



计算机组成原理与接口技术（实验） ——基于MIPS架构

Sep, 2020

并行IO接口实验（10学时，11-13周）

杨明
华中科技大学电子信息与通信学院
myang@hust.edu.cn



► 实验内容

- 目的
- 任务及时间安排
- 报告要求

► 原理回顾

- Nexys4实验板简介
- Nexys4怎么用？
- Xilinx的GPIO和INTC
- GPIO硬件设计
- GPIO应用软件设计
- 系统功能测试

- ▶ 掌握GPIO IP核的工作原理和使用方法
- ▶ 掌握中断控制方式的IO接口设计原理
- ▶ 掌握中断程序设计方法
- ▶ 掌握IO接口程序控制方法
 - 查询方式
 - 中断方式
 - 延时方式

► 任务

• 并行IO

- 所有实验任务要求分别采用程序控制方式、普通中断方式、快速中断方式实现，中断方式时，GPIO输入、延时都采用中断实现。每位同学仅完成其中的一个任务，且需完成的实验任务编号与学号后三位模3的取值一致。
- [实验任务0]
 - 嵌入式计算机系统将独立按键以及独立开关作为输入设备，LED 灯、七段数码管作为输出设备。LED 灯实时显示独立开关对应位状态，同时8 个七段数码管实时显示最近按下的独立按键位置编码字符（C,U,L,D,R）。
 - 程序控制方式提示：程序以七段数码管动态显示控制循环为主体，在循环体内的延时函数内读取开关值更新LED、读取按键值更新段码。

► 任务

• 并行IO

▪ [实验任务1]

– 嵌入式计算机系统将独立按键以及独立开关作为输入设备，LED 灯作为输出设备。修改实验示例程序代码，实现以下功能：

- ▶ 1) 按下BTNC 按键时，计算机读入一组16 位独立开关状态作为第一个输入的二进制数据，并即时显示输入的二进制数到16 位LED 灯上。（没有按下BTNC按键时，开关拨动不读入数据）
- ▶ 2) 按下BTNR 按键时，计算机读入另一组16 位独立开关状态作为第二个输入的二进制数据，并即时显示输入的二进制数到16 位LED 灯上。（没有按下BTNR按键时，开关拨动不读入数据）
- ▶ 3) 按下BTNU 按键时，将保存的2 组二进制数据做无符号加法运算，并将运算结果输出到LED 灯对应位。
- ▶ 4) 按下BTND 按键时，将保存的2 组二进制数据做无符号乘法运算，并将运算结果输出到LED 灯对应位。
- ▶ 程序控制方式提示：循环读取按键键值，根据按键的值读取开关状态，并做相应处理。

► 任务

• 并行IO

▪ [实验任务2]

– 嵌入式计算机系统将独立按键以及独立开关作为输入设备，七段数码管作为输出设备。实现以下功能：

- ▶ 1) 按下BTNC 按键时，计算机读入一组16 位独立开关状态作为一个二进制数据，并将该二进制数的低8 位对应的二进制数值0 或1 显示到8 个七段数码管上。
- ▶ 2) 按下BTNU 按键时，计算机读入一组16 位独立开关状态作为一个二进制数据，并将该16 进制数据各位数字对应的字符0~F 显示到低4 位七段数码管上（高4 位七段数码管不显示）。
- ▶ 3) 按下BTND 按键时，计算机读入一组16 位独立开关状态作为一个二进制数据，并将该数据表示的无符号十进制数各位数字对应的字符0~9 显示到低5 位七段数码管上（高3 位七段数码管不显示）。
- ▶ 程序控制方式提示：程序以七段数码管动态显示控制循环为主体，在循环体内的延时函数内循环读取按键键值以及开关状态，并根据按键值做相应处理。



- ▶ 实验任务
- ▶ 硬件电路框图
- ▶ 硬件实现步骤
- ▶ 查询方式、中断方式、延时方式
- ▶ 软件流程图
- ▶ 软件源代码加注释
- ▶ 心得体会

► 实验内容

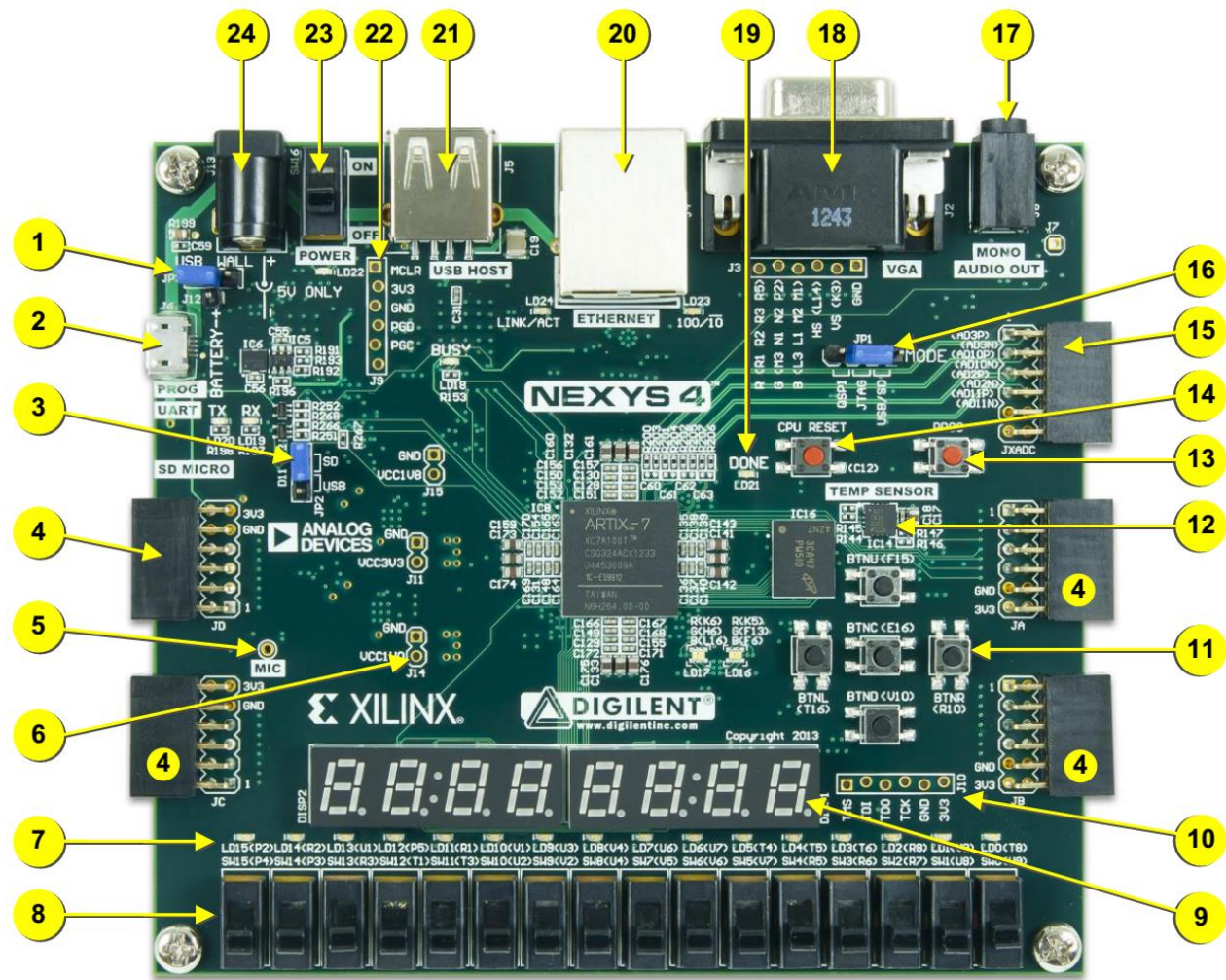
- 目的
- 任务及时间安排
- 报告要求

► 原理回顾

- Nexys4实验板简介
- Nexys4怎么用？
- Xilinx的GPIO和INTC
- GPIO硬件设计
- GPIO应用软件设计
- 系统功能测试

Nexys4实验板简介

外观

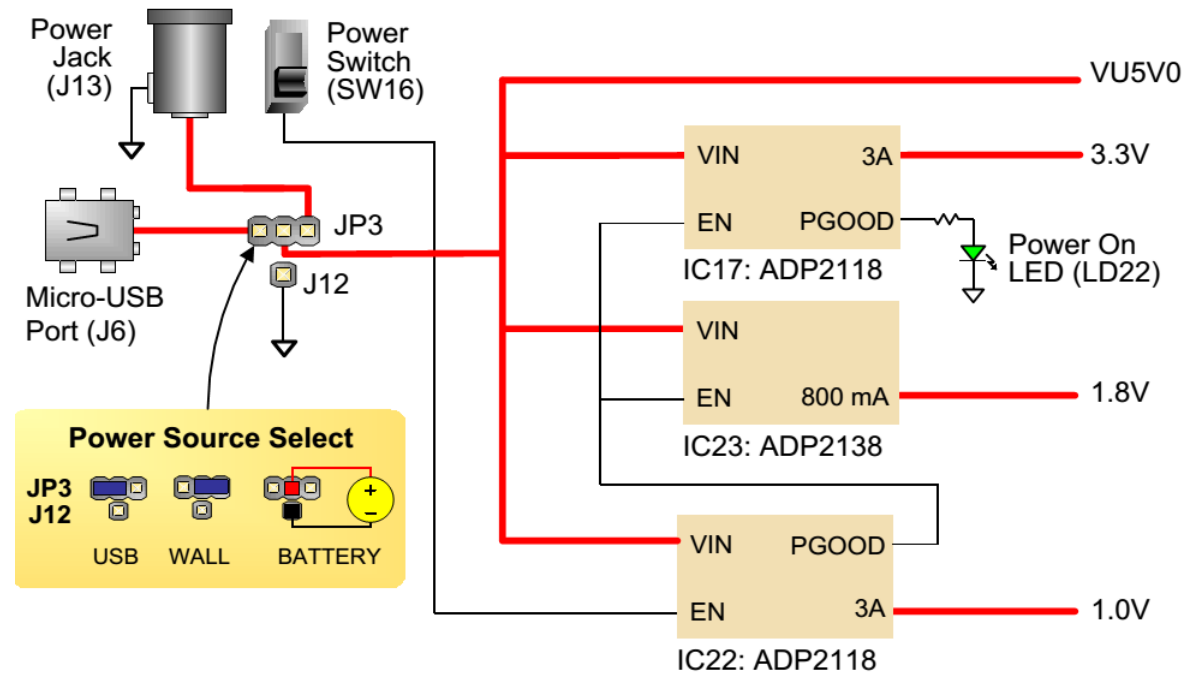


The Nexys4 board is a complete, ready-to-use digital circuit development platform based on the latest Artix-7 Field Programmable Gate Array (FPGA) from Xilinx. With its large, high-capacity FPGA (Xilinx part number XC7A100T-1CSG324C), generous external memories, and collection of USB, Ethernet, and other ports, the Nexys4 can host designs ranging from introductory combinational circuits to powerful embedded processors. Several built-in peripherals, including an accelerometer, temperature sensor, MEMs digital microphone, a speaker amplifier, and a lot of I/O devices allow the Nexys4 to be used for a wide range of designs without needing any other components.

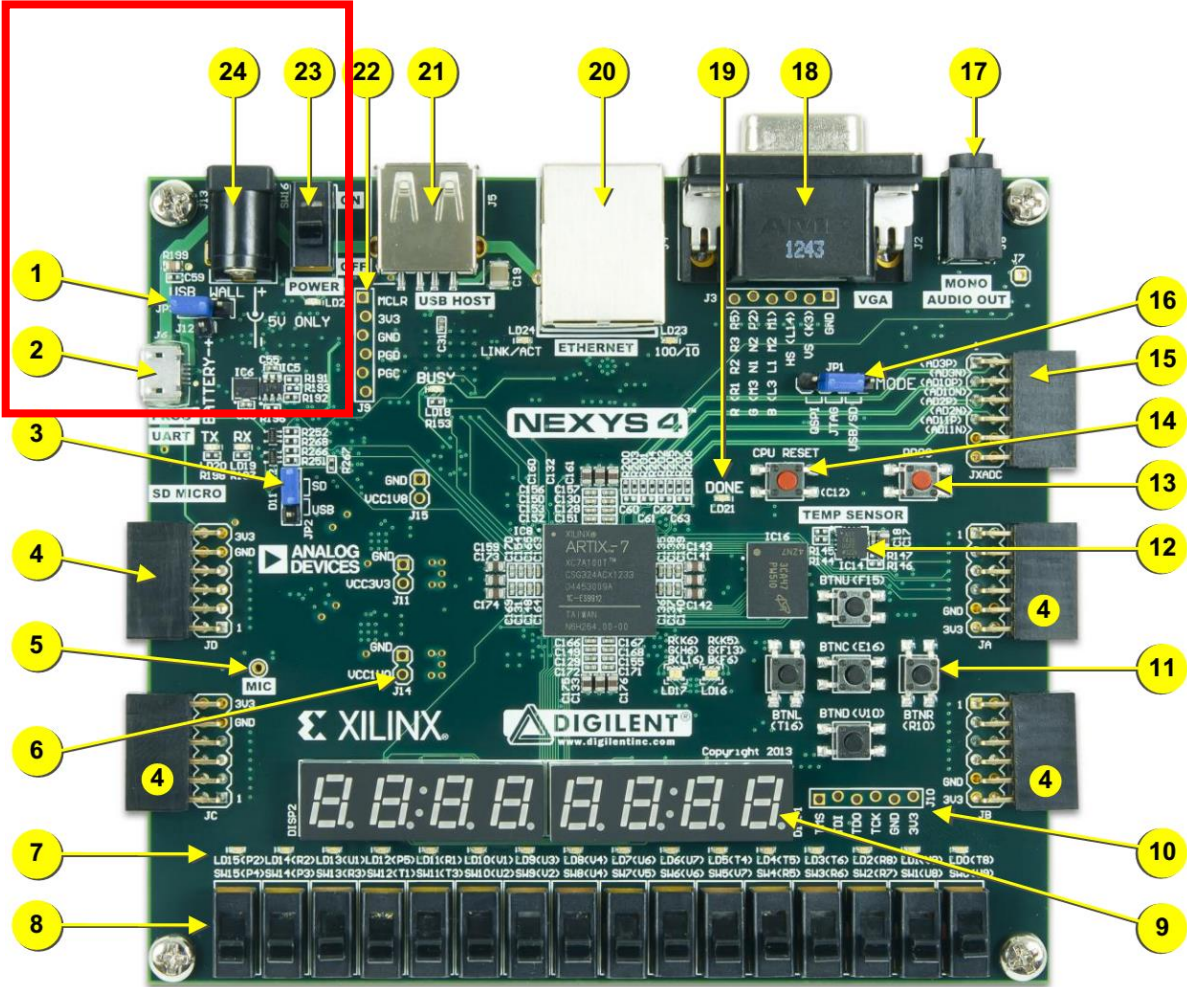
Callout	Component Description	Callout	Component Description
1	Power select jumper and battery header	13	FPGA configuration reset button
2	Shared UART/ JTAG USB port	14	CPU reset button (for soft cores)
3	External configuration jumper (SD / USB)	15	Analog signal Pmod connector (XADC)
4	Pmod connector(s)	16	Programming mode jumper
5	Microphone	17	Audio connector
6	Power supply test point(s)	18	VGA connector
7	LEDs (16)	19	FPGA programming done LED
8	Slide switches	20	Ethernet connector
9	Eight digit 7-seg display	21	USB host connector
10	JTAG port for (optional) external cable	22	PIC24 programming port (factory use)
11	Five pushbuttons	23	Power switch
12	Temperature sensor	24	Power jack



