**目录**

[第1章 PS/2 协议 5](#_Toc48147599)

[1.1 物理连接 5](#_Toc48147600)

[1.2 电气属性 5](#_Toc48147601)

[1.3 数据示意图 5](#_Toc48147602)

[1.4 总线上数据的收发 5](#_Toc48147603)

[1.4.1 设备向主机发送数据 5](#_Toc48147604)

[1.4.2 PS/2设备向PC机读取一个字节 6](#_Toc48147605)

[1.4.3 主机发送数据到设备 7](#_Toc48147606)

[第2章 LPC 总线 8](#_Toc48147607)

[2.1 物理连接 8](#_Toc48147608)

[2.2 电气属性 8](#_Toc48147609)

[2.3 Memory, I/O and DMA Cycle 9](#_Toc48147610)

[2.3.1 Start 9](#_Toc48147611)

[2.3.2 Cycle Type / Direction (CyCTYPE + DIR) 9](#_Toc48147612)

[2.3.3 Size 10](#_Toc48147613)

[2.3.4 Turn-Around (TAR) 10](#_Toc48147614)

[2.3.5 ADDR 10](#_Toc48147615)

[2.3.6 Channel 10](#_Toc48147616)

[2.3.7 DATA 10](#_Toc48147617)

[2.3.8 SYNC 11](#_Toc48147618)

[2.3.9 SYNC TimeOut 11](#_Toc48147619)

[2.3.10 SYNC Error Indication 11](#_Toc48147620)

[2.3.11 LFRAME# 12](#_Toc48147621)

[2.3.12 Start of Cycle 12](#_Toc48147622)

[2.3.13 Abort机制 13](#_Toc48147623)

[2.3.14 内存读写位域定义 13](#_Toc48147624)

[2.3.15 主机启动内存读 14](#_Toc48147625)

[2.3.16 主机启动内存写 14](#_Toc48147626)

[2.3.17 IO读写域定义 15](#_Toc48147627)

[2.3.18 主机启动IO读操作 16](#_Toc48147628)

[2.3.19 主机启动IO写操作 16](#_Toc48147629)

[2.4 Firmware Memory Cycle 17](#_Toc48147630)

[2.4.1 start 17](#_Toc48147631)

[2.4.2 IDSEL (Device Select) 17](#_Toc48147632)

[2.4.3 MADDR (Memory Address) 17](#_Toc48147633)

[2.4.4 MSIZE (Memory Size) 17](#_Toc48147634)

[2.4.5 TAR 18](#_Toc48147635)

[2.4.6 SYNC 18](#_Toc48147636)

[2.4.7 DATA 18](#_Toc48147637)

[2.4.8 Protocol 18](#_Toc48147638)

[2.4.9 Preamble 18](#_Toc48147639)

[2.4.10 Firmware Memory Read Cycle 19](#_Toc48147640)

[2.4.11 Firmware Memory Write Cycles 19](#_Toc48147641)

[2.4.12 Error Reporting 20](#_Toc48147642)

[2.4.13 固件存储器字段定义 20](#_Toc48147643)

[2.4.14 主机启动固件存储器读操作 21](#_Toc48147644)

[2.4.15 主机启动固件存储器写操作 21](#_Toc48147645)

[2.5 DMA 21](#_Toc48147646)

[2.5.1 DMA数据请求 21](#_Toc48147647)

[2.5.2 取消DMA请求 22](#_Toc48147648)

[2.5.3 DMA 传输定义 22](#_Toc48147649)

[2.5.4 Terminal Count 24](#_Toc48147650)

[2.5.5 取消传输 24](#_Toc48147651)

[2.5.6 SYNC field / LDRQ# Rules 25](#_Toc48147652)

[2.5.7 DMA读操作 25](#_Toc48147653)

[2.5.8 DMA写操作 25](#_Toc48147654)

[2.5.9 关于16和32位DMA的其他说明 25](#_Toc48147655)

[2.6 Bus Master Protocol 26](#_Toc48147656)

[2.6.1 周期格式和时间 26](#_Toc48147657)

[2.6.2 Bus Master Cycle 字段定义 26](#_Toc48147658)

[2.6.3 外设启动内存读操作 27](#_Toc48147659)

[2.6.4 外设启动内存写操作 27](#_Toc48147660)

[2.6.5 外设启动IO读操作 28](#_Toc48147661)

[2.6.6 外设启动IO写操作 28](#_Toc48147662)

[2.6.7 Request Assertion Rules(请求断言规则) 28](#_Toc48147663)

[2.7 电源管理（PM） 29](#_Toc48147664)

[2.7.1 CLKRUN# Protocol 29](#_Toc48147665)

[2.7.2 LPCPD# Protocol 29](#_Toc48147666)

[2.7.3 LPME# Usage 30](#_Toc48147667)

[2.8 总线复位机制 30](#_Toc48147668)

[第3章 eSPI 总线 31](#_Toc48147669)

[3.1 物理连接 31](#_Toc48147670)

[3.1.1 连接拓扑图 32](#_Toc48147671)

[3.2 电气属性 33](#_Toc48147672)

[3.3 总线协议 34](#_Toc48147673)

[3.3.1 Single Master-Single Slave 35](#_Toc48147674)

[3.3.2 Single Master-Multiple Slaves 35](#_Toc48147675)

[3.3.3 指令域（CMD） 36](#_Toc48147676)

[3.3.4 周转域 (TAR) 38](#_Toc48147677)

[3.3.5 响应域(RESP) 38](#_Toc48147678)

[3.3.6 Alert 域 42](#_Toc48147679)

[3.3.7 状态获取指令域 44](#_Toc48147680)

[3.3.8 获取和设置指令域 45](#_Toc48147681)

[3.3.9 Non-Posted 传输 46](#_Toc48147682)

[3.3.10 Posted 传输 48](#_Toc48147683)

[3.3.11 WAIT STATE 50](#_Toc48147684)

[3.4 传输层 51](#_Toc48147685)

[3.4.1 周期类型和数据包格式 51](#_Toc48147686)

[3.4.2 Channels 54](#_Toc48147687)

[3.4.3 Slave buffer management 75](#_Toc48147688)

[3.4.4 Transaction ordering rule 76](#_Toc48147689)

[3.4.5 Zero length read or write 77](#_Toc48147690)

[3.5 链路层(Link Layer) 78](#_Toc48147691)

[3.5.1 1,2,4 I/O模式 78](#_Toc48147692)

[3.5.2 CRC 81](#_Toc48147693)

[3.6 从机寄存器配置 83](#_Toc48147694)

[第4章 SMBus 总线(2.0) 91](#_Toc48147695)

[4.1 物理连接 92](#_Toc48147696)

[4.2 电气属性 92](#_Toc48147697)

[4.3 物理层 92](#_Toc48147698)

[4.4 数据链路层 93](#_Toc48147699)

[4.4.1 总线仲裁 94](#_Toc48147700)

[4.5 网络层 97](#_Toc48147701)

[4.5.1 设备标识–从机地址 97](#_Toc48147702)

[4.5.2 设备的使用 98](#_Toc48147703)

[4.5.3 数据包错误检查 98](#_Toc48147704)

[4.5.4 总线协议 100](#_Toc48147705)

[4.5.5 地址解析协议（ARP） 100](#_Toc48147706)

[4.6 SMBus 可选信号 101](#_Toc48147707)

[4.6.1 SMBSUS# 101](#_Toc48147708)

[4.6.2 SMBALERT# 102](#_Toc48147709)

[附录一：posted与non-posted 104](#_Toc48147710)

[附录二 CRC-8 计算方法 105](#_Toc48147711)

[附录三 I2C与SMBus的异同 106](#_Toc48147712)

[附录四 SMBus 设备地址分配 107](#_Toc48147713)

# PS/2 协议

## 物理连接

一般只有4个引脚有意义；分别是Clock(时钟脚)、DATA(数据脚)、+5V(电源脚)和Ground(电源地)，其中时钟线和数据线必须接上拉电阻。

## 电气属性

一般两设备间传输数据的最大时钟频率是33kHz，大多数ps/2设备工作在10~20kHz。推荐值在 15kHz左右，也就是说，Clock(时钟脚)高、低电平的持续时间都为40μs。

## 数据示意图

|  |  |
| --- | --- |
| 1 start bit. This is always 0 | 1 个起始位 总是为 0 |
| 8 data bits, least significant bit first. | 8 个数据位 低位在前 |
| 1 parity bit (odd parity) | 1 个校验位 奇校验 |
| 1 stop bit. This is always 1 | 1 个停止位 总是为 1 |
| 1 acknowledge bit | 1 个应答位 仅在主机对设备的通讯中 |

## 总线上数据的收发

### 设备向主机发送数据

设备到主机的通讯 当时钟为高 数据线改变状态 在时钟信号的下降沿，数据被锁存。

#### PS/2设备向PC机发送一个字节

(1)等待时钟为高电平，并延时50us；

(2)检测判断时钟信号是否为高，为高，则向下执行，为低，则转到(1)；

(3)检测数据线是否为高，如果为高则继续执行，如果为低，则放弃发送(此时PC机在向PS/2设备发送数据，所以PS/2设备要转移到接收程序处接收数据)；

(4)延时20μs(如果此时正在发送起始位，则应延时４0μs)；

(5)输出起始位(0)到数据线上。这里要注意的是：在送出每一位后都要检测时钟线，以确保PC机没有抑制PS/2设备，如果有则中止发送；

(6)输出8个数据位到数据线上；

(7)输出校验位；

(8)输出停止位(1)；

(9)延时30μs(如果在发送停止位时释放时钟信号则应延时50μs)；

#### 发送单个位

(1)准备数据位(将需要发送的数据位放到数据线上)；

(2)延时20μs；

(3)把时钟线拉低；

(4)延时40μs；

(5)释放时钟线；

(6)延时20μs。

### PS/2设备向PC机读取一个字节

这时为PC主机发送数据，所以PC机通过下拉时钟线大于 100μs来抑制通讯，然后再释放总线来发送数据，主机此时在时钟线变为低时准备数据到数据线，并在时钟上升沿锁存数据。而PS/2设备则要配合PC机才能读到准确的数据。

#### PS/2设备向PC机读取一个字节

(1)等待时钟线为高电平。

(2)判断数据线是否为低，为高则错误退出，否则继续执行。

(3)读地址线上的数据内容，共8个bit，每读完一个位，都应检测时钟线是否被PC机拉低，如果被拉低则要中止接收。

(4)读地址线上的校验位内容，1个bit。

(5)读停止位。

(6)如果数据线上为0(即还是低电平)，PS/2设备继续产生时钟，直到接收到1且产生出错号为止(因为停止位是1，如果PS/2设备没有读到停止位，则表明此次传输出错)。

(7) 输出应答位。

(8) 检测奇偶校验位，如果校验失败，则产生错误信号以表明此次传输出现错误。

(9)延时４5 μs，以便PC机进行下一次传输。

#### 读取一个位的步骤如下：

(1)延时20μs；

(2)把时钟线拉低

(3)延时４0μs

(4)释放时钟线

(5)延时20μs

(6)读数据线。

#### 发出应答位

(1)延时15μs；

(2)把数据线拉低；

(3)延时5μs；

(4)把时钟线拉低；

(5)延时４0μs；

(6)释放时钟线；

(7)延时5μs；

(8)释放数据线。

### 主机发送数据到设备

时钟都是由PS/2设备发出。主机要向设备发送数据，需要把时钟和数据线设置为“请求发送”状态。主机拉低信号线表示抑制通讯。

#### 主机发送数据到从机

(1) 主机拉低时钟线至少100us

(2) 吧数据线拉低，相当于请求发送信号，也就是起始位。

(3) 主机释放数据线。

(4) 主机释放数据线，并等待设备把时钟线拉低，主机拉低时钟到设备产生时钟时间不超过15ms

(5) 主机发送8位数据，将数据放置在数据线上，每设置一个数据都得等待设备吧时钟线拉高，在时钟上升沿时，对数据进行锁存，等待设备在把时钟拉低，如此反复8次发送数据。

(6) 主机释放数据线。如果如果主机在第11个时钟脉冲后不释放数据线，设备将继续产生时钟脉冲直到数据线被释放然后设备将产生一个错误。

(7) 等待设备把数据线拉低。

(8) 等待设备吧时钟线线拉低。

(9) 等待设备释放时钟线和数据线。

# LPC 总线

## 物理连接

LPC同时又是一个灵活的总线，它包括7个必选信号及6个可选信号。在实际应用中，并不是所有的信号都是必需的，可以根据情况对可选信号进行增减。

LPC 总线必选信号列表

|  |  |  |  |
| --- | --- | --- | --- |
| 信号 | 外设 | Host 设备 | 信号描述 |
| LAD[3:0] | I/O | I/O | 命令/数据/地址复用信号 |
| LFRAME# | I | O | 指示一个操作循环的开始 |
| LRESET# | I | I | 复位信号 |
| LCLK | I | I | 33MHz时钟 |

LPC总线可选信号列表

|  |  |  |  |
| --- | --- | --- | --- |
| 信号 | 外设 | Host 设备 | 信号描述 |
| LDRQ# | O | I | DMA master总线请求 |
| SERIRQ | I/O | I/O | 串行化中断信号 |
| CLKRUN# | OD | I/OD | Clock Run信号 |
| LPME# | OD | I/OD | 电源管理唤醒 |
| LPCPD# | I | O | Power Down |
| LSMI | OD | I | SMI 信号 |

**注: (1) OD 表示漏极开路; (2) 带#的信号线表示低电平有效；**

## 电气属性

上拉电阻的硬件要求。

|  |  |
| --- | --- |
| **Signal Name** | **Pull-Up** |
| LAD[3:0] | 15k - 100k ohm |
| LDRQ[1:0]# | 15k - 100k ohm |

## Memory, I/O and DMA Cycle

LPC总线支持多种总线Cycle，比如内存读、内存写、IO读、IO写、DMA读、DMA写等.一个Cycle通常按照如下的流程进行:

---主机Host有效FRAME#信号指示一个Cycle开始，并将相应的信息放在LAD[3:0]上.

---主机Host根据Cycle类型驱动相应的信息到LAD[3:0]上，比如DMA通道，访问地址等.同时Host也将Cycle的类型，读写的方向，传输数据大小等信息驱动到总线上。

---主机然后根据Cycle类型不同选择进行驱动数据或者是移交总线控制权.

---.外设获取总线后，将相应的数据驱动到LAD[3:0]上.表示该Cycle完成.

---外设释放总线控制权。至此该Cycle结束。

根据LPC信号线的不同组合及在整个Cycle中出现的位置，一个LPC总线Cycle可以划分为多个场/域(Field)，下面就分别对I0读写Cycle所涉及的Field进行简单说明。

### Start

这个场用于指示一个传输的开始或者结束。当FEAME#信号有效时所有的外设都要监视LAD[3:0]信号，并在FRAME#信号有效的最后一个时钟后进入START场。LAD[3:0]的值编码如下表。

|  |  |
| --- | --- |
| Bits[3:0] | 定义 |
| 0000 | 开始一个Cycle，用于外设Memory，IO，DMA操作 |
| 0001 | 保留 |
| 0010 | 相应 bus master0 |
| 0011 | 响应 Bus master1 |
| 0100-1100 | 保留 |
| 1101 | 开始Firmware Memory Read |
| 1110 | 开始Firmware Memory Write |
| 1111 | Stop/Abort 结束一个Cycle |

### Cycle Type / Direction (CyCTYPE + DIR)

该场由Host 驱动，对Cycle的传输类型（memory, IO, DMA）以及传输方向进行说明。LAD[0]在该场中被保留。作为外设应该忽略。可以得值见表。

|  |  |  |
| --- | --- | --- |
| Bit[3:2] | Bit[1] | 定义 |
| 00 | 0 | IO读 |
| 00 | 1 | IO写 |
| 01 | 0 | Memory 读 |
| 01 | 1 | Memory 写 |
| 10 | 0 | DMA 读 |
| 10 | 1 | DMA写 |
| 11 | X | 保留 |

### Size

该字段是一个时钟。 它在DMA传输中由主机驱动，在总线主存储器传输中由外设驱动，以确定要传输多少字节。 位[3：2]被保留，必须由驱动程序将其驱动为“ 00b”，并且必须被目标忽略。 其余位编码如下：

|  |  |
| --- | --- |
| Bit[1:0] | Size |
| 00 | 8 bit（1byte） |
| 01 | 16bit(2 bytes) |
| 10 | 保留 |
| 11 | 32bit(4 bytes) |

### Turn-Around (TAR)

TAR 场占用两个时钟宽度。Host驱动该场将总线控制权利移交给外部设备（进行数据读）；而外设驱动该场将总线控制权移交回来。

在第一个时钟相内，外设或Host驱动LAD[3:0]为1111b,在第二个时钟相内，外设或Host将LAD[3:0]置为高阻态。

### ADDR

这个场在进行IO操作时占用4个时钟，进行Memory操作时占用8个时钟，target 存取时由于Host 驱动，bus master 存取时由外设驱动。DMA操作不需要该场。当该场被驱动时，高位最先被驱动到总线上。比如，Memory操作时，第一个时钟包含了Address[31:28],最后一个时钟包含了Address[3:0]。

### Channel

这是一个时钟宽度的字段，由主机在DMA周期上驱动，以向外设指示已授予哪个DMA通道。 [2：0]位包含DMA通道号，[3]位包含基于ISA编码的TC（终端计数）线。 在目标或总线主控传输时，不会驱动该字段。

### DATA

该场占用两个时钟周期，用于传送一个字节数据。当数据流向外设时，该场由Host驱动;反之，当数据流向Host时，则该场由外设驱动。在传输过程的时候，低4位最先被驱动到总线上.具体的说，在第一个时钟，Data[3:0]被驰动，第二个时钟，Data[7:4]被驱动。

### SYNC

SYNC场用来加入等待状态。它占用时间为时钟周期的整数倍.在target或者DMA传输操作时，该场由外设驱动:在bus master操作时，由Host驱动。可能的组合见表

|  |  |
| --- | --- |
| Bits[3:0] | 定义 |
| 0000 | 准备 |
| 0001-0100 | 保留 |
| 0101 | 短等待 |
| 0110 | 长等待 |
| 0111-1000 | 保留 |
| 1001 | Ready More(DMA only) |
| 1010 | 错误 |
| 1011-1111 | 保留 |

当设备需要插入等待周期时，应先驱动为“0101"(短时等待)或者“0110"(长等待)，等到Ready状态来到时，设备或选择驱动为“0000"(准备好)，" 1010"(错误)或者“1001”(Ready More)。

### SYNC TimeOut

总线上通常可能发生一下几种潜在的错误:

1.当Host发起一个Cycle (Memory,IO,DMA)后，但是总线上没有设备驱动SYNC场，当Host检测到有3个连续的时钟内都没有响应时，就认为总线上没有外设响应此次Cycle操作.

Host驱动一个Cycle(Memory,IO,DMA )，一个设备驱动一个有效的SYNC场来插入等待(LAD[3:0]='0101b’或者’0110b')，但是却不完成该Cycle，这种情况在外设锁定的时候就发生了。在这种情况下，Host应采取以下措施以保护总线死锁：

·假如SYNC是，0101b'，那么最大SYNC时钟周期被假设为8。当Host检测到有多于8个时钟周期的SYNC场，那么Host将取消这个Cycle。

·假如SYNC是，0110b'，那么这里将没有最大SYNC长度的限制。外设必须设计有保护机制来完成这个Cycle。

当由于host来驱动SYNC时，因为延迟的原因，它可能不得不插入大量的等待周期，但外设不应该认为有time-out 发生。

### SYNC Error Indication

通过将LAD[3:0驱动为‘10l0b', SYNC协议允许外设报告一个错误。

假如Host从外设读取数据，数据仍将在下两个时钟内传输，这个数据可能是无效的，但外设仍将传输它。假如Host是写数据到外设中，那么该数据其实已经被传输了。

在多字节DMA传输中，一个错误SYNC将会终止这个Cycle。因此，如果Host正企图进行一个4字节的传输，但当在传输第一个字节时就出错了，那么剩下的3个字节将不会被传输。

在 Host接收到一个错误SYNC后，它可以有多种处理方式.对于ISA总线，它可以有效IOCHK#信号。

### LFRAME#

Host采用LFRAME#信号来开始一个Cycle，当一个abort或time-out出现时来终止一个Cycle，该信号也被外设用来决定什么时候监视总线。

这个信号通常是提醒设备LAD[3:0]线包含了Cycle开始或结束的有关信息，此时所有外设需要监视总线并决定该Cycle是否与自己有关。这样做的好外在于FRAME#信号可以使设备内部进入低功耗状态。假如一个外设没有被访问时，它就不需要监视总线，这样一来，设备就可以关闭内部时钟或者将状态机停下来。

当外设监测到FRAME#有效后，他们在下一个时钟立即停止驱动L.AD[3:0)信号线，并继续监视下一个总线信息。

### Start of Cycle

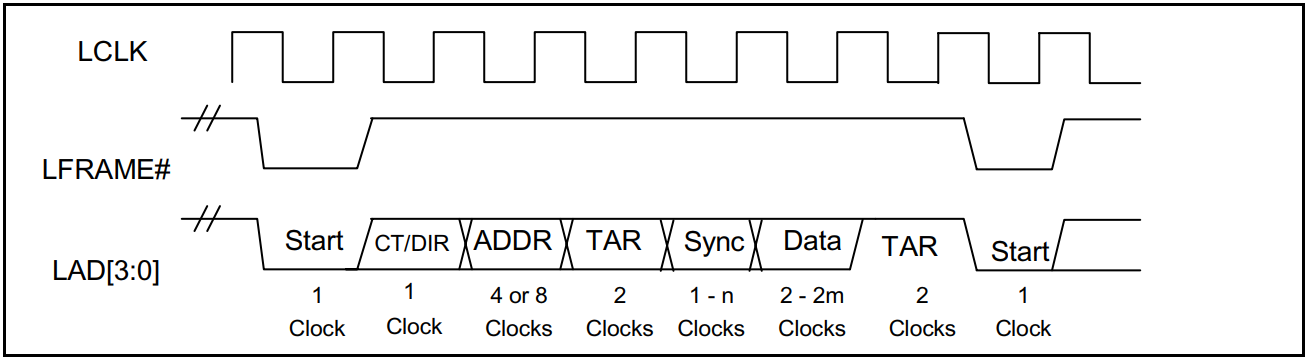
不同的Cycle都以相同的方式开始。

Host有效LFRAME#一个或多个时钟，并且驱动一个START值到LAD[3:0]信号线上。一旦发现LFRAME#信号有效后，所有的外设将停止驱动LAD[3:0],即使此时数据正在传输中。

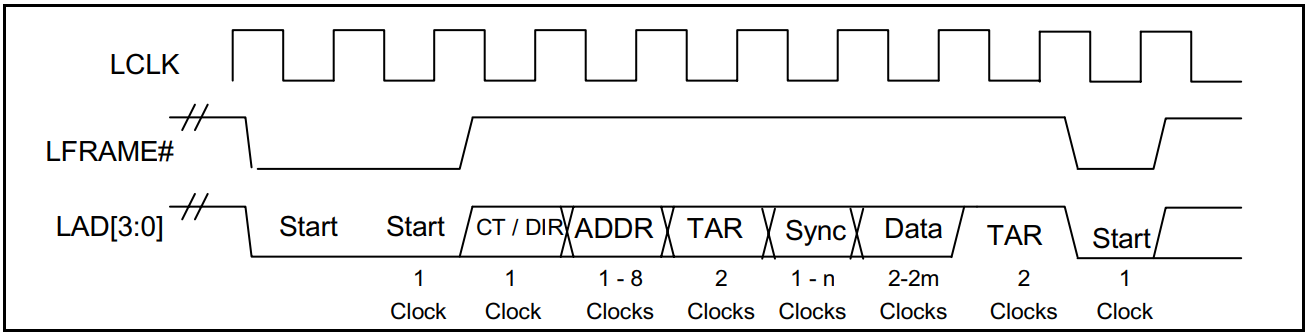
在Cycle的开始阶段，Host有可能保持FRAME#信号有效超一个时钟周期，START值也有可能发生变化。因此，外设必须总是使用最后一个值。比如，假如FRAME#有效两个时钟周期，那么外设应该忽略第一个时钟时的值而使用第二个时钟周期的值。

外设监视LAD[3:0]值并在LFRAME#有效时得到相应的START值，假如外设支持些种类型的Cycle，它应该试图解析Cycle余下的部分。假如不支持这个特定的类型，它将忽略Cycle余下的部分，直到FRAME#再次有效。

当最后一个START值被驱动后，Host就准备开始一个Cycle，并无效FRAME#信号。外设则根据START值开始进入工作状态。



典型的FRAME#时序

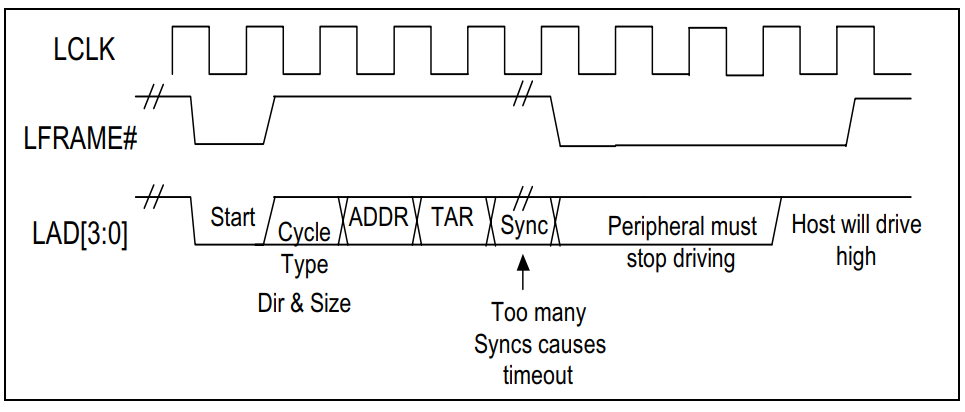


拓展的FRAME#时序

### Abort机制

FRAME#信号用来将一个外设kick出总线接口.Host驱动LFRAME#信号并驱动一个START值，’1111b’来产生一个Abort。因为有效LFRAME#通常表示一个Cycle的开始，所以即使当一个外设正在进行一个Cycle时，它必须认为Host正在取消当前的Cycle并且立刻停止驱动LAD[3:0]信号。为了确保一个Abort被正确识别，Host需要保持LFRAME#有效至少4个连续的时钟周期并且在第4个时钟内将LAD[3:0]驱动为’1111b'. Abort完成后，LFRAME#应被驱动为无效至少一个时钟周期。

Abort 通常发生在SYNC超时，Cycle 无响应或者一个设备驱动一个保留SYNC字的情况下。Abort被用来结束当前的Cycle以保证所有的设备进入下一个Cycle的准备状态。Abort也可以用来当作软件复位来使用。在使用长等待SYNC并不建议使用Abort机制。



Abort 机制

### 内存读写位域定义

在这些情况下，内存读取或写入周期用于内存映射的设备。 同步时间将取决于设备的速度。

|  |  |  |
| --- | --- | --- |
| **Field** | **# Clocks** | **Comment** |
| START | 1 | **Start of Cycle:** ‘0000b’ to indicate a start of a cycle. |
| CYCTYPE + DIR | 1 | **Cycle Type:** Indicates the type of cycle.Bits 3:2 must be ‘01b’ for memory cycle.Bit 1 indicates the direction of the transfer: ‘0b’ for read, ‘1b’ for write.Bit 0 is reserved. |
| ADDR | 8 | **Address Phase for Memory Cycle:** This is the 32-bit memory address.It is transferred with the most significant nibble first. |
| TAR | 2 | **Turn-Around Time:** The last component driving LAD[3:0] will drive it to “1111b” during the first clock, and tri-state it during the second clock. |
| SYNC | N | **Sync:** Allows peripheral or host to synchronize (add wait-states). Generally, the peripheral or host drives 0101 or 0110 until no more wait states are needed.At that point it will drive 0000.All other combinations are reserved.If the host sees a reserved combination, it is allowed to abortthe transfer. |
|  |  | |  |  | | --- | --- | | **Bits** | **Indication** | | 0000 | Sync Achieved with no error. | | 0101 | Indicates that Sync not Achieved yet, but the part is driving the bus. | | 0110 | Indicates that Sync not Achieved yet, but the part is driving the bus, and expects a long Sync. | | 1010 | Special Case: peripheral indicating errors.See Section 4.2.1.10 for details. | |
| Data | 2(1 byte) | **Data Phase:** The data byte is transferred least significant nibble first (D[3:0] on LAD[3:0], then D[7:4] on LAD[3:0]). |

### 主机启动内存读

|  |  |  |
| --- | --- | --- |
| **Memory Read** | **Driven By** | **Clocks** |
| START | Host | 1 |
| CYCTYPE + DIR | Host | 1 |
| ADDR | Host | 8 |
| TAR | Host | 2 |
| SYNC | Peripheral | 5 |
| DATA | Peripheral | 2 |
| TAR | Peripheral | 2 |
| **Total Clocks** |  | **21** |

在以上示例中，选择了5个时钟的SYNC值来读取第一个字节。 该值基于对EPROM等存储组件的已知访问时间，典型访问时间为120ns。

### 主机启动内存写

|  |  |  |
| --- | --- | --- |
| **Memory Read** | **Driven By** | **Clocks** |
| START | Host | 1 |
| CYCTYPE + DIR | Host | 1 |
| ADDR | Host | 8 |
| DATA | Host | 2 |
| TAR | Host | 5 |
| SYNC | Peripheral | 2 |
| TAR | Peripheral | 2 |
| **Total Clocks** |  | **17** |

在上面的示例中，假定外围设备不需要等待状态，因为存储设备将数据发布。 因此，只需准备就绪同步指示器

### IO读写域定义

在这些情况下，I / O读或写周期用于外设。 这些通常用于寄存器或FIFO访问，并且通常具有最小的同步时间。 字节之间的最小等待状态数为1。增强的并行端口周期将取决于连接器的速度。

|  |  |  |
| --- | --- | --- |
| START | 1 | **Start of Cycle:** ‘0000b’ to indicate a start of a cycle. |
| CYCTYPE +DIR | 1 | **Cycle Type:** Indicates the type of cycle.Bits 3:2 must be ‘00b’ for I/O cycle.Bit 1 indicates the direction of the transfer: ‘0b’ for read, ‘1b’ for write.Bit 0 is reserved and must be ignored by the peripheral. |
| ADDR | 4 | **Address Phase for I/O Cycle:** 16 bit I/O address transferred with the most significant nibble first. |
| TAR | 2 | **Turn-Around Time:** The last component driving LAD[3:0] will drive it to ‘1111b’ during the first clock, and tri-state it during the second clock. |
| Sync | N | **Synchronize:** Allows peripheral add wait-states.The peripheral drives ‘0101b’ or ‘0110b’ until no more wait-states are needed.At that point it will drive ‘0000b’.All other combinations are reserved. |
|  |  | |  |  | | --- | --- | | **Bits** | **Indication** | | 0000 | Sync Achieved with no error. | | 0101 | Indicates that Sync not Achieved yet, but the part is driving the bus. | | 0110 | Indicates that Sync not achieved yet, but the part is driving the bus, and expects a long Sync (probably due to EPP cycle). | | 1010 | Special Case: peripheral indicating errors.See Section 4.2.1.10for details. | |
| Data | 2 (1 byte) | **Data Phase:** The data byte is transferred with the least significant nibble first: (Data[3:0] on LAD[3:0], then Data[7:4] on LAD[3:0] ). |

### 主机启动IO读操作

|  |  |  |
| --- | --- | --- |
| **Memory Read** | **Driven By** | **Clocks** |
| START | Host | 1 |
| CYCTYPE + DIR | Host | 1 |
| ADDR | Host | 4 |
| TAR | Host | 2 |
| SYNC | Peripheral | 1 |
| DATA | Peripheral | 2 |
| TAR | Peripheral | 2 |
| **Total Clocks** |  | **13** |

在以上示例中，假定外设没有传递读取数据所需的等待状态，因为它很可能是从内部FIFO中拉出的。

### 主机启动IO写操作

|  |  |  |
| --- | --- | --- |
| **Memory Read** | **Driven By** | **Clocks** |
| START | Host | 1 |
| CYCTYPE + DIR | Host | 1 |
| ADDR | Host | 4 |
| DATA | Host | 2 |
| TAR | Host | 2 |
| SYNC | Peripheral | 1 |
| TAR | Peripheral | 2 |
| **Total Clocks** |  | **13** |

在以上示例中，假定外围设备上不需要等待状态。因此，所必要的是就绪同步指示器。

## Firmware Memory Cycle

本节介绍了用于系统BIOS固件的内存周期类型。

### start

该一个时钟字段指示一个周期的开始。 在最后一个时钟上LFRAME＃被采样为低电平有效。 下表中显示了用于循环的两个开始字段。

|  |  |
| --- | --- |
| AD[3:0] | 描述 |
| 1101 | Fireware memory Read |
| 1110 | Fireware memory Write |

### IDSEL (Device Select)

该一个时钟字段用于指示正在选择多个固件组件中的哪个。 将在此时钟期间通过AD [3：0]传输的四个位与绑定到固件组件引脚上的值进行比较。 如果存在匹配项，则固件组件将继续对cycle进行解码，以确定读取时请求了哪些字节或写入时要更新哪些字节。 如果不匹配，则固件组件可能会丢弃剩余的周期并进入待机电源状态。

### MADDR (Memory Address)

这是一个7时钟字段，提供28位存储器地址。 每个存储设备最多可容纳256MB，总共4GB的可寻址空间,先传高位。

### MSIZE (Memory Size)

下表中显示了此字段的编码。所有支持固件存储周期的设备都需要单字节传输。 多字节传输是可选的，主机和设备都需要一种机制来通知软件可用的支持（即功能寄存器位）。 注意，此机制的定义超出了本规范的范围。对于多字节写入，FLASH存储设备可能需要增加的编程电压（Vpp）才能执行4字节存储写入。如果没有更高的编程电压，但外设需要支持更高的编程电压以支持4字节存储器写，则应使用1或2字节存储器写。

|  |  |  |
| --- | --- | --- |
| Bits | Direction | Size of Transfer |
| 0000 | R/W | 1 byte |
| 0001 | R/W | 2bytes(可选)，访问必须与WORD边界对齐 |
| 0010 | R/W | 4bytes(可选)，访问必须与WORD边界对齐 |
| 0011 |  | Reserved |
| 0100 | R | 16 bytes,访问必须与16 bytes边界对齐 |
| 0101-0110 |  | Reserved |
| 0111 | R | 128 bytes,访问必须与128 bytes边界对齐 |
| 1000-1111 |  | reserved |

