总线接口

MENTOR GRAPHICS

这是　一个很好的解决方案

MUSBMHDRC

USB 2.0多点双重角色控制器

产品说明书和编程指南

机密的

2007年5月25日PSPG-40161.03-

对于此产品的客户支持：

- 拨打客户咨询服务**http://www.mentor.com/supportnet**
- 电子邮件**support_net@mentor.com**
- 电话**1-800-547-4303**（美国、墨西哥和加拿大免费）
  （世界其他地区的客户应联系当地Mentor Graphics支持办公室。）

客户支持手册中提供了完整的详细信息，该手册以Adobe Acrobat格式提供，格式为**custhb.pdf**，位于/Mentor Graphics Soft Cores CD上的**数据手册目录。**请注意《客户支持手册》中提供的联系客户支持时要采取的行动清单和必须提供的信息。

目录

1. 引言

MUSBMHDRC
USB 2.0多点
双重角色控制器

- 用作高速/全速USB外围设备的功能控制器或点对点的主机/外围设备或与其他USB功能进行多点通信
- 符合高速（480 Mbps）功能的USB 2.0标准以及USB 2.0规范的*On the Go*补充
- 支持与一个或多个高速、全速或低速设备的OTG通信
- 支持会话请求协议（SRP）和主机协商协议（HNP）
- 支持挂起和恢复信号
- 带可选ULPI链路包装器的UTMI+3级收发器接口
- 可选USB 1.1 PHY接口（仅适用于全速/低速操作），带有允许与I2C控制的PHY一起使用的可选I2C接口
- 软连接/断开连接
- 最多可配置15个附加发送端点和最多15个附加接收端点
- 提供端点的动态分配，以最大限度地增加支持的设备数量
- 可配置FIFO，包括动态FIFO大小的选项
- FIFO的同步RAM接口
- 支持对FIFO的DMA访问
- 高级AMBA™AHB兼容CPU接口（适用于各种AHB总线速度）
- 支持AHB总线上的多层操作
- 在硬件中执行所有事务调度
- 为核心配置提供图形用户界面

MUSBHDRC是一种多功能设计，在单核中提供：

- 高速/全速USB外围设备的功能控制器；
- 一个“双作用”USB控制器，用于与另一个USB功能（可以是高速、全速或低速）进行点对点“即时”（OTG）通信；和
- （当连接到集线器时）多点USB系统的主机控制器。

–进而允许使用MUSBHDRC核心的设备根据需要在这些不同的角色之间切换。

该内核既符合高速和全速功能的USB 2.0标准，也符合USB 2.0规范的*On The Go*补充。USB *On The Go*规范已被引入，为移动电话、PDA、数码相机和MP3播放器等消费者便携式设备提供低成本的连接解决方案。仅作为外围设备的设备

可以通过会话请求协议（SRP）发起USB流量，而双重角色设备同时支持SRP和主机协商协议（HNP）并且可以根据需要承担主机或外围设备的角色。MUSBHDRC还支持拆分事务，这反过来又允许它支持使用带有USB 2.0集线器的全速或低速设备。该核心还支持在不使用时关闭便携式设备的电源。

除端点0外，MUSBHDRC可供用户配置，最多可用于15个“发送”端点和/或最多15个“接收”端点。（这些端点用于IN事务和OUT事务取决于MUSBHDRC是用作外设还是用作主机。当用作外设时，IN事务通过TX端点处理，OUT事务通过Rx端点处理。当用作主机时，IN交易通过Rx终结点处理，OUT交易通过TX端点处理。）这些附加端点可以在软件中单独配置，以处理批量传输（也允许它们处理中断传输）、同步传输或控制传输。
此外，端点还可以动态分配给不同的目标设备功能，从而最大限度地增加可以同时支持的设备数量。

每个端点都需要一个与其相关的FIFO。MUSBHDRC有一个RAM接口，用于连接到用于所有端点FIFO的同步单端口RAM的单个块。（RAM块本身需要由用户添加。）

端点0的FIFO要求为64字节深，并将缓冲1个数据包。RAM接口可针对其它端点FIFO进行配置，其它端点FIFO的大小可为8到8192字节，并且可以缓冲1个或2个分组。单独的FIFO可以与每个端点相关联：或者，具有相同端点编号的TX端点和Rx端点可以配置为使用相同的FIFO，例如，以减少所需的RAM块的大小，前提是它们永远不能同时处于活动状态。

MUSBHDRC提供32位同步CPU接口，用于连接AMBA AHB总线[1]。该接口支持与以各种总线速度运行的AHB总线一起使用。还支持AHB总线上的多层操作。通过添加合适的封装器/桥接器，MUSBHDRC也可以很容易地连接到一系列其他标准总线。

还支持对端点FIFO的DMA访问。

MUSBHDRC提供了一个UTMI+Level 3兼容接口，用于连接到合适的USB高速/全速收发器。包括一个可选的ULPI链接包装器（在**musbmhdrc/docs**目录中包含的musbhdrc**_ULPI_An.pdf**文档中进行了描述），用于连接到ULPI兼容的PHY。还提供了一种替代接口，其允许使用具有核心的USB 1.1全速PHY，但仅用于全速和低速事务。（该接口在第8.1节中进行了描述）。

MUSBHDRC提供发送和接收USB数据包所需的所有编码、解码、检查和重新请求，仅当端点数据成功传输时才中断CPU。

当作为主机时，MUSBHDRC额外维护一个帧计数器，并自动安排SOF、Isochronous、Interrupt和Bulk传输。它还包括对点对点通信中使用的会话请求和主机协商协议的支持，其详细信息在*USB 2.0规范*的*USB On the Go*补充中给出。

MUSBHDRC提供一系列测试模式，主要是USB 2.0规范中描述的四种高速操作测试模式。它还包括允许强制进入全速模式、高速模式或主机模式的选项。最后一个可能有助于调试硬件中的PHY问题。

提供了图形用户界面脚本，用于根据用户需求配置核心。要使用的脚本取决于所选的CPU接口。*请注意*：在撰写本文时，核心仅在Verilog中可用。

本规范应与*USB On the Go*规范一起阅读，该规范还提供了电源要求、电压水平、连接器等的详细信息。

---

[1]参考ARM AMBA规范2.0版创建（第3章：AMBA AHB）

2. 功能描述

2.1. 离子操作模式

MUSBHDRC有两种主要操作模式——外围模式和主机模式。

在外设模式下，MUSBHDRC对发送和接收的所有USB数据包进行编码、解码、检查和引导。IN事务通过设备的TX FIFO处理，OUT事务通过其Rx FIFO处理。支持Control、Bulk、Isochronous和Interrupt事务。

在主机模式下，MUSBHDRC的行为方式取决于它是连接到另一个USB功能进行点对点通信，还是连接到集线器。当连接到另一个USB功能时，MUSBHDRC提供所需的一系列功能，以便在与该USB功能的点对点通信中充当主机。当连接到集线器时，它为同时受支持的多个设备提供了充当主机所需的设施。

当在主机模式下操作并用于与单个其他USB设备（可以是高速、全速或低速）的点对点通信时，MUSBHDRC可以支持Control、Bulk、Isochronous或Interrupt事务。IN事务通过Rx FIFO处理，OUT事务通过TX FIFO处理。除了对发送和接收的USB数据包进行编码、解码和检查外，MUSBHDRC还将自动调度等时端点和中断端点，以每*n*帧/微帧执行一次事务（如果选择了高带宽选项，则最多执行三次事务），其中*n*表示已为端点编程的轮询间隔。剩余的总线带宽在控制和批量端点之间平均共享（见第8.5.4节事务调度）。

当连接到集线器时，MUSBHDRC继续提供上述设施，但还需要对其进行详细编程：

- 目标设备的功能地址。
- 目标设备的操作速度（以便可以进行适当的速度转换）。
- 如果目标设备是通过高速集线器访问的全速或低速设备，则还需要使用集线器的功能地址和端口号对端点进行编程。

该设备可能需要将VBus充电至5V，作为连接的"A"设备（电源和默认主机），或者作为"B"设备（默认外围设备），通过将VBus充至2V来唤醒"A"装置。MUSBHDRC的输出指示何时需要这些充电选项。

MUSBHDRC最初是在主机模式下还是在外围模式下运行，取决于它是在"A"设备还是在"B"设备中使用，而这又取决于IDDIG输入是低还是高。当MUSBHDRC作为"A"设备运行时，它最初被配置为在主机模式下运行。当作为"B"设备运行时，MUSBHDRC最初配置为在外围模式下运行。但是，DevCtl寄存器中提供了一个"Host Req"位，CPU可以通过该位请求下次USB总线上没有活动时"B"设备成为主机。

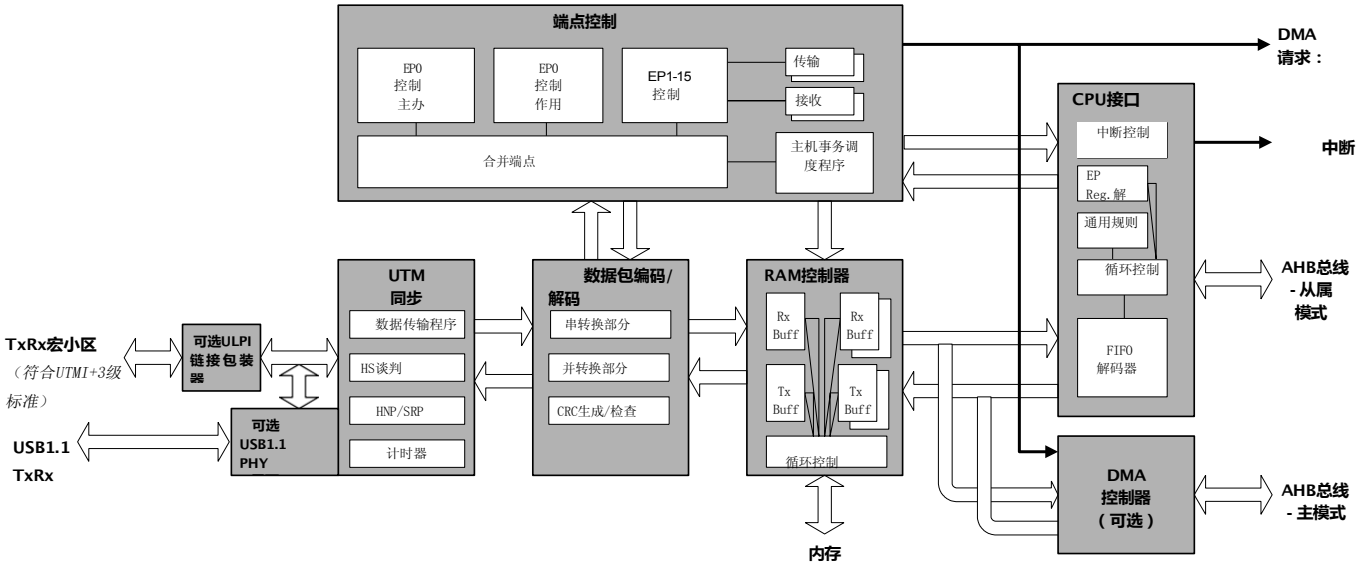
IDDIG输入反映了设备迷你AB插座的ID引脚的状态，IDDIG为低表示"A"插头，即作为"A"设备操作，IDDIG为高表示"B"插头和作为"B"设备操作。

关于MUSBHDRC是充当"A"设备还是充当"B"设备，以及它所连接的设备是高、全还是低速的信息，以及关于VBus相对于用于发出会话开始和会话结束信号的高和低电压阈值的电平的信息，也记录在DevCtl寄存器中。

第14节和第15节分别描述了会话请求和在连接任一端的设备之间传输主机/外围设备角色的过程。所进行的传输都遵循标准USB数据传输协议。

2.2. 集团

以下框图显示了MUSBHDRC内的主要功能块。（提供用于核心的任何桥接器的框图在**musbmhdrc/docs**目录中包含的该桥接器的单独规范中给出。）



2.3. 但是M S Y N C H R O N IZ A T IO N

UTM同步块的作用是在收发器宏小区60MHz时钟域和双重作用控制器的系统时钟CLK之间重新同步，该系统时钟CLK驱动核心的其余部分直到并包括CPU接口。这允许MUSBHDRC的其余部分以CPU总线速度运行，而不需要任何进一步的同步。该块还执行高速检测握手，并在与另一个USB OTG设备的点对点通信中处理HNP和SRP。

使用8位接口，该块首先将数据转换为16位，要求内核由运行频率至少超过30MHz的系统时钟驱动。系统时钟在域交叉上正确传输数据所需的实际最小频率是技术实现的函数。该实际最小频率在第4节中有详细定义。

2.4. PACKET ENCODING/DECODING

分组编码/解码块生成要发送的分组的报头，并对接收到的分组上的报头进行解码。它还为要发送的分组生成CRC，并检查接收到的分组的CRC。

2.5. ENdpoint Controllers

使用了两个控制器状态机：一个用于端点0上的控制传输，另一个用于终结点1到15上的批量/中断/等时事务。

2.6. CPU INTERFACE

CPU接口允许访问每个端点的控制/状态寄存器和FIFO。当数据包成功传输或接收时，以及当内核进入挂起模式或从挂起模式恢复时，它也会向CPU产生中断。

MUSBHDRC提供的接口是一个32位同步接口，遵循AMBA AHB总线接口的指定设计。可以通过向核心添加适当的封装器来实现与其他总线标准的接口。

2.7. RAM CONTROLLER

RAM控制器为单个同步单端口RAM块提供接口，用于缓冲CPU和USB之间的数据包。它从端点控制器获取FIFO指针，将它们转换为RAM块内的地址指针，并生成RAM访问控制信号。

2.8. DMA CONTROLLER Support

如果需要，MUSBHDRC可以包括用于有效加载/卸载端点FIFO的多通道DMA控制器。此DMA控制器最多可配置8个通道。

DMA控制器有它自己的控制寄存器块和它自己的中断控制器。它支持两种操作模式，每个通道都可以独立编程用于操作模式

替代地，MUSBHDRC可以与外部DMA控制器集成，用于有效地加载/卸载端点FIFO。MUSBHDRC为每个端点输出DMA请求信号。

2.9. TREE DIAGRAM

以下树状图显示了MUSBHDRC的层次结构。（与提供给核心使用的任何桥接器相关的模块显示在musbmhdrc/docs目录中该桥接器的单独规范中给出的等效树状图中。）*注意：*当配置为与可选的ULPI链接包装器一起使用时，核心还包括一个lpictl模块，该模块提供ULPI控制寄存器（详见ULPI链接包应用程序说明，以文件**musbhdrc_ULPI_an.pdf的形式**提供，该文件包含在**docs**目录中）。

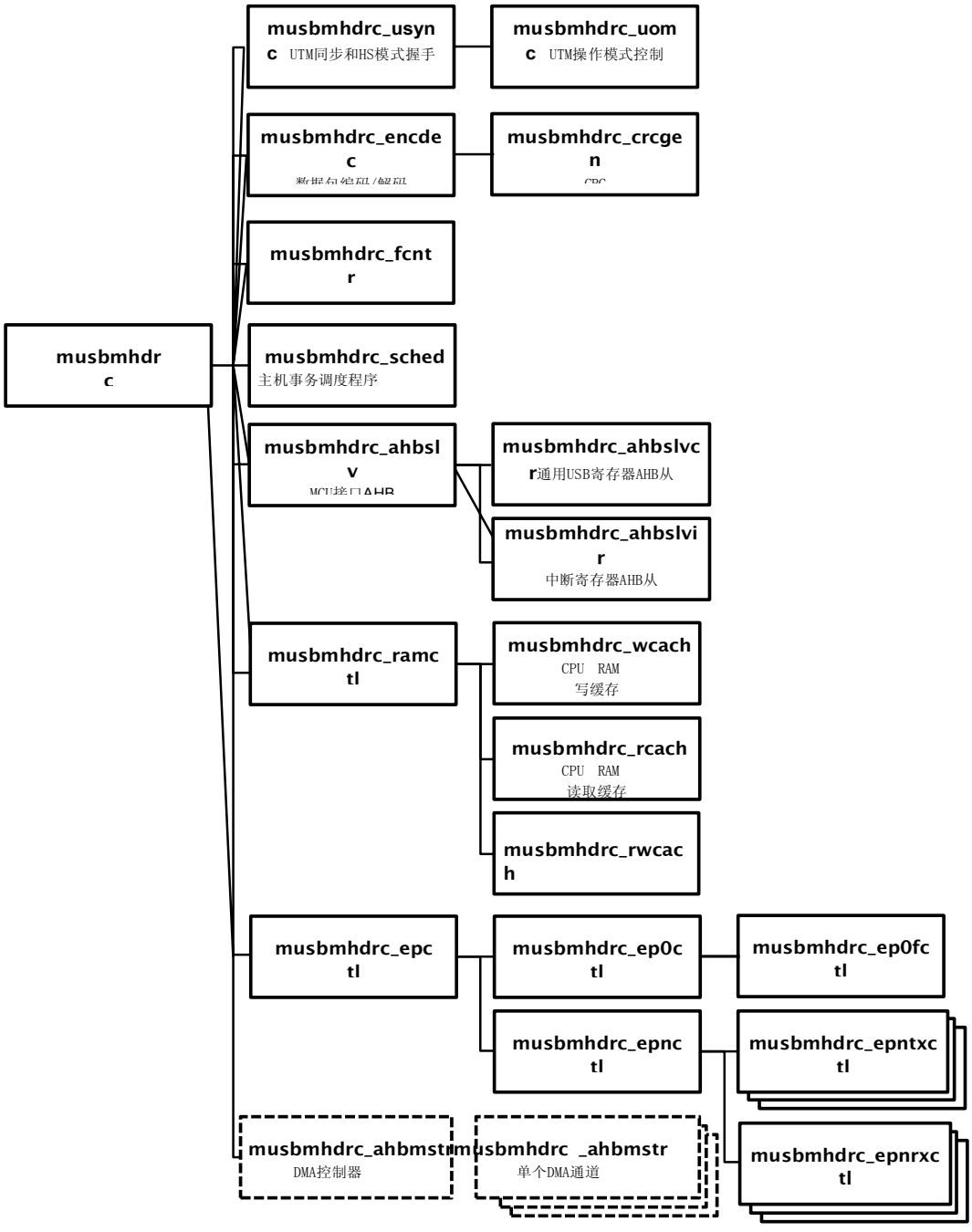


图1

MUSBHDRC可以针对以下方面进行配置：

1. 除了端点0之外，TX端点的数量。即1、3、5、7、11或15个额外的TX端点
2. 除了端点0之外，Rx端点的数量。即1、3、5、7、11或15个附加Rx端点
3. 这些端点中是否有任何一个支持高带宽等时传输。
4. 与每个端点相关联的FIFO大小（不包括端点0），或者动态分配给不同端点的总RAM大小（见第19节）。
5. 哪些FIFO（如果有的话）在TX端点和相应编号的Rx端点之间共享（除非动态调整大小）。
6. 批量传输 是否需要数据包的自动拆分/组合选项（见第8.4.1.4节和第8.4.2.4节）。
7. 使用UTMI+接口的核心版本 是否包括UTMI+VControl和VStatus寄存器，以及这些寄存器的大小（如果包括）（见第3.6.1节和第3.6.2节）。
8. 是否使用MUSBHDRC对多点（集线器支持）的支持。如果使用了额外的逻辑，则允许MUSBHDRC通过USB集线器管理连接。
9. 使用内置DMA控制器时支持的DMA通道数（如果使用外部DMA控制器，则为0）。

除了端点0之外，最多可以有15个TX端点和/或最多15个Rx端点。当在外围模式中使用MUSBHDRC时，每个TX端点用作IN端点，当在主机模式中使用MUSBMHDRC时用作OUT端点。类似地，当在外围模式中使用MUSBHDRC时，每个Rx端点被用作OUT端点，当在主机模式中使用MUSBMHDRC时被用作in端点。

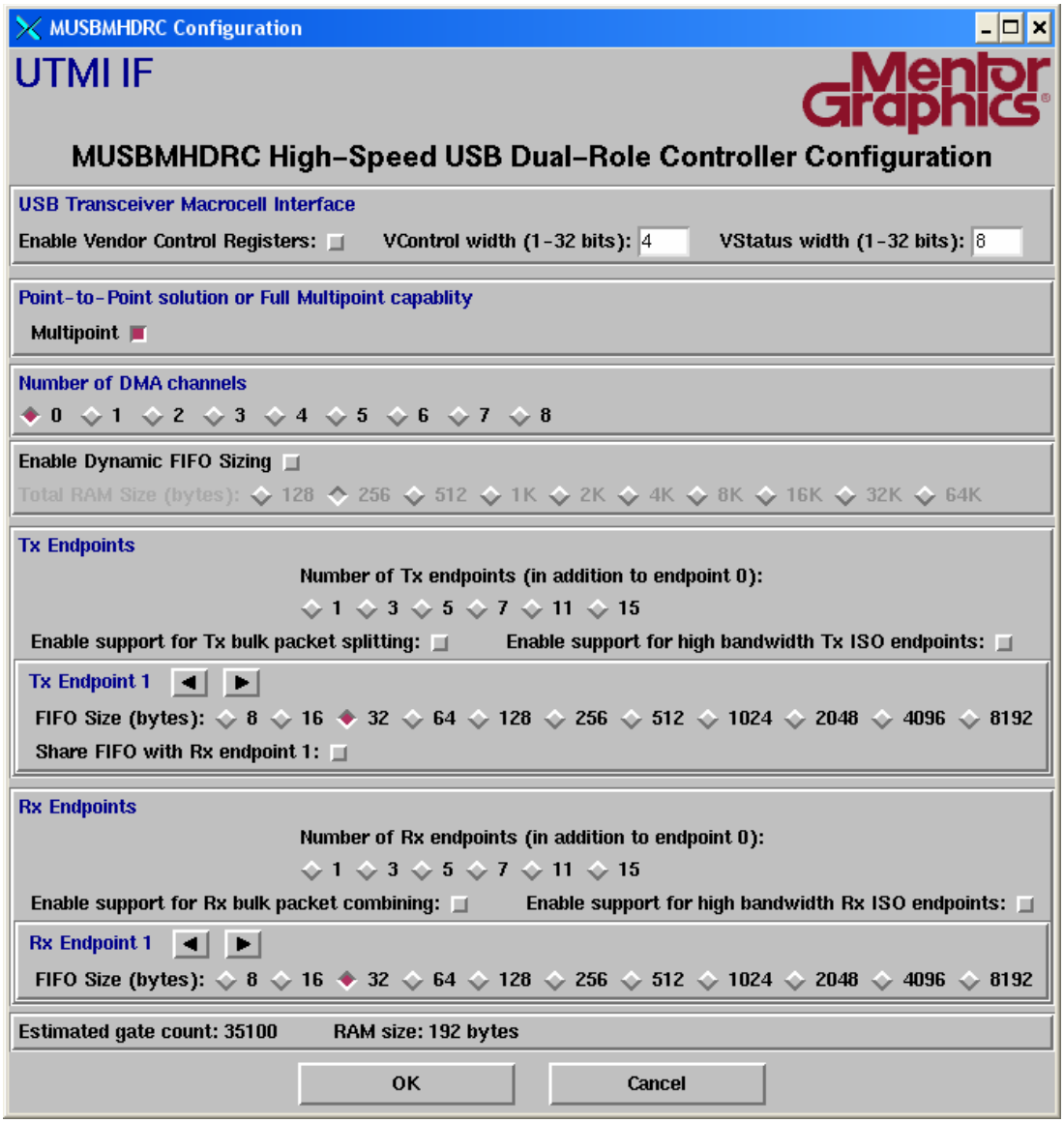
端点0的FIFO大小要求为64字节，并将缓冲一个数据包。其他端点的FIFO可以指定为8、16、32、64、128、256、512、1024、2048、4096或8192字节，并且除了8字节的FIFO之外，可以用于缓冲一个或两个分组。

但是，大于2048字节的FIFO大小只能与高带宽等时端点一起使用。如有需要，MUSBHDRC将自动将高达3072字节（3k）的数据包拆分／重新组合为2个或3个较小的数据包，以便通过总线进行传输／接收。

（*注意*：双缓冲对于批量或中断传输是可选的，但对于等时传输通常是必需的。您还应该注意USB规范对全速操作中批量、中断和等时传输的数据包大小的限制。）

单独的FIFO可以与每个端点相关联：或者，具有相同端点编号的TX端点和Rx端点可以配置为使用相同的FIFO，例如，以减少所需的RAM块的大小。

进一步的配置选项可以与提供用于MUSBHDRC核心的任何桥接器相关联（在**MUSBMHDRC/docs**目录中包括的该桥接器的单独规范中描述）。



在模拟或合成之前，通过运行图形用户界面脚本（类似于下面所示的脚本）来执行配置。这些步骤在《MUSBHDRC用户指南》第5节和模拟目录中包含的**config.readme**文件中进行了说明。配置文件（*_cfg.v）不应直接手动编辑。用户创建的配置文件要求是仅由交付的配置GUI修改的交付配置文件的结果。

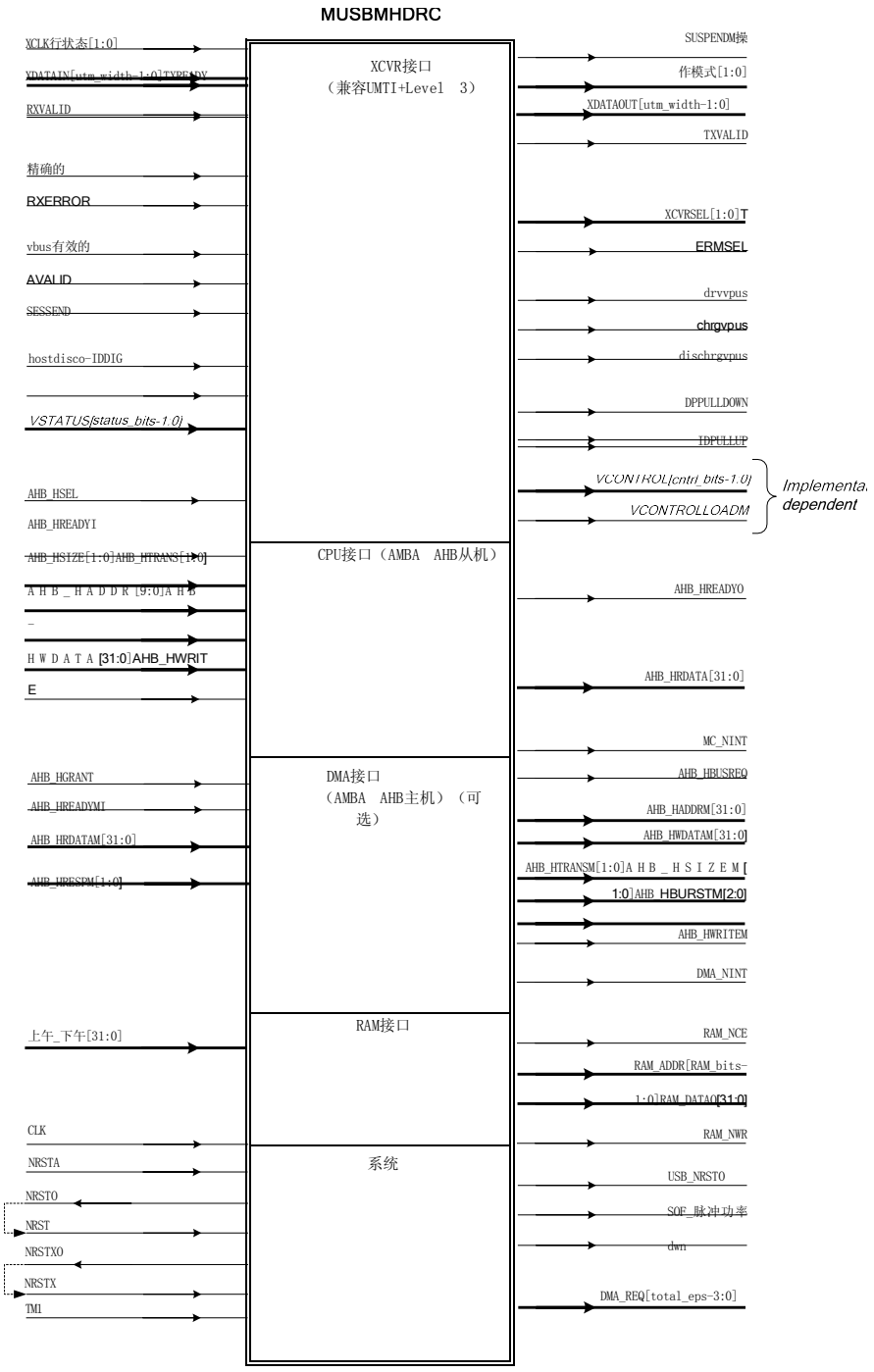
请注意：在向核心添加桥接器/包装器的情况下，可能会提供单独的脚本。有关详细信息，请参阅模拟目录中包含的**config.readme**文件或相应的桥接器/包装器规范。还提供了一个特殊的脚本（**config_fsp.tcl**），用于配置使用可选USB 1.1 PHY接口的核心。（第8.1节提供了更多相关信息。）

您还应注意，配置屏幕显示包括所选核心配置的估计门数和所需RAM数量。

在首次显示配置GUI时，会向core发出显示的配置。

请注意：*本节介绍MUSBHDRC核心本身的信号I/O。第8.1节描述了与核心一起使用可选USB 1.1 PHY接口时的信号I/O。添加桥接器时的信号I/O在**musbmhdrc/docs**目录中包含的适当桥接器规范中进行了描述。*

MUSBHDRC核心最多有438个外部信号；182个输入和256个输出。所有输入都在相关时钟的正（上升）沿上采样，并且输出随正时钟沿而变化。



<table>
<tr><th colspan="3">UTMI+接口信号（3级）</th></tr>
<tr><th>信号</th><th>类型</th><th>说明</th></tr>
<tr><td>XCLK</td><td>输入</td><td>收发器宏小区时钟。60MHz。</td></tr>
<tr><td>悬架dm</td><td>输出</td><td>异步挂起模式指示器（源自CLK和XCLK的信号
触发器）。当通过电源寄存器的位0启用时，当设备处于挂起模式时变低。否则会很高。（旨在驱动UTMI PHY。）</td></tr>
<tr><td>线性状态[1:0]</td><td>输入</td><td>显示单端接收器的当前状态。LINESTATE[0]反映了D+的状态；LINESTATE[1]反映了D-的状态。因此
<table>
<tr><th colspan="2">线性状态[1:0]</th><th></th></tr>
<tr><td>0</td><td>0</td><td>SE0</td></tr>
<tr><td>0</td><td>1.</td><td>“J”状态</td></tr>
<tr><td>1.</td><td>0</td><td>’K’状态</td></tr>
<tr><td>1.</td><td>1.</td><td>SE1</td></tr>
</table></td></tr>
<tr><td>操作模式[1:0]</td><td>输出</td><td>操作模式选择器
<table>
<tr><th colspan="2">操作模式[1:0]</th><th></th></tr>
<tr><td>0</td><td>0</td><td>正常操作</td></tr>
<tr><td>0</td><td>1.</td><td>非驾驶</td></tr>
<tr><td>1.</td><td>0</td><td>位填充和NRZI编码被禁用</td></tr>
<tr><td>1.</td><td>1.</td><td>保留</td></tr>
</table></td></tr>
<tr><td>XDATAIN[7:0]</td><td>输入</td><td>收到的数据。</td></tr>
<tr><td>扩展数据[7:0]</td><td>输出</td><td>要传输的数据。。</td></tr>
<tr><td>TXVALID</td><td>输出</td><td>传输数据有效。表示存在要传输的有效数据。</td></tr>
<tr><td>TXREADY</td><td>输入</td><td>传输数据准备就绪。表示变送器需要数据。</td></tr>
<tr><td>RXVALID</td><td>输入</td><td>接收数据有效。表示已接收到有效数据。</td></tr>
<tr><td>精确的</td><td>输入</td><td>指示正在接收有效的数据包。</td></tr>
<tr><td>RXERROR</td><td>输入</td><td>指示正在接收的数据包将由于错误而中止。</td></tr>
<tr><td>XCVRSEL[1:0]</td><td>输出</td><td>收发器选择。
<table>
<tr><th colspan="2">XCVRSEL[1:0]</th><th></th></tr>
<tr><td>0</td><td>0</td><td>HS收发器</td></tr>
<tr><td>0</td><td>1.</td><td>FS收发器</td></tr>
<tr><td>1.</td><td>0</td><td>LS收发器</td></tr>
<tr><td>1.</td><td>1.</td><td>FS收发器，LS包</td></tr>
</table></td></tr>
<tr><td>TERMSEL</td><td>输出</td><td>选择终止。0时，启用高速终止；当1时，启用全速终止。注：可用于切换D+上的上拉电阻。</td></tr>
<tr><td>VBUSVALID</td><td>输入</td><td>VBus与所选VBus有效阈值（要求在4.4V和4.75V之间）的比较。1=高于VBus有效阈值，0=低于VBus有效阀值。</td></tr>
<tr><td>AVALID</td><td>输入</td><td>VBus与“B”设备的会话有效阈值（要求在0.8V和2V之间）的比较。1=高于会话有效阈值，0=低于会话有效阈值。</td></tr>
<tr><td>SESSEND</td><td>输入</td><td>VBus与会话结束阈值（要求在0.2V和0.8V之间）相比。0=高于会话结束阈值，1=低于会话结束阈值。</td></tr>
<tr><td>DRVVBUS</td><td>输出</td><td>VBus电源启用（当MUSBHDRC作为“A”设备运行时使用）。</td></tr>
<tr><td>chrgvpus</td><td>输出</td><td>充电VBus（当MUSBHDRC作为“B”设备运行时，在会话请求期间使用）。</td></tr>
<tr><td>DISCHRGVBUS</td><td>输出</td><td>放电VBus（由“B”设备使用，以确保在启动会话请求协议（SRP）之前VBus足够低）。</td></tr>
</table>

UTMI+接口信号（3级）		
信号	**类型**	**说明**
hostdisco	输入	（*仅限主机模式*。）发生高速断开时需要断言（根据UTMI+规范）。*注*：全速/低速连接通过LINESTATE信号进行监控。
DPPULLDOWN	输出	启用D+线上收发器内的下拉电阻器。当MUSBHDRC作为外设运行时为低；当MUSBHDRC作为主机操作时为高。
dm下拉	输出	启用D线上收发器内的下拉电阻器。当MUSBHDRC用于点对点通信时，需要较高。
IDDIG	输入	表示MUSBHDRC连接器类型。高=>B型，低=>A型。
IDPULLUP	输出	启用IDDIG信号生成。
VSTATUS[cntrl_bits-1:0]	输入	PHY状态数据（最多可配置32位宽）——*如果实施*。
VCONTROL[cntrl_bits-1:0]	输出	PHY控制数据（最多可配置32位宽）——*如果实现*。
vcontrollodadm	输出	激活低信号，在读取新的控制信息时断言——*如果实施*。
CPU接口信号（AMBA AHB从机）*		
AHB_HSEL	输入	AHB选择。取高以选择MUSBHDRC设备。
AHB_HREADYI	输入	AHB就绪输入。
AHB_HREADYO	输出	AHB就绪输出。
AHB_HSIZE[1:0]	输入	AHB传输大小。
AHB_HTRANS[1:0]	输入	AHB传输类型。
AHB_HADDR[9:0]	输入	AHB地址总线。
AHB_HWDATA[31:0]	输入	AHB写入数据总线。
AHB_HRDATA[31:0]	输出	AHB读取数据总线。
AHB_HWRITE	输入	AHB写入未读取
MC_NINT	输出	CPU中断。低激活。
DMA接口信号（AMBA AHB主控）-可选		
AHB_HGRANT	输入	AHB总线主授权。
AHB_HREADYMI	输入	AHB主准备输入。
AHB_HRDATAM[31:0]	输入	AHB读取数据总线（主模式）
AHB_HRESPM[1:0]	输入	AHB响应（主模式）。
AHB_HBUSREQ	输出	AHB总线主请求。
AHB_HADDRM[31:0]	输出	AHB地址总线（主模式）。
AHB_HWDATAM[31:0]	输出	AHB写入数据总线（主模式）。
AHB_HTRANSM[1:0]	输出	AHB传输类型（主模式）。
AHB_HSIZEM[1:0]	输出	AHB传输大小（主模式）。
AHB_HBURSTM[2:0]	输出	AHB突发模式（主模式）。
AHB_HWRITEM	输出	AHB写入未读取（主模式）
DMA_NINT	输出	DMA控制器中断。低激活。

*在核心与桥接器一起使用的情况下，设备将具有一组不同的CPU接口信号——详见该桥接器的单独规范，该规范包含在**musbmhdrc/docs**目录中。

RAM接口信号		
信号	**类型**	**说明**
RAM_ADDR[RAM_bits-1:0]	输出	RAM地址总线。宽度取决于配置的端点的数量和类型。
上午_下午[31:0]	输入	RAM数据输入总线。
RAM_DATAO[31:0]	输出	RAM数据输出总线。
RAM_NCE	输出	RAM选择。低激活。
RAM_NWR	输出	RAM写入启用。低激活。
系统信号		
CLK	输入	系统时钟（由AHB总线时钟提供）。此时钟至少需要>30MHz。实际最小值见第4节。
NRSTA	输入	异步加电复位，低激活。
NRST	输入	与CLK同步复位。低激活。通常连接到输出NRSTO。
NRSTX	输入	与XCLK同步复位。低激活。通常连接到输出NRSTXO。
TM1	输入	测试模式。由提供的测试台使用，以减少计时器长度，从而减少测试台运行所需的时间。对于正常操作，此信号应连接为低电平。
NRSTO	输出	在与CLK时钟域同步之后，该信号等于异步输入NRSTA。低激活。该信号也可以通过寄存器7Fh（软复位）被断言。通常连接到输入NRST。
NRSTXO	输出	该信号等于同步到XCLK时钟域之后的异步输入NRSTA。低激活。该信号也可以通过寄存器7Fh断言 （软重置）。通常连接到输入NRSTX。
USB_NRSTO	输出	USB功能重置。低激活。当通过USB信号重置功能控制器时，该重置被断言。
SOF_PULSE	输出	帧同步脉冲。脉冲长度为1 CLK周期，脉冲频率在全速/低速模式下为1kHz，在高速模式下为8kHz，在外围模式下与接收到的SOF/uSOF数据包同步。
POWERDWN	输出	当CLK可以停止以节省电力时断言。（*注*：源自CLK和XCLK触发器、AVALID、VBUSVALID和 LINESTATE。）
DMA_REQ[总计_eps-3:0]	输出	DMA端点请求，每个附加的Rx端点和TX端点一个。如果总共定义了*N*个TX端点和*M*个Rx端点，则DMA_REQ[0]。。。 DMA_REQ[*N-2*]与TX端点1相关联。。。*N*-1；DMA_REQ[*N-1*]。。。 DMA_REQ[*N+M*-3]与Rx端点1相关联。。。*M*-1。

3. 记录

3.1. MUSBMHDRC REGISTER MAP

MUSBHDRC寄存器映射分为以下几个部分：

通用USB寄存器（00h–0Fh）–这些寄存器为整个核心提供控制和状态。

索引端点控制/状态寄存器（10h–1Fh）–这些寄存器为当前选定的端点提供控制和状态。映射到此部分的寄存器取决于核心是处于外围模式（DevCtl.D2=0）还是处于主机模式（DevCtl.D2=1）以及索引寄存器的值。

FIFOs（20h–5Fh）–此地址范围提供对端点FIFOs的访问。

附加控制和配置寄存器（60h–7Fh）–这些寄存器提供附加的设备状态和控制。

目标端点控制寄存器（80h–FFh）–当在配置GUI中启用多点选项时，这些寄存器为每个端点提供目标功能和集线器地址的详细信息。只有启用多点选项时，才能访问这些寄存器。

非索引端点控制/状态寄存器（100h及以上）–在10h-1Fh时可用的寄存器，可独立于索引寄存器的设置进行访问。100h–10Fh EP0寄存器；110h–11Fh EP1寄存器；120h–12Fh EP2；等等

DMA控制寄存器（200h及以上）——只有当设计被合成为包括可选DMA控制器时，这些寄存器才会出现（见第17节）。

RqPktCount寄存器（304h–33Ch）——这些寄存器在主机模式下与AutoReq一起使用（见第8.5.2节）。

DPktBufDis寄存器（340h–343h）–这些寄存器提供对禁用双数据包缓冲的直接用户控制。

C_T_UCH寄存器（344h–345h）–这些寄存器设置Chirp超时定时器。

C_T_HSRTN寄存器（346h–347h）–这些寄存器设置从高速恢复信号结束到启用UTM正常操作模式的延迟。

生成的内存映射如下图所示。

MUSBHDRC内存映射

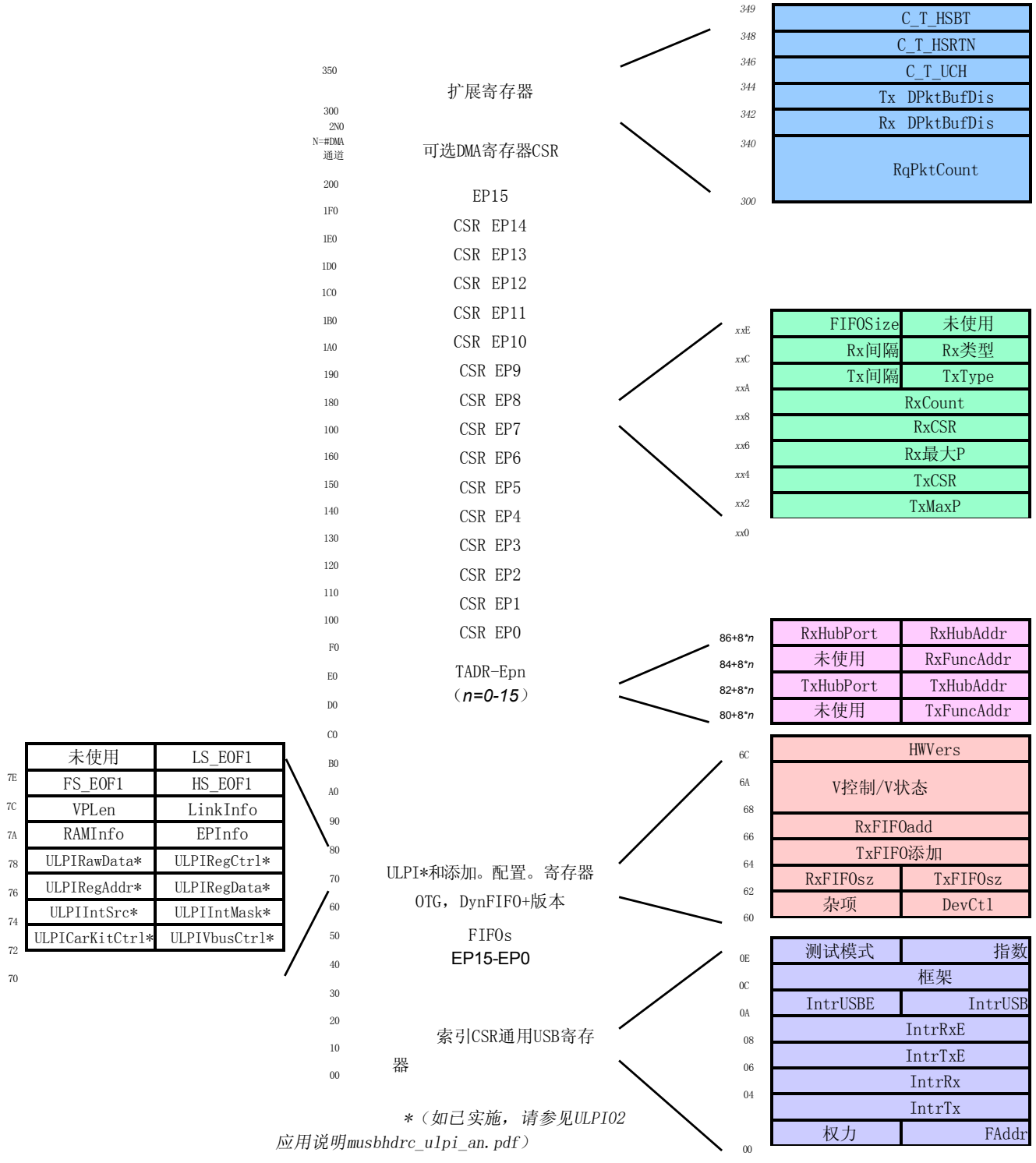


图2:

注：与提供给MUSBHDRC核心使用的任何桥接器相关的附加寄存器或使用该桥接器对以下寄存器的任何更改将在**MUSBMHDRC/docs**目录中包含的桥接器的单独规范中进行描述。

MUSBHDRC寄存器映射：通用USB寄存器（00h–0Fh）			
地址	**名称**	**说明**	**请参阅第节**
00	FAddr	函数地址寄存器。	3.2.1
01	权力	电源管理寄存器。	3.2.2
02，03	IntrTx	端点0加上TX端点1到15的中断寄存器。	3.2.3
04，05	IntrRx	Rx端点1至15的中断寄存器。	3.2.4
06，07	IntrTxE	IntrTx的中断启用寄存器。	3.2.5
08，09	IntrRxE	IntrRx的中断启用寄存器。	3.2.6
0A	IntrUSB	用于通用USB中断的中断寄存器。	3.2.7
0B	IntrUSBE	IntrUSB的中断启用寄存器。	3.2.8
0C，0D	框架	帧编号。	3.2.9
0E	指数	索引寄存器，用于选择端点状态和控制寄存器。	3.2.10
0F	测试模式	启用USB 2.0测试模式。	3.2.11

MUSBHDRC寄存器映射：索引寄存器–外围模式（10h–1Fh） （当DevCtl.D2=0时，索引寄存器选择的端点的控制状态寄存器）			
地址	**名称**	**说明**	**请参阅第节**
10，11	TxMaxP	外围TX端点的最大数据包大小。（索引寄存器设置为仅选择端点1–15）	3.3.7
12，13	CSR0L/H	终结点0的控制状态寄存器。（索引寄存器设置为选择终结点0）	3.3.1
	TxCSRL/H	外围TX端点的控制状态寄存器。（索引寄存器设置为选择端点1–15）	3.3.2
14，15	Rx最大P	外围Rx端点的最大数据包大小。（索引寄存器设置为仅选择端点1–15）	3.3.10
16，17	RxCSRL/H	外围Rx端点的控制状态寄存器。（索引寄存器设置为仅选择端点1–15）	3.3.11
18，19	计数0	终结点0 FIFO中接收的字节数。（索引寄存器设置为选择终结点0）	3.3.3
	RxCount	要从外围Rx端点FIFO读取的字节数。（索引寄存器设置为选择端点1–15）	3.3.13
1A–1B	–	*保留*。返回的值受主机模式下使用的影响（请参阅下页）。	
1C–1E	–	*未使用*，始终返回0。	
1F	ConfigData	返回核心配置的详细信息。（索引寄存器设置为选择终结点0。）	3.3.5

	FIFOSize	返回所选Rx FIFO和TX FIFO的配置大小（仅限端点1–15）。	3.3.18

MUSBHDRC寄存器映射：索引寄存器–主机模式（10h–1Fh） （当DevCtl.D2=1时，索引寄存器选择的端点的控制状态寄存器）			
地址	**名称**	**说明**	**请参阅第节**
10, 11	TxMaxP	主机TX终结点的最大数据包大小。（索引寄存器设置为仅选择端点1–15）	3.3.7
12, 13	CSR0L/H	终结点0的控制状态寄存器。（索引寄存器设置为选择终结点0）	3.3.1
	TxCSRL/H	主机TX终结点的控制状态寄存器。（索引寄存器设置为选择端点1–15）	3.3.2
14, 15	Rx最大P	主机Rx终结点的最大数据包大小。（索引寄存器设置为仅选择端点1–15）	3.3.10
16, 17	RxCSRL/H	主机Rx终结点的控制状态寄存器。（索引寄存器设置为仅选择端点1–15）	3.3.11
18, 19	计数0	终结点0 FIFO中接收的字节数。（索引寄存器设置为选择终结点0）	3.3.3
	RxCount	要从主机Rx端点FIFO读取的字节数。（索引寄存器设置为选择端点1–15）	3.3.13
1A	类型0	定义端点0的速度。（索引寄存器设置为选择终结点0）	3.3.4
	TxType	设置主机TX端点的事务协议、速度和外围端点编号。（索引寄存器设置为选择端点1–15）	3.3.14
1B	NAK限制0	设置终结点0上的NAK响应超时。（索引寄存器设置为选择终结点0）	3.3.6
	TxInterval	设置主机TX终结点的中断/ISOC事务的轮询间隔或批量事务的NAK响应超时。（索引寄存器设置为仅选择端点1–15）	3.3.15
1C	Rx类型	设置主机Rx终结点的事务协议、速度和外围终结点编号。（索引寄存器设置为仅选择端点1–15）	1.1.1
1D	RxInterval	设置主机Rx端点的中断/ISOC事务的轮询间隔或批量事务的NAK响应超时。（索引寄存器设置为仅选择端点1–15）	3.3.17
1E	–	*未使用*，总是返回0。	
1F	ConfigData	返回核心配置的详细信息。（索引寄存器设置为选择终结点0。）	3.3.5
	FIFOSize	返回所选Rx FIFO和TX FIFO的配置大小（仅限端点1–15）。	3.3.18

MUSBHDRC注册地图：FIFOs（20小时–5小时）			
地址	**名称**	**说明**	**请参阅第节**
20–5英尺	FIFOx	端点0–15的FIFOs。	3.4

MUSBHDRC寄存器映射：附加控制和配置寄存器（60h–7Fh）				
地址	**名称**	**说明**		**请参阅第节**
60	DevCtl	OTG设备控制寄存器。		3.2.12
61	杂项	杂项登记册		3.2.13
62	TxFIFOsz	TX端点FIFO大小	*仅在选择动态FIFO大小选项时使用。否则返回0。*	3.3.18
63	RxFIFOsz	Rx端点FIFO大小		
64，65	TxFIFO添加	TX端点FIFO地址		
66，67	RxFIFOadd	Rx端点FIFO地址		
68–6B	V控制/V状态	UTMI+PHY供应商寄存器		3.6.1，3.6.2
6C、6D	HWVers	硬件版本号寄存器		3.6.3
6E、6F	–	*未使用*		
70 – 77	–	*ULPI寄存器，仅在使用ULPI链接包装器的情况下实现。参见ULPI应用说明* ***musbhdrc_ULPI_an.pdf***。		
78	EPInfo	有关TX和Rx端点数量的信息。		3.7.1
79	RAMInfo	有关RAM宽度和DMA通道数量的信息。		3.7.2
7A	LinkInfo	有关要应用的延迟的信息。		1.1.1
7B	VPLen	VBus脉冲充电的持续时间。		3.7.4
7C	HS_EOF1	高速事务上可用的时间缓冲区。		3.7.5
7天	FS_EOF1	全速事务上可用的时间缓冲区。		3.7.6
7E	LS_EOF1	低速事务上可用的时间缓冲区。		3.7.7
7英尺	SOFT_RST	软重置。		3.7.8

MUSBHDRC寄存器映射：目标地址寄存器（80h–FFh） （只有在配置GUI中启用了多点选项时，这些寄存器才有效）				
80+8**n*	TxFuncAddr	传输端点*n*功能地址（仅限主机模式）	仅多点	3.5.1
81+8**n*	–	*未使用*，总是返回0。	仅多点	
82+8**n*	TxHubAddr	传输终结点*n*集线器地址（仅限主机模式）	仅多点	3.5.2
83+8**n*	TxHubPort	传输终结点*n*集线器端口（仅限主机模式）	仅多点	3.5.3
84+8**n*	RxFuncAddr	接收终结点*n*功能地址（仅限主机模式）	仅多点	3.5.1
85+8**n*	–	*未使用*，总是返回0。	仅多点	
86+8**n*	RxHubAddr	接收终结点*n*集线器地址（仅限主机模式）	仅多点	3.5.2
87+8**n*	RxHubPort	接收终结点*n*集线器端口（仅限主机模式）	仅多点	3.5.3

MUSBHDRC寄存器映射：扩展寄存器（200h–27Ch）				
地址	**名称**	**说明**		**请参阅第节**
200小时	DMA_INTR	DMA中断寄存器。	只有当MUSBHDRC配置为使用至少一个内部DMA通道时，DMA寄存器才可用。每个通道有一组寄存器。	3.9.1
204小时+（n-1）*10小时	DMA_CNTL	DMA通道n的DMA控制寄存器（通道1到8）。		0
208小时+（n-1）*10小时	DMA_ADDR	DMA通道n的DMA地址寄存器（通道1至8）。		3.9.3
20Ch+（n-1）*10h	DMA_COUNT	DMA通道n（通道1到8）的DMA计数寄存器。		3.9.4

MUSBHDRC寄存器映射：扩展寄存器（304h–347h）			
地址	**名称**	**说明**	**请参阅第节**
300+4**n*	RqPktCount	接收端点*n*请求的数据包数（仅端点1–15）	3.8.1
340, 341	Rx DPktBufDis	Rx端点1至15的双数据包缓冲区禁用寄存器	3.8.2.1
342, 343	TX DPktBufDis	TX端点1至15的双数据包缓冲区禁用寄存器	3.8.2.2
344, 345	C_T_UCH	此寄存器设置Chirp超时定时器	3.8.3
346, 347	C_T_HSRTN	该寄存器设置从高速恢复信令结束起的延迟，以启用UTM正常操作模式。	3.8.4
348	C_T_HSBT	此寄存器指定以64个高速位时间为增量添加到最小高速超时周期（736位时间）的值。	3.8.5

注：在以下位描述中：

“r”表示该位为只读
“rw”表示该位既可以读取也可以写入
“set”表示只能写入位来设置它
“r/set”表示可以读取或设置位，但不能清除
“清除”表示只能写入位以清除它
“r/clear”表示可以读取或清除位，

3.2. 注册通信

–在地址顺序中描述。

3.2.1. FADDR

FAddr是一个8位寄存器，应该使用事务外围部分的7位地址写入。

当MUSBHDRC在外设模式下使用时（DevCtl.D2=0），该寄存器应写入通过SET_address命令接收的地址，然后该命令将用于解码后续令牌包中的功能地址。

注意：仅限外围模式！！

核心配置多点支持：此寄存器仅适用于MUSBHDRC处于外设模式时执行的操作。在主机模式下，此寄存器被忽略。

核心配置不支持多点：此寄存器适用于当MUSBHDRC为主机和外设模式时执行的操作。

地址：00h；复位值：8'000

	D7	D6	D5	D4	D3	D2	D1	D0
	0	**函数地址**						
来自CPU	r	rw	rw	rw	rw	rw	rw	rw
从USB	-	r	R	r	r	r	r	r

一点	名称	作用
D7	**-**	*未使用，总是返回0。*
D6–D0	**函数加法器**	函数地址。

3.2.2. POWER

Power是一个8位寄存器，用于控制挂起和恢复信令以及MUSBHDRC的一些基本操作方面。

地址：01h；复位值：8'h20

		D7	D6	D5	D4	D3	D2	D1	D0
		国际标准化组织使现代化	**软连接器**	**HS激活**	**HS模式**	**重置**	**简历**	**待机模式**	**启用Suspend M**
Periphera	*中央处理器*	rw	rw	rw	r	r	rw	r	rw
模式	*通用串口*	r	r	r	rw	rw	r	rw	r

主办	*中央处理器*		–	rw	r	rw	rw	设置	rw
模式	*通用串口总线*	–	–	r	rw	rw	r/台	清楚的	r

一点	名称	作用
D7	**ISO更新**	当由CPU设置时，MUSBHDRC将在发送数据包之前，从设置TxPktRdy开始等待SOF令牌。如果在SOF令牌之前接收到IN令牌，则将发送零长度数据包。*注意：仅在外围模式下有效。此外，此位仅影响执行等时传输的端点。*
D6	**软连接器**	如果启用了软连接/断开连接功能，则当CPU设置该位时，USB D+/D-线将启用，当CPU清除该位时将三态。（参见第8.2节）*注：仅在外围模式下有效。*
D5	**HS激活**	当由CPU设置时，当集线器重置设备时，MUSBHDRC将协商高速模式。如果未设置，设备将仅在全速模式下运行。
D4	**HS模式**	设置时，此只读位表示USB重置期间成功协商的高速模式。在外围模式中，当USB重置完成时（如USB重置中断所示），该选项将变为有效。在主机模式下，当重置位被清除时变为有效。在会话期间保持有效。*注：*在确定要选择的传输速度时，考虑了Tiny-J信号。
D3	**重置**	当总线上存在复位信号时，设置此位。*注意：此位在主机模式下是从CPU读取/写入，但在外围模式下是只读。*
D2	**简历**	由CPU设置，以在设备处于挂起模式时生成恢复信号。在外设模式下，CPU应在10ms（最大15ms）后清除此位，以结束Resume信令。在主机模式下，CPU应在20ms后清除此位。
D1	**待机模式**	在主机模式下，此位由CPU设置为进入挂起模式。在外设模式中，此位在进入挂起模式时设置。当CPU读取中断寄存器或设置上面的恢复位时，它被清除。
D0	**启用SuspendM**	由CPU设置以启用SUSPENDM输出。

3.2. 3. I N T R T X

IntrTx是一个16位只读寄存器，用于指示端点0和TX端点1-15的当前活动中断。*注意：*与尚未配置的端点相关的位将始终返回0。还要注意，当读取此寄存器时，所有活动中断都会被清除。

*地址：*02h；*重置值：*16'0000

	D15	D14	D13	D12	D11	D10	D9	D8
	EP15 Tx	**EP14 Tx**	**EP13 Tx**	**EP12 Tx**	**EP11 Tx**	**EP10 Tx**	**EP9 Tx**	**EP8 Tx**
来自CPU从USB	r 设置	r 设置	r 设置	r 设置	r 设置	r 设置	r 设置	r 设置

	D7	D6	D5	D4	D3	D2	D1	D0
	EP7 Tx	**EP6 Tx**	**EP5 Tx**	**EP4 Tx**	**EP3 Tx**	**EP2 Tx**	**EP1 Tx**	**EP0**
来自CPU从USB	r 设置	r 设置	r 设置	r 设置	r 设置	r 设置	r 设置	r 设置

一点	名称	作用
D15	**EP15 TX**	TX端点15中断。
D14	**EP14 TX**	TX端点14中断。
D13	**EP13 TX**	TX端点13中断。
D12	**EP12 TX**	TX端点12中断。
D11	**EP11 TX**	TX端点11中断。
D10	**EP10 TX**	TX端点10中断。
D9	**EP9 TX**	TX端点9中断。
D8	**EP8 TX**	TX Endpoint 8中断。
D7	**EP7 TX**	TX Endpoint 7中断。
D6	**EP6 TX**	TX Endpoint 6中断。
D5	**EP5 TX**	TX Endpoint 5中断。
D4	**EP4 TX**	TX Endpoint 4中断。
D3	**EP3 TX**	TX端点3中断。
D2	**EP2 TX**	TX端点2中断。
D1	**EP1 TX**	TX端点1中断。
D0	**EP0**	终结点0中断。

3.2.4. INTRRX

IntrRx是一个16位只读寄存器，用于指示Rx端点1–15的哪些中断当前处于活动状态。*注意*：与尚未配置的端点相关的位将始终返回0。还要注意，当读取此寄存器时，所有活动中断都会被清除。

地址：04h；*重置值*：16'0000

	D15	D14	D13	D12	D11	D10	D9	D8
	EP15 Rx	**EP14 Rx**	**EP13 Rx**	**EP12 Rx**	**EP11 Rx**	**EP10处方**	**EP9 Rx**	**EP8 Rx**
来自CPU	r	r	r	r	r	r	r	r
从USB	设置	设置	设置	设置	设置	设置	设置	设置

