第 5 章 嵌入式输入/输出设备的接口

CHAPTER 5

学习目标与要求

1. 掌握嵌入式输入/输出接口的结构和数据传输方式。
2. 掌握常见的嵌入式输入/输出接口的实现原理。
3. 掌握常见的嵌入式输入/输出接口的使用方法。

5.1 I/O 接口

5.1.1 接口结构

嵌入式系统需要连接的外部输入/输出设备（简称外设）具备多样性、复杂性和异构性（处理速度不同）等特点，因此一般需要借助于接口电路来实现外设与 CPU 之间的连接。负责外设与 CPU 连接的中间接口电路即输入/输出接口电路，简称 I/O 接口。I/O 接口在嵌入式系统中所处的位置如图 5.1 所示。

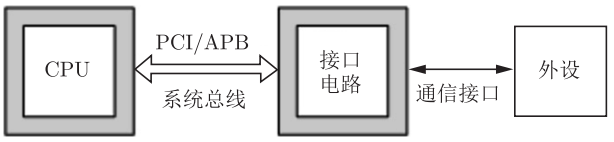

图 5.1 I/O 接口在嵌入式系统中所处的位置

5.1.2 I/O 接口组成

I/O 接口通常具备如下几方面的功能。

（1）数据缓存。

（2）CPU 命令接收和执行。

（3）信号电平转换。

（4）数据格式转换。

（5）外设选择。

（6）中断管理。

为实现上述功能，I/O 接口一般包括数据缓存、逻辑控制和外设连接 3 个基本的功能模块，I/O 接口的结构如图 5.2 所示。

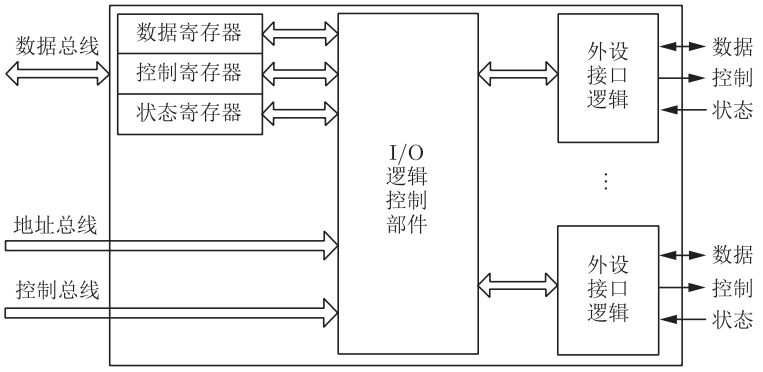

图 5.2　I/O 接口的结构

数据缓存功能模块主要对应一组寄存器，负责 I/O 交互过程的控制和记录等。数据缓存寄存器可根据存放数据类型的不同，划分为数据寄存器、控制寄存器和状态寄存器。其中数据寄存器用来存放交互过程中需传输的数据，控制寄存器用来存放外设、I/O 接口的控制命令（如 I/O 接口启动、中断使能等），状态寄存器则用来记录外设、I/O 接口的状态信息（如中断状态、系统故障等）。

逻辑控制功能模块主要接收 CPU 传输过来的地址，根据地址进行外设选择。

外设连接功能模块是 I/O 接口与外设的交互接口。同一个 I/O 控制器可与多个外设连接，因此需要设置一个或多个外设连接逻辑功能来实现外设与 CPU 的连接。

5.1.3　I/O 接口的数据传输

I/O 接口的数据传输方式一般包括程序查询方式、中断方式和直接访存方式 3 种。

程序查询方式是 CPU 负责查询外设的状态寄存器，判断外设的数据是否就绪。若就绪，则直接进行数据传输；否则，CPU 等待数据就绪或继续查询其他外设。

中断方式是当外设数据就绪时，外设向 CPU 发出中断服务请求信号。CPU 根据当前执行程序以及外设中断请求各自优先级的高低，判断是否暂停当前执行的程序以响应外设的中断服务请求。若外设中断服务请求的优先级高，则 CPU 暂停当前执行程序来完成外设数据传输；否则继续执行完当前程序再响应外设中断服务请求。从 CPU 响应的角度看，程序查询方式是一种主动的处理方式，程序中断方式是一种被动的处理方式。因此，后一种处理方式可实现 CPU 与外设的并行处理，使得 CPU 的利用率更高。