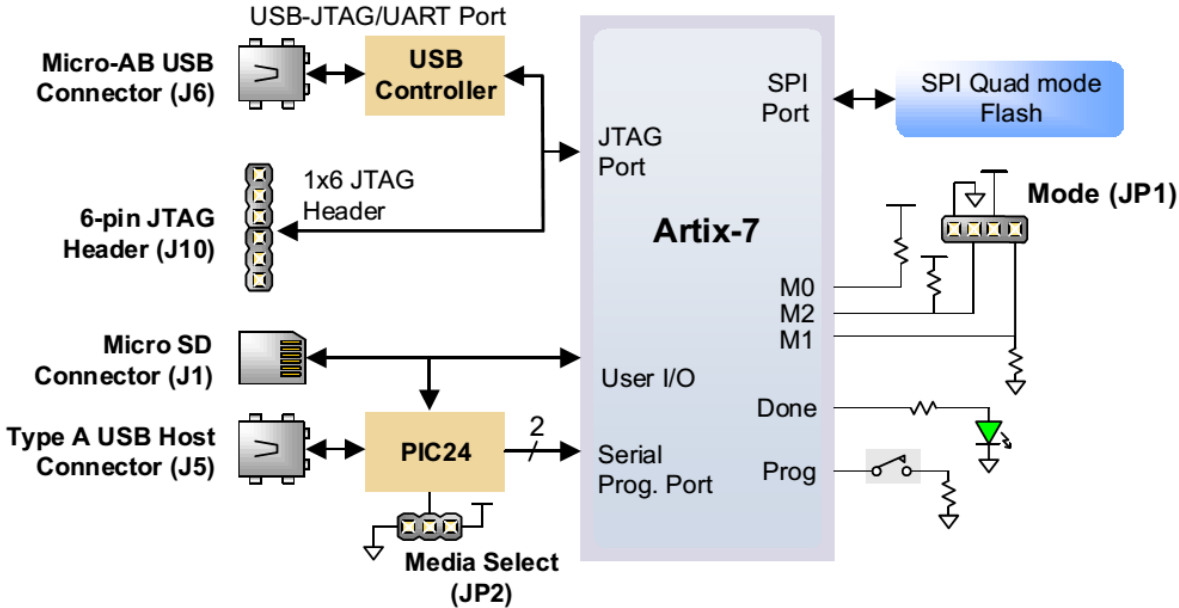
电源供电



- [1] JP3&J12缺省设置是按照USB供电
- [2] J6即作为USB供电，也是Jtag调试接口，还是USB-RS232的接口。



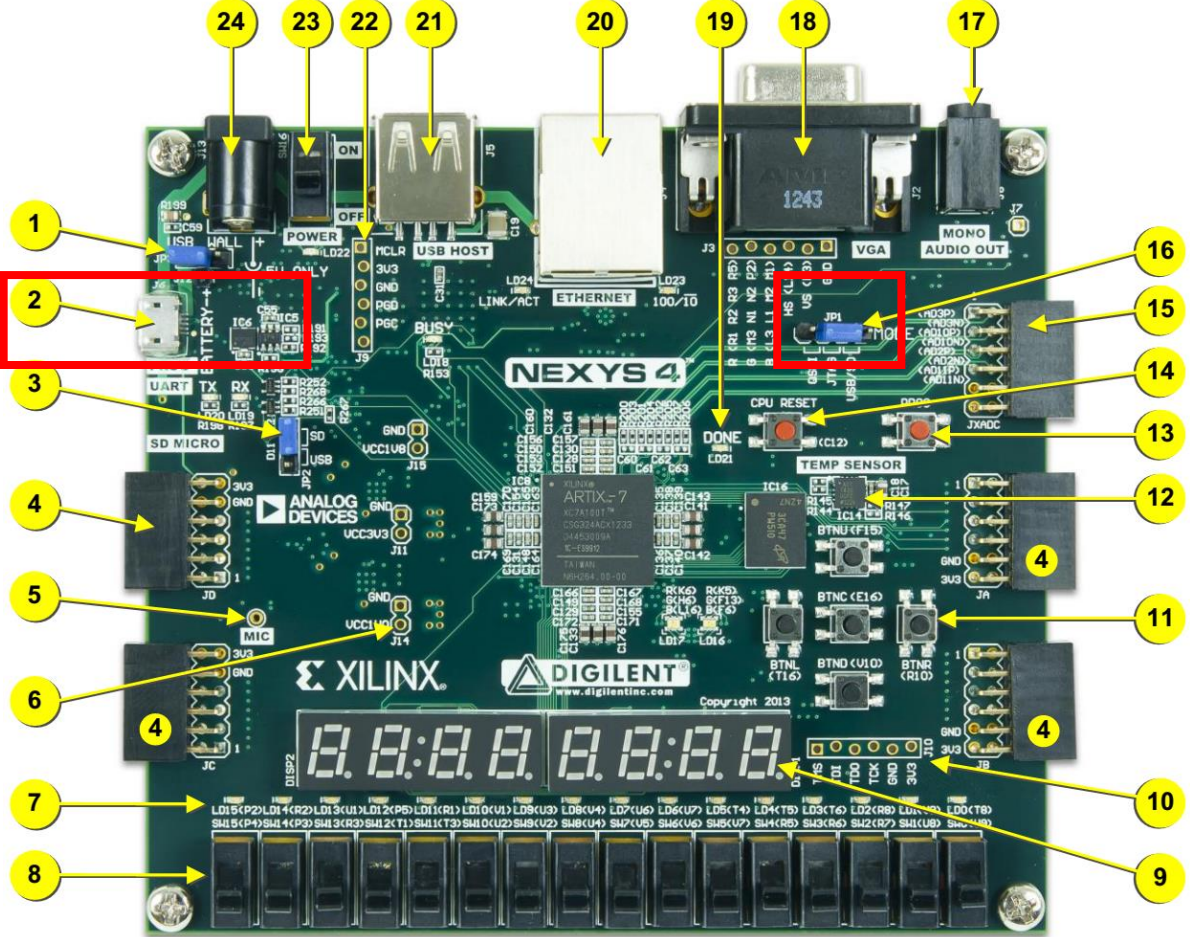
► FPGA配置



[1] JP2&JP1缺省设置是按照SPI Flash

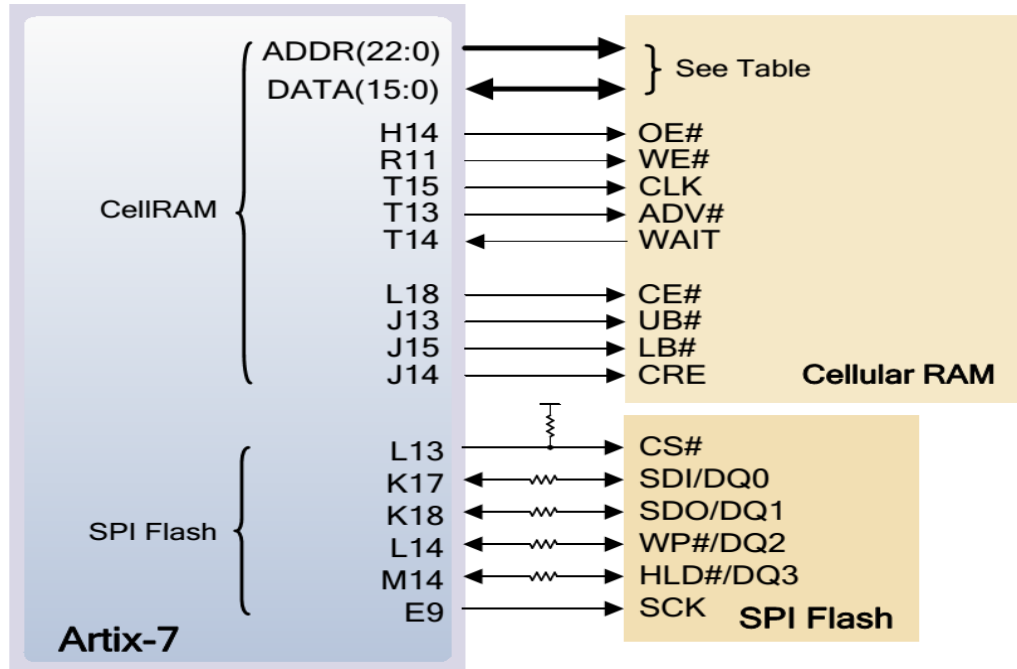
JP2	JP1	
NA		SPI Flash
NA		JTAG
		USB
		MicroSD

Programming Mode

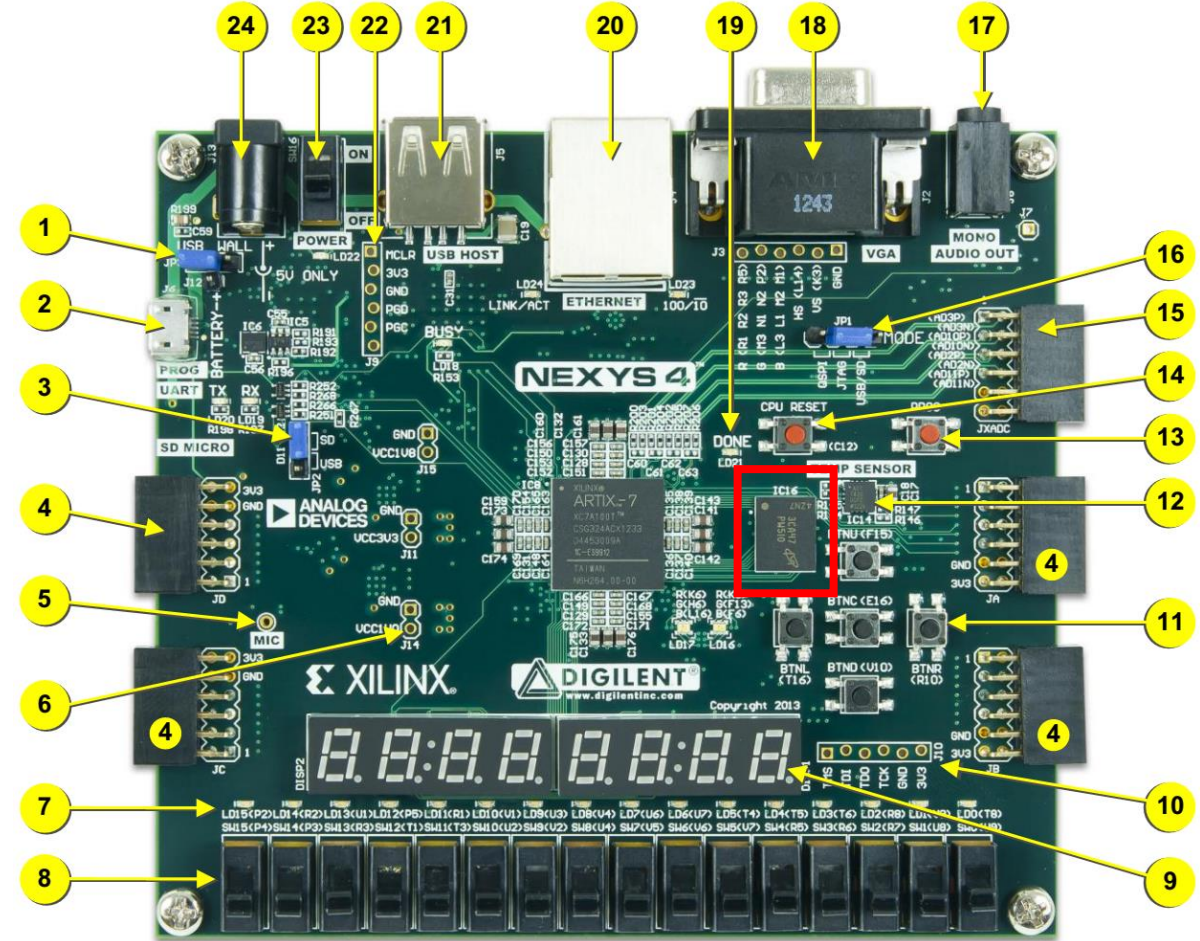


Nexys4实验板简介

► External Memories

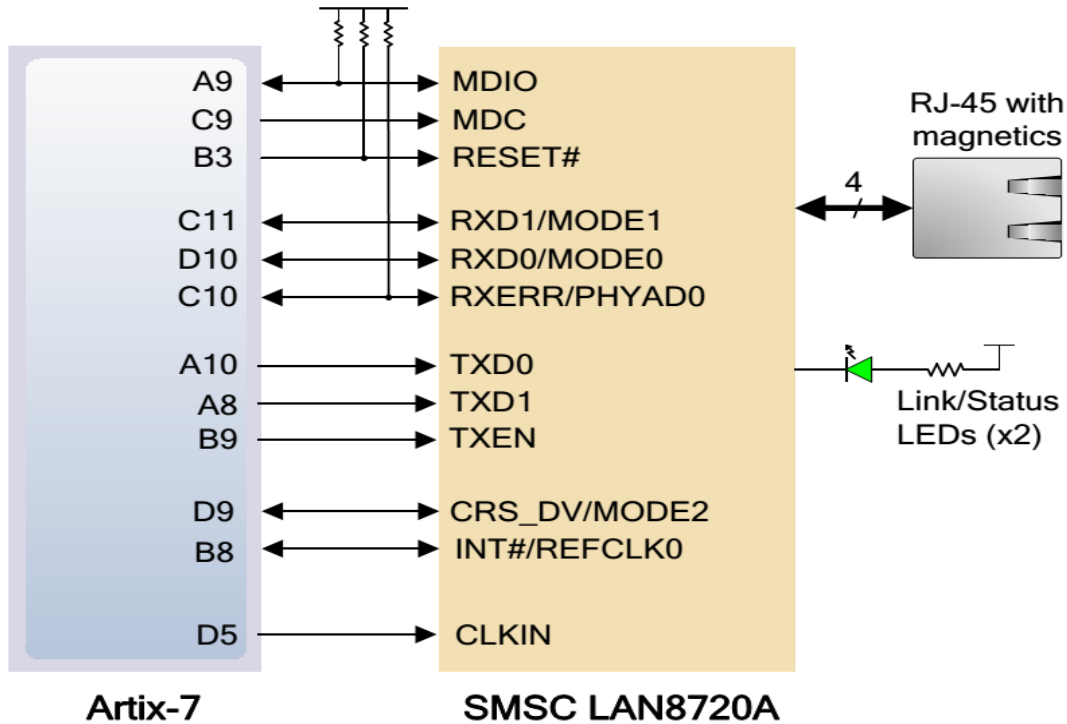


The Nexys4 board contains two external memories: a 128Mbit Cellular RAM (pseudo-static DRAM) and a 128Mbit non-volatile serial Flash device. The Cellular RAM has an SRAM interface, and the serial Flash is on a dedicated quad-mode (x4) SPI bus.

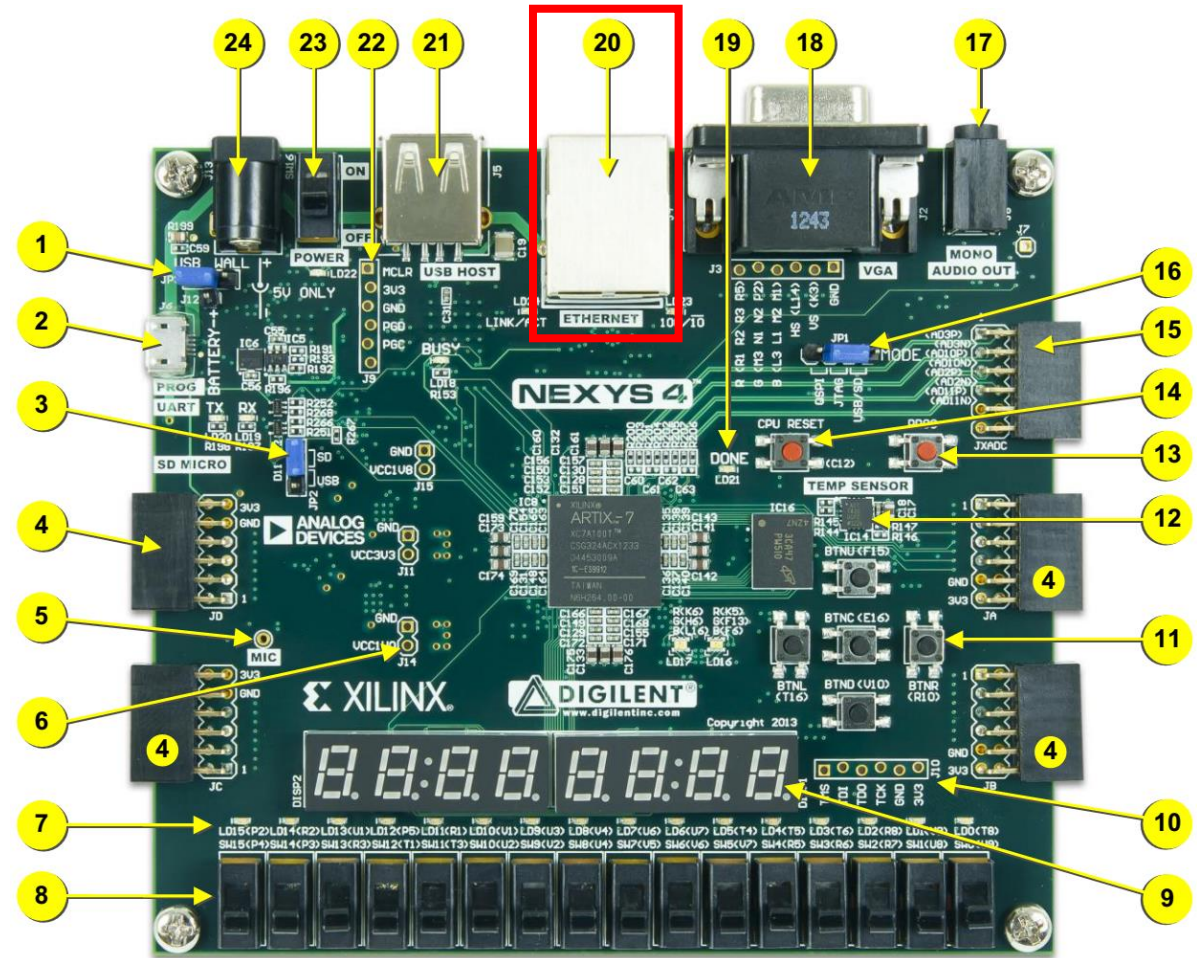


Nexys4实验板简介

► Ethernet PHY



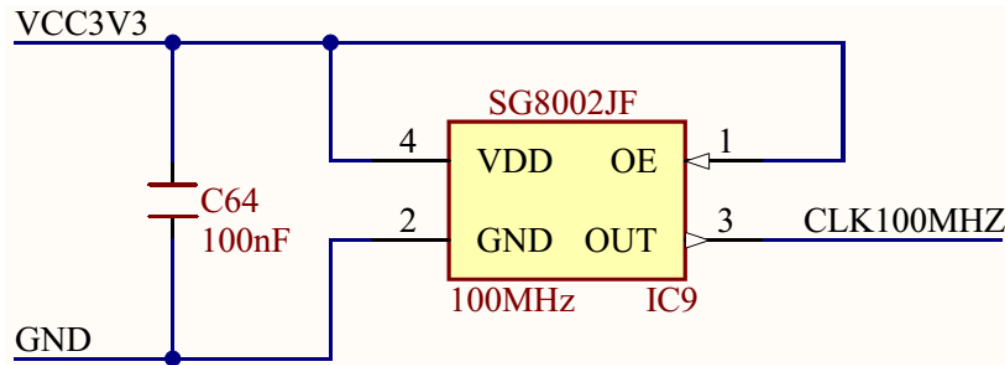
The Nexys4 board includes an SMSC 10/100 Ethernet PHY (SMSC part number LAN8720A) paired with an RJ-45 Ethernet jack with integrated magnetics.



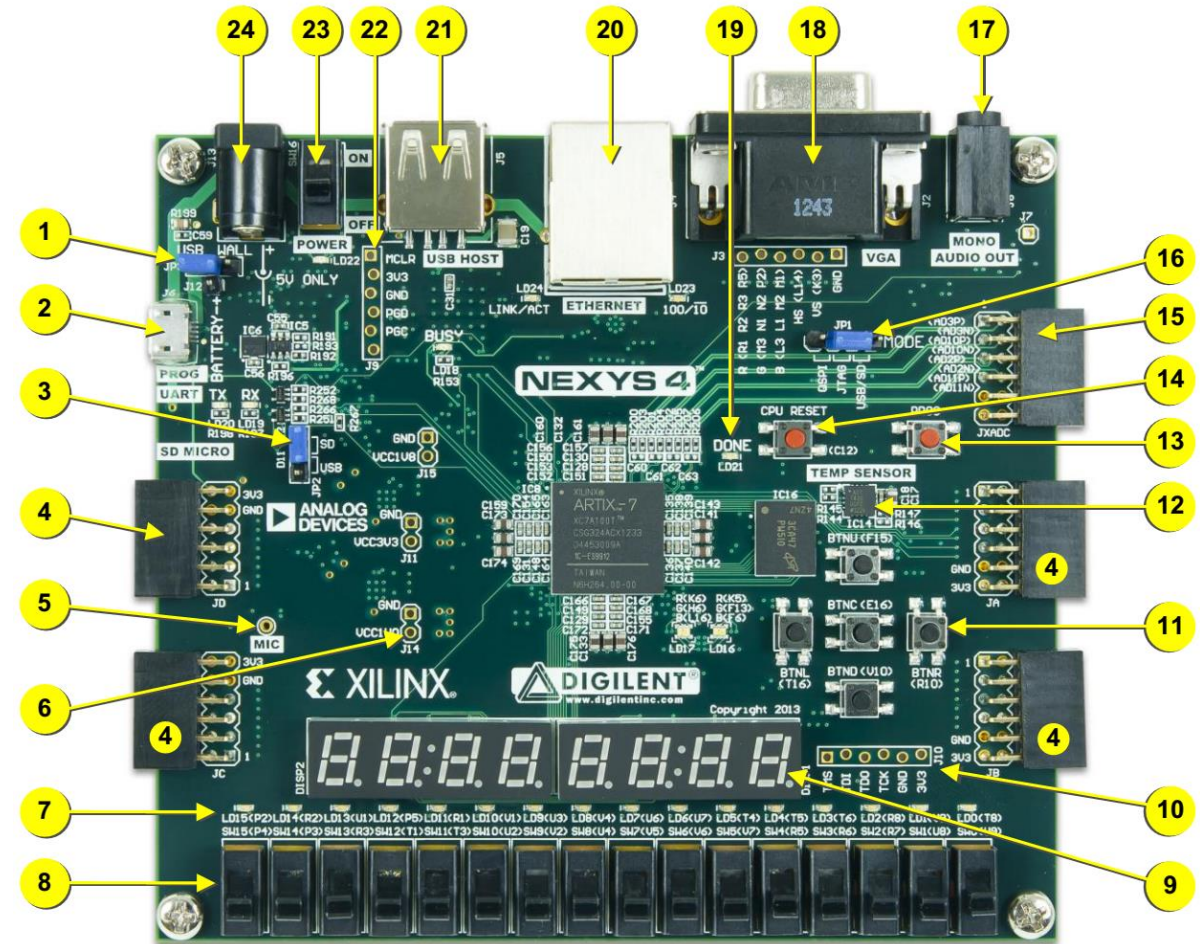
Nexys4实验板简介

► Oscillators/Clocks

- The Nexys4 board includes a single **100MHz** crystal oscillator **connected to pin E3** (E3 is a MRCC input on bank 35).

