STM32微控制器期末考试与学习全攻略秘籍 🎓

本学习指南根据您提供的课件（共9章）提炼而成，系统化梳理了所有核心知识点，并配有**期末考试必考点解析、原理详解、核心公式计算步骤和经典代码片段**。旨在帮助同学们快速建立知识脉络，冲刺高分！

📅 目录导航

第1章 ARM Cortex-M3嵌入式系统概述

1. 核心概念对比

- **微处理器 (MPU)**：仅包含中央处理器(CPU)的芯片，如Intel Core系列或ARM Cortex-A系列。需要外接ROM、RAM和接口芯片才能组成系统。
- **微控制器 (MCU/单片机)**：将**CPU、RAM、ROM、I/O接口、定时器**等资源集成到单一芯片上的微型计算机。STM32是典型的32位MCU。
- **嵌入式系统定义**：以**应用为中心**，以**计算机技术为基础**，**软硬件可裁剪**，适应应用系统对功能、可靠性、安全性、成本、体积、功耗、环境等有严格要求的**专用计算机系统**。

2. 嵌入式系统的特点

1. **专用性强**：与应用紧密绑定，软硬件可裁剪。
2. **实时性好**：能对外部事件作出足够快的响应。
3. **可靠性高**：具备看门狗、内存保护等容错机制。
4. **低功耗**：毫瓦级到瓦级。
5. **多样性**：处理器架构百家争鸣（ARM, RISC-V, MIPS等）。

> **Note**

实时系统分类：

- **硬实时系统**：必须在严格规定的时间内完成处理，否则会造成灾难性后果（如航天、车载安全系统）。
- **软（弱）实时系统**：偶尔违反时间限制不会对系统造成严重破坏（如视频播放、信息采集）。

3. ARM 商业模式与 Cortex 产品线

- **ARM模式**：自己不制造芯片，只授权IP核（设计方案）给各大半导体公司（如ST、NXP、TI），由它们添加外设后制造最终芯片。
- **Cortex三大系列 (A/R/M)**：
  - **Cortex-A (Application)**：面向性能级应用，支持富操作系统（如Android, Linux），用于智能手机、边缘计算。
  - **Cortex-R (Real-time)**：面向高实时性、高可靠性场景，用于汽车制动、硬盘控制器。
  - **Cortex-M (Microcontroller)**：面向微控制器领域，低功耗、低成本，替代传统8/16位单片机。

4. 嵌入式软件结构与常见RTOS

- **无操作系统（裸机 Bare-metal）**：
  - **引导程序**：汇编语言编写，上电运行，负责时钟、外设配置等初始化。
  - **应用程序**：C语言编写，直接架构在硬件上，通常在 `main()` 中无限循环。
- **典型嵌入式实时操作系统 (RTOS)**：
  - **FreeRTOS**：轻量级、开源、生态极度完善。
  - **RT-Thread**：中国主导开源，物联网友好，组件极其丰富。
  - **μC/OS-III**：可裁剪、抢占式、商业级RTOS，多用于工业和医疗。

第2章 STM32应用基础（🌟考试重中之重）

1. 冯·诺依曼结构 vs 哈佛结构

特性	冯·诺依曼结构	哈佛结构
存储空间	指令和数据统一编址，放在同一个存储器中。	指令和数据独立编址，分别放在两个存储器中。
总线结构	指令和数据共享同一套系统总线。	指线和数据线独立（分开的数据总线与指令总线）。

特性	冯·诺依曼结构	哈佛结构
执行效率	不能同时取指 and 取数，存在总线瓶颈。	可同时取指 and 取数，有利于流水线设计。
典型代表	传统PC、早期8051。	ARM Cortex-M3内核。

2. 大端模式与小端模式

计算机以字节为单位编址。对于多字节数据（如32位字 `0x12345678` 存放在 `0x8000` 开始的地址）：

- **小端模式 (Little-Endian)（STM32默认）：低字节存放在低地址，高字节存放在高地址。**
  - `0x8000` = `0x78` (LSB)
  - `0x8001` = `0x56`
  - `0x8002` = `0x34`
  - `0x8003` = `0x12` (MSB)
- **大端模式 (Big-Endian)：高字节存放在低地址，低字节存放在高地址。**
  - `0x8000` = `0x12` (MSB)
  - `0x8003` = `0x78` (LSB)

3. CISC 与 RISC 对比

- **CISC（复杂指令集）**：指令丰富且复杂，长度不固定，执行需要多个时钟周期。旨在减少编译器负担，代表：x86。
- **RISC（精简指令集）**：指令简单且长度固定（如32位），绝大多数单周期执行，依赖编译器优化，寄存器较多。代表：ARM。

