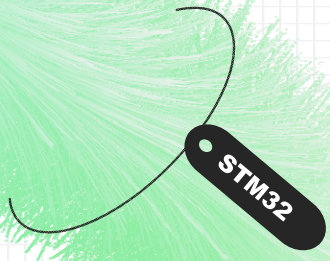
5. STM32 时钟和中断

北京科技大学
计算机与通信工程学院

目录

CONTENTS.



2

STM32时钟配置 (RCC)

01 讲解时钟树

02 重写时钟配置函数

3

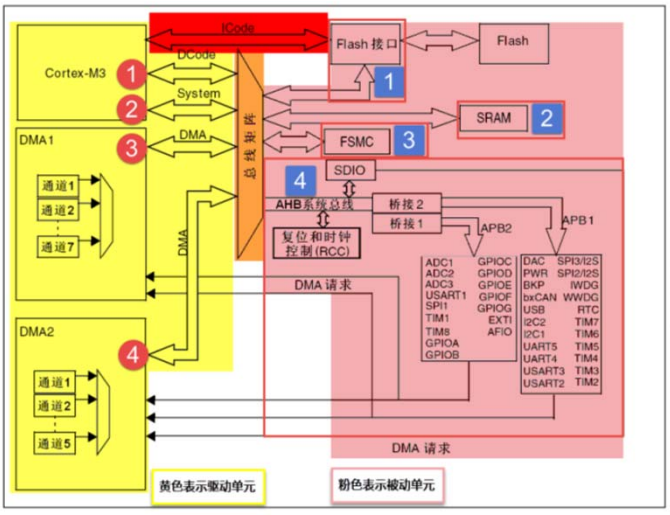
复位和时钟控制器

RCC （reset clock control ）主要作用:

- 设置系统时钟SYSCLK
- 设置AHB 分频因子（决定 HCLK 等于多少）
- 设置APB2 分频 因子（决定 PCLK2 等于多少）
- 设置 APB1 分频因子（决定 PCLK1 等于多少）
- 设置各个 外设的分频因子
- 控制 AHB、APB2 和 APB1 这三条总线时钟的开启
- 控制每个外设的时钟的开启

库函数对时钟的标准配置:

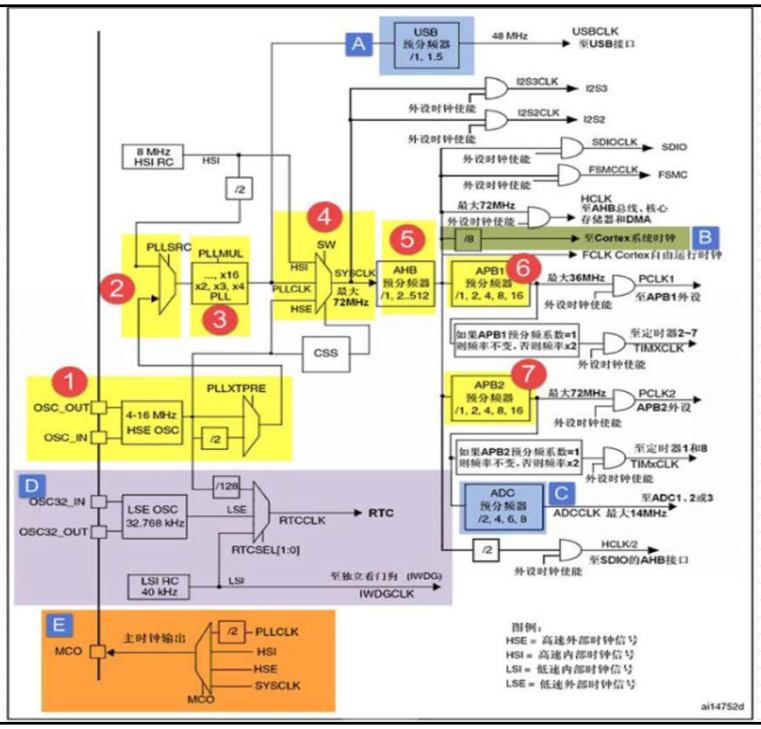
- PCLK2 = HCLK= SYSCLK=PLLCLK = 72M
- PCLK1=HCLK/2 = 36M。



STM32F10xx 系统框图

4

STM32 时钟树



5

时钟树 - HSE时钟

- HSE：High Speed External Clock signal，即高速的外部时钟。
- 来源：无源晶振（4-16M），通常使用8M。
- 控制：RCC_CR 时钟控制寄存器的位16：HSEON控制

6

时钟树 - HSI时钟

- HSI：Low Speed Internal Clock signal，高速的内部时钟。
- 来源：芯片内部，大小为8M，当HSE故障时，系统时钟会自动切换到HSI，直到HSE启动成功。
- 控制： RCC_CR 时钟控制寄存器的位0：HSION控制

7

时钟树 - 锁相环时钟

- 锁相环时钟：PLLCLK
- 来源：(HSI/2、HSE)经过倍频所得 。
- 控制：CFGR：PLLXTPRE、PLLMUL
- 注意：PLL时钟源头使用HSI/2的时候，PLLMUL最大只能是16，这个时候PLLCLK最大只能是64M，小于ST官方推荐的最大时钟72M。

8

时钟树 - 主系统时钟

- 系统时钟：SYSCLK，最高为72M（ST官方推荐的）
- 来源：HSI、HSE、PLLCLK。
- 控制：CFGR：SW
- 注意：通常的配置是SYSCLK=PLLCLK=72M。

