

1、嵌入式微处理器的体系结构	2
2、ARM 指令集的特征	3
CISC 与 RISC	3
x86 指令集与 arm 指令集	4
ARM 汇编	5
冯·诺伊曼体系与哈佛体系	6
3、复位的原理，流程	7
版本 1 PPT 80C51 复位 上电自动复位、按键复位	7
版本 2 PPT PIC MCU PIC 复位 6 种复位	8
版本 3 (from ChatGPT) 软件复位和硬件复位	10
版本 4 (from ChatGPT) 软件复位和硬件复位	11
4. 嵌入式攻击：	12
4.1 根据攻击对象分类	12
4.2 根据发起攻击的代理工具或手段分类	12
4.3 RFID 攻击	13
4.4 侧信道攻击 SCA (PPT 中一部分翻译为旁路攻击)	13
6. 嵌入式系统常见的工作模式 (低功耗，休眠，睡眠)，模式的特征，为什么需要这些模式	20
PPT 相关内容	20
CM3 低功耗模式需求：	20
CM3 低功耗模式的优势：	20
网上资料	21
STM32F1 低功耗模式	21
睡眠模式：	21
停止模式：	22
待机模式：	23
为什么嵌入式需要低功耗？ (GPT 答案)	23
7、嵌入式系统攻击分类	24
01 根据攻击对象分类	24
02 提高输入输出的可靠性	24
软件防御技术关键是代码分块的粒度粗细和等价实现的数量，这句话如何理解？	25
几种安全增强方法：	26

1、嵌入式微处理器的体系结构

冯·诺依曼 (von neumann)/普林斯顿(Princeton)体系结构:

程序存储器和数据存储器共用一个存储空间，统一编址；
采用统一的地址及数据总线，指令和数据的宽度相同；
使用灵活（例如 代码远程更新 OTA）。

哈佛体系结构

程序存储器和数据存储器是独立编址的两个存储空间；
这种分离的程序总线 and 数据总线可允许在一个机器周期内同时获取指令码（程序存储器）和操作数（数据存储器），从而提高执行速度，提高数据的吞吐率，可靠性高（大部分程序存储器是只读存储器）。

嵌入式微处理器指令系统

复杂指令集系统(Complex Instruction Set Computer, CISC)

早期的计算机采用复杂指令集计算机(CISC)体系，如 Intel 公司从 8086 到 Pentium 系列 CPU 采用 CISC 体系结构的计算机各种指令的使用频率相差悬殊。统计表明，大概有 20% 的比较简单指令被反复使用，使用量约占整个程序的 80%；而有 80% 左右的指令则很少使用，其使用量约占整个程序的 20%，即指令的 2/8 规律。

在 CISC 中，为了支持目标程序的优化，支持高级语言和编译程序，增加了许多复杂的指令，用一条指令来代替一串指令，简化了软件设计，却增加了硬件的复杂程度。而且这些复杂指令并不等于有利于缩短程序的执行时间。

在 VLSI(Very-large-scale integration)制造工艺中要求 CPU 控制逻辑具有规整性，而 CISC 为了实现大量复杂的指令，控制逻辑极不规整，给 VLSI 工艺造成很大困难

精简指令集系统(Reduced Instruction Set Computer, RISC)

RISC 是在 CISC 的基础上产生并发展起来的；
RISC 简化指令系统使计算机的结构更加简单合理，提高运算效率；
优先选取使用频率高的、很有用但不复杂的指令，避免使用复杂指令；
固定指令长度，减少指令格式和寻址方式种类；
指令之间各字段的划分比较一致，各字段的功能也比较规整；
采用 Load/Store 指令访问存储器，其余指令都在寄存器之间进行；
增加通用寄存器数量，算术/逻辑运算的操作数都在寄存器中存取；
大部分指令控制在一个或小于一个机器周期内完成；
以硬布线控制逻辑为主，不用或少用微码控制。

CISC 与 RISC 之间的主要差异

指令系统：RISC 设计者把主要精力放在那些经常使用的指令上，尽量使它们具有简单高效的特色。对不常用的功能，常通过组合指令来实现。而 CISC 的指令系统比较丰富，有专用指令来完成特定的功能。

存储器操作：RISC 对存储器操作有限制，使控制简单化；而 CISC 机器的存储器操作指令多，操作直接。

程序：RISC 汇编语言程序一般需要较大的内存空间，实现特殊功能时程序复杂，不易设计；而 CISC 汇编语言程序编程相对简单，科学计算及复杂操作的程序设计相对容易，效率较高。

CPU：由于 RISC CPU 包含较少的单元电路，因而面积小、功耗低；而 CISC CPU 包含丰富的电路单元，因而功能强、面积大、功耗大。

设计周期：RISC 微处理器结构简单，布局紧凑，设计周期短，且易于采用最新技术；CISC 微处理器结构复杂，设计周期长。

易用性：RISC 微处理器结构简单，指令规整，性能容易把握，易学易用；CISC 微处理器结构复杂，功能强大，实现特殊功能容易。

应用范围：RISC 更适用于嵌入式系统；而 CISC 则更适合于通用计算机。

2、ARM 指令集的特征

CISC 与 RISC

- **CISC: Complex Instruction Set Computer**，复杂指令集计算机：为了便于编程和提高存储器访问效率的芯片设计体系。在 20 世纪 90 年代中期之前，大多数的微处理器都采用 CISC 体系，包括 Intel 的 80x86 和 Motorola 的 68K 系列等。即通常所说的 x86 架构就是属于 CISC 体系的。其指令特征包括：
 - **使用微代码：**指令集可以直接在微代码存储器（比主存储器的速度快很多）里执行，新设计的处理器，只需增加较少的电晶体就可以执行同样的指令集，也可以很快地编写新的指令集程序。
 - **庞大的指令集：**可以减少编程所需要的代码行数，减轻程序员的负担。高级语言对应的指令集：包括双运算元格式、寄存器到寄存器、寄存器到存储器以及存储器到寄存器的指令。
 - **优点：**能够有效缩短新指令的微代码设计时间，允许设计师实现 CISC 体系机器的向上兼容。新的系统可以使用一个包含早期系统的指令超集合，也可以使用较早电脑上使用的相同软件。另外微程序指令的格式与高级语言相匹配，因而编译器并不一定要重新编写。
 - **缺点：**指令集以及芯片的设计比上一代产品更复杂，不同的指令，需要不同的时钟周期来完成，执行较慢的指令，将影响整台机器的执行效率。
- **RISC: Reduced Instruction Set Computer**，精简指令集计算机：为了提高处理器运行速度而设计的芯片设计体系。它的关键技术在于流水线操作（**Pipelining**）：在一个时钟周期里完成多条指令。而超流水线以及超标量技术已普遍在芯片设计中使用。RISC 体系多用于非 x86 阵营高性能微处理器 CPU，像 HOLTEK MCU 系列等。其指令特征包括：
 - **精简指令集：**包含了简单、基本的指令，通过这些简单、基本的指令，就可以组合成复杂指令。
 - **同样长度的指令：**每条指令的长度都是相同的，可以在一个单独操作里完成。
 - **单机器周期指令：**大多数的指令都可以在一个机器周期里完成，并且允许处理器在同一时间内执行一系列的指令。

- **优点：**在使用相同的芯片技术和相同运行时钟下，RISC 系统的运行速度将是 CISC 的 2~4 倍。由于 RISC 处理器的指令集是精简的，它的内存管理单元、浮点单元等都能设计在同一块芯片上。RISC 处理器比相对应的 CISC 处理器设计更简单，所需要的时间将变得更短，并可以比 CISC 处理器应用更多先进的技术，开发更快的下一代处理器。
- **缺点：**多指令的操作使得程序开发者必须小心地选用合适的编译器，而且编写的代码量会变得非常大。另外就是 RISC 体系的处理器需要更快的存储器，这通常都集成于处理器内部，就是 L1 Cache（一级缓存）。

