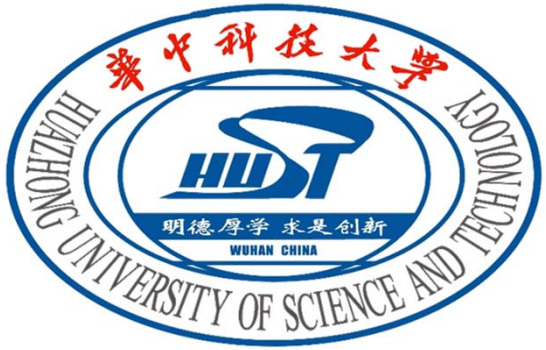
微机原理与接口技术

定时器中断应用示例

华中科技大学　左冬红



硬件定时器

数字钟　由计数器实现

秒钟进位信号的时间间隔60秒
分钟进位信号的时间间隔60分
} 固定时长定时器
嵌入式计算机系统的看门狗

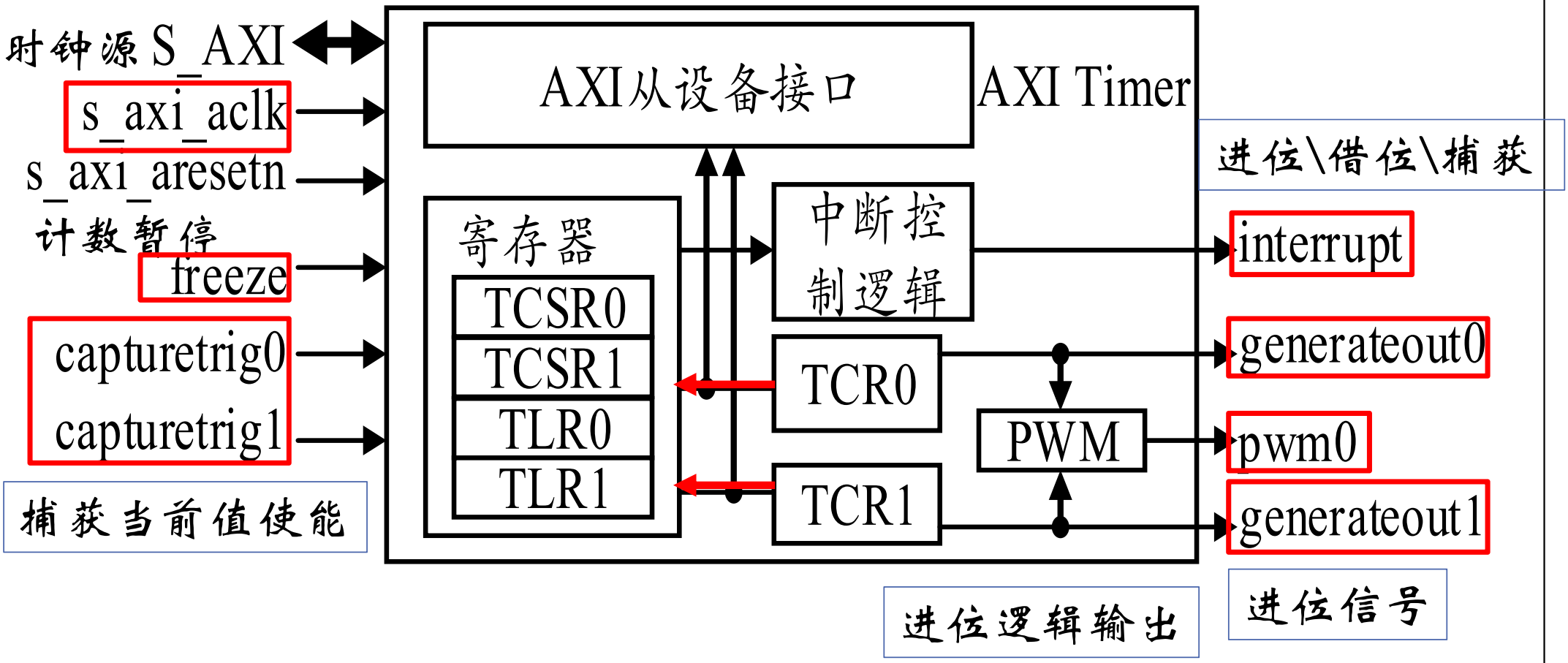
预置一次数值，之后多位计数器不停加计数　计算机系统时钟

若计数溢出之后自动装载新的预置值　定时间隔可调整

若将计数器的进位信号输出，则为周期性脉冲信号

若计数器的计数输出可锁存，则可测量外部信号周期（边沿触发）

AXI Timer定时器



Timer工作方式

- 定时
  - 单次定时：装载一次预置值；计数一轮；产生一个中断
  - 周期性定时：自动装载预置值；重复计数；周期性中断
- 定时模式：Generate