	D7	D6	D5	D4	D3	D2	D1	D0
	EP7 Rx	**EP6处方**	**EP5 Rx**	**EP4 Rx**	**EP3 Rx**	**EP2 Rx**	**EP1 Rx**	–
来自CPU	r	r	r	r	r	r	r	r
从USB	设置	设置	设置	设置	设置	设置	设置	r

一点	名称	作用
D15	**EP15 Rx**	Rx端点15中断。
D14	**EP14 Rx**	Rx端点14中断。
D13	**EP13 Rx**	Rx端点13中断。
D12	**EP12 Rx**	Rx端点12中断。
D11	**EP11 Rx**	Rx端点11中断。
D10	**EP10处方**	Rx端点10中断。
D9	**EP9 Rx**	Rx端点9中断。
D8	**EP8 Rx**	Rx Endpoint 8中断。
D7	**EP7 Rx**	Rx Endpoint 7中断。
D6	**EP6处方**	Rx Endpoint 6中断。
D5	**EP5 Rx**	Rx Endpoint 5中断。
D4	**EP4 Rx**	Rx端点4中断。
D3	**EP3 Rx**	Rx端点3中断。
D2	**EP2 Rx**	Rx端点2中断。
D1	**EP1 Rx**	Rx端点1中断。
D0	–	*未使用，总是返回0*

3.2.5. INTRTXE

IntrTxE是一个16位寄存器，为IntrTx中的中断提供中断启用位。如果一位被设置为1，则在IntrTx寄存器中相应的中断被设置时，MC_NINT将被断言。如果一位被设置为0，IntrTx中的中断仍被设置，但MC_NINT未被断言。在复位时，与端点0和设计中包括的TX端点相对应的比特被设置为1，而其余比特被设置成0。*注意：*与尚未配置的端点相关的位将始终返回0。

地址：06h；*重置值*：16'FFFF屏蔽，实现TX端点

	D15	D14	D13	D12	D11	D10	D9	D8
	EP15 Tx	**EP14 Tx**	**EP13 Tx**	**EP12 Tx**	**EP11 Tx**	**EP10 Tx**	**EP9 Tx**	**EP8 Tx**
来自CPU	rw	rw	rw	rw	rw	rw	rw	rw
从USB	r	r	r	r	r	r	r	r

	D7	D6	D5	D4	D3	D2	D1	D0
	EP7 Tx	**EP6 Tx**	**EP5 Tx**	**EP4 Tx**	**EP3 Tx**	**EP2 Tx**	**EP1 Tx**	**EP0**
来自CPU	rw	rw	rw	rw	rw	rw	rw	rw
从USB	r	r	r	r	r	r	r	r

3.2. 6. INTRXE

IntrRxE是一个16位寄存器，为IntrRx中的中断提供中断启用位。如果一位被设置为1，则在IntrRx寄存器中相应的中断被设置时，MC_NINT将被断言。如果一位被设置为0，IntrRx中的中断仍被设置，但MC_NINT未被断言。在复位时，与包括在设计中的Rx端点相对应的比特被设置为1，而其余比特被设置成0。*注意：与尚未配置的端点相关的位将始终返回0。*

地址：08h；重置值：16'FFFE屏蔽，实现Rx端点

	D15	D14	D13	D12	D11	D10	D9	D8
	EP15 Rx	**EP14 Rx**	**EP13 Rx**	**EP12 Rx**	**EP11 Rx**	**EP10处方**	**EP9 Rx**	**EP8 Rx**
来自CPU	rw	rw	rw	rw	rw	rw	rw	rw
从USB	r	r	r	r	r	r	r	r

	D7	D6	D5	D4	D3	D2	D1	D0
	EP7 Rx	**EP6处方**	**EP5 Rx**	**EP4 Rx**	**EP3 Rx**	**EP2 Rx**	**EP1 Rx**	–
来自CPU	rw	rw	rw	rw	rw	rw	rw	r
从USB	r	r	r	r	r	r	r	r

3.2. 7. INTRUSB

IntrUSB是一个8位只读寄存器，用于指示当前活动的USB中断。读取此寄存器时，所有活动中断都将被清除。

地址：0Ah；复位值：8'000

	D7	D6	D5	D4	D3	D2	D1	D0
	VBus错误	**Sess Req**	**Discon**	**Conn**	**软件**	**重置/巴贝尔**	**简历**	**悬**
来自CPU	r	r	r	r	r	r	r	r
从USB	设置	设置	设置	设置	设置	设置	设置	设置

一点	名称	作用
D7	**VBus错误**	在会话期间VBus降至VBus有效阈值以下时设置。*仅当MUSBHDRC是"A"设备时有效。*
D6	**Sess Req**	检测到会话请求信号时设置。*仅当MUSBHDRC是"A"设备时有效。*
D5	**Discon**	当检测到设备断开连接时，设置为主机模式。会话结束时设置为"外围"模式。*在所有交易速度下都有效。*
D4	**Conn**	检测到设备连接时设置。*仅在主机模式下有效。在所有交易速度下都有效。*
D3	**软件**	设置新帧开始的时间。
D2	**重置**	当在总线上检测到重置信号时，设置为外围模式。
	含糊不清地说	检测到牙牙学语时设置为主机模式。*注意：只有在发送了第一个SOF之后才处于活动状态。*
D1	**简历**	当MUSBHDRC处于挂起模式时，在总线上检测到恢复信号时设置。
D0	**悬**	在总线上检测到挂起信号时设置。*仅在外围模式下有效。*

3.2. 8. INTRUSBE

IntrUSBE是一个8位寄存器，为IntrUSB中的每个中断提供中断启用位。

地址：0Bh；复位值：8'06

	D7	D6	D5	D4	D3	D2	D1	D0
	VBus错误	**Sess Req**	**Discon**	**Conn**	**软件**	**重置/巴贝尔**	**简历**	**悬**
来自CPU	rw	rw	rw	rw	rw	rw	rw	rw
从USB	r	r	r	r	r	r	r	r

3.2. 9. FRAME

Frame是一个16位只读寄存器，用于保存最后接收到的帧编号。

地址：0Ch；重置值：16'0000

	D15 .. D11	D10	...	D0
	0 0 0 0 0	(MSB)	**帧编号**	(LSB)
来自CPU	r r		r	...r
从USB	w w	w	...	w

3.2. 10. INDEX

每个TX端点和每个Rx端点都有自己的一组控制/状态寄存器，位于100h-1FFh之间。此外，一组TX控制/状态和一组Rx控制/状态寄存器出现在10h–19h。索引是一个4位寄存器，用于确定访问哪些端点控制/状态寄存器。

地址：0Eh；重置值：4'b0000

	D3	D2	D1	D0
	(MSB)	**所选终结点**		(LSB)
来自CPU	rw	rw	rw	rw
从USB	r	r	r	r

在10h–19h访问端点的控制/状态寄存器之前，应将端点编号写入索引寄存器，以确保内存映射中出现正确的控制/状况寄存器。

3.2. 11. TEST MODE

测试模式是一个8位寄存器，主要用于将MUSBHDRC置于USB 2.0规范中描述的四种高速操作测试模式之一，以响应SET FEATURE:Testmode命令。它不用于正常操作。

地址：0Fh；复位值：8'000

	D7	D6	D5	D4	D3	D2	D1	D0
	强制主机	**FIFO_访问** *(自行清算)*	**Force_FS**	**Force_HS**	**测试包**	**测试_K**	**测试_J**	**测试_ SE0_NAK**
来自CPU	rw	设置	rw	rw	rw	rw	rw	rw
从USB	r	r	r	r	r	r	r	r

一点	名称	描述

<table>
<tr><th>一点</th><th>名称</th><th>描述</th></tr>
<tr><td>D7</td><td>强制主机</td><td>CPU设置该位，以指示核心在设置会话位时进入主机模式，无论其是否连接到任何外围设备。CID输入、HostDisconnect和LineState信号的状态被忽略。然后，核心将保持在主机模式，直到会话位被清除，即使设备已断开连接，并且如果Force_Host位保持设置，则在下次设置会话位时将重新进入主机模式。
在这种模式下，可以从DevCtl寄存器的位7读取来自PHY的HOSTDISCON信号的状态。
操作速度由Force_HS和Force_FS比特确定如下：
<table>
<tr><th>Force_HS</th><th>Force_FS</th><th>运行速度</th></tr>
<tr><td>0</td><td>0</td><td>低速</td></tr>
<tr><td>0</td><td>1.</td><td>全速</td></tr>
<tr><td>1.</td><td>0</td><td>高速</td></tr>
<tr><td>1.</td><td>1.</td><td>未定义</td></tr>
</table></td></tr>
<tr><td>D6</td><td>FIFO_访问</td><td>CPU设置此位以将端点0 TX FIFO中的数据包传输到端点0 Rx FIFO。它会自动清除。</td></tr>
<tr><td>D5</td><td>Force_FS</td><td>CPU将此位与上面的第7位一起设置，或者在接收到USB重置时强制MUSBHDRC进入全速模式。</td></tr>
<tr><td>D4</td><td>Force_HS</td><td>CPU将该位与上述第7位一起设置，或者在接收到USB重置时强制MUSBHDRC进入高速模式。</td></tr>
<tr><td>D3</td><td>测试包</td><td>（高速模式）CPU将此位设置为进入Test_Packet测试模式。在这种模式下，MUSBHDRC在总线上重复传输53字节的测试包，其形式在通用串行总线规范修订版2.0第7.1.20节（和第28.4节）中定义。注意：测试包具有固定格式，必须加载到
进入测试模式之前的终结点0 FIFO。</td></tr>
<tr><td>D2</td><td>测试_K</td><td>（高速模式）CPU将此位设置为进入Test_K测试模式。在这种模式下，MUSBHDRC在总线上传输连续的K。</td></tr>
<tr><td>D1</td><td>测试_J</td><td>（高速模式）CPU将此位设置为进入Test_J测试模式。在这种模式下，MUSBHDRC在总线上传输一个连续的J。</td></tr>
<tr><td>D0</td><td>测试_SE0_NAK</td><td>（高速模式）CPU将此位设置为进入Test_SE0_NAK测试模式。在该模式下，MUSBHDRC保持在高速模式，但用NAK响应任何有效的In令牌。</td></tr>
</table>

注：任何时候都只能设置D0–D6位中的一个。

3.2. 12 . D E VCT L

DevCtl是一个8位寄存器，用于选择MUSBHDRC是在外围模式还是在主机模式下运行，并用于控制和监控USB VBus线。如果PHY被挂起，则不接收PHY时钟（XCLK）并且不对VBus进行采样。

地址：60小时；*复位值*：8'h80

	D7	D6	D5	D4	D3	D2	D1	D0
	B-设备	**FSDev**	**LSDev**	**VBus[1]**	**VBus[0]**	**主机模式**	**主机需求**	**一场**
来自CPU	r	r	r	r	r	r	rw	rw
从USB	rw	rw	rw	rw	rw	rw	r/清除	rw

一点	名称	作用
D7	**B-设备**	此只读位指示MUSBHDRC是作为“A”设备还是作为“B”设备运行。0 ‘A’装置；1. ‘B’设备。*仅在会话正在进行时有效。* *注：*如果核心处于Force_Host模式（即会话已使用Testmode.D7=1启动），此位将指示PHY的HOSTDISCON输入信号的状态。
D6	**FSDev**	当检测到全速或高速设备连接到端口时，会设置此只读位。（通过在设备复位时检查高速啁啾，可以将高速设备与全速设备区分开来。）*仅在主机模式下有效。*
D5	**LSDev**	当检测到低速设备连接到端口时，会设置此只读位。 *仅在主机模式下有效。*
D4–D3	**VBus[1:0]**	这些只读位对当前VBus级别进行编码，如下所示： **D4** / **D3** / **意思** 0 / 0 / 低于会话结束 0 / 1. / 高于SessionEnd，低于AValid 1. / 0 / 高于AValid，低于VBusValid 1. / 1. / 高于VBusValid
D2	**主机模式**	此只读位是在MUSBHDRC充当主机时设置的。
D1	**主机需求**	设置后，当进入挂起模式时，MUSBHDRC将启动主机协商。当主机协商完成时，它被清除。参见第15节。*（仅限’B’设备）*
D0	**一场**	*当作为“A”设备操作时*，此位由CPU设置或清除，以启动或结束会话。 *当作为“B”设备操作时*，当会话开始/结束时，此位由MUSBHDRC设置/清除。它也由CPU设置为启动会话请求协议。当MUSBHDRC处于挂起模式时，CPU可以清除该位以执行软件断开连接。*注意：*当堆芯未挂起时清除此位将导致未定义的行为。

3.2. 13 . M I SC

MISC寄存器是一个包含各种常见配置位的8位寄存器。这些比特包括Rx/TX早期DMA使能比特。根据下表，配置位占用寄存器。

*地址：*61h；*复位值：*8’000

	D7	D6	D5	D4	D3	D2	D1	D0
	未使用	***未使用***	***未使用***	***未使用***	***未使用***	***未使用***	**tx_edma**	**rxedma**
来自CPU	R	r	r	r	r	r	rw	rw
从USB	r	r	r	r	r	r	rw	rw

一点	名称	作用
D7–D2	***未使用***	这些位是保留的
D1	**tx_edma**	当MAXP字节已写入端点时，所有IN端点的1’b0:DMA_REQ信号将被取消断言。这是后期模式。 当MAXP-8字节已写入端点时，所有IN端点的1’b1:DMA_REQ信号将被取消断言。这是早期模式。

D0	**rxedma**	当MAXP字节已被读取到端点时，所有OUT端点的1'b0:DMA_REQ信号将被取消断言。这是后期模式。 当MAXP-8字节已被读取到端点时，所有OUT端点的1'b1:DMA_REQ信号将被取消断言。这是早期模式。

3.3. INDEXED REGISTERS

*注意：*未配置所选端点时，以下寄存器的操作是未定义的。

3.3.1. CSR0L

CSR0L是一个8位寄存器，为端点0提供控制和状态位。*注：*寄存器的解释取决于MUSBHDRC是作为外设还是作为主机。用户还应注意，读取寄存器时返回的值反映了所获得的状态，例如写入寄存器的结果。

外围模式下的CSR0L： *地址：*12h（索引寄存器设置为0）；*复位值：*8'000

	D7	D6	D5	D4	D3	D2	D1	D0
	ServicedSetupEnd *（自清除）*	**服务RxKTRdy** *（自清）*	**SendStall** *（自行清算）*	**设置结束**	**DataEnd** *（自行清算）*	**SentStall**	**TxPktRdy** *（自行清算）*	**RxKTRdy**
来自CPU	设置	设置	设置	r	设置	r/清除	r/台	r
从USB	r	r	r	设置	r	设置	r	设置

一点	名称	外围模式下的功能
D7	**ServicedSetupEnd**	CPU向该位写入1以清除SetupEnd位。它会自动清除。
D6	**服务的xKTRdy**	CPU向该位写入1以清除RxPktRdy位。它会自动清除。
D5	**SendStall**	CPU向该位写入1以终止当前事务。STALL握手将被传输，然后该位将被自动清除。
D4	**设置结束**	当控制事务在设置DataEnd位之前结束时，将设置此位。此时将产生中断并刷新FIFO。该位由CPU向ServicedSetupEnd位写入1来清除。
D3	**DataEnd**	CPU设置此位： 1. 当为最后一个数据包设置TxPktRdy时。 2. 在卸载最后一个数据包后清除RxPktRdy时。 3. 当为零长度数据分组设置TxPktRdy时。它会自动清除。
D2	**SentStall**	此位是在传输STALL握手时设置的。CPU应该清除此位。
D1	**TxPktRdy**	CPU在将数据包加载到FIFO之后设置该位。当数据包已传输时，它会自动清除。此时也会生成一个中断（如果启用）。
D0	**RxKTRdy**	该比特是在接收到数据包时设置的。设置此位时会产生中断。CPU通过设置ServicedRxPktRdy位来清除该位。

主机模式下的CSR0L：　　*地址*：12h（索引寄存器设置为0）；*复位值*：8'000

	D7	D6	D5	D4	D3	D2	D1	D0
	NAK超时	**状态Pitt**	**ReqPkt**	**错误**	**设置Pkt**	**RxStall**	**TxPktRdy**	**RxKTRdy**
来自CPU	r/清除	rw	rw	r/清除	r/清除	r/清除	r/台	r/清除
从USB	设置	r	rw	设置	r	设置	清楚的	rw

一点	名称NAK	主机模式下的功能
D7	**超时**	当端点0在接收到NAK响应之后停止的时间长于NAKLimit0寄存器设置的时间时，将设置此位。CPU应清除此位，以允许端点继续。
D6	**状态Pitt**	CPU在设置TxPktRdy或ReqPkt位的同时设置该位，以执行状态阶段事务。设置此位可确保数据切换设置为1，从而DATA1数据包用于状态阶段事务。
D5	**ReqPkt**	CPU将此位设置为请求IN事务。当设置RxPktRdy时，它将被清除。
D4	**错误**	当已经进行了三次尝试来执行没有来自外围设备的响应的事务时，将设置该位。CPU应该清除此位。设置此位时会产生中断。
D3	**设置Pkt**	在设置TxPktRdy位的同时，CPU设置该位以发送用于事务的SETUP令牌而不是OUT令牌。*注意*：设置此位也会清除数据切换。
D2	**RxStall**	此位是在接收到STALL握手时设置的。CPU应该清除此位。
D1	**TxPktRdy**	CPU在将数据包加载到FIFO之后设置该位。当数据包已传输时，它会自动清除。此时也会生成一个中断（如果启用）。
D0	**RxKTRdy**	该比特是在接收到数据包时设置的。当设置此位时，将生成一个中断（如果启用）。当数据包已从FIFO中读取时，CPU应清除此位。

3.3. 2. C S R0H

CSR0H是一个8位寄存器，为端点0提供控制和状态位。*注*：寄存器的解释取决于MUSBHDRC是作为外设还是作为主机。用户还应注意，读取寄存器时返回的值反映了所获得的状态，例如写入寄存器的结果。

外围模式下的CSR0H： *地址*：13h（索引寄存器设置为0）；*复位值*：*8'000*

	D7	D6	D5	D4	D3	D2	D1	D0
	–	–	–	–	–	–	–	**FlushFIFO** *（自行清算）*
来自CPU	r	r	r	r	r	r	r	设置
从USB	r	r	r	r	r	r	r	r

一点	名称	外围模式下的功能
D7–D1	–	*未使用。读取时返回0。*
D0	**FlushFIFO**	CPU向该位写入1，以清除要从端点0 FIFO发送/读取的下一个数据包。FIFO指针被重置，TxPktRdy/RxPktRdy位（如下）被清除。*注意*：只有在设置了TxPktRdy/RxPktRdy时，才应使用FlushFIFO。在其他有时，它可能会导致数据损坏。

主机模式下的CSR0H：

地址：13h（索引寄存器设置为0）；*复位值*：*8'000* D7D6D5D4D3D2D1D0

	–	–	–	–	**Dis Ping**	**数据切换Wr。启用** *（自清除）*	**数据开关**	**FlushFIFO** *（自行清算）*
来自CPU	r	r	r	r	rw	设置	rw	设置
从USB	r	r	r	r	r	r	rw	r

一点	名称	主机模式下的功能
D7–D4	–	*未使用。读取时返回0。*
D3	**Dis Ping**	CPU向该位写入1，以指示核心不要在高速控制传输的数据和状态阶段发出PING令牌（用于不响应PING的设备）。
D2	**数据切换写入启用**	CPU向该位写入1，以启用要写入的端点0数据切换的当前状态（请参阅下面的数据切换位）。一旦写入新值，该位将自动清除。
D1	**数据开关**	读取时，此位指示端点0数据切换的当前状态。如果D10为高，则可以利用所需的数据切换设置来写入该位。如果D10为低，则忽略写入该位的任何值。
D0	**FlushFIFO**	CPU向该位写入1，以清除要从端点0 FIFO发送/读取的下一个数据包。FIFO指针被重置，TxPktRdy/RxPktRdy位（如下）被清除。 *注意*：只有在设置了TxPktRdy/RxPktRdy时，才应使用FlushFIFO。在其他情况下，它可能会导致数据损坏。

3.3. 3. 计数0

Count0是一个7位只读寄存器，用于指示端点0 FIFO中接收到的数据字节数。返回的值随着FIFO内容的变化而变化，并且仅在设置RxPktRdy（CSR0.D0）时有效。

地址：18h（索引寄存器设置为0）；*重置值*：7'0000000

	D6	D5	D4	D3	D2	D1	D0
	（MSB）			**终结点0接收计数**			（LSB）
来自CPU	r	r	r	r	r	r	r
从USB	w	w	w	w	w	w	w

3.3. 4. T Y PE 0

注意：仅限主机模式！

这些位仅在配置GUI中启用多点选项时适用。当多点被启用时，Type0是一个8位寄存器，其仅实现位6和7。这些位应该以目标设备的操作速度写入。

地址：1Ah；*重置值*：2'b00

D7	D6
速度	
来自	*CPUwrw*
来自	*USBrr*

一点	名称	作用
D7–D6	**速度**	目标设备的运行速度： 00：*未使用（注意：如果选择，则假设目标使用与核心相同的连接速度。）* 01:高 10:满 11:低

3.3. 5. CONFIGDATA

ConfigData是一个8位只读寄存器，用于返回有关所选核心配置的信息。

地址：1Fh（索引寄存器设置为0）；*重置值*：取决于配置

	D7	D6	D5	D4	D3	D2	D1	D0
	MPRxE	**MPTxE**	**BigEndian**	**HBRxE**	**HBTxE**	**DynFIFO尺寸**	**SoftConE**	**UTMI数据宽度**
来自CPU	r	r	r	r	r	r	r	r
从USB	r	r	r	r	r	r	r	r

一点	名称	作用
D7	**MPRxE**	当设置为"1"时，选择批量数据包的自动合并（见第22节）
D6	**MPTxE**	当设置为"1"时，选择批量数据包的自动拆分（请参阅第22节）
D5	**BigEndian**	始终为"0"。表示小恩迪亚排序。
D4	**HBRxE**	当设置为"1"时，表示已选择高带宽Rx ISO端点支持。
D3	**HBTxE**	当设置为"1"时，表示已选择高带宽TX ISO Endpoint Support。
D2	**DynFIFO尺寸**	当设置为"1"时，表示选择了动态FIFO大小选项。
D1	**SoftConE**	总是"1"。表示软连接/断开连接。
D0	**UTMI数据宽度**	表示所选的UTMI+数据宽度。总是0表示8位。

3.3. 6. NAKLIMIT0

注意：仅限主机模式！

NAKLimit0是一个5位寄存器，用于设置帧/微帧（高速传输）的数量，在该数量之后端点0在接收NAK响应流时应超时。（其他端点的等效设置可以通过其TxInterval和RxInterval寄存器进行：参见第3.3.15节和第3.3.17节。）

所选帧/微帧的数量为$2^{(m-1)}$（其中*m*是寄存器中设置的值，有效值为2-16）。如果主机接收到来自目标的NAK响应的帧数超过此寄存器中设置的限制所表示的帧数，则端点将被停止。*注意*：值为0或1将禁用NAK超时功能。

地址：1Bh（索引寄存器设置为0）；*重置值*：5'00000

	D4	...	D0
	(MSB)	**端点0 NAK限制（*m*）**	(LSB)
来自CPU	rw	...	rw
从USB	r	...	r

3.3. 7. TXMAXP

TxMaxP寄存器定义了在单个操作中可以通过所选TX端点传输的最大数据量。每个TX端点（端点0除外）都有一个TxMaxP寄存器。

地址：10h；*重置值*：11/13/16'0000

	D12/15		D11	D10	...	D0
	***m*–1**			（MSB）	**最大有效负载/事务**	（LSB）
来自CPU	rw	...	rw	rw	...	rw
从USB	r	...	r	r	...	r

位10:0定义（以字节为单位）在单个事务中传输的最大有效载荷。该值集最多可为1024字节，但受USB规范对全速和高速操作中批量、中断和等时传输的数据包大小的限制。

当配置核心时，如果选择了高带宽等时/中断端点或批量端点上的数据包拆分，则寄存器包括2或5个额外的位，这些位定义了乘数m，该乘数*m*等于记录值的一个以上。

在启用数据包拆分选项的批量端点的情况下，乘数*m*可以高达32，并定义指定有效载荷的"USB"数据包（即通过USB传输的数据包）的最大数量，在传输之前，放置在FIFO中的单个数据包应拆分为该数据包。（如果未启用数据包拆分选项，则不执行D15–D13，忽略D12–D11（如果包含）。）**注意：**数据包需要是比特10:0指定的有效载荷的精确倍数，比特10:0本身需要是8、16、32、64或（在高速传输的情况下）512字节。

对于在高速模式下运行并启用高带宽选项的等时/中断端点，*m*可以仅为2或3（分别对应于位11集或位12集），并且它指定了在单个微帧中可以发生的此类事务的最大数量。如果位11或位12为非零，则MUSBHDRC将自动将写入FIFO的任何数据包拆分为最多2或3个"USB"包，每个包包含指定的有效载荷（或更少）。每个事务的最大有效载荷是1024字节，因此这允许在每个微帧中传输多达3072字节。（对于全速模式下的等时传输，或者如果未启用高带宽，则忽略位11和12。）

写入位10:0的值（在高带宽等时传输的情况下乘以*m*）必须与相关端点的标准端点描述符的*wMaxPacketSize字段*中给定的值相匹配（见*USB规范*2.0版，第9章）。不匹配可能导致意外结果。

写入此寄存器的值所表示的数据总量（指定的有效负载◈ *m*） 不得超过TX端点的FIFO大小，如果需要双缓冲，则不得超过FIFO大小的一半。

如果在从端点发送数据包后更改了此寄存器，则在将新值写入此寄存器后，应完全刷新TX端点FIFO（使用TxCSRL中的FlushFIFO位）。

注意：为了在DMA模式1中正确生成中断，TxMaxP必须设置为偶数字节。

3.3. 8. TXCSRL

TxCSRL是一个8位寄存器，为通过当前选择的TX端点的传输提供控制和状态位。每个配置的TX端点（不包括端点0）都有一个TxCSRL寄存器。*注：*寄存器的解释取决于MUSBHDRC是作为外设还是作为主机。用户还应注意，读取寄存器时返回的值反映了所获得的状态，例如写入寄存器的结果。

在外设模式下：

地址：12h；复位值：8'000

	D7	D6	D5	D4	D3	D2	D1	D0
	IncompTx	**ClrDataTog**	**SentStall**	**SendStall**	**FlushFIFO** *（自行清算）*	**欠载**	**FIFO NotEmpty**	**TxPktRdy**
来自CPU	r/清除	设置	r/清除	rw	设置	r/清除	r/清除	r/台
从USB	设置	r/清除	设置	r	r	设置	设置	清楚的

一点	名称	外围模式下的功能
D7	**IncompTx**	当端点用于高带宽Isochronous时，此位被设置为指示大数据包已被拆分为2个或3个数据包进行传输但不足的位置 已接收到发送所有部件的IN令牌。*注意：在除等时传输以外的任何传输中，此位都将始终返回0。*
D6	**ClrDataTog**	CPU向该位写入1，以将端点数据切换重置为0。
D5	**SentStall**	此位是在传输STALL握手时设置的。FIFO被清除，TxPktRdy位被清除（见下文）。CPU应该清除此位。
D4	**SendStall**	CPU向该位写入1，以向IN令牌发出STALL握手。CPU清除该位以终止暂停条件。*注：当端点用于等时传输时，此位不起作用。*
D3	**FlushFIFO**	CPU向该位写入1，以清除来自端点TX FIFO的最新数据包。FIFO指针被重置，TxPktRdy位（下面）被清除，并产生中断。可以与TxPktRdy同时设置，以中止当前加载到FIFO中的数据包。*注意：只有*在设置了TxPktRdy时，才应使用FlushFIFO。在其他情况下，它可能会导致数据损坏。还要注意，如果FIFO是双缓冲的，则可能需要设置两次FlushFIFO才能完全清除FIFO。
D2	**欠载**	如果在未设置TxPktRdy时接收到IN令牌，则USB设置该比特。CPU应该清除此位。
D1	**FIFONotEmpty**	当TX FIFO中至少有1个数据包时，USB会设置此位。
D0	**TxPktRdy**	CPU在将数据包加载到FIFO之后设置该位。当数据包已传输时，它会自动清除。此时也会生成一个中断（如果启用）。在将第二数据包加载到双缓冲FIFO之前，TxPktRdy也被自动清除。

在主机模式下：

地址：12h；复位值：8'000

D7	D6	D5	D4	D3	D2	D1	D0
NAK超时/输入Tx	**ClrDataTog**	**RxStall**	**设置Pkt**	**FlushFIFO** *（自行清算）*	**错误**	**FIFO NotEmpty**	**TxPktRdy**
来自CPU从USB r/清除设置	设置 r/清除	r/清除 设置	rw r	设置 r	r/清除 rw	r/清除 设置	r/台 清楚的

一点	名称	主机模式下的功能
D7	**NAK超时** **IncompTx**	*仅限批量端点*：当收到NAK响应后，TX端点停止的时间超过TxInterval寄存器设置为NAK Limit的时间时，将设置此位。CPU应清除此位，以允许端点继续。 *仅限高带宽中断端点*：如果没有从要发送数据包的设备接收到响应，则会设置此位。
D6	**ClrDataTog**	CPU向该位写入1，以将端点数据切换重置为0。
D5	**RxStall**	此位是在接收到STALL握手时设置的。当设置该位时，任何正在进行的DMA请求都将停止，FIFO将被完全刷新，TxPktRdy位将被清除（见下文）。CPU应该清除此位。
D4	**设置Pkt**	在设置TxPktRdy位的同时，CPU设置该位以发送用于事务的SETUP令牌而不是OUT令牌。*注意*：设置此位也会清除数据切换。
D3	**FlushFIFO**	CPU向该位写入1，以清除来自端点TX FIFO的最新数据包。FIFO指针被重置，TxPktRdy位（下面）被清除，并产生中断。可以与TxPktRdy同时设置，以中止当前加载到FIFO中的数据包。*注意*：只有在设置了TxPktRdy时，才应使用FlushFIFO。在其他情况下，它可能会导致数据损坏。还要注意，如果FIFO是双缓冲的，则可能需要设置两次FlushFIFO才能完全清除FIFO。
D2	**错误**	当已经进行了3次发送分组的尝试并且没有接收到握手分组时，USB设置该比特。当位被设置时，产生中断，TxPktRdy被清除，FIFO被完全刷新。CPU应清除此位。*仅当端点在批量或中断模式下操作时有效。*
D1	**FIFONotEmpty**	当TX FIFO中至少有1个数据包时，USB会设置此位。
D0	**TxPktRdy**	CPU在将数据包加载到FIFO之后设置该位。当数据包已传输时，它会自动清除。此时也会生成一个中断（如果启用）。在将第二数据包加载到双缓冲FIFO之前，TxPktRdy也被自动清除。

3.3 . 9 . TX-CS-RH

TxCSRH是一个8位寄存器，它为通过当前选择的TX端点的传输提供额外的控制。每个配置的TX端点（不包括端点0）都有一个TxCSRH寄存器。*注：*寄存器的解释取决于MUSBHDRC是作为外设还是作为主机。用户还应注意，读取寄存器时返回的值反映了所获得的状态，例如写入寄存器的结果。

在外围模式下：

地址：13h；复位值：8'000

d7d6d4d3d2d1d0

	自动调压	国际标准化组织	模式	DMA ReqEnab	FrcDataTog	DMAReqMode	–	–
来自CPU	rw	rw	rw	rw	rw	rw	r	r
从USB	r	r	r	r	r	r	r	r

一点	名称	外围模式下的功能
D7	**自动调压**	如果CPU设置此位，则当最大数据包大小（TxMaxP中的值）的数据加载到TX FIFO时，TxPktRdy将自动设置。如果小于最大数据包大小的数据包是 则必须手动设置TxPktRdy。*注意：不应为高带宽等时端点或高带宽中断端点设置。*
D6	**国际标准化组织**	CPU设置此位以启用等时传输的TX端点，并清除它以启用批量或中断传输的TX终结点。*注意：此位仅在外围模式下有任何效果。在主机模式下，它总是返回零。*
D5	**模式**	CPU将该位设置为启用端点方向为TX，并清除该位以启用其为Rx。*注意：只有在TX和Rx事务都使用相同的端点FIFO时，此位才有任何效果。*
D4	**DMA ReqEnab**	CPU设置此位以启用对TX端点的DMA请求。
D3	**FrcDataTog**	CPU设置此位以强制端点数据切换到开关，并从FIFO中清除数据包，无论是否接收到ACK。这可由中断TX端点使用，中断TX端点用于通信等时端点的速率反馈。
D2	**DMAReqMode**	CPU将该位设置为选择DMA请求模式1，并将其清除为选择DMA申请模式0。 *注：在清除上述DMAReqEnab位之前或同一周期内，不得清除该位。*
D1–D0	–	*未使用，始终返回0。*

在主机模式下：

*地址：*13h；*复位值：8'000*

d7d6d4d3d2d1d0

	自动调压	–	**模式**	**DMA ReqEnab**	**FrcDataTog**	**DMAReqMode**	**数据切换 Wr。启用** *（自清除）*	**数据开关**
来自CPU	rw	r	rw	rw	rw	rw	设置	rw
从USB	r	r	r	r	r	r	r	rw

一点	名称	主机模式下的功能
D7	**自动调压**	如果CPU设置此位，则当最大数据包大小（TxMaxP中的值）的数据加载到TX FIFO时，TxPktRdy将自动设置。如果加载了小于最大数据包大小的数 据 包，则必须手动设置TxPktRdy。*注意：不应为高带宽等时端点或高带宽中断端点设置。*
D6	–	*未使用，总是返回零。*
D5	**模式**	CPU将该位设置为使能端点方向为TX，并将其清除为使能终点方向为Rx。*注意：只有在TX和Rx事务都使用相同的端点FIFO时，此位才有任何效果。*
D4	**DMA ReqEnab**	CPU设置此位以启用对TX端点的DMA请求。
D3	**FrcDataTog**	CPU设置此位以强制端点数据切换到开关，并从FIFO中清除数据包，无论是否接收到ACK。这可由中断TX端点使用，中断TX端点用于通信等时端点的速率反馈。
D2	**DMAReqMode**	CPU将该位设置为选择DMA请求模式1，并将其清除为选择DMA申请模式0。 *注：在清除上述DMAReqEnab位之前或同一周期内，不得清除该位。*
D1	**数据切换写入启用**	CPU向该位写入1，以启用要写入的TX端点数据切换的当前状态（请参阅下面的数据切换位）。一旦写入新值，该位将自动清除。
D0	**数据开关**	读取时，此位指示TX端点数据切换的当前状态。如果D1为高电平，则可以使用所需的数据切换设置来写入该位。如果D1为低，则忽略写入该位的任何值。

3.3. 10 . R X M A XP

RxMaxP寄存器定义了在单个操作中可以通过所选Rx端点传输的最大数据量。每个Rx端点（端点0除外）都有一个RxMaxP寄存器。

地址：14h；*重置值*：11/13/16'0000

	D12/15		D11	D10	...	D0
	***m*–1**			（MSB）	**最大有效负载/事务**	（LSB）
来自CPU	rw	...	rw	rw	...	rw
从USB	r	...	r	r	...	r

位10:0定义（以字节为单位）在单个事务中传输的最大有效载荷。该值集最多可为1024字节，但受USB规范对全速和高速操作中批量、中断和等时传输的数据包大小的限制。

当配置核心时，如果选择了高带宽等时/中断端点或批量端点上的数据包拆分，则寄存器包括2或5个额外的位，这些位定义了乘数m，该乘数*m*等于记录值的一个以上。

对于启用了分组拆分选项的批量端点，乘数*m*可以高达32，并且定义了要在FIFO内合并为单个数据分组的指定有效载荷的USB分组的数量。（如果未启用数据包拆分选项，则不执行D15–D13，忽略D12–D11（如果包含）。）

对于在高速模式下运行并启用高带宽选项的等时/中断端点，*m*只能是2或3（分别对应于位11集或位12集），并且它指定了在单个微帧中可以发生的此类事务的最大数量。如果位11或位12为非零，则MUSBHDRC将自动将在任何微帧中接收的单独USB分组组合成RX FIFO内的单个分组。每个事务的最大有效载荷是1024字节，因此这允许在每个微帧中接收多达3072字节。（对于全速模式下的等时传输，或者如果未启用高带宽，则忽略位11和12。）

写入位10:0的值（在高带宽等时传输的情况下乘以*m*）必须与相关端点的标准端点描述符的*wMaxPacketSize字段*中给定的值相匹配（见*USB规范*2.0版，第9章）。不匹配可能导致意外结果。

写入此寄存器的值所表示的数据总量（指定的有效负载□ *m*） 不得超过RX端点的FIFO大小，如果需要双缓冲，则不得超过FIFO大小的一半。

注意：RxMaxP必须设置为偶数字节，才能在DMA模式1中正确生成中断。

3.3. 11 . R X CS RL

RxCSRL是一个8位寄存器，为通过当前选择的Rx端点的传输提供控制和状态位。每个配置的Rx端点（不包括端点0）都有一个RxCSRL寄存器。*注：*寄存器的解释取决于MUSBHDRC是作为外设还是作为主机。用户还应注意，读取寄存器时返回的值反映了所获得的状态，例如写入寄存器的结果。

在外设模式下：

地址：16h；复位值：8'000

	D7	D6	D5	D4	D3	D2	D1	D0
	ClrDataTog	**SentStall**	**SendStall**	**FlushFIFO** *（自行清算）*	**数据错误**	**横行**	**FIFOFull** *（自行清算）*	**RxKTRdy**
来自CPU	设置	r/清除	rw	设置	r	r/清除	r	r/清除

一点	名称	外围模式下的功能
D7	**ClrDataTog**	CPU向该位写入1，以将端点数据切换重置为0。
D6	**SentStall**	此位是在传输STALL握手时设置的。CPU应该清除此位。
D5	**SendStall**	CPU向该位写入1以发出STALL握手。CPU清除该位以终止暂停条件。*注：当端点用于等时传输时，此位不起作用。*
D4	**FlushFIFO**	CPU将1写入该位，以清除将从端点Rx FIFO读取的下一个数据包。 FIFO指针被重置，RxPtRdy位（下面）被清除。*注意：*FlushFIFO只能在设置RxPktRdy时使用。在其他情况下，它可能会导致数据损坏。还要注意，如果FIFO是双缓冲的，则可能需要设置两次FlushFIFO才能完全清除FIFO。
D3	**数据错误**	如果数据包具有CRC或比特填充错误，则当设置RxBitRdy时设置该比特。当RxPktRdy被清除时，它被清除。*注意：此位仅在端点以ISO模式运行时有效。在批量模式下，它总是返回零。*
D2	**横行**	如果OUT数据包不能加载到Rx FIFO中，则设置此位。CPU应该清除此位。 *注意：此位仅在端点以ISO模式运行时有效。在批量模式下，它总是返回零。*
D1	**FIFOFull**	当不能将更多的数据包加载到Rx FIFO中时，设置该位。
D0	**RxKTRdy**	该比特是在接收到数据包时设置的。当数据包已从Rx FIFO卸载时，CPU应清除此位。位被设置时会产生一个中断。

在主机模式下：

地址：16h；复位值：8'000

	D7	D6	D5	D4	D3	D2	D1	D0
	ClrDataTog	**RxStall**	**ReqPkt**	**FlushFIFO**（*自行清算*）	**DataError/ NAK超时**	**错误**	**FIFOFull**（*自行清算*）	**RxKTRdy**
来自CPU	设置	r/清除	rw	设置	r（/清除）	r/清除	r	r/清除
从USB	r/清除	设置	rw	r	设置	设置	设置	设置

一点	名称	主机模式下的功能
D7	**ClrDataTog**	CPU向该位写入1，以将端点数据切换重置为0。
D6	**RxStall**	当接收到STALL握手时，设置该位并生成中断。CPU应该清除此位。
D5	**ReqPkt**	CPU向该位写入1以请求IN事务。当设置RxPktRdy时，它将被清除。
D4	**FlushFIFO**	CPU将1写入该位，以清除将从端点Rx FIFO读取的下一个数据包。 FIFO指针被重置，RxPtRdy位（下面）被清除。*注意*：FlushFIFO只能在设置RxPktRdy时使用。在其他情况下，它可能会导致数据损坏。还要注意，如果FIFO是双缓冲的，则可能需要设置两次FlushFIFO才能完全清除FIFO。
D3	**DataError/ NAK超时**	当在ISO模式下操作时，如果数据包具有CRC或位填充错误，则当设置RxPktRdy时设置该位，并且当清除RxPktRdy时清除该位。在批量模式下，当接收到NAK响应后Rx端点停止的时间超过RxInterval寄存器设置为NAK Limit的时间时，将设置此位。CPU应清除此位，以允许端点继续。
D2	**错误**	当已经进行了3次接收分组的尝试并且没有接收到数据分组时，USB设置该位。CPU应该清除此位。设置此位时会产生中断。 *注意：只有当Rx端点在批量或中断模式下操作时，此位才有效。在ISO模式下，它总是返回零。*
D1	**FIFOFull**	当不能将更多的数据包加载到Rx FIFO中时，设置该位。
D0	**RxKTRdy**	该比特是在接收到数据包时设置的。当数据包已从Rx FIFO卸载时，CPU应清除此位。位被设置时会产生一个中断。

3.3. 12 . R X CS RH

RxCSRH是一个8位寄存器，为通过当前选择的Rx端点的传输提供额外的控制和状态位。每个配置的Rx端点（不包括端点0）都有一个RxCSRH寄存器。*注：寄存器的解释取决于MUSBHDRC是作为外设还是作为主机。用户还应注意，读取寄存器时返回的值反映了所获得的状态，例如写入寄存器的结果。*

在外围模式下：

地址：17h；复位值：8'000

	D7	D6	D5	D4	D3	D2	D1	D0
	自动清除	**国际标准化组织**	**DMA ReqEnab**	**DisNyet /PID错误**	**DMAReqMode**	–	–	**IncompRx**
来自CPU	rw	rw	rw	rw/r	rw	r	r	r/清除

一点	名称	外围模式下的功能
D7	**自动清除**	如果CPU设置此位，则当RxMaxP字节的数据包已从Rx FIFO卸载时，RxPtRdy位将自动清除。当卸载小于最大数据包大小的数据包时，必须手动清除RxPktRdy。当使用DMA卸载RxFIFO时，无论RxMaxP如何，都会以4字节块的形式从RxFIFO读取数据。因此，RxPtRdy位将被清除如下： **余数（RxMaxP/4）** / **实际读取字节数** / **将清除RxPktRdy的数据包大小。** 0（即RXMaxP=64字节） / RXMAXP / RXMAXP、RXMAXP-1，RXMAXP-2，RXMAXP-3 3（即RXMaxP=63字节） / RXMAXP+1 / RXMAXP、RXMAXP-1、RXMAXP-2 2（即RXMaxP=62字节） / RXMAXP+2 / RXMAXP，RXMAXP-1 1（即RXMaxP=61字节） / RXMAXP+3 / RXMAXP *注意：不应为高带宽等时端点设置。*
D6	**国际标准化组织**	CPU将此位设置为启用等时传输的Rx端点，并将其清除为启用批量/中断传输的Rx端点。
D5	**DMA ReqEnab**	CPU设置此位以启用对Rx端点的DMA请求。
D4	**DisNyet** **PID错误**	**批量/中断事务：**CPU将此位设置为禁止发送NYET握手。设置后，所有成功接收到的Rx数据包都是ACK，包括在FIFO变满时。*注意：此位仅在高速模式下有任何作用，在高速模式中，应为所有中断端点设置此位。* **ISO事务：**核心设置此位以指示接收到的数据包中的PID错误。
D3	**DMAReqMode**	CPU将该位设置为选择DMA请求模式1，并将其清除为选择DMA申请模式0。
D2–D1	–	*未使用，始终返回0。*

D0	**IncompRx**	如果Rx FIFO中的数据包由于未接收到部分数据而不完整，则在高带宽等时/中断传输中设置此位。当RxPktRdy被清除时，它被清除。 *注意：在除等时传输以外的任何传输中，此位将始终返回0。*

在主机模式下：

地址：17h；复位值：8'000

	D7	D6	D5	D4	D3	D2	D1	D0
	自动清除	**AutoReq**	**DMA ReqEnab**	**PID错误**	**DMAReqMode**	**数据切换Wr。启用**（自清除）	**数据开关**	**IncompRx**
来自CPU	rw	rw	rw	r	rw	r	r	r/清除

一点	名称	主机模式下的功能
D7	**自动清除**	如果CPU设置此位，则当RxMaxP字节的数据包已从Rx FIFO卸载时，RxPtRdy位将自动清除。当卸载小于最大数据包大小的数据包时，必须手动清除RxPktRdy。当使用DMA卸载RxFIFO时，无论RxMaxP如何，都会以4字节块的形式从RxFIFO读取数据。因此，RxPtRdy位将被清除如下： **余数（RxMaxP/4）** | **实际读取字节数** | **将清除RxPktRdy的数据包大小。** 0（即RXMaxP=64字节） | RXMAXP | RXMAXP、RXMAXP-1、RXMAXP-2、rxaxp-3 3（即RXMaxP=63字节） | RXMAXP+1 | RXMAXP、RXMAXP-1，RXMAXP-2 2（即RXMaxP=62字节） | RXMAXP+2 | RXMAXP，RXMAXP-1 1（即RXMaxP=61字节） | RXMAXP+3 | RXMAXP *注意：不应为高带宽等时端点设置。*
D6	**AutoReq**	如果CPU设置该位，则当RxPktRdy位被清除时，ReqPkt位将被自动设置。 *注意：*当接收到短数据包时，此位会自动清除。
D5	**DMA ReqEnab**	CPU设置此位以启用对Rx端点的DMA请求。
D4	**PID错误**	**仅限ISO事务：**核心设置此位以指示接收到的数据包中的PID错误。 ***批量/中断事务：****此位的设置被忽略。*
D3	**DMAReqMode**	CPU将该位设置为选择DMA请求模式1，并将其清除为选择DMA申请模式0。
D2	**数据切换写入启用**	CPU向该位写入1，以启用要写入的端点0数据切换的当前状态（请参阅下面的数据切换位）。一旦写入新值，该位将自动清除。
D1	**数据开关**	读取时，此位指示端点0数据切换的当前状态。如果D10为高，则可以利用所需的数据切换设置来写入该位。如果D10为低，则忽略写入该位的任何值。
D0	**IncompRx**	如果接收到的数据包不完整，则此位将设置为高带宽同步传输或中断传输。当RxPtRdy被清除时，它将被清除。*注意：如果正确遵循USB协议，* *这个位永远不应该被设置。变为设置的位指示相关外围设备无法正常工作。（在除等时传输以外的任何传输中，此位将始终返回0。）*

3.3. 13 . R X计数

RxCount是一个13位只读寄存器，用于保存当前排队从Rx FIFO读取的数据包中的数据字节数。如果数据包是作为多个批量数据包传输的，则给定的数字将用于组合的数据包。
注意：返回的值随着FIFO的卸载而变化，并且仅在设置RxPktRdy（RxCSR.D0）时有效。

地址：18h；*重置值*：13'0000000000000

	D12	...	D0
	（MSB）	**端点Rx计数**	（LSB）
来自CPU	r	...	r
从USB	w	...	w

3.3. 14 . T X T Y PE

注意：仅限主机模式！

TxType是一个8位寄存器，应写入端点要针对的端点编号、当前所选TX端点使用的事务协议及其操作速度。每个配置的TX端点都有一个TxType寄存器（端点0除外，它在1Ah有自己的Type寄存器）。D6-D7仅在核心配置有多点选项时有效。

地址：1Ah；*复位值*：8'000

	D7	D6	D5	D4	D3	D2	D1	D0
	***Sp 预计起飞时间**		**Prot col**		**目标终结点编号**			
来自CPU	Rw	rw	rw	rw	rw	rw	rw	rw
从USB	R	r	r	r	r	r	r	r

注：D6-D7仅在启用多点选项时有效，否则不使用这些位。

一点	名称	作用
D7–D6	**速度**	当核心配置有多点选项时，目标设备的操作速度：00：*未使用（注意：如果选择，则假设目标使用与核心相同的连接速度。）*01：高 10:满 11:低 当核心没有配置多点选项时，不应访问这些位。
D5–D4	**协议**	CPU应将其设置为选择TX端点所需的协议：00:Control 01:等时 10：散装 11：中断
D3–D0	**目标终结点编号**	CPU应将该值设置为在设备枚举期间返回给MUSBHDRC的TX端点描述符中包含的端点编号。

3.3.15. TXINTERVAL

地址：1Bh；重置值：8'b00000000

	D7	...	D0
	Tx轮询间隔/NAK限制（*m*）		
来自CPU	rw	...	Rw
从USB	r	...	R

注意：仅限主机模式！

TxInterval是一个8位寄存器，用于中断和等时传输，定义当前所选TX端点的轮询间隔。对于批量端点，此寄存器设置帧/微帧的数量，在该数量之后，端点在接收NAK响应流时应超时。

每个配置的TX端点（端点0除外）都有一个TxInterval寄存器。在每种情况下，所设置的值定义帧/微帧（高速传输）的数量，如下所示：

传输类型	速度	有效值（*m*）	理解
打断	低速或全速	1 – 255	轮询间隔为*m*帧。
	高速	1 – 16	轮询间隔为$2^{(m-1)}$微帧。
等时的	全速或高速	1 – 16	轮询间隔为$2^{(m-1)}$帧/微帧。
大部分	全速或高速	2 – 16	NAK限制为$2^{(m-1)}$帧/微帧。*注意*：值为0或1将禁用NAK超时功能。

3.3.16. RXTYPE

注意：仅限主机模式！

RxType是一个8位寄存器，应写入端点要针对的端点编号、当前所选Rx端点使用的事务协议及其操作速度。每个配置的Rx端点都有一个RxType寄存器（端点0除外，它在1Ah处有自己的Type寄存器）。D6-D7仅在核心配置有多点选项时有效。

地址：1Ch；复位值：8'000

	D7	D6	D5	D4	D3	D2	D1	D0
	*Sp 预计起飞时间		Prot col		目标终结点编号			
来自CPU	rw	rw	rw	rw	rw	rw	rw	rw
从USB	r	r	r	r	r	r	r	r

一点	名称	作用
D7–D6	**速度**	当核心配置有多点选项时，目标设备的操作速度：00：*未使用（注意：如果选择，则假设目标使用与核心相同的连接速度。）*01：高 10:满 11:低 当核心没有配置多点选项时，不应访问这些位。
D5–D4	**协议**	CPU应将其设置为选择Rx端点所需的协议：00:Control 01:等时 10：散装 11：中断
D3–D0	**目标终结点编号**	CPU应将该值设置为在设备枚举期间返回给MUSBHDRC的Rx端点描述符中包含的端点编号。

注：D6-D7仅在启用多点选项时有效，否则不使用这些位。

3.3.17. RXINTERVAL

注意：仅限主机模式！

RxInterval是一个8位寄存器，用于中断和等时传输，定义当前所选Rx端点的轮询间隔。对于批量端点，此寄存器设置帧/微帧的数量，在该数量之后，端点在接收NAK响应流时应超时。每个配置的Rx端点（端点0除外）都有一个RxInterval寄存器。在每种情况下，所设置的值定义帧/微帧（高速传输）的数量，如下所示：

地址：1Dh；*重置值*：8'b00000000

	D7	...	D0
	Rx轮询间隔/NAK限制（*m*）		
来自CPU	rw	...	rw
从USB	r	...	r

传输类型	速度	有效值（*m*）	理解
打断	低速或全速	1 – 255	轮询间隔为*m*帧。
	高速	1 – 16	轮询间隔为$2^{(m-1)}$微帧。
等时的	全速或高速	1 – 16	轮询间隔为$2^{(m-1)}$帧/微帧。
大部分	全速或高速	2 – 16	NAK限制为$2^{(m-1)}$帧/微帧。*注意*：值为0或1将禁用NAK超时功能。

3.3.18. FIFOSIZE

FIFOSize是一个8位只读寄存器，用于返回与所选附加TX/Rx端点相关联的FIFO的大小。较低的半字节对所选择的TX端点FIFO的大小进行编码；上半字节对所选择的Rx端点FIFO的大小进行编码。值3–13对应于2^n字节（8–8192字节）的FIFO大小。如果尚未配置终结点，则将显示值0。在TX和Rx端点共享相同FIFO的情况下，Rx FIFO大小将编码为0xF。

注：只有当索引寄存器设置为选择端点1–15和动态之一时，寄存器才具有此解释

未选择"调整大小"。当索引寄存器被设置为选择端点0（见第3.3.4节）时，它有一个特殊的解释，而当使用动态FIFO大小时，返回的结果无效。

地址：1Fh（索引寄存器设置为仅选择端点1–15）；*重置值*：取决于配置

	D7...	D4	D3	...	D0
	Rx FIFO大小		**Tx FIFO大小**		
来自CPU	r	...r	r	...	r
从USB	r	...r	r	...	r

3.4. FIFOx(Addresses 20h–5Fh)

此地址范围为每个端点提供16个CPU访问FIFO的地址。写入这些地址会将数据加载到相应端点的TxFIFO中。从这些地址读取数据会从RxFIFO中卸载相应端点的数据。

地址范围为20h-5Fh，FIFO位于32位双字边界上（端点0位于20h，端点1位于24h...端点15位于5Ch）。

注：（i）根据需要，与FIFO之间的传输可以是8位、16位或32位，并且允许任何访问组合，前提是访问的数据是连续的。但是，与一个数据包相关的所有传输必须具有相同的宽度，以便数据一致地以字节、字或双字对齐。然而，为了完成奇数字节或奇数字传输，最后一次传输可以包含比先前传输更少的字节。
(ii) 根据FIFO的大小和预期的最大数据包大小，FIFO支持单数据包或双数据包缓冲。但是，不支持多数据包的突发写入，因为在写入每个数据包后需要设置标志。
(iii) 在端点1–15上出现STALL响应或TX Strike-Out错误后，相关的FIFO将被完全清除。

3.5.多点控制/STATUS寄存器的编辑

以下小节详细介绍了只有在配置GUI中启用多点选项时才有效的附加控制和状态寄存器。如果未启用多点选项，则不应访问这些重新注册器。

3.5.1. TXFUNCADDR/RXFUNCADDR

注意：在主机模式下是必需的！

TxFuncAddr和RxFuncAdder是7位读/写寄存器，记录要通过相关端点（*EPn*）访问的目标函数的地址。需要为所使用的每个TX端点定义TxFuncAddr；需要为所使用的每个Rx端点定义RxFuncAddr。

注意：必须为终结点0定义TxFuncAddr。EP0上不存在RxFuncAddr寄存器。

地址：分别为80+8*nh*和84+8*nh*；*重置值*7'000

	D6	...	D0
	目标函数地址		
来自CPU	rw	...	rw
从USB	r	...	r

3.5.2. TXHUBADDR/RXHUBADDR集线器ADDR

注意：仅在主机模式下相关！

TxHubAddr和RxHubAddr是8位读/写寄存器，与TxHubPort和RxHubPort一样，只需要在完全或低速设备通过高速USB 2.0集线器连接到TX/Rx Endpoint *EPn*的情况下写入，该集线器执行必要的事务转换以在高速传输和完全/低速传输之间转换。在这种情况下：

- 较低的7位应记录此USB 2.0集线器的地址
- 最高位应该记录集线器是否有多个事务转换器（如果是单个事务转换器，则设置为"0"；如果是多个事务转换程序，则设置"1"）。

注意：如果端点0连接到集线器，则必须为EP0定义TxHubAddr。EP0上不存在RxHubAddr寄存器。

地址：分别为82+8**n* h和86+*8*n*小时；*复位值*：8'000

	D7	D6	...	D0
	多个翻译器	**集线器地址**		
来自CPU	rw	rw	...	rw
从USB	r	r	...	r

3.5.3.TX-HU-BP或T/RX-HU-BP端口

注意：仅在主机模式下相关！

TxHubPort和RxHubPort只需要在完整或低速设备通过高速USB 2.0集线器连接到TX/Rx Endpoint *EPn*的情况下写入，该集线器执行必要的事务转换。在这种情况下，这些7位读/写寄存器需要用于记录USB 2.0集线器的端口，通过该端口访问与端点相关联的目标。

注意：如果终结点0连接到集线器，则必须定义TxHubPort。EP0上不存在RxHubPort寄存器。

地址：分别为83+8**n* h和87+*8*n*小时；*复位值*：7'000

	D6	...	D0
	集线器端口		
来自CPU	rw	...	rw
从USB	r	...	r

这些信息以及上面记录的集线器地址详细信息使MUSBHDRC能够支持拆分事务。

3.6. 在美国注册中心注册

3.6.1. trol上的VC

注意：只写！

VControl是一个UTMI+PHY供应商寄存器，当配置核心时，该寄存器可以选择性地包括在核心中。其大小也是可配置的，并且可以高达32位。寄存器的结构由系统设计者决定，但用户应该注意，UTMI+规范定义了一个4位的VControl寄存器。

该寄存器可以在地址68h处被访问。

用户还应注意：

(i) 如果指定的寄存器超过8位，则必须对整个寄存器进行任何写入，根据情况使用16位或32位写入。不支持写入单个字节。

(ii) 写入的延迟（如在AHB写入周期结束时的CLK的正边缘和加载UTMI+PHY V控制寄存器时的XCLK的正边缘之间测量的）将在Hc+3Xc和Hc+4Xc之间，其中Hc是CLK的周期，Xc是XCLK周期。因此，对核心的VControl、寄存器的连续写入之间的最小周期必须是Hc+4Xc，以确保该值在同步到XCLK域时不会损坏。

3.6. 2. V S T A T US

注意：只读！

VStatus是一个UTMI+PHY供应商寄存器，当配置核心时，该寄存器可以选择性地包括在核心中。其大小也是可配置的，并且可以高达32位。寄存器的结构由系统设计者决定，但用户应注意，UTMI+规范定义了一个8位VStatus寄存器。

该寄存器可以在地址68h处被访问。用户还应

注意：

(i) VSTATUS输入总线每6个XCLK周期采样一次。

(ii) 从PHY改变的VSTATUS输入总线和从VSTATUS寄存器读取的新值（在AHB读取周期结束时测量到CLK的正边缘）之间的延迟将在2Hc+Xc和3Hc+6Xc之间，其中Hc是CLK的周期，Xc是XCLK的周期。

3.6. 3. H W V E R S

HWVers寄存器是一个16位只读寄存器，它返回有关生成核心硬件的RTL版本的信息，特别是RTL版本号（*vxx.yyy*）。

地址：6Ch；*重置值*：取决于版本

	D15	D14	…	D10	D9	…	D0
	RC	***xx***			***yyy***		
来自CPU	r	r	…	r	r	…	r
从USB	r	r	…	r	r	…	r

一点	名称	作用
D15	**RC**	如果RTL是从候选版本中使用的，而不是从核心的完整版本中使用，则设置为“1”。
D14–D10	***xx***	主要版本号（范围0-31）。
D9–D0	***yyy***	次要版本号（范围0–999）。

3.7. ADDITIONAL CONFIGURATION REGISTERS

3.7.1. EPINFO

此8位只读寄存器允许读回设计中包含的TX和Rx端点的数量。

地址：78h；重置值：取决于实施

	D7	D6	D5	D4	D3	D2	D1	D0
	Rx端点				**Tx端点**			
来自CPU	r	r	r	r	r	r	r	r
从USB	r	r	r	r	r	r	r	r

一点	名称	作用
D7–D4	**Rx端点**	在设计中实现的Rx端点的数量。
D3–D0	**Tx端点**	在设计中实现的TX端点的数量。

3.7.2. RAMINFO

这个8位只读寄存器提供有关RAM宽度的信息。

地址：79h；重置值：取决于实施

	D7	D6	D5	D4	D3	D2	D1	D0
	DMA Chans				**RamBits**			
来自CPU	r	r	r	r	r	r	r	r
从USB	r	r	r	r	r	r	r	r

