# AxiCat

## Serial Protocol v1.3

Specification

*September 2016*

# Table of Contents

# Revision History

| Date | Authors | Description |
|---|---|---|
| 2014-07-13 | Peter S'heeren | Initial release v1.0. |
| 2014-09-17 | Peter S'heeren | Corrected section UART Set Rx Timeout.<br>Added N field in section TWI Slave Transmit.<br>Added special function 1-Wire.<br>Protocol v1.1.0. |
| 2014-09-28 | Peter S'heeren | Added 1-Wire enumeration.<br>Protocol v1.2.0. |
| 2014-12-12 | Peter S'heeren | Added 1-Wire probing.<br>Protocol v1.3.0. |
| 2016-09-13 | Peter S'heeren | Changed this document from technical note (TN001) to specification.<br>Serial protocol v1.3. |

# 1 Overview

The AxiCat is a multipurpose I/O card. The 26-pin interface features seventeen GPIO pins, two UARTs, an I2C bus and a SPI bus. The I/O can operate at 3.3 V and 5 V levels.

The microcontroller on the AxiCat contains the main application. The main application accepts incoming command packets and issues response packets over the USB interface. Command packets are translated to activities on the I/O interface or other actions like setting parameters. Response packets are sent when the I/O interface has completed activities or status must be reported.

Command and response packets and their behavior are part of the serial protocol. This document specifies the serial protocol. The main application of the AxiCat implements the serial protocol according to this specification.

# 2 Serial Port Settings

The AxiCat incorporates an FTDI FT245 USB-to-FIFO chip. Once enumerated, the USB function will appear as a serial port in your operating system. Linux and Windows operating systems assign a serial path to each serial port. You need to determine the serial path in order to open a handle to the serial port, choose serial settings and perform I/O with the AxiCat.

The USB function is an 8-byte full-duplex FIFO. It doesn't incorporate a physical serial interface. As such it doesn't need to be configured as a serial port.

| Attribute | Value |
|---|---|
| Baud rate | 9600[1] |
| Data bits | 8 |
| Stop bits | 1 |
| Parity | NONE |
| Handshaking | NONE |

How serial settings are handled specifically for the FT245 greatly depends on the driver that controls the FT245. In doubt, you're advised to set the serial settings shown in the table.

[1] Note that the baud rate has no influence on the throughput of the FT245. The given value is a standard baud rate, all operating systems should accept it as a valid serial setting.

# 3 Protocol

## Connection Model

The serial connection requires a point-to-point connection model. Each point is defined as follows:

- Host: The side of the link that communicates with the AxiCat. The host typically is a computer system running a program that controls the serial port.
- Device: The AxiCat.

## Data Direction

Data flows in two directions:

- Host-to-device (H►D): The host transmits data to the device.
- Device-to-host (D►H): The device transmits data to the host.

## Data Words

All data is transferred as 8-bit words. In this document, a data word is referred to as a data byte.

## Data Integrity

The data path between the host and the device is considered error-free. Each byte sent on one side of the connection will be delivered error-free to the other side. The protocol doesn't employ any error checking on the transmitted data.

The USB cable may be subject to data corruption. Since the FT245 relies on so-called bulk transfers, USB will try to correct the data transmission. If the data corruption can't be fixed, the USB function will be detached from the bus and the host will receive an error status.

The device checks the format of incoming command packets. If the host sends an ill-formatted command packet, the device will reject the packet. However, this is considered a bug on the host side and must never occur.

## Data Synchronization

The device will accept all incoming data bytes at any time. There's no timeout between the reception of data bytes.

The host mustn't apply a timeout between data bytes it receives from the device.

There's no concept of control and data bits or words. All bytes that travel between host and device in either direction represent data.

The recommendations for synchronization between host and device are:

- When the host connects to the device, it implements the initialization procedure as described later on.
- The host processes each incoming data byte from the device. The host breaks the

connection as soon as one invalid data byte is received.

- The host breaks the connection when the serial port reports an error. This situation typically occurs when the USB cable is unplugged.

## Packets

Both host and device transmit packets. The host transmits command packets, the device transmits response packets.

The length of a data packet is one or more data bytes. The format is fully determined by the state of the data bytes in the packet.

All packets start with a code. A command packet embeds an 8-bit command code that occupies the entire first data byte. A response packet embeds a 6-bit response code in the lower bits of the first data byte.

### Command Packets

| Code | Label | Type |
|---|---|---|
| 00h | CMD_GEN_NOP | S |
| 01h | CMD_GEN_INFO | S |
| 02h | CMD_GEN_VERSION | S |
| 04h | CMD_GPIO_SET_DIR | S |
| 05h | CMD_GPIO_WRITE | S |
| 06h | CMD_GPIO_READ | S |
| 08h | CMD_UART0_SET_DATA_BITS | S |
| 09h | CMD_UART1_SET_DATA_BITS | S |
| 0Ah | CMD_UART0_SET_STOP_BITS | S |
| 0Bh | CMD_UART1_SET_STOP_BITS | S |
| 0Ch | CMD_UART0_SET_BAUDRATE | S |
| 0Dh | CMD_UART1_SET_BAUDRATE | S |
| 0Eh | CMD_UART0_SET_BAUDRATE_RAW | S |
| 0Fh | CMD_UART1_SET_BAUDRATE_RAW | S |
| 10h | CMD_UART0_ENABLE | S |
| 11h | CMD_UART1_ENABLE | S |
| 12h | CMD_UART0_DISABLE | S |
| 13h | CMD_UART1_DISABLE | S |
| 14h | CMD_UART0_SET_RX_TIMEOUT | S |
| 15h | CMD_UART1_SET_RX_TIMEOUT | S |
| 16h | CMD_UART0_TX | A |
| 17h | CMD_UART1_TX | A |
| 20h | CMD_TWI_SET_SPEED | S |
| 21h | CMD_TWI_SET_SPEED_RAW | S |

| Code | Label | Type |
|---|---|---|
| 22h | CMD_TWI_ENABLE | S |
| 23h | CMD_TWI_DISABLE | S |
| 24h | CMD_TWI_MASTER_START | A |
| 25h | CMD_TWI_MASTER_STOP | A |
| 26h | CMD_TWI_MASTER_TX | A |
| 27h | CMD_TWI_MASTER_RX | A |
| 28h | CMD_TWI_SLAVE_ENABLE | S |
| 29h | CMD_TWI_SLAVE_DISABLE | S |
| 2Ah | CMD_TWI_SLAVE_TX | A |
| 2Bh | CMD_TWI_SLAVE_RX | A |
| 2Ch | CMD_TWI_CLEAR | S |
| 30h | CMD_SPI_SET_SPEED | S |
| 31h | CMD_SPI_SET_SPEED_RAW | S |
| 32h | CMD_SPI_SET_CFG | S |
| 33h | CMD_SPI_ENABLE | S |
| 34h | CMD_SPI_DISABLE | S |
| 35h | CMD_SPI_XFR | A |
| 38h | CMD_OW_ENABLE | S |
| 39h | CMD_OW_DISABLE | S |
| 3Ah | CMD_OW_RESET | A |
| 3Bh | CMD_OW_TOUCH_BITS | A |
| 3Ch | CMD_OW_ENUM | A |
| 3Dh | CMD_OW_PROBE | A |

There are two types of commands:

- Synchronous (S): The command is executed as soon as it arrives in the device. A response, if defined, is returned immediately. The response is also called synchronous.
- Asynchronous (A): The command is stored in a buffer and executed as a separate task. When an asynchronous command has completed the device sends an asynchronous response, if one is defined. An asynchronous command may spawn unsolicited responses.

### Response Packets

| Code | Label | Type |
|---|---|---|
| 01h | RSP_GEN_INFO | S |
| 02h | RSP_GEN_VERSION | S |
| 06h | RSP_GPIO_READ | S |
| 14h | RSP_UART0_TX_FREE | U |
| 15h | RSP_UART1_TX_FREE | U |
| 16h | RSP_UART0_RX | U |
| 17h | RSP_UART1_RX | U |
| 22h | RSP_TWI_BUS_ERROR | U |
| 23h | RSP_TWI_ARB_LOST | U |
| 24h | RSP_TWI_MASTER_START | A |

| Code | Label | Type |
|---|---|---|
| 25h | RSP_TWI_MASTER_STOP | A |
| 26h | RSP_TWI_MASTER_TX | A |
| 27h | RSP_TWI_MASTER_RX | A |
| 2Ah | RSP_TWI_SLAVE_TX | A |
| 2Bh | RSP_TWI_SLAVE_RX | A |
| 35h | RSP_SPI_XFR | A |
| 3Ah | RSP_OW_RESET | A |
| 3Bh | RSP_OW_TOUCH_BITS | A |
| 3Ch | RSP_OW_ENUM | A |
| 3Dh | RSP_OW_PROBE | A |

There are three types of responses:

- Synchronous (S): The response from a synchronous command.
- Asynchronous (A): The response from an asynchronous command that has completed.
- Unsolicited (U): The response has no corresponding command.

## *Buffer Management*

All asynchronous commands are stored in buffers. The device maintains several buffers. If an incoming command doesn't fit in the corresponding buffer, either the command blocks further reception of other commands or the command is discarded and no response is returned. As such the host must track the free space of the various buffers as to perform buffer management.

During initialization, the host must query the information structure. This structure contains the sizes of all buffers. Once the host has received the information, buffer management can begin. The host maintains a free bytes counter for each buffer.

Before the host sends an asynchronous command, it must first check whether the command will fit in the free space of the corresponding buffer. If the command won't fit, the host waits, else the host sends the command to device and decreases the buffer's free bytes counter.