直接访存方式 DMA 在外设与内存之间建立直接的数据传输通道，数据传输过程中不需要 CPU 的参与，从而进一步提升了数据传输效率。I/O 接口 3 种数据传输方式的对比如表 5.1 所示。

表 5.1　I/O 接口 3 种数据传输方式的对比

数据传输方式	优　　点	缺　　点	适 用 场 合
程序查询方式	实现成本低	CPU 利用率低 低速的字符外设 数据传输速度慢	低速的字符外设
中断方式	CPU 与外设可并行执行 CPU 利用率较高 数据传输速度较快	不适用于批处理数据传输	常见外设
直接访存方式	无须 CPU 参与 数据传输速度最快	需 DMA 控制器	高速外设

5.2　GPIO

5.2.1　GPIO 概述

GPIO（General Purpose Input/Output，通用输入/输出接口）用来实现外设与 CPU 的连接。I/O 接口的功能一般固定，但是 GPIO 可通过寄存器配置和编程实现功能复用。每个 GPIO 端口对应微处理器的一组 16 个引脚，GPIO 端口的功能通过一组寄存器来进行配置。GPIO 端口的内部结构如图 5.3 所示。

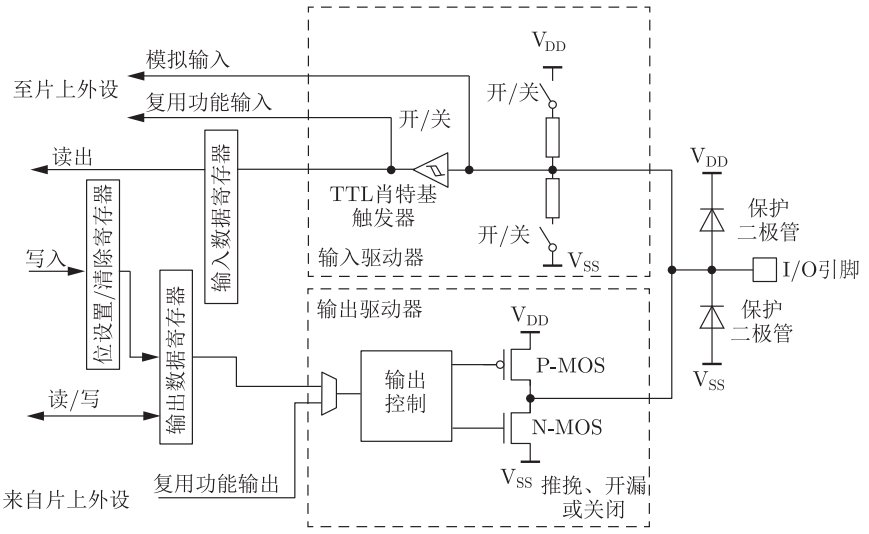

图 5.3　GPIO 端口内部结构图

以 Cortex-M4 微处理器为例，它包含 GPIO A～GPIO I 共 9 个 GPIO 端口，每个 GPIO 端口对应 1 个 32 位操作寄存器（GPIOx_BSRR）、1 个 32 位锁定寄存器（GPIOx_LCKR）、4 个 32 位配置寄存器（包括 GPIOx_MODER、GPIOx_OTYPER、GPIOx_OSPEEDR 和 GPIOx_PUPDR）、2 个 32 位数据寄存器（GPIOx_IDR 和 GPIOx

_ODR）和 2 个 32 位备用功能选择寄存器（GPIOx_AFRH 和 GPIOx_AFRL）。由于每个端口对应 16 个引脚，因此上述寄存器中的每 2 位控制一个引脚。Cortex-M4 的 GPIO 端口对应寄存器的功能如表 5.2 所示。

表 5.2　GPIO 寄存器功能描述

配置寄存器组		模式（输入/输出/模拟/备用）	GPIOx_MODER
		输出类型（推挽/开漏）	GPIOx_OTYPER
		输出速度	GPIOx_OSPEEDR
		上拉/下拉使能	GPIOx_PUPDR
数据寄存器组	输出数据	GPIOx_ODR	
	输入数据	GPIOx_IDR	
位操作寄存器		GPIOx_BSRR	
配置锁定寄存器		GPIOx_LCKR	
复用功能及重映射寄存器		GPIOx_AFRL, GPIOx_AFRH	