一点	名称	作用
D7–D4	**DMA Chans**	在设计中实现的DMA通道的数量。
D3–D0	**RamBits**	RAM地址总线的宽度。

3.7.3. 我知道

这个8位寄存器允许指定一些延迟。

地址：7Ah；复位值：8'h5C

	D7	D6	D5	D4	D3	D2	D1	D0
	WTCON				**WTID**			
来自CPU	rw	rw	rw	rw	rw	rw	rw	rw
从USB	r	r	r	r	r	r	r	r

一点	名称	作用
D7–D4	**WTCON**	以533.3ns为单位设置要应用的等待以允许用户的连接/断开过滤器。（默认设置对应于2.667μs。）
D3–D0	**WTID**	设置从断言的IDPULLUP到被视为有效的IDDIG的延迟，单位为4.369ms。（默认设置对应52.43ms。）

3.7 . 4.VPLEN

此8位寄存器设置VBus脉冲电荷的持续时间。

地址：7Bh；复位值：8'h3C

	D7	D6	D5	D4	D3	D2	D1	D0
	(msb)			**VPLEN**				**(lsb)**
来自CPU	rw	rw	rw	rw	rw	rw	rw	rw
从USB	r	r	r	r	r	r	r	r

一点	名称	作用
D7–D0	**VPLEN**	以546.1μs为单位设置VBus脉冲充电的持续时间。（默认设置对应于32.77ms。）

3.7. 5. H S _EO F 1

此8位寄存器设置最后事务的开始和高速事务的EOF之间允许的最小时间间隔。

地址：7Ch；复位值：8'h80

	D7	D6	D5	D4	D3	D2	D1	D0
	(msb)			**HS_EOF 1**				**(lsb)**
来自CPU	rw	rw	rw	rw	rw	rw	rw	rw
从USB	r	r	r	r	r	r	r	r

一点	名称	作用
D7–D0	**HS_EOF1**	为高速事务设置EOF之前停止开始新事务的时间，单位为133.3ns。（默认设置对应于17.07μs。）

3.7 . 6.FS_E，共1个

此8位寄存器设置最后一个事务的开始和全速事务的EOF之间允许的最小时间间隔。

地址：7Dh；复位值：8'h77

	D7	D6	D5	D4	D3	D2	D1	D0
	(msb)			**FS_EOF 1**				**(lsb)**
来自CPU	rw	rw	rw	rw	rw	rw	rw	rw

一点	名称	作用
D7–D0	**FS_EOF1**	为全速事务设置EOF之前停止开始新事务的时间，单位为533.3ns。（默认设置对应于63.46µs。）

3.7 . 7.LS _E，共1个

此8位寄存器设置最后事务的开始和低速事务的EOF之间允许的最小时间间隔。

地址：7Eh；复位值：8'h72

	D7	D6	D5	D4	D3	D2	D1	D0
	(msb)			**LS_EOF1**				**(lsb)**
来自CPU	rw	rw	rw	rw	rw	rw	rw	rw
从USB	r	r	r	r	r	r	r	r

一点	名称	作用
D7–D0	**LS_EOF1**	为低速事务设置EOF之前停止开始新事务的时间，单位为1.067µs。（默认设置对应于121.6µs。）

3.7. 8. S O FT _R ST

该8位寄存器将断言输出复位信号NRSTO和NRSTOX为LOW。此寄存器是自清除的，将通过输入NRST重置。

地址：7Fh；复位值：8'000

	未使用						D1	D0
	(msb)			**SOFT_RST**				**(lsb)**
从CPU从USB							rw r	rw r

一点	名称	作用
D7–D2	–	*未使用，总是返回零。*
D1	**NRSTX**	该位的默认值为1'b0；当1被写入该位时，输出NRSTXO将在CLK输入的7个周期的最小延迟内被断言（LOW）。输出NRSTXO将相对于XCLK异步断言和同步去断言。此寄存器是自清除的，将通过输入NRST重置。
D0	**NRST**	该位的默认值为1'b0；当1被写入该位时，输出NRSTO将在CLK输入的7个周期的最小延迟内被断言（LOW）。输出NRSTO将相对于CLK异步断言和同步去断言。此寄存器是自清除的，将通过输入NRST重置。

3.8. 正文摘要

以下小节详细介绍了控制和影响MUSBHDRC操作的附加寄存器。

3.8. 1. RQ PK TCOU NT

注意：仅限主机模式！

对于每个Rx端点1–15，MUSBHDRC提供一个16位RqPktCount寄存器。此读/写寄存器在主机模式下用于指定在一个或多个长度为MaxP的Bulk数据包到Rx Endpoint *n*的块传输中要传输的数据包数量。核心使用此寄存器中记录的值来确定在设置了AutoReq选项（包括在RxCSR寄存器中）的情况下发出的请求数量。

有关更多信息，请参阅第8.5.2节。

注意：在FIFO内，多个数据包组合成一个大容量数据包即为一个数据包。

*地址：300+4*n；重置值：16'0000*

	D15	…	D0
	(msb)	**RqPktCount**	**(lsb)**
来自CPU	rw	…	rw
从USB	rw	…	rw

一点	名称	作用
D15–D0	**RqPktCount**	设置要在块传输中传输的大小为MaxP的数据包数。*仅在设置了AutoReq时用于主机模式。在外设模式或未设置AutoReq时无效。*

3.8. 2. 执行UBL E P A C K ET B U FF ER DI S A B L E

对于每个Rx和Tx端点1–15，MUSBHDRC提供一个DPktBufDis位。这些位位于一组公共的DPktBuf Dis寄存器中。一组寄存器专用于Rx端点，另一组专用于TX端点。这些读/写比特用于在每个端点的基础上控制双分组缓冲的使用。当启用动态FIFO时，它将被忽略。当断言（DPktBufDis等于1'b1）时，无论端点FIFO大小和INMAXP大小关系如何，该位都将禁用对应端点的双数据包缓冲。当去断言（DPktBufDis等于1'b0）时，此位不一定启用双数据包缓冲，而是允许基于端点FIFO大小和INMAXP大小关系来确定双数据包缓存。详见第8.4节。

3.8. 2. 1. R X D P KT BU FD是

Rx DPktBufDis是一个16位寄存器，用于指示哪些Rx端点禁用了MUSBHDRC产品规范第8.4.2.2节中描述的双包缓冲区功能。

*注：*与尚未配置的端点相关的位可以通过写入其各自的寄存器"1"来断言；然而，禁用位将没有可观察到的效果。

地址：340h；重置值：16'0000

	D15	D14	D13	D12	D11	D10	D9	D8
	EP15 RxDis	**EP14 RxDis**	**EP13 RxDis**	**EP12 RxDis**	**EP11 RxDis**	**EP10 RxDis**	**EP9 RxDis**	**EP8 RxDis**
来自CPU	rw	rw	rw	rw	rw	rw	rw	rw
从USB	r	r	r	R	r	r	r	r

	D7	D6	D5	D4	D3	D2	D1	D0
	EP7 RxDis	**EP6 RxDis**	**EP5 RxDis**	**EP4 RxDis**	**EP3 RxDis**	**EP2 RxDis**	**EP1 RxDis**	**未使用**
来自CPU	rw	rw	rw	rw	rw	rw	rw	r
从USB	r	r	r	r	r	r	r	r

一点	名称	作用
D15	**EP15 RxDis**	端点15的Rx双数据包缓冲区禁用。
D14	**EP14 RxDis**	禁用端点14的Rx双数据包缓冲区。
D13	**EP13 RxDis**	禁用端点13的Rx双数据包缓冲区。
D12	**EP12 RxDis**	禁用端点12的Rx双数据包缓冲区。
D11	**EP11 RxDis**	端点11的Rx双数据包缓冲区禁用。
D10	**EP10 RxDis**	端点10的Rx双数据包缓冲区禁用。
D9	**EP9 RxDis**	端点9的Rx双数据包缓冲区禁用。
D8	**EP8 RxDis**	端点8的Rx双数据包缓冲区禁用。
D7	**EP7 RxDis**	端点7的Rx双数据包缓冲区禁用。
D6	**EP6 RxDis**	端点6的Rx双数据包缓冲区禁用。
D5	**EP5 RxDis**	端点5的Rx双数据包缓冲区禁用。
D4	**EP4 RxDis**	端点4的Rx双数据包缓冲区禁用。
D3	**EP3 RxDis**	端点3的Rx双数据包缓冲区禁用。
D2	**EP2 RxDis**	端点2的Rx双数据包缓冲区禁用。
D1	**EP1 RxDis**	端点1的Rx双数据包缓冲区禁用。
D0	**未使用**	保留

3.8.2.2 TXDPKTBUFDIS是

Tx-DPktBufDis是一个16位寄存器，用于指示哪个Tx端点禁用了MUSBHDRC产品规范第8.4.1.2节中描述的双分组缓冲功能。

注：与尚未配置的端点相关的位可以通过写入其各自的寄存器"1"来断言；然而，禁用位将没有可观察到的效果。

地址：342h；重置值：16'0000

	D15	D14	D13	D12	D11	D10	D9	D8
	EP15 TxDis	**EP14 TxDis**	**EP13 TxDis**	**EP12 TxDis**	**EP11 TxDis**	**EP10 TxDis**	**EP9 TxDis**	**EP8 TxDis**
来自CPU	rw	rw	Rw	rw	Rw	rw	rw	rw
从USB	r	r	R	r	R	r	r	r

	D7	D6	D5	D4	D3	D2	D1	D0
	EP7 TxDis	**EP6 TxDis**	**EP5 TxDis**	**EP4 TxDis**	**EP3 TxDis**	**EP2 TxDis**	**EP1 TxDis**	**未使用**
来自CPU	rw	rw	Rw	rw	Rw	rw	rw	r
从USB	r	r	R	r	R	r	r	r

一点	名称	作用
D15	**EP15 TxDis**	端点15的Tx双数据包缓冲区禁用。
D14	**EP14 TxDis**	禁用端点14的Tx双数据包缓冲区。
D13	**EP13 TxDis**	禁用端点13的Tx双数据包缓冲区。
D12	**EP12 TxDis**	禁用端点12的Tx双数据包缓冲区。
D11	**EP11 TxDis**	端点11的Tx双数据包缓冲区禁用。
D10	**EP10 TxDis**	端点10的Tx双数据包缓冲区禁用。
D9	**EP9 TxDis**	端点9的Tx双数据包缓冲区禁用。
D8	**EP8 TxDis**	端点8的Tx双数据包缓冲区禁用。
D7	**EP7 TxDis**	端点7的Tx双数据包缓冲区禁用。
D6	**EP6 TxDis**	端点6的Tx双数据包缓冲区禁用。
D5	**EP5 TxDis**	端点5的Tx双数据包缓冲区禁用。
D4	**EP4 TxDis**	端点4的Tx双数据包缓冲区禁用。
D3	**EP3 TxDis**	端点3的Tx双数据包缓冲区禁用。
D2	**EP2 TxDis**	端点2的Tx双数据包缓冲区禁用。
D1	**EP1 TxDis**	端点1的Tx双数据包缓冲区禁用。
D0	**未使用**	保留

3.8 . 3.C_T_U CH

此寄存器设置啁啾超时。这个数字乘以4表示超时发生之前的XCLK循环数。也就是说，如果XCLK是30MHz，则该数字表示超时发生之前的133ns时间间隔的数量。如果XCLK为60MHz，则此数字表示超时发生前67ns时间间隔的数量。尽管此位是由CLK域中的主机写入的，但使用此值的计数器位于XCLK域中。由于此寄存器中的值是静态的，因此不提供时域交叉。默认值是位于配置文件musbmhdrc_cfg.v中的同名编译器指令的值。

地址：344h；重置值：多种。

	D15	D14	D13	D12	D11	D10	D9	D8
	C_T_UCH[15:8]							
从CPU从USB	rw 不适用	rw 不适用	Rw 不适用	rw 不适用	Rw 不适用	rw 不适用	rw 不适用	rw 不适用
	D7	D6	D5	D4	D3	D2	D1	D0
	C_T_UCH[7:0]							
从CPU从USB	rw 不适用	rw 不适用	Rw 不适用	rw 不适用	Rw 不适用	rw 不适用	rw 不适用	rw 不适用

一点	名称	作用
D15-0	C_T_UCH	可配置Chirp超时定时器；默认值由musbhsfc_xcfg.v文件中的编译器指令确定。如果主机PHY数据宽度为16位（XCLK为30MHz），则默认值为203Ah；如果PHY数据长度为8位（XCLK为60Mhz），则对应于1.1ms的延迟，则默认为4074h。

3.8 . 4.C_T_HS R T N

该寄存器设置从高速恢复信令结束起的延迟，以启用UTM正常操作模式。这个数字乘以4表示超时发生之前的XCLK循环数。也就是说，如果XCLK是30MHz，则该数字表示超时发生之前的133ns时间间隔的数量。如果XCLK为60MHz，则此数字表示超时发生前67ns时间间隔的数量。尽管此位是由CLK域中的主机写入的，但使用此值的计数器位于XCLK域中。由于此寄存器中的值是静态的，因此不提供时域交叉。默认值是位于配置文件musbmhdrc_cfg.v中的同名编译器指令的值。

地址：346h；重置值：多种。

	D15	D14	D13	D12	D11	D10	D9	D8
	C_T_HSRTN[15:8]							
从CPU从USB	rw 不适用	rw 不适用	Rw 不适用	rw 不适用	rw 不适用	rw 不适用	rw 不适用	rw 不适用
	D7	D6	D5	D4	D3	D2	D1	D0
	C_T_HSRTN[7:0]							
从CPU从USB	rw 不适用	rw 不适用	Rw 不适用	rw 不适用	rw 不适用	rw 不适用	rw 不适用	rw 不适用

一点	名称	作用
D15-0	C_T_HSRTN	从高速恢复信令结束到启用UTM正常操作模式的延迟。默认值由musbhsfc_xcfg.v文件中的编译器指令确定。如果主机PHY数据宽度为16比特（XCLK为30MHz），则默认值为19h，如果PHY数据长度为8比特（XCLK为60Mhz），则对应于3us的延迟，则默认为32h。

3.8. 5. C _ T _ H S BT

根据USB 2.0第7.1.19.2节，如果数据包间延迟小于736位时间，则高速主机或设备期望对传输做出响应时，不得使事务超时；如果在816位时间内未看到任何信令，则必须使事务超时。该寄存器表示要加到736比特时间的最小高速超时周期的值。超时周期可以以64个高速比特时间（133ns）为增量来增加。有16个可能的值。默认情况下，加法器为0，因此将高速超时设置为其最小值。使用此寄存器将允许将高速超时设置为大于USB 2.0中指定的最大值的值，从而使MUSBHDRC不符合要求。

地址：348h；重置值：4'b0000

	D3	D2	D1	D0
	（MSB）	HS超时加法器		（LSB）
来自CPU从USB	rw r	rw r	rw r	rw r

一点	名称	作用
D3-D0	C_T_HSBT	以64个高速位时间为增量添加到最小高速超时周期（736位时间）的值。这允许将周转超时时间设置为16个可能的值，如下所示： 登记价值 / HS周转超时（HS位时间） / HS周转超时我们 0 / 736 / 1.534 1. / 800 / 1.667 2. / 864 / 1.801 3. / 928 / 1.934 4. / 992 / 2.067 5. / 1056 / 2.201 6. / 1120 / 2.334 7. / 1184 / 2.467 8. / 1248 / 2.601 9 / 1312 / 2.734 10 / 1376 / 2.868 11 / 1440 / 3.001 12 / 1504 / 3.134 13 / 1568 / 3.268 14 / 1632 / 3.401 15 / 1696 / 3.534

3.9. DMA REGISTERS

只有当MUSBHDRC配置为使用至少一个内部DMA通道时，DMA寄存器才可用。每个通道有一组寄存器。

3.9.1. DMA_INTR

该寄存器为每个DMA通道提供一个中断。此中断寄存器在读取时被清除。当该寄存器的任何位被设置时，输出DMA_NINT被断言为低。导致设置中断的事件在第17节（可选DMA控制器描述）中进行了描述。只有当相应通道的DMA中断使能位被启用时，该寄存器中的位才会被设置（寄存器DMA_CNTL.D3）。

地址：200小时；复位值：00h

	D7	D6	D5	D4	D3	D2	D1	D0
	DMA_INTR[7:0]							
从CPU从USB	rw 设置	rw 设置	rw 设置	rw 设置	rw 设置	r 设置	r 设置	R 设置

一点	名称	作用
D7	CH8 DMA_INTR	通道8 DMA中断
D6	CH7 DMA_INTR	通道7 DMA中断
D5	CH6 DMA_INTR	通道6 DMA中断

D4	CH5 DMA_INTR	通道5 DMA中断
D3	CH4 DMA_INTR	通道4 DMA中断
D2	CH3 DMA_INTR	通道3 DMA中断
D1	CH2 DMA_INTR	通道2 DMA中断
D0	CH1 DMA_INTR	通道1 DMA中断

3.9. 2. D M A _ CNTL

仅当MUSBHDRC配置为使用至少一个内部DMA通道时，此寄存器才可用。该寄存器为每个通道提供了DMA传输控制。使能、传输方向、传输模式、DMA突发模式都由该寄存器控制。

*地址：204h+（n-1）*10h；n=通道编号1至8；复位值：00h*

	D10	D9	D8
	DMA_BRSTM		DMA_ERR
从CPU从	rw	rw	rw
USB	r	r	rw

	D7	D6	D5	D4	D3	D2	D1	D0
	DMAEP				DMAIE	DMAMODE	DMA_DIR	DMA_EN
从CPU从	rw	rw	rw	rw	rw	rw	rw	rw
USB	r	r	r	r	r	r	r	r

一点	名称	作用
D10-D9	DMA_BRSTM	突发模式 00=突发模式0：未指定长度的突发01=突发模式1:INCR4或未指定长度 10=突发模式2:INCR8、INCR4或未指定长度 11=突发模式3:INCR16、INCR8、INCR4或未指定长度
D8	DMA_ERR	总线错误位。表示在输入AHB_HRESPM[1:0]上观察到总线错误。此位由软件清除。
D7-D4	DMAEP	此通道分配给的端点编号。
D3	DMAIE	DMA中断启用。
D2	DMAMODE	此位选择DMA传输模式。 0 = DMA Mode0 Transfer 1 = DMA Mode1 Transfer

D1	DMA_DIR	此位选择DMA传输方向。 0 = DMA Write (RX Endpoint) 1 = DMa Read (TX Endpoint)
D0	DMA_ENAB	此位启用DMA传输，并将使传输开始。

3.9. 3. DMA_ADDR

该寄存器标识相应DMA通道的当前存储器地址。写入该寄存器的初始存储器地址必须具有一个值，使其模4值等于0。也就是说，DMA_ADDR[1:0]必须等于2'b00。此寄存器的低两位是只读的，不能由软件设置。随着DMA传输的进行，内存地址将随着字节的传输而增加。

*地址：208h+（n-1）*10h；n=通道编号1至8；复位值：00h*

	D31	D30	D29	D28	D27	D26	D25	D24
	DMA_ADDR[31:24]							
从CPU从USB	rw rw	rw rw	rw rw	rw rw	rw rw	rw rw	rw rw	rw rw
	D23	D22	D21	D20	D19	D18	D17	D16
	DMA_ADDR[23:16]							
从CPU从USB	rw rw	rw rw	rw rw	rw rw	rw rw	rw rw	rw rw	rw rw
	D15	D14	D13	D12	D11	D10	D9	D8
	DMA_ADDR[15:8]							
从CPU从USB	rw rw	rw rw	rw rw	rw rw	rw rw	rw rw	rw rw	rw rw
	D7	D6	D5	D4	D3	D2	D1	D0
	DMA_ADDR[7:0]							
从CPU从USB	rw rw	rw rw	rw rw	rw rw	rw rw	rw rw	r rw	r rw

一点	名称	作用
D31–D0	DMA_ADDR	DMA内存地址。 **请注意，写入该寄存器的初始内存地址必须具有一个值，使其模4值等于0。也就是说，DMA_ADDR[1:0]必须等于2'b00。此寄存器的低两位是只读的，不能由软件设置。**

3.9. 4. DMA_COUNT

此寄存器标识传输的当前DMA计数。软件将设置传输的初始计数，用于识别整个传输长度。随着计数的进行，此计数会随着字节的传输而递减。

地址：20Ch+（n-1）*10h；n=通道编号1至8；复位值：00h

	D31	D30	D29	D28	D27	D26	D25	D24
	DMA_COUNT[31:24]							
从CPU从USB	rw rw	rw rw	rw rw	rw rw	rw rw	rw rw	rw rw	rw rw
	D23	D22	D21	D20	D19	D18	D17	D16
	DMA_COUNT[23:16]							
从CPU从USB	rw rw	rw rw	rw rw	rw rw	rw rw	rw rw	rw rw	rw rw
	D15	D14	D13	D12	D11	D10	D9	D8
	DMA_COUNT[15:8]							
从CPU从USB	rw rw	rw rw	rw rw	rw rw	rw rw	rw rw	rw rw	rw rw
	D7	D6	D5	D4	D3	D2	D1	D0
	DMA_COUNT[7:0]							
从CPU从USB	rw rw	rw rw	rw rw	rw rw	rw rw	rw rw	r rw	r rw

一点	名称	作用
D31–D0	DMA_COUNT	对应DMA通道的DMA内存地址。**注意：如果DMA是在计数为0的情况下启用，则不会请求总线，并且将生成DMA中断。**

3.10. DYNAMIC FIFO REGISTER

只有当MUSBHDRC配置为使用动态FIFO大小时，动态FIFO寄存器才可用。每个端点有一组寄存器，不包括端点0。这些是索引寄存器，因此要访问它们，地址0Eh处的INDEX寄存器必须设置为适当的终点。第19节介绍了动态Fifo寄存器的限制和使用。

3.10.1. TXFIFOSZ

TxFIFOsz是一个5位寄存器，用于控制所选TX端点FIFO的大小。

地址：62h；重置值：5'000

	D4	D3	D2	D1	D0
	DPB	SZ3	SZ2	SZ1	SZ0
来自CPU 从USB	rw r	rw r	rw r	rw r	rw r

一点	名称	作用
D4	**DPB**	定义是否支持双数据包缓冲。当"1"时，支持双数据包缓冲。当"0"时，仅支持单数据包缓冲。
D3–D0	**SZ[3:0]**	允许的最大数据包大小（*在传输前*在大容量/高带宽数据包的FIFO内进行任何拆分之*前*——见第8.4.1.3、8.4.1.4和8.5.3节） **SZ[3:0]** / **数据包大小（字节）** 0 0 0 0 — 8. 0 0 0 1. — 16 0 0 1. 0 — 32 0 0 1. 1. — 64 0 1. 0 0 — 128 0 1. 0 1. — 256 0 1. 1. 0 — 512 0 1. 1. 1. — 1024 1. 0 0 0 — 2048 1. 0 0 1. — 4096 如果DPB＝0，则FIFO也将是该大小；如果DPB＝1，则FIFO将是该大小的两倍。

3.10.2. RXFIFOSZ

RxFIFOsz是一个5位寄存器，用于控制所选Rx端点FIFO的大小。

地址：63h；*重置值*：5'000

	D4	D3	D2	D1	D0
	DPB	**SZ3**	**SZ2**	**SZ1**	**SZ0**
来自CPU	rw	rw	rw	rw	rw
从USB	r	r	r	r	r

一点	名称	作用
D4	**DPB**	定义是否支持双数据包缓冲。当"1"时，支持双数据包缓冲。当"0"时，仅支持单数据包缓冲。
D3–D0	**SZ[3:0]**	允许的最大数据包大小（*在接收到大容量/高带宽数据包后*，在FIFO内进行任何组合*后*——见第8.4.2.3、8.4.2.4和8.5.2节） **SZ[3:0]** / **数据包大小（字节）** 0 0 0 0 — 8. 0 0 0 1. — 16 0 0 1. 0 — 32 0 0 1. 1. — 64 0 1. 0 0 — 128 0 1. 0 1. — 256 0 1. 1. 0 — 512 0 1. 1. 1. — 1024 1. 0 0 0 — 2048 1. 0 0 1. — 4096 如果DPB＝0，则FIFO也将是该大小；如果DPB＝1，则FIFO将是该大小的两倍。

3.10 .3 . T X F I for A D D

TxFIFOadd是一个14位寄存器，用于控制所选Tx端点FIFO的起始地址。

地址：64h；重置值14'0000

	D13	D12	…	D0
	–	**AD12**	…	**AD0**
来自CPU	rw	rw	rw	rw
从USB	r	r	r	r

一点	名称	作用
D13	–	*保留以备将来使用。*
D12–D0	**广告[12:0]**	以8字节为单位的端点FIFO的起始地址如下：

广告[12:0]				起始地址
0	0	0	0	0000
0	0	0	1.	0008
0	0	0	2.	0010
		…		…
1.	F	F	F	FFF8

3.10 .4 . R X F I FOA D D

RxFIFOadd是一个14位寄存器，用于控制所选Rx端点FIFO的起始地址。

地址：66h；重置值14'0000

	D13	D12	…	D0
	–	**AD12**	…	**AD0**
来自CPU	rw	rw	rw	rw
从USB	r	r	r	r

一点	名称	作用
D13	–	*保留以备将来使用。*
D12–D0	**广告[12:0]**	以8字节为单位的端点FIFO的起始地址如下：

广告[12:0]				起始地址
0	0	0	0	0000
0	0	0	1.	0008
0	0	0	2.	0010
		…		…
1.	F	F	F	FFF8

4. 计时和复位

4.1. CL OC KI N G

MUSBHDRC被设计为从AHB总线时钟中获取其系统时钟CLK。这避免了任何重新同步逻辑

在MUSBHDRC和AHB之间，允许对MUSBHDSC寄存器和FIFO进行单周期访问。

核心可被计时的最大频率由MUSBHDRC可被合成的最大频率确定。这通常是0.35中的80MHz❍ 技术，0.25中的100MHz❍ 技术，以及0.18中的120MHz❍ 技术

核心（和AHB总线）可以被计时的最小频率取决于所选择的UTMI宽度和技术实现。对于某些技术实现，8位UTMI允许以接近30MHz的最低速度对内核进行计时。

对于特定技术实现的实际最小频率的更高分辨率，给出了以下边界方程：

$$Tclk <= (2Txclk) - (Tskew/2) - (Tetup/2) -$$

Tclk是保证在XCLK/CLK时域交叉上正确传输接收数据的信号CLK的最大周期。

Txclk是信号XCLK的周期。

Tsetup是该技术所需的设置延迟。

Thold是该技术所需的保持延迟。

Tskew是以下信号组之间传播延迟的最坏情况差异：Musbhdrc.usync_1.rxbuff0[15:0]

Musbhdrc.usync_1.rxbuff1[15:

0]Musbhdrc.usync_1-rxbfal0

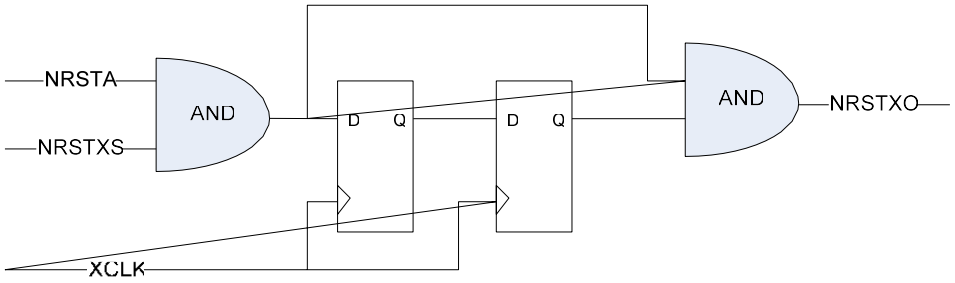
Musbhdrc.usync_1.rxbfal1

请注意，上面的等式假设了一个理想的时钟。请添加与技术相关的参数以获得更高级别的分辨率。

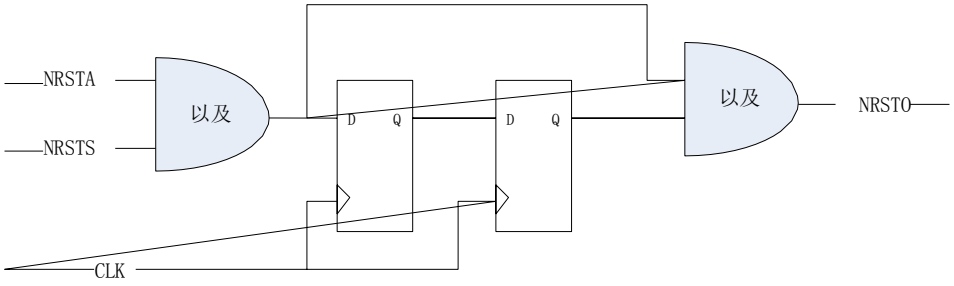
4.2. RESET

MUSBHDRC有两个时钟域；XCLK域是由PHY从接收到的数据中恢复的时钟，以及由AHB总线使用的CLK域。提供了两个输入复位信号（NRST和NRSTX）；每个时钟域一个。NRST可以异步断言，并且必须同步地去断言（与CLK同步）。NRSTX可以异步断言，并且必须同步地去断言（与XCLK同步）。如果这些同步复位不可用，可以使用MUSBHDRC提供的同步逻辑。在这种情况下，异步复位是在端口NRSTA上输入的。两个输出信号NRSTO和NRSTXO由NRSTA产生如下：

生成与XCLK同步的重置



产生与CLK同步的复位



信号NRSTXS和NRSTS是内部信号，它们为软件提供复位核心的能力。NRSTS和NRSTXS分别通过向寄存器7F写入位0和1而被断言。为了利用该同步逻辑，输出NRSTO必须连接到输入NRST，并且输出NRSTXO必须连接至输入NRSTX。如果客户不使用此同步逻辑，则输入NRSTA应被绑定为高电平或低电平，并且输出NRSTO和NRSTXO应保持未连接。NRSTA不用于MUSBHDRC中的任何其他目的。

5. CP U I N TE表面

基本的MUSBHDRC内核提供了一个32位同步CPU接口，该接口遵循为与AMBA AHB总线兼容而指定的格式。

所有输入都在CLK的正边缘上采样，并且输出跟随CLK的正向边缘而变化。

6. 日期

AHB数据接口为32位宽。

数据可以以单字节、16位半字或32位字的形式传输。

在半字和字传输中，字节通常先传输最低阶字节，如下所示：（B0表示要传输的第一个字节）

Little endian传输

传输大小	地址偏移	D[31:24]	D[23:16]	D15:8]	D[7:0]
32位	0	地下三层	地下二层	地下一层	B0
16位	0			地下一层	B0
16位	2.	地下一层	B0		

8位	0				B0
8位	1.			B0	
8位	2.		B0		
8位	3.	B0			

7. RAM接口

所有端点的FIFO都在一个同步单端口RAM块中实现。RAM不包括在MUSBHDRC中，但应通过RAM接口连接到MUSBHDrc。

RAM地址总线、输出数据总线和控制信号都在CLK的正沿之后变为有效。读取数据在下面的正边缘上被计时到核心中。（参见第20.3节和第20.4节中的时序图。）在CLK的负沿上计时的同步RAM块将需要小于半个时钟周期的访问时间。

RAM数据总线宽度为32位。

8. 美国

MUSBHDRC被设计为连接到符合UTMI+规范（级别3）的收发器宏小区。它可以被配置为连接到8位60MHz或16位30MHz收发器宏小区。（或者，内核可以与第8.1节中描述的可选USB 1.1全速PHY接口一起使用。）

对于*即时*操作，MUSBHDRC提供：

- DRVVBUS输出，可用于"a"设备配置，将+5V驱动到USB VBus导线上
- CHRGVBUS输出，可用于"B"设备配置，向VBus提供适当的脉冲以启动会话（例如，通过将VBus充电到会话启动阈值）
- DISCHRGVBUS输出，可在"B"设备配置中使用，以将VBus放电到足够低的水平，以启动会话请求协议（SRP）
- DPPULLDOWN和DMPULLDOWN输出，用于在核心用于与另一个USB设备的点对点通信时，根据需要连接/断开D+和D线上的下拉电阻器。（这些电阻器的尺寸应在*USB On-The-Go*规范中指定。）

MUSBHDRC还具有VBUSVALID、AVALID和SESSEND输入，以向其识别VBus相对于*移动*设备的会话控制中所涉及的各种阈值的水平。这些信号应连接到电压比较器，电压比较器分别检测VBus电压何时高于VBus有效阈值（要求在4.4V和4.75V之间）、何时高于"A"设备的会话有效阈值（需要在0.8V和2V之间）以及何时高于会话结束阈值（需要在0.2V和0.8V之间）。

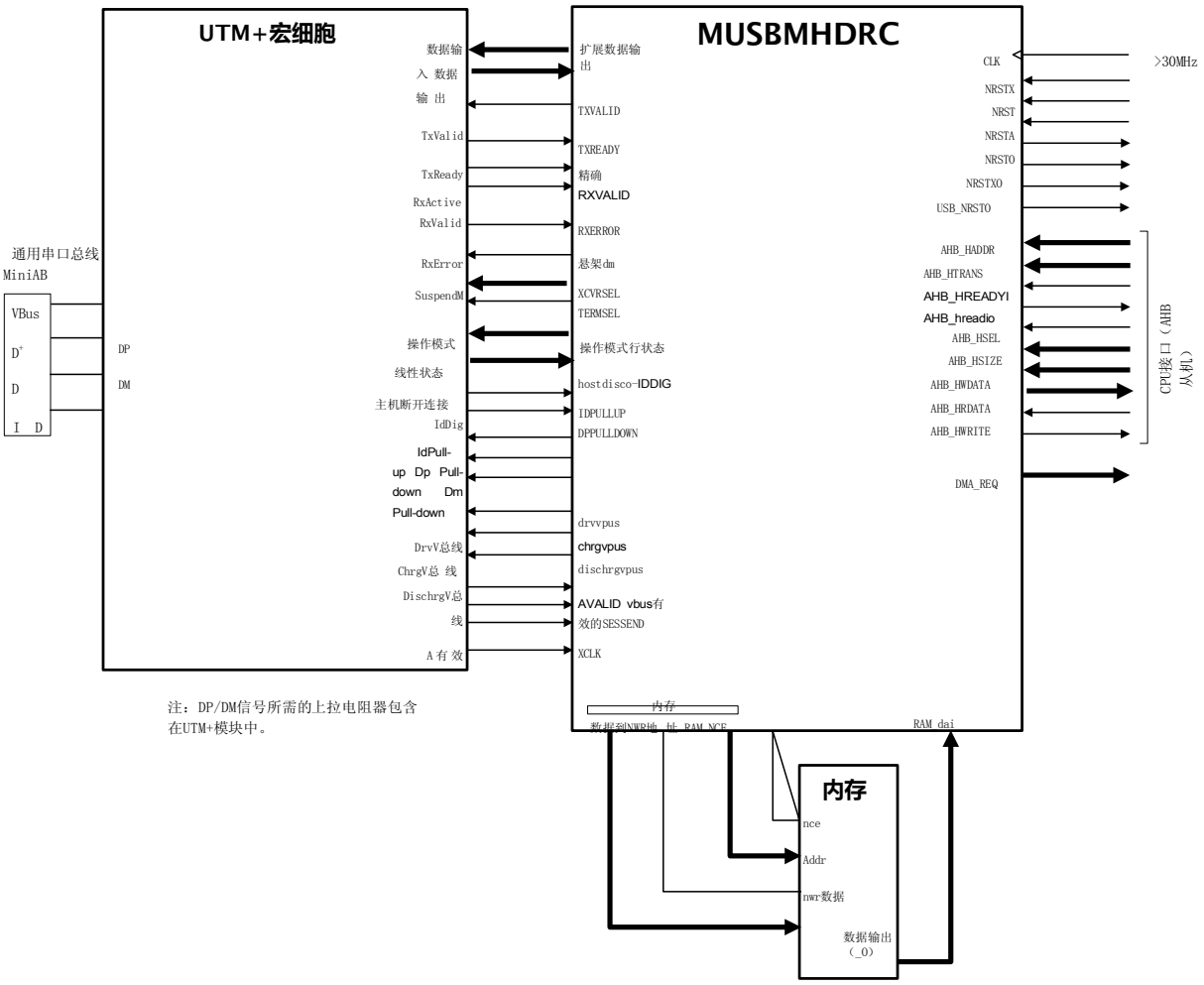
（这些电压范围的公差细节在*USB on the Go*规范中给出。）

DRVVBUS和CHRGVBUS信号的状态反映了在DevCtl寄存器中进行的选择。VBUSVALID、AVALID和SESSEND信号的状态可以从DevCtl寄存器的VBus[1:0]位推导出来，其值指示VBus相对于所选择的VBUSVALD、AVALID和SESSEND电平的当前电平。（UTMI+规范中给出了所需充电电流、时间等的详细信息。）

USB接口还具有IDDIG信号，该信号指示插入MUSBHDRC的设备是A型还是B型。当插入B型设备时，此输入应为高电平，当插入a型设备时应为低电平。设备类型通过对输入ID线进行采样来确定，只有在需要时，才能通过使用核心的IDPULLUP输出在ID线上切换适当的上拉电阻来启用此采样。显示示例连接的图表

如下页所示。

连接到UTM+宏小区的示例



注：DP/DM信号所需的上拉电阻器包含在UTM+模块中。

8.1. 选项A L USB 1.1 PHY INT E RF A C E

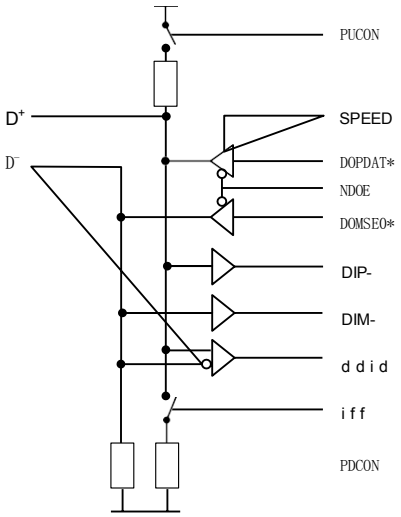
与MUSBHDRC核心一起提供的是一个模块（**fsi**），如果需要，可以与MUSBMDRC顶级模块（**MUSBMHDRC**）一起实例化，以允许核心使用USB 1.1 PHY而不是UTMI+PHY以全速或低速运行。如果需要，可以在**fsi**模块旁边实例化其他模块（**i2c**和**i2cmstr**），以支持使用带有i2c控制的USB 1.1 PHY的MUSBHDRC。

在Tx操作中，**fsi**模块提供8位并行到串行数据的转换、同步和分组结束位的自动添加、NRZI编码和自动位填充。在Rx操作中，该模块为接收数据、将串行数据转换为8位并行数据、剥离同步和分组结束位、NRZI解码和填充位错误检查提供数字锁相环。

fsi模块与提供的**MUSBMHDRC_fsp.v**包装器中的MUSBHDRC核心一起实例化。对信号I/O的影响如下文第8.1.1节所示。如果需要I2C总线选项（需要更多的**I2C**和**i2cmstr**模块），那么要使用的包装器是**musbmhdrc_I2C.v**，对I/O的影响如下文第8.1.2节所示。（**i2cmstr**模块在**i2c**模块中实例化。）

必须提供外部60MHz时钟来驱动MUSBHDRC的XCLK输入、**fsi**模块和**i2c**模块（如果使用）。此外，如果使用**musbmhdrc_i2c.v**包装器来应用可选的i2c总线接口，则除此之外，还必须编辑**musbmhdrc _pcfg.v**文件，以至少设置正确的PHY地址和内部寄存器配置（如文件本身中所述）。*注：***musbmhdrc_pcfg.v**文件是为与Philips ISP1301收发器一起使用而提供的设置文件。当核心与不同的I2C控制收发器一起使用时，将需要不同的定义。

核心（和配置脚本）还提供输出驱动程序格式的选择——DP/DM（即D+/D-）或DAT/SE0。默认选择DP/DM格式；DAT/SE0格式可以通过配置脚本或者通过定义C_ DATSE0配置参数来选择。



*在使用DAT/SE0输出驱动程序格式的情况下进行适当解码后

如上所述，DOPDAT和DOMSE0输出应通过缓冲器驱动D+和D-线，当NDOE为低时（在使用DAT/SE0输出驱动器格式的适当解码之后）启用缓冲器。同样，DIP和DIM输入应由连接到D+和D-的单端驱动器驱动。DIDIFF输入应由连接到D+和D-的差分驱动器驱动。SPEED信号可用于优化驱动器的转换速率以进行全速或低速操作。

类似地，PUCON、PU_LO和PDCON信号旨在用于将上拉电阻器或下拉电阻器连接到D+线。当MUSBHDRC作为主机运行时，下拉电阻器需要连接到USB D+和D-线。当它作为外设工作时，D线上需要相同的下拉电阻器，但USB上需要上拉电阻器

D+导线。当PUCON为高时，上拉电阻器应连接在D+导线和+3.3V之间。当PUCON为低电平时，应断开该上拉电阻器。类似地，当PDCON为高电平时，下拉电阻器应连接在D+和接地之间，而当PDCON低电平时，应断开下拉电阻器。PU_LO信号通过指示USB总线何时空闲，使设备能够在USB 2.0规范第7.1.5节允许的较高值和较低值上拉电阻器之间进行选择。

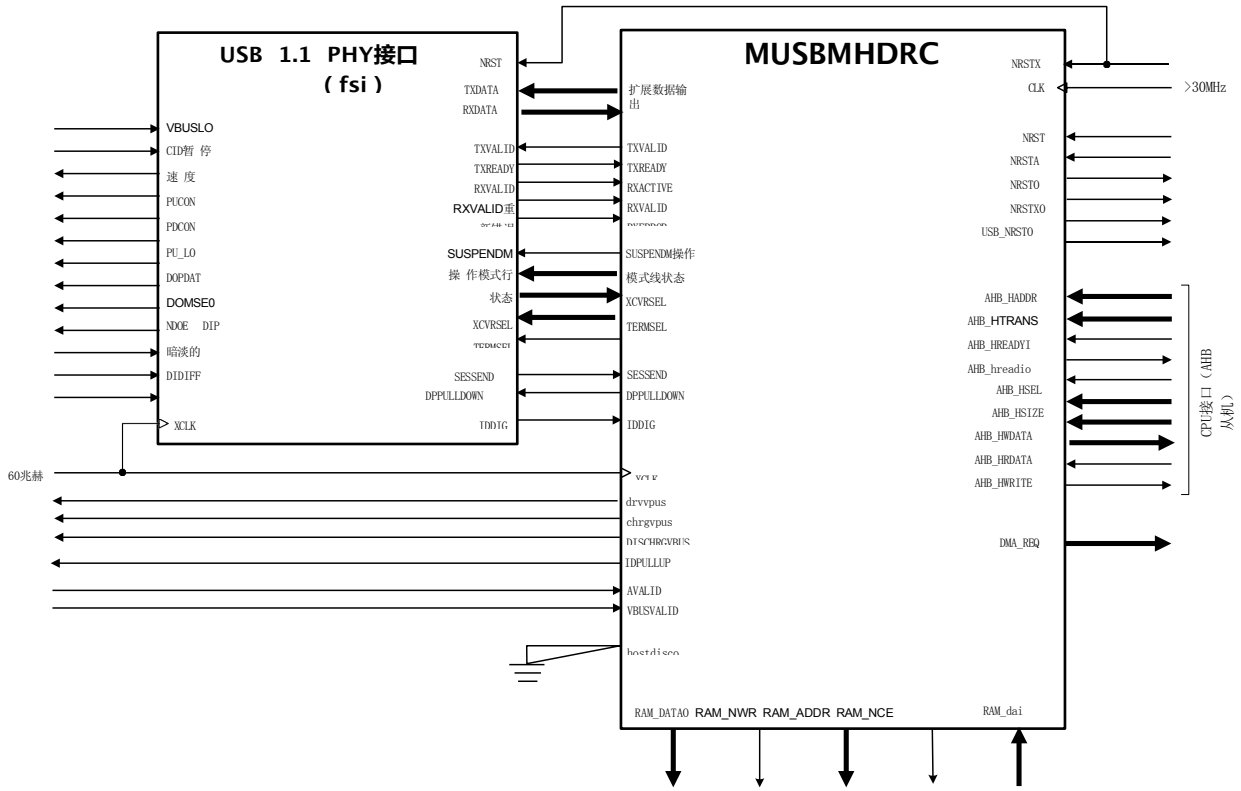
VBUSLO信号与VBUSVALID和AVALID一起用于会话请求和实现*USB On-The-Go*设备之间的主机协商协议。这些信号需要连接到电压比较器。CID用于指示插入并应（通过上拉）连接到迷你AB插座的ID引脚的连接器类型（迷你-A或迷你-B）。

注意：当MUSBHDRC处于挂起模式（Suspend为高电平）时，驱动器可能会断电。POWERDWN输出也被断言。为了省电，该信号可以用于停止CLK。但是，DIP的单端驱动器必须保持通电状态，以便MUSBHDRC能够检测到总线上的恢复信号。

（如果您需要有关将MUSBHDRC与USB 1.1 PHY一起使用的更多信息，请联系客户支持。）

8.1. 1. THE STANDARD USB 1。TERFACE中的1 PHY

下图显示了**fsi**模块如何连接到MUSBHDRC核心，以及使用**MUSBMHDRC_fsp.v**包装器时显示的接口。



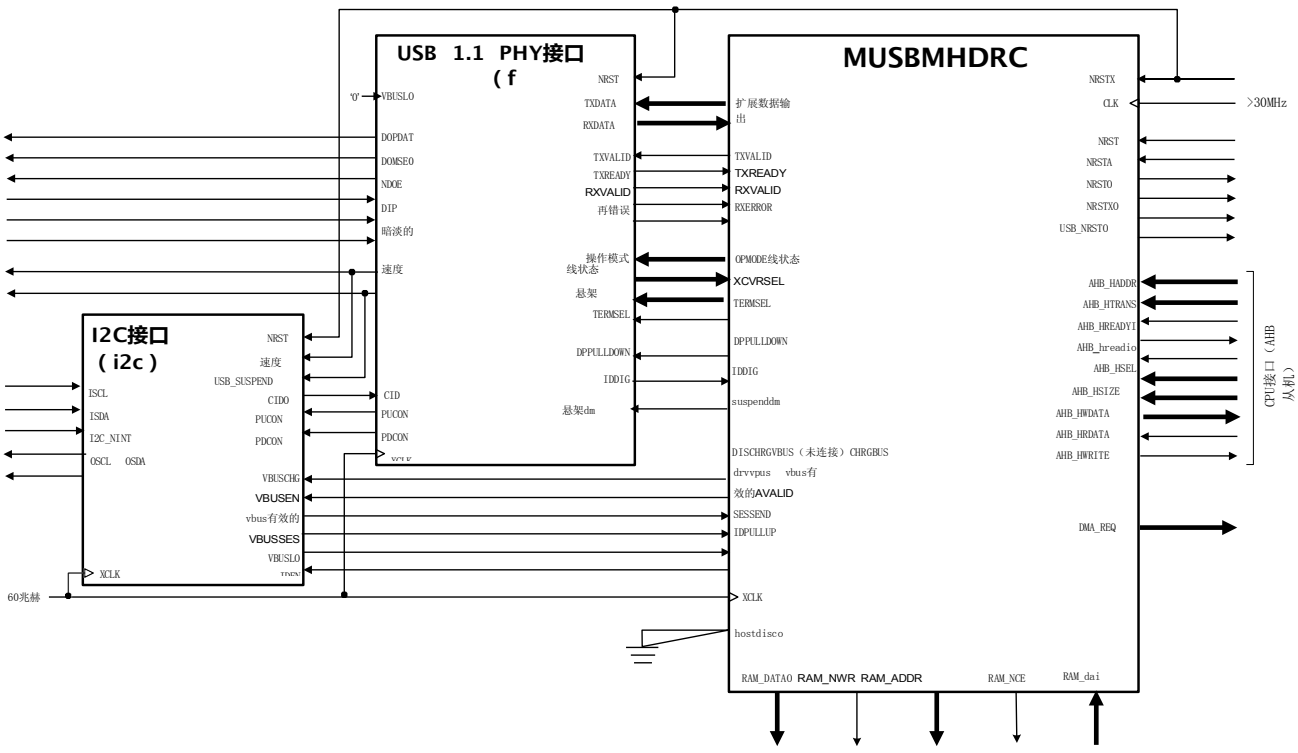
下表列出了将此模块添加到MUSBHDRC核心会引入整体引脚输出的信号。

信号	类型	说明
VBUSLO	输入	VBus与会话结束阈值（要求在0.2V和0.8V之间）相比。1=高于会话结束阈值，0=低于会话结束阈值。
中央信息区	输入	连接器ID，通过对设备ID行进行采样推导得出。1=B型，0=A型。

信号	类型	说明
暂停	输出	当MUSBHDRC处于挂起模式时，此信号变高。
速度	输出	收发器工作速度。1=全速，0=低速。
PUCON	输出	高电位时，应将上拉电阻器连接到D+。
PDCON	输出	当为高时，应将下拉电阻器连接到D+。（D-上所需的下拉电阻器应永久连接。）
PU_LO	输出	当为高时，该信号指示USB空闲，因此可以使用较低值的上拉电阻器（如果实现的话）。当为低时，表示USB正忙，因此应使用较高值的上拉电阻器。
DOPDAT	输出	根据核心配置提供D+输出或DAT输出。
DOMSE0	输出	根据核心配置提供D输出或SE0输出。
NDOE	输出	DOP、DOM的输出启用。低激活。
倾斜	输入	D+单端输入。
暗淡的	输入	D—单端输入。
DIDIFF	输入	差分输入。

8.1. 2. USB 1。1 PHY INTERFACE WITH I²C-BUS CONTROL OPTION

下图显示了**i2c**模块如何连接到核心，以及当使用**musbmhdrc_i2c.v**包装器。



注：目前仅支持具有单个I2C主机的I2C系统。

下表列出了使用**musbmhdrc_i2c.v**包装器将信号引入到MUSBHDRC核心的整体引脚输出中的信号。

信号	类型	说明
暂停	输出	当MUSBHDRC处于挂起模式时，此信号变高。
速度	输出	收发器工作速度。1=全速，0=低速。
DOPDAT	输出	根据核心配置提供D+输出或DAT输出。
DOMSE0	输出	根据核心配置提供D输出或SE0输出。
NDOE	输出	DOP、DOM的输出启用。低激活。
倾斜	输入	D+单端输入。
暗淡的	输入	D—单端输入。
DIDIF	输入	差分输入。
ISCL	输入	I2C时钟输入。
ISDA	输入	I2C数据输入。
I2C_NINT	输入	I2C中断（低电平有效）。
OSCL	输出	I2C时钟输出。
OSDA	输出	I2C数据输出。

注：I2C总线接口需要两个具有开路集电极（或开路漏极）输出和施密特输入的双向缓冲器。这些缓冲器的输入线应连接到ISDA/ISCL。这些缓冲器的输出线应连接到OSDA/OSL，使得当OSDA/OCL为低时，对应的输出缓冲器被启用（输出低），而当OSDA/SCL为高时，对应的输入缓冲器被禁用（输出高阻抗）。

有关此接口的更多信息，请参阅作为文件**musbmhdrc_i2c_an.pdf**提供的应用说明。

8.2. 软连接/DISCONNECT

注意：仅限外围模式！

MUSBHDRC可以通过软件控制其与USB总线的连接。

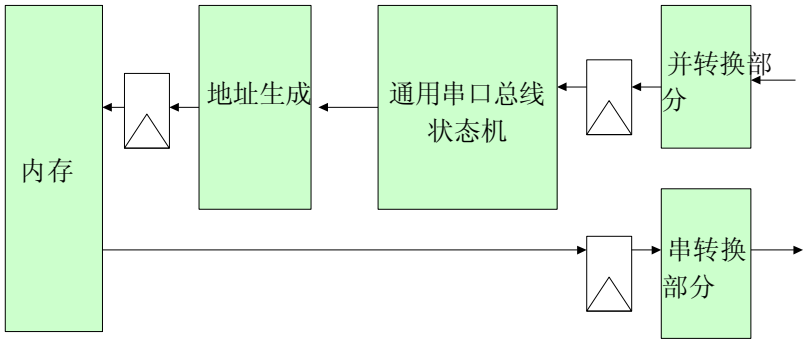
当MUSBHDRC在外围模式下操作时，通过设置/清除电源寄存器的第6位（被识别为软连接位），可以在正常模式和非驱动模式之间切换与MUSBHDrc一起使用的符合UTMI+的PHY。当该软连接位设置为1时，PHY被置于其正常模式，并且USB总线的D+/D-线被启用。当启用此功能且软连接位为零时，PHY将进入非驱动模式（OPMODE[1:0]设置为01b），D+和D-为三态。然后，MUSBHDRC将看起来已断开与USB总线上其他设备的连接。

硬件重置（NRST=0）后，软连接被清除为0。因此，MUSBHDRC将显示断开连接，直到软件将软连接设置为1。然后，应用软件可以选择何时将PHY设置为其正常模式。初始化过程较长的系统可能会使用此方法来确保初始化完成，并且系统已准备好在连接到USB之前执行枚举。

8.3.BUS TURN-AROUND TIME CONSID时代

通过任何MUSBHDRC实现的总线周转时间是所使用的PHY内的Rx和Tx延迟与MUSBHDRR内的SIE决策时间的组合结果。

MUSBHDRC内数据包的SIE决策路径如下所示：



数据包的SIE决策路径

PHY在XCLK域中操作，而CPU接口在CLK域中操作。因此，数据必须在这两个域之间传输。MUSBHDRC的体系结构将这种转换置于UTM接口。（将转换放置在CPU接口会将等待状态引入CPU数据传输，而将其放置在核心的任何其他点会导致核心不同部分之间的时序问题。）因此，对SIE决策时间有贡献的操作序列为：

1. 同步到CLK
2. 解码数据包
3. 决定响应
4. 获取数据（如果需要）
5. 对数据包进行编码
6. 同步到XCLK

如果所需的公交车转弯时间难以实现，我们建议提高CLK速度。

8.3 . 1.1. 1. 1.OPERATIONAS HOST或PERIPHERAL

MUSBHDRC可以在一系列不同的环境中使用。它可以用作连接到传统USB主机（如PC）的高速或全速“外围设备”。它可以在与另一个“外围”设备的点对点数据传输中用作主机或外围设备，或者，如果另一个设备也包含双重角色控制器，则这两个设备可以根据需要切换角色。（第二个设备可以是高速、全速或低速USB功能。）或者MUSBHDRC可以在“多点”设置中用作一系列此类外围设备的主机。

在任何情况下，MUSBHDRC和它所连接的设备之间都支持Control、Bulk、Isochronous或Interrupt事务。

MUSBHDRC是作为主机还是作为外设，取决于设备连接在一起的方式。每条USB电缆都有一个"A"端和一个"B"端。如果电缆的"A"端插入包含MUSBHDRC的设备，则MUSBHDSC将扮演主机设备的角色，并进入"主机"模式。（主机模式位（DevCtl.D2）将设置为"1"。）如果电缆的"B"端已插入，则MUSBHDRC将进入"外围"模式，主机模式位将设置为"0"。

如果MUSBHDRC连接到单个设备，并且该设备包含双重角色控制器，则可以使用信号切换两个设备的角色，而无需在设备之间切换电缆。第15节主机协商中解释了MUSBHDRC可以在外围角色和主机角色之间切换的条件。

注：MUSBHDRC的多点功能与一系列寄存器相关联，这些寄存器记录了设备功能分配给各个MUBMHDRC核心端点以及设备功能特性，如端点编号、操作速度和事务类型（见下页第5.1节）。尽管主要与将核心用作多个设备的主机有关，但当将核心用作单个目标设备的主机时，也需要设置这些寄存器。

8．4． 运行期间

当主机模式位（DevCtl.D2）被清除时，MUSBHDRC在外设模式下工作。

本节介绍核心在将MUSBHDRC用作外围设备时的Tx端点、Rx端点、进入/退出挂起模式和识别帧开始方面的操作。

第15节：主机协商解释了MUSBHDRC在外围模式下运行的条件。

8.4．1．生命周期

当MUSBHDRC在外围模式下运行时，in事务的数据通过MUSBHDrc的Tx FIFO进行处理。

端点1至15的Tx FIFO的大小由MUSBHDRC配置文件中的配置常数确定，或者在选择动态FIFO大小的情况下，通过TxFIFO2寄存器确定。可以放置在Tx端点的FIFO中用于传输的数据包的最大大小是可编程的，并且由写入该端点的TxMaxP寄存器的值（最大有效载荷　事务数/微帧（如适用））。

除了使用动态FIFO大小的情况外，当最大数据包大小设置为小于或等于FIFO大小的一半时，对IN事务启用双数据包缓冲，当最大信息包大小大于FIFO大小的二分之一时，启用单数据包缓冲。（在选择动态FIFO大小的情况下，单包或双包缓冲的使用是端点FIFO规范的一部分——见第19节）当启用双包缓冲时，两个数据包可以在FIFO中缓冲：当启用单包缓存时，即使数据包小于FIFO大小的一半，也只能缓冲一个数据包。

注意：为任何端点设置的最大数据包大小不得超过FIFO大小。您还应注意，当FIFO中有数据时，不应写入TxMaxP寄存器，因为可能会出现意外结果。

8.4．1．1.新的P A C K E T BUFF E R I NG

如果Tx端点FIFO的大小小于该端点的最大数据包大小的两倍（如在TxMaxP寄存器中设置的，或者在使用动态FIFO大小的情况下，在TxFIFO2寄存器中设置），则只能在FIFO中缓冲一个数据包，并且启用单数据包缓冲。

由于要发送的每个数据包都加载到Tx FIFO中，因此需要设置TxCSR中的TxPktRdy位。如果设置了TxCSR中的AutoSet位，当最大大小的数据包加载到FIFO时，TxPktRdy位将自动设置。对于小于最大值的数据包大小以及可能不使用AutoSet的情况（高带宽等时/中断事务），TxPktRdy将始终

必须手动设置（即通过CPU）。

当TxPktRdy比特被手动或自动设置时，该分组被认为准备发送。TxCSR中的FIFONotEmpty位也被设置。

当数据包已成功发送时，TxPktRdy和FIFONotEmpty都将被清除，并生成适当的Tx端点中断（如果启用）。然后可以将下一个数据包加载到FIFO中。

8.4. 1. 2.公共事业部

注：如果端点的相应DPktBufDis位在Tx DPktBuf Dis寄存器中被断言（等于1'b1），则禁用双数据包缓冲（有关详细信息，请参阅3.8.2）。此位的默认设置已启用（等于1'b0）。

如果Tx端点FIFO的大小至少是该端点的最大数据包大小的两倍（如在TxMaxP寄存器中设置的，或者在使用动态FIFO大小的情况下，在TxFIFO2寄存器中设置），则可以在FIFO中缓冲两个数据包，并启用双数据包缓冲。

由于要发送的每个数据包都加载到Tx FIFO中，因此需要设置TxCSR中的TxPktRdy位。如果设置了TxCSR中的AutoSet位，当最大大小的数据包加载到FIFO时，TxPktRdy位将自动设置。对于小于最大值的数据包大小以及可能不使用AutoSet（高带宽等时/中断事务）的情况，TxPktRdy将始终必须手动设置（即由CPU设置）。

当TxPktRdy比特被手动或自动设置时，该分组被认为准备发送。TxCSR中的FIFONotEmpty位也被设置。

在加载第一个数据包之后，TxPktRdy立即被清除，并且产生中断。现在可以将第二个数据包加载到Tx FIFO中，并再次设置TxPktRdy（如果数据包是最大大小，则手动或自动）。两个数据包现在都可以发送了。

成功发送每个数据包后，TxPktRdy将被清除，并生成适当的Tx端点中断（如果启用），以发出信号，表明现在可以将另一个数据包加载到Tx FIFO中。此时FIFONotEmpty位的状态指示可以加载多少数据包。如果设置了FIFONotEmpty位，则FIFO中还有另一个数据包，并且只能再加载一个数据包包。如果FIFONotEmpty位被清除，则FIFO中没有数据包，可以再加载两个数据包。

