[一、选择题](#6011-1608728581999)

[二、填空题](#6416-1568280279235)

[三、简答题](#3019-1568280296142)

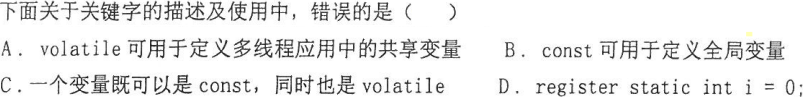
[PNP](#7IgP-1668003096962)

[NPN](#uq6B-1668003096146)

[四、编程题](#P5Yt-1668003289979)

[五、翻译题](#8662-1568281405327)

**一、选择题**



答案：D



答案：C

**嵌入式芯片的内核电源与片内I/O模块的电源是**

A、统一接在一起的

B、不同的，其中内核电源的电压要高

C、不同的，其中I/O模块电源的电压要高

D、上述都可以

答案：C

**UCOSIII操作系统不属于（     ）。**

A、RTOS

B、抢占式实时操作系统

C、非抢占式实时操作系统

D、嵌入式实时操作系统

答案：C

**在嵌入式芯片接口中，以下\_\_\_\_\_\_\_\_\_\_\_\_\_是音频接口。**

A.IIC B.IIS C.PWM D.以上都不是

答案：B

**在启动代码中，在关键的初始化代码中，可以使用\_\_\_\_\_\_\_\_\_\_\_\_\_\_语言。**

A.汇编 B.C语言 C.汇编+C语言 D.以上都不是

答案：A

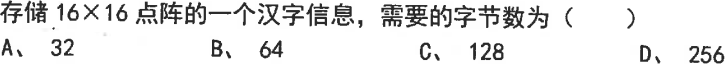
**在启动代码中，STM32复位处理放在\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。**

A.0x00000000 B.0x00000004 C.0x00000010 D.0x00000014

答案：B

clipboard.png

答案：C



答案：A

**二、填空题**

clipboard.png

答：

#ifndef x          //先测试x是否被宏定义过

#define x

    程序段1     //如果x没有被宏定义过，定义x，并编译程序段 1

#endif

    程序段2　 //如果x已经定义过了则编译程序段2的语句，“忽视”程序段 1

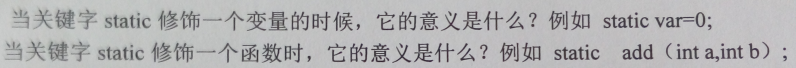
条件指示符#ifndef 的最主要目的是防止头文件的重复包含和编译。

**三、简答题**

**1.0x01<<2+3的值为多少?为什么?**

答：值为32。因为“+”号比“<<”号优先级高。

**2.**



答：

在C语言中static的作用如下

第一、在修饰变量的时候，static修饰的静态局部变量只执行一次，而且延长了局部变量的生命周期，直到程序运行结束以后才释放。

第二、static修饰全局变量的时候，这个全局变量只能在本文件中访问，不能在其它文件中访问，即便是extern外部声明也不可以。

第三、static修饰一个函数，则这个函数的只能在本文件中调用，不能被其他文件调用。Static修饰的局部变量存放在全局数据区的静态变量区。初始化的时候自动初始化为0；

（1）不想被释放的时候，可以使用static修饰。比如修饰函数中存放在栈空间的数组。如果不想让这个数组在函数调用结束释放可以使用static修饰

（2）考虑到数据安全性（当程想要使用全局变量的时候应该先考虑使用static）

**3.关键字volatile有何含义？请举3个例子。**

答案：volatile作为指令关键字，确保本条指令不会因编译器的优化而省略，且要求每次直接读值，即不是从寄存器里取备份值，而是去该地址内存存储的值.

举例：

1)并行设备的硬件寄存器（如：状态寄存器）

2)一个中断服务子程序中会访问到的非自动变量(Non-automatic variables)

3)多线程应用中被几个任务共享的变量

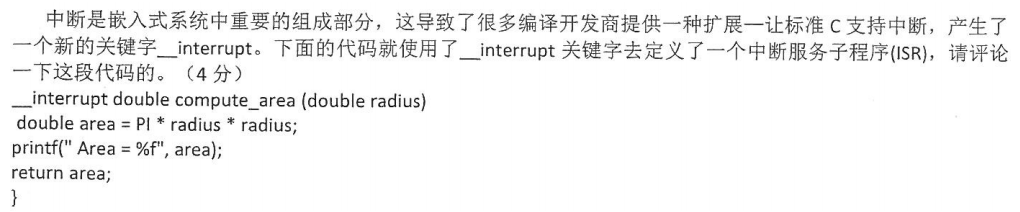
**4.一个参数既可以是const还可以是volatile吗？解释为什么。**

答案：是的。一个例子是只读的状态寄存器。它是volatile因为它可能被意想不到地改变。它是const因为程序不应该试图去修改它。

**5.一个指针可以是volatile 吗？解释为什么。**

答案：是的。尽管这并不很常见。一个例子是当一个中服务子程序修该一个指向一个buffer的指针时。

**6.**



少了大括号；

不能有返回值；

不能有参数传递；

不应该执行浮点运算且复杂的代码；

**7.**

clipboard.png

答：RCC库位于stm32f4xx\_rcc.c文件中。RTC库位于stm32f4xx\_rtc.c文件中。

**8.全局变量和局部变量在内存中是否有区别？**

答：全局变量保存在内存的全局存储区中，占用静态的存储单元；局部变量保存在栈中，只有在所在函数被调用时才动态地为变量分配存储单元。

**9.**

clipboard.png

答：

|  |
| --- |
|  |

**10.空指针与野指针的区别？**

答案：

空指针的值是0或NULL，表示没有指向任何的内存地址。

野指针的值是无效的值，表示指向无效的内存地址，指向了分配的有效内存空间之外。

**11.详细描述SPI接口与IIC（I2C）接口通信的异同点。**

答：

（1）工作模式与线的数目

IIC总线不是全双工，高电平采样，2根线SCL SDA。

SPI总线实现全双工，边沿采样，4根线SCK CS MOSI MISO，通信简单易理解。

（2）速度

SPI的速度要远远高于IIC

SPI总线速度为：几MHz - -几十MHz

IIC总线速度为：0Hz - - 3.4MHz

（3）流控方面

SPI缺乏流控机制，无论主器件还是从器件均不对消息进行确认，主器件无法知道从器件是否繁忙。

IIC存在ACK应答信号,所以有流控机制。

（4）设备的选择

IIC总线是多主机总线，通过SDA上的地址信息来锁定从设备。

SPI总线只有一个主设备，主设备通过CS片选来确定从设备。

（5）驱动编写方面

IIC总线读写时序比较固定统一，设备驱动编写方便。

SPI总线不同从设备读写时序差别比较大，因此必须根据具体的设备datasheet来实现读写，相对复杂一些。

相同点

IIC总线和SPI总线时钟都是由主设备产生，并且只在数据传输时发出时钟。

其他

IIC上拉电阻的作用：让空闲的设备保持在高电平状态，增强抗干扰能力。

**12.简述DMA操作的认识和理解。**

答案：

1）实现DMA传送的基本操作如下：

a.外设可通过DMA控制器向CPU发出DMA请求；

b.CPU响应DMA请求，系统转变为DMA工作方式，并把总线控制权交给DMA控制器；

c.由DMA控制器发送存储器地址，并决定传送数据块的长度；

d.执行DMA传送；

e.DMA操作结束，并把总线控制权交还CPU。

2）DMA方式主要适用于一些高速的I/O设备。这些设备传输字节或字的速度非常快。

**13.简述STM32异常的处理过程。**

答：

《Cortex M3与M4权威指南》章节4.5 P104

Exceptions are events that cause changes to program flow. When one happens, the processor suspends the current executing task and executes a part of the program called the exception handler. After the execution of the exception handler is completed, the processor then resumes normal program execution. **In the ARM architecture**, **interrupts are one type of exception**. Interrupts are usually generated from peripheral or external inputs, and in some cases they can be triggered by software. The exception handlers for interrupts are also referred to as Interrupt Service Routines (ISR)。