9

时钟树 - HCLK时钟

- HCLK：AHB高速总线时钟，速度最高为72M。为AHB总线的外设提供时钟、为Cortex系统定时器提供时钟（SysTick）、为内核提供时钟（FCLK）。
- AHB：advanced high-performance bus。
- 来源：系统时钟分频得到，一般设置HCLK=SYSCLK=72M
- 控制： CFGR：HPRE

10

时钟树 - PCLK1时钟

- PCLK1：APB1低速总线时钟，最高为36M。为APB1总线的外设提供时钟。2倍频之后则为APB1总线的定时器2-7提供时钟，最大为72M。
- 来源：HCLK分频得到，一般配置PCLK1=HCLK/2=36M
- 控制： RCC_CFGR 时钟配置寄存器的PPRE1位

11

时钟树 - PCLK2时钟

PCLK2时钟

- PCLK2：APB2高速总线时钟，最高为72M。为APB1总线的外设提供时钟。为APB1总线的定时器1和8提供时钟，最大为72M。
- 来源：HCLK分频得到，一般配置PCLK1=HCLK=72M
- 控制： RCC_CFGR 时钟配置寄存器的PPRE2位

12

时钟树 - 其他时钟

RTC时钟

- RTC时钟：为芯片内部的RTC外设提供时钟。
- 来源：HSE_RTC（HSE分频得到）、LSE（外部32.768KHZ的晶体提供）、LSI（32KHZ）。
- 控制： RCC备份域控制寄存器RCC_BDCR：RTCSEL位控制

独立看门狗时钟：IWDGCLK，由LSI提供

13

时钟树主系统时钟讲解

MCO时钟输出

- MCO：microcontroller clock output，微控制器时钟输出引脚，由PA8复用所得。
- 来源：PLLCLK/2，HSE、HSI、SYSCLK
- 控制：CRGR：MCO

14

系统主时钟配置流程

配置时钟的起始点：SystemInit() -> SetSysClock() -> SetSysClockTo72()

```
#if defined (STM32F10X_LD_VL) || (defined STM32F10X_MD_VL) || (defined STM32F10X_HD_VL)
 /* #define SYSCLK_FREQ_HSE    HSE_VALUE */
  #define SYSCLK_FREQ_24MHz  24000000
 #else
 /* #define SYSCLK_FREQ_HSE    HSE_VALUE */
 /* #define SYSCLK_FREQ_24MHz  24000000 */
 /* #define SYSCLK_FREQ_36MHz  36000000 */
 /* #define SYSCLK_FREQ_48MHz  48000000 */
 /* #define SYSCLK_FREQ_56MHz  56000000 */
 #define SYSCLK_FREQ_72MHz  72000000
 #endif
```

SetSysClockTo72() 代码见

system_stm32f10x.c

```
1  static void SetSysClockTo72(void)
2  {
3      __IO uint32_t StartUpCounter = 0, HSEStatus = 0;
4
5      // ① 使能 HSE，并等待 HSE 稳定
6      RCC->CR |= ((uint32_t)RCC_CR_HSEON);
7
8      // 等待 HSE 启动稳定，并做超时处理
9      do {
10         HSEStatus = RCC->CR & RCC_CR_HSERDY;
11         StartUpCounter++;
12     } while ((HSEStatus == 0) &&(StartUpCounter !=HSE_STARTUP_TIMEOUT))
13
14     if ((RCC->CR & RCC_CR_HSERDY) != RESET) {
15         HSEStatus = (uint32_t)0x01;
16     } else {
17         HSEStatus = (uint32_t)0x00;
18     }
19     // HSE 启动成功，则继续往下处理
20     if (HSEStatus == (uint32_t)0x01) {
21
22         //------------------------------------------------------------
23         // 使能 FLASH 预存取缓冲区 */
24         FLASH->ACR |= FLASH_ACR_PRFTBE;
25
```

系统主时钟配置流程

```
26         // SYSCLK 周期与闪存访问时间的比例设置，这里统一设置成 2
27         // 设置成 2 的时候，SYSCLK 低于 48M 也可以工作，如果设置成 0 或者 1 的时候，
28         // 如果配置的 SYSCLK 超出了范围的话，则会进入硬件错误，程序就死了
29         // 0：0 < SYSCLK <= 24M
30         // 1：24< SYSCLK <= 48M
31         // 2：48< SYSCLK <= 72M */
32         FLASH->ACR &= (uint32_t)((uint32_t)~FLASH_ACR_LATENCY);
33         FLASH->ACR |= (uint32_t)FLASH_ACR_LATENCY_2;
34         //------------------------------------------------------------
35
36         // ② 设置 AHB、APB2、APB1 预分频因子
37         // HCLK = SYSCLK
38         RCC->CFGR |= (uint32_t)RCC_CFGR_HPRE_DIV1;
39         //PCLK2 = HCLK
40         RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE2_DIV1;
41         //PCLK1 = HCLK/2
42         RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE1_DIV2;
43
44         //③ 设置 PLL 时钟来源，设置 PLL 倍频因子，PLLCLK = HSE * 9 = 72 MHz
45         RCC->CFGR &= (uint32_t)((uint32_t)
46                                 ~(RCC_CFGR_PLLSRC
47                                   | RCC_CFGR_PLLXTPRE
48                                   | RCC_CFGR_PLLMULL));
49         RCC->CFGR |= (uint32_t)(RCC_CFGR_PLLSRC_HSE
50                                 | RCC_CFGR_PLLMULL9);
51
52         // ④ 使能 PLL
53         RCC->CR |= RCC_CR_PLLON;
54
55         // ⑤ 等待 PLL 稳定
56         while ((RCC->CR & RCC_CR_PLLRDY) == 0) {
57         }
58
59         // ⑥ 选择 PLL 作为系统时钟来源
60         RCC->CFGR &= (uint32_t)((uint32_t)~(RCC_CFGR_SW));
61         RCC->CFGR |= (uint32_t)RCC_CFGR_SW_PLL;
62
63         // ⑦ 读取时钟切换状态位，确保 PLLCLK 被选为系统时钟
64         while ((RCC->CFGR&(uint32_t)RCC_CFGR_SWS) != (uint32_t)0x08){
65         }
66     } else {// 如果 HSE 启动失败，用户可以在这里添加错误代码出来
67     }
68 }
```