8.4. 1. 3.H I GH B A N D WIDT H I S ochronou S/I NT ERRU PT END P O I NT S

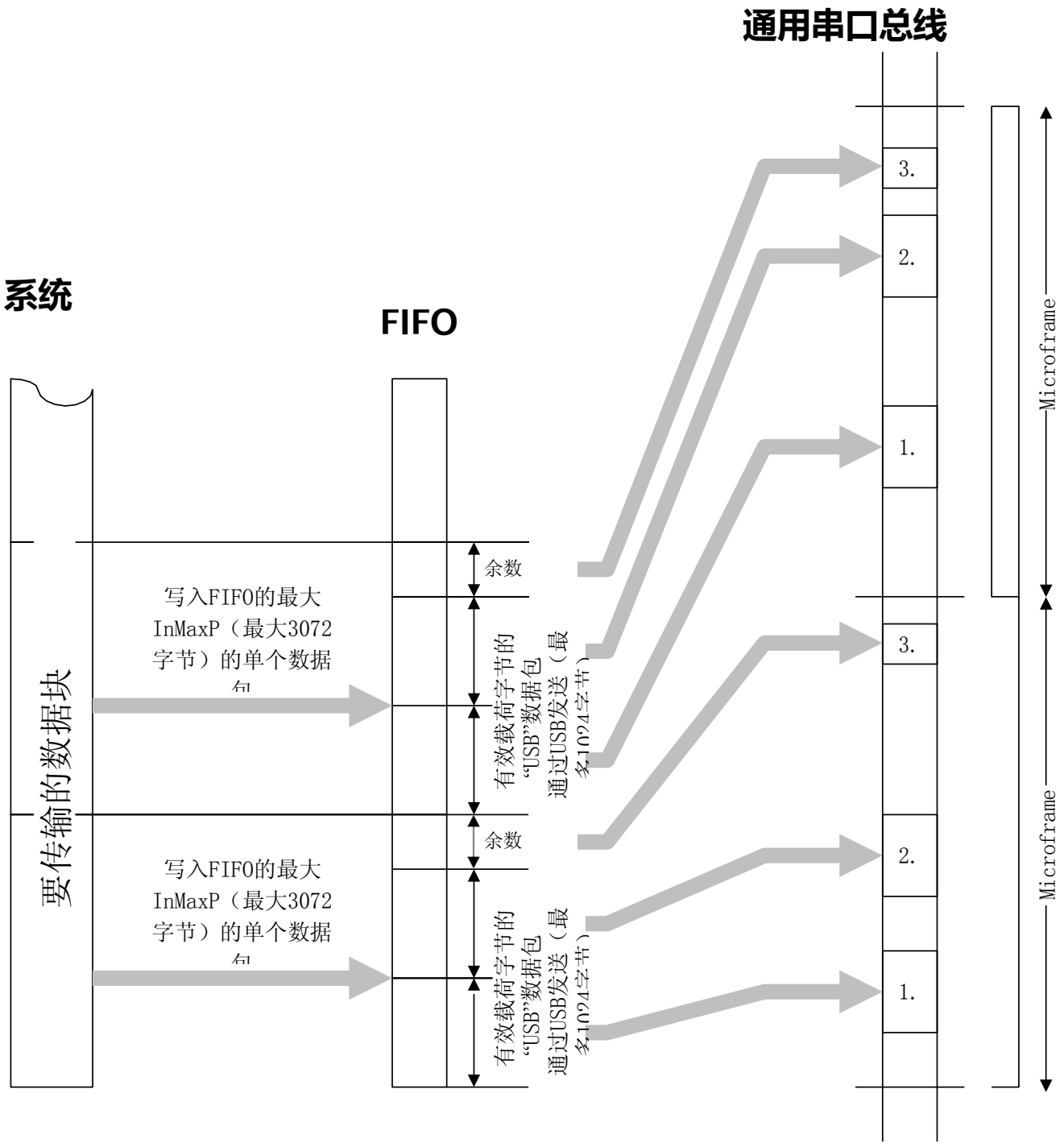
在高速模式下，为高带宽等时/中断事务设置的Tx端点可以在任何微帧中传输多达三个"USB"数据包，每个数据包的有效载荷高达1024字节，对应于每个微帧高达3072字节的数据传输速率。

MUSBHDRC通过允许用户加载高达3072字节（即3　1024字节）输入到相关联的FIFO中。从CPU中的软件的角度来看，操作与上述单包缓冲或双包缓冲（视情况而定）完全相同，不同之处在于TxPktRdy总是需要手动设置（即由CPU设置），因为AutoSet不以高带宽等时/中断传输进行操作。

加载到FIFO中的任何大于最大有效载荷的数据包都会自动拆分为最大有效载荷或更小的"USB"包，以便通过USB传输。每个微帧传输的USB数据包的数量和每个数据包中的最大有效载荷是通过TxMaxP寄存器定义的。TxMaxP寄存器的位10–0确定任何USB数据包中的最大有效载荷，而位12、11确定在一个微帧中可以发送的此类数据包的最大数量（2或3）。它们一起设置了可以加载到FIFO中的数据包的最大大小。

将始终发送至少一个USB数据包：在同一微帧中发送的其他USB数据包的数量将取决于加载到FIFO中的数据量。TxPktRdy位将被清除，并且只有当所有分组都已发送时才会产生中断。

每个USB数据包都是响应于in令牌而发送的。如果在微帧结束时，MUSBHDRC没有接收到足够的IN令牌来发送所有USB数据包（例如，因为接收到的IN令牌之一已损坏），则将从FIFO中清除剩余数据。TxPktRdy位随后将被清除，并且TxCSR寄存器中的IncompTx位被设置为指示并非所有加载到FIFO的数据都被发送。



8.4 . 1.4.关于一个特定的问题

批量操作中传输的数据包由USB规范定义为8、16、32、64或512字节大小，512字节选项仅适用于高速传输。然而，对于一些系统设计，应用软件在单个操作中将比在单个USB操作中传输的数据量更大的数据量写入端点可能更方便。一个特定的情况是，在某些情况下，相同的端点用于512字节的高速传输，但在其他情况下用于全速传输。当以全速操作时，单个操作中传输的最大数据量仅为64字节。

为了适应这种情况，MUSBHDRC包括一个"Tx批量数据包拆分"配置选项，如果选择该选项，则允许将较大的数据包写入批量Tx端点，然后将其拆分为适当（指定）大小的数据包，以便在USB总线上传输。（对于比单个USB数据包更大容量的Bulk Rx端点的读取，也存在类似的选项——请参见第8.4.2.4节。）

在此选项下，端点的TxMaxP寄存器增加到16位，寄存器的底部11位定义每个单独传输的有效*载荷*，而顶部5位定义*乘法器*。然后，应用软件可以写入大小*乘数*的数据包　有效*载荷*到FIFO，然后MUSBHDRC将其分割成所述有效载荷的单独分组以通过USB传输。从应用软件的角度来看，所产生的操作与单个USB数据包的传输没有什么不同，除了写入的数据包的大小。

该设施是作为配置选项而非标准功能提供的，因为它显著增加了闸门数量。

注意：此功能仅用于批量端点，并且根据USB规范，有效载荷必须为8、16、32、64或512字节，512字节选项仅适用于高速传输。TxMaxP寄存器中记录的有效负载还必须与端点的标准端点描述符的*wMaxPacketSize*字段匹配。相关联的FIFO也必须足够大，以在被分割之前容纳数据包。

8.4.2. OUT TRANSACTION HANDLING AS A PERIPHERAL组织

当MUSBHDRC在外围模式下操作时，OUT事务的数据通过MUSBHDrc的Rx FIFO进行处理。

端点1至15的Rx FIFO的大小由MUSBHDRC配置文件中的配置常数确定，或者在选择动态FIFO大小的情况下，通过RxFIFO2寄存器确定。Rx端点在任何帧或微帧（在高速模式下）中接收的最大数据量是可编程的，并且由写入该端点的RxMaxP寄存器的值来确定。（最大有效载荷　事务数/微帧（如适用））。

除了使用动态FIFO大小的情况外，当最大数据包大小设置为小于或等于FIFO大小的一半时，将为OUT事务启用双数据包缓冲，当最大信息包大小大于FIFO大小的二分之一时，将启用单数据包缓冲。（在选择动态FIFO大小的情况下，单包或双包缓冲的使用是端点FIFO规范的一部分——见第19节。）当启用双包缓冲时，两个数据包可以在FIFO中缓冲：当启用单包缓存时，即使数据包小于FIFO大小的一半，也只能缓冲一个数据包。

注意：最大数据包大小不得超过FIFO大小。

8.4.2.1新的PACKET BUFFERING

如果Rx端点FIFO的大小小于该端点的最大数据包大小的两倍（如在RxMaxP寄存器中设置的，或者在使用动态FIFO大小的情况下，在RxFIFO2寄存器中设置），则只能在FIFO中缓冲一个数据包，并且启用单数据包缓冲。

当数据包被接收并放置在Rx FIFO中时，RxCSR中的RxPtRdy位（D0）和FIFOFull位（D1）被设置，并且适当的Rx端点被生成（如果被启用）以发出信号，表示数据包现在可以从FIFO中卸载。

在数据包被卸载之后，RxPktRdy比特需要被清除，以便允许接收更多的数据包。如果设置了RxCSR（D15）中的AutoClear位，并且从FIFO中卸载了最大大小的数据包，则RxBitRdy位将自动清除。FIFOFull位也被清除。对于小于最大值的数据包大小，RxPktRdy将始终必须手动清除（即由CPU清除）（例外情况，请参阅寄存器描述）。

8.4.2.2公共事业部

注意：如果在Rx中断言端点的相应DPktBufDis位（等于1'b1），则禁用双数据包缓冲

DPktBufDis寄存器（详见3.8.2）。此位的默认设置已启用（等于1'b0）。

如果Rx端点FIFO的大小至少是端点的最大数据包大小的两倍（如在RxMaxP寄存器中设置的，或者在使用动态FIFO大小的情况下，在RxFIFO2寄存器中设置），则可以缓冲两个数据包，并启用双数据包缓冲。

当要接收的第一个数据包被加载到Rx FIFO中时，RxCSR中的RxPtRdy位被设置，并且适当的Rx端点中断被生成（如果被启用）以发出信号，表示数据包现在可以从FIFO中卸载。*注*：RxCSR中的FIFOFull位此时未设置：仅当接收到第二个数据包并将其加载到Rx FIFO中时才设置。

在每个数据包被卸载之后，RxPktRdy需要被清除，以便允许接收更多的数据包。如果设置了RxCSR中的AutoClr位，并且从FIFO中卸载了最大大小的数据包，则RxPtRdy位将自动清除。对于小于最大值的数据包大小，RxPktRdy将始终必须手动清除（即由CPU清除）。

如果清除RxBitRdy时FIFOFull位被设置为1，则MUSBHDRC将首先清除FIFOFull位。然后，它将再次设置RxPktRdy，以指示FIFO中有另一个数据包等待卸载。

8.4.2.3.High Bandwidth Isochronous/Interrupt Endpoints

在高速模式下，为高带宽等时事务设置的Rx端点可以在任何微帧中接收最多三个"USB"数据包，每个数据包的有效载荷最多为1024字节，对应于每个微帧最多3072字节的数据传输速率。高带宽中断事务同样可以在主机模式下接收，但请注意，在外围模式下不支持高带宽中断事件。

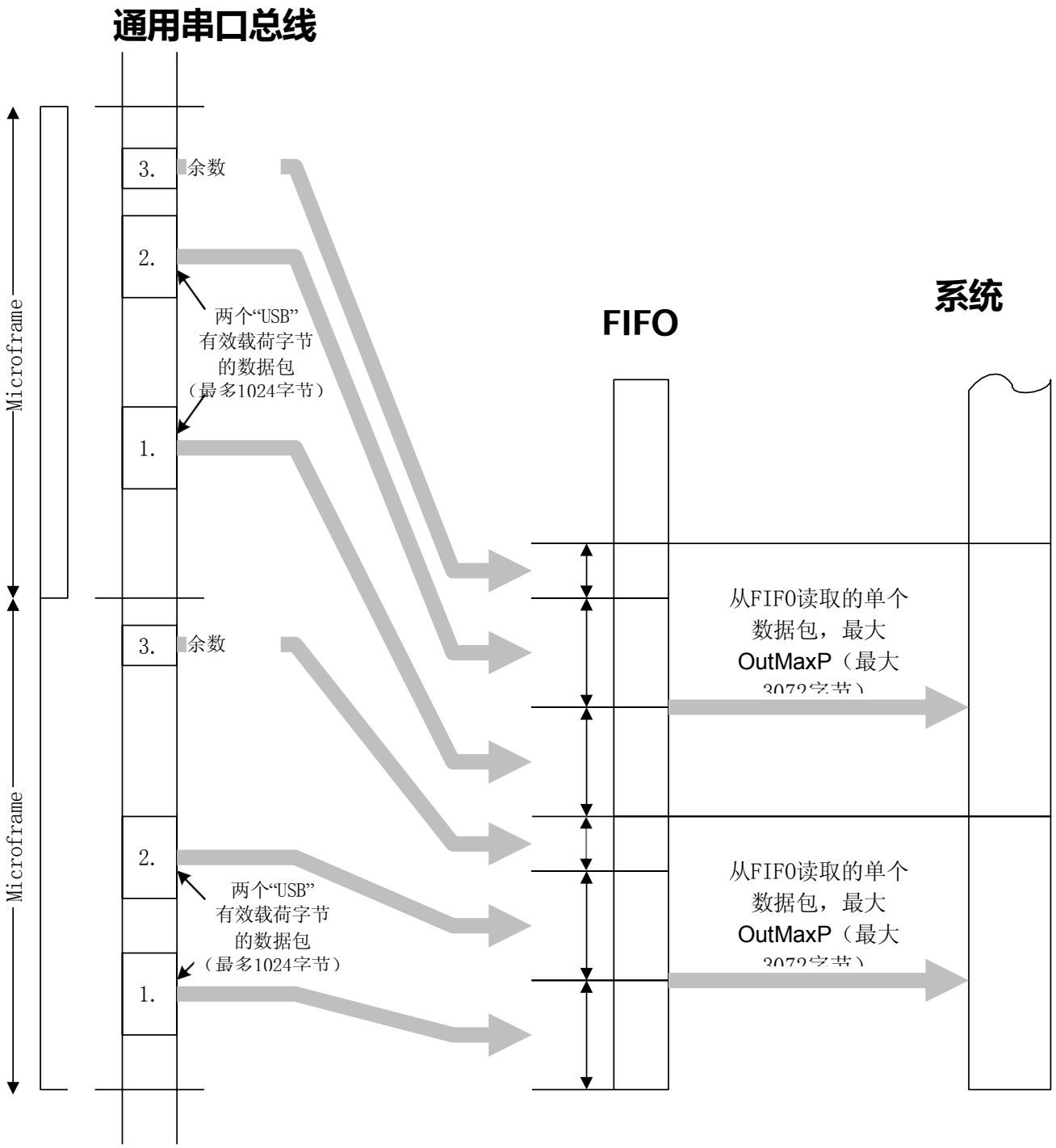
MUSBHDRC通过将在微帧期间接收到的所有USB数据包自动组合为高达3072字节（即3× 1024字节）。从CPU中的软件的角度来看，操作与上述单包缓冲或双包缓冲（视情况而定）完全相同，不同之*处在于，由于*AutoClear不以高带宽等时传输进行操作，因此总是需要手动（即由CPU）清除RxPktRdy。

可以在任何微帧中接收的USB分组的最大数量和这些分组的最大有效载荷通过RxMaxP寄存器来定义。RxMaxP寄存器的位10–0确定任何USB数据包中的最大有效载荷，而位12、11确定微帧中可以接收的这些数据包的最大数量（2或3）。

在任何微帧中发送的USB数据包的数量将取决于要传输的数据量，并且通过用于各个数据包的PID来指示。如果到微帧结束时还没有接收到所指示的分组数量，则RxCSR寄存器中的IncompRx位将被设置为指示FIFO中的数据不完整。同样，如果接收到错误数据类型的数据包，则PID错误位为RxCSR寄存器将被设置。然而，在每种情况下，仍将产生中断，以允许从FIFO读取已接收的数据。

注：报告PID错误或IncompRx的情况取决于接收到的数据包的精确顺序。当核心在外围模式下运行时，详细信息如下。（当核心在主机模式下运行时，一组单独的细节适用：请参阅第8.5.2节。）

预计pkt数量	收到的数据pkt	回答	预计pkt数量	收到的数据pkt	回答
1.	数据0（“0”）	好啊	3.	D0	好啊
	数据1（’D1’）	PID错误设置		D1	IncompRx集合
	数据2（’D2’）	PID错误设置		D2	IncompRx集合
	MDATA（’DM’）	PID错误设置		DM	IncompRx集合
2.	D0	好啊		DM D0	PID错误设置
	D1	IncompRx集合		DM D1	好啊
	D2	IncompRx设置 +PID错误设置		DM D2	IncompRx集合
	DM	IncompRx集合		DM DM	IncompRx集合
	DM D0	PID错误设置		DM DM D0	PID错误设置
	DM D1	好啊		DM DM D1	PID错误设置
	DM D2	PID错误设置		DM DM D2	好啊
	DM DM	PID错误设置		DM DM DM	PID错误设置



8.4 . 2.4.关于一个特定的问题

批量操作中传输的数据包由USB规范定义为8、16、32、64或512字节大小，512字节选项仅适用于高速传输。然而，对于一些系统设计，应用软件在单个操作中从端点读取比在单个USB操作中传输的数据量更大的数据量可能更方便。一个特定的情况是，在某些情况下，相同的端点用于512字节的高速传输，但在其他情况下用于全速传输。当以全速操作时，单个操作中传输的最大数据量仅为64字节。

为了适应这种情况，MUSBHDRC包括一个"Rx批量数据包合并"配置选项，如果选择该选项，则会使MUsbHDRC在应用软件读取之前，将通过USB总线接收的数据包合并为更大的数据包。（对于写入容量比单个USB数据包大的批量Tx端点，也存在类似的选项——请参见第8.4.1.4节。）

在此选项下，端点的RxMaxP寄存器增加到16位，寄存器的底部11位定义每个单独传输的有效*载荷*，而顶部5位定义*乘法器*。然后，MUSBHDRC将其接收的适当数量的USB数据包合并为一个大小*倍增*的单个数据包。 在断言RxPktRdy以提醒应用软件存在要在FIFO中读取的分组之前，在FIFO中的有效*载荷*。结果数据包的大小在RxCount中报告。从应用软件的角度来看，除了读取的数据包的大小之外，所产生的操作与接收单个USB数据包没有什么不同。

该设施是作为配置选项而非标准功能提供的，因为它增加了闸门数量。

注意：此功能仅用于批量端点，并且根据USB规范，有效载荷必须为8、16、32、64或512字节，512字节选项仅适用于高速传输。RxMaxP寄存器中记录的有效负载还必须与端点的标准端点描述符的*wMaxPacketSize*字段匹配。相关联的FIFO也必须足够大以容纳合并的数据分组。

还要注意，只有当接收到指定数量的数据包或接收到"短"USB数据包（即小于端点指定有效载荷的数据包）时，才会设置RxPktRdy。如果使用的协议是端点接收批量传输的协议，该批量传输是记录的有效载荷大小的倍数，并且没有短数据包来终止它，则RxMaxP寄存器不应被编程为期望比传输中的数据包更多的数据包（否则，软件在传输结束时不会中断）。

8.4.3.广告

MUSBHDRC核心自动响应USB总线上的某些条件或主机的操作。详情如下：

STALL ISsue DTO CONTROL transFER

在以下条件下，MUSBHDRC核心将自动向控制传输发出STALL握手：

1. 主机在控制传输的OUT data阶段发送的数据比SETUP阶段设备请求中指定的数据多。

   当CPU卸载最后一个OUT数据包并设置DataEnd后，主机发送OUT令牌（而不是IN令牌）时，MUSBHDRC会检测到这种情况。

2. 主机在控制传输的IN数据阶段请求的数据比在SETUP阶段设备请求中指定的数据多。

   当CPU已经清除TxPktRdy并响应于主机发出的ACK将DataEnd设置为应该是最后一个数据包之后，主机发送IN令牌（而不是OUT令牌）时，MUSBHDRC检测到这种情况。

3. 主机使用OUT数据令牌发送超过MaxP的数据。
4. 主机为OUT状态阶段发送超过零长度的数据包。

ZERO LENGTH OUT DATA PACKET SIN CONTROL TRANSFERS

使用零长度OUT数据包来指示控制传输的结束。在正常操作中，只有在传输了设备请求的整个长度之后（即，在CPU设置了DataEnd之后），才应该接收这样的分组。然而，如果主机在传输了整个长度的设备请求之前发送了一个零长度的OUT数据包，这表示传输提前结束。在这种情况下，MUSBHDRC将自动从FIFO中清除CPU加载的为数据阶段做好准备的任何In令牌，并设置SetupEnd。

8.4 . 4.怀疑端外围

当USB上3毫秒没有任何活动时，MUSBHDRC将进入挂起模式。如果已启用挂起中断，此时将生成一个中断。当处于挂起模式时，SUSPENDM输出将变低（如果启用）：这可用于将PHY置于挂起模式。此外，POWERDWN输出被断言：该信号可以用于停止CLK，从而在这种状态下节省电力。POWERDWN然后保持断言，直到从总线上断开电源（指示设备已断开连接）或在总线上检测到恢复信号或重置信号。

当检测到Resume（恢复）信号时，MUSBHDRC将退出Suspend（暂停）模式，并将SUSPENDM设为高电平。作为响应，PHY应退出挂起状态。如果Resume（恢复）中断已启用，将生成一个中断。CPU还可以通过设置电源寄存器中的恢复位来强制MUSBHDRC退出挂起模式。当设置此位时，MUSBHDRC将退出挂起模式，并将恢复信号驱动到总线上。CPU应在10毫秒（最大15毫秒）后清除此位，以结束Resume信令。

当CPU退出挂起模式时，不会生成恢复中断。

8.4. 5. 我 的 T A R T-O F F R A ME

当MUSBHDRC在外围模式下操作时，在全速模式下，它应该每毫秒从主机接收一次帧开始数据包，在高速模式下，则每125微秒从主机接收。

当接收到SOF分组时，包含在分组中的11比特帧号被写入帧寄存器，并且在SOF_pulse上产生持续一个CLK比特周期的输出脉冲。还会生成SOF中断（如果在IntrUSBE寄存器中启用）。

一旦MUSBHDRC已经开始接收SOF分组，它期望每毫秒（或每125微秒）接收一个。如果在1.00358毫秒（或125.125毫秒）之后没有接收到SOF数据包□ s） ，假设数据包已经丢失，并且尽管帧寄存器没有被更新，但SOF_PULSE（以及SOF中断，如果启用的话）仍然被生成。MUSBHDRC将继续每毫秒（或125微秒）生成SOF_PULSE，并且当再次成功接收到这些分组时，将这些脉冲重新同步到接收到的SOF分组。

8. 5. 在主机上运行

当主机模式位（DevCtl.D2）设置为“1”时，MUSBHDRC作为主机运行，用于与另一个USB设备进行点对点通信，或者当连接到集线器时，用于与多点设置中的整个设备进行通信。支持高速、全速和低速USB功能，既可用于点对点通信，也可用于通过集线器进行操作。（必要时，核心会自动执行必要的事务转换，以允许低速或全速设备与USB 2.0集线器一起使用。）

支持Control、Bulk、Isochronous或Interrupt事务。

本节介绍核心在将MUSBHDRC用作主机时应用的Tx端点、Rx端点、事务调度、进入/退出挂起模式和重置方面的操作。当核心连接到集线器时，会自动选择主机模式。第15节：主机协商解释了MUSBHDRC在主机模式下进行点对点操作的条件。

8.5 . 1.开发服务

只有在配置GUI中启用了多点配置时，以下设置要求才适用于core。如果未启用多点选项，则不应执行以下设置。

在作为主机访问任何设备之前——无论是点对点通信还是通过集线器进行多点通信——需要为每个使用的Rx或Tx端点设置相关的RxFuncAddr或TxFuncAdder寄存器，以记录被访问设备的功能地址。如果全速或低速设备通过高速USB 2.0集线器连接到MUSBHDRC，集线器地址和集线器端口的详细信息也需要记录在相应的RxHubAddr/TxHubAddr和RxHubPort/TxHubPort寄存器中。这允许核心支持拆分事务。

此外，设备运行的速度（高、满或低）需要记录在设备访问的每个端点的Type0（端点0）、TxType或RxType寄存器中。

RxFuncAddr、TxFuncAdder、RxHubAddr、TxHubAddr，RxHubPort、TxHubPort都是7位读/写寄存器。有关这些寄存器的更多信息，请参见第3.5.2节。第3.3.4节、第3.3.14节和第3.3.16节分别给出了关于Type0、TxType和RxType寄存器的信息。

对于多点通信，应该注意的是，这些寄存器中的设置记录了核心端点对与连接设备相关联的功能的*当前*分配。为了最大限度地增加支持的设备数量，MUSBHDRC允许动态更改此分配——只需更新这些寄存器中记录的地址和速度信息。

在受影响的端点上完成任何正在进行的事务后，需要对端点到设备功能的分配进行任何更改。

关于将端点分配给设备功能以及在不同分配之间切换的进一步信息在第11.1节中给出。

8.5.2. IN TRANSACTION HANDLING AS A主机

当MUSBHDRC作为主机操作时，IN事务以与当MUSBMDRC作为外围设备操作时OUT事务处理方式类似的方式处理，不同之处在于事务需要首先通过设置RxCSR中的ReqPkt位来启动。这向事务调度程序指示此端点上存在活动事务。然后，事务调度器向目标函数发送一个IN令牌。

当数据包被接收并放置在Rx FIFO中时，RxCSR中的RxPtRdy位被设置，并且适当的Rx端点中断被生成（如果被启用）以发出信号，表示数据包现在可以从FIFO中卸载。

当数据包已卸载时，应清除RxPktRdy。RxCSR寄存器中的AutoClear位可用于在从FIFO中卸载最大大小的数据包（例外情况，请参阅寄存器描述）时自动清除RxBitRdy。

RxCSR中还有一个AutoReq位，当RxPktRdy位被清除时，它会使ReqPkt位自动设置。AutoClear和AutoReq位可以与DMA控制器一起使用，以在没有CPU干预的情况下执行完整的批量传输。如果要传输已知数量的MaxP数据包，则应在与端点相关的RqPktCount寄存器中设置该数量（见第3.8.1节）。核心在每次请求后递减RqPktCount寄存器中的值。当该值从1递减到0时，AutoReq位被清除，以防止任何进一步的事务被尝试。对于传输大小未知的情况，RqPktCount应保留为零。AutoReq随后将保持设置，直到被短分组（即小于MaxP）的接收所清除，例如可能发生在批量传输结束时。

如果目标函数用NAK响应Bulk/Interrupt IN令牌，则MUSBHDRC将继续重试事务，直到达到已设置的任何NAK限制。但是，如果目标函数以STALL响应，则MUSBHDRC不会重试事务，而是会使用RxCSR寄存器组中的RxStall位中断CPU。如果目标函数在所需时间内没有响应IN令牌（或者数据包中存在CRC或比特填充错误），则MUSBHDRC将重试该事务。如果三次尝试后目标功能仍未响应，则MUSBHDRC将清除ReqPkt位，并用RxCSR设置中的Error位中断CPU。*注意：*在高带宽中断事务的情况下，主机将在单个微帧期间尝试2或3个事务，并在接收到所有数据包时生成中断。如果这些事务中的任何一个没有被目标确认，那么在同一微帧期间将不会尝试进一步的事务。

注：在任何微帧中发送的USB数据包的数量将取决于要传输的数据量，并通过用于单个数据包的PID来指示。如果到微帧结束时还没有接收到所指示的分组数量，则RxCSR寄存器中的IncompRx位将被设置为指示FIFO中的数据不完整。同样，如果接收到错误数据类型的数据包，则PID错误位为RxCSR寄存器将被设置。然而，在每种情况下，仍将产生中断，以允许从FIFO读取已接收的数据。

报告PID错误或IncompRx的情况取决于接收到的数据包的精确序列。当核心在外围模式下运行时，详细信息如下。（当堆芯在

外围模式：见第8.4.2.3节。）

预计pkt数量	收到的数据pkt	回答	预计pkt数量	收到的数据pkt	回答
1.	数据0（“0”）	好啊	2.	D1 DM	PID错误设置
	数据1（’D1’）	PID错误设置		D1 NR	IncompRx集合
	数据2（’D2’）	PID错误设置		D2 D0	PID错误设置
	MDATA（’DM’）	PID错误设置		D2 D1	PID错误设置
	无响应（“NR”）	IncompRx集合*		D2 D2	PID错误设置
2.	D0	好啊		D2 DM	PID错误设置
	D1 D0	好啊		D2 NR	IncompRx设置+PID错误设置
	D1 D1	PID错误设置		DM	PID错误设置
	D1 D2	PID错误设置		NR	IncompRx集合*
预计pkt数量	**收到的数据pkt**	**回答**	**预计pkt数量**	**收到的数据pkt**	**回答**
3.	D0	好啊	3.	D2 D2 D1	PID错误设置
	D1 D0	好啊		D2 D2 D2	PID错误设置
	D1 D1	PID错误设置		D2 D2 DM	PID错误设置
	D1 D2	PID错误设置		D2 D2 NR	IncompRx集合+PID错误设置
	D1 DM	PID错误设置			
	D1 NR	IncompRx集合		D2 DM D0	PID错误设置
	D2 D0	PID错误设置		D2 DM D1	PID错误设置
	D2 D1 D0	好啊		D2 DM D2	PID错误设置
	D2 D1 D1	PID错误设置		D2 DM DM	PID错误设置
	D2 D1 D2	PID错误设置		D2 DM NR	IncompRx设置+PID错误设置
	D2 D1 DM	PID错误设置		D2 NR	IncompRx集合
	D2 D1 NR	IncompRx集合		DM	PID错误设置
	D2 D2 D0	PID错误设置		NR	IncompRx集合*

*在这些情况下，仍将生成中断，但不会设置RxPktRdy，因为FIFO中没有放置任何数据。

8.5.3. OUTRANSACTION HANDLING AS A主机

当MUSBHDRC作为主机操作时，OUT事务的处理方式与当MUSBMDRC作为外设操作时in事务的处理方法类似。

当每个数据包被加载到TxFIFO中时，需要设置TxCSR寄存器中的TxPktRdy位。同样，当最大大小的数据包已加载到FIFO中时，在TxCSR中设置AutoSet位（如适用）将导致TxPktRdy自动设置。此外，AutoSet可以与DMA控制器一起使用，在没有CPU干预的情况下执行完整的批量传输。

如果目标函数用NAK响应OUT令牌，则MUSBHDRC将继续重试事务，直到达到已设置的任何NAK限制。但是，如果目标函数以STALL响应，则MUSBHDRC不会重试事务，而是会使用TxCSR寄存器组中的RxStall位中断CPU。如果目标函数在所需时间内没有响应OUT令牌（或者数据包中存在CRC或比特填充错误），则MUSBHDRC将重试该事务。如果三次尝试后目标功能仍未响应，则MUSBHDRC将刷新FIFO

并用TxCSR集合中的错误位中断CPU。

8.5 . 4.TR A N S A C T ION SC HE-DU LI N G公司

当作为主机操作时，MUSBHDRC维护一个帧/微帧计数器。如果目标功能是全速或高速设备，则MUSBHDRC将在每个帧/微帧开始时自动发送SOF/uSOF数据包。如果目标功能是低速设备，“K”状态将在总线上传输，作为“保持活动”，以阻止低速设备进入挂起模式。

在SOF/uSOF数据包被发送之后，MUSBHDRC将循环通过所有配置的端点来寻找活动事务。活动事务被定义为设置了ReqPkt比特的Rx端点，或者设置了TxPktRdy比特和/或FIFONotEmpty比特的Tx端点。

只有在帧/微帧的第一个事务调度程序周期上发现活动的等时或中断事务，并且该端点的间隔计数器已倒计时为零时，才会启动该事务。这确保了每个端点每*n*帧/微帧（如果选择高带宽支持，则最多三个）只发生一次中断/等时事务，其中*n*是通过TxInterval/RxInterval寄存器为该端点设置的间隔-请参阅第3.3.15节和第3.3.17节。

如果在下一个SOF/uSOF数据包到期之前，帧/微帧中有足够的时间来完成事务，则活动的批量事务将立即启动。如果事务需要重试（例如，因为接收到NAK或目标函数没有响应），则在事务调度器首先检查了所有其他端点的活动事务之前，不会重试事务。这确保了发送大量NAK的端点不会阻塞总线上的其他事务。核心还允许用户指定在端点超时之前从特定目标接收NAK的时间长度限制（见第3.3.4、3.3.15和3.3.17节）。

8.5. 5. B A BBLE

MUSBHDRC将不会启动事务，直到总线至少在最小的包间延迟内处于非活动状态。它也不会启动事务，除非它能在帧结束前完成。如果总线在一帧结束时仍处于活动状态，则MUSBHDRC将假定其连接的功能出现故障，并将暂停所有事务并生成一个间歇中断。

8.5. 6. 何模式苏消费

如果设置了Power寄存器中的SuspendMode位，则MUSBHDRC将完成当前事务，然后停止事务调度器和帧计数器。不会启动进一步的事务，也不会生成SOF数据包。

要退出挂起模式，CPU应设置恢复位并清除电源寄存器中的挂起位。当Resume位为高时，MUSBHDRC将在总线上生成Resume信号。20ms后，CPU应清除Resume位，此时帧计数器和事务调度程序将启动。

在挂起模式下，如果启用，SUSPENDM输出也将变低：这可能用于关闭USB驱动程序。此外，POWERDWN输出被断言：该信号可以用于停止CLK，从而在这种状态下节省电力。
但是，如果要支持远程唤醒，则必须保持PHY的电源，以便MUSBHDRC能够检测到总线上的Resume信令。

9. 美国

9.1. 在PE RIP H E R A L M O DE

当MUSBHDRC用作外设，并且在USB上检测到重置条件时，设备将执行以下操作：

- 将FAddr设置为0。
- 将“索引”设置为0。
- 刷新所有端点FIFO。
- 清除所有控制/状态寄存器。
- 启用所有端点中断。
- 生成重置中断。

如果功率寄存器（D5）中的HS Enab位已设置，则MUSBHDRC也会尝试协商高速操作。是否选择高速操作由HS模式位（Power.D4）指示。

当驱动MUSBHDRC的应用程序软件接收到重置中断时，它应该关闭任何打开的管道，并等待总线枚举开始。

9.2. 在HO-ST模式中

如果在MUSBHDRC处于主机模式时设置了电源寄存器中的Reset位，则MUSBHDRR将在总线上生成Reset信号。如果功率寄存器（D5）中的HS Enab位已设置，它也将尝试协商高速操作。

CPU应保持重置位设置至少20ms，以确保目标设备的正确重置。

CPU清除该位后，MUSBHDRC将启动其帧计数器和事务调度程序。是否选择高速操作将由HS模式位（Power.D4）指示。

10. 暂停/恢复

MUSBHDRC如何进入和退出挂起模式取决于它当前是作为主机还是作为外围设备运行。

10.1. 当MUSB MHDRC在I N G A S A PERI P HER A L运作时

(i) 进入挂起模式。当作为外设操作时，MUSBHDRC会监测USB上的活动，当3毫秒内没有活动发生时，它会进入挂起模式。如果已启用挂起中断，此时将生成一个中断。SUSPENDM输出也将变低（如果启用）。

此时，POWERDWN信号也被断言，以指示应用程序可以通过停止CLK来节省功率。POWERDWN然后保持断言，直到从总线上断开电源（指示设备已断开连接）或在总线上检测到恢复信号或重置信号。

(ii) 当总线上出现恢复信号时，如有必要，必须重新启动第一个CLK。然后，MUSBHDRC将自动退出挂起模式。如果Resume（恢复）中断已启用，将生成一个中断。

(iii) 启动远程唤醒。如果软件希望在MUSBHDRC处于挂起模式时启动远程唤醒，则应写入电源寄存器，将恢复位（D2）设置为“1”。（注意：如果CLK已停止，则需要重新启动它才能进行此写入。）软件应保持此位设置约10毫秒（最小2毫秒，最大2毫秒）

15毫秒），然后将其重置为0。此时，集线器应该已经接管了USB上的恢复信号的驱动。

注意：当软件启动远程唤醒时，不会产生恢复中断。

10.2. 当音乐H DRC在I N G A S A主机上运行时

(i) 进入挂起模式。当作为主机操作时，可以通过设置电源寄存器中的SuspendMode位，提示MUSBHDRC进入Suspend模式。当设置此位时，MUSBHDRC将完成当前事务，然后停止事务调度器和帧计数器。不会启动进一步的事务，也不会生成SOF数据包。

如果设置了Enable SuspendM位（Power.D0），当MUSBHDRC进入挂起模式并停止XCLK时，UTMI+PHY将进入低功率模式。

(ii) 发送恢复信号。当应用程序要求MUSBHDRC退出挂起模式时，它需要清除电源寄存器中的挂起位，设置恢复位，并保持设置20ms。当Resume位为高时，MUSBHDRC将在总线上生成Resume信号。20ms后，CPU应清除Resume位，此时帧计数器和事务调度程序将启动。

(iii) 响应远程唤醒。如果在MUSBHDRC处于挂起模式时从目标检测到Resume信令，*则*UTMI+PHY将退出低功率模式并重新启动XCLK。然后，MUSBHDRC将退出挂起模式，并自动将功率寄存器（D2）中的恢复位设置为"1"，以接管从目标生成恢复信号。如果Resume（恢复）中断已启用，将生成一个中断。

11. 支持多种设备

仅当MUSBHDRC在配置GUI中启用了多点选项时，本节才适用。当在主机模式下操作时，MUSBHDRC具有作为一系列USB外围设备（高速、全速或低速）的主机的功能，这些设备通过USB集线器连接到MUBMHDRC。

核心支持多个设备的关键特征是其允许将目标设备的功能单独分配给MUSBHDRC核心中实现的不同Rx和Tx端点的功能。此外，可以动态地进行这种分配，从而允许以不同的组合使用来自目标外围设备列表的设备。然而，可以一起使用的外围设备的组合受到在核心中实现的Tx和Rx端点的数量的限制。只能在需要的端点仍然可用的情况下添加其他设备。

11.1. 一个在I NG DEVI CES T O E N dpoints的L L O C

连接设备的单独功能通过一组三个寄存器分配给MUSBHDRC核心内的端点，这些寄存器与核心中实现的每个Rx或Tx端点（包括端点0）相关联。

相关寄存器为Tx/RxFuncAddr、Tx/RxHubAddr和Tx/RxHubPort。（这些寄存器的位置取决于正在寻址的MUSBHDRC端点。）

需要记录在Tx/RxFuncAddr寄存器中的信息是要通过所选端点访问的目标函数的地址。对于所使用的每个Tx和Rx端点，需要分别记录该信息。特别是，需要为端点0设置TxFuncAddr和RxFuncAddr。

Tx/RxHubAddr和Tx/RxHubPort寄存器用于全速或低速设备通过高速USB 2.0集线器连接到MUSBHDRC的情况，该集线器在高速传输和低速/全速传输之间执行所需的事务转换。在这种情况下，Tx/RxHubAddr和Tx/RxHubPort寄存器需要记录执行事务转换的集线器的地址以及相关联的Tx/Rx端点需要访问设备的集线器的端口。*注意：*如果端点0连接到集线器，则的Tx和Rx版本

需要为此端点设置这些寄存器。

Tx/RxHubAddr寄存器还用于记录集线器提供多个事务转换器还是仅提供单个事务转换器。这对可以实现的总体带宽具有显著影响。

这些寄存器的详细信息见第3.5.2节和第3.5.3节。

除了通过这三个寄存器记录目标功能的地址外，还需要记录目标设备的端点编号和操作速度以及将执行的事务类型。

对于Tx端点，当索引寄存器被设置为选择所需端点时，需要在位于1Ah的TxType寄存器中设置此信息。对于Rx端点，当索引寄存器设置为选择所需端点时，需要在位于1Ch的RxType寄存器中设置此信息。在这两种情况下，端点编号都记录在位D3–D0中，事务类型通过位D5–D4选择，操作速度通过位D7–D6选择。

在端点0的情况下，只需要设置速度（此端点仅具有处理控制事务的功能，因此始终与设备端点0相关联）。此速度设置是通过0型寄存器的位D7–D6进行的，当索引寄存器设置为0时，该寄存器位于1Ah。

Type0、TxType和RxType寄存器的详细信息分别在第3.3.4、3.3.14和3.3.16节中给出。

11.2.OPERATION

一旦向端点分配了功能，并记录了目标设备的操作速度，如第11.1节所述，多点设置中的大多数操作与核心连接到单个其他设备的等效操作没有什么不同。详细信息见本指南其他部分。

但是，还需要其他步骤：

其中采取了将功能分配动态切换到端点的选项（例如，允许支持更广泛的设备）

其中通过不同的端点来处理通常与端点0相关联的控制分组。

如果使用动态分配，则用户必须跟踪与端点以及分配给该端点的每个设备相关联的当前数据切换状态。当在一个设备和另一个设备之间进行切换时，该状态的知识对于允许*用户*选择正确的数据切换状态是必要的。（此操作由用户负责，因为核心无法确定在切换使用和停止使用功能时预期的数据切换状态。）

当核心处于主机模式时，可以通过向适当的TxCSRL/RxCSRL寄存器写入来设置这些寄存器中包括的数据切换写入启用和数据切换位，从而将数据切换状态从其当前状态切换。（见第3.3.9节和第3.3.11节）

（*注意*：当核心在主机模式下操作时，CSR0中还包括数据切换写入启用和数据切换位。但是，通过核心的端点0执行的控制操作通常应始终使数据切换处于预期状态。）

在通过端点0以外的端点处理控制数据包的情况下，用户还必须提示发送每个安装令牌。这包括设置核心在主机模式下操作时包含在TxCSR中的SetupPkt位以及TkPktRdy位。如果未设置SetupPkt比特，将发送OUT令牌。

总体而言，建议使用核心的端点0来处理连接到核心的所有设备的控制数据包，并酌情切换此端点的分配。然后处理发送正确令牌的问题，以及确保为此端点正确设置数据切换的问题。

如上所述，可以为此函数使用不同的端点，但需要注意的是：

(i) 控制功能必须分配给Rx/Tx端点对（即具有相同的端点编号）。

(ii) 所选择的每个端点都必须与FIFO相关联，这些FIFO能够以所选择的操作速度容纳与EP0事务相关联的数据包大小（即，低速或全速事务至少8个字节，而高速事务至少64个字节

11.3宽度为

多点系统处理等时事务（特别是）的能力由可用带宽决定。

一旦设置了端点，所有调度都在硬件中处理。但是，与基于PC的EHCI/OHCI/UHCI主机一样，在打开周期性管道（供同步或中断流量使用）之前，*软件*必须确定有足够的可用带宽。此外，如果通过高速集线器向全速设备打开周期性管道，则软件必须确认在本地高速总线和由集线器中的事务转换器生成的全速总线上都有足够的带宽可用。

不同事务所需的带宽可以使用与基于PC的主机所使用的算法类似的算法来确定（详见USB 2.0规范第5.11.3节）。

正如预期的那样，在所使用的集线器支持多个事务转换器的情况下，可用带宽将更大。

12. 连接CT/D我的连接CT

与连接和断开MUSBHDRC相关的特定行为涉及其在对等通信中的主机模式或外围模式中的使用。

12.1. 在HO-ST模式中

当MUSBHDRC在主机模式下操作时，CPU通过设置会话位（DevCtl.D0）来启动会话。功率是
然后施加到VBus，并且核心等待要连接的设备。

当检测到设备时，会产生连接中断（即IntrUSB.D4变高）。已经连接的设备的速度可以通过读取DevCtl寄存器来确定，其中FSDev位（D6）对于高速/全速设备将是高的，而LSDev比特（D5）对于低速设备将是低的。然后CPU应重置设备。如果设置了FSDev和HS Enab（电源.D5），则MUSBHDRC将尝试协商高速操作。这是否成功将由HS模式位（Power.D4）指示。

CPU应保持重置位设置20ms，以确保目标被重置。然后它可以开始设备枚举。

如果在会话进行期间设备断开连接，则会产生断开连接中断（即IntrUSB.D5变高）。

12.2. 在外围A L模式中

当MUSBHDRC在外围模式下运行时，当设备连接到主机时，不会产生中断。但是，当主机终止会话时，会生成断开连接中断（IntrUSB.D5）。

13. 程序设计方案

本节和以下部分将介绍控制MUSBHDRC核心的设备需要执行的操作，以及影响这一操作的核心操作方面。

在整个讨论中，假设控制设备是运行一些固件的微控制器，但它可以是定制的硬接线逻辑块。

1.1. 软连接/DISCONNECT

如果需要，MUSBHDRC将允许通过软件控制其与USB总线的连接。

当选择软连接/断开连接时，当MUSBHDRC在外围模式下操作时，通过设置/清除电源寄存器的位6（被识别为软连接位），可以在正常模式和非驱动模式之间切换与MUSBHDrc一起使用的符合UTMI+的PHY。当该软连接位设置为1时，PHY被置于其正常模式，并且USB总线的D+/D-线被启用。同时，MUSBHDRC处于“通电”状态，在该状态下，除USB重置外，它不会响应任何USB信号。

当该功能启用且软连接位为零时，PHY将进入非驱动模式，D+和D-为三态，MUSBHDRC在USB总线上的其他设备上显示，就好像它已经断开一样。

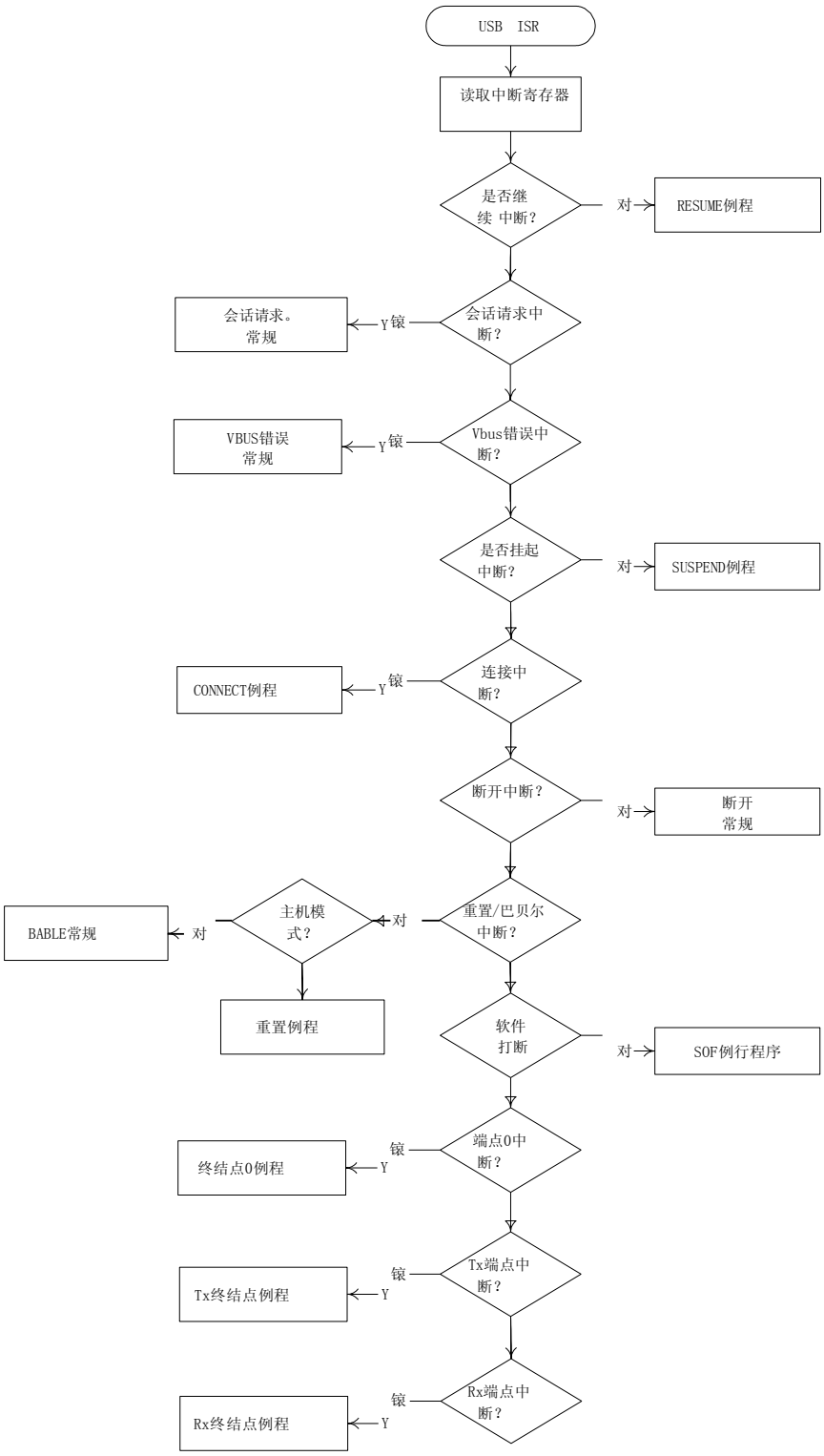
硬件重置（NRST=0）后，软连接被清除为0。因此，MUSBHDRC将显示断开连接，直到软件将软连接设置为1。然后，应用软件可以选择何时将PHY设置为其正常模式。初始化过程较长的系统可能会使用此方法来确保初始化完成，并且系统已准备好在连接到USB之前执行枚举。

一旦软连接位设置为1，软件也可以通过将该位清除为0来模拟断开连接。

1 3.2 我们的业务范围

当CPU被USB中断中断时，它需要读取中断状态寄存器，以确定是哪个端点导致了中断，并跳到适当的例程。如果多个端点导致了中断，则应首先为端点0提供服务，然后为其他端点提供服务。

以下流程图中给出了USB中断服务例程的流程图。



USB中断服务例程

14．其他问题

为了节省电源，*USB On-the-Go*补充功能允许VBus仅在需要时通电，并在总线未使用时关闭。MUSBHDRC通过允许在没有会话进行时停止CLK，进一步促进了额外的节能。当断言POWERDWN时，CLK可能会停止。

VBus始终由总线上的“A”设备提供。MUSBHDRC通过对来自UTMI+PHY的IDDIG输入进行采样来确定它是“A”设备还是“B”设备。，该信号被拉低，但当感测“B型”插头（表示MUsbHDRC是”B“设备）时，该信号变高。

14.1. START A SESSION

当包含MUSBHDRC的设备需要启动会话时，CPU需要在DevCtl寄存器中设置会话位（D0）。然后，MUSBHDRC将启用ID引脚感应。这导致IDDIG输入在检测到A型连接的情况下变低，或者在检测到B型连接的情形下变高。DevCtl寄存器（D7）中的B设备位也被设置为指示MUSBHDRC是采用了“A”设备还是“B”设备的角色。

如果MUSBHDRC是“A”设备：则MUSBHDSC将进入主机模式（“A”始终是默认主机），打开VBus并等待VBus超过VBus有效阈值，如VBUSVALID输入变高所示。（此事件还导致DevCtl寄存器（D4–D3）中的Vbus[1:0]位转到11b。）

然后，MUSBHDRC将等待连接外围设备。当检测到外围设备时，将生成连接中断（IntrUSB.D4）（如果启用），并且将根据检测到的是高速/全速外围设备还是低速外围设备来设置DevCtl寄存器中的FSDev或LSDev位（分别为D6/D5）。

然后CPU应该重置这个外设。如果设置了FSDev和HSEnab（Power.D5），则MUSBHDRC将在重置期间监测LINESTATE，以查看是否从外围设备接收到高速啁啾。如果接收到啁啾，MUSBHDRC将以高速啁啾响应并进入高速模式。

要结束会话，CPU应该清除会话位（DevCtl.D0）。如果检测到牙牙学语，MUSBHDRC也将自动结束会话。

如果MUSBHDRC是“B”设备：MUSBHDrc将使用*USB On the Go*补充中定义的会话请求协议请求会话，即它将首先断言DISCHRGVBUS以释放VBus。然后，当VBus低于会话结束阈值时（如SESSEND输入变高和DevCtl寄存器（D4–D3）中的VBus[1:0]位变为00b所示），并且线路状态为SE0超过2毫秒时，MUSBHDRC将首先对数据线进行脉冲，然后对VBus进行脉冲（通过CHRGVBUS变高）。

会话结束时，会话位被清除——通常由MUSBHDRC清除，但如果应用软件希望执行软件断开连接，也可以由CPU清除（见第3.2.12节中DevCtl的描述）。然后，MUSBHDRC将切换TERMSL，这将导致PHY切换出D+上的上拉电阻器。这会向“A”设备发出结束会话的信号。

14.2. 检测活动

当OTG设置的另一个设备希望启动会话时，如果它是“a”设备（如AVALID输入变高和DevCtl寄存器（D4–D3）中的VBus[1:0]位变为10b所示），它将使VBus升高到会话有效阈值以上，或者，如果它为“B”设备，它将首先对数据线进行脉冲，然后对VBus进行脉冲。根据这些动作中的哪一个，MUSBHDRC可以确定它是当前设置中的“A”设备还是“B”设备，并相应地采取如下行动：

如果VBus升高到会话有效阈值以上：则MUSBHDRC为“B”设备。MUSBHDRC将设置

DevCtl寄存器中的会话位（D0）。当在总线上检测到重置信号时，将生成重置中断（IntrUSB.D2）（如果启用），CPU应将其解释为会话的开始。

此时，MUSBHDRC将处于外设模式，因为"B"设备是默认的外设。

会话结束时，"A"设备将关闭VBus的电源。当VBus下降到会话有效阈值以下时（如AVALID输入变低和DevCtl寄存器（D4–D3）中的VBus[1:0]位变为01b所示），MUSBHDRC将检测到这一点并清除会话位以指示会话已结束。还将生成断开连接中断（IntrUSB.D5）（如果启用）

如果检测到数据线/VBus脉冲：则MUSBHDRC为"A"设备。它将生成会话请求中断（IntrUSB.D6–如果启用），以指示"B"设备正在请求会话。然后CPU应该通过设置会话位（DevCtl.D0）来启动会话。

15. 主机协商

当MUSBHDRC是"A"设备（IDDIG低，B设备（DevCtl.D7）=0）时，会话开始时，它将自动进入主机模式。

当MUSBHDRC是"B"设备（IDDIG高，B设备（DevCtl.D7）=1）时，当会话开始时，它将自动进入外围模式。然而，CPU可以通过在DevCtl寄存器（D1）中设置Host Req位来请求MUSBHDRC成为Host。该位可以在通过设置会话位（DevCtl.D0）请求会话启动的同时设置，也可以在会话启动后的任何时间设置。当MUSBHDRC下一次进入挂起模式（总线上3毫秒无活动）时，假设Host Req位保持设置，它将进入主机模式并通过切换TERMSEL开始主机协商（如*USB on the Go*补充中所规定），导致PHY断开D+线上的上拉电阻器。这将导致"A"设备切换到外围模式并连接其自身的上拉电阻器。当MUSBHDRC检测到这一点时，如果启用，它将生成连接中断（IntrUSB.D4）。它还将设置电源寄存器（D3）中的重置位，以开始重置"A"设备。（MUSBHDRC自动开始此重置序列，以确保在连接其上拉电阻器的"A"设备的1ms内按要求开始重置）。CPU应至少等待20毫秒，然后清除重置位并枚举"A"设备。

当基于MUSBHDRC的"B"设备完成使用总线时，CPU应通过设置电源寄存器（D1）中的挂起模式位将其置于挂起模式。"A"设备应该检测到这一点，并终止会话或恢复到主机模式。如果"A"设备是基于MUSBHDRC的，如果启用该功能，它将生成断开连接中断（IntrUSB.D5）。

16. 基本DMA支持

MUSBHDRC支持通过内置DMA控制器（如果实现）或外部DMA控制器对Tx端点1-15和Rx端点1-15的FIFO进行DMA访问。

支持内置DMA控制器或外部DMA控制器的基础是用于每个Tx端点和每个Rx端点的单独DMA请求行。如果总共定义了*N*个Tx端点和*M*个Rx端点（除了端点0），DMA_REQ[0]。。。DMA_REQ[*N*–1]与Tx端点1相关联。。。*N*DMA_REQ[*N*]。。。DMA_REQ[*N*+*M*–1]与Rx端点1相关联。。。*M*.这些请求行在使用内置DMA控制器的情况下在内部处理，但也可在核心的顶层使用，以允许外部DMA控制器使用它们。

DMA请求线通过适当的TxCSR控制寄存器（D12）或RxCSR控制寄存器（D23）中的DMAReqEnab位单独启用，并在两种模式下操作，称为DMA请求模式0和DMA请求模式1。操作模式的选择是通过DMAReqMode位（TxCSR.D10用于Tx端点；RxCSR.D11用于Rx端点）进行的。在使用外部DMA控制器和使用内置DMA控制器的情况下，都需要选择所需的请求模式。

对于在请求模式0下操作的Rx端点，当端点中有数据包可用时，DMA请求行变高

FIFO，并且通常在从最后一个字节开始处理的第8个字节的周期结束时变低（在包含该字节的传输之前发生两次传输减去一个CLK周期）。然而，如果正在使用外部DMA控制器，并且未采用早期DMA断言选项（见第3.2.13节），则请求行将在识别传输结束的周期结束时变低（发生在倒数第二次传输后的一个CLK周期）。如果CPU清除RxPtRdy位（RxCSRL.D0），则请求线也将变低。

在请求模式1中用于Rx端点的DMA请求线的行为是类似的，除了当接收到的分组具有最大分组大小（如在RxMaxP寄存器中设置的）时请求线将仅变高。如果接收到的数据包具有某种其他大小，则DMA请求行将保持低位。但是，请注意，如果请求模式从请求模式1切换到请求模式0，则如果FIFO中有数据包，则会断言请求行，以便下载此“预接收”数据包。

对于在请求模式0或请求模式1中操作的Tx端点，当端点FIFO能够接受数据包时，DMA请求线将变高。在最后一个TxMaxP字节的第8个字节加载到FIFO后，它通常会在一个CLK周期内变低，但如果正在使用外部DMA控制器，并且没有采用早期DMA断言选项（见第3.2.13节），则在所有TxMaxP字节加载后，请求行将在一个CL周期内变为低。如果CPU设置TxPktRdy位（TxCSRL.D0），则请求线也将变低。

注意：在主机模式下操作时，如果RxStall位（TxCSRL.D5）或Error位（TxCSRL.D2）在三次尝试传输数据包失败后被设置，则DMA请求行将被禁用，直到RxStall/Error位被清除。

所选模式也会影响端点中断的生成（如果启用）。在DMA请求模式0中，当接收到数据包时不会产生中断，但会产生适当的端点中断以提示加载所有数据包。在DMA请求模式1中，端点中断被抑制，除非在接收到短数据包（即小于RxMaxP字节之一）之后。

下表总结了生成Tx和Rx端点中断的条件。

EPI与正在设置的RxKTRdy相关联的中断

DMA ReqEnab	*DMAReqMode*	*是否生成中断?*
0	十、	对
1.	0	不
1.	1.	仅当短数据包

清除与TxPktRdy相关的EP中断

DMA ReqEnab	*DMAReqMode*	*是否生成中断?*
0	十、	对
1.	0	对
1.	1.	不

DMA请求模式0同样适用于批量、中断或同步传输。事实上，如果端点配置为等时传输，则在使用DMA的情况下，应始终选择DMA请求模式0。

DMA请求模式1在大数据块被传输到批量端点的情况下主要是有价值的。USB协议要求将这样的分组划分为端点的最大分组大小的一系列分组（高速为512字节，全速为64字节）。请注意，Tx/RxMaxP必须设置为偶数字节才能正确生成中断。DMA请求模式1可以与适当编程的DMA控制器一起使用，以避免在每个单独的数据包之后必须中断处理器的开销：相反，处理器只有在传输完成后才被中断。在某些情况下，传输的数据块将包括预定义数量的这些数据包，控制软件在传输过程中对这些数据包进行“计数”。在其他情况下，序列中的最后一个数据包可能小于最大数据包大小，接收器可以使用这个“短”数据包来发出传输结束的信号。（如果传输的总大小是最大数据包大小的精确倍数，则传输软件应发送一个空数据包供接收器检测。）