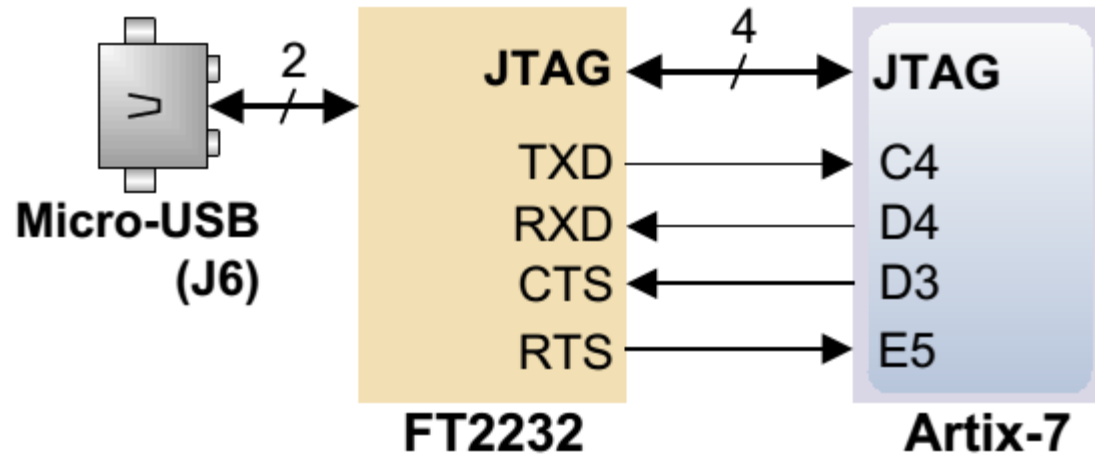


IO_L12P_T1_MRCC_35	E3	CLK100MHZ
IO_L12N_T1_MRCC_35	D3	UART CTS
IO_L13P_T2_MRCC_35	F4	PS2 CLK
	E3	PS2 DATA

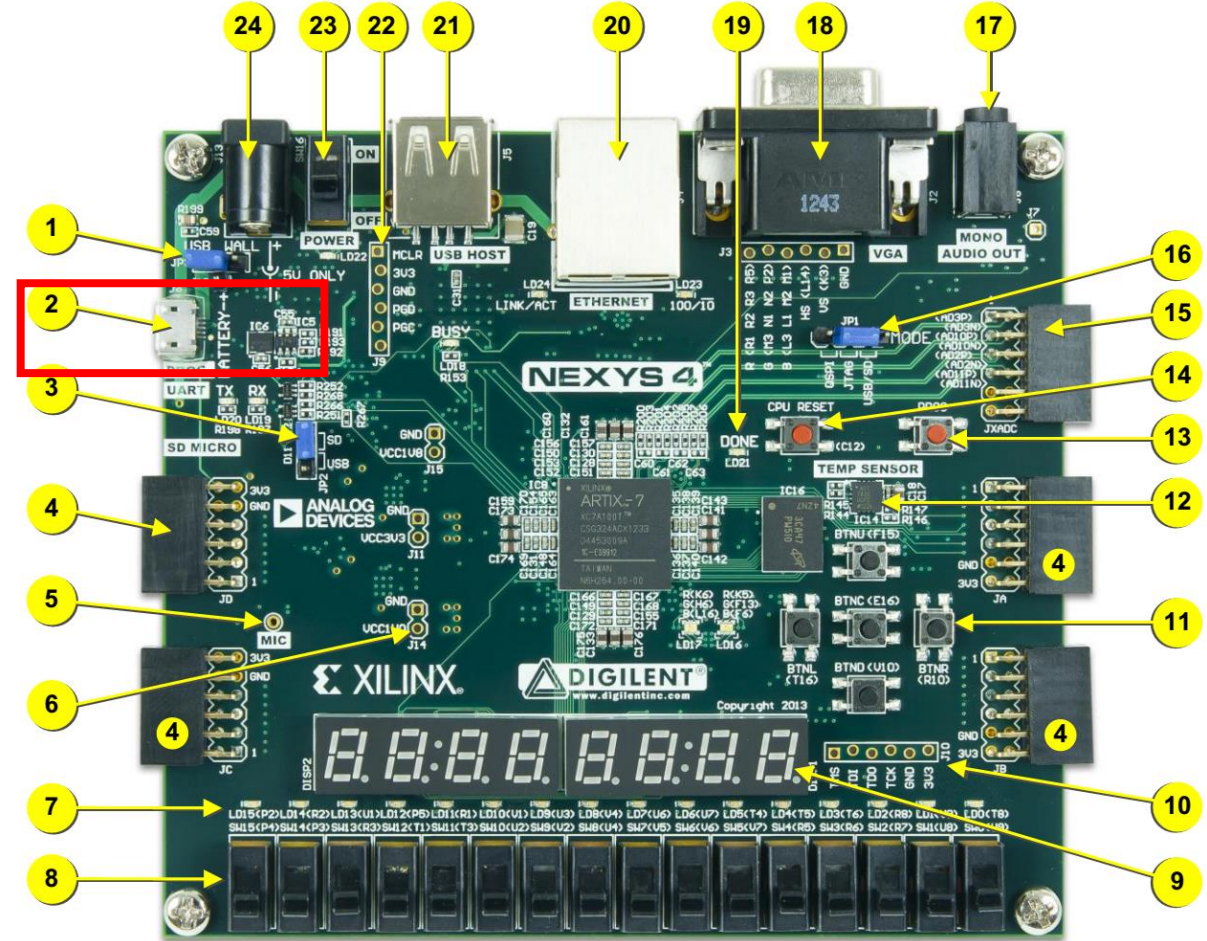


Nexys4实验板简介

► USB-UART Bridge (Serial Port)

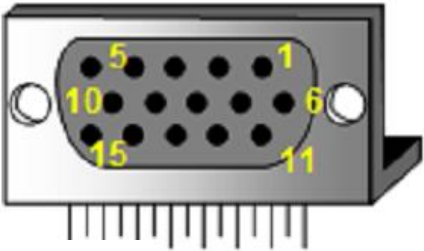


The Nexys4 includes an FTDI FT2232HQ USB-UART bridge (attached to connector J6) that allows you use PC applications to communicate with the board using standard Windows COM port commands.

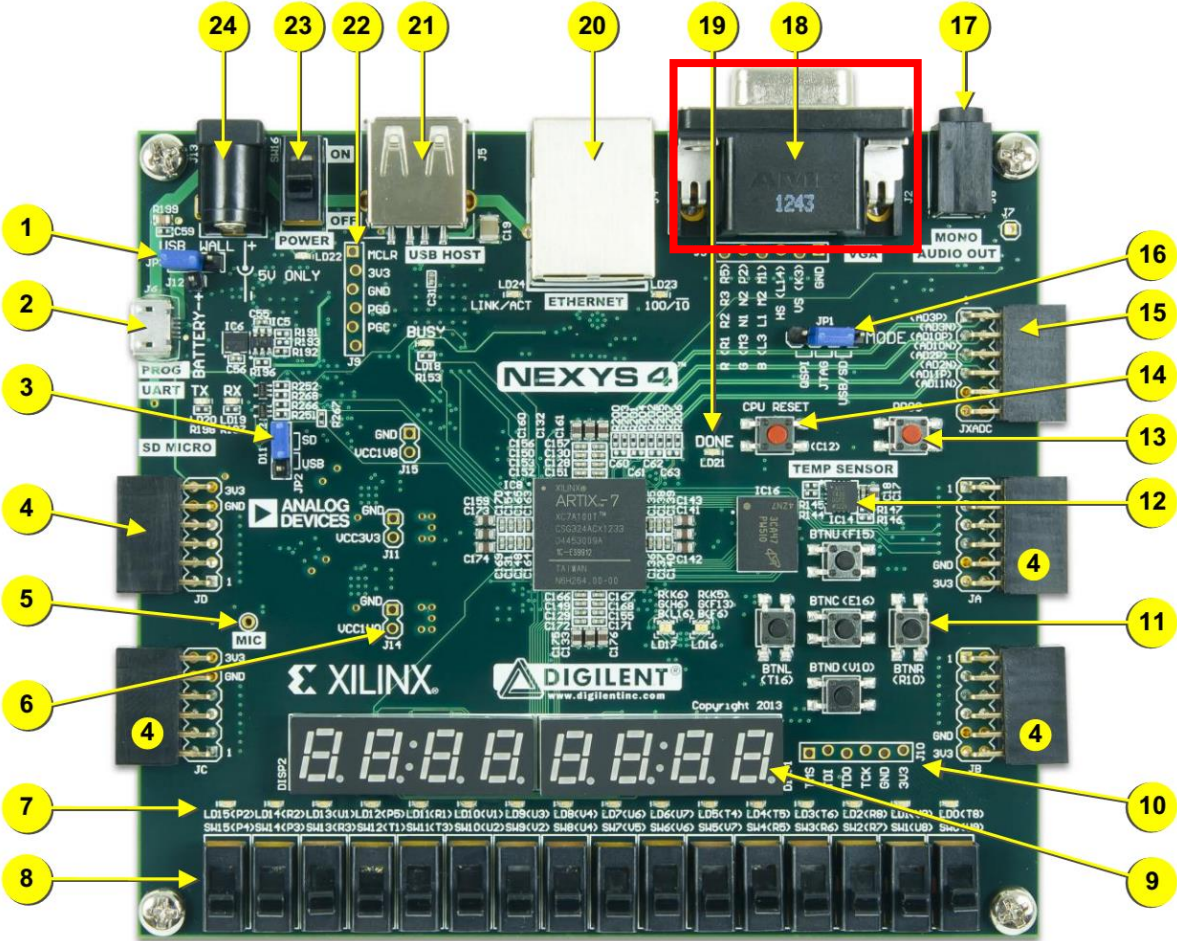
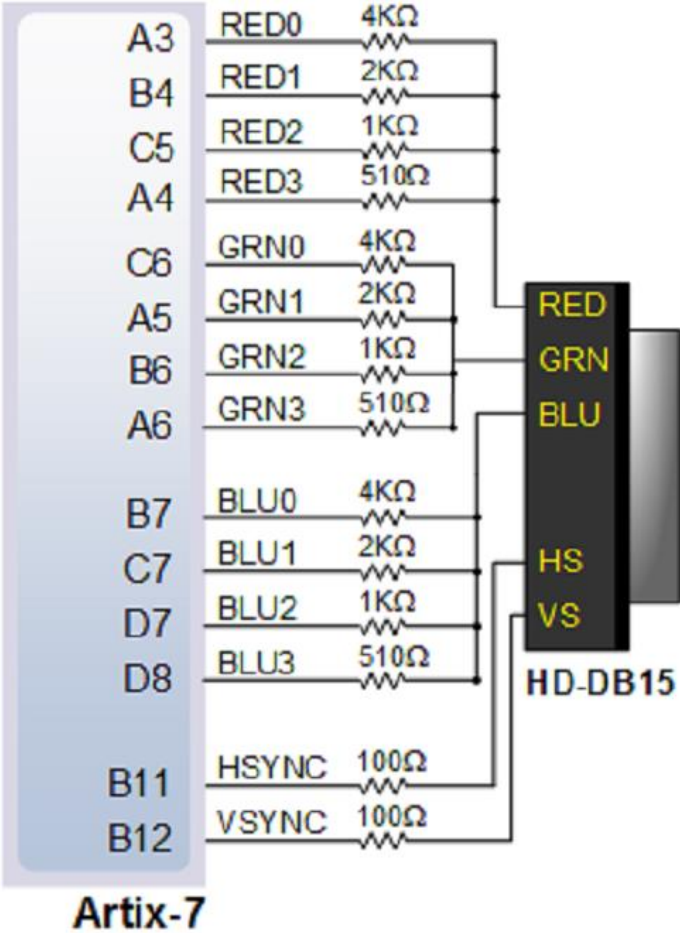


Nexys4实验板简介

► VGA Port



- Pin 1: Red
- Pin 2: Grn
- Pin 3: Blue
- Pin 13: HS
- Pin 14: VS
- Pin 5: GND
- Pin 6: Red GND
- Pin 7: Grn GND
- Pin 8: Blu GND
- Pin 10: Sync GND

