Texas Instruments

Z-Stack ZNP

ZigBee PRO Network Processor

Accelerate your ZigBee Development

Applications

- *ZigBee™ systems*
- *Home/Building automation*
- *Industrial control and monitoring*
- *Low power wireless sensor networks*
- *Set-top boxes and remote controls*
- *Automated Meter Reading*

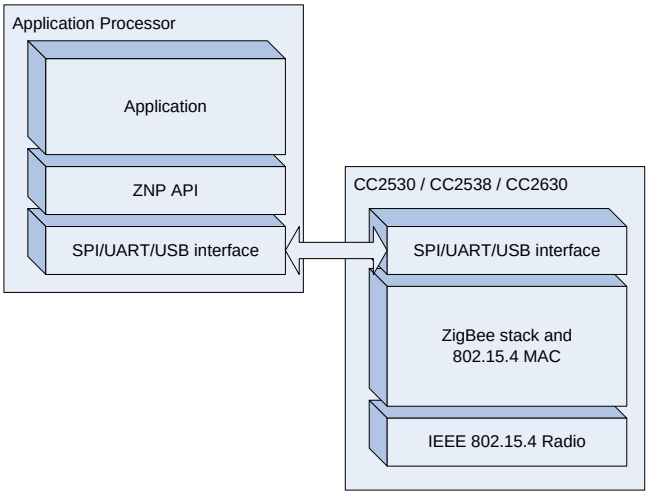
Description

The ***Z-Stack ZNP*** is a cost-effective, low power, ZigBee Network Processor that provides full ZigBee functionality with a minimal development effort.

In this solution, the ZigBee PRO stack runs on an SoC and the application runs on an external microcontroller. The ***Z-Stack ZNP*** handles all the ZigBee protocol tasks, and leaves the resources of the application microcontroller free to handle the application.

This makes it easy for users to add ZigBee to new or existing products at the same time as it provides great flexibility in choice of microcontroller.

Z-Stack ZNP interfaces to any microcontroller through a range of serial interfaces. For example, it can be combined with an MSP430 or Stellaris ARM Cortex-M3 microcontroller.



Key Features

- *All the powerful features of the ZigBee PRO system-on-chip with a simplified application interface.*
- *SPI[1], UART[2] or USB[3] interface to application processor with SPI speeds up to 2 MHz.*
- *Designed for low power operation when using SPI interface with maximum time spent in low power mode when using SPI interface.*
- *Access to 12-bit analog-to-digital converter, GPIO pins, non-volatile memory*

[1] SPI interface is supported on CC2530 and CC2538.

[2] UART interface is supported on CC2630/50, CC2530 and CC2538.

[3] USB interface is supported on CC2538 and CC2531 only.

References

[R1] NPI Users's Guide

[R2] Z-Stack Monitor and Test API

[R3] CC2630 Online Documentation : http://www.ti.com/product/cc2630

[R4] CC2538 Online Documentation : http://www.ti.com/product/cc2538

[R5] CC2530 Online Documentation : http://www.ti.com/product/cc2530

[R6] CC2531 Online Documentation : http://www.ti.com/product/cc2531

[R7] CC2591 Online Documentation : http://www.ti.com/product/cc2591

[R8] CC2592 Online Documentation : http://www.ti.com/product/cc2592

Acronyms

ADC	Analog to Digital Conversion (or Converter)
AF	ZigBee Application Framework
API	Application Programming Interface
AREQ	Asynchronous Request
CTS	Clear To Send
FCS	Frame Check Sequence
GPIO	General Purpose I/O
NPI	Network Processor Interface
NV	Non-Volatile
PA/LNA	Power Amplifier / Low Noise Amplifier (CC259x)
POLL	Poll request
RPC	Remote Procedure Call
RTS	Ready To Send
SAPI	Simple API
SoC	System on Chip
SOF	Start Of Frame
SPI	Serial Peripheral Interface bus
SREQ	Synchronous request
SRSP	Synchronous response
UART	Universal Asynchronous Receiver Transmitter
ZDO	ZigBee Device Object
ZNP	ZigBee Network Processor

1 Introduction

This document consolidates the existing ZNP Interface Specifications for all the devices supported in the ZigBee ® SimpleLink family (CC2530, CC2531, CC538 and CC2630)

To reduce duplicated and redundant information, simple summaries are provided here and references to external documents are provided where necessary.