翻译：

异常是导致程序流更改的事件。当发生这种情况时，处理器将挂起当前正在执行的任务，并执行程序中称为异常处理程序的一部分。异常处理程序的执行完成后，处理器将恢复正常的程序执行。在ARM体系结构中，中断是一种例外。中断通常由外设或外部输入产生，在某些情况下，它们可以由软件触发。中断的异常处理程序也称为中断服务例程（ISR）。

**14.简述STM32中断抢占优先级与响应优先级。**

答：

抢占优先级和响应优先级，其实是一个中断所包含的两个优先级，其中前者是对抢占优先级的级别划分，后者是相同抢占优先级的优先级别的划分。

比如：中断A抢占优先级比B高，那么A的中断可以在B里面触发，忽略响应优先级。

A和B抢占优先级相同，则A、B的响应优先级决定谁先响应。

15.硬实时任务和软实时任务的概念理解。

答案：所谓“硬实时”是区别“软实时”而言的。

硬实时对满足时限的要求比软实时严格，通常硬实时的系统响应时间在ms或μs级，而软实时对其响应时间的要求没有那么严格。

硬实时工作通常指超过时限要求就会造成严重损害的工作，而软实时即使超过时限也不会带来严重后果。

以核能电厂和看VCD为例，用在核能电厂的实时操作系统，如果超出时限可能会导致严重的损害，然而VCD播放器超出时限只不过让使用者感觉不舒服而已。所以前者是硬实时，后者是软实时。

**16.在程序中，使用宏定义有哪些优缺点。**

优点：

1. 提高了程序的可读性，同时也方便进行修改；
2. 提高程序的运行效率：使用带参的宏定义既可完成函数调用的功能，又能避免函数的出栈与入栈操作，减少系统开销，提高运行效率；
3. 宏是由预处理器处理的，通过字符串操作可以完成很多编译器无法实现的功能。比如##连接符。

缺点：

1. 由于是直接嵌入的，所以代码可能相对多一点；
2. 嵌套定义过多可能会影响程序的可读性，而且很容易出错；
3. 对带参的宏而言，由于是直接替换，并不会检查参数是否合法，存在安全隐患。

**17.看门狗的作用**

答：看门狗是用来防止万一单片机程序出错造成重大损失的。防错的原理很简单，它在硬件上就是一个定时器，当它溢出的时候就会让单片机强制复位使程序重新开始执行。

**18.**



答：

（1）定义

SPI(Serial Peripheral Interface：串行外设接口);
I2C(INTER IC BUS，内部集成电路总线)
UART(Universal Asynchronous Receiver Transmitter：通用异步收发器)

（2）区别

1. SPI总线由三条信号线组成：串行时钟(SCLK)、串行数据输出(MOSI)、串行数据输入(MISO)。SPI总线可以实现多个SPI设备互相连接。提供SPI串行时钟的SPI设备为SPI主机或主设备(Master)，其他设备为SPI从机或从设备(Slave)。主从设备间可以实现全双工通信，当有多个从设备时，还可以增加一条从设备选择线。

1. I2C总线是双向、两线(SCL、SDA)、串行、多主控（multi-master）接口标准，具有总线仲裁机制，非常适合在器件之间进行近距离、非经常性的数据通信。在它的协议体系中，传输数据时都会带上目的设备的设备地址，因此可以实现设备组网。

1. UART总线是异步串口，因此一般比前两种同步串口的结构要复杂很多，一般由波特率产生器(产生的波特率等于传输波特率的16倍)、UART接收器、UART发送器组成，硬件上由两根线，一根用于发送，一根用于接收。

**19.推挽输出概念与作用。**

推挽输出结构是由两个MOS或者三极管收到互补控制的信号控制，两个管子始终一个在导通，一个在截止。

作用：

推挽输出的最大特点是可以真正能真正的输出高电平和低电平，在两种电平下都具有驱动能力。

**20.开漏输出的概念与作用。**

推挽输出相对的就是开漏输出，对于开漏输出和推挽输出的区别最普遍的说法就是开漏输出无法真正输出高电平，即高电平时没有驱动能力，需要借助外部上拉电阻完成对外驱动。

作用：

1. 开漏输出的这一特性一个明显的优势就是可以很方便的调节输出的电平，因为输出电平完全由上拉电阻连接的电源电平决定。所以在需要进行电平转换的地方，非常适合使用开漏输出。
2. 开漏输出的这一特性另一个好处在于可以实现"线与"功能，所谓的"线与"指的是多个信号线直接连接在一起，只有当所有信号全部为高电平时，合在一起的总线为高电平；只要有任意一个或者多个信号为低电平，则总线为低电平。而推挽输出就不行，如果高电平和低电平连在一起，会出现电流倒灌，损坏器件。

**21.三极管与CMOS管的作用与区别。**

1. 关于晶体三极管

三极管，全称应为半导体三极管，也称双极型晶体管，晶体三极管，是一种电流控制电流的半导体器件，其作用是把微弱信号放大成辐值较大的电信号， 也用作无触点开关。晶体三极管，是半导体基本元器件之一，具有电流放大作用，是电子电路的核心元件。三极管是在一块半导体基片上制作两个相距很近的PN结，两个PN结把正块半导体分成三部分，中间部分是基区，两侧部分是发射区和集电区，排列方式有PNP和NPN两种。

1. 关于CMOS管

CMOS（Complementary Metal Oxide Semiconductor）即互补金属氧化物半导体，是一种大规模应用于集成电路芯片制造的原料，由PMOS管和NMOS管共同构成，它的特点是低功耗。

采用CMOS技术可以将成对的金属氧化物半导体场效应晶体管（MOSFET）集成在一块硅片上。该技术通常用于生产RAM和交换应用系统，在计算机领域里通常指保存计算机基本启动信息的RAM芯片。现在，CMOS制造工艺也被广泛应用于制作数码影像器材的感光元件等方面。

两者有一个本质的差别，三极管是电流控制电流源。

CMOS是电压控制电流源，所以三极管是电流为控制，CMOS是电压。

**22.无源晶振与有源晶振的区别。**

若考虑产品成本，建议可以选择无源晶振电路；若考虑产品性能，建议选择有源晶振电路，省时方便也能保证产品性能。

**23.实时系统中出现互斥锁死锁的原因，以及如何解除死锁。**

答：

死锁产生的4个必要条件：

（1） 互斥：某种资源一次只允许一个进程访问，即该资源一旦分配给某个进程，其他进程就不能再访问，直到该进程访问结束。

（2） 占有且等待：一个进程本身占有资源（一种或多种），同时还有资源未得到满足，正在等待其他进程释放该资源。

（3） 不可抢占：别人已经占有了某项资源，你不能因为自己也需要该资源，就去把别人的资源抢过来。

（4） 循环等待：存在一个进程链，使得每个进程都占有下一个进程所需的至少一种资源。

当以上四个条件均满足，必然会造成死锁，发生死锁的进程无法进行下去，它们所持有的资源也无法释放。这样会导致CPU的吞吐量下降。所以死锁情况是会浪费系统资源和影响计算机的使用性能的。那么，解决死锁问题就是相当有必要的了。