5.2.2　GPIO 功能特点

GPIO 具备如下资源。

（1）输出状态：推挽、开漏，上拉/下拉。

（2）输入状态：浮动、上拉/下拉、模拟。

（3）数据从输出数据寄存器（GPIOx_ODR）或外设（备用功能输出）输出。

（4）输入数据到输入数据寄存器（GPIOx_IDR）或外设（备用功能输入）。

（5）I/O 端口速度可配置。

（6）位设置和位重置寄存器（GPIOx_BSRR），然后可按位写入 GPIOx_ODR 寄存器。

（7）可通过配置锁定寄存器（GPIOx_LCKR）锁定 I/O 配置。

（8）可实现 ADC 和 DAC 模拟信号输入/输出。

（9）复用功能输入/输出选择寄存器（每个 I/O 端口最多支持 16 个 AFS 复用功能）。

（10）可实现每两个时钟周期快速切换。

（11）高度灵活的引脚复用功能，允许 I/O 引脚使用 GPIO 或几种外设功能之一。

5.2.3　GPIO 输入/输出模式

GPIO 端口对应的引脚可配置为如下 8 种模式，其中（1）～（4）为输入模式，（5）和（6）为输出模式，（7）和（8）为复用模式。

（1）浮空输入_IN_FLOATING：可以用作按键识别。

（2）带上拉输入_IPU：I/O 内部上拉电阻输入。

（3）带下拉输入_IPD：I/O 内部下拉电阻输入。

（4）模拟输入_AIN：应用 ADC 模拟输入，或者低功耗模式下进行节能操作。

（5）开漏输出_OUT_OD：I/O 输出 0 接 GND；I/O 输出 1 悬空，需外接上拉电阻才能实现高电平输出。

（6）推挽输出_OUT_PP：I/O 输出 0 接 GND；I/O 输出 1 接 VCC，读输入值未知。

（7）复用功能的推挽输出_AF_PP：片内外设功能（如 I^2C 的 SCL、SDA）。

（8）复用功能的开漏输出_AF_OD：片内外设功能（TX1、MOSI、MISO、SCK）。

5.2.4　GPIO 引脚复用

由于微处理器的引脚有限，为了提高 I/O 接口的利用率，更好地协调不同外设与 CPU 之间的交互，将通过多路复用器实现引脚的多路复用。引脚多路复用是通过寄存器配置，实现同一引脚在不同时刻与不同外设之间的交互，即不同的外设可共享相同的引脚连接。在某一个时刻，只允许引脚与某一个外设交互。

如表 5.2 所示，每个 GPIO 端口均对应一组寄存器，其中 GPIOx_AFRL 和 GPIOx_AFRH 两个寄存器用来实现 x 端口对应引脚的多路复用功能。GPIOx_AFRL 和 GPIOx_AFRH 两个 32 位寄存器使用寄存器的每 4 位来配置一个引脚。GPIOx_AFRL 寄存器配置 x 端口对应的第 0～7 号引脚，GPIOx_AFRH 配置 x 端口对应的第 8～15 号引脚。如以 STM32F4 系列芯片为例，每个端口包含的 16 个引脚将通过多路复用器一一对应连接到 16 个复用功能模块（AF0～AF15）。GPIOx_AFRL 寄存器用来选择 AF0～AF7 功能模块，GPIOx_AFRH 用来选择 AF8～AF15 功能模块。GPIO 引脚的多路复用器和多路复用配置如图 5.4 所示。

