1. **什么是嵌入式系统?嵌入式系统一般由哪几部分构成?**  
   答:以应用为中心，以计算机技术为基础，软件、硬件可裁减，适应应用系统对功能、可靠性、成本、体积、功耗严格要求的专用计算机系统。嵌入式系统采用“量体裁衣”的方式把所需的功能嵌入到各种应用系统中。  
   嵌入式系统一般由嵌入式微处理器、外围硬件设备、嵌入式操作系统（可选)，以及用户的应用软件系统等四个部分组成。

可以将嵌入式系统抽象成硬件层、中间层、操作系统层和应用软件层四个层次架构。

**2.**一般而言，嵌入式系统的构架可以分为4个部分:分别是嵌入式微处理器、外围硬件设备、嵌入式操作系统和用户应用软件。

3.嵌入式系统的硬件组成一般包括嵌入式处理器、存储器、I/O系统和外设

ARM体系结构使用两种方式存储字数据,被别是大端格式(big endian)和小端格式(little  
endian)。

1. 概念：**CISC** 复杂指令计算机 **RISC**精简指令计算机 ARM属于RISC类其指令长度是定长的

CISC复杂指令集 RISC精简指令集

指令系统 处理特殊任务效率高，规整性差 处理特殊任务效率低，指令规律齐全

存储操作 操作指令多操作直接 操作有限 控制简单

程序 程序设计相对容易 程序设计复杂依靠编译

中断 指令执行完响应 指令执行过程中响应中断

CPU 功能强 面积大 功耗高 面积小功耗低

设计周期 结构复杂设计周期长 布局紧凑 设计周期短

用户使用 功能强大 专业使用 指令规整 易学易用

应用范围 适合通用机 适用专用机

**5.ARM Cortex-M3有何特点?**答:采用哈佛结构的32位处理器内核，具有高性能、实时功能、数字信号处理、低功耗与低电压操作特性，同时还保持了集成度高和易于开发的特点。

**6.STM32中断优先级，分为抢占优先级和子优先级。**

抢占优先级决定了是否会有中断嵌套发生。

子优先级，又称为从优先级，仅在抢占优先级相同时才有影响，它标识了一个中断非抢  
占优先响应能力的高低。

ARM Cortex-M3内核定义了8位优先级。STM32微控制器只使用了其中的4位，具有16级可编程异常优先级。

**7.简述STM32的优先级划分与抢占的过程。**

答:STM32微控制器的每个中断源，有4位优先级，具有16级可编程异常优先级。用户可以根据实际应用需求通过编程设定4位优先级中抢占优先级的位数和子优先级的位数。级别关系是:抢占式优先级>响应(子）优先级>中断表中的排位顺序。只有在上一级优先级相同时下一级优先级才会起作用。

STM32对于中断的响应顺序，遵循以下原则:先比较抢占优先级，抢占优先级高的中断优先响应;

高抢占优先级的中断可以打断低抢占优先级程序的执行，即中断嵌套;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;

当上述两者都相同时，比较它们在中断向量表中的位置，位置低的中断优先响应。

**8.高级控制定时器主要由时基单元、输入捕获和输出比较等部分组成。**

**9.简述SysTick 的功能特点。**

答:SysTick的核心是1个24位递减计数器，使用时根据需要设置初值，启动后在系统时钟的作用下递减，减到0时置计数标志位并重装初值。系统可以查询计数标志位，也可以在中断允许时产生SysTick 中断。

**10.DMA** (Direct Memory Access，直接存储器存取）将外设的数据不经过CPU直接送入

内存储器，或从内存储器不经过CPU直接送往外部设备。

STM32的 DMA可实现存储器和存储器间、外设和存储器、存储器和外设之间的传输，

闪存、SRAM、外设的SRAM、APB1、APB2和AHB上的外设均可作为访问的源和目标。

STM32的 DMA1控制器有7个通道，每个通道专门用来管理来自于一个或多

个外设对存储器访问的请求。还有一个仲裁器﹒来协调各个DMA请求的优先权。

STM32的 DMA具有12个独立的可配置的通道/请求:DMAl有7个通道，DMA2有5个通道。

在STM32中，当在存储器之间使用DMA进行数据传输时，可使用任意DMA的任意

通道。在同一个DMA上，多个通道请求的优先级可以通过软件编程设置，当软件优先级设置相等时由硬件决定优先级高低。在 DMA处理时，一个事件发生后，外设发送一个请求信号到DMA控制器。DMA控制器根据通道的优先权处理请求。