When the device sends an asynchronous response, the host knows the corresponding command has completed and increases the buffer's free bytes counter. In case of the UARTs, the device sends an unsolicited response containing the free bytes counter of the

transmit buffer.

| Buffer | Related Packets | Description |
|---|---|---|
| all | CMD_GEN_INFO<br>RSP_GEN_INFO | During initialization, the host queries the information structure. |
| BUF_UART0_TX | CMD_UART0_TX<br>RSP_UART0_TX_FREE | Transmit buffer of UART0. The host writes data words to the buffer, the device transmits the data words and regularly reports the free buffer space. |
| BUF_UART1_TX | CMD_UART1_TX<br>RSP_UART1_TX_FREE | Transmit buffer of UART1. The host writes data words to the buffer, the device transmits the data words and regularly reports the free buffer space. |
| BUF_TWI_M | CMD_TWI_MASTER_START<br>RSP_TWI_MASTER_START<br>CMD_TWI_MASTER_STOP<br>RSP_TWI_MASTER_STOP<br>CMD_TWI_MASTER_TX<br>RSP_TWI_MASTER_TX<br>CMD_TWI_MASTER_RX<br>RSP_TWI_MASTER_RX | This buffer contains all I2C master commands. |
| BUF_TWI_STX | CMD_TWI_SLAVE_TX<br>RSP_TWI_SLAVE_TX | This buffer contains all I2C slave data transmission commands. |
| BUF_TWI_SRX | CMD_TWI_SLAVE_RX<br>RSP_TWI_SLAVE_RX | This buffer contains all I2C slave data reception commands. |
| BUF_SPI | CMD_SPI_XFR<br>RSP_SPI_XFR | This buffer contains all SPI transfer commands. |
| BUF_OW | CMD_OW_RESET<br>RSP_OW_RESET<br>CMD_OW_TOUCH_BITS<br>RSP_OW_TOUCH_BITS<br>CMD_OW_ENUM<br>RSP_OW_ENUM<br>CMD_OW_PROBE<br>RSP_OW_PROBE | This buffer contains all 1-Wire commands. |

The number of bytes a command occupies in a buffer is NOT equal to the length of the command packet. Each buffer stores commands in a specific way. The actual number of bytes required for each command is explained in the related sections.

When the device receives an asynchronous command that doesn't fit in the corresponding buffer, the device will discard the command, except for commands CMD_UART0_TX and CMD_UART1_TX. In the latter case, the device will block reception from the FT245 until the data is transmitted over the TXD line of the UART. As a result, the CMD_UARTn_TX command will choke the transfer of any further commands to the device. On the other hand, this behavior allows the host to send and queue up many CMD_UARTn_TX commands for a single UART if the host doesn't need to transmit any other commands.

## *USB Remote Wake-up*

The host can suspend attached devices to save on power consumption. Suspending devices is usually part of the power policy of the operating system, drivers and other software components. Many power saving scenarios are possible. The host may suspend individual devices or put the entire system in sleep mode.

USB demands that a USB function starts transitioning to the suspended state when it

detects 3 ms of idle state (no activity) on the USB bus. How and when this condition occurs is entirely up to the host system.

The USB function will resume from the suspended state when one of the following events occurs:

- The host resumes the USB function.
- The main application instructs the USB function to issue a resume signal to the USB controller. This is called USB remote wake-up.

The microcontroller isn't affected by the suspended state of the USB function meaning the main application keeps running. The main application triggers a USB remote wake-up under any of these conditions:

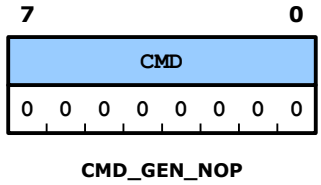
- It needs to send an asynchronous response to the host.
- It needs to send an unsolicited response to the host.

# *Generic*

## No-operation

The no-operation command packet occupies one byte. There's no corresponding response packet.

This command is part of the initialization procedure. It allows the host to flush out a possibly incomplete command packet that's being received by the device.

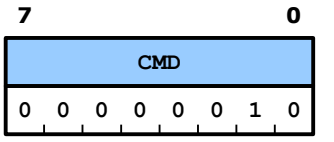



**CMD_GEN_NOP**

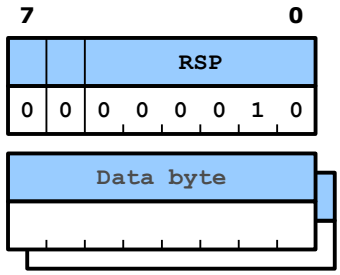
## Get Version

This command returns a string that denotes the version of the main application (also known as the firmware version).

The character set used in the string is ASCII. The string is zero-terminated. The last data byte is zero.




**CMD_GEN_VERSION**

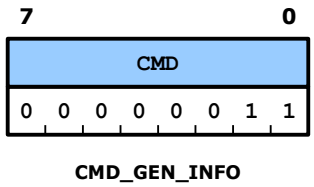



**RSP_GEN_VERSION**

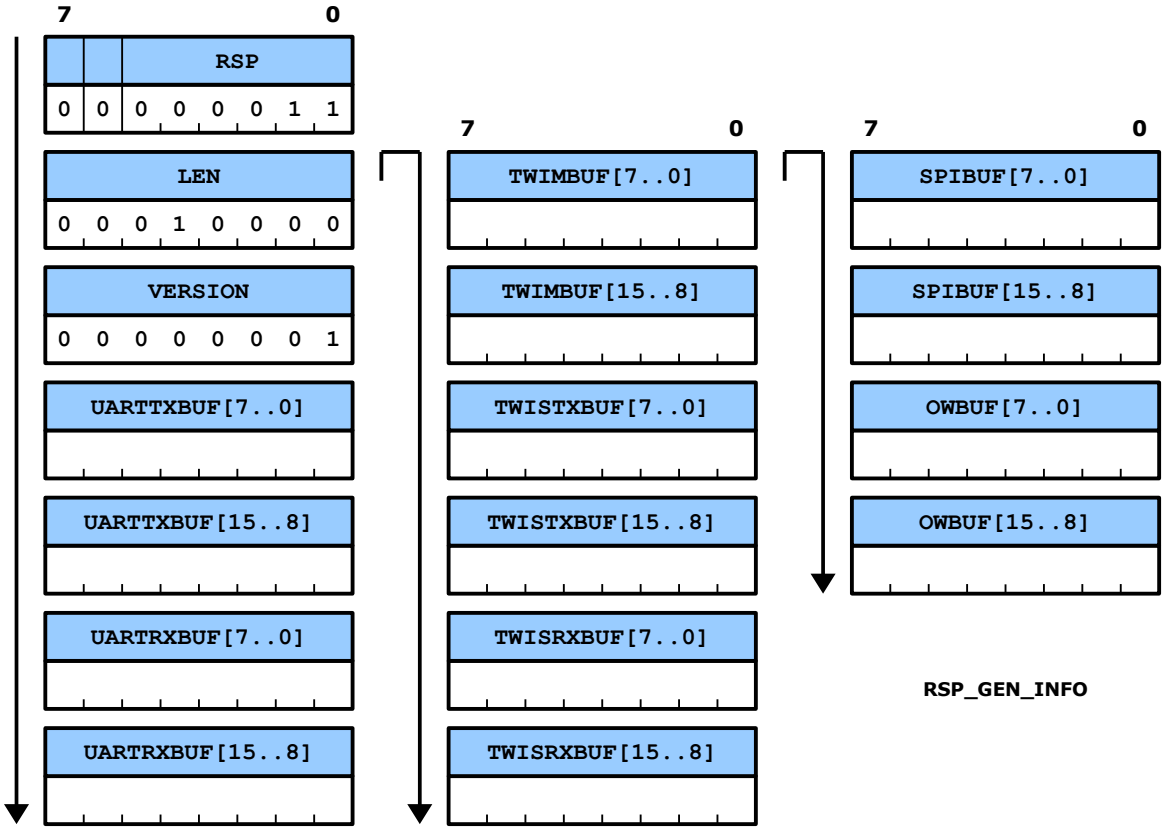
## Get Information

This command queries the information data.

The host needs this information for retrieving the firmware version and successfully handling buffer management.




CMD_GEN_INFO




RSP_GEN_INFO

Meaning of the fields:

- LEN: Length of information structure (bytes). This value includes all bytes except for the preceeding response code.
- VERSION: Firmware version.
- UARTTXBUF[15..0]: Size of UART0 and UART1 transmit buffer (bytes minus one).
- UARTRXBUF[15..0]: Size of UART0 and UART1 receive buffer (bytes minus one).
- TWIMBUF[15..0]: Size of TWI master buffer (bytes minus one).
- TWISTXBUF[15..0]: Size of TWI slave transmit buffer (bytes minus one).
- TWISRXBUF[15..0]: Size of TWI slave receive buffer (bytes minus one).
- SPIBUF[15..0]: Size of SPI transfer buffer (bytes minus one).
- OWBUF[15..0]: Size of 1-Wire command buffer (bytes minus one).

The LEN value may be greater than indicated here. This allows future versions of the AxiCat main application to add functionality.

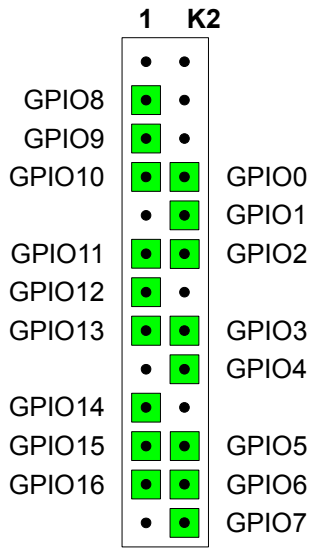
## *GPIO*

The device offers seventeen general-purpose input/output (GPIO) pins.

Most GPIO pins are shared with special functions (UART0, UART1, TWI, SPI). When a special function is enabled it overrides a group of GPIO pins. When the special function is disabled the group of GPIO pins is available again.

**Initial values**:

- All GPIO pins are active as none of the special functions are enabled after reset.
- All GPIO pins are initialized as input, active pull-up (output state set to one).

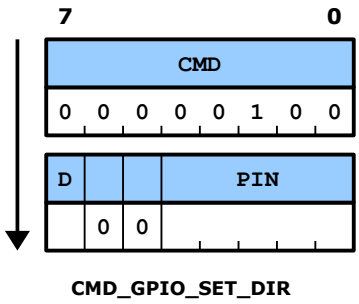



### Set Direction

This command sets the direction of the specified GPIO pin. Fields:

- PIN: Pin number (0..16).
- D: Direction: 0=input, 1=output.

The current output state immediately takes effect. The meaning depends on the direction of the GPIO pin.