### TAR

TAR 场占用两个时钟宽度。Host驱动该场将总线控制权利移交给外部设备（进行数据读）；而外设驱动该场将总线控制权移交回来。

在第一个时钟相内，外设或Host驱动LAD[3:0]为1111b,在第二个时钟相内，外设或Host将LAD[3:0]置为高阻态。

与2.3.4描述一致。

### SYNC

SYNC场用来加入等待状态。它占用时间为时钟周期的整数倍.在target或者DMA传输操作时，该场由外设驱动:在bus master操作时，由Host驱动。可能的组合见表

|  |  |
| --- | --- |
| Bits[3:0] | 定义 |
| 0000 | 准备 |
| 0001-0100 | 保留 |
| 0101 | 短等待 |
| 0110 | 长等待 |
| 0111-1000 | 保留 |
| 1001 | Ready More(DMA only) |
| 1010 | 错误 |
| 1011-1111 | 保留 |

当设备需要插入等待周期时，应先驱动为“0101"(短时等待)或者“0110"(长等待)，等到Ready状态来到时，设备或选择驱动为“0000"(准备好)，" 1010"(错误)或者“1001”(Ready More)。

与2.3.8描述一致。

### DATA

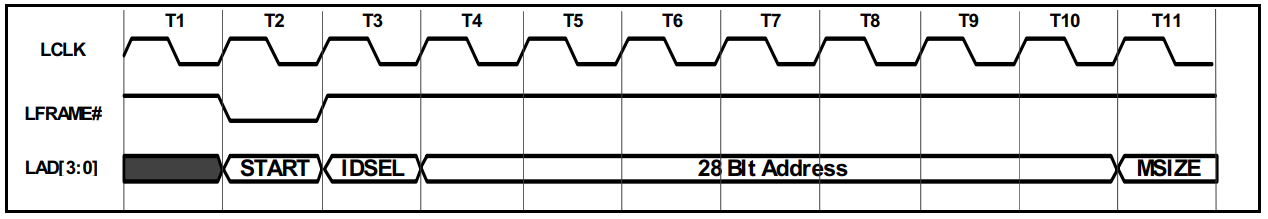
该字段为（2 \* N）个时钟宽度，表示由MSIZE字段确定的“ N”个数据字节的传输。 当数据流向外设时，主机在固件存储周期中驱动它（写周期），当数据流向主机时，则由外设驱动它（读周期）。每个字节的数据低位在前。这意味着对于每个字节，在第一个时钟上驱动Data [3：0]，在第二个时钟上驱动Data [7：4]。 这也意味着每个后续数据字节的地址通过传输顺序递增。

### Protocol

固件存储周期使用一系列事件，这些事件以START字段（LFRAME＃处于活动状态，并带有适当的AD [3：0]组合）开始，并以数据传输结束。

### Preamble

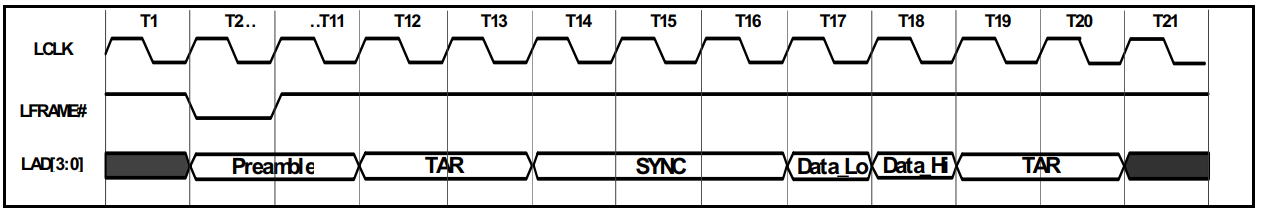
固件存储器周期的初始化如下图所示。固件存储器事务以LFRAME＃变低并在AD [3：0]上驱动START字段开始。 对于固件内存读取周期，START字段必须为“1101b”；对于固件内存写周期，START字段必须为“1110b”。在START字段之后是IDSEL字段。该字段就像片选一样，它指示哪个设备应响应当前事务。接下来的七个时钟是28位地址，从该地址开始在所选器件中进行读取。接下来，MSIZE值指示要传输的字节数。



Firmware Memory Cycle Preamble

### Firmware Memory Read Cycle

对于读取周期在Preamble之后，主机驱动TAR字段以将总线所有权授予固件组件。 在TAR阶段的第二个时钟之后，目标设备采用总线并开始驱动SYNC值。当目标设备准备就绪时，它将驱动数据（首先是最低有效字节），直到传输完所有数据，然后再进行TAR循环，以将控制权交还给主机。

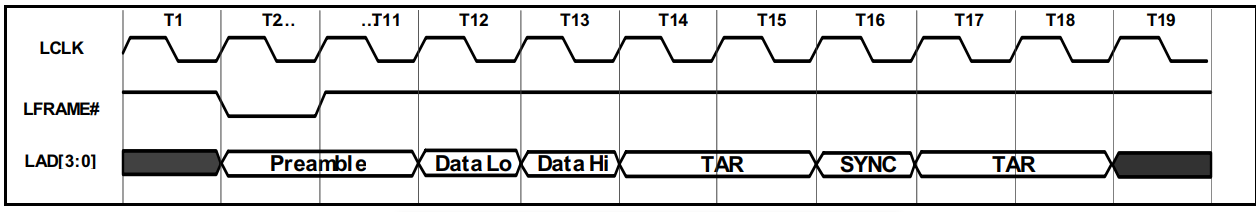


Firmware Memory Cycle Single Byte Read

上图显示了一个设备传输1字节数据并需要3个SYNC时钟来访问数据的过程，因为访问时间可以在地址阶段完成后开始，因此TAR阶段的两个时钟可以被视为设备访问时间的一部分。例如，访问时间为120ns的设备可以为SYNC阶段的时钟1和2设置“0101b”，为SYNC阶段的最后一个时钟设置“ 0000b”。 如果设备在Preamble阶段结束时开始访问，则相当于5个时钟的访问时间。一旦实现了SYNC，设备便会在随后的两个时钟中返回数据，并通过TAR阶段将总线所有权返还给主机。对于多字节读取，附加数据字节以紧接第一个字节之后的顺序传输（即在上图中的时间T18和T19之间），然后是TAR阶段。

### Firmware Memory Write Cycles

对于读取周期在Preamble之后，所有支持固件内存写周期的设备都必须支持单字节写操作。 进一步支持多字节固件存储器写周期是可选的。



Firmware Memory Cycle Single Byte Write

上图显示了一个固件存储器写周期，其中传输了一个字节。 主机将MSIZE值声明为0。在地址传输之后，2时钟数据阶段开始。在数据阶段之后，总线所有权通过TAR周期转移到固件组件。在TAR阶段之后，设备必须声明SYNC值“ 0000b”（就绪）或“ 1010b”（错误），以指示是否已接收到数据。有关可能的错误的说明，下部分有说明错误码。 然后，在另一个TAR阶段将总线所有权交还给主机。 对于多字节写入，附加数据字节紧接在第一个字节之后（即在上图中的时间T13和T14之间）以顺序方式传输，然后是第一个TAR阶段。

固件存储器写入仅允许一个时钟进入SYNC阶段。 固件存储器写周期结束时的TAR + SYNC + TAR阶段必须恰好是5个时钟。

### Error Reporting

固件组件不得通过LPC接口报告错误情况。 它只能报告等待状态和“就绪”状态。 它可以选择在内部记录要调试的错误，但一定不能通过LPC接口本身发出错误信号

### 固件存储器字段定义

固件存储器的读或写周期旨在用于PC系统引导固件，尽管它们可用于任何存储器周期。 同步时间将取决于设备的速度。

ADDR字段为28位，并以最高有效半字节优先发送。 尽管支持完整的28位寻址，但是存储设备可能支持的位数远少于此。

|  |  |  |
| --- | --- | --- |
| **Field** | **# Clocks** | **Comment** |
| START | 1 | **Start of Cycle:** ‘1101b’ to indicate a start of a firmware memory read cycle or ‘1110b’ to indicate start of a firmware memory write cycle. |
| IDSEL | 1 | **ID Select:** Selects the targeted firmware component based on device pin straps. |
| ADDR | 7 | **Address Phase for Firmware Memory Cycle:** This is the 28-bit memory address.It is transferred with the most significant nibble first.The device selected by IDSEL field uses this to address its internal memory array. |
| MSIZE | 1 | **Memory Size:** Indicates the number of bytes (N) to be transferred.See Table 4 above. |
| TAR | 2 | **Turn-Around Time:** The last component driving LAD[3:0] will drive it to “1111b” during the first clock, and tri-state it during the second clock. |
| SYNC | N | **Sync:** Allows peripheral or host to synchronize (add wait-states). Generally, the peripheral or host drives 0101 or 0110 until no more wait states are needed.At that point it will drive 0000.All other combinations reserved.If the host sees a reserved combination, it is allowed to abort the transfer. |
|  |  | |  |  | | --- | --- | | **Bits** | **Indication** | | 0000 | Sync Achieved with no error. | | 0101 | Indicates that Sync not Achieved yet, but the part is drivingthe bus | | 0110 | Indicates that Sync not Achieved yet, but the part is drivingthe bus, and expects a long Sync. | |
| Data | 2 \* N (N bytes) | **Data Phase:** The data byte is transferred least significant nibble first (D[3:0] on LAD[3:0], then D[7:4] on LAD[3:0]), and so on. |

### 主机启动固件存储器读操作

|  |  |  |
| --- | --- | --- |
| **Single Byte Memory Read** | **Driven By** | **Clocks** |
| START | Host | 1 |
| IDSEL | Host | 1 |
| ADDR | Host | 7 |
| MSIZE | Host | 1 |
| TAR | Host | 2 |
| SYNC | Peripheral | 3 |
| DATA | Peripheral | 2 |
| TAR | Peripheral | 2 |
| **Total Clocks** |  | **19** |

|  |  |  |
| --- | --- | --- |
| **128 Byte Memory Read** | **Driven By** | **Clocks** |
| START | Host | 1 |
| IDSEL | Host | 1 |
| ADDR | Host | 7 |
| MSIZE | Host | 1 |
| TAR | Host | 2 |
| SYNC | Peripheral | 3 |
| DATA | Peripheral | 256 |
| TAR | Peripheral | 2 |
| **Total Clocks** |  | **273** |

在上面的示例中，选择了3个时钟的SYNC值来读取第一个字节。

|  |  |  |
| --- | --- | --- |
| **4 Byte Memory Write** | **Driven By** | **Clocks** |
| START | Host | 1 |
| IDSEL | Host | 1 |
| ADDR | Host | 7 |
| MSIZE | Host | 1 |
| DATA | Host | 8 |
| TAR | Host | 2 |
| SYNC | Peripheral | 1 |
| TAR | Peripheral | 2 |
| **Total Clocks** |  | **23** |

### 主机启动固件存储器写操作

|  |  |  |
| --- | --- | --- |
| **Single Byte Memory Write** | **Driven By** | **Clocks** |
| START | Host | 1 |
| IDSEL | Host | 1 |
| ADDR | Host | 7 |
| MSIZE | Host | 1 |
| DATA | Host | 2 |
| TAR | Host | 2 |
| SYNC | Peripheral | 1 |
| TAR | Peripheral | 2 |
| **Total Clocks** |  | **17** |

在以上示例中，假定外围设备上不需要等待状态，因为存储设备将数据发布。 因此，所有必要的是就绪同步指示器。

## DMA

在LPC总线上，允许有8个DMA通道。

### DMA数据请求

LPC上的DMA通过使用来自外设的LDRQ＃线和来自主机的LAD [3：0]上的特殊编码来处理。 LPC接口支持Single, Demand, Verify, and Increment模式。 不支持Block, decrement, and cascade modes模式。 通道的0 -3是8位通道。 频道的5 -7是16位频道。 通道4被保留为通用总线主控器请求.

LPC接口上支持一种新的32位传输模式，该模式可用于根据每个请求从8位或16位通道传输多个字节，以提高吞吐量。

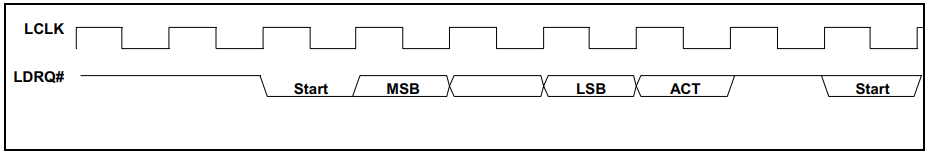
LDRQ＃与LCLK同步。其中LDRQ 接的上拉电阻，所以空闲状态为高电平，外设使用以下串行编码序列：

•通过将LDRQ＃置为低电平（起始位）来启动序列。 在空闲条件下，LDRQ＃为高。

•接下来的3位包含编码的DMA通道号（MSB在前）。

•下一位（ACT）指示对指示的DMA通道的请求是有效还是无效。 ACT位将为1（高）以指示它是否处于活动状态，而为0（低）以指示其是否处于活动状态。 ACT低的情况很少见，仅用于指示该信道的先前请求已被放弃。 有关放弃DMA请求的原因，请参见第6.3节。

•在启用/禁用指示之后，LDRQ＃信号必须至少在1个时钟周期内变为高电平。 在一个时钟之后，可以将LDRQ＃信号拉低到下一个编码序列。



DMA Request Assertion through LDRQ#

### 取消DMA请求

可以通过两种方式取消激活DMA请求：在错误情况下，通过将“ ACT”位设置为“ 0”来发送LDRQ＃消息，或者通常在DMA传输期间通过SYNC字段。

### DMA 传输定义

基本DMA传输的一般流程如下：

1.主机通过声明LFRAME＃在LAD [3：0]上声明“ 0000b”来开始传输。

2.Host根据DMA传输方向声明DMA的“周期类型”。

3.主机声明频道号，如果适用，还声明终端数。

4.主机指示传输的大小：8位，16位或32位。

5，如果DMA读取：

•主机驱动数据的前8位并调转总线。

•外设使用有效的SYNC确认数据。

•如果进行16位传输，则对接下来的8位重复该过程。

•如果是32位传输，则每8位数据重复该过程两次。

6，如果是DMA写：

•主机扭转总线并等待数据。

•外设通过SYNC指示数据准备就绪，并传输第一个字节。

•如果进行16位传输，则外设指示数据就绪并传输下一个字节。

•如果是32位传输，则外设再重复SYNC +数据传输两次。

7.外围设备绕过总线。

上述DMA传输中使用的字段显示为：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| START | 1 | | **Start of Cycle:** ‘0000b’ for DMA.The start condition is the last clock ofLFRAME# active. | | | | |
| CYCTYPE +DIR | 1 | | **Cycle Type:** Bits [3:2] are ‘10b’ to indicate DMA.Bit 2 indicates the transfer direction: ‘0’ for DMA read (host to peripheral), and ‘1’ for DMA write (peripheral to host).Bit 0 is reserved and must be ignored by the peripheral. | | | | |
| SIZE | 1 | | **Size of Transfer:** Bits [3:2] are reserved and must be ignored by the peripheral. | | | | |
|  | | | LAD[1:0] | | Transfer size | | |
| **00** | | **8 bit** | | |
| **01** | | **16 bit** | | |
| **10** | | **Reserved** | | |
| **11** | | **32 bit** | | |
| 32 bit transfers are optional.If the host drives a size field of ‘10b’, the peripheral must ignore the transfer. | | | | |
| TAR | | 1 | Channel #: Used only for DMA cycles to indicate channel # being granted.The LAD[2:0] bits indicate the channel number being granted, and LAD[3] indicates the TC bit. See Section 6.4.1 for a description of terminal count.The encoding on LAD[2:0] for channel number is as follows: | | | | |
|  | | | Bits | Indication | Bits | | Indication |
| 000 | Channel 0 | 100 | | Channel 4 |
| 001 | Channel 1 | 101 | | Channel 5 |
| 010 | Channel 2 | 110 | | Channel 6 |
| 011 | Channel 3 | 111 | | Channel 7 |
| If a peripheral sees an encoding for a DMA channel that it did not request, or an encoding for channel 4, it must ignore the cycle | | | | |
| SYNC | | 1 - N | Sync: Allows peripheral or host to synchronize (add wait-states).Generally,the peripheral or host drives ‘0101b’ or ‘0110b’ until no more wait-states areneeded.At that point it will drive ‘0000b’.All other combinations arereserved | | | | |
|  | | | Bits | | | Indication | |
| 0000 | | | Sync achieved with no error.Also indicates no more transfers desired for that channel, and DMA request is de-asserted. | |
| 0101 | | | Part indicating wait states. | |
| 0110 | | | Part indicating wait states, and many wait states will be added. | |
| 1010 | | | Sync achieved with error.Also indicates no more transfers desired for that channel, and DMA request is de-asserted. | |
| 1001 | | | Sync achieved with no error and more DMA transfers desired to continue after this transfer. | |
| If the host sees a SYNC value that is not one of the above combinations, it is allowed to abort the transfer. | | | | |
| DATA | | 1, 2, 4 bytes | Data Phase: Data bytes transferred with the least significant nibble first (D[3:0] on LAD[3:0], then D[7:4] on LAD[3:0]).For transfers greater than 1 byte, the bytes are sent with least significant byte first. | | | | |

### Terminal Count

终端计数通过LAD [3]在与DMA通道在LAD [2：0]上进行通信的同一时钟上进行通信。 此字段是“频道”字段。 终端计数根据传输的大小指示传输的最后一个字节。

### 取消传输

传输结束消息通过外设发送的特殊SYNC字段传送到主机。 LPC设备禁止通过取消置位LDRQ＃来尝试表示传输结束。

DMA代理在要传输的每个数据字节上使用SYNC编码，它向主机指示这是传输的最后一个字节还是请求更多的字节。 为了指示传输的最后一个字节，外设使用SYNC值“ 0000b”（就绪）或“ 1010b”（错误）。 这些编码告诉主机这是在DMA读取（主机到外围设备）上传输的最后一段数据，或者后面的字节是在DMA写入（外围设备到主机）上传输的最后一段数据。

当主机看到这两种编码中的一种时，它将在该字节之后结束DMA传输，并将DMA请求置为无效。 因此，如果主机指示16位传输，则外设可以通过将SYNC值指示为“ 0000b”或“ 1010b”来结束一个字节的传输。 主机将不会尝试传输第二个字节，并将在内部取消声明DMA请求。 对于32位传输中的任何字节也是如此。 因此，这使外设可以终止DMA传输。

如果外设希望保持DMA请求处于活动状态，则它使用SYNC值“ 1001b”（就绪以及更多数据）。这告诉8237控制器，在传输了当前字节之后，将请求更多数据字节，因此主机会将DMA请求保持对8237控制器的激活状态。 因此，在8位传输大小上，如果外设向主机指示SYNC值为“ 1001b”，则数据将被传输，并且DMA请求将对8237控制器保持有效。 稍后，主机将返回另一个START –>CYCTYPE-> CHANNEL-> SIZE等组合，以发起另一次到外围设备的传输。

### SYNC field / LDRQ# Rules

由于LPC上的DMA传输是通过LDRQ＃断言消息请求的，并且在DMA传输过程中通过SYNC字段结束，因此外设在从DMA通道启动背对背传输时必须遵循以下规则。

### DMA读操作

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DMA Read** | **Driven By** | **8 Bit** | **16 Bit** | **32 Bit** |
| START | Host | 1 | 1 | 1 |
| CYCTYPE + DIR | Host | 1 | 1 | 1 |
| CHANNEL | Host | 1 | 1 | 1 |
| SIZE | Host | 1 | 1 | 1 |
| DATA | Host | 2 | 4 | 8 |
| TAR | Host | 2 | 4 | 8 |
| SYNC | Peripheral | 1 | 2 | 4 |
| TAR | Peripheral | 2 | 4 | 8 |
| **Total Clocks** | | **11** | **18** | **32** |

在上面的示例中，假定当主机将数据传输到外围设备时，它已准备好立即接收数据，而不必添加等待状态。 每个字节都有一个SYNC-> TAR。 该图未显示主机从主内存中获取数据的额外延迟。因此，有效带宽将小于所显示的带宽，并减少一个由总体系统延迟确定的因素。

### DMA写操作

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DMA Read** | **Driven By** | **8 Bit** | **16 Bit** | **32 Bit** |
| START | Host | 1 | 1 | 1 |
| CYCTYPE + DIR | Host | 1 | 1 | 1 |
| CHANNEL | Host | 1 | 1 | 1 |
| SIZE | Host | 1 | 1 | 1 |
| TAR | Host | 2 | 2 | 2 |
| SYNC | Peripheral | 1 | 2 | 4 |
| DATA | Peripheral | 2 | 4 | 8 |
| TAR | Peripheral | 2 | 2 | 2 |
| **Total Clocks** | | **11** | **14** | **20** |

在上面的示例中，假定外围设备具有执行传输所需的所有数据，因此不需要添加等待状态。 此外，DMA写操作假定主机有足够的空间来容纳请求的数据，否则主机将不会断言传输要开始的宽度。例如，如果只有空间进行16位传输，则不会要求进行32位传输。

因此，外设无需在每个字节之间切换总线，而无需查看主机是否可以接受更多的传输。 这样可以在16位和32位传输时显着增加整体带宽。

### 关于16和32位DMA的其他说明

在默认操作下，主机将仅在8位通道上执行8位传输，而在16位通道上执行16位传输。为了在8位通道上启用16位和32位传输，并在16位通道上启用32位传输，外设必须与系统BIOS通信，以允许更大的传输大小。如果主机具有此功能，则BIOS将对主机进行编程以尝试更大的传输大小。

如果主机可以执行更大的传输大小，则它将在所有情况下尝试执行以下操作：

•对于8位通道（0-3），如果缓冲区不是在32位边界上启动，则主机将执行8位传输（1、2或3次），直到达到32位边界。届时它将开始尝试32位传输。这是为了使其内部缓冲区与32位内存对齐。

•对于16位通道（5-7），如果缓冲区不是在32位边界上启动的，则主机将执行16位传输，然后继续进行32位传输。

•对于8位通道（0-3），如果在终端计数之前剩余一个字节，则主机将不会尝试16位传输。相反，它将执行8位传输。

•对于8位通道（0-3），如果在终端计数之前剩余三个字节，则主机将不会尝试执行32位传输。相反，它将执行三个8位传输。这些8位传输中的最后一个将设置TC。

•对于16位通道（5-7），如果在达到终端计数之前仅剩下两个字节，则主机将不会尝试执行32位传输。相反，它将使用TC设置执行一次16位传输

## Bus Master Protocol

LPC主机和外围设备均可选，但强烈建议主机使用

总线主控器访问通过LDRQ＃信号线“ 100b”上的保留编码来执行。 通过使用这种保留的编码，外围设备可以创建任意数量的总线主控，而不仅限于系统中DMA通道的数量。

### 周期格式和时间

总线主机START字段与总线主机0（'0010b'）或总线主机1（'0011b'）相关联。 目前，LPC接口上仅支持两个总线主控。 发送此字段后，主机执行TAR将控制权转移到外围设备。

### Bus Master Cycle 字段定义

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **# Clocks** | **Comment** | |
| START | 1 | **Start of Cycle:** ‘0010b’ or ‘0011b’ to indicate a start of a cycle for one of the two supported bus masters. | |
| CYCTYPE + DIR | 1 | **Cycle Type:** Indicates the type of cycle.Bits (3:2) must be ‘01b’ for memory cycle or ‘00b’ for I/O cycle.Bit 1 indicates the direction of the transfer: ‘0b’ for read, ‘1b’ for write.Bit 0 is reserved and must be ignored. | |
| SIZE | 1 | **Size of Transfer:** Indicates the size of the transfer.Bits (3:2) are reserved and must be ignored by the host. | |
|  | | LAD[1:0] | Transfer size |
| 00 | 8 bit |
| 01 | 16 bit |
| 10 | Reserved |
| 11 | 32 bit |
| TAR | 2 | Turn-Around Time: The last component driving LAD[3:0] will drive it to‘1111b’ during the first clock, and tri-state it during the second clock. | |
| ADDR | 4 OR 8 | Address Phase for I/O or Memory Cycle: This is either the 16- bit I/O address or the 32-bit memory address.It is transferred with the most significant nibble first.Note that the address must be aligned to the size of the data transfer.Example: | |
|  | | Transfer Size | Address[1:0] |
| 8 ibt | XX (all combinations allowed) |
| 16 bit | x0 (bit 0 must be 0) |
| 32 bit | 00 (bits 1 and 0 must be 0 |
| The examples above state that a 16-bit transfer must be 16-bit aligned, and a 32-bit transfer must be 32 bit aligned | |
| SYNC | N | Sync: Allows host to add wait-states.The host drives ‘0110b’until no more wait-states are needed.At that point it will drive‘0000b’.All other combinations reserved. | |
|  | | Bits | Inducation |
| 0000 | Sync Achieved with no error. |
| 0110 | Indicates that Sync not Achieved yet, but the host is driving the bus, and expects a long Sync. |
| 1010 | Special Case: peripheral indicating errors. |
| DATA | 1,2, or 4 bytes | Data Phase: Data bytes transferred with the least significant nibble first (D[3:0] on LAD[3:0], then D[7:4] on LAD[3:0]).For transfers greater than 1 byte, the bytes are sent with the least significant byte first. | |

### 外设启动内存读操作

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Memory Read Cycle** | **Driven By** | **8 Bit** | **16 Bit** | **32 Bit** |
| START | Host | 1 | 1 | 1 |
| TAR | Host | 2 | 2 | 2 |
| CYCTYPE + DIR | Peripheral | 1 | 1 | 1 |
| ADDR | Peripheral | 8 | 8 | 8 |
| SIZE | Peripheral | 1 | 1 | 1 |
| TAR | Peripheral | 2 | 2 | 2 |
| SYNC | Host | 6 | 6 | 6 |
| DATA | Host | 2 | 4 | 8 |
| TAR | Host | 2 | 2 | 2 |
| **Total Clocks** |  | **25** | **27** | **31** |

在上面的示例中，选择了6个时钟作为主机启动的等待状态。 该假设基于合理的时间，以将地址传送到PCI总线并接收读取响应。 在负载较重的系统中，此长度可能会更长，并且肯定不会缩短。

### 外设启动内存写操作

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Memory Write Cycle** | **Driven By** | **8 Bit** | **16 Bit** | **32 Bit** |
| START | Host | 1 | 1 | 1 |
| TAR | Host | 2 | 2 | 2 |
| CYCTYPE + DIR | Peripheral | 1 | 1 | 1 |
| ADDR | Peripheral | 8 | 8 | 8 |
| SIZE | Peripheral | 1 | 1 | 1 |
| DATA | Peripheral | 2 | 4 | 8 |
| TAR | Peripheral | 2 | 2 | 2 |
| SYNC | Host | 6 | 6 | 6 |
| TAR | Host | 2 | 2 | 2 |
| **Total Clocks** |  | **25** | **27** | **31** |

### 外设启动IO读操作

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **I/O Read Cycle** | **Driven By** | **8 Bit** | **16 Bit** | **32 Bit** |
| START | Host | 1 | 1 | 1 |
| TAR | Host | 2 | 2 | 2 |
| CYCTYPE + DIR | Peripheral | 1 | 1 | 1 |
| ADDR | Peripheral | 4 | 4 | 4 |
| SIZE | Peripheral | 1 | 1 | 1 |
| TAR | Peripheral | 2 | 2 | 2 |
| SYNC | Host | 6 | 6 | 6 |
| DATA | Host | 2 | 4 | 8 |
| TAR | Host | 2 | 2 | 2 |
| **Total Clocks** |  | **21** | **23** | **27** |

### 外设启动IO写操作

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **I/O Write Cycle** | **Driven By** | **8 Bit** | **16 Bit** | **32 Bit** |
| START | Host | 1 | 1 | 1 |
| TAR | Host | 2 | 2 | 2 |
| CYCTYPE + DIR | Peripheral | 1 | 1 | 1 |
| ADDR | Peripheral | 4 | 4 | 4 |
| SIZE | Peripheral | 1 | 1 | 1 |
| DATA | Peripheral | 2 | 4 | 8 |
| TAR | Peripheral | 2 | 2 | 2 |
| SYNC | Host | 6 | 6 | 6 |
| TAR | Host | 2 | 2 | 2 |
| **Total Clocks** |  | **21** | **23** | **27** |

### Request Assertion Rules(请求断言规则)

声明总线主控器请求时，外设使用LDRQ＃线并发送编码为“ 100b”。 发送此消息后，除非外设获得总线主控器请求，否则LDRQ＃禁止声明另一个LDRQ＃消息“ 100b”。 它可能仍会发送有关DMA通道的消息。

授予总线主控通道后，它可以发送另一个LDRQ＃消息。 在驱动“ CYCTYPE + DIR”字段后，它仍然可以在LCLK上发送另一条消息。

## 电源管理（PM）

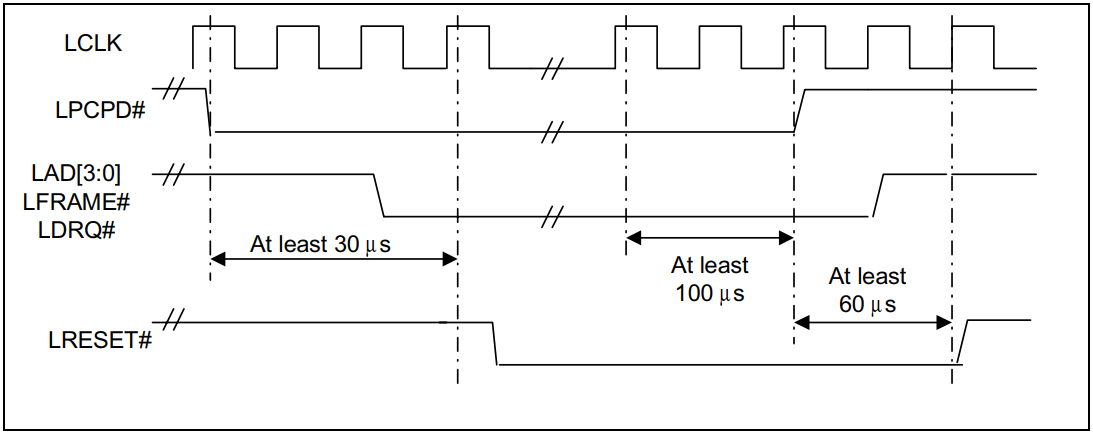
### CLKRUN# Protocol

在某些低功耗状态下，LCLK可能会停止。 要重新启动它，外设应置位PCI CLKRUN＃信号。 这将需要驱动LDRQ＃信号激活。

外设启动LDRQ＃序列后，主机不得取消置位CLKRUN＃导致时钟停止.

