第7章 中断技术

微处理器中断处理系统

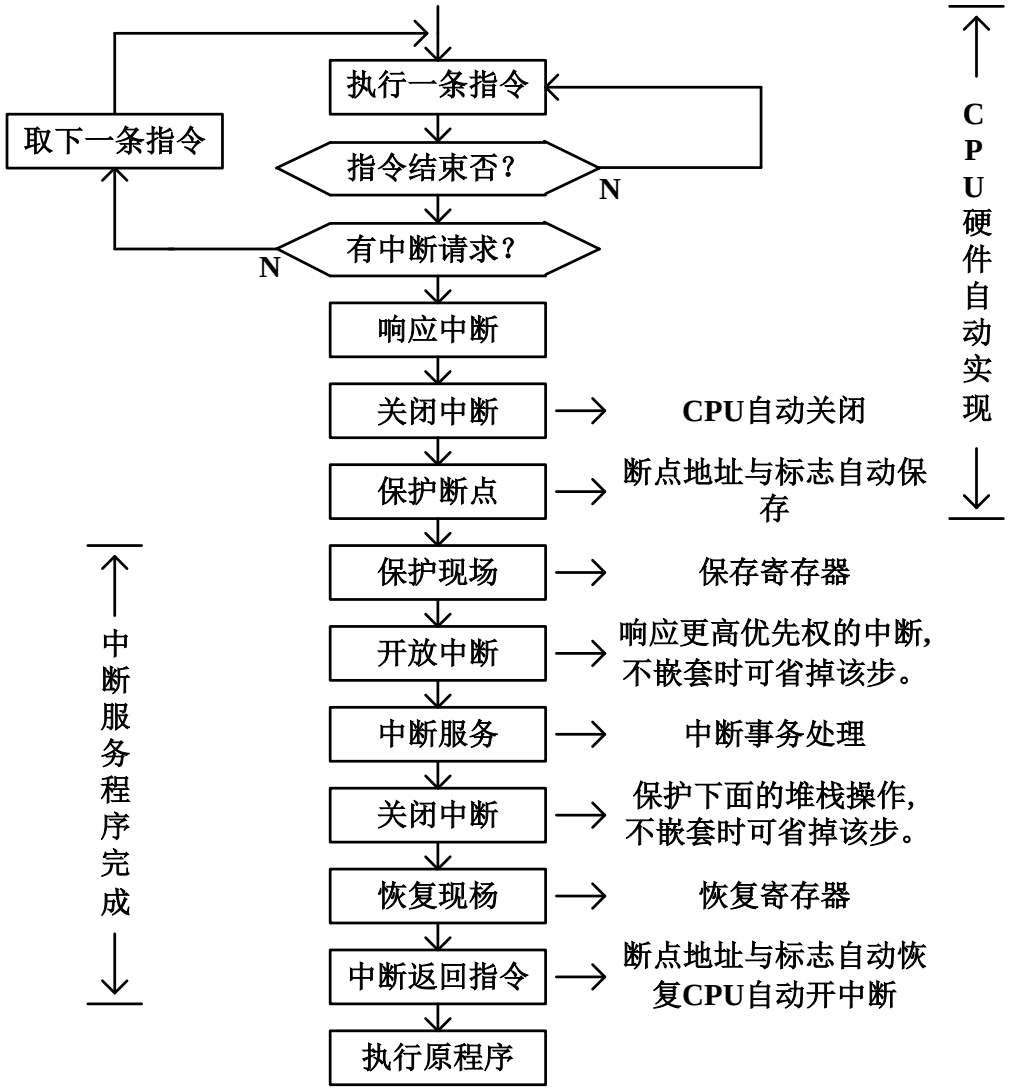
GPIO中断示例

华中科技大学
电子信息与通信学院
School of Electronic Information and Communications

学习目标

- 了解80x86微处理器实模式下中断处理过程（自学）
- 理解MicroBlaze微处理器的中断处理过程
- 熟悉MicroBlaze微处理器AXI总线的中断方式接口设计
- 熟悉MicroBlaze中断处理程序设计
- 掌握GPIO中断接口电路和软件设计