产生死锁需要四个条件，那么只要这四个条件中至少有一个条件得不到满足，就不可能发生死锁了。由于互斥条件是非共享资源所必须的，不仅不能改变，还应加以保证，所以破坏产生死锁条件如下。

破坏产生死锁的其他三个条件：

（1）破坏占有且等待条件

方法1：所有的进程在开始运行之前，必须一次性地申请其在整个运行过程中所需要的全部资源。

优点：简单易实施且安全。

缺点：因为某项资源不满足，进程无法启动，而其他已经满足了的资源也不会得到利用，严重降低了资源的利用率，造成资源浪费。使进程经常发生饥饿现象。

方法2：该方法是对第一种方法的改进，允许进程只获得运行初期需要的资源，便开始运行，在运行过程中逐步释放掉分配到的已经使用完毕的资源，然后再去请求新的资源。这样的话，资源的利用率会得到提高，也会减少进程的饥饿问题。

（2）破坏不可抢占条件

当一个已经持有了一些资源的进程在提出新的资源请求没有得到满足时，它必须释放已经保持的所有资源，待以后需要使用的时候再重新申请。这就意味着进程已占有的资源会被短暂地释放或者说是被抢占了。

该种方法实现起来比较复杂，且代价也比较大。释放已经保持的资源很有可能会导致进程之前的工作实效等，反复的申请和释放资源会导致进程的执行被无限的推迟，这不仅会延长进程的周转周期，还会影响系统的吞吐量。

（3）破坏循环等待条件

可以通过定义资源类型的线性顺序来预防，可将每个资源编号，当一个进程占有编号为i的资源时，那么它下一次申请资源只能申请编号大于i的资源。

**24.何谓临界区？**

答：

每个进程中访问临界资源的那段代码称为临界区（critical section），每次只允许一个进程进入临界区，进入后，不允许其他进程进入。不论是硬件临界资源还是软件临界资源，多个进程必须互斥的对它进行访问。多个进程涉及到同一个临界资源的的临界区称为相关临界区。使用临界区时，一般不允许其运行时间过长，只要运行在临界区的线程还没有离开，其他所有进入此临界区的线程都会被挂起而进入等待状态，并在一定程度上影响程序的运行性能。

**25.实时操作系统的任务调度点。**

答：

1. 释放信号量、释放互斥锁或者发送消息，也可通过配置相应的参数不发生任务调度。
2. 使用延时函数OSTimeDly()或者OSTimeDlyHMSM()。
3. 任务等待的事情还没发生(等待信号量，消息队列等)。
4. 任务取消等待。
5. 创建任务。
6. 删除任务。
7. 删除一个内核对象。
8. 任务改变自身的优先级或者其他任务的优先级。
9. 任务通过调用OSTaskSuspend()将自身挂起。
10. 任务解挂某个挂起的任务。
11. 退出所有的嵌套中断。
12. 通过OSSchedUnlock()给调度器解锁。
13. 任务调用OSSchedRoundRobinYield()放弃其执行时间片。
14. 用户调用OSSched()

**26.SPI中的CPOL与CPHA的含义。**

答：

CPOL(Clock Polarity)：时钟极性选择，为0时SPI总线空闲时，时钟线为低电平 ；为1时SPI总线空闲时，时钟线为高电平。

CPHA(Clock Phase)：时钟相位选择，为0时在SCLK第一个跳变沿，主机对MISO引脚电平采样；为1时在SCLK第二个跳变沿，主机对MISO引脚电平采样。

**27.请描述I2C总线中的起始条件、停止条件中SDA与SCL引脚电平的变化。**

答：

总线在空闲状态时，SCL和SDA都保持着高电平，当SCL为高电平而SDA由高到低的跳变，表示产生一个起始条件；

当SCL为高而SDA由低到高的跳变，表示产生一个停止条件。

**28.在项目代码，如何提高程序的抗干扰能力。**

答：

数字滤波方法

　　数字滤波是在对模拟信号多次采样的基础上，通过软件算法提取最逼近真值数据的过程。数字滤波的的算法灵活，可选择权限参数，其效果往往是硬件滤波电路无法达到的。

软件拦截技术

当窜入单片机系统的干扰作用在CPU 部位时，后果更加严重，将使系统失灵。最典型的故障是破坏程序计数器PC 的状态，导致程序从一个区域跳转到另一个区域，或者程序在地址空间内“乱飞”，或者陷入“死循环”。使用软件拦截技术可以拦截“乱飞”的程序或者使程序摆脱“死循环”，并将运行程序纳入正轨，转到指定的程序入口。

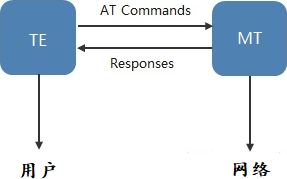
“软件看门狗”技术

PC 受到干扰而失控，引起程序“乱飞”，也可能使程序陷入“死循环”。当软件拦截技术不能使失控的程序摆脱“死循环”的困境时，通常采用程序监视技术WDT TIMER（WDT），又称“看门狗”技术，使程序脱离“死循环”。WDT 是一种软、硬件结合的抗程序跑飞措施，其硬件主体是一个用于产生定时T 的计数器或单稳，该计数器或单稳基本独立运行，其定时输出端接至CPU 的复位线，而其定时清零则由CPU 控制。在正常情况下，程序启动WDT 后，CPU 周期性的将WDT 清零，这样WDT 的定时溢出就不会发生，如同睡眠一般不起任何作用。在受到干扰的异常情况下，CPU 时序逻辑被破坏，程序执行混乱，不可能周期性的将WDT 清零，这样当WDT 的定时溢出时，其输出使系统复位，避免CPU因一时干扰而陷入瘫痪的状态。

**29.对于AT指令的了解程度。**

答：

在物联网中，AT命令集可用于控制&调测设备、通信模块入网等。AT命令，用来控制TE（Terminal Equipment）和MT(Mobile Terminal)之间交互的规则，在GSM网络中，用户可以通过AT命令进行呼叫、短信、电话本、数据业务、传真等方面的控制。



|  |  |  |
| --- | --- | --- |
| **类别** | **语法** | **说明** |
| 执行命令 | 有参数:AT+<x>=<...> 无参数:AT+<x> | 用来设置AT命令中的属性 |
| 测试命令 | AT+<x>=? | 用来显示AT命令设置的合法参数值有哪些（范围）。 |
| 查询命令 | AT+<x>? | 用来查询当前AT命令的设置的属性值。 |

**30.上拉电阻、下拉电阻的作用。**

**简介**

上拉就是将不确定的信号通过一个电阻嵌位在高电平，电阻同时起限流作用。 同理，下拉就是将不确定的信号通过一个电阻嵌位在低电平，电阻同时起限流作用。

**作用**

1. 避免电压的“悬浮”，造成电路的不稳定。
2. 提高驱动能力
3. 用单片机输出高电平，但由于后续电路的影响，输出的高电平不高，就是达不到VCC要求，影响电路工作。所以要接上拉电阻。
4. 下拉电阻情况，让单片机引脚输出低电平，结果由于后续电路影响输出的低电平达不到GND，所以接个下拉电阻。

**31.限流电阻的作用。**

限流电阻经常串联于电路中，用以限制所在支路电流的大小，以防电流过大烧坏所串联的元器件。同时限流电阻也能起分压作用。

**32.电容的定义与作用**

电容（或称电容量）是表现电容器容纳电荷本领的物理量。电容从物理学上讲，它是一种静态电荷存储介质，可能电荷会永久存在，这是它的特征，它的用途较广，它是电子、电力领域中不可缺少的电子元件。主要用于电源滤波、信号滤波、信号耦合、谐振、滤波、补偿、充放电、储能、隔直流等电路中。