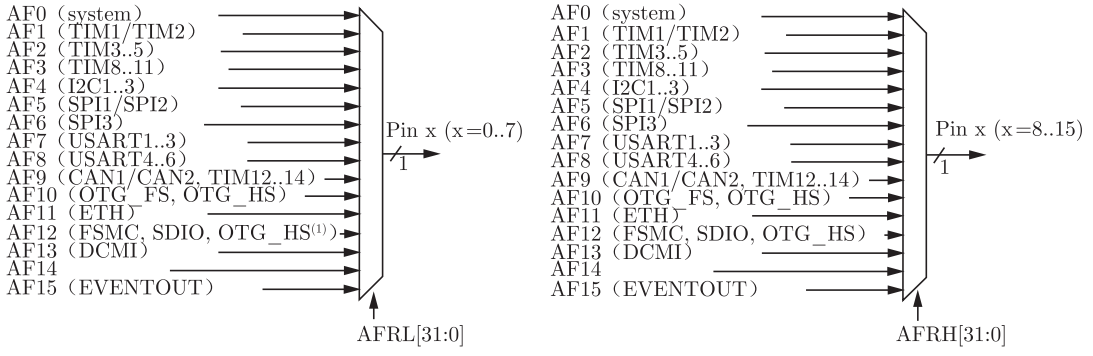

图 5.4　GPIO 引脚的多路复用器和多路复用配置

如图 5.4 所示，系统复位后，所有的 GPIO 引脚将连接到系统的备用功能 AF0，所有外设的复用功能将映射到 AF1～AF13，EVENTOUT 事件将映射到 AF15。

举例说明，假设开发板 x 端口对应的 11 号引脚 PC11 可作为 SPI3_MISO/U3_RX/U4_RX/SDIO_D3/DCMI_D4/I2S3ext_SD 等复用功能，而现在需要配置 PC11 为 SDIO_D3 功能使用。11 号引脚的复用功能将通过 GPIOx_AFRH[15:12] 来进行配置，因此需要选择 AF12，即设置 GPIOx_AFRH[15:12]=AF12。