16

使用库函数配置时钟示例 - HSI

HSI 必须 2 分频之后才能作为 PLL 的时钟来源，所以使用 HSI 时，最大的系统时 钟SYSCLK 只能是 HSI/2*16=4*16=64MHZ

函数调用举例：HSI_SetSysClock(RCC_PLLMul_9); 则设置系统时钟为：4MHZ * 9 = 36MHZ。

```
1 void HSI_SetSysClock(uint32_t pllmul)
2 {
3   __IO uint32_t HSIStartUpStatus = 0;
4
5  // 把 RCC 外设初始化成复位状态，这句是必须的
6   RCC_DeInit();
7
8   //使能 HSI
9   RCC_HSICmd(ENABLE);
10
11   // 等待 HSI 就绪
12   do{ HSIStartUpStatus = RCC->CR & RCC_CR_HSIRDY;}
13    while( HSIStartUpStatus ==0 )
14  // 只有 HSI 就绪之后则继续往下执行
15   if (HSIStartUpStatus == RCC_CR_HSIRDY) {
16   //-----------------------------------------------------//
17
18   // 使能 FLASH 预存取缓冲区
19       FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
20
21 // SYSCLK 周期与闪存访问时间的比例设置，这里统一设置成 2
22 // 设置成 2 的时候，SYSCLK 低于 48M 也可以工作，如果设置成 0 或者 1 的时候，
23 // 如果配置的 SYSCLK 超出了范围，则会进入硬件错误，程序就死了
24       // 0：0 < SYSCLK <= 24M
25       // 1：24< SYSCLK <= 48M
26       // 2：48< SYSCLK <= 72M
27       FLASH_SetLatency(FLASH_Latency_2);
28   //-------------------------------------------------------------//
29
```

17

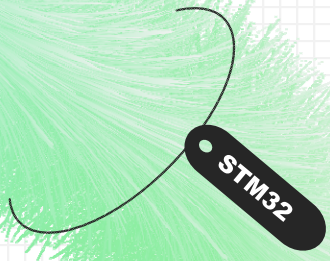
使用库函数配置时钟示例

```
30 // AHB 预分频因子设置为 1 分频，HCLK = SYSCLK
31       RCC_HCLKConfig(RCC_SYSCLK_Div1);
32
33 // APB2 预分频因子设置为 1 分频，PCLK2 = HCLK
34       RCC_PCLK2Config(RCC_HCLK_Div1);
35
36 // APB1 预分频因子设置为 1 分频，PCLK1 = HCLK/2
37       RCC_PCLK1Config(RCC_HCLK_Div2);
38
39 //-----------设置各种频率主要就是在这里设置----------------//
40 // 设置 PLL 时钟来源为 HSE，设置 PLL 倍频因子
41       // PLLCLK = 4MHz * pllmul
42       RCC_PLLConfig(RCC_PLLSource_HSI_Div2, pllmul);
43      //-- -------------------------------------------------------//
44
45 // 开启 PLL
46       RCC_PLLCmd(ENABLE);
47
48 // 等待 PLL 稳定
49       while (RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET) {
50       }
52 // 当 PLL 稳定之后，把 PLL 时钟切换为系统时钟 SYSCLK
53       RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
55 // 读取时钟切换状态位，确保 PLLCLK 被选为系统时钟
56       while (RCC_GetSYSCLKSource() != 0x08) {
57       }
58   } else {
59 // 如果 HSI 开启失败，那么程序就会来到这里，用户可在这里添加出错的代码处理
60 // 当 HSE 开启失败或者故障的时候，单片机会自动把 HSI 设置为系统时钟，
61 // HSI 是内部的高速时钟，8MHZ
62       while (1) {
63       }
64   }
65 }
```

18

目录

CONTENTS.



19

主讲内容

20

中断简介

- STM32 中断非常强大，每个外设都可以产生中断，所以中断的讲解放在哪一个外设里面去讲都不合适，这里单独抽出一章来做一个总结性的介绍。
- 本章如无特别说明，异常就是中断，中断就是异常。