注意：Tx/RxMaxP必须设置为偶数字节，才能在模式1中正确生成中断。有关使用DMA进行批量

传输的更多信息，请参见第22.3节。

根据需要，DMA传输可以是8位、16位或32位。但是，与一个数据包相关的所有传输（最后一个除外）必须具有相同的宽度，以便数据一致地以字节、字或双字对齐。最后一次传输可能包含比先前传输更少的字节，以便完成奇数字节或奇数字传输。

注意：在更改DMA请求模式之前，应禁用DMA请求。特别是，TxCSRH寄存器中的DMAReqMode位不应在相应DMAReqEnab位被清除到之前或在相同周期内设置为零

零

17. OPTIO N AL DM A controller

MUSBHDRC可以可选地包括多通道DMA控制器，最多可配置8个通道。

该DMA控制器支持两种DMA模式，称为DMA模式0和1，它可以处理高达8k的数据包大小（与第8.4.1.4节和第8.4.2.4节中描述的Tx批量数据包拆分、Rx批量数据包组合选项一起使用）。此外，控制器可以编程为使用INCR4、INCR8和INCR16 4/8/16节拍递增突发而不是未指定长度的突发进行传输。

在DMA模式0下操作时，DMA控制器只能编程为加载/卸载一个数据包，因此通过USB传输的每个数据包都需要处理器干预。此模式可用于任何端点，无论是使用控制、批量、等时事务还是中断事务（即包括端点0）。

当在DMA模式1中操作时，DMA控制器可以被编程为加载/卸载完整的批量传输（可以是许多数据包）。一旦设置好，DMA控制器将加载/卸载传输的所有数据包，仅当传输完成时才中断处理器。DMA模式1只能与使用批量事务的端点一起使用。

每个通道都可以针对选定的操作模式进行独立编程。

17.1. DM A REGI STERS

DMA控制器具有一个中断寄存器，用于指示哪些通道具有未决中断，以及用于每个配置通道的一组三个控制寄存器。DMA寄存器的完整描述见第3.8.5节"寄存器描述"一节。

住址	登记	描述
200小时	DMA_INTR	DMA中断寄存器。
204小时+（n-1）*10小时	DMA_CNTL	DMA控制寄存器。
208小时+（n-1）*10小时	DMA_ADDR	DMA地址寄存器。
20Ch+（n-1）*10h	DMA_COUNT	DMA计数寄存器。

*n =
通道编号1至8

1 7 .2. DM A B U S CY CLES

DMA控制器在AHB上使用递增突发。当它第一次被授予总线主控权时（无论是在USB数据包开始时，还是在数据包中途被丢弃后重新获得总线时），以及当AHB地址开始新的1K字节块时，它开始新的突发。*注：*在1K边界处启动新突发的要求由DMA控制器自动处理。用户在对DMA控制器进行编程时不需要考虑这些边界。

这些突发可以是4拍、8拍、16拍或具有未指定的长度，这取决于DMA信道的编程方式、正在传输的分组的大小以及相对于下一个1K边界的位置。CNTL寄存器的位D10–9选择突发模式0–3，该突发模式依次定义可使用的突发类型（见上文第17.1节）。例如，突发模式2的选择允许使用8拍（INCR8）、4拍（INCR4）突发和未指定长度但不是16拍（INCR16）突发的突发。

对于在DMA模式0或DMA模式1中传输所选择的突发模式没有限制。

分组的每次传输通常使用字传输（32位）来执行，但是在分组传输结束时可能存在额外的字节或半字传输。由于起始地址（写入DMA ADDR（*n*）寄存器）必须是字对齐的，因此数据包传输将从字传输开始，但可以在末尾添加半字和/或字节传输以处理任何残余。支持AHB拆分事务和重试。

17.3. 备份

如果DMA控制器在访问AHB上的内存时发生总线错误，DMA控制器将立即终止DMA传输，并用CNTL寄存器组的总线错误（D8）位中断处理器。

注意：此中断的生成不受DMA CNTL寄存器中中断启用位（位3）设置的影响。当CNTL。D3＝0。

17.4. TRANSFERRING PACKETS

使用内置DMA控制器访问MUSBHDRC FIFO需要对DMA控制器和MUSBHDFC端点进行适当编程。许多变化是可能的。以下部分详细介绍了用于传输单个数据包和多个数据包的基本操作的标准设置。

17.4.1. 在分区中：RX ENDPoint

单个数据包的传输通常将使用DMA模式0来执行。为此，MUSBHDRC Rx端点

应编程如下：

IntrRxE寄存器中的相关中断启用位设置为1。

相应RxCSR寄存器的DMAReqEnab位（D13）设置为0。（**注意：**此操作无需将MUSBHDRC设置为支持DMA。）

当MUSBHDRC接收到数据包时，它将生成适当的端点中断。然后，处理器应按如下方式对DMA控制器的选定通道进行编程：

ADDR：存储数据包的　　内存地址

COUNT：数据包的 大小（通过读取MUSBHDRC RxCount寄存器确定）

CNTL:DMA　使能（D0）=1；方向（D1）=0；DMA模式（D2）=0；中断使能（D3）=1；所需突发模式（D10–9）。

DMA控制器随后将请求总线主控权并将数据包传输到存储器。当它完成传输时，它将生成DMA中断（DMA_NINT取低）。然后，处理器应清除MUSBHDRC RxCSR寄存器中的RxPtRdy位。

17.4.2. INDIVIDUALPACKET：TXEndpoint

要使用DMA模式0执行此操作，MUSBHDRC Tx端点应编程如下：IntrTxE寄存器中的相关中断启用位设置为1。

适当的TxCSR寄存器的DMAReqEnab位（D12）设置为0。（**注意：**此操作无需将MUSBHDRC设置为支持DMA。）

当MUSBHDRC中的FIFO可用时，MUSBHDRR将使用适当的Tx Endpoint中断来中断处理器。然后，处理器应按以下方式对DMA控制器进行编程：

ADDR: 要 发送的 数据包的内存地

址COUNT : 要发送数据包 的大小

CNTL:DMA 使能（D0）=1；方向（D1）=1；DMA模式（D2）=0；中断使能（D3）=1；所需突发模式（D10–9）。

DMA控制器随后将请求总线主控权并将数据包传输到MUSBHDRC FIFO。当它完成传输时，它将生成一个DMA中断。然后，处理器应在MUSBHDRC TxCSR寄存器中设置TxPktRdy位。

17.4. 3μL IP LE P A C K E T S:RX EN DP OIN T

多个分组的传输通常将使用DMA模式1来执行。

在使用DMA模式1接收多个数据包的情况下，DMA控制器应编程如下：ADDR：存储传输 的 缓冲器的

内存地址

COUNT：数据缓冲区的 最大大小

CNTL:DMA 使能（D0）=1；方向（D1）=0；DMA模式（D2）=1；中断使能（D3）=1；所需突发模式（D10–9）。

并且MUSBHDRC Rx端点应当被编程如下：

IntrRxE寄存器中的相关中断启用位应设置为1。

相应RxCSR寄存器的自动清除（D15）、DMAReqEnab（D13）和DMAReqMode（D11）位应设置为1。在主机模式下，AutoReq（D14）位也应设置为1，RqPktCount寄存器应使用传输中的数据包数量进行编程。

当MUSBHDRC接收到每个数据包时，DMA控制器将请求总线主控权并将数据包传输到存储器。设置"自动清除"后，MUSBHDRC将自动清除RxBitRdy位。

在外围模式或RqPktCount为零的情况下，此过程将自动继续，直到MUSBHDRC接收到表示传输结束的"短数据包"（小于端点的最大数据包大小之一）。DMA控制器不会传输此"短数据包"：相反，MUSBHDRC将通过生成适当的Endpoint中断来中断处理器。然后，处理器可以读取MUSBHDRC RxCount寄存器，以查看"短数据包"的大小，并手动卸载它，或者在模式0下重新编程DMA控制器以卸载数据包。

在设置了AutoReq且RqPktCount为非零的主机模式下，核心将在每次请求后递减RqPktCount寄存器中的值。当该值从1递减到0时，AutoReq位被清除，以防止任何进一步的事务被尝试。

DMA控制器ADDR寄存器将随着数据包的卸载而递增，因此处理器可以通过将ADDR的当前值与存储器缓冲器的起始地址进行比较来确定传输的大小。

注意：如果传输的大小超过数据缓冲区的大小，DMA控制器将停止卸载FIFO，并通过DMA_NINT线中断处理器。

17.4. 4μL IP LE P A C K E T S:TX EN DP oint

要使用DMA模式1执行此操作，DMA控制器应编程如下：ADDR：要发送 的 数据块的内

存地址

COUNT：数据块的大小

CNTL:DMA 使能（D0）=1；方向（D1）=1；DMA模式（D2）=1；中断使能（D3）=1；所需突发模式（D10–9）。

并且MUSBHDRC Tx端点应当被编程如下：

IntrTxE寄存器中的相关中断启用位应设置为1（只是为了检测错误）。

适当TxCSR寄存器的AutoSet（D15）、DMAReqEnab（D12）和DMAReqMode（D10）位应设置为1。

当MUSBHDRC中的FIFO可用时，DMA控制器将请求总线主控权并将数据包传输到FIFO。设置AutoSet后，MUSBHDRC将自动设置TxPktRdy位。此过程将继续进行，直到整个数据块已传输到MUBMHDRC。DMA控制器随后将通过使DMA_NINT为低来中断处理器。如果要加载的最后一个数据包小于端点的最大数据包大小，则不会为此数据包设置TxPktRdy位：因此，处理器应通过设置TxPktRdy位以允许发送最后一个“短数据包”来响应DMA中断。如果要加载的最后一个数据包是最大数据包大小的，那么要采取的操作取决于传输是否在应用程序的控制下，例如Windows系统上的大容量存储软件，该软件会对发送的单个数据包进行计数。如果传输不在这样的控制之下，处理器仍然应该通过设置TxPktRdy位来响应DMA中断。这具有发送空包的效果，以便接收软件将其解释为指示传输结束。

18. VBUS事件

USB On The Go规范定义了一系列阈值，点对点通信中涉及的设备需要对这些阈值做出响应：

- VBus有效（要求大于4.4和4.75VV）
- 会话对“A”设备有效（要求在0.8V和2.1V之间）
- 会话结束（要求在0.2V和0.8V之间）

（特定设备中使用的实际阈值是通过MUSBHDRC核心外部的一系列比较器设置的，这些比较器根据VBus的电平将相应的VBUSVALID、AVALID和SESSEND输入设为高或低。）

这些阈值中的哪一个是关键的，以及控制MUSBHDRC的CPU需要响应的方式取决于设备是“A”设备还是“B”设备，以及事件发生的情况。所需的操作概述如下。

18. 1. 1. 1.A C T I ONS A S A N‘A’DEV I CE公司

VBus>VBus对由MUSBHDRC发起的会话有效（即VBus[1:0]（DevCtl.[D4:D3]）=11b，设置会话位（DevCtr.D0））。当VBus大于VBus Valid时，MUSBHDRC选择主机模式并等待连接设备。然后，它会生成一个连接中断（IntrUSB.D4）。CPU应重置并枚举连接的“B”设备。

VBus>Session由'B'设备启动的会话有效（即VBus[1:0]（DevCtl.[D4:D3]）=10b，会话位（DevCtr.D0）**清除）。**当VBus大于Session Valid时，MUSBHDRC将生成一个Session Request（会话请求）中断（IntrUSB.D6）。CPU应设置会话位。然后，MUSBHDRC将保持在主机模式或更改为外围模式，具体取决于“B”设备上拉电阻器的状态。所选择的模式将由主机模式位（DevCtl.D2）的状态指示。

低于VBus的VBus有效，同时会话位保持设置（即**VBus[1:0]**（DevCtl.[D4:D3]）◷ 11b、会话比特（DevCtl.D0）集合）。这表示VBus功率电平有问题。例如，电池功率可能下降得太低，无法维持VBus Valid。或者，“B”设备可能比“A”设备所能提供的电流更多。在任何一种情况下，MUSBHDRC都将自动终止会话并生成VBus错误中断（IntrUSB.D7）。

18.1. 1. 2. A C T I ONS A S A N‘B’DEV I CE公司

VBus>Session Valid（即VBus[1:0]（DevCtl.[D4:D3]）=10b，Session bit（DevCtr.D0）clear）。这表示来自“A”设备的活动。MUSBHDRC将设置会话位并使DPPULLDOWN输出为低电平，以便断开D+线上的下拉电阻器。

VBus＜会话有效，同时会话位保持设置（即VBus[1:0]（DevCtl.[D4:D3]）＝01b，**会话**位（DevCtr.D0）设**置）**。这表示"A"设备已断电（或断开连接）。MUSBHDRC将清除会话位（DevCtl.D0）并生成断开连接中断（IntrUSB.D5）。CPU应该结束会话。

VBus＜会话结束（即VBus[1:0]（DevCtl。[D4:D3]）=00b）。这是"B"设备可以发起会话请求的条件。如果设置了会话位（DevCtl.D0），那么在总线上SE0 2ms之后，MUSBHDRC将通过首先对数据线进行脉冲，然后对VBus进行脉冲（通过使CHRGVBUS变高）来启动SRP。

19. 活力四射

如果需要，可以将MUSBHDRC配置为具有128、256、512、1K...64K字节的单个总FIFO大小，当初始化MUSBHDrc时，可以将其区域分配给不同的端点。（见第3.3.18节）

注意：强烈建议您仅在MUSBHDRC用于在不同环境下需要不同FIFO大小的设备时使用此功能。如果FIFO大小不需要更改，最好通过标准配置选项设置这些大小，因为动态FIFO大小的选项会显著增加核心的大小，并需要更复杂的固件来处理。

FIFO空间到不同端点的分配要求每个Tx和Rx端点的规范为：

- RAM块内FIFO的起始地址
- 要支持的数据包的最大大小
- 是否需要双重缓冲

（最后两个共同定义了需要分配给FIFO的空间量。）

这些细节可以通过以下四个寄存器指定，当选择动态FIFO大小选项时，这些寄存器被添加到MUSBHDRC寄存器映射的索引区域：

MUSBMHDRC寄存器映射		
地址	**名称**	**说明**
62	TxFIFOsz	Tx端点FIFO大小
63	RxFIFOsz	Rx端点FIFO大小
64, 65	TxFIFO添加	Tx端点FIFO地址
66, 67	RxFIFOadd	Rx端点FIFO地址

寄存器描述第3.10节提供了这些寄存器的详细信息。

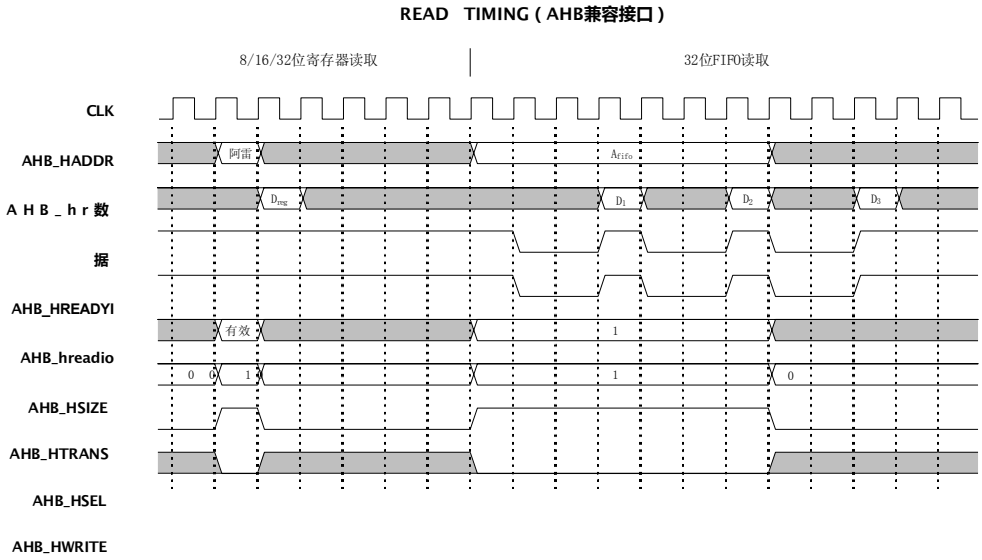
注：（i）动态设置FIFO大小的选项仅适用于端点1...15。端点0 FIFO具有固定大小（64字节）和固定位置（起始地址0）。
（ii）固件（和系统设计者）有责任确保在当前USB配置中活动的所有Tx和Rx端点具有专门分配给它们的RAM块，该RAM块至少与为端点设置的最大数据包大小一样大。

机密的

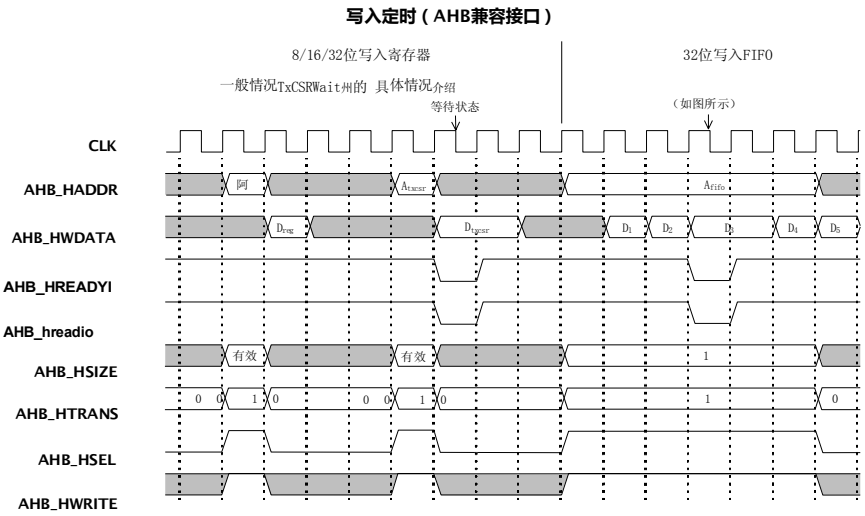
20. 定时波F O RMS

以下时序图用于指导MUSBHDRC的输入何时必须有效，以及输出何时有效。实际设置和延迟时间也将取决于MUSBHDRC所使用的技术和布局。**注意：**与所使用的任何替代网桥相关的时间可以在**musbmhdrc/docs**目录中的相关网桥规范中找到。

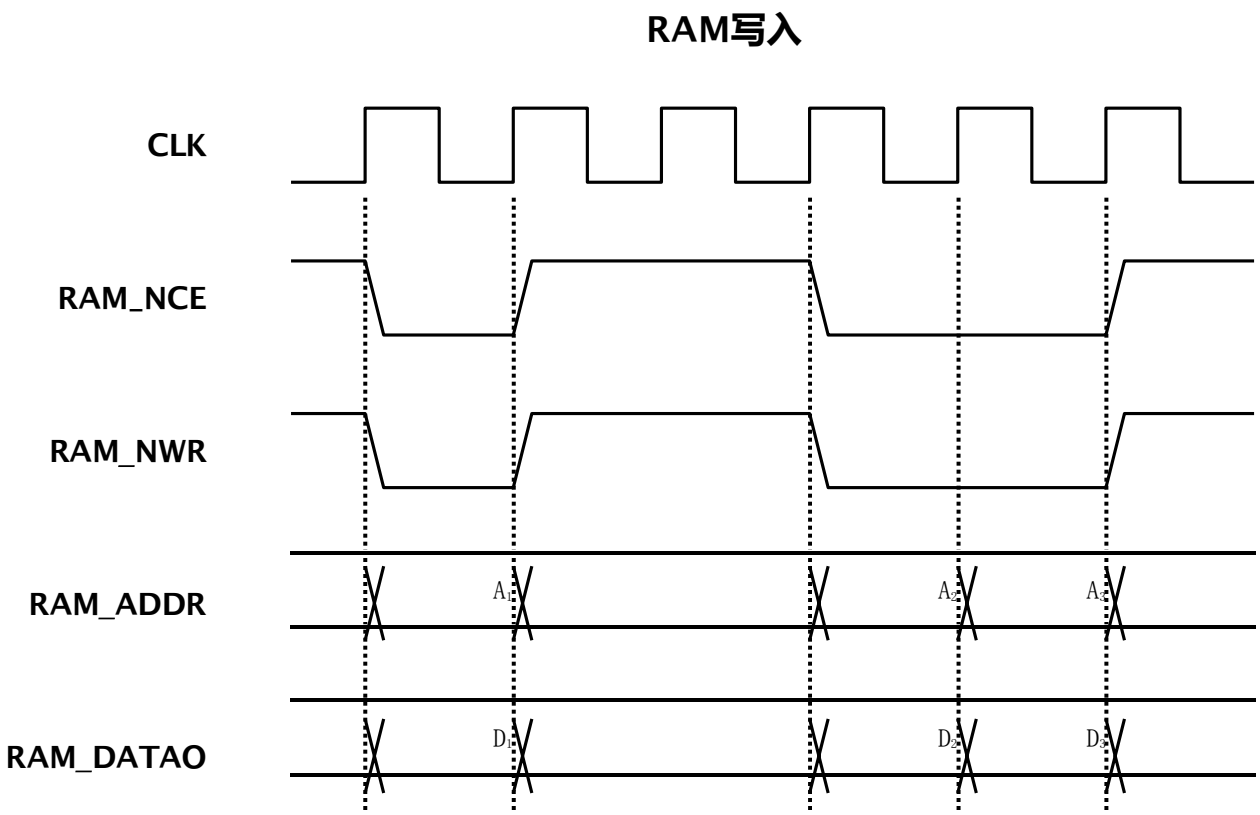
20.1. CP U R E A D



20.2. CP U令状

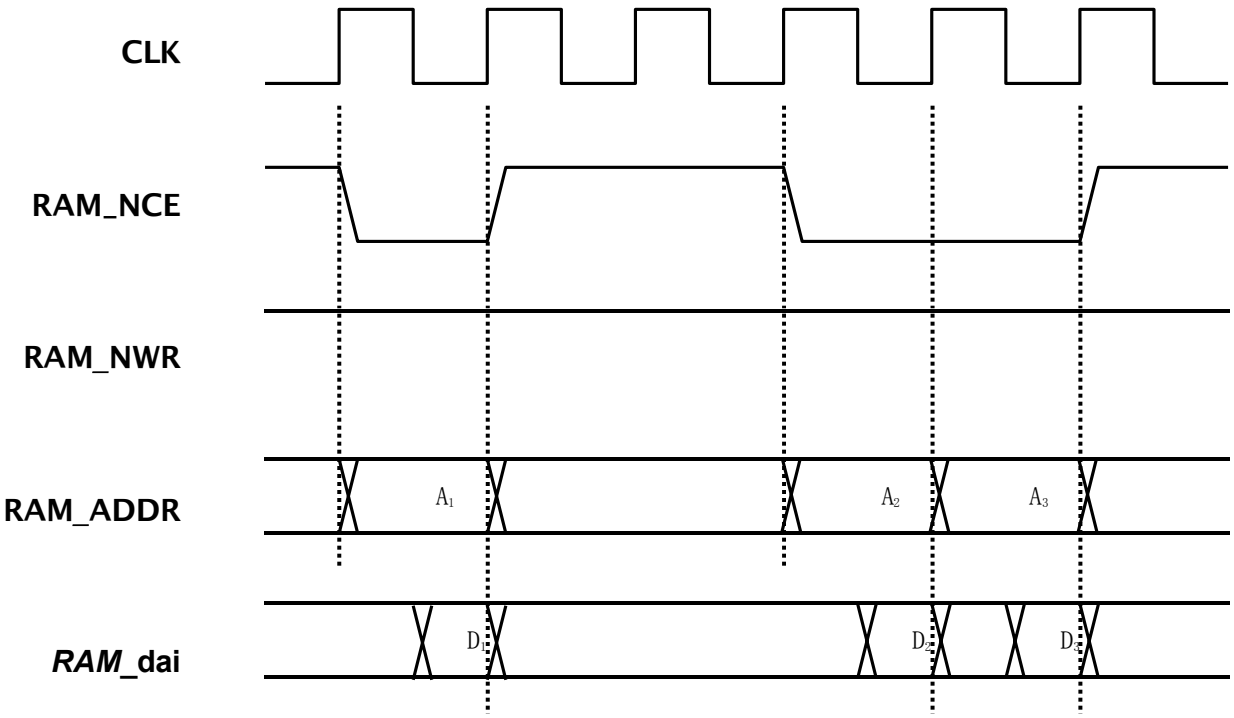


20.3.RAM WRITE



20.4. RAM READ

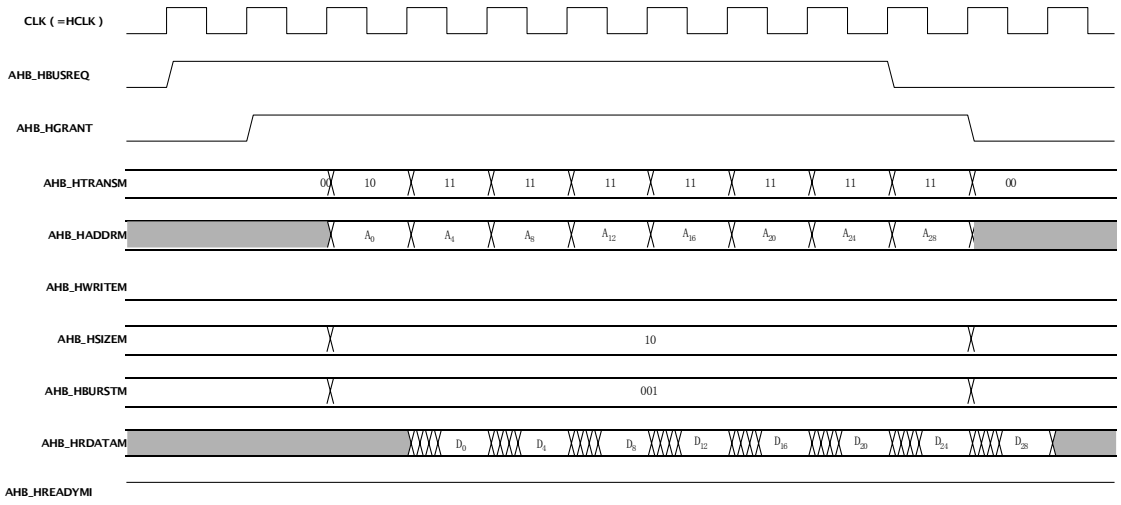
RAM读取



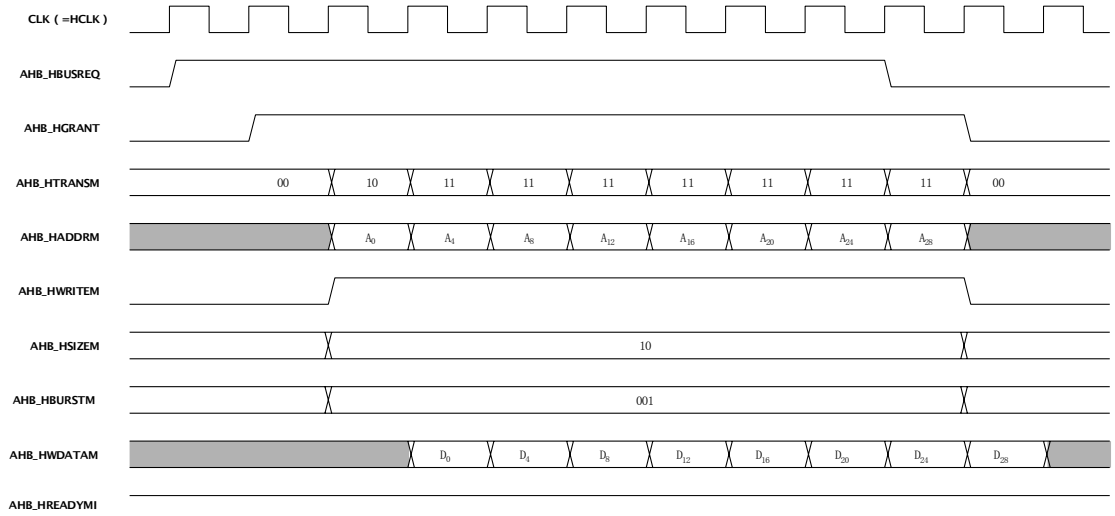
20.5. DMA时间

20.5.1. BUILT-在DMA控制器中

DMA加载到Tx端点（32字节）



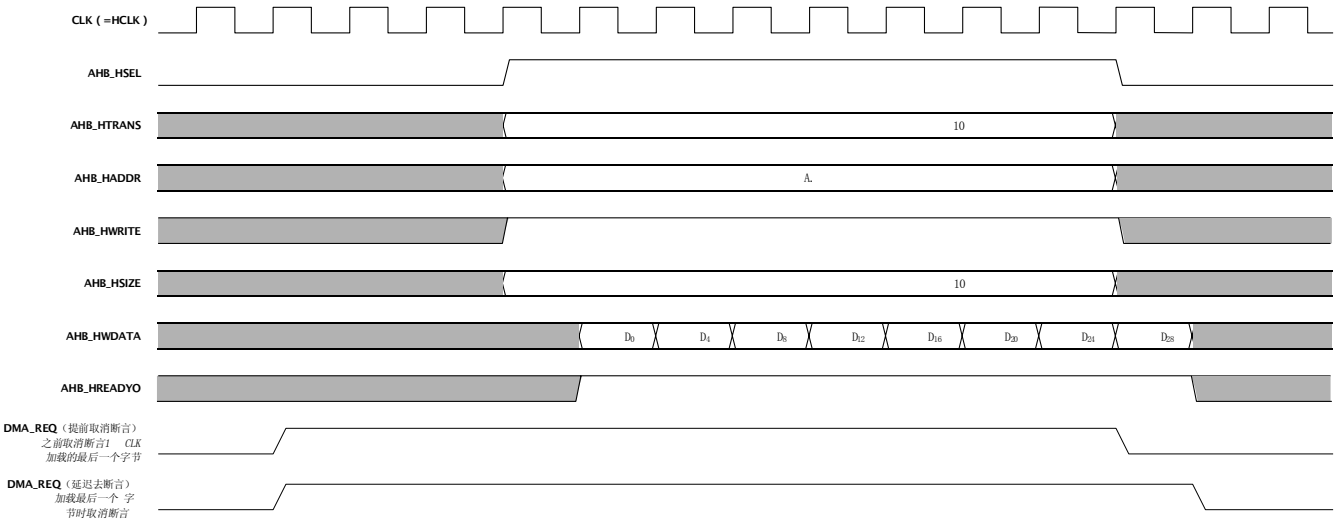
DMA从Rx端点卸载（32字节）



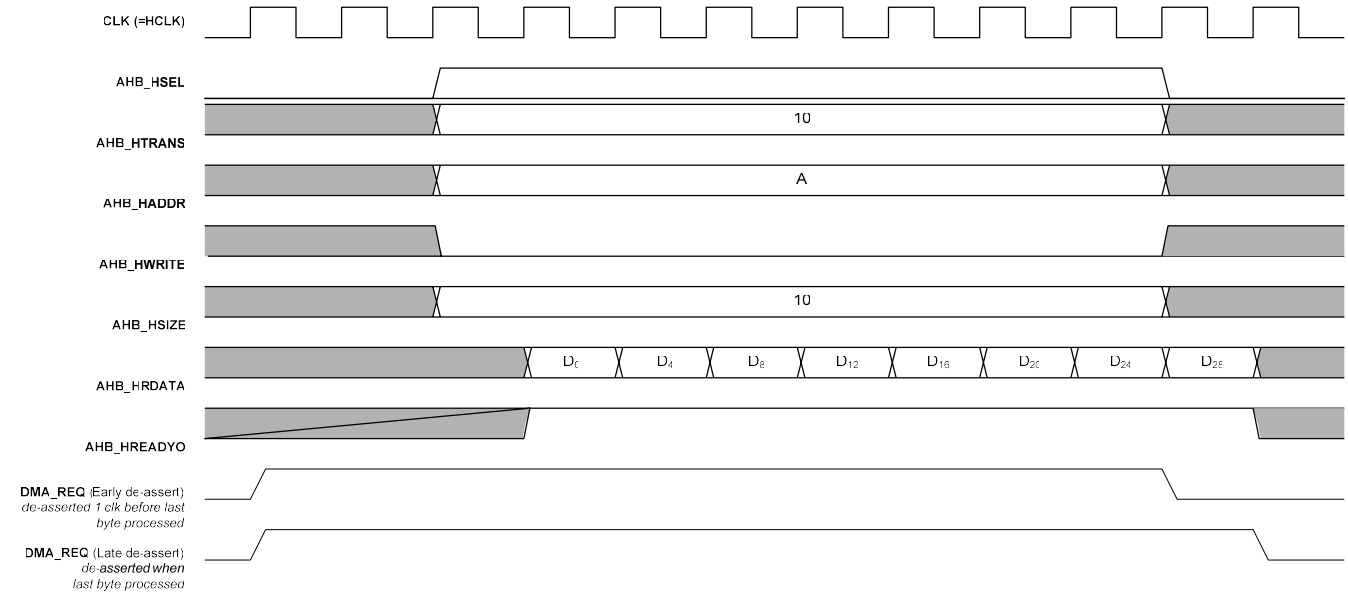
机密的

20. 5.2. 下一个DM A C O N T R O L E R I N T E R F A C E

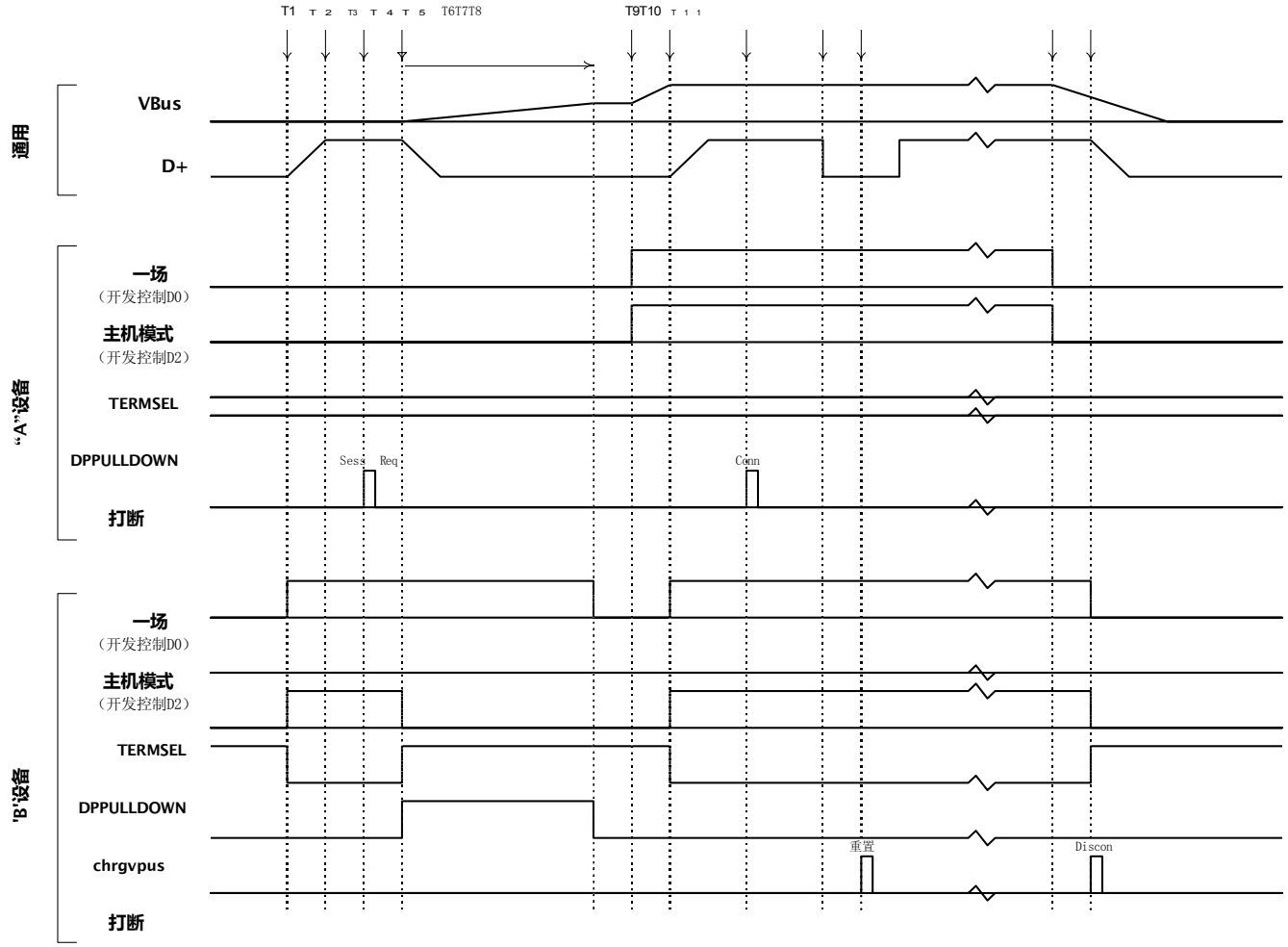
DMA加载到Tx端点（32字节）



DMA UNLOAD FROM Rx ENDPOINT (32 bytes)



20.6. sesion控制



T1'B'设备固件通过设置会话位（DevCtl.D0）请求会话启动。T2'B'设备对数据线进行脉冲。

T3'A'设备检测到数据线脉冲并生成会话请求中断（IntrUSB.D6）。T4'B'器件脉冲VBus（通过CHRGVBUS取高电平）。

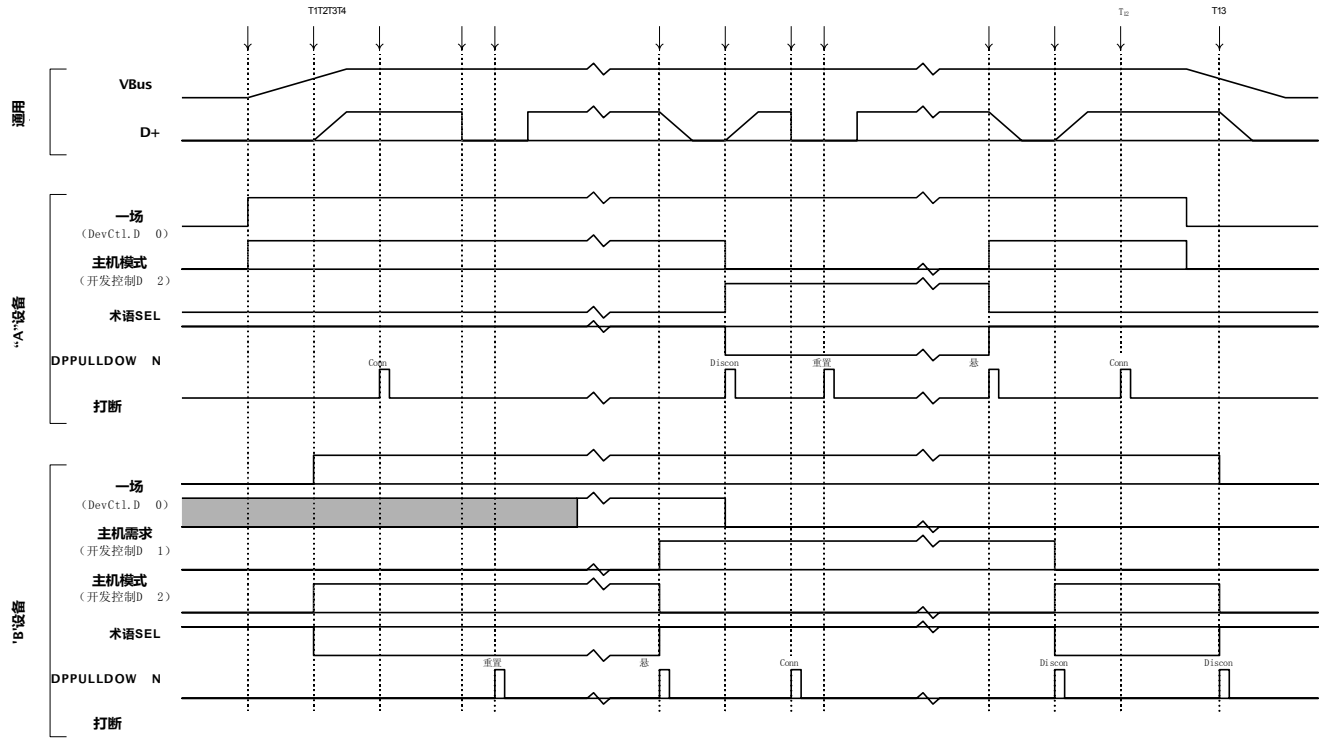
T5'A'设备固件通过设置会话位（DevCtl.D0）来启动会话。

T6'B'设备检测到会话启动并取TERMSEL高电平（可用于向D+施加上拉电阻器）。T7'A'设备检测到上拉电阻器的添加，并产生连接中断（IntrUSB.D4）。

T8'A'设备固件重置'B'设备，然后启动事务。T9'B'设备检测到重置并生成重置中断（IntrUSB.D2）。T10"A"设备固件通过清除会话位来结束会话。

T11 'B'设备检测到会话结束并生成断开连接中断（IntrUSB.D5）。

20.7. 主机谈判



T1'A'设备固件通过设置会话位（DevCtl.D0）启动会话。

T2'B'设备检测到会话启动并取TERMSEL高（可用于将上拉电阻器施加到D+）。T3'A'设备检测到'B'设备的存在，并生成连接中断（IntrUSB.D4）。

T4'A'设备固件重置'B'设备，然后启动事务。T5'B'设备检测到重置并生成重置中断（IntrUSB.D2）。

T6'B'设备在设置Host Req（DevCtl.D1）时检测到总线上的挂起模式，生成挂起中断（IntrUSB.D0）并将TERMSL设为低电平，从而消除D+上的上拉。

T7'A'设备检测到断开连接，生成断开连接中断（IntrUSB.D5），并将其TERMSEL设为高电平（再次可用于向D+施加上拉）。

T8'B'设备检测到'A'设备，生成连接中断（IntrUSB.D4）并启动'A'装置的重置。（"B"设备固件可以在20ms后清除重置位（Power.D3）并开始事务。）

T9'A'设备检测到重置并生成重置中断（IntrUSB.D2）。

T10'A'设备检测到总线上的挂起模式，生成挂起中断（IntrUSB.D0），并将TERMSEL设为低电平，消除上拉。

T11'B'设备检测到断开连接，生成断开连接中断（IntrUSB.D5），并将其TERMSEL设为高电平，将其上拉施加到D+。

T12'A'设备检测到'B'设备，然后通过清除会话位结束会话。

T13'B'设备检测到会话结束并生成断开连接中断（IntrUSB.D5）。

21. 控制事务S（通过终结点0）

2.1. 控制角色TRANSACTION SASAPERIPHERAL

端点0是核心的主要控制端点。因此，为端点0提供服务所需的例程比为其他端点提供服务所需要的例程更复杂。

需要该软件来处理可能通过端点0发送或接收的所有标准设备请求。*通用串行总线规范*2.0版第9章对此进行了说明。用于这些设备请求的协议涉及每次传输的不同数量和类型的事务。为了适应这种情况，CPU需要采用状态机方法来解码和处理命令。

USB外围设备接收的标准设备请求可分为三类：*零数据请求*（其中所有信息都包含在命令中）、*写入请求*（其中命令后面将有额外数据）和*读取请求*（其中设备需要将数据发送回主机）。

本节介绍软件处理这些不同类型的设备请求所必须执行的操作序列。

注意：与任何标准设备请求相关联的设置数据包应包括一个8字节的命令。任何包含8字节以外的命令字段的Setup数据包都将被MUSBHDRC核心自动拒绝。

21. 1. 1. 大小问题

零数据请求的所有信息都包含在8字节命令中，不需要传输额外的数据。“零数据”标准设备请求的示例有：SET_FEATURE、CLEAR_FEATURE、SET_ADDRESS、SET_CONFIGURATION、SET_INTERFACE。

当软件接收到Endpoint 0中断时，事件序列将与所有请求一样开始。RxPtRdy位（CSR0L.D0）也将被设置。然后应从端点0 FIFO读取8字节命令，对其进行解码并采取适当的操作。例如，如果命令是SET_ADDRESS，则应将命令中包含的7位地址值写入FAddr寄存器。

然后应写入CSR0寄存器以设置ServicedRxPktRdy位（D6）（指示已从FIFO读取命令）和设置DataEnd位（D3）（指示此请求不需要其他数据）。

当主机移动到请求的状态阶段时，将生成第二个Endpoint 0中断，以指示请求已完成。软件不需要进一步的操作：第二次中断只是确认请求成功完成。

如果该命令是无法识别的命令，或者由于某些其他原因无法执行，则在对其进行解码后，应写入CSR0寄存器以设置ServicedRxPktRdy位（D6）和SendStall位（D5）。当主机移动到请求的状态阶段时，MUSBHDRC将发送一个STALL，告诉主机请求没有执行。将生成第二个端点0中断，并设置SentStall位（CSR0L.D2）。

如果主机在设置DataEnd位后发送更多数据，则MUSBHDRC将发送STALL。将生成端点0中断，并设置SentStall位（CSR0L.D2）。

21. 1. 2. 书面要求

写入请求涉及在8字节命令之后从主机发送的额外数据包。“写入”标准设备请求的示例是：SET_DESCRIPTOR。

当软件接收到Endpoint 0中断时，事件序列将与所有请求一样开始。RxPtRdy位（CSR0L.D0）也将被设置。然后应从端点0 FIFO读取8字节命令并对其进行解码。

与零数据请求一样，应写入CSR0寄存器以设置ServicedRxPktRdy位（D6）（指示已从FIFO读取命令），但在这种情况下，不应设置DataEnd位（D3）（指示预期会有更多数据）。

当接收到第二个端点0中断时，应读取CSR0寄存器以检查端点状态。RxPtRdy比特（CSR0L.D0）应当被设置为指示已经接收到数据分组。然后应该读取COUNT0寄存器以确定该数据包的大小。然后可以从端点0 FIFO读取数据包。

如果与请求相关联的数据的长度（由命令中的*wLength*字段指示）大于端点0的最大数据包大小，则将发送更多的数据包。在这种情况下，应该写入CSR0来设置ServicedRxPktRdy位，但不应该设置DataEnd位。

当接收到所有预期的数据包时，应写入CSR0寄存器以设置ServicedRxPktRdy位和设置DataEnd位（表示不再需要数据）。

当主机移动到请求的状态阶段时，将生成另一个Endpoint 0中断，以指示请求已完成。软件不需要进一步的操作，中断只是对请求成功完成的确认。

如果该命令是无法识别的命令，或者由于某些其他原因无法执行，则在对其进行解码后，应写入CSR0寄存器以设置ServicedRxPktRdy位（D6）和SendStall位（D5）。当主机发送更多数据时，MUSBHDRC将发送一个STALL，告诉主机请求没有执行。将生成端点0中断，并设置SentStall位（CSR0L.D2）。

如果主机在设置DataEnd后发送更多数据，则MUSBHDRC将发送STALL。将生成端点0中断，并设置SentStall位（CSR0L.D2）。

21. 1. 3.问题

读取请求具有在8字节命令之后从函数发送到主机的数据包。“读取”标准设备请求的示例有：GET_CONFIGURATION、GET_INTERFACE、GET_DESCRIPTOR、GET_STATUS、SYNCH_FRAME。

当软件接收到Endpoint 0中断时，事件序列将与所有请求一样开始。RxPtRdy位（CSR0L.D0）也将被设置。然后应从端点0 FIFO读取8字节命令并对其进行解码。然后应写入CSR0寄存器，以设置ServicedRxPktRdy位（D6）（指示命令已从FIFO读取）。

然后，应将要发送到主机的数据写入端点0 FIFO。如果要发送的数据大于端点0的最大数据包大小，则只应将最大数据包尺寸写入FIFO。然后应该写入CSR0寄存器以设置TxPktRdy位（D1）（指示FIFO中有要发送的数据包）。当数据包已发送到主机时，将生成另一个Endpoint 0中断，并且可以将下一个数据包写入FIFO。

当最后一个数据包已写入FIFO时，应写入CSR0寄存器以设置TxPktRdy位并设置DataEnd位（D3）（指示在该数据包之后没有更多数据）。

当主机移动到请求的状态阶段时，将生成另一个Endpoint 0中断，以指示请求已完成。软件不需要进一步的操作：中断只是对请求成功完成的确认。

如果该命令是无法识别的命令，或者由于某些其他原因无法执行，则在对其进行解码后，应写入CSR0寄存器以设置ServicedRxPktRdy位（D6）和SendStall位（D5）。当主机请求数据时，MUSBHDRC将发送一个STALL，告诉主机该请求没有执行。将生成端点0中断，并设置SentStall位（CSR0L.D2）。

如果在设置了DataEnd（D3）之后主机请求更多数据，则MUSBHDRC将发送STALL。将生成端点0中断，并设置SentStall位（CSR0L.D2）。

21．1．4.结束 时间0

当MUSBHDRC作为外围设备运行时，端点0控制需要三种模式——IDLE、TX和RX——对应于控制传输的不同阶段以及端点0为传输的不同相位进入的状态。

通电或重置时的默认模式应为IDLE。

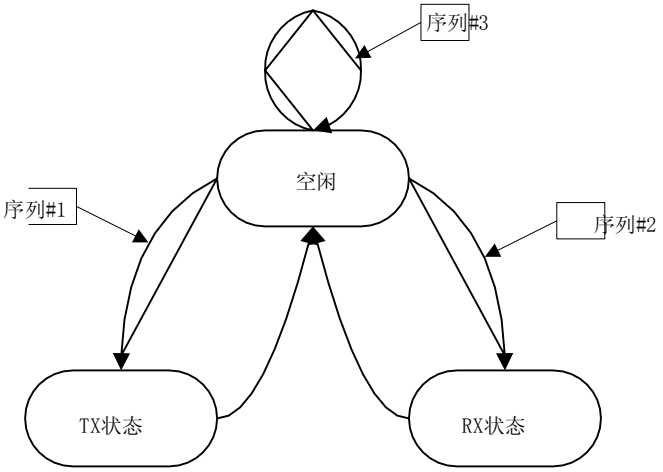
当端点0处于IDLE状态时，RxPktRdy（CSR0L.D0）被设置指示新的设备请求。一旦从FIFO中卸载设备请求，MUSBHDRC就对描述符进行解码，以确定是否存在数据阶段，如果存在，则确定控制传输的数据阶段的方向（以便设置FIFO方向）。

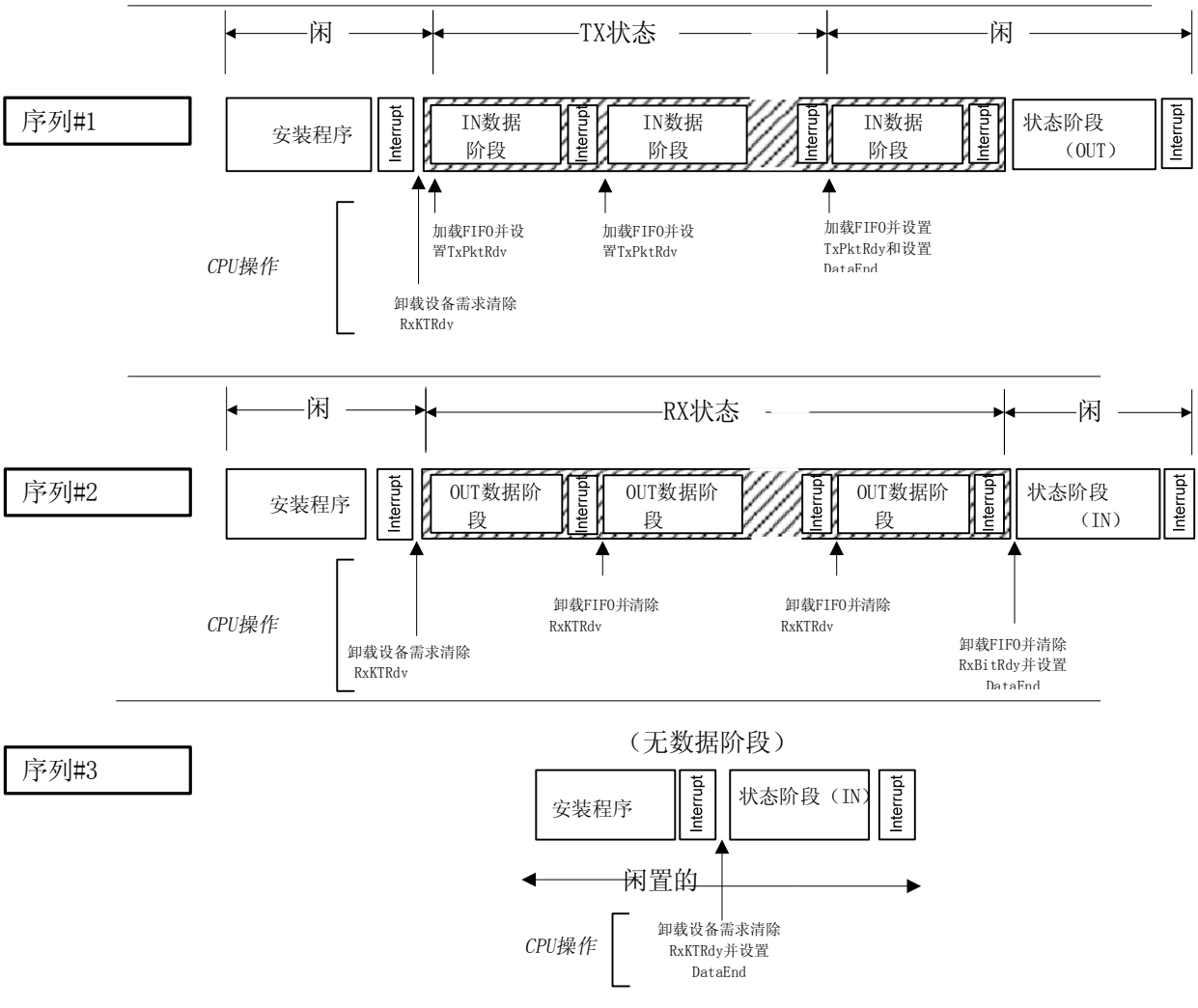
根据数据阶段的方向，端点0进入TX状态或RX状态。如果没有数据阶段，端点0将保持在IDLE状态以接受下一个设备请求。

CPU在可能传输的不同阶段需要采取的操作（例如加载FIFO、设置TxPktRdy）如下图所示。

注意，MUSBHDRC根据独立于CPU的数据相位的方向来改变FIFO方向。

图21-1外围设备的端点0状态





21.1.5. 结束0 SERVICE ROUTINE AS PERIPHERAL

将生成终结点0中断：

- 当核心在接收到有效令牌并且数据已写入FIFO之后设置RxPktRdy位（CSR0L.D0）时。
- 当FIFO中的数据包已成功传输到主机后，核心清除TxPktRdy位（CSR0L.D1）时。
- 当核心在控制事务由于协议冲突而结束之后设置SentStall位（CSR0L.D2）时。
- 当核心设置SetupEnd位（CSR0L.D4）时，因为在设置DataEnd（CSR0L.D3）之前控制传输已经结束。

每当进入Endpoint 0服务例程时，固件必须首先检查当前控制传输是否由于STALL条件或控制传输过早结束而结束。如果控制传输由于STALL条件而结束，则将设置SentStall位。如果控制传输由于控制传输过早结束而结束，则将设置SetupEnd位。在任何一种情况下，固件都应中止处理当前控制传输，并将状态设置为IDLE。

一旦固件确定中断不是由非法总线状态产生的，则采取的下一个操作取决于端点状态。

如果端点0处于IDLE状态，则可以生成中断的唯一有效原因是核心从USB总线接收数据。服务例程必须通过测试RxPktRdy（CSR0L.D0）位来检查这一点。如果设置了该位，则核心已接收到SETUP数据包。这必须从FIFO中卸载并解码，以确定核心必须采取的行动。
根据SETUP数据包中包含的命令，端点0将进入以下三种状态之一：

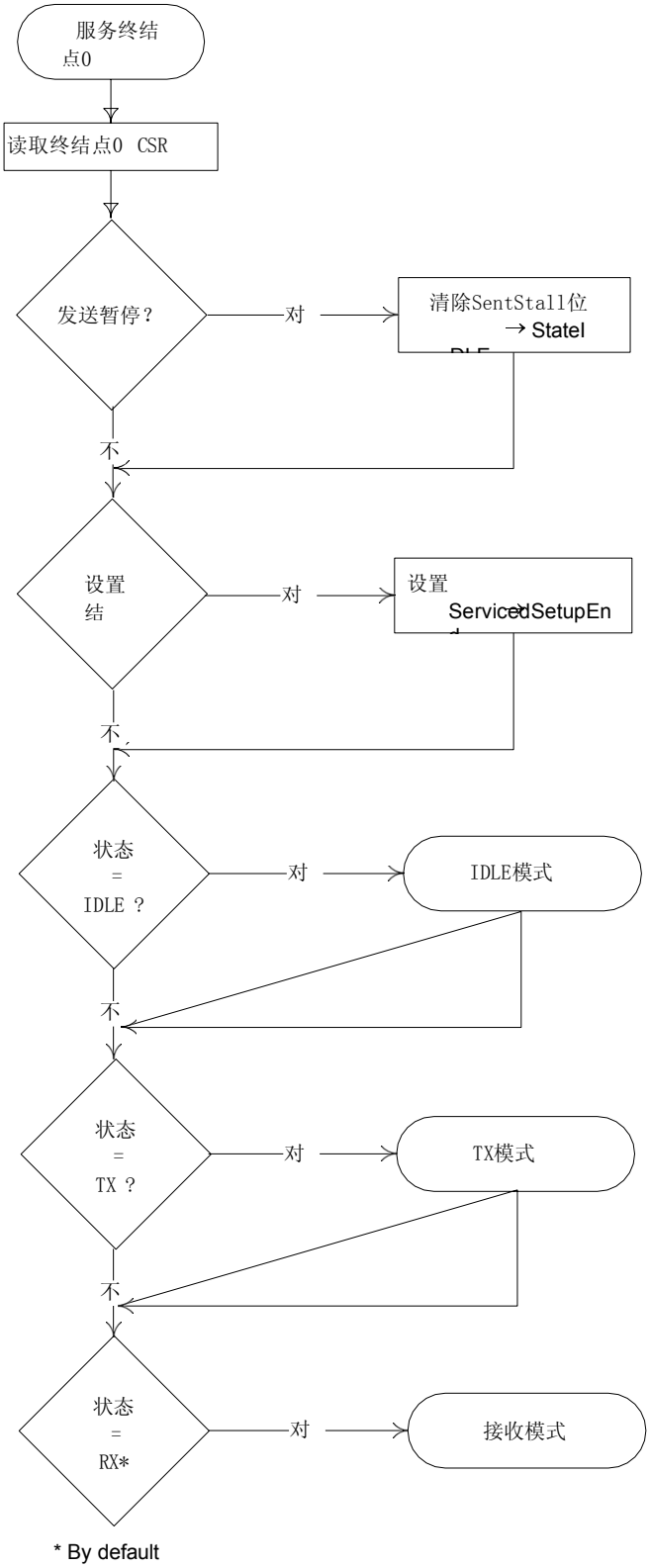
- 如果该命令是没有任何数据阶段的单个数据包事务（SET_ADDRESS、SET_INTERFACE等），则端点将保持在IDLE状态。
- 如果命令具有OUT数据阶段（SET_DESCRIPTOR等），则端点将进入RX状态。
- 如果命令具有IN数据阶段（GET_DESCRIPTOR等），则端点将进入TX状态。