**11.**STM32F103只有大容量的产品才有**DAC**。

STM32的 DAC模块是12位数字输入、电压输出型的DAC。 DAC可以配置为8位或12位模式，也可以与DMA控制器配合使用。

DAC工作在 12位模式时，数据可以设置成左对齐或右对齐。l32.DAC模块有②个输出通道，每个通道都有单独的转换器

**12.**STM32微控制器的外部中断/事件控制器**EXTI**，由19(或者20)根外部输入线、19(或

者20）个产生中断/事件请求的边沿检测器和APB外设接口等部分组成。对于互联型(CL)STM32微控制器产品，外部中断/事件控制器由20个产生事件/中断请求的边沿检测器组成。对于互联型产品以外的STM32微控制器,其外部中断/事件控制器(EXTI)由19个产生事件/中断要求的边沿检测器组成。每个输入线可以独立地配置成外部中断(interrupt)或事件(event)模式，并且可以配置其所对应的触发事件。

从优先级编号7开始的中断向量为外部中断，其优先级是可自行配置的。

外部中断/事件控制器(EXTI)用于管理外部中断。

EXTI每个中断/事件都有独立的触发和\_屏蔽，触发请求可以是上升沿、下降沿或者双边沿触发。挂起(Pending）寄存器保持着状态线的中断要求。

STM32的EXTI线16连接到PVD输出。Programmable voltage detector88.

STM32的EXTI 线17连接到RTC闹钟事件。

STM32的 EXTI 线18连接到USB唤醒事件。

1. **中断函数编写规则**

（1）不能有返回值（2）不能向ISR传递函数（3）ISR应尽可能的短小精悍（4）printf(char×IPFormatString,....)函数会带来重入和性能问题，不能在ISR中采用

38.利用TIM2实现0.5s精确延时，从而以1s为周期点亮LED。要求使用优先级分组1、占先优先级为1、子优先级为3。

1. 请写出TIM2配置的关键程序。void TIM\_Configuration(void){

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM2,ENABLE);

TIM\_TimeBaseInitTypeDef TIM\_BaseInitStructure;

TIM\_BaseInitStructure.TIM\_Prescaler=36000-1;

TIM\_BaseInitStructure.TIM\_Period=1000-1;

TIM\_BaseInitStructure.TIM\_ClockDivision=0;

TIM\_BaseInitStructure.TIM\_CounterMode=TIM\_CounterMode\_Up;

TIM\_BaseInitStructure.TIM\_RepetitionCounter=TIM\_CKD\_DIV1;

TIM\_TimeBaseInit(TIM2,&TIM\_BaseInitStructure);

TIM\_UpdateRequestConfig(TIM2,TIM\_UpdateSource\_Regular);

TIM\_Cmd(TIM2,ENABLE);

TIM\_ITConfig(TIM2,TIM\_IT\_Update,ENABLE);

}

1. 写出NVIC配置关键程序。

void NVIC\_Configuration(void)

{ NVIC\_InitTypeDef NVIC\_InitStructure;

NVIC\_PriorityGroupConfig(NVIC\_PriorityGroup\_1);

NVIC\_InitStructure.NVIC\_IRQChannel=TIM2\_IRQn;

NVIC\_InitStructure.NVIC\_IRQChannelPreemptionPriority=1;

NVIC\_InitStructure.NVIC\_IRQChannelSubPriority=3;

NVIC\_InitStructure.NVIC\_IRQChannelCmd=ENABLE;

NVIC\_Init(&NVIC\_InitStructure);}

（3）写出中断服务子程序。

void TIM2\_IRQHandler(void){

if(TIM\_GetITStatus(TIM2,TIM\_IT\_Update)==SET)

{ GPIOB->ODR^=LED;

