小川工作室编写，本书为 LM3S 的 USB 芯片编写，上传的均为草稿，还有没修改，可能还有很多地方不足，希望各位网友原谅！

QQ:2609828265

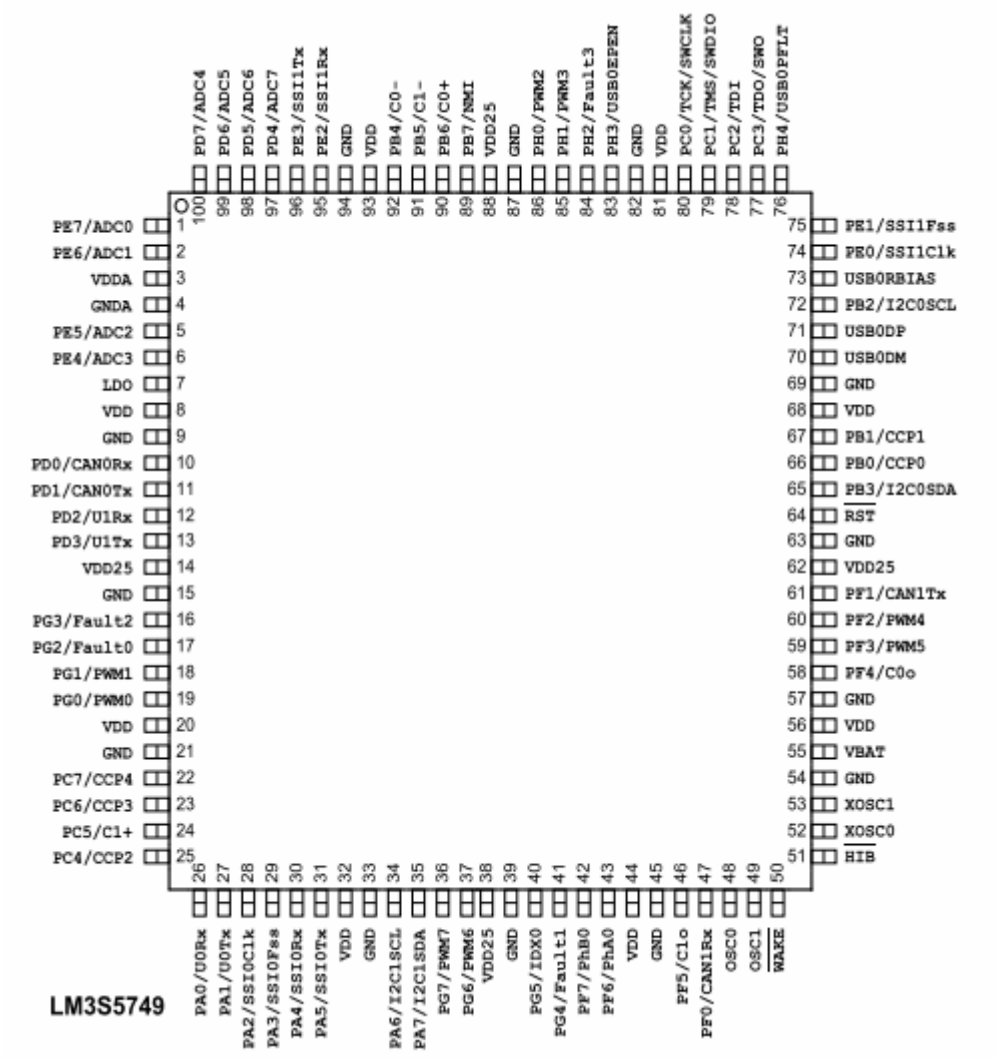
TEL:15882446438

E-mail:paulhyde@126.com

第二章 LM3S USB 处理器

2.1 LM3S 处理器简介

Luminary Micr 公司 Stellaris 所提供一系列的微控制器是首款基于 Cortex-m3 的控制器，它们为对成本尤其敏感的嵌入式微控制器应用方案带来了高性能的 32 位运算能力。 这些具备领先技术的芯片使用户能够以传统的 8 位和 16 位器件的价位来享受 32 位的性能，而且所有型号都是以小占位面积的封装形式提供。



Stellaris 系列芯片能够提供高效的性能、广泛的集成功能以及按照要求定位的选择，适用于各种关注成本并明确要求具有的过程控制以及连接能力的应用方案。LM3S3000、LM3S5000 系列，支持最大主频为 50 MHz，128 KByte FLASH,32~64 KByte SRAM，LQFP-64/LQFP-100 封装，集成 CAN 控制器、睡眠模块、正交编码器、ADC、带死区 PWM、温度传感器、模拟比较器、UART、SSI、通用定时器，I2C、CCP、DMA 控制器等外设，芯片内部固化驱动库，支持 USB Host/Device/OTG 功能，可运行在全速和低速模式，它符合 USB2.0 标准，包含挂起和唤醒信号。它包含 32 个端点，其中包含 2 个用于控制传输的专用连接端点(一个用于输入，一个用于输出)，其他 30 个端点带有可软件动态定义大小的 FIFO 并以支持多包队列。FIFO 支持 μDMA，可有效降低系统资源的占用。USB Device 启动方式灵活，可软件控制是否在启动时连接。USB 控制器遵从 OTG 标准的会话请求协议(SRP)和主机协商协议(HNP)。

Stellaris USB 模块特性：

a. 符合 USB-IF 认证标准

b. 支持 USB 2.0 全速模式(12 Mbps) 和低速模式(1.5 Mbps)

c. 集成 PHY

d. 传输类型：控制传输(Control)，中断传输(Interrupt)，批量传输(Bulk)，等时传输(Isochronous)

e.32 端点:1 个专用的输入控制端点和 1 个专用输出控制端点；15 个可配置的输入端点和 15 个可配置的输出端点。

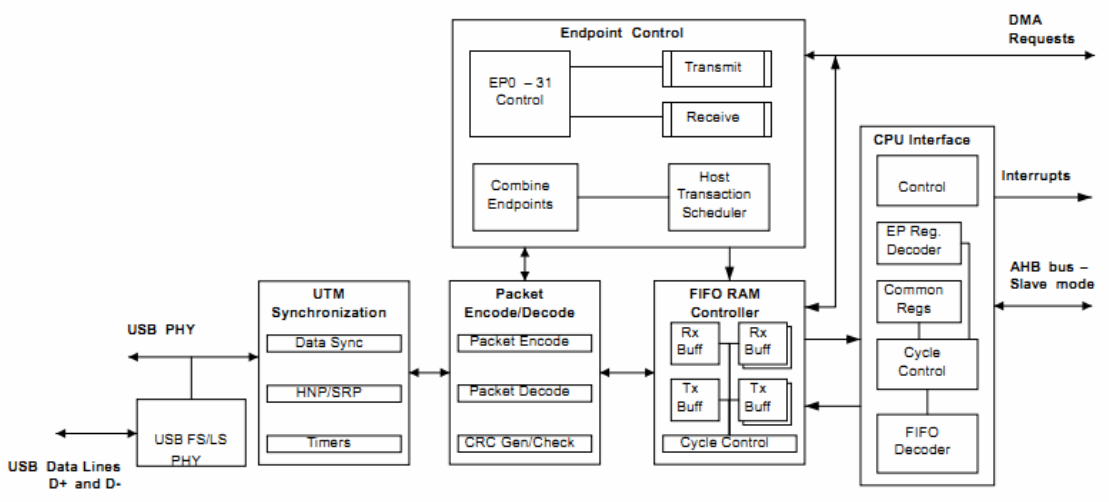
f.4 KB 专用端点内存空间：可支持双缓存的 1023 字节最大包长的等时传输。

g.支持 VBUS 电压浮动(droop)和有效 ID 检测，并产生中断信号

h.高效传输 μDMA :用于发送和接收的独立通道多达 3 个输入端点和 3 个输出端点，当 FIFO 中包含需要的大量数据时，触发通道请求。

本书主要讲 Stellaris USB 处理器的相关使用与 USB 协议，有关 LM3S 系列处理器其它外设操作与使用请参考《嵌入式系统原理与应用—基于 Cortex-m3 和 uc/os-II》，ISBN：978564709310，电子科技大学出版社。

2.2 Stellaris USB 模块



USB 模块框图

一些 USB 控制器的信号是 GPIO 的复用功能，这些管脚在复位时默认设置为 GPIO 信号。当需要使用 USB 功能时，应将相关 GPIO 备选功能选择寄存器（GPIOAFSEL）中的 AFSEL 位置位，表示启用 GPIO 的备选功能；同时还应将括号内的数字写入 GPIO 端口控制寄存器（GPIOCTRL）的 PMCn 位域，表示将 USB 信号分配给指定的 GPIO 管脚。USB0VBUS 和 USB0ID 信号通过清除 GPIO 数字使能寄存器(GPIODEN)中相应的 DEN 位来配置。关于配置 GPIO 的详细信息，请参阅《嵌入式系统原理与应用—基于 Cortex-m3 和 uc/os-II》（ISBN：978564709310，电子科技大学出版社）及相关数据手册。

当用于 OTG 模式时，由于 USB0VBUS 和 USB0ID 是 USB 专用的管脚，不需要配置，直接连接到 USB 连接器的 VBUS 和 ID 信号。如果 USB 控制器专用于主机或设备，USB 通用控制和状态寄存器(USBGPCS)中的 DEVMODOTG 和 DEVMOD 位用于连接 USB0VBUS 和 USB0ID 到内部固定电平，释放 PB0 和 PB1 管脚用于通用 GPIO。当用作自供电的设备时，需要监测 VBUS 值，来确定主机是否断开 VBUS，从而禁止自供电设备 D+/D-上的上拉电阻。此功能可通过将一个标准 GPIO 连接到 VBUS 实现。

2.2.1 功能描述

在管脚 USB0RBIAS 和地之间需要接一个 1%精度的 9.1 kΩ 电阻，且该电阻离 USB0RBIAS 管脚越近越好。由于损耗在该电阻上的功率很小，可以采用贴片电阻。Stellaris USB 支持

OTG 标准的回话请求协议(SRP)和主机协商协议(HNP)，提供完整的 OTG 协商。回话请求协议(SRP)允许连接在 USB 线缆 B 端的 B 类设备向连接在 A 端的 A 类设备发出请求，通知 A 类设备打开 VBUS 电源。主机协商协议(HNP)用于在初始会话请求协议提供供电后，决定 USB 线缆哪端的设备作为 USB Host 主控制器。当连接到非 OTG 外设或设备时，OTG 控制器可以探测出线缆的另一端接入是 USB Host 主机还是 Device 设备，并通过一个寄存器指示 OTG 运行在 Host 主机还是 Device 设备角色，上述过程是 USB 控制器自动完成的。基于这种自动探测机制，系统使用 A 类/B 类连接器取代 AB 类连接器，可支持与另外的 OTG 设备完整的 OTG 协商。

另外，USB 控制器支持接入非 OTG 外设或 Host 主控制器。它可以被设置为专用于 Host 或 Device 功能，此时 USB0VBUS 和 USB0ID 管脚可被设置作为的 GPIO 口使用。当 USB 控制器被用作自供电的 Device 时，必须将 VBUS 接到 GPIO 或模拟比较器的输入，在 VBUS 掉电时产生中断，用于关闭 USB0DP 的上拉电阻。