如果端点处于TX状态，则中断表示核心已接收到in令牌，并且已发送来自FIFO的数据。如果主机仍在等待更多数据[2]，固件必须通过在FIFO中放置更多数据来对此做出响应，或者通过设置DataEnd位来指示数据阶段已完成。一旦事务的数据阶段完成，端点0应返回到IDLE状态，以等待下一个控制事务。

如果端点处于RX状态，则中断指示已经接收到数据包。固件必须通过从FIFO中卸载接收到的数据来做出响应。然后固件必须确定它是否已经接收到所有期望的数据[2]。如果有，固件应设置DataEnd位并将Endpoint 0返回IDLE状态。如果预计会有更多数据，固件应设置ServicedRxPktRdy位（CSR0L.D6），以指示它已读取FIFO中的数据，并使端点处于RX状态。

[2]命令事务都包括一个字段，该字段指示主机期望接收或将要发送的数据量。

图21-2 MUSBHDRC作为外设运行时端点0服务例程的流程

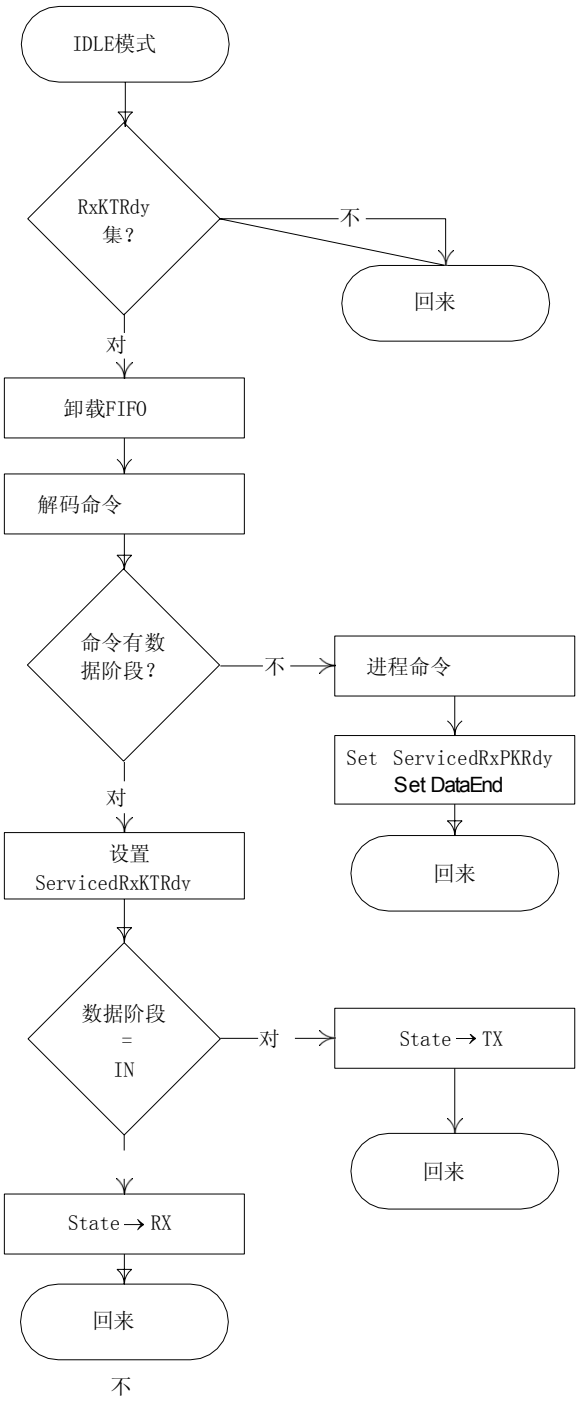


21. 1. 5. 1.身份证

IDLE模式是端点0控件在通电或重置时需要选择的模式，也是端点0控件应在RX和TX模式终止时返回的模式。

这也是处理控制传输的SETUP阶段的模式（如下图所示）。

图21-3 MUSBHDRC作为外设运行时控制传输的设置阶段流程



21.1. 5. 2. TX MODE

当端点处于TX状态时，所有到达的in令牌都需要作为数据阶段的一部分进行处理，直到将所需的数据量发送到主机为止。如果在端点处于TX状态时接收到SETUP或OUT令牌，这将导致出现SetupEnd条件，因为核心只期望in令牌。

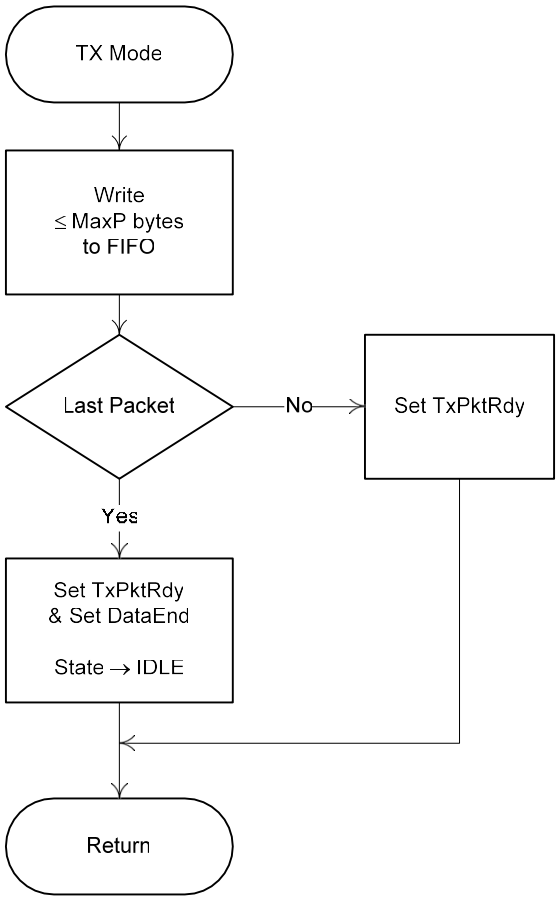
三个事件可能导致TX模式在发送预期数据量之前终止：

- 主机发送一个无效令牌，导致SetupEnd条件（CSR0L.D4集）
- 固件发送包含小于端点0（MaxP）的最大数据包大小的数据包
- 固件发送一个空数据包

在事务终止之前，固件只需在接收到指示数据包已从FIFO发送的中断时加载FIFO。（当TxPktRdy被清除时，会产生一个中断。）

当固件（通过发送短的或空的数据包）强制终止传输时，它应该设置DataEnd位（CSR0L.D3），以向核心指示数据阶段已经完成，并且核心接下来应该接收确认包。

图21-4当MUSBHDRC作为外设运行时，控制传输的IN数据阶段的流程



21. 1. 5. 3.RX MODE

在RX模式中，所有到达的数据都应被视为数据阶段的一部分，直到接收到预期的数据量。如果在端点处于RX状态时接收到SETUP或IN令牌，这将导致出现SetupEnd条件，因为核心只需要OUT令牌。

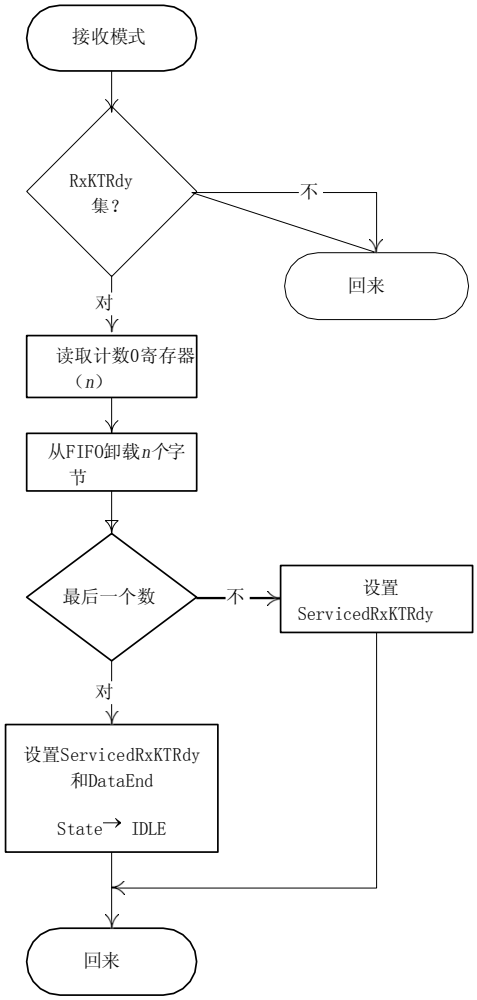
三个事件可导致RX模式在接收到预期数据量之前终止:

- 主机发送一个无效令牌，导致SetupEnd条件（CSR0L.D4集）
- 主机发送的数据包小于终结点0的最大数据包大小
- 主机发送一个空数据包

在事务终止之前，固件只需在接收到指示新数据已到达的中断时卸载FIFO（RxPktRdy（CSR0L.D0）设置），并通过设置ServicedRxPKtRy位（CSR0L.D6）来清除RxPktRdy。

当固件检测到传输终止时（通过接收预期的数据量或空数据包），它应该设置DataEnd位（CSR0L.D3）以向核心指示数据阶段已经完成，并且核心接下来应该接收确认包。

图21-5当MUSBHDRC作为外设运行时，控制传输的OUT数据阶段的流程



21. 1. 6.错误O R H A ND LI N G A S A PE RI P H ER A L

由于USB上的协议错误、主机过早结束传输，或者如果功能控制器软件希望中止传输（例如，因为它无法处理命令），控制传输可能会中止。

MUSBHDRC将自动检测协议错误，并在以下情况下向主机发送STALL数据包

条件

*1. 主机在写入请求的OUT data阶段发送的数据比命令中指定的要多。*当设置了DataEnd位（CSR0L.D3）之后主机发送OUT令牌时，会检测到这种情况。

*2. 主机在读取请求的IN数据阶段请求的数据多于命令中指定的数据。*当主机在CSR0寄存器中的DataEnd位被设置后发送IN令牌时，会检测到这种情况。

3. 主机在OUT数据包中发送超过MaxP个数据字节。

4. 主机在读取请求的STATUS阶段发送一个非零长度的DATA1数据包。

当MUSBHDRC发送了STALL数据包时，它设置SentStall位（CSR0L.D2）并生成中断。当软件接收到设置了SentStull位的Endpoint 0中断时，应中止当前传输，清除SentStul位，然后返回IDLE状态。

如果主机在传输请求的所有数据之前进入STATUS阶段，或在完成当前传输之前发送新的SETUP数据包，从而提前结束传输，则将设置SetupEnd位（CSR0L.D4），并生成Endpoint 0中断。当软件接收到设置了SetupEnd位的Endpoint 0中断时，应中止当前传输，设置ServicedSetupEnd位（CSR0L.D7），并返回IDLE状态。如果RxPktRdy位（CSR0L.D0）被设置，这表示主机已经发送了另一个SETUP数据包，然后软件应该处理该命令。

如果软件由于无法处理命令或存在其他内部错误而想要中止当前传输，则应设置SendStall位（CSR0L.D5）。然后，MUSBHDRC将向主机发送STALL数据包，设置SentStall位（CSR0L.D2），并生成Endpoint 0中断。

21.1.7.ADDITIONALACTIONS

当作为外设工作时，MUSBHDRC核心会自动响应USB总线上的某些条件或主机的操作。详情如下：

STALLISsueDTOCONTROLtranSFER

在以下条件下，MUSBHDRC核心将自动向控制传输发出STALL握手：

1. 主机在控制传输的OUT data阶段发送的数据比SETUP阶段设备请求中指定的数据多。

当CPU卸载最后一个OUT数据包并设置DataEnd后，主机发送OUT令牌（而不是IN令牌）时，MUSBHDRC会检测到这种情况。

2. 主机在控制传输的IN数据阶段请求的数据比在SETUP阶段设备请求中指定的数据多。

当CPU已经清除TxPktRdy并响应于主机发出的ACK将DataEnd设置为应该是最后一个数据包之后，主机发送IN令牌（而不是OUT令牌）时，MUSBHDRC检测到这种情况。

3. 主机使用OUT数据令牌发送超过MaxP的数据。

4. 主机为OUT状态阶段发送超过零长度的数据包。

ZERO-LENGTHOUTDATAPACKETSINCONTROLTRANSFERS

使用零长度OUT数据包来指示控制传输的结束。在正常操作中，只有在传输了设备请求的整个长度之后（即，在CPU设置了DataEnd之后），才应该接收这样的分组。然而，如果主机在传输了整个长度的设备请求之前发送了一个零长度的OUT数据包，这表示传输提前结束。在这种情况下，MUSBHDRC将自动从FIFO中清除CPU加载的为数据阶段做好准备的任何In令牌，并设置SetupEnd（CSR0L.D4）。

2 1.2 控制角色TR A N S A CTION S A S A H O ST

主机控制事务通过端点0进行，并且要求软件处理可能通过端点0发送或接收的所有标准设备请求（如*通用串行总线规范*，修订版2.0，第9章所述）。

对于USB外设，有三类标准设备请求需要处理：*零数据请求*（其中所有信息都包含在命令中）；*写入请求*（其中命令后面将跟着附加数据）；以及*读取请求*（其中要求设备将数据发送回主机）。

- 零数据请求包括SETUP命令和IN状态阶段
- 写入请求包括一个SETUP命令，后面是OUT数据阶段，而OUT数据阶段又是in状态阶段。
- 读取请求包括一个SETUP命令，后面跟着IN数据阶段，IN数据阶段后面跟着OUT状态阶段。

可以设置超时以限制MUSBHDRC将重试被目标持续NAKed的事务的时间长度。该限制可以在2到2个[15]帧/微帧之间，并通过NAKLimit0寄存器设置（见第3.3.6节）。

以下部分通过查看控制事务的不同阶段中要采取的步骤，描述CPU在发出这些不同类型的请求时需要采取的操作。

注意：在作为主机启动任何事务之前，需要将FAddr寄存器设置为寻址外围设备。首次连接设备时，应将FAddr设置为零。发出SET_ADDRESS命令后，应将FAddr设置为目标的新地址。

21. 2. 1.设置U PP H A S E A S A H O S T

对于控制事务的设置阶段，驱动主机设备的CPU需要：

1. 将所需设备请求命令的8个字节加载到端点0 FIFO
2. 然后设置SetupPkt和TxPtRdy（分别为位CSR0L.D3和CSR0L.D1）。*注意*：这些位需要设置在一起。

   然后，MUSBHDRC继续向被寻址设备的端点0发送SETUP令牌，然后发送8字节命令，必要时重试。（此操作的详细信息见第19.1节。）
3. 在发送数据的尝试结束时，MUSBHDRC将生成端点0中断（即设置IntrTx.D0）。然后CPU应读取CSR0以确定RxStall位（D2）、Error位（D4）或NAK Timeout位（D7）是否已被设置。

   如果设置了RxStall，则表示目标没有接受该命令（例如，因为目标设备不支持该命令），因此发出了STALL响应。

   如果设置了Error，则意味着MUSBHDRC已尝试发送SETUP数据包和以下数据包三次，但没有得到任何响应。

   如果设置了NAK Timeout（NAK超时），则意味着MUSBHDRC已接收到对每次发送SETUP数据包的尝试的NAK响应，时间长于NAKLimit0寄存器中设置的时间。然后，可以通过清除NAK超时位来指示MUSBHDRC继续尝试该事务（直到它再次超时），或者通过在清除NAK Timeout位之前刷新FIFO来中止该事务。
4. 如果没有设置RxStall、Error或NAK Timeout，则SETUP Phase已被正确确认，CPU应进入为特定标准设备请求指定的以下IN Data Phase、OUT Data Phase或IN Status Phase。

21. 2. 2. 输入D A T A P H A S E A S A H O S T

对于控制事务的IN数据阶段，驱动主机设备的CPU需要：

1. 设置ReqPkt（CSR0L.D5）。
2. 等待MUSBHDRC发送IN令牌并接收所需数据。（此操作的详细信息

如第19.1节所示。）

3. 当MUSBHDRC生成端点0中断（即设置IntrTx.D0）时，读取CSR0以确定是否设置了RxStall位（D2）、Error位（D4）、NAK Timeout位（D7）或RxPtRdy（D0）。

   如果设置了RxStall，则表示目标已发出STALL响应。

   如果设置了Error，则表示MUSBHDRC已尝试发送所需的IN令牌三次，但未得到任何响应。

   如果设置了NAK Timeout，则意味着MUSBHDRC已接收到对每次发送IN令牌的尝试的NAK响应，时间长于NAKLimit0寄存器中设置的时间。然后可以指示MUSBHDRC通过清除NAK超时位来继续尝试该事务（直到它再次超时），或者通过在清除NAK Timeout位之前清除ReqPkt来中止该事务。

4. 如果已设置RxPktRdy，则CPU应从端点0 FIFO读取数据，然后清除RxPktRdy。

5. 如果需要更多数据，CPU应重复步骤1-4。

成功接收所有数据后，CPU应进入控制事务的OUT状态阶段。

21.2.3. 输出DATA PHASE AS A HOST

对于控制事务的OUT数据阶段，驱动主机设备的CPU需要：

1. 将要发送的数据加载到端点0 FIFO中。

2. 然后设置TxPtRdy位（CSR0L.D1）。

   然后，MUSBHDRC继续发送OUT令牌，然后从FIFO向被寻址设备的端点0发送数据，必要时重试。（此操作的详细信息见第19.1节。）

3. 在发送数据的尝试结束时，MUSBHDRC将生成端点0中断（即设置IntrTx.D0）。然后CPU应读取CSR0以确定RxStall位（D2）、Error位（D4）或NAK Timeout位（D7）是否已被设置。

   如果设置了RxStall，则表示目标已发出STALL响应。

   如果设置了Error，则意味着MUSBHDRC已尝试发送OUT令牌和以下数据包三次，但没有得到任何响应。

   如果设置了NAK Timeout，则意味着MUSBHDRC已接收到对每次发送OUT令牌的尝试的NAK响应，时间长于NAKLimit0寄存器中设置的时间。然后，可以通过清除NAK超时位来指示MUSBHDRC继续尝试该事务（直到它再次超时），或者通过在清除NAK Timeout位之前刷新FIFO来中止该事务。

   如果未设置RxStall、Error或NAKLimit，则OUT数据已正确确认。

4. 如果需要发送更多数据，CPU应重复步骤1-3。

当所有数据都已成功发送时，CPU应进入控制事务的IN状态阶段。

21.2.4. 在STATUS PHASE AS A HOST中

（在设置阶段或退出数据阶段之后）

对于控制事务的IN状态阶段，驱动主机设备的CPU需要：

1. 设置StatusPkt和ReqPkt（分别为位CSR0L.D6和CSR0L.D5）。*注意：*这些位需要设置在一起，即在相同的写入操作中。

2. 等待MUSBHDRC同时发送IN令牌和接收来自USB外围设备的响应。（此操作的详细信息见第19.1节。）

3. 当MUSBHDRC生成端点0中断（即设置IntrTx.D0）时，读取CSR0以确定

RxStall位（D2）、Error位（D4）、NAK Timeout位（D7）或RxPktRdy（D0）已被设置。

如果设置了RxStall，则表示目标无法完成该命令，因此已发出STALL响应。

如果设置了Error，则表示MUSBHDRC已尝试发送所需的IN令牌三次，但未得到任何响应。

如果设置了NAK Timeout，则意味着MUSBHDRC已接收到对每次发送IN令牌的尝试的NAK响应，时间长于NAKLimit0寄存器中设置的时间。然后，可以通过清除NAK超时位来指示MUSBHDRC继续尝试该事务（直到它再次超时），或者通过在清除NAK Timeout位之前清除ReqPkt和StatusPkt来中止该事务。

4. CPU应清除StatusPitt位，以及RxPitRdy（即在与之相同的写入操作中）（如果已设置）。

21.2.5. 输出STATUS PHASE AS A HOST

（以下为数据阶段）

对于控制事务的OUT状态阶段，驱动主机设备的CPU需要：

1. 设置StatusPut和TxPktRdy（分别为位CSR0L.D6和CSR0L.D1）。注意：这些位需要设置在一起。
2. 等待MUSBHDRC同时发送OUT令牌和零长度DATA1数据包。（此操作的详细信息见第19.1节。）
3. 在发送数据的尝试结束时，MUSBHDRC将生成端点0中断（即设置IntrTx.D0）。然后CPU应读取CSR0以确定RxStall位（D2）、Error位（D4）或NAK Timeout位（D7）是否已被设置。

   如果设置了RxStall，则表示目标无法完成该命令，因此已发出STALL响应。

   如果设置了Error，则表示MUSBHDRC已尝试发送STATUS数据包和以下数据包三次，但未得到任何响应。

   如果设置了NAK Timeout，则意味着MUSBHDRC已接收到对每次发送IN令牌的尝试的NAK响应，时间长于NAKLimit0寄存器中设置的时间。然后，可以通过清除NAK超时位来指示MUSBHDRC继续尝试该事务（直到它再次超时），或者通过在清除NAK Timeout位之前刷新FIFO来中止该事务。
4. 如果未设置RxStall、Error或NAK Timeout，则STATUS Phase已正确确认。

22. Bulk tranSAC离子

22.1. HANDLING BULK TRANSACTIONS AS A PERPHERAL

22.1.1.业务单位KINTRANSACTIONS

Bulk IN事务用于将非周期性数据从功能控制器传输到主机。

四个可选功能可用于批量输入事务的外围模式中使用的Tx端点：

双数据包缓冲

除非使用动态FIFO大小，否则当写入TxMaxP寄存器的值小于或等于分配给端点的FIFO大小的一半时，将自动启用双数据包缓冲。（在选择动态FIFO大小的情况下，单包或双包缓冲的使用是端点FIFO规范的一部分——请参阅第3.3.18节）启用时，最多可以在FIFO中存储两个包，等待传输到主机。

DMA

如果为端点启用了DMA，则只要端点能够接受其FIFO中的另一个数据包，就会生成DMA请求。此功能可用于允许DMA控制器（例如MUSBHDRC设计中可选包含的控制器）

在没有处理器干预的情况下将数据包加载到FIFO中。如果使用DMA模式1，则TxMaxP[D10:0]必须设置为偶数，以便正确生成中断。

自动调压

当启用自动设置功能时，当TxMaxP字节的数据包已加载到FIFO时，TxPktRdy位（TxCSRL.D0）将自动设置。当DMA用于加载FIFO时，这尤其有用，因为它避免了在大批量传输期间加载单个数据包时需要任何处理器干预。

自动数据包拆分

对于一些系统设计，应用软件在单个操作中将比在单个USB操作中传输的数据量更大的数据量写入端点可能是方便的。一个特定的情况是，在某些情况下，相同的端点用于512字节的高速传输，但在其他情况下用于全速传输。当以全速操作时，单个操作中传输的最大数据量仅为64字节。为了适应这种情况，MUSBHDRC包括一个配置选项，如果选择该选项，则允许将较大的数据包写入批量端点，然后将批量端点拆分为适当（指定）大小的数据包，以便在USB总线上传输。是否选择此选项可以通过ConfigData寄存器的MPTxE位（D6）的设置来确定（见第3.3.5节）。通过TxMaxP寄存器设置必要的数据包大小信息（见第3.3.7节）。

22. 1. 1. 1.设置

在为批量事务配置Tx端点时，必须使用端点的最大数据包大小（以字节为单位）写入TxMaxP寄存器。此值应与终结点的标准终结点描述符的*wMaxPacketSize*字段相同。此外，IntrTxE寄存器中的相关中断启用位应设置为“1”（如果此端点需要中断），TxCSR寄存器的高字节应设置为如下所示（D9–D8位未使用）：

D15	**自动调压**	0/1	如果需要“自动设置”功能，请设置为1。
D14	**国际标准化组织**	0	设置为0可启用批量协议。
D13	**模式**	1.	设置为1以确保启用FIFO（仅当FIFO与Rx端点共享时才需要）。
D12	**DMA ReqEnab**	0/1	如果需要DMA请求，则设置为1。*注：*如果设置为1，还需要选择所选的DMAReqMode（TxCSRH.D2）。
D11	**FrcDataTog**	0	设置为0以允许正常的数据切换操作。

当首次配置端点时（在端点0上的SET_CONFIGURATION或SET_INTERFACE命令之后），应写入TxCSR的较低字节以设置ClrDataTog位（D6）。这将确保数据切换（由MUSBHDRC自动处理）以正确的状态开始。此外，如果FIFO中有任何数据包（由正在设置的FIFONotEmpty位（TxCSRL.D1）指示），则应通过设置FlushFIFO位（TxCSRL.D3）来刷新它们。*注意：*如果启用了双缓冲，则可能需要连续两次设置此位。

22.1. 1. 2. 操作程序

当数据要通过Bulk IN管道传输时，需要将数据包加载到FIFO中，并写入TxCSR寄存器以设置TxPktRdy位（D0）。当数据包已发送时，TxPktRdy位将被MUSBHDRC清除，并产生一个中断，以便下一个数据包可以加载到FIFO中。如果启用了双数据包缓冲，则在加载了第一个数据包并设置了TxPktRdy位之后，该TxPktRdy位将立即被MUSBHDRC清除，并产生中断，从而可以将第二个数据包加载到FIFO中。无论是否启用双数据包缓冲，软件都应以相同的方式运行，在接收到中断时加载数据包。

在一般情况下，数据包大小不得超过TxMaxP寄存器底部11位指定的大小。寄存器的这一部分定义了通过USB传输的有效载荷（数据包大小），并且USB规范要求其为8、16、32、64（全速或高速）或512字节（仅高速）。如果要传输超过此数量的数据，则需要将其作为多个USB数据包发送，这些数据包的大小都应为TxMaxP[D0:D0]，但保留剩余数据的最后一个数据包除外。

此规则的例外情况适用于在配置core时选择了自动批量数据包拆分选项的情况。

（这可以根据ConfigData寄存器的MPTxE位（D6）的设置来确定。）在选择此选项的情况下，可以将大小高达TxMaxP[D0:D0]指定的32倍的数据包写入FIFO（假设FIFO足够大，可以接受这些较大的数据包），然后由核心将其拆分为适当大小的数据包，以便通过USB传输。写入FIFO的数据包大小由*m*表示◈ *USB有效载荷*，其中TxMaxP[D15:D11]=*m*-1。为了利用这一功能，应用软件所需要做的就是在TxMaxP寄存器中设置适当的值（并确保写入位10:0的值与相关端点的标准端点描述符的*wMaxPacketSize字段*中给定的值匹配）。就应用软件而言，传输这些较大数据包的过程与用于传输标准大小的Bulk数据包的流程没有什么不同。

主机可以通过知道预期的数据总量来确定用于传输的所有数据已经被发送。或者，当它接收到小于所述有效载荷（TxMaxP[D0:D0]）的分组时，它可以推断出所有数据都已被发送。在后一种情况下，如果数据块的总大小是该有效载荷的倍数，则在发送了所有数据之后，函数将需要发送空分组。这是通过在接收到下一个中断时设置TxPktRdy来完成的，而不将任何数据加载到FIFO中。

如果正在传输大块数据，那么可以通过使用DMAError Handling来避免调用中断服务例程来加载每个数据包的开销

如果软件想要关闭Bulk IN管道，则应设置SendStall位（TxCSRL.D4）。当MUSBHDRC接收到下一个IN令牌时，它将向主机发送STALL，设置SentStall位（TxCSRL.D5）并生成中断。

当软件接收到设置了SentStall位（TxCSRL.D5）的中断时，它应该清除SentStalk位。但是，它应该保留SendStall位（TxCSRL.D4）设置，直到它准备好重新启用Bulk IN管道。*注意*：如果主机由于某种原因未能接收到STALL数据包，它将发送另一个IN令牌，因此建议保留SendStall位设置，直到软件准备好重新启用Bulk IN管道。重新启用管道时，应通过设置TxCSR寄存器（D6）中的ClrDataTog位来重新启动数据切换序列。

2 2 . 1.2. B U LK组织

Bulk OUT事务用于将非周期性数据从主机传输到功能控制器。

有四个可选功能可用于批量输出事务的外围模式中使用的Rx端点：

- **双数据包缓冲**

除非使用动态FIFO大小，否则当写入RxMaxP寄存器的值小于或等于分配给端点的FIFO大小的一半时，将自动启用双数据包缓冲。（在选择动态FIFO大小的情况下，单包或双包缓冲的使用是端点FIFO规范的一部分——见第3.3.18节）启用时，最多可以在FIFO中存储两个包。

- **DMA**

如果为端点启用了DMA，则每当端点的FIFO中有数据包时，都会生成DMA请求。此功能可用于允许DMA而无需处理器干预。如果使用DMA模式1，则RxMaxP[D10:0]必须设置为偶数，以便正确生成中断。

- **自动清除**

当启用自动清除功能时，当从FIFO卸载RxMaxP字节的数据包时，RxPktRdy位（RxCSRL.D0）将自动清除（例外情况，请参阅寄存器描述）。当DMA用于卸载FIFO时，这尤其有用，因为它避免了在大批量传输期间卸载单个数据包时需要任何处理器干预。

- **自动分组组合**

对于一些系统设计，应用软件在单个操作中从端点读取比在单个USB操作中传输的数据量更大的数据量可能是方便的。一个特定的情况是，在某些情况下，相同的端点用于512字节的高速传输，但在其他情况下用于全速传输。当以全速操作时，单个操作中传输的最大数据量仅为64字节。为了适应这种情况，MUSBHDRC包括一个配置选项，如果选择该选项，则会使MUSBHDRR将

在被应用软件读取之前，通过USB总线接收的分组被转换成更大的数据分组。是否选择此选项可以通过ConfigData寄存器的MPRxE位（D7）的设置来确定（见第3.3.5节）。必要的数据包大小信息通过RxMaxP寄存器设置（见第3.3.10节），而当前要读取的合并数据包的大小在RxCount寄存器中给出（见第3.3.1 3节）。

22. 1. 2. 1.设置

在为Bulk OUT事务配置Rx端点时，必须使用端点的最大数据包大小（以字节为单位）写入RxMaxP寄存器。此值应与终结点的标准终结点描述符的*wMaxPacketSize*字段相同。此外，IntrRxE寄存器中的相关中断启用位应设置为“1”（如果此端点需要中断），RxCSR寄存器的高字节应设置为如下所示（位D10–D8未使用/只读）：

D15	**自动清除**	0/1	如果需要自动清除功能，则设置为1。
D14	**国际标准化组织**	0	设置为0可启用批量协议。
D13	**DMA ReqEnab**	0/1	如果此终结点需要DMA请求，请设置为1。*注意*：如果设置为1，还需要选择所选的DMAReqMode（RxCSRH.D3）。
D12	**DisNyet**	0	设置为0以允许正常的PING流量控制。

当首次配置端点时（在端点0上的SET_CONFIGURATION或SET_INTERFACE命令之后），应写入RxCSR的较低字节以设置ClrDataTog位（D7）。这将确保数据切换（由MUSBHDRC自动处理）以正确的状态开始。此外，如果FIFO中有任何数据包（由正在设置的RxBitRdy位（RxCSRL.D0）指示），则应通过设置FlushFIFO位（RxCSRL.D4）来刷新这些数据包。*注意*：如果启用了双缓冲，则可能需要连续两次设置此位。

22.1. 2. 2. 操作程序

当Bulk-Rx端点接收到数据包时，RxPktRdy位（RxCSRL.D0）被设置，并且产生中断。软件应读取端点的RxCount寄存器，以确定数据包的大小。应从FIFO中读取数据包，然后清除RxPktRdy。如果清除RxBitRdy时FIFOFull位被设置为1，则MUSBHDRC将首先清除FIFOFull位。然后，它将再次设置RxPktRdy，以指示FIFO中有另一个数据包等待卸载。

接收到的数据包不应超过RxMaxP寄存器中指定的大小（因为这应该是发送到主机的端点描述符的*wMaxPacketSize*字段中设置的值）。当需要向函数发送大于*wMaxPacketSize*的数据块时，它将作为多个数据包发送。所有数据包的大小都将为*wMaxPacketSize*，但最后一个包含剩余数据的数据包除外。软件可以使用特定于应用的方法来确定块的总大小，从而确定何时接收到最后的分组。或者，当它接收到大小小于*wMaxPacketSize*的分组时，它可以推断整个块已经被接收。（如果数据块的总大小是*wMaxPacketSize*的倍数，则在数据之后将发送一个空数据包，表示传输完成。）

在一般情况下，应用软件需要单独从FIFO读取每个数据包。此规则的例外情况适用于在配置核心时选择了批量数据包的自动组合选项的情况。（这可以根据ConfigData寄存器的MPRxE位（D7）的设置来确定。）如果选择了此选项，核心一次最多可以接收32个数据包，并将它们组合成FIFO内的单个数据包（假设FIFO足够大，可以接受这些较大的数据包）。写入FIFO的数据包大小由*m*表示◈ *wMaxPacketSize*，其中RxMaxP[D15:D11]=*m*–1。为了利用这一功能，应用软件所需要做的就是在RxMaxP寄存器中设置适当的值（并确保写入位10:0的值与端点描述符的*wMaxPacketSize*字段中给定的值匹配）。就应用软件而言，传输这些较大数据包的过程与用于传输标准大小的Bulk数据包的流程没有什么不同。

如果正在传输大块数据，则可以通过使用DMA来避免调用中断服务例程来卸载每个数据包的开销。

22. 1. 2. 3.呃O R H A and LI N G

如果软件想要关闭Bulk OUT管道，则应设置SendStall位（RxCSRL.D5）。当MUSBHDRC接收到下一个数据包时，它将向主机发送STALL，设置SentStall位（RxCSRL.D6）并生成中断。

当软件接收到设置了SentStall位（RxCSRL.D6）的中断时，它应该清除该位。但是，它应该保留SendStall位（RxCSRL.D5）设置，直到它准备好重新启用Bulk OUT管道。*注意*：如果主机由于某种原因未能接收到STALL数据包，它将发送另一个数据包，因此建议保留SendStall位设置，直到软件准备好重新启用Bulk OUT管道。当Bulk OUT管道重新启用时，应通过设置RxCSR寄存器（D7）中的ClrDataTog位来重新启动数据切换序列。

22.2. H A NDLING BUL K T R A N S A CT IONS A S A主机

22. 2.1. 业务单位K I N T R A N S A C T I O N A S A H O S T

Bulk IN事务可以用于将非周期性数据从功能控制器传输到主机。有五个可选功能可用于在主机模

式下用于接收此数据的Rx端点：

- **双数据包缓冲**

当启用双数据包缓冲时，可以在读取另一个数据包时接收一个数据。除非使用动态FIFO大小，否则如果写入RxMaxP寄存器的值小于或等于分配给端点的FIFO大小的一半，则会自动启用双数据包缓冲。（在选择动态FIFO大小的情况下，单包或双包缓冲的使用是端点FIFO规范的一部分——参见第8.4.2.2节。）

- **DMA**

如果为端点启用了DMA，则每当端点的FIFO中有数据包时，都会生成DMA请求。此功能可用于允许DMA在无需处理器干预的情况下从FIFO卸载数据包。

- **自动请求（uest）**

当AutoReq（uest）功能被启用时，当RxKtRdy位被清除时，ReqPkt位（RxCSRL.D5）将被自动设置。此功能可与RqPktCount寄存器结合使用，以请求所需数量的最大大小数据包。

- **自动清除**

当启用自动清除功能时，当从FIFO卸载RxMaxP字节的数据包时，RxPktRdy位（RxCSRL.DO）将自动清除（例外情况，请参阅寄存器描述）。当DMA用于卸载FIFO时，这与自动请求一起特别有用，因为它避免了在大批量传输期间卸载单个数据包时需要任何处理器干预。

- **自动分组组合**

对于一些系统设计，应用软件在单个操作中从端点读取比在单个USB操作中传输的数据量更大的数据量可能是方便的。一个特定的情况是，在某些情况下，相同的端点用于512字节的高速传输，但在其他情况下用于全速传输。当以全速操作时，单个操作中传输的最大数据量仅为64字节。为了适应这种情况，MUSBHDRC包括一个配置选项，如果选择了该选项，则会使MUSBHDSC在被应用软件读取之前，将通过USB总线接收的数据包组合成更大的数据包。是否选择此选项可以通过ConfigData寄存器的MPRxE位（D7）的设置来确定（见第3.3.5节）。
必要的数据包大小信息通过RxMaxP寄存器设置（见第3.3.10节），而当前要读取的合并数据包的大小在RxCount寄存器中给出（见第3.3.1 3节）。

22. 2. 1. 1.设置

在主机模式下启动任何批量输入事务之前：

- 目标功能地址需要按照第8.5.1节所述进行设置。
- 要使用的MUSBHDRC端点的RxType寄存器需要写入位D7、D6以选择操作速度，位D5、D4=10（选择批量传输），位D3–D0设置为设备枚举期间返回给MUSBHDRR的in端点描述符中包含的端点编号值（见第3.3.16节）。
- MUSBHDRC端点的RxMaxP寄存器必须以传输的最大数据包大小（字节）写入。此值应与目标终结点的标准终结点描述符的*wMaxPacketSize*字段相同。
- RxInterval寄存器需要写入NAK限制所需的值（2–215帧/微帧），或者如果不需要NAK超时功能，则设置为零。
- IntrRxE寄存器中的相关中断启用位应设置为“1”（如果此端点需要中断）
- RxCSR寄存器的以下位应设置如下：

D15	**自动清除**	0/1	如果需要自动清除功能，则设置为1。
D14	**AutoReq**	0/1	如果需要自动请求功能，请设置为1。
D13	**DMA ReqEnab**	0/1	如果此终结点需要DMA请求，请设置为1。*注意：*如果设置为1，还需要选择所选的DMAReqMode（RxCSRH.D3）。
D12	**DisNyet**	0	设置为0以允许正常的PING流量控制。

当首次配置端点时，应通过使用数据切换写入启用和数据切换位（RxCSRH.D2和D9）来切换当前设置，或通过写入RxCSR的较低字节来设置ClrDataTog位（D7），将端点数据切换设置为0。这将确保数据切换（由MUSBHDRC自动处理）以正确的状态开始。此外，如果FIFO中有任何数据包（由正在设置的RxBitRdy位（RxCSRL.D0）指示），则应通过设置FlushFIFO位（RxCSRL.D4）来刷新这些数据包。*注意：*如果启用了双缓冲，则可能需要连续两次设置此位。

22. 2. 1. 2. 操作程序

当USB外围设备需要批量数据时，CPU应在相应的RxCSR寄存器（D5）中设置ReqPkt位。然后，MUSBHDRC将向选定的外围端点发送IN令牌，并等待返回数据。

如果数据被正确地接收，则RxPktRdy（RxCSRL.D0）被设置。如果USB外围设备以STALL响应，则设置RxStall（RxCSRL.D6）。如果收到NAK，MUSBHDRC会再次尝试，并继续尝试，直到事务成功或达到RxInterval寄存器中设置的NAKLimit。如果根本没有接收到响应，则在MUSBHDRC报告错误（RxCSRL.D2集）之前再进行两次尝试。

然后，MUSBHDRC生成适当的端点中断，因此CPU应读取相应的RxCSR寄存器，以确定是否设置了RxPtRdy、RxStall、Error或NAK Timeout位，并相应地采取行动。（如果设置了NAK超时位，则可以指示MUSBHDRC通过清除NAK Timeout位继续尝试此事务（直到它再次超时），或者通过在清除NAK超时前清除ReqPkt中止事务。）

接收到的数据包不应超过RxMaxP[D0:D0]指定的大小（因为这应该是
发送到主机的端点描述符的*wMaxPacketSize*字段）。

当需要发送大于*wMaxPacketSize*的数据块时，它将作为多个数据包发送。所有数据包的大小都将为*wMaxPacketSize*，但最后一个包含剩余数据的数据包除外。如果要传输的块的大小是已知的，则可以将要传输的*wMaxPacketSize*的包的数量写入RqPktCount寄存器和AutoReq选项集。当每个数据包被请求时，RqPktCount寄存器中的值将递减。在RqPktCount从1递减到0时，AutoReq被清除以停止任何进一步的请求。或者，可以推断出

当接收到大小小于*wMaxPacketSize*的分组时，已经接收到整个块。（如果数据块的总大小是*wMaxPacketSize*的倍数，则最后一个数据包的大小可能仍然小于*wMaxPacketSize*，因为空数据包通常在数据之后发送，以表示传输完成。）在这种情况下，RqPktCount应保留为零。然后，如果设置了AutoReq，则当接收到短数据包时，它将自动清除。

上面描述了应用软件单独从FIFO读取每个数据包的一般情况。在配置核心时选择了批量数据包的自动组合选项时，行为略有不同。（这可以根据ConfigData寄存器的MPRxE位（D7）的设置来确定。）如果选择此选项，核心一次最多可以接收32个数据包，并将它们组合成FIFO内的单个数据包（假设FIFO足够大，可以接受这些较大的数据包）。写入FIFO的数据包大小由*m*表示◇ *wMaxPacketSize*，其中RxMaxP[D15:D11]=*m*-1。为了利用这一功能，应用软件所需要做的就是在RxMaxP寄存器中设置适当的值（并确保写入位10:0的值与端点描述符的*wMaxPacketSize*字段中给定的值匹配）。然后基于*m*的分组来计算诸如要在RqPktCount寄存器中设置的数目之类的值◇ *wMaxPacketSize*，就应用软件而言，传输这些较大数据包的过程与传输标准大小的Bulk数据包的流程没有什么不同。

如果正在传输大块数据，则可以通过使用DMA来避免调用中断服务例程来卸载每个数据包的开销。

22.2.1.3. 呃 ORHA and LING

如果目标想要关闭Bulk IN管道，它将向IN令牌发送STALL响应。这将导致RxStall位（RxCSRL.D6）被设置。

22.2.2. BULK OUT TRANSACTIONASAHOST

Bulk OUT事务可以用于将非周期性数据从主机传输到功能控制器。四个可选功能可用于在主机模

式下用于传输此数据的Tx端点：

- **双数据包缓冲**

除非使用动态FIFO大小，否则当写入TxMaxP寄存器的值小于或等于分配给端点的FIFO大小的一半时，将自动启用双数据包缓冲。（在选择动态FIFO大小的情况下，单包或双包缓冲的使用是端点FIFO规范的一部分——参见第8.4.1.2节）启用时，最多可以在FIFO中存储两个包，等待传输到外围设备。

- **DMA**

如果为端点启用了DMA，则只要端点能够接受其FIFO中的另一个数据包，就会生成DMA请求。此功能可用于允许DMA在无需处理器干预的情况下将数据包加载到FIFO中。

- **自动调压**

当启用自动设置功能时，当TxMaxP字节的数据包已加载到FIFO时，TxPktRdy位（TxCSRL.D0）将自动设置。当DMA用于加载FIFO时，这尤其有用，因为它避免了在大批量传输期间加载单个数据包时需要任何处理器干预。

- **自动数据包拆分**

对于一些系统设计，应用软件在单个操作中将比在单个USB操作中传输的数据量更大的数据量写入端点可能是方便的。一个特定的情况是，在某些情况下，相同的端点用于512字节的高速传输，但在其他情况下用于全速传输。当以全速操作时，单个操作中传输的最大数据量仅为64字节。为了适应这种情况，MUSBHDRC包括一个配置选项，如果选择该选项，则允许将较大的数据包写入批量端点，然后将批量端点拆分为适当（指定）大小的数据包，以便在USB总线上传输。是否选择此选项可以通过ConfigData寄存器的MPTxE位（D6）的设置来确定（见第3.3.5节）。

通过TxMaxP寄存器设置必要的数据包大小信息（见第3.3.7节）。

22.2.2.1.设置

在启动任何批量输出交易之前：

- 目标功能地址需要按照第8.5.1节所述进行设置。
- 要使用的MUSBHDRC端点的TxType寄存器需要写入位D7、D6以选择操作速度，位D5、D4=10（选择批量传输），位D3–D0设置为设备枚举期间返回给MUSBHDRR的OUT端点描述符中包含的端点编号值（见第3.3.14节）。
- MUSBHDRC端点的TxMaxP寄存器必须以传输的最大数据包大小（字节）写入。此值应与目标终结点的标准终结点描述符的*wMaxPacketSize字段*相同。
- TxInterval寄存器需要写入所需的NAK限制值（2–215帧/微帧），或者如果不需要NAK超时功能，则设置为零。
- IntrTxE寄存器中的相关中断启用位应设置为“1”（如果此端点需要中断）
- TxCSR寄存器的以下位应设置如下：

D15	**自动调压**	0/1	如果需要“自动设置”功能，请设置为1。
D13	**模式**	1.	设置为1以确保启用FIFO（仅当FIFO与Rx端点共享时才需要）。
D12	**DMA ReqEnab**	0/1	如果此终结点需要DMA请求，请设置为1。*注：*如果设置为1，还需要选择所选的DMAReqMode（TxCSRH.D2）。
D11	**FrcDataTog**	0	设置为0以允许正常的数据切换操作。

当首次配置端点时，应通过使用数据切换写入启用和数据切换位（TxCSRH.D1和D0）来切换当前设置，或通过写入TxCSR的较低字节来设置ClrDataTog位（D6），将端点数据切换设置为0。这将确保数据切换（由MUSBHDRC自动处理）以正确的状态开始。此外，如果FIFO中有任何数据包（由正在设置的FIFONotEmpty位（TxCSRL.D1）指示），则应通过设置FlushFIFO位（TxCSRL.D3）来刷新它们。*注意：*如果启用了双缓冲，则可能需要连续两次设置此位。

22.2.2.2.操作程序

当需要将批量数据发送到USB外围设备时，CPU应将数据的第一个数据包写入FIFO（如果是双缓冲的，则写入两个数据包），并在相应的TxCSR寄存器（D0）中设置TxPktRdy位。然后，MUSBHDRC将向选定的外围端点发送OUT令牌，然后发送来自FIFO的第一个数据包。

如果外围设备正确地接收到数据，则应当接收ACK，从而MUSBHDRC将清除TxPktRdy（TxCSRL.D0）。如果USB外围设备以STALL响应，则设置RxStall（TxCSRL.D5）。如果收到NAK，MUSBHDRC会再次尝试，并继续尝试，直到事务成功或达到TxInterval寄存器中设置的NAKLimit。如果根本没有接收到响应，则在MUSBHDRC报告错误（TxCSRL.D2集）之前再进行两次尝试。

然后，MUSBHDRC生成适当的端点中断，于是CPU应该读取相应的TxCSR寄存器，以确定RxStall（D5）、Error（D2）或NAK Timeout（D7）位是否被设置并相应地动作。（如果设置了NAK Timeout（NAK超时）位，则可以指示MUSBHDRC通过清除NAK超时位来继续尝试此事务（直到它再次超时），或者通过在清除NAK逾时位之前刷新FIFO来中止事务。）

在一般情况下，数据包大小不应超过TxMaxP寄存器的底部11位指定的大小（其应设置为与适当端点描述符的*wMaxPacketSize字段*中设置的值相匹配）。当需要发送大于TxMaxP的数据块时，需要将其作为多个数据包发送——每个数据包都如上所述发送。这些数据包的大小都应该是TxMaxP[D0:D0]，除了最后一个包含残差的数据包。

此规则的例外情况适用于在配置core时选择了自动批量数据包拆分选项的情况。（这可以根据ConfigData寄存器的MPTxE位（D6）的设置来确定。）在选择此选项的情况下，可以将大小高达TxMaxP[D0:D0]指定的32倍的数据包写入FIFO（假设FIFO足够大，可以接受这些较大的数据包），然后由核心将其拆分为适当大小的数据包，以便通过USB传输。写入FIFO的数据包大小由*m*表示◈ *USB有效载荷*，其中TxMaxP[D15:D11]=*m*-1。应用软件利用这一功能所需要做的一切就是在TxMaxP寄存器中设置适当的值。就应用软件而言，传输这些较大数据包的过程与用于传输标准大小的Bulk数据包的流程没有什么不同。

如果数据块的总大小是TxMaxP的倍数，则主机可能需要在发送完所有数据之后发送空分组。这可以通过在接收到最后一个中断之后设置TxPktRdy来实现，而无需将任何数据加载到FIFO中。

如果正在传输大块数据，那么可以通过使用DMA来避免调用中断服务例程来加载每个数据包的开销。

22.2.2.3. 呃 O R H A and LI N G

如果目标想要关闭Bulk OUT管道，它将发送STALL响应。这由正在设置的RxStall位（TxCSRL.D5）来指示。

22.3. 就业DM A

采用DMA的优点是在加载或卸载FIFO时提高了总线和处理器的利用率。

DMA可以与任何类型的传输结合使用，但当要通过批量端点传输大块数据时，它特别有用。USB协议要求通过发送端点的最大数据包大小的一系列数据包（512字节用于高速，64字节用于全速）来传输大数据块。该系列中的最后一个分组可以小于最大分组大小。事实上，接收器可以使用该"短"分组的接收来用信号通知传输的结束（如果数据块的大小是最大分组大小的精确倍数，则可以在序列的末尾发送空分组）。

MUSBHDRC的DMA设施可以在外围模式或主机模式中使用，以避免在每个单独的数据包之后必须中断处理器的开销，而只是在传输完成之后才中断处理器。

以下部分概述了使用DMA以及一些标准类型的批量传输和批量接收传输所涉及的基本操作。这些动作可以使用内置DMA控制器（在核心中实现）或使用外部DMA控制器来执行。

22.3.1. U S ING D M A WITH BUL K T X endpoint S

对于Tx端点，当端点FIFO能够接受数据包时，DMA请求行变高，而当TxMaxP字节已加载到FIFO时，DMA要求行变低。或者，当TxCSR中的TxPktRdy位被设置时，请求线将变低。

要使用DMA通过Bulk Tx端点向USB主机发送大块数据，我们建议按如下方式设置DMA控制器和MUSBHDRC。

DMA控制器应该被编程为当端点的DMA请求线从低转换到高时执行端点的最大分组大小的突发DMA读取（高速为512字节，全速为64字节）。MUSBHDRC产品规范第17节中给出了内置DMA控制器的设置细节。控制器应继续对每个DMA请求执行这些突发读取，直到整个数据块已被传输。（然而，最后的突发可以小于最大分组大小）。然后它应该会中断CPU。

应通过设置TxCSR寄存器中的AutoSet、DMAReqEnab和DMAReqMode位（分别为位D15、D12和D10），将MUSBHDRC编程为启用AutoSet和DMA请求模式1。

如此编程，只要其FIFO中有空间接受数据包，MUSBHDRC就会将DMA请求线设为高电平。此外，在DMA控制器已经用最大分组大小的分组加载FIFO之后，TxPktRdy比特将被自动设置。然后，数据包就可以发送到主机了。当最后一个数据包已经由DMA控制器加载时，

控制器应当中断处理器。（内置控制器通过断言DMA_NINT来实现这一点。）如果最后加载的数据包小于最大数据包大小，则TxPktRdy位将没有被设置，因此需要手动设置（即由CPU）以允许发送最后的数据包。如果最后一个数据包具有最大数据包大小并且要发送空数据包以指示传输结束，则还需要手动设置TxPktRdy比特。

注：如果在主机模式下操作时，核心三次未能成功传输数据包，则TxCSR寄存器（TxCSR.D2）中的错误位将被设置，DMA请求行将被禁用，直到该错误位再次被清除。还应注意的是，TxCSR寄存器中的DMAReqMode位不得在相应DMAReqEnab位被清除之前或在相同周期内被清除。

22.3.2. USING DMA WITH BULK RX endpoints

Rx端点的DMA请求行的行为取决于通过RxCSR寄存器（D11）选择的DMA请求模式。在DMA请求模式0中，当数据包在端点FIFO中可用时，Rx DMA请求行变高，而当数据包的最后一个字节已被读取时，或当RxCSR中的RxPixRdy位被清除时，Rx-DMA请求行则变低。
在DMA请求模式1中，只有当接收到的数据包具有最大数据包大小（如在RxMaxP寄存器中设置的）时，DMA请求行才会变高。如果接收到的数据包具有某种其他大小，则DMA请求行保持低电平，结果数据包保留在设置了RxPktRdy的FIFO中。这将导致生成Rx端点中断（如果启用）。

DMA请求模式主要设计用于将大数据包传输到批量端点的情况。USB协议要求将这样的分组划分为一系列最大分组大小的分组（高速为512字节，全速为64字节）。序列中的最后一个分组可能小于最大分组大小（或者，如果传输的总大小是最大分组大小的精确倍数，则为空分组），并且接收器可以将此“短”分组解释为传送结束的信号。DMA请求模式1可以与适当编程的DMA控制器一起使用，以避免在每个单独的数据包之后必须中断处理器的开销，而只是在传输完成后中断处理器。
*注意：*如果请求模式从请求模式1切换到请求模式0，则如果FIFO中有数据包，则会断言请求行，以便下载此“预接收”数据包。

22.3.3 例外

以下部分描述了我们建议在接收大数据块时使用的设置——首先是在数据块的大小事先已知的情况下，然后是在该块的大小未知的情况下。*注：*一种情况使用MUSBHDRC内核的DMA请求模式0，而另一种情况则使用DMA请求模式1，但这*两种*操作都是使用内置DMA控制器的DMA模式1执行的。

22.3.3.1. CASE 1：经验的大小

如果要从USB主机接收的大数据块的大小在通过Bulk Rx端点发送之前是已知的，我们建议按如下方式设置DMA控制器和MUSBHDRC：

DMA控制器应该被编程为具有要传输的块的大小，并且当用于端点的DMA请求线从低转换到高时执行用于端点的包的最大大小的突发DMA写入（用于高速的512字节，用于全速的64字节）。*MUSBHDRC产品规范第17.4.3节中给出了内置DMA控制器的设置细节。*

MUSBHDRC应通过设置RxCSR寄存器（D13）中的DMAReqEnab位并确保DMAReqMode位（D11）为空，为DMA请求模式0编程。由于DMA请求模式0单独处理每个数据包，因此在这种情况下也建议选择自动清除（通过设置RxCSR.D15）。

这样编程，每当MUSBHDRC从主机接收到数据包时，它就会将DMA请求线设置为高电平，因此DMA控制器应该执行数据到存储器的突发写入。当DMA控制器已经从FIFO读取了最大数据包大小的数据包时，RxBitRdy位被自动清除。DMA控制器应该在每个DMA请求上继续执行这些突发写入，直到整个数据块已经被传输（最后一个突发可能小于最大分组大小）。

当DMA控制器读取了最后一个数据包时，它应该中断处理器。（内置DMA控制器通过断言DMA_NINT来实现这一点。）CPU对此中断的响应将取决于上次读取的数据包是否达到最大值

数据包大小与否。如果它小于最大数据包大小，RxPktRdy位将不会被清除，需要手动清除（即由CPU清除）。

22.3.3.2. CASE 2：经验数据的大小BLOCK NOT KNOWN

在要从USB主机接收的大数据块的大小未知的情况下，识别块的末尾的传统方法是通过发现接收到的分组小于最大分组大小。对于此操作，我们建议按如下方式设置DMA控制器和MUSBHDRC：

DMA控制器应该被编程为当端点的DMA请求线从低转换到高时执行对端点的最大分组大小的存储器的突发写入（高速为512字节，全速为64字节）。*MUSBHDRC产品规范第17.4.3节中给出了内置DMA控制器的设置细节。*

应通过设置RxCSR寄存器中的AutoClear、DMAReqEnab和DMAReqMode位（分别为位D15、D13和D11），将MUSBHDRC编程为启用AutoClear和DMA Request Mode 1，并应启用端点的中断。如果MUSBHDRC在主机模式下运行，还应选择AutoReq（RxCSR.D14）。

以这种方式编程，无论何时从主机接收到最大数据包大小的数据包，MUSBHDRC都会将DMA请求线设置为高电平（但不会产生任何中断）。当DMA控制器已从FIFO读取数据包时，RxBitRdy位将自动清除。当接收到小于最大数据包大小的数据包时，将不会生成DMA请求，因此数据包将保留在设置了RxPktRdy的FIFO中。这将导致MUSBHDRC生成相应的端点中断。在接收到此中断时，CPU应读取端点的RxCount寄存器以确定数据包的大小，然后手动读取此短数据包或重新编程DMA控制器以读取此数据包。RxPtRdy将需要手动清除（即由CPU清除）。

23. 全速度/低带宽H中断事务

23.1.在未来时期

中断IN事务使用与批量IN事务相同的协议（如第22.1.1节所述），并且可以以相同的方式使用。类似地，中断OUT事务使用与批量OUT事务几乎相同的协议（如第22.2.2节所述），并且可以以相同的方式使用。

但是，您应该注意，用作外围设备的MUSBHDRC上的Tx端点支持中断IN事务的一个功能，而这些功能在批量IN事务中不支持，因为它们支持数据切换位的连续切换。此功能可通过设置TxCSR寄存器（D11）中的FrcDataTog位来启用。当此位设置为"1"时，无论是否从主机接收到ACK，MUSBHDRC都将认为数据包已成功发送，并切换端点的数据位。

另一个区别是中断端点不支持PING流控制。这意味着MUSBHDRC不应以NYET握手进行响应，而应仅以ACK/NAK/STALL进行响应。为了确保这一点，RxCSR寄存器（D12）中的DisNyet位应设置为"1"，以禁止在高速模式下传输NYET握手。

虽然DMA可以与中断OUT端点一起使用，但它通常没有什么好处，因为中断端点通常被期望在单个数据包中传输所有数据。

23.2.中断TRANSACTIONS AS A HOST

当MUSBHDRC作为主机运行时，与USB外围设备上的中断端点的交互处理方式与等效批量事务（分别在第22.2.1节和第22.2.2节中描述）非常相似，但支持高带宽中断事务。

就操作步骤而言，主要区别在于RxType[5:4]和TxType[5:4需要设置为11（到

表示中断事务）而不是10。

还需要在RxInterval/TxInterval寄存器中设置所需的轮询间隔（见第3.3.17节和第3.3.15节）。

24. 全速/低带宽等时事务

24.1. HANDLING等时TRANSACTION离子ASA外围AL

24.1.1. 我是CHRONOUS IN TRANSACTIONS

Isochronous IN事务用于将周期性数据从功能控制器传输到主机。本节介绍全速等时传输端点和低带宽（每微帧1个数据包）高速等时传输终端在外围模式下的使用。高带宽高速（>8Mbps）端点将在后面的部分中进行描述。

三个可选功能可用于在外围模式下使用的Tx端点，用于等时in事务：

- **双数据包缓冲**

除非使用动态FIFO大小，否则当写入TxMaxP寄存器的值小于或等于分配给端点的FIFO大小的一半时，会自动启用双数据包缓冲。（在选择动态FIFO大小的情况下，单包或双包缓冲的使用是端点FIFO规范的一部分——参见第8.4.1.2节）启用时，最多可以在FIFO中存储两个包，等待传输到主机。*注意：*双数据包缓冲通常适用于等时事务，以避免运行不足错误（请参阅下面的"操作"）。

- **DMA**

如果为端点启用了DMA，则只要端点能够接受其FIFO中的另一个数据包，就会生成DMA请求。此功能可用于允许DMA控制器在无需处理器干预的情况下将数据包加载到FIFO中。
然而，这一功能对等时端点并不特别有用，因为传输的数据包通常不是最大数据包大小，并且需要在每个数据包之后访问TxCSR寄存器，以检查欠运行错误。

- **自动调压**

当为低带宽等时端点启用AutoSet功能时，当TxMaxP字节的数据包已加载到FIFO时，TxPktRdy位（TxCSRL.D0）将自动设置。然而，这一功能对等时端点并不特别有用，因为传输的数据包通常不是最大数据包大小，并且需要在每个数据包之后访问TxCSR寄存器，以检查欠运行错误。

24.1.1.1.设置

在为等时In事务配置Tx端点时，必须使用端点的最大数据包大小（以字节为单位）写入TxMaxP寄存器。此值应与终结点的标准终结点描述符的*wMaxPacketSize*字段相同。此外，IntrTxE寄存器中的相关中断启用位应设置为1（如果此端点需要中断），TxCSR寄存器的高字节应设置为如下所示（D9–D8位未使用）：

