

UCD30xx

PMBus

Programmer's Manual

Literature Number: xxxxxx

Date

Table of Contents

1	PMBus Interface	3
2	Slave Mode Initialization.....	3
2.1	Slave mode PMBus addressing.....	3
2.1.1	Automatic Slave Mode Address Response	3
2.1.2	Reading from the PMBus Status (PMBST) register.....	3
2.1.3	Manual Slave Address Acknowledge	3
2.2	Controlling how many bytes are acknowledged automatically	3
2.2.1	Acknowledging the command byte manually.....	3
2.2.2	Controlling the number of data bytes acknowledged.	3
2.3	PEC Enable Bit	3
2.4	Bus Speed Bits	3
3	Handling Key Slave Mode Message Types	3
3.1	Send Byte	3
3.2	Write Byte.....	3
3.3	Write Word	3
3.4	Block Write.....	3
3.5	Read Byte.....	3
3.5.1	Recognizing a read command.....	3
3.5.2	Responding to a read command.....	3
3.6	Read Word	3
3.7	Block Read.....	3
3.8	Group Commands	3
4	Using the PMBus Interface on the I2C bus.	3
5	PMBus with Clock Low Timeout.....	3
5.1	Clock Low Timeout Recovery.....	3
5.2	Detecting the need for recovery.....	3
6	PMBus Interface Registers Reference	3
6.1	PMBUS Control Register 1 (PMBCTRL1)	3
6.2	PMBus Transmit Data Buffer (PMBTXBUF).....	3
6.3	PMBus Receive Data Register (PMBRXBUF)	3
6.4	PMBus Acknowledge Register (PMBACK)	3
6.5	PMBus Status Register (PMBST).....	3
6.6	PMBus Interrupt Mask Register (PMBINTM).....	3
6.7	PMBus Control Register 2 (PMBCTRL2).....	3
6.8	PMBus Hold Slave Address Register (PMBHSA)	3
6.9	PMBus Control Register 3 (PMBCTRL3).....	3
6.10	PMBus Trim Register (PMBTRIM)	3
7	Master Mode Operation Reference.....	3
7.1	Quick Command	3
7.2	Send Byte	3
7.3	Receive Byte	3
7.4	Write Byte/Word.....	3

7.5	Read Byte/Read Word	3
7.6	Process Call.....	3
7.7	Block Write	3
7.8	Block Read.....	3
7.9	Block Write-Block Read Process Call.....	3
7.10	Alert Response	3
7.11	Extended Command – Write Byte/Word, Read Byte/Word.....	3
7.12	Group Command.....	3
8	Slave Mode Operation Reference	3
8.1	Slave Address Acknowledgement	3
8.2	Command Byte Acknowledgement	3
8.3	Receive Byte Configuration.....	3
8.4	Message Acknowledgement	3
8.5	Quick Command	3
8.6	Send Byte	3
8.7	Receive Byte	3
8.8	Write Byte/Word.....	3
8.9	Read Byte/Word.....	3
8.10	Process Call.....	3
8.11	Block Write	3
8.12	Block Read.....	3
8.13	Block Write-Block Read Process Call.....	3
8.14	Alert Response	3
8.15	Extended Command – Read Byte/Word, Write Byte/Word.....	3
8.16	Group Command.....	3

1 PMBus Interface

The PMBus interface is designed to implement most PMBus features using a combination of hardware and firmware. It supports 4 PMBus lines – Data, Clock, Alert, and Control. There are also 2 ADC pins equipped with a current source to support external PMBus address configuration with 2 resistors.

Both Master and Slave modes are supported, and there are two ways to support multiple addresses in slave mode. The primary use of the PMBus interface in power applications is in Slave mode, so that is covered first, and in more detail.

Chapters 2 and 3 are effectively the programmers manual – they describe how to implement a polled PMBus system which supports common commands with the PEC enabled for slave mode.

Next, use of the PMBus interface as an approximation of the I2C interface is discussed in chapter 4.

Chapter 5 discusses a work around which may be required for the PMBus in certain rare fault conditions.

Chapter 6 gives a detailed reference guide to all the registers in the PMBus interface.

Chapters 7 and 8 are a reference manual with descriptions of all the message types supported in both master and slave mode.

This document is best used in conjunction with TI supplied PMBus code.

This document also presumes the reader is familiar with the PMBus protocol.

2 Slave Mode Initialization

In slave mode, the UCD30xx only responds to messages from the Master. Messages cannot be initiated by the slave. The only exception to this is the Alert line, which all slave devices can assert to alert the master to a fault or other issue.

Slave mode can be accomplished either with interrupts or with PMBus polling. The approach used in the standard TI Digital Power programs is polling. This is done so that other tasks can be given higher priority. Highest priority – the fast interrupt on the ARM – is given to power supply faults and to the highest speed functions for power supply control. The standard interrupt is given to power supply functions that are not as urgent, but need to be done on a regular schedule, on a fixed interval. A timer is used to accomplish this regularly scheduled interrupt.

Since the PMBus supports clock stretching to enable the slave to delay a response, the PMBus is run in the background, and can be interrupted by either interrupt.

2.1 Slave mode PMBus addressing

The UCD30xx has flexible PMBus addressing capability. It can support one address automatically. It can also support some combinations of multiple addresses automatically. Any possible combination of addresses can be supported with clock stretching and firmware address detection. On a busy bus with many other peripherals, this could pose a serious overhead burden for the UCD30xx and the bus.

2.1.1 Automatic Slave Mode Address Response

For automatic slave mode address response, put the address into the Slave address bit field in PMBCTRL2:

```
PmbusRegs.PMBCTRL2.bit.SLAVE_ADDR = 0x11;
```

Make sure that MAN_SLAVE_ACK is in its default 0 state, to disable manual slave acknowledge:

```
PmbusRegs.PMBCTRL2.bit.MAN_SLAVE_ACK = 0;
```

To respond to only one address, make sure that all the bits in the slave mask are set, like so:

```
PmbusRegs.PMBCTRL2.bit.SLAVE_MASK = 0x7f;
```

The slave mask field supports automatic recognition of multiple addresses. Each bit that is set in the slave mask register causes the hardware to compare that same bit in the slave address field to the incoming slave address. If the bits do not match, the UCD30xx will not respond to that address.

If a bit is cleared in the slave mask register, its corresponding bit in the slave address register is not compared. Either a 1 or a zero is accepted as a valid bit for address compare purposes.

So, for example, suppose the same sequence as above was used, except the least significant bit is cleared in the slave mask:

```
PmbusRegs.PMBCTRL2.bit.SLAVE_MASK = 0x7e;
```

In this case, the PMBus slave address hardware would ignore the least significant bit in the address.

So it would respond to addresses 0x10 and 0x11.

Suppose the 2 least significant bits were cleared:

```
PmbusRegs.PMBCTRL2.bit.SLAVE_MASK = 0x7c;
```

Then addresses from 0x10 to 0x13 would be acknowledged.

Any combination of bits is acceptable for the slave mask register.

```
PmbusRegs.PMBCTRL2.bit.SLAVE_MASK = 0x7d;
```

The code above only clears the second bit. So addresses 0x11 and 0x13 would be the only addresses responded to.

```
PmbusRegs.PMBCTRL2.bit.SLAVE_MASK = 0x3f;
```

The code above clears only the most significant bit. So addresses 0x11 and 0x51 would be acknowledged.

When the slave mask register set to all ones, it is not necessary to know what address was acknowledged. With one or more bits cleared in the slave mask register, it will be necessary to read from the PMBus Hold Slave Address Register (PMBHSA) in order to determine which address was acknowledged:

```
this_address = PmbusRegs.PMBHSA.bit.SLAVE_ADDR;
```

2.1.2 Reading from the PMBus Status (PMBST) register

For manual slave address acknowledge, it will be necessary to read from the PMBST register. This register is a clear on read register. All bits will be cleared on each read.

It may be necessary to check multiple bits at different times, so it is best to copy the register to a temporary variable and do comparisons to that temporary variable.

For simplicity, this step will sometimes be omitted below. In actual applications, however, the comparisons to the PMBST register will generally be done to a stored register value rather than to the register itself.

2.1.3 Manual Slave Address Acknowledge

In manual slave acknowledge:

1. Set the Manual Slave Acknowledge bit:

```
PmbusRegs.PMBCTRL2.bit.MAN_SLAVE_ACK = 1;
```

2. When the slave address ready bit is set:

```
if(PmbusRegs.PMBST.bit.SLAVE_ADDR_READY == 1)
{
```

3. Read from the slave address register:

```
    this_address = PmbusRegs.PMBHSA.bit.SLAVE_ADDR;
```

4. Either acknowledge or not acknowledge the address:

```
    if(this_address == ok)
        PmbusRegs.PMBACK.bit.ACK = 1; //acknowledge
    else
        PmbusRegs.PMBACK.bit.ACK = 0; //don't acknowledge
}
```

If the slave address is acknowledged, the device will continue to process the message. If it is not acknowledged, the device will ignore the bus until the next start command is issued.

2.2 Controlling how many bytes are acknowledged automatically

There are several options for acknowledging bytes in the body of the message. The most efficient is to permit automatic acceptance of the command and as many message bytes as possible. This is also the default state of the registers on power up.

But it is possible to control how many bytes are acknowledged. This is intended for use in applications which require a NACK as soon as the slave recognizes that there is an error in the message.

There is also a separate bit for acknowledging the command byte.

2.2.1 Acknowledging the command byte manually

To acknowledge the command byte, set the manual command (MAN_CMD) bit.

```
PmbusRegs.PMBCTRL2.bit.MAN_CMD = 1;
```

Then, if the data ready bit is set, check the command and ACK or NACK:

```
if(PmbusRegs.PMBST.bit.DATA_READY == 1)
{
    this_command = PmbusRegs.PMBRXBUF.byte.BYTE0;
    if(this_command == ok)
        PmbusRegs.PMBACK.bit.ACK = 1;
    else
        PmbusRegs.PMBACK.bit.ACK = 0;
}
```

Note that the if statement with the “==ok” is simplified for convenience. It would probably be somewhat more complex in the actual application.

If the command byte is acknowledged automatically, it will still be in the same location in the PMBus receive buffer, it will just be acknowledged automatically, and may have one or more data bytes also in the buffer with it.

2.2.2 Controlling the number of data bytes acknowledged.

The receive byte acknowledge count RX_BYTE_ACK_CNT bit in the PMBCTRL2 register controls how many bytes are received before a software ACK is required. The maximum setting – 4 bytes – is the default setting. An acknowledge is required after each 4 bytes because that is all the received byte buffer can hold.

RX_BYTE_ACK_CNT can be programmed for 1 to 4 bytes. Bytes appear first in Byte 0 of the PMBRXBUF register, and then fill in from there, going up to Byte 3.