2 Physical Interface

The following sections describe the physical interface for the ZNP for each of the supported platforms.

2.1 CC2630/50

The CC2630 (and equivalently the CC2650) is the newest member of the family of Z–Stack ZNP platforms. On CC2630, ZNP includes some significant differences when compared with the other established ZNP paltforms:

- **TI-RTOS**: As with all Z-Stack software products on CC2630, it is built on top of TI-RTOS, a Real-Time Operating System developed by Texas Instruments.
- **NPI**: On CC2630, the ZNP architecture incorporates a new NPI (Network Processor Interface) subsystem. The NPI subsystem represents a convergance of Texas Instruments Network Processor-based software products (eg. ZigBee, BLE, MAC) onto a single common architecture. In the Network Processor approach, the core stack operations run on the embedded device, while applications run on the external host.

The following sections for CC2630 are provided as a simple summary. For more information on the NPI-based ZNP for CC2630, please refer to [R1].

NOTE: At this time, CC2630 ZNP only supports the UART NPI transport.

2.1.1 Network Processor Signals

The figure below shows how an application processor interfaces with the CC2630 ZNP.

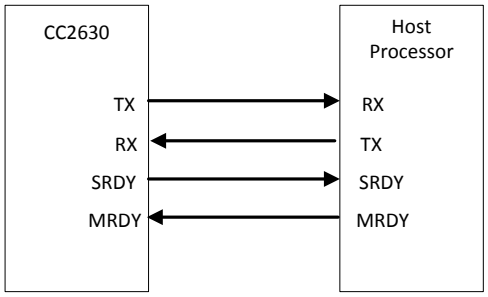


Figure 1 CC2630 Interface

The CC2630-ZNP uses the following signals for the hardware interface

- **RX/TX for UART:** These are the standard signals used for UART communication. Please refer to [R1] for details.

  NOTE: Hardware-based UART flow-control is currently not supported on the CC2630.
- **SRDY:** This active low signal is asserted by the CC2630 for power management and transaction control. The application processor can use a regular GPIO pin to poll the status of this signal, or connect it to a GPIO with edge configurable interrupt capability. Please refer to [R1] for details.
- **MRDY:** This active low signal is asserted by the application processor for power management and transaction control. Please refer to [R1] for details.

2.1.1.1 Pin Configuration

The Pin Configuration for ZNP on CC22630 is defined in the following table. Note that ZNP supports three different package sizes for the CC2630:

Transport	CC2630-ZNP signal	CC2630 7x7 PIN	CC2630 5x5 PIN	CC2630 4x4 PIN	*Direction (on C2630)*
POWER_SAVING	SRDY	DIO_12	DIO_4	DIO_3	*Out*
POWER_SAVING	MRDY	DIO_19	DIO_6	DIO_4	*In*

UART	TX	DIO_3	DIO_0	DIO_2	*Out*
UART	RX	DIO_2	DIO_1	DIO_1	*In*

2.1.2 Interface Configuration

2.1.2.1 IAR Project Configuration

The CC2630-ZNP IAR project that is included in the ZStack software package currently only supports UART for network processor host connectivity. Go to *Project->Options->C/C++ Compiler->Preprocessor->Defined Symbols* and ensure the following are defined:

NPI - enables the new NPI subsystem

NPI_USE_UART - enables the UART tansport layer of the NPI subsystem

NPI_UART_BR=11520 - configures the baudrate of the UART interface. This can be modified to suit your project's requirements.

NPI_SREQRSP - enables support for Synchronous REQ/RSP messaging.

POWER_SAVING - configures support for power savings operation which requires the connection and use of the MRDY and SRDY signals.