### LPCPD# Protocol

进入和退出掉电状态的一般时序如图所示。注意，这些时序仅适用于进入和退出低功耗状态。



Timing for Entering and Exiting the Power Down State

在进入低功耗状态之前，系统将声明LPCPD＃信号。在LCLK＃信号停止为低电平并将其他主机LPC I / F输出信号变为高阻态或驱动为低电平之前，至少要声明30微秒。

识别出已声明的LPCPD＃后，LPC接口上将不再有其他事务。外设应将LDRQ＃信号驱动为低电平或高阻态，直到LPCPD＃无效为止。这是为了防止外设将LPC接口信号驱动到可能掉电的主机中。

识别到LPCPD＃无效后，外设应将其LDRQ＃信号驱动为高电平。

LPCPD＃无效后，可以根据连接到LRESET＃的系统复位信号的特性来复位LPC接口。请注意，如果使用外设中的某些逻辑将系统从低功耗状态唤醒，则不应使用LRESET＃复位该逻辑。相反，如果唤醒逻辑需要某种类型的复位，则应通过软件机制将其复位。

外设必须异步识别LPCPD＃处于活动状态（它可能不符合LCLK的建立时间）。但是，它可以使用LCLK采样，因为它将运行至少30微秒（在LPCPD＃变低之后）。外设必须异步识别LPCPD＃变为非活动状态。但是，它可以使用LCLK采样，因为它将在LPCPD＃变高之前运行30毫秒以上。从LPCPD＃被识别为有效到LPCPD＃被识别为无效，外设应忽略LCLK。

### LPME# Usage

在典型的基于PCI的系统中，不应将LPME＃连接到PCI总线PME＃信号。 这是因为LPC设备通常不实现PCI设备所需的PCI电源管理配置空间。

## 总线复位机制

LPC接口的行为和LRESET＃的时序在一般系统复位方面不同于进入或退出低功耗状态。

主机和外围设备必须遵守以下规则：

1.当LRESET＃无效时，假定时钟正在运行。确切的时钟数将与PCI规范的时钟数相同。

2.当LRESET＃被断言时，主机的行为如下：

•LFRAME＃将被拉高

•LAD [3：0]将为高阻态

•LDRQ [n]＃被忽略

3.声明LRESET＃时，外设的行为如下：

•LFRAME＃可以忽略

•LAD [3：0]将为高阻态

•LDRQ [n]＃将被驱动为高

通用系统复位的LRESET＃断言可以随时发生，并且可以与LCLK异步。进入系统复位条件时，无法保证LPCPD＃与LRESET＃的时序关系。

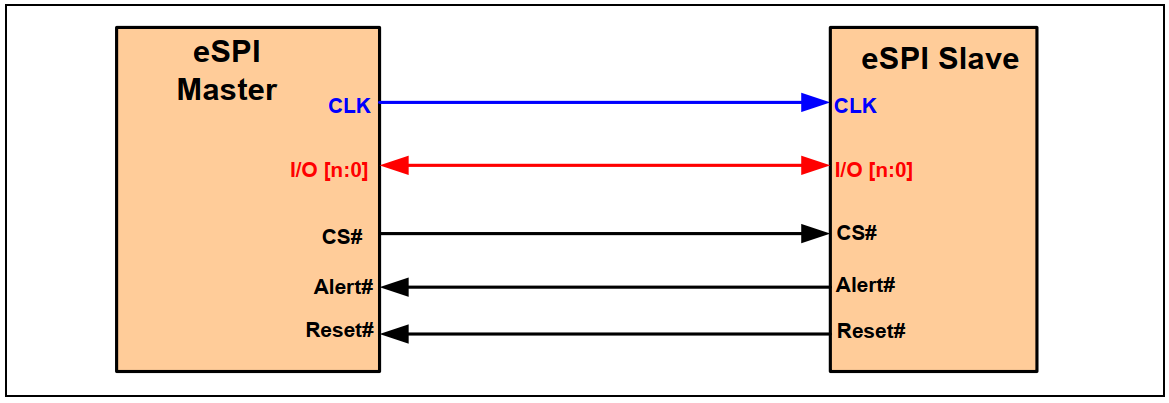
# eSPI 总线

## 物理连接

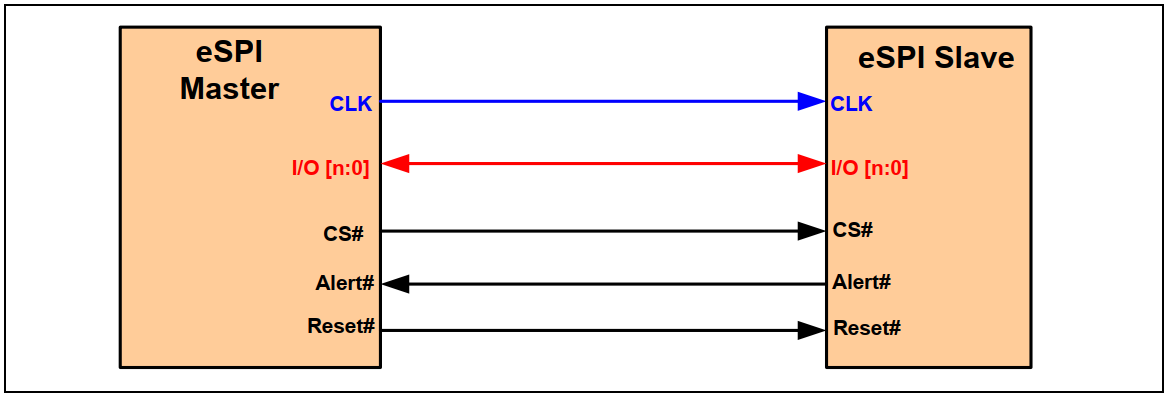
对于eSPI总线，只有一个eSPI主机和一个或多个eSPI从机。

|  |  |  |  |
| --- | --- | --- | --- |
| **Pin Name** | **Direction** | **Clock** | **Description** |
| **eSPI Reset#** | Master to Slave1 or Slave to Master2 | Asynchronous | **Reset#:** Reset the eSPI interface for both master and slaves.  **Note:**  1. eSPI Reset# is typically driven from eSPI master to eSPI slaves.  2. eSPI Reset# is generated by eSPI slave, driven from eSPI slave to eSPI master. |
| **Chip Select#** | Master to Slave | Asynchronous | Chip Select#: Driving Chip Select# low selects a particular eSPI slave for the transaction.  Each of the eSPI slaves is connected to a dedicated Chip Select# pin. |
| **Serial Clock** | Master to Slave | - | Clock: This pin provides the reference timing for all the serial input and output operations. |
| **I/O [n:0]** | Bi-directional | Serial Clock | I/O: These are bi-directional input/output pins used to transfer data between master and slaves.  The value of ‘n’ may be 1 or 3 depending on the I/O mode.  In Single I/O mode (n=1), I/O[0] is the eSPI master output/eSPI slave input (MOSI) whereas I/O[1] is the eSPI master input/eSPI slave output (MISO). |
| **Alert#** | Slave to Master | Asynchronous | Alert#: This pin is used by eSPI slave to request service from eSPI master. Alert# is either a driven, or an open drain output from the slave with default as a driven output.  This pin is optional for Single Master Single Slave configuration where I/O[1] can be used to signal the Alert event. |

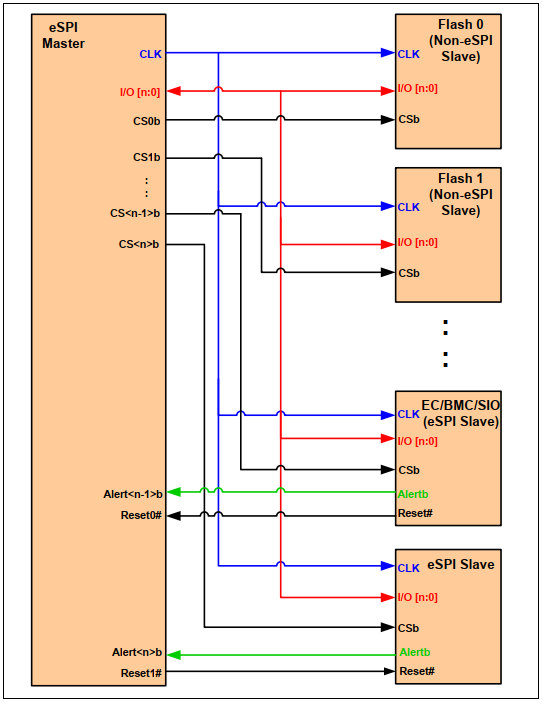
### 连接拓扑图



Single Master-Single Slave with eSPI Reset# from Slave to Master



Single Master-Single Slave with eSPI Reset# from Master to Slave



Single Master-Multiple Slaves with Two eSPI Reset#

时钟和数据引脚可以由多个SPi或者eSPi从机共享，每个从机都有专用的CS＃和Alert＃引脚。

允许诸如Flash和TPM之类的SPI从机与eSPI从机共享同一组时钟和数据引脚。这些非eSPI从机是使用专用的芯片选择＃引脚选择的，它们通过在eSPI总线上运行的SPI特定协议与eSPI主机进行通信。

## 电气属性

可以选择将CS#，Alert#以及Reset#信号接弱上拉。因为这几个信号是低电平有效。

## 总线协议

实际上eSPI总线和SPI协议很相似，但是还是具体的内容却大不相同，他们仅仅是相似的时序而已。

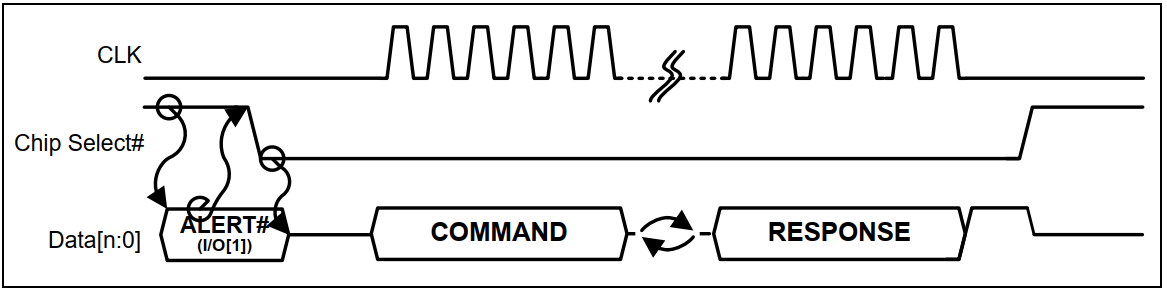
在eSPI传输过程中必须是8位对齐，或者1字节的倍数传输。



Basic eSPI Protocol

eSPI事务包括由主机驱动的命令阶段，Turn-Around（TAR）阶段和由从机驱动的响应阶段。命令阶段包括CMD，可选handler（HDR），可选DATA和CRC。响应阶段包括一个RSP，一个可选的handler（HDR），可选的数据，状态和CRC。对于始终在总线上传输CRC字节的所有eSPI事务，都必须生成CRC。但是，默认情况下，复位后CRC检查默认为禁用，并且由SET CONFIGURATION启用。禁用CRC检查时，接收器将忽略CRC字节。

主机可以通过设置cs＃来启动事务，启动时钟并将命令驱动到数据总线上。时钟保持切换状态，直到从机接收到完整的响应阶段为止。



Slave Triggered Transaction (Single Master-Slave)

从机可以通过向主机发出警报事件来启动事务。警报事件可以通过两种方式发出信号。在单主单从配置中，从机可以使用I / O [1]引脚指示警报事件。在单主多从配置中，需要专用的Alert＃引脚。

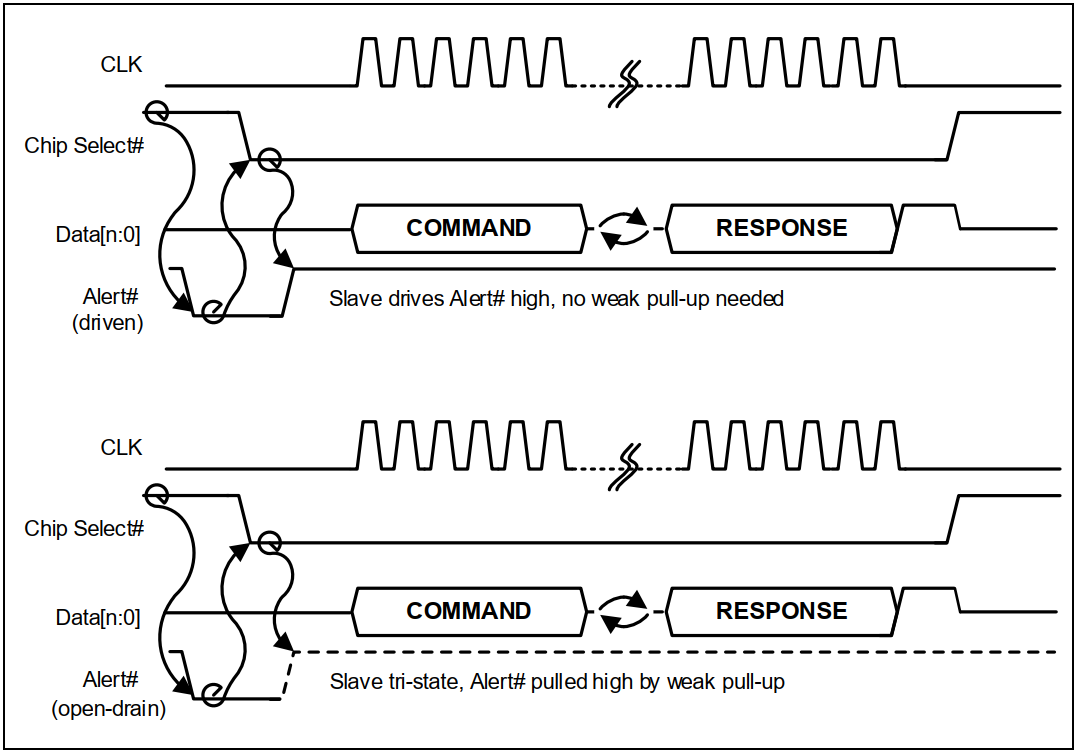
要注意的是当从机的CS＃为高电平时，从机只能发出警报事件。

### Single Master-Single Slave

当I / O [1]用于发出Alert＃事件的信号时，当从机决定请求服务时，它会从高阻态切换到低电平。然后，从机保持I / O [1]引脚的状态，直到主机确定片选信号为止。一旦设置CS＃，eSPI从器件就必须通过在tSLAZ时序内使该引脚处于高阻态来释放I / O [1]引脚的所有权，并且该引脚将被弱上拉电路拉高。然后，主机继续发出命令，从机中找出警报事件的原因，然后为请求提供服务。发送CRC后，在串行时钟的最后一个下降沿，eSPI从机必须将I / O [n：0]引脚驱动为高电平，直到取消选择CS＃。将其驱动为高电平可确保在取消使CS＃无效时I / O [1]不会产生错误的Alert＃事件。在CS＃的边沿时，这些I / O [n：0]引脚被从机设为高阻态，满足tSHQZ输出禁用时序，其中弱上拉将这些引脚保持在高电平，而主机继续为高阻态。为了在CS＃置为无效后发出警报事件，仅允许从机在tSHAA时序后重新置位I / O [1]引脚

当Alert＃引脚用于发出Alert＃事件信号时，从机将其从高电平切换为低电平或从高阻态切换为低电平。当从机决定请求服务时，I / O [n：0]引脚保持从机状态为高阻态。然后，从机将保持Alert＃引脚的状态，直到主机选择了CS＃。设置CS＃时，从机必须将Alert＃引脚驱动为高电平，或通过使该引脚处于高阻态来释放该引脚的所有权。然后，主机继续发出命令，从机中找出警报事件的原因，然后为请求提供服务。在发送CRC之后，串行时钟的最后一个下降沿，eSPI从机必须将I / O [n：0]引脚驱动为高电平，直到CS＃引脚变为高电平为止。CS＃无效后，这些I / O [n：0]引脚在满足tSHQZ输出禁用时序（弱上拉使这些引脚保持为高电平，而主机继续进入高阻态）之后被从机置于高阻态。 tSHAA时序不适用于Alert＃引脚。但是，当Alert＃引脚配置为漏极开路并置为有效时，该引脚上的弱上拉必须使CS＃的有效时间最短（使从机将Alert＃置于高阻态）。

### Single Master-Multiple Slaves



Slave Triggered Transaction (Multiple Slave)

该规范不会阻止将Alert＃引脚用于单主/单从配置。

在Alert事件设置与CS＃边缘情况下，从机仍将I / O [1]引脚设为高阻态或将Alert＃引脚驱动为高电平或高阻态。状态将在响应阶段返回，然后主机知道需要服务于从机的未完成请求。要注意的是，引脚上发出的警报事件与串行时钟异步。

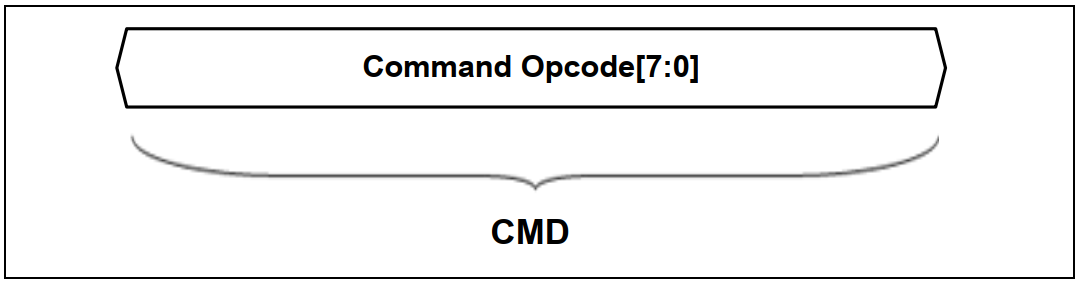
eSPI从机必须支持两种类型的警报机制。确定对每个eSPI从机使用哪种Alert机制是实现特定的方法。

eSPI被定义为使用基于数据包的拆分事务协议。在发送端，基于要发送的事务在传输层中形成数据包。链路层使用CRC字节扩展数据包。

同样，在接收端，启用CRC校验后，会在接收链路层检查CRC。一旦数据包通过了CRC校验，该数据包便被发送到“事务层”，在此进行解码并采取行动。

### 指令域（CMD）

eSPI主机使用命令阶段来启动到从机的事务，或响应从机的警报事件。 它由CMD，可选的标头（HDR），可选的DATA和CRC组成，CMD字段由命令操作码组成。



Command Opcode

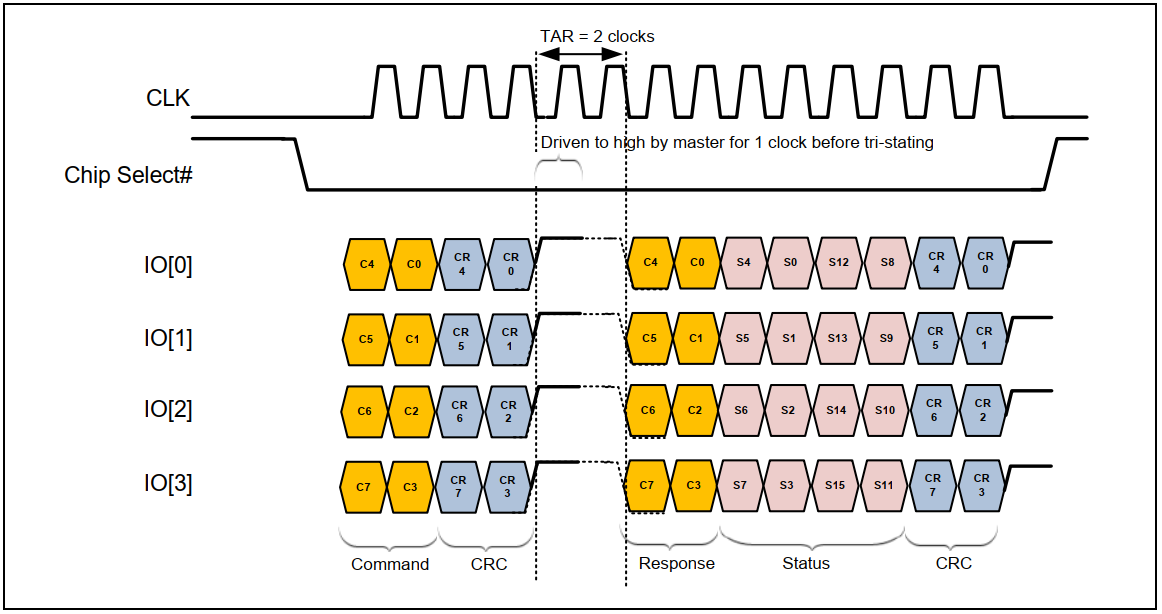
命令操作码为8位宽，Command Opcode用于指示特定于通道的命令并传达链接管理事件，如果从机收到带有无效指令操作码的数据包（本规范未定义），则从机不得响应该事务。 事务将在总线上以默认响应（NO\_RESPONSE）终止。

|  |  |  |  |
| --- | --- | --- | --- |
| 通道范围 | 操作指令 | 编码[7:0] | 描述 |
| **eSPI Peripheral Channel**  编码中的xx如下表示：  XX:  00: 1BYTE  01:2 BYTE  10:RESERVED  11:4BYTE | PUT\_PC | 00000000 | 放置一个已发布posted或hander和data |
| PUT\_NP | 00000010 | 放置未发布的标头和可选数据。 |
| GET\_PC | 00000001 | 获取一个已发布posted或hander和data |
| GET\_NP | 00000011 | 获取未发布的标头和可选数据。 |
| PUT\_IORD\_SHORT | 010000XX | 放置一个 non-posted IO读包指令 |
| PUT\_IOWR\_SHORT | 010001XX | 放置一个 non-posted IO写包指令 |
| PUT\_MEMRD32\_SHORT | 010010XX | 放置一个 non-posted 内存读32个数据包指令 |
| PUT\_MEMWR32\_SHORT | 010011XX | 放置一个 non-posted 内存写32个数据包指令 |
| Virtual  Wire  Channel | PUT\_VWIRE | 00000100 | 放置一个虚拟通道包指令 |
| GET\_VWIRE | 00000101 | 获取一个虚拟通道包指令 |
| OOB  Message  Channel | PUT\_OOB | 00000110 | 放置一个OOB（SMBus 通道）信息指令 |
| GET\_OOB | 00000111 | 获取一个OOB（SMBus 通道）信息指令 |
| Flash 访问通道 | PUT\_FLASH\_C | 00001000 | 设置Flash 访问完成指令。  在主机附加闪存共享模式下使用，主机可以将闪存访问完成返回给从机。 |
| GET\_FLASH\_NP | 00001001 | 获取non-posted Flash 访问请求指令 |
| 独立通道 | GET\_STATUS | 00100101 | 主机发起的命令，用于读取从机的状态寄存器。 |
| SET\_CONFIGURATION | 00100010 | 在初始化过程中设置从机功能的命令。通常在主机发现从机的功能之后执行此操作。 |
| GET\_CONFIGURATION | 00100001 | 在初始化过程中发现从机功能的命令 |
| RESET | 11111111 | 复位指令 |

### 周转域 (TAR)

在数据线上发送命令阶段的最后一位之后，数据线进入“Turn-Around”窗口。 要求eSPI主机在Turn-Around窗口的第一个时钟将所有数据线置1，然后将数据线置于高阻态。周转窗口的时钟数是固定的2个串行时钟，与eSPI I / O模式无关。如果从机需要额外的时间来获取命令并准备响应，则从机可以在TAR窗口后为任何eSPI事务插入WAIT\_STATE响应指令。

在所有eSPI I / O模式（单，双，四路I / O）的响应阶段之前，eSPI从机均不得驱动I / O [n：0]。特别是在单I / O模式下，从机必须在响应阶段之前才驱动I / O [1]（MISO）。 在周转周期之后，它必须立即在总线上驱动响应阶段，如下图所示



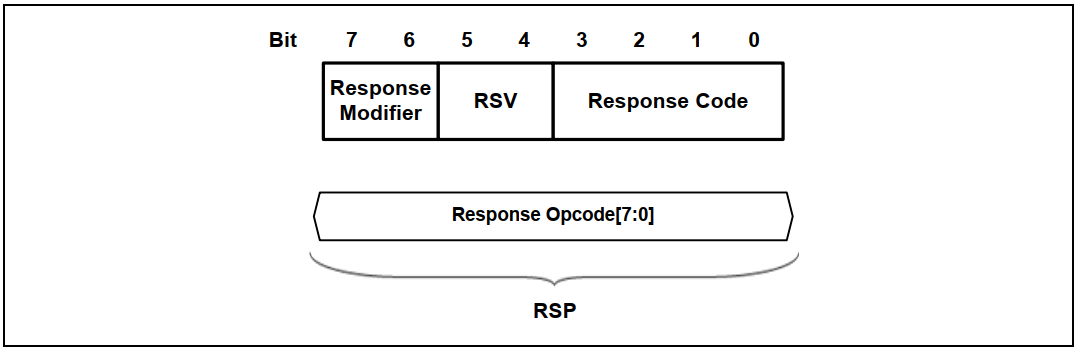
Turn-Around Time (TAR = 2 clock)

### 响应域(RESP)

响应阶段由eSPI从机驱动，以响应eSPI主机启动的命令。 它由RSP操作码，可选的hander（HDR），可选的数据，STATUS和CRC组成

#### 响应操作码

其中RSP操作码是一个8位字段，由响应代码和响应修饰符组成



Response Field

响应代码表示请求是成功，被推迟还是以错误或等待状态响应。

响应修饰符是为GET\_STATUS定义的2位字段，仅具有ACCEPT响应。对于所有其他响应，它的值必须始终为“ 00”，但NO\_RESPONSE的值为“11”。

响应修饰符字段指示是否将外围设备（通道0），虚拟线路（通道1）数据包或闪存访问（通道3）的GET\_STATUS指令的响应是否完成。仅当支持从属连接的闪存共享并在运行中时，闪存访问（通道3）完成才适用。

默认情况下，“响应修饰符”是禁用的。通过将“通用功能和配置”寄存器中的“响应修饰符启用”位设置为“ 1”来设置SET\_CONFIGURATION。

当从机驱动响应阶段时，必须将RSP操作码的保留（RSV）字段驱动为全0。它保留供规范将来使用。为了向后兼容，主机必须忽略保留（RSV）字段。

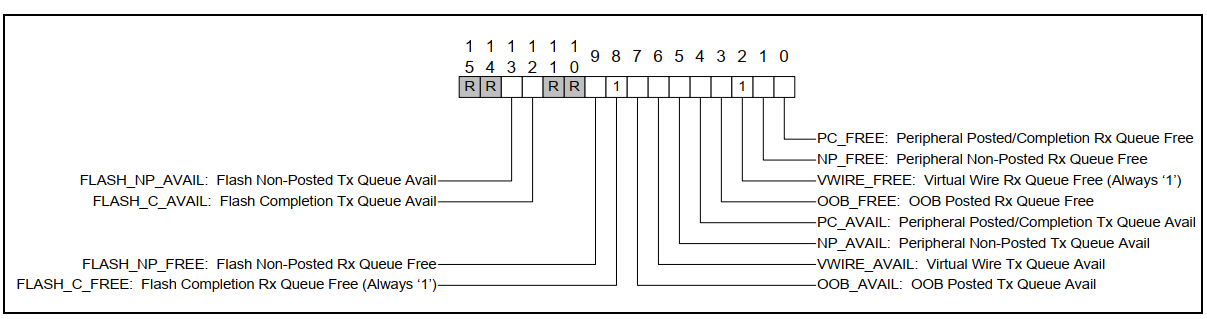
当响应阶段不受任何从机驱动时，默认为NO\_RESPONSE。 eSPI主机可以在检测到任何时候通过取消置位片选＃来终止传输。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| RESPONSE | 编码 | | | 描述 |
|  | [7:6] | [5:4] | [3:0] |  |
| ACCEPT | XX | RSV | 1000 | 命令已成功接收  如果命令是PUT\_NP，则ACCEPT的响应表示non-posted事务将作为“连接”事务完成。  XX:  00 : no append  01 : append外围设备  10：append虚拟通道  11：append flash访问通道 |
| DEFER | 00 | RSV | 0001 | 仅在响应PUT\_NP时有效。 已成功接收到未发布的命令，并且将完成未发布的事务推迟到以后的拆分完成中。 |
| NON\_FATAL\_ERROR | 00 | RSV | 0010 | 收到一个严重错误但不致命，但是该错误不会影响处理收到的命令 |
| FATAL\_ERROR | 00 | RSV | 0011 | 收到一个致命错误，该错误导致无法成功处理传输层数据包。 致命错误包括错误格式传输，总线没释放，无法获取有效值等等。 |
| WAIT\_STATE | 00 | RSV | 1111 | 响应总线上的事务时，增加一字节的延迟时间 |
| NO\_RESPONSE | 11 | RSV | 1111 | 无响应。 当由于数据线上的弱上拉而导致没有从机出现时，它是对GET\_CONFIGURATION的默认响应。 当在命令包上检测到致命的CRC错误，或者不支持命令操作码并且从机不得驱动响应阶段时，它也是默认响应。 |

#### 状态域

状态码由16位组成，如果通道已启用但尚未就绪，或者不支持的功能的状态位，则这些位无关紧要，必须由eSPI主机忽略。保留状态位必须由从机设置为0

状态域反映了传输过程中的从机实时状态，在考虑此事务中正在接收或发送的命令之后，AVAIL和FREE反映了队列状态。



Slave’s Status Register Definition

|  |  |  |  |
| --- | --- | --- | --- |
| 域 | Status | Bit Position | 描述 |
| 从机接受队列空闲 | PC\_FREE | 0 | 当为“ 1”时，表示从机可以自由接受至少一个通道0外设posted或完成报头以及最大有效负载大小的数据。 |
| NP\_FREE | 1 | 如果为“ 1”，则表示从机可以自由接受至少一个通道0外围non-posted的报头和1 DW的数据（如果适用） |
| VWIRE\_FREE | 2 | 该位必须始终为“ 1”。 通道1虚拟线的隧道不受流量控制 |
| OOB\_FREE | 3 | 当为“ 1”时，表示从机可以自由接受至少一个CH2 OOB（SMBus）消息，其数据最大为最大有效负载大小。 |
| 从机发送队列有效 | PC\_AVAIL | 4 | When ‘1’, indicates the slave has a channel 0 peripheral posted or completion header and optional data up to maximum payload size available to send |
| NP\_AVAIL | 5 | When ‘1’, indicates the slave has a channel 0 peripheral non-posted header available to send. |
| VWIRE\_AVAIL | 6 | When ‘1’, indicates the slave has a channel 1 tunneled virtual wire available to send. |
| OOB\_AVAIL | 7 | When ‘1’, indicates the slave has a channel 2 OOB (tunneled SMBus) message with data up to maximum payload size available to send. |
| 从机接收队列空闲 | FLASH\_C\_FREE | 8 | 该位始终唯1 |
| FLASH\_NP\_FREE | 9 | 如果为1，则表示从机可以自由接受至少一个通道3 Flash 访问non-posted的标头和最大有效载荷大小的数据。 |
| Reserved | 11:10 | 始终为0 |
| 从机接收队列有效 | FLASH\_C\_AVAIL | 12 | When ‘1’, indicates the slave has a channel 3 Flash Access completion header and data up to maximum payload size available to send. |
| FLASH\_NP\_AVAIL | 13 | When ‘1’, indicates the slave is free to accept at least one channel 3 Flash Access nonposted header and data up to maximum payload size. |
| Reserved | 15:14 | 始终为0 |

### Alert 域

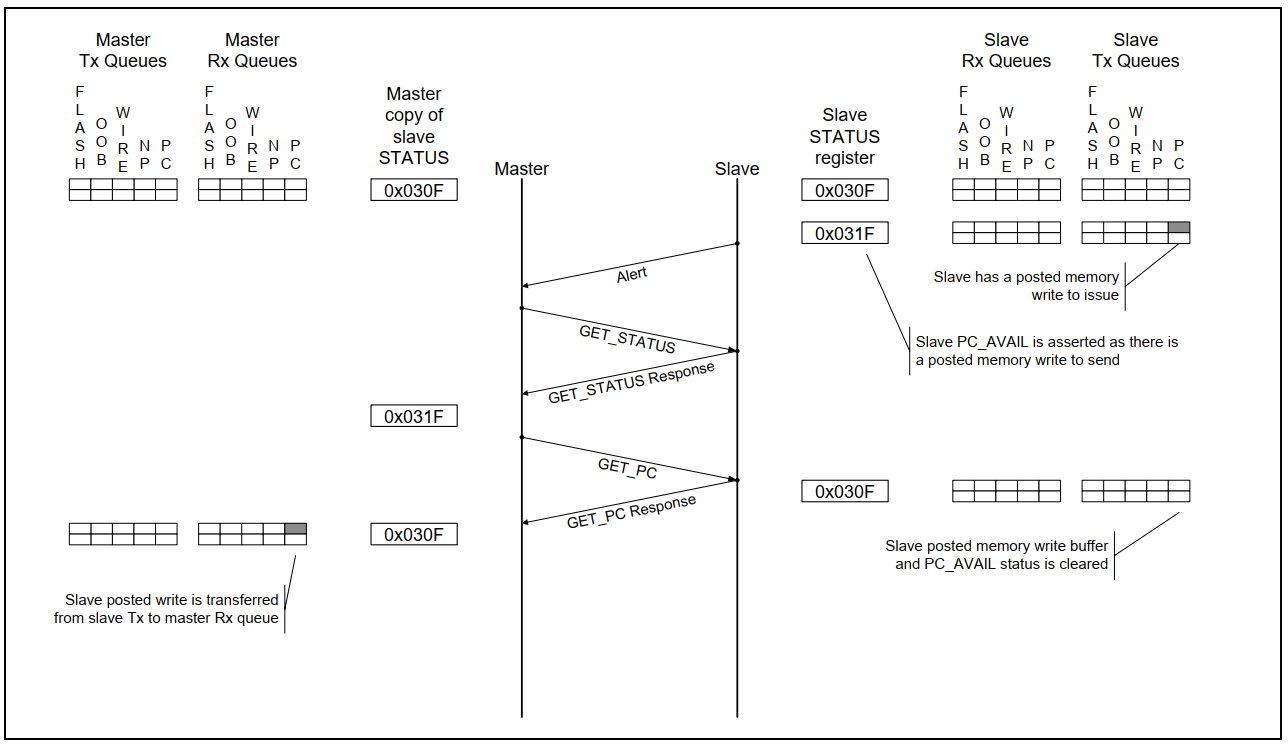
从机发出Alert信号以请求服务。主机可以向相应的从机发出GET\_STATUS命令，以获取Alert事件的原因。

从机可能由于以下任何原因而生成Alert事件：

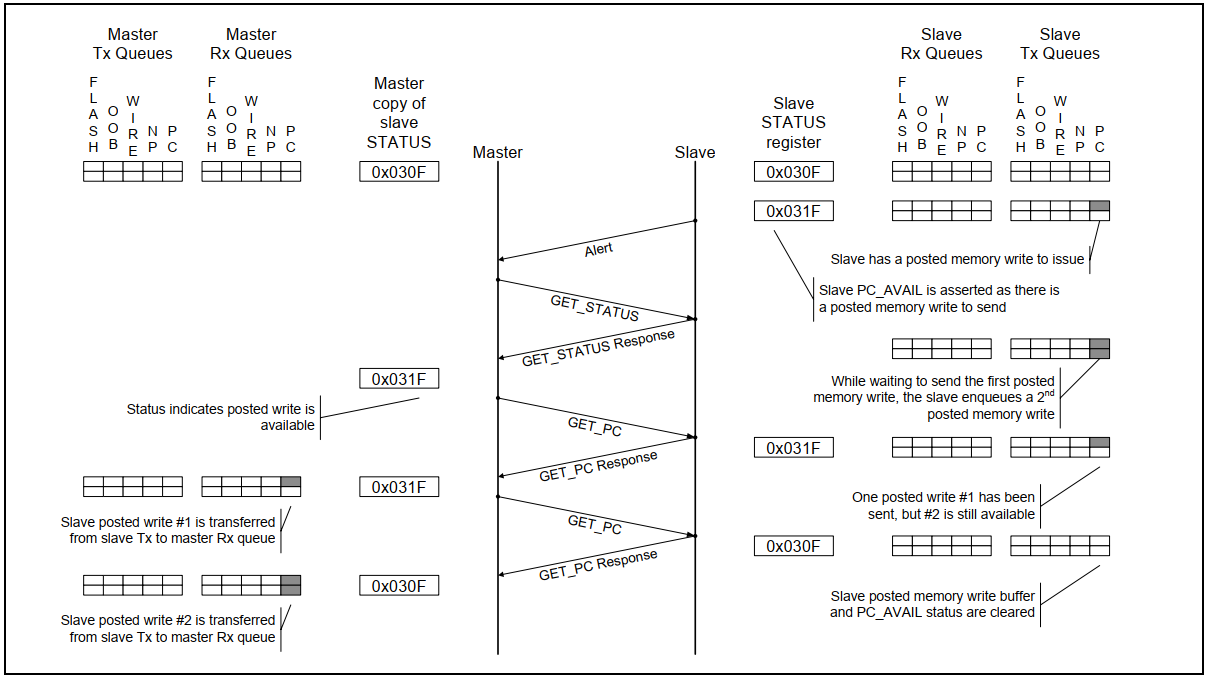
•从机有新请求。这可以是已发布，未发布，延迟完成，虚拟线路消息，OOB消息或Flash访问请求。

•从上次状态更新返回为非空闲以来，从机缓冲区空间已变为空闲。

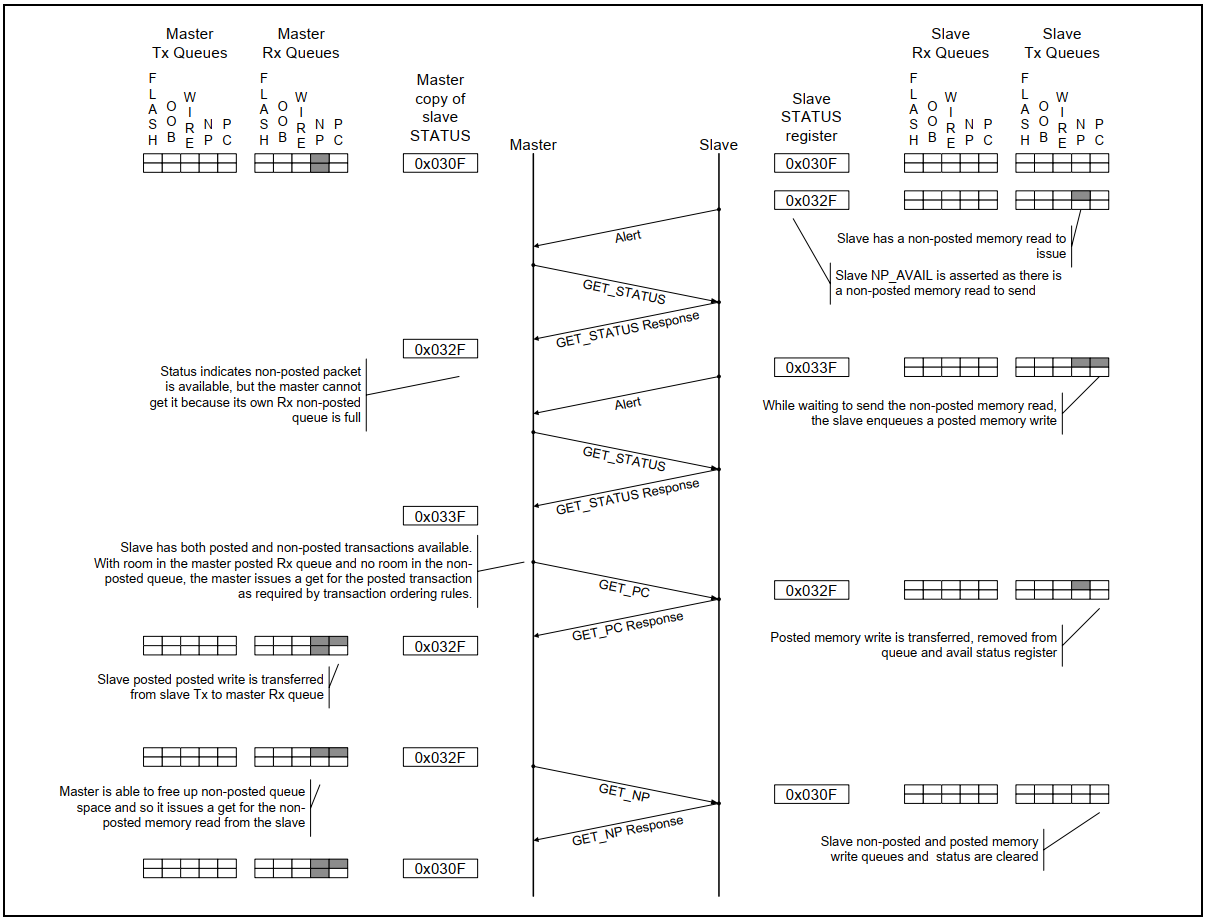
触发Alert事件的每个原因在STATUS寄存器中都有对应的位。当STATUS寄存器的状态不同于在上一个Response阶段返回的STATUS时，从机将生成一个新的Alert事件。 状态寄存器中的差异表示发生了新事件，需要主机提供服务。