The acknowledge or not acknowledge is handled as described above for address and command.

The availability of bytes for reading is indicated by the data ready bit.

Note that the data ready bit also indicates the availability of the command byte.

It is the responsibility of the firmware to sort out which state the PMBus interface is in. After an end of message or a NACK, the next data to arrive will be a command.

2.2.3 Acknowledgement requirements with automatic acknowledgement

Even with automatic acknowledgement, some writes to the PmbusRegs.PMBACK.bit.ACK bit are required.

All EOMs must be acknowledged. Also, if the message is longer than 4 bytes, each packet of 4 bytes must be acknowledged. This acknowledgement does not occur as an acknowledgement on the PMBus, but instead functions as an acknowledgement to the PMBus peripheral that the data has been read and the firmware is ready for more data.

This acknowledgement should always be done by writing a 1 to the PmbusRegs.PMBACK.bit.ACK register. Writing 0 to this register in automatic mode, or whenever the bus is not waiting for an acknowledgement, will cause an issue. The NACK will not be cleared until the next message. The next message will be NACKed!

2.3 PEC Enable Bit

The only other bit which requires initialization before a slave mode command can be received is the PEC enable bit. Setting the PEC enable bit enables hardware which checks the last byte in a message for a valid Packet Error Check value. If the value is valid, the PEC_VALID bit in PMBST will be set. The PEC byte will still be read into the PMBRXBUF register.

2.4 Bus Speed Bits

There are also 2 bits in PMBCTRL3 which need to be set for the proper state even for slave mode.

There is a data setup requirement for the bus which varies between the Speed modes (i.e. 250ns for Standard mode, 100ns for Fast Mode and 50ns for Fast Plus Mode). Other timing parameters related to start and stop conditions also vary between the different modes.

If both bits are cleared (default), the bus is set up for 100 KHz.

For 400 KHz., set the FAST_MODE_ENA bit:

```
PmbusRegs.PMBCTRL3.bit.FAST_MODE_ENA = 1;
```

For 1 MHz., set the FAST_MODE_PLUS_ENA bit.

```
PmbusRegs.PMBCTRL3.bit.FAST_MODE_PLUS_ENA = 1;
```

3 Handling Key Slave Mode Message Types

This section describes some of the message types for PMBus and how to determine which message type is being received.

It is oriented toward the most efficient mode of operation – with automatic address and command acknowledgment. It is also oriented toward having PEC enabled.

If automatic address acknowledgement is disabled, all messages will start with a SLAVE_ADDR_READY.

Read commands will have two, one for the read, and one for the write.

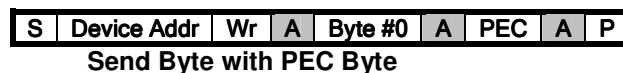
If automatic command acknowledgement is enabled, a DATA READY will occur for that as well.

If the message has no PEC, the number of bytes available will be one less. For example, with PEC, a QUICK COMMAND will have one byte, the PEC. With no PEC, a QUICK COMMAND will have zero bytes.

Note that the byte count does not increment as bytes arrive. no bits are set in the PMBST register until a stop message is received, the receive buffer is full, or a fault occurs. Then all appropriate bit values are placed in the register together.

All that is necessary to receive a quick command is to ACK the message by writing a 1 to the PMBACK register.

3.1 Send Byte



When a Send Byte message is received, the Data Ready bit is set, along with the EOM, and, assuming that the PEC is valid, the PEC valid bit. The read byte count (RD_BYTE_COUNT) register will contain a 2.

All that is necessary to receive a send byte command is to ACK the message by writing a 1 to the PMBACK register. Before doing the ACK, read the byte from the lowest byte of the PMBRXBUF register.

3.2 Write Byte



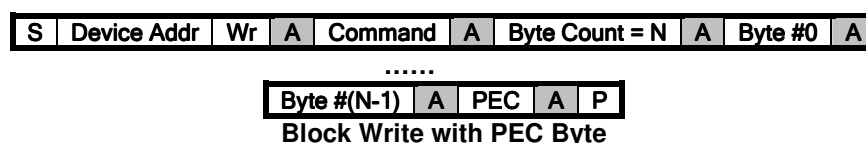
The Write Byte message will look exactly the same as the Send Byte, except the RD_BYTE_COUNT register will contain a 3.

3.3 Write Word



The Write Word message will have a RD_BYTE_COUNT of 4.

3.4 Block Write



3.5 Read Byte

S	Device Addr	Wr	A	Command	A	Sr	Device Addr	Rd	A	Byte #0	A	PEC	NA	P
---	-------------	----	---	---------	---	----	-------------	----	---	---------	---	-----	----	---

Read Byte with PEC Byte

The Read Byte command is more complex, as the UCD30xx is required to respond with data.

3.5.1 Recognizing a read command

When the repeated start (Sr) signal is received, the Data Ready bit will be asserted with a RD_BYTE_COUNT of 1. At this point, the operation cannot be distinguished from a group command send byte message.

When the same device address is sent out with a read, then the Data Request bit will be asserted.

If data has already been written to the PMTXBUF register before the Device Addr is received, then the Data Request bit will not be asserted.

So if group commands are also expected, it may be necessary to read the Data Ready with a RD_BYTE_COUNT of 1, and then wait and see whether the next event is an EOM or a Data Request.

If it is an EOM, then the command should be processed as a group send byte. If it is a Data Request, then the command should be processed as a read. Depending on the command, it could be a read byte, word, or block.

If the PMBus interface is polled, it is possible that both the Data Ready and the Data Request bits will be set between polling intervals, so that possibility should be considered in the design of the firmware.

3.5.2 Responding to a read command

Once the read command is recognized, it is necessary to respond by writing data to the PMBus Transmit Data Buffer (PMBTXBUF). It is also necessary to make sure that the values in the PMBCTRL2 register are correct. The transmit byte count must be correct, of course, and the PEC bits must also be correct.

For a read byte, the transmit byte count can be loaded with a 1:

```
PmbusRegs.PMBCTRL2.bit.TX_COUNT = 1;
```

And if the transmission of a PEC is desired, the TX_PEC bit should be set.

```
PmbusRegs.PMBCTRL2.bit.TX_PEC = 1;
```

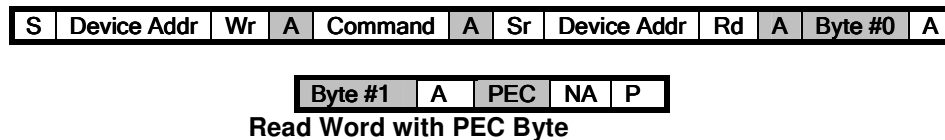
Once the PMBCTRL2 register bits are all correct, then the data can be written to PMBTXBUF. The write to PMBTXBUF starts the data transmission, so it must be done last.

```
PmbusRegs.PMBTXBUF.all = data_out;
```

It is best to write to the entire PMBTXBUF at once. Partial writes will also start the transmission, only the data will not all be ready.

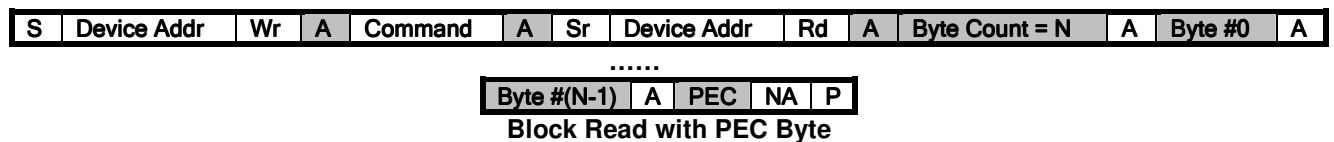
After the master receives the message, it will NACK the last byte to indicate that the correct number of bytes have been received. This will cause the EOM bit to be set in the PMBST register, indicating to the UCD30xx firmware that the Read Byte message is complete.

3.6 Read Word



The Read Word command is handled exactly the same as the Read Byte command, except TX_COUNT is loaded with a 2 instead of a 1, and PMBTXBUF is loaded with two bytes instead of one.

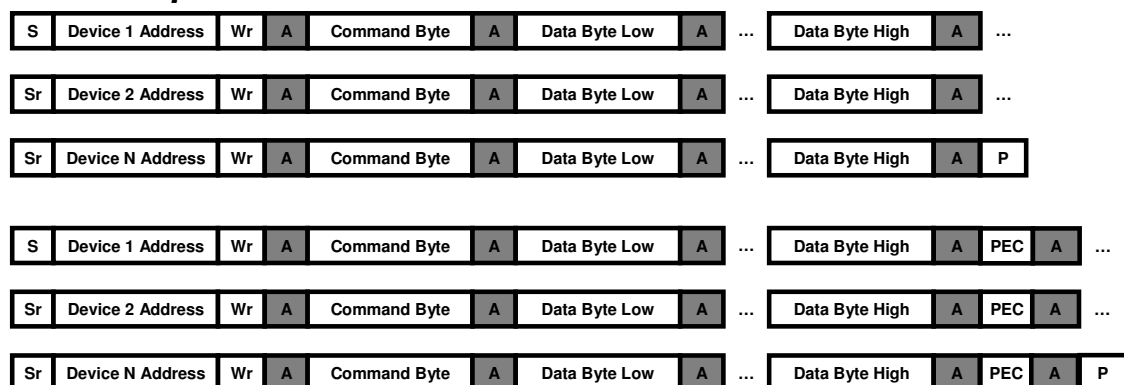
3.7 Block Read



Block read starts the same as Read Word or Read Byte, but TX_COUNT is loaded with a 4 the first time, and TX_PEC is not set. Instead of waiting for an EOM after the first transmission, the firmware instead waits for a Data Request, indicating that the master is ready for more data. Until the last 4 or less bytes, the firmware simply writes a 4 to TX_COUNT and then writes the 4 bytes to PMBTXBUF. TX_PEC is left cleared.

Then when the last 4 or fewer bytes are to be transmitted, the firmware writes out the appropriate byte count, sets the TX_PEC bit, and writes the data to PMBTXBUF. The PMBus interface will write out the data, followed by the PEC, and then the EOM bit will be set when the master NACKs the PEC.

3.8 Group Commands



For Group Commands, the data ready bit will be set as soon as the repeated start is received. The data can then be read into memory. But the data should not be acted upon until the EOM bit is set, which will occur when all of the messages have been received.

Other than this delayed EOM, there is no difference for the slave firmware in receiving a Group Command than any other write message.

4 Using the PMBus Interface on the I2C bus.

The PMBus is closely related to the I2C bus. In normal use the differences will not cause problems.

The primary difference is the clock high and clock low timeout limits that are required in the PMBus but are not required in the I2C bus. They are intended to prevent the bus from being locked up by a device which fails in the middle of a message. The clock low timeout occurs after 25 milliseconds, and the clock high timeout occurs after 50 milliseconds. The timeouts are only checked for during message transmission, i.e. between a start and a stop or NACK signal on the bus.