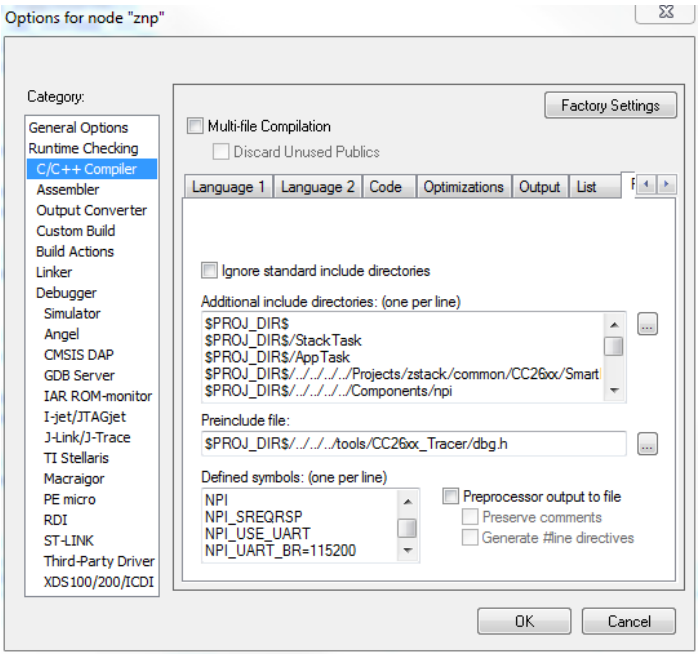


Figure 2 IAR Setup for CC2630 ZNP

2.1.2.2 CC2630 ZNP IAR Build Configurations

Due to code size constraints, the CC2630 ZNP does not support the "All Devices" build configuration where a single compiled image includes full support for all three ZigBee device types (Coordinator, Router and End Device).

Within the ZNP IAR work space for CC2630, there are nine build configurations: for each device type there are three predefined build configurations with increasing levels of security.

2.1.3 UART Transport

2.1.3.1 Configuration

The following UART configuration is supported:

- Baud rate: up to 115200
- 8-N-1 byte format.

2.1.3.2 Frame Format

UART transport frame format is shown in the following figure. The left-most field is transmitted first over the wire. This is the same General Serial Packet defined by the Monitor and Test (MT) specification [R2].

Bytes: 1	3-253	1
SOF	General format frame	FCS

Figure 3 UART Transport Frame Format

SOF: Start of frame indicator. This is always set to 0xFE.

General frame format: This is the general frame format as described in 2.1.3.4.

FCS: Frame-check sequence. This field is computed as an XOR of all the bytes in the general format frame fields.

2.1.3.3 Sample FCS Calculation

Shown below is a C example for the FCS calculation:

```
unsigned char calcFCS(unsigned char *pMsg, unsigned char len)
{
  unsigned char result = 0;
  while (len--)
  {
    result ^= *pMsg++;
  }
  return result;
}
```

2.1.3.4 General Frame Format

The general frame format is shown in the following figure. The left-most field is transmitted first over the wire. For multi-byte fields, the lowest order byte is transmitted first. This is the same General Frame Format defined by the Monitor and Test (MT) specification [R2].

Bytes: 1	2	0-250
Length	Command	Data

Figure 4 General Frame Format

Length: The length of the data field of the frame. The length can range from 0-250.

Command: The command of the frame.

Data: The frame data. This depends on the command field and is described for each command in Section 3.

2.1.3.4.1 Command Field

The command field is constructed of two bytes. The bytes are formatted as shown in the following figure. The Cmd0 byte is transmitted first.

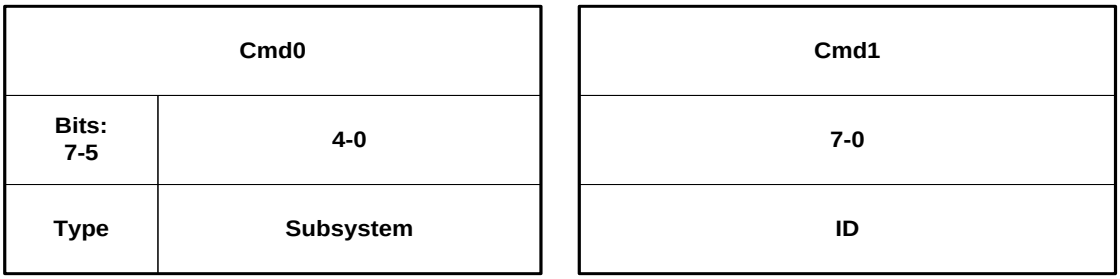


Figure 5 Command Field

Type: The command type has one of the following values:

- **0: POLL**. For CC2630, this command type is not supported.
- **1: SREQ**: A synchronous request that requires an immediate response. For example, a function call with a return value would use an SREQ command.
- **2: AREQ**: An asynchronous request. For example, a callback event or a function call with no return value would use an AREQ command.
- **3: SRSP**: A synchronous response. This type of command is only sent in response to a SREQ command. For an SRSP command the subsystem and ID are set to the same values as the corresponding SREQ. The length of an SRSP is generally nonzero, so an SRSP with length=0 can be used to indicate an error.
- **4-7: Reserved**.