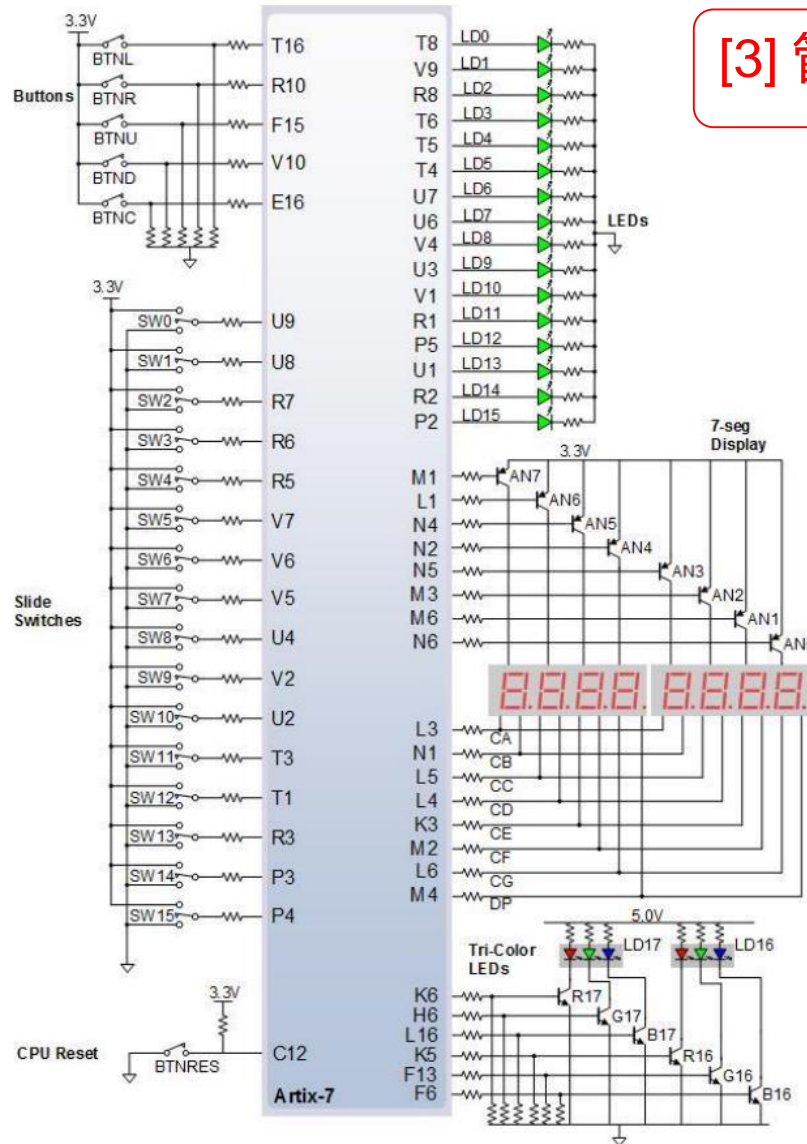


Nexys4实验板简介

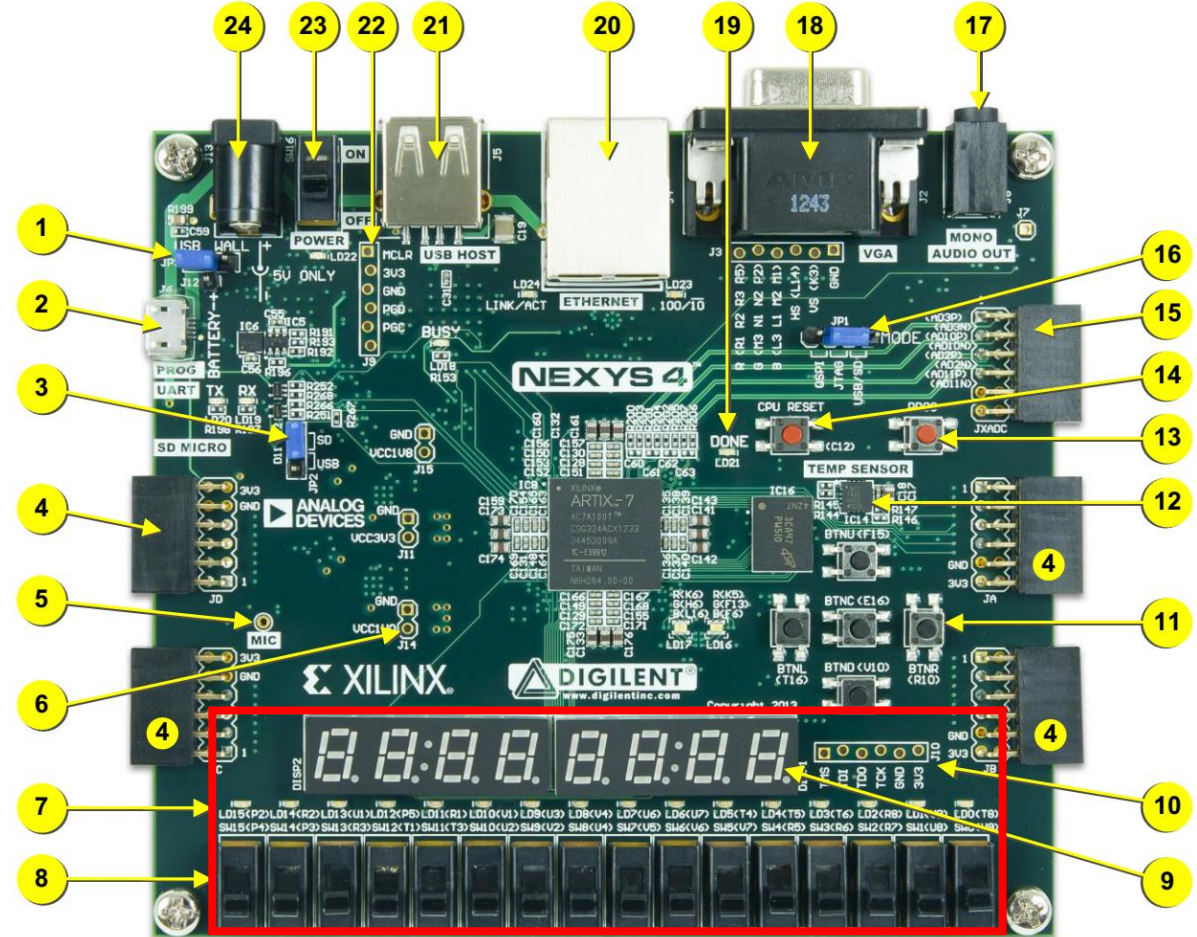
► Basis IO

- Button
- Switch
- LED
- 7-seg Dis
- Reset
- Tri-C LEDs

[1] 各种IO
会在实验中
频繁使用，
要会看管脚
分配。
[2] 本实验
中用到部分

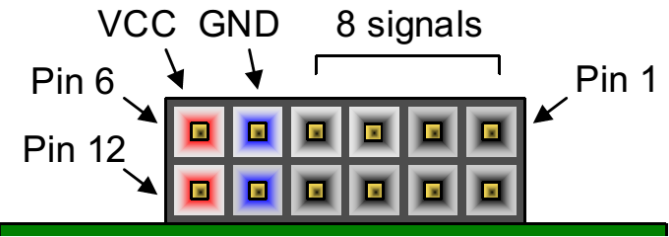


[3] 管脚分配详见文件：Nexys4_Master.ucf

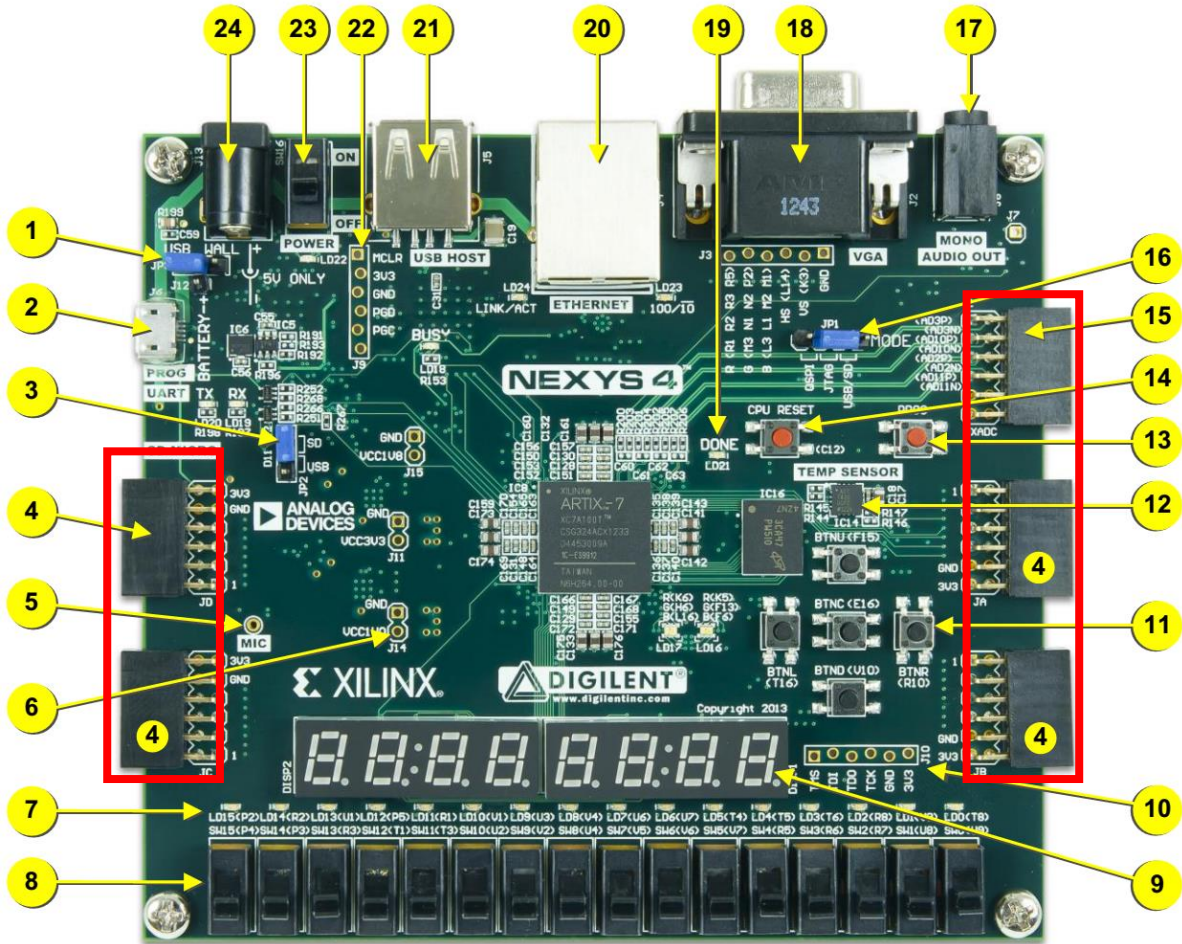


► Pmod Connectors

- [1] 提供外部扩展能力
- [2] 本实验中可用

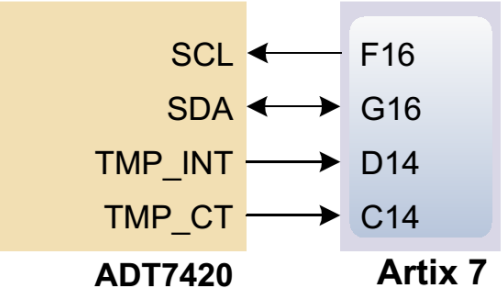


Pmod JA	Pmod JB	Pmod JC	Pmod JD	Pmod XDAC
JA1: B13	JP1: G14	JC1: K2	JD1: H4	JXADC1: A13
JA2: F14	JB2: P15	JC2: E7	JD2: H1	JXADC2: A15
JA3: D17	JB3: V11	JC3: J3	JD3: G1	JXADC3: B16
JA4: E17	JB4: V15	JC4: J4	JD4: G3	JXADC4: B18
JA7: G13	JB7: K16	JC7: K1	JD7: H2	JXADC7: A14
JA8: C17	JB8: R16	JC8: E6	JD8: G4	JXADC8: A16
JA9: D18	JB9: T9	JC9: J2	JD9: G2	JXADC9: B17
JA10: E18	JB10: U11	JC10: G6	JD10: F3	JXADC10: A18



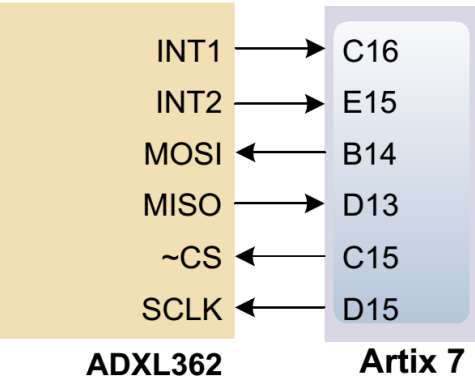
► 传感器

- 温度

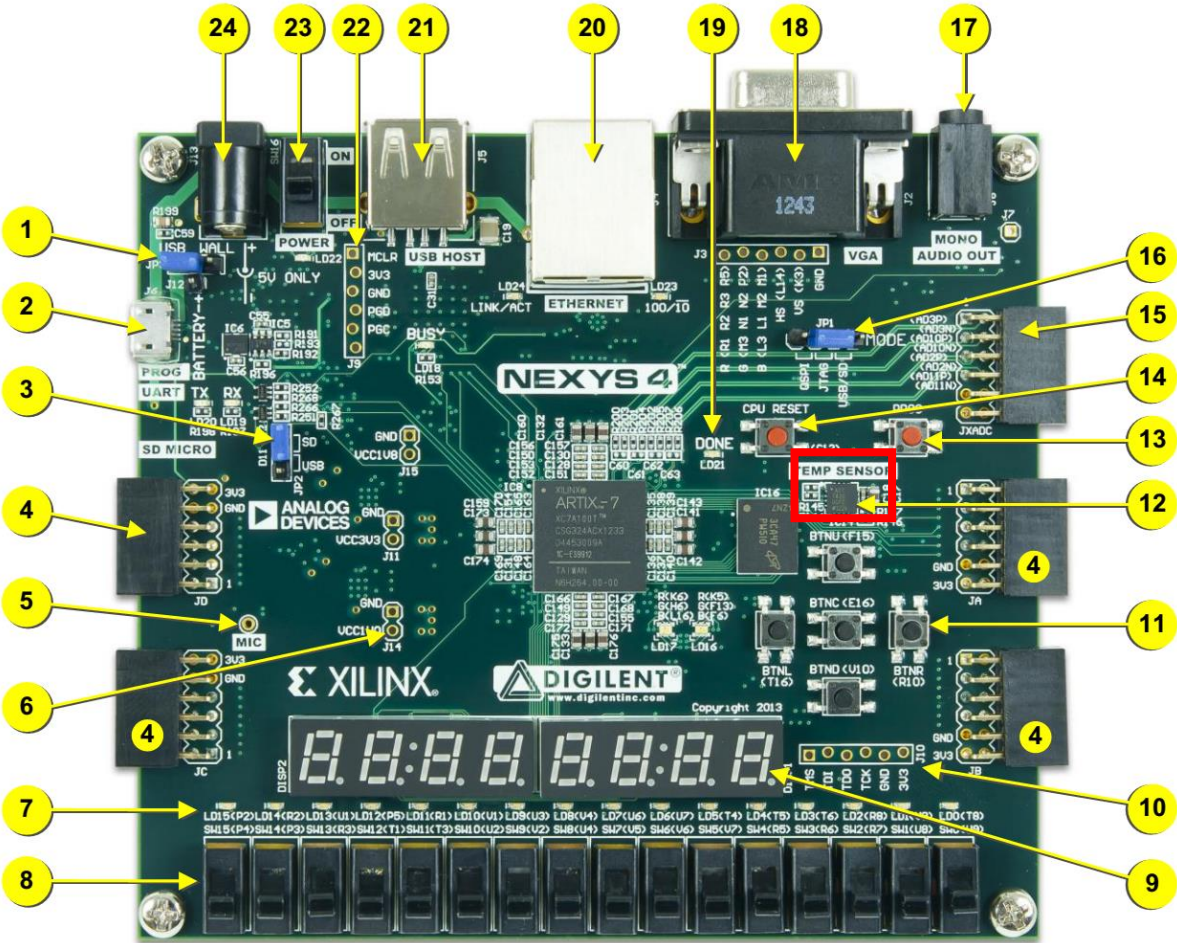


SCL: I2C Serial Clock
SDA: I2C Serial Data
TMP_INT: Over-temperature and Under-temperature Indicator
TMP_CT: Critical Over-temperature Indicator

- 加速度



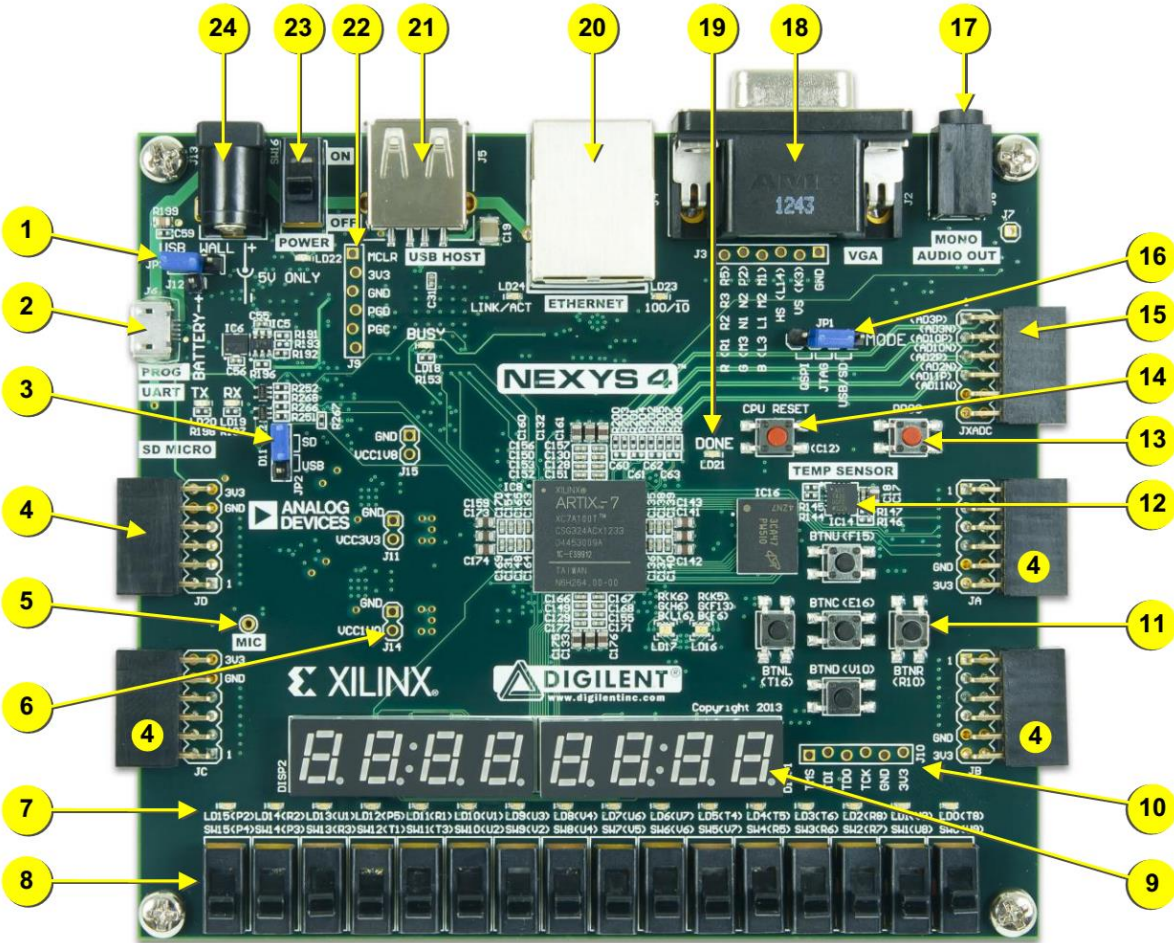
INT1: Interrupt One
INT2: Interrupt Two
MOSI: Master Out Slave In
MISO: Master In Slave Out
~CS: Slave Select (Active Low)
SCLK: Serial Clock



Nexys4实验板简介

- 其它
 - 键盘
 - 鼠标
 - IIC
 - SPI
 -

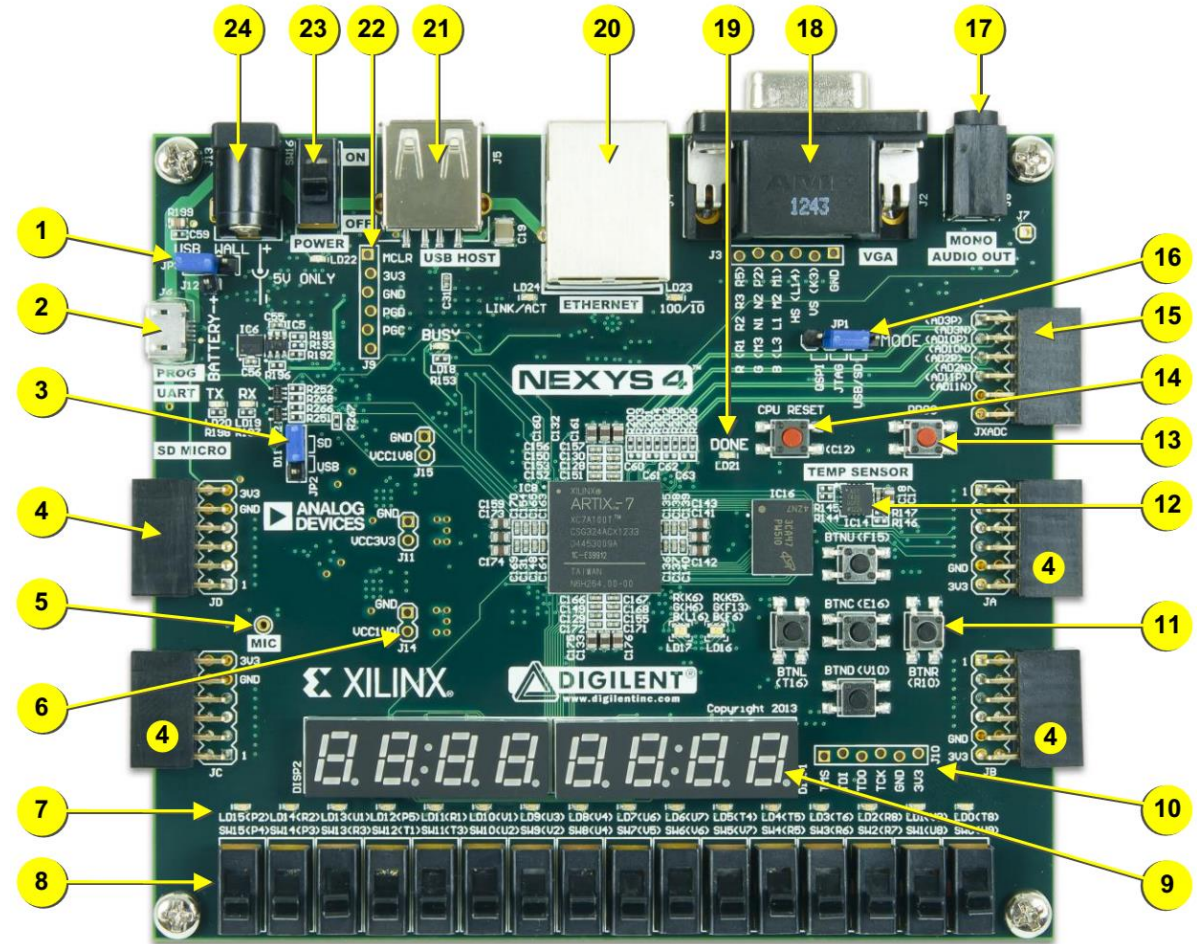
详见 “Nexys4_RM_VB2_Final_5.pdf”
—— www.digilentinc.com



Nexys4实验板简介

► 基于Nexys4可以做 ???

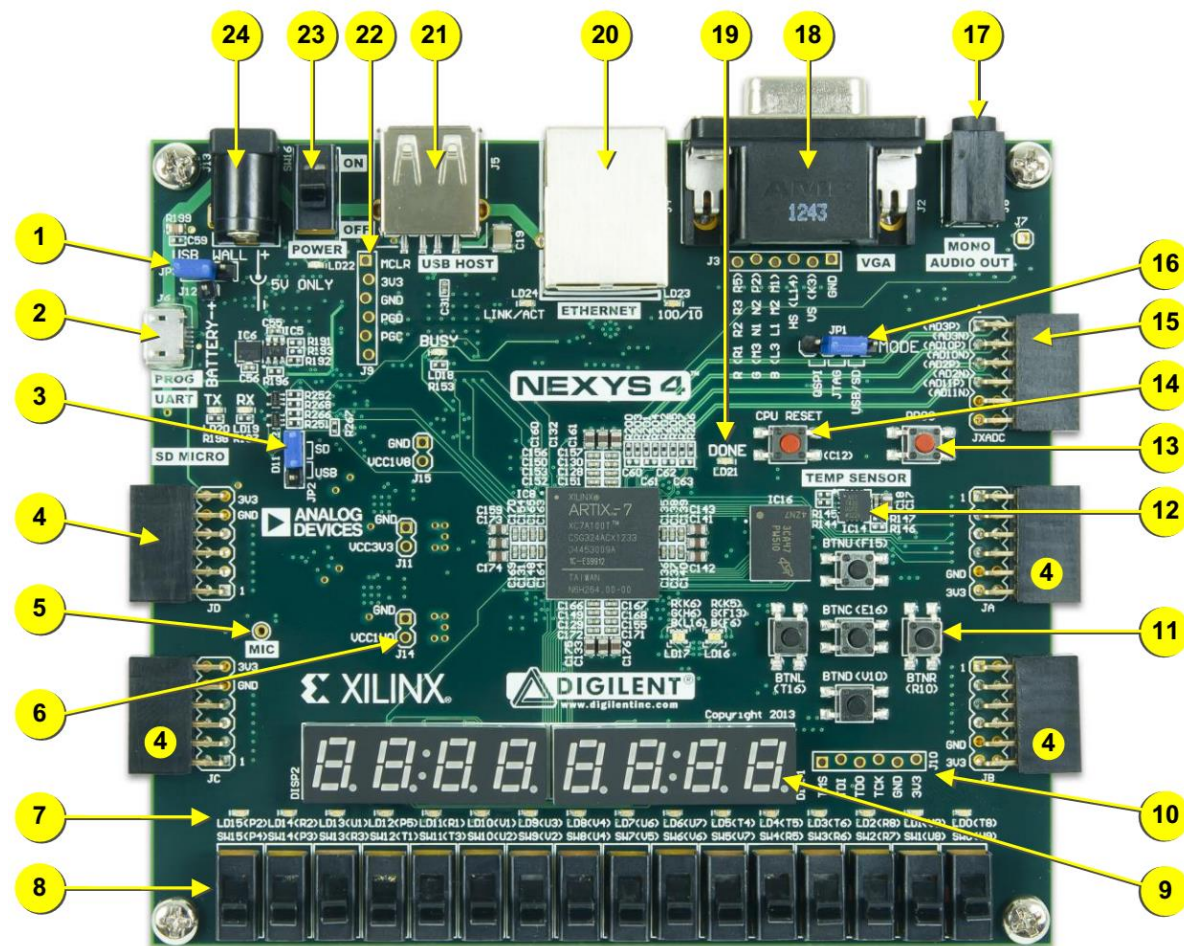
- 简易数字示波器
- 简易数字信号源
- VGA贪食蛇小游戏
- 超声波测距仪
- 简易手绘画图仪
- 简易电子琴
- 加速度测量仪 (跑步计步器)
-