5.2.5　GPIO 配置

GPIO 引脚的配置需完成输入配置、输出配置和复用配置等。

1. 输入配置

当 GPIO 引脚被配置为输入时，其输入配置如图 5.5 所示。

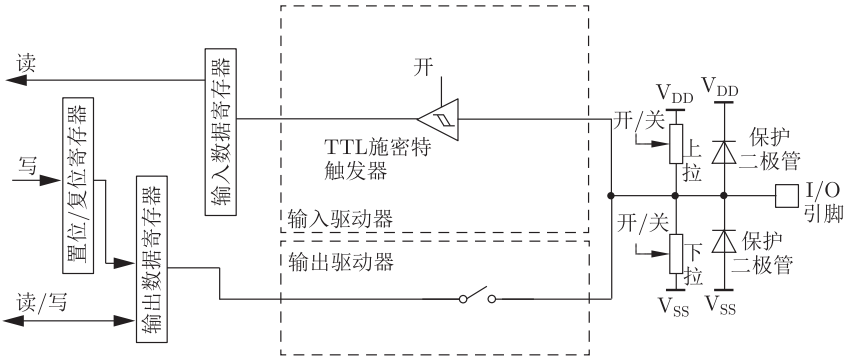

图 5.5　输入配置

（1）输出缓冲器被禁止。

（2）施密特触发输入被激活。

（3）根据寄存器 GPIOx_PUPDR 中的值选择引脚为上拉或下拉。

（4）GPIO 引脚上的数据在每个 AHB1 时钟周期被采样到输入数据寄存器。

（5）对输入数据寄存器的读访问可获得 GPIO 状态。

2. 输出配置

当 GPIO 引脚被配置为输出时，其输出配置如图 5.6 所示。

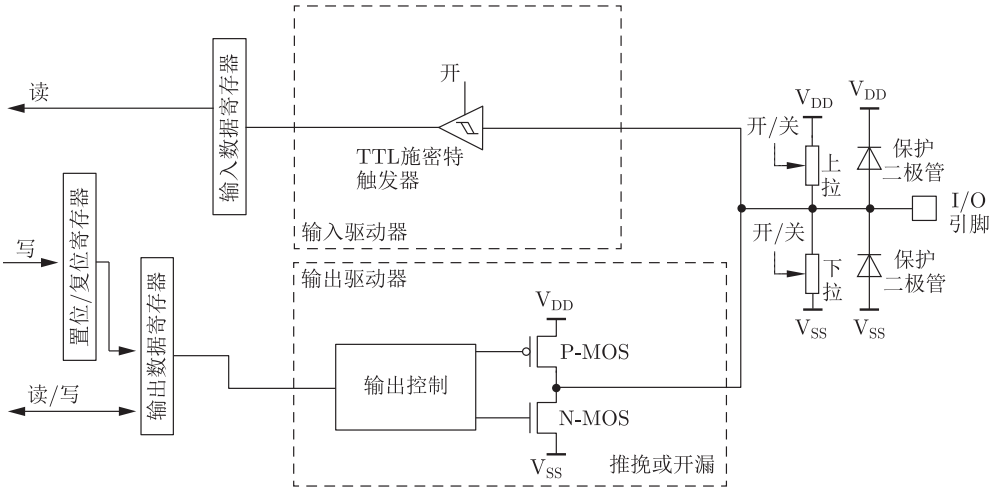

图 5.6　输出配置

（1）输出缓冲器被激活。

① 开漏模式：输出寄存器上的 0 激活 NMOS，而输出寄存器上的 1 将端口置于高阻状态（PMOS 从不被激活）。

② 推挽模式：输出寄存器上的 0 激活 NMOS，而输出寄存器上的 1 将激活 PMOS。

（2）施密特触发器输入被激活。

（3）根据寄存器 GPIOx_PUPDR 中的值选择引脚为弱上拉或弱下拉。

（4）GPIO 引脚上的数据在每个 AHB1 时钟周期被采样到输出数据寄存器 GPIOx-ODR。

（5）对输入数据寄存器的读访问可得到 GPIO 状态。

（6）对输出数据寄存器的读访问得到最后一次写入的值。

3. 复用配置

当 GPIO 引脚被配置为输出时，其复用配置如图 5.7 所示。

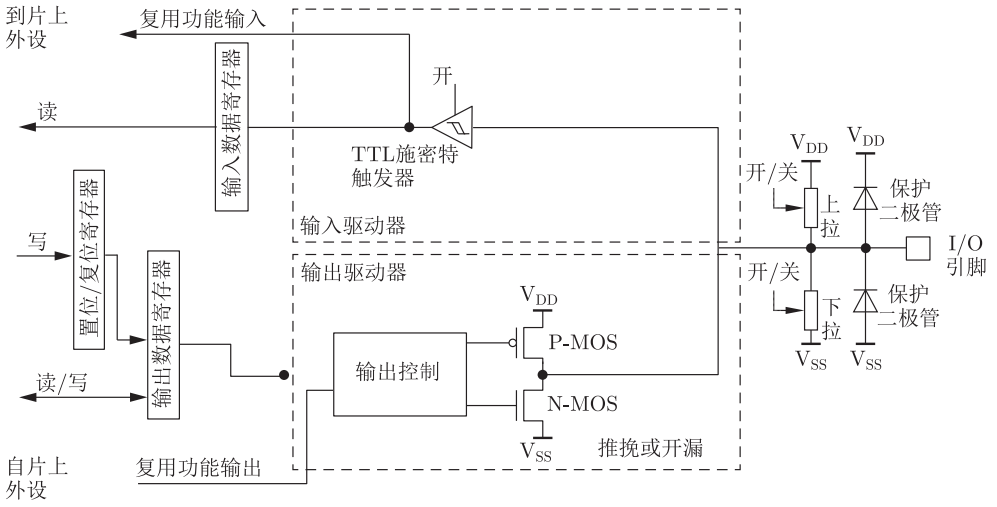

图 5.7　复用功能配置

（1）输出缓冲器可以被配置成开漏或推挽。

（2）输出缓冲器被片上外设信号驱动。

（3）施密特触发器输入被激活。

（4）根据 GPIOx_PUPDR 寄存器中的值选择引脚为弱上拉或弱下拉。

（5）GPIO 引脚上的数据在每个 AHB1 时钟周期被采样到输入数据寄存器 GPIOx-IDR。

（6）对输入数据寄存器的读访问将得到 GPIO 状态。

（7）对输出数据寄存器的读访问得到最后一次写入的值。

根据以上机制，可以设计一个通用的配置函数来进行 GPIO 配置。该函数的代码如下。

```
// GPIO 通用设置，GPIOx: GPIOA～GPIOI
// BITx: 0X0000～0XFFFF，位设置，每个位代表一个 I/O
// 第0位代表 Px0，第1位代表 Px1，以此类推。比如0X0101代表同时设置Px0和Px1
// MODE：0～3模式选择，0输入（系统复位默认状态）；1普通输出；2复用功能；
//              3 模拟输入
// OTYPE: 0/1，输出类型选择，0推免输出；1开漏输出
// OSPEED: 0～3，输出速度设置，0 2MHz；1 25MHz；2 50MHz；3 100MHz
// PUPD: 0～3，上下拉设置，0不带上下拉；1上拉； 2下拉； 3保留
// 注意：在输入模式（普通输入/模拟输入）下，OTYPE 和 OSPEED 参数无效

void GPIO_Set(GPIO_TypeDef* GPIOx,u32 BITx, u32 MODE, u32 OTYPE, u32
    OSPEED, u32 UPPD)
{
        u32 pinpos=0, pos=0, curpin=0;
        for(pinpos=0;pinpos<16;pinpos++)
        {
                pos=1<<pinpos;                  //一位位检查
                curpin=BITx&pos;
                if(curpin==pos)                 //需要设置
                {
                        GPIO->MODER&=~(3<<(pinpos*2));      //清除原来的