If the clock low timeout occurs, the PMBus state machine is reset. The clock low timeout flag is also asserted, and a clock low interrupt can be generated if it is enabled.

The clock high timeout is more often a problem with I2C compatibility. Therefore, the clock high timeout only sets a clock high bit. The PMBus state machine is not reset, and so the clock high timeout can be considered an optional feature. To conform to the PMBus interface specification, the firmware must reset the PMBus interface when the clock high timeout bit is set.

The PMBus interface will also not support the continuous read function from the I2C spec.

There are also subtle differences in the signal levels and drive capabilities specified for the PMBus and I2C bus. Consult the UCD30xx data sheets for more information on this issue. Generally these differences do not cause any problems.

5 PMBus with Clock Low Timeout

The PMBus uses SMBus for the basis of its physical layer. SMBus defines a maximum time (25ms to 35ms) that the clock line can be held low. When this time is exceeded, the active master device on the bus should create a stop condition on the bus. The slave device should respond by resetting its communication engine and begin looking for a start condition.

The timeout for clock low is like a watchdog, if it expires then a device should reset to recover.

This fault should be a very rare event that is an indication of an out of specification design or of a major failure in the system.

With a proper system design this fault should never occur. If it does then the root cause should be fixed and not count on this corrective action to mitigate another potential design issue.

5.1 Clock Low Timeout Recovery

If a clock low timeout occurs while the UCD30xx is in slave mode and is pulling the data line low, then the master device can not issue a stop condition as the data line is being held low. When the UCD30xx detects the clock low timeout, it performs a PMBus reset. This action will not release the data line. The data line under this condition is released after a peripherals reset.

When the peripherals reset is performed the peripherals affected are ADC-12, MAC, PMBus, GPIO, SCI, SPI, and the Timer Module. The Fusion Power Peripherals are not affected by the peripherals reset, so power supply control can continue through the reset.

To reset the peripherals, it is necessary to clear and then set the PBUF_ENA bit in the memory controller:

```
MmcRegs.PCTRL.bit.PBUF_ENA = 0;  
MmcRegs.PCTRL.bit.PBUF_ENA = 1;
```

This should be done in privileged mode. The best way to accomplish this is to use a software interrupt call to reset the peripherals.

After the peripherals reset the affected peripherals must be reinitialized. On a reset, all fields in the affected peripherals will return to their default states.

It should only take a few microseconds to return key peripherals to full functionality. These key peripherals should be reinitialized first after the reset.

The peripherals reset should be done at the time when the reset has will not cause operational concerns. Given that the actual timeout specification ranges from 25ms to 35ms, the system can choose to delay the execution of the peripherals reset by several milliseconds.

5.2 Detecting the need for recovery

There are 3 options to detect that a Clock Low Timeout has occurred with a data line low.

1. Simply reset the interface whenever the Clock Low Timeout bit is set. This will cause some unnecessary resets, but will guarantee the quickest recovery.
2. After a Clock Low Timeout bit is set, wait a predetermined time. If a PMBus message is received during this predetermined time, then the interface is still functional, so there is no need for a reset. If a PMBus message is not received, then the interface needs to be reset. This interval needs to be determined for each application based on the expected frequency of messages from the host. If the PMBus is busy with communications to other modules, it is possible to switch to manual address acknowledge mode. This give some visibility to messages to other modules, and can speed up verification of continued bus functioning.
3. It is also possible to hook up another GPIO pin to the data line. If this GPIO pin is configured as an input, the state of the data line can be monitored. If it is consistently low, especially if the bus is inactive, then the reset must be performed.

6 PMBus Interface Registers Reference

6.1 *PMBUS Control Register 1 (PMBCTRL1)*

Address FFF7F600

Bit Number	20	19	18	17
Bit Name	PRC_CALL	GRP_CMD	PEC_ENA	EXT_CMD
Access	R/W	R/W	R/W	R/W
Default	0	0	0	0

Bit Number	16	15:8	7:1	0
Bit Name	CMD_ENA	BYTE_COUNT	SLAVE_ADDR	RW
Access	R/W	R/W	R/W	R/W
Default	0	0000_0000	000_0000	0

Bit 20: PRC_CALL – Master Process Call Message Enable

0 = Default state for all messages besides Process Call message (Default)

- 1 = Enables transmission of Process Call message
- Bit 19: GRP_CMD** – Master Group Command Message Enable
0 = Default state for all messages besides Group Command message (Default)
1 = Enables transmission of Group Command message
- Bit 18: PEC_ENA** – Master PEC Processing Enable
0 = Disables PEC processing (Default)
1 = Enables PEC byte transmission/reception
- Bit 17: EXT_CMD** – Master Extended Command Code Enable
0 = Use 1 byte for Command Code (Default)
1 = Use 2 bytes for Command Code
- Bit 16: CMD_ENA** – Master Command Code Enable
0 = Disables use of command code on Master initiated messages (Default)
1 = Enables use of command code on Master initiated messages
- Bits 15-8: BYTE_COUNT** – Indicates number of data bytes transmitted in current message. Byte count does not include any device addresses, command words or block lengths in block messages. In block messages, the PMBus Interface automatically inserts the block length into the message based on the byte count setting. The firmware only needs to load the address, command words and data to be transmitted. PMBus Interface supports byte writes up to 255 bytes.
- Bits 7-1: SLAVE_ADDR** – Specifies the address of the slave to which the current message is directed towards.
- Bit 0: RW** – Indicates if current Master initiated message is read operation or write operation.
0 = Message is a write transaction (data from Master to Slave) (Default)
1 = Message is a read transaction (data from Slave to Master)

6.2 PMBus Transmit Data Buffer (PMBTXBUF)

Address FFF7F604

Bit Number	31:24	23:16	15:8	7:0
Bit Name	BYTE3	BYTE2	BYTE1	BYTE0
Access	R/W	R/W	R/W	R/W
Default	0000_0000	0000_0000	0000_0000	0000_0000

- Bits 31-24: BYTE3** – Last data byte transmitted from Transmit Data Buffer
Bits 23-16: BYTE2 – Third data byte transmitted from Transmit Data Buffer
Bits 15-8: BYTE1 – Second data byte transmitted from Transmit Data Buffer
Bits 7-0: BYTE0 – First data byte transmitted from Transmit Data Buffer

6.3 PMBus Receive Data Register (PMBRXBUF)

Address FFF7F608

Bit Number	31:24	23:16	15:8	7:0
Bit Name	BYTE3	BYTE2	BYTE1	BYTE0
Access	R	R	R	R
Default	-	-	-	-

- Bits 31-24: BYTE3** – Last data byte received in Receive Data Buffer
Bits 23-16: BYTE2 – Third data byte received in Receive Data Buffer
Bits 15-8: BYTE1 – Second data byte received in Receive Data Buffer
Bits 7-0: BYTE0 – First data byte received in Receive Data Buffer

6.4 *PMBus Acknowledge Register (PMBACK)*

Address FFF7F60C

Bit Number	0
Bit Name	ACK
Access	R/W
Default	0

Bit 0: ACK – Allows firmware to acknowledge or not acknowledge received data
0 = NACK received data (Default)
1 = Acknowledge received data, bit clears upon issue of ACK on PMBus

6.5 PMBus Status Register (PMBST)

Address FFF7F610

Bit Number	19	18	17	16
Bit Name	CONTROL_RAW	ALERT_RAW	CONTROL_EDGE	ALERT_EDGE
Access	R	R	R	R
Default	-	-	-	-

Bit Number	15	14	13	12	11
Bit Name	MASTER	LOST_ARB	BUS_FREE	UNIT_BUSY	RPT_START
Access	R	R	R	R	R
Default	-	-	-	-	-

Bit Number	10	9	8
Bit Name	SLAVE_ADDR_READY	CLK_HIGH_TIMEOUT	CLK_LOW_TIMEOUT
Access	R	R	R
Default	-	-	-

Bit Number	7	6	5	4	3
Bit Name	PEC_VALID	NACK	EOM	DATA_REQUEST	DATA_READY
Access	R	R	R	R	R
Default	-	-	-	-	-

Bit Number	2:0
Bit Name	RD_BYTE_COUNT
Access	R
Default	-

- Bit 19: CONTROL_RAW** – Control Pin Real Time Status
0 = Control pin observed at logic level low
1 = Control pin observed at logic level high
- Bit 18: ALERT_RAW** – Alert Pin Real Time Status
0 = Alert pin observed at logic level low
1 = Alert pin observed at logic level high
- Bit 17: CONTROL_EDGE** – Control Edge Detection Status
0 = Control pin has not transitioned
1 = Control pin has been asserted by another device on PMBus
- Bit 16: ALERT_EDGE** – Alert Edge Detection Status
0 = Alert pin has not transitioned
1 = Alert pin has been asserted by another device on PMBus
- Bit 15: MASTER** – Master Indicator
0 = PMBus Interface in Slave Mode or Idle Mode
1 = PMBus Interface in Master Mode
- Bit 14: LOST_ARB** – Lost Arbitration Flag
0 = Master has attained control of PMBus
1 = Master has lost arbitration and control of PMBus
- Bit 13: BUS_FREE** – PMBus Free Indicator
0 = PMBus processing current message
1 = PMBus available for new message
- Bit 12: UNIT_BUSY** – PMBus Busy Indicator

- 0 = PMBus Interface is idle, ready to transmit/receive message
- 1 = PMBus Interface is busy, processing current message
- Bit 11: RPT_START** – Repeated Start Flag
 - 0 = No Repeated Start received by interface
 - 1 = Repeated Start condition received by interface
- Bit 10: SLAVE_ADDR_READY** – Slave Address Ready
 - 0 = Indicates no slave address is available for reading
 - 1 = Slave address ready to be read from Receive Data Register (Bits 6:0)
- Bit 9: CLK_HIGH_TIMEOUT** – Clock High Timeout Status
 - 0 = No clock high timeout detected
 - 1 = Clock High timeout detected
- Bit 8: CLK_LOW_TIMEOUT** – Clock Low Timeout Status
 - 0 = No clock low timeout detected
 - 1 = Clock low timeout detected, clock held low for greater than 35ms
- Bit 7: PEC_VALID** – PEC Valid Indicator
 - 0 = Received PEC not valid (if EOM is asserted)
 - 1 = Received PEC is valid
- Bit 6: NACK** – Not Acknowledge Flag Status
 - 0 = Data transmitted has been accepted by receiver
 - 1 = Receiver has not accepted transmitted data
- Bit 5: EOM** – End of Message Indicator
 - 0 = Message still in progress or PMBus in idle state.
 - 1 = End of current message detected
- Bit 4: DATA_REQUEST** – Data Request Flag
 - 0 = No data needed by PMBus Interface
 - 1 = PMBus Interface request additional data. PMBus clock stretching enabled to stall bus until firmware provides transmit data.
- Bit 3: DATA_READY** – Data Ready Flag
 - 0 = No data available for reading by processor
 - 1 = PMBus Interface read buffer full, firmware required to read data prior to further bus activity. PMBus clock stretching enabled to stall bus until data is read by firmware.
- Bits 2-0: RD_BYTE_COUNT** – Number of Data Bytes available in Receive Data Register
 - 0 = No received data
 - 1 = 1 byte received. Data located in Receive Data Register, Bits 7-0
 - 2 = 2 bytes received. Data located in Receive Data Register, Bits 15-0
 - 3 = 3 bytes received. Data located in Receive Data Register, Bits 23-0
 - 4 = 4 bytes received. Data located in Receive Data Register, Bits 31-0