4. 流水线技术与吞吐率计算（📝必考计算题）

Cortex-M3 采用三级流水线：**取指 (Fetch) -> 译码 (Decode) -> 执行 (Execute)**。

- **完成 n 条指令所需时间**：

$$\text{总时间} = \text{完成第一条指令所需时间} + (n-1) \times \text{时间最长的流水线阶段时间}$$

- **吞吐率 (Throughput)**：单位时间内完成的指令数。

$$\text{吞吐率} = \frac{\text{指令条数 } n}{\text{总时间}}$$

> **Tip**
>
> **期末计算真题演练：**
> 已知某三级指令流水线：取指周期 = 1ns，译码周期 = 3ns，执行周期 = 2ns。求完成 100 条指令的吞吐率。

解答步骤：

1. 第一条指令完成时间 = $1 + 3 + 2 = 6\text{ns}$（此时流水线填满）。
2. 从第二条指令开始，瓶颈在最长阶段（译码阶段，3ns），即每过 3ns 产出一条指令。
3. 总时间 = $6\text{ns} + (100 - 1) \times 3\text{ns} = 6 + 297 = 303\text{ns}$。
4. 吞吐率 = $\frac{100}{303\text{ns}} \approx 0.33$ 条指令/ns (或 3.3×10^8 条指令/秒)。

5. Cortex-M3 内核核心组件

- **NVIC（嵌套向量中断控制器）**：处理中断嵌套，低延迟。
- **SysTick（系统嘀嗒定时器）**：**24位向下递减计数器**，用作OS心跳或精准延时。
- **MPU（存储保护单元）**：保护关键内存区，防止应用越权或跑飞。
- **总线矩阵**：连接内核总线（I-bus, D-bus, S-bus）与DMA总线，进行并行仲裁。

6. STM32 总线与外设映射

STM32F103 的主控单元包括：**I-bus** (指令总线), **D-bus** (数据总线), **S-bus** (系统总线), **DMA总线**。被控单元包括：内部SRAM、内部Flash、AHB到APB桥。

- **APB1总线（低速，最高36MHz）**：挂载 TIM2~7, USART2~5, SPI2~3, I2C1~2, CAN, USB, DAC, PWR, BKP。
- **APB2总线（高速，最高72MHz）**：挂载 GPIOA~G, AFIO, EXTI, ADC1~3, TIM1/8, SPI1, USART1。
- **AHB总线**：挂载 DMA1/2, RCC, SDIO, FSMC, Flash 接口。

7. 电源管理与电压监视

- **供电分区**：
  1. **数字电源 (VDD/VSS)**：通常为 3.3V，给I/O及内核调压器供电。
  2. **模拟电源 (VDDA/VSSA)**：供电给ADC、复位电路、振荡器。
  3. **参考电压 (VREF+/VREF-)**：ADC的参考基准，必须介于 $2.4\text{V} \sim V_{DDA}$。
  4. **后备电源 (VBAT)**：当主电源VDD掉电时，自动切至VBAT给 **RTC**（实时时钟）和 **后备寄存器 (BKP)** 供电。
- **PVD（可编程电压监测器）**：监测VDD电压，当低于或高于预设阈值时产生中断，进行安全处理。
- **复位类型**：系统复位（不复位备份域和时钟CSR标志）、电源复位（复位除备份寄存器外所有）、备份域复位。

8. 低功耗模式对比

模式	调压器状态	CPU状态	时钟状态	SRAM/寄存器内容	唤醒源	唤醒后时钟
睡眠模式 (Sleep)	开启	停止	运行	保留，外设继续工作	任意中断/事件	恢复前状态
停止模式 (Stop)	低功耗模式	停止	关断 (PLL/HSE/HSI全停)	寄存器与SRAM保留	任意 **EXTI中断/事件**	**HSI (不稳定的8MHz)**
待机模式 (Standby)	关断	停止	关断	**全部丢失** (后备寄存器保留)	WKUP引脚/RTC/NRST/IWDG	相当于**软件复位**

9. 启动配置 (BOOT[1:0])

在系统复位后，SYSCLK第4个上升沿锁存 BOOT 引脚的状态：

- **BOOT0 = 0，BOOT1 = X**：从**主Flash存储器**启动（起始地址 `0x08000000` 重映射到 `0x00000000`）。
- **BOOT0 = 1，BOOT1 = 0**：从**系统存储器**启动（内置ISP串口下载 Bootloader，地址 `0x1FFFF000`）。
- **BOOT0 = 1，BOOT1 = 1**：从**内置SRAM**启动（多用于仿真调试，地址 `0x20000000`）。