Nexys4怎么用？

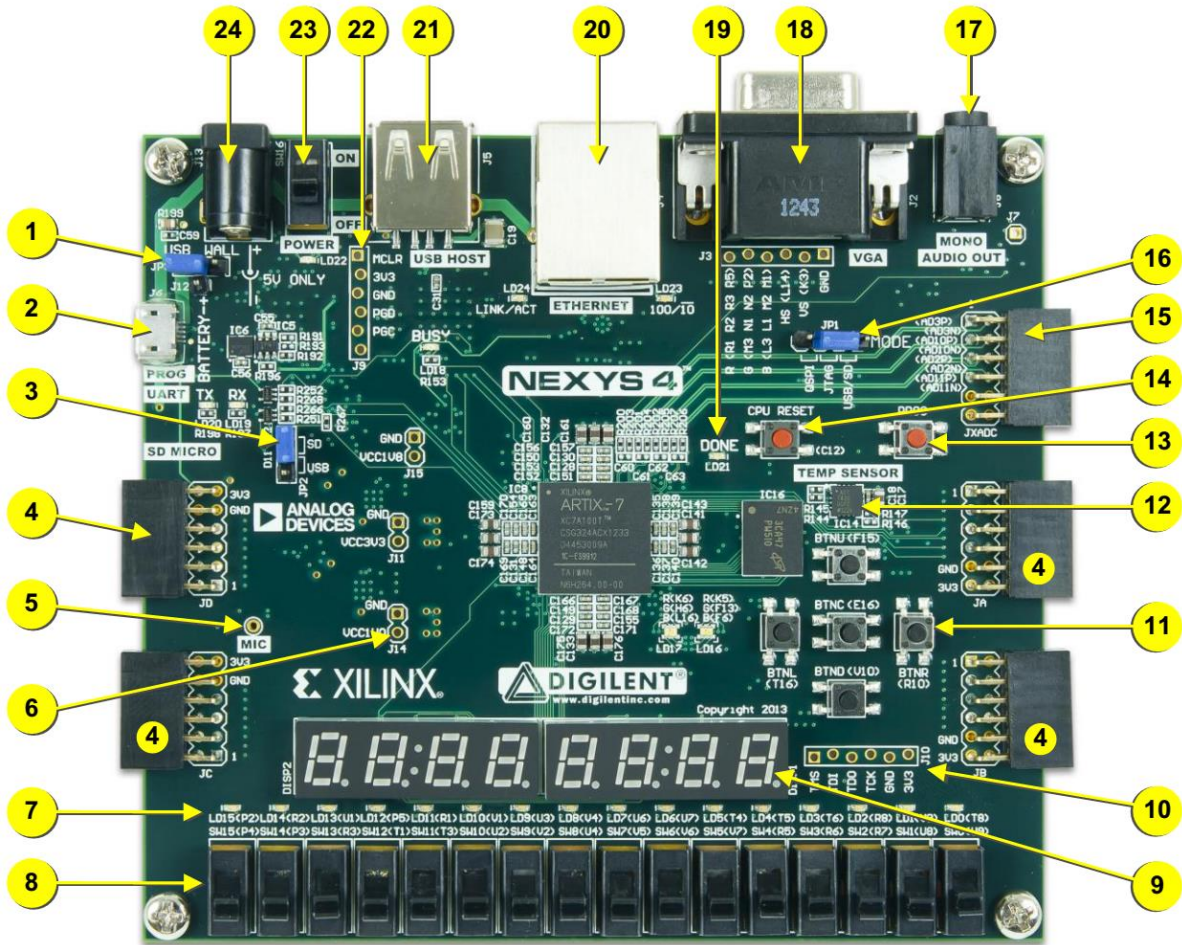
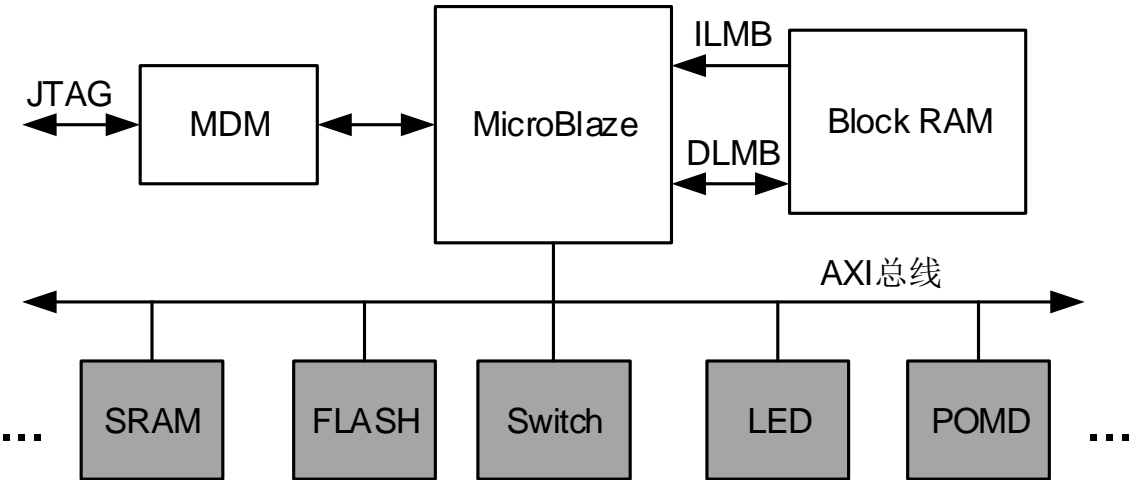
► 基于Nexys4怎么做？？？

- 纯FPGA方案
 - 所有功能用硬件描述语言实现
- 嵌入式处理器 + 应用软件方案
 - FPGA中设计处理器，并运行相应的用户程序，从而实现相应的功能。
 - 实验课中的IO接口都是采用此方案。

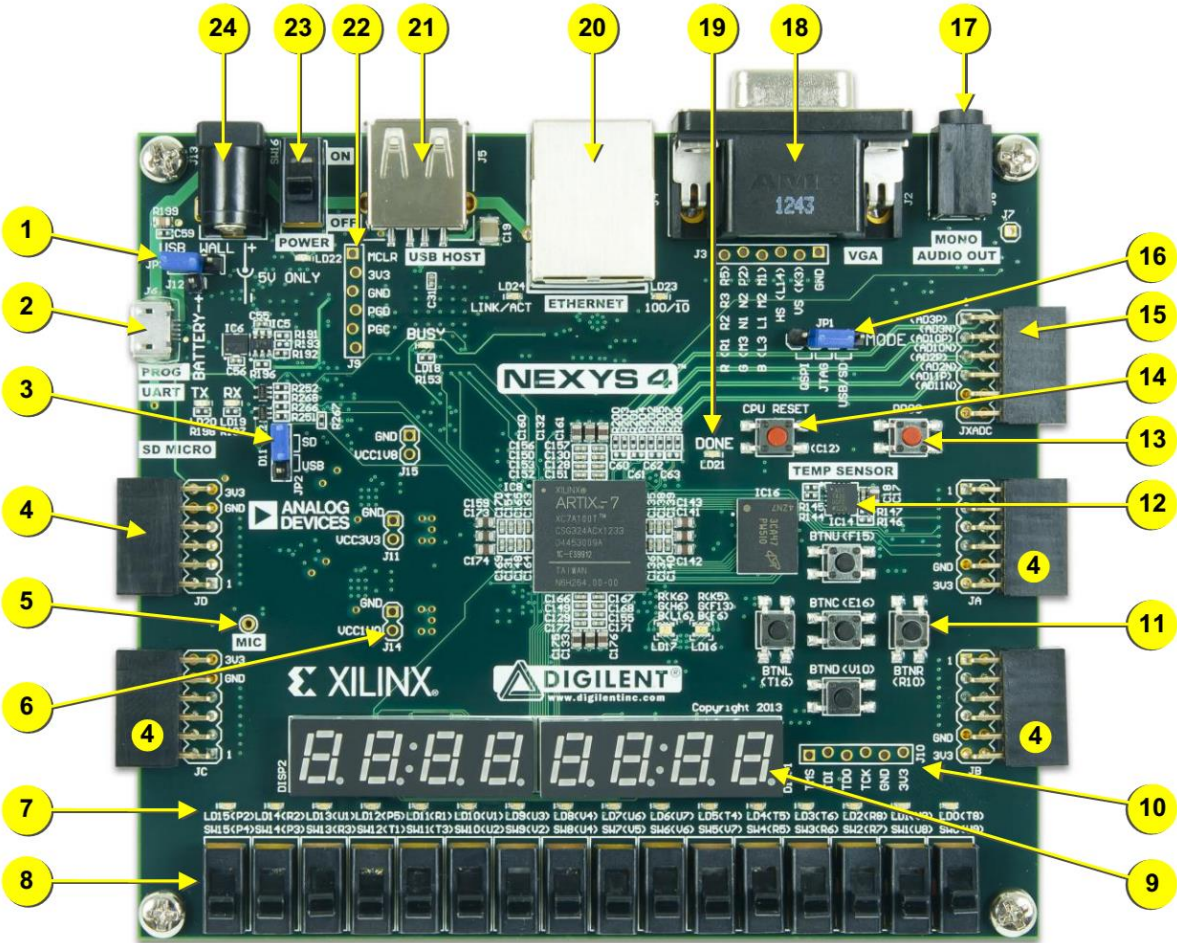
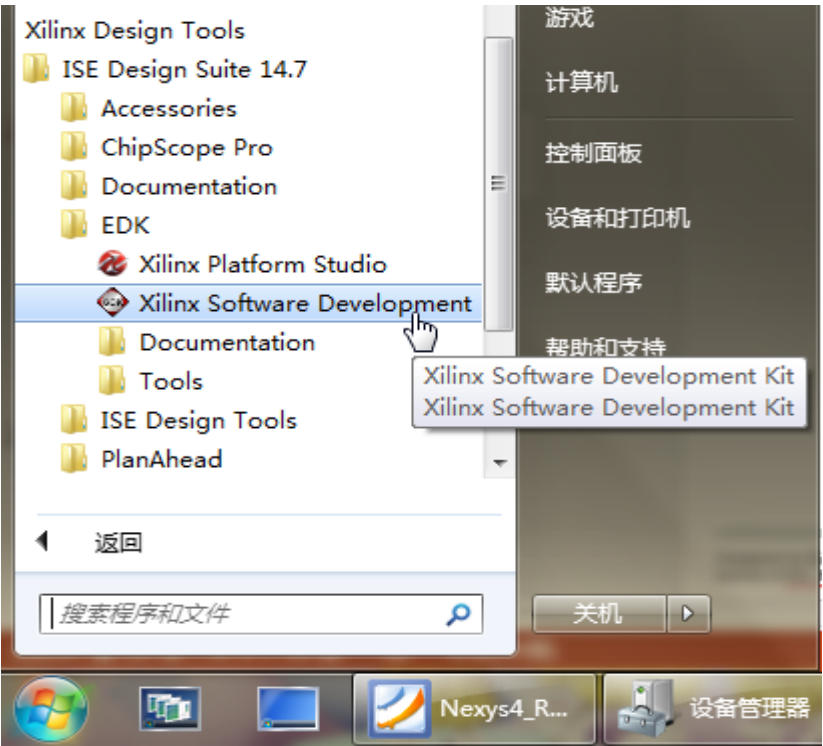


▶ 怎么在Nexys4中建立处理器硬件系统？

- 硬件开发环境（Vavado）
 - 建立最小系统
 - 添加基本硬件模块
- 详见：超星平台4.1



- ▶ 怎么写处理器运行的用户程序？
 - 软件开发环境SDK (Software Development Kit)
 - C语言
 - 详见： 超星平台4.2、4.3



► 实验内容

- 目的
- 任务及时间安排
- 报告要求

► 原理回顾

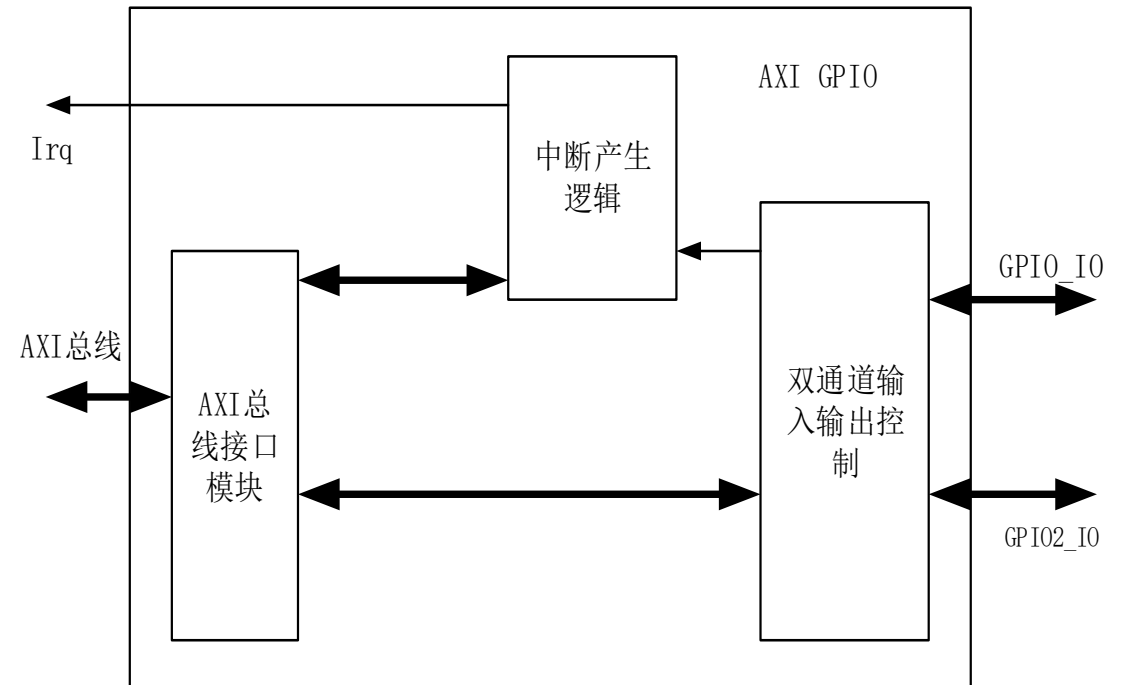
- Nexys4实验板简介
- Nexys4怎么用？
- Xilinx的GPIO和INTC
- GPIO硬件设计
- GPIO应用软件设计
- 系统功能测试

► GPIO

- GPIO (general purpose IO) 是通用并行IO接口的简称。它将总线信号转换为IO设备要求的信号类型，实现地址译码、输出数据锁存、输入数据缓冲的功能

► Xilinx AXI总线GPIO IP核

- 包括AXI总线接口模块、中断产生逻辑、双通道I/O模块
- 每个通道都可以支持1~32位的数据输入输出，可以配置为单输入、单输出或双向输入输出 —— 怎么配置？？
(通过寄存器)



Xilinx的GPIO和INTC

► GPIO内部框图及其寄存器

• I/O方向

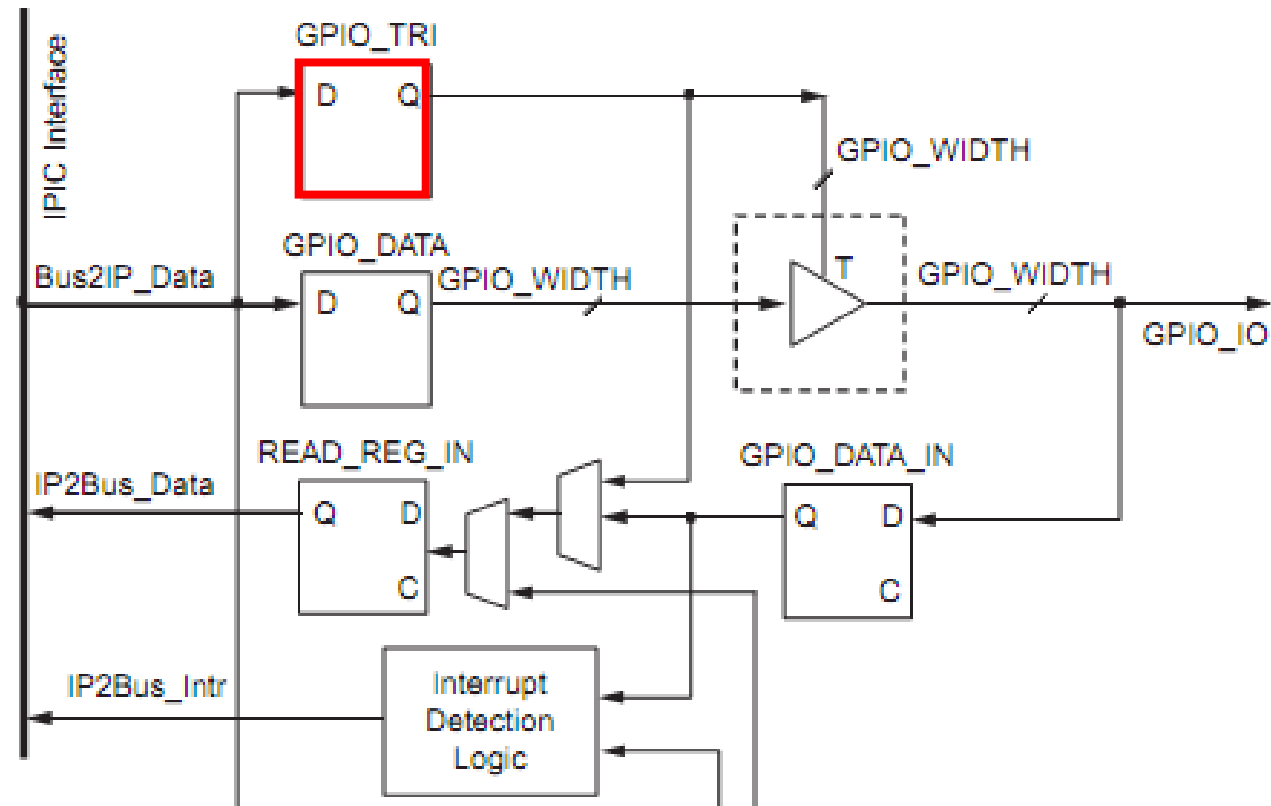
- 当GPIO_TRI某位为0时，GPIO相应的IO引脚配置为输出；
- 当GPIO_TRI某位为1时，GPIO相应的IO引脚配置为输入；

• I/O读写

```
# include "xil_io.h"
```

```
...  
Xil_In8(Addr);  
Xil_In16(Addr);  
Xil_In32(Addr);
```

```
Xil_Out8(Addr, Value);  
Xil_Out16(Addr, Value);  
Xil_Out32(Addr, Value);
```



寄存器名称	偏移地址	初始值	含义	读写操作
GPIO_DATA	0x0	0	通道1数据寄存器	通道1数据
GPIO_TRI	0x4	0	通道1三态控制寄存器	写控制通道1传输方向
GPIO2_DATA	0x8	0	通道2数据寄存器	通道2数据
GPIO2_TRI	0xC	0	通道2三态控制寄存器	写控制通道2传输方向

► GPIO内部框图及其寄存器

- I/O模块的**基地址**
 - XPS和SDK中均可查看
 - xparameters.h文件也可查看

```
# include "xil_io.h"
```

```
...
Xil_In8(Addr);
Xil_In16(Addr);
Xil_In32(Addr);
```

```
Xil_Out8(Addr, Value);
Xil_Out16(Addr, Value);
Xil_Out32(Addr, Value);
```

The screenshot shows the Xilinx IDE interface. The top window displays the 'Addresses' tab of the 'microblaze_0's Address Map'. The table below is a transcription of the data shown in this window:

Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)
microblaze_0_d_bram_ctrl	C_BASEADDR	0x00000000	0x00007FFF	32K	SLMB
microblaze_0_i_bram_ctrl	C_BASEADDR	0x00000000	0x00007FFF	32K	SLMB
Button	C_BASEADDR	0x40000000	0x4000FFFF	64K	S_AXI
Dip	C_BASEADDR	0x40040000	0x4004FFFF	64K	S_AXI
RS232	C_BASEADDR	0x40600000	0x4060FFFF	64K	S_AXI
			FFFF	64K	S_AXI
			FFFF	64K	S_AXI

The bottom window shows the 'lab11_hw_platform Hardware Platform Specification'. Under 'Design Information', it lists:

- Target FPGA Device: xc7a100t
- Created With: EDK 14.7
- Created On: Fri Nov 13 12:09:07 2015
- XPS Design Report: file:///F:/EDA/Xilinx/User/Nexys4/lab13.5.1/SDK/SDK_Export/hw/system.html

Under 'Address Map for processor microblaze_0', it lists the following components and their address ranges:

- microblaze_0_d_bram_ctrl 0x00000000 0x00007fff
- microblaze_0_i_bram_ctrl 0x00000000 0x00007fff
- debug_module 0x41400000 0x4140ffff
- rs232 0x40600000 0x4060ffff
- button 0x40000000 0x4000ffff
- dip 0x40040000 0x4004ffff
- axi_intc_0 0x41200000 0x4120ffff