两种计算机的对比如下表所示：

	CISC	RISC
指令的形成	指令复杂，故采用微指令码控制单元的设计	90%是由硬件直接完成，只有 10%的指令是由软件以组合的方式完成，指令执行时间上较短，但所需 ROM 空间相对较大
寻址模式	需要较多的寻址模式，占用的汇流排周期较多	只有少数的寻址模式
指令的执行	指令的格式长短不一，执行时的周期次数也不统一，执行步骤过多，闲置的单元电路等待时间增长，不利于平行处理的设计，效能上占下风	与 CISC 刚好相反，适合采用流水线处理架构的设计，进而可以达到平均一周期完成一指令的方向努力。效能更好，但 RISC 因指令精简化后造成应用程式码变大，需要较大的存储器空间，且存在指令种类较多等等的缺点。

x86 指令集与 arm 指令集

- **x86 指令集：**是 Intel 为其第一块 16 位 CPU(i8086)专门开发的，后来的电脑中为提高浮点数据处理能力而增加的 X87 芯片系列数学协处理器以及使用 X87 指令，以后就将 X86 指令集和 X87 指令集统称为 X86 指令集。虽然随着 CPU 技术的不断发展，Intel 陆续研制出更新型的 i80386、i80486，但为了保证电脑能继续运行以往开发的各类应用程序以保护和继承丰富的软件资源，所以 Intel 公司所生产的所有 CPU 仍然继续使用 X86 指令集，所以它的 CPU 仍属于 X86 系列。由于 Intel X86 系列及其兼容 CPU 都使用 X86 指令集，所以就形成了今天庞大的 X86 系列及兼容 CPU 阵容。除了具备上述 CISC 的诸多特性外，X86 指令集有以下几个突出的缺点：
 - **通用寄存器组——对 CPU 内核结构的影响。**X86 指令集只有 8 个通用寄存器，所以，CISC 的 CPU 执行是大多数时间是在访问存储器中的数据，而不是寄存器中的。这就拖慢了整个系统的速度。RISC 系统往往具有非常多的通用寄存器，并采用了重叠寄存器窗口和寄存器堆等技术使寄存器资源得到充分的利用。
 - **解码——对 CPU 的外核的影响。**解码器（Decode Unit），这是 x86 CPU 才有的东西。其作用是把长度不定的 x86 指令转换为长度固定的类似于 RISC 的

指令，并交给 RISC 内核。解码分为硬件解码和微解码，对于简单的 x86 指令只要硬件解码即可，速度较快，而遇到复杂的 x86 指令则需要进行微解码，并把它分成若干条简单指令，速度较慢且很复杂。Athlon 也好，PIII 也好，老式的 CISC 的 X86 指令集严重制约了他们的性能表现。

- **寻址范围小——约束了用户需要。**即使 AMD 研发出 X86-64 架构时，虽然也解决了传统 X86 固有的一些缺点，比如寻址范围的扩大，但这种改善并不能直接带来性能上的提升。
- **arm 指令集：**以 RISC 为架构体系的 ARM 指令集的指令格式统一，种类比较少，寻址方式也比复杂指令集少。当然处理速度就提高很多。ARM 处理器都是所谓的精简指令集处理机（RISC）。其所有指令都是利用一些简单的指令组成的，简单的指令意味着相应硬件线路可以尽量做到最佳化，而提高执行速率，相对的使得一个指令所需的时间减到最短。而因为指令集的精简，许多工作都必须组合简单的指令来完成，而针对较复杂组合的工作便需要由编译器(compiler) 来执行，而 CISC 体系的 X86 指令集因为硬体所提供的指令集较多，所以许多工作都能够以一个或是数个指令来代替，编译器的工作因而减少许多。除了具备上述 RISC 的诸多特性之外，可以总结 ARM 指令集架构的其它一些特点如下：
 - 体积小，低功耗，低成本，高性能；
 - 支持 Thumb（16 位）/ARM（32 位）双指令集，能很好的兼容 8 位 /16 位器件；
 - 大量使用寄存器，指令执行速度更快；
 - 大多数数据操作都在寄存器中完成；
 - 寻址方式灵活简单，执行效率高；
 - 指令长度固定；
 - 流水线处理方式；
 - load-store 结构。
- 另外 ARM 还具有一些非 RISC 思想的指令架构：
 - 允许一些特定指令的执行周期数字可变，以降低功耗，减小面积和代码尺寸；
 - 增加了桶形移位器来扩展某些指令的功能；
 - 使用了 16 位的 Thumb 指令集来提高代码密度；
 - 使用条件执行指令来提高代码密度和性能；
 - 使用增强指令来实现数字信号处理的功能。

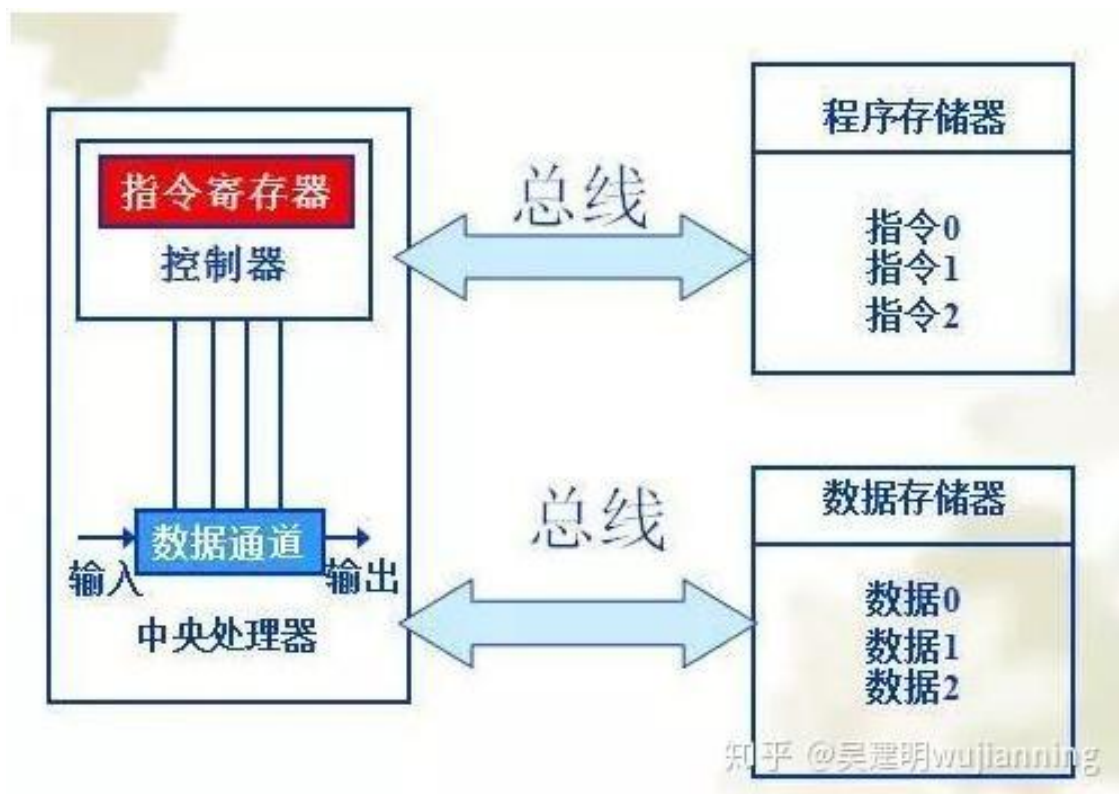
ARM 汇编

- 两种风格：
 - ARM 官方的 ARM 汇编风格：指令一般用大写，Windows 中的 IDE 开发环境
 - GNU 风格的 ARM 汇编：指令一般用小写
- ARM 汇编的特点：
 - 采用 RISC 架构，CPU 本身不能直接读取内存，而需要先将内存中内容加载入 CPU 中通用寄存器中才能被 CPU 处理
 - ldr（load register）指令将内存内容加载入通用寄存器
 - str（store register）指令将寄存器内容存入内存空间中
 - ldr/str 组合用来实现 ARM CPU 和内存数据交换
- 8 种寻址方式：
 - 寄存器寻址 mov r1, r2