微处理器响应中断的一般过程

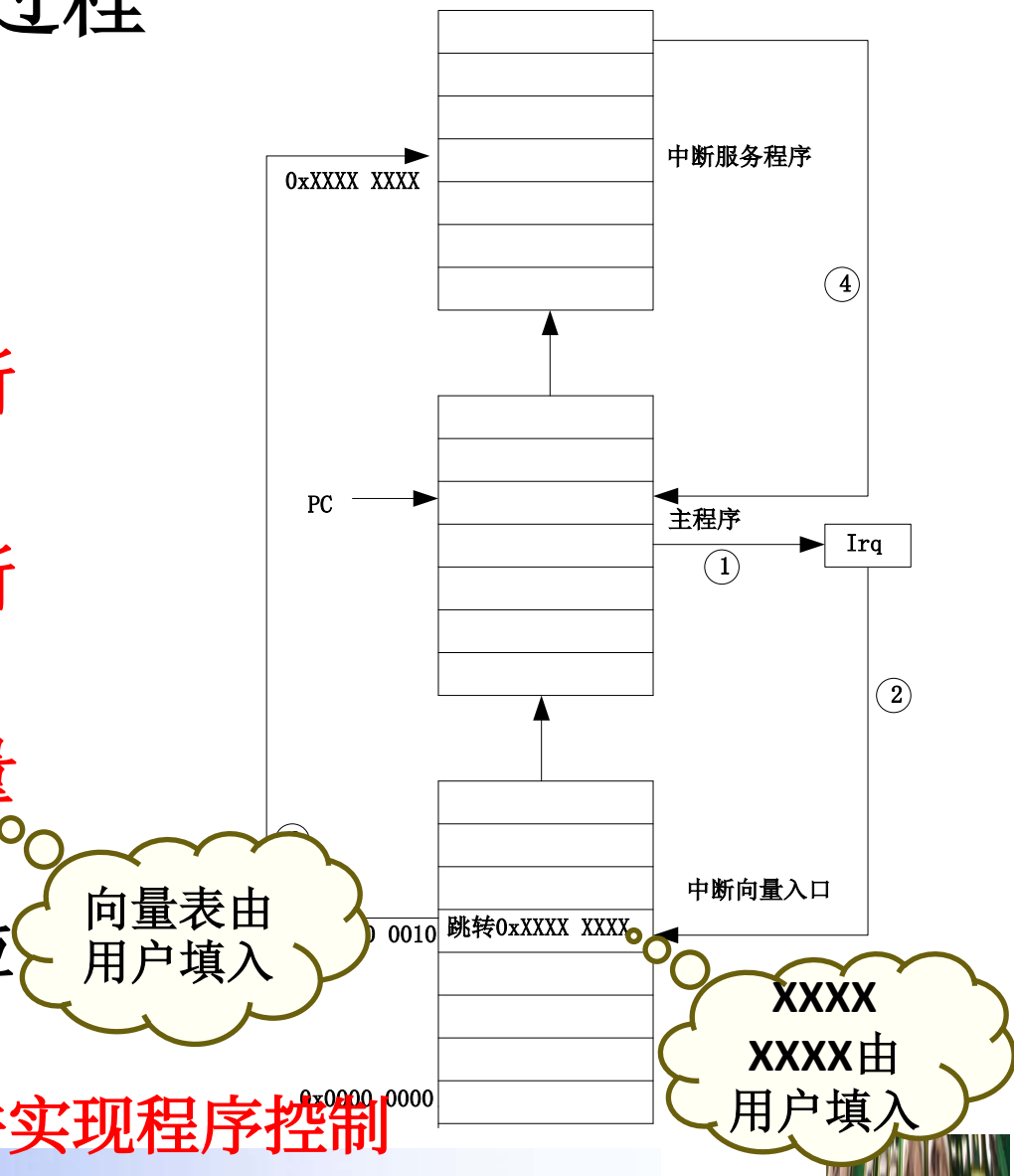


MicroBlaze中断系统介绍_中断源分类

中断类型	中断向量地址	保存断点的寄存器
复位	0x00000000-0x00000007	-
用户异常	0x00000008-0x0000000F	-
中断	0x00000010-0x00000017	R14
不可屏蔽硬件中断	0x00000018-0x0000001F	R16
硬件打断（break）		
软件打断（break）		
硬件异常	0x00000020-0x00000027	R17