寄存器名称	偏移地址	初始值	含义	读写操作
GPIO_DATA	0x0	0	通道1数据寄存器	通道1数据
GPIO_TRI	0x4	0	通道1三态控制寄存器	写控制通道1传输方向
GPIO2_DATA	0x8	0	通道1数据寄存器	通道2数据
GPIO2_TRI	0xC	0	通道1三态控制寄存器	写控制通道1传输方向

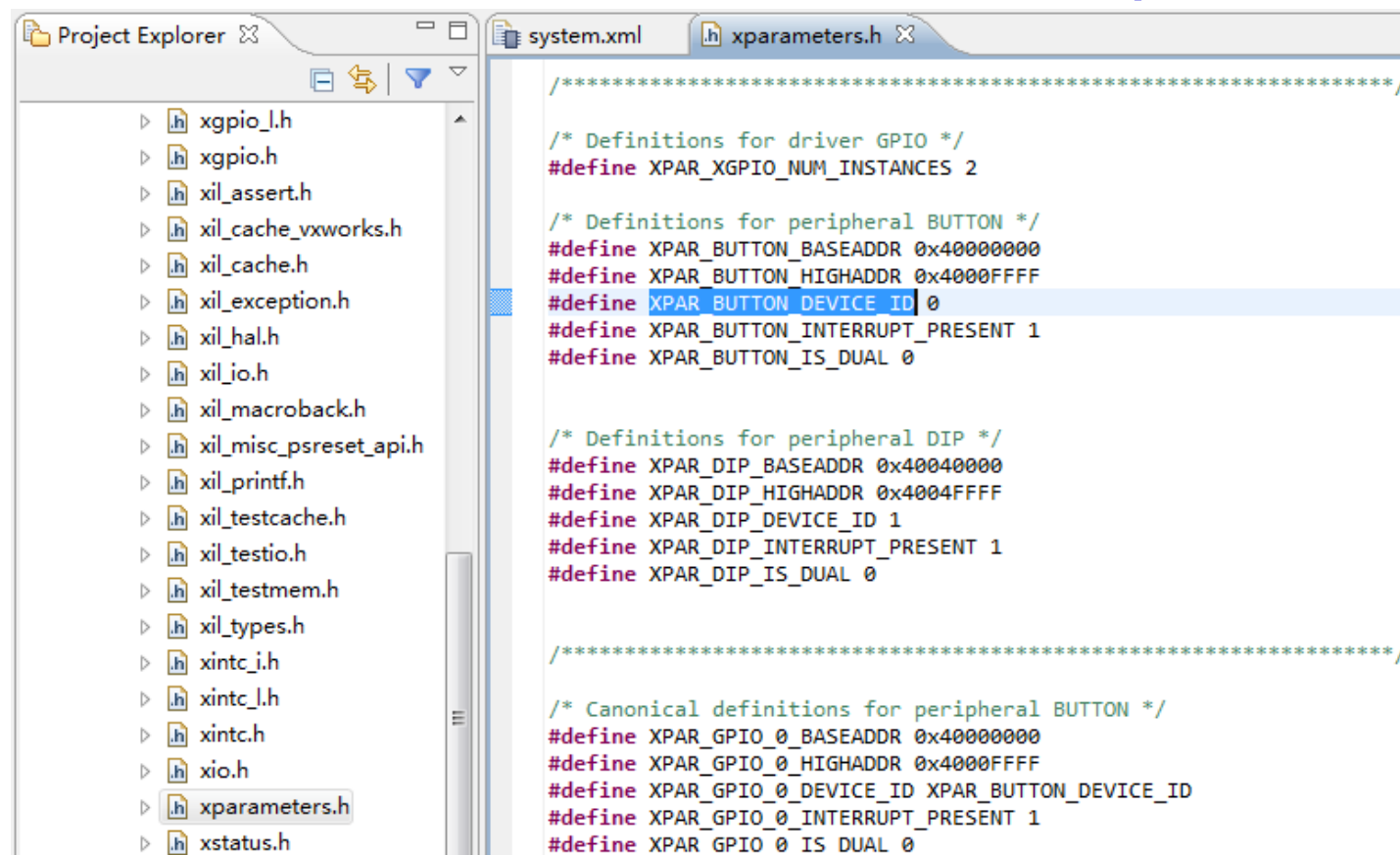
► GPIO内部框图及其寄存器

- I/O模块的**基地址**
 - XPS和SDK中均可查看
 - xparameters.h**文件也可查看

```
# include "xil_io.h"
```

```
...
Xil_In8(Addr);
Xil_In16(Addr);
Xil_In32(Addr);
```

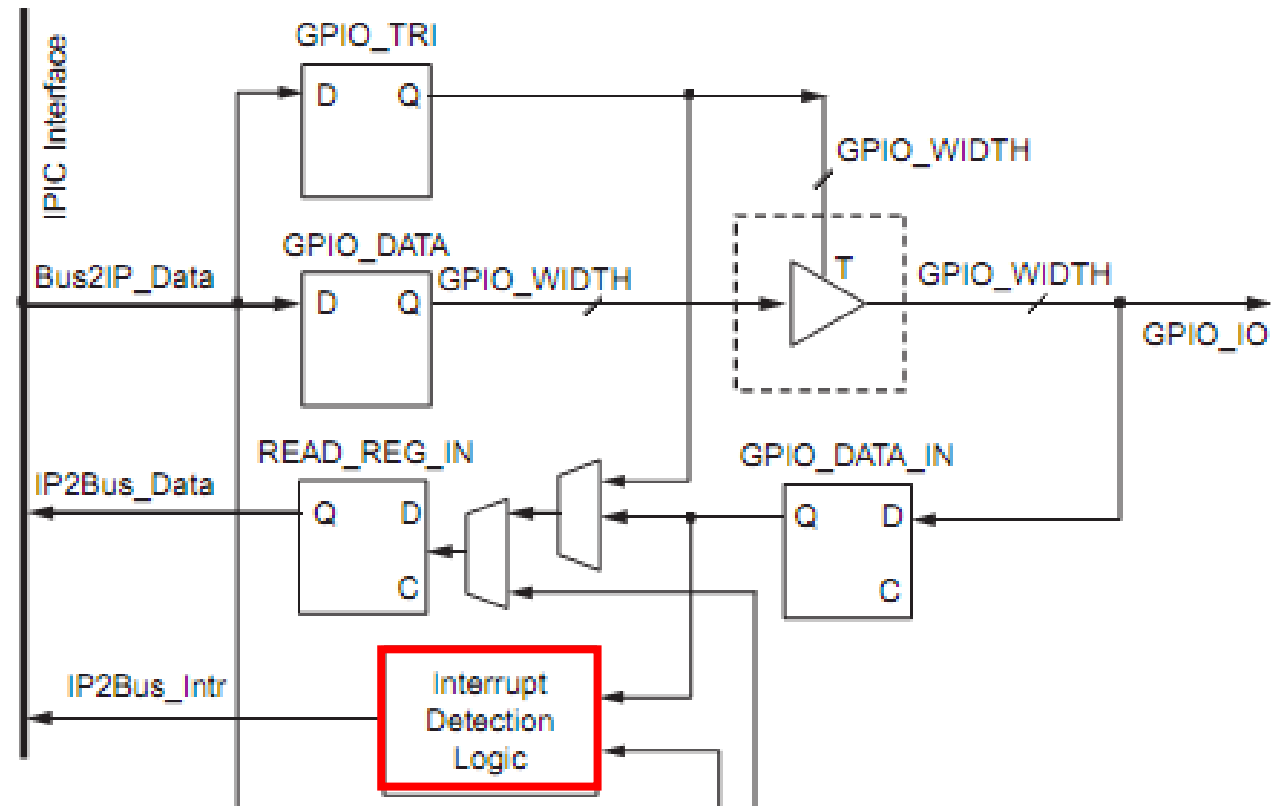
```
Xil_Out8(Addr, Value);
Xil_Out16(Addr, Value);
Xil_Out32(Addr, Value);
```



寄存器名称	偏移地址	初始值	含义	读写操作
GPIO_DATA	0x0	0	通道1数据寄存器	通道1数据
GPIO_TRI	0x4	0	通道1三态控制寄存器	写控制通道1传输方向
GPIO2_DATA	0x8	0	通道1数据寄存器	通道2数据
GPIO2_TRI	0xC	0	通道1三态控制寄存器	写控制通道1传输方向

► GPIO内部框图及其寄存器

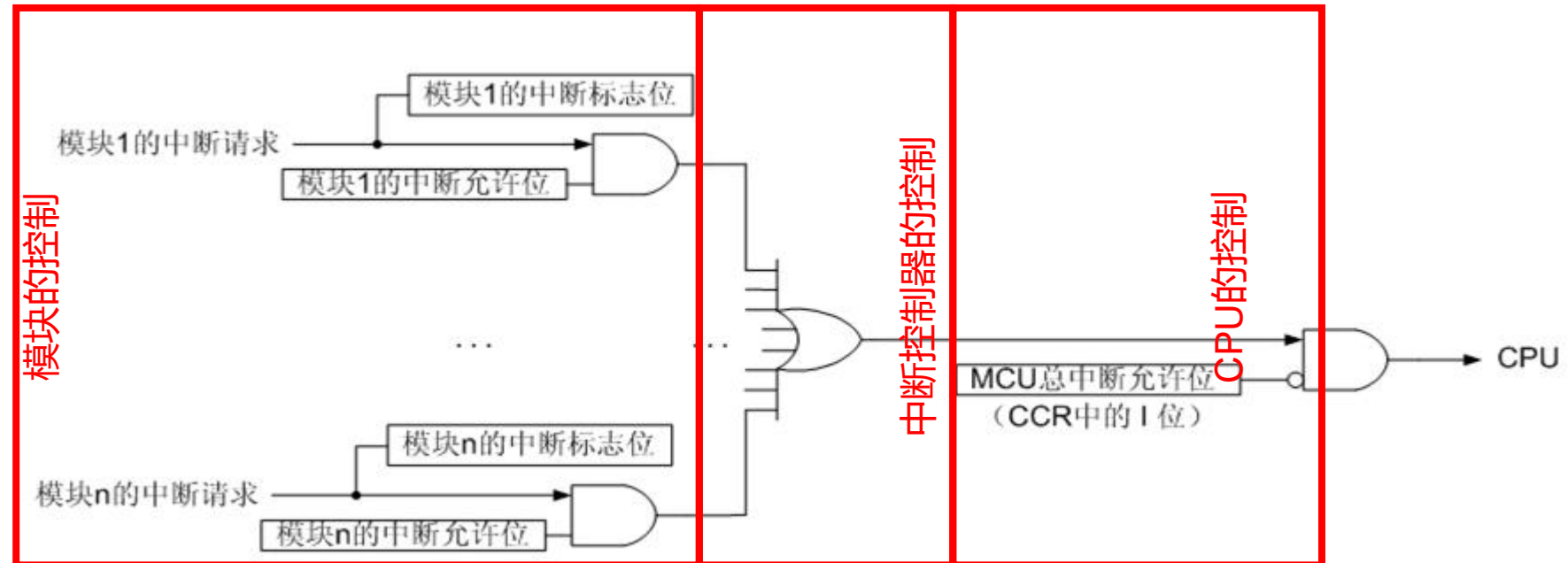
- I/O中断



名称	偏移地址	含义	读写操作
GIER	0x11C	全局中断屏蔽寄存器	最高位bit31控制GPIO是否输出中断信号Irq
IP IER	0x128	中断屏蔽寄存器	控制各个通道是否允许产生中断 bit0-通道1；bit1-通道2
IP ISR	0x120	中断状态寄存器	各个通道的中断请求状态，写1将清除相应位的中断状态 bit0-通道1；bit1-通道2

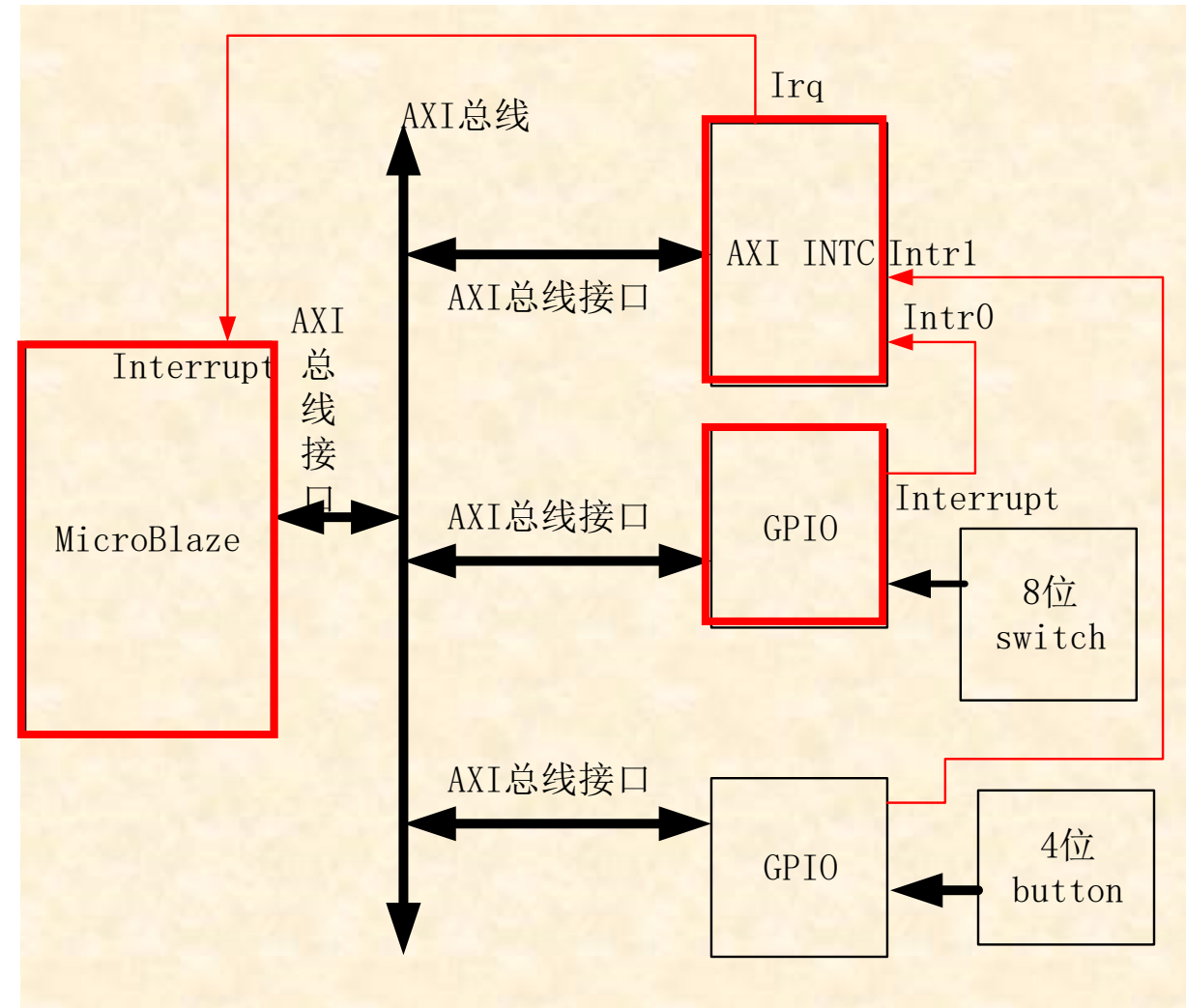
中断

- 中断是指CPU停止当前操作，保存好CPU当前寄存器的值，转而执行中断服务子程序ISR（Interrupt Service Routine），然后恢复CPU寄存器值并返回执行中断前的操作的过程。多数中断是因为硬件事件引起的。例如I/O管脚上的边沿信号或者定时器的溢出等等。
- CPU能否收到模块的中断请求取决于：
 - 模块的中断控制
 - 中断控制器的控制
 - CPU的控制



► 中断系统硬件电路框图

- 中断的三个层次
 - 模块(GPIO)
 - 中断控制器(INTC)
 - CPU(MicroBlaze)
- CPU能否收到模块的中断请求取决于
 - 模块的中断控制
 - 中断控制器的控制
 - CPU的控制

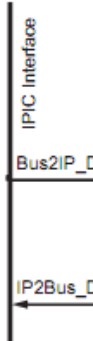


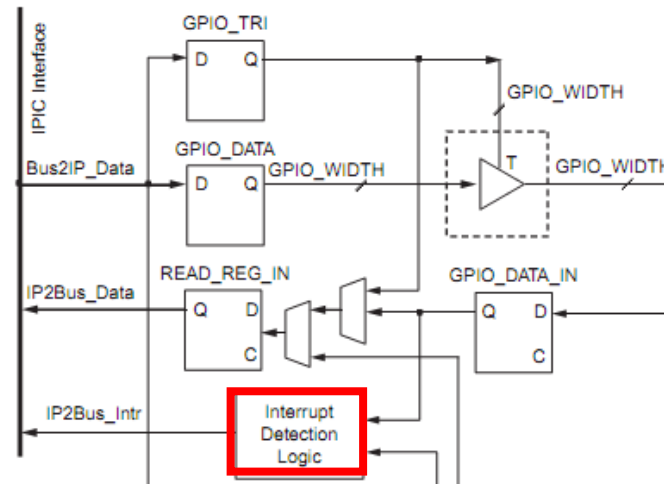

```

+ XGpio_Initialize(XGpio*, u16) : int
+ XGpio_LookupConfig(u16) : XGpio_Config*
+ XGpio_CfgInitialize(XGpio*, XGpio_Config*, u32) : int
+ XGpio_SetDataDirection(XGpio*, unsigned, u32) : void
+ XGpio_GetDataDirection(XGpio*, unsigned) : u32
+ XGpio_DiscreteRead(XGpio*, unsigned) : u32
+ XGpio_DiscreteWrite(XGpio*, unsigned, u32) : void
+ XGpio_DiscreteSet(XGpio*, unsigned, u32) : void
+ XGpio_DiscreteClear(XGpio*, unsigned, u32) : void
XGpio_SelfTest(XGpio*) : int
XGpio_InterruptGlobalEnable(XGpio*) : void
XGpio_InterruptGlobalDisable(XGpio*) : void
XGpio_InterruptEnable(XGpio*, u32) : void
XGpio_InterruptDisable(XGpio*, u32) : void
XGpio_InterruptClear(XGpio*, u32) : void
XGpio_InterruptGetEnabled(XGpio*) : u32
XGpio_InterruptGetStatus(XGpio*) : u32

```

xgpio.h文件中的驱动函数

- 模块的中断控制
 - 如果要允许I/O模块的中断，则需要对GIER、IP IER进行编程。
 - I/O的API函数
 - Xil_Out(addr, data), Xil_In()
 - 中断控制器的控制
 - CPU的控制
- 
- The diagram shows a vertical line representing the IPIC Interface. It has two horizontal connections: one to Bus2IP_D and another to IP2Bus_D.