> **Important**
>
> **Cortex-M3复位启动流程**：
>
> 1. 首先从 `0x00000000` 处取出 **MSP** (主堆栈指针) 的初始值。
> 2. 然后从 `0x00000004` 处取出 **PC** 的初始值（即复位向量地址）。
> 3. **注意**：由于内核工作在 Thumb 状态下，向量表中的地址其**最低有效位 (LSB) 必须为 1**（即地址为奇数）。
> 4. 执行复位中断服务程序 `Reset_Handler`，在汇编内调用 `SystemInit()` 配置时钟，最后跳转至C语言 `main()`。

10. 存储器映射与位带 (Bit-Banding) 操作（📝计算高频考点）

- **寻址空间**：32位CPU最大寻址空间为 4GB（统一编址），共分为8个块，每个块 512MB。
- **位带操作概念**：将位带区中的每一个位 (bit) 映射到别名区的一个字 (32-bit Word)。通过读写别名区地址的字，实现对位带区单bit的原子（不被打断）操作。
- **公式（必须背熟）**：

- **SRAM位带区**：范围 `0x20000000 ~ 0x200FFFFF` (1MB)，别名区起始地址：`0x22000000`。

$$\text{别名地址} = 0x22000000 + (\text{位带区地址} - 0x20000000) \times 32 + \text{位序号 } n \times 4$$

- **外设位带区**：范围 `0x40000000 ~ 0x400FFFFF` (1MB)，别名区起始地址：`0x42000000`。

$$\text{别名地址} = 0x42000000 + (\text{位带区地址} - 0x40000000) \times 32 + \text{位序号 } n \times 4$$

第3章 Keil MDK软件与工程模板创建

1. CMSIS 标准架构

- **定义**：Cortex Microcontroller Software Interface Standard (Cortex微控制器软件接口标准)。由ARM定义，旨在统一不同MCU厂商的底层接口，提高代码的可移植性。
- **分层**：**用户应用层 -> 标准外设库(HAL) -> CMSIS-Core -> 硬件寄存器**。

2. 标准外设库重要文件汇总

- `system_stm32f10x.c/h`： 系统时钟配置文件，包含系统级时钟初始化函数 `SystemInit()`，上电后自动将系统时钟倍频到最高主频（如 72MHz）。
- `stm32f10x.h`： 核心器件头文件，定义了所有外设结构体、寄存器基地址及中断向量。包含 `__IO` 的宏定义（即 `volatile`，防止编译器将硬件寄存器优化掉）。
- `startup_stm32f10x_hd.s`： 大容量产品（如 STM32F103ZET6）的汇编启动代码。
- `misc.c/h`： 内核NVIC（中断通道管理）的底层驱动。

3. Keil 工程配置必踩坑 (Define 与 Include)

- **Preprocessor Symbols Define**：必须添加 `USE_STDPERIPH_DRIVER, STM32F10X_HD`。
  - `USE_STDPERIPH_DRIVER`： 表示使用标准外设库驱动，启用 `stm32f10x_conf.h` 头文件。
  - `STM32F10X_HD`： 说明芯片是大容量器件 (High-Density)，编译对应的寄存器宏定义。这两个宏之间**必须用逗号 `,` 隔开**，错用句号 `.` 会导致编译报 `#992: invalid macro definition` 错误。

第4章 通用目的输入输出口（GPIO）

1. GPIO 配置寄存器

每个端口有 16 个引脚（如 PA0~PA15），由以下 7 个寄存器控制：

- `GPIOx_CRL`：端口配置低寄存器（控制引脚 Pin 0~7），4位控制一个引脚。
- `GPIOx_CRH`：端口配置高寄存器（控制引脚 Pin 8~15），4位控制一个引脚。
- `GPIOx_IDR`：端口输入数据寄存器（只读，16位对应16个引脚当前状态）。
- `GPIOx_ODR`：端口输出数据寄存器（可读写，配置输出电平，也可开启上/下拉）。
- `GPIOx_BSRR`：端口位设置/清除寄存器（32位。低16位写1置位，高16位写1复位，**原子操作**，推荐！）。
- `GPIOx_BRR`：端口位清除寄存器（16位，写1复位引脚）。
- `GPIOx_LCKR`：端口配置锁定寄存器（锁定当前IO配置，直至复位）。