TIM\_ClearITPendingBit(TIM2,TIM\_FLAG\_Update);}

TIM的PWM输出实验：

TIM时基配置结构体定义 typedef struct{ uint16\_t TIM\_Prescaler;

uint16\_t TIM\_CounterMode;(设置计数器模式)Uint16\_t TIM\_Period; (设置计数器周期，即ARR)

uint16\_t TIM\_ClockDivision;(设置TIM的内部时钟CLK\_INT与输入捕获滤波器之间的时钟分割系数)

uint8\_t TIM\_RepetitionCounter;(设置重复计数器数值)}TIM\_TimeBaseInitTypeDef;

输出比较OC初始化定义 typedef struct{ uint16\_t TIM\_OCMode;(设置定时器模式)

uint16\_t TIM\_OutputState; uint16\_t TIM\_OutputNState;uint16\_t TIM\_Pulse;(装载到CCRx中的数值) uint16\_t TIM\_OCPolarity;(指明OC输出高电平有效还是低电平有效) uint16\_t TIM\_OCNPolarity;

uint16\_t TIM\_OCIdleState; uint16\_t TIM\_OCNIdleState;}TIM\_OCInitTypeDef;

int main(void){

RCC\_Configuration(); GPIO\_Configuration();

PrescalerValue=(uint16\_t)(SystemCoreClock/24000000)-1;

TIM\_TimeBaseStructure.TIM\_Period=666-1;

TIM\_TimeBaseStructure.TIM\_Prescaler=PrescalerValue;

TIM\_TimeBaseStructure.TIM\_ClockDivision=0;

TIM\_TimeBaseStructure.TIM\_CounterMode=TIM\_CounterMode\_Up;

TIM\_TimeBaseInit(TIM3,&TIM\_TimeBaseStructure);

TIM\_OCInitStructure.TIM\_OCMode=TIM\_OCMode\_PWM1;

TIM\_OCInitStructure.TIM\_OutputState=TIM\_OutputState\_Enable;

TIM\_OCInitStructure.TIM\_Pulse=CCR1\_Val;

TIM\_OCInitStructure.TIM\_OCPolarity=TIM\_OCPolarity\_High;

TIM\_OC1Init(TIM3,&TIM\_OCInitStructure);

TIM\_OC1PreloadConfig(TIM3,TIM\_OCPreload\_Enable);

TIM\_OCInitStructure.TIM\_OutputState=TIM\_OutputState\_Enable;

TIM\_ARRPreloadConfig(TIM3,ENABLE);

TIM\_Cmd(TIM3,ENABLE);

while(1){}

}

void RCC\_Configuration(void){

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM3,ENABLE);//使能GPIOA的时钟

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA|RCC\_APB2Periph\_GPIOB|RCC\_APB2Periph\_GPIOC|RCC\_APB2Periph\_AFIO,ENABLE);//使能uSART1的时钟，USART1挂接到APB2上。}

void GPIO\_Configuration(void)

{ GPIO\_InitTypeDef GPIO\_InitStructure;//声明GPIO初始化结构变量。

GPIO\_InitStructure.GPIO\_Pin=GPIO\_Pin\_6|GPIO\_Pin\_7;//配置管脚3

GPIO\_InitStructure.GPIO\_Speed=GPIO\_Speed\_50MHz;

GPIO\_InitStructure.GPIO\_Mode=GPIO\_Mode\_AF\_PP; //IO口配置为推挽输出口

GPIO\_Init(GPIOA,&GPIO\_InitStructure);//初始化PA8口

GPIO\_InitStructure.GPIO\_Pin=GPIO\_Pin\_0|GPIO\_Pin\_1;

GPIO\_Init(GPIOB,&GPIO\_InitStructure); }

39.在某工程中，需要检测一个低频压力传感器输出的信号。假设压力传感器输出的电压范围为0-10mV。请基于STM32F103设计一个电压采样方案，通过ADC1采集电压，并通过串口发送到PC机。假设利用ADCI的通道1和USART2。ADC采样周期为 55.5个时钟周期。串行口采用USART2，使用1位起始位、8位数据位、无校验位和1停止位，波特率位9600bps。