- 立即(立即数)寻址 `mov r0, #0xFF00`
- 寄存器移位寻址 `mov r0, r1, lsl #3`
- 寄存器间接寻址 `ldr r1, [r2]` 表示内存，内存地址存在 r2 这个寄存器中，把内存地址里的值给 r1
- 基址变址寻址 `ldr r1, [r2, #4]` 内存地址在 r2+4 里面
- 多寄存器寻址 `ldmia r1!, {r2-r7, r12}` 一次访问多个寄存器
- 堆栈寻址 `stmfd sp!, {r2-r7, lr}`
- 相对寻址 `beq flag`
- 指令后缀：
 - B (byte) 功能不变，操作长度变为 8 位
 - H (half word) 功能不变，长度变为 16 位
 - S (signed) 功能不变，操作数变为有符号
 - 如 `ldr ldrb ldrrh ldrsb ldrrsh`
 - S (S 标志) 功能不变，影响 CPSR 标志位
 - 如 `mov` 和 `movs` `movs r0, #0`

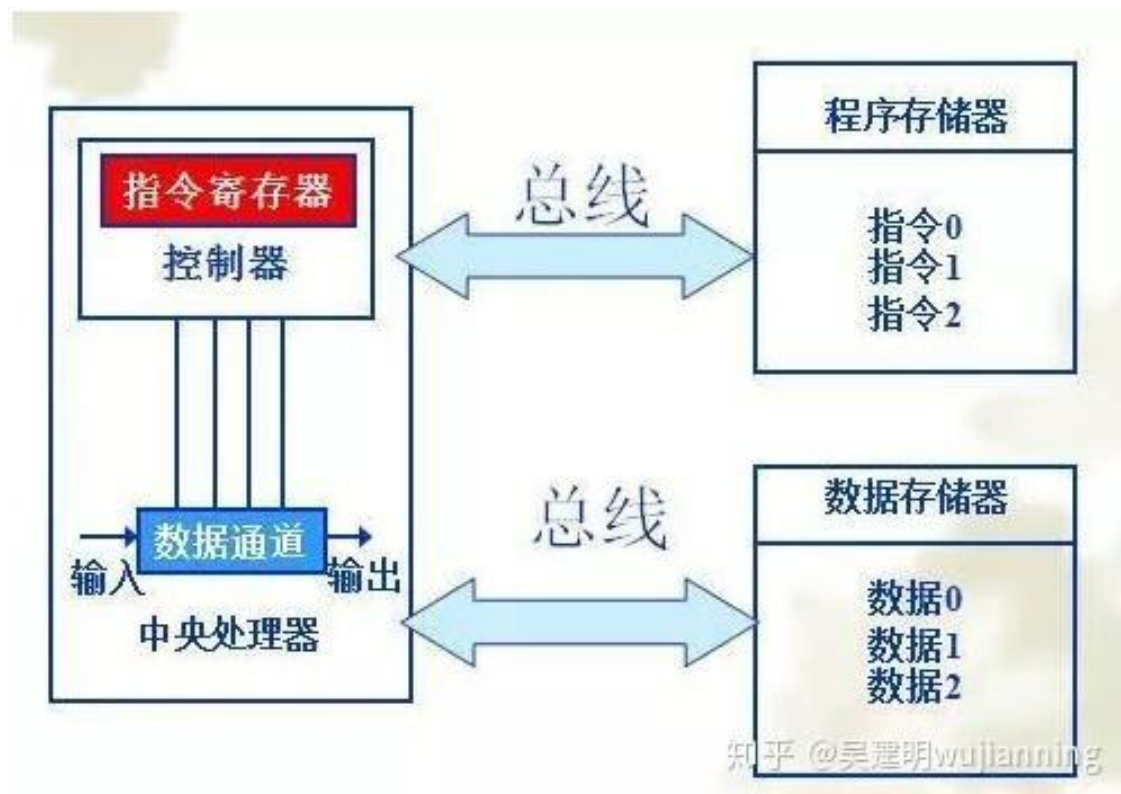
冯·诺伊曼体系与哈佛体系

冯·诺伊曼体系：



- 数据与指令都存储在同一存储区中，取指令与取数据利用同一数据总线。
- 被早期大多数计算机所采用。
- ARM7——冯诺依曼体系结构简单,但速度较慢。取指不能同时取数据。

哈佛体系：



- 程序存储器与数据存储器分开.
- 提供了较大的存储器带宽，各自有自己的总线。
- 适合于数字信号处理.
- 大多数 DSP 都是哈佛结构.
- ARM9 是哈佛结构，取指和取数在同一周期进行，提高速度，改进哈佛体系结构分成三个存储区：程序、数据、程序和数据共用。

3、复位的原理，流程

版本 1 PPT 80C51 复位 上电自动复位、按键复位

1) 复位操作

◆ 复位是单片机的初始化操作，主要功能是把 PC 初始化为 0000H，使单片机从 0000H 单元开始执行程序。当由于程序运行出错或操作错误使系统处于死锁状态时，为摆脱困境，可以按复位键以重新启动，也可以通过 WDT 看门狗定时器来强迫复位（WDT 可在单片机系统受干扰使程序不能正常运行时，自动产生复位信号。

◆ 除 PC 之外，复位操作还对其它一些特殊功能寄存器有影响

◆ 复位操作还对单片机的个别引脚信号有影响。例如在复位期间，ALE 和/PSEN 信号变为无效状态，即 ALE=0，/PSEN =1

2) 复位信号

◆ RST 引脚是复位信号的输入端。复位信号是高电平有效，其有效时间应持续 24 个振荡周期(即 2 个机器周期) 以上。若使用频率为 6MHz 的晶振，则复位信号应持续 4μs 以上

◆ 整个复位电路包括芯片内、外两部分。外部电路产生的复位信号送施密特触发器，再由片内复位操作。

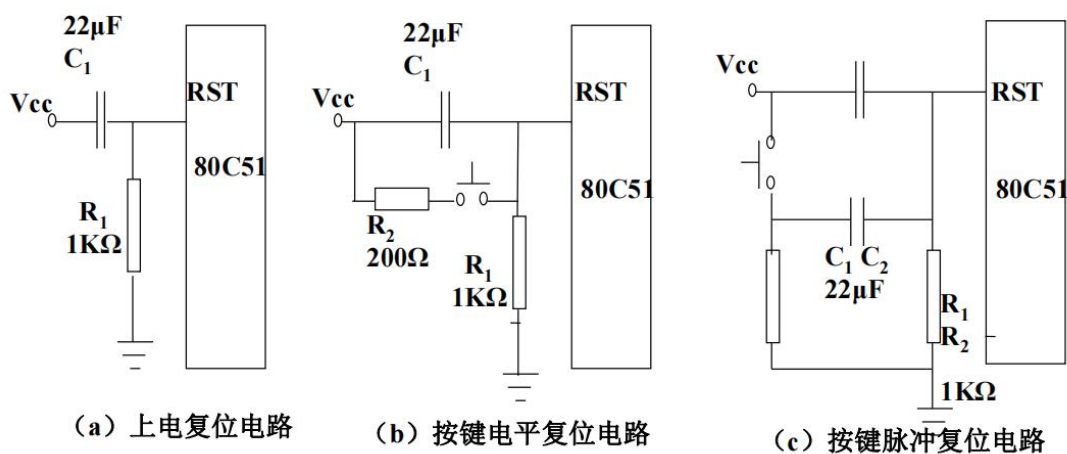
◆ 有上电自动复位、按键复位两种方式

上电自动复位是通过外部复位电路的电容充电来实现的，只要电源的上升时间不超过 1ms，就可以实现上电自动复位，即接通电源就完成了系统的复位初始化。

按键复位有电平方式和脉冲方式两种。

- 按键电平复位是通过使复位端经电阻与 Vcc 电源接通而实现的
- 按键脉冲复位则是利用 RC 微分电路产生的正脉冲来实现的

电路图如下：



版本 2 PPT PIC MCU PIC 复位 6 种复位

PIC16F/LF722A/723A 可以区分各种复位：

- 上电复位 (POR)
- 正常工作期间的 WDT 复位
- 休眠期间的 WDT 复位
- 正常工作期间的 MCLR 复位
- 休眠期间的 MCLR 复位
- 欠压复位 (BOR)