更详细的描述：<https://zhuanlan.zhihu.com/p/51761646>

**33.电感的定义与作用**

电感器(Inductor)是能够把电能转化为磁能而存储起来的元件。电感器的结构类似于变压器，但只有一个绕组。电感器具有一定的电感，它只阻碍电流的变化。如果电感器在没有电流通过的状态下，电路接通时它将试图阻碍电流流过它；如果电感器在有电流通过的状态下，电路断开时它将试图维持电流不变。电感器又称扼流器、电抗器、动态电抗器。

阻交通直

对于直流电，电感是相当于短路的；而对于交流电，电感是对其有阻碍作用的，交流电的频率越高，电感对它的阻碍作用越大。

电感还能实现变压器、RL的低通滤波器、RL的高通滤波器。

**34.堆和栈**

栈区（stack），由编译器自动分配释放 ，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈。

堆区（heap）， 一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回收 。

**35.何为MSB和LSB？**

**36.什么是可重入函数？**

可重入（reentrant）函数可以由多于一个任务并发使用，而不必担心数据错误。可重入函数可以在任意时刻被中断，稍后再继续运行，不会丢失数据；可重入函数要么使用本地变量，要么在使用全局变量时保护自己的数据。

**37.什么是不可重入函数？**

不可重入函数在实时系统设计中被视为不安全函数。

满足下列条件的函数多数是不可重入的：

1. 函数体内使用了静态的数据结构；
2. 函数体内调用了malloc()或者free()函数；
3. 函数体内调用了标准I/O函数。

**38.按键抖动的原因与解决办法。**

原因：按键在按下时，活动触点击打固定触点会有机械振动，因而造成输出波形抖动，因按键形态和触点材料的不同，抖动的过程一般会持续数毫秒，金属触点的按键可能达到10ms，而软性触点（如导电橡胶或薄膜）则可能在1ms以内甚至没有抖动。

解决方法：按键去抖动的实用做法是定时查询，定时器资源往往是极易复用的，一般10ms左右的查询间隔对于用户按键也是足够的，用户操作按键不可能达到50次每秒，按下的持续时间也不可能短于10ms。

**39.extern "C"的作用**

加上extern "C"后，会指示编译器这部分代码按C语言（而不是C++）的方式进行编译。因为C、C++编译器对函数的编译处理是不完全相同的，尤其对于C++来说，支持函数的重载，编译后的函数一般是以函数名和形参类型来命名的。

例如函数void fun(int, int)，编译后的可能是\_fun\_int\_int(不同编译器可能不同，但都采用了类似的机制，用函数名和参数类型来命名编译后的函数名)；而C语言没有类似的重载机制，一般是利用函数名来指明编译后的函数名的，对应上面的函数可能会是\_fun这样的名字。

**40.static与extern**

static主要有三种用法：在函数内用于修饰变量、用于修饰函数、用于修饰本.c文件全局变量。

extern用于声明多个模块共享的全局变量、声明外部函数。

**41.assert的概念与作用**

断言（assert）就是用于在代码中捕捉这些假设，可以将断言看作是异常处理的一种高级形式。断言表示为一些布尔表达式，程序员相信在程序中的某个特定点该表达式值为真。

可以在任何时候启用和禁用断言验证，因此可以在测试时启用断言，而在部署时禁用断言。同样，程序投入运行后，最终用户在遇到问题时可以重新启用断言。注意assert()是一个宏，而不是函数。

**42.do {}while(0)**

do {} while(0)的使用时为了保证宏定义的使用者能无编译错误的用宏。

定义一个宏：

#define SAFE\_FREE(p) do{free(p); p=NULL} while(0)

假设这里去掉do{....} while(0)，及定义为：

#define SAFE\_FREE(p) free(p); p=NULL；

那么一下代码

if(NULL!=p)

SAFE\_FREE(p)

else

.......

会被展开成：

if(NULL!=p)

free(p); p=NULL；

else

.......

展开存在两个问题：

因为if分支后面有两个语句，导致else分支没有对应的if，编译失败。

假设没有else分支，则SAFE\_FREE中的第二个语句无论if测试是否通过，都会执行。

如何解决以上问题呢？

有人说给SAFE\_FREE的定义加上{}就可以解决上述问题了，即：

#define SAFE\_FREE(p) { free(p); p=NULL; }

代码展开如下：

if(NULL!=p)

{ free(p); p=NULL; }

else

.......

但是，在C程序中，每个语句后面加分号是一种约定俗成的习惯，那么代码如下：

If(NULL!=p)

{ free(p); p=NULL; }；

else

.......

问题又来了，这样else又没有对应的if了，编译还是失败。假设用了do{} while(0)就可以解决上面的一系列问题了，代码如下：

If(NULL!=p)

do { free(p); p=NULL; } while(0)；

else

.......

所以do {} while(0)的使用时为了保证宏定义的使用者能无编译错误的用宏。

**43.字节对齐**

***1. 什么是字节对齐？***

在C语言中，结构是一种复合数据类型，其构成元素既可以是基本数据类型（如int、long、float等）的变量，也可以是一些复合数据类型（如数组、结构、联合等）的数据单元。在结构中，编译器为结构的每个成员按其自然边界（alignment）分配空间。各个成员按照它们被声明的顺序在内存中顺序存储，第一个成员的地址和整个结构的地址相同。

为了使CPU能够对变量进行快速的访问,变量的起始地址应该具有某些特性,即所谓的”对齐”. 比如4字节的int型,其起始地址应该位于4字节的边界上,即起始地址能够被4整除.

***2. 字节对齐有什么作用？***

字节对齐的作用不仅是便于cpu快速访问，同时合理的利用字节对齐可以有效地节省存储空间。

对于32位机来说，4字节对齐能够使cpu访问速度提高，比如说一个long类型的变量，如果跨越了4字节边界存储，那么cpu要读取两次，这样效率就低了。但是在32位机中使用1字节或者2字节对齐，反而会使变量访问速度降低。所以这要考虑处理器类型，另外还得考虑编译器的类型。在vc中默认是4字节对齐的，GNU gcc 也是默认4字节对齐。

***3. 更改C编译器的缺省字节对齐方式***

在缺省情况下，C编译器为每一个变量或是数据单元按其自然对界条件分配空间。一般地，可以通过下面的方法来改变缺省的对界条件：

· 使用伪指令#pragma pack (n)，C编译器将按照n个字节对齐。

· 使用伪指令#pragma pack ()，取消自定义字节对齐方式。

另外，还有如下的一种方式：

· \_\_attribute((aligned (n)))，让所作用的结构成员对齐在n字节自然边界上。如果结构中有成员的长度大于n，则按照最大成员的长度来对齐。

· \_\_attribute\_\_ ((packed))，取消结构在编译过程中的优化对齐，按照实际占用字节数进行对齐。

***4. 举例说明***

1. 例1

struct test

{

char x1;

short x2;

float x3;

char x4;

};

由于编译器默认情况下会对这个struct作自然边界（有人说“自然对界”我觉得边界更顺口）对齐，结构的第一个成员x1，其偏移地址为0，占据了第1个字节。第二个成员x2为short类型，其起始地址必须2字节对界，因此，编译器在x2和x1之间填充了一个空字节。结构的第三个成员x3和第四个成员x4恰好落在其自然边界地址上，在它们前面不需要额外的填充字节。在test结构中，成员x3要求4字节对界，是该结构所有成员中要求的最大边界单元，因而test结构的自然对界条件为4字节，编译器在成员x4后面填充了3个空字节。整个结构所占据空间为12字节。

