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。

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

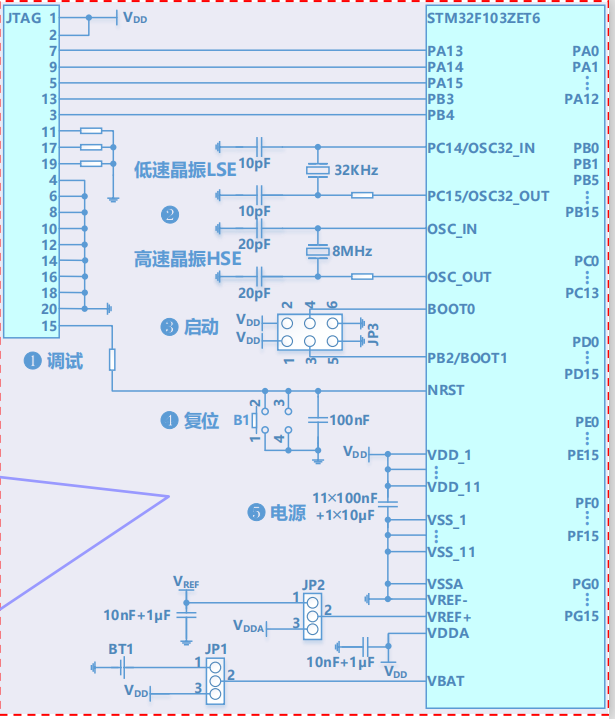
8.7 何为引脚功能复用？有何意义？如何实现，请以 STM32F103 为例，举例具体说明。

引脚功能复用：将某几个功能分配到一个引脚。

如果将芯片全部功能都安排专用引脚，庞大引脚数量会提升产品成本，装焊困难，加大应用成本， 而且实际应用中极少会用到器件全部资源。

以 STM32F103RCT6 为例，2号引脚上有PC13/TAMPER/RTCP三种功能。

8.11 请设计一个 STM32 最小系统。



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？

功能：延时， 使**CPU**保持复位，暂不进入工作状态(防止执行错误指令)，确保其及各部件处于确定的初始状态，直至电压稳定复位电路直接影响系统稳定性和可靠性。

系统复位，电源复位，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？

方便进行分频，时钟输入源与最终达到外设处的时钟速率不再有固定的关系。

2） 从左至右，相关时钟依次可分为大致哪几种？

输入时钟、 系统时钟和由系统时钟分频所得其他时钟。

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。

可通过个别使能/禁止外部功能和降低外部时钟频率优化功耗。

使用**HSE**及**ST**库函数，配置时钟(参数)流程如下：  
➀ 将**RCC**重新设置为默认值， **RCC\_DeInit**➁ 打开**HSE**， **RCC\_HSEConfig**(**RCC\_HSE\_ON**)  
➂ 等待**HSE**工作， **HSEStartUpStatus=RCC\_WaitForHSEStartUP**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  
微机原理与嵌入式系统 51  
➃ 设置**AHB**时钟， **RCC\_HCLKConfig**；  
➄ 设置高速/低速**AHB**时钟，  
**RCC\_PCLK2Config/RCC\_PCLK1Config**➅ 设置**PLL**， **RCC\_PLLConfig**；  
打开**PLL**， **RCC\_PLLCmd**(**ENABLE**)  
➆ 等待**PLL**工作，  
**while**(**RCC\_GetFlagStatus**(**RCC\_FLAG\_PLLRDY**)==**RESET**))  
➇ 设置系统时钟， **RCC\_SYSCLKConfig**➈ 判断**PLL**是否是系统时钟，  
**while**(**RCC\_GetSYSCLKSource**()**!=0x08**)  
➉ 打开要用外设时钟， **RCC\_APB2PeriphClockCmd**/  
**RCC\_APB1PeriphClockCmd**

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。

复用功能重映射，可分时复用某些外设，虚拟增加了端口数量， 可用于优化引脚配置和 PCB 布线设计， 并减少信号交叉干扰。

如，要将 USART2 的 Tx 和 Rx 从默认的引脚 PA2 和 PA3 重新映射到引脚 PD5 和 PD6，根据图述步骤，可使用库函数编程如下：RCC\_ APB2PeriphClockCmd (RCC\_APB2Periph\_GPIOD,ENABLE); //使能引脚 PD5 和 PD6 的时钟RCC\_APB2PeriphClockCmd (RCC\_APB2Periph\_AFIO,ENABLE); //使能引脚 PD5 和 PD6 上 AFIO 时钟//根据 USART2 的 Tx 和 Rx，分别将引脚 PD5 和 PD6 设置为推挽复用和浮空输入模式  
GPIO\_InitStructure.GPIO\_Pin=GPIO\_Pin\_5;  
GPIO\_InitStructure.GPIO\_Mode=GPIO\_Mode\_AF\_PP;  
GPIO\_Init (GPIOD,&GPIO\_InitStructure);  
GPIO\_InitStructure.GPIO\_Pin=GPIO\_Pin\_6;  
GPIO\_InitStructure.GPIO\_Mode=GPIO\_Mode\_IN\_FLOATING;  
GPIO\_Init (GPIOD,&GPIO\_InitStructure);  
//使能要进行 I/O 引脚重映射的外设 USART2 的时钟  
RCC\_APB1PeriphClockCmd (RCC\_APB1Periph\_USART2,ENABLE);