MicroBlaze中断处理过程

- 主中断服务程序
- 不同种类中断源识别，读取中断控制器的中断请求状态寄存器，
- 调用特定中断源的中断服务程序
- 软件维护一个中断向量表（中断向量数组），并根据中断源查找相应中断服务程序



软件识别不同外部中断源，并实现程序控制

中断方式接口程序设计

- 中断程序构成

不同系统常驻内存的实现方式不同

断点保护，标志寄存器，通用寄存器保护

恢复断点，标志寄存器，通用寄存器

先后顺序可互换

开放特定中断请求，设定中断优先级

中断服务程序常驻内存

将中断向量填入正确的中断向量表地址

保护现场

中断处理

清除中断请求寄存器

恢复现场

中断返回指令

屏蔽特定中断请求

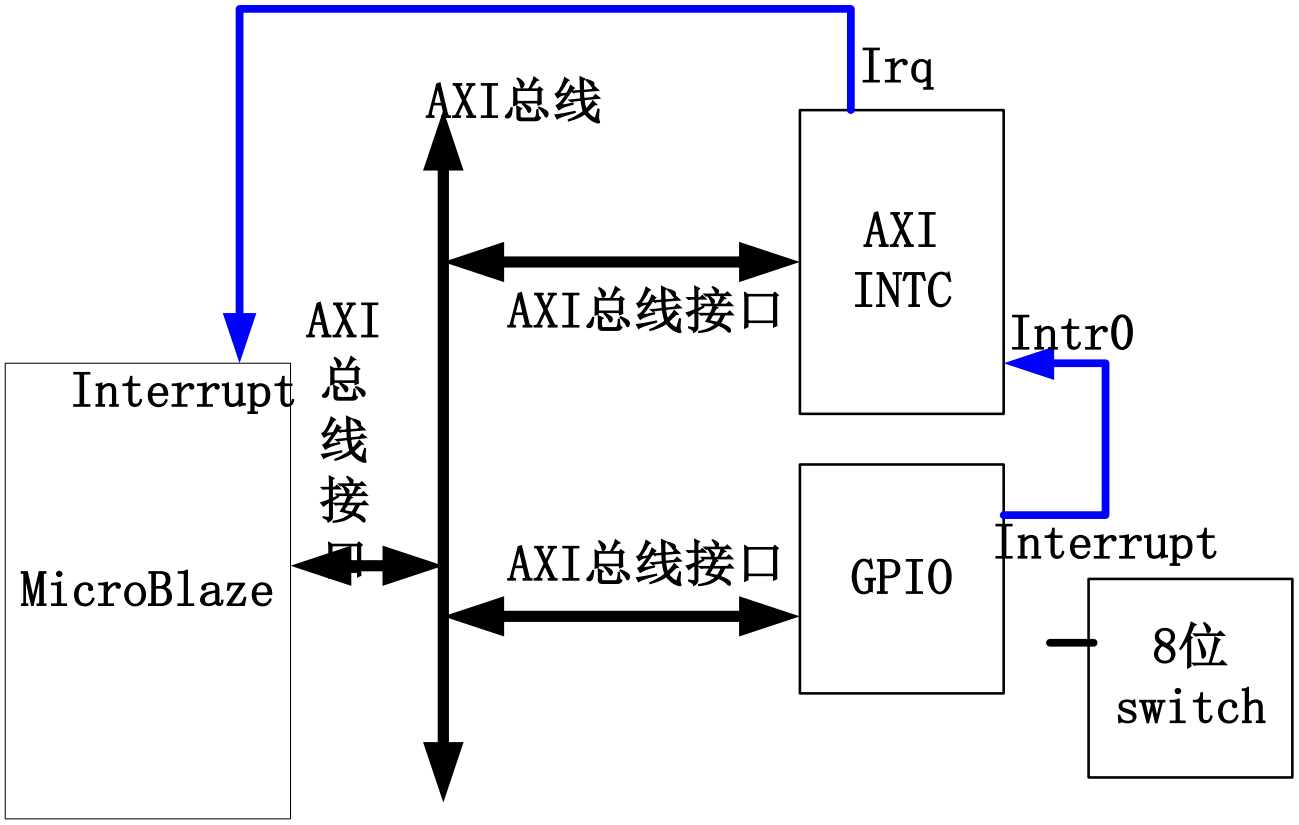
(A)启用中断传送

(B)中断服务程序

(C) 停止中断传送

设计实例一

- 中断方式实现8个开关状态输入——硬件电路



AXI GPIO并行接口控制器中断原理简介

- GPIO内部中断相关寄存器

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

GPIO中断产生逻辑模块当检测到GPIO_DATA_IN输入数据发生变化时，就可以产生中断信号，但是是否输出中断信号，受中断允许控制寄存器控制。中断控制逻辑与AXI INTC类似，但是没有优先级判断，仅2个中断源。

程序设计——不采用INTC,GPIO API

- 开放中断