6.6 PMBus Interrupt Mask Register (PMBINTM)

Address FFF7F614

Bit Number	8	7	6	5
Bit Name	LOST_ARB	CONTROL	ALERT	EOM
Access	R/W	R/W	R/W	R/W
Default	1	1	1	1

Bit Number	4	3	2
Bit Name	SLAVE_ADDR_READY	DATA_REQUEST	DATA_READY
Access	R/W	R/W	R/W
Default	1	1	1

Bit Number	1	0
Bit Name	BUS_LOW_TIMEOUT	BUS_FREE
Access	R/W	R/W
Default	1	1

Bit 8: LOST_ARB – Lost Arbitration Interrupt Mask

- 0 = Generates interrupt upon assertion of Lost Arbitration flag
- 1 = Disables interrupt generation upon assertion of Lost Arbitration flag

Bit 7: CONTROL – Control Detection Interrupt Mask

- 0 = Generates interrupt upon assertion of Control flag
- 1 = Disables interrupt generation upon assertion of Control flag

Bit 6: ALERT – Alert Detection Interrupt Mask

- 0 = Generates interrupt upon assertion of Alert flag
- 1 = Disables interrupt generation upon assertion of Alert flag

Bit 5: EOM – End of Message Interrupt Mask

- 0 = Generates interrupt upon assertion of End of Message flag
- 1 = Disables interrupt generation upon assertion of End of Message flag

Bit 4: SLAVE_ADDR_READY – Slave Address Ready Interrupt Mask

- 0 = Generates interrupt upon assertion of Slave Address Ready flag
- 1 = Disables interrupt generation upon assertion of Slave Address Ready flag

Bit 3: DATA_REQUEST – Data Request Interrupt Mask

- 0 = Generates interrupt upon assertion of Data Request flag
- 1 = Disables interrupt generation upon assertion of Data Request flag

Bit 2: DATA_READY – Data Ready Interrupt Mask

- 0 = Generates interrupt upon assertion of Data Ready flag
- 1 = Disables interrupt generation upon assertion of Data Ready flag

Bit 1: BUS_LOW_TIMEOUT – Clock Low Timeout Interrupt Mask

- 0 = Generates interrupt upon assertion of Clock Low Timeout flag
- 1 = Disables interrupt generation upon assertion of Clock Low Timeout flag

Bit 0: BUS_FREE – Bus Free Interrupt Mask

- 0 = Generates interrupt upon assertion of Bus Free flag
- 1 = Disables interrupt generation upon assertion of Bus Free flag

6.7 PMBus Control Register 2 (PMBCTRL2)

Address FFF7F618

Bit Number	22:21	20	19	18:16
Bit Name	RX_BYTE_ACK_CNT	MAN_CMD	TX_PEC	TX_COUNT

Access	R/W	R/W	R/W	R/W
Default	11	0	0	000

Bit Number	15	14:8	7	6:0
Bit Name	PEC_ENA	SLAVE_MASK	MAN_SLAVE_ACK	SLAVE_ADDR
Access	R/W	R/W	R/W	R/W
Default	0	111_1111	0	111_1100

Bit 22-21: RX_BYTE_ACK_CNT – Configures number of data bytes to automatically acknowledge when receiving data in slave mode.

00 = 1 byte received by slave. Firmware is required to manually acknowledge every received byte.

01 = 2 bytes received by slave. Hardware automatically acknowledges the first received byte. Firmware is required to manually acknowledge after the second received byte.

10 = 3 bytes received by slave. Hardware automatically acknowledges the first 2 received bytes. Firmware is required to manually acknowledge after the third received byte.

11 = 4 bytes received by slave. Hardware automatically acknowledges the first 3 received bytes. Firmware is required to manually acknowledge after the fourth received byte.

Bit 20: MAN_CMD – Manual Command Acknowledgement Mode

0 = Slave automatically acknowledges received command code (Default)

1 = Data Ready flag generated after receipt of command code, firmware required to issue ACK to continue message

Bit 19: TX_PEC – Asserted when the slave needs to send a PEC byte at end of message.

PMBus Interface will transmit the calculated PEC byte after transmitting the number of data bytes indicated by TX Byte Cnt(Bits 19:17).

0 = No PEC byte transmitted (Default)

1 = PEC byte transmitted at end of current message

Bit 18-16: TX_COUNT– Number of valid bytes in Transmit Data Register

0 = No bytes valid (Default)

1 = One byte valid, Byte #0 (Bits 7:0 of Receive Data Register)

2 = Two bytes valid, Bytes #0 and #1 (Bits 15:0 of Receive Data Register)

3 = Three bytes valid, Bytes #0-2 (Bits 23:0 of Receive Data Register)

4 = Four bytes valid, Bytes #0-3 (Bits 31:0 of Receive Data Register)

Bit 15: PEC_ENA – PEC Processing Enable

0 = PEC processing enabled (Default)

1 = PEC processing disabled

Bit 14-8: SLAVE_MASK – Used in address detection, the slave mask enables acknowledgement of multiple device addresses by the slave. Writing a '0' to a bit within the slave mask enables the corresponding bit in the slave address to be either '1' or '0' and still allow for a match. Writing a '0' to all bits in the mask enables the PMBus Interface to acknowledge any device address. Upon power-up, the slave mask defaults to 7Fh, indicating the slave will only acknowledge the address programmed into the Slave Address (Bits 6-0).

Bit 7: MAN_SLAVE_ACK– Manual Slave Address Acknowledgement Mode

0 = Slave automatically acknowledges device address specified in SLAVE_ADDR, Bits 6-0 (Default)

1 = Enables the Manual Slave Address Acknowledgement Mode. Firmware is required to read received address and acknowledge on every message

Bits 6-0: SLAVE_ADDR – Configures the current device address of the slave. Used in automatic slave address acknowledge mode (default mode). The PMBus Interface will compare the received device address with the value stored in the Slave Address bits and the mask configured in the Slave Mask bits. If matching, the slave will acknowledge the device address.

6.8 PMBus Hold Slave Address Register (PMBHSA)

Address FFF7F61C

Bit Number	7:1	0
Bit Name	SLAVE_ADDR	SLAVE_RW
Access	R	R
Default	-	-

Bits 7-1: SLAVE_ADDR – Stored device address acknowledged by the slave

Bit 0: SLAVE_RW – Stored R/W bit from address acknowledged by the slave

0 = Write Access

1 = Read Access

6.9 PMBus Control Register 3 (PMBCTRL3)

Address FFF7F620

Bit Number	5	4	3
Bit Name	CNTL_INT_EDGE	FAST_MODE_PLUS_ENA	FAST_MODE_ENA
Access	R/W	R/W	R/W
Default	0	0	0

Bit Number	2	1	0
Bit Name	BUS_LO_INT_EDGE	ALERT_ENA	RESET
Access	R/W	R/W	R/W
Default	0	0	0

Bit 5: CNTL_INT_EDGE – Control Interrupt Edge Select

0 = Interrupt generated on falling edge of Control (Default)

1 = Interrupt generated on rising edge of Control

Bit 4: FAST_MODE_PLUS_ENA – Fast Mode Plus Enable

0 = Standard 100 KHz mode enabled (Default)

1 = Fast Mode Plus enabled (1MHz operation on PMBus)

Bit 3: FAST_MODE_ENA – Fast Mode Enable

0 = Standard 100 KHz mode enabled (Default)

1 = Fast Mode enabled (400KHz operation on PMBus)

Bit 2: BUS_LO_INT_EDGE – Clock Low Timeout Interrupt Edge Select

0 = Interrupt generated on rising edge of clock low timeout (Default)

1 = Interrupt generated on falling edge of clock low timeout

Bit 1: ALERT_ENA – Slave Alert Enable

0 = PMBus Alert is not driven by slave, pulled up high on PMBus (Default)

1 = PMBus Alert driven low by slave

Bit 0: RESET – PMBus Interface Synchronous Reset

0 = No reset of internal state machines (Default)

1 = Control state machines are reset to initial states

6.10 PMBus Trim Register (PMBTRIM)

Address FFF7F624

Bit Number	4	3	2:0
Bit Name	IBIAS_B_ENA	IBIAS_A_ENA	IBIAS_TRIM
Access	R/W	R/W	R/W
Default	0	0	000

Bit 4: IBIAS_B_ENA – PMBus Current Source B Control

0 = Disables Current Source for PMBUS address detection thru ADC (Default)

1 = Enables Current Source for PMBUS address detection thru ADC

Bit 3: IBIAS_A_ENA – PMBus Current Source A Control

0 = Disables Current Source for PMBUS address detection thru ADC (Default)

1 = Enables Current Source for PMBUS address detection thru ADC

Bits 2-0: IBIAS_TRIM – Trim bits for PMBus Address Detection, Bits will be programmed during test and should not be overwritten by firmware.

7 Master Mode Operation Reference

The PMBus Interface has the capability to initiate any of the available message protocols used for communication with connected slave modules. Initiating a message begins with programming the Master Control Register (Address 0h). Upon programming of this register, the message will begin transmission on the PMBus once the bus is idle and ready for additional messages.

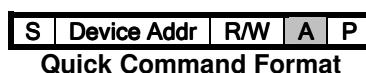
Within the bits of the Master Control Register, a number of options are provided that help to configure the PMBus message. The PMBus Interface includes an optional PEC enable bit (Bit 18). Enabled the PEC_EN bit forces the PMBus Interface to append a PEC byte onto the end of the message. The firmware is not required to calculate the PEC value or account for the PEC byte when entering the number of bytes in the message.

The Byte Count bits (Bits 15-8) within the Master Control Register configures the number of data bytes within the outgoing message. The firmware is required to program the byte count at the start of each message. A byte count of zero will result in the transmission of a Quick Command message. The byte count does not include any command bytes, PEC bytes, address bytes or the block length (found in Block Write/Read messages). The PMBus Interface in Master Mode automatically terminates a valid message based on the values programmed in the byte count bits. In cases of a slave NACK, the PMBus Interface also automatically initiates a stop condition to terminate the message and provides the appropriate alarms to the firmware through the Status Register.

