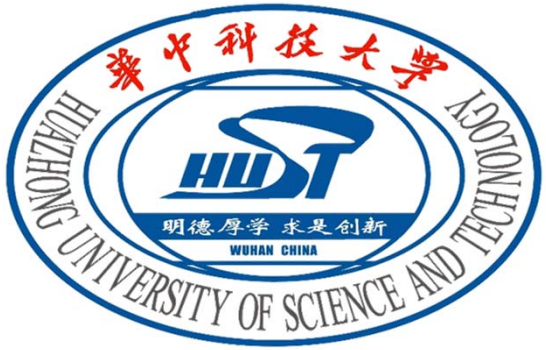
微机原理与接口技术

多中断源程序设计示例

华中科技大学　左冬红



多中断源应用系统示例

基于嵌入式微处理器设计一个同时支持多种并行IO设备工作的嵌入式实时MIMO系统。基本输入输出设备有：16个独立LED灯，16个独立开关、2个独立按键，4个七段数码管。且都通过GPIO并行输入输出接口连接到单核微处理器计算机系统的同一总线上。

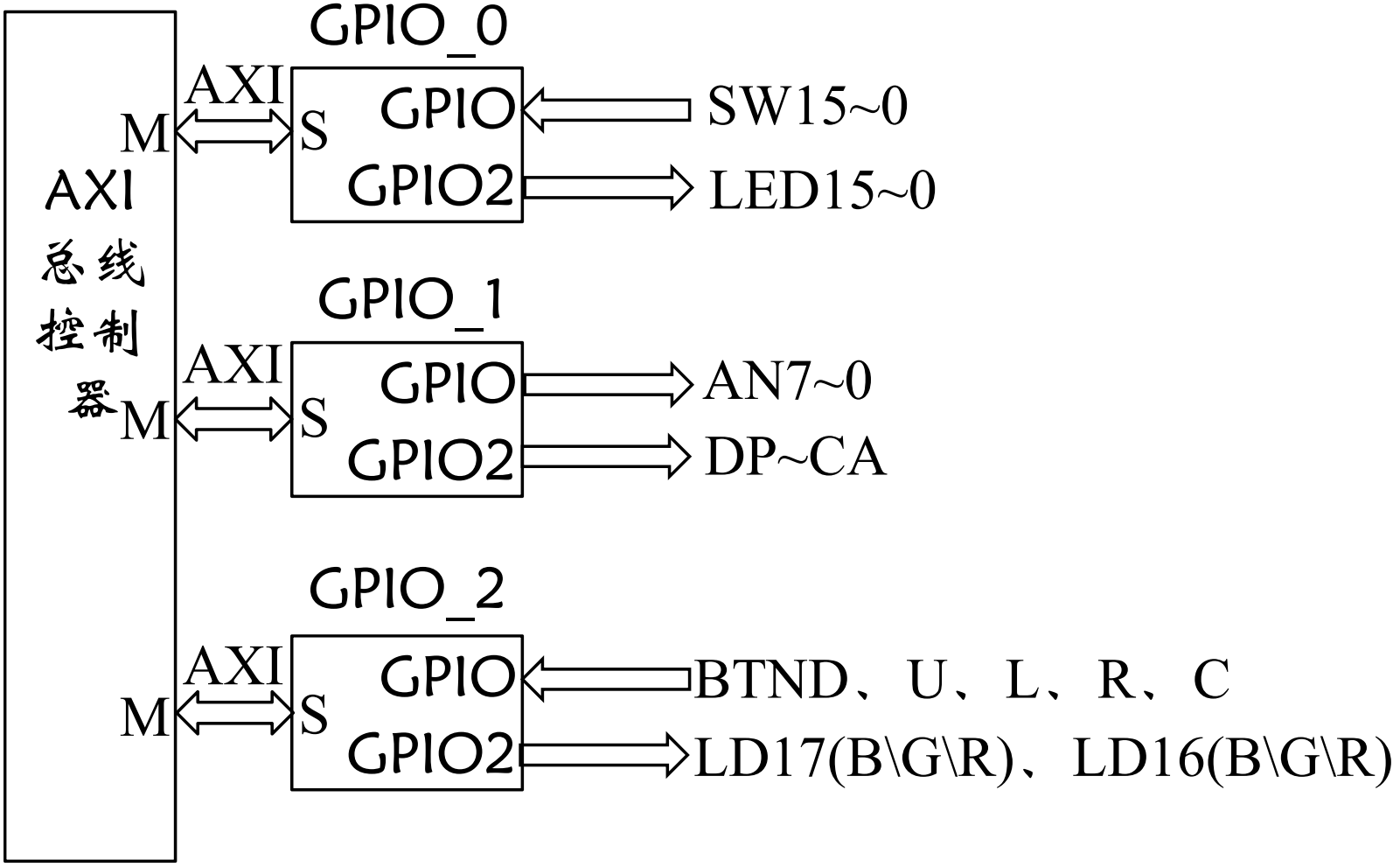
要求实现的基本功能为：

1) 16个LED灯走马灯式轮流循环亮灭。且循环速度可通过两个独立按键步进控制，其中一个按键每按下一次步进增速，另一个按键每按一次步进减速。