  - MicroBlaze IE开放

microblaze_enable_interrupts();

  - INTC 总中断输出开放

Xil_Out32(INTC_BASEADDR+XIN_MER_OFFSET,0x3);

  - INTC 某个INTR引脚输入开放(INTR0)

Xil_Out32(INTC_BASEADDR+XIN_IER_OFFSET,0x1);

  - GPIO总中断输出开放

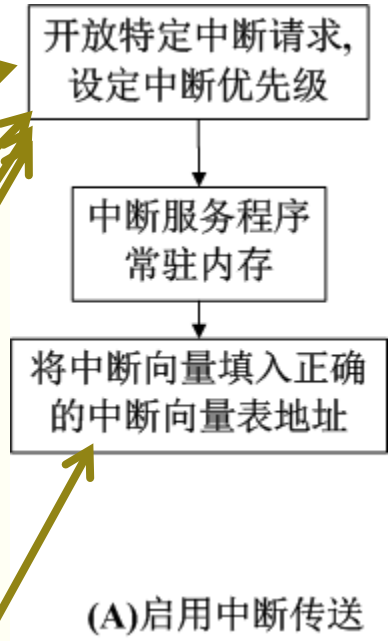
Xil_Out32(BUTTON_BASEADDR+XGPIO_GIE_OFFSET,0x80000000);

  - GPIO某个通道中断开放

Xil_Out32(BUTTON_BASEADDR+XGPIO_IER_OFFSET,0x1);

- 填写中断向量表
  - 注册总中断服务程序地址

void My_ISR() __attribute__ ((interrupt_handler));



软件设计——读写寄存器

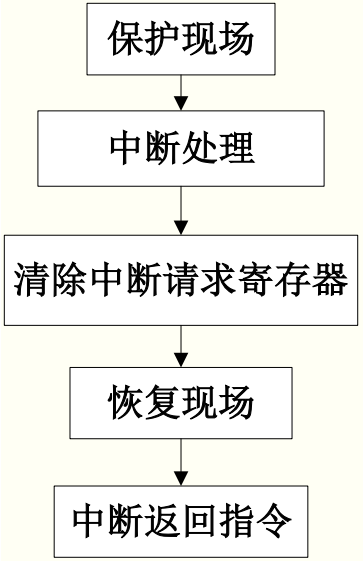
- 中断服务程序
  - 总中断服务程序
    - 读取INTC 中断状态寄存器

      status=Xil_In32(INTC_BASEADDR+XIN_ISR_OFFSET);
    - 识别中断源并调用用户中断服务程序
    - 重复步骤2直到所有的中断处理完毕
    - 清除INTC中断请求状态