2. GPIO 的 8 种工作模式（🌟必考填空题）

每个引脚共有4种输入模式和4种输出模式：

1. **输入浮空 (Input Floating)**：无上拉/下拉电阻。引脚电平完全由外部输入决定，常用于 USART RX、I2C 接收。
2. **输入上拉 (Input Pull-up)**：引脚内部接弱上拉电阻到 VDD。无外部输入时，引脚保持高电平。
3. **输入下拉 (Input Pull-down)**：引脚内部接弱下拉电阻到 VSS。无外部输入时，引脚保持低电平。
4. **模拟输入 (Analog Input)**：切断数字输入通道（禁用施密特触发器）。外部电平直接输入到内部ADC，功耗最低。
5. **推挽输出 (Push-Pull)**：双MOS管工作。输出高电平时PMOS导通，输出低电平时NMOS导通。**驱动能力强，输出电流大，速度快**。
6. **开漏输出 (Open-Drain)**：仅NMOS工作。输出低电平时导通拉低，输出高电平时输出为高阻态。**需要外部上拉电阻**才能输出高电平，适用于多设备线与、电平转换（如I2C总线）。
7. **推挽复用输出 (AF Push-Pull)**：引脚输出受片内外设（如SPI的MOSI，USART的TX）控制，而非 ODR。
8. **开漏复用输出 (AF Open-Drain)**：引脚复用功能且以开漏形式输出（如 I2C 的 SCL/SDA）。

3. GPIO 输出速度与重映射

- **引脚输出速度 (Slew Rate)**：分为 `2MHz`、`10MHz`、`50MHz`。它是驱动电路的上升/下降沿跳变速率，非信号频率。建议设置为信号最高速率 of 5~10倍。例如：LED/USART 设为 2MHz 即可，I2C 设为 10MHz，SPI/FSMC 设为 50MHz。
- **复用重映射 (Remap)**：为方便硬件布线，将某外设的功能引脚从默认位置转移到备用引脚（如将 USART1 的 TX/RX 从 PA9/PA10 重映射到 PB6/PB7）。

4. 经典 GPIO 库函数初始化代码

```
// 1. 使能 GPIO 外设时钟（GPIO挂载在 APB2 总线上）
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

// 2. 定义 GPIO 初始化结构体
```

```
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;              // 选择第 5 号引脚
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;       // 推挽输出
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;      // 50MHz速度

// 3. 调用库函数初始化
GPIO_Init(GPIOB, &GPIO_InitStructure);
```

第5章 中断系统与基本应用（EXTI）

1. 中断的基本原理与特点

- **中断机制优势**：解决CPU高效的运算能力与外设慢速的数据交互之间的矛盾。保证实时处理和紧急异常处理。
- **中断服务函数 (ISR)**：
  - **无参数，无返回值**，由硬件中断源触发自动调用，程序员不能主动调用。
  - 必须**短小精悍**，避免在ISR中进行复杂的延时或大量计算。
  - 在ISR中修改并在主程序中访问的变量，声明时**必须加 `volatile` 修饰符**（告知编译器每次直接读取内存地址，防止因优化将寄存器值放入缓存）。

2. NVIC 优先级划分与嵌套规则（🌟必考计算题）

Cortex-M3 使用4个二进制位来配置中断优先级，共划分为：**抢占优先级 (Preemption Priority)** 和 **响应（子/副）优先级 (Subpriority)**。

- **抢占优先级**：高抢占优先级的中断可以打断正在执行的低抢占优先级中断，形成**中断嵌套**。
- **响应优先级**：若抢占优先级相同，两中断同时到来时，高响应优先级的中断先被响应，但**不能打断**正在执行的同抢占优先级中断（不形成嵌套）。
- **硬件默认优先级**：若抢占优先级和响应优先级均相同，则按**中断向量表中的位置（中断号越小，优先级越高）**判定。

优先级分组 (SCB->AIRCR:PRIGROUP)

通过 `NVIC_PriorityGroupConfig()` 划分为5个组：

分组级别	抢占优先级位数 (值范围)	响应优先级位数 (值范围)
Group 0	0 位 (无抢占)	4 位 (0 ~ 15)
Group 1	1 位 (0 ~ 1)	3 位 (0 ~ 7)
Group 2	2 位 (0 ~ 3)	2 位 (0 ~ 3)
Group 3	3 位 (0 ~ 7)	1 位 (0 ~ 1)
Group 4	4 位 (0 ~ 15)	0 位 (无子优先级)

> ⚠ **Warning**
>
> 在整个系统设计中，**优先级分组只能配置一次**，一般在 `main()` 开头初始化，各外设中断在此分组框架内分配具体优先级。

3. EXTI（外部中断/事件控制器）

- **通道映射**：STM32 的 EXTI 共 19 个通道。
  - `EXTI0 ~ EXTI15` 映射到 I/O 引脚（Pin 0~15）。所有 Port 的相同 Pin 共用一个中断线（例如 PA0, PB0... PG0 都连接到 EXTI0）。使用时需要借助 **AFIO**（复用功能IO寄存器）进行具体通道选择配置。
  - `EXTI16` 连接 PVD。`EXTI17` 连接 RTC 闹钟。`EXTI18` 连接 USB 唤醒。