注意： 当使用 USB 模块时，系统必须运行在 20MHz 频率或以上。

2.2.2 作为设备

USB 控制器作为 USB 设备(Device)操作时，输入事务通过使用端点的发送端点寄存器，由端点的发送接口进行控制。输出事务通过使用端点的接收端点寄存器，由端点的接收接口进行控制。当配置端点的 FIFO 大小时，需要考虑最大数据包大小。

a.批量传输：批量端点的 FIFO 可配置为最大包长(最大 64 字节)，如果使用双包缓存，则需要配置为 2 倍于最大包长大小(后面的章节将详细描述)。

b.中断传输：中断端点的 FIFO 可配置为最大包长(最大 64 字节)，如果使用双包缓存，则需要配置为 2 倍于最大包长大小。

c.等时传输：等时端点的 FIFO 比较灵活，最大支持 1023 字节。

d.控制传输：USB 设备可能指定一个独立的控制端点，但在大多数情况，USB 设备使用 USB 控制器的端点 0 作为专用的控制端点。

① 端点

USB 控制器作为设备运行时，提供两个专用的控制端点(输入和输出)和用于与主机通讯的 30 个可配置的端点(15 输入和 15 输出) 。端点的端点号和方向与对应的相关寄存器有直接联系。比如，当主机发送到端点 1，所有的配置和数据存在于端点 1 发送寄存器接口中。端点 0 是专用的控制端点,用于枚举期间的端点 0 的所有控制传输或其他端点 0 的控制请求。端点 0 使用 USB 控制器 FIFO 内存(RAM)的前 64 字节，此内存对于输入事务和输出事务是共享的。其余 30 个端点可配置为控制端点、批量端点、中断端点或等时端点。他们应被作为 15 个可配置的输入端点和 15 个可配置的输出端点来对待。这些成对的端点的输入和输出端点不需要必须配置为相同类型，比如端点对(endpoint pairs)的输出部分可以设置为批量端点，而输入部分可以设置为中断端点。每个端点的 FIFO 的地址和大小可以根据应用需求来修改。

② 输入事务

输入事务是相对主机而言的，输入事务的数据通过发送端点的 FIFO 来处理。15 个可配置的输入端点的 FIFO 大小由 USB 发送 FIFO 起始地址寄存器(USBTXFIFOADD)决定，传输时发送端点 FIFO 中的最大数据包大小可编程配置，该大小由写入该端点的 USB 端点 n 最大发送数据寄存器 (USBTXMAXPn)中的值决定，USBTXMAXPn 值不能大于 FIFO 的大小。端点的 FIFO 可配置为双包缓存或单包缓存，当双包缓存使能时， FIFO 中可缓冲两个数据包，这需要 FIFO 至少为两个数据包大小。当不使用双包缓存时，即使数据包的大小小于 FIFO 大小的一半，也只能缓冲一个数据包。

单包缓冲：如果发送端点 FIFO 的大小小于该端点最大包长的两倍时(由 USB 发送动态 FIFO 大小寄存器 USBTXFIFOSZ 设定)，只能使用单包缓冲，在 FIFO 中缓冲一个数据包。当数据包已装载到 TXFIFO 中时，USB 端点 n 发送控制和状态低字节寄存器 USBTXCSRLn 中的 TXRDY 位必须被置位，如果 USB 端点 n 发送控制和状态高字节寄存器 USBTXCSRHn 中的 AUTOSET 位被置 1，TXRDY 位将在最大包长的包装载到 FIFO 中时自动置位；如果数据包小于最大包长，位 TXRDY 必须手动置位。当 TXRDY 位被手动或自动置 1 时，表明要发送的数据包已准备好。如果数据包成功发送，TXRDY 位和 FIFONE 位将被清 0，同时产生相应的中断信号，此时下一包数据可装载到 FIFO 中。

双包缓存：如果发送端点 FIFO 的大小至少两倍于该端点最大包长时，允许使用双包缓存，FIFO 中可以缓冲两个数据包。当数据包已装载到 TXFIFO 中时，USBTXCSRLn 中的 TXRDY 位必须被置位，如果寄存器 USBTXCSRHn 中的 AUTOSET 位被置 1，TXRDY 位将在最大包长的包装载到 FIFO 中时自动置位；如果数据包小于最大包长，位 TXRDY 必须手动置位。当 TXRDY 位被手动或自动置 1 时，表明要发送的数据包已准备好。在装载完第一个包后，TXRDY 位立即清除，同时产生中断信号；此时第二个数据包可装载到 TXFIFO 中，TXRDY 位重新置位(手动或自动)，此时，两个要发送的包都已准备好，如果任一数据包成功发送，TXRDY 位和 FIFONE 位将被清 0，同时产生相应的发送端点中断信号，此时下一包数据可装载到 TXFIFO 中。寄存器 USBTXCSRLn 中的 FIFONE 位的状态表明此时可以装载几个包，如果 FIFONE 位置 1，表明 FIFO 中还有一个包未发送，只能装载一个数据包；如果 FIFONE 位为 0，表明 FIFO 中没有未发送的包，可以装载两个数据包。

如果 USB 发送双包缓存禁止寄存器 USBTXDPKTBUFDIS 中的 EPn 位置位，相应的端点禁止双包缓存。此位缺省为置 1，需要使能双包缓存时必须清 0 该位。

③ 输出事务

输出事务是相对主机而言的，输出事务的数据通过接收端点的 FIFO 来处理。15 个可配置的输出端点的 FIFO 大小由 USB 接收 FIFO 起始地址寄存器(USBRXFIFOADD)决定，传输时接收端点 FIFO 中的最大数据包大小可编程配置，该大小由写入该端点的 USB 端点 n 最大接收数据寄存器 (USBRXMAXPn)中的值决定。端点的 FIFO 可配置为双包缓存或单包缓存，当双包缓存使能时 FIFO 中可缓冲两个数据包。当不使用双包缓存时，即使数据包的大小小于 FIFO 大小的一半，也只能缓冲一个数据包。

单包缓存：如果接收端点 FIFO 的大小小于该端点最大包长的两倍时，只能使用单包缓冲，在 FIFO 中缓冲一个数据包。当数据包已接收到 RXFIFO 中时，USB 端点 n 接收控制和状态低字节寄存器 USBRXCSRLn 中的 RXRDY 和 FULL 位置位，同时发出相应的中断信号，表明接收 FIFO 中有一个数据包需要读出。当数据包从 FIFO 中读出时，RXRDY 位必须被清 0 以允许接收后面的数据包，同时向 USB 主机发送确认信号。如果 USB 端点 n 接收控制和状态高字节寄存器 USBRXCSRHn 中的 AUTOCl 位被置 1，RXRDY 和 FULL 位将在最大包长的包从 FIFO 中读出时自动清 0；如果数据包小于最大包长，位 RXRDY 必须手动清 0。

双包缓存：如果接收端点的 FIFO 大小不小于该端点的最大包长的 2 倍时，可以使用双缓冲机制缓存两个数据包。当第一个数据包被接收缓存到 RXFIFO，寄存器 USBRXCSRLn 中的 RXRDY 位置位，同时产生相应的接收端点中断信号，指示有一个数据包需要从 RXFIFO 中读出。当第一个数据包被接收时，寄存器 USBRXCSRLn 的 FULL 位不置位，该位只有在第二个数据包被接收缓存到 FIFO 时才置位。当从 FIFO 从读出一个包时，RXRDY 位必须清 0 以允许接收后面的包。如果 USB 端点 n 接收控制和状态高字节寄存器 USBRXCSRHn 中的 AUTOCl 位被置 1，RXRDY 位将在最大包长的包从 FIFO 中读出时自动清 0；如果数据包小于最大包长，RXRDY 位必须手动清 0。当 RXRDY 位清 0 时，FULL 位为 1，USB 控制器先清除 FULL 位，然后再置位 RXRDY 位，表明 FIFO 中的另一个数据包等待被读出。

如果 USB 接收双包缓存禁止寄存器 USBRXDPKTBUFDIS 中的 EPn 位置位，相应的端点禁止双包缓存。此位缺省为置 1，需要使能双包缓存时必须清 0 该位。

④ 调度

传输事务由 Host 主机控制器调度决定，Device 设备无法控制事务调度。设备等待 Host 主控制器发出请求，随时可建立传输事务。当传输事务完成或由于某些原因被终止时，会产生中断信号。当 Host 主控制器发起请求，而 Device 设备还没有准备好，设备会返回一个 NAK 忙信号。

⑤ 设备挂起(SUSPEND)

USB 总线空闲达 3ms 时，USB 控制器自动进入挂起(SUSPEND)模式。如果 USB 中断使能寄存器 USBIE 中使能挂起(SUSPEND)中断，此时会发出一个中断信号。当 USB 控制器进入挂起模式，USB PHY 也将进入挂起模式。当检测到唤醒(RESUME)信号时，USB 控制器退出挂起模式，同时使 USB PHY 退出挂起模式，此时如果唤醒中断使能，将产生中断信号。设置 USB 电源寄存器 USBPOWER 中的 RESUME 位同样可以强制 USB 控制器退出挂起模式。当此位置位，USB 控制器退出挂起模式，同时在总线上发出唤醒信号。RESUME 位必须在 10ms(最大 15ms)后清 0 来结束唤醒信号。为满足电源功耗需求，LM3S USB 控制器可进入深睡眠模式。

⑥ 帧起始

当 USB 控制器运行在设备模式，它每 1ms 收到一次主机发出的帧起始包(SOF)。当收到 SOF 包时，包中包含的 11 位帧号写入 USB 帧值寄存器 USBFRAME 中，同时发出 SOF 中断信号，由应用程序处理。一旦 USB 控制器开始收到 SOF 包，它将预期每 1ms 收到 1 次。如果超过 1.00358ms 没有收到 SOF 包，将假定此包丢失，寄存器 USBFRAME 也将不更新。当 SOF 包重新成功接收时，USB 控制器继续，并重新同步这些脉冲。

使用 SOF 中断可以每 1ms 获得一个基本时钟，可当 tick 使用，并且使用方便。后面会有重要应用。

⑥ USB 复位

当 USB 控制器处于设备模式，如果检测到 USB 总线上复位信号，USB 控制将自动：清除寄存器 USBFADDR，清除 USB 端点索引寄存器(USBEPIDX)，清空所有端点 FIFO，清除所有控制/状态寄存器，使能所有端点中断，产生复位中断信号。刚接入到主机的 USB 设备，USB 复位意味着要进行枚举了。

2.2.3 作为主机

当 Stellaris USB 控制器运行在主机模式时，可与其他 USB 设备的点对点通讯，也可用于连接到集线器，与多个设备进行通讯。USB 控制器支持全速和低速设备。它自动执行必要的事务传输，允许 USB 2.0 集线器使用低速设备和全速设备。支持控制传输、批量传输、等时传输和中断传输。输入事务由端点的接收接口进行控制；输出事务使用端点的发送端点寄存器。当配置端点的 FIFO 大小时，需要考虑最大数据包大小。

① 端点

端点寄存器用于控制 USB 端点接口，通过接口可与设备进行通讯。主机端点由 1 个专用控制输入端点、1 个专用控制输出端点、15 个可配置的输出端点和 15 个可配置的输入端点组成。控制端点只能与设备的端点 0 进行控制传输，用于设备枚举或其他使用设备端点 0 的控制功能。控制端点输入输出事务共享一个 FIFO 存储空间，并在 FIFO 的前 64 字节。其余输入和输出端点可配置为：控制传输端点、批量传输端点、中断传输端点或等时传输端点。输入和输出控制有成对的三组寄存器，它们可以与不同类型的端点以及不同设备的不同端点进行通讯。例如，第一对端点可分开控制，输出部分与设备的批量输出端点 1 通讯，同时输入部分与设备的中断端点 2 通讯。FIFO 的地址和大小可以软件设置，并且可以指定用于某一个端点输入或者输出传输。