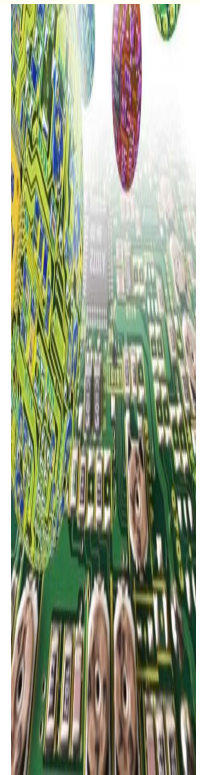
      Xil_Out32(INTC_BASEADDR+XIN_IAR_OFFSET,status);
  - 用户中断服务程序
    - 读取开关值

      Xil_In32(DIP_BASEADDR+XGPIO_DATA_OFFSET);
    - 清除GPIO中断

      Xil_Out32(DIP_BASEADDR+XGPIO_ISR_OFFSET,1);



(B)中断服务程序



软件源码——读写寄存器

```
#include "xil_io.h"
#include "stdio.h"
#define INTC_BASEADDR 0x41200000
#define XIN_ISR_OFFSET      0   /* Interrupt Status Register */
#define XIN_IER_OFFSET      8   /* Interrupt Enable Register */
#define XIN_IAR_OFFSET      12  /* Interrupt Acknowledge Register */
#define XIN_MER_OFFSET      28  /* Master Enable Register */
#define DIP_BASEADDR 0x40040000
#define XGPIO_DATA_OFFSET       0x0   /**< Data register for 1st channel */
#define XGPIO_TRI_O
#define XGPIO_GIE_O
#define XGPIO_ISR_O
#define XGPIO_IER_O
void SwitchHandler(
void My_ISR()  __att
int pshDip;//作为中
int state2;//存储Di
int main()
{
        Xil_Out32(DIP_BASEADDR+XGPIO_TRI_OFFSET,0xff);//设定Dips为输入方式
        Xil_Out32(DIP_BASEADDR+XGPIO_IER_OFFSET,0x1);//DIP GPIO中断使能
        Xil_Out32(DIP_BASEADDR+XGPIO_GIE_OFFSET,0x80000000);
        Xil_Out32(INTC_BASEADDR+XIN_IER_OFFSET,0x3);//对中断控制器进行中断源使能
        Xil_Out32(INTC_BASEADDR+XIN_MER_OFFSET,0x3);
        microblaze_enable_interrupts();//允许处理器处理中断
        while(1)
        {
                if(pshDip)//若拨动Switch开关，则打印相关信息
                {
                        xil_printf("Switch Interrupt Triggered!!!the state is 0x%X\n\r",state2);
                        pshDip=0;
                }
        }
        return 0;
}
void My_ISR()
{       int status;
        status=Xil_In32(INTC_BASEADDR+XIN_ISR_OFFSET);//读取ISR
        if((status&0x1)==0x1)
                SwitchHandler();//调用用户中断服务程序
        Xil_Out32(INTC_BASEADDR+XIN_IAR_OFFSET,status);//写IAR

}
void SwitchHandler()
{

        state2=Xil_In32(DIP_BASEADDR+XGPIO_DATA_OFFSET);//读取Switch开关的状态值
        pshDip=1;
        Xil_Out32(DIP_BASEADDR+XGPIO_ISR_OFFSET,1);//清除中断标志位
}
```

standalone操作系统中断相关API

- MicroBlaze微处理器中断系统调用
  - void microblaze_enable_interrupts(void)
    - 该函数的功能是使得MSR中的IE位为1，允许MicroBlaze微处理器响应外部中断。
  - void microblaze_disable_interrupts(void)
    - 该函数的功能是使得MSR中的IE位为0，禁止MicroBlaze微处理器响应外部中断。
  - void microblaze_register_handler(XInterruptHandler Handler, void *DataPtr)
    - 该函数的功能是将中断控制器主中断服务程序的地址与跳转指令结合后填入中断向量地址0x0000 0010处，并且将中断服务程序需要处理的参数地址传给中断服务程序。
    - void handler() __attribute__ ((interrupt_handler));

