第7章 中断技术

定时器应用

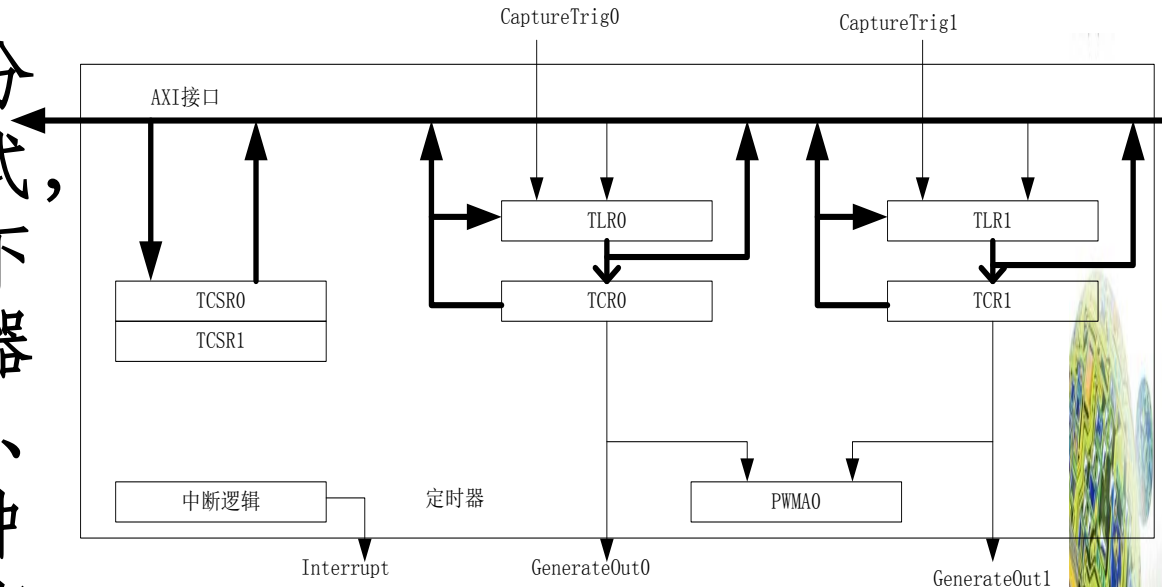
华中科技大学
电子信息与通信学院
School of Electronic Information and Communications

学习目标

- 掌握硬件时钟中断原理和应用

AXI Timer定时器简介

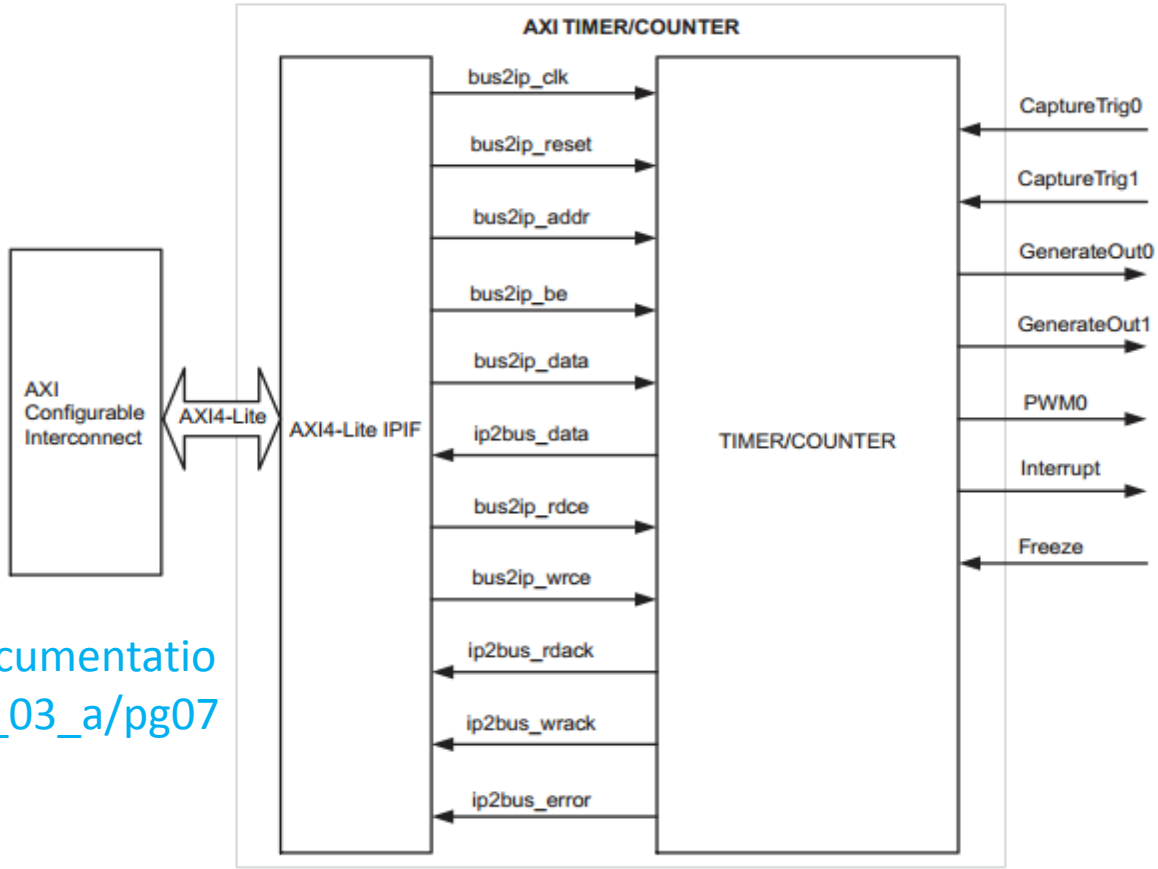
- 定时模式可以分为两种计数方式，向上计数和向下计数。该定时器可以工作在8位、16位、32位三种模式，采用小字节序