无论点对点通信还是集线器通信，在访问设备之前，必须设置端点 n 的接收功能地址寄存器 USBRXFUNCADDRn 和端点 n 的发送功能地址寄存器 USBTXFUNCADDRn。LM3S 系列 USB 控制器支持通过集线器连接设备，一个寄存器就可以实现，该寄存器说明集线器地址和每个传输的端口。在本书中的所有例程都未使用集线器访问设备。

② 输入事务

输入事务，相对主机而言是数据输入，与设备输出事务类似的，但传输数据必须通过设置寄存器 USBCSRL0 中的 REQPKT 位开始，向事务调度表明此端点存在一个活动的传输。此时事务调度向目标设备发送一个输入令牌包。当主机 RXFIFO 中接收到数据包时，寄存器 USBCSRL0 的 RXRDY 位置位，同时产生相应的接收端点中断信号，指示 RXFIFO 中有数据包需要读出。

当数据包被读出时，RXRDY 位必须清 0。寄存器 USBRXCSRHn 中的 AUTOCL 位可用于当最大包长的包从 RXFIFO 中读出时将 RXRDY 位自动清 0。寄存器 USBRXCSRHn 中的 AUTORQ 位用于当 RXRDY 位清 0 时将 REQPKT 位自动置位。AUTOCL 和 AUTORQ 位用于 μDMA 访问，在主处理器不干预时完成批量传输。

当 RXRDY 位清 0 时，控制器向设备发送确认信号。当传输一定数据包时，需要将端点 n 的 USBRQPKTCOUNTn 寄存器配置为要传输包数量，每一次传输， USBRQPKTCOUNTn 寄存器中的值减 1，当减到 0 时，AUTORQ 位清 0 来阻止后面试图进行的数据传输。如传输数量未知，USBRQPKTCOUNTn 寄存器必须清 0,AUTORQ 位保持置位,直到收到结束包,AUTORQ 位才清 0（小于 USBRXMAXPn 寄存器中的 MAXLOAD 值）。

用图表示传输过程序。

③ 输出事务

当数据包装载到 TXFIFO 中时， USBTXCSRLn 寄存器中的 TXRDY 位必须置位。如果置位了 USBTXCSRHn 寄存器中的 AUTOSET 位，当最大包长的数据包装载到 TXFIFO 中时，TXRDY 位自动置位。此外，AUTOSET 位与 μDMA 控制器配合使用，可以在不需要软件干预的情况下完成批量传输。

④ 调度

调度由 USB 主机控制器自动处理。中断传输可以是每 1 帧进行一次，也可以每 255 帧进行一次，可以在 1 帧到 255 之间以 1 帧增量调度。批量端点不允许调度参数，但在设备的端点不响应时，允许 NAK 超时。等时端点可以在每 1 帧到每 216 帧之间调度(2 的幂)。

USB 控制器维持帧计数，并发送 SOF 包，SOF 包发送后，USB 主机控制器检查所有配置好的端点，寻找激活的传输事务。REQPKT 位置位的接收端点或 TXRDY 或 FIFONE 位置位的发送端点，被视为存在激活的传输事务。

如果传输建立在一帧的第一个调度周期，而且端点的间隔计数器减到 0，则等时传输和中断传输开始。所以每个端点的中断传输和等时传输每 N 帧才发生一次，N 是通过 USB 主机端点 n 的 USBTXINTERVALn 寄存器或 USB 主机端点 n 的 USBRXINTERVALn 寄存器设置的间隔。

如果在帧中下一个 SOF 包之前有足够的提供足够的时间完成传输，则激活的批量传输立即开始。如果传输需要重发时(例如，收到 NAK 或设备未响应)，需要在调度器先检查完其他所有端点是否有激活的传输之后，传输才能重传。这保证了一个发送大量 NAK 响应的端点不阻塞总线上的其他传输正常进行。

⑤ USB集线器

以下过程只适用于 USB2.0 集线器的主机。当低速设备或全速设备通过 USB 2.0 集线器连接到 USB 主机时，集线器地址和端口信息必须记录在相应的 USB 端点 n 的 USBRXHUBADDRn 寄存器和 USB 端点 n 的 USBRXHUBPORTn 寄存器或者 USB 端点 n 的 USBTXHUBADDRn 寄存器和 USB 端点 n 的 USBTXHUBPORTn 寄存器。

此外，设备的运行速度(全速或低速)必须记录在 USB 端点 0 的 USBTYPE0 寄存器，和设

备访问主机 USB 端点 n 的 USBTXTYPEn 寄存器，或者主机 USB 端点 n 的 USBRXTYPEn 寄存器。

对于集线器通讯，这些寄存器的设置记录了 USB 设备当前相应端点的配置。为了支持更多数量的设备，USB 主机控制器允许通过更新这些寄存器配置来实现。

2.2.4 OTG 模式

OTG 就是 On The Go，正在进行中的意思，可以进行“主机与设备”模式切换。USB OTG 允许使用时才给 VBUS 上电，不使用 USB 总线时，则关断 VBUS 。VBUS 由总线上的 A 设备提供电源。OTG 控制器通过 PHY 采样 ID 输入信号分辨 A 设备和 B 设备。ID 信号拉低时，检测到插入 A 设备(表示 OTG 控制器作为 A 设备角色)；ID 信号为高时，检测到插入 B 设备(表示 OTG 控制器作为 B 设备角色)。注意当在 OTG A 和 OTG B 之间切换时，控制器保留所有的寄存器内容。关于 OTG 使用不在本书重点介绍。

2.3 寄存器描述

使用 USB 处理器进行 USB 主机、设备开发离不开相关寄存器操作，USB 本身就相当复杂，寄存器相当多，下面就几个常用的寄存器进行介绍。本节主要讲主机与设备模式下的寄存器使用，关于 GTO 模式下寄存器使用不具体讲解，因为 GTO A 设备相当于主机，GTO B 设备相当于设备。USB 寄存器如下表：

寄存器名称	类型	复位值	寄存器描述
USBFADDR	R/W	0x00	USB 地址
USBPOWER	R/W	0x20	USB 电源
USBTXIS	RO	0x0000	发送中断状态
USBRXIS	RO	0x0000	接收中断状态
USBTXIE	R/W	0xFFFF	发送中断使能
USBRXIE	R/W	0xFFFE	接收中断使能
USBIS	RO	0x00	USB 处理模块中断状态
USBIE	R/W	0x06	USB 处理模块中断使能
USBFRAME	RO	0x0000	USB 帧值
USBEPIDX	R/W	0x00	端点索引
USBTEST	R/W	0x00	测试模式
USBFIFOn	R/W	0x0000.0000	端点 n FIFO
USBDEVCTL	R/W	0x80	设备控制
USBTXFIFOSZ	R/W	0x00	动态 TXFIFO 大小
USBRXFIFOSZ	R/W	0x00	动态 RXFIFO 大小
USBTXFIFOADD	R/W	0x0000	动态 TXFIFO 起始地址
USBRXFIFOADD	R/W	0x0000	动态 RXFIFO 起始地址
USBCONTIM	R/W	0x5C	USB 连接时序
USBVPLEN	R/W	0x3C	USB OTG VBUS 脉冲时序
USBFSEOF	R/W	0x77	USB 全速模式下最后的传输与帧结束时序
USBLSEOF	R/W	0x72	USB 低速模式下最后的传输与帧结束时序

USBTXFUNCADDRn	R/W	0x00	发送端点 n 功能地址
USBTXHUBADDRn	R/W	0x00	发送端点 n 集线器(Hub)地址
USBTXHUBPORTn	R/W	0x00	发送端点 n 集线器(Hub)端口
USBRXFUNCADDRn	R/W	0x00	接收端点 n 功能地址（n 不为 0）
USBRXHUBADDRn	R/W	0x00	接收端点 n 集线器(Hub)地址（n 不为 0）
USBRXHUBPORTn	R/W	0x00	接收端点 n 集线器(Hub)端口（n 不为 0）
USBCSRL0	W1C	0x00	端点 0 控制和状态低字节
USBCSRH0	W1C	0x00	端点 0 控制和状态高字节
USBCOUNT0	RO	0x00	端点 0 接收字节数量
USBTYPE0	R/W	0x00	端点 0 类型
USBNAKLMT	R/W	0x00	USB NAK 限制
USBTXMAXPn	R/W	0x0000	发送端点 n 最大传输数据
USBTXCSRLn	R/W	0x00	发送端点 n 控制和状态低字节
USBTXCSRHn	R/W	0x00	发送端点 n 控制和状态高字节
USBRXMAXPn	R/W	0x0000	接收端点 n 最大传输数据
USBRXCSRLn	R/W	0x00	接收端点 n 控制和状态低字节
USBRXCSRHn	R/W	0x00	接收端点 n 控制和状态高字节
USBRXCOUNTn	RO	0x0000	端点 n 接收字节数量
USBTXTYPEn	R/W	0x00	端点 n 主机发送配置类型
USBTXINTERVALn	R/W	0x00	端点 n 主机发送间隔
USBRXTYPEn	R/W	0x00	端点 n 主机配置接收类型
USBRXINTERVALn	R/W	0x00	端点 n 主机接收巡检间隔
USBRQPKTCOUNTn	R/W	0x0000	端点 n 块传输中请求包数量
USBRXDPKTBUFDIS	R/W	0x0000	接收双包缓存禁止
USBTXDPKTBUFDIS	R/W	0x0000	发送双包缓存禁止
USBEPC	R/W	0x0000.0000	USB 外部电源控制
USBEPCRIS	RO	0x0000.0000	USB 外部电源控制原始中断状态
USBEPCIM	R/W	0x0000.0000	USB 外部电源控制中断屏蔽
USBEPCISC	R/W	0x0000.0000	USB 外部电源控制中断状态和清除
USBDRRIS	RO	0x0000.0000	USB 设备唤醒(RESUME)原始中断状态
USBDRIM	R/W	0x0000.0000	USB 设备唤醒(RESUME)中断屏蔽
USBDRISC	W1C	0x0000.0000	USB 设备唤醒(RESUME)中断状态和清除
USBGPCS	R/W	0x0000.0000	USB 通用控制和状态
USBVDC	R/W	0x0000.0000	VBUS 浮动控制
USBVDCRIS	RO	0x0000.0000	VBUS 浮动控制原始中断状态
USBVDCIM	R/W	0x0000.0000	VBUS 浮动控制中断屏蔽
USBVDCISC	R/W	0x0000.0000	VBUS 浮动控制中断状态\清除

USBIDVRIS	RO	0x0000.0000	ID 有效检测原始中断状态
USBIDVIM	R/W	0x0000.0000	ID 有效检测中断屏蔽
USBIDVISC	R/W1C	0x0000.0000	ID 有效检测中断状态与清除
USBDMASEL	R/W	0x0033.2211	DMA 选择

表 1. USB 寄存器

2.3.1 控制状态寄存器

USBFADDR，设备地址寄存器。包含 7 位设备地址，将通过 SET ADDRESS(USBREQ_SET_ADDRESS)请求收到的地址（pUSBRequest->wValue 值）写入该寄存器。USBFADDR 寄存器描述如下表：