■ 一些寄存器在任何复位条件下都不受影响；其状态在 POR 时未知，在任何其他复位时不变。

■ 多数其他寄存器在以下复位时会复位为“复位状态”：

- 上电复位 (POR)
- MCLR 复位
- 休眠期间的 MCLR 复位
- WDT 复位
- 欠压复位 (BOR)

■ 多数寄存器不受 WDT 唤醒的影响，因为 WDT 唤醒被视为恢复正常工作。在不同的复位

条件下，会将 TO 和 PD 位置 1 或清零。这些位在软件中用来确定复位的性质。

MCLR

- PIC16F/LF722A/723A在MCLR复位电路中有一个**噪声滤波器**。滤波器能检测并滤除小脉冲干扰信号。
- 施加到引脚的电压超出其规范将导致MCLR 复位以及ESD 事件期间电流超出器件规范。**建议不将MCLR 引脚直接与VDD 连接**，使用图给出的RC 网络。
- 内部MCLR 选项通过清零配置字寄存器中的MCLRE 位来使能。当MCLRE = 0 时，将内部生成到芯片的复位信号。当MCLRE = 1 时， RE3/MCLR 引脚将成为外部复位输入。在此模式下，**RE3/MCLR 引脚具有到VDD 的弱上拉**。选择内部MCLR 选项不会影响在线串行编程。

上电复位（POR）

- 片上POR 电路将芯片保持在复位状态，直到VDD 达到正常工作所需的电平为止。如果使能BOR，则不应用最大上升时间规范。BOR 电路将器件保持在复位状态，直到VDD 达到VBOR 。
- 器件开始正常工作（退出复位状态）时，必须满足器件工作参数（即，电压、频率以及温度等）才能确保其正常工作。如果这些条件未满足，则器件必须保持在复位状态，直到满足工作条件为止。

上电延时定时器（PWRT）

- 上电延时定时器仅在上电时（从上电复位或欠压复位）提供一个**固定的64 ms（标称值）**超时。
- 上电延时定时器根据WDT 振荡器工作。只要PWRT 有效，器件就保持在复位状态。PWRT 延时允许VDD 上升到所能接受的电平。**配置位PWRTS 可以禁止（如果置1）或使能（如果清零或编程）上电延时定时器**。使能欠压复位时，尽管不是必需的，还是应该使能上电延时定时器。
- **各个芯片的上电延时定时器延时都会有所不同，原因是：**
 - VDD 不同
 - 温度不同
 - 工艺不同

看门狗定时器（WDT）

- WDT 具有如下特性：
 - 与Timer0 共用一个8 位预分频器
 - 超时周期从17 ms 到2.2 s （标称值）
 - 由配置位使能
- WDT 振荡器
WDT 以31 kHz 的内部振荡器作为其时基。
- WDT 控制
WDTE 位位于配置字寄存器1中，置1 时， WDT 将持续运行。
OPTION 寄存器的PSA 和PS<2:0> 位控制WDT周期。

欠压复位（BOR）

- 欠压复位通过**编程配置寄存器**中的BOREN<1:0> 位来使能。使用两个位来使能BOR。当BOREN = 11 时，将始终使能BOR。当BOREN = 10 时，将使能BOR，但在休眠期间禁止BOR。当BOREN = 0X 时，将禁止BOR。
- 欠压跳变点可以通过配置寄存器中的BORV 位在两个跳变点之间进行选择。
- **在上电复位和欠压复位之间，可为执行保护覆盖实现整个电压范围。**
- 如果VDD 低于VBOR 的时间大于参数TBOR，则欠压条件将复位器件。无论VDD压摆率如何，都将发生此情况。如果VDD 低于VBOR的时间大于参数TBOR，复位不一定会发生。如果上电延时定时器运行时VDD 下降到低于VBOR，芯片将返回到欠压复位状态，并将重新初始化上电延时定时器。一旦VDD 上升到高于VBOR 时，上电延时定时器将执行64 ms 的复位。

版本 3 （from ChatGPT）软件复位和硬件复位

在嵌入式系统中，复位是指将系统恢复到初始状态的过程。复位可以分为**软件复位**和**硬件复位**两种方式。

硬件复位是通过物理电路实现的，当系统接收到复位信号时，会强制将处理器、存储器、外设等所有硬件设备恢复到初始状态。硬件复位通常由复位电路或复位按钮触发，其原理是将处理器的复位引脚拉低，从而强制处理器进入复位状态。硬件复位的流程如下：

1. 系统接收到复位信号。
2. 处理器进入复位状态，停止执行任何指令。
3. 所有外设和存储器设备被清空或初始化。
4. 处理器根据启动地址开始执行初始化程序。

软件复位是通过软件指令实现的，当系统执行软件复位指令时，处理器会将所有寄存器、存储器、外设等设备恢复到初始状态。软件复位的原理是通过在程序中执行一段特定的代码

来实现。软件复位的流程如下：

1. 程序执行软件复位指令。
2. 处理器将所有寄存器、存储器、外设等设备恢复到初始状态。
3. 处理器根据启动地址开始执行初始化程序。

总之，无论是硬件复位还是软件复位，其目的都是将系统恢复到初始状态，以便重新启动或重新初始化系统。

版本 4（from ChatGPT）软件复位和硬件复位

在嵌入式系统中，复位（Reset）是一个重要的过程，用于将系统的所有组件和状态恢复到初始状态，以确保系统在启动时处于可预测和可控制的状态。复位过程主要包括**硬件复位**和**软件复位**两个方面。

1. 硬件复位：

硬件复位是通过电路和信号控制来实现的，其目的是将系统中的所有硬件元件恢复到初始状态。硬件复位可以分为两种类型：电源复位和外部复位。

- 电源复位（Power-on Reset）：电源复位是系统在通电时进行的一种复位操作。当嵌入式系统上电时，电源管理电路会向系统中的所有电路提供电源供应，并在电源稳定后发送一个复位信号，使得整个系统处于初始状态。
- 外部复位（External Reset）：外部复位是通过外部信号触发的一种复位操作。通常，嵌入式系统中会提供一个物理按键或者引脚，当用户按下该按键或者外部触发信号到来时，系统会收到一个复位信号，从而触发复位操作。

2. 软件复位：

软件复位是通过软件控制实现的，其目的是将系统中的软件执行状态恢复到初始状态。软件复位通常是由处理器执行的特定指令或者软件操作触发的。

- 处理器复位指令（Reset Instruction）：大多数处理器都提供了特殊的复位指令，当处理器执行该指令时，会将处理器内部状态以及与之相关的寄存器和标志位恢复到初始状态。具体的复位指令名称和使用方法根据处理器架构的不同而有所差异。
- 软件触发复位（Software-triggered Reset）：除了处理器复位指令外，还可以通过软件编写的特定程序代码来触发软件复位。例如，程序可以检测到某个特定的事件或者条件，然后执行一系列复位相关的操作，使系统恢复到初始状态。