两个定时器中的任意一个计数结束都会产生中断，如果允许输出中断信号，则使Interrupt输出高电平，从而产生中断请求。

定时计数器的其他功能

- 产生周期性的脉冲信号（generate模式）
- 统计外部周期信号的周期（capture模式）
- 产生PWM波



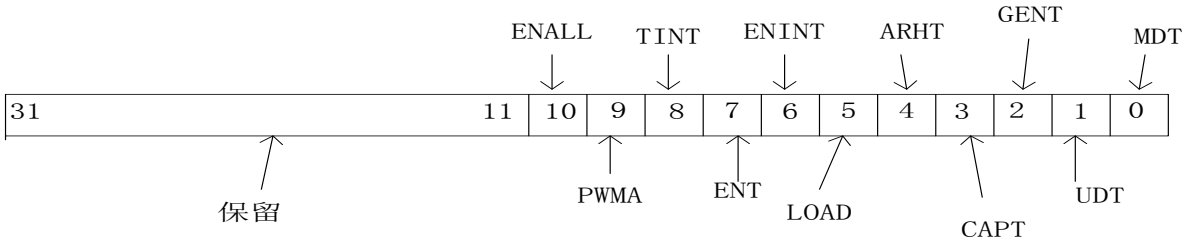
http://www.xilinx.com/support/documentation/ip_documentation/axi_timer/v1_03_a/pg079-axi-timer.pdf

产生中断信号的时间间隔

- 加计数：T=(TCRmax-TLR)*AXI_CLK_PERIOD
- 减计数：T=TLR* AXI_CLK_PERIOD

寄存器名称	偏移地址	功能描述	寄存器名称	偏移地址	功能描述
TCSR0	0x00	定时器0控制寄存器	TCSR1	0x10	定时器1控制寄存器
TLR0	0x04	定时器0预置数寄存器	TLR1	0x14	定时器1预置数寄存器
TCR0	0x08	定时器0计数寄存器	TCR1	0x18	定时器1计数寄存器