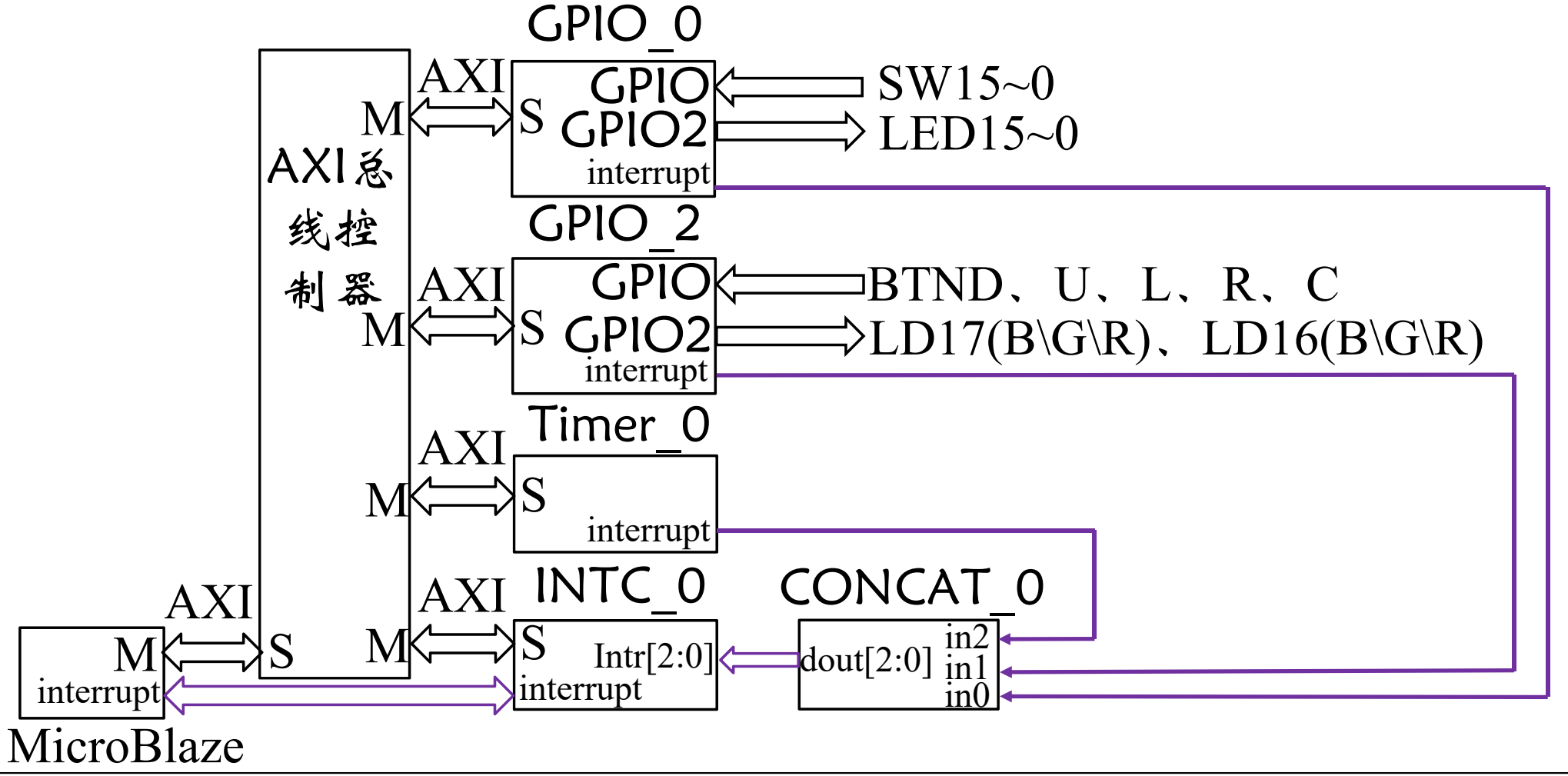
2) 4个七段数码管实时显示16位独立开关表示的十六进制数。

四个中断源
- 两个GPIO
- 两个Timer

GPIO与外设连接关系



并行IO中断系统


GPIO_0
AXI
M
S
GPIO
GPIO2
interrupt
SW15~0
LED15~0
AXI总线控制器
GPIO_2
AXI
M
S
GPIO
GPIO2
interrupt
BTND、U、L、R、C
LD17(B\G\R)、LD16(B\G\R)
Timer_0
AXI
M
S
interrupt
AXI
M
S
INTC_0
AXI
M
S
Intr[2:0]
interrupt
CONCAT_0
dout[2:0]
in2
in1
in0
AXI
M
S
interrupt
MicroBlaze

普通中断方式——初始化流程

- 各GPIO工作方式设定：输入、输出
- 各Timer工作方式设定
  - 停止Timer
  - 设定初值
    - 装载初值
  - 启动Timer、开中断、自动装载、减计数
- 开放中断系统
  - GPIO_0开中断
  - GPIO_2开中断
  - INTC_0开中断
  - MicroBlaze_0开中断
- 注册总中断服务程序