Subsystem: The subsystem of the command. Values are shown below:

Subsystem Value	Subsystem Name
0	RPC Error interface
1	SYS interface
2	Reserved
3	Reserved
4	AF interface
5	ZDO interface
6	Simple API interface
7	UTIL interface
8	Reserved
9	APP Interface
10-31	Reserved

ID: The command ID. The ID maps to a particular interface message. Value range: 0-255.

When the ZNP cannot recognize an SREQ command from the host processor, the following SRSP is returned:

SRSP:

1	1	1	1	1	1
Length = 0x03	Cmd0 = 0x60	Cmd1 = 0x00	ErrorCode	ReqCmd0	ReqCmd1

Attributes:

<table>
<tr><th>Attribute</th><th>Length (byte)</th><th>Description</th></tr>
<tr><td>ErrorCode</td><td>1</td><td>The error code maps to one of the following enumerated values.
<table><tr><th>Value</th><th>Description</th></tr><tr><td>0x01</td><td>Invalid subsystem</td></tr><tr><td>0x02</td><td>Invalid command ID</td></tr><tr><td>0x03</td><td>Invalid parameter</td></tr><tr><td>0x04</td><td>Invalid length</td></tr></table></td></tr>
<tr><td>ReqCmd0</td><td>1</td><td>The Cmd0 value of the processed SREQ</td></tr>
<tr><td>ReqCmd1</td><td>1</td><td>The Cmd1 value of the processed SREQ</td></tr>
</table>

2.1.4 Initialization Procedures

2.1.4.1 ZNP power-up procedure

The recommended power-up procedure is as follows:

1. Application processor and ZNP power up.
2. The application processor initializes its UART interface.
3. The application processor sends a `SYS_RESET_REQ` to the ZNP.
4. The application processor receives the `SYS_RESET_IND` message from the ZNP.

The ZNP can be reset when the application processor sends a `SYS_RESET_REQ` message.

2.1.5 Additional Information

For more information on the physical interface and transport layer details for CC2630/50, please refer to the NPI User's Guide [R1].

2.2 CC2538

2.2.1 Network processor signals

The figure below shows how an application processor interfaces with the CC2538.

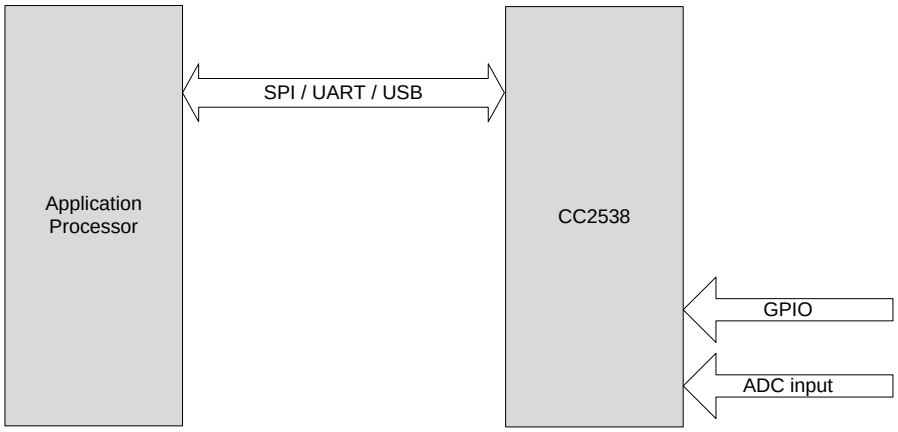


Figure 6 CC2538 Interface

The CC2538-ZNP uses the following signals for the hardware interface

- **MISO/MOSI/CLK/CS for SPI, and RX/TX/RTS/CTS for UART:** These are the standard signals used for SPI or UART communication. See sections 2.2.3.3 (for SPI) and 2.2.4.4 (for UART) for details.
- **SRDY:** This signal is asserted by the CC2538 for power management and transaction control when using SPI transport. The application processor can use a regular GPIO pin to poll the status of this signal, or connect it to a GPIO with edge configurable interrupt capability. See section 2.2.3.3 for details.
- **MRDY:** This signal is asserted by the application processor for power management and transaction control when using SPI transport. This is typically hardwired to the SS pin and does not have to be controlled by a separate GPIO from the application processor. See section 2.2.3.3 for details.