21

中断类型

- 系统异常，体现在内核水平
- 外部中断，体现在外设水平

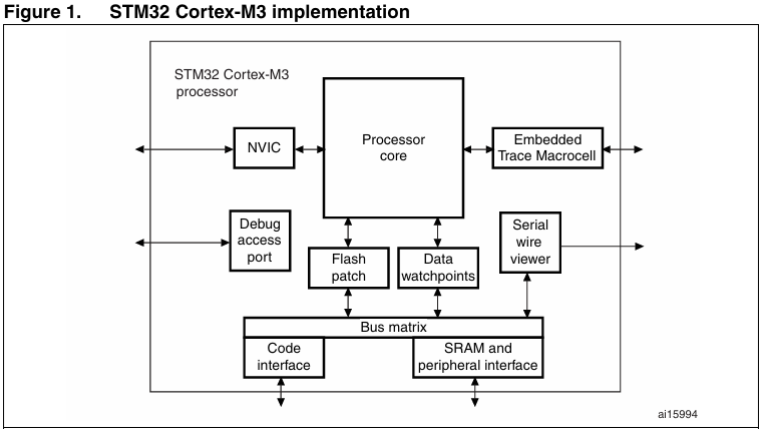
有关系统异常和外部中断的清单可查阅参考手册第9章的向量表部分。

22

NVIC简介

NVIC：嵌套向量中断控制器，属于内核外设，管理着包括内核和片上所有外设的中断相关的功能。

Figure 1. STM32 Cortex-M3 implementation



两个重要的库文件：core_cm3.h和misc.h

23

NVIC寄存器

NVIC寄存器简介，core_cm3.h定义

```
typedef struct {
    __IO uint32_t ISER[8];        // 中断使能寄存器
    uint32_t RESERVED0[24];
    __IO uint32_t ICER[8];        // 中断清除寄存器
    uint32_t RSERVED1[24];
    __IO uint32_t ISPR[8];        // 中断使能悬起寄存器
    uint32 t RESERVED2[24];
    __IO uint32_t ICPR[8];        // 中断清除悬起寄存器
    uint32 t RESERVED3[24];
    __IO uint32_t IABR[8];        // 中断有效位寄存器
    uint32 t RESERVED4[56];
    __IO uint8_t  IP[240];        // 中断优先级寄存器(8Bit wide)
    uint32 t RESERVED5[644];
    __O  uint32_t STIR;           // 软件触发中断寄存器
}  NVIC_Type;
```

24

中断优先级的定义

优先级设定：NVIC->IPRx

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
用于表达优先级				未使用，读回为 0			

优先级分组：SCB->AIRCR:PRIGROUP[10:8]

优先级分组	主优先级	子优先级	描述
NVIC_PriorityGroup_0	0	0-15	主-0bit，子-4bit
NVIC_PriorityGroup_1	0-1	0-7	主-1bit，子-3bit
NVIC_PriorityGroup_2	0-3	0-3	主-2bit，子-2bit
NVIC_PriorityGroup_3	0-7	0-1	主-3bit，子-1bit
NVIC_PriorityGroup_4	0-15	0	主-4bit，子-0bit

25

中断编程的顺序

1-使能中断请求

2-配置中断优先级分组

3-配置NVIC寄存器，初始化NVIC_InitTypeDef;

4-编写中断服务函数

26

中断编程

1.使能中断请求

如何使能，需要配置哪个寄存器?

由每个外设的相关中断使能位控制。

例如：

串口有发送完成中断，接收完成中断，这两个中断都由串口控制寄存器的相关中断使能位控制。

25.6.4 控制寄存器 1(USART_CR1)

地址偏移：0x0C

复位值：0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
res		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位6	**TCIE**：发送完成中断使能 (Transmission complete interrupt enable) 该位由软件设置或清除。 0：禁止产生中断； 1：当USART_SR中的TC为'1'时，产生USART中断。
位5	**RXNEIE**：接收缓冲区非空中断使能 (RXNE interrupt enable) 该位由软件设置或清除。 0：禁止产生中断； 1：当USART_SR中的ORE或者RXNE为'1'时，产生USART中断。
位4	IDLEIE：IDLE中断使能 (IDLE interrupt enable)

27

中断编程

2.中断优先级分组

如何配置，需要配置哪个寄存器

内核外设 SCB 的寄存器AIRCR (应用程序中断及复位控制寄存器)的PRIGROUP[10:8]位

设置优先级分组可调用库函数

NVIC_PriorityGroupConfig()实现，有关 NVIC 中断相关的库函数都在库文件 misc.c 和 misc.h 中。

4.4.4 Application interrupt and reset control register (SCB_AIRCR)

Address offset: 0x0C

Reset value: 0xFA05 0000

Required privilege: Privileged

The AIRCR provides priority grouping control for the exception model, endian status for data accesses, and reset control of the system.

To write to this register, you must write 0x5FA to the VECTKEY field, otherwise the processor ignores the write.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
VECTKEYSTAT[15:0](read)/ VECTKEY[15:0](write)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ENDIANNESS	Reserved				PRIGROUP			Reserved					SYSRESETREQ	VECTCLRACTIVE	VECTRESET
r					rw	rw	rw						w	w	w

SCB是System Control Block的缩写，用于系统的控制和管理,包含了多个寄存器，如AIRCR（Application Interrupt and Reset Control Register），用于控制系统的各种异常和中断处理。此外，SCB还负责时钟控制、电源管理等

28

中断编程

3.初始化 NVIC_InitTypeDef 结构体

1-NVIC_IRQChannel：中断源

2-NVIC_IRQChannelPreemptionPriority：抢占优先级

3-NVIC_IRQChannelSubPriority：子优先级

4-NVIC_IRQChannelCmd：使能或者失能

设置抢占优先级和子优先级，使能中断请求。NVIC_InitTypeDef 结构体在固件库头文件 misc.h 中定义。

中断源定义在 stm32f10x.h 头文件里面的 IRQn_Type 结构体定义。

IRQn_Type 中断源结构体

```
typedef enum IRQn {
    //Cortex-M3 处理器异常编号
    NonMaskableInt_IRQn       = -14,
    MemoryManagement_IRQn     = -12,
    BusFault_IRQn             = -11,
    UsageFault_IRQn           = -10,
    SVCall_IRQn               = -5,
    DebugMonitor_IRQn         = -4,
    PendSV_IRQn               = -2,
    SysTick_IRQn              = -1,
    //STM32 外部中断编号
    WWDG_IRQn                 = 0,
    PVD_IRQn                  = 1,
    TAMP_STAMP_IRQn           = 2,

    // 限于篇幅，中间部分代码省略，具体的可查看库文件 stm32f10x.h

    DMA2_Channel2_IRQn        = 57,
    DMA2_Channel3_IRQn        = 58,
    DMA2_Channel4_5_IRQn      = 59
} IRQn_Type;
```