> 🖉 **Note**
>
> **中断 vs 事件**：
>
> - **中断通道**：检测到边沿后向 NVIC 发送信号，CPU 挂起当前任务，转去执行 ISR（软件参与，有延迟）。
> - **事件通道**：检测到边沿后，直接通过硬件连线向其他外设（如 DMA, ADC）发送触发脉冲，**无需软件介入，硬件自动联动，零延迟，大大降低CPU负担**。

4. 外部中断 EXTI 初始化配置步骤

1. 开启 GPIO 时钟和 **AFIO** 时钟（非常重要！映射中断线必须开 AFIO 时钟）。
2. 配置 GPIO 为输入模式（上拉或浮空）。
3. 通过 `GPIO_EXTILineConfig()` 建立 GPIO 引脚与 EXTI 中断线的映射。
4. 初始化 EXTI：指定中断线、触发方式（上升沿、下降沿或双边沿）并使能。
5. 初始化 NVIC：分配抢占/响应优先级，并使能对应中断通道。
6. 编写中断服务函数（在 `stm32f10x_it.c` 中）：**检查中断挂起标志位，处理应用，退出前清除标志位**。

```
// 典型 EXTI 中断服务程序
void EXTI0_IRQHandler(void)
{
    if(EXTI_GetITStatus(EXTI_Line0) != RESET) // 1. 确认中断来源
    {
        // 2. 执行用户处理代码
        LED_Toggle();

        // 3. 清除 EXTI0 中断挂起标志（不清除会导致重复进入中断死机）
        EXTI_ClearITPendingBit(EXTI_Line0);
```

```
    }
}
```

第6章 定时器与脉冲宽度调制（PWM）及输入捕获

1. 定时器分类与对比

STM32F103 最多支持 8 个定时器，分为三类，功能互为子集关系：
类型	定时器通道	计数器位数	计数模式	核心功能	挂载总线
基本定时器	TIM6, TIM7	16 位	仅向上计数	无通道，无I/O脚。主要用于精准定时和驱动DAC触发。	APB1
通用定时器	TIM2, TIM3, TIM4, TIM5	16 位	向上、向下、中央对齐	4个独立通道。支持输入捕获、输出比较、PWM输出、单脉冲。	APB1
高级定时器	TIM1, TIM8	16 位	向上、向下、中央对齐	4个独立通道。除通用功能外，支持**三相 complementary PWM 输出（带死区时间控制）**、刹车输入（Break）、重复计数器。	APB2

2. 时基单元与溢出时间计算（📝必考计算题）

时基单元包括：计数器寄存器（`TIMx_CNT`）、预分频器寄存器（`TIMx_PSC`）、自动重装载寄存器（`TIMx_ARR`）。

- **定时器溢出时间 T_{out} 计算公式**：

$$T_{out} = \frac{(\mathrm{ARR}+1)\times(\mathrm{PSC}+1)}{T_{clk}}$$

  - T_{clk}：定时器的输入时钟源频率（若挂在APB1上，系统时钟72MHz倍频后也是72MHz）。
  - 因为预分频和自动重装载计数都是从0开始计数，所以在公式中对应的寄存器值都要**加 1**。

> **Tip**
>
> **期慢计算真题演练**：
> 输入时钟 $T_{clk} = 72\mathrm{MHz}$，需要产生一个精度为 $1\mathrm{ms}$ 的定时中断。请给出 ARR 和 PSC 的配置值。
>
> **解答步骤**：
>
> 1. 根据公式：$T_{out} = \frac{(ARR+1)\times(PSC+1)}{72{,}000{,}000} = 0.001\mathrm{s}$。
> 2. 两数之积为：$(ARR+1)\times(PSC+1) = 72{,}000$。

3. 为了方便计算，我们常令预分频器将时钟分频为 10kHz（即计数一次为 $100\mu\text{s}$）或 1MHz（计数一次为 $1\mu\text{s}$）。
4. 设预分频系数 $PSC = 71$（即 72 分频，计数器频率 = $72\text{MHz}/(71+1) = 1\text{MHz}$）。
5. 那么自动重载值 $ARR = 999$（计数 1000 次，即 $1000 \times 1\mu\text{s} = 1\text{ms}$）。
6. 结果：$PSC = 71$，$ARR = 999$。