位	名称	类型	复位	描述
8	保留	RO	0	保持不变
[7:0]	FUNCADDR	R/W	0	设备地址

表 2. USBFADDR 寄存器

例如，在枚举时会用到设置 USB 设备地址：

```
void USBDevAddrSet(unsigned long ulBase, unsigned long ulAddress)
{
    HWREGB(ulBase + USB_O_FADDR) = (unsigned char)ulAddress;
}
```

其中 ulBase 为设备基地址，LM3S USB 处理器一般只有一个 USB 模块，所以只有 USB0_BASE（0x40050000）；ulAddress 为主机 USBREQ_SET_ADDRESS 请求命令时 tUSBRequest.wValue 值。

USBPOWER，USB 电源控制寄存器，8 位。主机模式下，USBPOWER 寄存器描述如下表：

位	名称	类型	复位	描述
[7:4]	保留	RO	0x02	保持不变
3	RESET	R/W	0	总线复位
2	RESUME	R/W	0	总线唤醒，置位后 20mS 后软件清除
1	SUSPEND	R/W1S	0	总线挂起
0	PWRDNPHY	R/W	0	PHY 掉电

表 2. USBPOWER 寄存器

USBPOWER，USB 电源控制寄存器，8 位。设备模式下，USBPOWER 寄存器描述如下表：

位	名称	类型	复位	描述
[5:4]	保留	RO	0x02	保持不变
7	ISOUP	R/W	0	ISO 传输时，TXRDY 置位时，收到输入令牌包，将发送 0 长度的数据包。
6	SOFTCONN	R/W	0	软件连接/断开 USB
3	RESET	RO	0	总线复位
2	RESUME	R/W	0	总线唤醒，置位后 10mS 后软件清除
1	SUSPEND	RO	0	总线挂起
0	PWRDNPHY	R/W	0	PHY 掉电

表 3. USBPOWER 寄存器

```
#define USB_POWER_ISOUP         0x00000080  // Isochronous Update
#define USB_POWER_SOFTCONN      0x00000040  // Soft Connect/Disconnect
#define USB_POWER_RESET         0x00000008  // RESET Signaling
#define USB_POWER_RESUME        0x00000004  // RESUME Signaling
#define USB_POWER_SUSPEND       0x00000002  // SUSPEND Mode
#define USB_POWER_PWRDNPHY      0x00000001  // Power Down PHY
```

例如：在主机模式下，把 USB 总线挂起。

```
void USBHostSuspend(unsigned long ulBase)
{
    //ulBase 为设备基地址
    HWREGB(ulBase + USB_O_POWER) |= USB_POWER_SUSPEND;
}
```

例如：在主机模式下，复位 USB 总线。

```
void USBHostReset(unsigned long ulBase, tBoolean bStart)
{
    //ulBase 为设备基地址，bStart 确定复位开始/结束。
    if(bStart)
    {
        HWREGB(ulBase + USB_O_POWER) |= USB_POWER_RESET;
    }
    else
    {
        HWREGB(ulBase + USB_O_POWER) &= ~USB_POWER_RESET;
    }
}
```

例如：在主机模式下，唤醒 USB 总线，唤醒开始 20ms 后必须手动结束。

```
void USBHostResume(unsigned long ulBase, tBoolean bStart)
{
    //ulBase 为设备基地址，bStart 确定唤醒开始/结束。
    if(bStart)
    {
        HWREGB(ulBase + USB_O_POWER) |= USB_POWER_RESUME;
    }
    else
    {
        HWREGB(ulBase + USB_O_POWER) &= ~USB_POWER_RESUME;
    }
}
```

例如：在设备模式下，设备连接/断开主机。

```
void USBDevConnect(unsigned long ulBase)
{
    //ulBase 为设备基地址。
    HWREGB(ulBase + USB_O_POWER) |= USB_POWER_SOFTCONN;
}
void USBDevDisconnect(unsigned long ulBase)
{
    //ulBase 为设备基地址。
    HWREGB(ulBase + USB_O_POWER) &= (~USB_POWER_SOFTCONN);
}
```

USBFRAME，16 位只读寄存器，保存最新收到的帧编号。USBFRAME 寄存器描述如下表：

位	名称	类型	复位	描述
[15:11]	保留	RO	0	保持不变
[10:0]	FRAME	RO	0	帧编号

表 4. USBFRAME 寄存器

例如：获取最新帧编号。

```
unsigned long USBFrameNumberGet(unsigned long ulBase)
{
    //ulBase 为设备基地址。
    return(HWREGH(ulBase + USB_O_FRAME));
}
```

USBTEST，测试模式，回应 USBREQ_SET_FEATURE 的 USB_FEATURE_TEST_MODE 命令，使 USB 控制器进入测试模式，任何时候，只能有一位被设置，但不用于正常操作。在主机模式下，USBTEST 寄存器描述如下表：

位	名称	类型	复位	描述
[4:0]	保留	RO	0	保持不变
7	FORCEH	R/W	0	强制为主机模式
6	FIFOACC	R/W1S	0	将端点 0 的 TXFIFO 数据传送到 RXFIFO
5	FORCEFS	R/W	0	强制全速模式

表 5. USBTEST 寄存器

在设备模式下，USBTEST 寄存器描述如下表：

位	名称	类型	复位	描述
[4:0]、7	保留	RO	0	保持不变
6	FIFOACC	R/W1S	0	将端点 0 的 TXFIFO 数据传送到 RXFIFO
5	FORCEFS	R/W	0	强制全速模式

表 6. USBTEST 寄存器

USBDEVCTL，USB 设备控制器，提供 USB 控制器当前状态信息，可指示接入设备是全速还是低速。USBDEVCTL 寄存器描述如下表：

位	名称	类型	复位	描述
7	DEV	RO	1	设备模式，OTG A 端。
6	FSDEV	RO	0	全速设备
5	LSDEV	RO	0	低速设备
[4:3]	VBUS	RO	0	VBUS 电平
2	HOST	RO	0	Host 主机模式
1	HOSTREQ	R/W	0	主机请求
0	SESSION	R/W	0	会话开始/结束

表 7. USBDEVCTL 寄存器

```
#define USB_DEVCTL_DEV          0x00000080  // Device Mode
#define USB_DEVCTL_FSDEV        0x00000040  // Full-Speed Device Detected
#define USB_DEVCTL_LSDEV        0x00000020  // Low-Speed Device Detected
#define USB_DEVCTL_VBUS_M       0x00000018  // VBUS Level
#define USB_DEVCTL_VBUS_NONE    0x00000000  // Below SessionEnd
#define USB_DEVCTL_VBUS_SEND    0x00000008  // Above SessionEnd, below AValid
#define USB_DEVCTL_VBUS_AVALID  0x00000010  // Above AValid, below VBUSValid
#define USB_DEVCTL_VBUS_VALID   0x00000018  // Above VBUSValid
#define USB_DEVCTL_HOST         0x00000004  // Host Mode
#define USB_DEVCTL_HOSTREQ      0x00000002  // Host Request
#define USB_DEVCTL_SESSION      0x00000001  // Session Start/End
```

例如：获取当前 USB 工作模式。

```
unsigned long USBModeGet(unsigned long ulBase)
{
    return(HWREGB(ulBase + USB_O_DEVCTL) &
           (USB_DEVCTL_DEV | USB_DEVCTL_HOST | USB_DEVCTL_SESSION |
            USB_DEVCTL_VBUS_M));
}
```

返回下面参数：

```
#define USB_DUAL_MODE_HOST      0x00000001
#define USB_DUAL_MODE_DEVICE    0x00000081
#define USB_DUAL_MODE_NONE      0x00000080
#define USB_OTG_MODE_ASIDE_HOST 0x0000001d
```

```
#define USB_OTG_MODE_ASIDE_NPWR 0x00000001
#define USB_OTG_MODE_ASIDE_SESS 0x00000009
#define USB_OTG_MODE_ASIDE_AVAL 0x00000011
#define USB_OTG_MODE_ASIDE_DEV  0x00000019
#define USB_OTG_MODE_BSIDE_HOST 0x0000009d
#define USB_OTG_MODE_BSIDE_DEV  0x00000099
#define USB_OTG_MODE_BSIDE_NPWR 0x00000081
#define USB_OTG_MODE_NONE       0x00000080
```

USBCONTIM，连接时序控制寄存器，用于控制 USB 连接。USBCONTIM 寄存器描述如下表：

位	名称	类型	复位	描述
[7:4]	WTCON	R/W	0x5	连接等待
[3:0]	WTID	R/W	0xC	ID 等待

表 8. USBCONTIM 寄存器

WTCON，此位域可按需求配置等待延时，满足连接/断开的滤波需求；单位为 533.3ns，复位后为 0x5，默认为 2.667μs。

WTID，此位域配置 ID 等待延时，等待 ID 值有效时才使能 OTG 的 ID 检测。单位为 4.369 ms，复位后为 0xC，默认为 52.43 ms。

USBVPLEN，VBUS 脉冲时序配置寄存器，控制 VBUS 脉冲充电的持续时间。USBVPLEN 寄存器描述如下表：

位	名称	类型	复位	描述
[7:0]	VPLEN	R/W	0x3c	VBUS 脉冲宽度

表 9. USBVPLEN 寄存器

VPLEN，用于配置 VBUS 脉冲充电的持续时间，单位 546.1 μs，默认为 32.77 ms。

USBFSEOF，用于配置全速模式下最后传输开始与帧结束(EOF)之间的最小时间间隔。USBFSEOF 寄存器描述如下表：

位	名称	类型	复位	描述
[7:0]	FSEOFG	R/W	0x77	全速模式帧间隙

表 10. USBFSEOF 寄存器

FSEOFG，在全速传输中用于配置最后的传输与帧结束之间的最小时间间隙，单位 533.3ns，默认为 63.46 μs。

USBLSEOF，用于配置低速模式下最后传输开始与帧结束(EOF)之间的最小时间间隔。USBLSEOF 寄存器描述如下表：

位	名称	类型	复位	描述
[7:0]	LSEOFG	R/W	0x72	低速模式帧间隙

表 11. USBLSEOF 寄存器

FSEOFG，在低速传输中用于配置最后的传输与帧结束之间的最小时间间隙，单位 1.067 μs，默认为 121.6 μs。

注意：USBCONTIM、USBVPLEN、USBFSEOF、USBLSEOF 一般不用配置，保持默认情况就能满足 USB 通信。

USBGPCS，USB 通用控制状态寄存器，提供内部 ID 信号的状态。

USBGPCS 寄存器描述如下表：

位	名称	类型	复位	描述
[31:2]	保留	RO	0	保持不变

1	DEVMODOTG	R/W	0	使能 ID 控制模式
0	DEVMOD	R/W	0	模式控制

表 12. USBGPCS 寄存器

```
#define USB_GPCS_DEVMODOTG      0x00000002  // Enable Device Mode
#define USB_GPCS_DEVMOD         0x00000001  // Device Mode
```

例如：设置主机模式。

```
void USBHostMode(unsigned long ulBase)
{
    //ulBase 为设备基地址。
    HWREGB(ulBase + USB_O_GPCS) &= ~(USB_GPCS_DEVMOD);
}
```

