5.3　RP2040 芯片引脚和功能

5.3.1　RP2040 芯片的封装和引脚功能

RP2040 芯片是方形扁平无引脚封装(quad flat no-leads package,QFN),56 个引脚隐藏于芯片底部四周边缘,它的引脚布局和名称如图 5-10 所示。

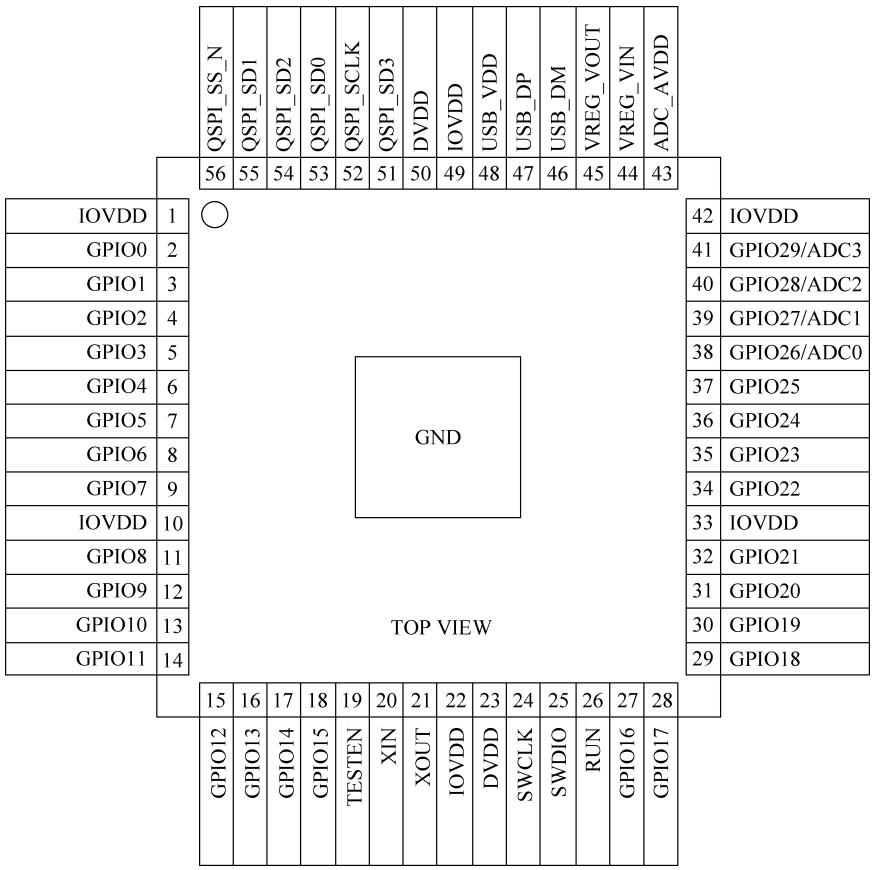

图 5-10　RP2040 引脚布局和名称

RP2040 的引脚分为电源引脚、复位引脚、振荡器引脚、USB 引脚、QSPI 引脚和 GPIO 引脚,其引脚功能如表 5-1 所示。

表 5-1　RP2040 引脚功能表

名　　称	描　　述
GPIOx	通用输入输出引脚,RP2040 也让内部其他外设共享这些引脚
GPIOx/ADCy	通用输入输出和 ADC 共用引脚,因为需要兼容模拟信号,与其他只兼容数字输入输出的 GPIO 电路结构有所不同

在读写过程中，任何一方随时都可以通过在 SCL 高电平时拉低 SDA 来结束通信，或者在接收到数据后给出 noACK 来结束数据传输。

10.5　RP2040 芯片的 I²C 控制器

10.5.1　RP2040 芯片的 I²C 控制器的特性

RP2040 芯片的 I²C 控制器使用 2 个 GPIO 引脚作为 SCL、SDA，这两个引脚必须按照 I²C 要求进行配置：上拉模式、斜率限制和输入使能施密特触发器。上拉模式相当于集电极开路，输出低电平可以拉低，高电平时上拉电阻拉成高电平。斜率限制使得输出边沿不是特别陡峭，减少高频噪声。输入施密特触发功能可以减小输入噪声的影响。