GPIO\_PinRemapConfig (GPIO\_Remap\_USART2,ENABLE); //进行 USART2 的 I/O 引脚重映射

以上语句中所用库函数可参考本章相关小节及技术手册。

8.31 简述通用定时器的输入捕获过程。

输入时，当捕获的输入脉冲在电平发生翻转时， **CCR**加载**CNT**当前计数值，实现脉冲频率测量。  
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。

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

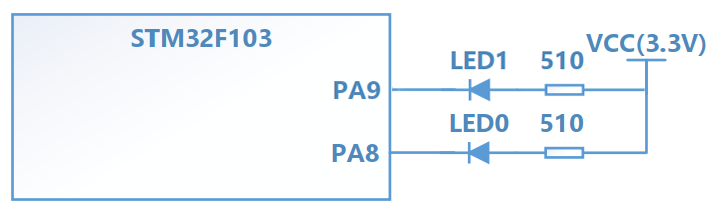
TIM\_SelectSlaveMode(TIM2,TIM\_SlaveMode\_Reset);

//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。

输出时， **CCR**用来存储一个数值，将此数值与**CNT**当前计数值比较，根据比较结果进行不同的电平输出。

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。



# include "stm32f103.h"

void LED0\_Config(void) ;

void LED0\_On(void) ;

void LED0\_Off(void) ;

void LED1\_Config(void) ;

void LED1\_On(void) ;

void LED1\_Off(void) ;

void delay\_us(u32 nTimer)；

void delay\_ms(u32 nTimer)；

int main(void)

{

LED0\_Config() ;

LED1\_Config() ;

while (1)

{

LED0\_On() ;

delay\_ms(u32 5000)；

LED0\_Off() ;

LED1\_On() ;

delay\_ms(u32 5000)；

LED1\_Off() ;

}

}

void LED0\_Config(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

//Enable GPIO\_LED0 clock

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA, ENABLE);

//GPIO\_LED0 Pin (PA8) Configuration

GPIO\_InitStructure.GPIO\_Pin=GPIO\_Pin\_8;

GPIO\_InitStructure.GPIO\_Mode=GPIO\_Mode\_Out\_PP;

GPIO\_InitStructure.GPIO\_Speed=GPIO\_Speed\_2MHz;

GPIO\_Init(GPIOA, &GPIO\_InitStructure) ;

}

void LED0\_On(void)

{

GPIO\_ResetBits(GPIOA,GPIO\_Pin\_8) ;

}

void LED0\_Off(void)

{

GPIO\_SetBits(GPIOA,GPIO\_Pin\_8) ;

}

void LED1\_Config(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

//Enable GPIO\_LED1 clock

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA, ENABLE);

//GPIO\_LED0 Pin (PA9) Configuration

GPIO\_InitStructure.GPIO\_Pin=GPIO\_Pin\_9;

GPIO\_InitStructure.GPIO\_Mode=GPIO\_Mode\_Out\_PP;

GPIO\_InitStructure.GPIO\_Speed=GPIO\_Speed\_2MHz;

GPIO\_Init(GPIOA, &GPIO\_InitStructure) ;

}

void LED1\_On(void)

{

GPIO\_ResetBits(GPIOA,GPIO\_Pin\_9) ;

}

void LED1\_Off(void)

{

GPIO\_SetBits(GPIOA,GPIO\_Pin\_9) ;

}

void delay\_us(u32 nTimer)

{

u32 i=0;

for(i=0;i<nTimer;i++){

\_\_NOP();\_\_NOP();

}

}

void delay\_ms(u32 nTimer)

{

u32 i=1000\*nTimer;

delay\_us(i);

}

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。

每个中断源有 4 位优先级， 具有 16 级可编程中断优先级。可根据实际应用需求编程设定4 位优先级中抢占优先级的位数。

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 8.56 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：  
❒ 外部信号从①引脚进入；  
❒ 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；  
❒ 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；  
❒ 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；  
❒ 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。

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）：  
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”。

# 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。

优点：  
⚫ 支持全双工， 推挽 push-pull 驱动性能比开漏 open-drain 完整性更好；  
⚫ 支持高速（100MHz 以上） 传输；  
⚫ 协议支持字长不限于 8bits，可灵活选择；硬件连接简单。

缺点：  
⚫ 比 I2C 多两根线；  
⚫ 无寻址机制，靠片选选择不同设备；  
⚫ 从设备接收无 ACK，主设备不知发送是否成功；  
⚫ 典型应用只支持单主控；  
⚫ 相比 RS232/RS485/CAN 总线，传输距离短。