从机掌握外围设备的流程图发表于写

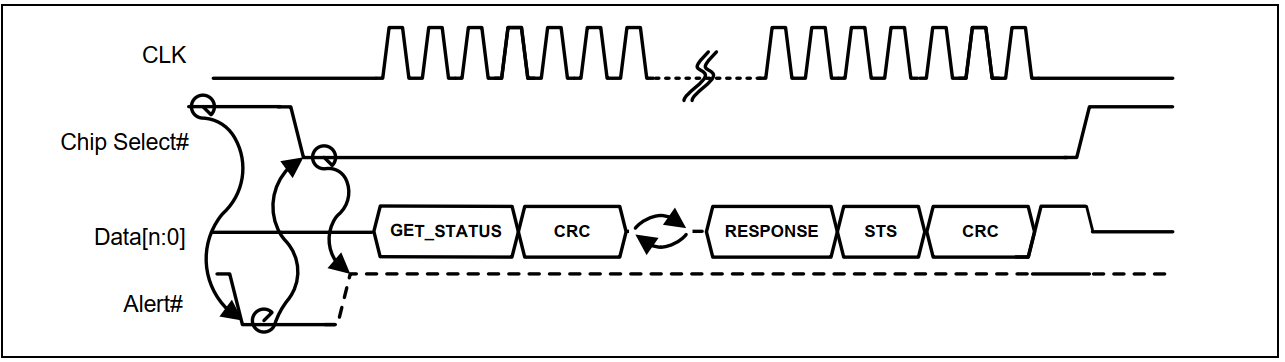


Flow Diagram for a Back-to-back Slave to Master Peripheral Posted Write



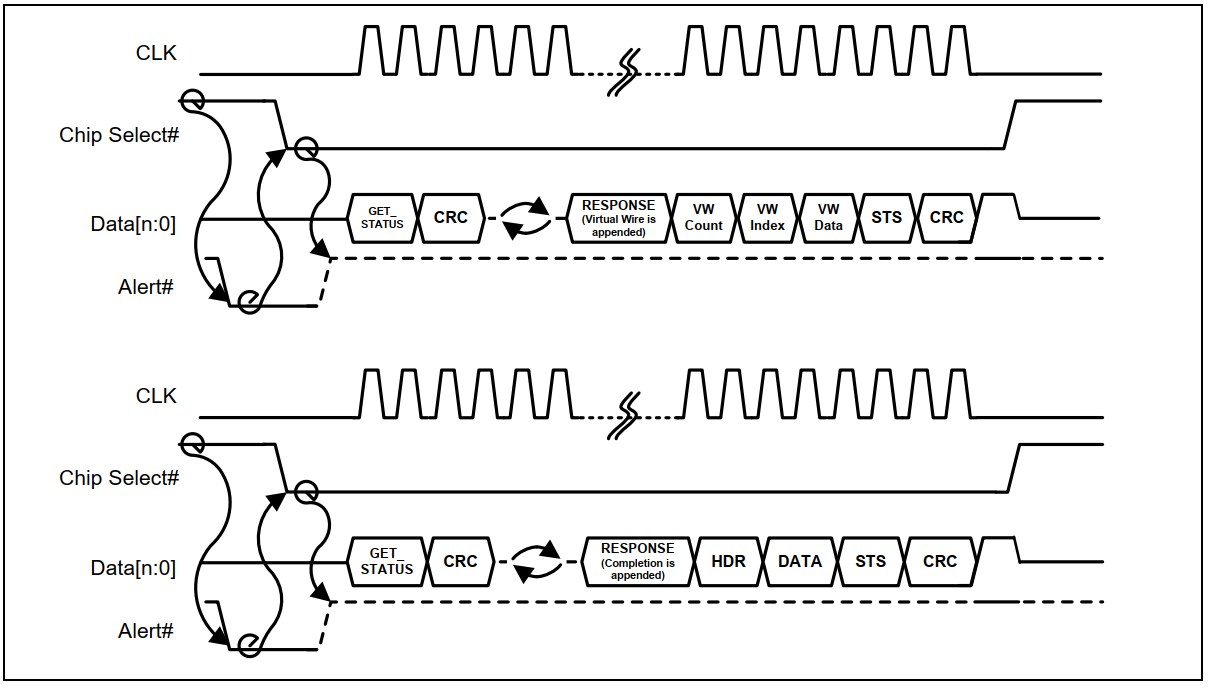
Flow Diagram for a Slave to Master Peripheral Posted Write passes Nonposted

### 状态获取指令域



GET\_STATUS Command

GET\_STATUS是与通道无关的命令，用于查询状态寄存器的内容。 状态寄存器的状态将在响应阶段返回。此命令通常用于响应来自eSPI从机的Alert事件，以确定Alert事件的原因并随后为Slave服务。



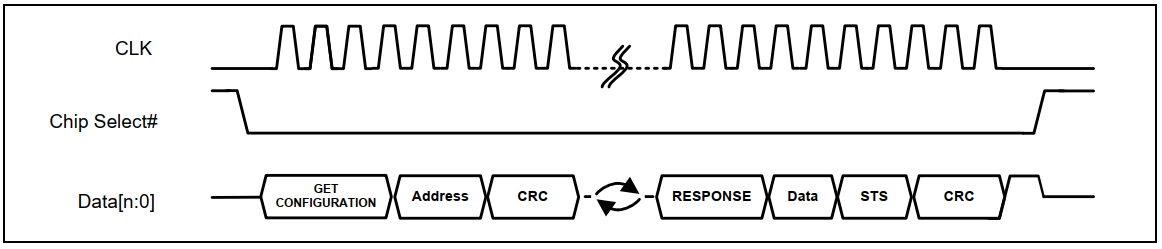
GET\_STATUS Command (with Response Modifier)

### 获取和设置指令域

SET\_CONFIGURATION和GET\_CONFIGURATION命令是与通道无关的命令，用于访问eSPI从机侧的通道功能和配置寄存器。 仅支持DWord访问。 由于没有启用字节，如果修改少于一个完整的DWord，则需要软件执行读-修改-写访问。

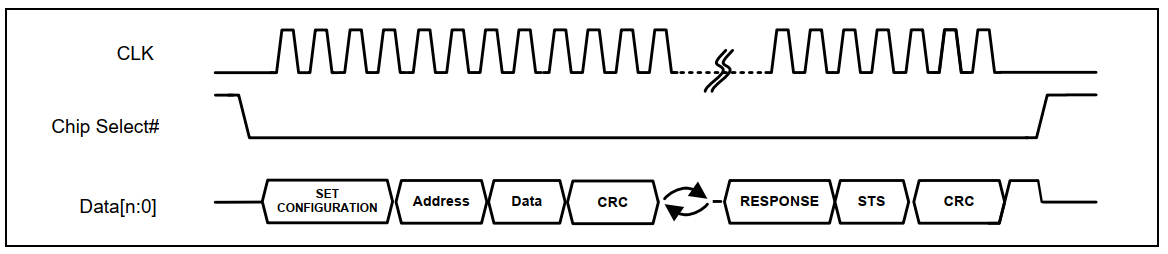
注:DWORD全称Double Word，是指注册表的键值，每个word为2个字节的长度，DWORD 双字即为4个字节，每个字节是8位，共32位

SET\_CONFIGURATION和GET\_CONFIGURATION命令永远不会被推迟，必须在同一周期内完成。



GET\_CONFIGURATION Command

GET\_CONFIGURATION命令用于读取eSPI从机上的通道功能和配置寄存器。 GET\_CONFIGURATION命令阶段包括一个8位命令操作码，一个16位地址和一个8位CRC。 响应阶段包括8位响应，1 DW数据，16位状态和8位CRC。

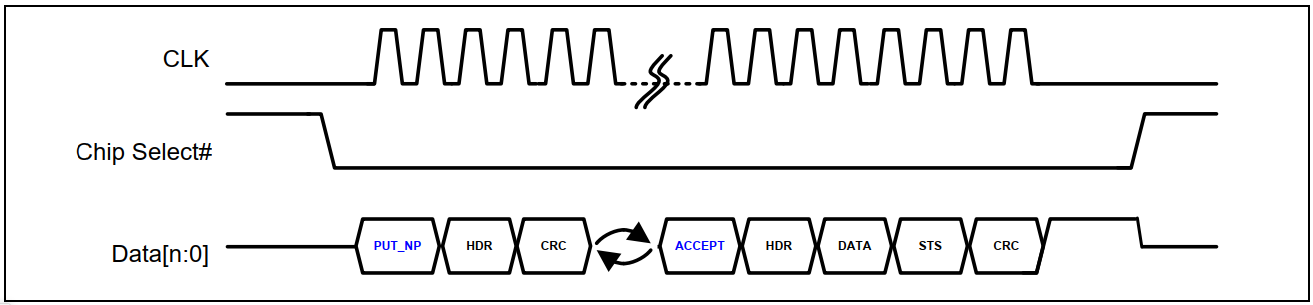


SET\_CONFIGURATION Command

SET\_CONFIGURATION命令用于在eSPI从机上写入通道功能和配置寄存器。 SET\_CONFIGURATION命令阶段包括一个8位命令操作码，一个16位地址，1个DW数据和一个8位CRC。 响应阶段包括一个8位响应，一个16位状态和一个8位CRC。

Note：从eSPI Reset＃中退出后，eSPI主机可以向特定的eSPI从机启动GET\_CONFIGURATION周期，以确定是否存在eSPI从机。 如果不存在eSPI从机，则在Turn-Around时间后eSPI数据线仍保持上拉状态。 eSPI机可以使用此行为来推断总线上不存在eSPI从机,如果存在eSPI从机，则eSPI从机必须在Turn-Around时间结束后响应。

### Non-Posted 传输

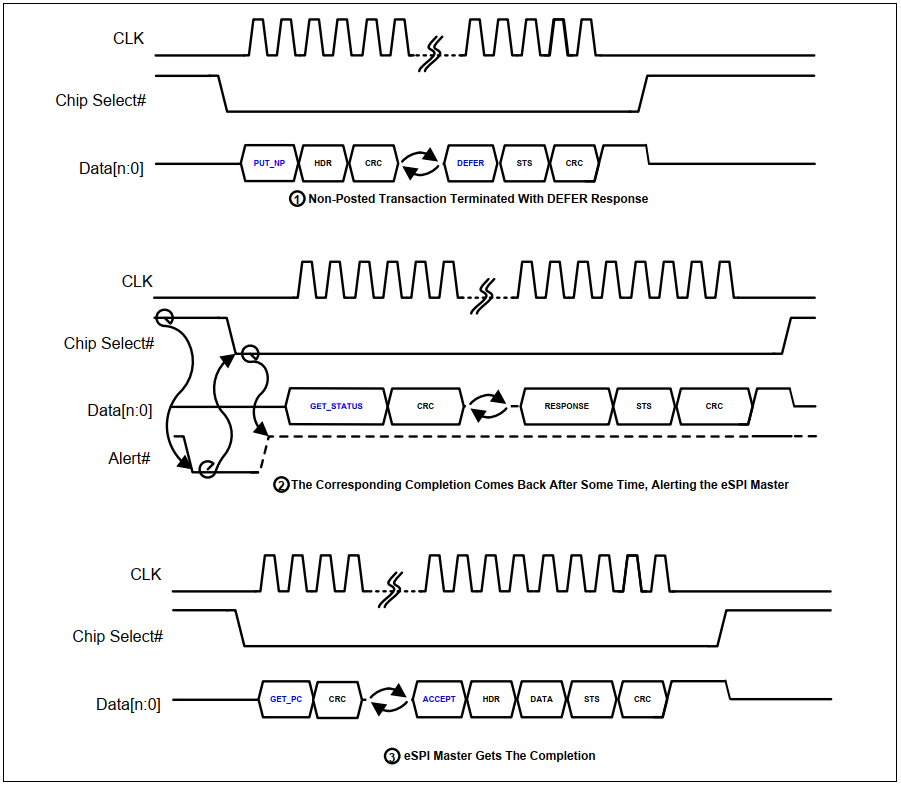


Connected Master Initiated Non-Posted Transaction

如果eSPI主机发起的non-posted传输无法立即获得的数据或其他信息，则non-posted请求将以“ DEFER”响应终止。当数据或信息最终可用时，可以在以后的一段时间内返回延迟的完成。只要保留指令规则，该总线就可以在延迟完成返回之前用于其他事务。

当返回延迟完成，唯一有效的响应是ACCEPT。 对于将以错误终止的non-posted，要求从机在连接时以FATAL ERROR或NONFATAL ERROR响应，而不会延迟事务。

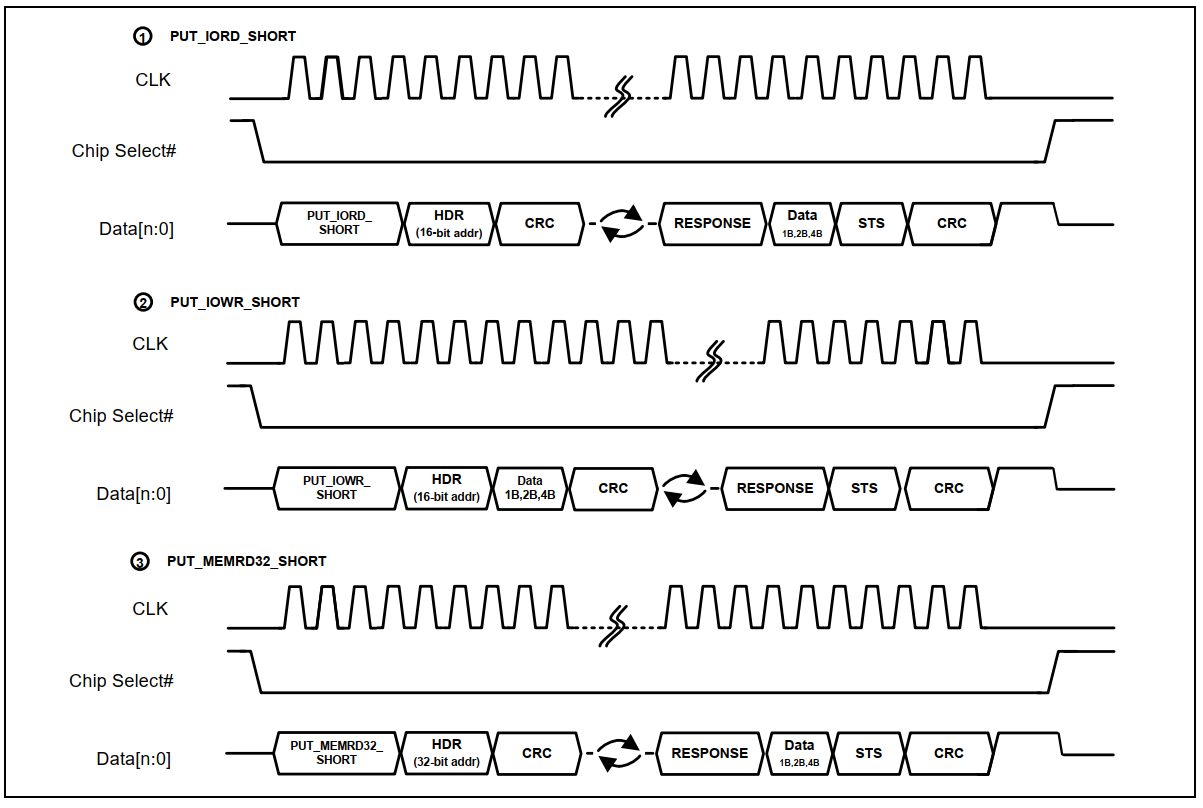
eSPI从机可以通过连接或延迟的多个拆分完成来完成non-posted的命令。 有关分割完成的情况，请参阅第5.1.3节。 如果拆分完成中的一个完成状态不成功，则将不返回其余拆分完成



Deferred Master Initiated Non-Posted Transaction

对于长度为1、2或4个字节的请求，eSPI支持从主机到从机的短时间non-posted的事务，这些事务开销较小，因此效率更高。 唯一的操作码指示non-posted传输的类型和请求长度。 hander仅包含地址，并且操作码隐含事务的地址字节数。 简短的non-posted传输记录没有“标记”字段。 Tag字段隐含为全0，从机将在完成hander中将其返回。

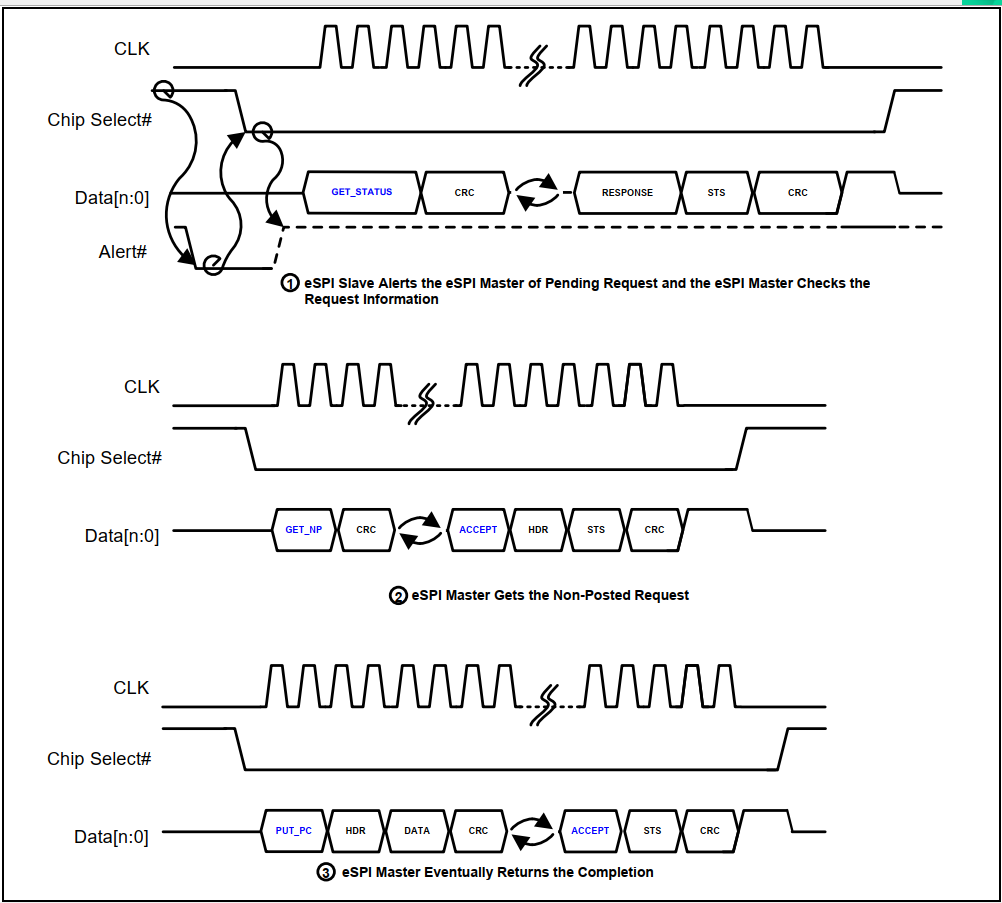
短期non-posted传输可以作为已关联或延迟完成而终止。但是，为了优化短事务的性能，从机应尽可能以在已连接状态完成事务。



Master Initiated Short Non-Posted Transaction

当存在未处理的non-posted传输时，eSPI从机可以生成警报。对此，eSPI主机将发出GET\_STATUS命令以检查待定请求信息

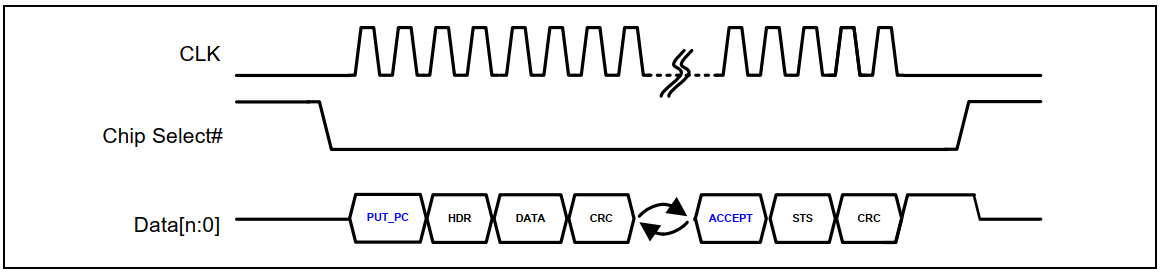
然后，eSPI主机将生成GET\_NP命令以获取non-posted传输。 一旦完成数据和返回响应所需的信息有效，eSPI主机就会将拆分完成内容返回给eSPI从机。由eSPI从机发起的non-posted请求的完成总是被拆分。



Slave Initiated Non-Posted Transaction

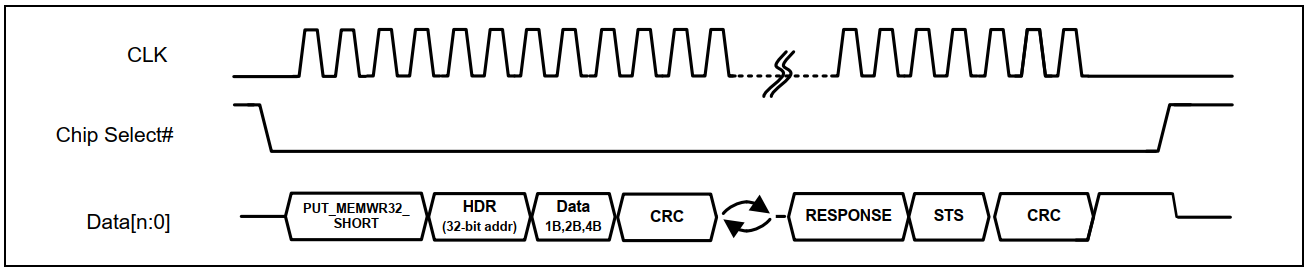
### Posted 传输

由eSPI主机启动posted传输的有效响应为ACCEPT，致命错误和非致命错误。posted传输的DEFER响应无效。



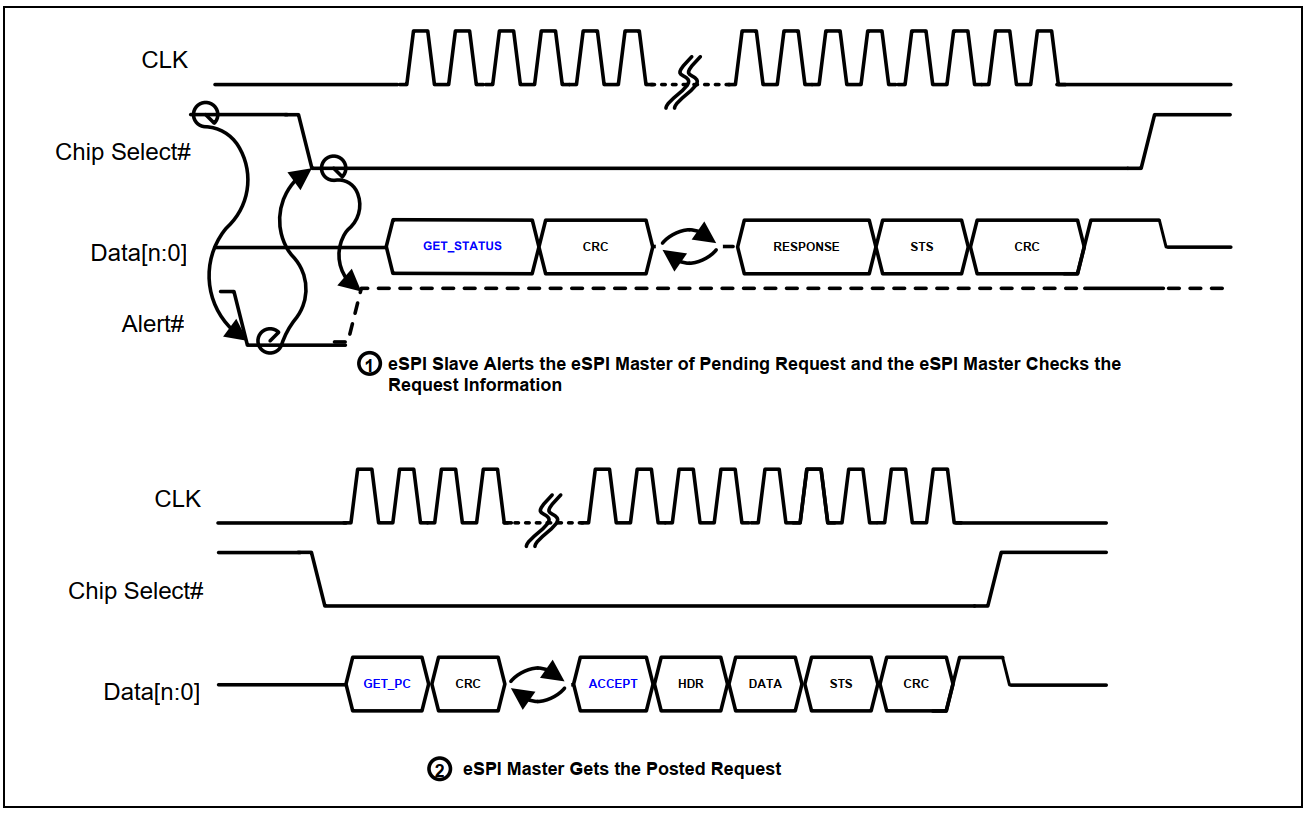
Master Initiated Posted Transaction

对于长度为1、2或4字节的请求，eSPI支持从主机到从机的短时间posted事务，这些事务开销较小，因此效率更高。 唯一的操作码指示短posted 传输和请求长度。 hander仅包含地址，并且操作码隐含事务的地址字节数。

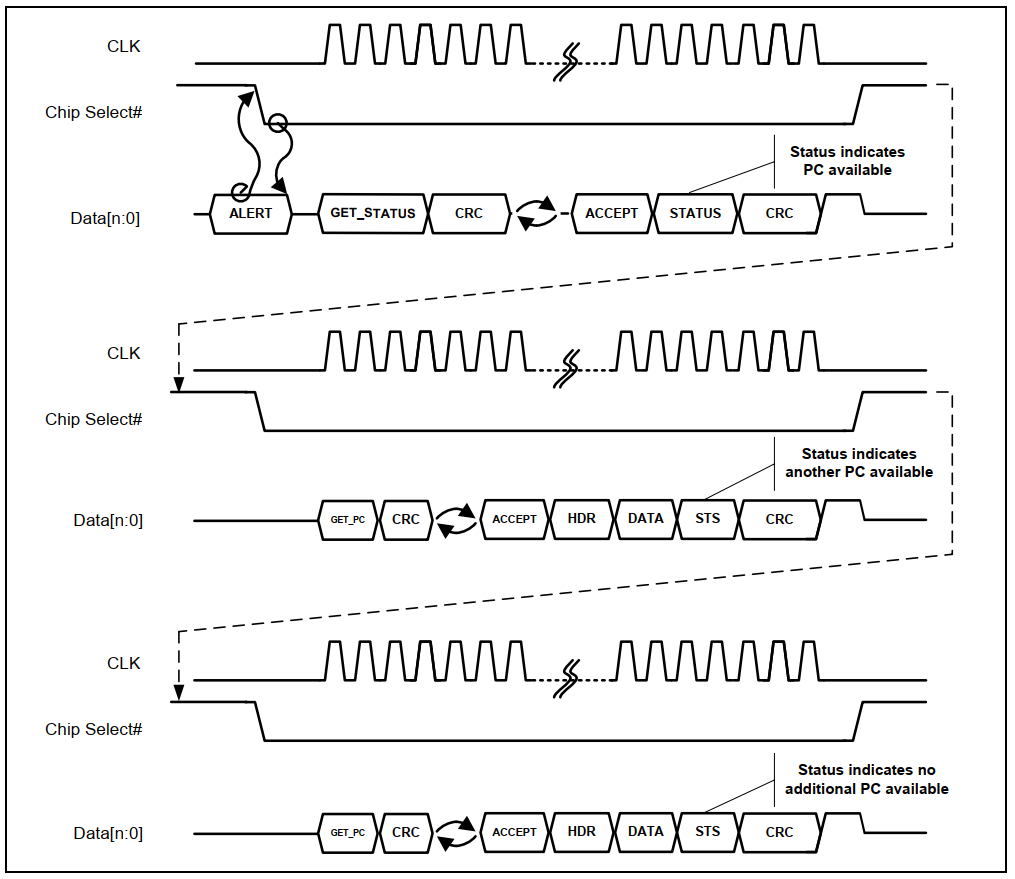


Master Initiated Short Posted Transaction

当有待定的posted传输时，eSPI从机可以生成警报。对此，eSPI主机将发出GET\_STATUS命令以检查pending请求信息.然后，eSPI主机将生成GET\_PC命令以获取posted传输。



Slave Initiated Posted Transaction



Pipelined Back-to-Back Bus Mastering Posted Write Transactions

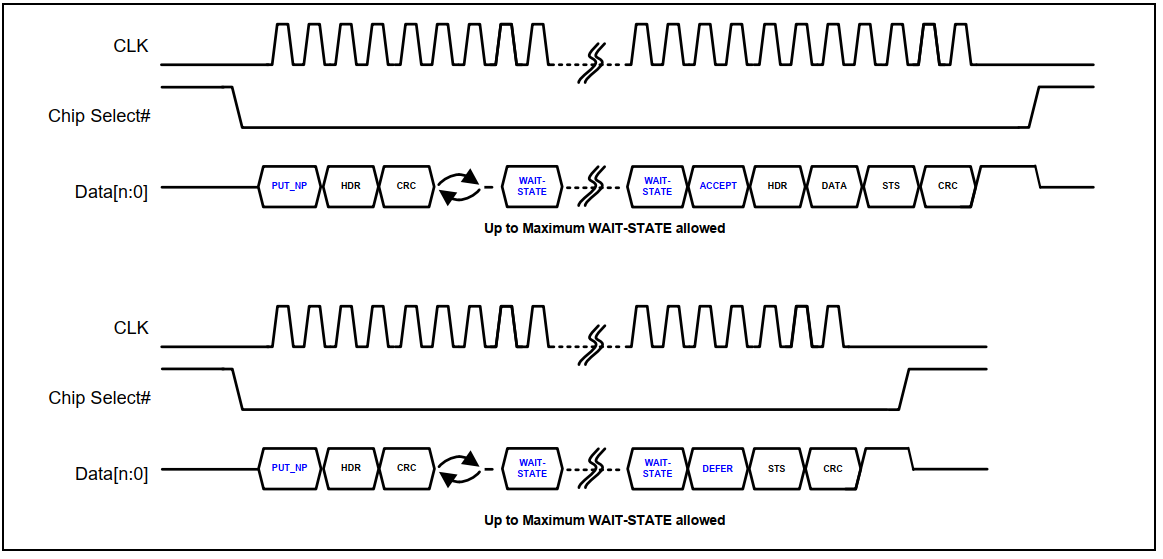
### WAIT STATE

在响应阶段，所有eSPI事务均由eSPI从机支持WAIT\_STATE。 在2个时钟的Turn-Around（TAR）窗口之后，允许eSPI从机以WAIT\_STATE响应代码进行响应，然后再以ACCEPT，DEFER，NON-FATAL ERROR或FATAL ERROR终止事务。

eSPI从机可以在响应阶段开始时插入一个或多个WAIT\_STATE响应代码，直到eSPI主机配置的最大WAIT\_STATE值为止。

如果从机需要采样命令并准备响应，则WAIT\_STATE功能可提供超出Turn-Around（TAR）窗口之外的时间。当eSPI从机需要超过Turn-Around（TAR）窗口的额外延迟时，它允许主机发起的non-posted事务（否则将立即DEFER响应）在同一事务中完成。 WAIT\_STATE响应码提供的额外延迟是一个字节的时间，分别对应于Single I / O模式下的8个串行时钟，Dual I / O模式下的4个串行时钟或Quad I / O模式下的2个串行时钟。

CRC计算中不包括WAIT\_STATE响应代码。它定义为与所有其他响应编码相比至少具有2位差异码。在接收到WAIT\_STATE响应代码时，要求eSPI主机处理等待状态。