D15	**自动调压**	0/1	如果需要"自动设置"功能，请设置为1。
D14	**国际标准化组织**	1.	设置为1以启用Isochronous协议。
D13	**模式**	1.	设置为1以确保启用FIFO（仅当FIFO与Rx端点共享时才需要）。
D12	**DMA ReqEnab**	0/1	如果此终结点需要DMA请求，请设置为1。*注：*如果设置为1，还需要选择所选的DMAReqMode（TxCSRH.D2）。
D11	**FrcDataTog**	0	在等时模式下被忽略。

24.1. 1. 2. 操作程序

Isochronous端点不支持数据重试，因此，如果要避免运行中的数据，则必须在接收IN令牌之前将要发送到主机的数据加载到FIFO中。主机将每帧（或高速模式下的微帧）发送一个IN令牌，但是帧（或微帧）内的定时可能会变化。如果在一帧结束时收到IN令牌，然后在下一帧开始时收到，则几乎没有时间重新加载FIFO。出于这个原因，端点的双重缓冲通常是必要的。

AutoSet功能可以与低带宽Isochronous Tx端点一起使用，方法与批量Tx端点相同。然而，除非数据以与主机的帧时钟同步的绝对一致的速率从源到达，否则发送到主机的分组的大小将不得不从帧到帧（或从微帧到微帧）增加或减少，以匹配源数据速率。这意味着实际的数据包大小并不总是TxMaxP大小，从而使AutoSet功能变得无用。

每当数据包被发送到主机时都会产生中断，并且软件可以使用该中断将下一个数据包加载到FIFO中，并以与批量Tx端点相同的方式在TxCSR寄存器（D0）中设置TxPktRdy位。由于中断几乎可以在一帧（/微帧）内的任何时间发生，这取决于主机何时调度事务，这可能导致FIFO加载请求的定时不规则。如果端点的数据源来自某些外部硬件，则在加载FIFO之前等待每个帧（/微帧）结束可能会更方便，因为这将最大限度地减少对额外缓冲的需求。这可以通过使用SOF中断（IntrUSB.D3）或来自MUSBHDRC的外部SOF_PULSE信号来触发下一个数据包的加载。当接收到SOF分组时，每帧（/微帧）生成一次SOF_PULSE。（MUSBHDRC还维护一个外部帧（/微帧）计数器，因此当SOF数据包丢失时，它仍然可以生成SOF_PULSE。）中断仍可用于设置TxCSR（D0）中的TxPktRdy位，并检查数据是否超支/运行不足（请参阅下面的"错误处理"）。

启动双缓冲等时IN管道可能是问题的根源。双缓冲要求数据包在加载后的帧（/微帧）之前不传输。如果函数在主机设置管道（并因此开始发送IN令牌）之前至少一帧（/微帧）加载第一个数据包，则没有问题。但是，如果主机在加载第一个数据包时已经开始发送IN令牌，则数据包可以在与加载时相同的帧（/微帧）中发送，这取决于它是在接收到IN令牌之前还是之后加载的。这个潜在的问题可以通过在功率寄存器（D7）中设置ISO更新位来避免。当该位被设置为1时，加载到等时Tx端点FIFO中的任何数据包将不会被发送，直到接收到下一个SOF包之后，从而确保数据包不会发送得太早。

24. 1. 1. 3. 呃 O R H A and LI N G

当接收到in令牌时，如果端点的FIFO中没有数据，它将向主机发送一个空数据包，并在TxCSR寄存器中设置欠运行位（D2）。这表明软件向主机提供数据的速度不够快。由应用程序决定如何处理这种错误情况。

如果软件正在每帧（/微帧）加载一个数据包，并且它发现TxCSR寄存器（D0）中的TxPktRdy位在它想要加载下一个数据包时被设置，这表明数据包没有被发送（可能是因为来自主机的in令牌被破坏）。这取决于应用程序如何处理这种情况：它可以选择通过设置TxCSR寄存器（D3）中的FlushFIFO位来刷新未发送的数据包，也可以选择跳过当前数据包。

2 4 . 1.2. I S O CHRO NOU S OUT T R A N S A CT I O N S输入输出

Isochronous OUT事务用于将周期性数据从主机传输到功能控制器。本节介绍全速等时接收端点和低带宽（每微帧1个数据包）高速等时接收端在外围模式下的使用。高带宽高速（>8 Mbps）端点将在后面的部分中进行描述

三个可选功能可用于在外围模式下用于等时OUT事务的Rx端点：

- **双数据包缓冲**

除非使用动态FIFO大小，否则当写入RxMaxP寄存器的值小于或等于分配给端点的FIFO大小的一半时，会自动启用双数据包缓冲。（在选择动态FIFO大小的情况下，单包或双包缓冲的使用是端点FIFO规范的一部分——参见第8.4.2.2节）启用时，最多可以在FIFO中存储两个包，等待传输到主机。注：双数据包缓冲

一般建议用于等时交易，以避免超限错误（请参阅下面的“操作”）。

□ DMA

如果为端点启用了DMA，则每当端点的FIFO中有数据包时，都会生成DMA请求。此功能可用于允许DMA控制器在无需处理器干预的情况下从FIFO卸载数据包。然而，这一功能对等时端点并不特别有用，因为传输的数据包通常不是最大数据包大小，并且需要在每个数据包之后访问RxCSR寄存器，以检查Overrun或CRC错误。

□ 自动清除

当启用自动清除功能时，当从FIFO卸载RxMaxP字节的数据包时，RxPktRdy位（RxCSRL.D0）将自动清除（例外情况，请参阅寄存器描述）。然而，这一功能对等时端点并不特别有用，因为传输的数据包通常不是最大数据包大小，并且需要在每个数据包之后访问RxCSR寄存器，以检查Overrun或CRC错误。

24. 1. 2. 1.设置

在为等时OUT事务配置Rx端点时，必须使用端点的最大数据包大小（以字节为单位）写入RxMaxP寄存器。此值应与终结点的标准终结点描述符的*wMaxPacketSize*字段相同。此外，IntrRxE寄存器中的相关中断启用位应设置为1（如果此端点需要中断），RxCSR寄存器的高字节应设置为如下所示（位D10–D8未使用/只读）：

D15	**自动清除**	0/1	如果需要自动清除功能，则设置为1。
D14	**国际标准化组织**	1.	设置为1以启用Isochronous协议。
D13	**DMA ReqEnab**	0/1	如果此终结点需要DMA请求，请设置为1。注意：如果设置为1，还需要选择所选的DMAReqMode（RxCSRH.D3）。
D12	**DisNyet**	0	在等时模式下被忽略。

24.1. 2. 2. 操作程序

Isochronous端点不支持数据重试，因此，如果要避免数据溢出，则在接收到数据包时，FIFO中必须有空间接受数据包。主机将每帧（或高速模式下的微帧）发送一个数据包，但帧内的时间可能会有所不同。如果在一帧（/微帧）的末尾附近接收到一个数据包，而另一个在下一帧的开头到达，则几乎没有时间卸载FIFO。出于这个原因，端点的双重缓冲通常是必要的。

自动清除功能可以与等时接收端点一起使用，与批量接收端点相同。然而，除非数据宿以绝对一致的速率接收数据并与主机的帧时钟同步，否则从主机发送的分组的大小将不得不从帧到帧（或从微帧到微帧）增加或减少，以匹配所需的数据速率。这意味着实际数据包的大小并不总是RxMaxP，从而使自动清除功能变得无用。

每当从主机接收到数据包时都会产生中断，并且软件可以使用此中断从FIFO卸载数据包，并以与Bulk Rx端点相同的方式清除RxCSR寄存器（D0）中的RxPtRdy位。由于中断几乎可以在一帧（/微帧）内的任何时间发生，这取决于主机何时调度事务，FIFO卸载请求的时间可能不规则。如果端点的数据接收器要连接到一些外部硬件，则最好在卸载FIFO之前等待每个帧（/微帧）结束，从而最大限度地减少对额外缓冲的要求。这可以通过使用SOF中断（IntrUSB.D3）或来自MUSBHDRC的外部SOF_PULSE信号来触发数据包的卸载来实现。当接收到SOF分组时，每帧（/微帧）生成一次SOF_PULSE。（MUSBHDRC还维护一个外部帧（/微帧）计数器，因此当SOF数据包丢失时，它仍然可以生成SOF_PULSE。）中断仍可用于清除RxCSR中的RxPktRdy位，并检查数据是否超支/运行不足（请参阅下面的“错误处理”）。

24. 1. 2. 3. 呃 O R H A and LI N G

当从主机接收数据包时，如果FIFO中没有空间存储数据包，则RxCSR寄存器（D2）中的OverRun位将被设置。这表明软件卸载数据的速度不够快。由应用程序决定

确定如何处理此错误条件。

如果MUSBHDRC发现接收到的数据包有CRC错误，它仍将数据包存储在FIFO中，并设置RxKtRdy位（RxCSRL.D0）和DataError位（RxCSRL.D3）。如何处理这种错误情况取决于应用程序。

24.2. HANDLING等时TRANSACTIONS AS A主机

24.2.1. 我是CHRONOUS IN TRANSACTIONS

Isochronous IN事务用于将周期性数据从功能控制器传输到主机。本节介绍全速等时接收端点和低带宽（每微帧1个数据包）高速等时接收端在主机模式下的使用。高带宽高速（>8Mbps）端点将在后面的部分中进行描述。

四个可选功能可用于在主机模式下用于接收此数据的Rx端点：

- **双数据包缓冲**

当启用双数据包缓冲时，可以在读取另一个数据包时接收一个数据。除非使用动态FIFO大小，否则当写入RxMaxP寄存器的值小于或等于分配给端点的FIFO大小的一半时，将自动启用双数据包缓冲。（在选择动态FIFO大小的情况下，单包或双包缓冲的使用是端点FIFO规范的一部分——见第8.4.1.2节）。*注*：双数据包缓冲通常适用于等时事务，以避免数据溢出（请参阅下面的"操作"）。

- **DMA**

如果为端点启用了DMA，则每当端点的FIFO中有数据包时，都会生成DMA请求。此功能可用于允许DMA控制器在无需处理器干预的情况下从FIFO卸载数据包。然而，此功能对于等时端点并不是特别有用，因为传输的数据包通常不是最大数据包大小。

- **自动清除**

当启用自动清除功能时，当从FIFO卸载RxMaxP字节的数据包时，RxPktRdy位（RxCSRL.D0）将自动清除（例外情况，请参阅寄存器描述）。然而，这一功能对等时端点并不特别有用，因为传输的数据包通常不是最大数据包大小，并且需要在每个数据包之后访问RxCSR寄存器，以检查Overrun或CRC错误。

- **自动请求（uest）**

当AutoReq（uest）功能被启用时，当RxKtRdy位被清除时，ReqPkt位（RxCSRL.D5）将被自动设置。

24.2.1.1.设置

在启动等时IN事务之前：

- 目标功能地址需要按照第8.5.1节所述进行设置。
- 要使用的MUSBHDRC端点的RxType寄存器需要写入位D7、D6以选择操作速度，位D5、D4=01（选择等时传输），位D3–D0设置为端点值
  在设备枚举期间返回给MUSBHDRC的in端点描述符中包含的数字（见第3.3.16节）。
- MUSBHDRC端点的RxInterval寄存器需要按照所需的事务间隔（通常每帧/微帧一个事务）写入——请参阅第3.3.17节。
- MUSBHDRC端点的RxMaxP寄存器必须以传输的最大数据包大小（字节）写入。此值应与目标终结点的标准终结点描述符的*wMaxPacketSize*字段相同。
- IntrRxE寄存器中的相关中断启用位应设置为"1"（如果此端点需要中断）
- RxCSR寄存器的以下位应设置如下：

D15	**自动清除**	0/1	如果需要自动清除功能，则设置为1。
D14	**AutoReq**	0/1	如果需要自动请求功能，请设置为1。
D13	**DMA ReqEnab**	0/1	如果此终结点需要DMA请求，请设置为1。注意：如果设置为1，还需要选择所选的DMAReqMode（RxCSRH.D3）。
D12	**DisNyet**	0	在等时模式下被忽略。

24.2.1.2. 操作程序

操作从CPU设置ReqPkt（RxCSRL.D5）开始。这导致MUSBHDRC向目标发送IN令牌。

当接收到数据包时，生成中断，软件可以使用该中断从FIFO卸载数据包，并以与Bulk Rx端点相同的方式清除RxCSR寄存器（D0）中的RxPtRdy位。由于中断几乎可以在一帧（/微帧）内的任何时间发生，FIFO卸载请求的定时可能是不规则的。如果端点的数据接收器要连接到一些外部硬件，则最好在卸载FIFO之前等待到每个帧结束，从而最大限度地减少对额外缓冲的要求。这可以通过使用来自MUSBHDRC的SOF_PULSE信号来触发数据包的卸载来实现。SOF_PULSE每帧（/微帧）生成一次。中断仍然可以用于清除RxCSR中的RxPtRdy比特。

自动清除功能可以与等时接收端点一起使用，与批量接收端点相同。然而，除非数据宿以绝对一致的速率接收数据并与MUSBHDRC的帧时钟同步，否则分组的大小将从一帧到另一帧（或从微帧到微帧）增加或减少，以匹配所需的数据速率。这意味着实际数据包的大小并不总是RxMaxP，从而使自动清除功能变得无用。

24.2.1.3. 呃 ORHA and LING

如果在接收数据包的过程中发生CRC或比特填充错误，则数据包仍将被存储在FIFO中，但DataError比特（RxCSRL.D3）被设置为指示数据可能被破坏。

24.2.2. ISOCHRONOUS OUT TRANSACTIONS输入输出

Isochronous OUT事务用于将周期性数据从主机传输到功能控制器。本节介绍全速等时传输端点和低带宽（每微帧1个数据包）高速等时传输终端的使用。高带宽高速（>8Mbps）端点将在后面的部分中进行描述。

三个可选功能可用于在主机模式下用于传输此数据的Tx端点：

- **双数据包缓冲**

除非使用动态FIFO大小，否则当写入TxMaxP寄存器的值小于或等于分配给端点的FIFO大小的一半时，会自动启用双数据包缓冲。（在选择动态FIFO大小的情况下，单包或双包缓冲的使用是端点FIFO规范的一部分——见第8.4.2.2节。）。启用时，最多可将两个数据包存储在FIFO中，等待传输到外围设备。

- **DMA**

如果为端点启用了DMA，则只要端点能够接受其FIFO中的另一个数据包，就会生成DMA请求。然而，此功能对于等时端点并不是特别有用，因为传输的数据包通常不是最大数据包大小。

- **自动调压**

当使用低带宽等时端点启用AutoSet功能时，当TxMaxP字节的数据包已加载到FIFO时，TxPktRdy位（TxCSRL.D0）将自动设置。然而，此功能对于等时端点并不是特别有用，因为传输的数据包通常不是最大数据包大小。

24. 2. 2. 1.设置

在启动等时OUT事务之前：

- 目标功能地址需要按照第8.5.1节所述进行设置。
- 要使用的MUSBHDRC端点的TxType寄存器需要写入位D7、D6以选择操作速度，位D5、D4=01（选择等时传输），位D3–D0设置为端点值
  在设备枚举期间返回给MUSBHDRC的OUT端点描述符中包含的数字（见第3.3.14节）。
- MUSBHDRC端点的TxInterval寄存器需要按照所需的事务间隔写入（通常每帧/微帧一个事务）——请参阅第3.3.15节。
- MUSBHDRC端点的TxMaxP寄存器必须以传输的最大数据包大小（字节）写入。此值应与目标终结点的标准终结点描述符的*wMaxPacketSize*字段相同。
- IntrTxE寄存器中的相关中断启用位应设置为"1"（如果此端点需要中断）
- TxCSR寄存器的以下位应设置如下：

D15	**自动调压**	0/1	如果需要"自动设置"功能，请设置为1。
D13	**模式**	1.	设置为1以确保启用FIFO（仅当FIFO与Rx端点共享时才需要）。
D12	**DMA ReqEnab**	0/1	如果此终结点需要DMA请求，请设置为1。*注*：如果设置为1，还需要选择所选的DMAReqMode（TxCSRH.D2）。
D11	**FrcDataTog**	0	在等时模式下被忽略。

24. 2. 2. 2. 操作程序

当CPU写入FIFO然后设置TxPktRdy（TxCSRL.D0）时，操作开始。这触发MUSBHDRC发送OUT令牌，然后发送来自FIFO的第一个数据包。

每当发送数据包时都会产生中断，并且软件可以使用该中断将下一个数据包加载到FIFO中，并以与批量Tx端点相同的方式在TxCSR寄存器（D0）中设置TxPktRdy位。由于中断几乎可以在一帧内的任何时间发生，这取决于主机何时调度事务，这可能导致FIFO加载请求的定时不规则。如果端点的数据源来自某些外部硬件，则在加载FIFO之前等待到每帧结束可能会更方便，因为这将最大限度地减少对额外缓冲的要求。这可以通过使用来自MUSBHDRC的SOF_PULSE信号来触发下一个数据包的加载来实现。SOF_PULSE每帧（/微帧）生成一次。中断仍然可以用于设置TxCSR中的TxPktRdy比特。

AutoSet功能可以与低带宽Isochronous Tx端点一起使用，方法与批量Tx端点相同。然而，除非数据以与MUSBHDRC的帧时钟同步的绝对一致的速率从源到达，否则数据包的大小将从一帧到另一帧（或从微帧到微帧）增加或减少，以匹配源数据速率。这意味着实际的数据包大小并不总是TxMaxP大小，从而使AutoSet功能变得无用。

25. 高带宽 S O C HRO无US/中断事务

高带宽等时/中断事务使用与其他等时/断事务几乎相同的协议。然而，进行高带宽事务也有一些特殊功能。

1. 只有在MUSBHDRC内核已配置为支持这些事务的情况下，才能进行高带宽等时/中断事务（请参阅《MUBMHDRC用户指南》第3节）。

还需要对核心进行配置，以便用于高带宽事务的端点具有足够大小的FIFO，以保存至少一个高带宽数据包（超过1K字节，最多3K字节）的数据。

2. 当在TxMaxP/RxMaxP寄存器中设置端点处理的最大数据包大小时，还需要通过该寄存器的位D11和D12设置每个微帧的最大事务数。

此最大事务数（2或3）还表示任何单个“高带宽”数据包可以在其中传输的最大段数，这反过来又将数据包的最大大小设置为同一寄存器中为端点指定的最大有效载荷的2或3倍。

注意：在任何事务中可以发送的最大有效负载为1Kbyte。

3. 发送数据包时，TxPktRdy需要由应用软件进行设置。类似地，当从Rx端点FIFO卸载数据包时，应用软件需要清除RxPktRdy。

AutoSet和AutoClear函数不能用于在高带宽事务中设置和清除这些位。

4. 作为多个部分的数据包传输引入了另一种类型的错误——不完整数据包的传输。

对于Tx端点，该问题主要适用于MUSBHDRC处于外围模式时，并且发生在MUSBHDRR未能从主机接收到足够的in令牌以发送数据包的所有部分时。它也可以应用于主机模式下的高带宽中断事务，在主机模式下，核心不会从发送数据包的设备接收到任何响应。在这两种情况下，MUSBHDRC都将设置TxCSR寄存器（D7）中的IncompTx位。

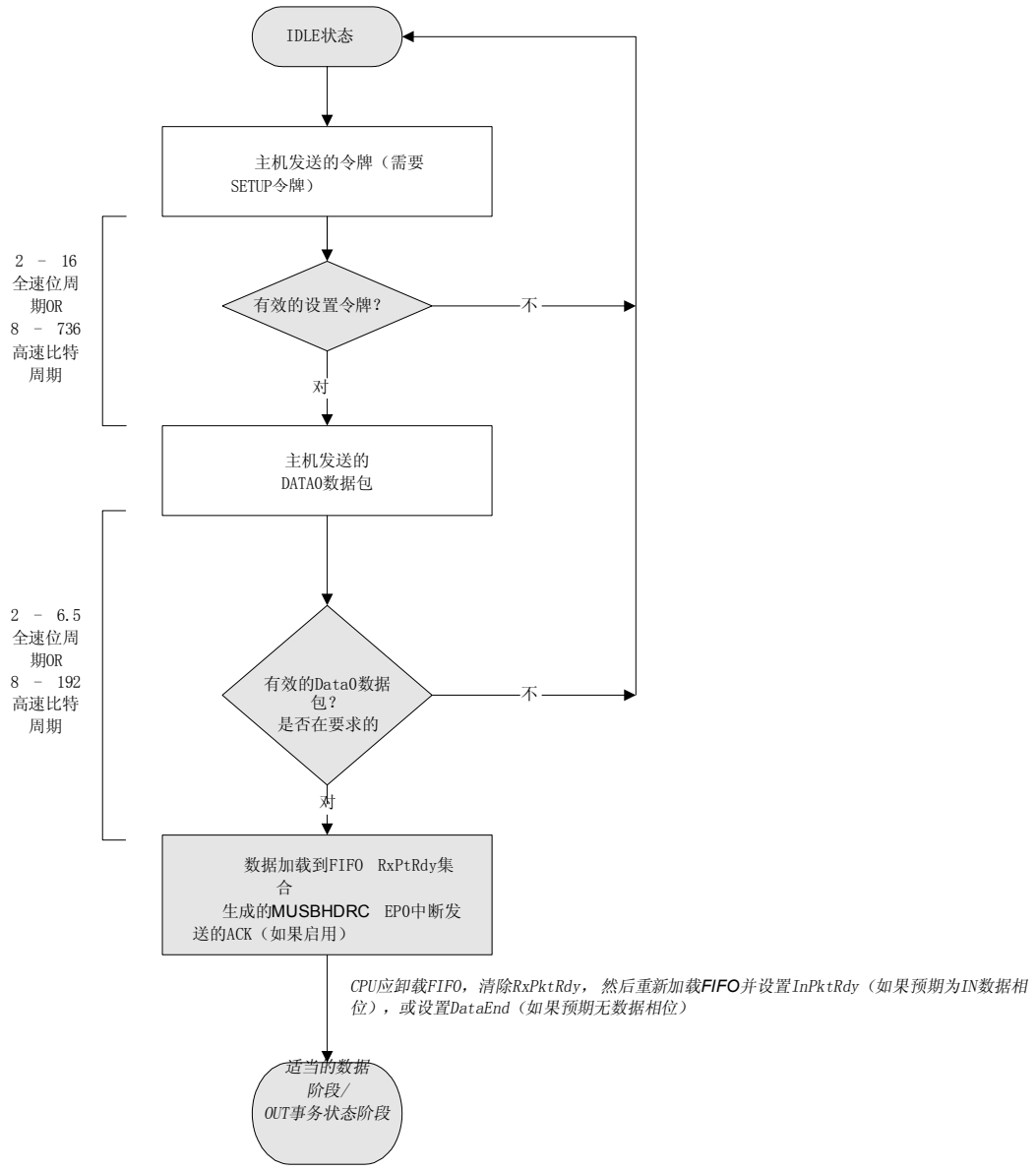
对于Rx端点，当数据包的已接收部分的PID显示尚未接收到数据包的一个或多个部分时，就会出现问题。当这种情况发生时，MUSBHDRC会在RxCSR寄存器（D8）中设置IncompRx位。总的来说，只有当核心在外围模式下操作时才会设置此位，但如果（且仅当）与MUSBHDRC通信的设备未能根据USB协议做出响应，则也可以在主机模式下设置此位。

26. 事务作为外围设备流动

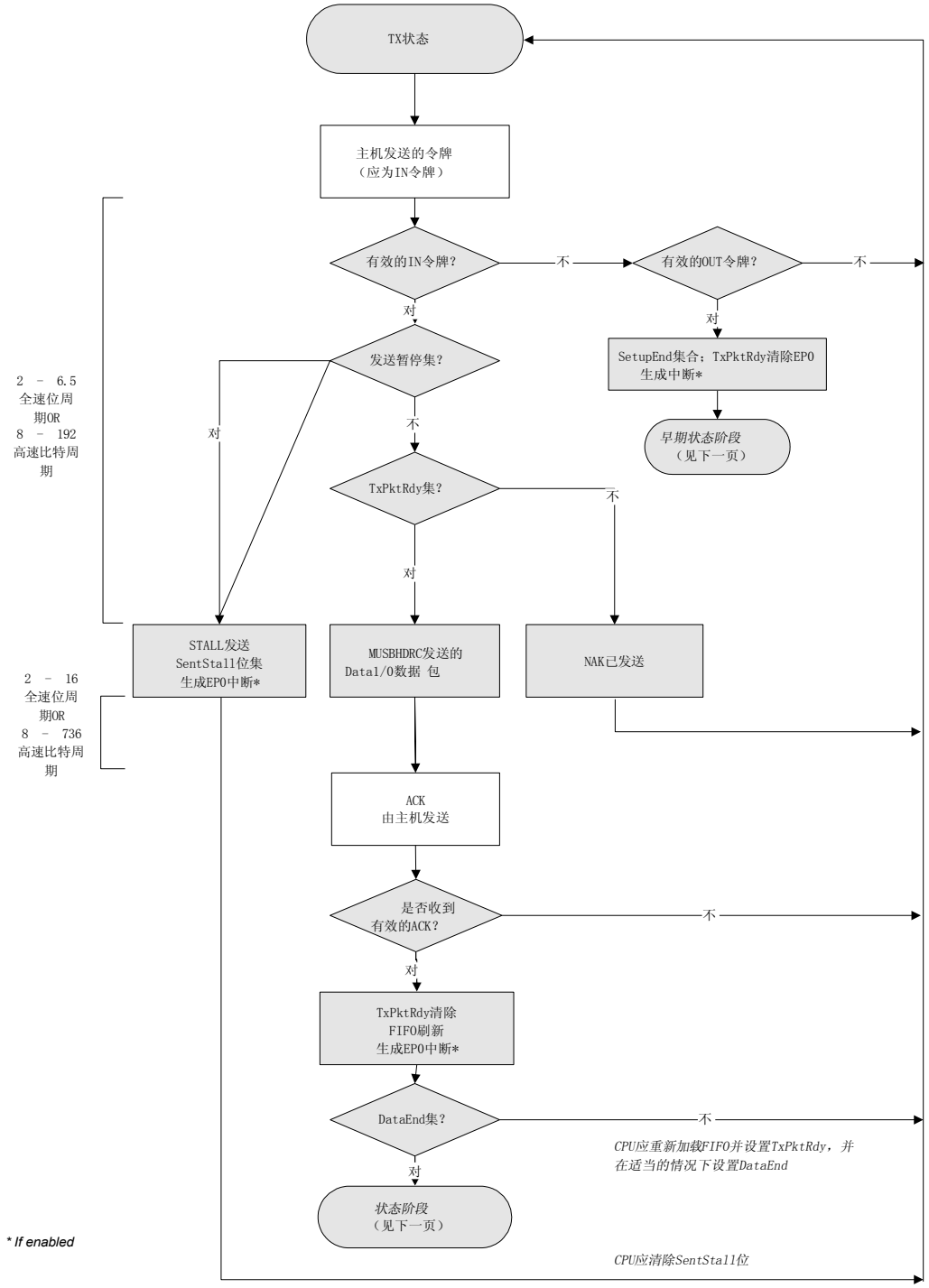
注意：主机操作显示在白色背景下。MUSBHDRC动作以阴影显示。

26.1. 控制角色任务

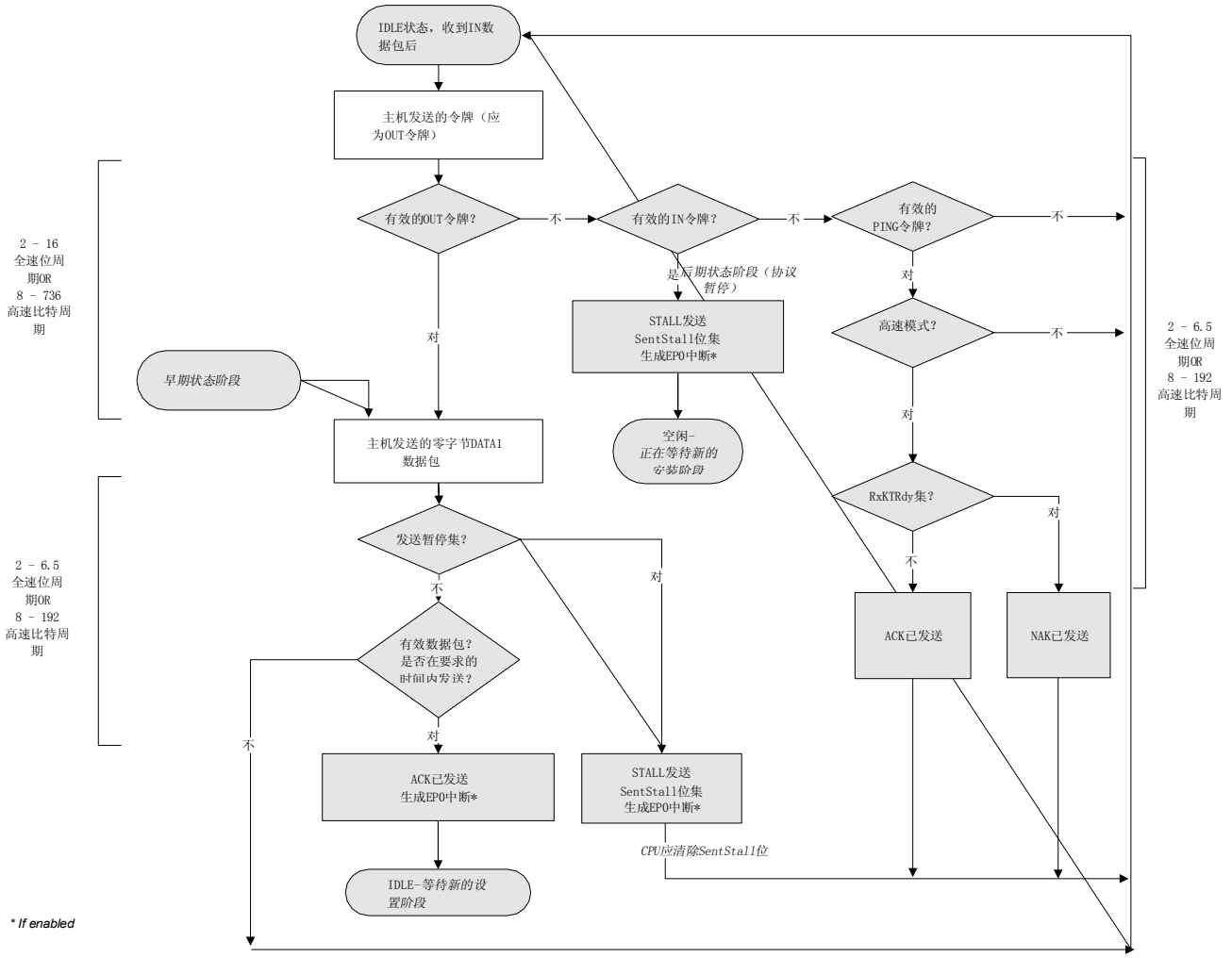
26.1.1.设置PHASE



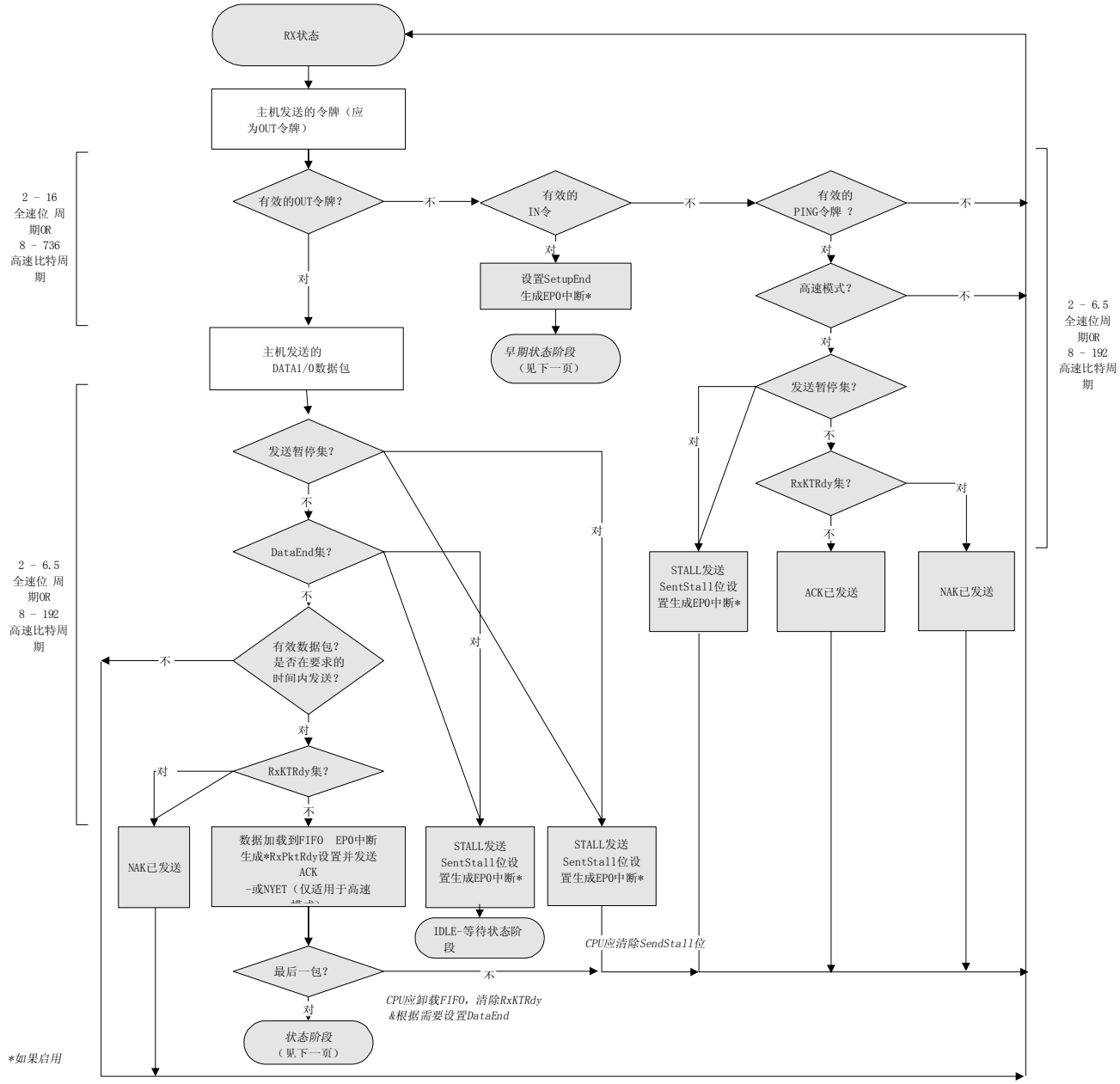
26.1.2.在DATAPHASE中



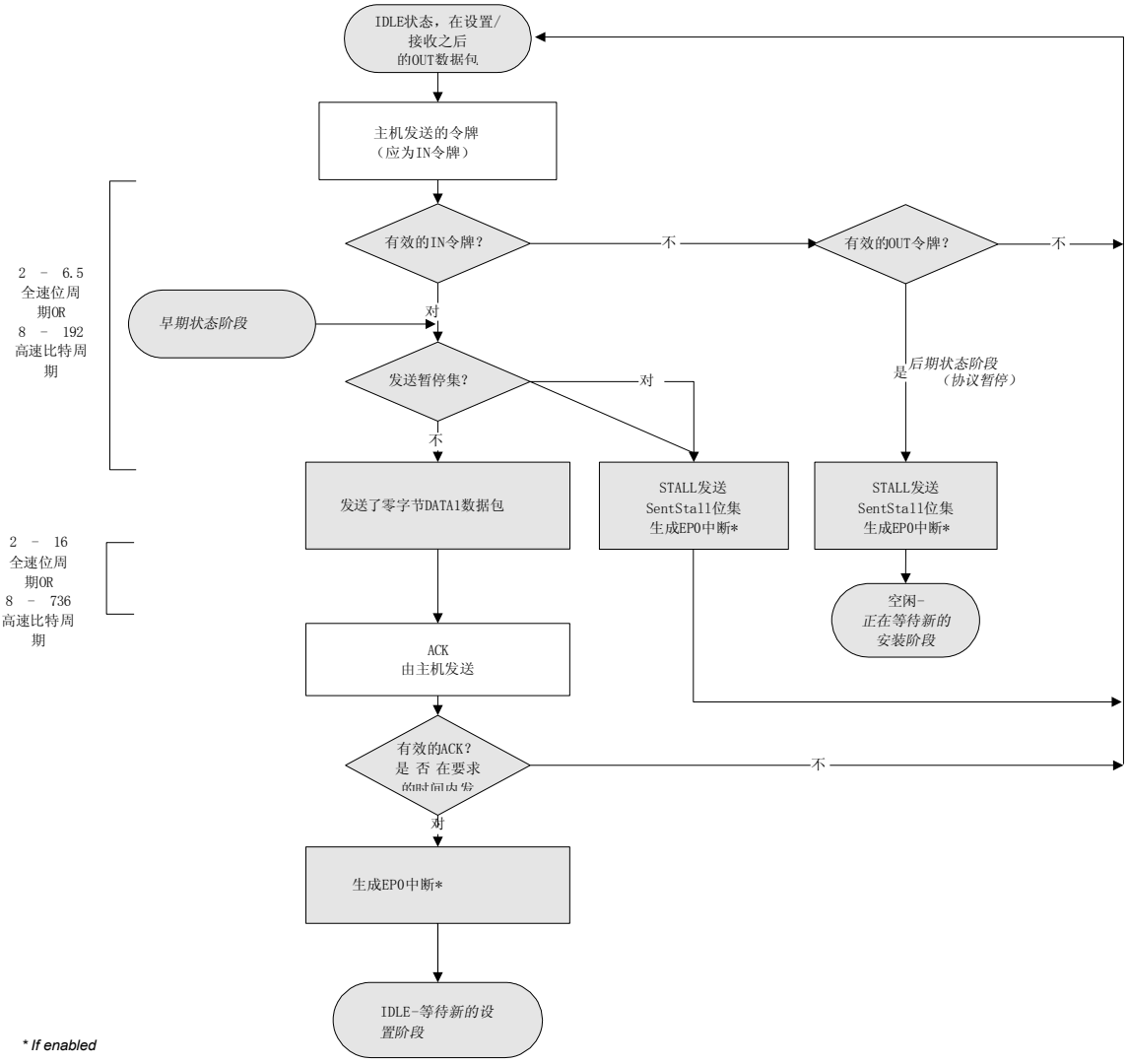
26.1.147. FOLLOWING THE STATUS PHASE



26.1.4.OUT DATA PHASE

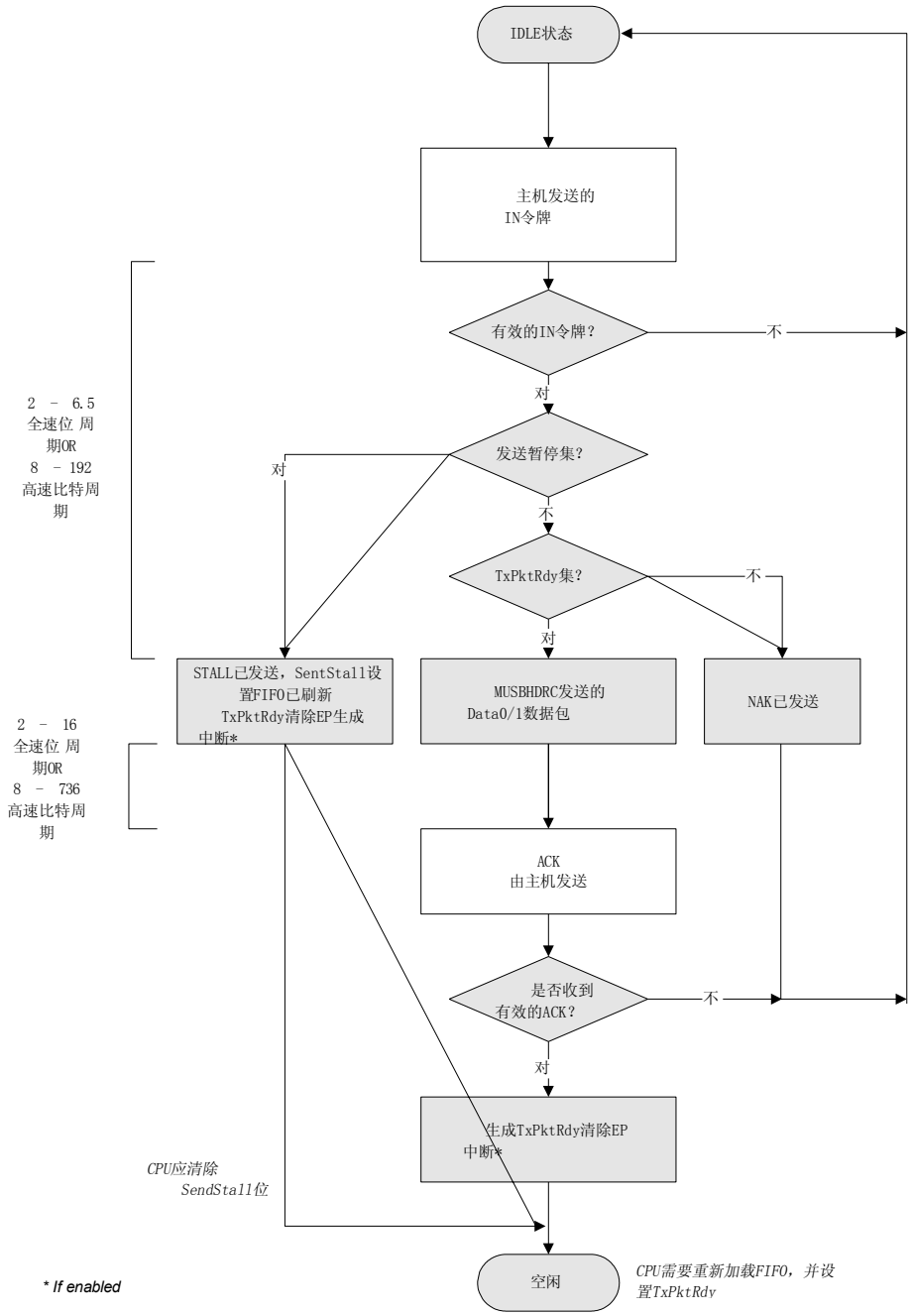


26.1.149. FOLLOWING THE STATUS PHASE

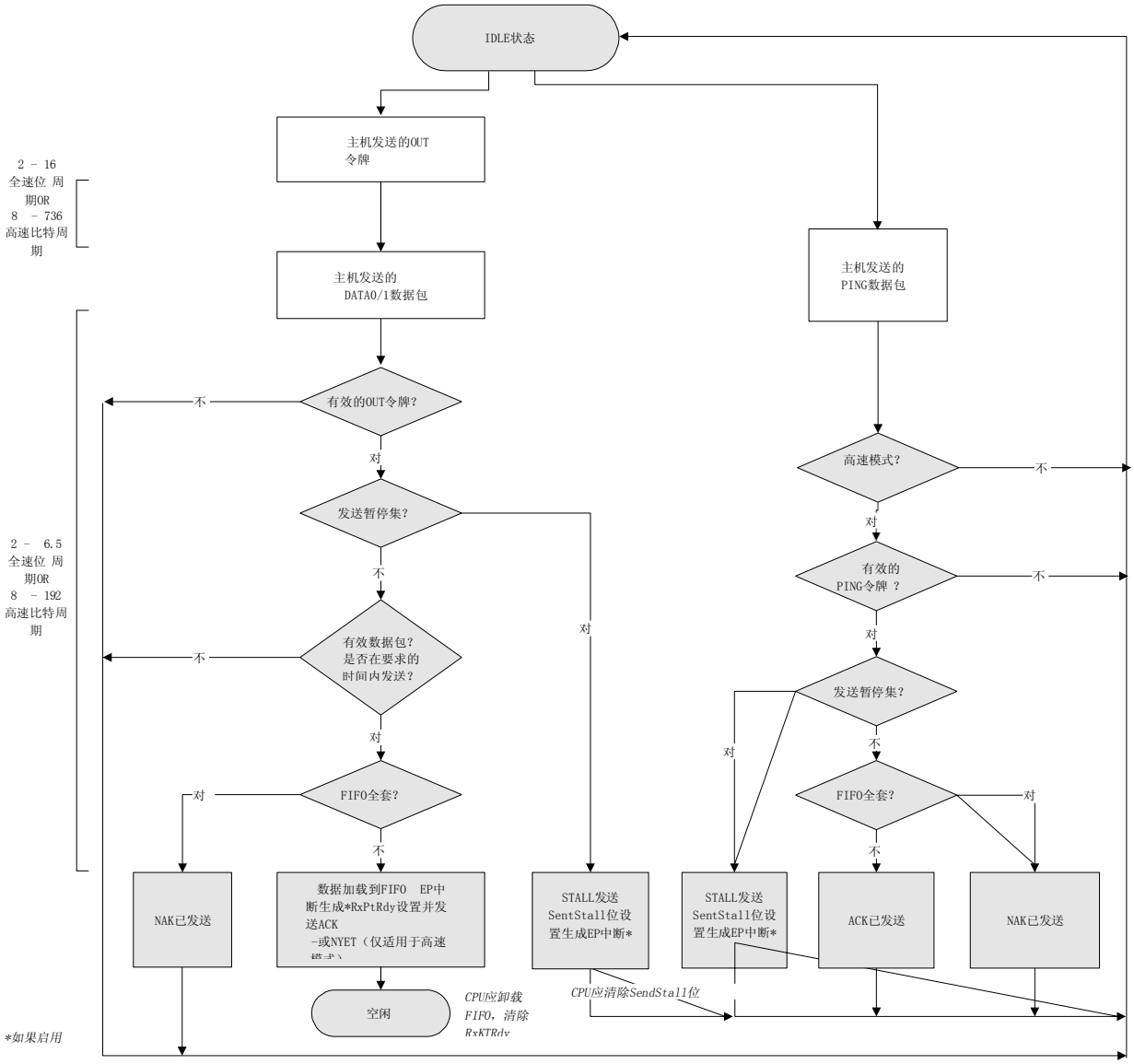


26.2.BULK/LOW-BANDWIDTH INTERRUPT TRANSACTIONS(地面上的地面宽度)

26.2.1.输入RANSACTION

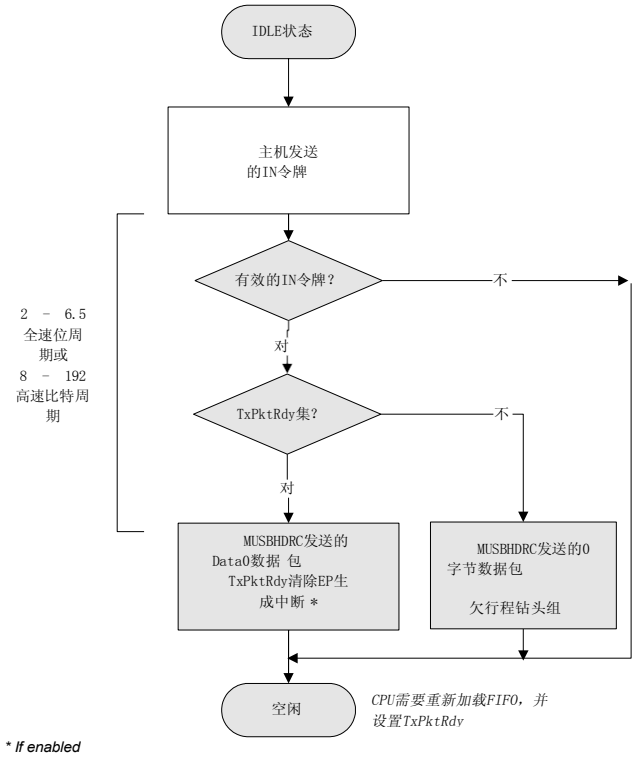


26.2.2.输出TRANSACTION

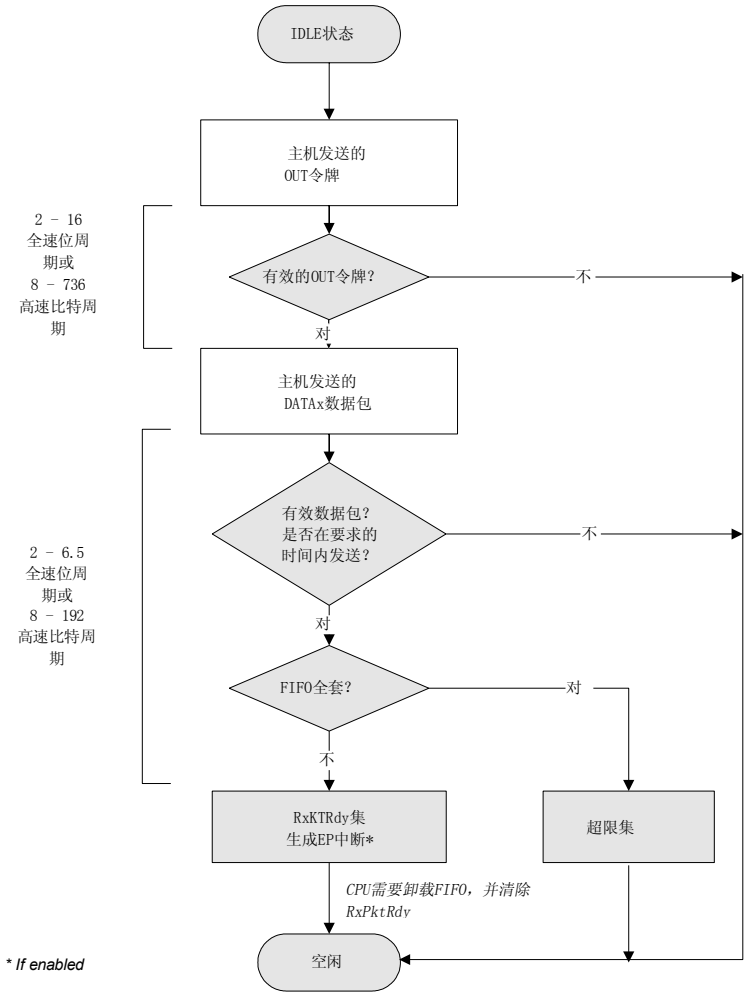


26.3.全速/低速-BANDWIDTH ISOCHRONOUS TRANSACTIONS

26.3.1.输入TRANSACTION

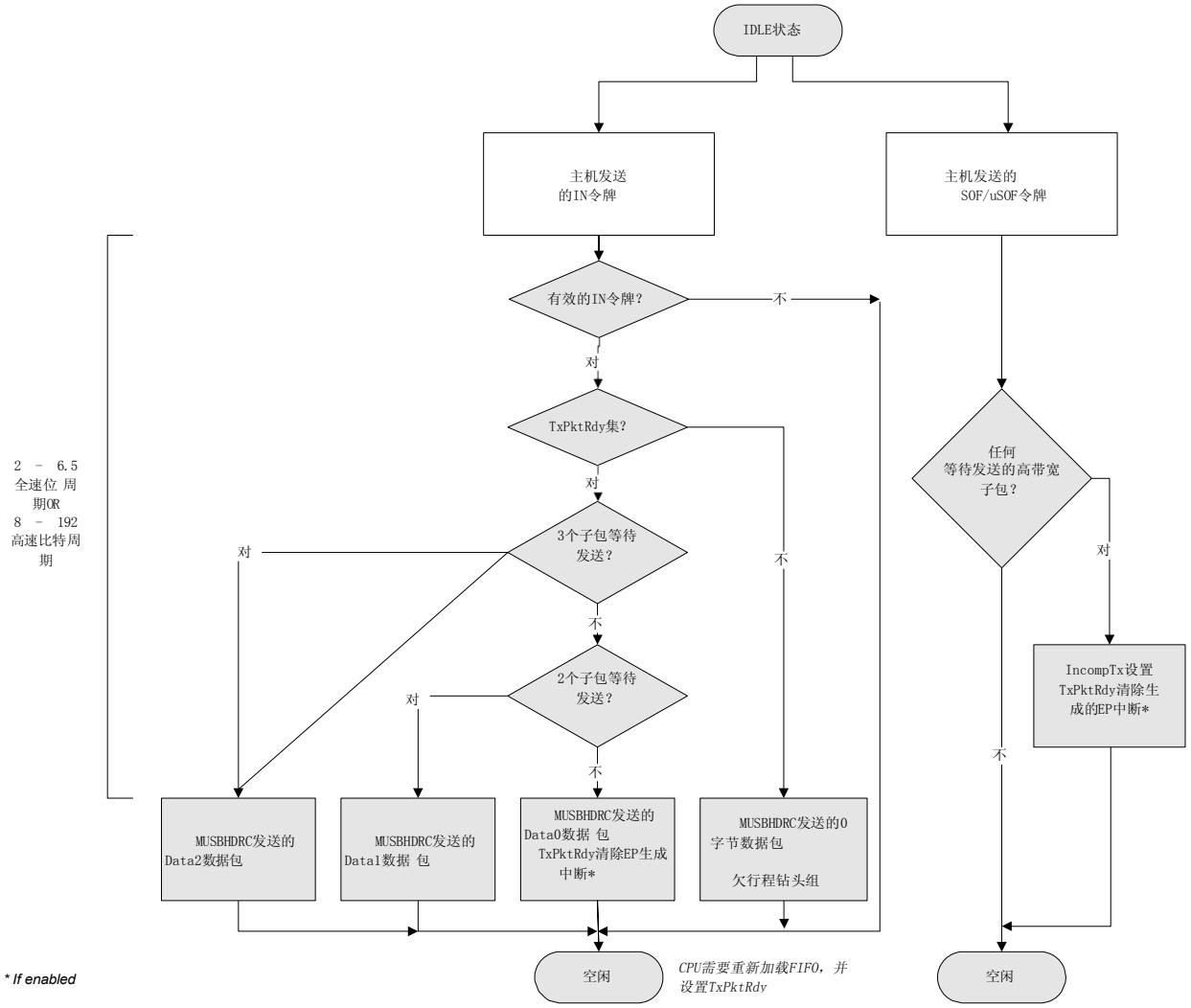


26.3.2.输出TRANSACTION

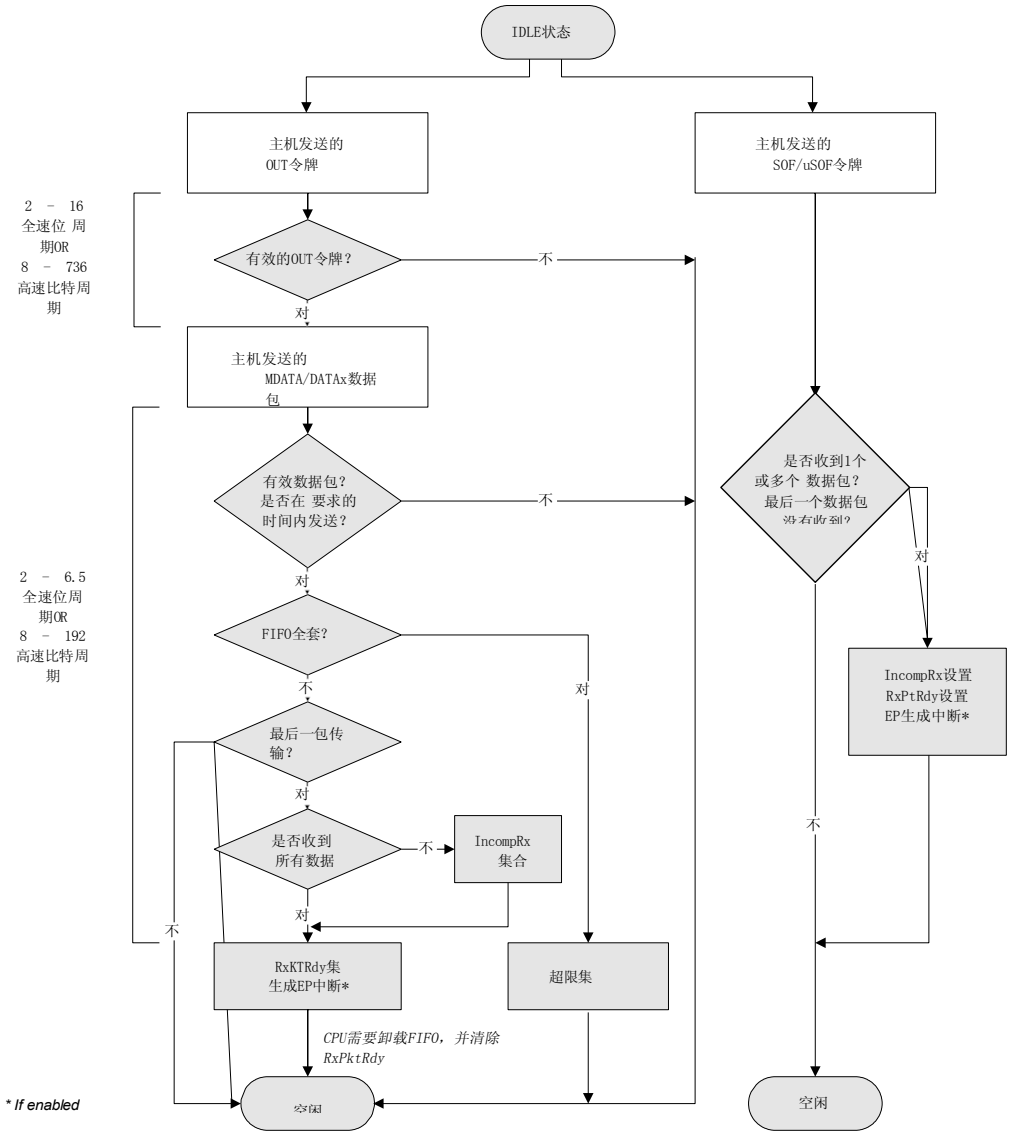


26.4. 高BANDWIDTH TRANSACTIONS (ISOCHRONOUS/INTERRUPT)