2.2.1.1 Pin Configurations

The CC2538-ZNP Pin configurations are described in the following sections.

2.2.1.1.1 Default pin configuration

By default, the Pin Configurations is the following:

Transport	CC2538-ZNP signal	CC2538 PIN	CC2538 NAME	*Direction (on C2538)*
SPI	**MOSI**	P1_18	PA4	*In*
SPI	**CS**	P1_14	PA3	*In*
SPI	**CLK**	P1_16	PA2	*In*
SPI	**MISO**	P1_20	PA5	*Out*
SPI	**SRDY**	P1_03	PB0	*Out*
SPI	**MRDY**	P1_08	PC5	*In*

SPI	**GND**	GND	GND	*In/Out*
UART	**TX**	P1_09	PA1	*Out*
UART	**RX**	P1_07	PA0	*In*
UART	**CTS**	P1_03	PB0	*In*
UART	**RTS**	P2_18	PD3	*Out*
UART	**GND**	GND	GND	*In/Out*

2.2.1.1.2 Alternate pin configuration

Go to Project-> Options-> C/C++Compiler-> Preprocessor-> DefinedSymbols and add ZNP_ALT. The ZNP_ALT supports RESET pin. The Pin Configurations is the following:

Transport	CC2538-ZNP signal	CC2538 PIN	CC2538 NAME	*Direction (on C2538)*
SPI	**MOSI**	P1_18	PA4	*In*
SPI	**CS**	P1_14	PA3	*In*
SPI	**CLK**	P1_16	PA2	*In*
SPI	**MISO**	P1_20	PA5	*Out*
SPI	**SRDY**	P2_10	PD5	*Out*
SPI	**MRDY**	P1_11	PB2	*In*
SPI	**GND**	GND	GND	*In/Out*
SPI	**RESET**	P2_15	EM_RESET	*In*
UART	**TX**	P1_09	PA1	*Out*
UART	**RX**	P1_07	PA0	*In*
UART	**CTS**	N/A	N/A	*In*
UART	**RTS**	N/A	N/A	*Out*
UART	**GND**	GND	GND	*In/Out*

2.2.1.1.3 Beagle Bone White with CC2538 pin configuration

Go to Project-> Options-> C/C++Compiler-> Preprocessor-> Defined Symbols and add BB_ZNP. The Pin Configurations is the following:

Transport	CC2538-ZNP signal	CC2538 PIN	CC2538 NAME	*Direction (on C2538)*
SPI	**MOSI**	P1_18	PA4	*In*
SPI	**CS**	P1_14	PA3	*In*
SPI	**CLK**	P1_16	PA2	*In*
SPI	**MISO**	P1_20	PA5	*Out*
SPI	**SRDY**	P1_03	PB0	*Out*
SPI	**MRDY**	P2_18	PD3	*In*
SPI	**GND**	GND	GND	*In/Out*

2.2.1.1.4 USB pin configuration

In this configuration, the CC2538-ZNP will use the USB transport the pin-out of the CC2538 can be found in the datasheet. The USB transport exposes the CDC (communication device class) class USB interface and exposes a virtual COM port to the host. The host processor would then access this device as a regular COM port device and communicate with the ZNP using the UART

Transport. For more information on the USB driver go to CC2538 foundation firmware package [R4]. For the USB to survive system reset, Soft Reset has been introduced. For more information on SYS_RESET_REQ please refer to [R2].

2.2.2 Interface Configuration

The CC2538-ZNP supports SPI, UART, or USB interface to the application processor.

2.2.2.1 IAR project configuration

The CC2538-ZNP IAR project that is included in the ZStack software package cabn be configured to UART, USB or SPI by using proprocessor defines. Go to Project->Options->C/C++ Compiler->Preprocessor->Defined Symbols and configure as following:-

For SPI add “HAL_SPI=TRUE”. This is out-of-the box configuration.