Master Initiated Non-Posted Transaction Responded with WAIT STATE

## 传输层

### 周期类型和数据包格式



General eSPI Packet Format

#### Cycle Types

表中将支持的cycle类型的方向指定为“up”或“down”。 “up”是指从eSPI从机到eSPI主机的方向，“down”是指从eSPI主机到eSPI从机的方向。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Cycle type | Encoding  [7:0] | Direction | Command  Type | Description |
| espi 外设通道 | Memory  Read 32 | 00000000 | Up/down | Non-posted | 32位寻址内存读取请求。  有关数据包格式，请参见图36。 |
| Memory  Read 64 | 00000010 | Up/down | Non-posted | 64位寻址内存读取请求。  有关数据包格式，请参见图36。 |
| Memory  write 32 | 00000001 | Up/down | posted | 32位寻址内存写请求。  有关数据包格式，请参见图34。 |
| Memory  write 64 | 00000011 | Up/down | posted | 64位寻址内存写请求。  有关数据包格式，请参见图34。 |
| Message | 0001r2r1r00 | Up/down | posted | 消息请求。  有关数据包格式，请参见图38。 |
| Message  With data | 0001r2r1r01 | Up/down | posted | 带有数据有效负载的消息请求。  有关数据包格式，请参见图38。 |
| Successful  Completion  Without  data | 00000110 | Up | completion | 对应于I / O写入。 有关数据包格式，请参见图39。 |
| Successful  Completion  With data | 00001p1p­01 | Up/down | completion | 对应于内存读取或I / O读取。  有关数据包格式，请参见图39。 |
| Unsuccessful  Completion  Without  data | 00001p1p­00 | Up/down | completion | 对应于内存或I / O。  有关数据包格式，请参见图39。 |
| OOB  消息通道 | OOB  (Tunneled  SMBus)  Message | 00100001 | Up/down | posted | SMBus带外消息。  有关数据包格式，请参见图45：OOB（隧道SMBus）消息数据包格式。 |
| 内存访问通道 | Flash read | 00000000 | Up | Non-posted | Flash读。  有关数据包格式，请参见图48。 |
| Flash write | 00000001 | Up | Non-posted | Flash写。  有关数据包格式，请参见图48。 |
| Flash erase | 00000010 | Up | Non-posted | flash擦除指令。  有关数据包格式，请参见图48。 |
| Successful  Completion  Without  data | 00000110 | down | completion | 对应于Flash Write或Flash Erase。 有关数据包格式，请参见图39。 |
| Successful  Completion  With data | 00001p1p­01 | down | completion | 对应于Flash Read。 |
| Unsuccessful  Completion  Without  data | 00001p1p­00 | down | completion | 对应于Flash访问 |

上表中P1P0的编码定义如下

|  |  |
| --- | --- |
| **Encoding P1P0** | **Description** |
| 00 | Indicates the middle completion of a split completion sequence. |
| 01 | Indicates the first completion of a split completion sequence. |
| 10 | Indicates the last completion of a split completion sequence. |
| 11 | Indicates the only completion for a split transaction. |

对于和没有数据的失败传输，P1必须始终为“ 1”，因为这始终是最后或唯一的传输。

命令操作码和周期类型编码的组合必须唯一。不要求周期类型编码在命令操作码之间必须唯一

上表中r2r1r0的编码定义如下:

|  |  |
| --- | --- |
| Encoding r2r1r0 | 描述 |
| 000 | Local – Terminated at receiver. |
| 001-111 | Reserved |

#### tag

对于同一通道上需要完成的多个未完成的non-posted请求，“tag”字段可以是唯一的。有关标签及其与完成顺序的关联的更多详细信息，请参见第5.4节-事务排序规则。

4位标签字段允许在任何时候最多处理16个唯一的non-posted请求，彼此之间没有排序要求。

#### length

长度字段是指以字节为单位指定的请求大小或数据有效负载。 长度字单位长为1。 全零的值表示4 KB的长度。

对于内存读取和flash读取，长度字段是指请求的数据有效负载大小

对于内存写，flash写，带数据的OOB消息和带数据的传输，长度字段指定数据包中返回的实际数据量。

对于没有数据的传输或不成功的传输，长度字段必须由启动器驱动为零。 接收者必须忽略长度字段

对于“short I / O”和“short memory”，没有定义长度字段，因为事务长度嵌入在支持1、2或4字节访问的命令操作码本身中。 Short command不支持3字节访问。

有关闪存擦除长度字段的定义，请参阅“主附加flash共享（MAFS）”和“从属附加flash共享（SAFS）”部分以了解详细信息。

内存读取和flash读取请求可以一次或多次拆分完成。不超过相应通道的最大有效负载大小的内存读取或闪存读取必须一次完成。

#### address

eSPI外设通道存储器事务支持32位和64位寻址格式。 对于外设通道I / O周期，仅使用16位地址

对于4 GB以下的地址，内存事务必须使用32位寻址格式。 使用64位寻址格式时，高32位地址[63:32]不能全为0

对于具有总线主控能力的eSPI从机，必须支持64位寻址存储器事务。 eSPI主机必须同时支持32位和64位寻址存储器事务

对于其他内存事务，该地址可以在任何字节边界处开始或结束。 但是，地址和有效载荷长度的组合不得越过相应的“最大有效载荷大小”的自然对齐的地址边界。 它不能跨越4 KB地址边界。

对于Flash Access Channel和OOB消息，只要有效载荷长度不超过相应的最大有效载荷大小，就没有地址对齐要求。

#### data

无论地址对齐如何，有效数据字段始终从字节0开始。

### Channels

通道提供了一种方法，允许多个独立的数据流共享同一条物理总线。

put \_ \* / get \_ \* / \* \_ avail / \* \_ free的每组都与相应通道的命令和响应关联。每个通道都有其专用资源，例如队列和流控制。来自不同通道之间的数据流没有关系。

在初始化过程中，可以通过eSPI主机向eSPI机发出的GET\_CONFIGURATION命令来发现特定eSPI从机支持的通道的数量和类型。

通道类型和通道号的分配是固定的。eSPI从机只能发送支持哪些通道。

eSPI外设事务始终使用通道0。PUT\_PC/ PUT\_NP / GET\_PC / GET\_NP / PC\_FREE / NP\_FREE / PC\_AVAIL / NP\_AVAIL命令和状态字段用于通道0访问。

虚拟线通过通道1进行通信。PUT\_VWIRE / GET\_VWIRE / VWIRE\_AVAIL命令和状态字段用于通道1的访问。

OOB消息和Flash 访问分别使用通道2和通道3。

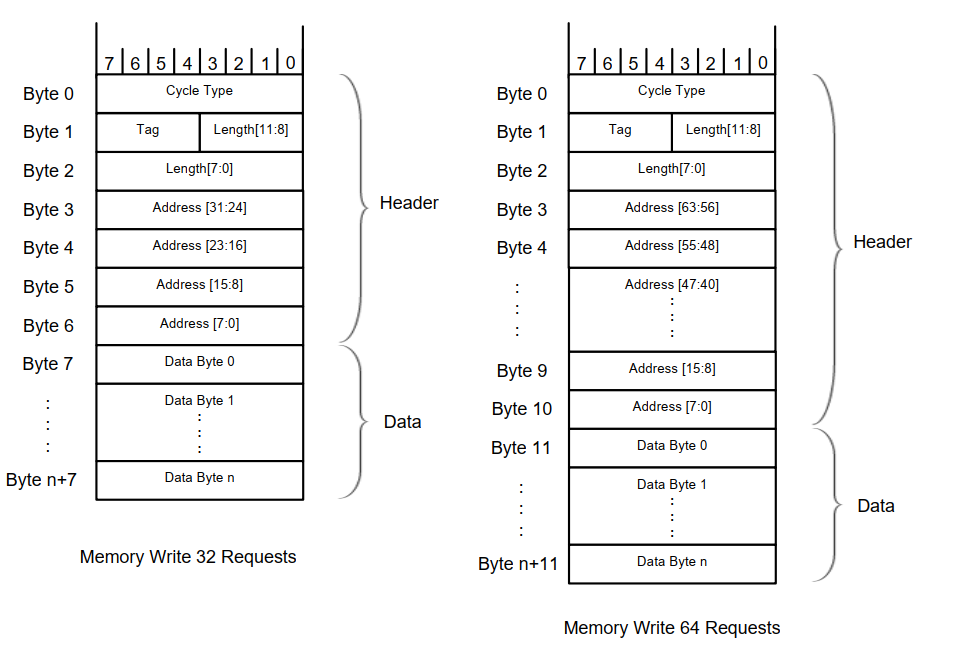
诸如GET\_STATUS，SET\_CONFIGURATION和GET\_CONFIGURATION之类的命令不与任何特定通道关联。

#### 外设通道

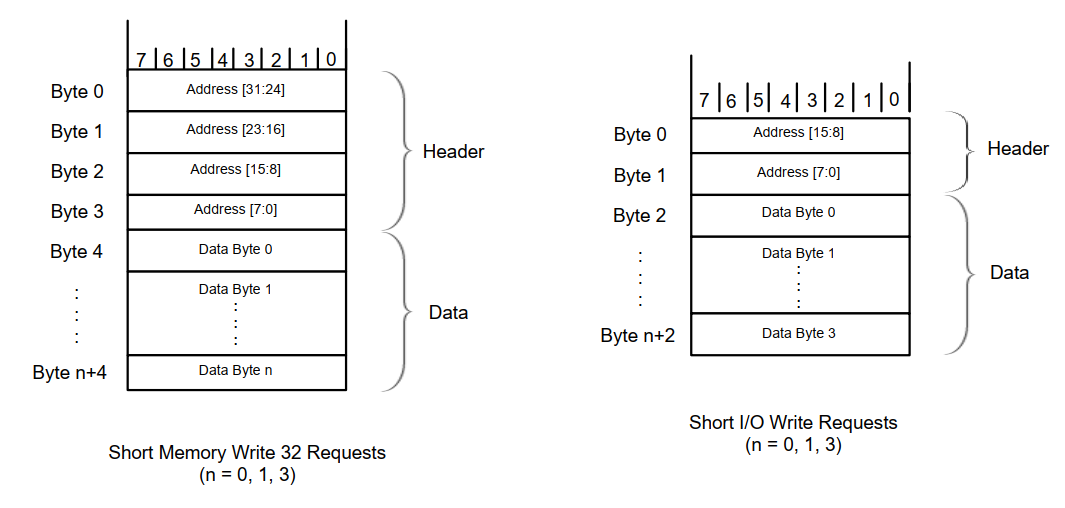
eSPI外围存储请求包，I / O请求包，消息请求包和完成的格式如下所示。

仅对于请求长度为1、2或4个字节的Short I / O命令操作码支持I / O事务。 不支持3字节I / O事务。

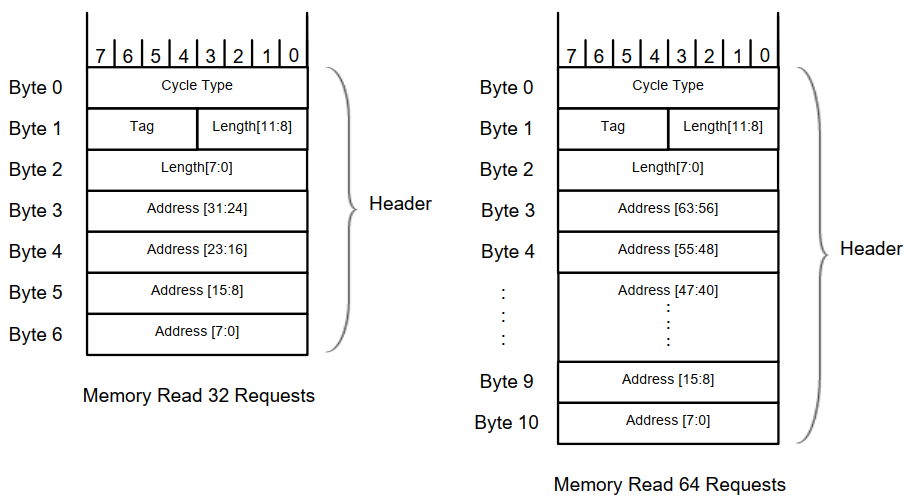
标签和频道信息用于将补全与相应的请求进行匹配。



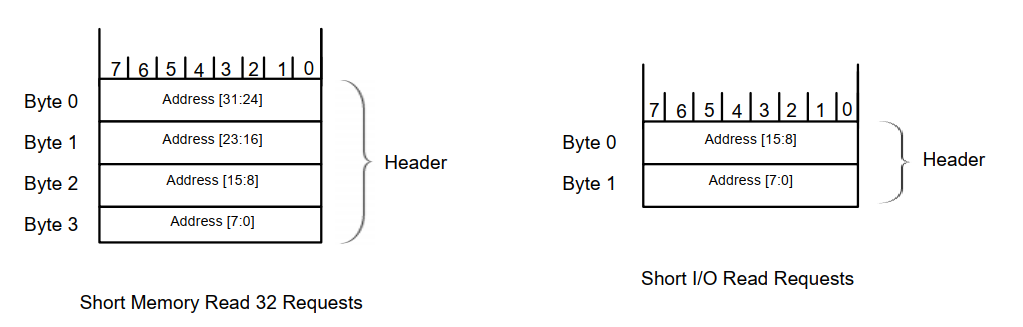
Peripheral Memory Write Packet Format



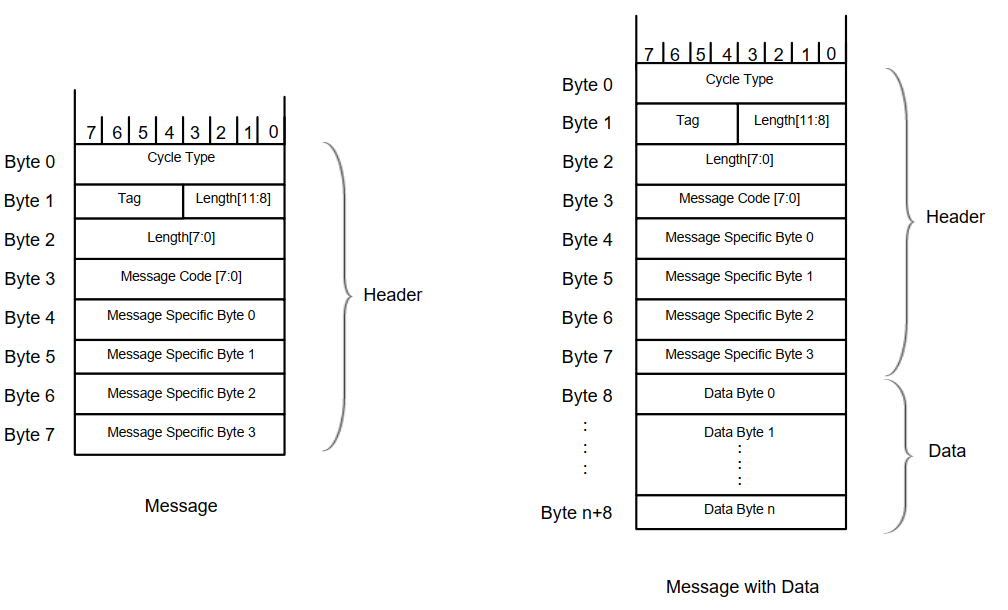
Short Peripheral Memory or Short I/O Write Packet Format (Master Initiated only)



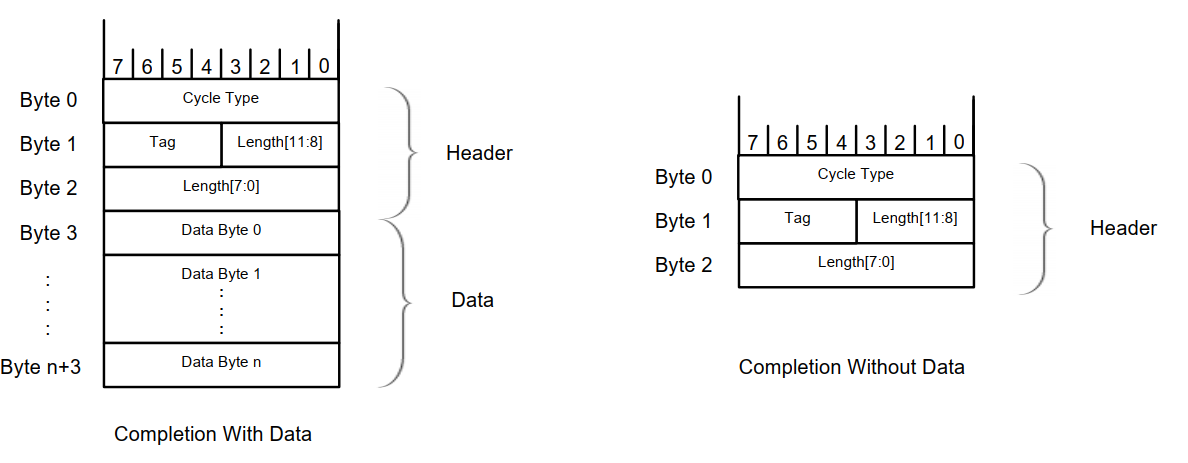
Peripheral Memory Read Packet Format



Short Peripheral Memory or Short I/O Read Packet Format (Master Initiated only)



Peripheral Message Packet Format



Peripheral Memory or I/O Completion With and Without Data Packet Format

内存读取和写入请求可以由eSPI主机和从机发起。短I / O读和写请求以及短内存读和写请求只能由eSPI主机启动。

消息和带有数据周期的消息类型是使用PUT\_PC和GET\_PC命令操作码发布的事务。 消息和带有数据的消息包都使用相同的头格式。长度字段指定带有数据的消息中有效负载的大小。 消息标题中不包括消息头中的4个消息特定字节。 对于消息周期类型，“长度”字段为保留字段，并且必须发送全0

包头中的消息代码中定义消息的功能和用法。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Cycle Type** | **Message**  **Code [7:0]** | **Routing**  **r2r1r0** | **Direction** | **Description** |
| LTR | Message | 0000\_0001 | 000 | Up | Latency Tolerance Reporting. |

延迟容忍报告（Latency Tolerance Reporting ; LTR）使eSPI从机能够报告其对外设通道存储器读写的服务延迟要求，因此可以在考虑eSPI从机服务要求的情况下实施电源管理。

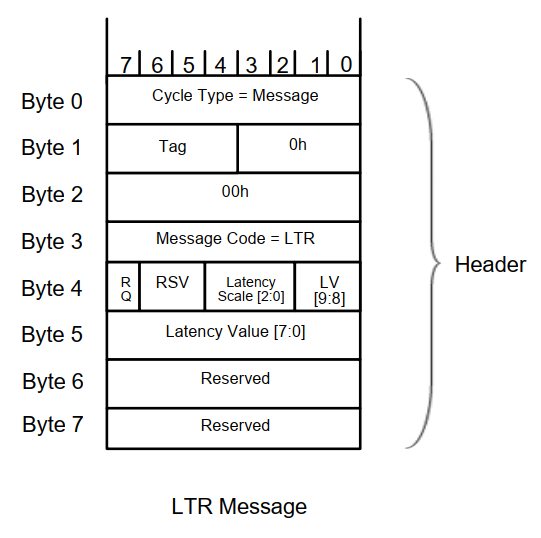
所有eSPI主机必须支持LTR消息。对于使用外设通道存储器请求支持总线主控的eSPI从机，必须使用LTR。

若将“延迟值”字段设置为全0表示eSPI从机将受到任何延迟的影响，并要求提供最佳服务

如果eSPI从机没有服务要求，则必须清除Requirement位。支持LTR的eSPI从机必须在发出任何存储器请求之前发送初始LTR消息。 建议在使能外设通道后，eSPI从机立即发送LTR消息。每当服务请求发生变化时，eSPI从机应发送更新的LTR消息。

强烈建议eSPI从机在任何500μs的时间段内发送不超过两个LTR消息。 如果在500μs的时间内接收到两个以上的LTR消息，则eSPI主机不得产生错误。

LTR使用没有数据有效载荷的消息周期类型。



LTR Message Format

|  |  |
| --- | --- |
| Message field | Description |
| Requirement | “ 0”表示eSPI从机没有服务要求。 当该位为1时，其余字段有效以指示eSPI从机的等待时间要求。 |
| Latency scale | 这是“延迟值”（LV [9：0]）字段的乘数，以产生延迟延迟的绝对时间值   |  |  | | --- | --- | | LS[2:0] | Description | | 000 | Value times 1ns | | 001 | Value times 32ns | | 010 | Value times 1024ns | | 011 | Value times 32768ns | | 100 | Value times 1048576ns | | 101 | Value times 33554432ns | | 110-111 | Reserved | |
| Latency Value | 它指定了eSPI从机可容忍的服务延迟。 |

#### 虚拟线通道

虚拟线通道用于通过带内消息传递边带引脚或GPIO的状态。 串行IRQ中断作为带内消息通过此通道进行通信。

命令阶段包括命令操作码，虚拟线数据包和CRC

虚拟线数据包以虚拟线计数作为头字节开头，其中计数指示数据包通信的虚拟线组的数量。 然后是一个或多个Virtual Wire组。 每个虚拟线组由2个字节组成，即虚拟线索引和虚拟线数据。 允许在同一数据包中发送最多64个组的多个虚拟线路组。

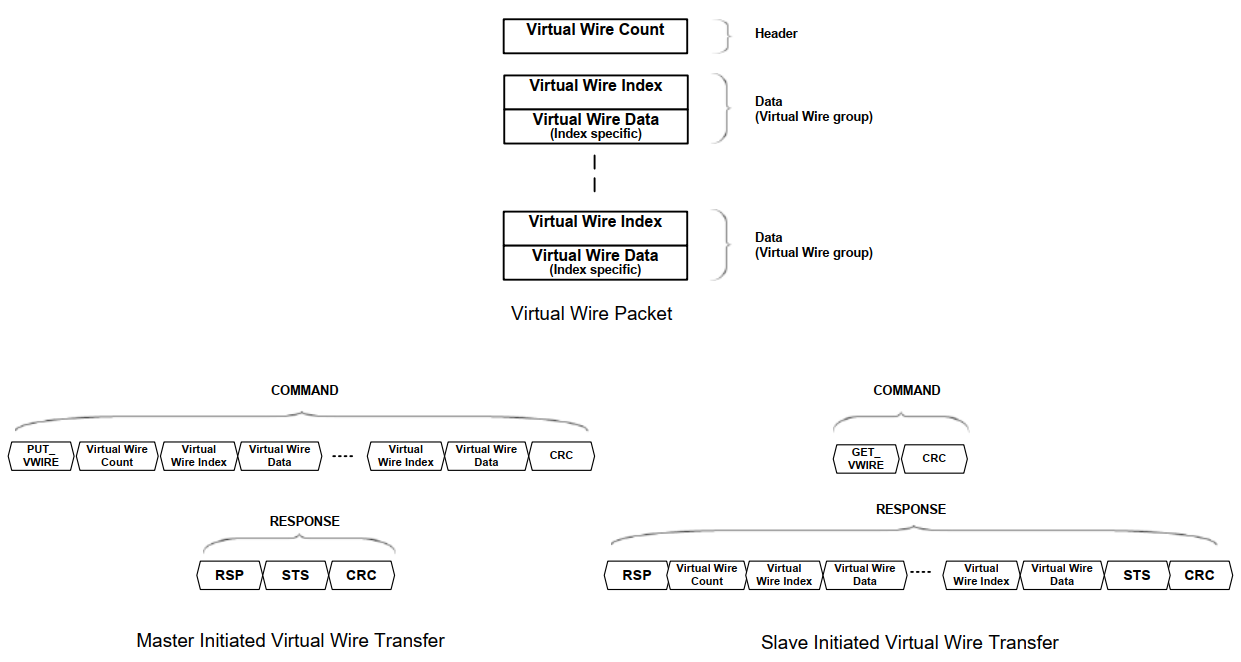
虚拟线路计数的定义如下：

|  |  |
| --- | --- |
| **Bits** | **Description** |
| 7:6 | Reserved |
| 5:0 | 6位计数字段允许在同一数据包中最多通信64个虚拟线组。 这是从0开始的计数。 |

特定虚拟线索引中未使用的虚拟线槽将被保留。 发起方必须将预留字段驱动为“ 0”，接收方必须忽略预留字段

虚拟线路数据包不受流量控制。 只要启用了通道，接收器就必须随时准备好接收虚拟线数据包。虚拟线数据包的长度为（1 + 2 \* n）字节，其中n是数据包中虚拟线组的数量。与其他通道的流量相比，虚拟线应具有最高优先级，以确保将延迟保持在最低水平。

下图显示了虚拟线数据包格式



Virtual Wire Packet Format

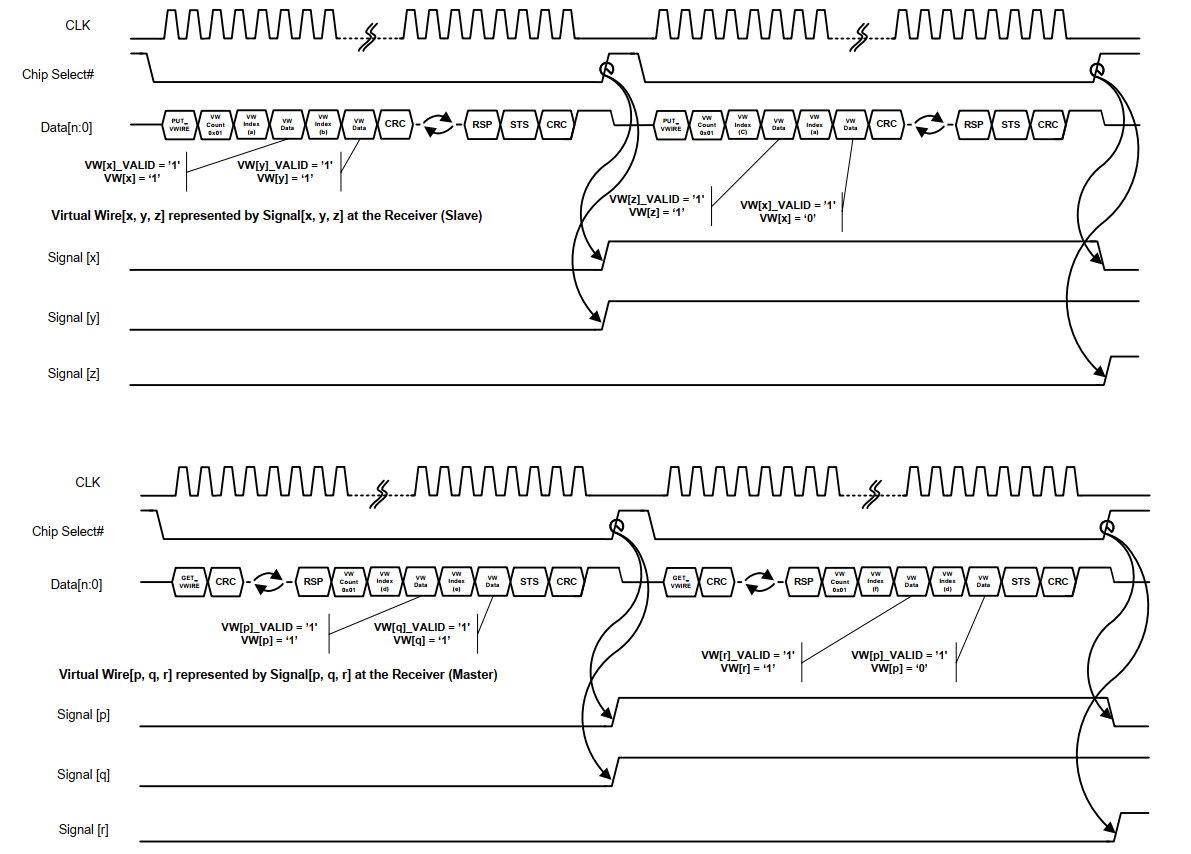
一些虚拟线路（例如PME和中断）可以由多个eSPI从机共享。 eSPI主机必须为每个eSPI从机独立地跟踪各个虚拟线的逻辑状态，并负责聚合不同组的虚拟线。

从eSPI主机到从机的消息仅是单播，这意味着它们只能针对一个特定的eSPI从机。

当通过链接接收到指示平台重置的PLTRST＃消息时，应将“平台重置”重置的各个虚拟线初始化为默认值。 为避免行为不明确，建议不要将通过Platform Reset（平台重置）重置的Virtual Wire与PLTRST＃Virtual Wire一起发送到单个Virtual Wire数据包中。 在任何情况下，PLTRST＃都具有优先权，并且不管这些虚拟线在数据包中发送的级别如何，都会重置这些虚拟线。

如果收到不支持的虚拟线路消息，则接收方应静默丢弃该消息。

仅当在事务处理结束时取消选择CS＃时，通过eSPI通道的虚拟线才会在接收器上生效。 启动器只能假定虚拟线是在Chip Select＃的取消声明边缘发送的，目的是使物理引脚状态更改与所通信的虚拟线同步。



Virtual Wires at the Receiver

|  |  |  |  |
| --- | --- | --- | --- |
| Virtual Wire Index | | Virtual Wire Group | Virtual Wire Data Format |
| Start | End |
| 0 | 1 | Interrupt event | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | L | IRQ Line | | | | | | |  |  |  | | --- | --- | | Bit | Description | | 7 | 中断电平  0b:low 1b:high | | 6:0 | 指定要发送到中断控制器的中断（IRQ）线  Index= 0h IRQ 0-127  Index=1h:IRQ 128-255 |   虚拟线中断事件定义的是从机到主机的过程。  中断电平高（“ 1”）表示中断声明。 中断事件虚拟线为高电平有效。 |

|  |  |  |  |
| --- | --- | --- | --- |
| Virtual Wire Index | | Virtual Wire Group | Virtual Wire Data Format |
| Start | End |
| 2 | 7 | system event | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | Valid | | | | Level | | | |  |  |  | | --- | --- | | Bit | Description | | 7:4 | valid：此字段指示一对一对应的Level位的有效性  0b: 无效 1b:有效  该位用作“掩码”位。  如果为“ 0”，则对应的虚拟线路必须保留其先前的值，并且不得为此虚拟线路包进行更新。 | | 3:0 | level：此字段中的每个位指示要传送的虚拟有线信号的状态。  0b: low 1b:high | |
| 8 | 63 | Reserved | Reserved |
| 64 | 127 | Platform specific | Platform specific |

|  |  |  |  |
| --- | --- | --- | --- |
| Virtual Wire Index | | Virtual Wire Group | Virtual Wire Data Format |
| Start | End |
| 128 | 255 | 通用I / O扩展器 | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | Valid | | | | Level | | | |  |  |  | | --- | --- | | Bit | Description | | 7:4 | valid：此字段指示一对一对应的Level位的有效性  0b: 无效 1b:有效  该位用作“掩码”位。  如果为“ 0”，则对应的虚拟线路必须保留其先前的值，并且不得为此虚拟线路包进行更新。 | | 3:0 | level：该字段中的每个位指示要通信的虚拟GPIO的状态。  0b: low 1b:high | |

eSPI规范定义了以下系统事件虚拟线，涉及平台独立的标准边带信号。

除非明确指出，否则默认情况下所有系统事件（虚拟线索引2到7）都是电平。其中包括经过级别触发的ERROR FATAL和ERROR NON-FATAL虚拟线。系统事件的任何状态更改都会导致新状态与相应的电平（“ 0”或“ 1”）进行通信。 各个虚拟线的默认重置状态如下虚拟表中所述。

如果支持，则必须根据规范实施这些标准虚拟线路，以实现互操作性和兼容性。 但是，这些虚拟线的支持是特定于平台的。

|  |  |
| --- | --- |
| **Virtual Wire Index** | **2** |
| **Virtual Wire Group** | **System Event** |
| **Reset** | **eSPI Reset# 1** |
| **Direction** | **Master to Slave** |

|  |  |  |
| --- | --- | --- |
| Bit | Virtual | Description |
| 7 |  | Reserved |
| 6 | SLP\_S5＃有效：该位指示位[2]上SLP\_S5＃虚拟线的有效性。  0:无效 1:有效 |
| 5 | SLP\_S5＃有效：该位指示位[2]上SLP\_S5＃虚拟线的有效性。  0：无效 1：有效 |
| 4 | SLP\_S3＃有效：该位指示位[0]上SLP\_S3＃虚拟线的有效性。  0：无效 1：有效 |
| 3 | RSV | Reserved |
| 2 | SLP\_S5# | S5睡眠控制：当应在S5中关闭非关键系统的电源时发送（软关机）。  极性：低电平有效。 reset：有效。 |
| 1 | SLP\_S4# | S4睡眠控制：当应在S4（挂起磁盘）中关闭非关键系统的电源时发送。  极性：低电平有效。 reset：有效。 |
| 0 | SLP\_S3# | S3睡眠控制：当应在S3中关闭非关键系统的电源（挂到RAM）时发送。  极性：低电平有效。 reset：有效。 |

|  |  |
| --- | --- |
| **Virtual Wire Index** | **3** |
| **Virtual Wire Group** | **System Event** |
| **Reset** | **eSPI Reset#** |
| **Direction** | **Master to Slave** |

|  |  |  |
| --- | --- | --- |
| Bit | Virtual | Description |
| 7 |  | Reserved |
| 6 | OOB\_RST\_WARN有效：该位指示[2]上OOB\_RST\_WARN虚拟线的有效性。  0:无效 1:有效 |
| 5 | PLTRST＃有效：该位指示位[1]上PLTRST＃虚拟线的有效性。  0：无效 1：有效 |
| 4 | SUS\_STAT＃有效：该位指示位[0]上SUS\_STAT＃虚拟线的有效性。  0：无效 1：有效 |
| 3 | RSV | Reserved |
| 2 | OOB\_RST\_WARN | OOB复位警告：由主机在OOB处理器即将进入复位状态之前发送。收到后，EC或BMC必须刷新并使其OOB通道上游请求队列停顿，并在完成所有未完成的事务后声明OOB\_RST\_ACK VWire。主机随后完成所有未完成的posted事务，然后通过写从机的配置寄存器来禁用OOB通道。  极性：低电平有效。 reset：有效。 |
| 1 | PLTRST# | 平台重置：用于指示平台重置声明和取消声明的命令。  极性：低电平有效。 reset：有效。 |
| 0 | SUS\_STAT# | 挂起状态：在系统即将进入低功耗状态时发送。  极性：低电平有效。 reset：有效。 |

|  |  |
| --- | --- |
| **Virtual Wire Index** | **4** |
| **Virtual Wire Group** | **System Event** |
| **Reset** | **eSPI Reset#** |
| **Direction** | **Slave to Master** |

|  |  |  |
| --- | --- | --- |
| Bit | Virtual | Description |
| 7 |  | PME＃有效：该位指示位[3]上PME＃虚拟线的有效性。  0:无效 1:有效 |
| 6 | WAKE＃有效：此位指示位[2]上WAKE＃虚拟线的有效性。  0:无效 1:有效 |
| 5 | Reserved |
| 4 | OOB\_RST\_ACK有效：该位指示位[0]上OOB\_RST\_ACK虚拟线的有效性。  0：无效 1：有效 |
| 3 | PME# | PCI电源管理事件：eSPI从机生成了PCI PME＃事件。 从机用来通过PCI定义的PME＃从Sx唤醒主机。  如果事件在系统处于S0时发生，则会生成SCI。 由平台上的多个PCI设备共享。  极性：低电平有效。 reset：有效。 |
| 2 | WAKE# | Wake＃：在任何事件下，从机都使用它从Sx唤醒主机； LID开关或AC插入等时唤醒的通用事件。它用于生成特定于eSPI器件的非PME＃唤醒。 如果事件在系统处于S0时发生，则会生成SCI。 由多个eSPI端点共享。  注意：eSPI WAKE＃虚拟线不等同于PCIe WAKE＃引脚，并且不能用作PCIe WAKE＃。  极性：低电平有效。 reset：有效。 |
| 1 | RSV | Reserved |
| 0 | OOB\_RST\_ACK | 注意：eSPI WAKE＃虚拟线不等同于PCIe WAKE＃引脚，并且不能用作PCIe WAKE＃。  极性：低电平有效。 reset：有效。 |

|  |  |
| --- | --- |
| **Virtual Wire Index** | **5** |
| **Virtual Wire Group** | **System Event** |
| **Reset** | **eSPI Reset#** |
| **Direction** | **Slave to Master** |

|  |  |  |
| --- | --- | --- |
| Bit | Virtual | Description |
| 7 |  | SLAVE\_BOOT\_LOAD\_STATUS有效：该位指示SLAVE\_BOOT\_LOAD\_STATUS虚拟线的有效性[3]  0:无效 1:有效 |
| 6 | ERROR\_NONFATAL有效：该位指示ERROR\_NONFATAL虚拟线在位[2]上的有效性  0:无效 1:有效 |
| 5 | ERROR\_FATAL有效：该位指示第[1]位上ERROR\_FATAL虚拟线的有效性。  0：无效 1：有效 |
| 4 | SLAVE\_BOOT\_LOAD\_DONE有效：该位指示位[0]上SLAVE\_BOOT\_LOAD\_DONE虚拟线的有效性。  0：无效 1：有效 |
| 3 | SLAVE\_BOOT\_  LOAD\_STATUS | 从引导加载状态：完成从主闪存连接的从引导加载后，由EC或BMC发送。  0:引导映像已损坏，不完整或无法使用。  1:引导代码加载成功，并且映像的完整性是完整的，或者不需要从主附加闪存加载引导代码。  注意：必须以与Slave\_Boot\_Load\_Done相同或先前的虚拟有线消息发送Slave\_Boot\_Load\_Status。  极性：如上所述。  重置：“ 0”。 |
| 2 | ERROR\_NONFATAL | 错误非致命错误：当检测到非致命错误时由从机发送。  极性：低电平有效。 reset：有效。 |
| 1 | ERROR\_FATAL | 错误致命：检测到致命错误时由从机发送。  极性：低电平有效。 reset：有效。 |
| 0 | SLAVE\_BOOT\_  LOAD\_DONE | 从引导加载完成：当EC或BMC完成其引导过程时发送，以指示eSPI主机继续从G3到S0退出。 eSPI主机会在继续进行SLP\_S5＃取消声明之前等待该虚拟线路的声明。  极性：低电平有效。 reset：有效。 |

|  |  |
| --- | --- |
| **Virtual Wire Index** | **6** |
| **Virtual Wire Group** | **System Event** |
| **Reset** | **PLTRST#** |
| **Direction** | **Slave to Master** |

|  |  |  |
| --- | --- | --- |
| Bit | Virtual | Description |
| 7 |  | HOST\_RST\_ACK有效：该位指示HOST\_RST\_ACK虚拟线路在位[3]上的有效性。  0:无效 1:有效 |
| 6 | RCIN＃有效：该位指示位[2]上的RCIN＃虚拟线的有效性。  0:无效 1:有效 |
| 5 | SMI＃有效：该位指示位[1]上SMI＃虚拟线的有效性。0：无效 1：有效 |
| 4 | SCI＃有效：该位指示位[0]上SCI＃虚拟线的有效性。  0：无效 1：有效 |
| 3 | HOST\_RST\_ACK | 主机复位确认：由从机响应HOST\_RST\_WARN虚拟线路发送。  极性：低电平有效。 reset：有效。 |
| 2 | RCIN# | 重置CPU INIT＃：发送代表键盘控制器请求CPU重置。  极性：低电平有效。 reset：有效。 |
| 1 | SMI# | 系统管理中断（SMI）：作为通用警报发送，导致BIOS调用SMI代码。  极性：低电平有效。 reset：有效。 |
| 0 | SCI# | 系统控制器中断（SCI）：作为通用警报发送，导致OS调用ACPI方法。  极性：低电平有效。 reset：有效。 |

|  |  |
| --- | --- |
| **Virtual Wire Index** | **7** |
| **Virtual Wire Group** | **System Event** |
| **Reset** | **PLTRST#** |
| **Direction** | **Master to Slave** |