26.4.1. 输入RANSACTION



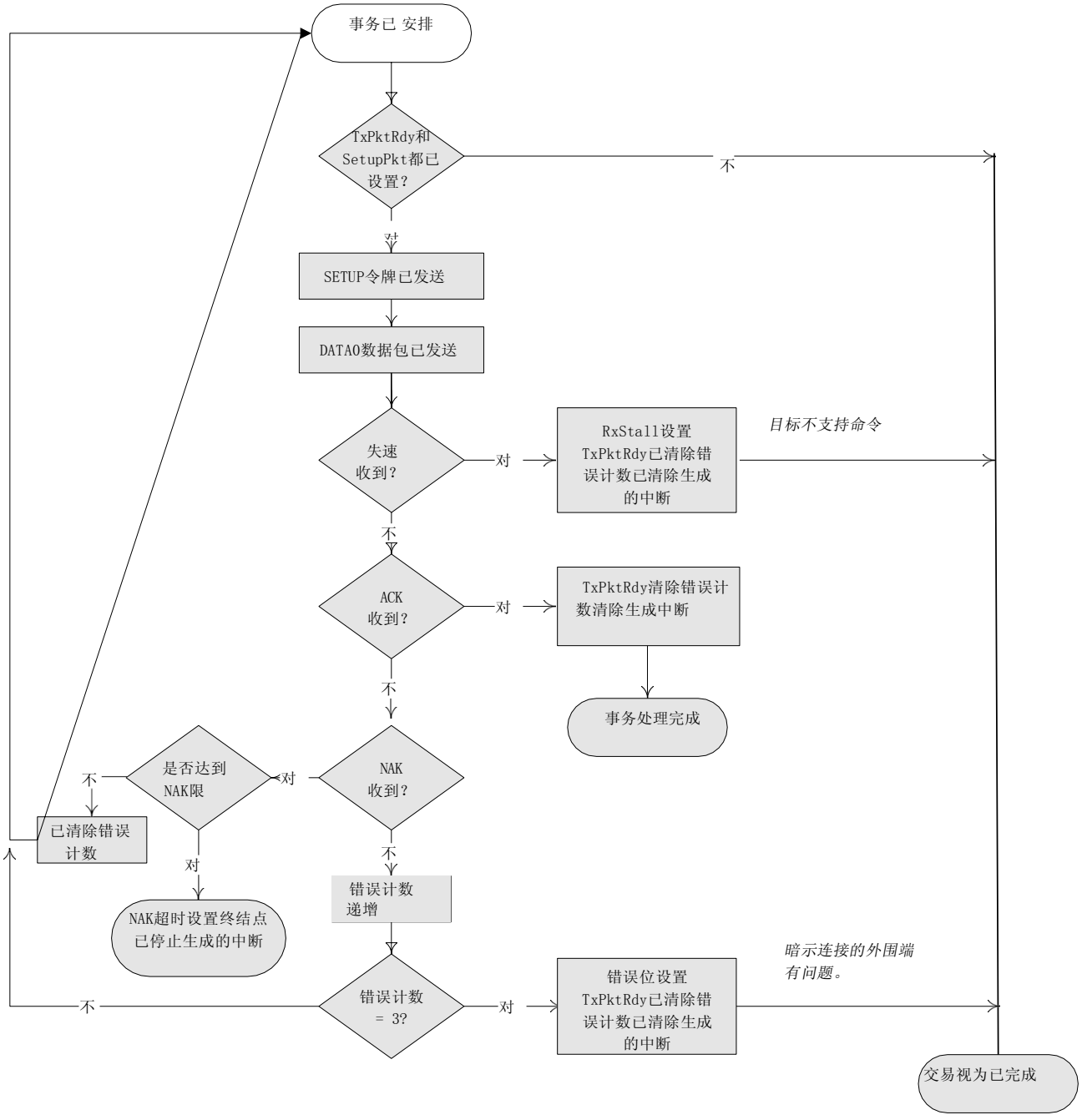
26.4.2.输出TRANSACTION



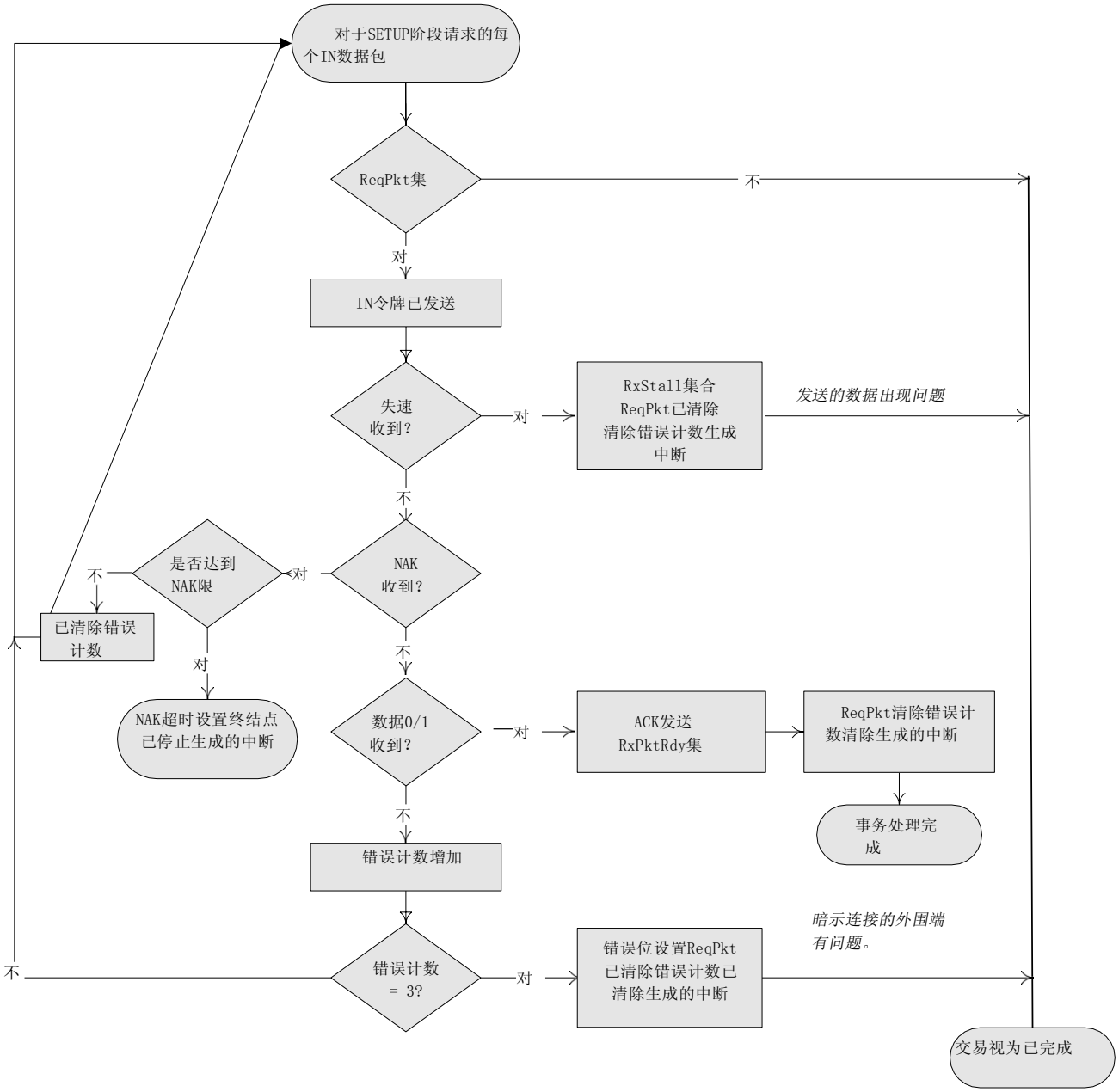
27.作为主机的事务流

2 7.1控制　角色任务

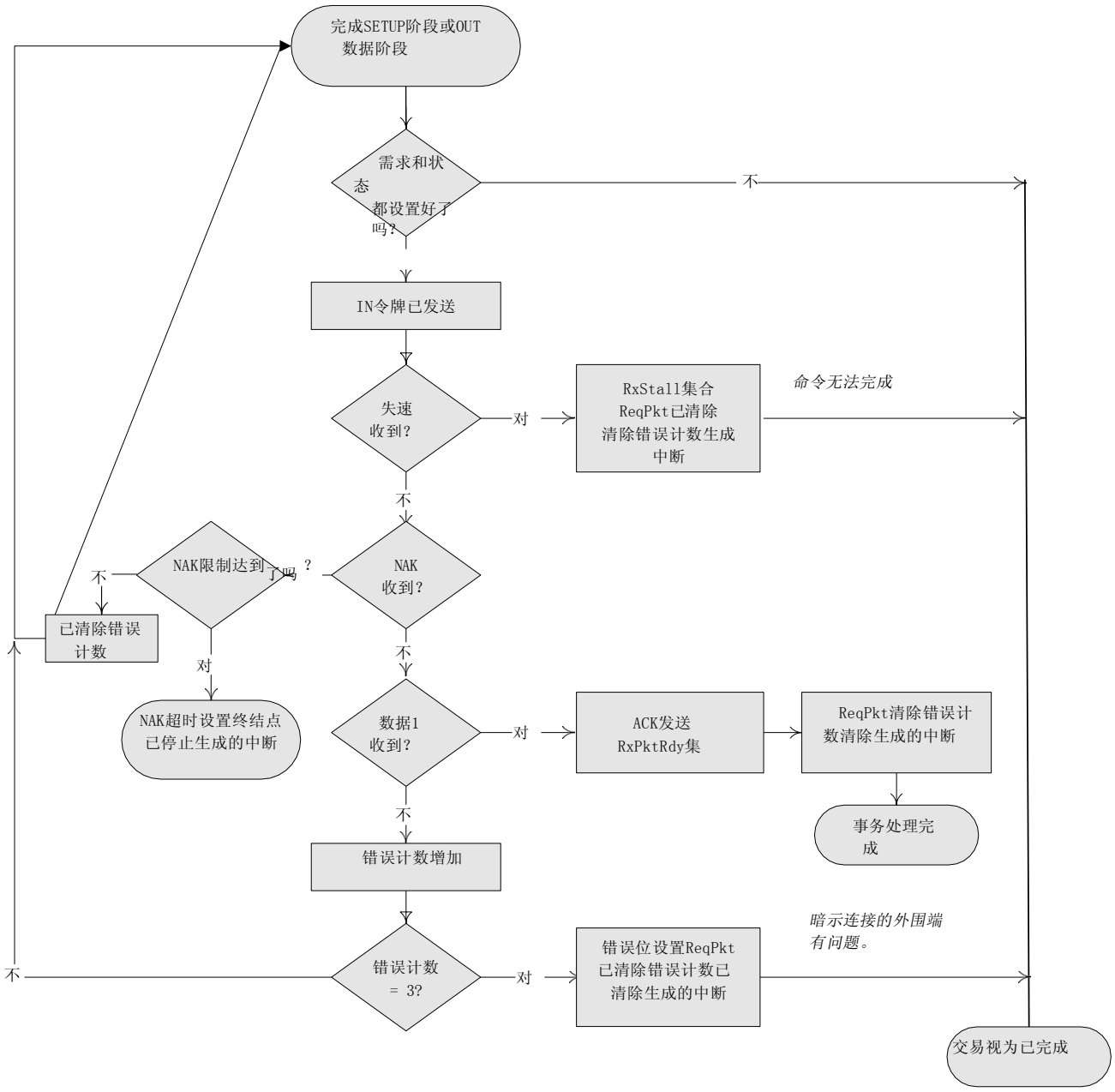
27. 1. 1. 设置P H A S E



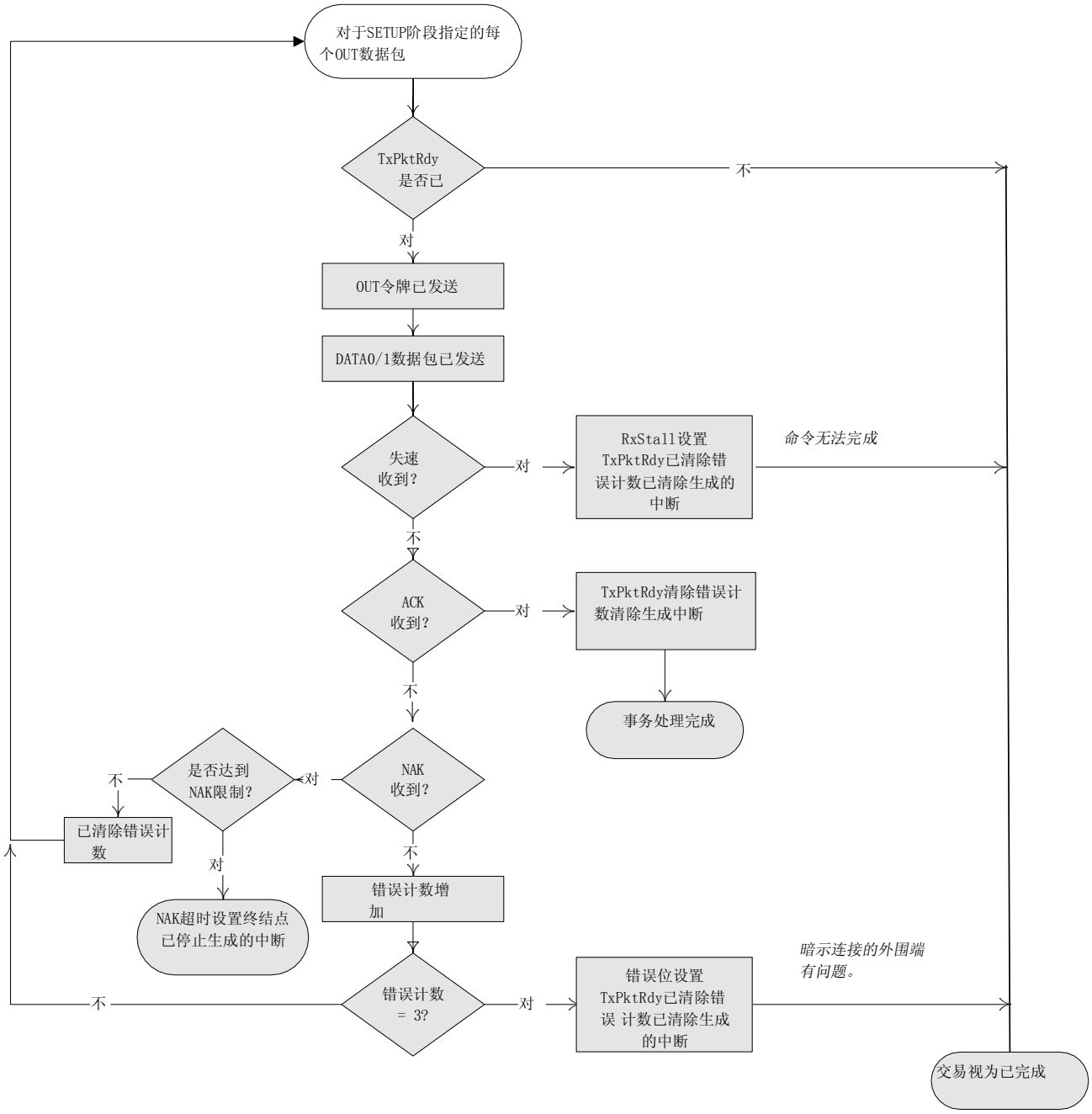
27.1.2. 在DATAPHASE...



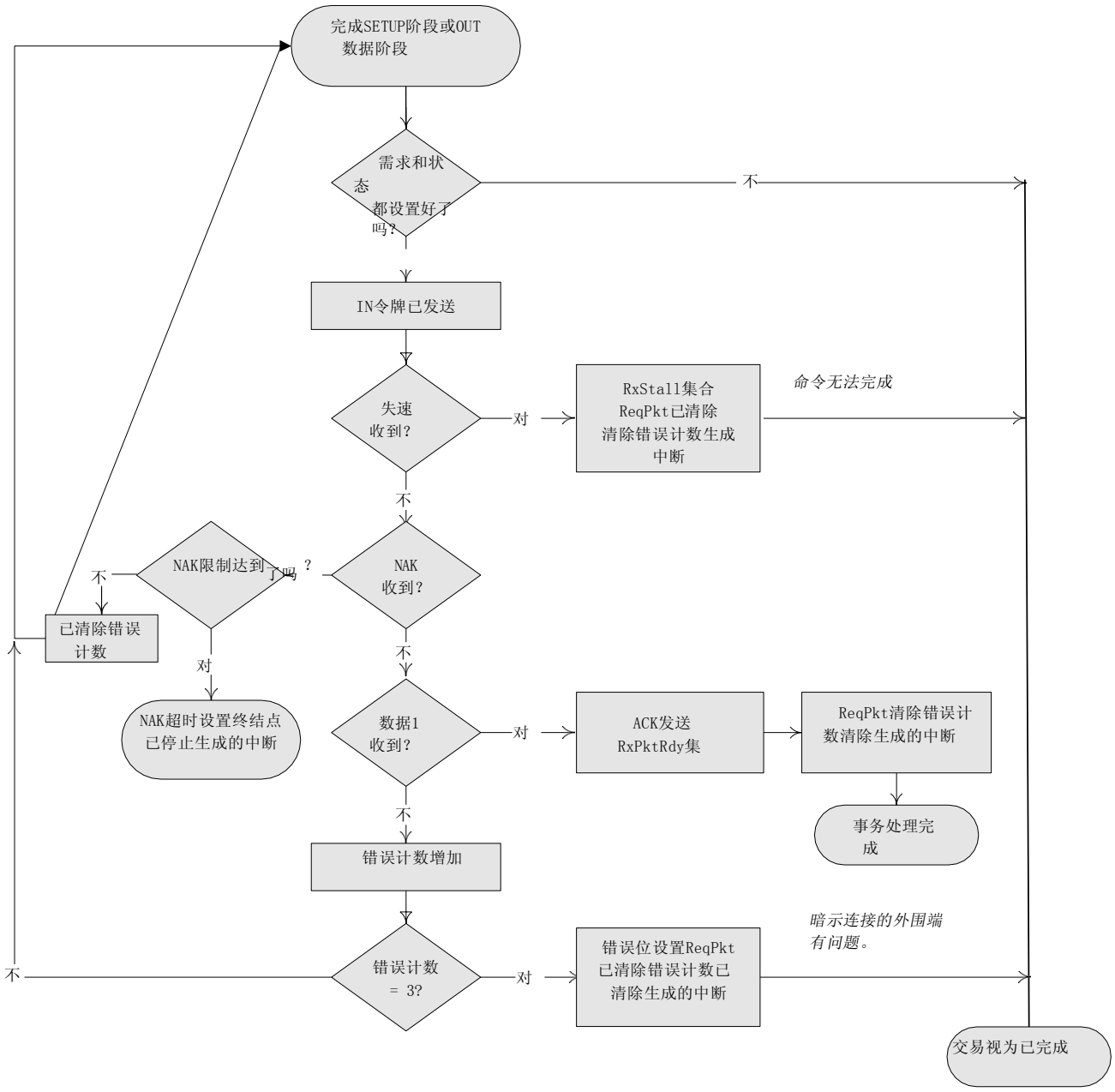
27.1.158. FOLLOWING THE STATUS PHASE



27.1.4. 出了DATA PHASE...

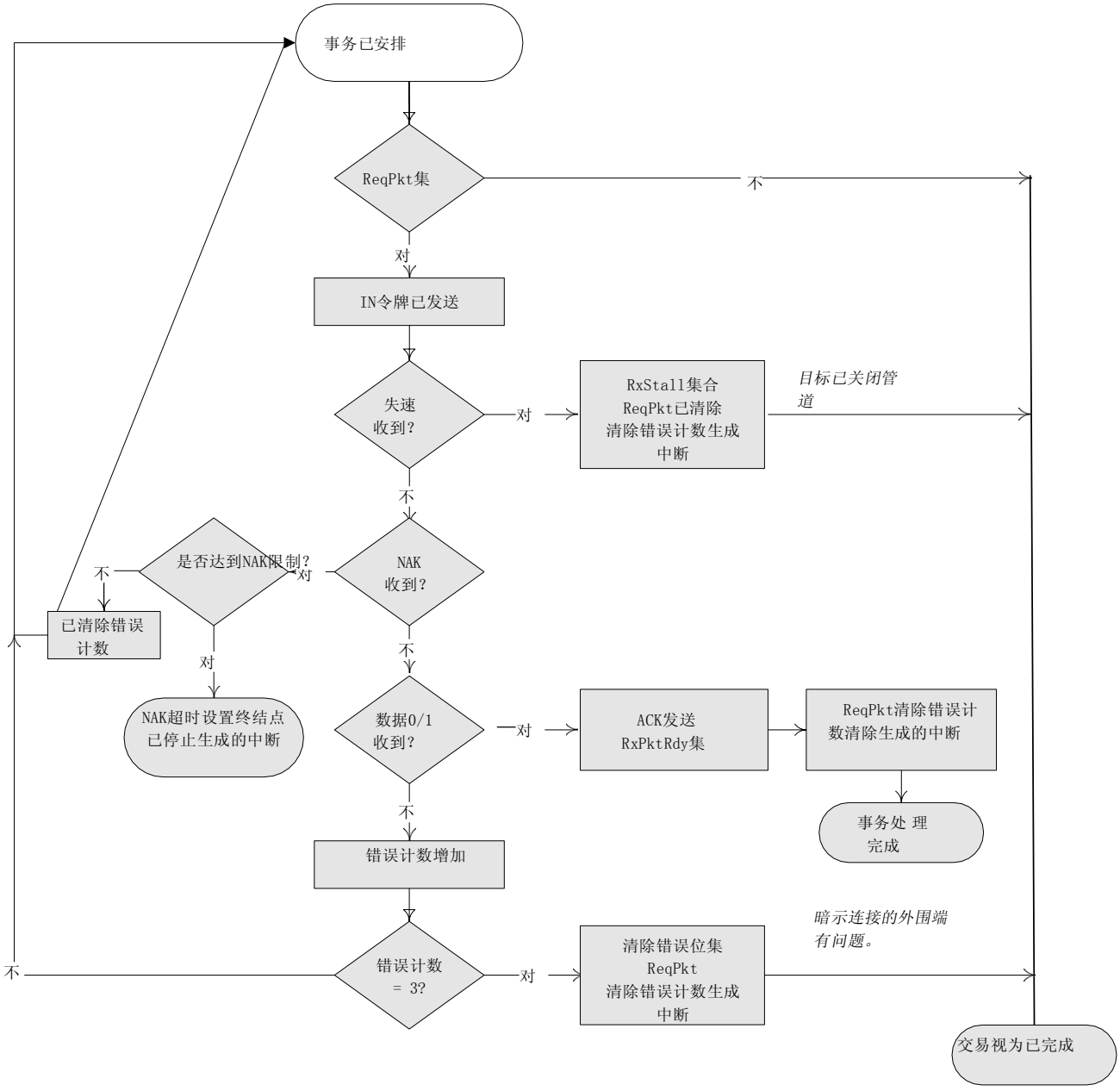


27.1.160. FOLLOWING THE STATUS PHASE

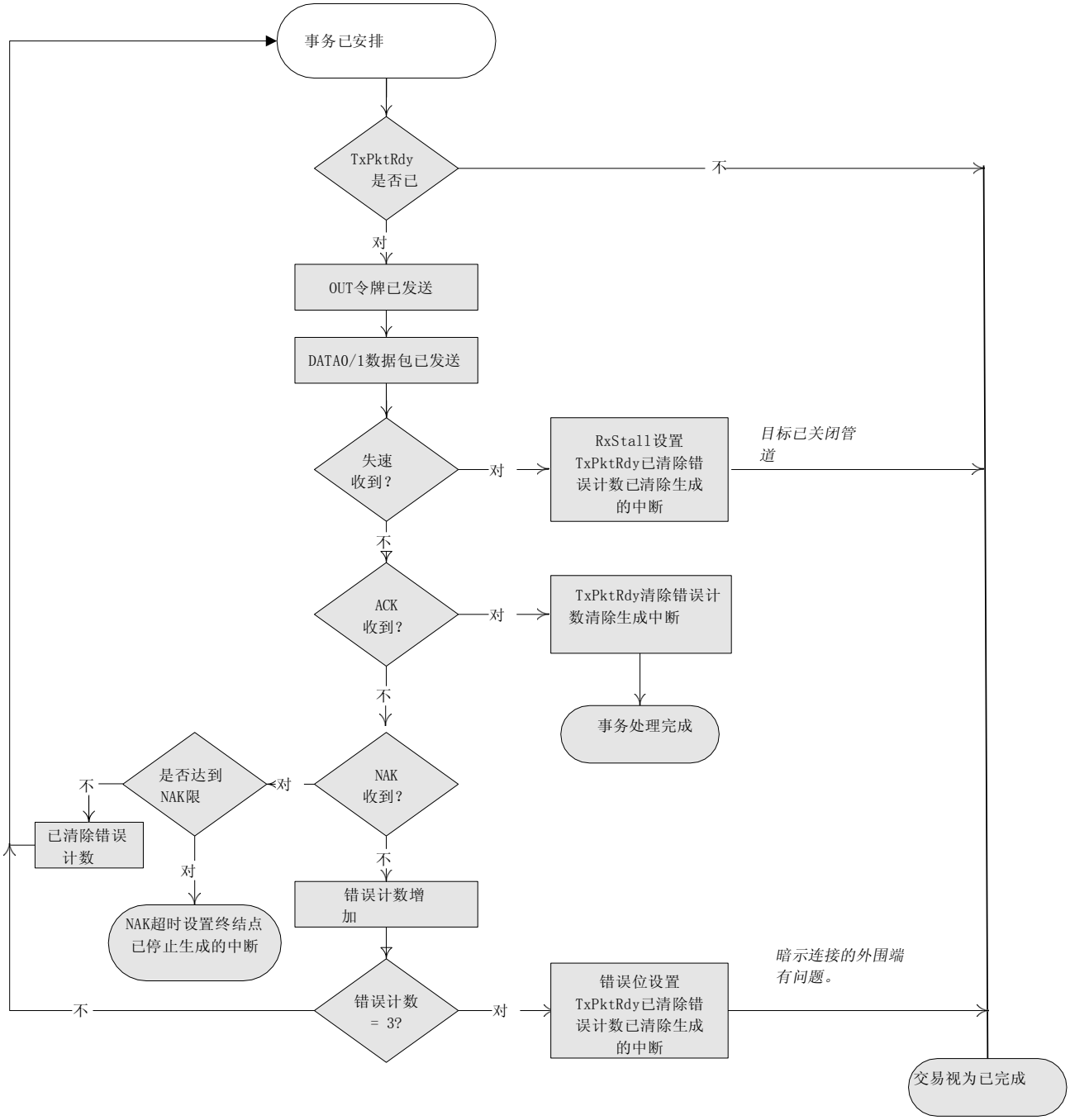


27.2. BULK/LOW-BANDWIDTH INTERRUPT TRANSACTIONS(地面上的地面宽度)

27.2.1. 输入RANSACTION

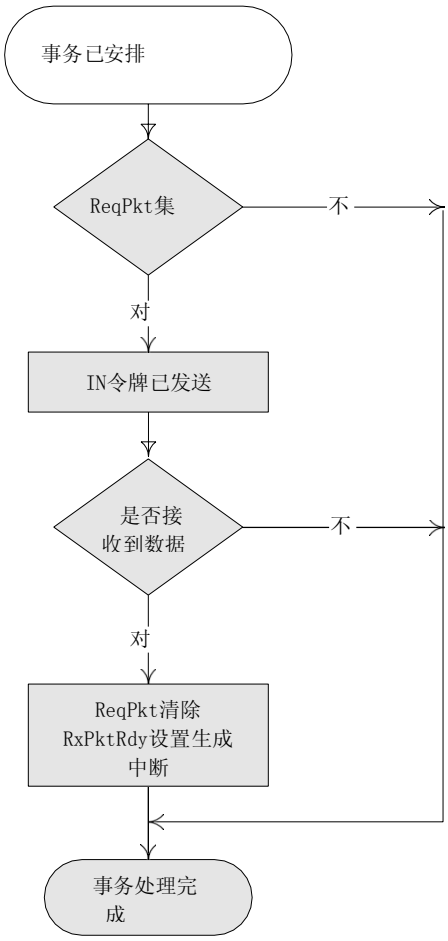


27.2.2. OUT TRANSACTION

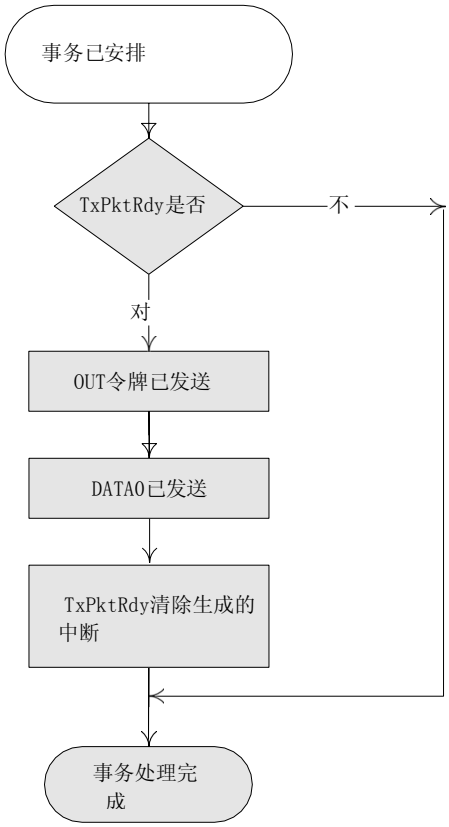


27.3. 全速/低速-BANDWIDTH ISOCHRONOUS TRANSACTIONS

27.3.1. 输入RANSACTION

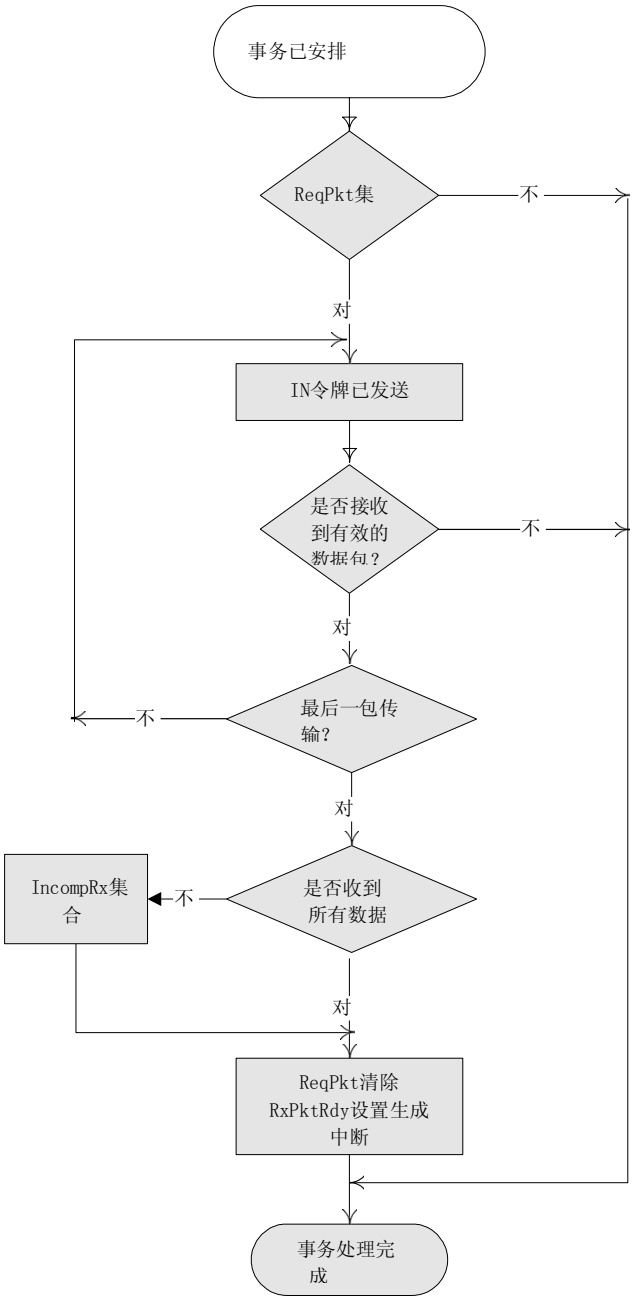


27.3.2. OUT TRANSACTION

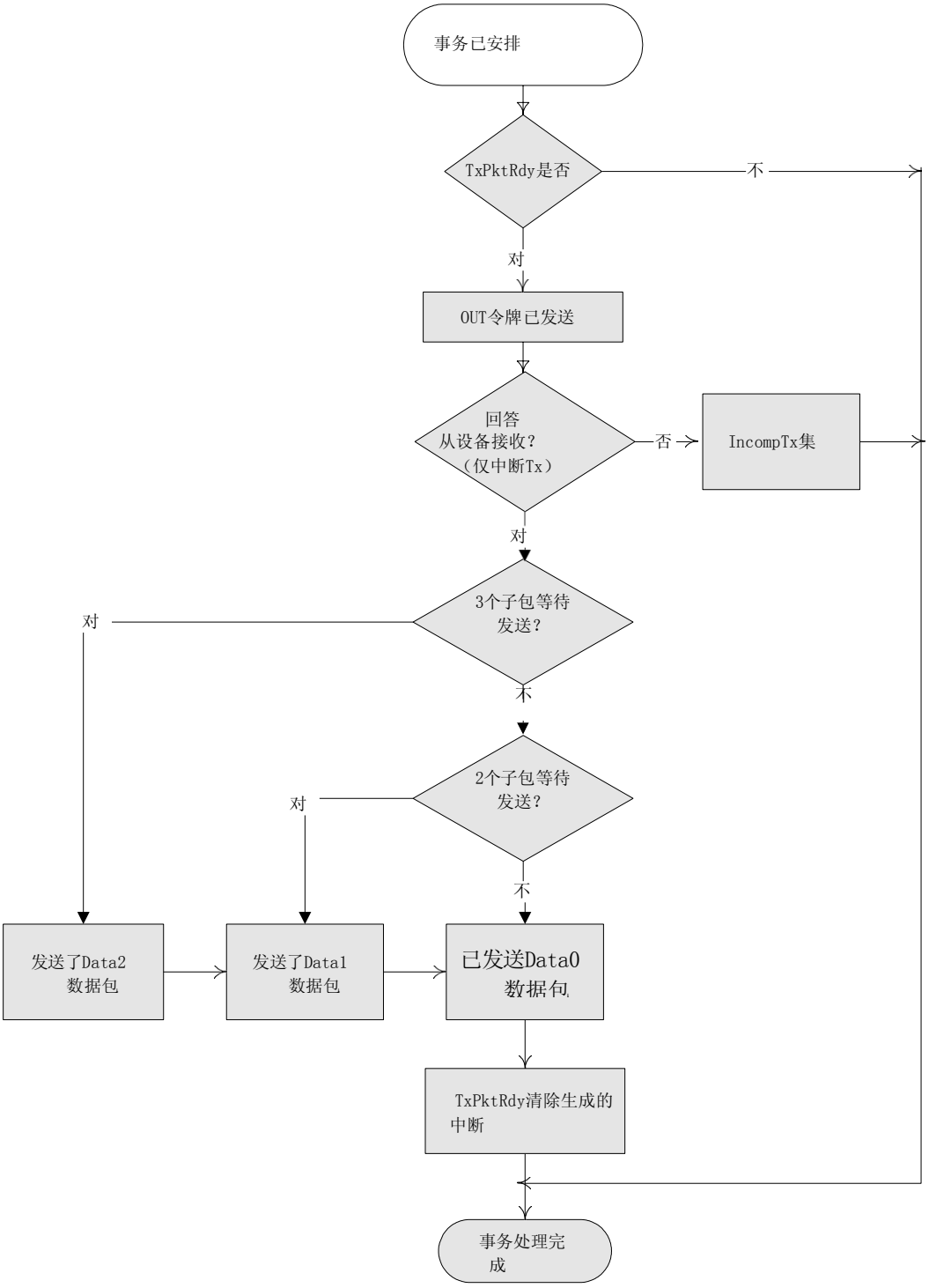


27.4. 高BANDWIDTH TRANSACTIONS(我是CHRONOUS/在陆地上)

27.4.1. 输入RANSACTION

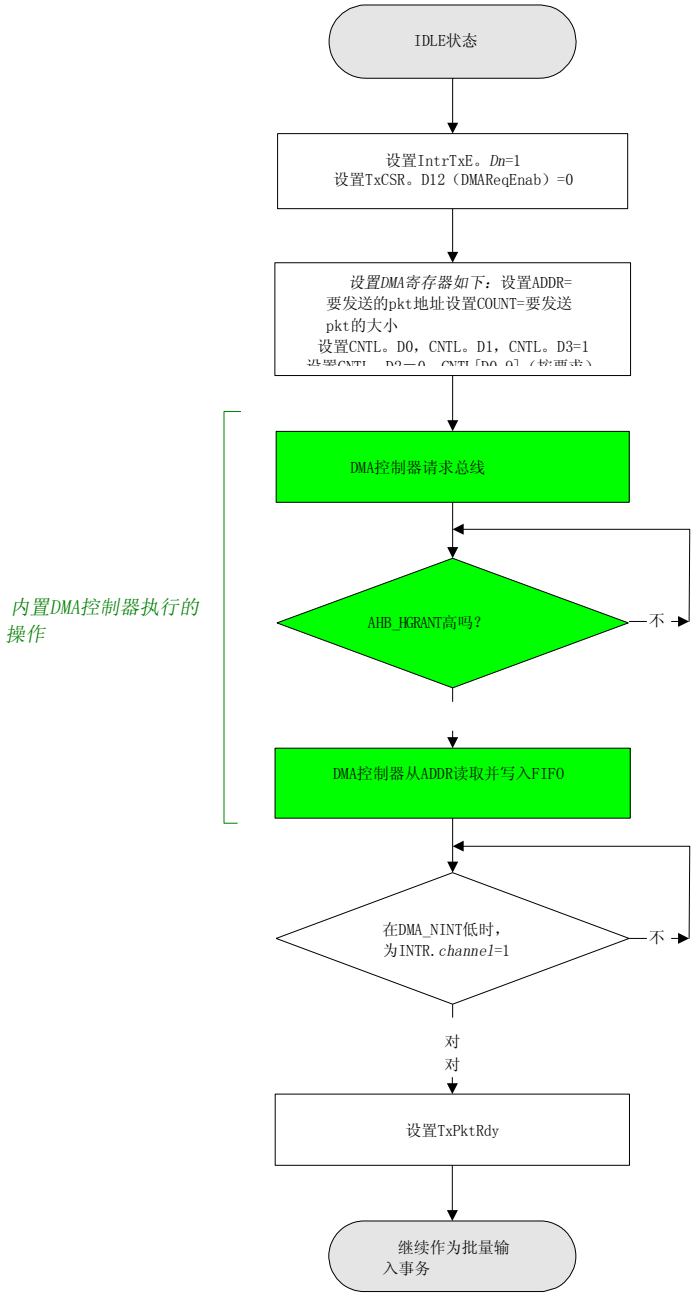


27.4.2. 输出TRANSACTION

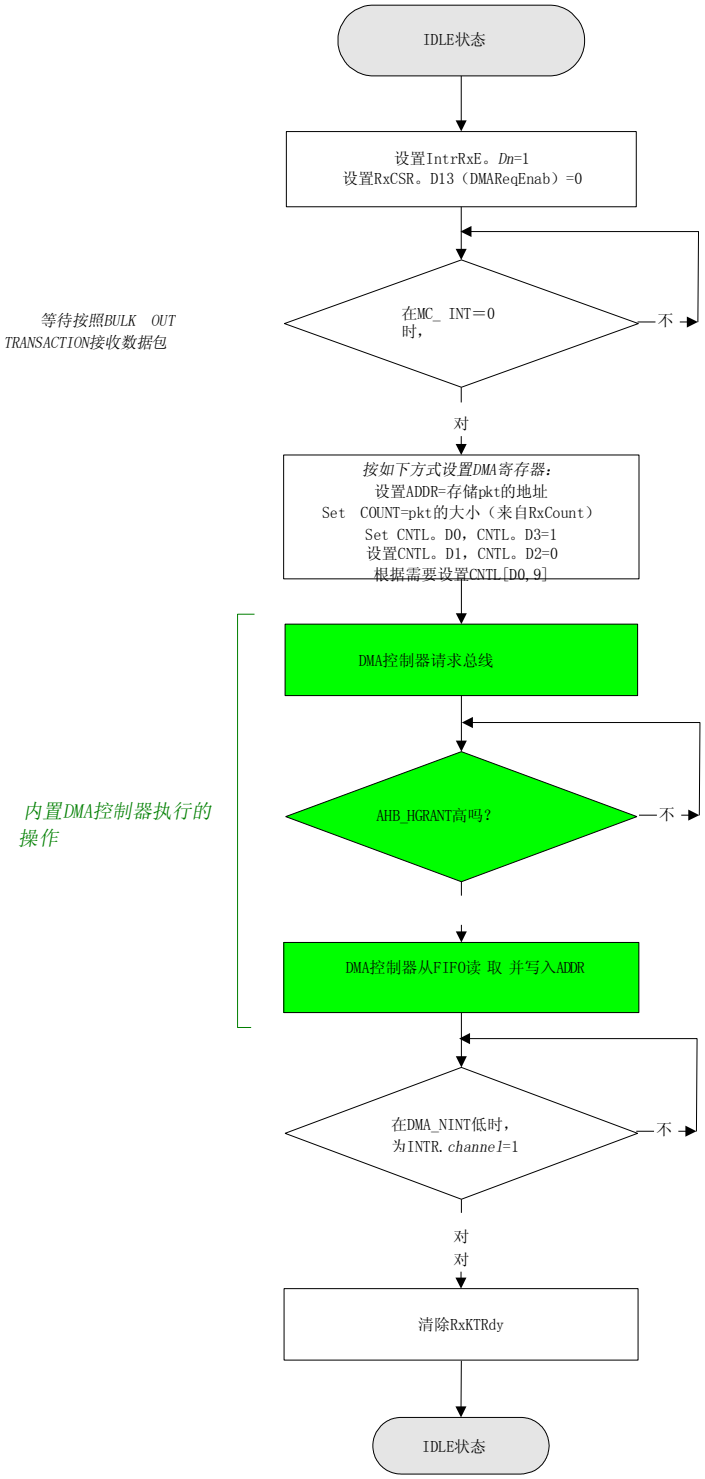


27.5. IONS上的DMA OPER(在DMA CONTROLLER中使用BUILT)

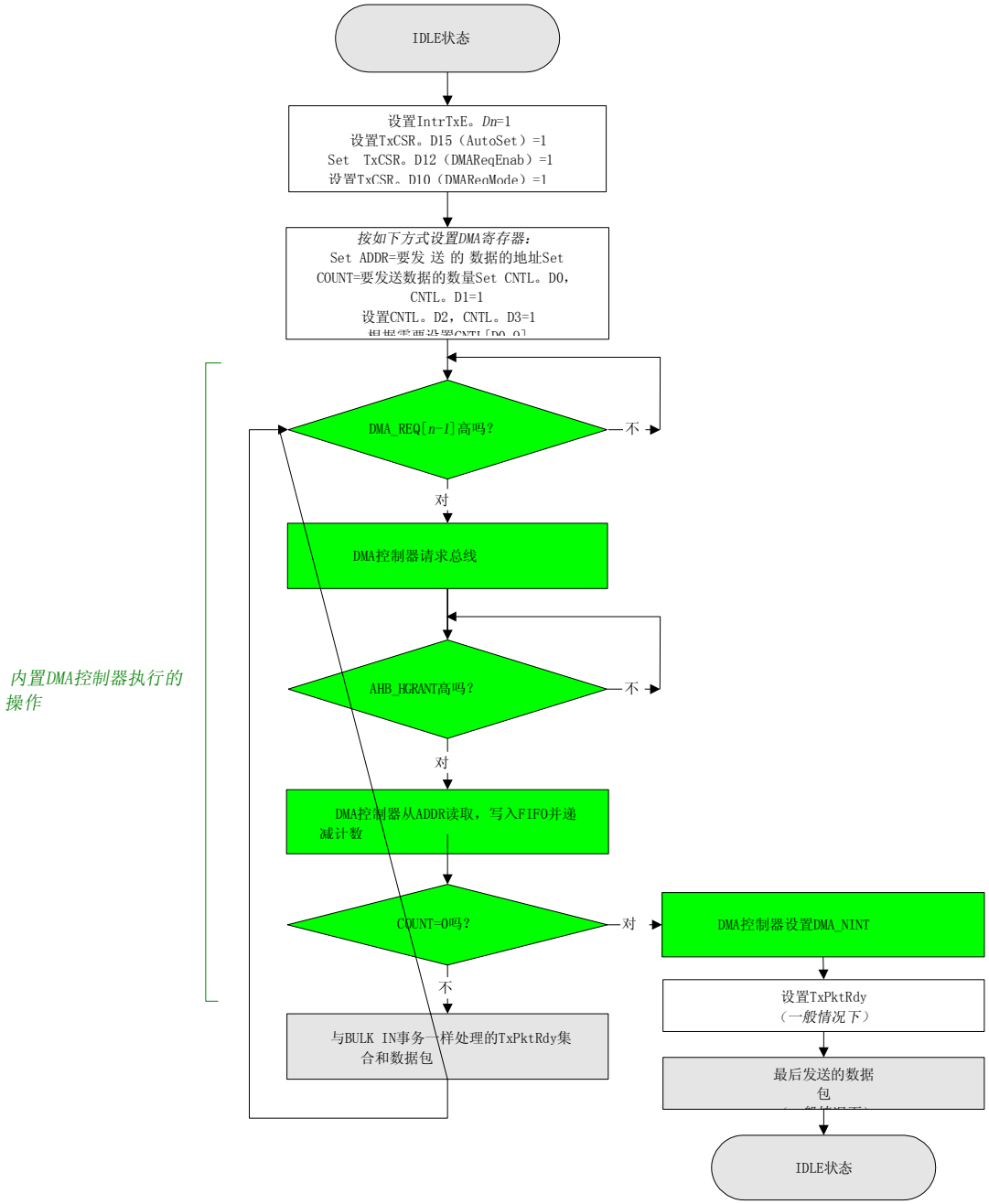
27.5.1. 犯罪



27.5.2. 成本

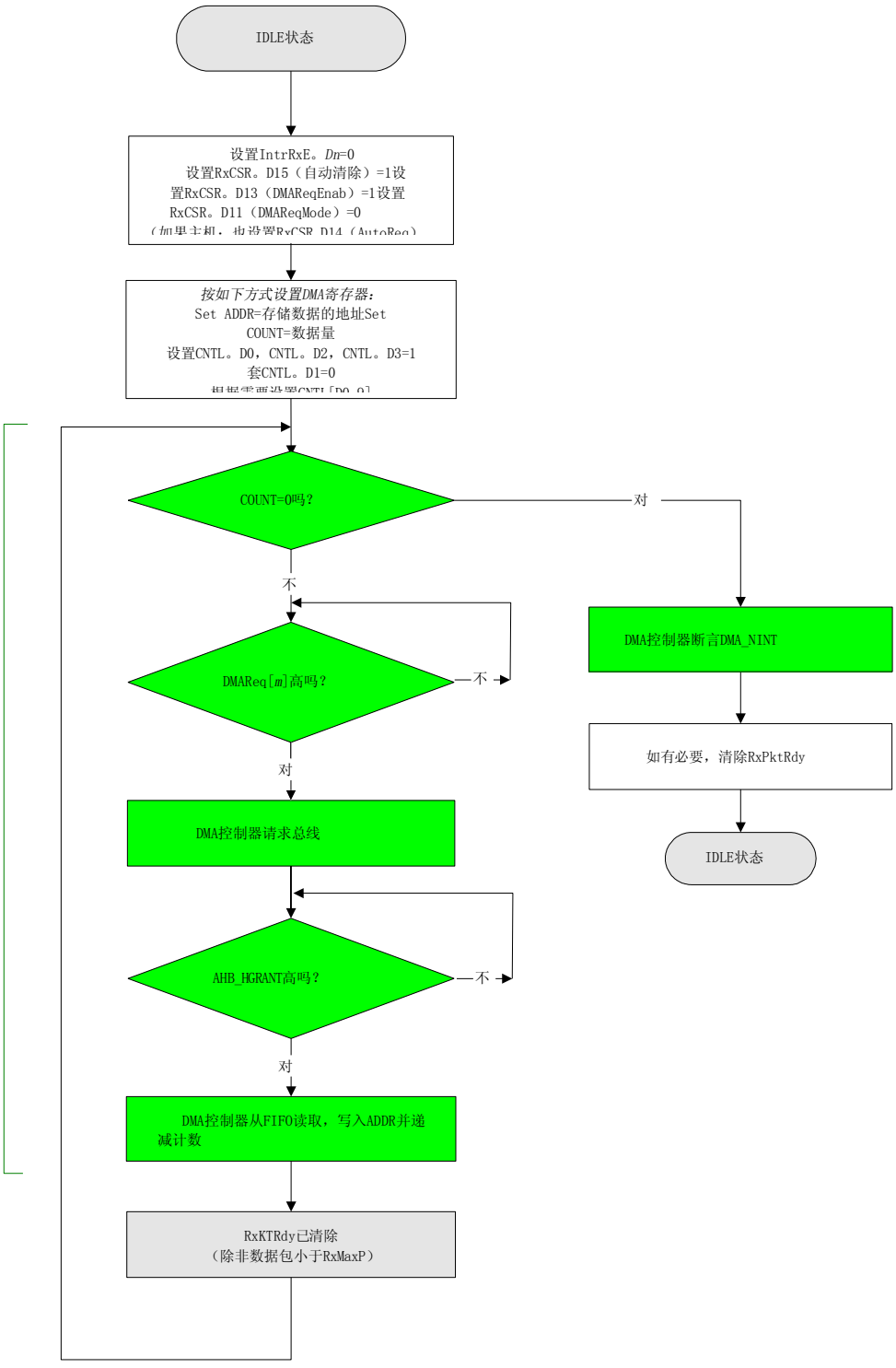


27.5. 3μLIP LE PACKET X

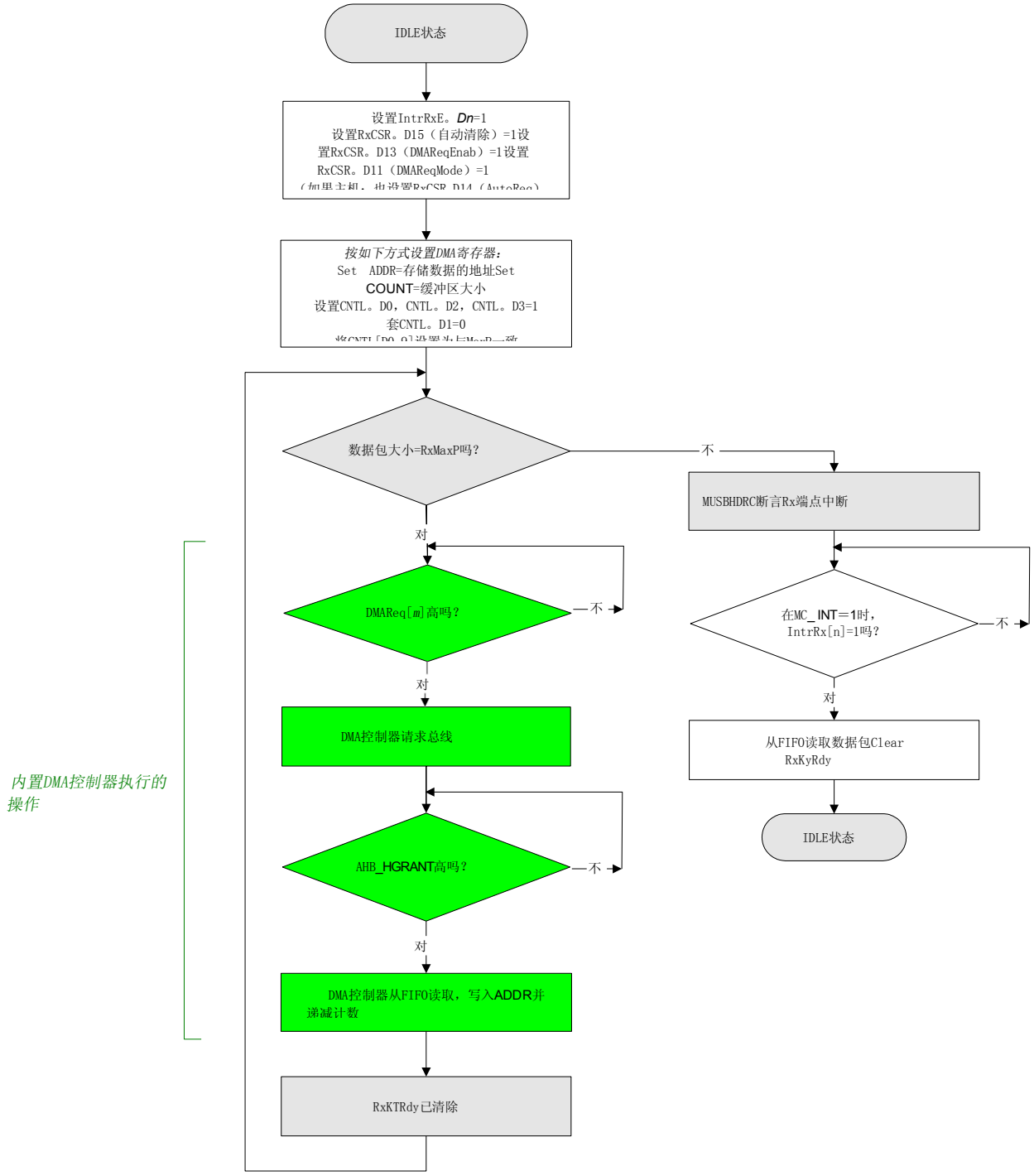


27.5. 4μL IP LE PACKETRX

如果数据块大小已知：



如果数据块大小未知：



28. 测试M O DES

MUSBHDRC支持为高速功能定义的四种USB 2.0测试模式。它还支持额外的"FIFO访问"测试模式，可用于测试CPU接口和RAM块的操作。

通过写入TestMode寄存器（地址0Fh）进入测试模式。测试模式通常由主机向端点0发送SET_FEATURE请求来请求。当软件接收到请求时，它应该等待，直到Endpoint 0传输完成（当它接收到指示状态阶段已完成的Endpoint 0中断时），然后写入TestMode寄存器。

注：这些测试模式在正常操作中没有任何用途。

28.1 .测　试_SE0_N A K

要进入Test_SE0_NAK测试模式，软件应通过将8'h01写入TestMode寄存器来设置Test_SE0.NAK位。然后，MUSBHDRC将进入一种模式，在该模式中，它用NAK响应任何有效的in令牌。

28.2 .测试_J

要进入Test_J测试模式，软件应通过将8'h02写入TestMode寄存器来设置Test_J位。然后，MUSBHDRC将进入在总线上传输连续J的模式。

28.3 .测试_K

要进入Test_K测试模式，软件应通过将8'04写入TestMode寄存器来设置Test_K位。然后，MUSBHDRC将进入在总线上传输连续K的模式。

28.4 .测试_ A C KET

要执行Test_Packet测试，软件应：

(i) 启动会话（如果核心正在主机模式下使用）。

(ii) 将标准测试数据包（如下所示）写入端点0 FIFO。

(iii) 将7'h08写入TestMode寄存器以进入Test_Packet测试模式。

(iv) 在CSR0寄存器（D1）中设置TxPktRdy位。

要加载的53字节测试数据包如下（所有字节均为十六进制）。测试包只需加载一次；MUSBHDRC将在没有来自软件的任何进一步干预的情况下继续重新发送测试分组。

00	00	00	00	00	00	00	00
00	AA	AA	AA	AA	AA	AA	AA
AA	EE	EE	EE	EE	EE	EE	EE
EE	FE	FF	FF	FF	FF	FF	FF
FF	FF	FF	FF	FF	7英尺	BF	DF
EF	F7	FB	FD	常设费	7E	BF	DF

用
EF F7 FB FD 7E

该数据序列在*通用串行总线规范*2.0版第7.1.20节中进行了定义。MUSBHDRC将DATA0 PID添加到数据序列的开头，并将CRC添加到末尾。

28.5. FIFO_acc-ESS

FIFO访问测试模式允许用户通过将一个高达64字节的数据包加载到端点0 FIFO中，然后再次读取，来测试CPU接口和RAM块的操作。使用端点0是因为它是双向端点，其Tx和Rx FIFO使用相同的RAM区域。

注意：内核不需要连接到USB总线即可运行此测试。如果已连接，则在运行测试时不应进行任何会话。

试验程序如下：

1. 使用任一CPU访问将最多64字节的数据包加载到端点0 Tx FIFO中。
2. 设置TxPktRdy（CSR0L.D1）。
3. 将8'M40写入测试模式寄存器。
4. 再次使用任一CPU访问从Endpoint Rx FIFO卸载数据包。
5. 设置ServicedRxPktRdy（CSR0L.D6）。

将0x40写入测试模式寄存器会导致以下事件序列：

1. 端点0 CPU指针（记录要传输的字节数）被复制到端点0 USB指针（记录接收的字节数。
2. 终结点0 CPU指针已重置。
3. TxPktRdy（CSR0L.D1）被清除。
4. RxPtRdy（CSR0L.D0）被设置。
5. 将生成端点0中断（如果已启用）。

这些步骤的效果是使端点0控制器起作用，就好像加载到Tx FIFO中的数据包已经被刷新并且通过USB接收到相同的数据包一样。加载到Tx FIFO的数据现在可以从Rx FIFO中读出。

28.6. 强制主机

强制主机测试模式使用户能够指示核心在主机模式下操作，无论它是否实际连接到任何外围设备，即CID输入的状态以及LINESTATE和HOSTDISCON信号被忽略。（在这种模式下，HOSTDISCON信号的状态可以从DevCtl寄存器的第7位读取。）

此模式通过在测试模式寄存器中设置位7来选择，允许实现USB Test_Force_Enable（7.1.20）。它还可以用于调试硬件中的PHY问题。

当Force_Host位保持设置时，核心将在会话位设置时进入主机模式，并保持在主机模式，直到会话位被清除，即使连接的设备在会话期间断开连接。此模式下的操作速度由测试模式寄存器的Force_HS和Force_FS位的设置确定，详见第3.2.11节。

29. 硬件读回

MUSBHDRC包括用于读取有关核心配置和创建核心的核心RTL版本的信息的设施。

29.1. HARDWARE在离子READBACK处配置UR

有关如何配置MUSBHDRC内核的各种详细信息，可从内核的ConfigData寄存器中获得

（如第3.3.5节所述），同时已配置Tx和Rx端点的数量以及RAM地址总线的宽度等信息可从EPInfo和RAMInfo寄存器获得（见第3.7.1节和第3.7.2节）。此外，与每个端点相关联的FIFO大小以及FIFO是否在Tx端点和Rx端点之间共享的详细信息可以从FIFOSize寄存器中获得（见第3.3.18节）。

ConfigData和FIFOSize寄存器都包含在核心的一组索引寄存器中，并且都位于偏移量0x1F处。当为端点0读取此位置的寄存器时（即索引寄存器设置为0），它将返回以下ConfigData信息：

一点	价值	理解
D7	0/1	当设置为"1"时，表示已选择批量数据包的自动组合。
D6	0/1	当设置为"1"时，表示已选择批量数据包的自动拆分。
D5	0/1	小恩迪亚总是"0"。
D4	0/1	当设置为"1"时，表示已选择高带宽Rx ISO端点支持。
D3	0/1	当设置为"1"时，表示已选择高带宽Tx I SO端点支持。
D2	0/1	当设置为"1"时，表示选择了动态FIFO大小选项。
D1	0/1	对于软连接/断开连接，始终为"1"。
D0	0/1	对于UTMI+8位的数据宽度，始终为"0"。

当读取端点编号1–15时，寄存器返回已配置相应Tx和Rx端点的FIFO的详细信息，如下所示：（对于端点0，有一个64字节FIFO用于Rx和Tx事务。）

位	价值	理解
0 – 3	0	没有具有此终结点编号的Tx终结点
	3 – 11	TxFIFO大小=^{2n}（8–2048字节）
4 – 7	0	没有具有此终结点编号的Rx终结点
	3 – 11	RxFIFO大小=^{2n}（8–2048字节）
	15	Tx和Rx端点共享相同的FIFO（大小由位0–3给出）

注意：在使用动态FIFO大小选项的情况下，FIFOSize寄存器不会返回有效信息。

29.2.RTL版本READBACK

可以从HWVers寄存器中读取关于创建MUSBHDRC核心的特定实现的RTL版本的详细信息。

这个寄存器位于6Ch，主要记录创建核心的RTL的版本号。这些版本号的格式为*xx.yyy*，其中*xx*是主要修订号，*yyy*是次要修订号。主要修订号记录在HWVers寄存器的第14-10位，而次要修订号记录在此寄存器的第9-0位。

核心使用寄存器的顶部位来指示核心是从"候选版本"而不是核心的完整版本创建的。发布候选者不应用于测试以外的任何其他目的。它们不应用于制造成品硅。

30. 李谈嗨故事

30.1. 第1期

2006年8月1日-将《MUSBHDRC程序员指南》的内容合并到《MUSBMDRC产品规范》中。

30.2. 第2期

2006年10月17日。添加了内部DMA的内容。将 所有寄存器描述移至第3节。

30.3. 第3期

2007年5月25日-版本1.900；针对以下缺陷进行了修改。

- dts100383116-使编译器指令C_T_HSBT可编程
- dts100398841-清除内部DMA文档，这意味着起始地址可以是非字边界。
- dts100402572-musbmhdrc_pspg.pdf第20.5.2节中的DMA加载/卸载时序图不正确。
- dts100402896-第4.2节中的时钟同步图不正确。
- dts100386440-ULPI链路AN错误地引用了MUSBHDR_ulpictl而不是MUSBHDRC_lpictl