                            设置
                        GPIO->MODER|=(MODE<<(pinpos*2)); //设置新的模式
                        if(MODE==0X01)||(MODE==0X02)
                        {
                                GPIO->OSPEED&=~(3<<(pinpos*2));
                                GPIO->OSPEED|=(OSPEED<<(pinpos*2));
                                GPIO->OTYPE&=~(1<<pinpos);
                                GPIO->OTYPE|=(OTYPE<<pinpos);
                        }
                        GPIO->PUPDR&=~(3<<(pinpos*2));
                        GPIO->PUPDR|=(PUPDR<<(pinpos*2));
                }
        }
}
```

5.3 外部中断/事件

5.3.1 外部中断/事件概述

外部中断/事件是实现外设与微处理器之间通信的主要方式之一。微处理器中包含一

个外部中断/事件控制器（External Interrupt/Event Controller，EXTI），用于管理微处理器中的外部中断/事件线。每个中断/事件线都对应一个边沿检测器，实现输入信号的上升沿或下降沿检测。EXTI 可以对每个中断/事件线进行单独配置，配置中断或者事件类别以及触发时间的属性。EXTI 还包含一个挂起寄存器，用于记录每个中断/事件线的状态。

Cortex-M4 中的 EXTI 可支持如下所示的 23 个外部中断/事件：

（1）EXTI 线 0~15：对应 GPIO 的输入中断。

（2）EXTI 线 16：连接到 PVD 输出。

（3）EXTI 线 17：连接到 RTC 闹钟事件。

（4）EXTI 线 18：连接到 USB OTG FS 唤醒事件。

（5）EXTI 线 19：连接到以太网唤醒事件。

（6）EXTI 线 20：连接到 USB OTG HS（在 FS 中配置）唤醒事件。

（7）EXTI 线 21：连接到 RTC 入侵和时间戳事件。

（8）EXTI 线 22：连接到 RTC 唤醒事件。

5.3.2　EXTI 结构和外部中断/事件响应过程

如图 5.8 所示为 EXTI 结构和外部中断/事件响应过程。图中的实线箭头标注了外部中断信号的传输路径。响应具体的过程是：外设的中断信号从编号为 1 的芯片引脚进入，经过编号为 2 的边沿检测电路，通过编号为 3 的或门进入中断挂起请求寄存器。最后，经过编号为 4 的与门输出到内嵌向量中断控制器（Nested Vectored Interrupt Controller，NVIC）检测电路。该边沿检测电路受上升沿或下降沿选择寄存器控制，用户可以通过它们控制在哪一个边沿产生中断。由于上升沿或下降沿分别受两个并行的选择寄存器控制，所以用户可以选择上升沿或下降沿，或者同时选择上升沿和下降沿。如果只有一个寄存器控制，则只能选择一种边沿触发。编号 3 所对应或门的另一个输入是软件中断/事件寄存器，软件可以优先于外部信号请求中断或事件，即当软件中断/事件寄存器的对应位为 1 时，不管外部信号如何，编号为 3 的或门都会输出有效信号。中断或事件请求信号经过编号为 3 的或门后进入挂起请求寄存器，挂起请求寄存器中记录了外部信号的电平变化。在此之前，中断和事件的信号传输路径一致。外部请求信号经过编号为 4 的与门后，向 NVIC 发出一个中断请求。如果中断屏蔽寄存器的对应位为 0，则该请求信号不能传输到与门的另一端，从而实现了中断屏蔽；反之，则正常响应中断，进入中断服务程序。