Inclusion of command bytes in the message initiated by the Master is configured through the CMD_EN and EXT_CMD bits (Bits 17-16 of the Master Control Register). Enabling CMD_EN forces the PMBus Interface to include a single command byte in the message. On the first programming of the Transmit Data Register, bits 7-0 will represent the command byte. When enabling EXT_CMD, support for extended commands is enabled. Bits 15-0 of the Transmit Data Register, after the first program of the Transmit Data Register, represent the extended command bytes.

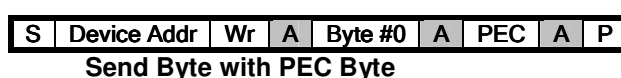
Additional control bits within the Master Control Register enable various message protocols for PMBus applications. Protocols such as Process Call and Group Command Messages require additional programming with these bits.

7.1 Quick Command



Quick commands are initiated in Master Mode by simply programming the desired slave device address into the Master Control Register. The byte count within the Master Control Register is configured to 0 bytes by writing all zeros to bits 15-8. Upon transmission of the device address, the PMBus Interface will monitor the slave acknowledgement of the address. If the address is not acknowledged, the Naked bit within the status register is enabled and the PMBus Interface automatically enables a stop condition on the PMBus to terminate the message. If the address is acknowledged, a data request is issued to the processor. The firmware writes a '0' to the Acknowledge Register to terminate the message, forcing the PMBus Interface to write a stop condition onto the PMBus.

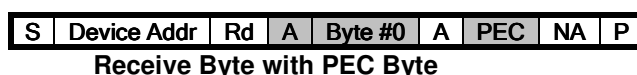
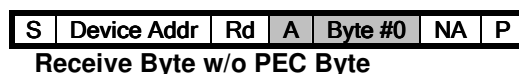
7.2 Send Byte



A Send Byte message consists of the device address, a single data byte and an optional PEC byte. To initiate a Send Byte message, the data byte to be transmitted to the slave is loaded into bits 7-0 of the Transmit Data Register. The Master Control Register is configured with the device address. To transmit a PEC byte with the message, the PEC_EN bit within the Master Control Register is asserted high when the address is programmed.

After programming the Master Control Register, the PMBus Interface initiates the Send Byte message onto the PMBus. The firmware can wait for an End of Message interrupt from the PMBus Interface. Upon receipt of the EOM interrupt, the Status Register is read to verify the slave properly acknowledged the transmitted data.

7.3 Receive Byte



A Receive Byte message consists of the device address, a single data byte and an optional PEC byte. Data is being read from the slave in a Receive Byte message. To initiate a Receive Byte message, the firmware programs the device address, the R/W bit and the optional PEC_EN into the Master Control Register. The R/W bit is enabled high to indicate a read message type (data transmitted from Slave to Master).

After programming the Master Control Register, the PMBus Interface initiates a Receive Byte message onto the PMBus. The firmware can wait for an End of Message interrupt from the PMBus Interface to verify the accuracy of the message transmission. Upon receipt of the EOM interrupt, the Status Register is read to verify proper slave acknowledgement of the device address and to determine if any data is available for reading in the Receive Data Register. If PEC_EN was asserted in the Master Control Register, the PEC_VALID bit in the Status Register is also checked to ensure a proper PEC byte was received from the Slave with the received data.

7.4 Write Byte/Word

S	Device Addr	Wr	A	Command	A	Byte #0	A	P
---	-------------	----	---	---------	---	---------	---	---

Write Byte w/o PEC Byte

S	Device Addr	Wr	A	Command	A	Byte #0	A	PEC	A	P
---	-------------	----	---	---------	---	---------	---	-----	---	---

Write Byte with PEC Byte

S	Device Addr	Wr	A	Command	A	Byte #0	A	Byte #1	A	P
---	-------------	----	---	---------	---	---------	---	---------	---	---

Write Word w/o PEC Byte

S	Device Addr	Wr	A	Command	A	Byte #0	A	Byte #1	A	PEC	A	P
---	-------------	----	---	---------	---	---------	---	---------	---	-----	---	---

Write Word with PEC Byte

The Write Byte and Write Word messages consist of a device address, a command byte, transmitted data bytes and an optional PEC byte. Write Byte messages include a single byte, while the Write Word messages support transmission of 2 bytes to the corresponding slave module. Similar to the Send Byte protocol, the Master Control Register is configured to send 1 or 2 bytes, the CMD_EN bit is set to enable command byte transmission and the optional PEC_EN bit is set.

With the command byte transmission enabled, the format of the Transmit Data Register differs from the Send Byte protocol. In Bits 7-0 of the Transmit Data Register, the firmware must program the command byte to be sent to the slave. The data byte(s) are programmed into bits 15-8 and bits 23-16.

After programming the Master Control Register, the PMBus Interface initiates a Write Byte/Word message on the PMBus. The firmware can wait for an End of Message interrupt from the interface to verify the accuracy of the message transmission. The Status Register indicates if the slave acknowledged the message properly.

7.5 Read Byte/Read Word

S	Device Addr	Wr	A	Command	A	Sr	Device Addr	Rd	A	Byte #0	NA	P
---	-------------	----	---	---------	---	----	-------------	----	---	---------	----	---

Read Byte w/o PEC Byte

S	Device Addr	Wr	A	Command	A	Sr	Device Addr	Rd	A	Byte #0	A	PEC	NA	P
---	-------------	----	---	---------	---	----	-------------	----	---	---------	---	-----	----	---

Read Byte with PEC Byte

S	Device Addr	Wr	A	Command	A	Sr	Device Addr	Rd	A	Byte #0	A
---	-------------	----	---	---------	---	----	-------------	----	---	---------	---

Byte #1	NA	P
---------	----	---

Read Word w/o PEC Byte

S	Device Addr	Wr	A	Command	A	Sr	Device Addr	Rd	A	Byte #0	A
---	-------------	----	---	---------	---	----	-------------	----	---	---------	---

Byte #1	A	PEC	NA	P
---------	---	-----	----	---

Read Word with PEC Byte

The Read Byte and Read Word messages consist of a device address, a command byte, received data bytes from a slave and an optional PEC byte. Read Byte messages include a single byte, while the Read Word message protocol supports receipt of 2 bytes from the slave. Similar to the Receive Byte Protocol, the Master Control Register is configured to receive 1 or 2 bytes, the CMD_EN bit is set and the PEC_EN is configured to expect or not expect a PEC byte appended to the message. The PMBus Interface will automatically terminate the message after the expected number of bytes is received from the slave or if the slave does not properly acknowledge any portion of the message.

In addition to programming the Master Control Register, the firmware is expected to load the command byte into bits 7-0 of the Transmit Data Register. Any data received from the slave will be found in the Receive Data Register.

After programming the Master Control Register, the PMBus Interface initiates a Read Byte/Read Word message on the PMBus. The firmware can wait for an End of Message interrupt from the interface. Upon the EOM interrupt, the Status Register is read to determine the number of bytes received (1 for a Read Byte/2 for a Read Word). The received data bytes are found in Bits 15-0 of the Receive Data Register. If PEC is enabled for the message, the PEC_VAL bit in the Status Register can be verified to check the accuracy of the received PEC byte from the Slave.

7.6 Process Call

S	Device Addr	Wr	A	Command	A	Byte #0	A	Byte #1	A	Sr	Device Addr	Rd	A
---	-------------	----	---	---------	---	---------	---	---------	---	----	-------------	----	---

Byte #0	A	Byte #1	NA	P
---------	---	---------	----	---

Process Call w/o PEC Byte

S	Device Addr	Wr	A	Command	A	Byte #0	A	Byte #1	A	Sr	Device Addr	Rd	A
---	-------------	----	---	---------	---	---------	---	---------	---	----	-------------	----	---

Byte #0	A	Byte #1	A	PEC	NA	P
---------	---	---------	---	-----	----	---

Process Call with PEC Byte

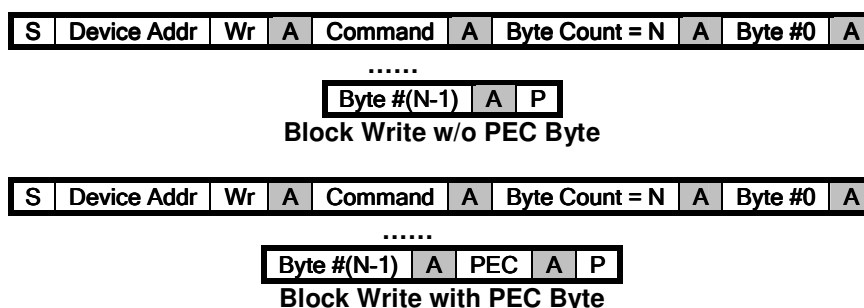
The Process Call protocol consists of a Write Word message, followed by a Read Word message, without a stop condition between the two messages. A PEC byte can be appended to the read data from the slave as an option to the message protocol. The Master Control Register includes a PRC_CALL bit, which enables the transmission of a Process Call message onto the PMBus. The PMBus Interface will automatically generate a repeated start condition and initiate the Read Word portion of the message when the process call bit is enabled.

To complete the Write Word portion of the Process Call, the Transmit Data Register is loaded with the command byte in Bits 7-0 and the data bytes are loaded into Bits 23-8 of the register.

After programming the Master Control Register, the PMBus Interface initiates the Process Call Message on the PMBus. The firmware can wait for an End of Message interrupt from the interface to determine the validity of the message. Upon the receipt of the EOM, the Status Register should indicate the receipt of 2 bytes from the Read Word portion of the Process Call message and the status of the Slave acknowledgement of the transmit data. If PEC processing is enabled, the PEC_VAL bit within the Status Register indicates the accuracy of the PEC byte received from the Slave during the Read Word part of the message.

The PRC_CALL bit within the Master Control Register should be disabled on the next message not of the Process Call protocol. Please note that any write to the Master Control Register initiates a message, so reconfiguration of the Master is not recommended until the firmware requires a new message to be transmitted on the PMBus.

7.7 Block Write



The Block Write protocol is similar to a Write Word in its structure, with the exception of transmission of more than 2 data bytes in the message. Additionally, the first data byte following the command byte specifies the length of the block of data bytes. As with a majority of the message protocols, the PEC byte can be appended to the end of the write data to the Slave.

To initiate a Block Write message on the PMBus, the Master Control Register is programmed with the block length in the Byte Count bits. The block length is the number of data bytes, excluding the command byte and the first data byte that contains the block length. The PMBus Interface will automatically insert the block length into the message if the number of data bytes specified by the firmware exceeds 2. The initial write data is loaded into the Transmit Data Register. With bits 7-0 representing the command byte, the remaining 3 bytes represent the first three data bytes following the block length.