For UART, remove “HAL_SPI”. No preprocessor defines are needed to setup as UART.

For USB, add “HAL_UART_USB” and “USB_SETUP_MAX_NUMBER_OF_INTERFACES=5”

PLEASE NOTE: Make sure that only one of the above defines are used at any point.

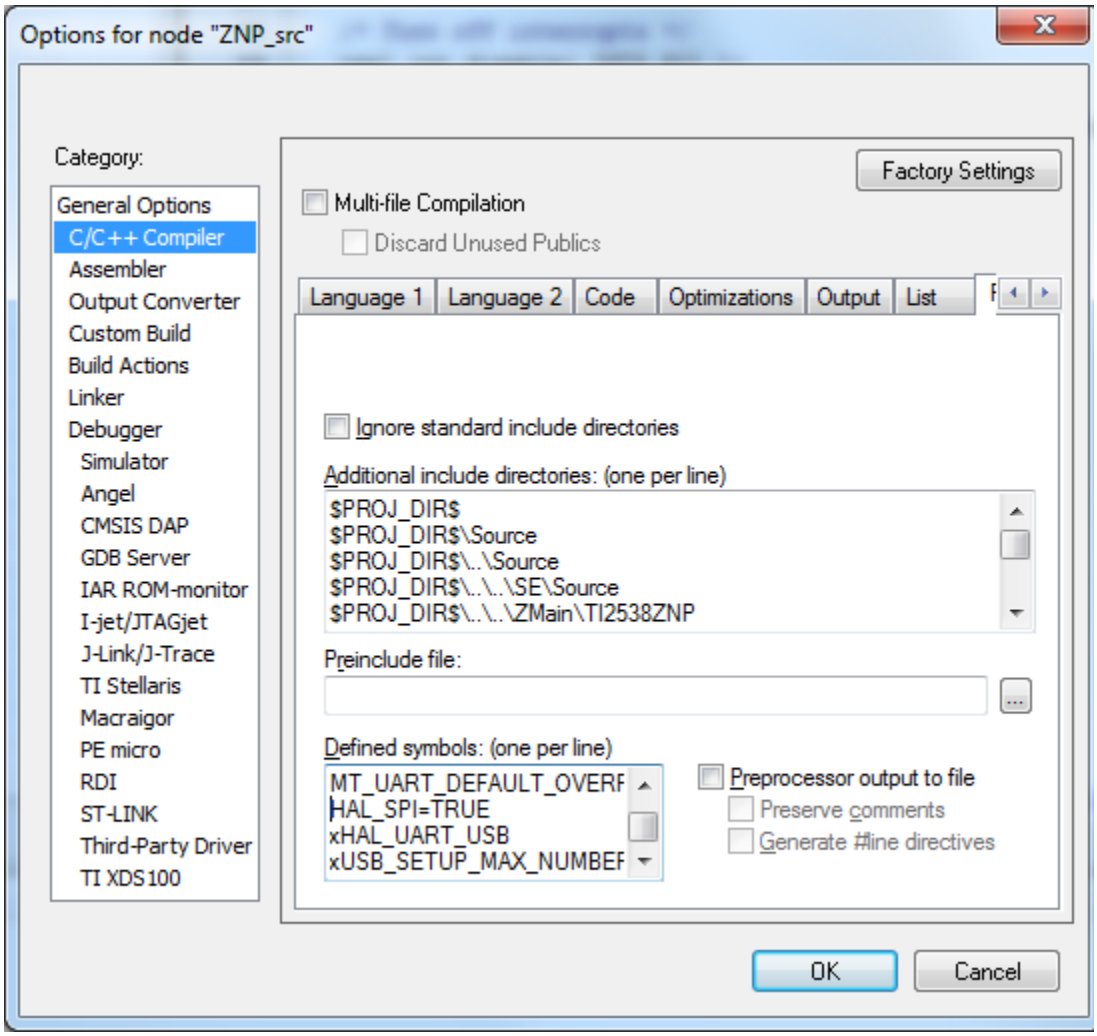


Figure 7 IAR Setup foc CC2538 ZNP

2.2.3 SPI Transport

2.2.3.1 Configuration

The following SPI configuration is supported:

- SPI slave.
- Clock speed up to 2 MHz.
- Clock polarity 1 and clock phase 1 on CC2538.
- Bit order MSB first.

