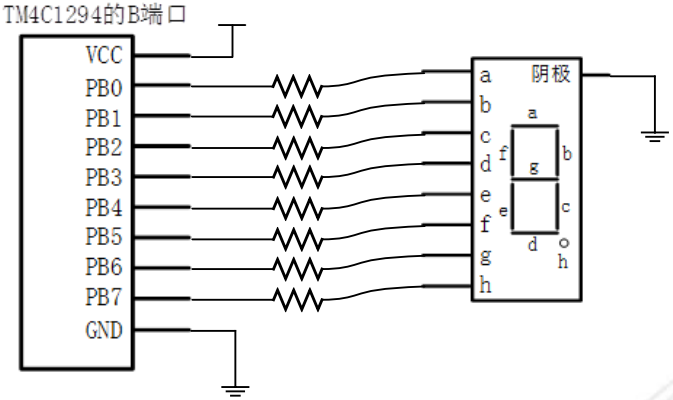
系统节拍定时器（SysTick）

程晨闻

东南大学电气工程学院

作业

– 使用PB口驱动一个7段数码管，画出端口与数码管的电路接线。写出端口的初始化程序。在数码管上显示数字7，写出相关的程序代码。




```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,
GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 |
GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7);
```

```
uint16_t LEDtb[]={3fh, 06h, 5bh, 4fh, 66h, 6dh, 7dh, 07h, 7fh, 6fh};
void LEDDisp(int value)
{
GPIOPinWrite(GPIO_PORTB_BASE,0xff, LEDtb [value]);
}
LEDDisp(7);
```

- 中断和异常基本概念
- 中断的响应流程
  - 外设级(GPIO)，NVIC级，CPU级
- 中断控制寄存器的访问
  - IntMasterEnable() (PRIMASK,CPU级)
  - IntEnable() (ENn, NVIC级)
  - IntDisable() (DISn, NVIC级)
  - GPIOIntTypeSet(), GPIOIntEnable()(外设级)
  - GPIOIntClear(), GPIOIntStatus()(外设级)

➢ 中断函数的注册

- 根据中断向量号注册中断服务函数
- 将中断向量表从FLASH移动到SRAM中
- 可通过VTABLE改变向量表起始地址
- GPIOIntRegister(uint32_t ui32Port, void (*pfnIntHandler)(void))(调用IntRegister()和IntEnable())

```
//
// Get the interrupt number associated with the specified GPIO.
//
ui32Int = _GPIOIntNumberGet(ui32Port);
```

- IntRegister(uint32_t ui32Interrupt, void (*pfnHandler)(void))

➢ 中断服务函数的编写

- 判断中断源（调用GPIOIntStatus函数）
- 执行具体的操作
- 手动清除中断标志（调用GPIOIntClear函数）

```
void
IntGPIOj(void)
{
        if(GPIOIntStatus(GPIO_PORTJ_AHB_BASE,true)==GPIO_INT_PIN_0){
                // do something
                GPIOIntClear(GPIO_PORTJ_AHB_BASE,GPIO_INT_PIN_0);
        }
        if(GPIOIntStatus(GPIO_PORTJ_AHB_BASE,true)==GPIO_INT_PIN_1){
                // do something
                GPIOIntClear(GPIO_PORTJ_AHB_BASE,GPIO_INT_PIN_1);
        }
}
```

➢ 中断驱动的简单按键和扫描按键

- 按键防抖所需的延时函数，降低了程序的效率，因为在这段延时的过程中，CPU无事可做。
- 如何充分利用CPU的时间？

```
void IntGPIOj(void)
{
//首先判断是那个引脚产生的中断
        if(GPIOIntStatus(GPIO_PORTJ_AHB_BASE,true)==GPIO_INT_PIN_0){
//延迟一段时间后，检查引脚是否还是低电平
            SysCtlDelay(g_ui32SysClock/30);
            //如果是低电平，则反转输出
            if(GPIOPinRead(GPIO_PORTJ_AHB_BASE,GPIO_PIN_0)==0){
                if(GPIOPinRead(GPIO_PORTN_BASE,GPIO_PIN_0)){
                    GPIOPinWrite(GPIO_PORTN_BASE,GPIO_PIN_0,0);
                }else{
                    GPIOPinWrite(GPIO_PORTN_BASE,GPIO_PIN_0,GPIO_PIN_0);
                }

            }
```