2.3.2 中断控制

USBTXIS，发送中断状态寄存器，16 位只读寄存器，读取数据时，清除相应标志位。USBTXIS 寄存器描述如下表：

位	名称	类型	复位	描述
15..0	EPn	RO	0	端点发送中断标志

表 13. USBTXIS 寄存器

USBRXIS，接收中断状态寄存器。USBRXIS 寄存器描述如下表：

位	名称	类型	复位	描述
15..1	EPn	R/W	0	端点接收中断标志

表 14. USBRXIS 寄存器

USBTXIE，发送中断使能寄存器，16 位只读寄存器，读取数据时，清除相应标志位。USBTXIE 寄存器描述如下表：

位	名称	类型	复位	描述
15..0	EPn	RO	0	端点发送中断标志

表 15. USBTXIE 寄存器

USBRXIE，接收中断使能寄存器。USBRXIE 寄存器描述如下表：

位	名称	类型	复位	描述
15..1	EPn	R/W	0	端点接收中断标志

表 16. USBRXIE 寄存器

USBIS，通用中断状态寄存器，8 位只读寄存器，读取数据时，清除相应标志位。USBIS 寄存器描述如下表：

位	名称	类型	复位	描述
7	VBUSERR	RO	0	VBUS 中断（只有主机模式使用）
6	SESREQ	RO	0	会话请求中断（只有主机模式使用）
5	DISCON	RO	0	连接断开中断
4	CONN	RO	0	连接中断
3	SOF	RO	0	帧起始中断
2	BABBLE	RO	0	Babble 中断
1	RESUME	RO	0	唤醒中断
0	SUSPEND	RO	0	挂起中断（只有设备模式使用）

表 17. USBIS 寄存器

USBIE，通用中断使能寄存器，USBIE 寄存器描述如下表：

位	名称	类型	复位	描述

7	VBUSERR	RO	0	VBUS 中断使能（只有主机模式使用）
6	SESREQ	RO	0	会话请求中断使能（只有主机模式使用）
5	DISCON	RO	0	连接断开中断使能
4	CONN	RO	0	连接中断使能
3	SOF	RO	0	帧起始中断使能
2	BABBLE	RO	0	Babble 中断使能
1	RESUME	RO	0	唤醒中断使能
0	SUSPEND	RO	0	挂起中断使能（只有设备模式使用）

表 18. USBIE 寄存器

此外还有 USBEPC、USBEPCRIS、USBEPC、USBEPCRIS、USBEPCIM、USBEPCISC 电源管理中断；USBIDVISC、USBIDVIM、USBIDVRIS 等 ID 检测中断控制；USBDRRIS、USBDRIM、USBDRISC 唤醒中断控制；USBVDC、USBVDCRIS、USBVDCIM、USBVDCISC 的 VBUS Droop 控制中断。这些中断使用较少，不在此详细描述。

例如，写一组函数，控制、管理上面中断。

USBIntEnable()、USBIntDisable()的 ulFlags 参数， USBIntStatus()返回参数：

```
#define USB_INTCTRL_ALL          0x000003FF  // All control interrupt sources
#define USB_INTCTRL_STATUS       0x000000FF  // Status Interrupts
#define USB_INTCTRL_VBUS_ERR     0x00000080  // VBUS Error
#define USB_INTCTRL_SESSION      0x00000040  // Session Start Detected
#define USB_INTCTRL_SESSION_END 0x00000040  // Session End Detected
#define USB_INTCTRL_DISCONNECT   0x00000020  // Disconnect Detected
#define USB_INTCTRL_CONNECT      0x00000010  // Device Connect Detected
#define USB_INTCTRL_SOF          0x00000008  // Start of Frame Detected
#define USB_INTCTRL_BABBLE       0x00000004  // Babble signaled
#define USB_INTCTRL_RESET        0x00000004  // Reset signaled
#define USB_INTCTRL_RESUME       0x00000002  // Resume detected
#define USB_INTCTRL_SUSPEND      0x00000001  // Suspend detected
#define USB_INTCTRL_MODE_DETECT 0x00000200  // Mode value valid
#define USB_INTCTRL_POWER_FAULT 0x00000100  // Power Fault detected
//端点中断控制
#define USB_INTEP_ALL            0xFFFFFFFF  // Host IN Interrupts
#define USB_INTEP_HOST_IN        0xFFFE0000  // Host IN Interrupts
#define USB_INTEP_HOST_IN_15     0x80000000  // Endpoint 15 Host IN Interrupt
#define USB_INTEP_HOST_IN_14     0x40000000  // Endpoint 14 Host IN Interrupt
#define USB_INTEP_HOST_IN_13     0x20000000  // Endpoint 13 Host IN Interrupt
#define USB_INTEP_HOST_IN_12     0x10000000  // Endpoint 12 Host IN Interrupt
#define USB_INTEP_HOST_IN_11     0x08000000  // Endpoint 11 Host IN Interrupt
#define USB_INTEP_HOST_IN_10     0x04000000  // Endpoint 10 Host IN Interrupt
#define USB_INTEP_HOST_IN_9      0x02000000  // Endpoint 9 Host IN Interrupt
#define USB_INTEP_HOST_IN_8      0x01000000  // Endpoint 8 Host IN Interrupt
#define USB_INTEP_HOST_IN_7      0x00800000  // Endpoint 7 Host IN Interrupt
#define USB_INTEP_HOST_IN_6      0x00400000  // Endpoint 6 Host IN Interrupt
#define USB_INTEP_HOST_IN_5      0x00200000  // Endpoint 5 Host IN Interrupt
#define USB_INTEP_HOST_IN_4      0x00100000  // Endpoint 4 Host IN Interrupt
#define USB_INTEP_HOST_IN_3      0x00080000  // Endpoint 3 Host IN Interrupt
#define USB_INTEP_HOST_IN_2      0x00040000  // Endpoint 2 Host IN Interrupt
#define USB_INTEP_HOST_IN_1      0x00020000  // Endpoint 1 Host IN Interrupt
#define USB_INTEP_DEV_OUT        0xFFFE0000  // Device OUT Interrupts
#define USB_INTEP_DEV_OUT_15     0x80000000  // Endpoint 15 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_14     0x40000000  // Endpoint 14 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_13     0x20000000  // Endpoint 13 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_12     0x10000000  // Endpoint 12 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_11     0x08000000  // Endpoint 11 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_10     0x04000000  // Endpoint 10 Device OUT Interrupt
```

```
#define USB_INTEP_DEV_OUT_9     0x02000000  // Endpoint 9 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_8     0x01000000  // Endpoint 8 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_7     0x00800000  // Endpoint 7 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_6     0x00400000  // Endpoint 6 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_5     0x00200000  // Endpoint 5 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_4     0x00100000  // Endpoint 4 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_3     0x00080000  // Endpoint 3 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_2     0x00040000  // Endpoint 2 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_1     0x00020000  // Endpoint 1 Device OUT Interrupt
#define USB_INTEP_HOST_OUT      0x0000FFFE  // Host OUT Interrupts
#define USB_INTEP_HOST_OUT_15   0x00008000  // Endpoint 15 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_14   0x00004000  // Endpoint 14 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_13   0x00002000  // Endpoint 13 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_12   0x00001000  // Endpoint 12 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_11   0x00000800  // Endpoint 11 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_10   0x00000400  // Endpoint 10 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_9    0x00000200  // Endpoint 9 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_8    0x00000100  // Endpoint 8 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_7    0x00000080  // Endpoint 7 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_6    0x00000040  // Endpoint 6 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_5    0x00000020  // Endpoint 5 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_4    0x00000010  // Endpoint 4 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_3    0x00000008  // Endpoint 3 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_2    0x00000004  // Endpoint 2 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_1    0x00000002  // Endpoint 1 Host OUT Interrupt
#define USB_INTEP_DEV_IN        0x0000FFFE  // Device IN Interrupts
#define USB_INTEP_DEV_IN_15     0x00008000  // Endpoint 15 Device IN Interrupt
#define USB_INTEP_DEV_IN_14     0x00004000  // Endpoint 14 Device IN Interrupt
#define USB_INTEP_DEV_IN_13     0x00002000  // Endpoint 13 Device IN Interrupt
#define USB_INTEP_DEV_IN_12     0x00001000  // Endpoint 12 Device IN Interrupt
#define USB_INTEP_DEV_IN_11     0x00000800  // Endpoint 11 Device IN Interrupt
#define USB_INTEP_DEV_IN_10     0x00000400  // Endpoint 10 Device IN Interrupt
#define USB_INTEP_DEV_IN_9      0x00000200  // Endpoint 9 Device IN Interrupt
#define USB_INTEP_DEV_IN_8      0x00000100  // Endpoint 8 Device IN Interrupt
#define USB_INTEP_DEV_IN_7      0x00000080  // Endpoint 7 Device IN Interrupt
#define USB_INTEP_DEV_IN_6      0x00000040  // Endpoint 6 Device IN Interrupt
#define USB_INTEP_DEV_IN_5      0x00000020  // Endpoint 5 Device IN Interrupt
#define USB_INTEP_DEV_IN_4      0x00000010  // Endpoint 4 Device IN Interrupt
#define USB_INTEP_DEV_IN_3      0x00000008  // Endpoint 3 Device IN Interrupt
#define USB_INTEP_DEV_IN_2      0x00000004  // Endpoint 2 Device IN Interrupt
#define USB_INTEP_DEV_IN_1      0x00000002  // Endpoint 1 Device IN Interrupt
#define USB_INTEP_0             0x00000001  // Endpoint 0 Interrupt
```

中断使能函数：

```
void USBIntEnable(unsigned long ulBase, unsigned long ulFlags)
{
    //发送中断控制
    if(ulFlags & (USB_INT_HOST_OUT | USB_INT_DEV_IN | USB_INT_EP0))
    {
        HWREGH(ulBase + USB_O_TXIE) |=
            ulFlags & (USB_INT_HOST_OUT | USB_INT_DEV_IN | USB_INT_EP0);
    }
    //接收中断控制
    if(ulFlags & (USB_INT_HOST_IN | USB_INT_DEV_OUT))
    {
        HWREGH(ulBase + USB_O_RXIE) |=
            ((ulFlags & (USB_INT_HOST_IN | USB_INT_DEV_OUT)) >>
             USB_INT_RX_SHIFT);
    }
    //通用中断控制
```

```
    if(ulFlags & USB_INT_STATUS)
    {
        HWREGB(ulBase + USB_O_IE) |=
            (ulFlags & USB_INT_STATUS) >> USB_INT_STATUS_SHIFT;
    }
    //电源中断控制
    if(ulFlags & USB_INT_POWER_FAULT)
    {
        HWREG(ulBase + USB_O_EPCIM) = USB_EPCIM_PF;
    }
    //ID 中断控制
    if(ulFlags & USB_INT_MODE_DETECT)
    {
        HWREG(USB0_BASE + USB_O_IDVIM) = USB_IDVIM_ID;
    }
}
```

中断禁止函数：

```
void USBIntDisable(unsigned long ulBase, unsigned long ulFlags)
{
    //发送中断控制
    if(ulFlags & (USB_INT_HOST_OUT | USB_INT_DEV_IN | USB_INT_EP0))
    {
        HWREGH(ulBase + USB_O_TXIE) &=
            ~(ulFlags & (USB_INT_HOST_OUT | USB_INT_DEV_IN | USB_INT_EP0));
    }
    //接收中断控制
    if(ulFlags & (USB_INT_HOST_IN | USB_INT_DEV_OUT))
    {
        HWREGH(ulBase + USB_O_RXIE) &=
            ~((ulFlags & (USB_INT_HOST_IN | USB_INT_DEV_OUT)) >>
              USB_INT_RX_SHIFT);
    }
    //通用中断控制
    if(ulFlags & USB_INT_STATUS)
    {
        HWREGB(ulBase + USB_O_IE) &=
            ~((ulFlags & USB_INT_STATUS) >> USB_INT_STATUS_SHIFT);
    }
    //电源中断控制
    if(ulFlags & USB_INT_POWER_FAULT)
    {
        HWREG(ulBase + USB_O_EPCIM) = 0;
    }
    //ID 中断控制
    if(ulFlags & USB_INT_MODE_DETECT)
    {
        HWREG(USB0_BASE + USB_O_IDVIM) = 0;
    }
}
```