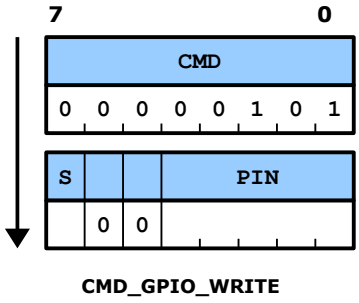



**CMD_GPIO_SET_DIR**

### Write Output State

This command sets the output state of the specified GPIO pin. The meaning depends on the direction of the GPIO pin. Fields:

- PIN: Pin number (0..16).
- S: Output state:
  - Input pin: 0=no pull-up, 1=pull-up active.
  - Output pin: Set output to logic 0 or logic 1.




**CMD_GPIO_WRITE**

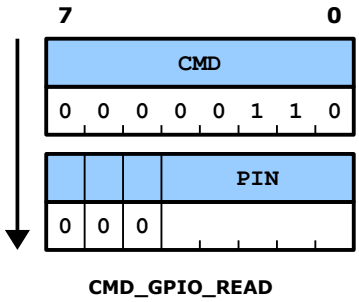
## Read Information

The read information command returns a response packet with information of the specified pin. Fields:

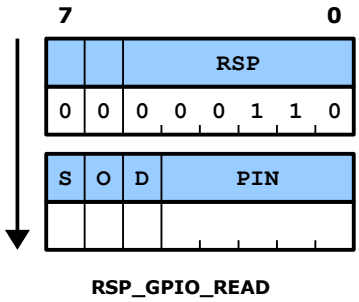
- PIN: Pin number (0..16).
- D: Direction: 0=input, 1=output.
- O: Output state (logic 0 or logic 1).
- S: Sensed state (logic 0 or logic 1).

Sensed state is the same as input state. The GPIO pin is sampled when the command is executed.

The response packet also includes the PIN field. This allows the host to issue multiple read information commands without the need to keep track of the order in which the commands were sent. This way the host can consider the commands as asynchronous.

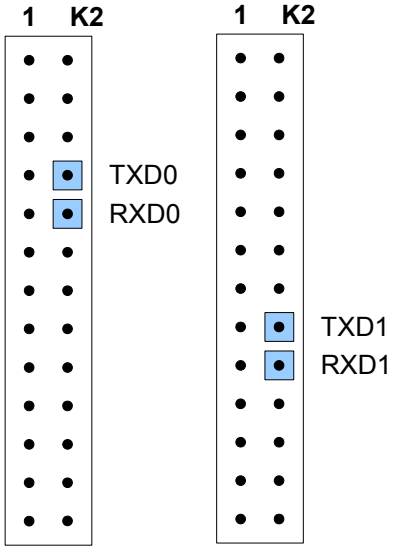



**CMD_GPIO_READ**




**RSP_GPIO_READ**

## *UART0 and UART1*




Special functions UART0 and UART1 provide a serial bus with full duplex operation. The two UARTs operate completely independent.

Serial settings data bits, stop bits, speed and raw speed can be set no matter the UART is enabled or disabled.

All command codes and response codes embed a 1-bit field (U) that specifies the UART:

- U=0: UART0 (TXD0, RXD0).
- U=1: UART1 (TXD1, RXD1).

When an UART is enabled, it overrides the group of GPIO pins as shown in the picture.

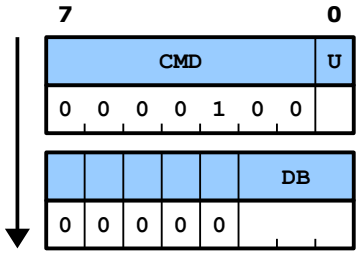
During the period an UART is enabled, all changes made to the direction and output state of the overridden GPIO pins must be deemed lost.

### Set Data Bits

This command sets the data bits of the specified UART.

**Initial value**: DB=011b

| DB | Data bits | DB | Data bits |
|---|---|---|---|
| 000b | 5 | 100b | 9 |
| 001b | 6 | 101b | N.A. |
| 010b | 7 | 110b | N.A. |
| 011b | 8 | 111b | N.A. |



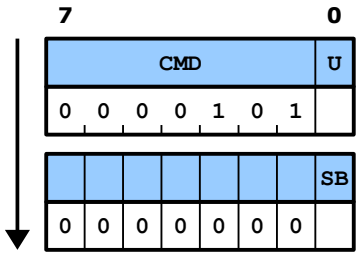

**CMD_UARTn_SET_DATA_BITS**

### Set Stop Bits

This command sets the data bits of the specified UART.

**Initial value**: SB=0

| SB | Stop bits |
|---|---|
| 0 | 1 |
| 1 | 2 |



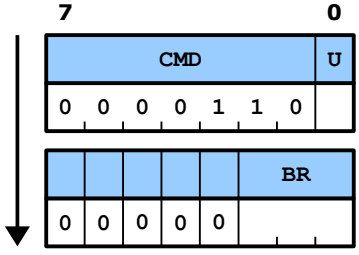

**CMD_UARTn_SET_STOP_BITS**

### Set Baud Rate

This command sets the baud rate of the specified UART.

**Initial value**: BR=111b

| BR | Baud rate | BR | Baud rate |
|---|---|---|---|
| 000b | 1200 | 100b | 19200 |
| 001b | 2400 | 101b | 38400 |
| 010b | 4800 | 110b | 57600 |
| 011b | 9600 | 111b | 115200 |

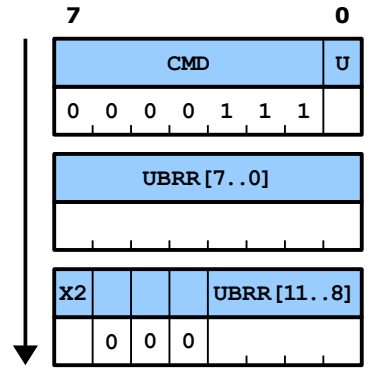



**CMD_UARTn_SET_BAUDRATE**

## Set Baud Rate Raw

This command allows you to directly set the baud rate register and double speed bit in the microcontroller. The target registers in the microcontroller are:

- U=0:
  - Field UBRR[11..0] → register UBRR0[11..0].
  - Field X2 → U2X0 in register UCSR0A.
- U=1:
  - Field UBRR[11..0] → register UBRR1[11..0].
  - Field X2 → U2X1 in register UCSR1A.

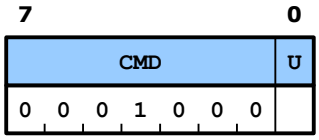



CMD_UARTn_SET_BAUDRATE_RAW

## Enable

This command enables the specified UART.

The device sends an unsolicited RSP_UARTn_TX_FREE response when it receives this command even when the special function is already enabled.




CMD_UARTn_ENABLE

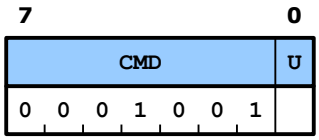
## Disable

This command disables the specified UART.

All data transfers in progress will be immediately stopped.

Any data that is stored in the BUF_UARTn_TX buffer will be flushed.

Any data that has been received in the Rx buffer will be lost.



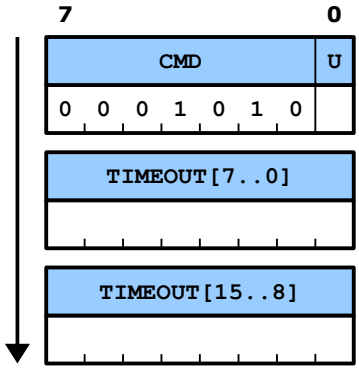

CMD_UARTn_DISABLE

## Set Rx Timeout

This command specifies how long the AxiCat has to wait before it sends an RSP_UARTn_RX response packet when the UART has received data on the RXD pin.

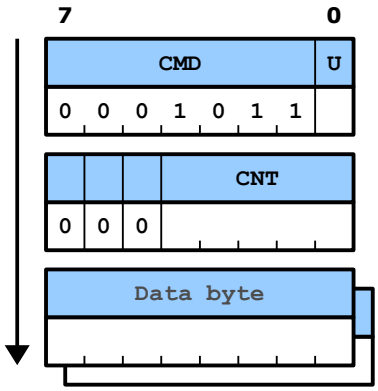
The TIMEOUT value is 0..65535 and specifies a timeout in milliseconds.

**Initial value**: TIMEOUT=50




CMD_UARTn_SET_RX_TIMEOUT

## Transmitting Data Words




CMD_UARTn_TX

Command CMD_UARTn_TX transmits one or more data words over the TXD line of the specified UART. This is an asynchronous command.

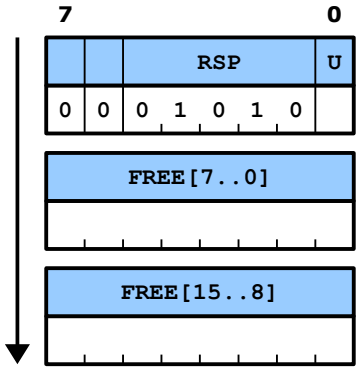
Field CNT specifies the number of data bytes minus one.

The actual number of data words to transmit depends on the data bits setting:

- 5..8: One byte per data word.
- 9: Two bytes per data word. The data bytes in the packet are ordered LSB → MSB. When an odd number of bytes is embedded, the last byte will be discarded.

When the UART is enabled, the data bytes end up in buffer BUF_UARTn_TX. The host must subtract the number of data bytes from the buffer's free bytes counter.

It's possible to send more data bytes than the buffer can hold. If this is the case, the device will block reception of bytes from the FT245. As data words are transmitted over the TXD line and buffer BUF_UARTn_TX drains, the device reads some more data from the FT245 until the buffer is full again. This process repeats until the command has been fully received.



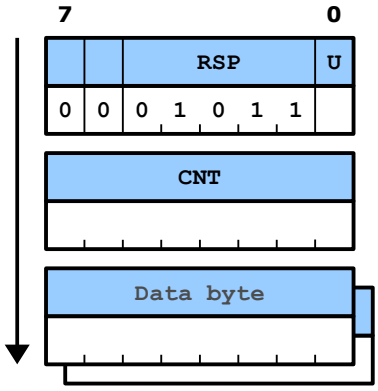

RSP_UARTn_TX_FREE

The device transmits the data words in the background. As buffer BUF_UARTn_TX drains, the device sends unsolicited RSP_UARTn_TX_FREE responses. Field FREE contains the number of free bytes minus one. The free bytes counter of buffer BUF_UARTn_TX must be updated to FREE plus one.