Following programming of the Master Control Register, the Block Write message is initiated on the PMBus. If the block length exceeds three bytes, the PMBus Interface will provide a data request interrupt, indicating the need for additional data bytes in the Transmit Data Register. The PMBus Interface assumes that if more than 4 bytes are needed to complete the message, the firmware will utilize all 4 bytes when programming the Transmit Data Register. If less than 4 bytes are needed to finish the Block Write message, the firmware only needs to program the appropriate bits of the Transmit Data Register.

Upon completion of the message, the PMBus Interface issues an EOM interrupt. The interface can be checked to verify the slave accepted the block of write data.

7.8 Block Read

S	Device Addr	Wr	A	Command	A	Sr	Device Addr	Rd	A	Byte Count = N	A	Byte #0	A
---	-------------	----	---	---------	---	----	-------------	----	---	----------------	---	---------	---

.....

Byte #(N-1)	NA	P
-------------	----	---

Block Read w/o PEC Byte

S	Device Addr	Wr	A	Command	A	Sr	Device Addr	Rd	A	Byte Count = N	A	Byte #0	A
---	-------------	----	---	---------	---	----	-------------	----	---	----------------	---	---------	---

.....

Byte #(N-1)	A	PEC	NA	P
-------------	---	-----	----	---

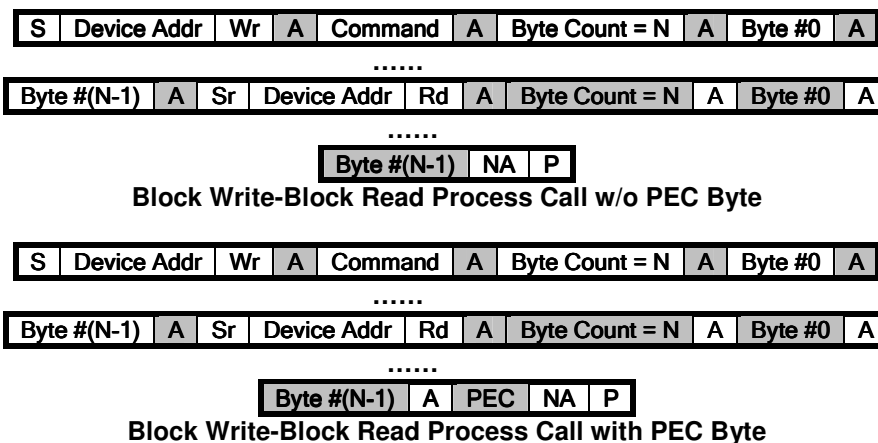
Block Read with PEC Byte

The Block Read protocol is similar to a Read Word in its structure, with the exception of reception of more than 2 data bytes from the Slave. The first data byte transmitted by the Slave represents the block length of the data being written by the slave. If PEC processing is enabled, the Slave appends a PEC byte to the end of the message.

To initiate a Block Read message on the PMBus, the Master Control Register is programmed with the block length in the Byte Count bits. This count excludes the command byte, any slave address and the block length bytes in the message. The command byte to be transmitted to the Slave is written into bits 7-0 of the Transmit Data Register prior to the programming of the Master Control Register.

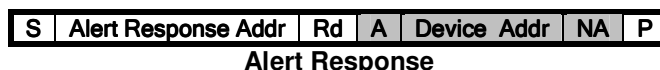
After configuring the Master Control Register, the Block Read message is initiated on the PMBus. The interface interrupts the firmware upon receipt of 4 data bytes from the slave. If the block length is 3, the EOM interrupt will be received concurrently with the data ready interrupt. Otherwise, a data ready interrupt is asserted, indicating 4 bytes are ready for reading by the firmware. At the end of the message, less than 4 bytes may be stored in the Receive Data Register. The RX Byte Count bits in the Status Register indicate the number of bytes available in the final data transfer. The firmware may verify the received PEC upon detection of the End of Message interrupt.

7.9 Block Write-Block Read Process Call



The Block Read-Block Write Process Call protocol combines the Block Write and Block Read protocols, removing the stop condition between the two messages. The operation of the Master is similar to a Block Write operation. Loading the block length into the byte count bits of the Master Control Register provides the length of the Block Write portion of the message. In addition, the PRC_CALL bit within the Master Control Register must be enabled. Upon completion of the Block Write part of the message, the PMBus Interface will automatically issue a Repeated Start condition on the PMBus and start transmission of the Block Read portion of the message. Operation of the PMBus Interface after the Repeated Start condition is the same as would be in a simple Block Read Message.

7.10 Alert Response



The Alert Response Message is utilized when the Master detects an alert condition from a Slave on the PMBus. In Master mode, the Alert Response Message is simply a Receive Byte message with PEC disable and the Slave Address set to 0xC (Alert Response Address). The PMBus Interface detects the Alert condition on an input and interrupts the firmware indicating the assertion of an alert condition (Slave desires to communicate with Master). Programming the Master Control Register with the Alert Response Address, initiates the Alert Response message and provides the device address of the Slave requesting service. The device address will be found in the Receive Data Register following receipt of the EOM interrupt.

7.11 Extended Command – Write Byte/Word, Read Byte/Word

S	Device Addr	Wr	A	Command Byte	A	Ext Command Byte	A	Sr	Device Addr	Wr	A
---	-------------	----	---	--------------	---	------------------	---	----	-------------	----	---

Byte #0	A	P
---------	---	---

Extended Command Write Byte w/o PEC Byte

S	Device Addr	Wr	A	Command Byte	A	Ext Command Byte	A	Sr	Device Addr	Wr	A
---	-------------	----	---	--------------	---	------------------	---	----	-------------	----	---

Byte #0	A	PEC	A	P
---------	---	-----	---	---

Extended Command Write Byte with PEC Byte

S	Device Addr	Wr	A	Command Byte	A	Ext Command Byte	A	Sr	Device Addr	Wr	A
---	-------------	----	---	--------------	---	------------------	---	----	-------------	----	---

Byte #0	A	Byte #1	A	P
---------	---	---------	---	---

Extended Command Write Word w/o PEC Byte

S	Device Addr	Wr	A	Command Byte	A	Ext Command Byte	A	Sr	Device Addr	Wr	A
---	-------------	----	---	--------------	---	------------------	---	----	-------------	----	---

Byte #0	A	Byte #1	A	PEC	A	P
---------	---	---------	---	-----	---	---

Extended Command Write Word with PEC Byte

S	Device Addr	Wr	A	Command Byte	A	Ext Command Byte	A	Sr	Device Addr	Rd	A
---	-------------	----	---	--------------	---	------------------	---	----	-------------	----	---

Byte #0	NA	P
---------	----	---

Extended Command Read Byte w/o PEC Byte

S	Device Addr	Wr	A	Command Byte	A	Ext Command Byte	A	Sr	Device Addr	Rd	A
---	-------------	----	---	--------------	---	------------------	---	----	-------------	----	---

Byte #0	A	PEC	NA	P
---------	---	-----	----	---

Extended Command Read Byte with PEC Byte

S	Device Addr	Wr	A	Command Byte	A	Ext Command Byte	A	Sr	Device Addr	Rd	A
---	-------------	----	---	--------------	---	------------------	---	----	-------------	----	---

Byte #0	A	Byte #1	NA	P
---------	---	---------	----	---

Extended Command Read Word w/o PEC Byte

S	Device Addr	Wr	A	Command Byte	A	Ext Command Byte	A	Sr	Device Addr	Rd	A
---	-------------	----	---	--------------	---	------------------	---	----	-------------	----	---

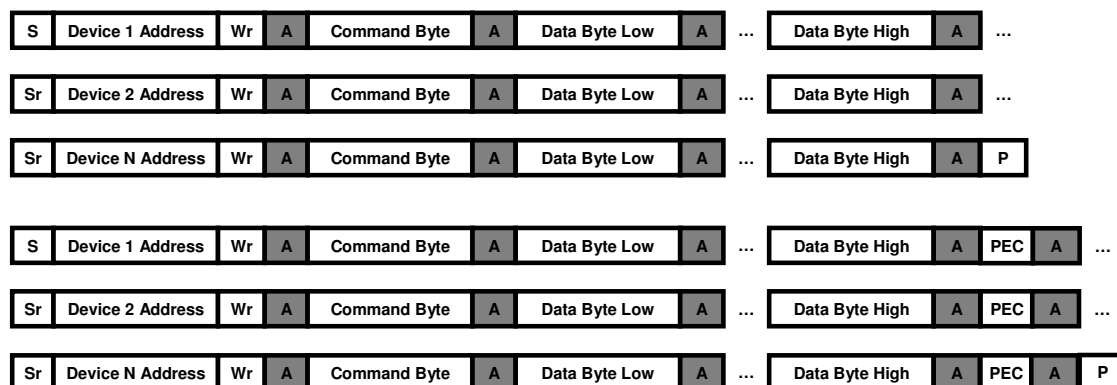
Byte #0	A	Byte #1	A	PEC	NA	P
---------	---	---------	---	-----	----	---

Extended Command Read Word with PEC Byte

The PMBus Interface provides support for extended commands which allow for an extra 256 command codes. By asserting the EXT_CMD bit within the Master Control Register, two command bytes are transmitted on the message protocol. Extended commands can be added to the Read Byte, Read Word, Write Byte and Write Word protocols. Operation of the PMBus interface in extended command mode is similar to these formats. In programming the write data or first part of the read message, the second command byte is loaded into Bits 15-8 of the Transmit Data Register with the remaining data bytes. The remaining operation of the PMBus is identical to the previous protocols, except for the inclusion of a Repeated Start condition and slave address in the write messages. No support is required by firmware for

these additional bytes in the write messages. The interface will interpret the EXT_CMD bit and make the appropriate format changes.

7.12 Group Command



The PMBus Interface must support the Group Command Protocol. The Group Command Protocol is used to send commands to more than one device within the same message. When devices on the PMBus detect the stop condition at the conclusion of the Group Command message, the received commands are executed concurrently. To initiate a Group Command, the GRP_CMD bit within the Master Control Register must be set when programming the slave address for the first device in the message. The rest of the message is processed as a write byte/word message. At the conclusion of the first part of the Group Command message, the firmware programs the next device address in the Master Control Register. The PMBus Interface will initiate a repeated start on the bus and start the next part of the message. When programming the last device address of the Group Command message, the firmware must disable the GRP_CMD bit when programming the Master Control Register.

8 Slave Mode Operation Reference

In addition to supporting Master capability, the PMBus Interface also provides support to act as a slave on the PMBus. Configuration of the slave mode is accomplished through the programming of the Slave Control Register. The interface supports automatic and manual acknowledgement of the address and command bytes. Receive data from the master is read from the Receive Data Register, as performed in the master mode. Transmit data is loaded into the Transmit Data Register. The Status Register provides updated info on data ready for reading or data requests for data to transmit over the PMBus to the Master.