要求：

（1）画出并简述系统组成框图

（2）画出程序流程图

（3）写出配置ADC的关键语句

void ADC\_Configuration(void)

{

ADC\_InitTypeDef ADC\_InitStructure;

GPIO\_InitTypeDef GPIO\_InitStructure;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_ADC1 | RCC\_APB2Periph\_GPIOA, ENABLE);//开启ADC1的时钟

ADC\_TempSensorVrefintCmd(ENABLE);

RCC\_ADCCLKConfig(RCC\_PCLK2\_Div6);//12MHZ

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_0;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AIN;

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

ADC\_InitStructure.ADC\_Mode = ADC\_Mode\_Independent;//选择ADC工作在独立模式或者双ADC模式

ADC\_InitStructure.ADC\_ScanConvMode = DISABLE; //权责工作在扫描模式（多通道）还是单次（单通道）模式

ADC\_InitStructure.ADC\_ContinuousConvMode = DISABLE;//选择ADC工作在连续还是单次模式。

ADC\_InitStructure.ADC\_ExternalTrigConv = ADC\_ExternalTrigConv\_None;//定义使用外部触发来启动规则通道的ADC

ADC\_InitStructure.ADC\_DataAlign = ADC\_DataAlign\_Right;//设置数据左对齐还是右对齐

ADC\_InitStructure.ADC\_NbrOfChannel = 1;//设置进行规则转换的ADC通道数目

ADC\_Init(ADC1, &ADC\_InitStructure);

ADC\_Cmd(ADC1, ENABLE);//使能ADC1

ADC\_ResetCalibration(ADC1); // 复位ADC1校准寄存器

while(ADC\_GetResetCalibrationStatus(ADC1)); //等待校准复位结束

ADC\_StartCalibration(ADC1);//开始校准

while(ADC\_GetCalibrationStatus(ADC1)); //等待校准结束

}

（4）写出配置USART的关键语句

void USART\_Configuration(void){

USART\_InitTypeDef USART\_InitStructure;

USART\_InitStructure.USART\_BaudRate=115200;

USART\_InitStructure.USART\_WordLength = USART\_WordLength\_8b;

USART\_InitStructure.USART\_StopBits=USART\_StopBits\_1;

USART\_InitStructure.USART\_Parity=USART\_Parity\_No;

USART\_InitStructure.USART\_HardwareFlowControl=USART\_HardwareFlowControl\_None;

USART\_InitStructure.USART\_Mode=USART\_Mode\_Tx;

USART\_Init(USART1,&USART\_InitStructure);

USART\_Cmd(USART1,ENABLE);

}

1.  
对于72M频率来说，计算pwm频率是：  
频率：  
//Fpwm = 72M / ((arr+1)\*(psc+1))(单位：Hz)  
占空比：  
//duty circle = TIM3->CCR1 / arr(单位：%)

//PWM输出初始化  
//arr：自动重装值  
//psc：时钟预分频数

2.

产生5ms定时，即200Hz。则总分频数目=72000000/200=360000

令PSC=36、CNT=10000,则可以产生200Hz定时信号

4.点灯

#include "stm32fl0x.hM

void RCC\_Configuration(void);

void delay\_nms(ul6 time);

int main(void)

{

GPIO」nitTypeDef GPIOInitStructure;

RCC\_Configuration();

GPIO」nitStructuw.GPIO\_Mode=GPIO\_Mode\_Out\_PP;

GPIO」nitStructuw.GPIO\_Speed=GPIO\_Speed\_50MHz;

GPIO」nitStructuw.GPIO\_Piii=GPIO\_Piii\_6|GPIO\_Piii\_7|GPIO\_Piii\_8;

GPIO」nit(GPIOB, gGPIO—InitStructure);

GPIO\_SetBits(GPIOB,GPIO\_InitStructure.GPIO\_Pin); while(l)

{

GPIO\_ResetBits(GPIOB,GPIO\_Pin\_6);

delay\_nms(500);

GPIO\_SetBits(GPIOB,GPIO\_Pin\_6);

GPIO\_ResetBits(GPIOB,GPIO\_Pin\_7);

delay\_nms(500);

