微嵌第八章习题参考答案

8.3 嵌入式的开发与通用计算机系统相比，有何独特之处？交叉开发环境的主要组成部分和特点是什么？

与通用计算机应用系统的开发相比，嵌入式系统的开发环境、开发工具和调试方式都有着明显区别。对于通用计算机应用系统开发而言，系统的开发机器即是系统的运行机器，系统的开发环境即是系统的运行环境。而对于嵌入式系统开发而言，系统的开发机器不是系统的运行机器，系统的开发环境不是系统的运行环境。这就需要专门的开发环境、开发工具和调试方法。

嵌入式系统的开发环境称为交叉开发环境(cross development environment)，主要由宿主机 host、目标机 target 及它们之间的连接构成。

嵌入式系统的开发环境中，宿主机和目标机（嵌入式系统）是不同的机器，嵌入式软件在宿主机上使用嵌入式开发工具进行编写、编译、链接和定位，生成可在目标机上执行的二进制代码，然后通过 JTAG 接口、串口或网口将代码下载到目标机（嵌入式系统）上调试，调试完成后，将最终调试好的二进制代码烧写到目标机（嵌入式系统）微处理器的 ROM 中运行。

8.7 何为引脚功能复用？有何意义？如何实现，请以 STM32F103 为例，举例具体说明。

引脚功能复用就是指特意将某几个功能分配到一个引脚，通过编程分别将芯片不同功能引出。

意义：大大减少了引脚数，解决了引脚资源不够的问题，方便更高效地利用引脚资源、降低成本以及焊装的复杂度。

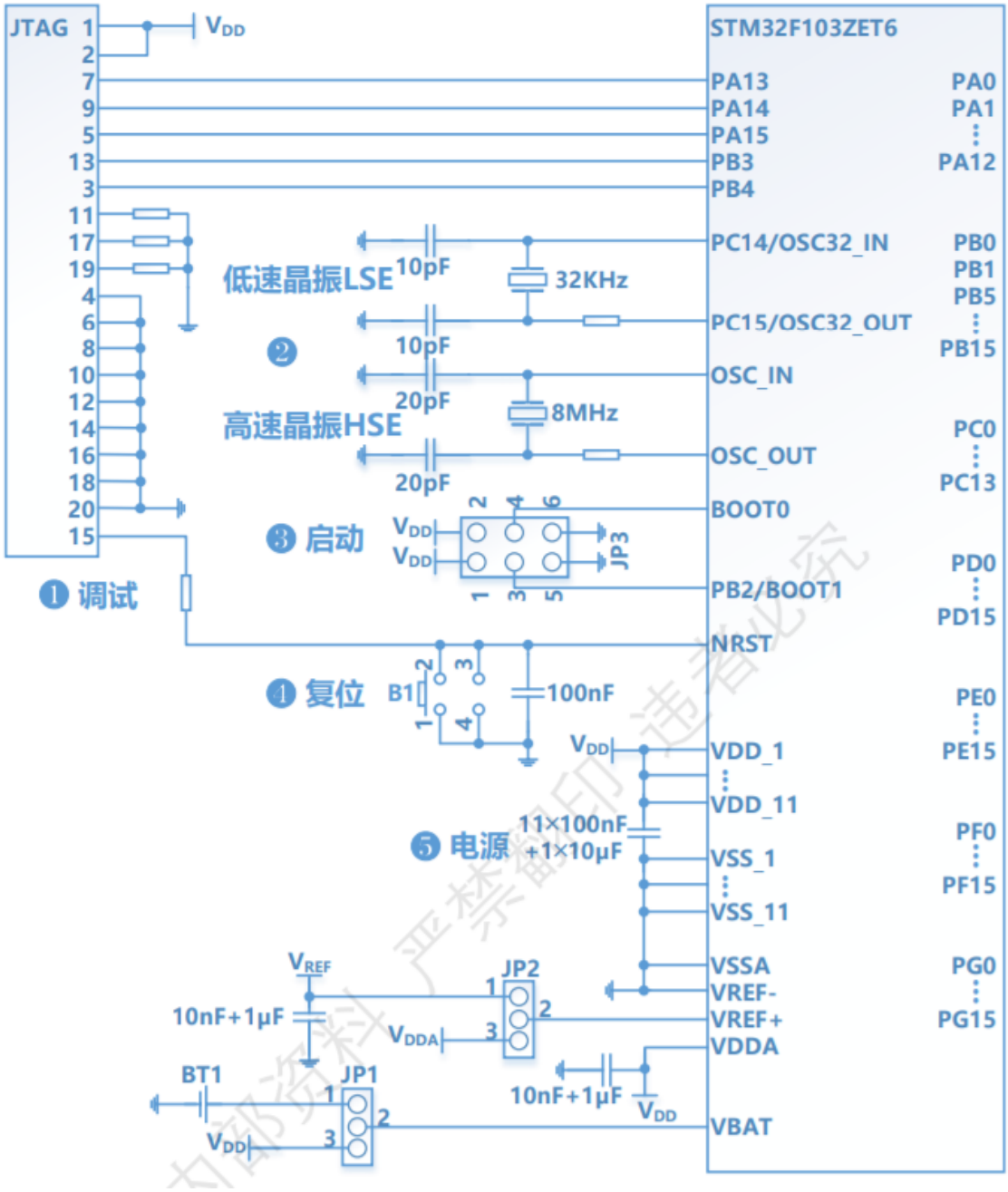
实现方法：配置好引脚复用的具体功能；在实际使用时，通过PINSELx编程实现对多路开关的控制，连通引脚与某功能模块，实现引脚功能复用。