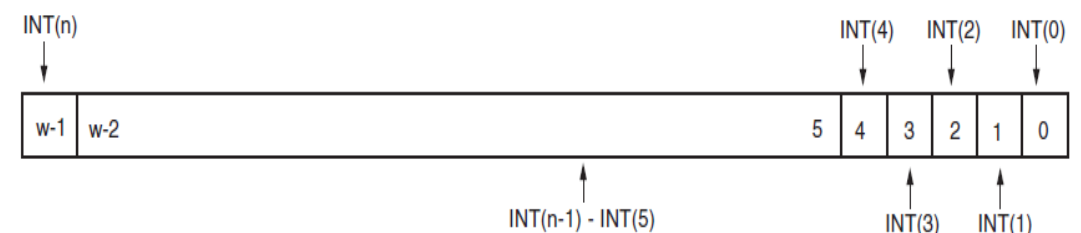
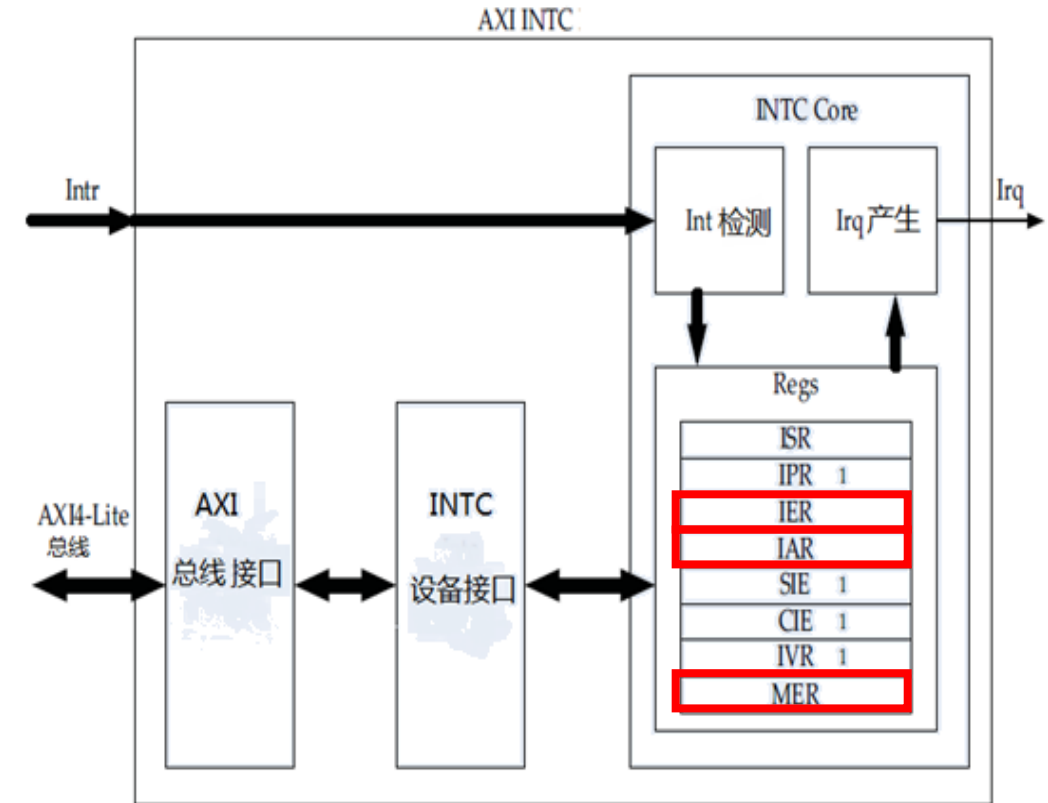
名称	偏移地址	含义	读写操作
GIER	0x11C	全局中断屏蔽寄存器	最高位bit31控制GPIO是否输出中断信号Irq
IP IER	0x128	中断屏蔽寄存器	控制各个通道是否允许产生中断 bit0-通道1；bit1-通道2
IP ISR	0x120	中断状态寄存器	各个通道的中断请求状态，写1将清除相应位的中断状态 bit0-通道1；bit1-通道2

Xilinx的GPIO和INTC

► CPU能否收到模块的中断请求取决于：

- 模块的中断控制
- 中断控制器的控制
 - 则需要对**IER**、**MER**进行编程。
 - INT的API函数
 - Xil_Out(addr, data) , Xil_In()

• CPU的控制



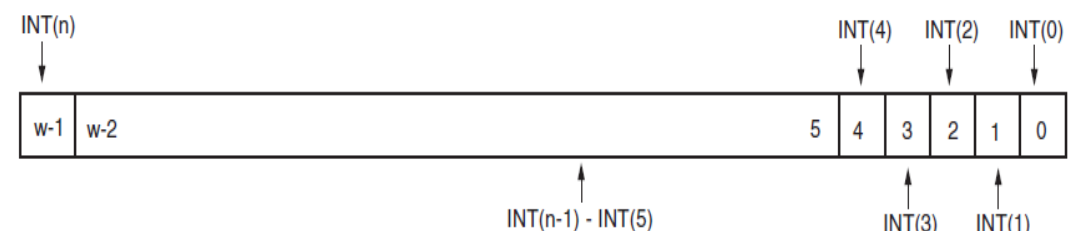
Xilinx的GPIO和INTC

► CPU能否收到模块的中断请求取决于：

- 模块的中断控制
- 中断控制器的控制
 - 则需要对**IER**、**MER**进行编程。
 - INT的API函数
 - Xil_Out(addr, data) , Xil_In()

• CPU的控制

寄存器名称	偏移地址	允许操作	初始值	含义
ISR	0x0	Read / Write	0x0	中断请求状态寄存器
IPR (可选)	0x4	Read	0x0	中断悬挂寄存器
IER	0x8	Read / Write	0x0	中断屏蔽寄存器
IAR	0xC	Write	0x0	中断响应寄存器
SIE (可选)	0x10	Write	0x0	中断允许设置寄存器
CIE (可选)	0x14	Write	0x0	中断允许清除寄存器
IVR (可选)	0x18	Read	0x0	中断类型码寄存器
MER	0x1C	Read / Write	0x0	主中断屏蔽寄存器



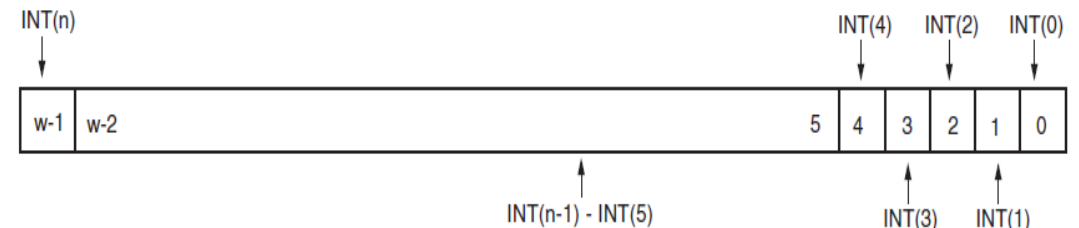
Xilinx的GPIO和INTC

► CPU能否收到模块的中断请求取决于：

- 模块的中断控制
- 中断控制器的控制
 - 则需要对IER、MER进行编程。
 - INT的API函数
 - Xil_Out(addr, data) , Xil_In()
- CPU的控制

```
+ XIntc_Initialize(XIntc*, u16) : int
+ XIntc_Start(XIntc*, u8) : int
+ XIntc_Stop(XIntc*) : void
+ XIntc_Connect(XIntc*, u8, XInterruptHandler, void*) : int
+ XIntc_Disconnect(XIntc*, u8) : void
+ XIntc_Enable(XIntc*, u8) : void
+ XIntc_Disable(XIntc*, u8) : void
+ XIntc_Acknowledge(XIntc*, u8) : void
+ XIntc_LookupConfig(u16) : XIntc_Config*
+ XIntc_ConnectFastHandler(XIntc*, u8, XFastInterruptHandler) : int
+ XIntc_SetNormalIntrMode(XIntc*, u8) : void
+ XIntc_VoidInterruptHandler(void) : void
+ XIntc_InterruptHandler(XIntc*) : void
+ XIntc_SetOptions(XIntc*, u32) : int
+ XIntc_GetOptions(XIntc*) : u32
+ XIntc_SelfTest(XIntc*) : int
+ XIntc_SimulateIntr(XIntc*, u8) : int
```

xintc.h文件中的驱动函数



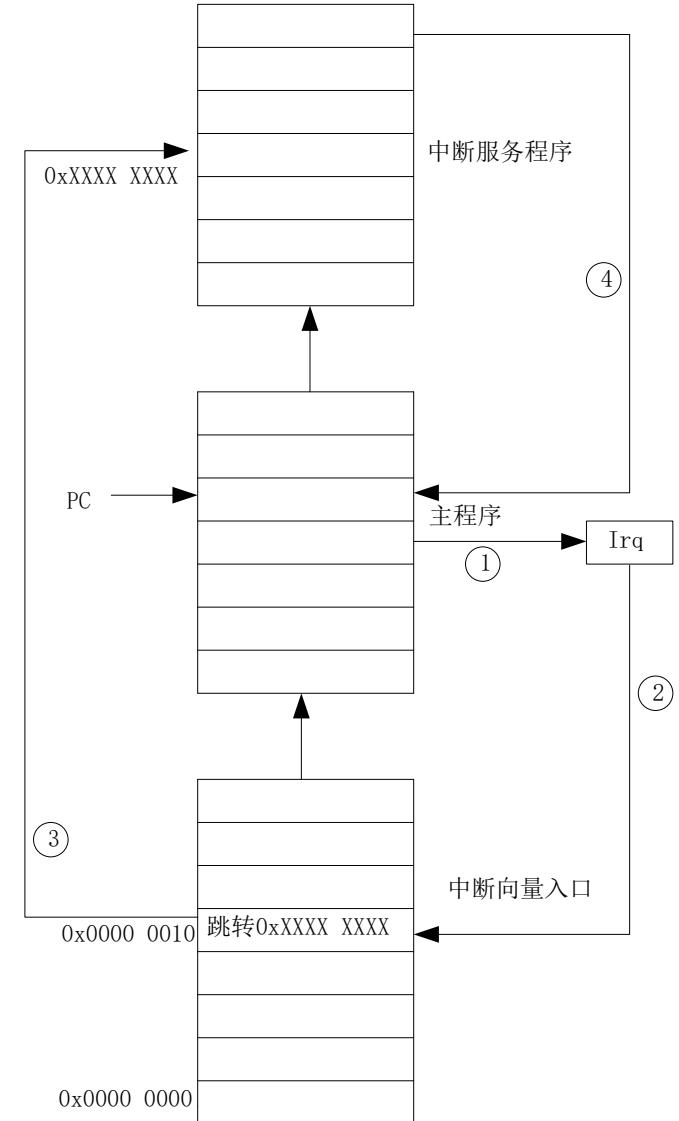
Xilinx的GPIO和INTC

► CPU能否收到模块的中断请求取决于：

- 模块的中断控制
- 中断控制器的控制
- CPU的控制
 - 则需要对MSR中的I位进行编程、需要设定中断服务函数入口。
 - CPU的API函数

mb_interface.h文件中的驱动函数

```
microblaze_enable_interrupts(void) : void
microblaze_disable_interrupts(void) : void
microblaze_enable_icache(void) : void
microblaze_disable_icache(void) : void
microblaze_enable_dcache(void) : void
microblaze_disable_dcache(void) : void
microblaze_enable_exceptions(void) : void
microblaze_disable_exceptions(void) : void
microblaze_register_handler(XInterruptHandler, void*) : void
```

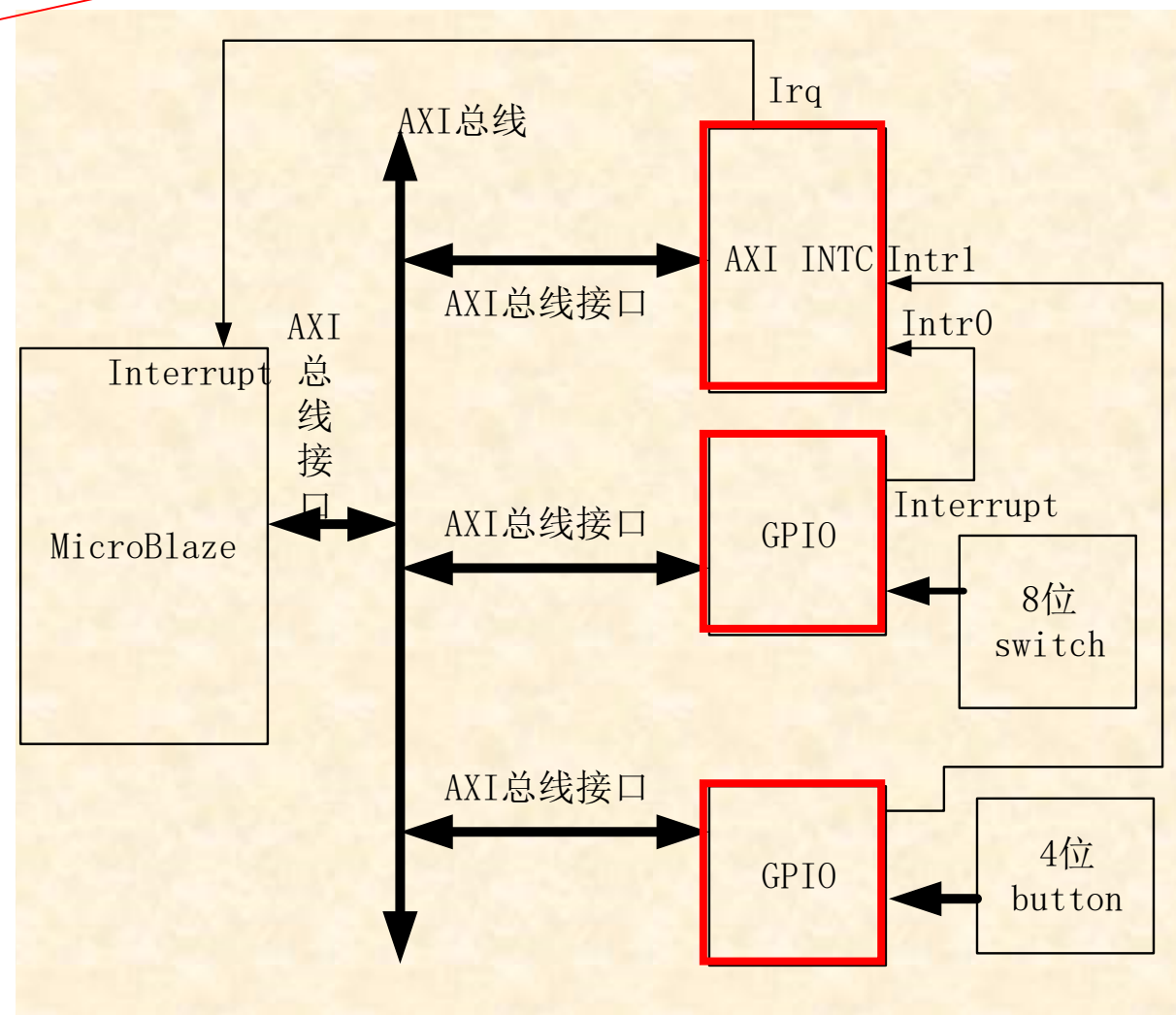


- ▶ CPU能否收到模块的中断请求取决于：
 - 模块的中断控制
 - 中断控制器的控制
 - CPU的控制
- ▶ 对模块、中断控制器、CPU的编程可以采用两种不同方法：
 - IO/INC等模块的API函数
 - 直接地址读、写(Xil_In、Xil_Out)
 - 上述两种方法详见左老师课件 “chap7——第十七讲 GPIO中断输入接口.pdf”

GPIO、AXI 的配置，模块间的连接等均通过图形菜单来完成，提高了设计效率

► 硬件电路框图

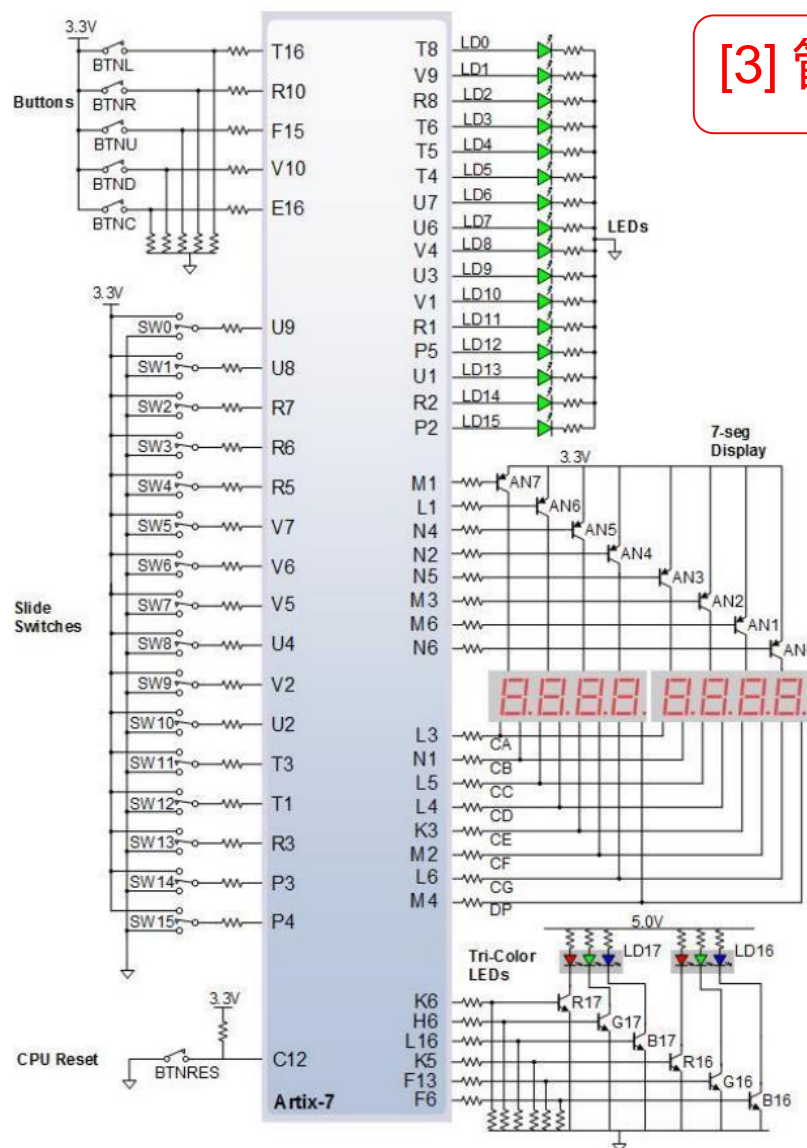
- MicroBlaze最小系统：第x章
- GPIO模块 & AXI INTC：在最小系统基础上，用Vivado软件添加