获取中断标志并清除函数：

```
unsigned long USBIntStatus(unsigned long ulBase)
{
    unsigned long ulStatus;
    ulStatus = (HWREGB(ulBase + USB_O_TXIS));
    ulStatus |= (HWREGB(ulBase + USB_O_RXIS) << USB_INT_RX_SHIFT);
    ulStatus |= (HWREGB(ulBase + USB_O_IS) << USB_INT_STATUS_SHIFT);
    if(HWREG(ulBase + USB_O_EPCISC) & USB_EPCISC_PF)
    {
        ulStatus |= USB_INT_POWER_FAULT;
```

```
            HWREGB(ulBase + USB_O_EPCISC) |= USB_EPCISC_PF;
        }

        if(HWREG(USB0_BASE + USB_O_IDVISC) & USB_IDVRIS_ID)
        {
            ulStatus |= USB_INT_MODE_DETECT;
            HWREG(USB0_BASE + USB_O_IDVISC) |= USB_IDVRIS_ID;
        }
        return(ulStatus);
    }
```

端点中断使用较为频繁，可单独控制。

```
    void USBIntDisableEndpoint(unsigned long ulBase, unsigned long ulFlags)
    {
        HWREGH(ulBase + USB_O_TXIE) &=
            ~(ulFlags & (USB_INTEP_HOST_OUT | USB_INTEP_DEV_IN | USB_INTEP_0));
        HWREGH(ulBase + USB_O_RXIE) &=
            ~((ulFlags & (USB_INTEP_HOST_IN | USB_INTEP_DEV_OUT)) >>
              USB_INTEP_RX_SHIFT);
    }
    void USBIntEnableEndpoint(unsigned long ulBase, unsigned long ulFlags)
    {
        HWREGH(ulBase + USB_O_TXIE) |=
               ulFlags & (USB_INTEP_HOST_OUT | USB_INTEP_DEV_IN | USB_INTEP_0);
        HWREGH(ulBase + USB_O_RXIE) |=
            ((ulFlags & (USB_INTEP_HOST_IN | USB_INTEP_DEV_OUT)) >>
             USB_INTEP_RX_SHIFT);
    }
    unsigned long USBIntStatusEndpoint(unsigned long ulBase)
    {
        unsigned long ulStatus;
        ulStatus = HWREGH(ulBase + USB_O_TXIS);
        ulStatus |= (HWREGH(ulBase + USB_O_RXIS) << USB_INTEP_RX_SHIFT);
        return(ulStatus);
    }
```

2.3.3 端点寄存器

USBEPIDX，USB 端点索引寄存器。设置端点 FIFO 大小和起始地址要配合 USBEPIDX 使用。USBEPIDX 寄存器描述如下表：

位	名称	类型	复位	描述
[7:4]	保留	RO	0	保留
[3:0]	EPIDX	R/W	0	端点索引

表 19. USBEPIDX 寄存器

USBRXFIFOSZ、USBTXFIFOSZ ，USB 接收/发送 FIFO 大小寄存器。USBRXFIFOSZ、USBTXFIFOSZ 寄存器描述如下表：

位	名称	类型	复位	描述
[7:5]	保留	RO	0	保留
4	DPB	R/W	0	双包缓存
[3:0]	SIZE	R/W	0	最大包 8*(2^SIZE)

表 20. USBRXFIFOSZ、USBTXFIFOSZ 寄存器

USBRXFIFOADD、USBTXFIFOADD ，USB 接收/发送 FIFO 首地址寄存器。USBRXFIFOADD、USBTXFIFOADD 寄存器描述如下表：

位	名称	类型	复位	描述
[15：9]	保留	RO	0	保留

[8:0]	ADDR	R/W	0	最大包 8*SIZE

表 21. USBRXFIFOADD、USBTXFIFOADD 寄存器

注意：访问要用到索引寄存器的 FIFO 寄存器只有 USBRXFIFOSZ、USBTXFIFOSZ、USBRXFIFOADD、USBTXFIFOADD 四个。

例如：写一个函数，控制端点的 FIFO。

```
static void  USBIndexWrite(unsigned long ulBase, unsigned long ulEndpoint,
              unsigned long ulIndexedReg, unsigned long ulValue,
              unsigned long ulSize)
{
    unsigned long ulIndex;
    // 保存当前索引寄存器值.
    ulIndex = HWREGB(ulBase + USB_O_EPIDX);
    // 写新值到端点索引寄存器
    HWREGB(ulBase + USB_O_EPIDX) = ulEndpoint;
     //根据寄存器大小写入新值到 FIFO 寄存器。
    if(ulSize == 1)
    {
        HWREGB(ulBase + ulIndexedReg) = ulValue;
    }
    else
    {
        HWREGH(ulBase + ulIndexedReg) = ulValue;
    }
    //恢复以前索引寄存器值。
    HWREGB(ulBase + USB_O_EPIDX) = ulIndex;
}
```

USBTXFUNCADDRn，发送端点功能地址寄存器。在主机模式下，用于设置端点 n 访问的目标地址。USBTXFUNCADDRn 寄存器描述如下表：

位	名称	类型	复位	描述
7	保留	RO	0	保留
[6:0]	ADDR	R/W	0	设备总线地址

表 22 USBTXFUNCADDRn 寄存器

USBTXHUBPORTn，发送端点 n 集线器端口号寄存器。USBTXHUBPORTn 寄存器描述如下表：

位	名称	类型	复位	描述
7	保留	RO	0	保留
[6:0]	ADDR	R/W	0	集线器端口

表 23 USBTXHUBPORTn 寄存器

USBTXHUBADDRn，发送端点 n 集线器地址寄存器，USBTXHUBADDRn 寄存器描述如下表：

位	名称	类型	复位	描述
7	MULTTRAN	RO	0	多路开关
[6:0]	ADDR	R/W	0	集线器地址

表 24 USBTXHUBADDRn 寄存器

USBRXFUNCADDRn，接收端点功能地址寄存器。在主机模式下，用于设置端点 n 访问的目标地址。USBRXFUNCADDRn 寄存器描述如下表：

位	名称	类型	复位	描述
7	保留	RO	0	保留
[6:0]	ADDR	R/W	0	设备总线地址

表 25 USBRXFUNCADDRn 寄存器

USBRXHUBPORTn，接收端点 n 集线器端口号寄存器。USBRXHUBPORTn 寄存器描述如下表：

位	名称	类型	复位	描述
7	保留	RO	0	保留
[6:0]	ADDR	R/W	0	集线器端口

表 26 USBRXHUBPORTn 寄存器

USBRXHUBADDRn，接收端点 n 集线器地址寄存器，USBRXHUBADDRn 寄存器描述如下表：

位	名称	类型	复位	描述
7	MULTTRAN	RO	0	多路开关
[6:0]	ADDR	R/W	0	集线器地址

表 27 USBRXHUBADDRn 寄存器

注意：USBTXFUNCADDR0、USBTXHUBPORT0 、USBTXHUBADDR0 同时用于端点 0 的接收和发送。

例如：写一个函数，主机端点访问某个设备地址。

USBHostAddrSet() 的 ulEndpoint 参数：

```
#define USB_EP_0                0x00000000  // Endpoint 0
#define USB_EP_1                0x00000010  // Endpoint 1
#define USB_EP_2                0x00000020  // Endpoint 2
#define USB_EP_3                0x00000030  // Endpoint 3
#define USB_EP_4                0x00000040  // Endpoint 4
#define USB_EP_5                0x00000050  // Endpoint 5
#define USB_EP_6                0x00000060  // Endpoint 6
#define USB_EP_7                0x00000070  // Endpoint 7
#define USB_EP_8                0x00000080  // Endpoint 8
#define USB_EP_9                0x00000090  // Endpoint 9
#define USB_EP_10               0x000000A0  // Endpoint 10
#define USB_EP_11               0x000000B0  // Endpoint 11
#define USB_EP_12               0x000000C0  // Endpoint 12
#define USB_EP_13               0x000000D0  // Endpoint 13
#define USB_EP_14               0x000000E0  // Endpoint 14
#define USB_EP_15               0x000000F0  // Endpoint 15
#define NUM_USB_EP              16          // Number of supported endpoints
void USBHostAddrSet(unsigned long ulBase, unsigned long ulEndpoint,
                    unsigned long ulAddr, unsigned long ulFlags)
{
    //根据 ulFlags 设置发送还是接地址
    if(ulFlags & USB_EP_HOST_OUT)
    {
        HWREGB(ulBase + USB_0_TXFUNCADDR0 + (ulEndpoint >> 1)) = ulAddr;
    }
    else
    {
        HWREGB(ulBase + USB_0_TXFUNCADDR0 + 4 + (ulEndpoint >> 1)) = ulAddr;
    }
}
```

例如：写一个函数，主机端点通过集线器访问某个设备地址。

```
void USBHostHubAddrSet(unsigned long ulBase, unsigned long ulEndpoint,
                       unsigned long ulAddr, unsigned long ulFlags)
{
    ////根据 ulFlags 设置发送还是接地址
    if(ulFlags & USB_EP_HOST_OUT)
    {
        HWREGB(ulBase + USB_0_TXHUBADDR0 + (ulEndpoint >> 1)) = ulAddr;
    }
    else
    {
```

```
            HWREGB(ulBase + USB_O_TXHUBADDR0 + 4 + (ulEndpoint >> 1)) = ulAddr;
        }
    }
```