- AXI INTC中断控制器驱动API

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_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

AXI INTC中断控制器驱动相关数据结构

```
typedef struct {
XInterruptHandler Handler;//中断服务程序入口地址
void *CallBackRef;// 中断服务程序参数地址
} XIntc_VectorTableEntry;
```

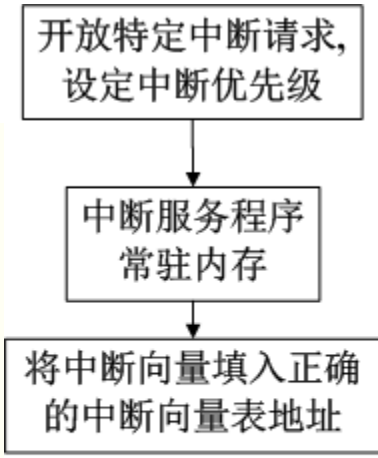
```
typedef struct {
u16 DeviceId;                    //操作系统维护的设备ID号
u32 BaseAddress;   //设备对应的基地址
u32 AckBeforeService;//何时写中断响应寄存器选项
u32 Options;                     //总中断服务程序处理方式
XIntc_VectorTableEntry HandlerTable[XPAR_INTC_MAX_NUM_INTR_INPUTS]; //中断向量表
} XIntc_Config;
```

```
typedef struct {
u32 BaseAddress;    //控制器基地址
u32 IsReady;                     // 控制器是否已初始化标志
u32 IsStarted;        //控制器是否已开启标志
u32 UnhandledInterrupts; // 统计未处理的中断请求数目
XIntc_Config *CfgPtr;            //中断控制器配置项
} XIntc;
```

GPIO API

- 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

软件设计



(A)启用中断传送

- 开放中断
  - MicroBlaze IE开放

  microblaze_enable_interrupts();
  - INTC 总中断输出开放

  XIntc_Start(XIntc * InstancePtr, XIN_REAL_MODE)
  - INTC 某个IR引脚输入开放

  XIntc_Enable(XIntc * InstancePtr, u8 Id)
  - GPIO总中断输出开放

  XGpio_InterruptGlobalEnable(XGpio * InstancePtr)
  - GPIO某个通道中断开放

  XGpio_InterruptEnable(XGpio * InstancePtr, u32 Mask)
- 填写中断向量表
  - 注册INTC总中断服务程序地址

  microblaze_register_handler(XInterruptHandler Handler, void *DataPtr)
  - 注册GPIO中断服务程序地址

  XIntc_Connect(XIntc * InstancePtr, u8 Id, XInterruptHandler Handler, void *CallBackRef)

软件设计

- 中断服务程序
  - 总中断服务程序

XIntc_InterruptHandler(XIntc * InstancePtr)

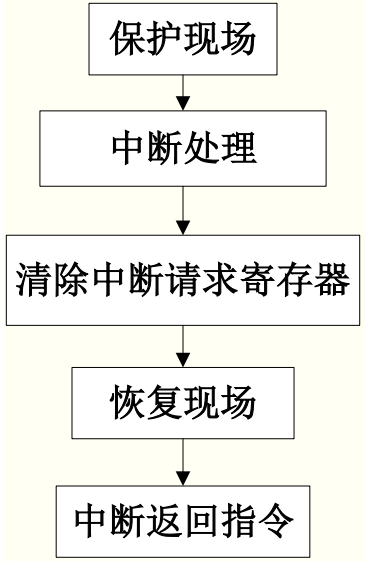
* This function is the primary interrupt handler for the driver. It must be
* connected to the interrupt source such that is called when an interrupt of
* the interrupt controller is active. It will resolve which interrupts are
* active and enabled and call the appropriate interrupt handler. It uses
* the AckBeforeService flag in the configuration data to determine when to
* acknowledge the interrupt. Highest priority interrupts are serviced first.