以 STM32F103RCT6 为例，引脚 PB10，主功能是 PB10，默认复用功能是 I2C2 的时钟端 SCL 和 USART3 的发送端 Tx。

8.11 请设计一个 STM32 最小系统。

以 STM32F103 微处理器为例，典型的最小系统如图所示，包含以下功能部件：STM32芯片、电源电路、复位电路、时钟系统、调试和下载电路及启动电路。

电源：3.3V（一般可由供电电源 12V 通过可调稳压电路及低压差稳压芯片转为 3.3V）；
复位：手动复位按键，复位信号低有效；
时钟：外接晶振频率为 8MHz（高速 HSE）及 32kHz（低速 LSE）两个时钟源；
调试下载：通过此电路连接上位机、仿真器与目标板，下载和调试程序；
启动：可选择从用户 Flash 或系统 Flash 或片内 SRAM 上运行代码。



8.12 STM32F103 的复位电路有何功能？常见的复位方式有哪些？

当微处理器上电时，电压不是直接跳变到微处理器可工作的范围(如 3.3V)，而是一个逐步上升的过程。此时，微处理器无法正常工作，会引起芯片内程序无序执行。同样的情况也会发生在微处理器的供电电压波动不稳定时。因此需要复位电路给它延时，使微处理器保持复位，暂不进入工作状态，防止 CPU 执行错误指令，确保 CPU 及各部件处于确定的初始状态，直至电压稳定。复位电路的设计直接影响到系统稳定性和可靠性。未添加复位电路或复位电路设计不可靠可能会引起“死机”及“程序跑飞”等现象。

STM32F10x 支持 3 种复位形式，分别为**系统复位**、**电源复位**和**备份区域复位**。

8.16 参照 STM32 的时钟树，请做以下思考，并解释 1）设计成这种形式的主要目的和理由为何？2）从左至右，相关时钟依次可分为大致哪几种？3）时钟输出的使能有何意义，并以一个实例（如 TIM）说明一下大体流程。

1. STM32为了实现低功耗，设计了一个功能完善但略显复杂的时钟系统。因为有倍频、分频和一系列外设时钟的开关，STM32 的时钟树才会显得如此复杂。首先，倍频是考虑到电磁兼容性，如果直接外接一个 72MHz 的晶振，过高的振荡频率会给制作电路板带来难度。其次，分频是因为 STM32F103 各个片上外设的工作频率不尽相同，既有高速外设又有低速外设，需要把高速外设和低速外设分开管理，如同 PC 中的北桥和南桥一样。最后，每个 STM32F103 外设都配备了时钟开

