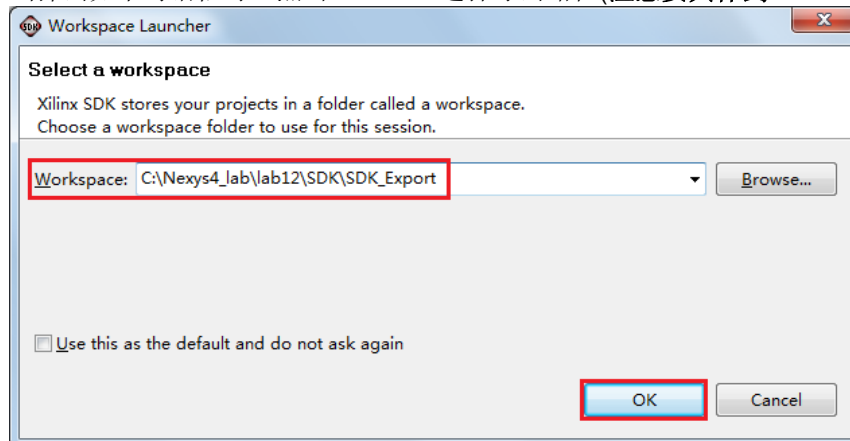
4-2-1. 点击页面左侧的 **Export Design** 按钮。



弹出的对话框点击 **Export & Launch SDK**。



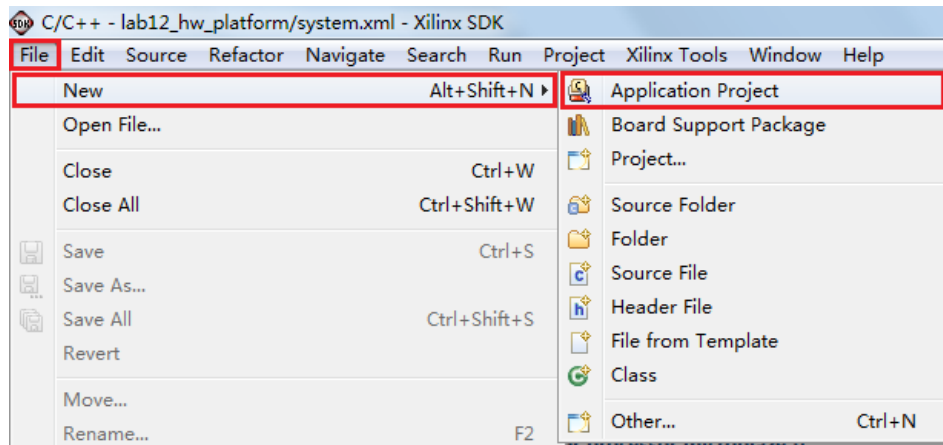
4-2-2. 当弹出如下对话框时，点击 **Browse** 选择导出路径(注意要具体到..\SDK\SDK_Export)，点击 **OK**



