时钟设定、配置

函数

1. RCC_APB1PeriphClockCmd()
2. RCC_APB2PeriphClockCmd()

GPIO接口部分

GPIO配置、设定流程

函数

1. GPIO_Init()
2. GPIO_SetBits()
3. GPIO_ResetBits()
4. ReadInputDataBit()

结构

1. GPIO_InitTypeDef。其中字段有：

```
typedef struct{
    GPIO_Pin;
    GPIO_Speed;
    GPIO_Mode;
} GPIO_InitTypeDef;
```

示例

```
/*
    GPIO口配置示例。
    设定PA1口作为输出控制LED灯（低电平有效），PA0口作为按钮输入检测。
*/

// 时钟配置
void rcc_init()
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
}

// GPIO口配置
void gpio_init()
{
    GPIO_InitTypeDef gpio_init;
    gpio_init.GPIO_Pin = GPIO_Pin_1;
    gpio_init.GPIO_Speed = GPIO_Speed_50MHz;
    gpio_init.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(GPIOA, &gpio_init);
    gpio_init.GPIO_Pin = GPIO_Pin_0;
    gpio_init.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_Init(GPIOA, &gpio_init);
}

// 灯亮
void led_light()
{
    GPIO_ResetBits(GPIOA, GPIO_Pin_1);
}

// 灯灭
void led_dark()
{
    GPIO_SetBits(GPIOA, GPIO_Pin_1);
}

// 获取按键信息
char read_key()
{
    return !ReadInputDataBit(GPIOA, GPIO_Pin_0);
}

int main()
{
    rcc_init();
    gpio_init();
    while(1)
    {
        if(read_key())
            led_light();
        else led_dark();
    }
    return 0;
}
```

STM32中断部分

STM32 NVIC嵌套向量中断控制器配置

函数

1. NVIC_Init()
2. NVIC_PriorityGroupConfig()
3. 中断处理函数。

结构

1. NVIC_InitTypeDef。其中字段有：

```
typedef struct{
    NVIC_IRQChannel;
    NVIC_IRQChannelCmd;
    NVIC_IRQChannelPeermptionPriority;
    NVIC_IRQChannelSubPriority;
} NVIC_InitTypeDef;
```

2. 用于设定优先级组别：

```
NVIC_PriorityGroup_0        // 选择第0组
NVIC_PriorityGroup_1        // 选择第1组
NVIC_PriorityGroup_2        // 选择第2组
NVIC_PriorityGroup_3        // 选择第3组
NVIC_PriorityGroup_4        // 选择第4组
```

EXTI外部中断配置流程

函数

1. EXTI_Init()
2. EXTI_ClearITPendingBit();
3. GPIO_EXTILineConfig();

注意，外部中断线中，Px0 ~ Px4在NVIC的channel配置中，使用EXTI0_IRQn ~ EXTI4_IRQn。而Px5 ~ Px9使用EXTI9_5_IRQn，Px10 ~ Px15使用EXTI15_10_IRQn。

结构

1. EXTI_InitTypeDef。其中字段为：

```
typedef struct{
    EXTI_Line;
    EXTI_LineCmd;
    EXTI_Mode;
    EXTI_Trigger;
} EXTI_InitTypeDef;
```

示例

```
/*
    修改GPIO点亮LED灯程序，改为用中断实现。
*/

// 初始化EXTI外部中断
void exti_init()
{
    EXTI_InitTypeDef exti_init;
    exti_init.EXTI_Line = EXTI_Line0;   // PA0中断检测
    exti_init.EXTI_LineCmd = ENABLE;
    exti_init.EXTI_Mode = EXTI_Mode_Interrupt;
    exti_init.EXTI_Trigger = EXTI_Trigger_Rising_Falling;
    EXTI_Init(&exti_init);
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PortSource0);
}

// 初始化NVIC()
void nvic_init()
{
    NVIC_InitTypeDef nvic_init;
    nvic_init.NVIC_IRQChannel = EXTI0_IRQn;
    nvic_init.NVIC_IRQChannelCmd = ENABLE;
    nvic_init.NVIC_IRQChannelPeermptionPriority = 0;
    nvic_init.NVIC_IRQChannelSubPriority = 0;
    NVIC_Init(&nvic_init);
}

// 中断处理函数
void EXTI0_IRQHandler()
{
    if(read_key())
        led_light();
    else led_dark();

    EXTI_ClearITPendingBit(EXTI_Line0);
}

int main()
{
    gpio_init();
    exti_init();
    nvic_init();
    while(1);

    return 0;
}
```