RP2040 芯片的 I²C 控制器支持独立的主器件和从器件，支持 7 位和 10 位地址模式，接收和发送具有 FIFO。支持三种传输速度：100kHz 标准模式(standard mode)、400kHz 快速模式(fast mode)和 1MHz“快速＋”模式(fast mode ＋)。

发送器从发送 FIFO 中取得 12 位发送字，其中 8 位发送数据和 4 位命令编码。发送器根据 12 位发送字的内容自动发送数据、产生停止位或应答等。

当地址匹配时，接收器自动应答，并把接收的数据自动存入接收 FIFO 中，不用 CPU 干预。

10.5.2　发送 FIFO 中的数据和命令

发送 FIFO 中的数据和命令统一编码成 11 位信息，低 8 位数据的高 3 位是传送附加命令信息。对地址 IC_DATA_CMD 写就是写入发送 FIFO，而相同地址读就是读取接收 FIFO。发送字各个位域的名称与含义如表 10-3 所示。

表 10-3　IC_DATA_CMD 各个位域的名称与含义

位　域	名　称	含　义
10	Restart	只写，发送该数据之前发送起始位和地址，以开始读写过程
9	Stop	只写，发送该数据后发送停止位
8	CMD	只写，读写命令位，0-写入；1-读出
7:0	data	读写，8 位数据

因此，I²C 总线通信的过程可以通过写入 FIFO 中的数据流有效控制。

图 10-18 给出了作为主器件发送的时序图。首先，写入空的发送 FIFO，FIFO 变成非空，触发起始位开始发送地址帧。收到应答后，发送数据并自动读取 FIFO 中的内容。如果数据没有附加停止位，即使 FIFO 变空，也不会发送停止位，而是