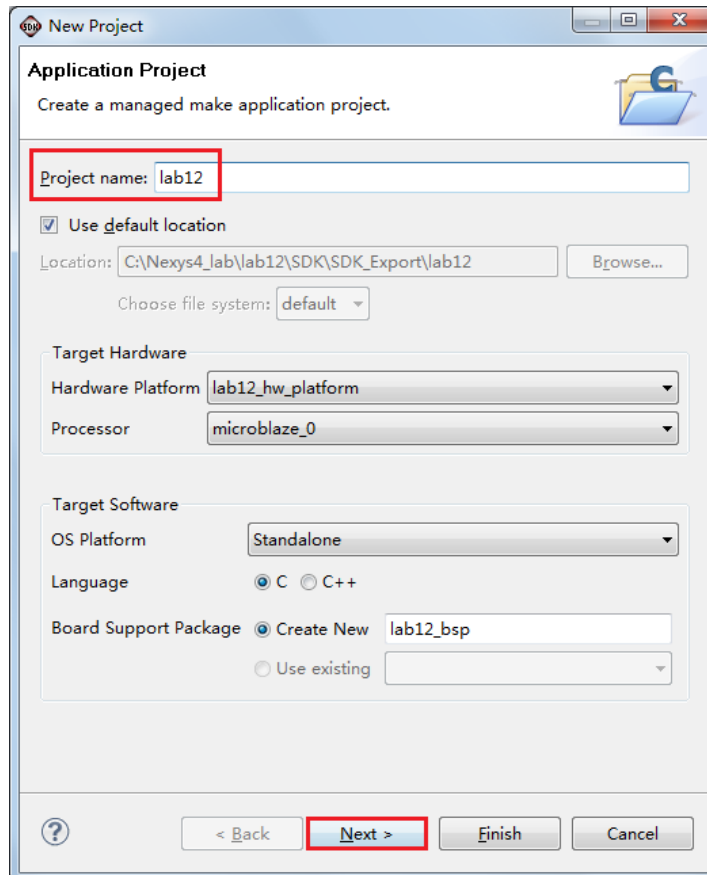
第五步 添加 APP

5-1. 新建工程及板级支持包

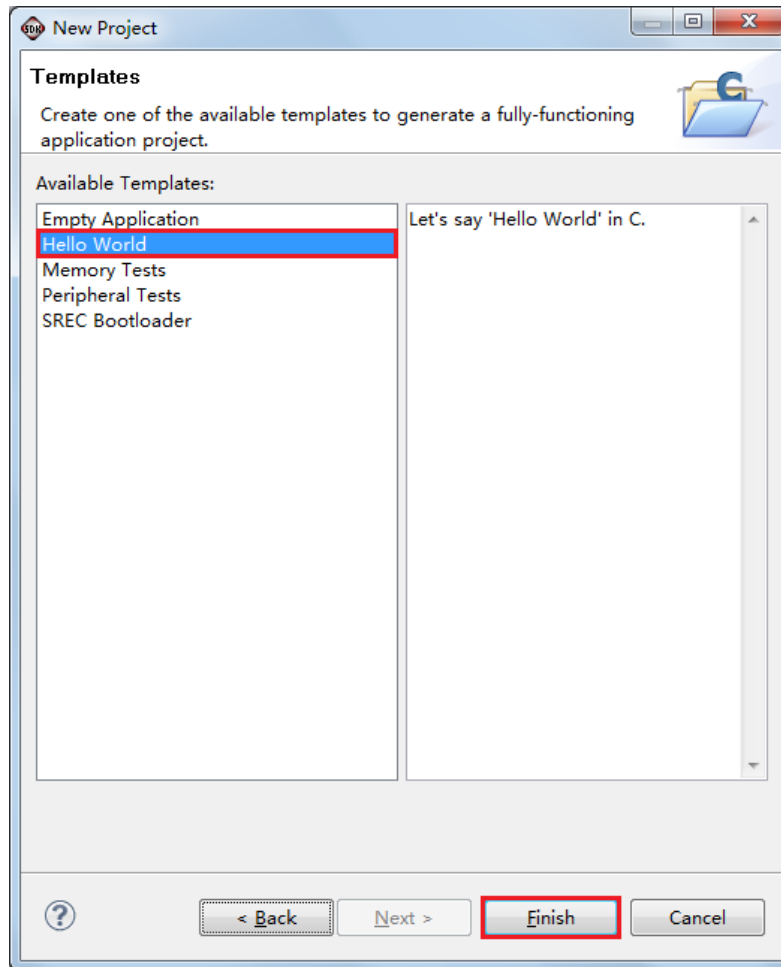
5-1-1. 点击 File→New→Application Project



5-1-2. 填写工程名如 lab12（注意不要有中文和空格），其它选项保持默认，点击 Next



5-1-3. 选择 Hello World，点击 Finish



5-2. 配置 IP 核

- 5-2-1. 打开 C:\Nexys4_lab\lab12\drivers\audio_ip_v1_00_a\src 目录下的 **audio_ip_selftest.c** 文件，修改以下两处，保存后关闭。