普通中断方式——总中断服务程序

读INTC ISR

判断intr0　　调用开关中断事务处理函数

判断intr1　　调用按键中断事务处理函数

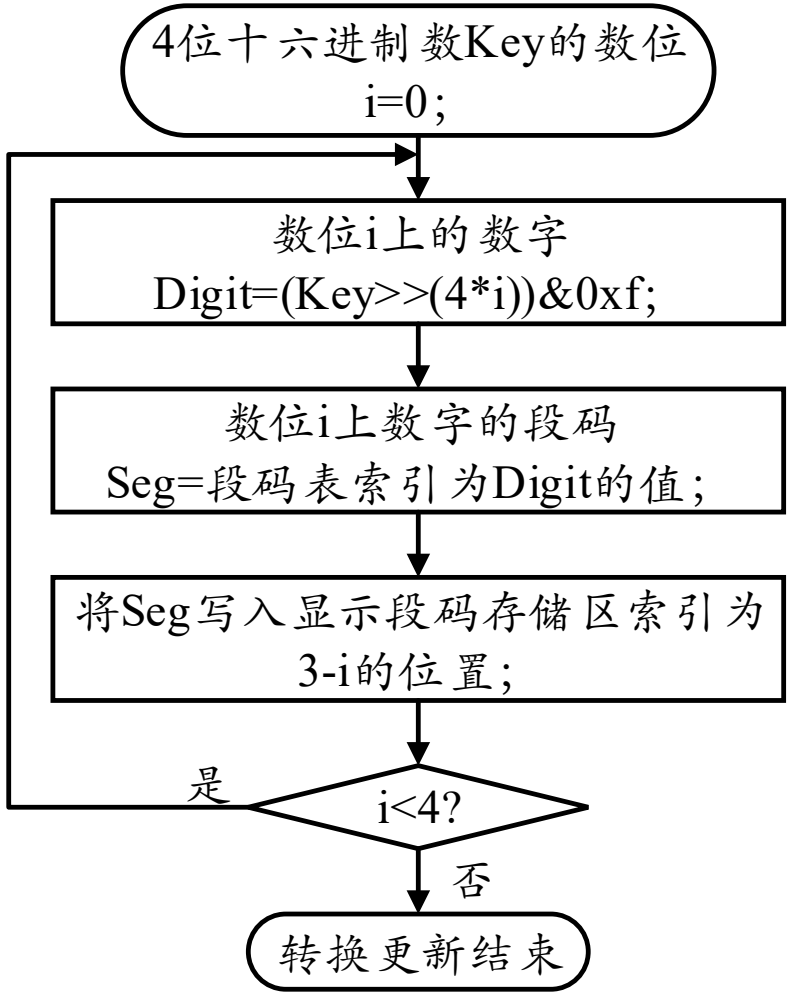
判断intr2　　调用定时器中断事务处理函数

写INTC IAR清除ISR状态

开关中断事务处理程序

读取开关值、更新数码管显示缓冲区

读后写GPIO_0 ISR清除ISR状态



按键中断事务处理程序

读取按键值，改变走马灯定时器计数初值

判断增速按键

将定时器初值减少一固定值

判断减速按键

将定时器初值增加一固定值

读后写GPIO_2 ISR清除ISR状态

定时器总中断事务处理程序

判断定时器0中断

调用定时器0中断事务处理函数

读后写定时器0 TCSR 清除定时器0中断状态

判断定时器1中断

调用定时器1中断事务处理函数

读后写定时器1 TCSR 清除定时器1中断状态

走马灯定时器中断事务处理程序

控制当前位置LED灯点亮，并将当前位置修改为下一位

数码管动态扫描定时器中断事务处理程序

输出当前位置数码管段码、位码，并将当前位置修改为下一位

普通中断方式初始化程序代码

```
int main()
{
Xil_Out8(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_TRI_OFFSET,0x1f);
Xil_Out16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_TRI_OFFSET,0xffff);
Xil_Out16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_TRI2_OFFSET,0x0);
Xil_Out16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_DATA2_OFFSET,0x1);
Xil_Out8(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_TRI_OFFSET,0x0);
Xil_Out8(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_TRI2_OFFSET,0x0);
Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_IER_OFFSET,XGPIO_IR_CH1_MASK);//
Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_GIE_OFFSET,XGPIO_GIE_GINTR_ENABLE_MASK);//
Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_IER_OFFSET,XGPIO_IR_CH1_MASK);//
Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_GIE_OFFSET,XGPIO_GIE_GINTR_ENABLE_MASK);//
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET,Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET)&~XTC_CSR_ENABLE_TMR_MASK);
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TLR_OFFSET,RESET_VALUE0);//
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET,Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET)|XTC_CSR_LOAD_MASK);//
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET,(Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET)&~XTC_CSR_LOAD_MASK)
|XTC_CSR_ENABLE_TMR_MASK|XTC_CSR_AUTO_RELOAD_MASK|XTC_CSR_ENABLE_INT_MASK|XTC_CSR_DOWN_COUNT_MASK);
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET,
Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET)&~XTC_CSR_ENABLE_TMR_MASK);//
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TLR_OFFSET,RESET_VALUE1);//
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET,
Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET)|XTC_CSR_LOAD_MASK);//
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET,
(Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET)&~XTC_CSR_LOAD_MASK)\
|XTC_CSR_ENABLE_TMR_MASK|XTC_CSR_AUTO_RELOAD_MASK|XTC_CSR_ENABLE_INT_MASK|XTC_CSR_DOWN_COUNT_MASK);
Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_IER_OFFSET,XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK
|XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK|XPAR_AXI_TIMER_0_INTERRUPT_MASK);//
Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_MER_OFFSET,XIN_INT_MASTER_ENABLE_MASK|XIN_INT_HARDWARE_ENABLE_MASK);
microblaze_enable_interrupts();
microblaze_register_handler((XInterruptHandler)My_ISR, (void *)0);
}
```