串行通信

配置流程

函数

1. USART_Init()
2. USART_Cmd()
3. USART_SendData()
4. USART_ReceiveData()
5. USART_GetFlagStatus()

结构

1. USART_InitTypeDef。字段有：

```
typedef struct{
    USART_BaudRate;
    USART_WordLength;
    USART_StopBits;
    USART_Parity;
    USART_HardwareFlowControl;
    USART_Mode;
} USART_InitTypeDef;
```

示例

```
/*
    配置串口查询方式接收主机信息并发送回给主机。
*/

// 初始化时钟
void rcc_init()
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1|RCC_APB2Periph_GPIOA|RCC_APB2Periph_AFIO, ENAE
}

// 初始化PA9和PA10，其中PA9是输出口，PA10是输入口。
void gpio_init()
{
    // PA9 复用TX引脚为输出口
    GPIO_InitTypeDef gpio_init;
    gpio_init.GPIO_Pin = GPIO_Pin_9;
    gpio_init.GPIO_Mode = GPIO_Mode_AF_PP;      // 复用推挽输出
    gpio_init.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &gpio_init);

    // PA10 复用RX引脚为输入口
    gpio_init.GPIO_Pin = GPIO_Pin_10;
    gpio_init.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &gpio_init);
}

void usart_init()
{
    USART_InitTypeDef usart_init = {
        .USART_BaudRate = 9600,
        .USART_WordLength = USART_WordLength_8b,
        .USART_StopBits = USART_StopBits_1,
        .USART_Parity = USART_Parity_No,
        .USART_HardwareFlowControl = USART_HardwareFlowControl_None,
        .USART_Mode = USART_Mode_TX|USART_Mode_RX,
    }

    USART_Init(USART1, &usart_init);
    USART_Cmd(USART1, ENABLE);
}

void usart_send_byte(unsigned char data)
{
    USART_SendData(data);

    while(!USART_GetFlagStatus(USART1, USART_FLAG_TC));
}

unsigned char usart_recv_byte()
{
    while(!USART_GetFlagStatus(USART1, USART_FLAG_RXNE));

    return USART_ReceiveData(USART1);
}
```

```
int main()
{
    rcc_init();
    gpio_init();
    usart_init();
    while(1)
    {
        data = usart_recv_byte();
        usart_send_byte(data);
    }

    return 0;
}
```

定时器

定时器的配置流程

函数

1. TIM_TimeBaseInit()
2. TIM_OCxInit()
3. TIM_ICInit()
4. TIM_ITConfig()：配置中断
5. TIM_ETRClockMode1Config()
6. TIM_Cmd()
7. TIM_GetITStatus()
8. TIM_ClearITPendingBit()
9. TIM_OCxPolarityConfig()
10. TIM_PWMIConfig()
11. TIM_GetCapture1()
12. TIM_GetCapture2()

结构

1. TIM_TimeBaseInitTypeDef。字段为：

```
typedef struct{
    TIM_Period;
    TIM_Prescaler;
    TIM_ClockDivision;
    TIM_CounterMode;
    TIM_RepetitionCounter;
} TimeBaseInitTypeDef;
```

2. TIM_OCInitTypeDef。字段为：

```
typedef struct{
    u16 TIM_OCMode;
    u16 TIM_OutputState;
    u16 TIM_OutputNState;
    u16 TIM_Pulse;
    u16 TIM_OCPolarity;
    u16 TIM_OCNPolarity;
    u16 TIM_OCIdleState;
    u16 TIM_OCNIdleState;
} TIM_OCInitTypeDef
```

3. TIM_ICInitTypeDef。字段为：

```
typedef struct{
    u16 TIM_Channel;
    u16 TIM_ICPolarity;
    u16 TIM_ICSelection;
    u16 TIM_ICScaler;
    u16 TIM_ICFilter;
} TIM_ICInitTypeDef;
```