GPIO\_SetBits(GPIOB,GPIO\_Pin\_7);

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

delay\_nms(500);

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

}

1. 如下程序为L1602液晶显示器“写”操作的实现函数。请根据该程序，画出时序

图。

void write(uchar del)//1602写数据函数，输入的是需要写入1602的数据

{//根据命令集，按照时序来使能LCD

DATA(del);

RS(l);//RS=0?选择写指令寄存器；RS=1,选择写数据寄存器。

RW(0);//RW=0,进行写操作；RW=1,进行读操作

E(0); //E,使能位

delay\_ms(10);

玖1);//使能

delay\_ms(10);

E(0);

}

1. 嵌入式系统是以 应用中心.以计算机技术为基础，软件硬件 可裁剪，适应应用系 统对功能、可靠性、成本、体积、功耗严格要求的 专用 计算机系统。
2. 一般而言，嵌入式系统的构架可以分为4个部分：分别是嵌入式微处理器、外围硬 件设备、嵌入式操作系统和用户应用软件。
3. 可以将嵌入式系统抽象成硬件层、中间层、操作系统层和应用软件层四个层次架构。
4. 微处理器有两种总线架构，数据和指令使用同一接口的是冯诺伊曼结构,分开的指 令和数据接口、取指和数据访问可以并行进行的是哈弗结构。

根据指令集的不同，微处理器的架构可分为**（复杂指令集CISC）和（精简指令集RISC）** 两种。

1. 根据嵌入式系统使用的微处理器，可以将嵌入式系统分为嵌入式微控制器，嵌入式DSP

处理器， 嵌入式微处理器 以及片上系统SOC。

1. 微处理器是整个系统的核心，通常由3大部分组成：控制单元、算术逻辑 单元

和 寄存器 。

1. 计算机有CISC和RISC两种类型，以ARM微处理器为核心的计算机属于 **RISC***类* 型，其指令长度是定长的。
2. 操作系统是联接硬件与应用程序的系统程序，其基本功能有进程管理、进程间通信、 內存管理、I/O资源管理。
3. 从嵌入式操作系统特点可以将嵌入式操作系统分为（实时操作系统）和分时操作系统，其 中实时系统亦可分为（硬实时系统）和软实时系统。
4. ARM体系结构使用两种方式存储字数据,被别是大端格式(big endian)和小端格式(little endian)。
5. 在大端格式中,鬲位数字存放在低位字节中，在小端格式中,鬲位数字存放在鬲位字节 中。ARM系统默认采用小端格式存储。
6. ARM将存储器看做是一个从0开始的线性递增的字节集合。
7. 基于ARM內核的芯片具有许多的外设，对这些外设访问的标准方法是使用存储器映射 I/O,为外设的每个寄存器都分配一个地址。
8. 通常，向ARM的存储器映射I/O地址装载数据用于逵△，向这些地址保存数据用于筠 生。有些地址的装载和保存用于外设的控制功能，而不是输入或输出功能。
9. Cortex-M3是基于ARMv7-M体系结构的微控制器核。
10. Cortex-M3有多个寄存器，其中大部分用于通用寄存器，有小部分作为专用寄存器， **R15** 寄存器用于存储PC, R13通常用来存储 堆栈指针**SP** 。
11. STM32的 嵌套向量中断控制器(NVIC) 管理着包括Cortex-M3核异常等中断，其 和ARM处理器核的接口紧密相连，可以实现 低延迟 的中断处理.并有效地处理 晚 到中断。
12. STM32中断优先级，分为抢占优先级和子优先级。
13. 抢占优先级决定了是否会有中断嵌套发生。
14. 子优先级,又称为从优先级，仅在抢占优先级相同时才有影响，它标识了一个中断非抢 占优先响应能力的高低。
15. ARM Cortex-M3內核定义了 &位优先级。STM32微控制器只使用了其中的生位，具有 16级可编程异常优先级。
16. Cortex-M3在嵌套中断时.高优先级中断可以抢占低优先级中断服务子程序，这与 传统ARM处理器中的快中断(FIQ)的功能类似。
17. STM32对Cortex-M3的中断向量表重新进行了剪裁，并为异常和中断编排了默认优先 级编号。其中，优先级编号-3〜6的中断向量定义为系统异常,编号为负数的內核异常优 先级是固定的，无法通过软件进行配置，包括：复位(Reset, -3)、不可屏蔽中断(NMI,