总中断服务程序代码

```
void My_ISR()
{
int status;
status=Xil_In32(XPAR_AXI_INTC_0_BASEADDR+XIN_ISR_OFFSET);//
if((status&XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK)==XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK)
switch_handle();
if((status&XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK)==XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK)
button_handle();
if ((status&XPAR_AXI_TIMER_0_INTERRUPT_MASK)==XPAR_AXI_TIMER_0_INTERRUPT_MASK)
timer_handle();
Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_IAR_OFFSET,status);//
}
```

开关中断事务处理程序代码

```
void switch_handle()
{
short hex=Xil_In16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_DATA_OFFSET);
int segcode_index=3;
for(int digit_index=0;digit_index<4;digit_index++)
{
segcode[segcode_index]=segtable[(hex>>(4*digit_index))&0xf];
segcode_index--;
}
Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_ISR_OFFSET,
Xil_In32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_ISR_OFFSET));//
}
```

按键中断事务处理程序

```
void button_handle()
{
    char button;
    button = Xil_In8(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_DATA_OFFSET)&0x1f;
    if(button==0x2)
        Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TLR_OFFSET,
                Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TLR_OFFSET)-STEP_PACE);
    if (button==0x10)
        Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TLR_OFFSET,
                Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TLR_OFFSET)+STEP_PACE);
    Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_ISR_OFFSET,
                Xil_In32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_ISR_OFFSET));//
}
```

定时器总中断服务程序代码

```
void timer_handle()
{
int status;
status=Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET);//
if((status&XTC_CSR_INT_OCCURED_MASK)==XTC_CSR_INT_OCCURED_MASK)
timer0_handle();
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET,
Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET));
status=Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET);//
if((status&XTC_CSR_INT_OCCURED_MASK)==XTC_CSR_INT_OCCURED_MASK)
timer1_handle();
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET,
Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET));
}
```

定时器中断事务处理程序代码

走马灯定时

```
void timer0_handle()
{
    ledbits++;
    if(ledbits==16)
        ledbits=0;
    Xil_Out16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_DATA2_OFFSET,1<<ledbits);
}
```

数码管动态扫描定时

```
void timer1_handle()
{
    Xil_Out16(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_DATA2_OFFSET,segcode[pos]);
    Xil_Out16(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_DATA_OFFSET,poscode[pos]);
    pos++;
    if(pos==4)
            pos=0;
}
```

全部变量及函数申明

```
#include "xil_io.h"
#include "stdio.h"
#include "xintc_l.h"
#include "xtmrctr_l.h"
#include "xgpio_l.h"
#define RESET_VALUE0       100000000-2
#define RESET_VALUE1       100000-2
#define STEP_PACE 10000000
void My_ISR();
void switch_handle();
void button_handle();
void timer_handle();
void timer0_handle();
void timer1_handle();
char segtable[16]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x98,0x88,0x83,0xc6,0xa1,0x86,0x8e};
char segcode[4]={0xc0,0xc0,0xc0,0xc0};
short poscode[4]={0xf7,0xfb,0xfd,0xfe};
int ledbits=0;
int pos=0;
```