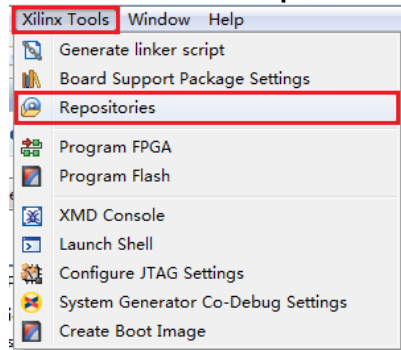
```
#include "audio_ip.h"
#include "stdio.h"
#include "xil_io.h"
#include "xparameters.h"

/***** Constant De

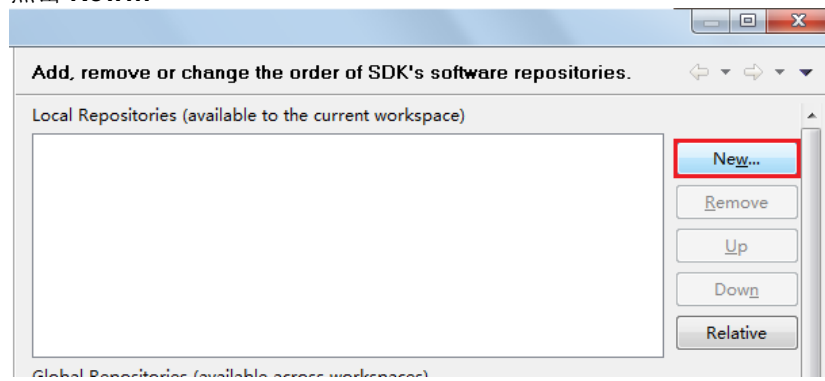
#define READ WRITE MUL FACTOR 0x10
#define AUDIO_IP_USER_NUM_REG 0x01

/***** Variable De
```

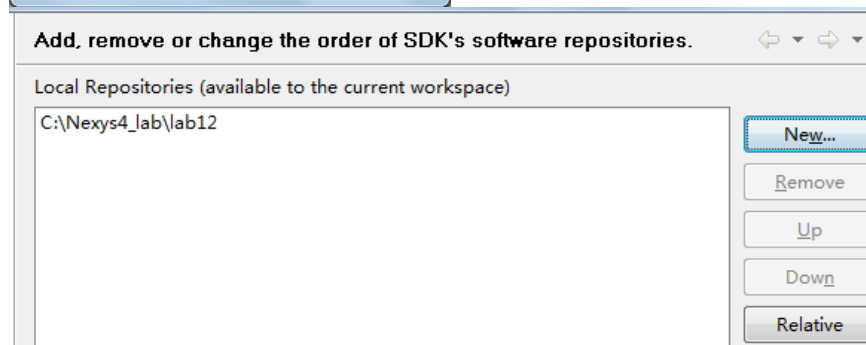
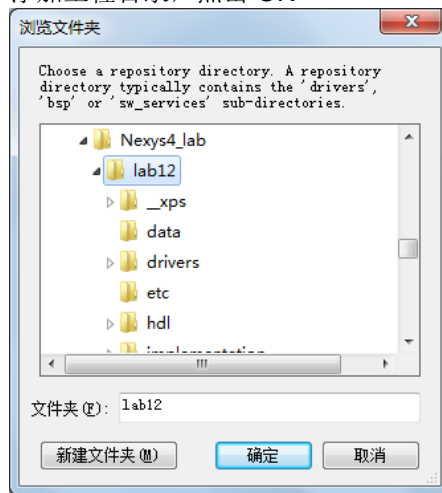
5-2-2. 点击 Xilinx Tools->Repositories



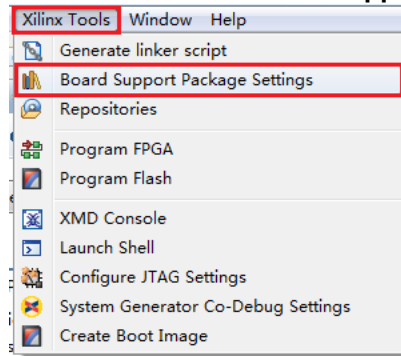
点击 New...