1. 例2

#pragma pack(1) //让编译器对这个结构作1字节对齐

struct test

{

char x1;

short x2;

float x3;

char x4;

};

#pragma pack() //取消1字节对齐，恢复为默认4字节对齐

这时候sizeof(struct test)的值为8。

1. 例3

#define GNUC\_PACKED \_\_attribute\_\_((packed))

struct PACKED test

{

char x1;

short x2;

float x3;

char x4;

}GNUC\_PACKED;

这时候sizeof(struct test)的值仍为8。

**44.stm32中字节对齐问题（\_\_align(n)，\_\_packed用法）**

***1.\_\_align(num)***

这个用于修改最高级别对象的字节边界。在汇编中使用LDRD或者STRD时，就要用到此命令\_\_align(8)进行修饰限制，来保证数据对象是相应对齐。 这个修饰对象的命令最大是8个字节限制,可以让2字节的对象进行4字节对齐,但是不能让4字节的对象2字节对齐。

\_\_align是存储类修改,他只修饰最高级类型对象，不能用于结构或者函数对象。

比如：\_\_align(4) u8 mem1base[MEM1\_MAX\_SIZE];//保证分配的数组空间4字节对齐，同时保证数组首地址可被4整除。

***2.\_\_packed***

\_\_packed是进行一字节对齐

(1) 不能对packed的对象进行对齐

(2) 所有对象的读写访问都进行非对齐访问

(3) float及包含float的结构联合及未用\_\_packed的对象将不能字节对齐

(4) \_\_packed对局部整形变量无影响

(5) 强制由unpacked对象向packed对象转化是未定义,整形指针可以合法定义为packed。\_\_packed int\* p; //\_\_packed int 则没有意义

(6) 对齐或非对齐读写访问带来问题

\_\_packed struct STRUCT\_TEST

{

char a;

int b;

char c;

} ; //定义如下结构此时b的起始地址一定是不对齐的

//在栈中访问b可能有问题,因为栈上数据肯定是对齐访问[from CL]

//将下面变量定义成全局静态不在栈上

static char\* p;

static struct STRUCT\_TEST a;

void main()

{

\_\_packed int\* q; //此时定义成\_\_packed来修饰当前q指向为非对齐的数据地址下面的访问则可以

p = (char\*)&a;

q = (int\*)(p+1);

\*q = 0x87654321;

/\*

得到赋值的汇编指令很清楚

ldr r5,0x20001590 ; = #0x12345678

[0xe1a00005] mov r0,r5

[0xeb0000b0] bl \_\_rt\_uwrite4 //在此处调用一个写4byte的操作函数

[0xe5c10000] strb r0,[r1,#0] //函数进行4次strb操作然后返回保证了数据正确的访问

[0xe1a02420] mov r2,r0,lsr #8

[0xe5c12001] strb r2,[r1,#1]

[0xe1a02820] mov r2,r0,lsr #16

[0xe5c12002] strb r2,[r1,#2]

[0xe1a02c20] mov r2,r0,lsr #24

[0xe5c12003] strb r2,[r1,#3]

[0xe1a0f00e] mov pc,r14

\*/

/\*

如果q没有加\_\_packed修饰则汇编出来指令是这样直接会导致奇地址处访问失败

[0xe59f2018] ldr r2,0x20001594 ; = #0x87654321

[0xe5812000] str r2,[r1,#0]

\*/

//这样可以很清楚的看到非对齐访问是如何产生错误的

//以及如何消除非对齐访问带来问题

//也可以看到非对齐访问和对齐访问的指令差异导致效率问题

}

比如：

typedef \_\_packed struct READ\_Command

{

u\_char code;

u\_int addr;

u\_char len;

} READ\_Command;

与

typedef struct READ\_Command

{

u\_char code;

u\_int addr;

u\_char len;

} READ\_Command;

的区别是什么啊？

回答:没有\_\_packed的会出现字对齐等也就是，char型的有可能是占用4个字节的长度的内存空间有\_\_packed 的就不会，就肯定是1个字节的内存空间，是gcc编译器的关键字。(不止vc下面32位的系统里面的内存数据的存取是32位的，处理的时候都是4个字节为单位，通常也就是int的长度。如果不定义压缩方式，也就是编译选项没有诸如#pragma pack(1)之类的，那么系统会进行4字节对齐)。

注意:\_packed只是某种编译器的格式压缩，有的是pack呢，对不同的CPU压缩的对齐方式也不一样,在使用了该关键以后在进行操作时需要格外小心。

声明结构类型时，可以包含一个保留字packed，用于实现压缩数据存储。

当一个记录类型在 {$A-} 状态下声明或者声明中包括了保留字 packed 时，记录中的字段不被调整，而替换为赋予连续的偏移量。这样一个压缩记录的总尺寸就是所有字段的尺寸的和。因为数据调整尺寸可能改变（如不同版本的编译器对同一种数据类型的调整值可能不同），所以当想要把记录写入磁盘时或者在内存中传递到另一模块而该模块由不同版本的编译器编译时，最好还是压缩所有的记录。(delphi borland 中也有该关键字)

***3.在 Cotex-M3 programming manual 中有提到对齐问题***

(1) 通常编译器在生成代码的时候都会进行结构体填充，保证（结构体内部成员）最高性能的对齐方式。

(2) 编译器自动分配出来结构体的内存（比如定义为全局变量或局部变量）肯定是对齐的。

(3) 查阅帮助文档的malloc部分,mdk的标准malloc申请的内存区时8字节对齐的。

(4) 若自定义的malloc函数本身没有对分配的内存实现4字节或以上的对齐操作，分配出来的不对齐的内存，编译器是不知道的，所以很可能会产生问题。

此时最好的解决方式在内存池数组前添加\_\_align(4)关键字，只需保证自定义malloc分配出来的首地址是4字节对齐。比如：\_\_align(4) u8 mem1base[MEM1\_MAX\_SIZE];

**48.使用过哪些操作系统、编程语言及MCU（极光尔沃嵌入式软件开发面试题目考试）。**

clipboard.png

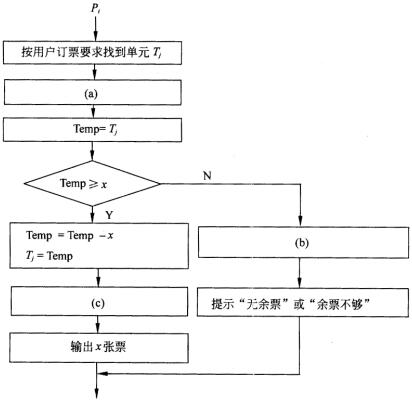
**49.对一系列数据进行排序有几种方法（极光尔沃嵌入式软件开发面试题目考试）。**

clipboard.png

**50.考核操作系统的同步与互斥。**

**题1**

某航空公司机票销售系统有n个售票点，该系统为每个售票点创建一个进程Pi(i=1，2，…，n)管理机票销售。假设Tj(j=1，2，…，m)单元存放某日某航班的机票剩余票数，Temp为Pi进程的临时工作单元，x为某用户的订票张数。初始化时系统应将信号量S赋值为\_\_\_\_\_\_\_\_\_\_(1)。Pi进程的工作流程如下图所示，若用P操作和V操作实现进程间的同步与互斥，则图中空(a)、空(b)和空(c)处应分别填入\_\_\_\_\_\_\_\_\_\_(2)。