3. PWM 脉宽调制工作原理

- **占空比 (Duty Cycle)**：一周期内高电平所占比例。有效电压 $V_{eff} = V_{max} \times$ 占空比。
- **PWM 模式1 vs 模式2**：
  - **PWM 模式 1**：在向上计数模式中，只要当前计数器 $CNT < CCRx$ (捕获比较值)，通道输出有效电平（高），否则输出无效电平（低）。
  - **PWM 模式 2**：在向上计数模式中，只要当前计数器 $CNT < CCRx$，通道输出无效电平（低），否则输出有效电平（高）。
- **极性配置**：`CCyP` 决定高电平还是低电平是“有效电平”。

4. 输入捕获测量高电平脉宽原理

用于测量外接脉冲信号宽度。

1. 首先配置输入通道为**上升沿捕获**。
2. 当捕获到上升沿时，触发捕获中断，读取此时的 $CCRx$ 寄存器值（记录为 T_1），然后将捕获极性切换为**下降沿捕获**。
3. 当捕获到下降沿时，触发中断，读取此时的 $CCRx$ 值（记录为 T_2）。
4. **注意溢出处理**：如果高电平持续时间过长，计数器可能会溢出，需要在更新中断 (Update Event) 中累加溢出次数 N。
5. 最终脉冲宽度为：

$$\text{脉宽} = (T_2 - T_1) + N \times (ARR + 1)$$

单位：系统时钟周期（通常乘上计数一次的时间 $\text{PSC} + 1/T_{clk}$ 即可换算为秒）。

第7章 串行通信接口（USART）

1. 通信基础理论

- **串行 vs 并行**：
  - **并行**：多条线同时发多位。速度快但成本高、抗干扰差，适合板内短距离传输。
  - **串行**：单条线一位位发。成本低但速度相对慢，适合长距离传输。
- **单工、半双工与全双工**：
  - **单工**：只能单向发送（如收音机）。
  - **半双工**：可以双向，但同一时刻只能有一方发（如对讲机）。

    - **全双工**：双方可同时进行发送和接收（如电话、USART）。
- **异步 vs 同步**：
    - **同步**：需要时钟线同步收发双方（如 SPI, I2C）。效率极高。
    - **异步**：不需要时钟线，靠帧格式的起始/停止位来界定字符（如 UART）。时钟容错率高。
- **波特率 (Baud Rate) 与 比特率 (Bit Rate)**：
    - **波特率**：每秒传输的码元个数。
    - **比特率**：每秒传输的二进制数（bits）。
    - 在二进制调制中（一个码元代表一个bit，STM32串口即为此），**波特率在数值上等于比特率**。双方波特率必须一致才能通信。

2. 串口协议层格式与电平规范

- **一帧异步串口数据**：由 **1个起始位 (低电平 0) + 5~9个数据位 (LSB最低有效位在前) + 校验位 (可选) + 停止位 (0.5/1/1.5/2个高电平 1)** 构成。
- **电平规范区别**：
    - **TTL电平**：MCU引脚直接输出。逻辑 `1` $\approx 3.3\text{V}$，逻辑 `0` $\approx 0\text{V}$。
    - **RS-232电平**：工业标准，抗干扰强。逻辑 `1` = $-3\text{V} \sim -15\text{V}$，逻辑 `0` = $+3\text{V} \sim +15\text{V}$。
    - **转换连接**：TTL电平与RS-232电平互联必须通过电平转换芯片（如 **MAX3232**）。
    - **电脑通信**：电脑常用USB口通信，需要通过 **CH340** 或 PL2303 转换为 TTL 串口。

3. 波特率计算公式与 BRR 寄存器映射（📝必考计算题）

STM32 串口波特率发生器包含一个无符号定点数 `USARTDIV`（由12位整数和4位小数组成）。

- **计算公式**：

$$\text{BaudRate} = \frac{f_{CK}}{16 \times \text{USARTDIV}}$$

    - f_{CK}：USART 外设时钟源。**USART1 挂载在 APB2 上 (72MHz)，USART2~5 挂载在 APB1 上 (36MHz)**！
- **BRR 寄存器存储**：
    - `DIV_Fraction` (位 [3:0])：存放 `USARTDIV` 小数部分 $\times 16$ 后的近似整数。
    - `DIV_Mantissa` (位 [15:4])：存放 `USARTDIV` 的整数部分。

> **Tip**
>
> **期末计算真题演练 (1)**：
> 要求使用挂在 APB2 上的 USART1，波特率配置为 $115,200\text{bps}$。求 `USARTDIV` 值以及 `USART_BRR` 寄存器的配置值。
>
> **解答步骤**：

1. 根据公式：$\text{USARTDIV} = \frac{f_{CK}}{16 \times \text{BaudRate}} = \frac{72{,}000{,}000}{16 \times 115{,}200} = 39.0625$。
2. 得到整数部分 $\text{Mantissa} = 39 = 0x27$。
3. 得到小数部分 $\text{Fraction} = 0.0625 \times 16 = 1.0 = 0x01$。
4. 故 `USART_BRR` = `0x271` 。