When the UART is disabled, any incoming CMD_UARTn_TX command is discarded. This means you can't fill up buffer BUF_UARTn_TX before enabling the UART.

The device sends an unsolicited RSP_UARTn_TX_FREE response each time it receives CMD_UARTn_ENABLE.

## Receiving Data Words




RSP_UARTn_RX

When the UART is enabled, the device receives data words over the RXD line. The data words are stored in the Rx buffer.

As soon as the Rx buffer is non-empty, the Rx timeout timer kicks in. When the timer expires, the device transfers the Rx data in an unsolicited RSP_UARTn_RX response.

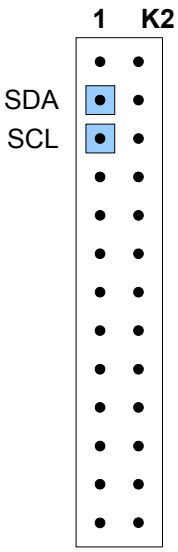
Field CNT denotes the number of data bytes minus one. The actual number of data words in the response packet depends on the data bits setting:

- 5..8: One byte per data word.
- 9: Two bytes per data word. The data bytes in the packet are ordered LSB → MSB.

The timer is skipped when the Rx buffer is full or the timeout value is set to zero. In these cases a RSP_UARTn_RX response is sent immediately after the reception of the data words.

Buffer management is not needed for the Rx buffers.

# TWI




The Two-Wire Interface (TWI) implements an I2C bus controller with elaborate master and slave functionality.

The master supports standard 100 kHz and 400 kHz speeds as well as a multitude of user-defined "raw" speed settings. Other features include multi-master arbitration, repeated start signaling, clock stretching and unlimited data payloads.

Slave features include 7-bit address and general call address recognition, clock stretching and unlimited data payloads.

When special function TWI is enabled, it overrides the group of GPIO pins as shown in the picture.

During the period the special function is enabled, all changes made to the direction and output state of the overridden GPIO pins must be deemed lost.

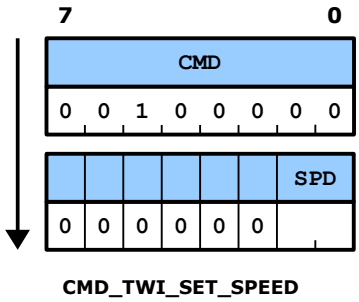
## Set Speed

This command sets the bus speed of the I2C master.

The bus speed can be set when special function TWI is enabled or disabled.

**Initial value**: SPD=01b

| SPD | Bus speed |
|---|---|
| 00b | 50000 Hz |
| 01b | 100000 Hz |
| 10b | 200000 Hz |
| 11b | 400000 Hz |



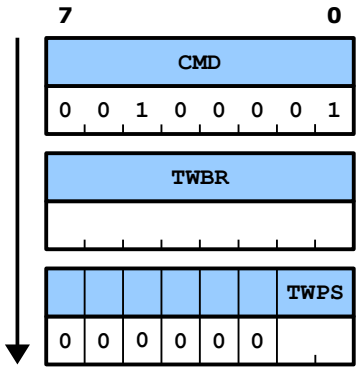

**CMD_TWI_SET_SPEED**

## Set Speed Raw

This command allows you to directly set the registers that control the bus speed in the microcontroller. The target registers in the microcontroller are:

- Clock rate: field TWBR → register TWBR.
- Prescaler: field TWPS → TWPS[1..0] in register TWSR.

The raw speed can be set when special function TWI is enabled or disabled.



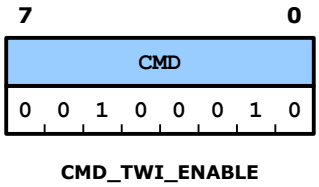

**CMD_TWI_SET_SPEED_RAW**

## Enable

This command enables special function TWI.

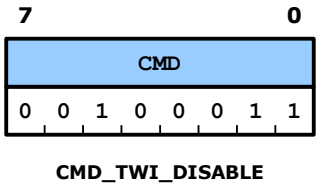
The device starts executing the first master command in buffer BUF_TWI_M, if one is present.

This command has no effect if the special function is already enabled.




**CMD_TWI_ENABLE**

## Disable




**CMD_TWI_DISABLE**

This command disables special function TWI.

The I2C bus is released. The device appears disconnected from the I2C bus.

The following actions will always be performed whether or not the special function is enabled:

- If a master command is being executed, it's completed immediately (S=0). All remaining master commands are completed as skipped (S=1).
- If a slave transmit command is being executed, it's completed immediately (S=0). All remaining slave transmit commands are completed as skipped (S=1).
- If a slave receive command is being executed, it's completed immediately (S=0). All remaining slave receive commands are completed as skipped (S=1).

## Master Clock Stretching

The master holds SCL low when it controls the I2C bus but can't continue generating signals because no master command is present in buffer BUF_TWI_M.

## Master Operation

When the TWI function is enabled, a state machine composed of master states comes in effect.

The IDLE state is the initial master state. Whilst the master is in the IDLE state, it releases SCL. In all other states the master controls the I2C bus.

In each state, the master can execute certain types of master commands stored in buffer TWI_BUF_M. If an unexpected master command is next, the device completes the command as skipped and proceeds to the next command, if any. In any state other than IDLE, if there's no command available, the master pulls SCL low.

The master transitions from the current state to the next state when it executes a command. From the resulting asynchronous response the host can determine the new master state.

A state transition only occurs when a command is completed normally. A skipped command doesn't trigger a state transition.

There's nothing wrong with commands being skipped. The host is encouraged to write master commands in advance as much as possible in order to increase I2C performance (and to keep clock stretching to a minimum). The host can't predict how the state machine will transition exactly as the behavior of I2C slaves and other factors influence the state machine. For example, if the host writes three transmit commands containing 32 data bytes each, but the slave responds with a NACK after receiving the 10$^{th}$ data byte, the latter two transmit commands will be skipped.

The state machine of the master is extensive. Rather than presenting an all-in-one diagram, each master state is shown individually along with its related transitions.

**IDLE**

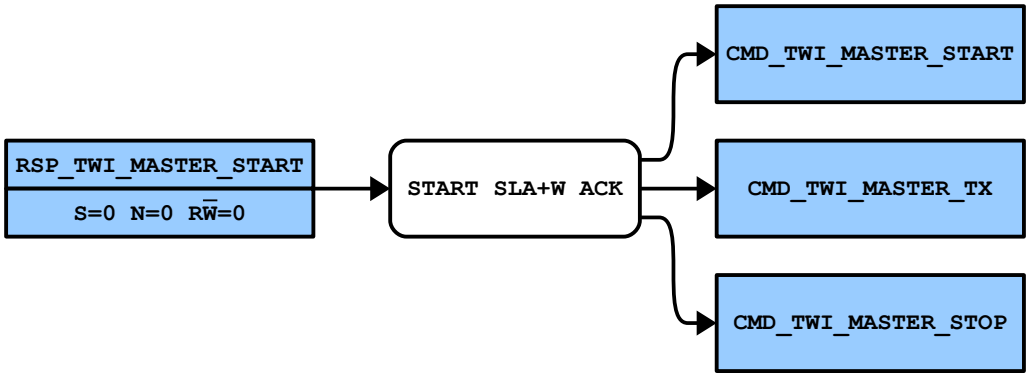
The master releases the I2C bus.


**START NACK**

The master has read back a NACK signal after SLA+W or SLA+R.

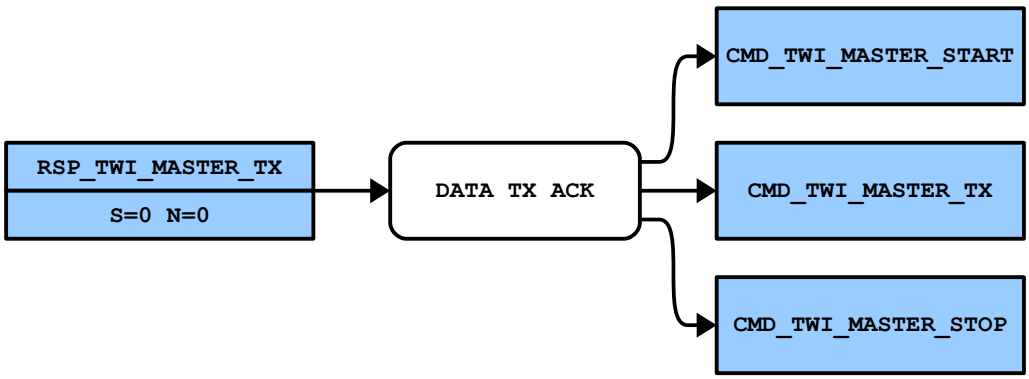

**START SLA+W ACK**

The slave has returned an ACK response after recognizing SLA+W.




**DATA TX ACK**

- The master has replied with ACK to every data byte of the transmit command.
- The host has disabled the TWI function while the transmit command was active.



**DATA TX NACK**

The slave has returned a NACK response to a data byte of the transmit command.


**START SLA+R ACK**

The slave has returned an ACK response after recognizing SLA+R.


**DATA RX ACK**

- The master has received the number of data bytes specified in the receive command. The master has generated an ACK response after receiving each data byte.
- The host has disabled the TWI function while the receive command was active.


**DATA RX NACK**

The master has received the number of data bytes specified in the receive command. The master has generated a NACK response after receiving the last data byte. This concludes the reception of data bytes from the slave.

**SKIP PAST STOP**

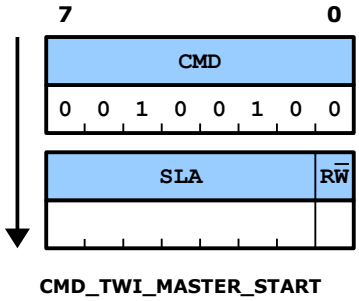
The master completes all commands as skipped up to and including the following master stop command.