8.1 Slave Address Acknowledgement

The PMBus Interface in Slave Mode supports two modes of address acknowledgement, automatic and manual. Automatic mode provides acknowledgement of the received device address without direct firmware control. Manual mode requires the firmware to program the Acknowledge Register to accept a received device address. The address acknowledgement mode is set by programming bit 7 of the Slave Control Register. A '0' enables automatic mode, while a '1' sets the PMBus Interface into manual address acknowledgement mode.

In utilizing automatic address acknowledgement mode, the firmware must program the slave address and slave mask within the Status Control Register. The slave address bits (Bits 6-0) configure the slave address of the PMBus Interface in slave mode. Upon receipt of this address from the Master, the interface will acknowledge the received address and continue processing the message. To provide capability to answer to a wider range of slave addresses, the slave mask bits (Bits 14-8) have been provided. A '1' in any bit of the slave mask requires the corresponding bit of the received address to match the programmed slave address. A '0' allows for a slave address match for the corresponding bit for any bit. After reset, the slave mask defaults to all ones, meaning the received slave address must exactly match the programmed slave address. The programmed slave address defaults to 7Ch.

After configuring the slave address and mask, the PMBus interface will respond to receive device addresses based on the programmed settings. An acknowledge enable or not acknowledge is executed without control from the firmware. To verify the address acknowledgement, the Hold Slave Address register stores the received device address in automatic mode. The firmware can read the register to confirm the interface is operating successfully or to determine which device address was received when the slave mask has been modified to receive multiple addresses.

Manual address mode requires the firmware to read the received slave address and issue an acknowledgement write if the received address equals the slave address. After the PMBus interface receives a device address from the Master in manual mode, an interrupt is generated and the SA_RDY bit within the Status Register is enabled. The firmware can read the received address from Bits 7-0 of the Receive Data Register. If the address matches the slave address expected by the firmware, a '1' is written to the Acknowledge Register. Otherwise, the firmware writes a '0' to the Acknowledge Register. A write to the Acknowledge register is required in manual mode, since the PMBus Interface will hold the PMBus clock low until the firmware responds to the received slave address.

8.2 Command Byte Acknowledgement

The PMBus Interface in Slave Mode provides the capability to manually acknowledge the received command byte. In default mode, the interface will automatically acknowledge all received command bytes and store the byte(s) into the Receive Data Register. The firmware reads the command byte with the other data bytes received from the Master. In manual mode, a data ready interrupt is enabled by the PMBus Interface after receiving the command byte. The firmware must read the command byte from Bits 7-0 of the Receive Data Register and acknowledge the command byte by programming the Acknowledge

Register. In manual mode, it is possible to write a 0 to the Acknowledge register and cause the hardware to NACK the command byte.

8.3 Receive Byte Configuration

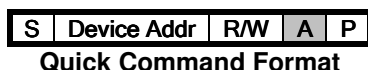
The PMBus Interface in Slave Mode provides the capability to configure the number of bytes to receive before issuing a data ready interrupt. Bits 22:21 of the PMBus Slave Control Register configure the number of bytes to receive in the PMBus Receive Data Buffer before alerting the firmware. A setting of "00" forces the firmware to acknowledge every byte received by the PMBus slave. Setting bits 22:21 (RX_BYTE_ACK_CNT) to "01" requires the firmware to acknowledge after every 2 received bytes. The default for slave operation is "11", which provides full capability of the receive buffer and limits servicing by the firmware. The following discussions on the read capability in slave mode assume a setting of "11" for the RX_BYTE_ACK_CNT bits.

Acknowledgement is done as always by writing to the Acknowledge Register.

8.4 Message Acknowledgement

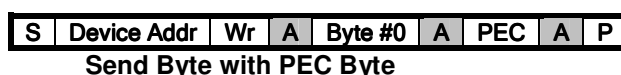
The PMBus Interface in Slave Mode requires the firmware to acknowledge the receipt of the EOM on each message from the PMBus Master. Acknowledgement of each message is performed by writing a '1' to the Acknowledge Register following the receipt of the EOM bit (Bit 5) in the Status Register. Failure to properly acknowledge each message will disable the slave from acknowledging its device address on the next message over the PMBus. Writing a 0 to the Acknowledge Register after an EOM will not cause a NACK of the current message because that message is already complete. Instead a NACK will be saved up for the next message.

8.5 Quick Command



Quick commands received by the PMBus Interface in Slave mode require a simple acknowledgement of the received device address. In automatic address acknowledge mode (See Section 8.1), the interface processes the quick command without firmware interaction. Upon receipt of the end of message, the firmware has the option to read the received address in the Hold Slave Address Register. In manual address acknowledge mode, the address is acknowledged through firmware control, as discussed in Section 5.1 earlier.

8.6 Send Byte

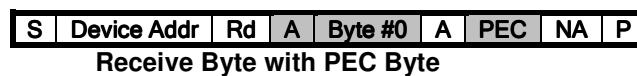
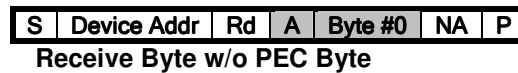


A Send Byte message consists of the device address, a single data byte and an optional PEC byte. To process the PEC byte correctly, PEC processing needs to be enabled in the Slave Control Register. In automatic address acknowledge mode, the data and optional PEC byte are acknowledged without firmware interaction. The firmware receives an End of Message Interrupt, reads the status register and finds the data ready indication bit set. In manual mode, the address is acknowledged by the firmware, while the remaining data and PEC bytes are acknowledged always by the interface.

The PMBus Interface stores Data Byte #0 into the Receive Data Register. The data byte will be stored into bits 7-0. In non-PEC mode, the Rx Byte Count in the Status Register will indicate one byte received.

If PEC processing is enabled, the PEC byte is also stored into the Receive Data Register, with the PEC byte residing in bits 15-8. The Rx Byte Count in the Status Register will indicate two bytes received. The PEC Valid bit in the Status Register indicates the validity of the received PEC byte.

8.7 Receive Byte



A Receive Byte message consists of the device address, a single data byte and an optional PEC byte. In automatic address acknowledge mode, the firmware receives a data request interrupt following reception of the slave address. The data byte to be sent to the Master is stored into bits 7-0 of the Transmit Data Register and Transmit Byte Count bits within the Slave Control Register is set to a value of one. If PEC processing is enabled, the Transmit PEC bit (Bit 19) within the Slave Control Register is set to '1', along with the Enable PEC bit (Bit 15). The interface will automatically append the calculated PEC byte at the completion of the message.

8.8 Write Byte/Word



Write Byte w/o PEC Byte



Write Byte with PEC Byte



Write Word w/o PEC Byte



Write Word with PEC Byte

The Write Byte and Write Word messages consist of a slave address, a command word, transmitted data bytes and an optional PEC byte. In automatic address acknowledge mode, the data bytes and optional PEC byte are acknowledged without firmware interaction. The acknowledgement of the command word is configured through the Slave Control Register. The firmware receives an End of Message Interrupt in all cases except for Write Word with PEC message, reads the status register and finds the data ready indication bit set.

In the case of a Write Word with PEC byte message, the data ready interrupt is enabled after receiving 4 bytes (Command Byte, the 2 data bytes and the PEC byte). The firmware reads the data from the Receive Data Register and must write the Acknowledge Register to acknowledge back to the master. The PMBus Interface holds the PMBus until the firmware responds to the received data.

In all other cases, the EOM interrupt is received and data can be read from the Receive Data Register. The firmware is not required to send an acknowledgement back to the Master.

8.9 Read Byte/Word

S	Device Addr	Wr	A	Command	A	Sr	Device Addr	Rd	A	Byte #0	NA	P
---	-------------	----	---	---------	---	----	-------------	----	---	---------	----	---

Read Byte w/o PEC Byte

S	Device Addr	Wr	A	Command	A	Sr	Device Addr	Rd	A	Byte #0	A	PEC	NA	P
---	-------------	----	---	---------	---	----	-------------	----	---	---------	---	-----	----	---

Read Byte with PEC Byte

S	Device Addr	Wr	A	Command	A	Sr	Device Addr	Rd	A	Byte #0	A
---	-------------	----	---	---------	---	----	-------------	----	---	---------	---

Byte #1	NA	P
---------	----	---

Read Word w/o PEC Byte

S	Device Addr	Wr	A	Command	A	Sr	Device Addr	Rd	A	Byte #0	A
---	-------------	----	---	---------	---	----	-------------	----	---	---------	---

Byte #1	A	PEC	NA	P
---------	---	-----	----	---

Read Word with PEC Byte

The Read Byte and Read Word messages consist of a slave address, a command word, received data bytes from a slave and an optional PEC byte. Address and Command acknowledgement is configured through the Slave Control Register. In automatic mode, following receipt of the repeated start and slave address, the PMBus Interface provides a data ready and data request interrupt. The received command byte is found in bits 7-0 of the Receive Data Register. The firmware responds to the data request by programming the data bytes into the Transmit Data Register and the TX Byte Count bits in the Slave Control Register. If PEC processing is enabled, the Transmit PEC bit should also be asserted. An EOM interrupt indicates completion of the message to the Master.