期末计算真题演练 (2)（带进位情况）：
计算波特率产生小数部分需做四舍五入。当 $USARTDIV = 50.99$ 时：

1. 整数部分 $\text{Mantissa} = 50 = 0x32$。
2. 小数部分：$0.99 \times 16 = 15.84 \approx 16$。
3. 由于小数乘16后进位（$16 = 0x10$），必须向整数部分进位1。
4. 最终整数部分 = $50 + 1 = 51 = 0x33$；小数部分 = $0 = 0x0$。
5. 故 `USART_BRR` = `0x330` 。

4. 硬件流控制与寄存器标志位

- **nRTS (Request to Send)**：输出引脚。为低电平（有效）时，说明自身接收缓冲区未满，通知对方可以发送。
- **nCTS (Clear to Send)**：输入引脚。检测为低电平时，允许自身发送下一帧数据；若为高电平，则在当前帧发送完毕后暂停。
- **重要标志位**：
  - `TXE` (发送数据寄存器空)：表明数据已被拷贝进移位寄存器， `USART_DR` 可以写入新字节。
  - `TC` (发送完成)：表明一帧数据（包括停止位）已全部发送完毕。
  - `RXNE` (接收寄存器非空)：表明收到了一个字节，需及时读取 `USART_DR` 取出，否则会报 `ORE` 溢出错误。

第8章 直接存储器访问（DMA）

1. DMA 的基本概念与优势

- **定义**：Direct Memory Access (直接存储器访问)。由专门的硬件控制器接管系统总线，在存储器与外设之间，或存储器与存储器之间进行快速数据传送，**完全不需要 CPU 参与**。
- **作用**：解放CPU，避免大批量重复性的数据复制（如ADC多通道扫描采集、DAC数据输出等）消耗处理器时间。

2. STM32 DMA 特征与通道分配

- **双控制器架构**：
  - **DMA1**：有 7 个独立通道。
  - **DMA2**：有 5 个独立通道（仅限大容量和互联型芯片）。

- **优先级仲裁**：
  1. **软件配置**：各通道可配置4种软件优先级（Very High, High, Medium, Low）。
  2. **硬件决定**：若软件优先级配置相同，则通道编号越小的优先级越高（即 Channel 1 > Channel 2），同时 DMA1 优先级高于 DMA2。
- **总线仲裁机制**：DMA与Cortex-M3内核共享数据总线。当它们冲突时，总线仲裁器会采用循环调度算法，**保证 CPU 至少能分得 50% 的总线带宽**，防止内核“饥饿”。

3. 数据对齐与传输模式

- **地址增量模式**：外设地址（`DMA_PeripheralInc`）和内存地址（`DMA_MemoryInc`）可以使能或失能。
  - *外设到内存的ADC多通道传输*：外设寄存器地址是固定的（`&ADC1->DR`），故外设地址增量**失能**；内存是数组首地址，需按位存入，故内存地址增量**使能**。
- **数据宽度对齐**：源宽度和目标宽度可设为 Byte(8位)、HalfWord(16位)、Word(32位)。如果两端宽度不一致，DMA会自动执行“数据打包”或“拆包”。
- **普通模式 vs 循环模式**：
  - **普通模式**：传输计数器计数到 0 时，DMA 通道关闭。
  - **循环模式**：计数到 0 后，自动重载计数器，重新指向首地址继续循环传输，适用于持续的ADC多通道电压采集。

4. DMA 初始化配置结构体成员详解

```
DMA_InitTypeDef DMA_InitStructure;
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)&ADC1->DR; // 外设物理地址
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)ADC_Value;     // 内存目标数组基地址
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;              // 外设作为数据源（外设->内存）
DMA_InitStructure.DMA_BufferSize = 4;                           // 传输数量（计数寄存器 CNDTR = 4）
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable; // 外设地址不递增
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;         // 内存地址递增
DMA_InitStructure.DMA_PeripheralDataSize =
DMA_PeripheralDataSize_HalfWord; // 16位宽度
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
// 16位宽度
DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;                 // 循环模式
DMA_InitStructure.DMA_Priority = DMA_Priority_High;             // 软件高优先级
DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;                    // 关闭内存到内存（使用外设触发）
```

```
DMA_Init(DMA1_Channel1, &DMA_InitStructure);                    // 针对通道
1 进行初始化
```