2.2.3.2 Frame Format

SPI transport uses the general frame format described in 2.2.5.

2.2.3.3 Signal Description

The following standard SPI signals are used:

- CLK: Serial clock.
- CS: Chip select.
- MOSI: Master-output slave-input data.
- MISO: Master-input slave-output data.

Two additional signals are required for SPI transaction handling and power management:

- MRDY: Master ready, an active low signal. This signal is set by the application processor when it has data ready to send to the CC2538. This signal can either be controlled independently or it can be hardwired to the slave select signal.
- SRDY: Slave ready, a bi-modal signal. This signal is set by the CC2538 when it is ready to receive or send data. When set low, it indicates the CC2538 is ready to receive data. When set high during an SPI POLL or SREQ transaction it indicates the CC2538 is ready to send data. When set high during an SPI AREQ transaction it indicates the CC2538 is done receiving data.

2.2.3.4 Signal Operation

The signals operate according to the following rules:

1. The application processor initiates a transaction by setting MRDY low and then waits for SRDY to go low.
2. The application processor shall never set MRDY high to end a transaction before all bytes of the frame have been transferred.
3. When receiving a POLL or SREQ, the CC2538 shall set SRDY high when it has data ready for the application processor.
4. When receiving an AREQ, the CC2538 shall set SRDY high when all bytes of the frame have been received.

2.2.3.5 Protocol Scenarios

2.2.3.5.1 AREQ Command

The following figure shows an AREQ command sent from the application processor to the CC2538.

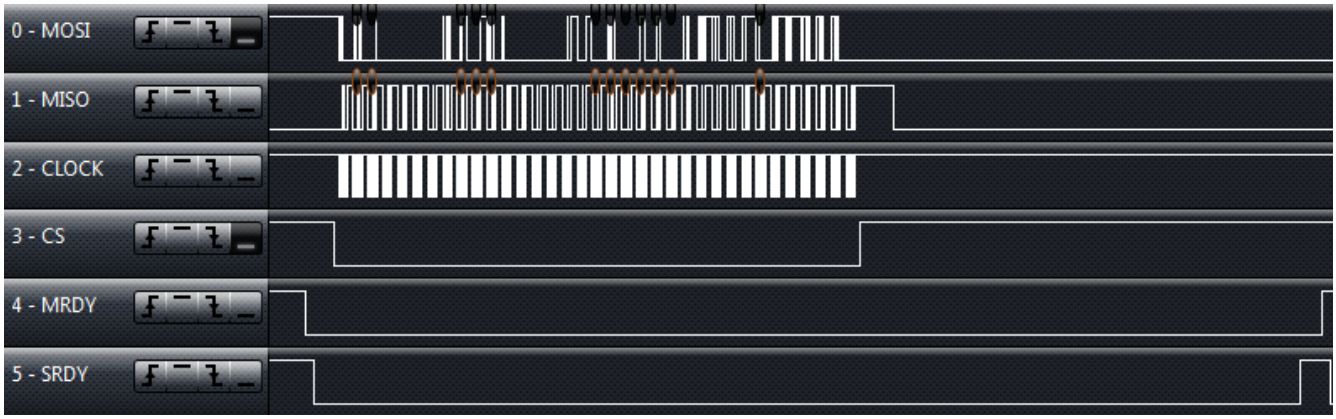


Figure 8 AREQ Command

The following sequence of events occurs on the application processor and CC2538:

1. Application processor has an AREQ frame to send. Set MRDY low and wait for SRDY to go low.
2. CC2538 receives falling edge of MRDY. When ready to receive data set SRDY low.
3. Application processor reads SRDY low. Then set CS low. Start data transmission.
4. Application processor transmits data until frame is complete. Then set CS high.
5. CC2538 receives data until frame is complete.
6. Application processor waits for SRDY to go high.
7. CC2538 receives complete frame and sets SRDY high.
8. Application processor reads SRDY high. Set MRDY high.

2.2.3.5.2 POLL Command

The following figure shows a POLL command sent from the application processor to the CC2538-ZNP.