关，当使用某个外设时，一定要打开该外设的时钟；而当不使用某个外设时，可以把这个外设时钟关闭，从而降低 STM 32 的整体功耗。

2. 从左至右，相关时钟依次可主要分为 3 种：输入时钟、系统时钟和由系统时钟分频得到的其他时钟
3. 时钟输出中大多带使能控制，如AHB总线、内核、各种APB1外设、APB2外设等时钟。当需使用它们时，须先使能对应时钟。
   流程如下：使用HSE为例子
   1. 使用外部晶振，并等待外部晶振起振;
   2. 采用外部高速晶振HSE做PLL源，并配置PLL;
   3. PLL输出时钟信号通过AHB分频器分频后送给各模块使用，AHB分频器可选择1、2、4、8、16、64、128、256、512分频;
   4. AHB分频器输出的时钟送给5大模块使用，要配置好TIM，包括了两个模块：一、送给APB1分频器。APB1分频器可选择1、2、4、8、16分频，其输出一路供APB1外设PCLK1使用，另一路送给定时器(Timer)2、3、4倍频器使用。该倍频器可选择1或者2倍频，时钟输出供定时器2~7使用；二、送给APB2分频器。APB2分频器可选择1、2、4、8、16分频，其输出一路供APB2外设PCLK2使用，另一路送给定时器(Timer)1倍频器使用。该倍频器可选择1或者2倍频，时钟输出供定时器1/8使用。

8.27 GPIO 的复用功能重映射有何意义？如何实现的，举一个例子说明。

GPIO 的复用功能重映射可把某外设复用功能从(某默认)引脚转移至(某备用)引脚上，可分时复用外设，虚拟增加端口数量，优化引脚配置和布线设计PCB，同时减少信号交叉干扰。

从 I/O 引脚角度看，引脚 PB10主功能是 PB10，默认复用功能是 I2C2 的时钟端 SCL 和 USART3 的发送端 Tx，重定义功能是 TIM2_CH3。上电复位后，PB10 默认为普通输出，而 I2C2 的 SCL 和 USART3 的 Tx 是它的默认复用功能。定时器 2(TIM2)进行 I/O 引脚重映射后， TIM2_CH3 也可成为 PB10 的复用功能。若想使用 PB10 的默认复用功能 USART3，则需编程配置 PB10 为复用推挽输出模式，同时使能 USART3 并保持 I2C2 禁止状态。若要使用PB10的重定义复用功能 TIM2_CH3，则需编程对 TIM2 进行重映射，然后再按复用功能方式配置。

8.31 简述通用定时器的输入捕获过程。

输入时， 通过检测TIMx_CHx通道上的边沿信号，在边沿信号发生跳变（比如上升沿/下降沿）的时候，将当前定时器的值（TIMx_CNT）存放到对应的捕获/比较寄存器（TIMx_CCRx）里面，完成一次捕获。同时，还可以配置捕获时是否触发中断/DMA等。

8.32 参照书中例子，采用 TIM2 通道 2 进行频率测量，利用库函数实现其设置。

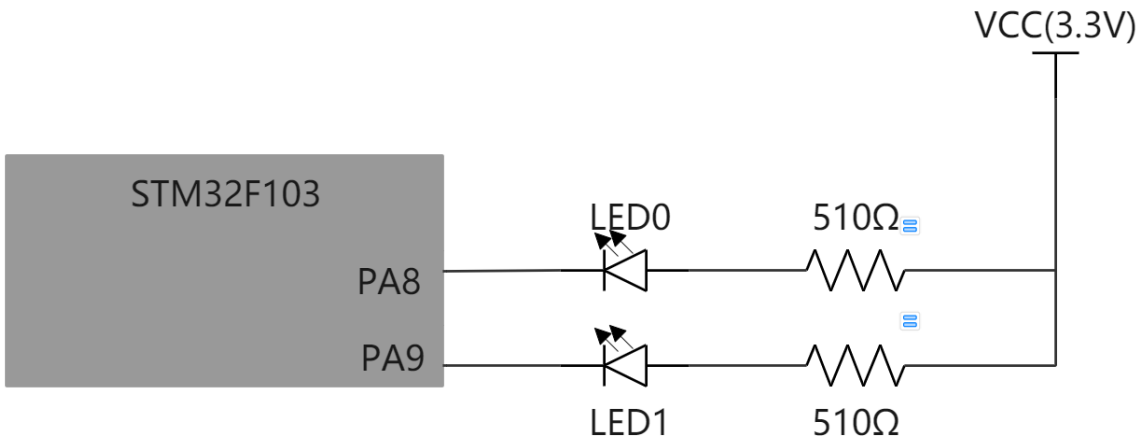
```
//TIM2 的 1/3 通道设置：
TIM_ICInitStructure.TIM_ICMode=TIM_ICMode_ICAP; //配置为输入捕获模式
TIM_ICInitStructure.TIM_Channel=TIM_Channel_2; //选择通道 2
TIM_ICInitStructure.TIM_ICPolarity=TIM_ICPolarity_Rising; //输入上升沿捕获
TIM_ICInitStructure.TIM_ICSelection=TIM_ICSelection_DirectTI; //通道方向选择
TIM_ICInitStructure.TIM_ICPrescaler=TIM_ICPSC_DIV1; //每次检测到输入捕获就触发一次捕获
TIM_ICInitStructure.TIM_ICFilter=0x0;
TIM_ICInit(TIM2,&TIM_ICInitStructure);

//输入捕获配置：
TIM_SelectInputTrigger(TIM2,TIM_TS_TI1FP1);//选择滤波后 TI1 作为输入触发源，触发下面程序的复位
TIM_SelectSlaveMode(TIM2,TIM_SlaveMode_Reset);
//复位模式-选中的触发输入（TRGI）的上升沿初始化计数器，并且产生一个更新信号
TIM_SelectMasterSlaveMode(TIM2,TIM_MasterSlaveMode_Enable);//主从模式选择
```