8.10 Process Call

S	Device Addr	Wr	A	Command	A	Byte #0	A	Byte #1	A	Sr	Device Addr	Rd	A
---	-------------	----	---	---------	---	---------	---	---------	---	----	-------------	----	---

Byte #0	A	Byte #1	NA	P
---------	---	---------	----	---

Process Call w/o PEC Byte

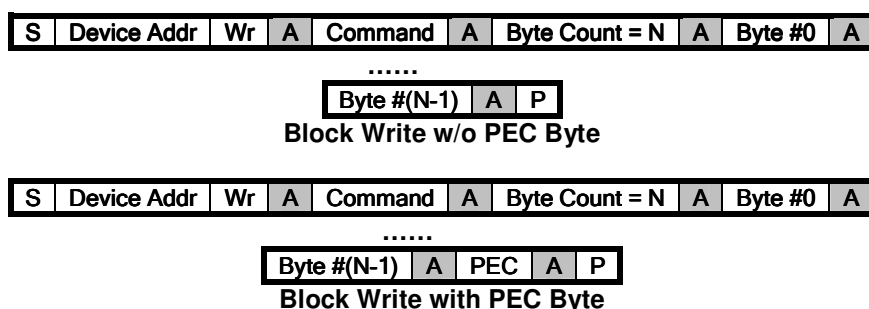
S	Device Addr	Wr	A	Command	A	Byte #0	A	Byte #1	A	Sr	Device Addr	Rd	A
---	-------------	----	---	---------	---	---------	---	---------	---	----	-------------	----	---

Byte #0	A	Byte #1	A	PEC	NA	P
---------	---	---------	---	-----	----	---

Process Call with PEC Byte

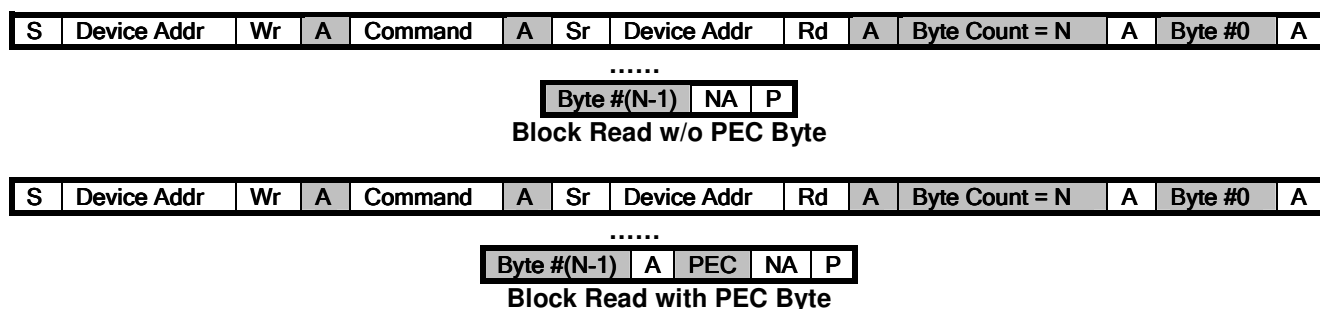
The Process Call protocol consists of a Write Word message, followed by a Read Word message, without a stop condition between the two messages. Address and Command acknowledgement is configured through the Slave Control Register. In automatic mode, following receipt of the repeated start and slave address, the PMBus Interface provides a data ready and a data request interrupt. The repeated start bit is set in the Status Register to indicate the receipt of the first part of the Process Call message. The received command byte is found in bits 7-0 of the Receive Data Register, while the two data bytes received from the Master can be found in bits 23-8. Upon receipt of the repeated start and a data request from the interface, the firmware programs the Transmit Data Register with the 2 data bytes to be sent to the Master. If PEC processing is enabled, the Transmit PEC bit within the Slave Control Register is asserted. The EOM interrupt will indicate the read word portion of the Process Call message has been completed by the interface.

8.11 Block Write



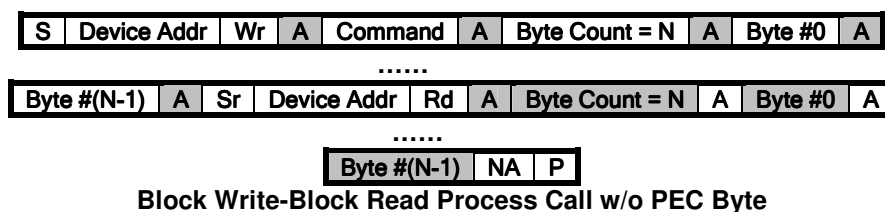
The Block Write protocol is similar to a Write Word in its structure, with the exception of transmission of more than 2 data bytes in the message. Following the receipt of the command byte, the block length and 2 data bytes, the PMBus Interface provides a data ready interrupt. The interface waits for the firmware to read the received data and program the acknowledge register. While waiting from acknowledgement from the firmware, the interface will drive the clock line low, stalling the bus. The data ready interrupts will continue for the duration of the message at a frequency of every 4 data bytes. The number of bytes received can be found within the Status Register. At the end of the message, less than 4 bytes may be stored in the Receive Data Register. The PEC Valid can be checked to determine if the received PEC value is accurate.

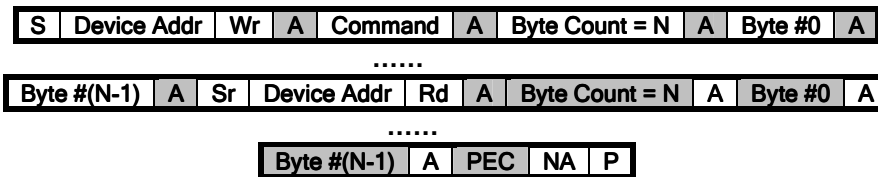
8.12 Block Read



The Block Read protocol is similar to a Read Word in its structure, with the exception of reception of more than 2 data bytes from the Slave. Following the receipt of the repeated slave address, a data ready and data request interrupt is generated by the PMBus Interface. The command byte received from the Master can be found in bits 7-0 of the Receive Data Register. The PMBus is stalled until the firmware programs data bytes into the Transmit Data Register. The firmware is required to load the block length into bits 7-0 of the Transmit Data Register during the initial programming of the register. After 4 bytes have been transmitted, the interface will issue a data request interrupt and stall the PMBus until the firmware has programmed additional data into the Transmit Data Register.

8.13 Block Write-Block Read Process Call

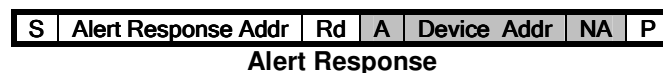




Block Write-Block Read Process Call with PEC Byte

The Block Read-Block Write Process Call protocol combines the Block Write and Block Read protocols, removing the stop condition between the two messages. The processing of the Block Read-Block Write Process Call message is similar to the mode of operation for the Process Call message. After acknowledgement of the address and command bytes, the PMBus Interface generates a data ready interrupt upon detection of 4 data bytes or a repeated start condition. After receiving the repeated start, the firmware will be required to load transmit data to send to the Master. Bits 7-0 of the initial programming of the Transmit Data Register must represent the byte count of the block data sent to the Master.

8.14 Alert Response



Alert Response

The Alert Response Message is utilized when the Master detects an alert condition from a Slave on the PMBus. In automatic address acknowledge mode, upon detection of the Alert Response Address, the PMBus Interface provides an acknowledgement to the Master and sends the programmed slave address within the Slave Control Register. The interface only responds to the message if the Alert En within Miscellaneous Control Register has been previously set. After receiving the Alert Response message, the interface will clear the Alert condition and enable bit within the Miscellaneous Register.

In manual address acknowledge mode, the firmware must read the received address from the Receive Data Register and transmit the desired slave address back to the Master. The Misc Control Register must be reprogrammed to disable the Alert En bit used to initiate the Alert Response message from the Master.

8.15 Extended Command – Read Byte/Word, Write Byte/Word



Extended Command Write Byte w/o PEC Byte



Extended Command Write Byte with PEC Byte



Extended Command Write Word w/o PEC Byte



Byte #0	A	Byte #1	A	PEC	A	P
---------	---	---------	---	-----	---	---

 Extended Command Write Word with PEC Byte

S	Device Addr	Wr	A	Command Byte	A	Ext Command Byte	A	Sr	Device Addr	Rd	A
---	-------------	----	---	--------------	---	------------------	---	----	-------------	----	---

Byte #0	NA	P
---------	----	---

 Extended Command Read Byte w/o PEC Byte

S	Device Addr	Wr	A	Command Byte	A	Ext Command Byte	A	Sr	Device Addr	Rd	A
---	-------------	----	---	--------------	---	------------------	---	----	-------------	----	---

Byte #0	A	PEC	NA	P
---------	---	-----	----	---

 Extended Command Read Byte with PEC Byte

S	Device Addr	Wr	A	Command Byte	A	Ext Command Byte	A	Sr	Device Addr	Rd	A
---	-------------	----	---	--------------	---	------------------	---	----	-------------	----	---

Byte #0	A	Byte #1	NA	P
---------	---	---------	----	---

 Extended Command Read Word w/o PEC Byte

S	Device Addr	Wr	A	Command Byte	A	Ext Command Byte	A	Sr	Device Addr	Rd	A
---	-------------	----	---	--------------	---	------------------	---	----	-------------	----	---

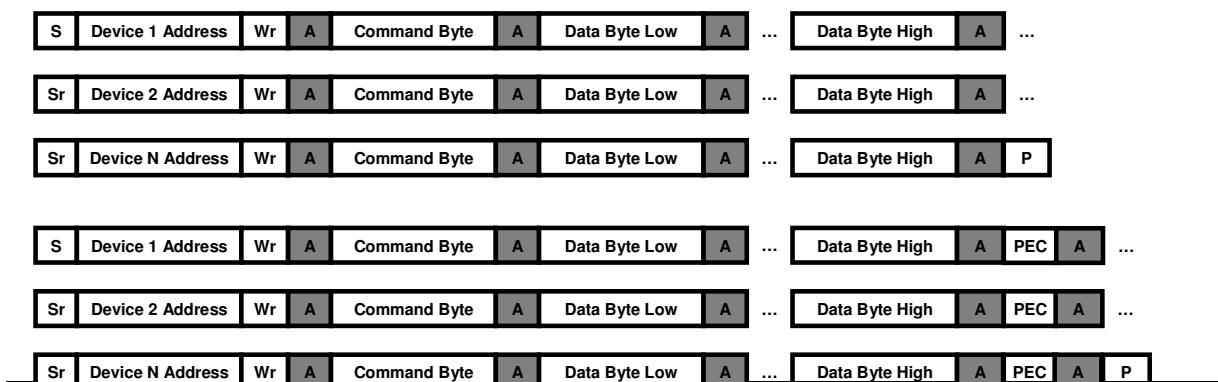
Byte #0	A	Byte #1	A	PEC	NA	P
---------	---	---------	---	-----	----	---

 Extended Command Read Word with PEC Byte

The PMBus Interface provides support for extended commands which allow for an extra 256 command codes. In Slave Mode, the PMBus Interface stores both command words within the Receive Data Register, along with the data bytes. In recognizing the extended command messages, the Repeated Start bit and the Rd Byte Count Bits within the Status Register are utilized. For Extended Command Write Byte/Write Word messages, the two command bytes are stored in bits 15-0 of the Receive Data Register. The initial command byte should hold the command extension code, representing utilization of the extended command protocol. The Repeated Start bit is also set, received after the retransmission of the device address. The Rd Byte Count equals three for an Ext Cmd Write Byte message and four for a Ext Cmd Write Word message.

For the Extended Command Read Byte/Read Word messages, the interface will generate a data ready and data request interrupt following reception of the repeated device address. The two command bytes will be found in bits 15-0 of the Receive Data Register, with the initial command byte matching the command extension code. The firmware will be required to load transmit data to complete the message back to the Master.

8.16 Group Command



The PMBus Interface must support the Group Command Protocol. The Group Command Protocol is used to send commands to more than one device within the same message. When devices on the PMBus detect the stop condition at the conclusion of the Group Command message, the received commands are executed concurrently. Following address and command acknowledgement, the interface provides a data ready interrupt upon detection of 4 data bytes or the transmission of a repeated start on the PMBus. The firmware should wait for the EOM interrupt before processing the received command, as required by the use of the Group Command message.