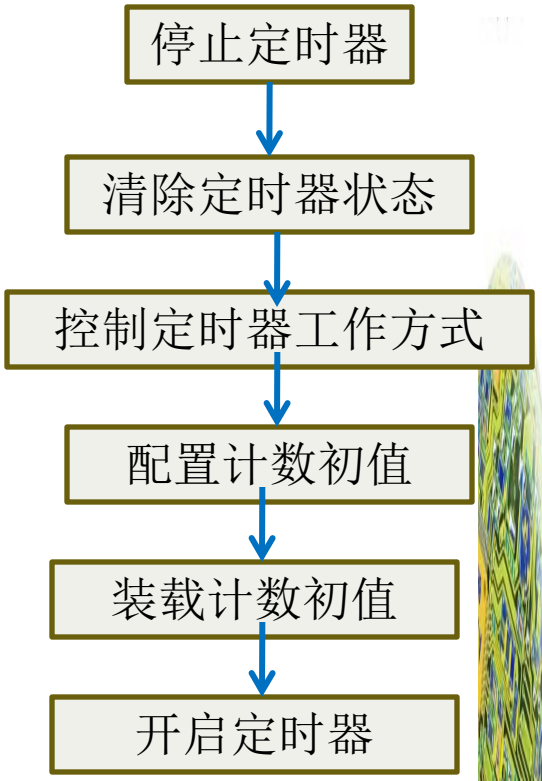
TCSR寄存器各位的定义



名称	含义	位置	读	写
MDT	工作模式	Bit0	设置值	1 capture 模式（外部触发）；0 Generate模式（定时输出）
UDT	计数方式	Bit1	设置值	1 减计数；0 加计数
GENT	使能GenerateOut输出	Bit2	设置值	0 禁止定时输出；1 允许定时输出
CAPT	Capture trig外部触发信号使能	Bit3	设置值	0关闭外部触发信号；1使能外部触发信号
ARHT	自动装载	Bit4	设置值	1 TCR自动装载TLR的值；0 TCR保持不变
LOAD	装载命令	Bit5	设置值	0不装载TLR到TCR；1装载TLR到TCR
ENINT	中断使能	Bit6	设置值	1产生中断输出；0不产生中断输出
ENT	定时器使能	Bit7	设置值	1定时器运行；0 定时器停止
TINT	定时器中断状态	Bit8	1中断 0，无	1清除中断状态；0无影响
PWMA	脉宽调制使能	Bit9	设置值	0使脉宽调制输出无效 1且MDT0,MDT1必须为0，GENT0,GENT1也同时为1时，脉宽输出调制有效，
ENALL	所有定时器使能	Bit10	设置值	1使能所有定时器，写0则清除ENALL位，对ENT0,ENT1无影响
保留		其余位		

控制定时器定时结束时产生中断的基本流程

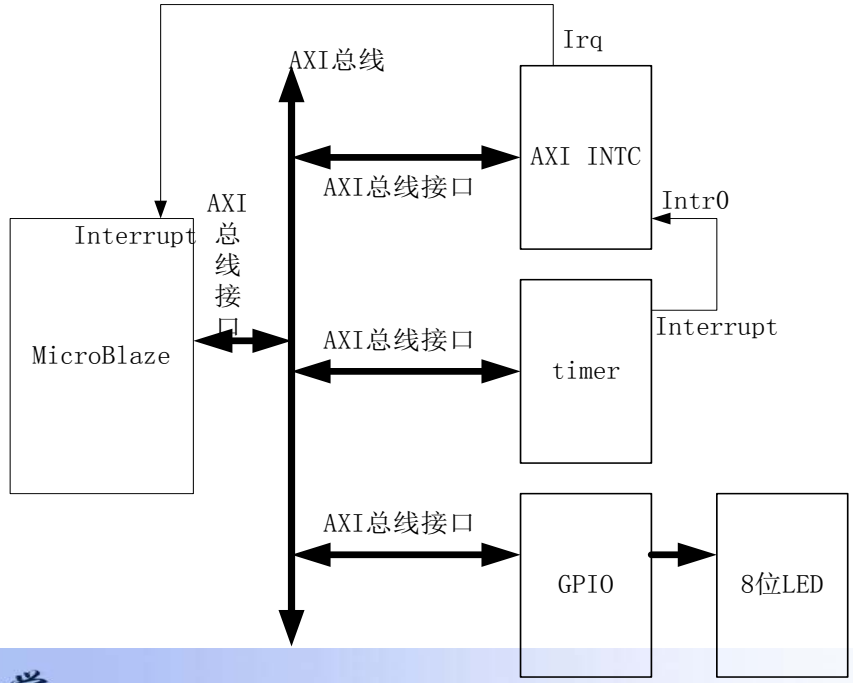
- 初始化定时器
  - 停止定时器，写TCSR使ENT=0；
  - 清除中断标志，写TINT=1；
  - 清除MDT,使其为0
  - 设置UDT为0或1，进行加或减计数；
  - 设置ARHT=1，控制定时器计数结束时自动装载预置值
  - 使能中断，写TCSR使ENINT=1；
  - 写TLR，配置计数初始值
  - 装载TCR，写TCSR使LOAD=1；
- 运行定时器，写TCSR使ENT=1，LOAD=0；