► Basis IO

- Button
- Switch
- LED
- 7-seg Dis
- Reset
- Tri-C LEDs

[1] 各种IO
会在实验中
频繁使用，
要会看管脚
分配。
[2] 本实验
中用到部分



[3] 管脚分配详见文件：Nexys4_Master.ucf

```

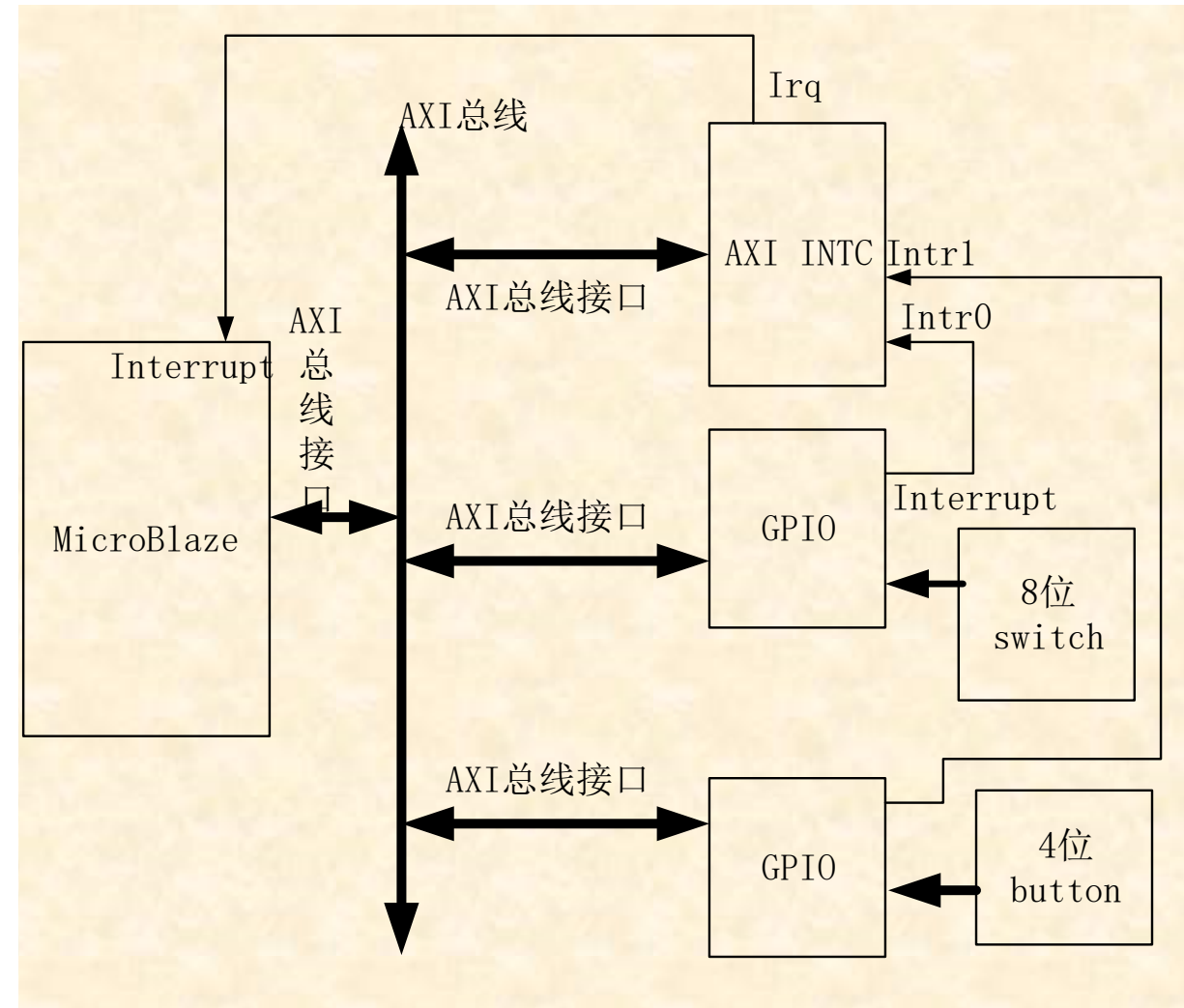
Nexys4_Master.ucf x
1  ## This file is a general .ucf for the Nexys4 rev B board
2  ## To use it in a project:
3  ## - uncomment the lines corresponding to used pins
4  ## - rename the used signals according to the project
5
6  ## Clock signal
7  #NET "clk" LOC = "E3" | IOSTANDARD = "LVCMOS33";
8  #NET "clk" TNM_NET = sys_clk_pin;
9  #TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 100 MHz HIGH 50%;
10
11 ## Switches
12 #NET "sw<0>" LOC = "U9" | IOSTANDARD = "LVCMOS33";
13 #NET "sw<1>" LOC = "U8" | IOSTANDARD = "LVCMOS33";
14 #NET "sw<2>" LOC = "R7" | IOSTANDARD = "LVCMOS33";
15 #NET "sw<3>" LOC = "R6" | IOSTANDARD = "LVCMOS33";
16 #NET "sw<4>" LOC = "R5" | IOSTANDARD = "LVCMOS33";
17 #NET "sw<5>" LOC = "V7" | IOSTANDARD = "LVCMOS33";
18 #NET "sw<6>" LOC = "V6" | IOSTANDARD = "LVCMOS33";
19 #NET "sw<7>" LOC = "V5" | IOSTANDARD = "LVCMOS33";
20 #NET "sw<8>" LOC = "U4" | IOSTANDARD = "LVCMOS33";
21 #NET "sw<9>" LOC = "V2" | IOSTANDARD = "LVCMOS33";
    
```


► 使用SDK软件，设计应用程序(详见13.5.1)

- 由MicroBlaze CPU执行
- 管理AXI、GPIO模块，实现输入、中断等功能

► 函数接口

- 可以基于已有驱动提供的API
- 也可以直接通过libc提供的输入输出语句对硬件进行直接控制



实验书例子采用，需要知道这些函数的具体含义。程序模块化程度高。

▶ 函数接口

- 可以基于已有驱动提供的API —— GPIO位于头文件`xgpio.h`中，INTC位于头文件`xintc.h`中，CPU位于头文件`mb_interface.h`中

```

+ XGpio_Initialize(XGpio*, u16) : int
+ XGpio_LookupConfig(u16) : XGpio_Config*
+ XGpio_CfgInitialize(XGpio*, XGpio_Config*, u32) : int
+ XGpio_SetDataDirection(XGpio*, unsigned, u32) : void
+ XGpio_GetDataDirection(XGpio*, unsigned) : u32
+ XGpio_DiscreteRead(XGpio*, unsigned) : u32
+ XGpio_DiscreteWrite(XGpio*, unsigned, u32) : void
+ XGpio_DiscreteSet(XGpio*, unsigned, u32) : void
+ XGpio_DiscreteClear(XGpio*, unsigned, u32) : void
+ XGpio_SelfTest(XGpio*) : int
+ XGpio_InterruptGlobalEnable(XGpio*) : void
+ XGpio_InterruptGlobalDisable(XGpio*) : void
+ XGpio_InterruptEnable(XGpio*, u32) : void
+ XGpio_InterruptDisable(XGpio*, u32) : void
+ XGpio_InterruptClear(XGpio*, u32) : void
+ XGpio_InterruptGetEnabled(XGpio*) : u32
+ XGpio_InterruptGetStatus(XGpio*) : u32
    
```

xgpio.h文件中的驱动函数

```

+ XIntc_Initialize(XIntc*, u16) : int
+ XIntc_Start(XIntc*, u8) : int
+ XIntc_Stop(XIntc*) : void
+ XIntc_Connect(XIntc*, u8, XInterruptHandler, void*) : int
+ XIntc_Disconnect(XIntc*, u8) : void
+ XIntc_Enable(XIntc*, u8) : void
+ XIntc_Disable(XIntc*, u8) : void
+ XIntc_Acknowledge(XIntc*, u8) : void
+ XIntc_LookupConfig(u16) : XIntc_Config*
+ XIntc_ConnectFastHandler(XIntc*, u8, XFastInterruptHandler) : int
+ XIntc_SetNormalIntrMode(XIntc*, u8) : void
+ XIntc_VoidInterruptHandler(void) : void
+ XIntc_InterruptHandler(XIntc*) : void
+ XIntc_SetOptions(XIntc*, u32) : int
+ XIntc_GetOptions(XIntc*) : u32
+ XIntc_SelfTest(XIntc*) : int
+ XIntc_SimulateIntr(XIntc*, u8) : int
    
```

xintc.h文件中的驱动函数

mb_interface.h文件中的驱动函数

```

+ microblaze_enable_interrupts(void) : void
+ microblaze_disable_interrupts(void) : void
+ microblaze_enable_icache(void) : void
+ microblaze_disable_icache(void) : void
+ microblaze_enable_dcache(void) : void
+ microblaze_disable_dcache(void) : void
+ microblaze_enable_exceptions(void) : void
+ microblaze_disable_exceptions(void) : void
+ microblaze_register_handler(XInterruptHandler, void*) : void
    
```

► 函数接口

左老师视频中有采用，直接对硬件进行操作，简洁，需要知道端口地址。

- 也可以直接通过libc提供的输入输出语句对硬件进行直接控制 —— 位于xil_io.h中

```

/*****
**
** Perform an input operation for an 8-bit memory location by reading from the
** specified address and returning the value read from that address.
**
** @param Addr contains the address to perform the input operation at.
** @return The value read from the specified input address.
** @note None.
**
*****/
#define Xil_In8(Addr) (*(volatile u8 *)(Addr))

/*****
**
** Perform an input operation for a 16-bit memory location by reading from the
** specified address and returning the value read from that address.
**
** @param Addr contains the address to perform the input operation at.
** @return The value read from the specified input address.
** @note None.
**
*****/
#define Xil_In16(Addr) (*(volatile u16 *)(Addr))

/*****
**
** Perform an input operation for a 32-bit memory location by reading from the
** specified address and returning the value read from that address.
**
** @param Addr contains the address to perform the input operation at.
** @return The value read from the specified input address.
** @note None.
**
*****/
#define Xil_In32(Addr) (*(volatile u32 *)(Addr))

```

```

/*****
**
** Perform an output operation for an 8-bit memory location by writing the
** specified value to the specified address.
**
** @param Addr contains the address to perform the output operation at.
** @param value contains the value to be output at the specified address.
** @return None
** @note None.
**
*****/
#define Xil_Out8(Addr, Value) \
    (*(volatile u8 *)(Addr)) = (Value)

/*****
**
** Perform an output operation for a 16-bit memory location by writing the
** specified value to the specified address.
**
** @param Addr contains the address to perform the output operation at.
** @param value contains the value to be output at the specified address.
** @return None
** @note None.
**
*****/
#define Xil_Out16(Addr, Value) \
    (*(volatile u16 *)(Addr)) = (Value)

/*****
**
** Perform an output operation for a 32-bit memory location by writing the
** specified value to the specified address.
**
** @param Addr contains the address to perform the output operation at.
** @param value contains the value to be output at the specified address.
** @return None
** @note None.
**
*****/
#define Xil_Out32(Addr, Value) \
    (*(volatile u32 *)(Addr)) = (Value)

```

► 查询程序设计思路

- 主程序不停的读取GPIO的ISR寄存器，当其对应的位为1时，读取GPIO的数据寄存器并输出到console（xil_printf函数实现，stdio.h），并写ISR相应位
 - XIL_IN
 - XIL_OUT

► 中断程序设计思路（详见实验书&& “chap7——第十七讲 GPIO中断输入接口.pdf”）

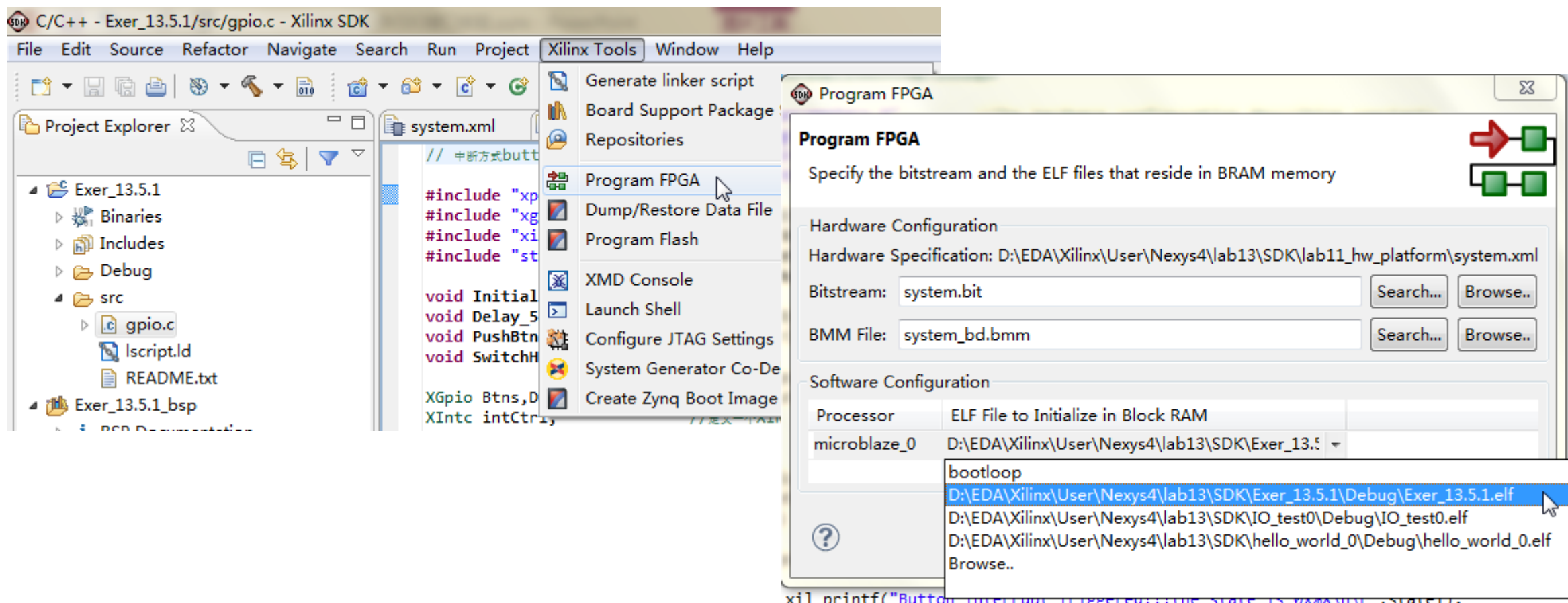
- 主程序开放microblaze，INTC，GPIO中断，不停的检测输出标志是否为1，是则输出数据到console，并将输出标志设置为0
- 中断服务程序读取数据（或输出数据）并设立输出标志位为1

► 延时方式

- For循环
 - For(i=0;i<constant;i++);
 - Constant的值决定延时的时间长短

► 测试步骤

- 将硬件设计目标文件（.bit和.bmm）和应用软件目标文件（.elf）下载至Nexys4开发板，验证功能是否实现；若功能不正确，则进行：软件Debug or 硬件检查。



► 查询程序代码(参考)

```

1 // 查询方式button按键以及switch输入的测试程序
2 #include "xparameters.h" //The hardware configuration describing constants
3 #include "stdio.h"
4 #include "xil_io.h" // IO functions
5 #include "xil_types.h"
6
7 #define btn_DATA 0x40000000 // button 数据寄存器地址
8 #define btn_TRI 0x40000004 // button 控制寄存器地址
9 #define btn_ISR 0x40000120 // button 中断状态寄存器地址
10
11 #define sw_DATA 0x40040000 // switch 数据寄存器地址
12 #define sw_TRI 0x40040004 // switch 控制寄存器地址
13 #define sw_ISR 0x40040120 // switch 中断状态寄存器地址
14
15 // 按键标志位
16 short pshBtn, pshSw;
17
18 int main(void)
19 {
20     short btn, sw;
21     xil_printf("\r\nRunning GpioInput Test(Poll)!\r\n");
22     Xil_Out8(btn_TRI, 0xff); // btn is used as input
23     Xil_Out8(sw_TRI, 0xff); // sw is used as input
24     pshBtn = 0x00;
25     pshSw = 0x00;
26     while(1)
27     {
28         // ... (code continues)
29     }
30     return 0;
31 }

```

```

26 while(1)
27 {
28     pshBtn = Xil_In8(btn_ISR);
29     pshSw = Xil_In8(sw_ISR);
30     if(pshBtn) //若按下按键, 则打印相关信息
31     {
32         btn = Xil_In8(btn_DATA);
33         Xil_Out8(btn_ISR, 0x01);
34         xil_printf("Button Pushed!!!the state is 0x%X\n\r", btn);
35     }
36     if(pshSw) //若拨动Switch开关, 则打印相关信息
37     {
38         sw = Xil_In8(sw_DATA);
39         Xil_Out8(sw_ISR, 0x01);
40         xil_printf("Switch Pushed!!!the state is 0x%X\n\r", sw);
41     }
42 }

```


► 中断程序 (No API) 代码(参考)

```
1 // 中断方式button按键以及switch输入的测试程序
2
3 #include "xparameters.h" //The hardware configuration describing constants
4 #include "xintc.h" //Interrupt Controller API functions
5 #include "stdio.h"
6 #include "xil_io.h" // IO functions
7 #include "xil_types.h"
8 #include "mb_interface.h"
9 #include "xgpio.h" //GPIO API functions
10
11 #define btn_DATA 0x40000000 // button 数据寄存器地址
12 #define btn_TRI 0x40000004 // button 控制寄存器地址
13 #define btn_GIER 0x4000011c // button 中断全局允许寄存器地址
14 #define btn_IER 0x40000128 // button 中断通道允许寄存器地址
15 #define btn_ISR 0x40000120 // button 中断状态寄存器地址
16
17 #define sw_DATA 0x40040000 // switch 数据寄存器地址
18 #define sw_TRI 0x40040004 // switch 控制寄存器地址
19 #define sw_GIER 0x4004011c // switch 中断全局允许寄存器地址
20 #define sw_IER 0x40040128 // switch 中断通道允许寄存器地址
21 #define sw_ISR 0x40040120 // switch 中断状态寄存器地址
22
23 #define intc_ISR 0x41200000
24 #define intc_IER 0x41200008
25 #define intc_IAR 0x4120000C
26 #define intc_MER 0x4120001C
27
28 // 注册总中断服务程序地址
29 void My_ISR (void) __attribute__((interrupt_handler));
30
31 void Initialize(); //初始化函数 (包含中断初始化)
```



► 中断程序 (No API) 代码(参考) — 续

```
32 void PushBtnHandler();           //按键的处理函数
33 void SwitchHandler();           //拨动开关的处理函数
34 void Delay_50ms();              //延时函数
35
36 short flag_Sw, flag_Btn;         // 按键标志位
37 short sw, btn;                  // 按键键值
38
39 int main(void)
40 {
41     xil_printf("\r\nRunning GpioInput Interrupt Test(No APP)!\r\n");
42     Initialize();
43     while(1)
44     {
45         if(flag_Sw)               //若拨动Switch开关, 则打印相关信息
46         {
47             xil_printf("Switch Interrupt Triggered!!!the result is 0x%X\n\r", sw);
48             flag_Sw=0;
49         }
50         if(flag_Btn)              //若按下按键, 则打印相关信息
51         {
52             xil_printf("Button Interrupt Triggered!!!the result is 0x%X\n\r", btn);
53             flag_Btn=0;
54         }
55         int status;
56         status = Xil_In32(intc_ISR); // 读取ISR
57         status = Xil_In32(sw_ISR);   // 读取ISR
58         status = Xil_In32(btn_ISR);  // 读取ISR
59     }
60     return 0;
61 }
62 }
```


► 中断程序 (No API) 代码(参考) — 续

```
63
64 void Initialize()
65 {
66     flag_Sw = 0x00;
67     flag_Btn = 0x00;
68     sw = 0x00;
69     btn = 0x00;
70
71     Xil_Out8(sw_TRI, 0xff);           // sw is used as input
72     Xil_Out8(sw_IER, 0x01);          // channel 0 inter enable
73     Xil_Out32(sw_GIER, 0x80000000);  // sw Interrupt enable
74
75     Xil_Out8(btn_TRI, 0xff);          // btn is used as input
76     Xil_Out8(btn_IER, 0x01);          // channel 0 inter enable
77     Xil_Out32(btn_GIER, 0x80000000);  // btn Interrupt enable
78
79     Xil_Out32(intc_IAR, 0xffffffff);  // claer all irq requests
80     Xil_Out32(intc_IER, 0x03);        // Intr[1:0] Interrupt enable
81     Xil_Out32(intc_MER, 0x03);        // INTC Interrupt enable
82
83     microblaze_enable_interrupts();   // CPU interrupt enable
84 }
```

► 中断程序 (No API) 代码(参考) — 续

```

85
86 void My_ISR(void)
87 {
88     int status;
89     status = Xil_In32(intc_ISR);           // 读取ISR
90     if(status & 0x02)                     // ISR[1]=1, 说明是Switch中断
91     {
92         SwitchHandler();
93     }
94     if(status & 0x01)                     // ISR[0]=1, 说明是PshButton中断
95     {
96         PushBtnHandler();
97     }
98
99     Xil_Out32(intc_IAR, status);          // 写IAR清INTC中断标志
100 }
101
102 void SwitchHandler()
103 {
104     sw = Xil_In8(sw_DATA);               // 读取Switch开关的状态值
105     flag_Sw=1;
106     int isr_status;
107     isr_status = Xil_In32(sw_ISR);
108     Xil_Out32(sw_ISR, 0x01);             // 清除中断标志位
109 }

```

```

110
111 void PushBtnHandler()
112 {
113
114     btn = Xil_In8(btn_DATA);             // 读取Switch开关的状态值
115     flag_Btn=1;
116     Xil_Out8(btn_IER, 0x00);             // channel 0 inter disable
117     Delay_50ms();
118     int isr_status;
119     isr_status = Xil_In32(btn_ISR);
120     Xil_Out32(btn_ISR, 0x01);            // 清除中断标志位
121     Xil_Out8(btn_IER, 0x01);            // channel 0 inter enable
122 }
123
124 void Delay_50ms()
125 {
126     int i;
127     for(i=0; i<5000000; i++);
128 }

```

Thanks