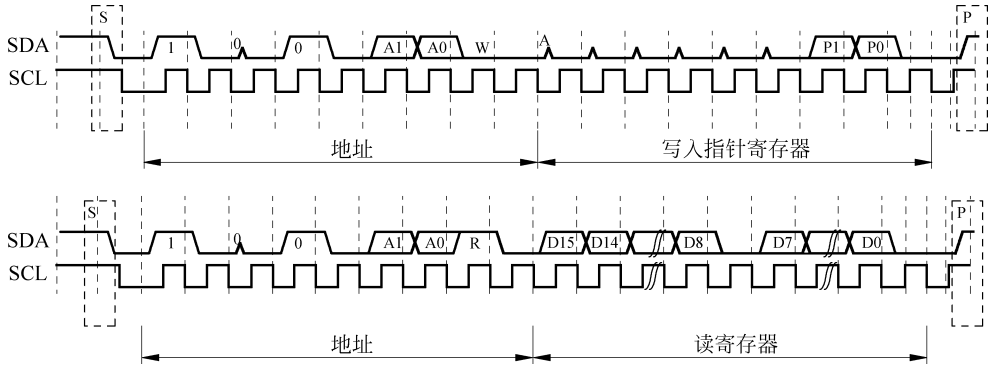

图 11-16　ADS101x 读寄存器时序

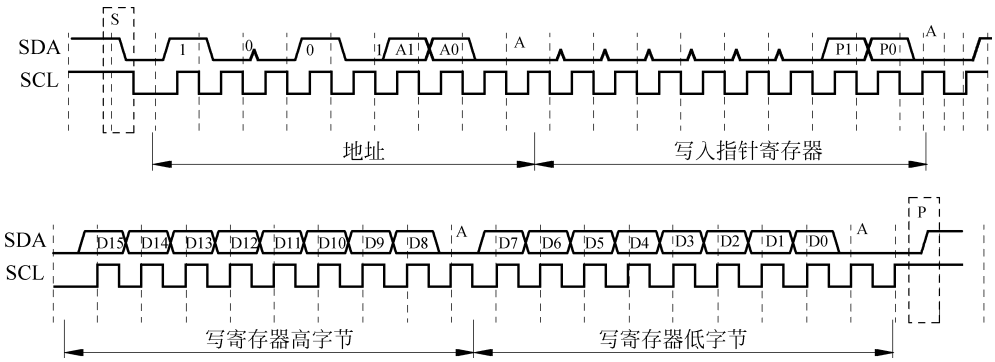

图 11-17　ADS101x 写寄存器时序

11.3　RP2040 芯片内置的 A/D 及编程

11.3.1　RP2040 芯片内置 A/D 转换器

RP2040 芯片内置 12 位逐次比较型 A/D 转换器，在 48MHz 时钟下可以达到 500kSPS 的性能。它有 5 路输入，4 路引脚输入电压、1 路片内温度传感器用来测量芯片温度。它支持 4 个字深度的 FIFO 存储转换结果，支持 DMA 和中断，并且包含一个分数分频器用来给连续采样作为定时。图 11-18 给出了 RP2040 芯片内置 A/D 转换器结构框图。

使能 ADC 需要在控制寄存器 CS 中 EN 位域写入 1，经过一个短暂的复位过程之后，CS.READY 位域变成 1，ADC 就准备好了，可以在任何时候在 CS.EN 中写入 0 关闭 ADC。

ADC 支持 2 种操作模式：单次模式和连续模式。

是在 CPU 空闲时决定下一个占用 CPU 的是哪个进程。

抢占式多任务系统就是在协同的基础上增加了 tick 中断，通过时间片的划分，强制各个任务进行轮换，每个任务可以不必主动放弃 CPU 的占用。

协同式多任务系统由于没有频繁的 tick 中断，而只在必要时调用任务调度算法，因此系统效率可以大大提高，在性能较低的 CPU 中可以考虑使用。但是，每个任务都需要适时地放弃 CPU 的占用，需要每个任务都进行合适的编码，设计上较复杂。一旦一个任务不能及时放弃 CPU 占用，就会造成系统锁死，任务无法切换。

抢占式多任务系统虽然耗费更多资源用来进行任务调度，但是每个任务可以不必考虑主动放弃 CPU 的占用，即使每个任务都不主动放弃 CPU 占用系统也不会锁死，使得系统可靠性提高。

13.1.6　任务调度算法

当前任务时间片用尽或放弃执行时，需要任务调度算法决定下一个时间片由就绪状态任务中的哪个任务占用。对于实时多任务操作系统内核来说，最基本的调度算法有两种：基于优先级的算法和基于时间片轮转的算法。

基于优先级的算法是对每一个进程都赋予一个优先级，每次进行进程调度时优先级高者优先。这种算法的特点是只根据优先级进行调度，如果高优先级的进程不放弃占用 CPU，则低优先级的进程就没有机会获得执行。

基于时间片轮转的算法则完全不考虑进程的优先级，平等对待所有进程。该算法维持一个队列，当前进程执行完当前的时间片后排到队列尾部，由队列头部的进程取得时间片。这样每个进程都是平等排队，缺点是对于实时性高的进程没办法提高其响应速度。

更好的方法是把优先级和时间片轮转结合起来，比如维持多个队列，不同优先级进程排到不同队列中，而调度时给高优先级的队列更多的执行机会。

总之，调度算法是多任务系统复杂且关键的部分，这里不详细讨论。

13.2　ARM CM0 中多任务的实现方法

13.2.1　主堆栈和线程堆栈

由于堆栈在任务实现中起着重要作用，因此，ARM CM0 对堆栈操作做了专门的优化，适合多任务的实现。

前面讲过，堆栈指针是 R13(SP)，实际上这里的堆栈指针是指主堆栈指针(main stack pointer，MSP)，另外还有一个线程堆栈指针(thread stack pointer，PSP)专门供多任务操作系统中的任务代码使用。由于复位后默认一直使用 MSP，