定时器初始化流程

定时器中断程序设计实例

- 基于MicroBlaze微处理器AXI总线设计硬件接口电路以及控制程序，要求微处理器控制8位LED灯轮流亮灭，且1秒钟更换一个。并采用硬件定时器中断方式实现延时控制。



- AXI timer时钟信号来自AXI总线时钟AXI_CLK。若AXI_CLK=100MHz，那么定时1s，就需要计100M个时钟脉冲。如果采用减计数，TLR的初始值就是100M；如果采用加计数，TLR的初始值就是$2^{32}-1-100M$。

定时器接口软件——读写寄存器

```
#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 LED_BASEADDR 0x40000000
#define XGPIO_DATA_OFFSET       0x0   /**< Data register for 1st channel */
#define XGPIO_TRI_OFFSET        0x4   /**< I/O direction reg for 1st channel */
#define XGPIO_GIE_OFFSET        0x11C /**< Glogal interrupt enable register */
#define XGPIO_ISR_OFFSET        0x120 /**< Interrupt status register */
#define XGPIO_IER_OFFSET        0x128 /**< Interrupt enable register */
#define TIMER_BASEADDR 0x41C00000
#define XTC_TCSR_OFFSET         0       /**< Control/Status register */
#define XTC_TLR_OFFSET          4       /**< Load register */
#define XTC_TCR_OFFSET          8       /**< Timer counter register */
#define RESET_VALUE      0x5F5E100
void TimerCounterHandler();
void My_ISR() __attribute__ ((interrupt_handler));//总中断服务程序
u32 LedBits;
```

```c
int main()
{
    Xil_Out32(LED_BASEADDR+XGPIO_TRI_OFFSET,0x00);//设定LED为输出方式
    Xil_Out32(TIMER_BASEADDR+XTC_TCSR_OFFSET,0x152);//写TCSR,停止计数器，自动装载，允许中断，减计数
    Xil_Out32(TIMER_BASEADDR+XTC_TLR_OFFSET,RESET_VALUE);//写TLR,预置计数初值
    Xil_Out32(TIMER_BASEADDR+XTC_TCSR_OFFSET,0x172);//装载计数初值
    Xil_Out32(TIMER_BASEADDR+XTC_TCSR_OFFSET,0x1d2);//开始计时运行
    Xil_Out32(INTC_BASEADDR+XIN_IER_OFFSET,0x1);//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);
    return 0;
}
void My_ISR()
{       int status;
    status=Xil_In32(INTC_BASEADDR+XIN_ISR_OFFSET);//读取ISR
    if((status&0x1)==0x1)
        TimerCounterHandler();//调用用户中断服务程序
    Xil_Out32(INTC_BASEADDR+XIN_IAR_OFFSET,status);//写IAR

}
void TimerCounterHandler()
{
    Xil_Out32(LED_BASEADDR,1<<LedBits);
    Xil_Out32(TIMER_BASEADDR+XTC_TCSR_OFFSET,0x1d2);//清除中断
    LedBits++;
    if(LedBits==8)
        LedBits=0;
}
```

定时器API

- XTmrCtr_Initialize(XTmrCtr*, u16) : int
- XTmrCtr_Start(XTmrCtr*, u8) : void
- XTmrCtr_Stop(XTmrCtr*, u8) : void
- XTmrCtr_GetValue(XTmrCtr*, u8) : u32
- XTmrCtr_SetResetValue(XTmrCtr*, u8, u32) : void
- XTmrCtr_GetCaptureValue(XTmrCtr*, u8) : u32
- XTmrCtr_IsExpired(XTmrCtr*, u8) : int
- XTmrCtr_Reset(XTmrCtr*, u8) : void
- XTmrCtr_LookupConfig(u16) : XTmrCtr_Config*
- XTmrCtr_SetOptions(XTmrCtr*, u8, u32) : void
- XTmrCtr_GetOptions(XTmrCtr*, u8) : u32
- XTmrCtr_GetStats(XTmrCtr*, XTmrCtrStats*) : void
- XTmrCtr_ClearStats(XTmrCtr*) : void
- XTmrCtr_SelfTest(XTmrCtr*, u8) : int
- XTmrCtr_SetHandler(XTmrCtr*, XTmrCtr_Handler, void*) : void
- XTmrCtr_InterruptHandler(void*) : void