8.34 简述通用定时器的比较输出过程。

当CNT计数值(变动至)与CCR值相等时，把相应输出引脚可根据所设置编程模式选择以下赋值 (并输出)：置位、复位、翻转或不变，并将状态寄存器SR中相应标志位置位。同时，如果相应中断屏蔽位置位且中断使能，则产生中断。最后，如果DMA请求使能置位，则产生DMA请求。

8.36 参照书中例子，设计一个红绿灯系统，要求：红灯亮 5 秒钟，熄灭，切换绿灯亮 5 秒钟， 熄灭，循环往复。给出硬件原理图，并编写代码实现。

硬件设计：




```
# include <stm32f10x.h>
# include "delay.h"

int main(void)
{
    GPIO_InitTypeDef  GPIO_InitStrxuct;
    delay_init();

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_2MHz;

    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_8;
    GPIO_Init(GPIOA,&GPIO_InitStruct);
    GPIO_ResetBits(GPIOA, GPIO_Pin_8);

    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_9;
    GPIO_Init(GPIOA,&GPIO_InitStruct);
    GPIO_ResetBits(GPIOA, GPIO_Pin_9);

    while(1)
    {
        GPIO_SetBits(GPIOA, GPIO_Pin_8);
        GPIO_ResetBits(GPIOA, GPIO_Pin_9);
        delay_ms(5000);

        GPIO_SetBits(GPIOA, GPIO_Pin_9);
        GPIO_ResetBits(GPIOA, GPIO_Pin_8);
        delay_ms(5000);

    }
}
```

此处也可以使用TIM实现精确延时，利用如下子函数即可：

```
void TIM2_Delay5000MS ( )
{
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure ;
RCC_APB1PeriphClockCmd (RCC_APB1Periph_TIM2, ENABLE) ; // Enable TIM2 clock
/* TIM2 Time Base Configuration:
TIM2CLK /((TIM_Prescaler+1) * (TIM_Period+ 1))=TIM2 Frequency
TIM2CLK= 72MHz , TIM2 Frequency= 2Hz,
TIM_Prescaler= 36000-1, TIM_Period= 1000-1 */
TIM_TimeBaseStructure.TIM_Prescaler= 36000- 1 ;
TIM_TimeBaseStructure.TIM_Period= 10000 - 1 ;
TIM_TimeBaseStructure.TIM_CounterMode= TIM_CounterMode_Up ;
TIM_TimeBaseInit (TIM2, &TIM_TimeBaseStructure) ;
TIM_ClearFlag ( TIM2 , TIM_FLAG_Update ) ; // Clear TIM2 update pending flag
TIM_Cmd (TIM2, ENABLE) ; // Enable TIM2 counter
while (TIM_GetFlagStatus (TIM2, TIM_FLAG_Update) ==RESET) ;
}
```