  - 周期脉冲
- 测频：无需预置；重复计数；中断时获取计数当前值
  - 两次中断计数值只差即为被测信号的周期
- 计时模式：Capture
- PWM波：周期性定时；自动装载预置值；重复计数；周期脉冲
- PWM模式：同时使用两个计数器

Timer定时间隔

加计数：T=(TCRmax-TLR+2)*AXI_CLK_PERIOD
减计数：T=(TLR+2)* AXI_CLK_PERIOD

Timer寄存器存储映像

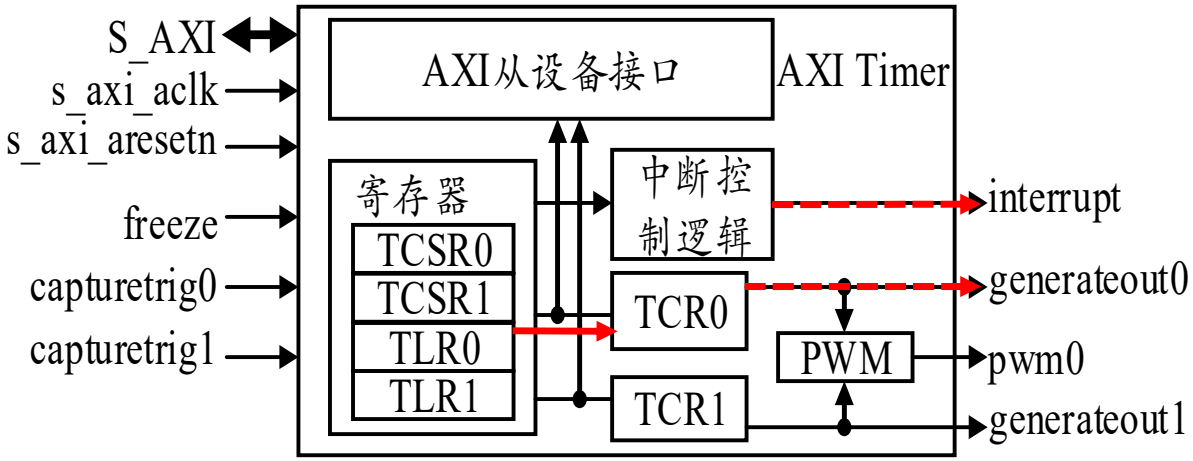
名称	偏移地址	功能描述	名称	偏移地址	功能描述
TCSR0	0x00	T0控制状态寄存器	TCSR1	0x10	T1控制状态寄存器
TLR0	0x04	T0装载寄存器	TLR1	0x14	T1装载寄存器
TCR0	0x08	T0计数寄存器	TCR1	0x18	T1计数寄存器