定时器中断服务程序

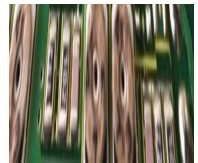
```
void TimerCounterHandler(void *CallBackRef, u8 TmrCtrNumber)
{
        Xil_Out32(XPAR_LEDS_8BITS_BASEADDR,1<<LedBits);
//产生中断时，输出LED显示值
        LedBits++;//修改显示位置指向下一位
        if(LedBits==8)
//由于只有8位LED灯，因此位置不能大于等于8，继续从bit0开始循环
                LedBits=0;
}
```

建立定时器中断系统函数

```
static int TmrCtrSetupIntrSystem(XIntc* IntcInstancePtr,XTmrCtr* TmrCtrInstancePtr,
u16 DeviceId, u16 IntrId, u8 TmrCtrNumber)
{                int Status;
Status = XIntc_Initialize(IntcInstancePtr, INTC_DEVICE_ID);//初始化中断控制器数据结构
                if (Status != XST_SUCCESS) {
                                return XST_FAILURE;
                }
Status = XIntc_Connect(IntcInstancePtr, IntrId,
                (XInterruptHandler)XTmrCtr_InterruptHandler,(void *)TmrCtrInstancePtr);
//将定时器主中断服务程序注册到相应Intr引脚对应的中断向量
                if (Status != XST_SUCCESS) {
                                return XST_FAILURE;       }
                Status = XIntc_Start(IntcInstancePtr, XIN_REAL_MODE);
//设置中断控制器接受硬件中断，并发出中断请求
                if (Status != XST_SUCCESS) {
                                return XST_FAILURE;       }
                XIntc_Enable(IntcInstancePtr, IntrId);//使能定时器对应Intr的中断请求
microblaze_register_handler((XInterruptHandler)XIntc_InterruptHandler, IntcInstancePtr);
//将中断控制器的总中断服务程序注册到0X0000 0010地址处
                microblaze_enable_interrupts();//使能微处理器的中断控制位
                return XST_SUCCESS;
}
```