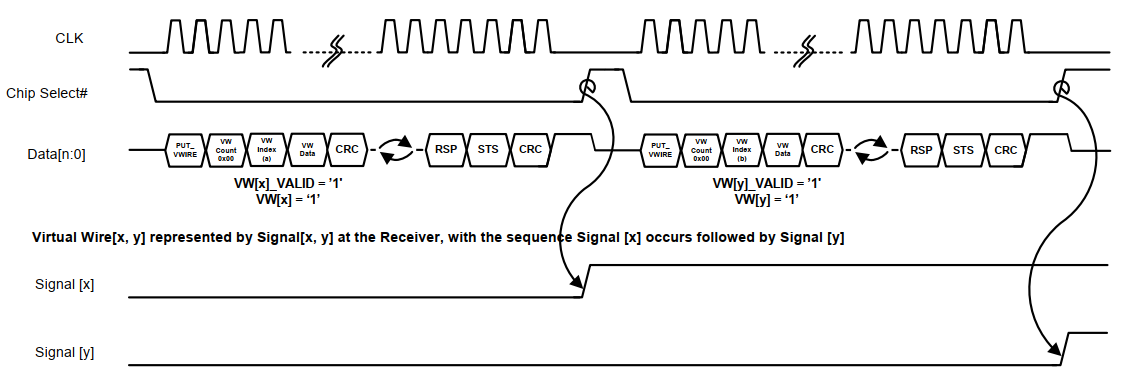
|  |  |  |
| --- | --- | --- |
| Bit | Virtual | Description |
| 7 |  | RESVERED |
| 6 | NMIOUT＃有效：该位指示位[2]上NMIOUT＃虚拟线的有效性。  0:无效 1:有效 |
| 5 | SMIOUT＃有效：该位指示位[1]上SMIOUT＃虚拟线的有效性。  0：无效 1：有效 |
| 4 | HOST\_RST\_WARN有效：该位指示位[0]上的HOST\_RST\_WARN虚拟线的有效性。  0：无效 1：有效 |
| 3 | RSV | RESVERED |
| 2 | NMIOUT# | NMI输出：由主机发送，以指示发生NMI＃事件。 该虚拟线路上的“ 0”和“ 1”分别对应于NMI＃对CPU的断言和断言。  极性：低电平有效。 reset：有效。 |
| 1 | SMIOUT# | SMI输出：由主机发送，以指示发生SMI＃事件。 该虚拟线路上的“ 0”和“ 1”分别对应于SMI＃对CPU的断言和断言。  极性：低电平有效。 reset：有效。 |
| 0 | HOST\_RST\_WARN | 主机重置警告：由主机在主机即将进入重置之前发送。 接收后，EC或BMC必须刷新并停止其上游外围设备请求队列，并在完成所有未完成的事务后声明HOST\_RST\_ACK VWire。 主机随后完成所有未完成的posted事务或完成，然后通过写从机的配置寄存器来禁用外围通道。  极性：低电平有效。 reset：有效。 |

通过虚拟有线频道传达的某些事件可能是计时事件

例如，特定引脚的断言可以指示一个事件。 在一定时间段内对同一引脚的长时间置位可能表示发生了不同的事件。

一种解决方案是针对每个事件发送两种不同的消息。当引脚置为有效时发送一条消息，而引脚已确定为一定时间段则发送另一条消息. 此方法要求消息的源实现一个计时器，该计时器对引脚声明的持续时间进行计时，该超时在超时后会导致发送不同的消息。

在同一数据包中通信的多条虚拟线将在CS# 信号取消声明边缘同时更改接收器的状态。 如果需要传递虚拟线的序列，则必须在下一个CS＃设置中传递以后发生的虚拟线，以表示该序列



Virtual Wires with Sequence Communicated

从eSPI从机到虚拟线通道的主机都支持中断事件。 定义了虚拟线索引0和1用于通信中断事件，并且可以通过eSPI总线在带内最多传输256条IRQ线。

为简单起见，eSPI中断事件虚拟线默认为无效（低电平），从而消除了传统的SERIRQ行为，即使预期不会产生中断，默认情况下SERIRQ线也默认拉高（“1”）。

|  |  |  |  |
| --- | --- | --- | --- |
| IRQ Source (In Slave) | Source Level | IRQ Enable  (0=Disable, 1=Enable) | Slave to Master  (Active High) IRQ Virtual Wire |
| Active high | 0 | 0 | Default. No IRQ VW sent |
| 0 | 0 -> 1  1 -> 0 | No IRQ VW sent |
| 0 -> 1 | 1 | Assertion. IRQ VW (Level=’1’) sent |
| 1 -> 0 | 1 | Deassertion. IRQ VW (Level=’0’) sent |
| 1 | 0 -> 1 | Assertion. IRQ VW (Level=’1’) sent |
| 1 | 1 -> 0 | Deassertion. IRQ VW (Level=’0’) sent |
| 0 -> 1  1 -> 0 | 0 | No IRQ VW sent |
|  | 1 | 0 | Default. No IRQ VW sent |
|  | 1 | 0 -> 1  1 -> 0 | No IRQ VW sent |
|  | 0 -> 1 | 1 | Deassertion. IRQ VW (Level=’0’) sent |
|  | 1 -> 0 | 1 | Assertion. IRQ VW (Level=’1’) sent |
|  | 0 | 0 -> 1 | Assertion. IRQ VW (Level=’1’) sent |
|  | 0 | 1 -> 0 | Deassertion. IRQ VW (Level=’0’) sent |
|  | 0 -> 1  1 -> 0 | 0 | No IRQ VW sent |

中断支持电平触发和边沿触发的中断

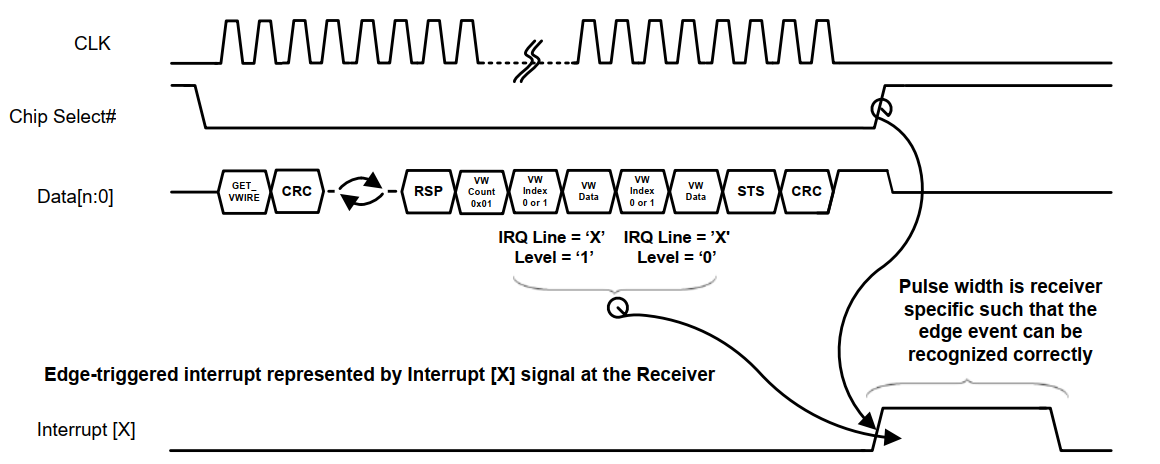
对于电平触发的中断，只要中断线的状态从“ 0”变为“ 1”（反之亦然），中断线的电平就会通过虚拟线进行通信。在初始化期间，必须将分配的中断线配置到从机。 允许从机上的多个中断源在一条电平触发的中断线上共享。 共享中断线在所有中断得到服务之前不会改变状态，这将触发从机发送的中断事件虚拟线。

为避免在使用电平触发的中断时产生虚假中断，建议驱动程序和eSPI从机执行以下操作：驱动程序完成中断服务例程后，应向eSPI从机发出posted的内存写操作，以清除中断。然后，驱动程序应向同一中断清除寄存器发出读操作。在eSPI从机上，应在清除中断的posted内存写入与返回后续内存读取完成之间按清除方式发送中断事件虚拟线路

包括中断事件虚拟线路的每个虚拟线路在虚拟线路数据包内所传递的跃迁不得超过2个，否则该虚拟线路行为未能被定义。 启动器必须确保不违反此要求。 接收者可以检查是否违反此规定。 这两个转换可能是“ 0”到“ 1”到“ 0”或“ 1”到“ 0”到“ 1”的转换。 接收器必须在接收到中断事件虚拟线时跟踪其状态，并将其转换为在CS＃的无效位沿生效的中断事件脉冲。

eSPI从机可以使用两个单独的虚拟线数据包通信边沿触发的中断，即在一个虚拟线数据包中声明中断，然后在随后的虚拟线数据包中取消断言。 但是，从机有责任确保始终保持中断声明边沿的通信不丢失，例如在能够发出前一个中断解除声明虚拟线之前发生下一个中断事件的边界。

由于中断事件虚拟线是通过来自外设访问的独立通道进行通信的，因此在将中断传递给CPU之前，可能无法保证数据在主存储器中。为了保证数据的一致性， 软件必须对从机中的任何寄存器执行读取操作，以便在访问由从机写入主存储器的数据之前刷新从机的posted写缓冲区



Edge-triggered Interrupt through Virtual Wire

如果将虚拟GPIO配置为输出引脚，则eSPI主机通过带内消息传递建立虚拟GPIO引脚的状态，并且eSPI从机在接收到该消息后，会在物理上位于从机侧的GPIO引脚上反映该状态。 虚拟GPIO被配置为输出引脚，eSPI主机通过带内消息传递建立虚拟GPIO引脚的状态，并且eSPI从机在接收到该消息后会反映物理上位于从机侧的GPIO引脚上的状态。

如果将虚拟GPIO配置为输入引脚，则eSPI从机会采样物理GPIO引脚的状态，然后在任何引脚状态转换时通过带内消息传递GPIO引脚的状态

共享相同索引号的所有GPIO引脚必须配置为相同方向。 它们可以配置为所有输入或所有输出，但不能配置为输入和输出的组合

同样，一组共享相同索引的虚拟GPIO将共享相同的复位。 可以通过eSPI reset#或平台重置来复位该重置

虚拟GPIO引脚到eSPI从机侧的物理GPIO引脚的映射是特定于实现的，不在规范范围内

#### OOB消息通道

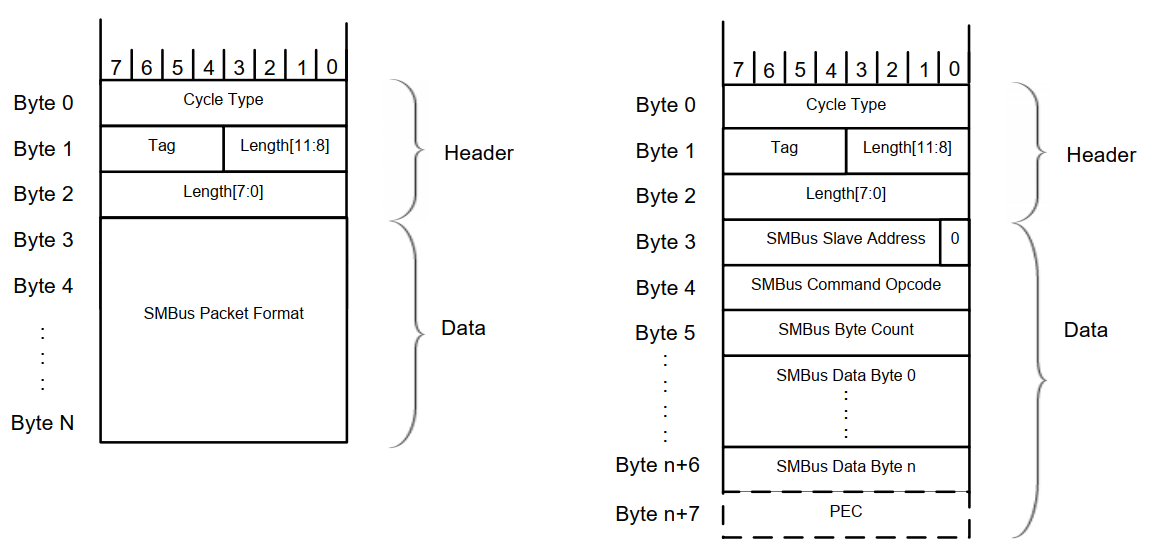
SMBus数据包可以作为带外（OOB）消息通过eSPI进行隧道传输。整个SMBus数据包作为数据嵌入到eSPI OOB消息中

仅SMBus块写入通过eSPI OOB消息进行隧道传输。这些包括基于SMBus块写入协议的SMBus管理组件传输协议（MCTP）数据包。SMBus从机地址，SMBus命令操作码，SMBus字节计数，SMBus数据字段和可选的PEC字节作为数据发送到eSPI OOB消息包中。SMBus字节计数字段不包括PEC字节。 它包含SMBus块写入数据包本身的实际有效负载，其中不包括3个SMBus标头字节。

除了3个标头字节（即SMBus从地址，SMBus命令操作码和SMBus字节数）和一个可选的PEC字节之外，OOB消息的Length字段还包括SMBus字节计数字段的计数。

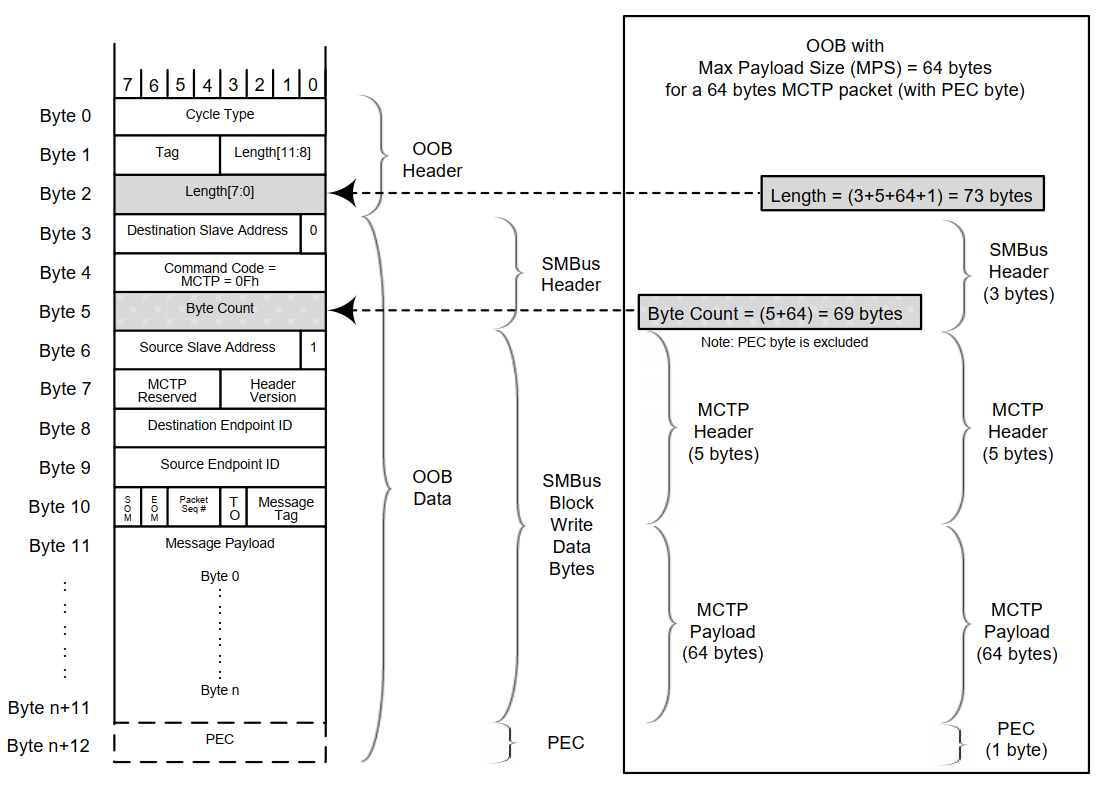
SMBus PEC的存在是通过eSPI OOB标头长度字段和SMBus字节数之间的简单算术运算确定的。

OOB消息通道的最大有效负载大小（MPS）适用于通过该通道隧穿的数据包中嵌入的协议的实际有效负载，例如但不限于MCTP和通用SMBus块写入。



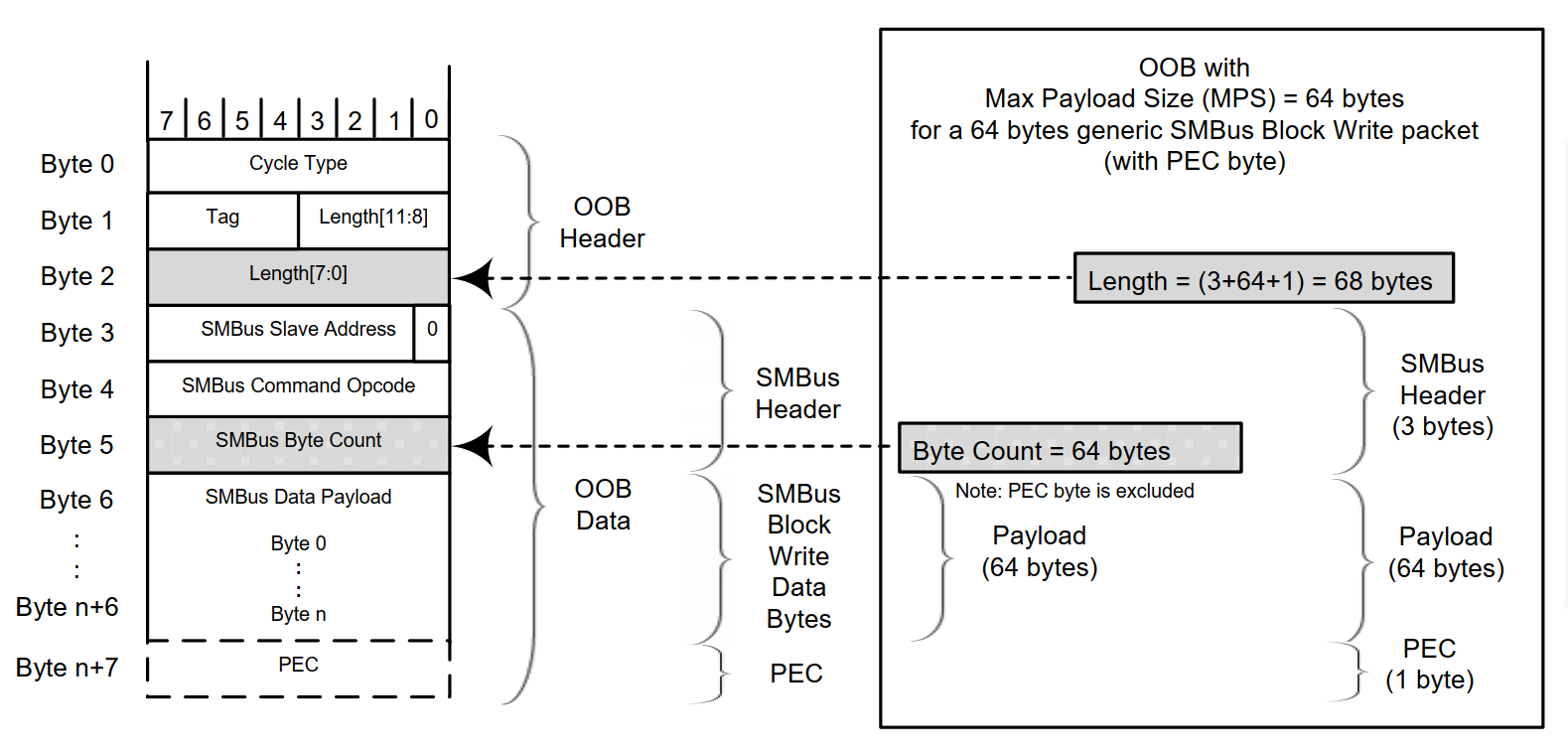
OOB (Tunneled SMBus) Message Packet Format

SMBus上的MCTP是SMBus块写入数据包的一种特殊形式，其SMBus命令操作码为0Fh（即MCTP）。 MCTP标头和MCTP有效负载被嵌入为SMBus块写入数据字节。 对于eSPI OOB MCTP数据包，最大有效负载大小（MPS）适用于MCTP有效负载本身，不包括MCTP标头和可选的PEC字节。 例如，64字节的MPS允许在OOB消息通道上传输具有高达64字节MCTP有效负载的MCTP数据包。 对于带有可选PEC字节的64字节MCTP有效负载，SMBus字节计数字段和OOB标头长度字段分别为69字节和73字节。



OOB MCTP Packet

对于eSPI OOB通用SMBus块写入数据包，最大有效负载大小（MPS）适用于数据包中允许的SMBus块写入数据字节数（不包括可选PEC字节）。 例如，64字节的MPS允许通过OOB消息通道传输具有多达64字节数据有效载荷的通用SMBus块写入数据包。 对于带有可选PEC字节的64字节数据有效载荷，SMBus字节计数字段和OOB标头长度字段分别为64字节和68字节。



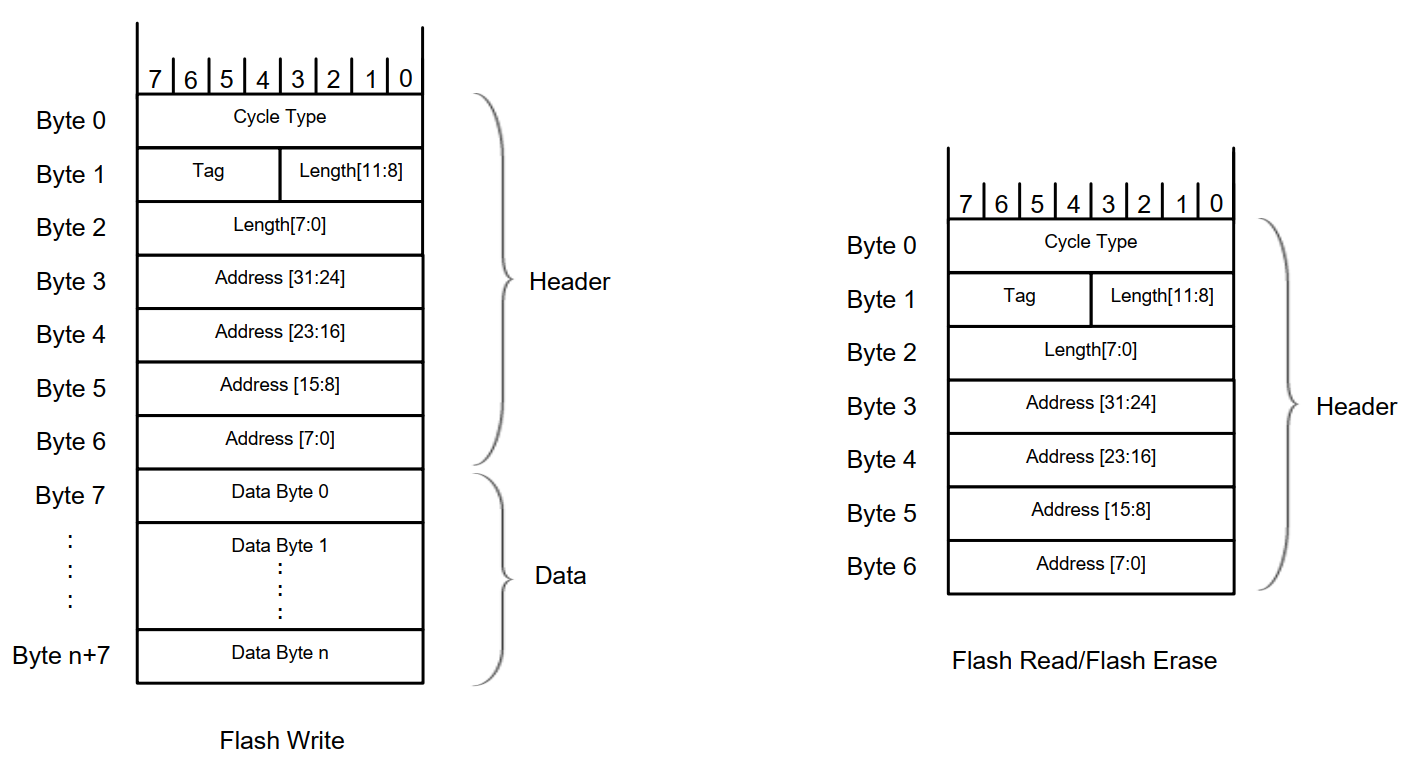
OOB Generic SMBus Block Write Format

#### 运行时Flash访问通道

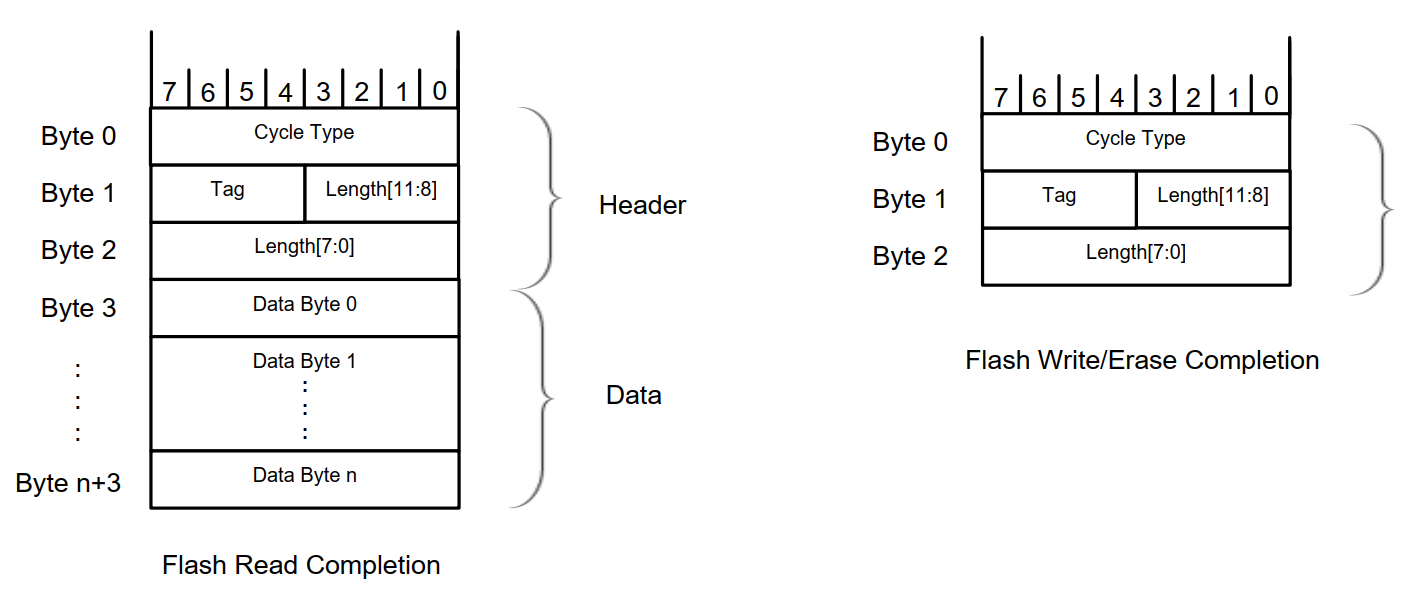
flash访问通道提供了一条路径，允许flash组件在芯片组和需要进行flash访问的eSPI从机（例如EC和BMC）之间共享运行时

芯片组中的flash控制器完成flash初始化后，将在eSPI从机侧启用闪存访问通道。flash访问通道使用与eSPI外设事务相同的数据包格式。

标签字段用于将完成与请求匹配。 具有相同标签的Flash访问请求必须按顺序完成。

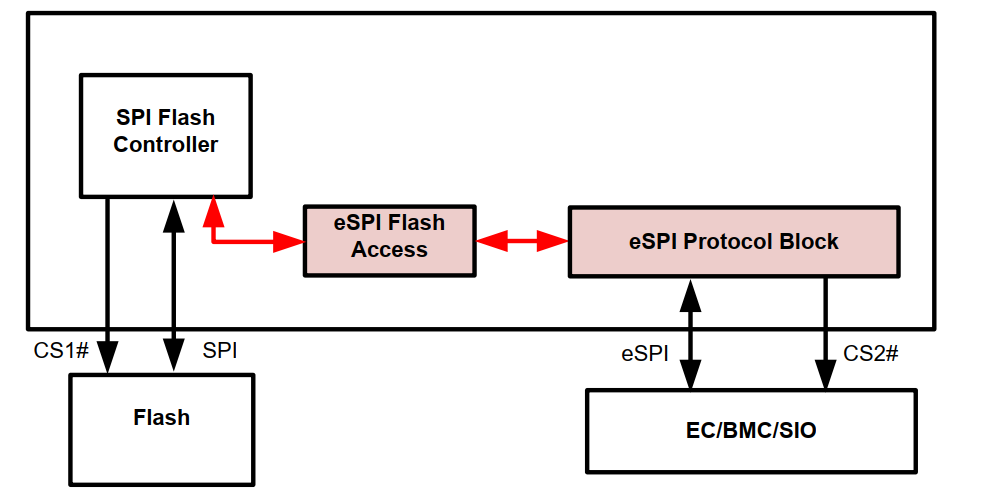


Flash Access Request Packet Format

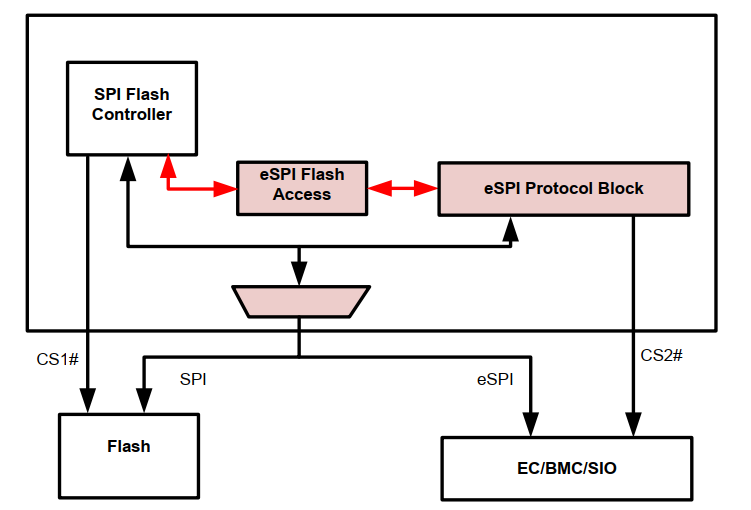


Flash Access Completion Packet Format

主机连接的flash共享是指将flash组件连接到eSPI主机（例如芯片组）的方案。 允许eSPI从机通过Flash Access通道访问共享的Flash组件。 根据系统配置，flash组件可以位于独立的SPI接口上，也可以位于共享的SPI / eSPI接口上。但是，长度字段编码“ 011”不适用于主附件闪存共享（MAFS）中的闪存擦除。