29

中断编程

4.编写中断服务函数

1-中断服务函数名要怎么写？写错了怎么办？

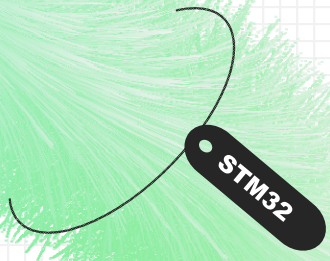
2-中断服务函数要写在什么地方？

在启动文件 startup_stm32f10x_hd.s 中预先为每个中断都写了一个中断服务函数，只是这些中断函数都是为空，为的只是初始化中断向量表。实际的中断服务函数都需要我们重新编写，为了方便管理我们把中断服务函数统一写在 stm32f10x_it.c 这个库文件中。

30

目录

CONTENTS.



31

内容

32

EXTI简介

EXTI：External interrupt / event controller

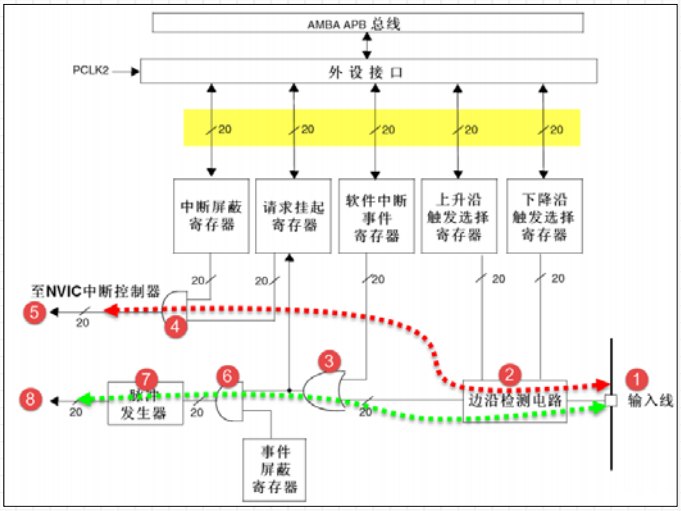
外部 中断/事件 控制器管理了控制器的 20个中断/事件线。

每个中断/事件线都对应有一个边沿检测器，可以实现输入信号的上升沿检测和下降沿的检测。

EXTI 可以实现对每个中断/事件线进行单独配置，可以单独配置为中断或者事件，以及触发事件的属性。

33

EXTI功能框图



EXTI 可分为两大部分功能，一个是产生中断，另一个是产生事件，这两个功能从硬件上就有所不同。

红色虚线是产生中断的电路流程。它是一个产生中断的线路，最终信号流入到 NVIC 控制器内。

绿色虚线是产生事件的电路流程。它是一个产生事件的线路，最终输出一个脉冲信号。

产生事件线路是在编号 3电路之后与中断线路有所不同，之前电路都是共用的。

34

EXTI功能框图

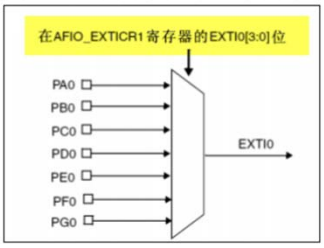
输入线

1、输入线总共有多少，具体是哪一些?

2、通过配置哪个寄存器来选择?

EXTI 中断/事件线

中断/事件线	输入源
EXTI0	PX0 (X 可为 A, B, C, D, E, F, G, H, I)
EXTI1	PX1 (X 可为 A, B, C, D, E, F, G, H, I)
EXTI2	PX2 (X 可为 A, B, C, D, E, F, G, H, I)
EXTI3	PX3 (X 可为 A, B, C, D, E, F, G, H, I)
EXTI4	PX4 (X 可为 A, B, C, D, E, F, G, H, I)
EXTI5	PX5 (X 可为 A, B, C, D, E, F, G, H, I)
EXTI6	PX6 (X 可为 A, B, C, D, E, F, G, H, I)
EXTI7	PX7 (X 可为 A, B, C, D, E, F, G, H, I)
EXTI8	PX8 (X 可为 A, B, C, D, E, F, G, H, I)
EXTI9	PX9 (X 可为 A, B, C, D, E, F, G, H, I)
EXTI10	PX10 (X 可为 A, B, C, D, E, F, G, H, I)
EXTI11	PX11 (X 可为 A, B, C, D, E, F, G, H, I)
EXTI12	PX12 (X 可为 A, B, C, D, E, F, G, H, I)
EXTI13	PX13 (X 可为 A, B, C, D, E, F, G, H, I)
EXTI14	PX14 (X 可为 A, B, C, D, E, F, G, H, I)
EXTI15	PX15 (X 可为 A, B, C, D, E, F, G, H, I)
EXTI16	PVD 输出
EXTI17	RTC 闹钟事件
EXTI18	USB 唤醒事件
EXTI19	以太网唤醒事件（只适用互联型）