综合硬件复位和软件复位，嵌入式系统中复位的流程可以描述如下：

1. 系统上电或者外部复位信号到来。
2. 电源管理电路稳定后发送复位信号，触发电源复位。
3. 系统中的所有硬件元件收到复位信号，将其状态恢复到初始状态。
4. 处理器接收到复位信号，执行处理器复位指令，将处理器内部状态恢复到初始状态。
5. 处理器根据复位向量（Reset Vector）跳转到特定的内存地址，该地址通常是存储系统的初始启动代码的位置。
6. 启动代码执行初始化操作，包括初始化寄存器、设备和外设等。
7. 启动代码加载操作系统或应用程序，并跳转到相应的入口点，系统进入正常运行状态。

需要注意的是，具体的复位流程可能因嵌入式系统的架构、处理器型号和设计要求而有所不同。上述流程只是一种一般性的描述，实际应用中可能会有细微的差异。

4. 嵌入式攻击：

4.1 根据攻击对象分类

利用嵌入式自身的安全缺陷，或者利用其外部不受信任的环境，嵌入式系统面临诸多攻击威胁，根据攻击对象不同，可将其分为隐私数据攻击、可用性攻击、代码完整性攻击等。

隐私数据攻击

这种攻击的目的是获取嵌入式系统内存储、传递或操作的敏感信息数据；防范这类攻击的主要手段是对敏感信息数据进行加密保护，但实现加密保护需要密钥，密钥的创建、存储、使用和销毁等，需要引入能够信任的密钥管理机制以保障其安全性。此外，还可通过访问控制对敏感信息数据进行保护。

可用性攻击

这类攻击挪用系统资源，扰乱系统的正常工作，使系统不能执行相应正常操作。可以通过在嵌入式系统中，添加可靠的资源分配管理组件，来防范这类攻击。

代码完整性攻击

这种攻击的目的是获取嵌入式这种攻击试图修改嵌入式系统相关数据或代码。防范这类攻击的重点是保证嵌入式系统自身代码的完整性，可以在运行前通过对嵌入式系统相关代码进行安全度量，检测代码是否被篡改。

4.2 根据发起攻击的代理工具或手段分类

软件攻击

如病毒，木马，蠕虫等通过软件代理对终端系统结构的薄弱环节发起的攻击。这类攻击是耗费代价较小，是较为常见的一种攻击。

硬件攻击

如硬件分解、电磁干扰、使用探针对嵌入式芯片内部的交互信息进行窃听等。这类攻击需要较为昂贵的基础设施要求，较难实现。

缓存溢出攻击

向缓存中传送超出容量的数据，并伴以一段恶意执行代码及用来覆盖调用程序返回地址的地址数据，造成缓存溢出。当功能返回时，开始执行恶意代码。

4.3 RFID 攻击

针对标签和阅读器的攻击方法：

软件技术

软件攻击使用 RFID 的通信接口，寻求安全协议、加密算法及其物理实现的弱点；

例如在软件的执行过程中插入窃听程序。利用潜藏的命令、不良参数与缓冲器溢出、文件存取、恶意进程、通信协议、加密协议等。逻辑攻击者利用这些缺陷诱骗卡泄露机密数据或允许非期望的数据修改。

窃听技术

采用高时域精度的方法，分析电源接口在微处理器正常工作过程中产生的各种电磁辐射的模拟特征；

用于实现物理攻击的手段和工具包括化学溶剂、蚀刻与着色材料、显微镜、亚微米探针台以及粒子束 FIB 等。

故障产生技术和旁路攻击

通过产生异常的应用环境条件，使处理器产生故障，从而获得额外的访问途径。

利用物理量来分析和更改 RFID 的行为。通过观察电路中的某些物理量，如能量消耗、电磁辐射、时间等的变化规律，来分析加密数据；或通过干扰电路中的某些物理量，如电压、电磁辐射、温度、光和 X 射线、频率等，来操纵 RFID 的行为。

RFID 详细物理攻击在 PPT 第五章第九页之后

4.4 侧信道攻击 SCA（PPT 中一部分翻译为旁路攻击）

说明：这里资料来源于 gjy 学长的整理，其中的 PPT 指的是去年 dyl 老师的一个 ppt 旁路攻击与防御（比较详细），在 th 老师的 PPT 第五章中介绍了侧信道攻击的方法，还包括了差分功耗分析（DPA），简单功耗分析（SPA），这里就不再给出。

在 dyl 老师的 ppt 中给出了一些 SCA 方法的定义，这里在后面给出。

旁路攻击SCA及防御

- 密码攻击
 - 数学攻击
 - 以**线性攻击**和**差分攻击**为代表的传统密码分析方法，利用加密算法的统计特性，通过分析选择的明密文对来获取密钥
 - 实施这类攻击通常需要俘获和处理异常巨大的数据量，在现实中并不总是可行
 - 实体攻击

通过去除芯片或电路的封装，观察、操纵和干预系统内部芯片的某些点的状态及芯片与外围电路之间的通讯来获得信息；或者通过改变电路结构，直接获取存储数据等

- 攻击方法
 - 微探针技术

攻击者通常在去除芯片封装之后，通过恢复芯片功能焊盘与外界的电气连接，最后可以使用微探针获取感兴趣的信号，从而分析出电路的有关设计信息和存储结构，甚至直接读取出存储器的信息进行分析

- 版图重构

利用高倍光学及射电显微镜研究电路的连接模式，可以迅速识别芯片上的一些基本结构，如数据线、地址线、控制线、RNG等。其结果是存储内容被攻击者非法获得，旁路某些敏感电路

- 实现攻击

加密算法的实现电路经常会泄露一些有用信息，这些信息能够被用来分析加密算法的敏感数据或密钥，这即是实现攻击

- 分类
 - 主动攻击(失效分析攻击)
 - 是一种侵入式攻击，通过**引入故意错误**如数据抖动、电源或时钟突变来影响加密电路的正常运算过程，从而分析出密钥
 - 已知干扰方式
 - 时钟信号短暂扰动；电源供应短暂扰动；外部电磁场短暂扰动；光、温度、射线等扰动
 - **时钟信号扰动**是目前最容易和最有效的干扰方式
 - 干扰攻击目的(PPT 旁路攻击SCA及防御 P11)
 - 干扰攻击机理(PPT 旁路攻击SCA及防御 P12)

- 被动式攻击(旁路攻击SCA)

它是指加密系统能够以各种形式泄漏信息，比如温度、声波、能量耗散、执行时间、电磁泄漏和光信号等，通过提取加密电路中的泄漏物理量，分析泄漏物理量变化规律来从而分析出密钥

- 旁道攻击不同于传统经典的、专注于数学理论而对密码系统进行研究的方式，而是一种针对密码系统实现上的物理攻击方式。但它既没有系统的攻击方式，也没有系统的解决方法
- 常用攻击方法
 - 利用电磁辐射的**电磁攻击方法**

电磁信号的噪声比能量信号的噪声小，保留了泄露的秘密密钥的模式

- 利用能量消耗的**能量攻击方法**

能量信号暴露了芯片在该位置的的电位差

电磁攻击和能量攻击这两种攻击方法可以互相补充，有时可以联合使用

- 旁路攻击是基于统计理论的物理攻击方法
- 旁路攻击的必要条件
 - 有足够多的采样样本
 - 各密钥相关状态的准确采样值
- 攻击可分为两个阶段
 - 泄露信息的采集阶段
 - 密钥分析阶段
- 旁路攻击分类
 - 时间攻击
 - 故障攻击
 - 功耗攻击
 - 电磁攻击
 - 声音攻击
 - 可见光攻击
 - 组合分析攻击