## Master Start

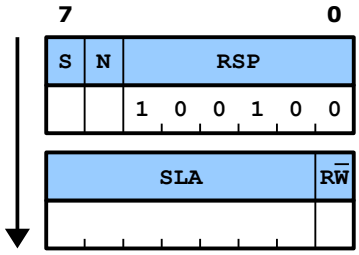
This asynchronous command generates a START or REPEATED START signal followed by SLA+W or SLA+R on the I2C bus.

Fields:

- SLA: Slave address (0..127).
- $\overline{RW}$: SLA+W (0) or SLA+R (1).
- S: Command was skipped (1) or executed (0).
- N: ACK (0) or NACK (1) received after sending SLA+W or SLA+R. Ignore this field in case the command was skipped (S=1).




CMD_TWI_MASTER_START




RSP_TWI_MASTER_START

It may take a while before the master begins executing a start command. When another master controls the I2C bus, no master can initiate a START signal.

The master begins executing a start command in one of the following situations:

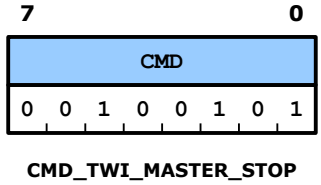
- The I2C bus is free. The master initiates START. Arbitration commences as well since other masters may initiate a START at the same time.
- The master already controls the I2C bus. The master initiates REPEATED START. Arbitration is not required in this situation.

## Master Stop

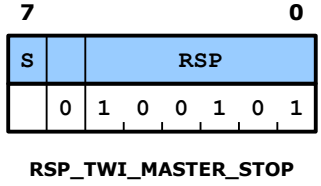
This asynchronous command generates a STOP signal on the I2C bus. The master then releases the I2C bus.

The master stop command also serves as the end marker of a sequence of commands which the master must execute atomically, thus without releasing the I2C bus.

When the master loses arbitration or detects a bus error, all commands up to and including the stop command are skipped. In either situation, the master will release the I2C bus and no STOP signal is to be generated, though a master stop command must be encountered before the master will



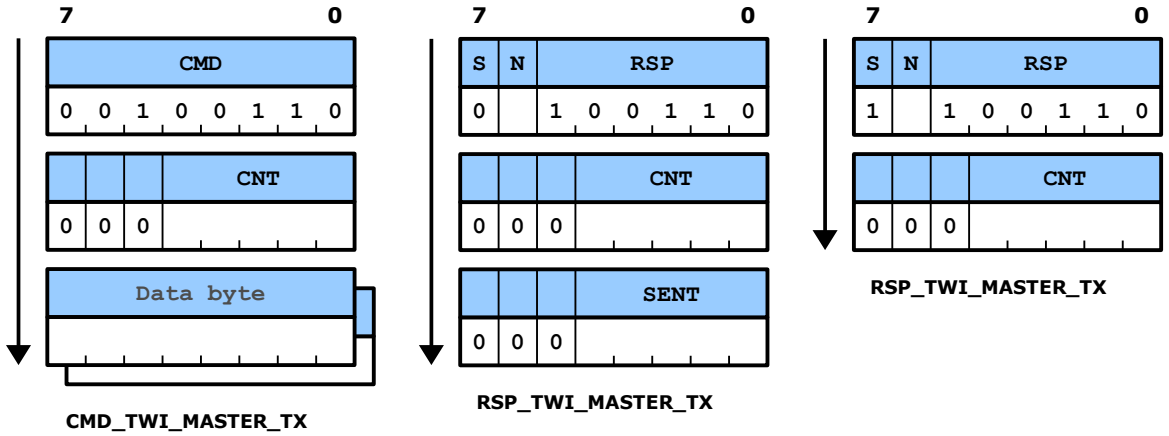

CMD_TWI_MASTER_STOP




RSP_TWI_MASTER_STOP

execute a new start command.

## Master Transmit




This asynchronous command instructs the master to transmit 1 to 32 data bytes to the slave that was addressed by a previously executed master start command.

Fields:

- CNT: Number of data bytes minus one present in the command packet.
- SENT: Number of data bytes minus one transmitted to the slave.
- S: Command was skipped (1) or executed (0).
- N: ACK (0) or NACK (1) received after sending the last data byte. Ignore the field in case the command was skipped (S=1).

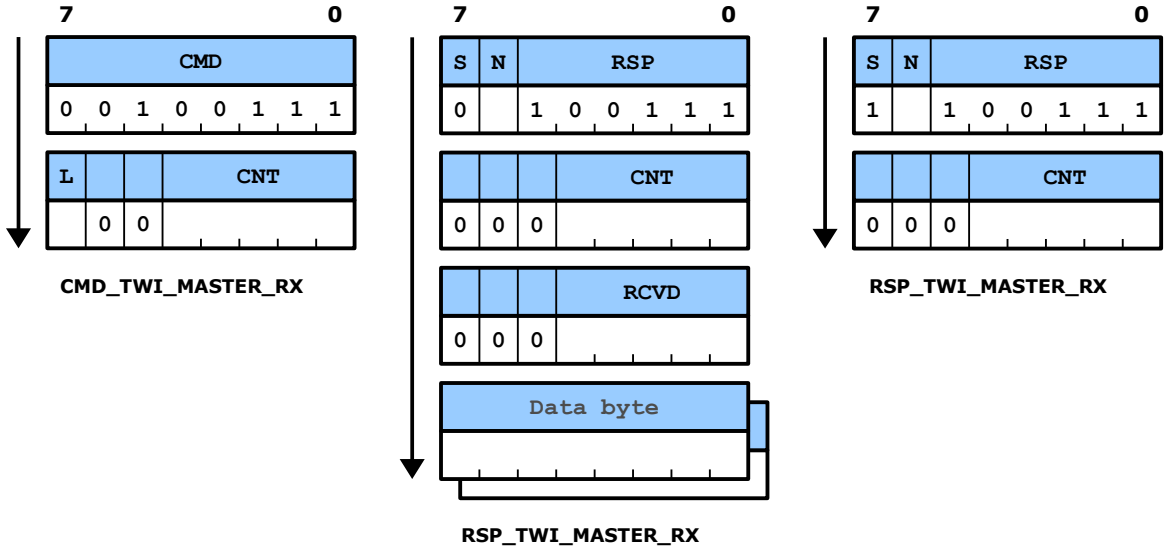
If the slave returns a NACK response after receiving a data byte, the master will cease further transmission of any data bytes. The data byte with NACK response will be added to the SENT field. This means, even when the slave returns a NACK response after the first data byte, the device will return a response packet with S=0 to the host.

The host may issue multiple master transmit commands consecutively to create one logical payload of data bytes. Each master transmit command can carry a random number of data bytes, so the host isn't required to put 32 data bytes in a master transmit command when another follows.

There's no limit to the size of the logical payload.

When the host disables special function TWI while the master is executing a master transmit command, the command is completed immediately but not marked as skipped (S=0).

## Master Receive




This asynchronous command instructs the master to receive 1 to 32 data bytes from the slave that was addressed by a previously executed master start command.

Fields:

- CNT: Number of data bytes minus one to receive.
- L: Last master receive command (1) or more to come (0).
- RCVD: Number of data bytes minus one received from the slave.
- S: Command was skipped (1) or executed (0).
- N: ACK (0) or NACK (1) sent after receiving the last data byte. Ignore this field in case the command was skipped (S=1).

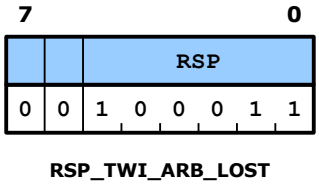
The host issues one or more master receive commands consecutively to create one logical payload of data bytes. The final command must be marked as last (L=1). Each master receive command can carry a random number of data bytes, so the host isn't required to put 32 data bytes in a master receive command when another follows.

If L=1 the master will send a NACK response after it has received the last data byte. In doing so, the master concludes the transmission of the payload.

There's no limit to the size of the logical payload.

When the host disables special function TWI while the master is executing a master receive command, the command is completed immediately but not marked as skipped (S=0).

## Arbitration Lost



RSP_TWI_ARB_LOST

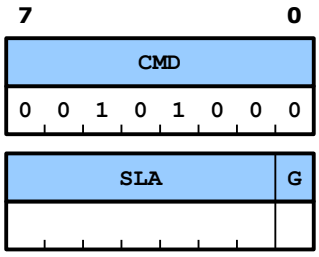
When the master loses arbitration, the device sends an arbitration lost response packet to the host.

The current command is completed. All further master commands up to and including the first stop command are skipped. It's the responsibility of the host to send a master stop command at some point, else the master won't be able to start executing a new master start command.

## Master Probe

SLA+W followed by a STOP signal or a REPEATED START signal without transmission of actual data bytes is a valid sequence. It's useful for probing the presence of an I2C slave.

## Slave Enable



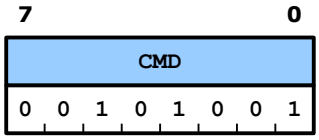
CMD_TWI_SLAVE_ENABLE

This command enables the slave role of special function TWI. The slave starts listening to SLA+W and SLA+R transmissions and checks the slave address field.

Fields:

- SLA: Slave address (0..127).
- G: Do (1) or do not (0) listen to the general call address (0000000b).

## Slave Disable



CMD_TWI_SLAVE_DISABLE

This command tells the slave to stop listening to SLA+W and SLA+R transmissions.

If a slave transmit or receive command is being executed, it'll continue to execute and possibly any following commands in the buffer until the last one (L=1) is processed, or a bus error occurs.