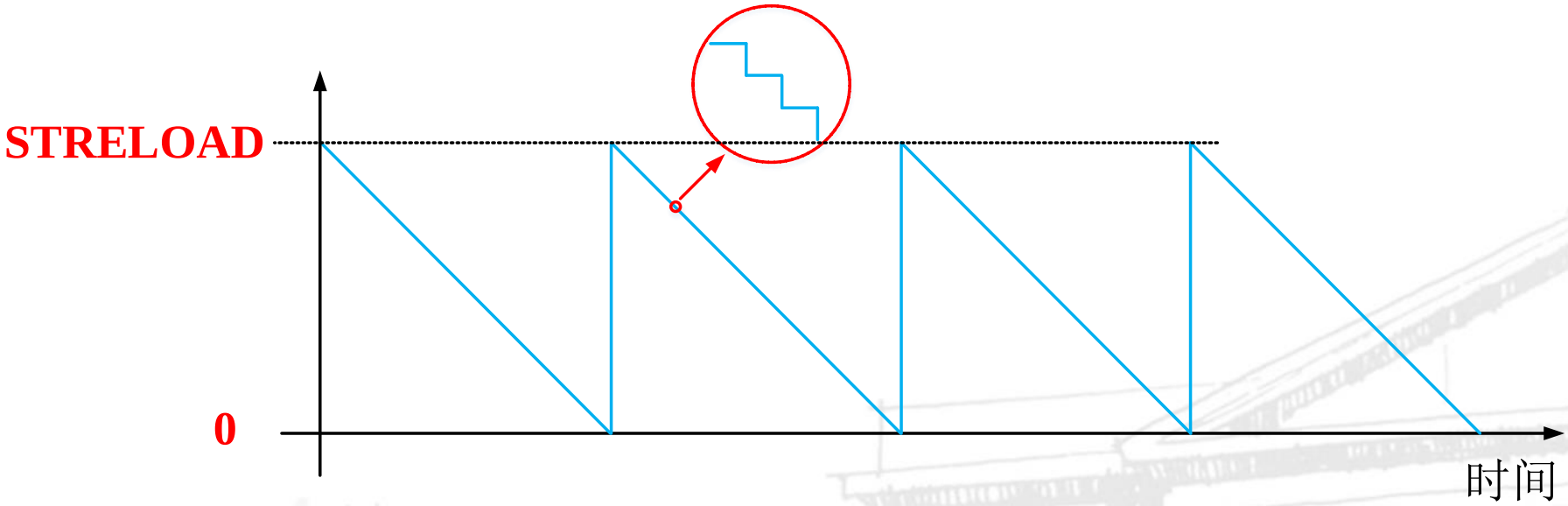
- 系统节拍定时器原理
- 系统节拍定时器的操作方法

➢ 系统节拍定时器SysTick

- 24位向下计数，使用系统时钟
- 用于实时操作系统（RTOS）的节拍计数器
- 用于精确延时
- 用于测量一段程序的执行时间。
- 由三个寄存器控制，使用简单：
  - 节拍控制和状态寄存器STCTRL：配置时钟、使能、使能中断、查看状态
  - 节拍计数初值寄存器STRELOAD: 计数器从这个值开始计数，向下计数，计到零后，又重新从这个值开始计数
  - 节拍当前值寄存器STCURRENT：计数器的当前值

➢ 系统节拍定时器SysTick的工作原理

- 使能后，系统节拍定时器的计数器开始向下递减计数，每过一个系统时钟周期，计数值减一
- 读取STCURRENT寄存器可以获知当前的计数值



➢ 控制和状态寄存器STCTRL

SysTick Control and Status Register (STCTRL)

Base 0xE000.E000
Offset 0x010
Type RW, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															COUNT
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved													CLK_SRC	INTEN	ENABLE
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- COUNT：计数器计数到零后，COUNT位置1；读取该寄存器，使COUNT清零；写STCURRENT寄存器，也可以使COUNT清零
- CLK_SRC：时钟源选择，0：PIOSC/4；1：系统时钟
- INTEN：中断使能控制，0：关闭SysTick中断；1：计数到0后，产生SysTick中断到NVIC
- ENABLE：计数器使能控制，0：关闭计数器；1：开启计数器，计数器从STRELOAD开始计数。

➢ 节拍计数初值寄存器STRELOAD

SysTick Reload Value Register (STRELOAD)

Base 0xE000.E000
Offset 0x014
Type RW, reset -

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved								RELOAD							
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