(1) B

A．0 B．1 C．2 D．3

(2) A

A．P(S)、V(S)和V(S)

B．P(S)、P(S)和V(S)

C．V(S)、P(S)和P(S)

D．V(S)、V(S)和P(S)

解析：在此题中，信号量S是一个互斥信号量，作用是避免多个进程同时对余票数量进行处理导致

问题，所以初值应置为1。程序中(a)应使用P(S)操作，代表占用资源，(c)是成功完成订票时释放资

源进行的操作，应该进行V(S)操作，(b)是指当前没有足够的余票，未完成出票，此时，也要释放资源，

所以也应该进行V(S)操作。

**题2**

若系统中存在n个等待事务Ti(i=0,1,2,…,n-1),其中：T0正等待被T1锁住的数据项A1,T1正等待被T2锁住的数据项A2,......,Ti正等待被Ti+1锁住的数据项Ai+1,......,Tn-1正等待被T0锁住的数据项A0,则系统处于(\_\_\_\_\_\_)状态。

A.封锁

**B.死锁**

C.循环

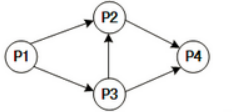
D.并发处理

答案：B

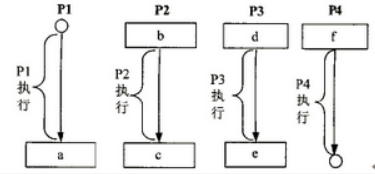
解析：题目所描述的情况为环路等待，此时系统处于死锁状态。

**题3**

进程P1、P2、P3和P4的前趋图如下：



若用PV操作控制进程P1~P4并发执行的过程，则需要设置5个信号量S1、S2、S3、S4和S5,且信号量S4-S5的初值都等于0。下图中a、b和c处应分别填写(\_\_\_\_\_\_)；d、e和f处应分别填写(\_\_\_\_\_\_)。



A.V(S1)V(S2)、P(S1)V(S3)和V(S4)

B.P(S1)V(S2)、P(S1)P(S2)和V(S1)

**C.V(S1)V(S2)、P(S1)P(S3)和V(S4)**

D.P(S1)P(S2)、V(S1)P(S3)和V(S2)

**A.P(S2)、V(S3)V(S5)和P(S4)P(S5)**

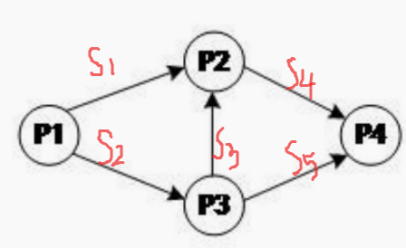
B.V(S2)、P(S3)V(S5)和V(S4)P(S5)

C.P(S2)、V(S3)P(S5)和P(S4)V(S5)

D.V(S2)、V(S3)P(S5)和P(S4)V(S5)

解析：本题只要将信号量在前趋图中标识出来，题目就非常容易解了。在前趋图中，每个箭头

统架构对应一个信号量，编号从左至右，从上至下，由小到大，如图所示。



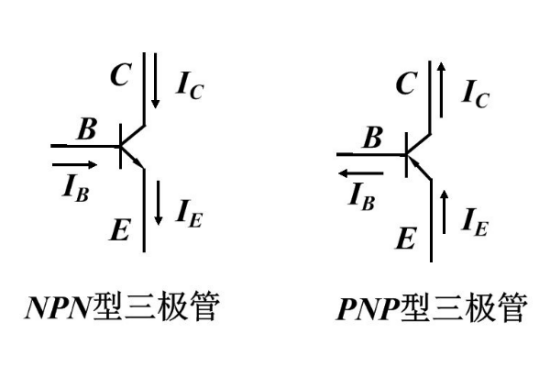
标出信号量之后，从某进程引出的信号量，在该进程末尾需要对信号量执行V操作，而信号量箭头指