## Slave Clock Stretching

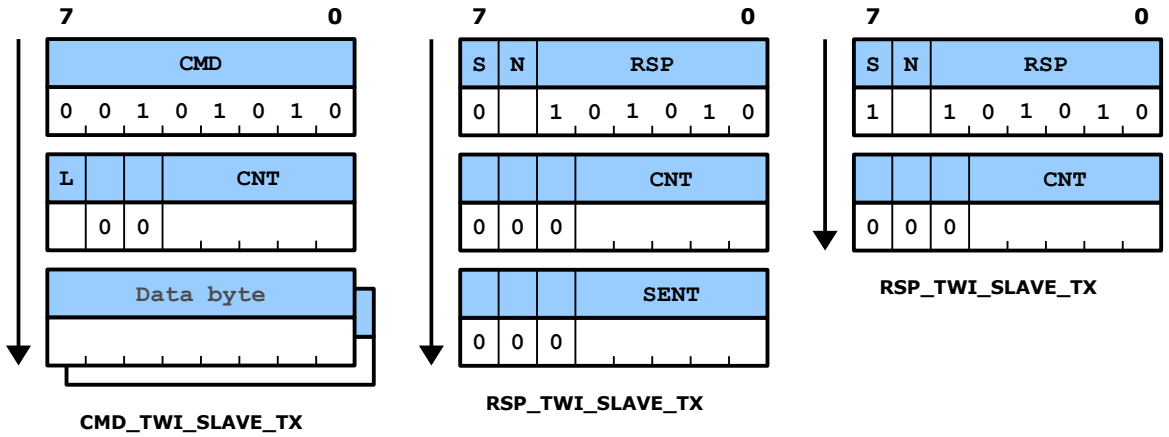
Clock stretching is an I2C feature that allows a slave to "freeze" the I2C bus for an unspecified amount of time by holding low the SCL line.

When the slave is addressed by SLA+W, it requires the presence of a slave transmit command in buffer BUF_TWI_STX. Likewise, when the slave is addressed by SLA+R, it requires the presence of a slave receive command in buffer BUF_TWI_SRX.

If no command is present when needed, the slave performs clock stretching until the host issues the required command. If another command is required later on, the slave again performs clock stretching until the command is available. This process repeats until the last command of a logical payload has been processed.

Clock stretching enables the host to evaluate incoming data bytes before producing an answer.

## Slave Transmit




This asynchronous command instructs the slave to transmit 1 to 32 data bytes to the master.

Fields:

- CNT: Number of data bytes minus one to receive.
- L: Last slave receive command (1) or more to come (0).
- SENT: Number of data bytes minus one transmitted to the master.
- S: Command was skipped (1) or executed (0).
- N: ACK (0) or NACK (1) received after sending the last data byte. Ignore the field in case the command was skipped (S=1).

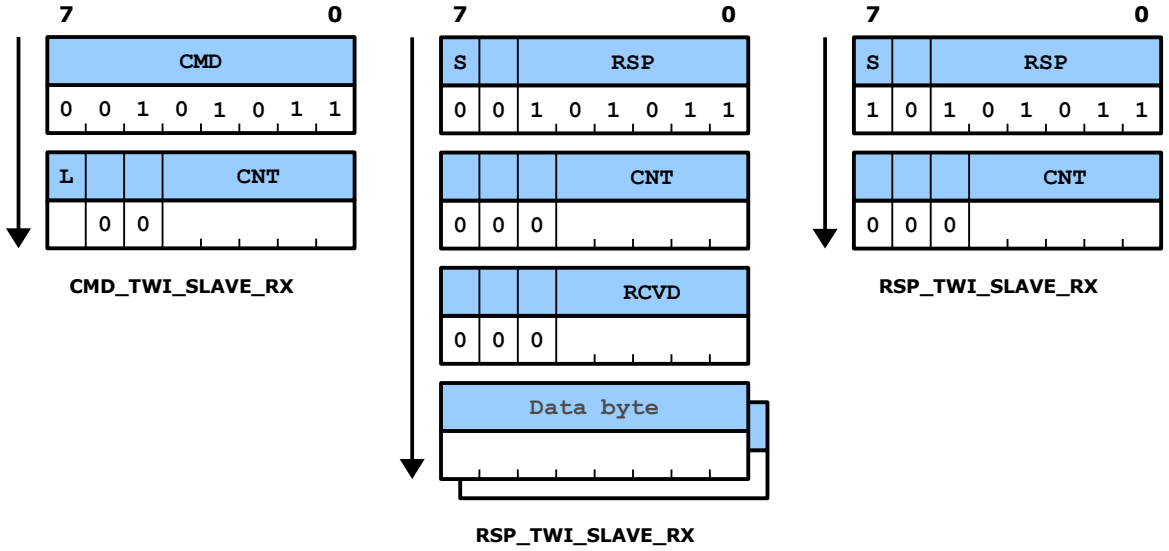
The host issues one or more slave transmit commands consecutively to create one logical payload of data bytes. The final command must be marked as last (L=1). Each slave transmit command can specify a random number of data bytes, so the host isn't required to request 32 data bytes in a slave transmit command when another follows.

The master returns an ACK response for each data byte that it successfully receives. If the master reads more data bytes than the payload holds, the slave will transmit data bytes with all bits set to one, thus with value FFh. The host receives a response for each completed command, but will not know about the extra data bytes that the slaves sends to the master.

If the host receives less data bytes than the payload provides by returning a NACK response, the slave will complete all commands that comprise the payload.

There's no limit to the size of the logical payload.

## Slave Receive




This asynchronous command instructs the slave to receive 1 to 32 data bytes from the master.

Fields:

- CNT: Number of data bytes minus one to receive.
- L: Last slave receive command (1) or more to come (0).
- RCVD: Number of data bytes minus one received from the master.
- S: Command was skipped (1) or executed (0).

The host issues one or more slave receive commands consecutively to create one logical payload of data bytes. The final command must be marked as last (L=1). Each slave receive command can specify a random number of data bytes, so the host isn't required to request 32 data bytes in a slave receive command when another follows.

The slave returns an ACK response for each data byte that fits in the logical payload. If the master transmits more data bytes, the slave will discard the unexpected data bytes and return NACK responses. The host receives a response for each completed command, but will not know about the discarded data bytes.

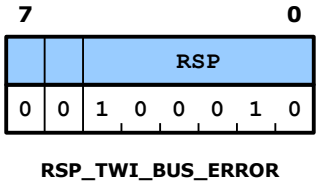
If the host sends less data bytes than the payload provides, the slave will complete all commands that comprise the payload.

If the host sends no data bytes at all (probing), the slave won't complete the command. Note that a command must be present nevertheless, else the slave will perform clock stretching until the host issues a command.

There's no limit to the size of the logical payload.

## Bus Error

The device sends this unsolicited response packet to the host when it's detected an erroneous state on the I2C bus.



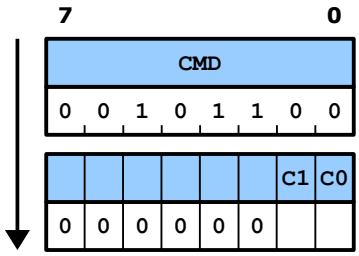

**RSP_TWI_BUS_ERROR**

## Clear

This command clears asynchronous commands from their buffer by completing them.

Fields:

- C0: If C0=1, the device completes all slave transmit commands as skipped.
- C1: If C1=1, the device completes all slave receive commands as skipped.




**CMD_TWI_CLEAR**

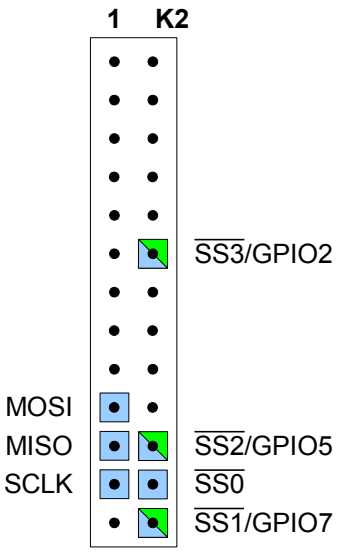
## Buffer Management

Number of bytes occupied in the various buffers:

| Command | Buffer | Bytes occupied |
|---|---|---|
| Master Start | `BUF_TWI_M` | **2** |
| Master Stop | `BUF_TWI_M` | **1** |
| Master Transmit | `BUF_TWI_M` | **CNT+4** [1] |
| Master Receive | `BUF_TWI_M` | **CNT+4** [1] |
| Slave Transmit | `BUF_TWI_STX` | **CNT+3** [1] |
| Slave Receive | `BUF_TWI_SRX` | **CNT+3** [1] |

[1] Field CNT is value minus one.

## *SPI*




The Serial Peripheral Interface (SPI) implements a three-wire synchronous serial bus with master operation and four slave select lines.

Note that SPI slave operation is not offered. A good functioning SPI slave would require specific firmware that can immediately react to incoming data byte thus without the intermediate USB interface.

When special function SPI is enabled, it overrides GPIO pins GPIO6, GPIO14, GPIO15 and GPIO16 as indicated by the single-colored blue squares in the picture.

During the period the special function is enabled, all changes made to the direction and output state of the overridden GPIO pins must be deemed lost.

GPIO2, GPIO5 and GPIO7 are NOT overridden when the special function is enabled. The host can still use them as true GPIO pins. When $\overline{SS1}$, $\overline{SS2}$ or $\overline{SS3}$ is specified in an SPI transfer command, the device will change the output state of that pin.

When using $\overline{SS1}$, $\overline{SS2}$ or $\overline{SS3}$, it's the responsibility of the host to initialize the corresponding GPIO pins to direction output and logic one before sending SPI transfers to the device.

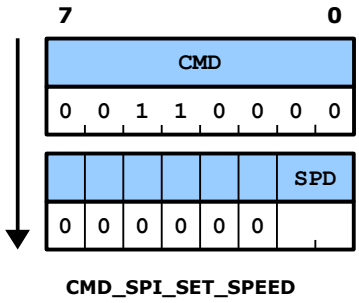
### Set Speed

This command sets the bus speed of the SPI master.

The bus speed can be set when special function SPI is enabled or disabled.

**Initial value**: SPD=00b

| SPD | Bus speed |
|---|---|
| 00b | 750000 Hz |
| 01b | 1500000 Hz |
| 10b | 3000000 Hz |
| 11b | 6000000 Hz |



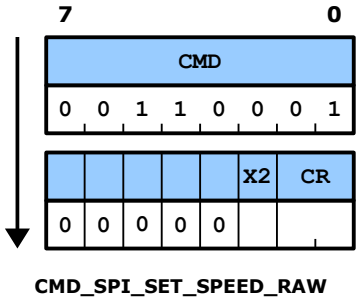

**CMD_SPI_SET_SPEED**

### Set Speed Raw

This command allows you to directly set the registers that control the bus speed in the microcontroller. The target registers in the microcontroller are:

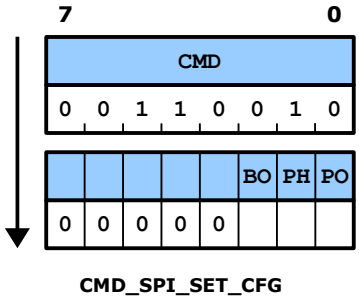
- Clock rate: field CR → SPR[1..0] in register SPCR.
- Double speed: field X2 → SPI2X in register SPCR.