在诸多旁路攻击手段中，能量攻击和电磁攻击最为强大和高效

- 旁路攻击分析技术
 - 故障分析技术
 - 侵入分析技术
 - 时间分析技术
 - 简单功耗分析技术
 - 差分功耗分析技术
 - 电磁辐射分析技术
 - 高阶差分技术
 - 汉明差分技术
 - 模板分析技术

- 对旁路攻击的防御技术
 - 防御立足点就是破坏旁路功耗分析的两个必要条件
 - 消除密码算法实现中的信息泄露
 - 增加攻击的难度
 - 增加噪声
 - 减少有效信息量
 - 关于旁路信息泄露的几条公理
 - 计算并且只有计算才能产生泄漏信息
 - 相同的计算在不同的计算机上泄漏的信息不同
 - 信息泄漏与选择的测量方法密切相关
 - 所有的旁路泄漏信息都与计算机内部配置相关
 - 已有的抗攻击研究

已有的抗攻击研究主要可以分为算法级和电路级。现有的比较有效的方法是算法级掩码和电路级的双轨和掩码技术

。 电路级抗攻击

- 。 目前电路级的安全措施主要利用了3个技术
 - 。 互补电路
 - 。 电路级Masking
 - 。 预充电
- 。 利用噪声发生器产生随机的噪声

加入到可能会被攻击的信号端，以降低功耗分析的信噪比，从而使得功耗分析的攻击方法变得更加困难。但是该方法存在一些缺陷，攻击者可以通过更多的采样和信号处理等方法完成对硬件的攻击

。 异步电路

由于其在平均功耗、能量信号的峰值以及电磁辐射等方面的优势，使得它在抵御功耗分析方面也有作用，但由于芯片设计中异步电路本身运用的不是很广泛，相关的设计软件也不是很齐全，这为异步电路在安全芯片中的应用带来了不小的麻烦

。 双轨技术

通过加入一个互补电路，平衡了逻辑0 和逻辑1 的功耗，从而破坏了第一个必要条件；而掩码技术通过内部引入随机数，使得内部数据也具有随机特性，从而破坏了第二个必要条件

。 对旁路攻击的软件防御

- 。 使用软件方法应对硬件攻击的关键就是要看到
 - 。 任何硬件攻击都不能随意改变自身的运行方式，这也是"软件防御"思想的基本内涵
- 。 主要方法
 - 。 数据冗余

使用数据单元、多备份

为防止硬件攻击行为修改存储体上的静态数据和程序代码，可以在存储数据（及程序代码）的时候增加一定长度的冗余数据，用来保证数据的完整性。一种简单的方法就是在每组数据的后面增加循环冗余码。如果对于数据完整性有着更加严格的要求，可以采用一个简单的哈希函数计算出每组数据的杂凑值。通过数据冗余，还可以有效地防止主动攻击对关键数据的修改（比如安全位、指令下载控制位等），并且由于仅仅是对数据存储格式的变化，这种措施可以在不影响程序结构的情况下，加固安全性

- 控制冗余

前序执行顺序、入口检查
控制冗余的主要思想就是在包含有关键代码的函数中添加多重控制，以最大程度上保证程序执行是处于完全安全、可信的环境之中。比如，设置多个状态位，在程序的执行过程中不断检查这些状态位的状态，如果有一个发生了改变，程序马上退出执行

- 执行冗余

对密码算法中独立的（逻辑上没有继承关系、时间上没有先后关系）的代码按随机顺序执行
对同一个功能构造多个不同但是等价的实现，实现随机选择执行
随机延时和随机功耗插入
未用资源的随机介入
SLEEP躲避
数据单元的等价实现、同步及随机访问

- 特定加密算法的安全实现

- 软件防御技术关键是代码分块的粒度粗细和等价实现的数量

（推荐打印 dyl 老师的相关 ppt）

时间分析方法：

该方法通过对密码芯片运算过程中执行时间信息的采集，结合密码算法的内部实现，证实了算法指令和执行时间存在相关性，从而推测出密钥信息。

故障分析方法：

该方法利用密码 计算过程中的故障信息来破解密码系统。1997 年 Biham 和 Shamir 将这种攻击方法应用于对称密码体制，首次提出“差分故障攻击”的概念，并成功地攻击了 DES 算法。

功耗分析：

功耗攻击是一种利用密码设备运行时的功耗泄露信息来推测密码系统所进行的操作和秘密信息的攻击方法。功耗攻击已经被用于破解几乎所有的对称密码和公钥密码，Paul 声称仅需要数十条功耗泄漏曲线，就可以在几分钟内迅速地破解没有防御措施的大多数智能卡。

功耗分析比较重要，若要举例说明，可以举例汉明重量模型和能耗之间的线性关系（可以用于模版攻击，相关系数 CPA 攻击），如 AES 的每一个子密钥是 0-255 的 8 位密钥，0-255 是用 8 位二进制表示的，汉明重量为这八位数中 1 的个数，如 255 二进制为 11111111，汉明重量就是 8，0 汉明重量为 0，这样会有 9 个汉明重量，在模版攻击中 9 个汉明重量对应功耗 9 个模版。

电磁攻击：

出电磁泄露的信息非常大，其攻击方法比时间攻击和能量攻击更为有效，电磁攻击不仅可以与能量攻击相媲美，而且对于某些能量攻击无法破解的系统，电磁攻击也能够破解。

声音攻击：

从理论上说，主板上电容器发出的声音能提供一定的解密线索，其泄漏方式与电磁泄漏或者能量波动十分相似。Shamir 发现 PC 机运行时产生的声音信息量十分惊人，由于电容器发出的音频信号高于风扇产生的频率，所以很容易将其过滤出来并用作密码分析。

处理器运算时的声音泄露和所执行的操作之间存在相关性。例如，研究人员发现对于不同的密钥，RSA signature / decryption 的声音是不同的，通过观察音频信号则可以重现密码操作的时间序列，并可以为时间攻击 (Timing Attack) 提供有价值的信息。

可见光攻击：

Kuhn 的研究工作表明 CRT 显示器散射出的可见光可以用于重新构建 CRT 显示器上的信息。可见光攻击不同于时间攻击和功耗攻击，该攻击方法不需要和被测的物理设备进行连接，因此更容易被攻击者使用。Loughry 则将该攻击方法应用于 LED 状态显示器件的攻击。

组合分析攻击：

组合分析攻击：将功耗分析、电磁辐射分析、时间攻击、故障攻击等方法组合使用，组合攻击往往比单一途径的分析方法更加有效。

模版攻击：

模版攻击根据密码设备泄漏旁路信息的数据相关性和操作相关性进行攻击，首先为密钥空间中所有的密钥分别构建一个泄漏信息特征的模版，之后根据获取的一份或多份泄漏的信息寻找最匹配的模版，进而推断最可能的正确密钥或有效缩小密钥搜索空间