Independent Flash SPI and eSPI Interface



Shared SPI and eSPI Interface

在为运行时flash共享启用flash访问通道之前，EC和BMC等eSPI从机必须能够与其他存储设备（例如其自己的ROM）执行的代码一起正常运行。

主附加闪存共享方案使用2个专用的eSPI命令操作码：GET\_FLASH\_NP和PUT\_FLASH\_C（表3）

通过eSPI接口对Flash组件进行的任何运行时访问都将通过eSPI主机，该主机随后将周期路由到Flash Access模块，然后再将周期转发到SPI Flash控制器。 SPI闪存控制器将代表eSPI从机执行对flash的访问。

eSPI从机在Flash Access事务中使用的Flash访问地址是物理Flash线性地址。 物理地址覆盖了整个Flash寻址空间。但是，SPI Flash控制器可能会对eSPI从机启动的Flash Access事务施加访问权限控制。

未经访问权限访问Flash区域的任何尝试均被视为错误。需要SPI Flash控制器进行检查，并将完成失败的信息返回eSPI从机。

Flash Read和Flash Write事务是non-posted的事务。每个事务将具有一个相应的完成状态，该完成状态指示请求的操作的状态，并在周期为快速读取时指示数据。完成状态将传送回eSPI从机。

通道功能和配置寄存器中的Flash访问通道最大读取请求大小参数已定义为允许eSPI主机将Flash读取请求大小,限制为eSPI主机支持的大小。

同样，通道功能和配置寄存器中的flash访问通道最大有效负载大小参数已定义为允许eSPI主机限制flash写入数据有效负载大小。

Flash Erase是non-posted的请求，没有数据。此命令指示SPI Flash控制器擦除分配给eSPI从机的部分区域。地址字段指定擦除块的开头，长度字段的最低3位指定要擦除的块的大小。长度字段的最低3位的编码与通道功能和配置寄存器的“闪存块擦除大小”字段的值匹配。指定的地址必须与块擦除大小对齐。支持的擦除块大小是可编程的，并由eSPI主机通过通道功能和配置寄存器传递给从机。

eSPI主机会将事务直接转发到flash控制器。然后，flash控制器将执行必要的检查，以确保支持该周期，然后再将其发送到flash。如果由于无效的寻址模式（32位与24/26位寻址），不支持的命令，不支持的块擦除大小或任何其他原因而不支持该周期，则flash控制器将为eSPI主机合成不成功的完成信息，这将导致然后将完成信息通过eSPI总线转发到eSPI从机，而无需发送指令到flash。

### Slave buffer management

eSPI协议定义了简化的缓冲区管理机制。 eSPI从机通过响应阶段的“状态”字段传达用于传输的新事务的可用性和用于存储传入事务的接收缓冲区的可用性

eSPI主机不需要将队列信息传递给eSPI从机。 eSPI主机将等待直到其相关的内部队列可用，然后再处理从机的请求。如果命令规则允许，eSPI主机可以选择服务另一个具有可用队列资源的请求，同时等待相关的队列资源释放。

eSPI从机负责确保在将请求提交给eSPI主机之前进行命令。 eSPI主机在满足其他未决请求的命令要求之前，不应通过“status”字段看到该请求。例如，如果在posted的写入之后对non-posted的传输队列顶部的non-posted的读取进行了排序，则除非所有已发布的写入位于状态寄存器之前，否则non-posted的读取不应将状态寄存器中的NP\_AVAIL位置1。

仅当eSPI从机接收器缓冲区可以接受至少一个最大有效载荷大小的数据包时，才能设置相应的空闲指示。作为响应阶段的一部分返回的状态字段中的空闲指示必须包含当前事务消耗的缓冲区大小。例如，如果eSPI主机启动了发布写操作，而该写操作耗尽了eSPI从属接收器的发布/完成队列，则必须在相应命令的响应阶段的状态字段中清除PC\_FREE指示。

当eSPI主机发出GET\_ \*命令时，“响应阶段”中的“状态”字段必须反映与GET\_ \*命令关联的缓冲区的下一个状态。 例如，如果eSPI主机发出GET\_PC且PC\_AVAIL状态位在该命令的“响应阶段”期间被置位，则表明此命令之后还有另一个已发布或完成的事务。 如果PC\_AVAIL状态位被清除，则表明在报告时此命令之后没有其他可用的posted或完成事务。

\_AVAIL或\_FREE一经声明，就必须继续由eSPI从机提交，直到主机对相关从机的缓冲区采取相应的操作为止。例如，曾经由从机声明的PC\_AVAIL仅受主机的GET\_PC命令影响，曾经由从机声明的PC\_FREE仅受主机的PUT\_PC命令影响，依此类推。声明后，由于其他不相关的事件，从机不允许更改\_AVAIL或\_FREE指示。



eSPI Slave Buffer Design (Conceptual)

### Transaction ordering rule

此处指定的命令规则适用于同一通道内的事务，并共享同一CS＃引脚。 在不同渠道上的传输之间没有命令要求。在具有相同通道号但涉及使用不同CS＃引脚的eSPI从器件的传输之间没有命令要求。

|  |  |  |
| --- | --- | --- |
| **Row pass Column?** | **Posted Request or Completion** | **Non-Posted Request** |
| **Posted Request or Completion** | No1 | Yes3 |
| **Non-Posted Request** | No2 | No4 |

NO-表示不允许在后续事务之前完成后续事务，以保留系统中的顺序。

YES-表示必须允许后续事务在前一个事务或死锁发生之前完成

note：

1.posted的请求不能通过posted的请求，以确保最新的数据写在最后。 完成必须将发布的写数据拉回原始总线，以避免陈旧的数据。

2. non-posted的请求必须推送posted的写数据，以避免读取过时的数据。

3.必须允许已发布的请求或完成内容传递non-posted的请求，以避免死锁。

4.non-posted的请求不需要彼此通过。

事务命令规则要求eSPI主机和eSPI从机为它们发起的每个non-posted的事务预分配完成缓冲区。这样可以确保返回的完成不会阻止任何posted交易的运行情况。

为避免可能的死锁，发送（Tx）和接收（Rx）之间必须没有进出依存关系，特别是eSPI代理不能根据相应Tx队列的前移进度释放通道的Rx队列。 这适用于外设通道，虚拟有线通道，OOB通道和闪存访问通道的所有对应的Tx / Rx对。

必须以请求顺序返回与具有相同标签的相同通道对应的所有完成。不需要从相同的通道完成操作，而是按请求顺序返回不同的标签。 不需要以请求顺序返回来自不同渠道的完成。

### Zero length read or write

不支持零长度内存，I / O和Flash读写。

## 链路层(Link Layer)

### 1,2,4 I/O模式

无论主机还是从机都必须支持单 IO模式，我们也可以通过配置寄存器设置为单，双，或者四I/O模式。eSPI从机可以独立支持双I / O和四I / O模式。

默认情况下，来自eSPI Reset＃的主机和从机都在Single I / O模式下运行。 主机可以使用SET\_CONFIGURATION命令更改操作模式。SET\_CONFIGURATION以当前的操作模式完成。新的操作模式仅在CS＃的取消激活边缘生效。

在单I / O模式下，I / O [1：0]引脚是单向的。 eSPI主机在命令阶段驱动I / O [0]，而从机的响应在I / O [1]上驱动。 在命令阶段，要求eSPI从机将I / O [1]置于高阻态，因为I / O [1]可以由eSPI主机驱动，例如在启动带内复位命令时。

在双I / O模式下，I / O [1：0]引脚变为双向以形成双向数据总线，并且所有命令和响应阶段都同时在两个双向引脚上传输，从而有效地使了 单I / O模式的传输速率

在四线I / O模式下，I / O [3：0]引脚是双向数据总线，并且所有命令和响应阶段都同时通过四个双向引脚进行传输，相对于双I / O模式，从而有效地使传输速率提高了一倍。

eSPI事务的每个字段都按照定义的顺序相应移出。 对于包含多个字节的字段，在eSPI总线上移出的字节顺序如下：（LSB =最低有效字节，MSB =最高有效字节）

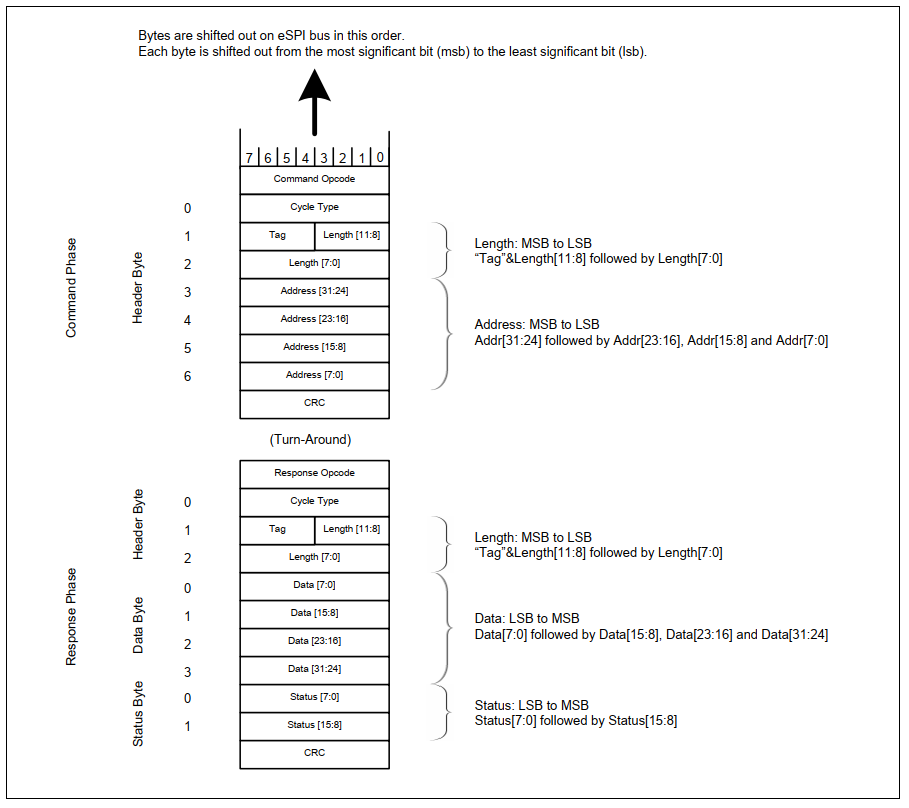
•：Header:

。长度：从MSB（带标签字段）到LSB

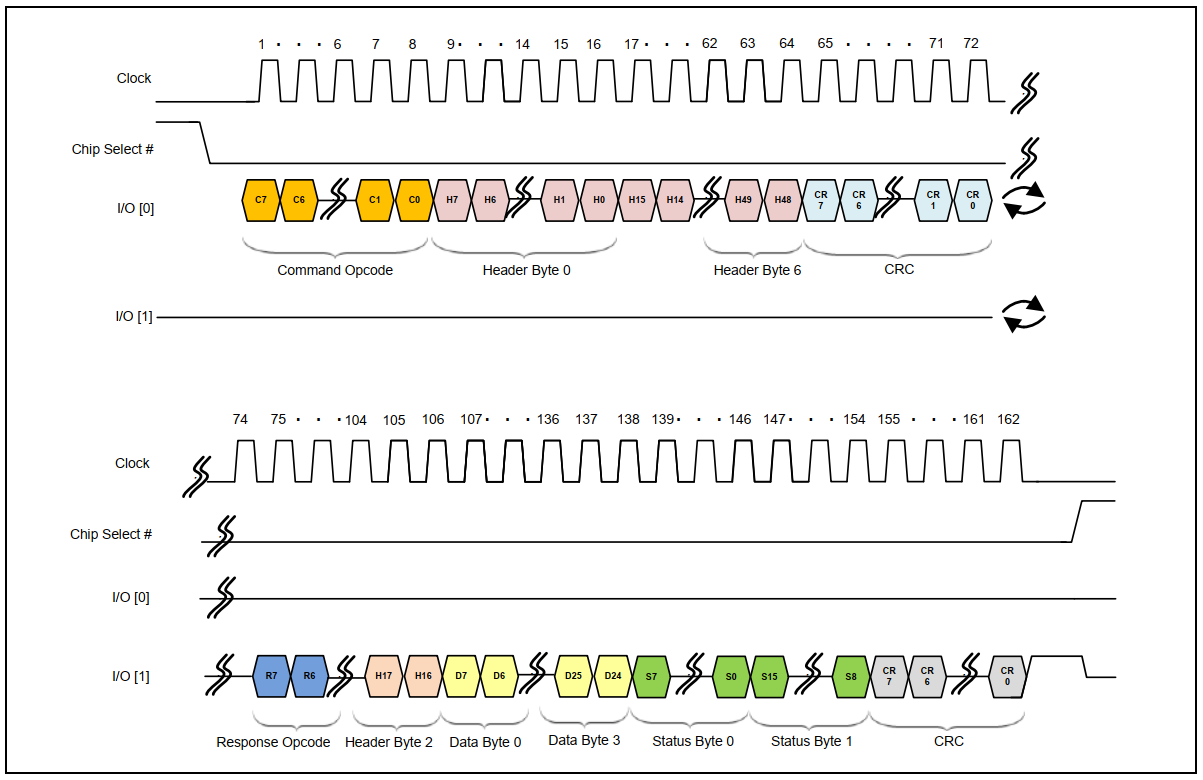
。地址：从MSB到LSB。 这适用于地址为GET\_CONFIGURATION和SET\_CONFIGURATION的eSPI事务。

* Data：从LSB到MSB
* Status从LSB到MSB

其中每个字节都从最高有效位（位[7]）移到最低有效位（位[0]）



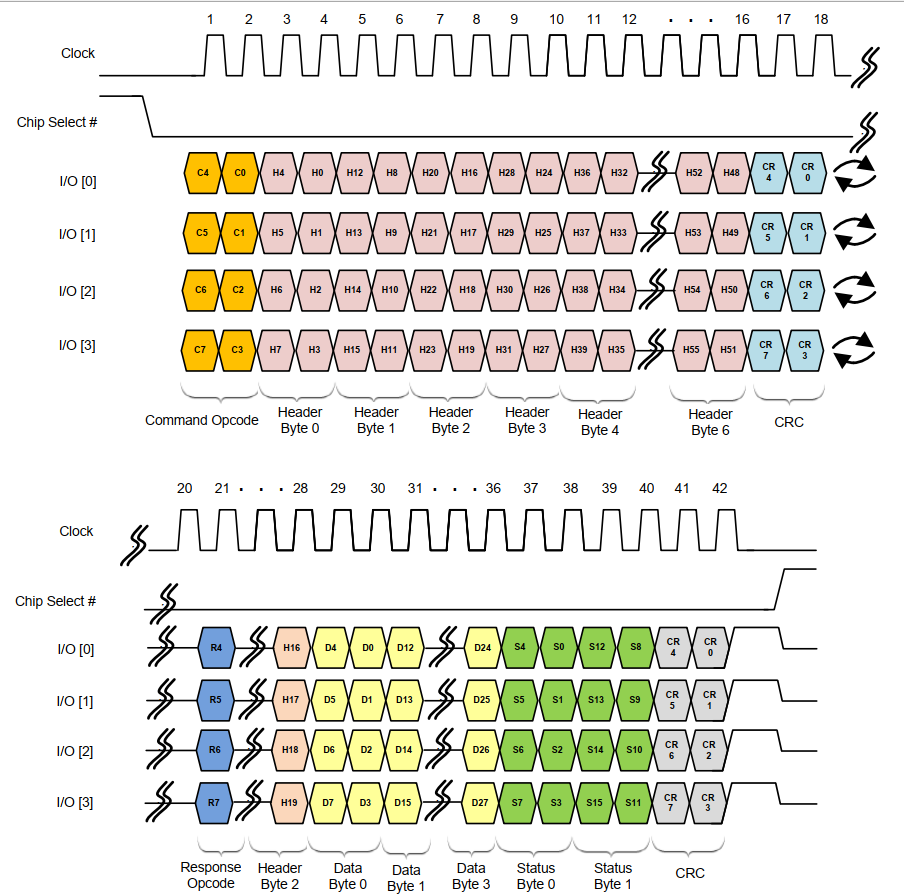
Byte Ordering on the eSPI Bus



Single I/O Mode



Dual I/O Mode



Quad I/O Mode

### CRC

CRC-8用于保护eSPI事务数据包。 命令阶段和响应阶段包含各自的CRC字节。 对于命令阶段，CRC计算包括命令阶段中的所有字节，例如命令操作码，标头（如果存在）和数据（如果存在）。 对于响应阶段，CRC计算包括响应阶段中的所有字节，例如响应代码（不包括在CRC计算中的WAIT\_STATE响应代码），标头（如果存在），数据（如果存在）和状态。

CRC值使用以下规则计算：

•多项式表示为：x8 + x2 + x + 1。

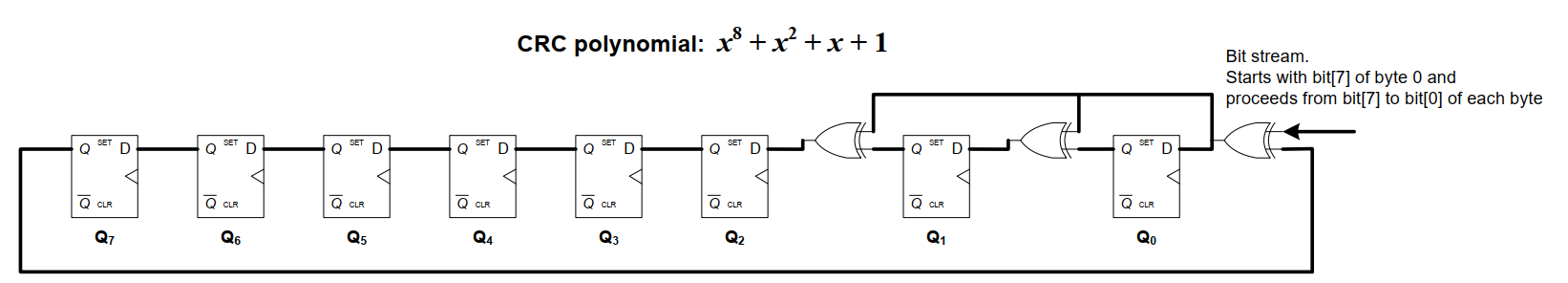
•用于CRC计算的多项式的系数表示为“ 07h”。

•种子值为“ 00h”。 在计算任何CRC字节之前，将CRC存储寄存器复位为初始值00h。

•CRC计算从字节0的位[7]开始，从每个字节的位[7]到位[0]

对于eSPI，必须强制生成CRC。 但是，默认情况下，在eSPI reset＃之后，CRC检查默认为禁用，并通过将CRC检查启用位设置为SET\_CONFIGURATION来启用，从而在成功设置SET\_CONFIGURATION时，将在CS＃的无效沿上在eSPI主机和eSPI从机上启用CRC检查。 eSPI主机和eSPI从机都必须始终能够支持CRC检查，因为平台要求确定是否将为eSPI接口启用CRC检查。禁用CRC检查时，接收器将忽略CRC字节

#### CRC Polynomial Representation



CRC Polynomial Representation

#### CRC Byte with Input Data D7:D0 (⊕ denotes logical XOR)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **1st Clock** | **2nd Clock** | **3rd Clock** | **4th Clock** |
| Q0 | D7 | D6 | D5 | D4 |
| Q1 | D7 | D7⊕D6 | D6⊕D5 | D5⊕D4 |
| Q2 | D7 | D7⊕D6 | D7⊕D6⊕D5 | D6⊕D5⊕D4 |
| Q3 | 0 | D7 | D7⊕D6 | D7⊕D6⊕D5 |
| Q4 | 0 | 0 | D7 | D7⊕D6 |
| Q5 | 0 | 0 | 0 | D7 |
| Q6 | 0 | 0 | 0 | 0 |
| Q7 | 0 | 0 | 0 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **5th Clock** | **6th Clock** | **7th Clock** | **8th Clock. CRC = Q7:0** |
| Q0 | D3 | D2 | D7⊕D1 | D7⊕D6⊕D0 |
| Q1 | D4⊕D3 | D3⊕D2 | D7⊕D2⊕D1 | D6⊕D1⊕D0 |
| Q2 | D5⊕D4⊕D3 | D4⊕D3⊕D2 | D7⊕D3⊕D2⊕D1 | D6⊕D2⊕D1⊕D0 |
| Q3 | D6⊕D5⊕D4 | D5⊕D4⊕D3 | D4⊕D3⊕D2 | D7⊕D3⊕D2⊕D1 |
| Q4 | D7⊕D6⊕D5 | D6⊕D5⊕D4 | D5⊕D4⊕D3 | D4⊕D3⊕D2 |
| Q5 | D7⊕D6 | D7⊕D6⊕D5 | D6⊕D5⊕D4 | D5⊕D4⊕D3 |
| Q6 | D7 | D7⊕D6 | D7⊕D6⊕D5 | D6⊕D5⊕D4 |
| Q7 | 0 | D7 | D7⊕D6 | D7⊕D6⊕D5 |

**注: 具体的计算方法可参考附录二。**

## 从机寄存器配置

增强型串行外设接口（eSPI）定义了一组从寄存器。 这些寄存器是枚举，配置和为eSPI总线定义的各个独立通道的正常运行所必需的。

#### 状态寄存器

状态寄存器位通过eSPI reset＃复位。状态寄存器的在响应阶段的相应状态字段中返回。

关于状态寄存器的编码在之前介绍过，在3.3.5 响应域中的状态域。

#### 功能和配置寄存器

功能和配置寄存器位通过eSPI Reset＃重置。 此外，可以通过相应寄存器部分中所述的其他复位来复位寄存器。

GET\_CONFIGURATION和SET\_CONFIGURATION命令用于访问这些寄存器。 只能以DWord宽度访问寄存器。当使用SET\_CONFIGURATION命令配置寄存器时，从机的新寄存器值仅在片选＃的置低沿生效。 因此，SET\_CONFIGURATION命令基于当前的预配置设置在eSPI总线上运行。

从偏移量800h到FFFh的寄存器保留为特定于平台的。 这为平台特定的应用程序提供了2KB的寄存器空间。

**Table 20: Slave Registers**

|  |  |  |
| --- | --- | --- |
| **Start (Hex)** | **End (Hex)** | **Register Name** |
| 000 | 003 | Reserved |
| 004 | 007 | Device Identification |
| 008 | 00B | General Capabilities and Configurations |
| 00C | 00F | Reserved |
| 010 | 013 | Channel 0 Capabilities and Configurations |
| 014 | 01F | Reserved |
| 020 | 023 | Channel 1 Capabilities and Configurations |
| 024 | 02F | Reserved |
| 030 | 033 | Channel 2 Capabilities and Configurations |
| 034 | 03F | Reserved |
| 040 | 043 | Channel 3 Capabilities and Configurations |
| 044 | 7FF | Reserved |
| 800 | FFF | Platform Specific registers |

##### Device Identification

|  |  |  |  |
| --- | --- | --- | --- |
| **Bit** | **Type** | **Default** | **Description** |
| 31:8 | RO | 0 | **Reserved.** |
| 7:0 | RO | 01h | **Version ID：**固定为0x01 |

##### General Capabilities and Configurations

|  |  |  |  |
| --- | --- | --- | --- |
| Bit | Type | Default | Description |
| 31 | RW | 0b | 启用CRC校验：eSPI主机将该位置1，以启用eSPI总线上的CRC校验。默认情况下，CRC校验被禁用。  0 :禁能CRC校验 1：使能CRC校验 |
| 30 | RW | 0b | 响应修饰符使能：此位设置为1，以允许eSPI从机使用响应修饰符来附加外设（通道0）完成，虚拟线路（通道1）数据包或闪存访问（通道3）完成 到GET\_STATUS响应阶段。 当该位为0时，eSPI从机只能使用响应修饰符“ 00”，即无附加。 默认情况下，“响应修饰符”是禁用的。 |
| 29 | RO | 0 | Reserved |
| 28 | RW | 0b | Alert模式：该位用于配置从机使用的警报机制，以在eSPI接口上启动事务。  0b：I / O [1]引脚用于指示警报事件。  1b：Alert＃引脚用于发信号通知Alert事件。  Note：在单个主单从拓扑中，该位可设置为0或者1。 对于单主多从拓扑，该位必须设置为“ 1”。  在运行时，在此情况下提供的单个或多个从属拓扑中，警报模式都可以从默认的“ 0”更改为“ 1”，仅启用了一个从属即可生成Alert＃事件。 |
| 27:26 | RW | 00b | I / O模式选择：eSPI主机对该字段进行设置以启用适当的操作模式，该模式将在CS＃的取消置位沿生效。  主机和从机都必须支持在此字段中配置的I / O模式。 默认情况下支持单I / O模式。   |  |  | | --- | --- | | Encoding | Mode | | 00 | Single I/O | | 01 | Dual I/O | | 10 | Quad I/O | | 11 | Reserved. | |
| 25:24 | RO | HwInit | I / O模式支持：此字段指示从机支持的I / O模式。   |  |  | | --- | --- | | Encoding | Mode | | 00 | Single I/O | | 01 | Single I/O and Dual I/O | | 10 | Single I/O and Quad I/O | | 11 | Single ,Dual ,and Quad I/O | |
| 23 | RW | 0 | 漏极开路Alert＃选择：eSPI主机将此位设置为“ 1”，以将Alert＃引脚配置为漏极开路输出。  默认情况下，Alert＃引脚用作驱动输出。 如果从机支持漏极开路Alert＃引脚，则只能将此位编程为1。  当警报模式位为1时，该位必须有效，表示警报＃引脚用于发信号通知警报事件。  0b：Alert＃引脚为驱动输出。  1b：Alert＃引脚为漏极开路输出。 |
| 22:20 | RW | 000b | 工作频率设置   |  |  | | --- | --- | | Bits | Frequency(MHz) | | 000 | 20 | | 001 | 25 | | 010 | 33 | | 011 | 50 | | 100 | 66 | | others | Reserved | |
| 19 | RO | HwInit | 支持漏极开路Alert＃:硬件是否支持开漏输出。  0： 不支持 1： 支持 |
| 18:16 | RO | HwInit | 获取从机支持的最大操作频率   |  |  | | --- | --- | | Bits | Frequency(MHz) | | 000 | 20 | | 001 | 25 | | 010 | 33 | | 011 | 50 | | 100 | 66 | | others | Reserved | |
| 15:12 | RW | 0 | 允许的最大等待状态：eSPI主机设置从机必须允许从机响应的最大等待状态，然后从机必须使用接受，延迟，非致命错误或致命错误响应代码进行响应。  这是字节时间粒度中从1开始的字段。 当为“ 0”时，表示16字节时间。  一个字节时间对应于Single I / O模式下的8个串行时钟，Dual I / O模式下的4个串行时钟或Quad I / O模式下的2个串行时钟。 |
| 11:8 | RO | 0 | Reserved |
| 7:0 | RO | HwInit | 支持的通道：置位时，每个位指示从机支持相应的通道   |  |  | | --- | --- | | Bits | channel | | 0 | Peripheral Channel | | 1 | Virtual Wire Channel | | 2 | OOB Message Channel | | 3 | Flash Access Channel | | 4:7 | Reserved for platform specific channels | |

**HwInit:表示由硬件功能决定。**

##### Channel 0 Capabilities and Configurations

|  |  |  |  |
| --- | --- | --- | --- |
| Bit | Type | Default | Description |
| 31:15 | RO | 0 | Reserved |
| 14:12 | RW | 001b | 外围通道最大读取请求大小：eSPI主机设置外围通道的最大读取请求大小   |  |  | | --- | --- | | Bit | 最大读取请求大小 | | 000b | reserved | | 001b | 64 bytes address aligned max read request size | | 010b | 128 bytes address aligned max read request size | | 011b | 256 bytes address aligned max read request size | | 100b | 512 bytes address aligned max read request size | | 101b | 1024 bytes address aligned max read request size | | 110b | 2048 bytes address aligned max read request size | | 111b | 4096 bytes address aligned max read request size | |
| 11 | RO | 0 | Reserved |
| 10:8 | RW | 001b | 已选择：eSPI主机设置外设通道的最大有效负载大小。   |  |  | | --- | --- | | Bit | 最大有效负载大小 | | 000b | resvered | | 001b | 64 bytes address aligned max payload size | | 010b | 128 bytes address aligned max payload size | | 011b | 256 bytes address aligned max payload size | | 100b-111b | resvered | |
| 7 | RO | 0 | Reserved |
| 6:4 | RO | HwInit | 从机支持最大负载大小   |  |  | | --- | --- | | Bit | 最大有效负载大小 | | 000b | resvered | | 001b | 64 bytes address aligned max payload size | | 010b | 128 bytes address aligned max payload size | | 011b | 256 bytes address aligned max payload size | | 100b-111b | resvered | |
| 3 | RO | 0 | Reserved |
| 2 | RW | 0b | 总线主机使能：当该位为0时，它将禁止从机在外设通道上生成总线主机周期。 当该位为1时，它允许从机在外设通道上生成总线主控周期。  在将总线主控使能位从“ 1”清除为“ 0”之前，从机不能有未完成的non-posted周期。 |
| 1 | RO | 0b | 外围通道就绪：当该位为1时，表明从机已准备好接受外围通道上的事务。  启用通道后，eSPI主机应轮询此位，然后在此通道上运行从机的任何事务。  0b: Channel is not ready 1b: Channel is ready |
| 0 | RW | 1b | 外设通道使能：一般情况下，reset#之后默认使能。  将此位从1清除为0，将促发该通道复位。 |

##### Channel 1 Capabilities and Configurations

|  |  |  |  |
| --- | --- | --- | --- |
| Bit | Type | Default | Description |
| 31:22 | RO | 0 | Reserved |
| 21:16 | RW | 0 | Virtual Wire最大操作数：单个Virtual Wire数据包中可以发送的Virtual Wire组的最大数量。  这是从0开始的计数。 默认值0表示计数为1。 |
| 15:14 | RO | 0 | Reserved |
| 13:8 | RO | HwInit | 支持的最大Virtual Wire数：此字段发布从机支持的最大Virtual Wire数。  eSPI从机必须在此字段中发布“ 000111b”或更大的值，以指示在单个虚拟线数据包中通信的至少8个虚拟线组的支持 |
| 7:2 | RO | 0 | Reserved |
| 1 | RO | 0 | 虚拟有线通道就绪：当该位为1时，表示从机已准备好接受虚拟有线通道上的事务。  启用通道后，eSPI主机应轮询此位，然后在此通道上运行从机的任何事务。  0b: Channel is not ready 1b: Channel is ready |
| 0 | RW | 0 | 虚拟线通道使能：eSPI主机将此位设置为“ 1”以启用虚拟线通道。  将此位从“ 1”清除为“ 0”不会重置虚拟线通道，因此必须继续在内部保持所有虚拟线的状态。 当该位为“ 0”时，在虚拟线路通道上不得进行任何传输。  该通道在复位之后默认禁能。 |

##### Channel 2 Capabilities and Configurations

|  |  |  |  |
| --- | --- | --- | --- |
| Bit | Type | Default | Description |
| 31:11 | RO | 0 | Reserved |
| 10:8 | RW | 001b | eSPI主机设置OOB消息通道的最大有效负载大小。  最大有效负载大小适用于OOB数据包中嵌入的协议的实际有效负载。 有关OOB消息有效负载的详细信息，请参见5.2.3节。   |  |  | | --- | --- | | Bit | 最大有效负载大小 | | 000b | resvered | | 001b | 64 bytes address aligned max payload size | | 010b | 128 bytes address aligned max payload size | | 011b | 256 bytes address aligned max payload size | | 100b-111b | resvered | |
| 7 | RO | 0b | Reserved |
| 6:4 | RO | HwInit | 从机支持最大负载大小   |  |  | | --- | --- | | Bit | 最大有效负载大小 | | 000b | resvered | | 001b | 64 bytes address aligned max payload size | | 010b | 128 bytes address aligned max payload size | | 011b | 256 bytes address aligned max payload size | | 100b-111b | resvered | |
| 3:2 | RO | 0 | Reserved |
| 1 | RO | 0b | OOB消息通道就绪标志位：当该位为1时，表示从机已准备好接受OOB 消息通道上的事务。  0b: Channel is not ready 1b: Channel is ready |
| 0 | RW | 0b | OOB消息通道使能：eSPI主机将该位置1，以启用OOB消息通道。  将此位从“ 1”清除为“ 0”会触发对OOB消息通道的重置。  该通道在复位之后默认禁能。 |

##### Channel 3 Capabilities and Configurations

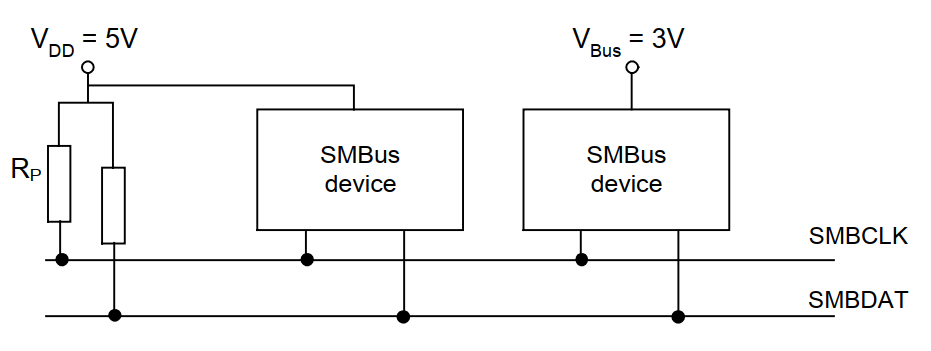
|  |  |  |  |
| --- | --- | --- | --- |
| Bit | Type | Default | Description |
| 31:15 | RO | 0 | Reserved |
| 14:12 | RW | 001b | eSPI主机设置Flash Access通道的最大读取请求大小。   |  |  | | --- | --- | | Bit | 最大读取请求大小 | | 000b | reserved | | 001b | 64 bytes max read request size | | 010b | 128 bytes max read request size | | 011b | 256 bytes max read request size | | 100b | 512 bytes max read request size | | 101b | 1024 bytes max read request size | | 110b | 2048 bytes max read request size | | 111b | 4096 bytes max read request size | |
| 11 | RO | 0b | Flash共享模式：  0b: Master attached flash sharing.  1b: Slave attached flash sharing |
| 10:8 | RW | 001b | eSPI主机设置Flash 访问通道的最大有效负载大小   |  |  | | --- | --- | | Bit | 最大有效负载大小 | | 000b | resvered | | 001b | 64 bytes address aligned max payload size | | 010b | 128 bytes address aligned max payload size | | 011b | 256 bytes address aligned max payload size | | 100b-111b | resvered | |
| 7:5 | RO | HwInit | 从机支持最大负载大小   |  |  | | --- | --- | | Bit | 最大有效负载大小 | | 000b | resvered | | 001b | 64 bytes address aligned max payload size | | 010b | 128 bytes address aligned max payload size | | 011b | 256 bytes address aligned max payload size | | 100b-111b | resvered | |
| 4:2 | RW | 01b | 闪存块擦除大小：eSPI主机设置此字段，以将块擦除大小传达给从机。  该字段仅适用于主连接的flash共享方案。   |  |  | | --- | --- | | Bit | 大小 | | 000b | Reserved | | 001b | 4 Kbytes | | 010b | 64 Kbytes | | 011b | Both 4 Kbytes and 64 Kbytes are supported | | 100b | 128 Kbytes | | 101b | 256 Kbytes | | 110-111b | Reserved | |
| 1 | RO | 0b | Flash 访问通道就绪标志位：当该位为1时，表示从机已准备好接受flash 访问通道上的事务。  0b: Channel is not ready 1b: Channel is ready |
| 0 | RW | 0b | Flash 访问通道使能：eSPI主机将该位置1，以启用flash访问通道。  将此位从“ 1”清除为“ 0”会触发对Flash 访问通道的重置。  该通道在复位之后默认禁能。 |