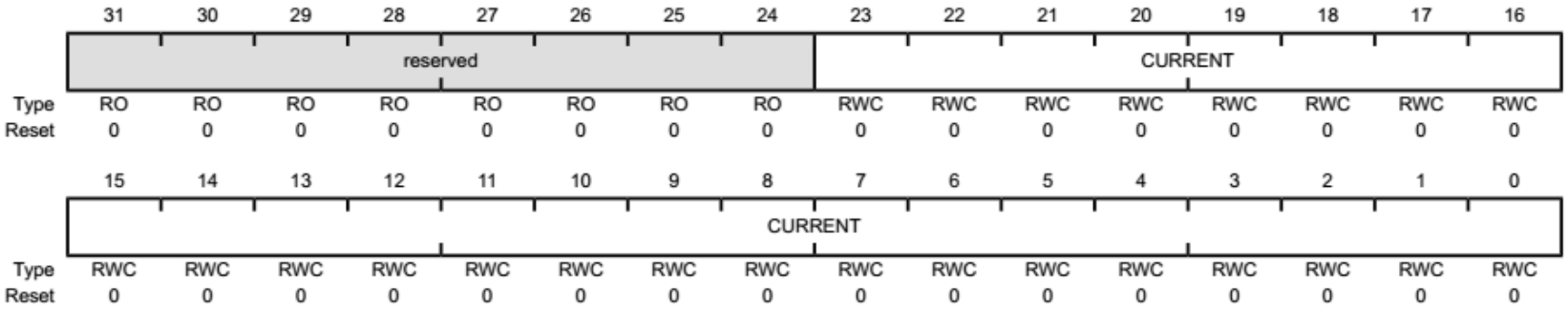
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RELOAD															
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- RELOAD： 24位计数初值，可以是1到0x00FF FFFF之间的任何数
- 如果写0的话，会关闭计数器，使计数器不工作，但不会产生中断
- 计数器从1数到0的过程中，可配置产生中断

➢ 节拍当前值寄存器STCURRENT

SysTick Current Value Register (STCURRENT)

Base 0xE000.E000
Offset 0x018
Type RWC, reset -



– 写入任何数据，都会使CURRENT清零，并清除STCTRL的COUNT位，然后从24位计数初值STRELOAD重新开始计数

➢ 系统节拍定时器的使用步骤

– 1 设置计数初值STRELOAD
– 2 向STCURRENT写任意值，使STCURRENT清零
– 3 设置STCTRL并开启计数

➢ 系统节拍定时器的使用步骤

– 1 设置计数初值STRELOAD

- 与节拍定时器有关的函数和宏定义在driverlib/systick.h和inc/hw_nvic.h中，因此用到节拍定时器后，要包含这两个文件
- 可以直接操作STRELOAD寄存器
  - HWREG(NVIC_ST_RELOAD) =……;
- 调用库函数SysTickPeriodSet

```
void
SysTickPeriodSet(uint32_t ui32Period)
{
    // Check the arguments.
    ASSERT((ui32Period > 0) && (ui32Period <= 16777216));
    // Set the period of the SysTick counter.
    HWREG(NVIC_ST_RELOAD) = ui32Period - 1;
}
```

  - 由于COUNT置位和中断请求都发生在从1数到0的时刻，而且计数第一个周期是从0变为STRELOAD，因此设置在STRELOAD中的计数值应该减1

➢ 系统节拍定时器的使用步骤

– 2 向STCURRENT写任意值，使STCURRENT清零

- HWREG(NVIC_ST_CURRENT)=0;

➢ 系统节拍定时器的使用步骤

– 3 设置STCTRL并开启计数

- 库函数提供了下面四个函数，控制计数器的使能和计数器中断的使能。
- void SysTickEnable (void)
- void SysTickDisable (void)
- void SysTickIntEnable (void)
- void SysTickIntDisable (void)

➢ 系统节拍定时器的使用步骤

– 3 设置STCTRL并开启计数

- 库函数提供了下面四个函数，控制计数器的使能和计数器中断的使能。
- SysTickEnable (void)

```
#define NVIC_ST_CTRL_CLK_SRC    0x00000004  // Clock Source
#define NVIC_ST_CTRL_ENABLE     0x00000001  // Enable
```

```
void
SysTickEnable(void)
{
    HWREG(NVIC_ST_CTRL) |= NVIC_ST_CTRL_CLK_SRC | NVIC_ST_CTRL_ENABLE;
}
```

- SysTickDisable (void)

```
void
SysTickDisable(void)
{
    HWREG(NVIC_ST_CTRL) &= ~(NVIC_ST_CTRL_ENABLE);
}
```

➢ 控制和状态寄存器STCTRL

SysTick Control and Status Register (STCTRL)