向的进程开始处应有信号量的P操作。以P1为例，S1与S2从P1发出，所以在P1末尾处要执行V

(S1)V(S2),而在P2开始处应执行P(S1)P(S3)。

特殊解法：根据前趋图快速求出每个进程要执行P操作和V操作的个数，根据答案找出匹配的即可。

**52.三极管有哪三极。**



1. 基极

用于激活晶体管。（名字的来源，最早的点接触晶体管有两个点接触放置在基材上，而这种基材形成了底座连接。）

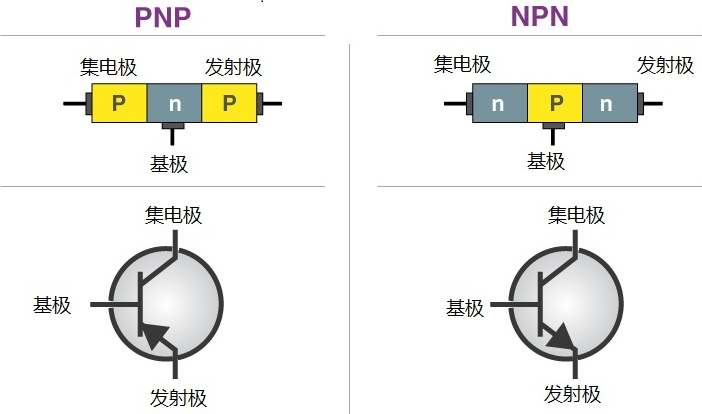
1. 集电极

三极管的正极。（因为收集电荷载体）

1. 发射极

三极管的负极。（因为发射电荷载流子）

**51.描述NPN与PNP的工作原理。**



**PNP**

**PNP**是一种BJT，其中一种n型材料被引入或放置在两种p型材料之间。在这样的配置中，设备将控制电流的流动。PNP晶体管由2个串联的晶体二极管组成。二极管的右侧和左侧分别称为集电极-基极二极管和发射极-基极二极管。

**NPN**

**NPN**中有一种 p 型材料存在于两种 n 型材料之间。**NPN晶体管基本上用于将弱信号放大为强信号**。在 NPN 晶体管中，电子从发射极区移动到集电极区，从而在晶体管中形成电流。这种晶体管在电路中被广泛使用。

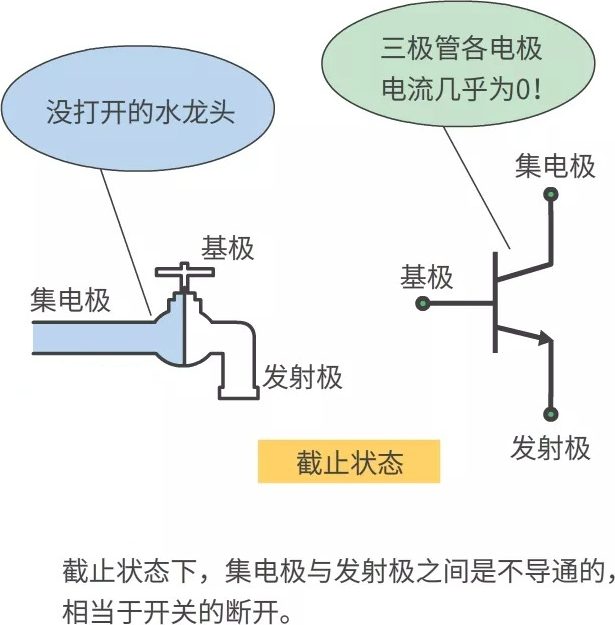
**52.三极管的3种工作状态**

分别是截止状态、放大状态、饱和状态。接下来分享我在微信公众号（借用一下，原作者看到的话，提醒一下）看到的一种通俗易懂的讲法：

1、三极管工作原理-截止状态

三极管的截止状态，这应该是比较好理解的，当三极管的发射结反偏，集电结反偏时，三极管就会进入截止状态。

这就相当于一个关紧了的水龙头，水龙头里的水是流不出来的。



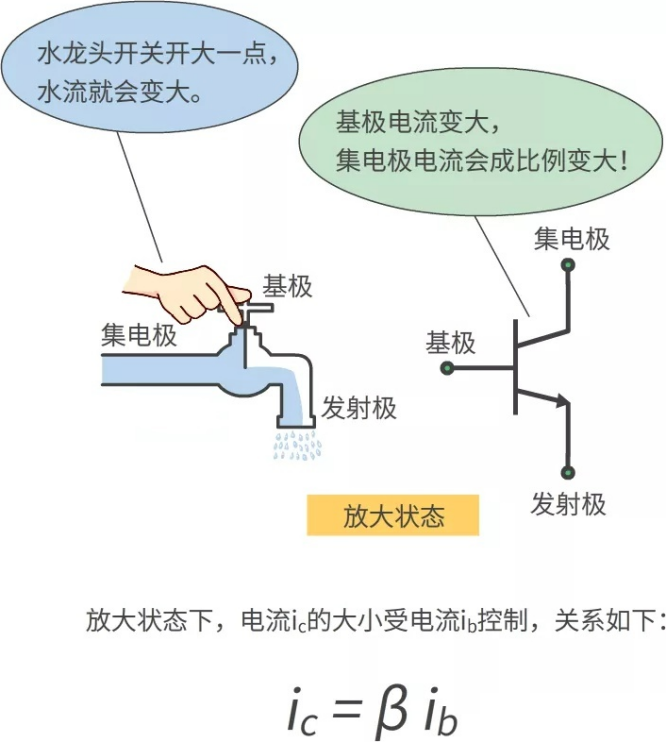
截止状态下，三极管各电极的电流几乎为0，集电极和发射极互不相通。

2、三极管工作原理-放大状态

当三极管发射结正偏，集电结反偏，三极管就会进入放大状态。

在放大状态下，三极管就相当于是一个受控制的水龙头，水龙头流出水流的大小受开关（基极）控制，开关拧大一点，流出的水就会大一点。

也就是放大状态下，基极的电流大一点，集电极的电流也会跟着变大！并且ic与ib存在一定比例关系，ic = β ib，β是直流电流放大系数，表示三极管放大能力的大小。

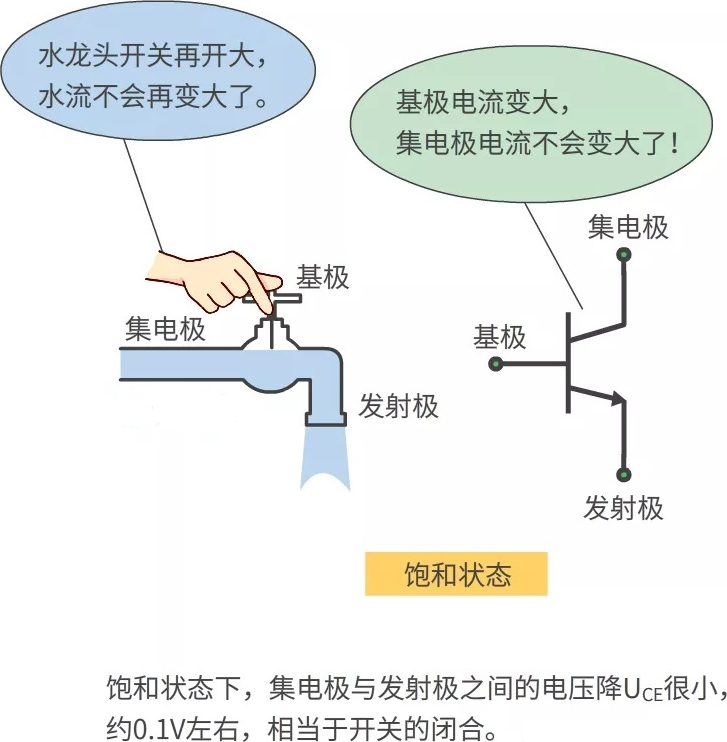


3、三极管工作原理-饱和状态

当三极管发射结正偏，集电结正偏时，三极管工作在饱和状态。

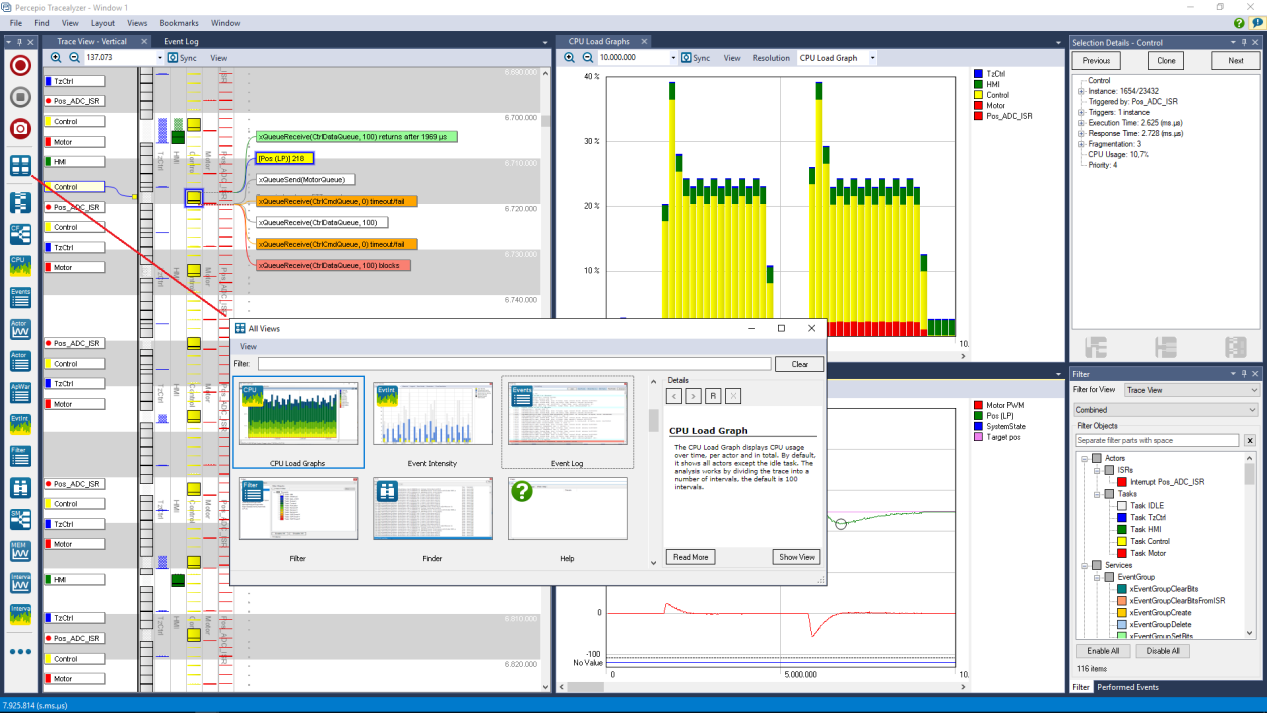
在饱和状态下，三极管集电极电流ic的大小已经不受基极电流ib的控制，ic与ib不再成比例关系。