示例

```
/*
    例程1：通过定时器实现1s的延迟，控制LED1灯每秒闪烁一次。
*/

// TIM1时钟模块使能
void rcc_init()
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1|RCC_APB2Periph_GPIOA, ENABLE);
}

// 配置TIM时钟模块
void tim_init()
{
    TIM_TimeBaseInitTypeDef tim_init;
    tim_init.TIM_Period = 7200 - 1;
    tim_init.TIM_Prescaler = 10000 - 1;
    tim_init.TIM_ClockDivision = TIM_CKD_DIV1;
    tim_init.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM1, &tim_init);
    TIM_Cmd(TIM1);
}

// 配置定时器中断
void nvic_init()
{
    NVIC_InitTypeDef nvic_init;
    TIM_IT_Config(TIM1, TIM_IT_Update, ENABLE);
    nvic_init.NVIC_IRQChannel = TIM1_UP_IRQn;
    nvic_init.NVIC_IRQChannelCmd = ENABLE;
    nvic_init.NVIC_IRQChannelPeermptionPriority = 0;
    nvic_init.NVIC_IRQChannelSubPriority = 0;
    NVIC_Init(&nvic_init);
}

// 中断处理函数
void TIM1_UP_IRQHandler()
{
    if(TIM_GetFlagStatus(TIM1, TIM_IT_Update))
        led_twinkle();

    TIM_ClearITPendingBit(TIM1, TIM_IT_Update);
}


/*
    例程2：计数器功能。
*/

// TIM2模块时钟使能

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

// 初始化计数器
void TIM2_Configuration()
```

```
{
    TIM_TimeBaseInitTypeDef tim_init;
    tim_init.TIM_Period = 0xFFFFFF;
    tim_init.TIM_Prescaler = 0;
    tim_init.TIM_ClockDivision = TIM_CKD_DIV1;
    tim_init.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM2, &tim_init);
    TIM_ETRClockMode1Config(TIM2, TIM_ExtTRGPrescaler_OFF, TIM_ExtTRGPolarity_NonInverted, 0);
    TIM_Cmd(TIM2, ENABLE);
}

// 初始化中断

void nvic_init()
{
    NVIC_InitTypeDef nvic_init;
    TIM_ITConfig(TIM2, TIM_IT_Trigger, ENABLE);
    nvic_init.NVIC_IRQChannel = TIM2_IRQn;
    nivc_init.NVIC_IRQChannelCmd = ENABLE;
    nvic_init.NVIC_IRQChannelPreemptionPriority = 0;
    nvic_init.NVIC_IRQChannelSubPriority = 0;
    NVIC_Init(&nvic_init);
}

void TIM2_IRQHandler()
{
    if(TIM_GetITStatus(TIM2, TIM_IT_Trigger))
        usart_send_byte(TIM_GetCounter(TIM2));

    TIM_ClearITPendingBit(TIM2, TIM_IT_Trigger);
}

/*
    例程3：实现对按钮按下去的时间长度进行测量。
*/

// 使能TIM2时钟模块
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

void TIM2_Configuration()
{
    TIM_TimeBaseInitTypeDef tim_init();
    tim_init.TIM_Period = 72 - 1;
    tim_init.TIM_Prescaler = 0;
    tim_init.TIM_ClockDivision = TIM_CKD_DIV1;
    tim_init.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM1, &tim_init);
    TIM_Cmd(TIM1, ENABLE);

    TIM_ICInitTypeDef tim_ic_init;
    tim_ic_init.TIM_Channel = TIM_Channel_1;
    tim_ic_init.TIM_ICFilter = 0;
    tim_ic_init.TIM_ICPolarity = TIM_ICPolarity_Falling;
    tim_ic_init.TIM_ICSelection = TIM_ICSelection_DirectT1;
    tim_ic_init.TIM_ICPrescaler = TIM_ICPSC_DIV1;
```

```
    TIM_ICInit(TIM1, &tim_ic_init);

    NVIC_InitTypeDef nvic_init;
    TIM_ITConfig(TIM2, TIM_IT_Update|TIM_IT_CC1, ENABLE);
    nvic_init.IRQChannel = TIM2_IRQn;
    nvic_init.IRQChannelCmd = ENABLE;
    nvic_init.IRQChannelPeermptionPriority = 0;
    nvic_init.IRQChannelSubPriority = 0;
    NVIC_Init(&nvic_init);
}

extern struct capture_st m_capture;

void TIM2_IRQHandler()
{
    if(TIM_GetFlagStatus(TIM2, TIM_IT_Update))
    {
        if(m_capture.m_start == 1)
            m_capture.period_count++;
        TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
    }
    if(TIM_GetFlagStatus(TIM2, TIM_IT_CC))
    {
        if(m_capture.m_start == 0)
        {
            m_capture.m_start = 1;
            m_capture.period_count = 0;
            m_capture.ccr_value = 0;
            TIM_OC1PolarityConfig(TIM2, TIM_ICPolarity_Rising);
            TIM_SetCounter(TIM2, 0);
        }
        else
        {
            TIM_ITConfig(TIM2, TIM_IT_Update|TIM_IT_CC1, DISABLE);
            TIM_OC1PolarityConfig(TIM2, TIM_ICPolarity_Falling);