第9章 模拟数字转换器（ADC）

1. ADC 转换的三个基本步骤

模拟信号转为数字信号的核心过程：

1. **采样**：在时间上对模拟信号进行离散化抽样（采样周期 T）。
2. **量化**：将连续的幅值转换为离散的台阶阶梯。
3. **编码**：将量化后的数值转换为二进制输出（STM32是 12 位分辨率，输出范围 $0 \sim 4095$）。

2. STM32 ADC 主要特征与时钟配置

- **分辨率与输入范围**：12 位逐次逼近型ADC。输入电压范围：$V_{REF-} \leq V_{IN} \leq V_{REF+}$（即最小 0V，最大 3.3V）。
- **时钟源限制**：ADC 的输入时钟 $ADCCLK$ 来源于 APB2 时钟通过预分频器（除以 2/4/6/8）获得，且 $ADCCLK$ **最大不能超过** 14MHz，否则转换精度无法保证。
  - 当 APB2 主频为最高时钟时（72MHz），分频器只能选择 **6分频 (得到12MHz)** 或 8分频 (得到9MHz)。**绝不能选择 2/4分频**。

3. 规则通道组 vs 注入通道组

- **规则通道组 (Regular Group)**：类似于**主程序**。最多支持 16 个通道，转换结果统一存放在单一的数据寄存器 `ADC_DR` 中。**多通道连续扫描转换时，必须使用 DMA 及时读走**，否则新数据会覆盖旧数据。
- **注入通道组 (Injected Group)**：类似于**中断**。最多支持 4 个通道，其优先级高，转换可以打断规则通道的转换。拥有 4 个独立的专用数据寄存器 `ADC_JDR1~4`，不会存在覆盖冲突。

4. 数据对齐与自校准

- **对齐方式**：由于转换结果是12位，而数据寄存器是16位，可通过 `ALIGN` 位选择：
  - **右对齐 (Right Alignment)**（常用）：高4位填充0（规则组）或符号位。
  - **左对齐 (Left Alignment)**：低4位填充0。
- **自校准**：大幅消除电容器组变化产生的系统性误差。**建议在每次上电后进行一次自校准**，且校准必须在 ADC 处于上电状态且完成上电延迟（`ADON = 1`）后才能启动。

5. 转换时间计算（📝必考计算题）

- **计算公式**：

$$T_{conv} = \text{采样时间} + 12.5 \text{ 个 ADC 时钟周期}$$

- 采样时间可由软件编程配置（如 1.5, 7.5... 239.5个周期）。量化编码时间固定为 12.5 个周期。

> **Tip**
>
> **期末计算真题演练**：
> 已知 $ADCCLK = 14\text{MHz}$，通道采样时间配置为最快阶段的 1.5 个周期。求该通道的转换时间。
>
> **解答步骤**：
>
> 1. 根据公式：$T_{conv} = 1.5 + 12.5 = 14$ 个 ADC 时钟周期。
> 2. 由于时钟频率为 14MHz，每个周期时间为 $\frac{1}{14\text{MHz}}\mu\text{s}$。
> 3. 故该通道的转换时间为：$T_{conv} = 14 \times \frac{1}{14} = 1\mu\text{s}$。

6. ADC 运行转换模式

- **单次转换模式 (Single Conversion)**：每次使能或触发只转换选中的那一个通道一次，然后停止。
- **连续转换模式 (Continuous Conversion)**：转换结束后自动开启下一轮转换，常用于连续采样的热敏电阻、电位器。
- **扫描模式 (Scan Mode)**：用于多通道。按照 `SQRx` 或 `JSQR` 寄存器里规定的顺序自动对通道列表挨个转换。
- **间断模式 (Discontinuous Mode)**：将待转换的序列分成数个子组，每次收到触发脉冲，仅转换其中一个子组的 n 个通道。

7. 内部温度传感器

- 连接在 **`ADC1_IN16`** 输入通道上，支持测量范围 $-40^{\circ}\text{C} \sim 125^{\circ}\text{C}$，精度约为 $\pm 1.5^{\circ}\text{C}$。
- 使用前必须将 `ADC_CR2` 寄存器中的 **`TSVREFE`** 位（第23位）置1，以激活该通道和内部参考电压通道。

💡 期末高分复习建议

1. **分值大头**：第2章（基础架构与位带）、第4章（GPIO模式）、第5章（NVIC与优先级配置）、第6章（定时器时间计算与PWM机制）占分最高。
2. **公式速背**：流水线吞吐率、位带地址映射、定时器溢出时间、波特率发生器、ADC转换时间计算。每年必考至少2道大计算题。
3. **读代码与改错**：能够识别标准外设库的初始化步骤（开时钟 -> 赋结构体 -> 调用Init -> 使能外设）。注意如“忘记清除中断挂起标志位”等高频程序BUG。

祝同学们期末考试顺利，全部拿A+！💯