饱和状态下的三极管基极电流ib变大时，集电极电流ic也不会变大了，这就相当于水龙头的开关已经开得比较大了，开关再开大时，流出的水流也不会再变大了。



**53.你用过嵌入式系统调试工具有哪些，请详细描述。**

Tracealyzer是Percepio 公司开发的一款用于RTOS或基于linux的嵌入式软件系统的可视化跟踪工具，对系统运行时的行为提供了前所未有的洞察方法。帮助开发人员加快固件的开发，减少对系统验证和性能优化所需要的时间。目前Tracealyzer提供了30多种相互关联的运行时行为视图，包括任务调度、中断、任务之间的相互作用，以及从应用程序代码中记录的用户事件。Tracealyzer作为传统调试的补充，提供更高层次的调试视图，非常适合理解典型的实时问题。



**Tracealyzer支持的OS**

FreeRTOS、Keil RTX5、Linux、On Time RTOS-32、ThreadX、µC/OS-III、VxWorks、Zephyr、OpenVX/Synopsys

**为什么要使用Tracealyzer**

源代码本身并不能完全描述多任务软件系统在运行时的实际行为。实时行为还取决于许多其他因素，如任务和中断的时间，它们的相互作用和输入。基于RTOS固件的有效开发需要很好地理解实时操作系统任务、中断和操作系统内核之间的执行、时序和交互。使用Tracealyzer，你可以记录实时行为，软件跟踪与强大的可视化相结合，提高软件的开发效率和质量，从而提升开发效率，降低项目风险和缩短产品上市时间。

● **更快的排除故障Tracealyzer可以捕获难以复现和分析的罕见、不定时发生的错误**。许多嵌入式软件的问题，可以短时间内被解决。Tracealyzer记录可以与在实验室内现有的调试器同步使用，或部署在系统中捕获罕见的偶发性错误。

● **更好的软件质量Tracealyzer不仅是用于特定困难问题的“灭火器”，你还可以用于发现和避免潜在的问题，如阻塞超时的系统调用**。此外，Tracealyzer让你更好地了解当前系统运行动态，所以在设计新的功能时，可以避免不当的设计可能导致的定时、CPU使用、调度或其他任务交互的相关问题。

● **更好的软件性能改进的洞察力也可以让你找到提高嵌入式软件性能的新方法**。可能是意外惊喜，任务时间上的小变化可能会带来显著的性能提升。Tracealyzer提供了多种方式寻找“热点”，任务被更高优先级的任务延迟，可能可以早或晚减少资源冲突。如果没有合适的可视化，这样的优化是很难发现的。这样，你就可以得到一个响应更快速的软件系统，或者在同一个硬件平台上融入更多的软件功能。

● **优化控制系统控制系统开发人员可以受益于绘制自定义应用程序数据的支持，我们称之为用户事件**。例如控制信号（传感器输入和执行器输出）图可以与软件时序相互关联，以便更好地了解控制性能的异常。此外，任务时序（例如周期）的图可以用来研究影响控制性能的软件时序变化。

● **其他的好处改进的洞察力水平也允许更快的引入新的开发人员，以及评估新的第三方软件性能**，如嵌入式数据库，触摸屏驱动程序或通信协议栈的更好的可能性。最后，由于我们支持常见的几种针对嵌入式软件的操作系统，你可以经常保持Tracealyzer支持，即使切换到另一个操作系统。

**54.alpha 测试（α测试）与beta测试（β测试）的区别?**

定义：α测试是在用户组织模拟软件系统的运行环境下一种验收测试，又用户或第三方测试公司进行测试，模拟各类用户为对即将面市的软件产品进行测试，试图发现并修改错误。

β测试是用户公司组织各方面的典型终端用户在日常工作中实际使用beta版本，并要求用户报告异常情况，提出批评意见。

两者的主要区别：

1、测试的场所不同。

α测试是指把用户请到开发方的产所来测试，β测试是指在一个或多个用户的场所进行测试。

2、测试环境不同。

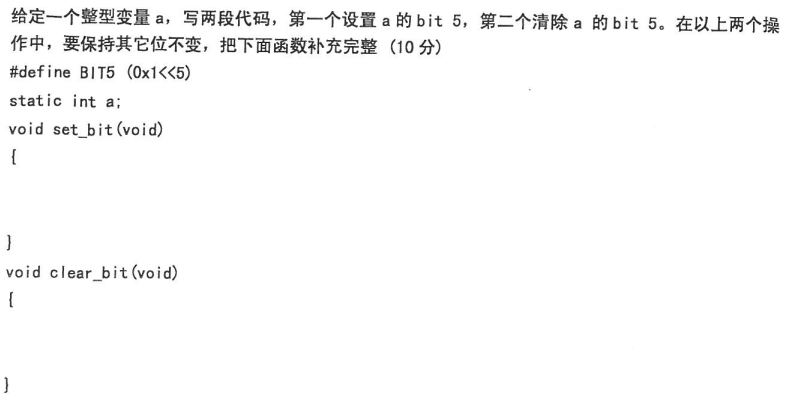
α测试的环境受开发方控制的用户的数量相对较少，时间比较集中。而β测试的环境是不受开发方控制的，谁也不知道用户如何折磨软件，用户数量相对较多，时间不集中。

3、测试周期的不同。

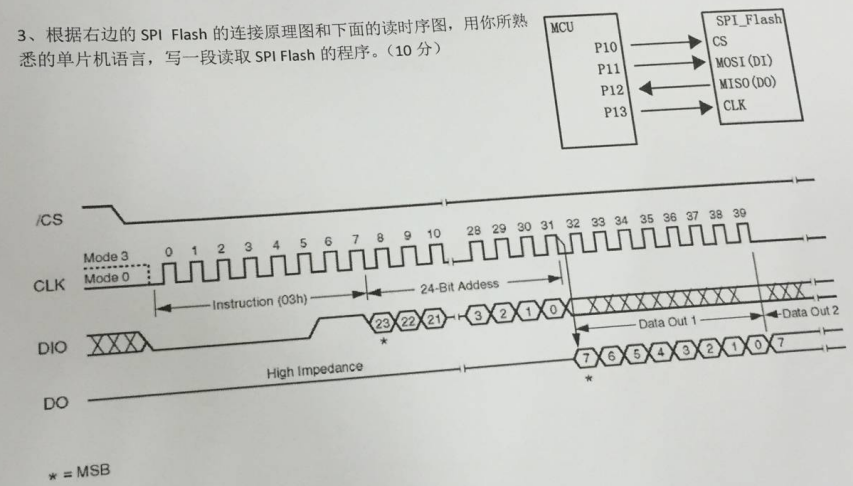
一般地，α测试先于β测试执行通用软件产品需要较大规模的β测试，测试周期较长。

**四、编程题**

**1.**



**2.**



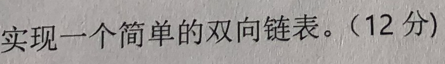
**3.请给出相应的滤波算法，给相应的代码（极光尔沃嵌入式软件开发面试题目考试）。**

clipboard.png

**4.请给出printf的实现方法（极光尔沃嵌入式软件开发面试题目考试）。**

clipboard.png

**5.实现一个简单的双向链表（极光尔沃嵌入式软件开发面试题目考试）。**



**6.请写出swap函数的实现方法（极光尔沃嵌入式软件开发面试题目考试）。**

clipboard.png

**7.请写出strcpy函数的实现方法（极光尔沃嵌入式软件开发面试题目考试）。**

clipboard.png

**8.请写出strlen函数的实现方法（极光尔沃嵌入式软件开发面试题目考试）。**

clipboard.png

**五、翻译题**