快速中断——初始化程序

```
int main()
{
Xil_Out8(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_TRI_OFFSET,0x1f);
Xil_Out16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_TRI_OFFSET,0xffff);
Xil_Out16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_TRI2_OFFSET,0x0);
Xil_Out16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_DATA2_OFFSET,0x1);
Xil_Out8(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_TRI_OFFSET,0x0);
Xil_Out8(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_TRI2_OFFSET,0x0);
Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_IER_OFFSET,XGPIO_IR_CH1_MASK);//
Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_GIE_OFFSET,XGPIO_GIE_GINTR_ENABLE_MASK);//
Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_IER_OFFSET,XGPIO_IR_CH1_MASK);//
Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_GIE_OFFSET,XGPIO_GIE_GINTR_ENABLE_MASK);//
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET,Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET)&~XTC_CSR_ENABLE_TMR_MASK);
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TLR_OFFSET,RESET_VALUE0);//
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET,Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET)|XTC_CSR_LOAD_MASK);//
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET,(Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET)&~XTC_CSR_LOAD_MASK)
|XTC_CSR_ENABLE_TMR_MASK|XTC_CSR_AUTO_RELOAD_MASK|XTC_CSR_ENABLE_INT_MASK|XTC_CSR_DOWN_COUNT_MASK);
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET,
Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET)&~XTC_CSR_ENABLE_TMR_MASK);//
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TLR_OFFSET,RESET_VALUE1);//
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET,
Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET)|XTC_CSR_LOAD_MASK);//
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET,
(Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET)&~XTC_CSR_LOAD_MASK)\
|XTC_CSR_ENABLE_TMR_MASK|XTC_CSR_AUTO_RELOAD_MASK|XTC_CSR_ENABLE_INT_MASK|XTC_CSR_DOWN_COUNT_MASK);
Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_IER_OFFSET,XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK
|XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK|XPAR_AXI_TIMER_0_INTERRUPT_MASK);//
Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_MER_OFFSET,XIN_INT_MASTER_ENABLE_MASK|XIN_INT_HARDWARE_ENABLE_MASK);
microblaze_enable_interrupts();
microblaze_register_handler((XInterruptHandler)My_ISR, (void *)0);
}
```

快速中断-初始化程序

配置快速模式

```
Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_IMR_OFFSET,XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK
|XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK|XPAR_AXI_TIMER_0_INTERRUPT_MASK);//
```

填写INTC IVAR

```
Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_IVAR_OFFSET+
        4*XPAR_AXI_INTC_0_AXI_GPIO_0_IP2INTC_IRPT_INTR,(int)switch_handle);
Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_IVAR_OFFSET+
        4*XPAR_AXI_INTC_0_AXI_GPIO_2_IP2INTC_IRPT_INTR,(int)button_handle);
Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_IVAR_OFFSET+
        4*XPAR_AXI_INTC_0_AXI_TIMER_0_INTERRUPT_INTR,(int)timer_handle);
```

全部变量及函数申明

```
#include "xil_io.h"
#include "stdio.h"
#include "xintc_l.h"
#include "xtmrctr_l.h"
#include "xgpio_l.h"
#define RESET_VALUE0       100000000-2
#define RESET_VALUE1       100000-2
#define STEP_PACE 10000000
void My_ISR();
void switch_handle() __attribute__ ((fast_interrupt));
void button_handle() __attribute__ ((fast_interrupt));
void timer_handle() __attribute__ ((fast_interrupt));
void timer0_handle();
void timer1_handle();
char segtable[16]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x98,0x88,0x83,0xc6,0xa1,0x86,0x8e};
char segcode[4]={0xc0,0xc0,0xc0,0xc0};
short poscode[4]={0xf7,0xfb,0xfd,0xfe};
int ledbits=0;
int pos=0;
```

中断事务处理程序函数体不变

程序控制方式实现——请自行设计

跑马灯延时

数码管动态扫描延时

查询开关

查询按键

小结

- 多中断源
  - 产生中断时，中断服务程序处理各自相应业务
  - 中断事务尽可能简单，否则阻塞其他中断源
    - 开启MicroBlaze中断
    - 关闭MicroBlaze中断
- 快速模式-普通模式对比

下一讲：DMA技术