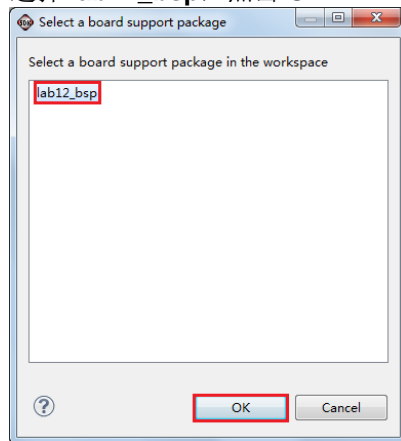
添加工程目录，点击 OK



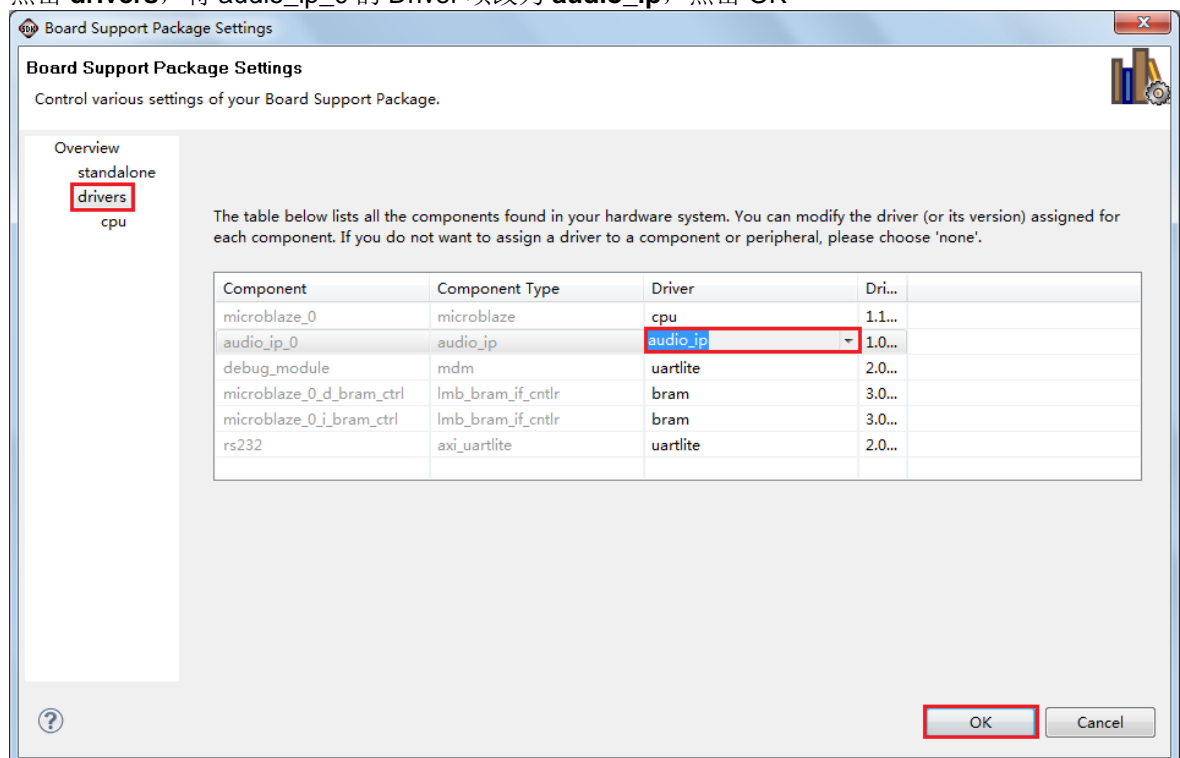
5-2-3. 点击 **Xilinx Tools**→**Board Support Package Settings**