定时器初始化函数

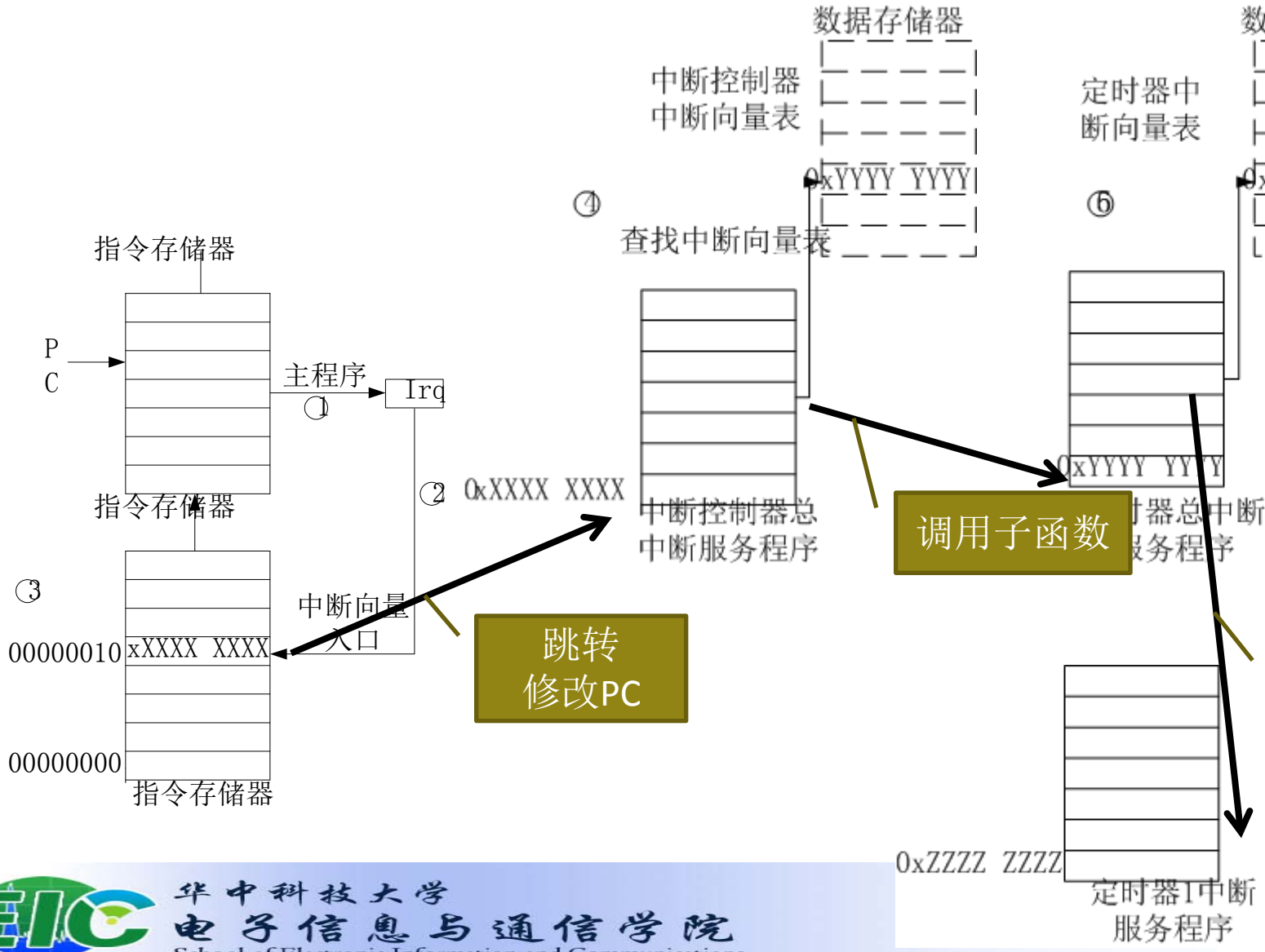
```
int TmrCtrIntrExample(XIntc* IntcInstancePtr,XTmrCtr* TmrCtrInstancePtr,u16 DeviceId,u16 IntrId,u8 TmrCtrNumber)
{
        int Status;
        Status = XTmrCtr_Initialize(TmrCtrInstancePtr, DeviceId);
//初始化定时器数据结构，并清除定时器中断产生标志以及装载标志
        if (Status != XST_SUCCESS) {
                return XST_FAILURE;
        }
        Status = TmrCtrSetupIntrSystem(IntcInstancePtr,TmrCtrInstancePtr,DeviceId,IntrId,TmrCtrNumber);
        if (Status != XST_SUCCESS) {
                return XST_FAILURE;
        }
        XTmrCtr_SetHandler(TmrCtrInstancePtr, TimerCounterHandler,  TmrCtrInstancePtr);//注册定时器中断服务程序
        XTmrCtr_SetOptions(TmrCtrInstancePtr, TmrCtrNumber,
        XTC_INT_MODE_OPTION|XTC_AUTO_RELOAD_OPTION| XTC_DOWN_COUNT_OPTION);
//设置定时器0：允许中断、自动装载、减计数
        XTmrCtr_SetResetValue(TmrCtrInstancePtr, TmrCtrNumber, RESET_VALUE);
//设置定时器初始值到TLR
        XTmrCtr_Start(TmrCtrInstancePtr, TmrCtrNumber);
//首先设置LOAD标志为1，然后在清除load标志的同时，设置定时器使能标志ENT=1
        return XST_SUCCESS;
}
```

主函数

```
int main(void)
{
        int Status;
        LedBits=0;
        Xil_Out32(XPAR_LEDS_8BITS_BASEADDR+0x4,0x0);
//控制通道1 LED GPIO为输出
        Status = TmrCtrIntrExample(&InterruptController,
                                   &TimerCounterInst,
                                   TMRCTR_DEVICE_ID,
                                   TMRCTR_INTERRUPT_ID,
                                   TIMER_CNTR_0);
        if (Status != XST_SUCCESS) {
                return XST_FAILURE;
        }
        While(1);//死循环
        return XST_SUCCESS;
}
```

定时器中断程序执行过程



作业

- 8