-2)、硬故障(HardFault, \_])。

1. 外部中断/事件控制器(EXTI)用于管理外部中断o
2. EXTI每个中断/事件都有独立的触发和屏蔽,触发请求可以是上升沿、下降沿或者 双边沿触发。挂起(Pending)寄存器保持着状态线的中断要求。
3. STM32 的 EXTI 线 16 连接到 PVD 输出 。Programmable voltage detector
4. STM32的EXTI线17连接到 RTC闹钟事件 。
5. STM32的EXTI线18连接到 USB唤醒事件 。
6. STM32的通用定时器TIM,均由一个通过可编程预分频器驱动的16 位自动装载计 数器构成。
7. 系统滴答定时器（SysTick）提供了 1个24位二进制递减计数器.具有灵活的控制机 制。

SysTick的核心是1个24位递减计数器,使用时根据需要设置初值，启动后在系统时钟 的作用下递减，减到0时置计数标志位并重装初值。

1. DMA （Direct Memory Access,直接存储器存取）将外设的数据不经过CPU直接送入 内存储器，或从內存储器不经过CPU直接送往外部设备。
2. STM32的DMA可实现存储器和存储器间、外设和存储器、存储器和外设之间的传输. 闪存、SRAM,外设的SRAM, APBK APB2和AHB上的外设均可作为访问的源和目 标。
3. STM32的DMA1控制器有 7 个通道.每个通道专门用来管理来自于一个或多

个外设对存储器访问的请求。还有一个 仲裁器 来协调各个DMA请求的优先权。

1. STM32的DMA具有旦个独立的可配置的通道/请求：DMA1有Z个通道，DMA2 < 5 个通道。
2. 在STM32中，当在存储器之间使用DMA进行数据传输时，可使用任意DMA的任意 通道。
3. 在同一个DMA上，多个通道请求的优先级可以通过软件编程设置.当软件优先级设置 相等时由硬件决定优先级高低。
4. 在DMA处理时，一个事件发生后，外设发送一个请求信号到DMA控制器。DMA控 制器根据通道的 优先权 处理请求。

11& STM32芯片內部集成的12位ADC是一种逐次逼近型模拟数字转换器，具有\_ 个 通道，可测量16个外部和2个內部信号源。

1. STM32的ADC由时钟控制器提供的ADCCLK时钟和PCLK2 （APB2时钟）同步。
2. STM32ADC各通道的A/D转换可以单次（single）、连续（continuous）、扫描（scan）或 间断（discontinuous）模式执行，有规则通道组和注入通道组,每次转换结束可 产生中断（DMA）o
3. STM32 ADC转换原理为逐次逼近型A/D转换，分为注入通道和规则通道。
4. 在STM32中,只有在 规则通道 的转换结束时才产生DMA请求，并将转换的数据 从 ADC->DR 寄存器传输到用户指定的目的地址。
5. 在有两个ADC的STM32器件中，可以使用 双ADC模式。
6. 在双ADC模式里，根据 ADC CR1 寄存器中 DUALMOD「2:0］位所选的模式. 转换的启动可以是ADC1主和ADC2从的交替触发或同时触发。
7. ADC的校准模式通过设置 ADC CR2 寄存器的 CAL 位来启动。
8. 在STM32中.ADC CR2 寄存器的 ALIGN 位选择转换后数据储存的对齐方式。
9. STM32F103只有大容量的产品才有DAC。
10. STM32的DAC模块是12位数字输入、电压输出型的DAC。DAC可以配置为8位 或12位模式.也可以与DMA控制器配合使用。
11. DACX作在12位模式时，数据可以设置成左对齐或右对齐。
12. DAC模块有*2*个输出通道.每个通道都有单独的转换器
13. STM32F103系列最少都拥有2\_个ADC。
14. ADC的输入时钟ADCCLK不得超过14MHz,它是由PCLK2经分频产生。