            m_capture.m_finish = 1;
            m_capture.m_start = 0;
            m_capture.ccr_value = TIM2->CCR;
        }
        TIM_ClearITPendingBit(TIM2, TIM_IT_CC1);
    }
}

/*
    例程4：PWM输出呼吸灯效果。
*/

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

void TIM2_Configuration()
{
    TIM_TimeBaseInitTypeDef tim_init;
    tim_init.TIM_Period = 1024 - 1;
    tim_init.TIM_Prescaler = 200 -1;
    tim_init.TIM_ClockDivision = TIM_CKD_DIV1;
```

```
    tim_init.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM2, &tim_init);

    TIM_OCInitTypeDef tim_oc_init;
    tim_oc_init.TIM_OCMode = TIM_OCMode_PWM1;
    tim_oc_init.TIM_OCPolarity = TIM_OCPolarity_Low;
    tim_oc_init.TIM_OutputState = TIM_OutputState_Enable;
    tim_oc_init.TIM_Pulse = 0;
    TIM_OC2Init(TIM2, &tim_oc_init);

    NVIC_InitTypeDef nvic_init;
    TIM_ClearFlag(TIM2, TIM_IT_Update);
    TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
    nvic_init.NVIC_IRQChannel = TIM2_IRQn;
    nvic_init.NVIC_IRQChannelCmd = ENABLE;
    nvic_init.NVIC_IRQChannelPreemptionPriority = 0;
    nvic_init.NVIC_IRQChannelSubPriority = 0;
    NVIC_Init(&nvic_init);
    TIM_Cmd(TIM2, ENABLE);
}

void TIM2_IRQHandler()
{
    if(TIM_GetFlagStatus(TIM2, TIM_IT_Update) == Set)
    {
        if(!--period_class)
        {
            period_class = 10;
            if(!++data_index >= 110)
                data_index = 0;
            TIM_SetCompare2(TIM2, data[data_index]);
        }
        TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
    }
}

/*
    通过PA6引脚作为输出产生PWM，然后PA8对PA6产生的PWM波形进行测量。
*/

void gpio_init()
{
    GPIO_InitTypeDef gpio_init;
    gpio_init.GPIO_Pin = GPIO_Pin_6;
    gpio_init.GPIO_Mode = GPIO_Mode_AF_PP;
    gpio_init.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &gpio_init);
    gpio_init.GPIO_Pin = GPIO_Pin_8;
    gpio_init.GPIO_Mode = GPIO_Mode_IPD;
    GPIO_Init(GPIOA, &gpio_init);
}

void TIM_Configuration()
{
    TIM_TimeBaseInitTypeDef tim_init;
```

```
    TIM_ICInitTypeDef tim_ic_init;
    TIM_OCInitTypeDef tim_oc_init;
    tim_init.TIM_Period = 8 - 1;
    tim_init.TIM_Prescaler = 72 - 1;
    tim_init.TIM_ClockDivision = TIM_CKD_DIV1;
    tim_init.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM3, &tim_init);

    tim_oc_init.TIM_OCMode = TIM_OCMode_PWM1;
    tim_oc_init.TIM_OCPolarity = TIM_OCPolarity_High;
    tim_oc_init.TIM_OCIdleState = TIM_OCIdleState_Reset;
    tim_oc_init.TIM_OutputState = TIM_OutputState_Enable;
    tim_oc_init.TIM_OutputNState = TIM_OutputNState_DISABLE;
    tim_oc_init.TIM_Pulse = 4;
    TIM_OC1Init(TIM3, &tim_oc_init);
    TIM_Cmd(TIM3, ENABLE);

    tim_in_init.TIM_Channel = TIM_Channel_1;
    tim_in_init.TIM_ICPolarity = TIM_ICPolarity_Rising;
    tim_in_init.TIM_ICPrescaler = TIM_ICPSC_DIV1;
    tim_in_init.TIM_ICSelection = TIM_ICSelection_DirectT1;
    tim_ic_init.TIM_ICFilter = 0;
    TIM_PWMIConfig(TIM1, &tim_ic_init);

    TIM_SelectInputTrigger(TIM1, TIM_TS_TI1FP1);
    TIM_SelectSlaveMode(TIM1, TIM_SlaveMode_Enable);
    TIM_SelectMasterSlaveMode(TIM1, TIM_MasterSlaveMode_Enable);
    TIM_ClearFlag(TIM1, TIM_FLAG_CC1);
    nvic_config();
    TIM_ITConfig(TIM1, TIM_IT_CC1, ENABLE);
    TIM_Cmd(TIM1, ENABLE);
}

void nvic_config(void)
{
    NVIC_InitTypeDef nvic_init;
    nvic_init.NVIC_IRQChannel = TIM1_CC_IRQn;
    nvic_init.NVIC_IRQChannelCmd = ENABLE;
    nvic_init.NVIC_IRQChannelPreemptionPriority = 0;
    nvic_init.NVIC_IRQChannelSubPriority = 0;
    NVIC_Init(&nvic_init);
}

void TIM1_CC_IRQHandler()
{
    uint16_t ic1 = TIM_GetCapture1(TIM1);
    uint16_t ic2 = TIM_GetCapture2(TIM1);

    if(ic1)
        printf("%d %d\n", ic1 + 1, ic2 + 1);

    TIM_ClearITPendingBit(TIM1, TIM_IT_CC1);
}
```