EXTI0 输入源选择

35

EXTI初始化结构体

EXTI_InitTypeDef

1-EXTI_Line：用于产生 中断/事件 线
2-EXTI_Mode：EXTI模式（中断/事件）
3-EXTI_Trigger：触发（上/下/上下）
4-EXTI_LineCmd：使能或者失能（IMR/EMR）

标准库函数对每个外设都建立了一个初始化结构体，结构体成员用于设置外设工作参数，并由外设初始化配置函数，比如 EXTI_Init()调用，这些设定参数将会设置外设相应的寄存器，达到配置外设工作环境的目的。

初始化结构体和初始化库函数配合使用是标准库精髓所在，理解了初始化结构体每个成员意义基本上就可以对该外设运用自如了。

36

实验设计

1、PA0连接到EXTI用于产生中断，PA0的电平变化通过按键来控制

2、产生一次中断，LED反转一次

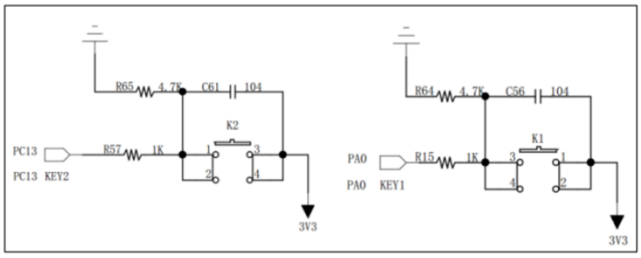


图 18-3 按键电路设计

37

编程要点

1-初始化要连接到EXTI的GPIO

2-初始化EXTI用于产生中断/事件

3-初始化NVIC，用于处理中断

4-编写中断服务函数

5-main函数

38

1-初始化要连接到EXTI的GPIO

```
//bsp_exti.h
#ifndef __BSP_EXTI_H
#define __BSP_EXTI_H

#include "stm32f10x.h"

#define KEY1_INT_GPIO_PIN          GPIO_Pin_0
#define KEY1_INT_GPIO_PORT         GPIOA
#define KEY1_INT_GPIO_CLK          RCC_APB2Periph_GPIOA

void EXIT_Key_Config(void);

#endif /* __BSP_EXTI_H */
```

```
//bsp_exti .c
void EXIT_Key_Config(void)
{
  GPIO_InitTypeDef  GPIO_InitStruct;

  // 初始化GPIO
  RCC_APB2PeriphClockCmd(KEY1_INT_GPIO_CLK, ENABLE);
  GPIO_InitStruct.GPIO_Pin = KEY1_INT_GPIO_PIN;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN_FLOATING;

  GPIO_Init(KEY1_INT_GPIO_PORT, &GPIO_InitStruct);
}
```

39

2-初始化EXTI用于产生中断/事件

```
//bsp_exti .c
void EXIT_Key_Config(void) {
        EXTI_InitTypeDef  EXTI_InitStruct;

        // 初始化EXTI
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
        GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource0); /* 选择 EXTI 的信号源 */
        EXTI_InitStruct.EXTI_Line = EXTI_Line0;
        EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;  /* EXTI 为中断模式 */
        EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Rising; /* 上升沿中断 */
        EXTI_InitStruct.EXTI_LineCmd = ENABLE;/* 使能中断 */
        EXTI_Init(&EXTI_InitStruct);
}
```

40

3-初始化NVIC，用于处理中断

```
static void NVIC_Configuration(void) {
NVIC_InitTypeDef NVIC_InitStructure;
/* 配置 NVIC 为优先级组 1 */
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
/* 配置中断源：按键 1 */
NVIC_InitStructure.NVIC_IRQChannel =  EXTI0_IRQn;
/* 配置抢占优先级：1 */
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
/* 配置子优先级：1 */
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
/* 使能中断通道 */
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
}
```

```
//bsp_exti .c
void EXIT_Key_Config(void) {
        ......
        /* 配置 NVIC 中断*/
        NVIC_Configuration();
        ......
}
```

41

4-编写中断服务函数

```
//stm32f10x_it.c
1  void EXTI0_IRQHandler(void)
2 {
3   //确保是否产生了 EXTI Line 中断
4    if (EXTI_GetITStatus(EXTI_Line0) != RESET) {
5     // LED1 取反
6      LED1_TOGGLE;
7    //清除中断标志位
8     EXTI_ClearITPendingBit(EXTI_Line0);
9    }
10 }
```

42

5-main函数

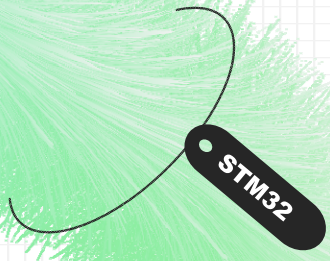
18-6 主函数

```
1  int main(void)
2  {
3      /* LED 端口初始化 */
4      LED_GPIO_Config();
5
6      /* 初始化 EXTI 中断，按下按键会触发中断，
7      *  触发中断会进入 stm32f10x_it.c 文件中的函数
8      *  KEY1_IRQHandler 和 KEY2_IRQHandler，处理中断，反转 LED 灯。
9      */
10     EXTI_Key_Config();
11
12     /* 等待中断，由于使用中断方式，CPU 不用轮询按键 */
13     while (1) {
14     }
15 }
```

43

目录

CONTENTS.