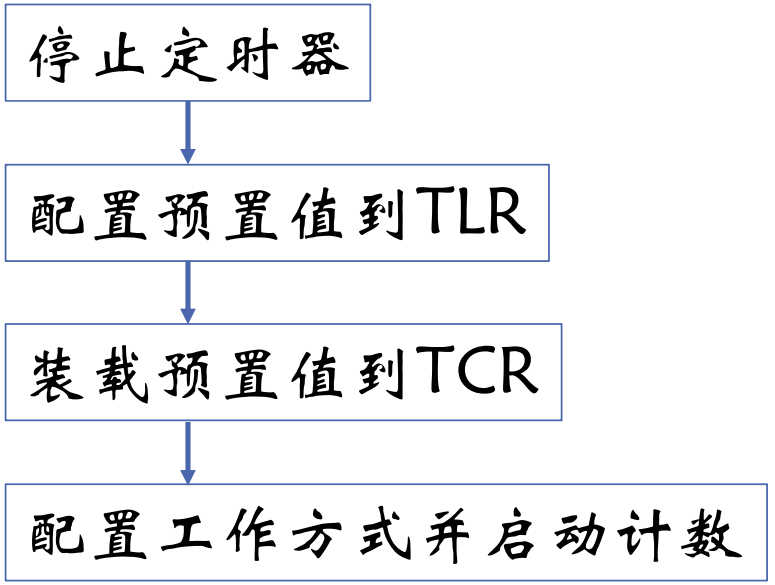
32位预置值或计数值

Timer TCSR寄存器

位	名称	含义
0	MDTx	工作模式：0-定时，1-计时
1	UDTx	计数方式：0-加计数，1-减计数
2	GENTx	generateout输出使能：0-禁止，1-使能
3	CAPTx	capturetrig触发使能：0-禁止，1-使能
4	ARHTx	自动重复装载：0-禁止，1-使能
5	LOADx	装载：0-不装载，1-装载
6	ENITx	中断使能：0-禁止，1-使能
7	ENTx	计数器运行使能：0-停止，1-启动
8	TxINT	计数器中断状态：读：0-无中断，1-有中断;写：0-无意义，1-清除中断状态
9	PWMAx	脉宽调制使能： 0-无效，1且MDTx=0、GENTx=1-脉宽输出
10	ENALL	所有计数器使能：0-清除ENALL位，对ENTx无影响；1-使能所有计数器
11	CASC	级联模式1-级联，0-独立 仅TCSR0此位有意义

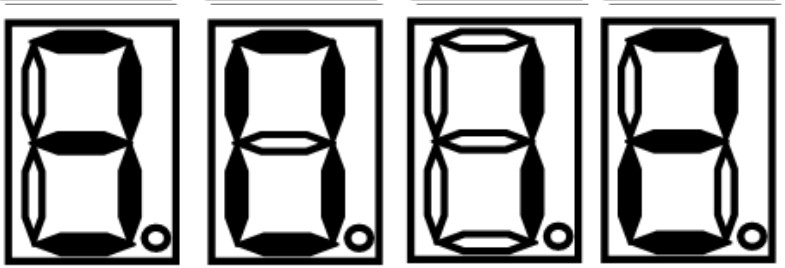
Timer工作原理-定时模式





Timer定时中断示例

Microblaze微处理器计算机系统采用AXI总线控制4位七段数码管动态显示接口滚屏显示数字0~3序列，要求采用AXI Timer定时中断实现，试设计接口电路和控制程序。