USBTXMAXPn，发送端点 n 最大传输数据寄存器，定义发送端点单次可传输的最大数据长度。USBTXMAXPn 寄存器描述如下表：

位	名称	类型	复位	描述
[15:11]	保留	RO	0	保留
[10:0]	MAXLOAD	R/W	0	单次传输数据最大字节数

表 28 USBTXMAXPn 寄存器

USBRXCOUNTn，USB 端点 n 接收字节数寄存器，从 RXFIFO 中读出数据的字节数。USBRXCOUNTn 寄存器描述如下表：

位	名称	类型	复位	描述
[15:13]	保留	RO	0	保留
[12:0]	COUNT	R/W	0	接收字节数

表 29 USBRXCOUNTn 寄存器

USBTXTYPEn，主机发送类型寄存器，配置发送端点的目标端点号，传输协议，以及其运行速度。USBTXTYPEn 寄存器描述如下表：

位	名称	类型	复位	描述
[7:6]	SPEED	R/W	0	运行速度
[5:4]	PROTO	R/W	0	协议
[3:0]	TEP	R/W	0	目标端点号

表 30 USBTXTYPEn 寄存器

USBTXTYPEn，主机发送类型寄存器，配置发送端点的目标端点号，传输协议，以及其运行速度。USBTXTYPEn 寄存器描述如下表：

位	名称	类型	复位	描述
[7:6]	SPEED	R/W	0	运行速度
[5:4]	PROTO	R/W	0	协议
[3:0]	TEP	R/W	0	目标端点号

表 31 USBTXTYPEn 寄存器

USBTXINTERVALn，主机发送间隔寄存器。USBTXINTERVALn 寄存器描述如下表：

位	名称	类型	复位	描述
[7:0]	TXPOLL / NAKLMT	R/W	0	NAK 超时时间间隔

表 32 USBTXINTERVALn 寄存器

USBRXTYPEn，主机接收类型寄存器，配置接收端点的目标端点号，传输协议，以及其运行速度。USBRXTYPEn 寄存器描述如下表：

位	名称	类型	复位	描述
[7:6]	SPEED	R/W	0	运行速度
[5:4]	PROTO	R/W	0	协议
[3:0]	TEP	R/W	0	目标端点号

表 33 USBRXTYPEn 寄存器

USBRXINTERVALn，主机接收间隔寄存器。USBRXINTERVALn 寄存器描述如下表：

位	名称	类型	复位	描述
[7:0]	TXPOLL / NAKLMT	R/W	0	NAK 超时时间间隔

表 34 USBRXINTERVALn 寄存器

USBRQPKTCOUNTn，块传输请求包数量寄存器。USBRQPKTCOUNTn 寄存器描述如下表：

位	名称	类型	复位	描述
[15:0]	COUNT	R/W	0	块传输包数量

表 35 USBRQPKTCOUNTn 寄存器

USBRXDPKTBUFDIS，接收双包缓存禁止寄存器，USBRXDPKTBUFDIS 寄存器描述如下表：

位	名称	类型	复位	描述
15..0	EPn	R/W	0	接收双包缓存禁止

表 36 USBRXDPKTBUFDIS 寄存器

USBTXDPKTBUFDIS，发送双包缓存禁止寄存器，USBTXDPKTBUFDIS 寄存器描述如下表：

位	名称	类型	复位	描述
15..0	EPn	R/W	0	发送双包缓存禁止

表 37 USBTXDPKTBUFDIS 寄存器

USBFIFOn，FIFO 端点寄存器，主要进行 FIFO 访问。写操作，将向 TXFIFO 中写入数据；读操作，将从 RXFIFO 中读出数据。USBFIFOn 寄存器描述如下表：

位	名称	类型	复位	描述
[31:0]	EPDATA	R/W	0	端点数据

表 38 USBFIFOn 寄存器

USBCSRL0，端点 0 控制和状态低字节寄存器，为端点 0 提供控制和状态位。USBCSRL0 寄存器描述如下表：

位	名称	类型	复位	描述　主机/设备
7	NAKTO/SETENDC	R/W	0	NAK 超时/SETEND 清 0
6	STATUS/RXRDYC	R/W	0	状态包/清 RXRDY 位
5	REQPKT/STALL	R/W	0	请求包/发送 STALL 握手
4	ERROR/SETEND	R/W	0	错误/Setup End
3	SETUP/DATAEND	R/W	0	建立令牌包/数据结束
2	STALLED	R/W	0	端点挂起
1	TXRDY	R/W	0	发送包准备好
0	RXRDY	R/W	0	接收包准备好

表 39 USBCSRL0 寄存器

USBCSRH0，端点 0 控制和状态高字节寄存器，为端点 0 提供控制和状态位。USBCSRH0 寄存器描述如下表：

位	名称	类型	复位	描述
[7:3]	保留	RO	0	保留
2	DTWE	R/W	0	数据切换写使能（只有主机模式）
1	DT	R/W	0	数据切换（只有主机模式）
0	FLUSH	R/W	0	清空 FIFO

表 40 USBCSRH0 寄存器

USBTXCSRLn，发送端点 n(非端点 0)控制和状态低字节寄存器，为端点 n 提供控制和状态位。USBTXCSRLn 寄存器描述如下表：

位	名称	类型	复位	描述　主机/设备
7	NAKTO	R/W	0	NAK 超时（只有主机）
6	CLRDT	R/W	0	清除数据转换
5	STALLED	R/W	0	端点挂起
4	SETUP/STALL	R/W	0	建立令牌包/发送 STALL

3	FLUSH	R/W	0	清空 FIFO
2	ERROR/UNDRN	R/W	0	错误/欠运转
1	FIFONE	R/W	0	FIFO 不空
0	TXRDY	R/W	0	发送包准备好

表 41 USBTXCSRLn 寄存器

USBTXCSRHn，发送端点 n(非端点 0)控制和状态高字节寄存器，为端点 n 提供控制和状态位。USBTXCSRHn 寄存器描述如下表：

位	名称	类型	复位	描述
7	AUTOSET	R/W	0	自动置位
6	ISO	R/W	0	ISO 传输（只有设备模式）
5	MODE	R/W	0	模式
4	DMAEN	R/W	0	DMA 请求使能
3	FDT	R/W	0	强制数据切换
2	DMAMOD	R/W	0	DMA 请求模式
1	DTWE	R/W	0	数据切换写使能（只有主机模式）
0	DT	R/W	0	数据切换（只有主机模式）

表 42 USBTXCSRHn 寄存器

USBRXCSRLn，接收端点 n 控制和状态低字节寄存器，USBRXCSRLn 寄存器描述如下表：

位	名称	类型	复位	描述　主机/设备
7	CLRDT	R/W	0	清除数据转换
6	STALLED	R/W	0	端点挂起
5	REQPKT/ STALL	R/W	0	请求包/发送 STALL 握手
4	FLUSH	R/W	0	清空 FIFO
3	DATAERR\NAKTO/ DATAERR	R/W	0	数据错误\NAK 超时 /数据错误
2	ERROR/OVER	R/W	0	错误/Overrun
1	FULL	R/W	0	错误
0	RXRDY	R/W	0	接收包准备好

表 43 USBRXCSRLn 寄存器

USBRXCSRHn,接收端点 n 控制和状态高字节寄存器，为接收端点提供额外的控制和状态位。USBRXCSRHn 寄存器描述如下表：

位	名称	类型	复位	描述
7	AUTOCL	R/W	0	自动清除
6	AUTORQ	R/W	0	自动请求
5	DMAEN	R/W	0	DMA 请求使能
4	PIDERR	R/W	0	PID 错误
3	DMAMOD	R/W	0	DMA 请求模式
2	DTWE	R/W	0	数据切换写使能（只有主机模式）
1	DT	R/W	0	数据切换（只有主机模式）
0	保留	RO	0	保留

表 44 USBRXCSRHn 寄存器

例如：写一个函数，配置端点。

USBHostEndpointConfig()和 USBDevEndpointConfigSet()的 ulFlags 参数：

```
#define USB_EP_AUTO_SET         0x00000001  // Auto set feature enabled
#define USB_EP_AUTO_REQUEST     0x00000002  // Auto request feature enabled
```

```
#define USB_EP_AUTO_CLEAR       0x00000004  // Auto clear feature enabled
#define USB_EP_DMA_MODE_0       0x00000008  // Enable DMA access using mode 0
#define USB_EP_DMA_MODE_1       0x00000010  // Enable DMA access using mode 1
#define USB_EP_MODE_ISOC        0x00000000  // Isochronous endpoint
#define USB_EP_MODE_BULK        0x00000100  // Bulk endpoint
#define USB_EP_MODE_INT         0x00000200  // Interrupt endpoint
#define USB_EP_MODE_CTRL        0x00000300  // Control endpoint
#define USB_EP_MODE_MASK        0x00000300  // Mode Mask
#define USB_EP_SPEED_LOW        0x00000000  // Low Speed
#define USB_EP_SPEED_FULL       0x00001000  // Full Speed
#define USB_EP_HOST_IN          0x00000000  // Host IN endpoint
#define USB_EP_HOST_OUT         0x00002000  // Host OUT endpoint
#define USB_EP_DEV_IN           0x00002000  // Device IN endpoint
#define USB_EP_DEV_OUT          0x00000000  // Device OUT endpoint
```

ulFIFOSize 参数：

```
#define USB_FIFO_SZ_8           0x00000000  // 8 byte FIFO
#define USB_FIFO_SZ_16          0x00000001  // 16 byte FIFO
#define USB_FIFO_SZ_32          0x00000002  // 32 byte FIFO
#define USB_FIFO_SZ_64          0x00000003  // 64 byte FIFO
#define USB_FIFO_SZ_128         0x00000004  // 128 byte FIFO
#define USB_FIFO_SZ_256         0x00000005  // 256 byte FIFO
#define USB_FIFO_SZ_512         0x00000006  // 512 byte FIFO
#define USB_FIFO_SZ_1024        0x00000007  // 1024 byte FIFO
#define USB_FIFO_SZ_2048        0x00000008  // 2048 byte FIFO
#define USB_FIFO_SZ_4096        0x00000009  // 4096 byte FIFO
#define USB_FIFO_SZ_8_DB        0x00000010  // 8 byte double buffered FIFO
#define USB_FIFO_SZ_16_DB       0x00000011  // 16 byte double buffered FIFO
#define USB_FIFO_SZ_32_DB       0x00000012  // 32 byte double buffered FIFO
#define USB_FIFO_SZ_64_DB       0x00000013  // 64 byte double buffered FIFO
#define USB_FIFO_SZ_128_DB      0x00000014  // 128 byte double buffered FIFO
#define USB_FIFO_SZ_256_DB      0x00000015  // 256 byte double buffered FIFO
#define USB_FIFO_SZ_512_DB      0x00000016  // 512 byte double buffered FIFO
#define USB_FIFO_SZ_1024_DB     0x00000017  // 1024 byte double buffered FIFO
#define USB_FIFO_SZ_2048_DB     0x00000018  // 2048 byte double buffered FIFO
```

端点转化到其状态控制器寄存器地址：

```
#define EP_OFFSET(Endpoint)     (Endpoint - 0x10)
```

主机端点配置函数：

```
void USBHostEndpointConfig(unsigned long ulBase, unsigned long ulEndpoint,
                    unsigned long ulMaxPayload,
                    unsigned long ulNAKPollInterval,
                    unsigned long ulTargetEndpoint, unsigned long ulFlags)
{
    unsigned long ulRegister;
    //判断是否是端点 0，端点 0 的发送与接收配置使用同一寄存器。
    if(ulEndpoint == USB_EP_0)
    {
        HWREGB(ulBase + USB_O_NAKLMT) = ulNAKPollInterval;
        HWREGB(ulBase + EP_OFFSET(ulEndpoint) + USB_O_TYPE0) =
            ((ulFlags & USB_EP_SPEED_FULL) ? USB_TYPE0_SPEED_FULL :
             USB_TYPE0_SPEED_LOW);
    }
    //端点 1-15 配置
    else
    {
        ulRegister = ulTargetEndpoint;
        if(ulFlags & USB_EP_SPEED_FULL)
        {
            ulRegister |= USB_TXTYPE1_SPEED_FULL;
        }
```

```
else
{
    ulRegister |= USB_TXTYPE1_SPEED_LOW;
}
switch(ulFlags & USB_EP_MODE_MASK)
{
    case USB_EP_MODE_BULK:
    {
        ulRegister |= USB_TXTYPE1_PROTO_BULK;
        break;
    }
    case USB_EP_MODE_ISOC:
    {
        ulRegister |= USB_TXTYPE1_PROTO_ISOC;
        break;
    }
    case USB_EP_MODE_INT:
    {
        ulRegister |= USB_TXTYPE1_PROTO_INT;
        break;
    }
    case USB_EP_MODE_CTRL:
    {
        ulRegister |= USB_TXTYPE1_PROTO_CTRL;
        break;
    }
}
//发送/接收端点配置
if(ulFlags & USB_EP_HOST_OUT)
{
    HWREGB(ulBase + EP_OFFSET(ulEndpoint) + USB_O_TXTYPE1) =
        ulRegister;
    HWREGB(ulBase + EP_OFFSET(ulEndpoint) + USB_O_TXINTERVAL1) =
        ulNAKPollInterval;
    HWREGB(ulBase + EP_OFFSET(ulEndpoint) + USB_O_TXMAXP1) =
        ulMaxPayload;
    ulRegister = 0;
    if(ulFlags & USB_EP_AUTO_SET)
    {
        ulRegister |= USB_TXCSRH1_AUTOSET;
    }
    if(ulFlags & USB_EP_DMA_MODE_1)
    {
        ulRegister |= USB_TXCSRH1_DMAEN | USB_TXCSRH1_DMAMOD;
    }
    else if(ulFlags & USB_EP_DMA_MODE_0)
    {
        ulRegister |= USB_TXCSRH1_DMAEN;
    }
    HWREGB(ulBase + EP_OFFSET(ulEndpoint) + USB_O_TXCSRH1) =
        (unsigned char)ulRegister;
}
else
{
    HWREGB(ulBase + EP_OFFSET(ulEndpoint) + USB_O_RXTYPE1) =
        ulRegister;
    HWREGB(ulBase + EP_OFFSET(ulEndpoint) + USB_O_RXINTERVAL1) =
        ulNAKPollInterval;
    ulRegister = 0;
```

```
            if(ulFlags & USB_EP_AUTO_CLEAR)
            {
                ulRegister |= USB_RXCSRH1_AUTOCL;
            }
            //DMA 控制
            if(ulFlags & USB_EP_DMA_MODE_1)
            {
                ulRegister |= USB_RXCSRH1_DMAEN | USB_RXCSRH1_DMAMOD;
            }
            else if(ulFlags & USB_EP_DMA_MODE_0)
            {
                ulRegister |= USB_RXCSRH1_DMAEN;
            }
            HWREGB(ulBase + EP_OFFSET(ulEndpoint) + USB_O_RXCSRH1) =
                (unsigned char)ulRegister;
        }
    }
}
```