6. 嵌入式系统常见的工作模式（低功耗，休眠，睡眠），模式的特征，为什么需要这些模式

低功耗系统的一种软件架构是让系统始终处于睡眠模式，仅在运行单个中断服务程序(ISR)时唤醒，然后立即返回睡眠状态。

PPT 相关内容

CM3 低功耗模式需求：

表 9.1 典型的低功耗需求和相关考虑	
需求	典型的低功耗考虑
动态电流	通常以 $\mu\text{A}/\text{MHz}$ 衡量。动态电流主要是指存储器、外设以及处理器所需的动态功耗。为了简化计算，经常要假定微控制器的功耗和时钟频率直接成比例（严格说来是不对的）。另外，所使用的实际程序代码也可以影响结果
休眠模式电流	通常以 μA 衡量。这是因为最低功耗休眠模式下所有的时钟信号一般都会停止，其主要由晶体管的漏电流和一些模拟电路及 I/O 板组成。一般来说，在测量这类电流时多数外设都是关闭的。不过，在实际应用中，可能需要一些外设处于活动状态
能效效率	测量时一般会基于 Dhrystone ($\text{DMIPS}/\mu\text{W}$) 或 CoreMark ($\text{CoreMark}/\mu\text{W}$) 等常见的测试平台，不过，这些平台可能和自己应用中实际的数据处理行为不同
唤醒等待	通常以时钟周期数来衡量，有时则是以微秒为单位。一般来说，它是硬件请求（如外设中断）到处理器继续执行程序的时间。若单位为微秒，则时钟频率会直接影响结果。在一些设计中，可以使用非常低功耗的休眠模式，不过唤醒所需的时间可能会更长（例如，PLL 等时钟回路可能会被关闭，继续正常的时钟输出则会花费较长的时间）。产品设计人员需要确定应用中要使用哪种休眠模式

CM3 低功耗模式的优势：

CM3的系统控制

低功耗：

- 减小电池体积
- 增强安全性
- 电磁干扰更小，提高无线通信质量
- 电源设计更简单，无需考虑散热问题
- 可以考虑系统通过其他能源供电

.....

网上资料

STM32F1 低功耗模式

STM32 本身就支持低功耗模式，共有三种低功耗模式：

- 睡眠(Sleep)模式。
- 停止(Stop)模式。
- 待机(Standby)模式。

模式名称	进入	唤醒	对 1.2V 域时钟的影响	对 VDD 域时钟的影响	调压器
睡眠(立即休眠或退出休眠)	WFI	任意中断	CPU CLK 关闭对其它时钟或模拟时钟源无影响	无	开启
	WFE	唤醒事件			
停止	PDDS 和 LPDS 位 + SLEEPDEEP 位 + WFI 或 WFE	任意 EXTI 线(在 EXTI 寄存器中配置, 内部线和外部线)	所有 1.2 V 域时钟都关闭	HSI 和 HSE 振荡器关闭	开启或处于低功耗模式
待机	PDDS 位 + SLEEPDEEP 位 + WFI 或 WFE	WKUP 引脚上升沿、RTC 闹钟(闹钟 A 或闹钟 B)RTC 唤醒事件、RTC 入侵事件、RTC 时间戳事件、NRST 引脚外部复位、IWDG 复位	所有 1.2 V 域时钟都关闭	HSI 和 HSE 振荡器关闭	关闭

CSDN @木士易

(退出时休眠这一特性的作用是, 当 ISR 完成时, 使能该特性会使 MCU 立即进入睡眠状态, 而不是让处理器每次都进入和退出 ISR, 浪费完堆栈的开销。通过这样做, MCU 仍然被配置为运行 ISR, 因此当它再次触发时, 系统唤醒并立即执行 ISR, 开销最小。)

睡眠模式:

描述: 在休眠模式下, 系统会关闭大部分的功能模块和外设, 以降低功耗。系统处于较低的功耗状态, 只保持必要的基本功能, 例如时钟和少量的输入/输出功能。休眠模式下的系统可以很快地被唤醒, 以便响应外部事件或命令。

进入睡眠模式：进入睡眠模式有两种指令：WFI(等待中断)和 WFE(等待事件)。根据 Cortex-M 内核的 SCR(系统控制)寄存器可以选择使用立即休眠还是退出时休眠。当 SCR 寄存器的 SLEEPONEXIT(bit1)位为 0 的时候使用立即休眠，当为 1 的时候使用退出时休眠。

退出睡眠模式：如果使用 WFI 指令进入休眠模式的话那么任意一个中断都会将 MCU 从休眠模式中唤醒，如果使用 WFE 指令进入休眠模式的话那么当有事件发生的话就会退出休眠模式，比如配置一个 EXIT 线作为事件。

当 STM32F103 处于休眠模式的时候 Cortex-M3 内核停止运行，但是其他外设运行正常，比如 NVIC、SRAM 等。休眠模式的功耗比其他两个高，但是休眠模式没有唤醒延时，应用程序可以立即运行。

停止模式：

描述：当停机模式设置完毕后，CPU 一旦遇到 WFI 或 WFE 指令就会停止工作，HSI 和 HSE 也进入关闭状态。但 Flash 和 SRAM 将会继续保持电源供应，所以此时 STM32 的所有工作状态仍然是保留着的。

停止模式基于 Cortex-M3 的深度休眠模式与外设时钟门控，在此模式下 1.2V 域的所有时钟都会停止，PLL、HSI 和 HSE RC 振荡器会被禁止，但是内部 SRAM 的数据会被保留。调压器可以工作在正常模式，也可配置为低功耗模式。如果有必要的话可以通过将 PWR_CR 寄存器的 FPDS 位置 1 来使 Flash 在停止模式的时候进入掉电状态，当 Flash 处于掉电状态的时候 MCU 从停止模式唤醒以后需要更多的启动延时。停止模式的进入和退出如表所示：

停止模式	描述
进入模式	在以下条件下执行 WFI(等待中断)或 WFE(等待事件)指令： — 设置 Cortex-M3 系统控制寄存器中的 SLEEPDEEP 位。 — 清除电源控制寄存器(PWR_CR)中的 PDDS 位。 — 通过设置 PWR_CR 中 LPDS 位选择电压调节器的模式 注：为了进入停止模式，所有的外部中断的请求位(挂起寄存器(EXTI_PR))和 RTC 的闹钟标志必须被清除，否则停止模式的进入流程将会被跳过，程序继续运行。
退出模式	如果执行 WFI 进入停止模式： 设置任一外部中断线为中断模式(在 NVIC 中必须使能相应的外部中断向量)。 如果执行 WFE 进入停止模式： 设置任一外部中断线为事件模式。

CSDN @木士易

待机模式：

描述：当进入待机模式后，所有的 SRAM 数据、Cortex-M3 处理器的寄存器和 STM32 的寄存器内容都将会被清零。效果等同于硬件复位。

相比于前面两种低功耗模式，待机模式的功耗最低。待机模式是基于 Cortex-M3 的深度睡眠模式的，其中调压器被禁止。1.2V 域断电，PLL、HSI 振荡器和 HSE 振荡器也被关闭。除了备份区域和待机电路相关的寄存器外，SRAM 和其他寄存器的内容都将丢失。待机模式的进入和退出如表所示：

待机模式	描述
进入模式	在以下条件下执行 WFI(等待中断)或 WFE(等待事件)指令： — 设置 Cortex™-M3 系统控制寄存器中的 SLEEPDEEP 位。 — 设置电源控制寄存器(PWR_CR)中的 PDDS 位。 — 清除电源控制/状态寄存器(PWR_CSR)中的 WUF 位。
退出模式	WKUP 引脚的上升沿、RTC 闹钟事件的上升沿、NRST 引脚上外部复位、IWDG 复位。 CSDN @木士易

退出待机模式的话会导致 STM32F1 重启，所以待机模式的唤醒延时也是最大的。实际应用中要根据使用环境和要求选择合适的待机模式。

为什么嵌入式需要低功耗？（GPT 答案）

- 延长电池寿命：许多嵌入式系统依赖电池供电，例如便携设备、无线传感器节点等。通过降低功耗，可以延长电池的使用时间，减少频繁充电或更换电池的频率，提供更长的续航能力。
- 节约能源：对于使用外部电源的嵌入式系统，低功耗意味着能耗更低，可以减少能源消耗和运行成本。这对于大规模部署的嵌入式系统（如智能家居、工业自动化等）尤为重要。
- 热管理：功耗较高的嵌入式系统会产生较多的热量，这可能导致温度升高，影响系统的可靠性和性能。通过降低功耗，可以减少热量的产生，降低温度，提高系统的稳定性和可靠性。
- 环境要求：在一些特殊环境下，例如嵌入式系统用于无人机、卫星、远程传感器等场景，能源供应可能有限，或者无法进行频繁的维护和充电。在这些情况下，低功耗是必要的，以确保系统能够长时间稳定运行。