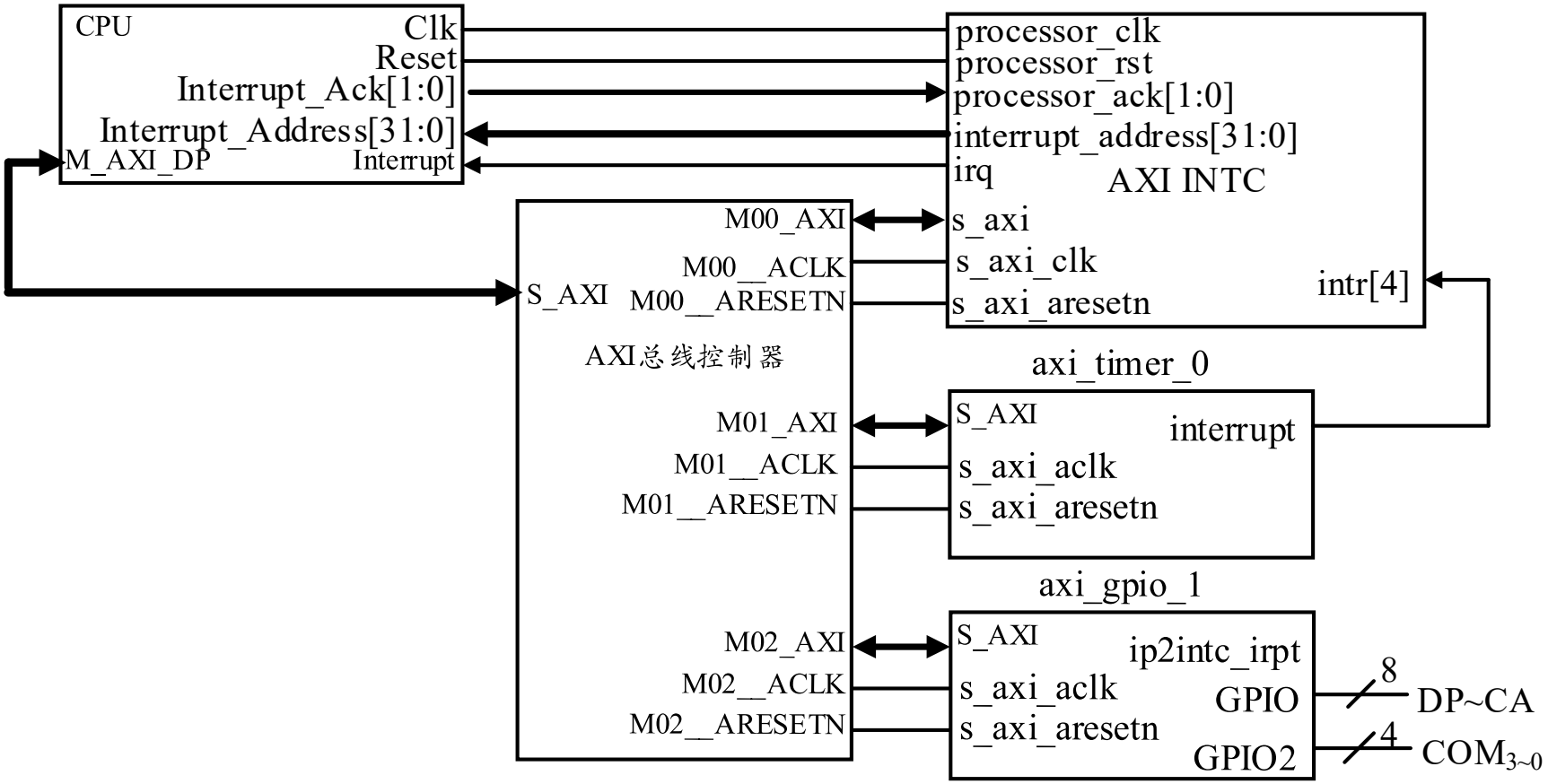
4位七段数码管滚屏显示数字0~3序列，即4位七段数码管首次显示数字串0123，之后每隔一段时间再依次更换显示数字串1230；2301；3012；并循环