设备端点配置函数：

```
void USBDevEndpointConfigSet(unsigned long ulBase, unsigned long ulEndpoint,
                             unsigned long ulMaxPacketSize, unsigned long ulFlags)
{
    unsigned long ulRegister;
    if(ulFlags & USB_EP_DEV_IN)
    {
        HWREGB(ulBase + EP_OFFSET(ulEndpoint) + USB_O_TXMAXP1) =
            ulMaxPacketSize;
        ulRegister = 0;
        if(ulFlags & USB_EP_AUTO_SET)
        {
            ulRegister |= USB_TXCSRH1_AUTOSET;
        }
        if(ulFlags & USB_EP_DMA_MODE_1)
        {
            ulRegister |= USB_TXCSRH1_DMAEN | USB_TXCSRH1_DMAMOD;
        }
        else if(ulFlags & USB_EP_DMA_MODE_0)
        {
            ulRegister |= USB_TXCSRH1_DMAEN;
        }
        if((ulFlags & USB_EP_MODE_MASK) == USB_EP_MODE_ISOC)
        {
            ulRegister |= USB_TXCSRH1_ISO;
        }
        HWREGB(ulBase + EP_OFFSET(ulEndpoint) + USB_O_TXCSRH1) =
            (unsigned char)ulRegister;
        HWREGB(ulBase + EP_OFFSET(ulEndpoint) + USB_O_TXCSRL1) =
            USB_TXCSRL1_CLRDT;
    }
    else
    {
        HWREGB(ulBase + EP_OFFSET(ulEndpoint) + USB_O_RXMAXP1) =
            ulMaxPacketSize;
        ulRegister = 0;
        if(ulFlags & USB_EP_AUTO_CLEAR)
        {
            ulRegister = USB_RXCSRH1_AUTOCL;
        }
        if(ulFlags & USB_EP_DMA_MODE_1)
```

```
        {
            ulRegister |= USB_RXCSRH1_DMAEN | USB_RXCSRH1_DMAMOD;
        }
        else if(ulFlags & USB_EP_DMA_MODE_0)
        {
            ulRegister |= USB_RXCSRH1_DMAEN;
        }
        if((ulFlags & USB_EP_MODE_MASK) == USB_EP_MODE_ISOC)
        {
            ulRegister |= USB_RXCSRH1_ISO;
        }
        HWREGB(ulBase + EP_OFFSET(ulEndpoint) + USB_O_RXCSRH1) =
            (unsigned char)ulRegister;
        HWREGB(ulBase + EP_OFFSET(ulEndpoint) + USB_O_RXCSRL1) =
            USB_RXCSRL1_CLRDT;
    }
}
```

从 FIFO 中读取数据函数：

```
long USBEndpointDataGet(unsigned long ulBase, unsigned long ulEndpoint,
                        unsigned char *pucData, unsigned long *pulSize)
{
    unsigned long ulRegister, ulByteCount, ulFIFO;
    if(ulEndpoint == USB_EP_0)
    {
        ulRegister = USB_O_CSRL0;
    }
    else
    {
        ulRegister = USB_O_RXCSRL1 + EP_OFFSET(ulEndpoint);
    }
     //判断数据是否准备好。
    if((HWREGH(ulBase + ulRegister) & USB_CSRL0_RXRDY) == 0)
    {
        *pulSize = 0;
        return(-1);
    }
    //获取要读取的数据个数
    ulByteCount = HWREGH(ulBase + USB_O_COUNT0 + ulEndpoint);
    ulByteCount = (ulByteCount < *pulSize) ? ulByteCount : *pulSize;
    *pulSize = ulByteCount;
    ulFIFO = ulBase + USB_O_FIFO0 + (ulEndpoint >> 2);
    //从 FIFO 中读取
    for(; ulByteCount > 0; ulByteCount--)
    {

        *pucData++ = HWREGB(ulFIFO);
    }
    return(0);
}
```

写数据到 FIFO 中函数：

```
long USBEndpointDataPut(unsigned long ulBase, unsigned long ulEndpoint,
                        unsigned char *pucData, unsigned long ulSize)
{
    unsigned long ulFIFO;
    unsigned char ucTxPktRdy;
    if(ulEndpoint == USB_EP_0)
    {
        ucTxPktRdy = USB_CSRL0_TXRDY;
    }
```

```
    else
    {
        ucTxPktRdy = USB_TXCSRL1_TXRDY;
    }
   //判断是否可以写入数据
    if(HWREGB(ulBase + USB_O_CSRLO + ulEndpoint) & ucTxPktRdy)
    {
        return(-1);
    }
    //计算 FIFO 地址
    ulFIFO = ulBase + USB_O_FIFOO + (ulEndpoint >> 2);
    //写入 FIFO
    for(; ulSize > 0; ulSize--)
    {
        HWREGB(ulFIFO) = *pucData++;
    }
    return(0);
}
```

发送刚写入 FIFO 的数据函数：

```
long USBEndpointDataSend(unsigned long ulBase, unsigned long ulEndpoint,
                    unsigned long ulTransType)
{
    unsigned long ulTxPktRdy;
    if(ulEndpoint == USB_EP_0)
    {
        ulTxPktRdy = ulTransType & 0xff;
    }
    else
    {
        ulTxPktRdy = (ulTransType >> 8) & 0xff;
    }
    // 判断发送数据包是否准备好
    if(HWREGB(ulBase + USB_O_CSRLO + ulEndpoint) & USB_CSRLO_TXRDY)
    {
        return(-1);
    }
    //发送刚写入的数据
    HWREGB(ulBase + USB_O_CSRLO + ulEndpoint) = ulTxPktRdy;
    return(0);
}
```

以上介绍了常用 USB 寄存器，不常用寄存器请读者参考相关数据手册。通过以上寄存器操作可以开发 USB 主机与设备模块，可以完成全部 USB2.0 支持的全速和低速系统开发。一般情况下，不使用寄存器级编程，为了方便使用，LM3S 处理器官方已经制作好了 C 语言“driverlib.lib”库，并免费为开发人员提供，在 LM3S5000 系列 MCU 中已将“driverlib.lib”库预先放入 ROM 中，节约程序储存空间，并且调用速度更快。第三章将重点介绍 Stellaris 处理器的 USB 底层驱动函数。已经编译到 driverlib.lib 中，只要加入 usb.h 头文件就可以进行开发。第四章及以后的章节会介绍使用 usblib.lib 开发 USB 主机与设备模块。

2.4 USB 处理配置使用

与其它处理器一样，在使用其相关外设资源时，要先进行初始化配置。初始化与配置分为四步：

① 内核配置

使用 USB 处理器前必须配置 RCGC2 寄存器，使能其外设时钟；使能 USB 的 PLL 为 PHY 提供时钟；使能相应 USB 中断。

② 管脚配置

配置 RCGC2 使能相应 GPIO 模块；配置 GPIOPCTL 寄存器中的 PMCn 位，分配 USB 信号到合适的管脚上（根据具体芯片配置，有的芯片不需要此步）；作为设备时，必须禁止向 VBUS 供电，使用外部主机控制器供电。通常使用 USB0EPEN 信号用于控制外部稳压器，禁止使能外部稳压器，避免同时驱动电源管脚 USB0VBUS。

③ 端点配置

在主机模式，在和设备端点建立连接时配置；在设备模式，设备枚举之前配置。实际使用时端点很少需要设置，但都很重要。

④ 建立通信

硬件配置完成后还需进行协议配置，比如描述符、类协议之类，与 USB 通信相关的协议部分都需要考虑。

例如：使用 USB 处理器做鼠标，开始进要进行以下配置（①、②步骤，③、④后面详细讲）。

```
//配置内核 CPU 时钟，作设备时不能低于 20MHz.
HWREG(SYSCTL_RCC) &= ~(SYSCTL_RCC_MOSCDIS);
SysCtlDelay(524288);
HWREG(SYSCTL_RCC) = ((HWREG(SYSCTL_RCC) &
                      ~(SYSCTL_RCC_PWRDN | SYSCTL_RCC_XTAL_M |
                        SYSCTL_RCC_OSCSRC_M)) |
                     SYSCTL_RCC_XTAL_8MHZ | SYSCTL_RCC_OSCSRC_MAIN);
SysCtlDelay(524288);
HWREG(SYSCTL_RCC) = ((HWREG(SYSCTL_RCC) & ~(SYSCTL_RCC_BYPASS |
                     SYSCTL_RCC_SYSDIV_M)) | SYSCTL_RCC_SYSDIV_4 |
                     SYSCTL_RCC_USESYSDIV);
//使能 USB 设备时钟
SysCtlPeripheralEnable(SYSCTL_PERIPH_USB0);
// 打开 USB Phy 时钟.
SysCtlUSBPLLEnable();
//清除中断标志。
USBIntStatusControl(USB0_BASE);
USBIntStatusEndpoint(USB0_BASE);
//使能相关中断
USBIntEnableControl(USB0_BASE, USB_INTCTRL_RESET |
                               USB_INTCTRL_DISCONNECT |
                               USB_INTCTRL_RESUME |
                               USB_INTCTRL_SUSPEND |
                               USB_INTCTRL_SOF);
USBIntEnableEndpoint(USB0_BASE, USB_INTEP_ALL);
//开总中断
IntEnable(INT_USB0);
//引脚配置
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,0xf0);
GPIOPinWrite(GPIO_PORTF_BASE,0xf0,0);
GPIOPinTypeGPIOInput(GPIO_PORTF_BASE,0x0f);
HWREG(GPIO_PORTF_BASE+GPIO_O_PUR) = 0x0f;
```