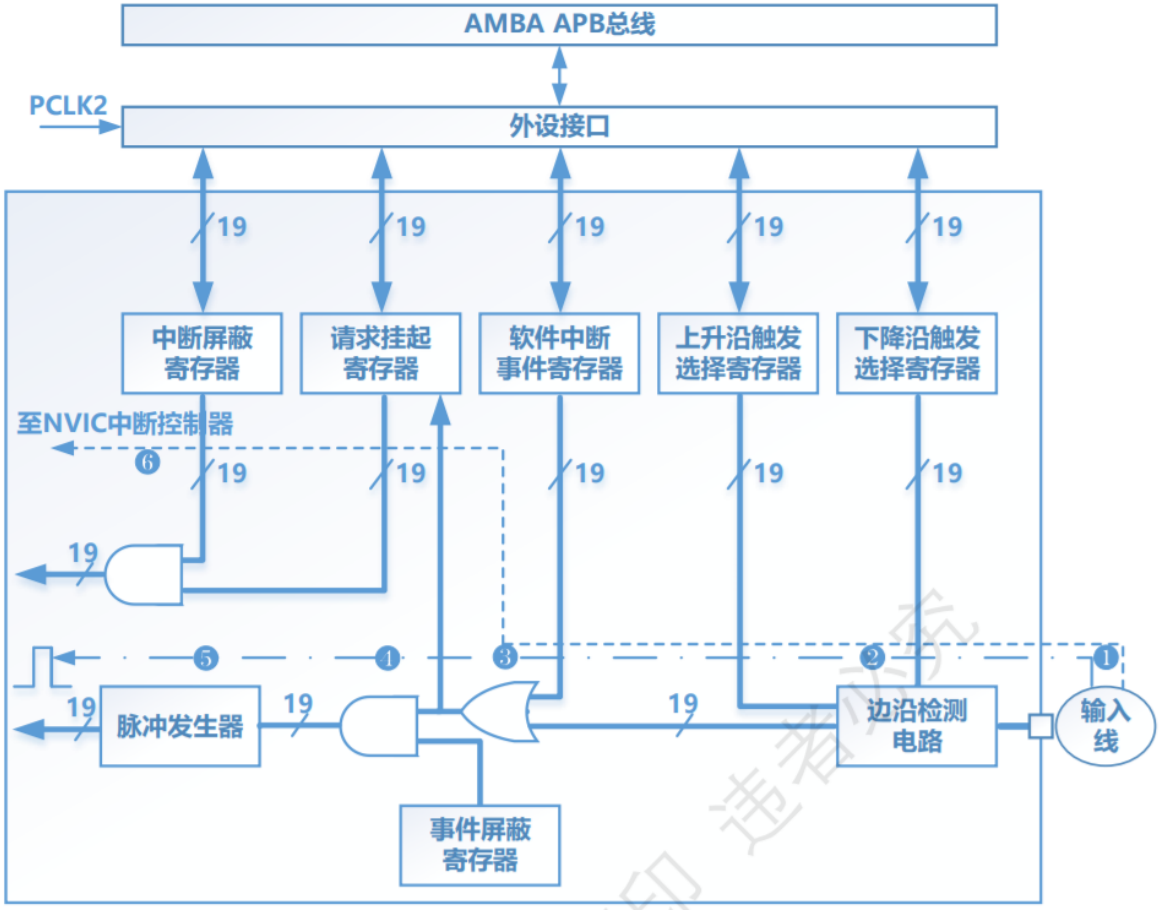
8.39 在中断优先级中，抢占优先级和子优先级如何发挥各自作用？实际应用中是怎么分组并设置的，以 STM32F103 举例说明。

抢占优先级，又称主/组/占先优先级，标识一个中断抢占式优先响应能力高低，决定是否会有中断嵌套发生。如，一个具有高抢占先优先级的中断会打断当前正在 执行的中断服务程序（转而执行其所对应 ISR）；子优先级，又称从优先级，仅在抢占优先级相同时才有影响，标识一个中断非抢占式优先响应能力高低。在抢占优先级相同时：如果有中断正被处理，高子优先级中断须等待正被响应的低子优先级中断处理结束后才能得到响应；如果没有中断正被处理，高子优先级中断将优先被响应。中断响应顺序遵循如下原则：先比较抢占优先级，抢占优先级高的中断优先响应； 当抢占优先级相同时，比较子优先级，子优先级高的中断优先响应；抢占优先级和子优先级都相同时，比较中断向量表中位置，位置低的中断优先响应。