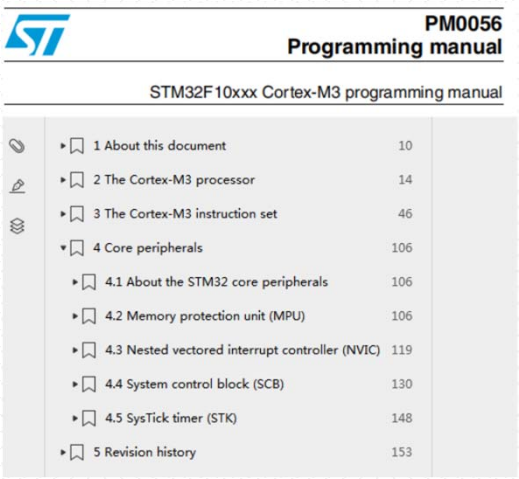
44

内容

45

SysTick简介

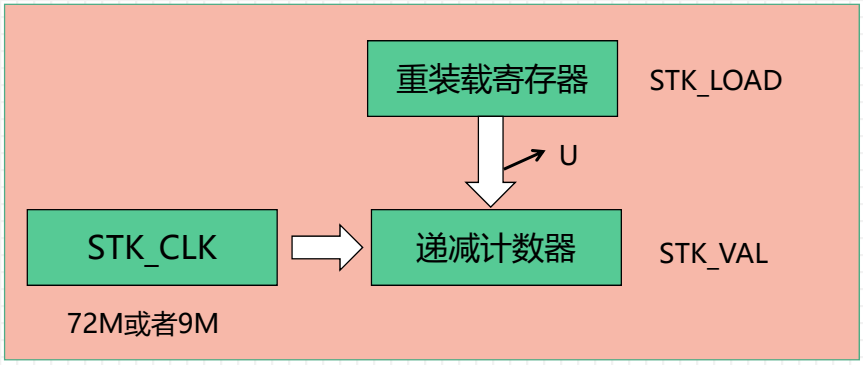
SysTick：系统定时器，24位，只能递减，存在于内核，嵌套在NVIC中，所有的Cortex-M内核的单片机都具有这个定时器。


PM0056
Programming manual

STM32F10xxx Cortex-M3 programming manual

46

SysTick功能框图



counter在时钟的驱动下，从reload初值开始往下递减计数到0，产生中断和置位COUNTFLAG标志。然后又从reload值开始重新递减计数，如此循环。

47

SysTick寄存器

表 18-2 SysTick 控制及状态寄存器

位段	名称	类型	复位值	描述
16	COUNTFLAG	R/W	0	如果在上次读取本寄存器后， SysTick 已经计到了 0，则该位为 1。
2	CLKSOURCE	R/W	0	时钟源选择位，0=AHB/8，1=处理器时钟 AHB
1	TICKINT	R/W	0	1=SysTick 倒数计数到 0 时产生 SysTick 异常请求， 0=数到 0 时无动作。也可以通过读取 COUNTFLAG 标志位来确定计数器是否递减到 0
0	ENABLE	R/W	0	SysTick 定时器的使能位

表 18-3 SysTick 重装载数值寄存器

位段	名称	类型	复位值	描述
23:0	RELOAD	R/W	0	当倒数计数至零时，将被重装载的值

表 18-4 SysTick 当前数值寄存器

位段	名称	类型	复位值	描述
23:0	CURRENT	R/W	0	读取时返回当前倒计数的值，写它则使之清

48

SysTick定时时间计算

- 1-t：一个计数循环的时间，跟reload和CLK有关
- 2-CLK：72M或者9M，由CTRL寄存器配置
- 3-RELOAD：24位，用户自己配置

49

SysTick定时时间计算

- t = reload * (1/clk)
- Clk = 72M时，t = (72) *(1/ 72 M)= 1US
- Clk = 72M时，t = (72000) *(1/ 72 M)= 1MS

时间单位换算:

1s = 1000ms = 1000 000 us = 1000 000 000ns

50

SysTick寄存器

SysTick寄存器结构体

在固件库文件：core_cm3.h中定义

```
typedef struct
{
  __IO uint32_t CTRL;                    /*!< 控制及状态寄存器 */
  __IO uint32_t LOAD;                    /*!< 重装载数值寄存器*/
  __IO uint32_t VAL;                     /*!< 当前数值寄存器*/
  __I  uint32_t CALIB;                   /*!< 校准寄存器 */
} SysTick_Type;
```

51

SysTick库函数

SysTick配置库函数

在固件库文件：core_cm3.h中定义

```
// 这个 固件库函数 在 core_cm3.h中
static __INLINE uint32_t SysTick_Config(uint32_t ticks)
{
  // reload 寄存器为24bit, 最大值为2^24
  if (ticks > SysTick_LOAD_RELOAD_Msk)  return (1);

  // 配置 reload 寄存器的初始值
  SysTick->LOAD  = (ticks & SysTick_LOAD_RELOAD_Msk) - 1;

  // 配置中断优先级为 1<<4-1 = 15, 优先级为最低
  NVIC_SetPriority (SysTick_IRQn, (1<<__NVIC_PRIO_BITS) - 1);

  // 配置 counter 计数器的值
  SysTick->VAL   = 0;

  // 配置systick 的时钟为 72M
  // 使能中断
  // 使能systick
  SysTick->CTRL  = SysTick_CTRL_CLKSOURCE_Msk |
                   SysTick_CTRL_TICKINT_Msk |
                   SysTick_CTRL_ENABLE_Msk;
  return (0);
}
```

52

SysTick库函数

SysTick配置库函数

在固件库文件：core_cm3.h中定义

```
static __INLINE void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)
{
  /* 设置优先级 for Cortex-M3 系统中断 */
  if(IRQn < 0) {
    SCB->SHP[((uint32_t)(IRQn) & 0xF)-4] = ((priority << (8 - __NVIC_PRIO_BITS)) & 0xff); }
  /* 设置优先级 for 外设中断  */
  else {
    NVIC->IP[(uint32_t)(IRQn)] = ((priority << (8 - __NVIC_PRIO_BITS)) & 0xff);    }
}
```

53

SysTick中断优先级

1-SysTick属于内核里面的外设，他的中断优先级跟片上的外设的中断优先级相比，哪个高?