The raw speed can be set when special function SPI is enabled or disabled.




**CMD_SPI_SET_SPEED_RAW**

## Configure

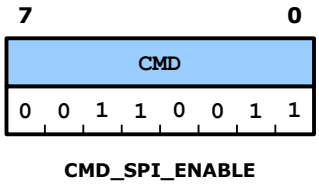



This command configures the SPI bus. Fields:

- Clock polarity:
  - PO=0: SCK is low when idle.
  - PO=1: SCK is high when idle.
- Clock phase:
  - PH=0: Sample on leading edge.
  - PH=1: Sample on trailing edge.
- Bit order:
  - BO=0: Most significant bit first (MSb → LSb).
  - BO=1: Least significant bit first (LSb → MSb).

**Initial values**: PO=0, PH=0, BO=0

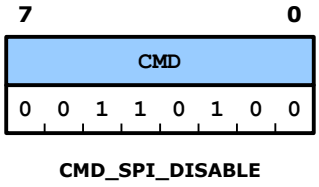
## Enable




This command enables special function SPI.

The device starts executing the first SPI transfer command, if one is present in buffer BUF_SPI.

## Disable

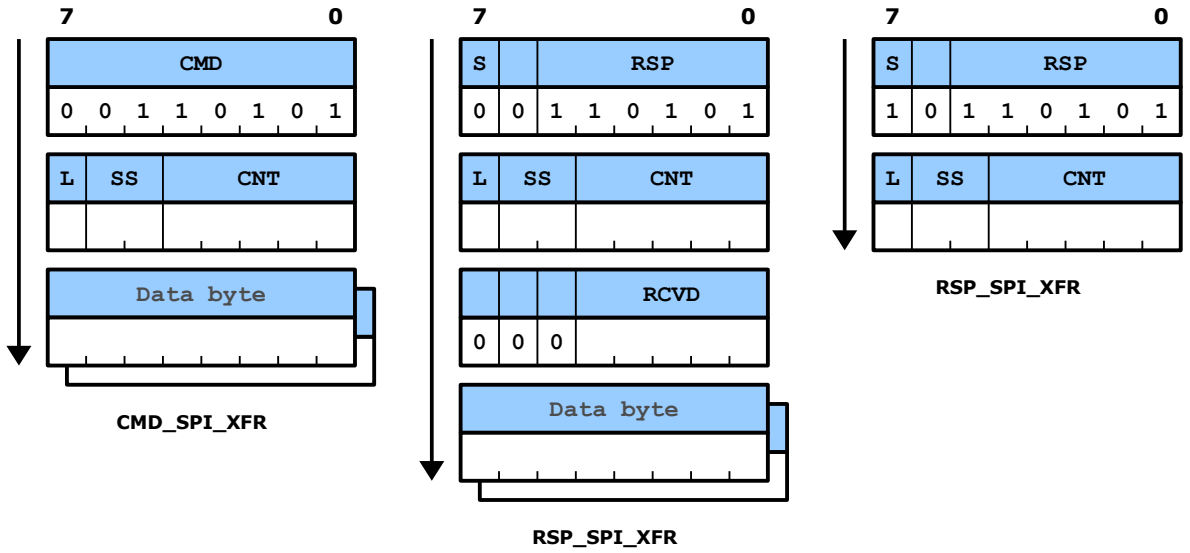



This command disables special function SPI.

The direction and output state of GPIO6, GPIO14, GPIO15 and GPIO16 are undefined. If any of $\overline{SS1}$, $\overline{SS2}$ or $\overline{SS3}$ where used in an SPI transfer, the direction and output state of the corresponding GPIO pins is also undefined.

If an SPI command is active, it's completed. Any further SPI commands in buffer BUF_SPI will be completed with the skipped bit set to one.

## Transfer




An SPI transfer writes data bytes to the MOSI line while reading data bytes from the MISO line.

Field CNT denotes to number of data bytes minus one to transfer.

Field SS tells the device which slave select line to use: 00b for $\overline{SS0}$, 01b for $\overline{SS1}$, 10b for $\overline{SS2}$, 11b for $\overline{SS3}$.

Field RCVD indicates the number of transferred data bytes minus one. In most cases this value is the same as CNT in the command packet. When the device receives command CMD_SPI_DISABLE while an SPI transfer is in progress, the transfer may be prematurely completed and RCVD would be less than CNT.

A transfer command can schedule up to 32 data bytes. If the host needs to transfer more bytes, it must send a group of commands to form a logical data payload of the required number of bytes. Field L indicates whether the command is the last of a group of commands. There's no limit to the size of the logical payload.

Each transfer command can specify a random number of data bytes, even when it's part of a logical data payload, so the host isn't required to specify 32 data bytes in a transfer command that preceeds another transfer command.

The device always asserts (sets to zero) the slave select line when it starts executing a transfer command. On the other hand, the device only releases (sets to one) the slave select line when it's done with a command with L=1. As a consequence the host must take care of the following:

- Set L=1 in the last transfer command even when there's only one command in a group. If not, the slave select line remains asserted.
- Always specify the same slave select line (field SS) in a group of commands.

## Buffer Management

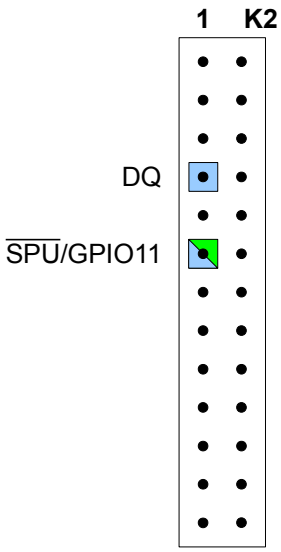
SPI transfer commands are cached in buffer BUF_SPI. The number of bytes occupied by a command is:

| **CNT + 3** [1] |
|---|

[1] Field CNT is value minus one.

Example: If the SPI transfer command carries 10 data bytes, the number of bytes occupied is: (10-1) + 3 = 12.

# *1-Wire*




The 1-Wire Interface implements master operation, support for strong pull-up and a versatile enumeration procedure.

When special function 1-Wire is enabled, it overrides pin GPIO10 as indicated by the single-colored blue square in the picture.

During the period the special function is enabled, all changes made to the direction and output state of the overridden GPIO pin must be deemed lost.

GPIO11 is NOT overridden when the special function is enabled. The host can still use it as a true GPIO pin. When strong pull-up is selected in a 1-Wire command, the device will change the output state of GPIO11.

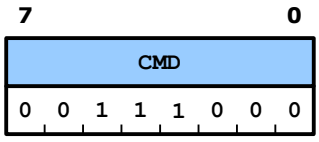
When using strong pull-up control, it's the responsibility of the host to initialize SPU/GPIO11 to direction output and logic one before sending 1-Wire commands to the device.

The 1-Wire special function supports standard speed.

## Enable

This command enables special function 1-Wire.

The device starts executing the first 1-Wire command, if one is present in buffer BUF_OW.

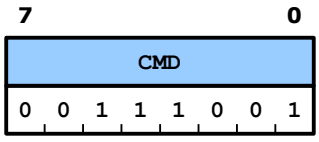



**CMD_OW_ENABLE**

## Disable

This command disables special function 1-Wire.

If a 1-Wire command is active, it's completed. Any further 1-Wire commands in buffer BUF_OW will be completed with the skipped bit set to one.



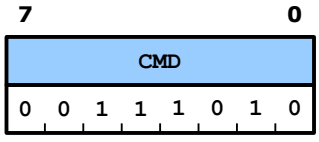

**CMD_OW_DISABLE**

## Reset

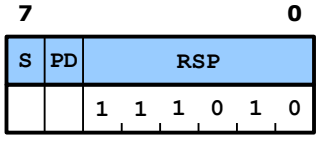
The command generates a 1-Wire reset signal on the 1-Wire bus.

Any 1-Wire slave that recognizes the reset signal will report its presence to the master. The PD flag is set when one or more slaves are present on the 1-Wire bus.

If strong pull-up was activated during the execution of a previous command, the device will drive pin SPU/GPIO11 high before the reset signal is generated.

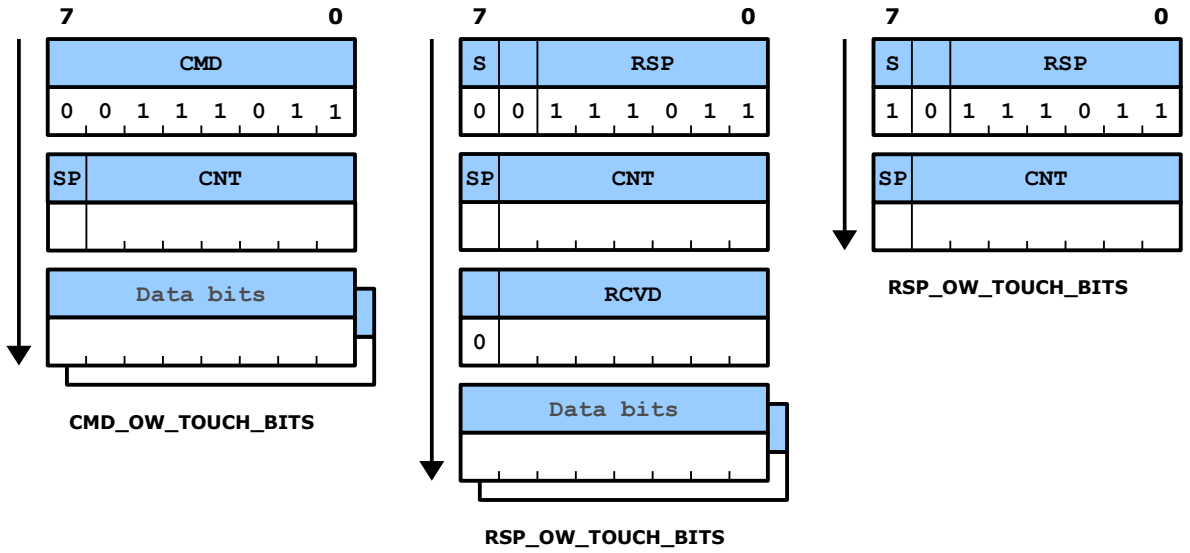



**CMD_OW_RESET**




**RSP_OW_RESET**

## Touch Bits




A 1-Wire touch bits command writes data bits and simultaneously reads back data bits on the DQ line.