如图 5.8 所示，虚线箭头标注了外部事件信号的传输路径。外部事件请求信号经过编号为 3 的或门后，进入编号为 5 的与门。这个与门的作用与编号为 4 的与门类似，用于引入事件屏蔽寄存器的控制。最后，编号 6 所对应脉冲发生器的跳变信号转变为一个单脉冲，输出到芯片中的其他功能模块。如图 5.8 所示，从外部激励信号来看，中断和事件的产生源可以一样。中断和事件的区别在于，中断需要 CPU 参与后续的中断响应，包括上下文切换、执行中断服务程序、恢复现场并返回等；但是事件仅触发脉冲发生器产生一个单脉冲，进而由硬件自动完成事件的响应，比如 DMA 操作、AD 转换等。

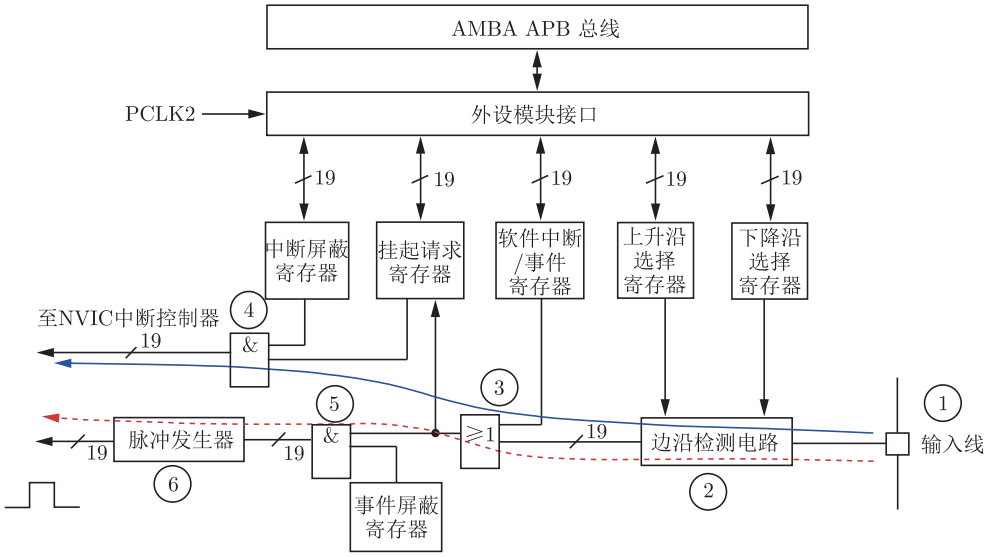

图 5.8　EXTI 结构和外部中断/事件响应过程

5.3.3　外部中断/事件的配置

如果要产生中断/事件，则必须配置中断/事件线。首先，选择中断/事件的边沿触发信号，设置两个触发选择寄存器，同时在中断/事件屏蔽寄存器相应位写 1，允许中断/事件请求。当外部中断/事件线发生了配置的边沿触发信号时，将产生一个中断/事件请求，对应挂起位将被置为 1。在挂起寄存器对应位写 1 将清除该中断。通过在软件中断/事件寄存器写 1，也可以由软件产生中断/事件请求。

具体而言，外部中断/事件的中断配置如下。

1. 硬件中断选择配置

通过如下过程可配置 23 个中断线路作为中断源。

（1）配置 23 个中断线的屏蔽位（EXTI_IMR）。

（2）配置所选中断线路的触发选择位（EXTI_RTSR 和 EXTI_FTSR）。

（3）配置对应到外部中断控制器（EXTI）的 NVIC 中断通道的使能和屏蔽位，使得 23 个中断线路中的中断请求可以被正确响应。

2. 硬件事件选择配置

通过如下过程可配置 23 个线路作为事件源。

（1）配置 23 个事件线的屏蔽位（EXTI_EMR）。

（2）配置所选事件线的触发选择位（EXTI_RTSR 和 EXTI_FTSR）。

3. 软件中断/事件选择配置

19 个线路可配置为软件中断/事件线，其配置如下：

（1）配置 19 个中断/事件线的屏蔽位（EXTI_IMR 和 EXTI_EMR）。

（2）设置软件中断寄存器的请求位（EXTI_SWIER）。