  - 用户中断服务程序
    - 读取开关值

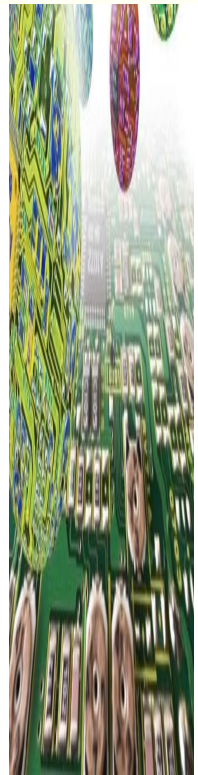
XGpio_DiscreteRead(XGpio * InstancePtr, unsigned Channel)

    - 清除GPIO中断

XGpio_InterruptClear(XGpio * InstancePtr, u32 Mask)



(B)中断服务程序



软件设计——其他

- 结构体初始化
  - 中断控制器

  XIntc_Initialize(XIntc * InstancePtr, u16 DeviceId)
  - GPIO

  XGpio_Initialize(XGpio * InstancePtr, u16 DeviceId)
- GPIO输入方向控制

  XGpio_SetDataDirection(XGpio * InstancePtr, unsigned Channel, u32 DirectionMask)

程序源码

```
#include "xparameters.h"  //The hardware configuration describing constants
#include "xgpio.h"       //GPIO API functions
#include "xintc.h"//Interrupt Controller API functions
#include "stdio.h"
void SwitchHandler(void *CallBackRef);//拨动开关的中断服务程序
XGpio Dips;//定义GPIO外设变量
XIntc intCtrl;//定义XINTC外设变量
int pshDip;//作为中断标志
int main()
{
        XGpio_Initialize(&Dips,XPAR_DIP_DEVICE_ID);//初始化Dips实例，并设定其为输入方式
        XGpio_SetDataDirection (&Dips, 1, 0xff);
        XIntc_Initialize(&intCtrl,XPAR_AXI_INTC_0_DEVICE_ID );//初始化intCtrl实例
        XGpio_InterruptEnable(&Dips, 1);//GPIO中断使能
        XGpio_InterruptGlobalEnable(&Dips);
        XIntc_Enable(&intCtrl,XPAR_AXI_INTC_0_DIP_IP2INTC_IRPT_INTR);//对中断控制器进行中断源使能
        XIntc_Connect(&intCtrl,XPAR_AXI_INTC_0_DIP_IP2INTC_IRPT_INTR,//注册用户中断服务函数
                      (XInterruptHandler)SwitchHandler,(void *)0);
        microblaze_enable_interrupts();//允许处理器处理中断
        //注册总中断处理函数
        microblaze_register_handler((XInterruptHandler)XIntc_InterruptHandler, (void *)&intCtrl);
        XIntc_Start(&intCtrl, XIN_REAL_MODE);//启动中断控制器，开放总中断
        xil_printf("\r\nRunning GpioInputInterrupt!\r\n");
        while(1)
        {
           if(pshDip)//若拨动Switch开关，则打印相关信息
                {
                xil_printf("Switch Interrupt Triggered!!!the state is 0x%X\n\r",state2);
                pshDip=0;
                }
        }
        return 0;
}

void SwitchHandler(void *CallBackRef)
{

        state2=XGpio_DiscreteRead(&Dips,1);//读取Switch开关的状态值
        pshDip=1;
        XGpio_InterruptClear(&Dips,1);//清除中断标志位
}
```

作业

- 9
  - 采用API和读写寄存器两种方式实现pushbutton按键的输入
  - 由于pushbutton按键按下时会马上反弹，因此会产生多次中断。避免多次中断的方式之一：进入中断后，在中断服务程序里马上屏蔽GPIO中断（IER)，读入按键值，延时一段时间（50ms）后，在退出中断服务程序前再开放中断.