实际应用：

1. 建立中断向量表
2. 分配栈空间并初始化
3. 设置中断优先级：①设置中断优先级分组(的位数)。4位中，抢占优先级和子优先级各占几位(二者和为4)， 可有5种方式：抢占优先级0~4位/子优先级4~0位 NVIC_PriorityGroup_0~4。②设置抢占优先级和子优先级，根据①中分组情况，分别设置抢占和子优先级(即在各自取值范围中设置优先级确定值)
4. 通过失效中断总屏蔽位和分屏蔽位，可使能对应中断
5. 编写对应ISR代码

8.46 EXTI 的外部中断/事件请求的产生和传输过程为何？以 STM32F103 为例，具体说明。



对照图中各单元编号①…⑥，从输入（外部输入线）到输出（外部中断/事件请求信号），外部中断/事件请求的产生和传输过程，依次如下：

1. 外部信号从①引脚进入；
2. 经过②边沿检测电路；此电路受上升沿触发选择寄存器和下降沿触发选择寄存器（这两个平行寄存器）控制，可配置选择上升沿、下降沿或双边沿（即同时选择上升沿和下降沿）产生中断/事件；
3. 经过③或门；此或门另一输入是软件中断/事件寄存器，软件可优先于外部信号产生一个中断或事件请求，即当软件中断/事件寄存器对应位为 1 时，不管外部信号如何，或门都会输出有效信号；至此，无论中断或事件，外部请求信号传输路径一致；
4. 外部请求信号进入④与门；此与门另一输入是事件屏蔽寄存器，其对应位为 0，则屏蔽某外部事件，该外部请求信号不能传输到与门另一端；其对应位为 1，则与门产生有效输出并送至⑤脉冲发生器；脉冲发生器把一个跳变信号转变为一个单脉冲，输出到其他功能模块；至此为外部事件请求信号传输路径，如图 8.56 中点划线箭头所示，即①→②→③→④→⑤；
5. 外部请求信号进入请求挂起寄存器（记录外部信号电平变化）；然后进入⑥与门；此与门功能和④与门类似，引入中断屏蔽寄存器控制；仅当其对应位为 1 时，该外部请求信号才被送至中断控制器 NVIC，从而发出一个中断请求，否则，屏蔽之；至此为外部中断请求信号传输路径，如图 8.56 中虚线箭头所示，即①→②→③→⑥。

8.50 简述 STM32F103 的 USART 数据接收/发送过程。

数据收发核心是两个移位寄存器：发送移位寄存器和接收移位寄存器， 负责收发数据并做并串转换。

1. **USART 数据发送过程**

   内核指令或 DMA 外设先将数据从内存（变量）写入发送数据寄存器 TDR。然后，发送控制器适时地自动把数据从 TDR 加载到发送移位寄存器，将数据一位一位地通过 Tx 引脚发送出去。当数据完成从 TDR 到发送移位寄存器的转移后，会产生 TDR 已空事件 TXE。其后，当数据从发送移位寄存器全部发送到 Tx 后，会产生数据发送完成事件 TC。可在 SR 中查询这些事件。 数据发送，须设置相关寄存器各位，过程如下：

   ①CR1.UE 置位 1，激活 USART；

②CR1.M 定义字长；

③CR2.STOP 定义停止位位数；

④若采用多缓冲器通信，配置 CR3.DMAT 使能 DMA，（另外配置 DMA）；

⑤利用 BRR 选择波特率；

⑥置位 CR1.TE，发送一个空闲帧作为第一次数据发送；

⑦将要发送数据写入 DR（此动作会清除 SR.TXE）；一个缓冲区情况下，重复⑦。

2. **USART 数据接收过程**

   是数据发送的逆过程。数据从 Rx 引脚一位一位地输入到接收移位寄存器中。然后，接收控制器自动将接收移位寄存器的数据转移到接收数据寄存器 RDR 中。最后，内核指令或 DMA 将 RDR 数据读入内存（变量）中。当接收移位寄存器的数据转移到 RDR 后，会产生 RDR 非空/已满事件 RXNE。 数据接收配置如下（前 5 步与发送相同）：