ADC

ADC配置流程

函数

1. ADC_Init()
2. ADC_StartCalibration()
3. ADC_SoftwareStartConvCmd()
4. ADC_RegularChannelConfig()
5. ADC_GetConversionValue()

结构

1. ADC_InitTypeDef结构。字段如下：

```
typedef struct
{
    u32 ADC_Mode;           // ADC工作模式（独立、双ADC）
    FunctionalState ADC_ScanConvMode;   // ADC转换工作在扫描还是单次模式
    FunctionalState ADC_ContinuousConvMode;     // 工作在连续模式还是单次模式
    u32 ADC_ExternalTrigConv;   // 使用外部触发规则来启动规则通道的模数转换
    u32 ADC_DataAlign;      // ADC数据向左边对齐还是向右边对齐
    u8 ADC_NbrOfChannel;    // 按顺序进行转换的ADC通道数目
} ADC_InitTypeDef;
```

示例

```
/*
    利用滑动变阻器产生的电压，通过STM32芯片的PC0引脚采集。
    并通过串口把这个电压值传送到电脑。
*/

void adc_init()
{
    ADC_InitTypeDef adc_init;
    ADC_DeInit(ADC1);
    adc_init.ADC_Mode = ADC_Mode_Independent;
    adc_init.ADC_ScanConvMode = DISABLE;
    adc_init.ADC_ContinuousConvMode = ENABLE;
    adc_init.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    adc_init.ADC_DataAlign = ADC_DataAlign_Right
    adc_init.ADC_NbrOfChannel = 1;
    ADC_Init(ADC1, &adc_init);
    RCC_ADCCLKConfig(RCC_PCLK2_Div2);
    ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 1, ADC_SampleTime_239Cycles5);
    ADC_Cmd(ADC1, ENABLE);
    ADC_ResetCalibrationsStatus(ADC1);
    while(ADC_GetCalibrationsStatus(ADC1));
    ADC_StartCalibrationStatus(ADC1);
    while(ADC_GetCalibrationsStatus(ADC1));
}

int main()
{
    rcc_init();
    gpio_init();
    usart_init();
    adc_init();
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
    while(1)
    {
        if(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC))
        {
            unsigned char voltage = 3.3f/4096 * ADC_GetConverionValue(ADC1);
            usart_send_data(voltage);
            delay_moment();
        }
    }
    return 0;
}
```