# SMBus 总线(2.0)

系统管理总线（SMBus）是两线接口，各种系统组件芯片可通过该接口相互通信，并与系统的其余部分通信。它基于I2C 的操作原理。

SMBus提供用于系统和电源管理相关任务的控制总线。 系统可以使用SMBus将消息传递到设备或从设备传递消息，而不用使单个控制线跳闸。 卸下各个控制线可减少引脚数。 接受消息可确保将来的可扩展性

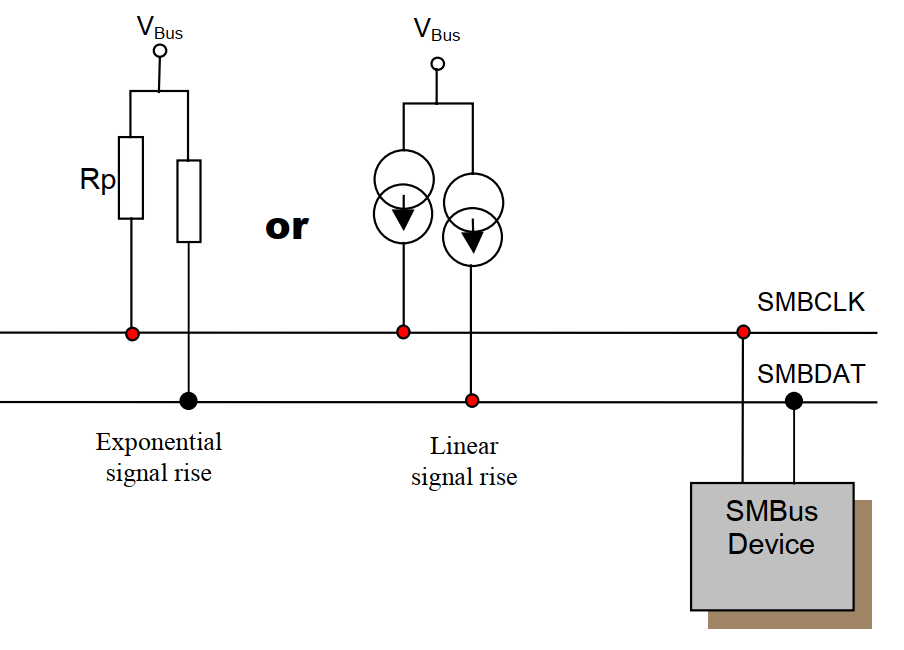
使用系统管理总线，设备可以提供制造商信息，告诉系统其型号/零件号是什么，保存其状态以暂停事件，报告不同类型的错误，接受控制参数并返回其状态。



SMBus Topology

VDD可能为3至5V+/- 10％，并且可能有SMBus器件直接由总线VDD供电。 SMBCLK和SMBDAT线都是双向的，通过上拉电阻或电流源或其他类似电路连接到正电源电压。 总线空闲时，两条线都高。

总线可以通过上拉电阻或电流源拉高。在涉及较高总线电容的情况下，可以使用更复杂的电路，该电路可以限制下拉电流，同时在从低到高的过渡期间提供足够的电流，以保持SMBus的上升时间规格。



SMBus pull-up circuitry

## 物理连接

SMBus 与I2C连接一致，所以采用的两线协议，分别为时钟线与数据线，数据流双向。

## 电气属性

SMBus 与I2C连接一致，所以采用的两线协议，都必须有上拉电阻。

## 物理层

略，这个部分详细介绍SMBUS的一些硬件部分的时序时钟相关内容。

## 数据链路层

SMBus 与I2C相同

在数据链路层中，数据必须在时钟高电平期间间保持稳定；

当时钟为高电平期间，数据线从高电平到低电平变化过程就是数据发送的开始信号；

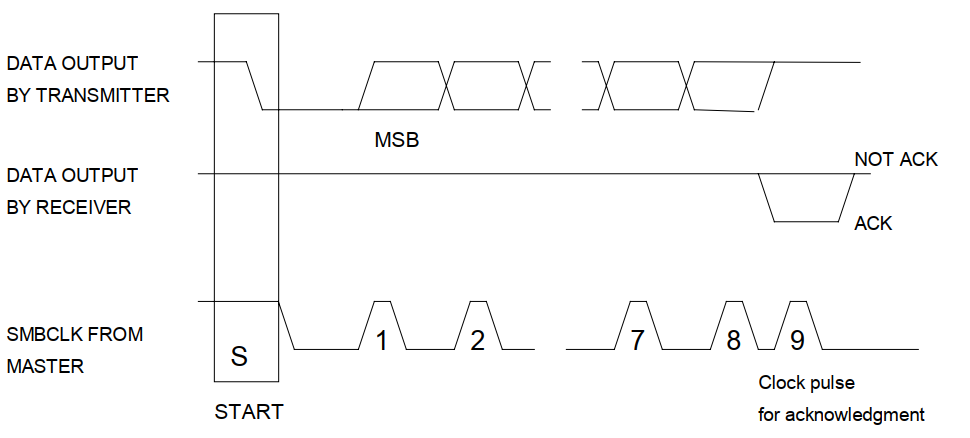
当时钟为高电平期间，数据线从低电平到高电平变化过程就是数据发送的停止信号；

当时钟线与信号线都为高电平的时候，表示总线空闲状态。

当时钟为高电平期间，主机释放数据线，从从机将数据线拉低，表示ACK，反之亦然。

每个字节由8位组成。 总线上传输的每个字节都必须跟随一个应答位。 字节先传送最高有效位（MSB）。

下图显示了相对于其他数据的确认（ACK）和未确认（NACK）脉冲的位置



ACK and NACK signaling of SMBus

SMBus设备必须始终确认（ACK）它自己的地址。SMBus使用此信号来检测总线上是否存在可拆卸设备。

在以下情况下，SMBus从设备可能会决定对地址字节以外的其他字节进行NACK：

•从设备正忙于执行实时任务，或者请求的数据不可用。主机在检测到NACK条件时必须产生STOP条件以中止传输。 请注意，作为替代方案，从设备可以在本规范的范围内延长时钟低电平周期，以完成其任务并继续传输。

•从站设备检测到无效命令或无效数据。在这种情况下，从设备必须对收到的字节进行NACK。 主机在检测到此条件后必须生成一个STOP条件，然后重试该事务。

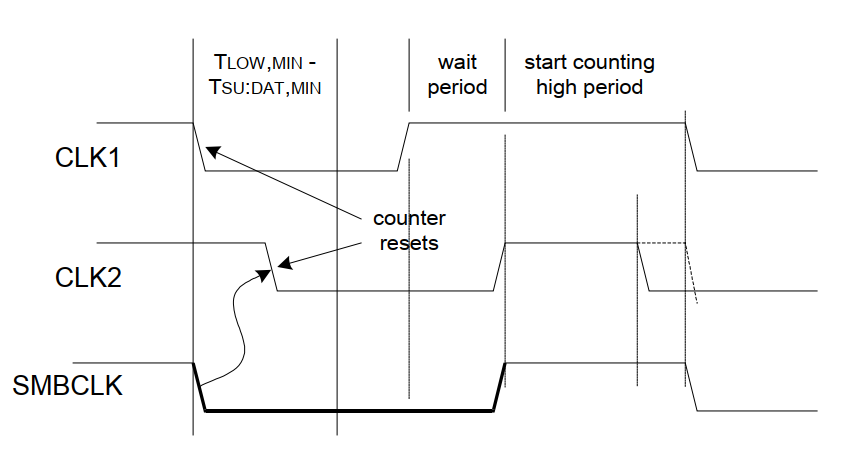
•如果事务中涉及到主接收器，则必须通过在从机时钟输出的最后一个字节上产生一个NACK，向从机发送器发送数据结束信号。 从机发送器必须释放数据线以允许主机产生停止条件。

### 总线仲裁

SMbus总线上的仲裁分两部分：SCL线的同步和SDA线的仲裁。

#### 时钟同步

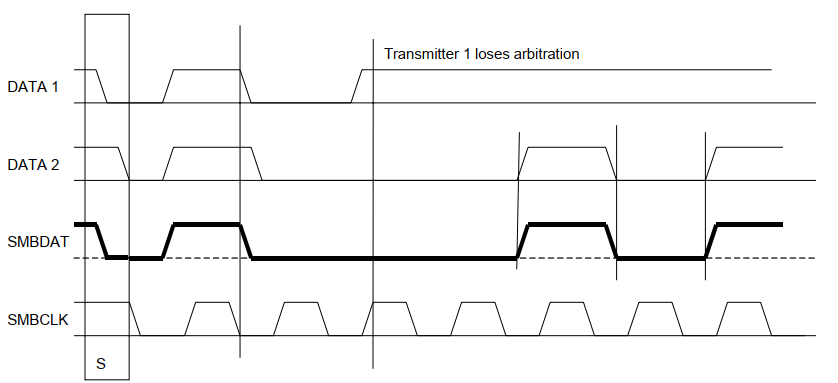
SCL同步是由于总线具有线“与”的逻辑功能，即只要有一个节点发送低电平时，总线上就表现为低电平。当所有的节点都发送高电平时，总线才能表现为高电平。正是由于线“与”逻辑功能的原理，当多个节点同时发送时钟信号时，在总线上表现的是统一的时钟信号。这就是SCL的同步原理



SMBus clock synchronization

#### **SDA仲裁**

SDA线的仲裁也是建立在总线具有线“与”逻辑功能的原理上的。节点在发送1位数据后，比较总线上所呈现的数据与自己发送的是否一致。是，继续发送；否则，退出竞争。SDA线的仲裁可以保证I2C总线系统在多个主节点同时企图控制总线时通信正常进行并且数据不丢失。总线系统通过仲裁只允许一个主节点可以继续占据总线



SMBus arbitration

上图是以两个节点为例的仲裁过程。DATA1和DATA2分别是主节点向总线所发送的数据信号，SDA为总线上所呈现的数据信号，SCL是总线上所呈现的时钟信号。当主节点1、2同时发送起始信号时，两个主节点都发送了高电平信号。这时总线上呈现的信号为高电平，两个主节点都检测到总线上的信号与自己发送的信号相同，继续发送数据。第2个时钟周期，2个主节点都发送低电平信号，在总线上呈现的信号为低电平，仍继续发送数据。在第3个时钟周期，主节点1发送高电平信号，而主节点2发送低电平信号。根据总线的线“与”的逻辑功能，总线上的信号为低电平，这时主节点1检测到总线上的数据和自己所发送的数据不一样，就断开数据的输出级，转为从机接收状态。这样主节点2就赢得了总线，而且数据没有丢失，即总线的数据与主节点2所发送的数据一样，而主节点1在转为从节点后继续接收数据，同样也没有丢掉SDA线上的数据。因此在仲裁过程中数据没有丢失。

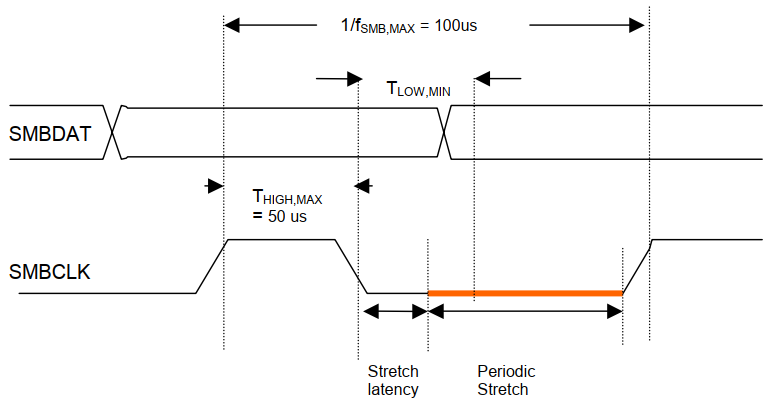
总结：SDA仲裁和SCL时钟同步处理过程没有先后关系，而是同时进行的。

#### Clock low extending

SMBus提供了一种时钟同步机制，以允许不同速度的设备共存于总线上。 除总线仲裁程序外，还可在位或字节传输期间使用时钟同步机制，以允许较慢的从属设备处理较快的主设备。

在位级别，设备可以通过定期延长时钟低电平间隔来减慢总线速度。

在一条消息的传输过程中，允许设备将时钟延长到本文档AC规范中描述的最大限制。 但是，设计为周期性延长每个时钟周期的设备必须保持10KHz的fSMB，MIN频率（fSMB，MIN-1 = 100µs），以保留SMBus带宽。

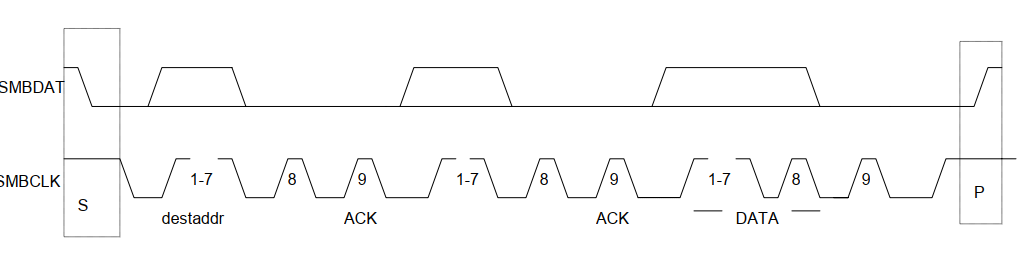


Periodic clock stretching by a slave SMBus device

#### Data transfer formats

在启动条件S之后，主机将要寻址的从设备的7位地址放在总线上。 地址是7位长，其后是指示数据传输方向（R / W＃）的第八位。 零表示传输（写），而零表示对数据的请求（读）。 数据传输总是由主机产生的STOP（P）条件终止。

特定的SMBus协议要求主机先产生一个重复的START条件，然后是从机地址，而不先产生STOP条件。



Data transfer over SMBus

## 网络层

系统管理总线规范涉及三种类型的设备。slaver是正在接收或响应命令的设备。master是发出命令，生成时钟并终止传输的设备。host是特殊的master，提供与系统CPU的主要接口。host必须是slave-master，并且必须支持SMBus主机通知协议。一个系统中最多可以有一个host。

可以对设备进行设计使其永远只是从机，当然也可以设置为只可能为主机的。设备在大多数情况下扮演的slave，但是也可能为master，系统host是设备的一个范例，该设备大多数情况下为主机，但是包含了从机的行为。

### 设备标识–从机地址

系统管理总线上作为从机存在的任何设备都有一个称为从机地址的唯一地址。 作为参考，以下地址是保留的，不得由任何SMBus从机使用或分配给任何SMBus从机。

|  |  |  |
| --- | --- | --- |
| **Slave Address Bits 7-1** | **R/W# bit Bit 0** | **Comment** |
| 0000 000 | 0 | General Call Address |
| 0000 000 | 1 | START byte |
| 0000 001 | X | CBUS address |
| 0000 010 | X | Address reserved for different bus format |
| 0000 011 | X | Reserved for future use |
| 0000 1XX | X | Reserved for future use |
| 0101 000 | X | Reserved for ACCESS.bus host |
| 0110 111 | X | Reserved for ACCESS.bus default address |
| 1111 0XX | X | 10-bit slave addressing |
| 1111 1XX | X | Reserved for future use |
| 0001 000 | X | SMBus Host |
| 0001 100 | X | SMBus Alert Response Address |
| 1100 001 | X | SMBus Device Default Address |

所有其他地址均可用于动态地址设备和/或其他设备的地址分配。 第5.2.1.4节讨论了其他设备地址

SMBus设备默认地址保留供SMBus地址解析协议使用，该协议允许动态分配地址。 详见5.6节

几个SMBus设备可以在实际系统中同时使用。在设备地址冲突的情况下，设计人员可以使用SMBus设备中实现的可编程功能来解决此类冲突，也可以使用同一系统中的多个SMBus分支来扩展使用同一地址的设备。

实际的SMBus系统中当前使用几种类型的地址。

#### Reserved addresses

这个类型的地址在上面表中已经说明，预留了一些功能地址。

#### Purpose-assigned addresses

这些地址由SMBus工作组分配给特定类型的设备。获得分配地址的每种设备类型都必须具有与其关联的SMBus规范。也就是说功能不同地址不同。

其他SMBus实施不仅仅依靠设备地址来标识设备的功能。 在这些应用程序中，设备的地址可能与目的分配的地址重叠。

#### Prototype Addresses

|  |  |
| --- | --- |
| **Slave Address** | **Description** |
| 1001 0XX | Prototype Addresses |

原型地址（1001 0XX）保留用于设计阶段的地址，不再生产阶段使用。

#### Miscellaneous device addresses

制造商已经生产并且可能会继续生产用于特定系统目的的SMBus兼容设备，他们不需要实现完整的SMBus规范，或者他们不需要操作系统的明确支持。 例如，此类设备可以是端口扩展器，D / A电路等。

#### Dynamically assigned addresses

SMBus 2.0版引入了动态分配地址的概念。 这将在第5.6节中详细介绍

### 设备的使用

典型的SMBus设备将具有一组命令，通过这些命令可以读取和写入数据。所有命令均为8位（1字节）长。命令参数和返回值的长度可以不同。访问不存在或不支持的命令可能会导致错误情况。根据该规范，最高有效位首先被传输。

对于任何给定的设备，有十一种可能的命令协议。 从设备可以使用十一个协议中的任何一个或全部来进行通信。 协议是快速命令，发送字节，接收字节，写字节，写字，读字节，读字，过程调用，块读取，块写入和块写块读取过程调用。

### 数据包错误检查

SMBus设备对数据包错误检查的实现对于SMBus设备是可选的，但对于参与该过程且仅在ARP处理期间的设备是必需的。实现数据包错误检查的SMBus设备必须能够与主机和其他未实现数据包错误检查机制的设备进行通信。

只要适用，就可以通过在每次消息传输结束时附加一个数据包错误代码（PEC）来实现数据包错误检查。每种协议（快速命令和后面部分中介绍的主机通知协议除外）都有两种变体：一种带有分组错误代码（PEC）字节，另一种没有。 PEC是一个CRC-8错误校验字节，根据所有消息字节（包括地址和读/写位）计算得出。 PEC由提供最后一个数据字节的设备附加到消息中。

#### ACK/NACK and Packet Error Checking

SMBus字节中的ACK / NACK位与SMBus数据包中的任何其他位一样容易受到破坏。因此，PEC末尾的ACK不能保证PEC正确。在写操作结束时接收到ACK的主发送器不一定要假定写操作已成功在写操作的从接收器中进行，尽管很有可能是这样。

由主发送器在PEC之后接收到的NACK表示从接收器的链路层已及时意识到传输中的错误，以在PEC字节的末尾提供NACK。这可能是由于错误的PEC或任何其他错误引起的。如果设备足够快地发现并在NACK到期时指示错误，则也可以用NACK指示在数据链路层上方发现的错误。

由主发送器在PEC之后接收到的ACK意味着从接收器中的链路层无法及时确定错误以提供NACK。这可能是因为接收器无法实时检查PEC的有效性。

如果主发送器要确保在目标设备上正确执行写操作，则它必须使用某种高层机制来验证这一点。这可以采用数据的PEC读取的形式。收到正确的PEC将可靠地表明原始写入已发生而没有错误。

当主接收器从发送器读取数据时，在事务结束时，由主接收器提供的ACK / NACK除了标记最后一个字节的末尾以外，没有其他意义。从属发送器正在提供数据，并且如果从属发送器提供的PEC是正确的，则主接收器可以假定数据是在从属发送器接收时接收到的。如果不是，则由主接收器采取任何适当的补救措施。

#### Master implementation

主机可以在任何传输中使用PEC。要求主机具有目标从站是否支持PEC或确定目标从站是否支持PEC的方法。PEC的使用受上层协议（例如设备驱动程序），规范（例如SMBus ARP协议的要求）或给定类别的设备（例如智能电池）的检测算法支配。

#### Slave implementation

必须准备实现包错误检查的从设备，以在有或没有PEC的情况下接收和发送数据。 在从设备接收传输期间，在设备识别出协议和命令之后，必须接受并检查附加PEC的有效性。

在从机发送传输期间，从机发送器必须在最后一个字节传输之后响应其他时钟，并向请求它的主接收器提供PEC。

每个总线事务都需要每个数据包内的发送方和接收方都进行数据包错误代码（PEC）计算。 PEC使用每个读或写总线事务的8位循环冗余校验（CRC-8）来计算分组错误码（PEC）。可以通过符合多项式C（x）= x8 + x2 + x1 + 1多项式表示的CRC-8的任何方式来计算PEC，并且必须按接收到的比特顺序进行计算。

根据5.5节中的协议，将PEC附加到消息中。PEC计算包括传输中的所有字节，包括地址，命令和数据。PEC计算不包括ACK，NACK，START，STOP或重复的START位。 这意味着PEC是从第一个START条件开始计算整个消息的。

### 总线协议

#### 快速命令

#### 发送一个字节

#### 接受一个字节

#### 写一个字节或者半字

#### 读一个字节或者半字

#### 流程调用

#### 块读写

#### 块写块读取过程调用

#### SMBus主机通知协议

### 地址解析协议（ARP）

#### 唯一设备标识符（UDID）

#### 上电复位

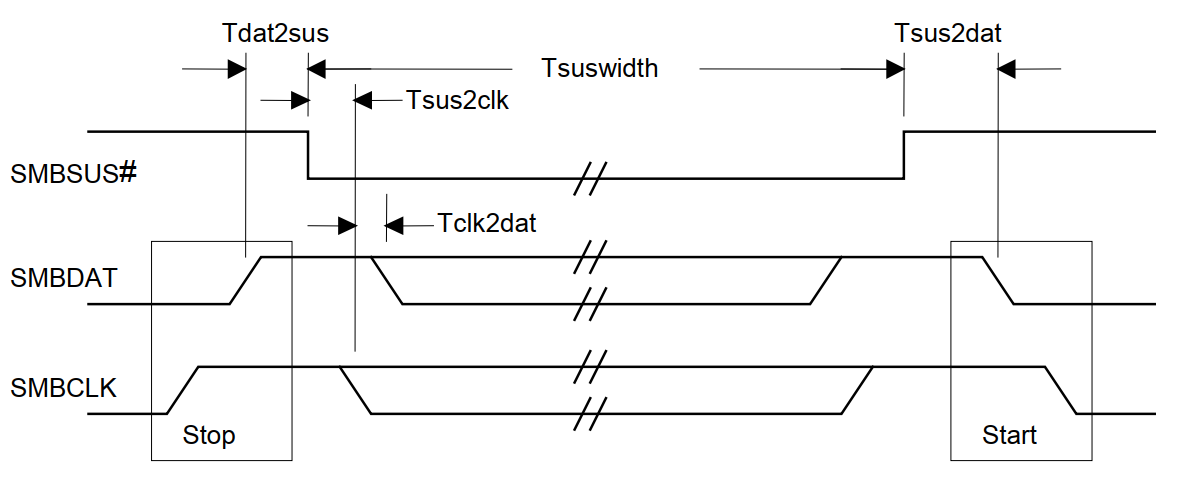
#### ARP 命令

## SMBus 可选信号

### SMBSUS#

系统进入挂起模式时，可选的第三信号SMBSUS＃变低。挂起模式是一种低功耗模式，在该模式下，大多数设备都处于停止或关机状态。恢复后，SMBSUS＃返回高电平。 然后，系统将所有设备返回到该状态。

系统可以使用SMBCLK和SMBDAT线来编程设备行为。在正常操作模式下，系统可能会向不同设备发出配置命令。这些命令可以告诉设备每当SMBSUS＃线变为活动状态时的行为。



SMBus During Suspend

|  |  |  |
| --- | --- | --- |
| Timing | Min | Typical |
| TDAT2SUS | 0ns | tens of ms |
| Tsus2clk | 0ns | tens of ns |
| Tclk2dat | 0ns | 0ns |
| Tsuswidth | minutes, hours, weeks |  |
| Tsus2dat | 0ns | hundreds of ms |

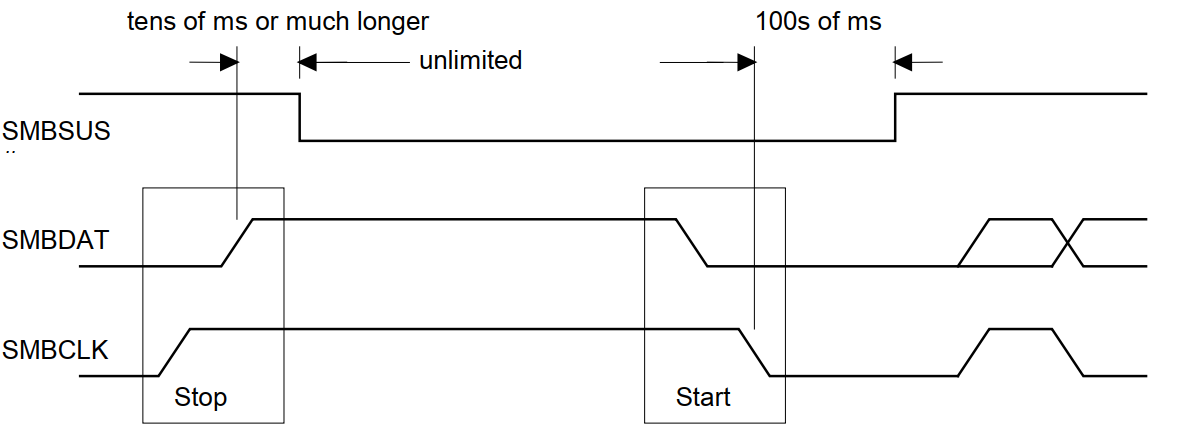
SMBus suspend parameters

SMBSUS＃不是线或信号。它是控制系统管理功能的设备的输出，是其他所有设备的输入。

挂起期间，除非使用SMBus从挂起模式恢复，否则系统管理总线上没有任何活动。 退出暂停状态后，活动恢复。

处于STOP条件后的任何时间，SMBSUS＃信号都可能变为低电平有效，表示系统正在进入挂起模式。这可以立即发生（最小值= 0ns），但可能需要更长的时间。实际上，最终的SMBus消息可能会在SMBSUS＃变低之前几分钟或几小时终止。挂起模式可能持续几秒钟，几分钟，几小时或几周。在系统管理总线可以发送另一条消息之前，系统必须退出挂起模式，该过程称为恢复。尽管SMBus可能在挂起期间处于唤醒状态，但是恢复过程可能仍然必须向系统管理总线供电。恢复过程可能需要很长时间，可能需要几百毫秒。 SMBCLK和SMBDAT上拉的仔细断电排序将防止器件错误地看到总线上的START条件。

如果在挂起期间为系统管理总线供电，则设备可以使用它来唤醒系统。主机或其他设备将监视总线上的START条件。该设备启动恢复序列。当系统退出挂起状态时，总线上的通信将恢复。



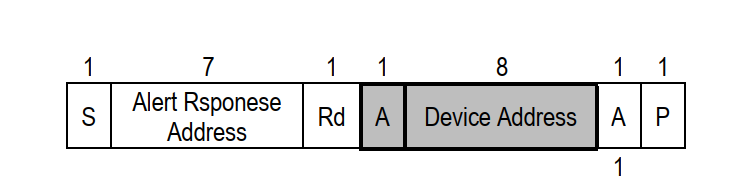
Using SMBus to Resume from Suspend

由于SMBSUS＃是可选信号，因此某些系统设备可能不知道系统是否处于挂起模式。 这样的设备可以假设，如果SMBCLK和SMBDAT线都为高电平，则表明总线处于活动状态。 该设备可能会尝试向另一个接受SMBSUS＃信号并因此处于睡眠状态的设备发送关键消息。 因此，重要的是，如果系统管理总线上驻留了不可暂停的主机，则系统能够在START条件下恢复，并且主机可以将关键消息发送到已挂起的设备。

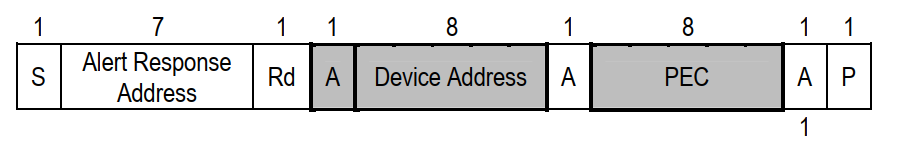
### SMBALERT#

另一个可选信号是设备的中断线，这些设备希望将其掌握能力换成引脚。 就像SMBCLK和SMBDAT信号一样，SMBALERT＃是线与信号。SMBALERT＃与SMBus通用呼叫地址一起使用。 用SMBus调用的消息长2个字节。

仅从属设备可以通过SMBALERT＃向主机发出要通话的信号。主机处理该中断，并同时通过警报响应地址（ARA）访问所有SMBALERT＃设备。只有将SMBALERT＃拉低的设备才会确认警报响应地址。主机执行修改后的接收字节操作。从发送设备提供的7位设备地址位于该字节的7个最高有效位中。 第八位可以是零或一。



7-bit-Addressable Device Responds to an ARA



7-bit-Addressable Device Responds to an ARA with PEC

如果有多个设备将SMBALERT＃拉低，则在从站地址传输期间，最高优先级（最低地址）的设备将通过标准仲裁赢得通信权。

确认从站地址后，设备必须脱离其SMBALERT＃下拉菜单。 如果主机在消息传输完成后仍看到SMBALERT＃为低电平，则表示它会再次读取ARA。

未实现SMBALERT＃信号的主机可能会定期访问ARA。

# 附录一：posted与non-posted

Post翻译过来就是邮寄的意思，当我们写了一封信然后丢进邮筒，那就要做好一去不复返遥遥无回应的准备。

按照这样理解的话，如果不需要对方响应的数据传输，就是Posted；如果需要对方立即或者稍后响应响应的就是Non-Post。Non-Posted总线事务是指PCI主机向PCI目标设备进行数据传送的时候，数据必须到达目的地之后，才能结束当前总线事务。PCI总线在没有结束当前总线事务时必须等待。

PCI总线规定Memory Read, I/O Read/Write, Config Read/Write都只能采用Non-Posted方式传输。

首先读操作肯定得是Non-Posted，读操作需要目标设备返回读回来的数据，如果在返回数据的过程中运行插入其他的操作，这样主机就很可能接收不到读回来的这笔数据了；其次，IO Write也是需要等待IO设备的响应的，不能允许中间插入其他操作，而Config Read/Write的本质其实是IO Read/Write。这样看来，将Memory Read, I/O Read/Write, Config Read/Write规定为Non-Posted传输就十分容易理解了。

Posted总线事务指PCI主机向PCI目标设备进行数据传输时，当数据到达PCI桥后，即由PCI桥接管来自上游总线的总线事务，并将其转发到下游总线。采用这种数据传送方式，在数据还没到达目标设备之前，PCI总线就可以结束当前总线事务，进行其他总线事务。很显然，采用Posted传送方式，这个Posted总线事务通过某条PCI总线后，就可以释放PCI总线的资源。PCI总线规定只有Memory Write可以采用Posted传送方式。

相对于Non-Posted传输来说，使用Posted方式的Memory Write显得更为高效些，主机在发起Memory Write传输后就可以进行下一笔数据传送，无须等待任何回应。这么做有点副作用，那就是无法确认写操作是否真的完成，但是和所提高的总线性能相比，Memory Write牺牲的这点命中率也不足道哉了。

# 附录二 CRC-8 计算方法

**模2除法**

模2除法与算术除法类似，但每一位除的结果不影响其它位，即不向上一位借位，所以实际上就是异或。在循环冗余校验码（CRC）的计算中有应用到模2除法。

例子：

使用多项式：x8+x5+x4+1（二进制为：100110001）

计算一个字节：0x11（二进制为：00010001）

计算步骤：

A、 因为采用顺序异或，所以需要计算的数据左移8位，

移位后数据为：0001 0001 0000 000

B、 先进行高9bit异或（多项式为9bit），0001 0001 0000 0000，因为高9bit的最高bit为0，不需要进行异或，同理，接下来的两bit也是0，也不需要进行异或。 这样处理后数据为：1 0001 0000 0000；

C、 接下来最高位为1，需要进行异或操作了

1 0001 0000 0000 (因为是CRC8 所以data左移8位)

^ 1 0011 0001 0000(这个是 二项式公式参数化，要最高位对齐，后面补0)

---------------------------------------------

0 0010 0001 0000

^ 0 0010 0110 0010

----------------------------------------------

0 0000 0111 0010 (bit7 以前的数为0，结束异或，结果为0x72)

**这里的例子与文档中的二项式不一样，要注意变通。**

# 附录三 I2C与SMBus的异同

**相同点：**

　　2条线的总线协议(1个时钟，1个数据)

　　主-从通信，主设备提供时钟

　　多主机功能

　　SMBus数据格式类似于I2C的7位地址格式

**不同点**

尽管SMBus源自I2C，但两条总线的规格在电气，时序，协议和操作模式方面有几个主要差异。

|  |  |
| --- | --- |
| SMBus | I2C |
| 最大传输速度 100kHz | 最大传输速度400kHz |
| 最小传输速度 10kHz | 无最小传输速度 |
| 35ms时钟低超时 | 无时钟超时 |
| 固定的逻辑电平 | 逻辑电平由VDD决定 |
| 不同的地址类型(保留、动态等) | 7位、10位和广播呼叫从地址类型 |
| 不同的总线协议（快速命令、处理呼叫等) | 无总线协议 |

# 附录四 SMBus 设备地址分配

|  |  |
| --- | --- |
| **Slave Address** | **Description** |
| 0001 000 | SMBus Host |
| 0001 001 | Smart Battery Charger |
| 0001 010 | Smart Battery Selector Smart Battery System Manager |
| 0001 011 | Smart Battery |
| 0001 100 | SMBus Alert Response |
| 0101 000 | ACCESS.bus host |
| 0101 100 | Reserved by previous versions of the SMBus specification for LCD Contrast Controller. This address may be reassigned in future versions of the SMBus specification. |
| 0101 101 | Reserved by previous versions of the SMBus specification for CCFL Backlight Driver. This address may be reassigned in future versions of the SMBus specification. |
| 0110 111 | ACCESS.bus default address |
| 1000 0XX | Reserved by previous versions of the SMBus specification for PCMCIA Socket Controllers (4 addresses) . These addresses may be reassigned in future versions of the SMBus specification. |
| 1000 100 | Reserved by previous versions of the SMBus specification for (VGA) Graphics Controller. This address may be reassigned in future versions of the SMBus specification. |
| 1001 0XX | Unrestricted addresses |
| 1100 001 | SMBus Device Default Address |