两处定时：

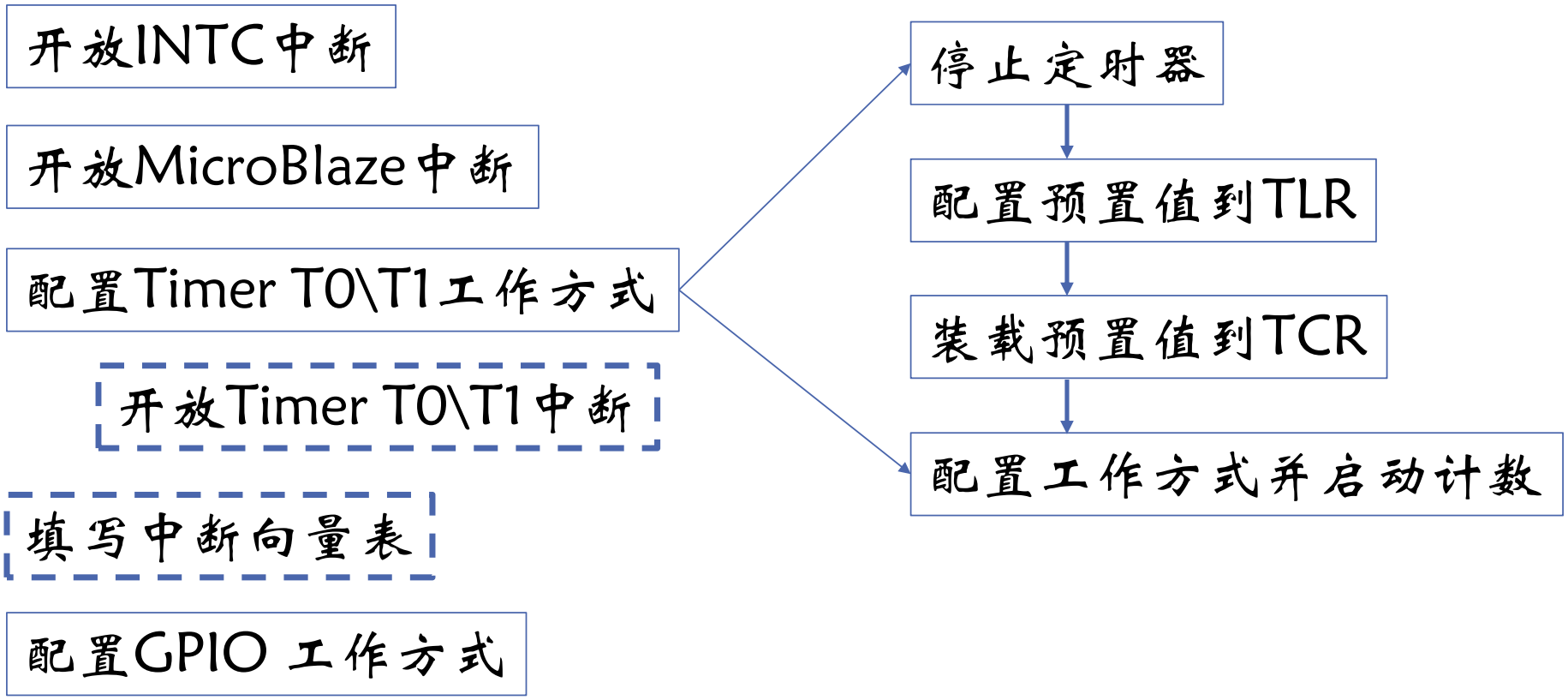
1）4位七段数码管动态显示扫描间隔延时；

2）显示数字串更新延时。

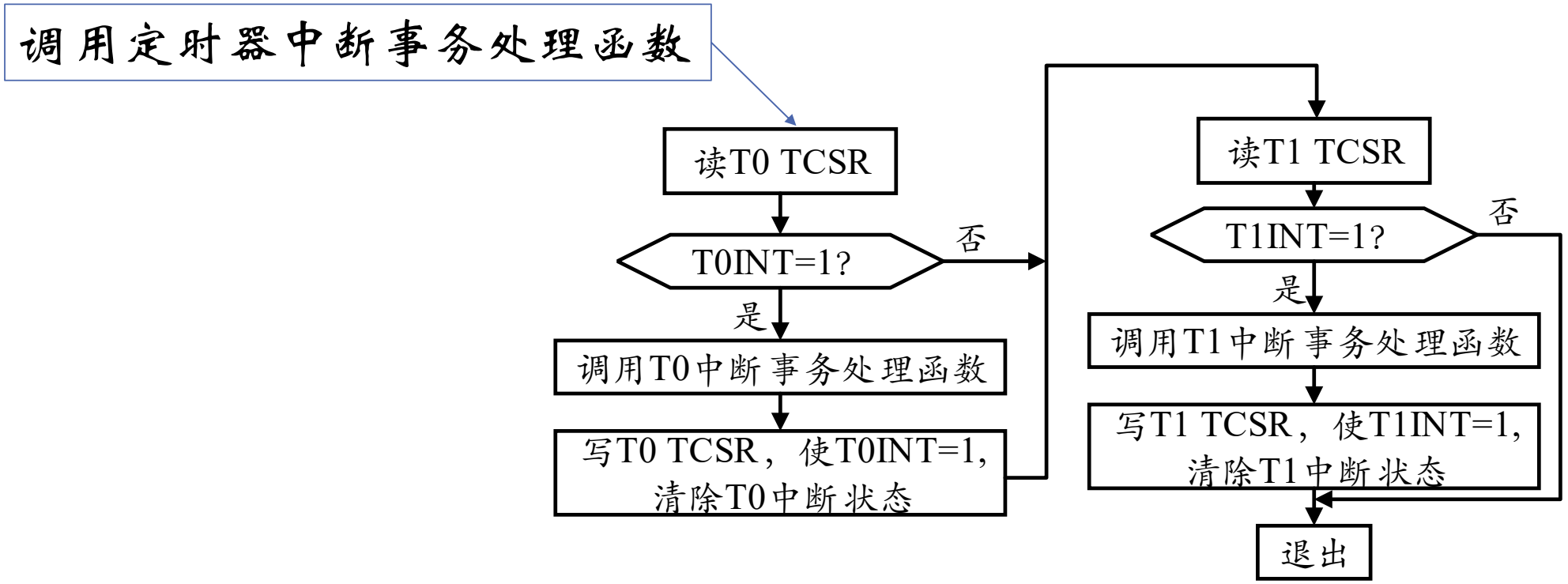
Timer定时中断示例接口电路-快速中断



中断初始化程序

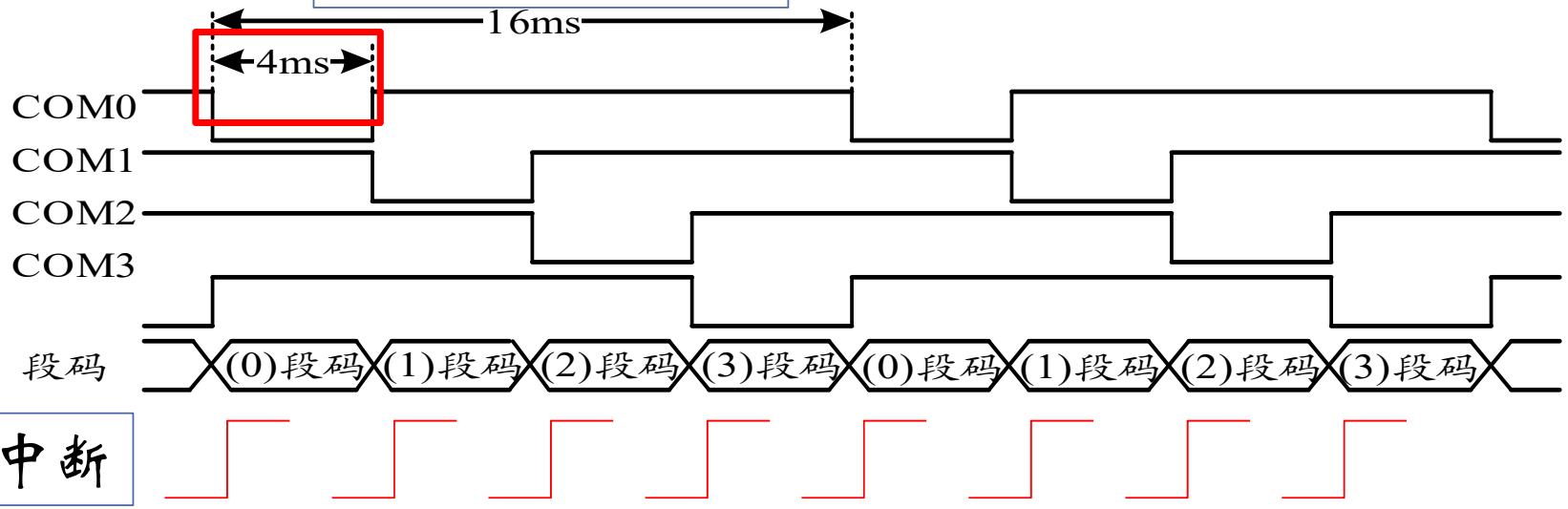

开放INTC中断
开放MicroBlaze中断
配置Timer T0\T1工作方式
开放Timer T0\T1中断
填写中断向量表
配置GPIO 工作方式
停止定时器
配置预置值到TLR
装载预置值到TCR
配置工作方式并启动计数

中断服务程序


调用定时器中断事务处理函数
读T0 TCSR
T0INT=1?
否
是
调用T0中断事务处理函数
写T0 TCSR，使T0INT=1，
清除T0中断状态
读T1 TCSR
T1INT=1?
否
是
调用T1中断事务处理函数
写T1 TCSR，使T1INT=1，
清除T1中断状态
退出