2-systick中断优先级配置的是scb->shprx寄存器；而外设的中断优先级配置的是nvic->iprx，有优先级分组，有抢占优先级和子优先级的说法。

54

SysTick中断优先级

1-STM32里面无论是内核还是外设都是使用4个二进制位来表示中断优先级。

2-中断优先级的分组对内核和外设同样适用。当比较的时候，只需要把内核外设的中断优先级的四个位按照外设的中断优先级来分组来解析即可，即人为的分出抢占优先级和子优先级。

55

实验设计

1-编写一个微秒延时函数

2-编写一个毫秒延时函数

```
#include "stm32f10x.h"
#include "bsp_led.h"
#include "bsp_systick.h"

int main(void)
{
	// 来到这里的时候，系统的时钟已经被配置成72M。
	LED_GPIO_Config();

	while(1)
	{
		LED_G(OFF);
		SysTick_Delay_ms(500);

		LED_G(ON);
		SysTick_Delay_us(50000);
	}
}
```

56

实验设计

创建了两个文件：bsp_SysTick.c 和 bsp_ SysTick.h 文件
用来存放 SysTick 驱动程序及相关宏定义，中断服务函数放在 stm32f10x_it.h 文件中

```
// bsp_systick.h

#ifndef __BSP_SYSTICK_H
#define __BSP_SYSTICK_H

#include "stm32f10x.h"
#include "core_cm3.h"

void SysTick_Delay_us(uint32_t us);
void SysTick_Delay_ms(uint32_t ms);

#endif /* __BSP_SYSTICK_H */
```

57

实验设计

```
// bsp_systick.c
#include "bsp_systick.h"
void SysTick_Delay_us(uint32_t us)
{
        uint32_t i;
        SysTick_Config(72);

        for(i=0; i<us; i++)
        {        // 当计数器的值减小到 0 的时候，CRTL 寄存器的位 16 会置 1
                while( !((SysTick->CTRL) & (1<<16)) ); // 计数循环如果没结束，就等待
        }
        // 关闭 SysTick 定时器
        SysTick->CTRL &= ~ SysTick_CTRL_ENABLE_Msk; // 停止systick
}
```

58

实验设计

```
// bsp_systick.c
#include "bsp_systick.h"

void SysTick_Delay_ms(uint32_t ms)
{
        uint32_t i;
        SysTick_Config(72000);

        for(i=0; i<ms; i++)
        {        // 当计数器的值减小到 0 的时候，CRTL 寄存器的位 16 会置 1
                while( !((SysTick->CTRL) & (1<<16)) );
        }
        // 关闭 SysTick 定时器
        SysTick->CTRL &= ~ SysTick_CTRL_ENABLE_Msk;
}
```

以上可以工作了，但是轮询方式效率低

```
/* 配置SysTick 为10us中断一次 */
```

59

实验设计 - 中断方式

```
//main.c
#include "stm32f10x.h"
#include "bsp_SysTick.h"
#include "bsp_led.h"

int main(void)
{
  /* LED 端口初始化 */
    LED_GPIO_Config();
  /* 配置SysTick 为10us中断一次 */
  SysTick_Init();

 for(;;){
     LED1( ON );
    Delay_us(100000);       // 100000 * 10us = 1000ms
    LED1( OFF );
    Delay_us(100000);       // 100000 * 10us = 1000ms
 }
}
```

```
//bsp_SysTick.c
#include "bsp_SysTick.h"
#include "core_cm3.h"
#include "misc.h"

static __IO u32 TimingDelay;

/**
 * @brief us 延时程序,10us 为一个单位
 * @param
 * @arg nTime: Delay_us( 1 ) 则实现的延时为
1 * 10us = 10us
 * @retval 无
 */
 void Delay_us(__IO u32 nTime)
 {
 TimingDelay = nTime;

 while (TimingDelay != 0);
 }
```

60

实验设计 - 中断方式

```
//stm32f10x_it.c
#include "stm32f10x_it.h"

extern void TimingDelay_Decrement(void);

void SysTick_Handler(void)
{
   TimingDelay_Decrement();
}
```

```
//bsp_SysTick.c
#include "bsp_SysTick.h"
#include "core_cm3.h"
#include "misc.h"

static __IO u32 TimingDelay;
void TimingDelay_Decrement(void){
        if (TimingDelay != 0x00)
        {
                TimingDelay--;
        }
}
```

61

实验设计 - 中断方式

```
//bsp_SysTick.c
#include "bsp_SysTick.h"
#include "core_cm3.h"
#include "misc.h"

/**
  * @brief  启动系统滴答定时器 SysTick
  * @param  无
  * @retval 无
  */
void SysTick_Init(void)
{
        /* SystemCoreClock / 100000        10us中断一次
         * SystemCoreClock / 1000000       1us中断一次
         */
        if (SysTick_Config(SystemCoreClock / 100000))        // ST3.5.0库版本
        {
                /* Capture error */
                while (1);
        }
}
```

62