When the device write a one, the slave can report either zero or one. When the device writes a zero, the bit value that's read back always equals zero.

Field CNT denotes the number of data bits minus one to transfer.

Field SPU controls the strong pull-up signal on pin $\overline{\text{SPU}}$. If SPU is one, the device will drive pin $\overline{\text{SPU}}$ low after the last data bit has been transferred.

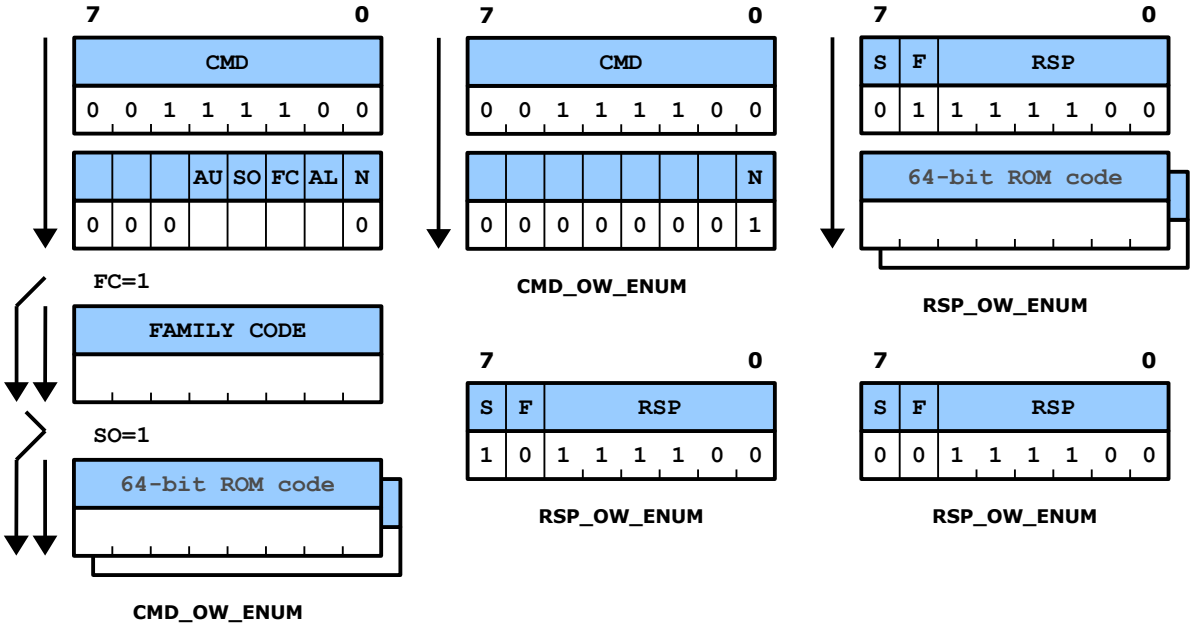
If strong pull-up was activated during the execution of a previous command, the device will drive pin $\overline{\text{SPU}}$ high before the first data bit is transferred.

Field RCVD indicates the number of transferred data bits minus one. In most instances this value is the same as CNT in the command packet. When the device receives command CMD_OW_DISABLE while 1-Wire bit transfers are in progress, the command may be prematurely completed and RCVD would be less than CNT.

A touch bits command can contain up to 128 data bits or 16 data bytes. If the host needs to transfer more bits, it must send a group of commands to form a logical data payload of the required number of bits. There's no limit to the size of the logical payload.

Each touch bits command can specify a random number of data bits, even when it's part of a logical data payload, so the host isn't required to specify 128 data bits in a touch bits command that preceeds another touch bits command.

## Enumerate




The enumeration command searches for a 1-Wire slave on the 1-Wire bus and reports the ROM code if a slave is found. 1-Wire enumeration is a procedure that requires one or more iterations to complete, one iteration for each enumerated 1-Wire slave.

Each enumeration command schedules one iteration of a 1-Wire enumeration. The command either initiates an enumeration procedure (N=0) or continues the procedure (N=1). Both variations of the command generate one of the indicated responses.

If field N (next) is zero, the command establishes a new enumeration context based on the specified search criteria, and executes the first iteration of the enumeration procedure. In other words, the command searches for the first 1-Wire slave.

If field N is one, the command continues from where the previous command left off and searches for the next 1-Wire slave, if any.

If AL, FC and SO are zero, the command will start a search for all 1-Wire slaves that are visible on the 1-Wire bus. You can narrow down the enumeration by specifying any combination of the following search criteria:

- AL=1 (Alarm condition search): Only 1-Wire slaves that have an alarm condition are enumerated. The device issues 1-Wire command Alarm Search (ECh) instead of Search ROM (F0h).
- FC=1 (Family code search): Only 1-Wire slaves bearing the specified family code are enumerated.
- SO=1 (DS2409 smart ON): If you choose this option, only 1-Wire slaves that reside behind the main or auxiliary gate of a DS2409 are enumerated. You have to choose between main and auxiliary gate and you've to provide the ROM code of the target DS2409.

The length of the command packet depends on the state of fields FC and SO. If FC is set, the device expects a 1-byte family code. If SO=1, the device expects an 8-byte ROM code.

If field F (found) is one, an 8-byte ROM code is included in the response packet and the enumeration procedure remains active.

If field F is zero, no more 1-Wire slave could be found and the enumeration procedure has ended.

The device does not check the CRC of ROM codes in any command or response.

If strong pull-up was activated during the execution of a previous command, the device will drive pin SPU/GPIO11 high before the enumeration is started.

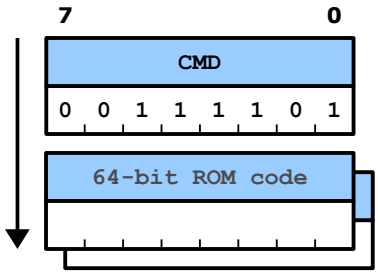
## Probe

The probing command searches for the presence of a specific 1-Wire slave on the 1-Wire bus. The 1-Wire slave is identified by its ROM code.

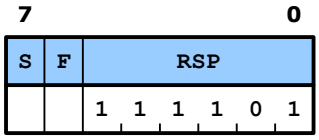
If field F (found) is one, the probing procedure has detected the presence of the 1-Wire slave with the given ROM code. If field F is zero, the 1-Wire slave hasn't been detected.

Field F is always zero when field S is one.

If strong pull-up was activated during the execution of a previous command, the device will drive pin SPU/GPIO11 high before probing is started.




**CMD_OW_PROBE**




**RSP_OW_PROBE**

## Buffer Management

1-Wire commands are cached in buffer BUF_OW.

| Command | Bytes occupied |
|---|---|
| Reset | **1** |
| Touch Bits | **CNT+4** [1] |
| Enumerate | **11** |
| Probe | **9** |

[1] Field CNT is value minus one.

## Host Initialization Procedure

When the host connects to the device, it has to synchronize with the device. There are a number of challenges:

- The device may be in the middle of receiving a command. Worst case the device may be expecting 32 data bytes, for example, as part of a CMD_UART0_TX command.
- The device may be in any state. The host can't assume any state since a previous connection may have been closed at any point during I/O with the device.
- The FT245 may hold data bytes from a previous connection in its Rx channel. The host may receive these data bytes upon connecting with the device in case the Rx channel isn't purged.
- One or more special functions may be enabled. For example, an enabled UART could be receiving data and returning RSP_UARTn_RX responses.
- The device may not be an AxiCat at all.

Here's the recommended initialization procedure:

1. Write 32 CMD_GEN_NOP commands.
2. Write CMD_UART0_DISABLE.
3. Write CMD_UART1_DISABLE.
4. Write CMD_SPI_DISABLE.
5. Write CMD_OW_DISABLE.
6. Write CMD_TWI_SLAVE_DISABLE.
7. Write CMD_TWI_DISABLE → CMD_MASTER_START → CMD_TWI_DISABLE. This sequence forces the device to complete the master start command as skipped, thus returning a response packet. Address and direction in the master start command are irrelevant.
8. Read data from the device, thus purging the Rx channel. If an AxiCat is present, at least RSP_MASTER_START must come in. Incoming data may span multiple reads. Use these recommended timeouts:
   - Timeout for reading first data chunk: 2 seconds. If this timeout occurs, the device didn't return any data bytes and initialization fails.
   - Timeout for reading subsequent data chunks: 200 milliseconds. If this timeout occurs, the Rx channel has been purged.
9. At this point, the initialization procedure has succeeded. The host and the device are synchronized.

# 4 Serial Protocol Revision History

| Version | Description |
|---|---|
| 1.0 | ▪ Initial release. |
| 1.1.0 | ▪ Added special function 1-Wire.<br>▪ Added GEN INFO command.<br>▪ Removed GEN BUF INFO command. |
| 1.2.0 | ▪ Added 1-Wire enumeration. |
| 1.3.0 | ▪ Added 1-Wire probing. |
| 1.3 | ▪ Changed versioning scheme of serial protocol to major and minor version numbers. |

# 5 Legal Information

## *Disclaimer*

Axiris products are not designed, authorized or warranted to be suitable for use in space, nautical, space, military, medical, life-critical or safety-critical devices or equipment.

Axiris products are not designed, authorized or warranted to be suitable for use in applications where failure or malfunction of an Axiris product can result in personal injury, death, property damage or environmental damage.

Axiris accepts no liability for inclusion or use of Axiris products in such applications and such inclusion or use is at the customer's own risk. Should the customer use Axiris products for such application, the customer shall indemnify and hold Axiris harmless against all claims and damages.

## *Trademarks*

All product names, brands, and trademarks mentioned in this document are the property of their respective owners.

# 6 Contact Information

Official website: http://www.axiris.eu/