- 系统集成和散热设计：低功耗可以降低系统的整体功耗，减少散热要求。这对于嵌入式系统的集成和散热设计来说是有益的，可以简化硬件设计和系统布局，减小尺寸和成本。

7、嵌入式系统攻击分类

01 根据攻击对象分类

利用嵌入式自身的安全缺陷，或者利用其外部不受信任的环境，嵌入式系统面临诸多攻击威胁，根据攻击对象不同，可将其分为**隐私数据攻击**、**可用性攻击**、**代码完整性攻击**等。

- **隐私数据攻击** 这种攻击的目的是获取嵌入式系统内存储、传递或操作的敏感信息数据；防范这类攻击的主要手段是对敏感信息数据进行加密保护，但实现加密保护需要密钥，密钥的创建、存储、使用和销毁等，需要引入能够信任的密钥管理机制以保障其安全性。此外，还可通过访问控制对敏感信息数据进行保护。
- **可用性攻击** 这类攻击挪用系统资源，扰乱系统的正常工作，使系统不能执行相应正常操作。可以通过在嵌入式系统中添加可靠的资源分配管理组件来防范这类攻击。
- **代码完整性攻击** 这种攻击的目的是获取嵌入式这种攻击试图修改嵌入式系统相关数据或代码。防范这类攻击的重点是保证嵌入式系统自身代码的完整性，可以在运行前通过对嵌入式系统相关代码进行安全度量，检测代码是否被篡改。

02 根据发起攻击的代理工具或手段分类

- **软件攻击** 如病毒，木马，蠕虫等通过软件代理对终端系统结构的薄弱环节发起的攻击。这类攻击是耗费代价较小，是较为常见的一种攻击。
- **硬件攻击** 如硬件分解、电磁干扰、使用探针对嵌入式芯片内部的交互信息进行窃听等。这类攻击需要较为昂贵的基础设施要求，较难实现。
- **缓存溢出攻击** 向缓存中传送超出容量的数据，并伴以一段恶意执行代码及用来覆盖调用程序返回地址的地址数据，造成缓存溢出。当功能返回时，开始执行恶意代码。

02 提高输入输出的可靠性

在嵌入式系统中，可以采用多种软件方法来提高输入输出的可靠性。包括：

玉龙真言：不需这么复杂，看复习 PPT，两句话就够了

1. 数据冗余：通过数据冗余技术可以提高输入输出的可靠性。数据冗余可以通过存储多个数据副本来实现。当一个数据副本出现错误时，可以使用其他副本来恢复正确的数据。这可以应用于输入数据的冗余存储和输出数据的冗余备份。同时，可以结合使用校验和、循环冗余校验（CRC）等技术来检测和纠正数据或备份中的错误
2. 控制冗余：控制冗余是指在输入输出过程中引入冗余控制以提高可靠性。前序检查

是一种常见的控制冗余方法，即在执行任何关键操作之前，先进行输入数据的检查和验证。入口检查则是在系统接收输入之前对输入进行检查，确保输入满足预期条件。

3. 执行冗余：执行冗余指的是通过多种方式执行相同的操作来增加可靠性。随机顺序执行是一种执行冗余方法，即通过以不同的顺序执行相同的操作来防御传统侧信道分析。等价实现是指在软件层面实现相同功能的多个实现，通过对比结果来检测和纠正错误。随机延时和功耗是一种通过引入随机延迟来消除电源噪声等问题的执行冗余方法。这些执行冗余方法也可以在一定程度上抵御侧信道分析。
4. 特定算法：使用特定算法可以提高输入输出的可靠性。例如，冗余校验算法如海明码、纠错码等可以用于检测和纠正数据传输中的错误。其他算法如决策树、状态机等也可以用于输入数据的验证和处理。
5. 软件陷阱：软件陷阱就是一条引导指令，强行将乱飞的程序引向一个指定的地址，在那里有一段专门对程序出错进行处理的程序。通过执行某个指令进入特定的程序处理模块，相当于由外部中断响应。一般软件陷阱有现场保护功能。软件陷阱用于抗干扰时，首先检查是否是干扰引起的，并判断造成影响的程度，如不能恢复，则强制进入复位状态，如干扰已撤消，则可立即恢复执行原来的程序。
6. SLEEP 躲避：在系统没必要响应或用户察觉不到的操作间隙，可以插入睡眠操作来躲避电磁等干扰。
7. 定期自检：在系统中引入自检机制，定期检查输入输出设备的状态和正确性。这可以通过执行自检程序或使用内建的自检功能来实现。如果发现任何问题，系统可以采取相应的措施，如重新初始化设备或发出警告。
8. 输入输出缓冲区管理：有效地管理输入输出缓冲区可以提高可靠性。使用适当的缓冲区大小和缓冲区溢出检测机制来避免数据丢失或损坏。
9. 安全性措施：实施适当的安全性措施，如数据加密、访问控制和身份验证，以保护输入输出数据的完整性和机密性。
10. 超时机制：在进行输入输出操作时，实施合适的超时机制。同时，也可以在程序运行过程中引入超时机制，如看门狗等。如果在预定时间内未能完成输入输出操作，或未能定时喂狗，系统可以采取相应的措施，如重新尝试、报告错误或回滚操作。

软件防御技术关键是代码分块的粒度粗细和等价实现的数量,这句话如何理解?

ChatGPT

这句话可以理解为在软件防御技术中，关键因素包括代码的分块方式（粒度粗细）以及等价实现的数量。

首先，“代码分块的粒度粗细”指的是将软件的代码按照一定的规则 and 标准进行划分，形成不同的代码块或模块。这里的粒度指的是代码块的大小或复杂度，粗细表示代码块的粒度大小不同。粒度粗的代码块包含更多的代码逻辑，而粒度细的代码块则将代码细分为更小的单元。选择适当的代码分块粒度可以影响软件防御技术的效果。

其次, "等价实现的数量"是指在软件开发过程中, 为了实现相同的功能或逻辑, 可能存在多种不同的代码实现方式。这些不同的实现方式可以称为等价实现。等价实现的数量表示了软件中存在多少种不同的代码编写方式或算法选择。这些等价实现可能在效率、安全性或易维护性等方面存在差异。

因此, 这句话的含义是在软件防御技术中, 重要的因素之一是如何选择代码分块的粒度大小, 以及需要考虑到等价实现的数量。通过合理选择代码分块的粒度和控制等价实现的数量, 可以对软件进行更有效的防御。

几种安全增强方法:

软件增强

- **使用嵌入式安全操作系统** 安全操作系统通过参照监视器监视系统的运行, 防止违反安全策略的动作产生, 当前嵌入式安全操作系统一般都具备身份认证、自主访问控制和安全审计等功能。第四代防火墙的设计以这种安全操作系统为基础。

硬件增强

- **增添加密运算模块** 使用应用程序特定的集成电路 (ASIC) 在硬件上实现给定的加密算法, 只需要很少的成本且可批量生产。
- **添加专用安全存储模块** 参考 PC 端可信计算思想, 在嵌入式系统中添加一个专用的安全硬件模块, 并将敏感数据保护在一个安全设计十分牢靠的物理设备中, 可采用以下几种先进技术: 防篡改技术、物理安全技术、硅工艺技术

架构设计增强

- **引入 TrustZone 架构** 从体系架构角度出发, 引入 Trustzone 架构增强嵌入式系统安全。TrustZone 提供硬件隔离, 在尽量不影响原有处理器设计的情况下保护安全内存、加密块、键盘和显示器等外设。