   ①CR1.UE 置位 1，激活 USART；

   ②CR1.M 定义字长；

   ③CR2.STOP 定义停止位位数；

   ④若采用多缓冲器通信，配置 CR3.DMAT 使能 DMA，（另外配置 DMA）；

   ⑤利用 BRR 选择波特率；

   ⑥置位 CR1.RE，激活接收器，开始寻找起始位；

   ⑦接收到一个字符时，SR.RXNE 被置位，表明移位寄存器内容被转移到 RDR，此时若 CR1.RXNEIE=1（即中断使能），则产生中断；接收期间若检测到帧/溢出/噪声错误， 相应标志会置位（供查询）。

   ⑧SR.RXNE 清零：多缓冲器，由 DMA 读 SR 完成；单缓冲模式，软件读 SR 完成；也可 通过对其写 0 完成。清零须在下一字符接收结束前完成，避免溢出错误。

8.52 编程实现通过串口 1 发送字符串“you are welcome”。

重定向 printf 到 USART1，通过 USART1 向 PC 串口发送“you are welcome”。USART1 与 PC 串口间通信速率和通信协议规定如下：数据传输速率为 115200bps，数据格式为 8 位数据位，无奇偶校验位，2 位停止位，无数据流控制。

```
# include "stm32f10x.h"
int fputc (int ch, FILE * f)
{
while (USART_GetFlagStatus (USART1, USART_FLAG_TC) = = RESET) ;
USART_SendData (USART1, (uint8_t) ch) ;
return ch;
}
void USART1_Config (unsigned int baud) ;
int main (void)
{
USART1_Config (115200) ; //USART1 Init with 115200bps
printf ("you are welcome") ;
while (1){ }
}
void USART1_Config (unsigned int baud)
{
GPIO_InitTypeDef GPIO_InitStructure;
USART_InitTypeDef USART_InitStructure;
RCC_APB2PeriphClockCmd (RCC_APB2Periph_GPIOA, ENABLE ) ; // Enable GPIO clock
// Configure USART1 Tx (PA9)
GPIO_InitStructure.GPIO_Pin= GPIO_Pin_9;
```

```
GPIO_InitStructure.GPIO_Mode= GPIO_Mode_AF_PP;
GPIO_InitStructure.GPIO_Speed= GPIO_Speed_50MHz ;
GPIO_Init (GPIOA, &GPIO_InitStructure) ;
// Configure USART1 Rx ( PA10)
GPIO_InitStructure.GPIO_Pin= GPIO_Pin_10;
GPIO_InitStructure.GPIO_Mode= GPIO_Mode_IN_FLOATING;
GPIO_InitStructure.GPIO_Speed= GPIO_Speed_50MHz ;
GPIO_Init (GPIOA, &GPIO_InitStructure) ;
RCC_APB2PeriphClockCmd ( RCC_APB2Periph_USART1, ENABLE); //Enable USART1 clock
/* Configure USART:
BaudRate=baud Word Length= 8Bits Two Stop Bit No parity
Hardware flow control disabled(RTS and CTS signals) Receive and transmit enabled
*/
USART_InitStructure.USART_BaudRate= baud;
USART_InitStructure.USART_WordLength= USART_WordLength_8b;
USART_InitStructure.USART_StopBit s= USART_StopBits_2 ;
USART_InitStructure.USART_Parity= USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl=USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode=USART_Mode_Rx|USART_Mode_Tx;
USART_Init (USART1, &USART_InitStructure);
USART_ClearFlag (USART1, USART_FLAG_TC); //Clear USART1 Transmission complete
flag
USART_Cmd (USART1, ENABLE) ; //Enable USART1
}
```

8.57 SPI 的环形总线结构有何特点，简述之。

主要特点包括如下几个方面：

1. SPI1 位于高速 API2 总线上，其他 SPI（如 SPI2、SPI3 等）位于 APB1 总线上；
2. 既可作为主设备，也可作为从设备；
3. 主模式和从模式下均可由软件或硬件进行 NSS 管理，动态改变主/从操作模式；
4. 可编程时序，由时钟极性和时钟相位决定；
5. 可编程数据格式，8/16 位数据帧，LSB 或 MSB 在前的数据顺序；
6. 可编程传输速率，最高可达 18MHz；
7. 可触发中断的两个标志位：发送标志位TXE（发送缓冲区空）和接收标志位RXNE （接收缓冲区非空）；
8. 支持 DMA 功能的 1B 发送和接收缓冲区，分别产生发送和接收请求；
9. 带或不带第三根双向数据线的双线单工同步传输；
10. 支持以多主配置方式工作。