4位七段数码管动态显示扫描间隔延时

硬件定时器定时



定时中断

中断事务：

从显示缓冲区读取一位数字对应的段码、位码并输出

记录段码、位码对应值，中断次数%4

数字串移位更新延时间隔

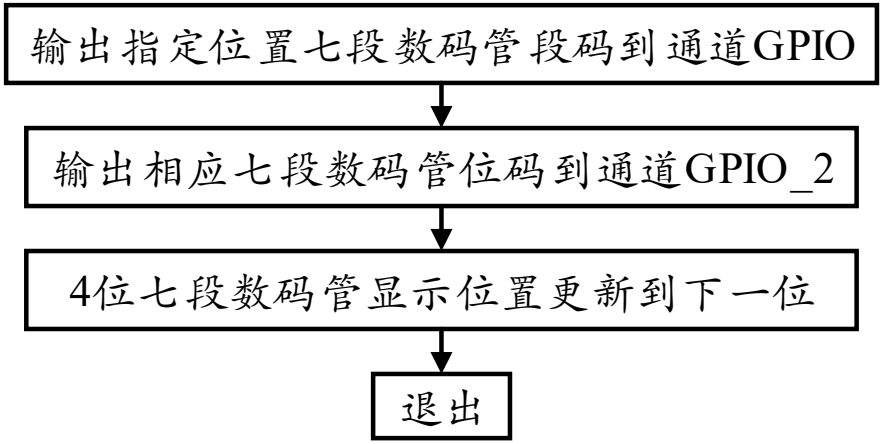
间隔时间应为数秒或更长

中断事务：

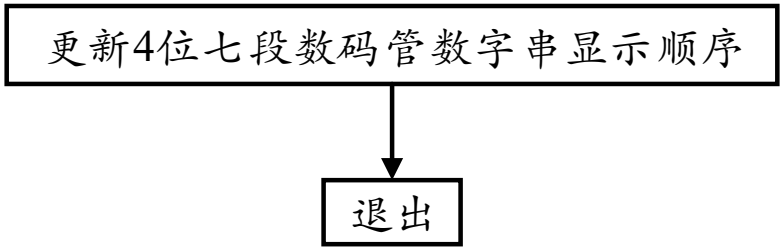
更新数码管显示缓冲器

各位数字段码在显示缓冲区中存储的位置向左循环移动一位

定时器中断事务处理



T0中断事务处理函数流程



T1中断事务处理函数流程

中断初始化程序代码——快速中断

```
void setup_interrupt_system(void)
{
    Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IAR_OFFSET,XPAR_AXI_TIMER_0_INTERRUPT_MASK);
    Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IER_OFFSET,XPAR_AXI_TIMER_0_INTERRUPT_MASK);
    Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IMR_OFFSET,XPAR_AXI_TIMER_0_INTERRUPT_MASK);
    Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_MER_OFFSET,
   XIN_INT_MASTER_ENABLE_MASK|XIN_INT_HARDWARE_ENABLE_MASK);
    Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IVAR_OFFSET+
          4*XPAR_INTC_0_TMRCTR_0_VEC_ID,(int)Timerhandler);
    Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET,
                Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET )|XTC_CSR_INT_OCCURED_MASK);
    Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET,
                  Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET )| XTC_CSR_ENABLE_INT_MASK);
    Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET,
          Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET )
          | XTC_CSR_INT_OCCURED_MASK);
    Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET,
          Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET )
          | XTC_CSR_ENABLE_INT_MASK);
    microblaze_enable_interrupts();
}
```

IO控制初始化程序

```
int main(void)
{
    int tcsr0,tcsr1;
    setup_interrupt_system();
    Xil_Out32(XPAR_GPIO_1_BASEADDR+XGPIO_TRI_OFFSET,0x0);
    Xil_Out32(XPAR_GPIO_1_BASEADDR+XGPIO_TRI2_OFFSET,0x0);
    Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TLR_OFFSET,T0_RESET_VALUE);
    Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TLR_OFFSET,T1_RESET_VALUE);
    tcsr0=Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET);
    Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET,tcsr0|XTC_CSR_LOAD_MASK);
    tcsr1=Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET);
    Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET,tcsr1|XTC_CSR_LOAD_MASK);
    Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET,
  tcsr0|XTC_CSR_ENABLE_TMR_MASK|XTC_CSR_DOWN_COUNT_MASK
  |XTC_CSR_AUTO_RELOAD_MASK|XTC_CSR_EXT_GENERATE_MASK);
    Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET,
tcsr1|XTC_CSR_ENABLE_TMR_MASK|XTC_CSR_DOWN_COUNT_MASK|XTC_CSR_AUTO_RELOAD_MASK
            |XTC_CSR_EXT_GENERATE_MASK);
    return 0;
}
```

全局变量及函数申明

```
#define T0_RESET_VALUE 100000-2 //0.001s
#define T1_RESET_VALUE 100000000-2 //1s
void T0handler(void);
void T1handler(void);
void Timerhandler(void) __attribute__ ((fast_interrupt));
void setup_interrupt_system(void);
char segcode[4]={0xc0,0xf9,0xa4,0xb0};
char poscode[4]={0xf7,0xfb,0xfd,0xfe};
int loop=0,pos=0;
```