选择 **lab12_bsp**, 点击 **OK**

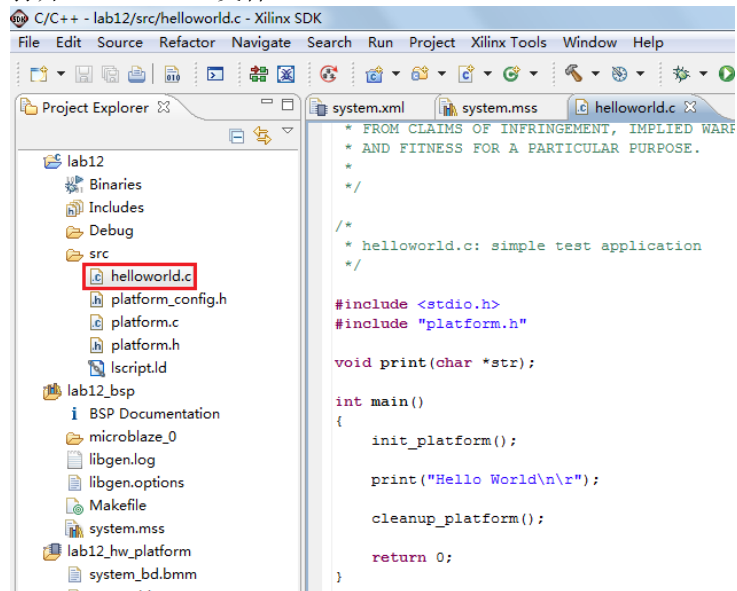


点击 **drivers**, 将 **audio_ip_0** 的 Driver 项改为 **audio_ip**, 点击 **OK**



5-3. 修改 C 语言程序

5-3-1. 打开 helloworld.c 文件



5-3-2. 修改 helloworld.c 文件如下，保存

```
#include <stdio.h>
#include "platform.h"
#include "xparameters.h"
#include "audio_ip.h"

void print(char *str);

int main()
{
    init_platform();
    char *tone="";
    int i;
    print("You can press the key from 1 to 8\r\n");
    while(1)
    {
        *tone = inbyte();
        print(tone);
        print("\r\n");
        switch (*tone)
        {
            case '1': AUDIO_IP_mWriteSlaveReg0(XPAR_AUDIO_IP_0_BASEADDR, 0, 0b00111); break;
            case '2': AUDIO_IP_mWriteSlaveReg0(XPAR_AUDIO_IP_0_BASEADDR, 0, 0b01001); break;
            case '3': AUDIO_IP_mWriteSlaveReg0(XPAR_AUDIO_IP_0_BASEADDR, 0, 0b01011); break;
            case '4': AUDIO_IP_mWriteSlaveReg0(XPAR_AUDIO_IP_0_BASEADDR, 0, 0b01100); break;
            case '5': AUDIO_IP_mWriteSlaveReg0(XPAR_AUDIO_IP_0_BASEADDR, 0, 0b01110); break;
            case '6': AUDIO_IP_mWriteSlaveReg0(XPAR_AUDIO_IP_0_BASEADDR, 0, 0b10000); break;
            case '7': AUDIO_IP_mWriteSlaveReg0(XPAR_AUDIO_IP_0_BASEADDR, 0, 0b10010); break;
            case '8': AUDIO_IP_mWriteSlaveReg0(XPAR_AUDIO_IP_0_BASEADDR, 0, 0b10011); break;
            default : AUDIO_IP_mWriteSlaveReg0(XPAR_AUDIO_IP_0_BASEADDR, 0, 0); break;
        }
        for (i=0; i<1000000; i++){
            AUDIO_IP_mWriteSlaveReg0(XPAR_AUDIO_IP_0_BASEADDR, 0, 0);
        }
        cleanup_platform();
        return 0;
    }
}
```

其中 tone 通过串口接收了用户的按键，按键 1 到 8 分别对应 8 种音符。通过 switch case 语句判断出用户的按键之后将相应音符的发音值写入 ip 核寄存器中。for 循环用来延时，当向寄存器中写入 0 时不发声。

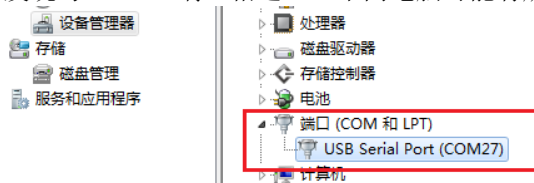
第六步 上板验证

6-1. 将 Nexys4 与 PC 的 USB 接口连接

6-2. 查看端口号

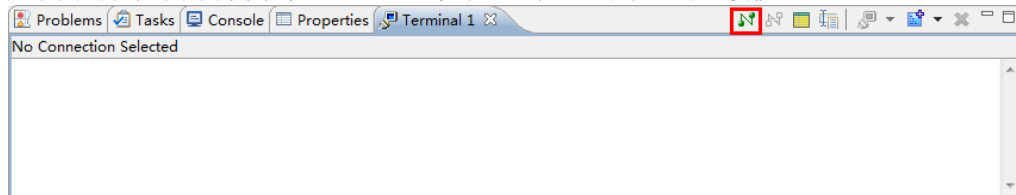
右键“我的电脑” — “属性” — 在页面左侧选择“设备管理器”