Base 0xE000.E000
Offset 0x010
Type RW, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															COUNT
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved													CLK_SRC	INTEN	ENABLE
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- COUNT：计数器计数到零后，COUNT位置1。读取该寄存器，使COUNT清零。写STCURRENT寄存器，也可以使COUNT清零。
- CLK_SRC：时钟源选择。0：PIOSC/4。1：系统时钟
- INTEN：中断使能控制。0：关闭SysTick中断。1：计数到0后，产生SysTick中断到NVIC。
- ENABLE：计数器使能控制。0：关闭计数器。1：开启计数器，计数器从STRELOAD开始计数。

➢ 系统节拍定时器的使用步骤

– 3 设置STCTRL并开启计数

- 库函数提供了下面四个函数，控制计数器的使能和计数器中断的使能。
- SysTickIntEnable (void)

```
#define NVIC_ST_CTRL_INTEN      0x00000002  // Interrupt Enable
```

```
void
SysTickIntEnable(void)
{
    HWREG(NVIC_ST_CTRL) |= NVIC_ST_CTRL_INTEN;
}
```

- SysTickIntDisable (void)

```
void
SysTickIntDisable(void)
{
    HWREG(NVIC_ST_CTRL) &= ~(NVIC_ST_CTRL_INTEN);
}
```

➢ 系统节拍定时器的使用步骤

– 注册中断服务函数SysTickIntRegister

```
#define FAULT_SYSTICK           15          // System Tick
```

```
void
SysTickIntRegister(void (*pfnHandler)(void))
{
    //
    // Register the interrupt handler, returning an error if an error occurs.
    //
    IntRegister(FAULT_SYSTICK, pfnHandler);

    //
    // Enable the SysTick interrupt.
    //
    HWREG(NVIC_ST_CTRL) |= NVIC_ST_CTRL_INTEN;
}
```

- SysTickIntRegister调用IntRegister来注册中断服务函数，然后开启节拍计数器的中断。
- IntRegister将向量表移动到SRAM中，然后修改向量表，注册中断服务函数。

系统异常列表

异常编号	异常类型	优先级	描述
1	复位	-3（最高）	复位
2	NMI	-2	不可屏蔽中断（外部NMI输入）
3	硬件错误	-1	所有错误都可能会引发，前提是相应的错误处理未使能
4	内存管理错误	可编程	存储器管理错误，存储器管理单元（MPU）冲突或访问非法位置
5	总线错误	可编程	当高级高性能总线（AHB）接口收到从总线的错误响应时产生
6	使用错误	可编程	程序错误或试图访问协处理器导致的错误
7~10	保留	NA	—
11	SVC	可编程	请求管理调用。一般用于OS环境且允许应用任务访问系统服务
12	调试监控	可编程	在使用基于软件的调试方案时，断点和监视点等调试事件的异常
13	保留	NA	—
14	PendSV	可编程	可挂起的服务调用。OS一般用该异常进行上下文切换
15	SYSTICK	可编程	当其在处理器中存在时，有定时器外设产生，可用于OS或简单的定时器外设
16	中断	可编程	由外设发出信号或软件请求产生，并通过NVIC（设置优先级）关联的异常

➢ 系统节拍定时器的使用步骤

– 获得当前计数值SysTickValueGet

```
uint32_t
SysTickValueGet(void)
{
    //
    // Return the current value of the SysTick counter.
    //
    return(HWREG(NVIC_ST_CURRENT));
}
```

使用中断

- void SysTickIntSrv();
- ……
- SysTickIntEnable();
- SysTickIntRegister(SysTickIntSrv);

- ……
- void SysTickIntSrv(){
- SysTickIntCount++;
- }

练习

- 初始化系统节拍定时器，每12000个系统时钟周期产生一次节拍计数器中断，中断服务函数为SysTickIntSrv。
- 如果要在SysTickIntSrv查看COUNT位，COUNT位为1，设置变量SysTickCount=1，否则SysTickCount=0，如何实现？
- 如果将计数器时钟改为PIOSC/4，写出实现的代码。

答案

- 初始化系统节拍定时器，每12000个系统时钟周期产生一次节拍计数器中断，中断服务函数为SysTickIntSrv。
- 如果要在SysTickIntSrv查看COUNT位，COUNT位为1，设置变量SysTickCount=1，否则SysTickCount=0，如何实现？
- 如果将计数器时钟改为PIOSC/4，写出实现的代码。

```
SysTickPeriodSet(12000);
HWREG(NVIC_ST_CURRENT)=0;
SysTickIntRegister(SysTickIntSrv);
SysTickEnable();

void SysTickIntSrv()
{
  SysTickCount=(HWREG(NVIC_ST_CTRL)&NVIC_ST_CTRL_COUNT)>>
16;
  }

HWREG(NVIC_ST_CTRL)&=~NVIC_ST_CTRL_CLK_SRC;
```

谢谢！