宏定义头文件

```
#include "xintc_l.h"
#include "xtmrctr_l.h"
#include "xgpio_l.h"
#include "xparameters.h"
#include "xio.h"
#include "xil_exception.h"
```

定时器中断服务程序

```
void Timerhandler(void)
{
    int tcsr;
    tcsr=Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET);
    if((tcsr&XTC_CSR_INT_OCCURED_MASK)==XTC_CSR_INT_OCCURED_MASK)
    {
        T0handler();
        Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET,
                tcsr|XTC_CSR_INT_OCCURED_MASK);
    }
    tcsr=Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET
                +XTC_TCSR_OFFSET);
    if((tcsr&XTC_CSR_INT_OCCURED_MASK)==XTC_CSR_INT_OCCURED_MASK)
    {
        T1handler();
        Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET
            +XTC_TCSR_OFFSET,tcsr|XTC_CSR_INT_OCCURED_MASK);
    }
}
```

中断事务处理函数

```
void T0handler(void)
{
    Xil_Out32(XPAR_GPIO_1_BASEADDR+XGPIO_DATA_OFFSET,segcode[(loop+pos)%4]);
    Xil_Out32(XPAR_GPIO_1_BASEADDR+XGPIO_DATA2_OFFSET,poscode[pos]);
    pos++;
    if(pos==4)
        pos=0;
}
```

```
void T1handler(void)
{
    loop++;
    if(loop==4)
        loop=0;
}
```

显示缓冲区段码数字
0,0xc0
1,0xf9
2,0xa4
3,0xb0

	T0 pos			
	0	1	2	3
T1 loop	数码管显示的数字串			
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

普通中断方式

```
#define T0_RESET_VALUE 100000-2 //0.001s
#define T1_RESET_VALUE 100000000-2 //1s
void T0handler(void);
void T1handler(void);
void Timerhandler(void) __attribute__ ((interrupt_handler));
void setup_interrupt_system(void);
char segcode[4]={0xc0,0xf9,0xa4,0xb0};
char poscode[4]={0xf7,0xfb,0xfd,0xfe};
 int loop=0,pos=0;
```

定时器中断服务程序

```
void Timerhandler(void)
{
    int tcsr;
    tcsr=Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET);
    if((tcsr&XTC_CSR_INT_OCCURED_MASK)==XTC_CSR_INT_OCCURED_MASK)
    {
        T0handler();
        Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET,
                tcsr|XTC_CSR_INT_OCCURED_MASK);
    }
    tcsr=Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET
                +XTC_TCSR_OFFSET);
    if((tcsr&XTC_CSR_INT_OCCURED_MASK)==XTC_CSR_INT_OCCURED_MASK)
    {
        T1handler();
        Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET
            +XTC_TCSR_OFFSET,tcsr|XTC_CSR_INT_OCCURED_MASK);
    }
    Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IAR_OFFSET,
    XPAR_AXI_TIMER_0_INTERRUPT_MASK);//写INTC IAR
}
```

中断初始化程序代码——普通中断

```
void setup_interrupt_system(void)
{
    Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IAR_OFFSET,XPAR_AXI_TIMER_0_INTERRUPT_MASK);
    Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IER_OFFSET,XPAR_AXI_TIMER_0_INTERRUPT_MASK);
    Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IMR_OFFSET,XPAR_AXI_TIMER_0_INTERRUPT_MASK);
    Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_MER_OFFSET,
   XIN_INT_MASTER_ENABLE_MASK|XIN_INT_HARDWARE_ENABLE_MASK);
    Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IVAR_OFFSET+
          4*XPAR_INTC_0_TMRCTR_0_VEC_ID,(int)Timerhandler);
    Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET,
                Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET )|XTC_CSR_INT_OCCURED_MASK);
    Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET,
                  Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET )| XTC_CSR_ENABLE_INT_MASK);
    Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET,
          Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET )
          | XTC_CSR_INT_OCCURED_MASK);
    Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET,
          Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET )
          | XTC_CSR_ENABLE_INT_MASK);
    microblaze_enable_interrupts();
}
```

小结

- 定时器工作在循环定时方式的初始化控制流程
  - 停止定时器
  - 写计数初值
  - 装载计数初值
  - 启动定时器并配置工作模式、启用中断
- 快速中断与普通中断方式区别
  - 快速中断INTC无需查询(多中断源)、无需写IAR
  - 快速中断需填写INTC中断向量表