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）

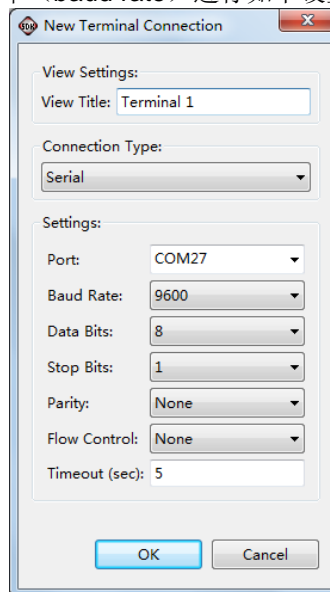


6-3. 在 SDK 中打开串口：

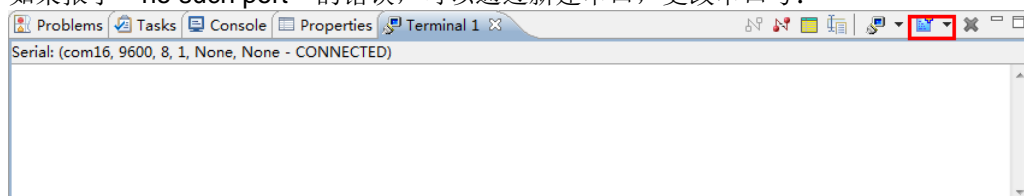
6-3-1. 在下面的在页面下方找到 terminal 选项卡，然后点击绿色的连接按钮。



6-3-2. 按照端口号和 XPS 中的波特率（baud rate）进行如下设置：

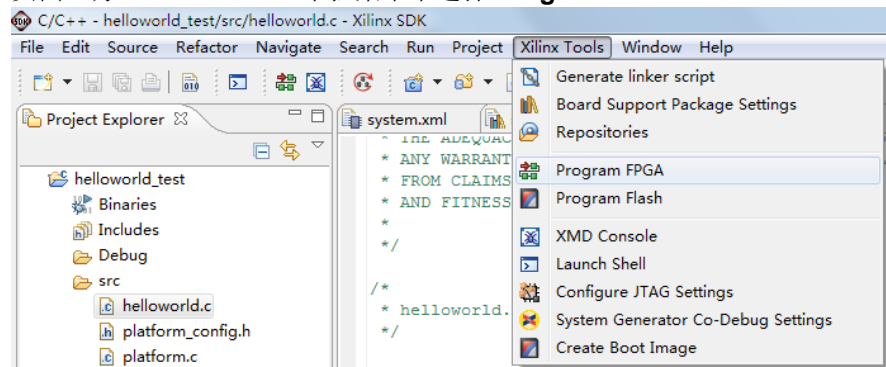


如果报了“no such port”的错误，可以通过新建串口，更改串口号：

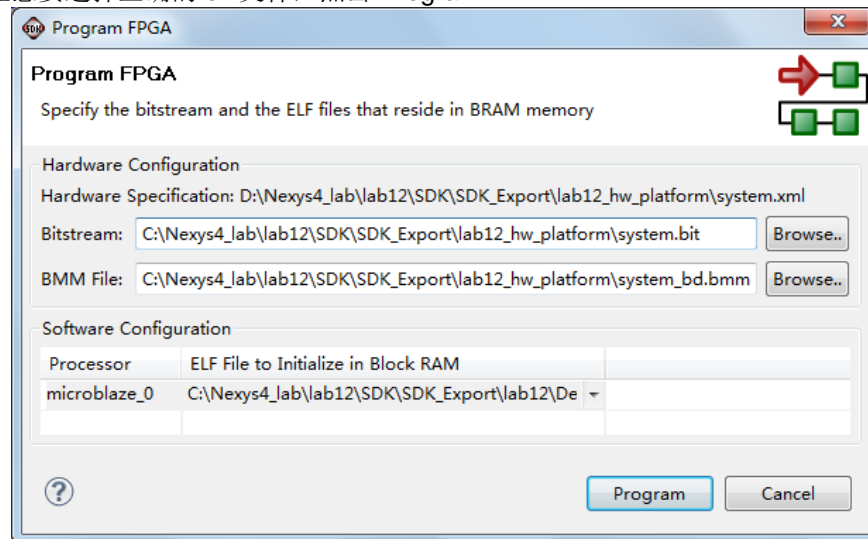


6-4. 将程序下载到板子上并运行

6-4-1. 在页面上方，Xilinx Tools 下拉菜单中选择 Program FPGA



6-4-2. 注意要选择正确的 elf 文件，点击 Program



6-5. 将耳机接到板卡右上角的 MONO AUDIO OUT 接口上



键盘上 1 到 8 的按键分别对应 “do re mi fa so la si do” 8 个音符。
例如按下 1 后将在串口显示 1 并从耳机中听到 “do” 的声音。