Figure 9 POLL command

The following sequence of events occurs on the application processor and CC2538:

1. CC2538 has an AREQ frame to send. When ready to receive data set SRDY low.
2. Application processor detects SRDY low and sets MRDY low. Prepare POLL command and set CS low and start data transmission.
3. Application processor transmits data until frame is complete. Then raise the CS.
4. CC2538 receives data until frame is complete.
5. Application processor waits for SRDY to go high.

6. CC2538 prepares AREQ frame for transmission. When ready to transmit set SRDY high.
7. Application processor reads SRDY high. Set CS low and start data reception.
8. Application processor receives data until frame is complete.
9. CC2538 transmits data until frame is complete.
10. Application processor receives complete frame. Then raise CS high. Set MRDY high.

2.2.3.5.3 SREQ Command

The following figure shows a SREQ command sent from the application processor to the CC2538-ZNP.

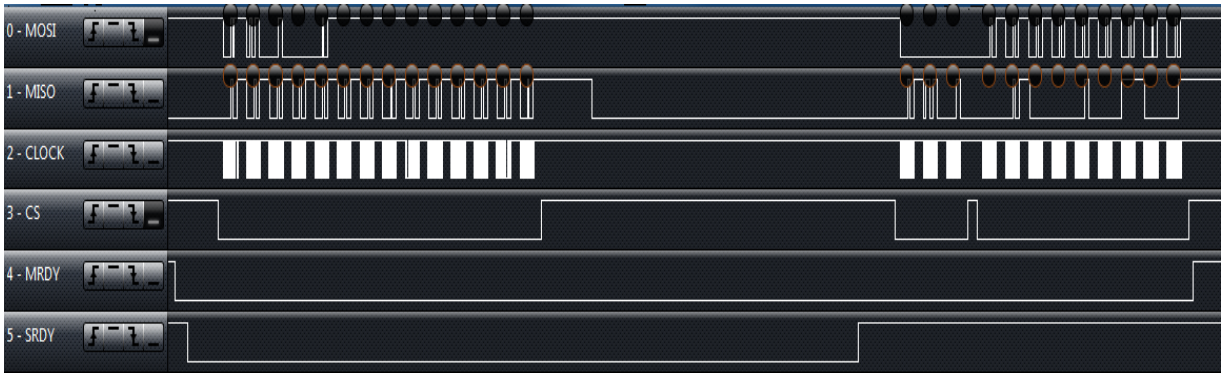


Figure 10 SREQ command

The following sequence of events occurs on the application processor and CC2538-ZNP:

1. Application processor has an SREQ frame to send. Set MRDY low and wait for SRDY to go low.
2. CC2538 receives falling edge of MRDY. When ready to receive data set SRDY low.
3. Application processor reads SRDY low. Start data transmission.
4. Application processor Sets CS low and transmits data until frame is complete, then sets CS high.
5. CC2538 receives data until frame is complete.
6. Application processor waits for SRDY to go high.
7. CC2538 processes SREQ command and executes function
8. CC2538 prepares SRSP frame. When ready to transmit data set SRDY high.
9. Application processor reads SRDY high. Start data reception.
10. Application processor, sets CS low and receives data until frame is complete, then sets CS high.
11. CC2538 transmits data until frame is complete.
12. Application processor receives complete frame. Set MRDY high.

2.2.3.5.4 ZNP SPI Error Recovery

The ZNP SPI protocol is a half duplex protocol. Though SPI, is full duplex, valid bytes are either on MOSI or MISO. When the Slave is transmitting bytes out on MISO, the CC2538 ZNP Slave depends on MRDY de-assert for End of transmission. That is when the Master de-asserts MRDY, the Slave considers this end of transmit and moves to the next stae. Hence, even if the Master is slow to de-assert the MRDY, the Slave will not move to the next transaction

If there is ever an RX FIFO overlflow on the Slave, the Slave will raise the SRDY and immediately transmit out zeros. Though the current packet is lost, this ensures that the next packet is received.

2.2.4 UART Transport

2.2.4.1 Configuration

The following UART configuration is supported:

- Baud rate: 115200
- Hardware (RTS/CTS) flow control.
- 8-N-1 byte format.

2.2.4.2 Frame Format

Please refer to section 2.1.3.2.

2.2.4.3 Sample FCS Calculation

Please refer to section 2.1.3.3 for the sample code.

2.2.4.4 Signal Description

The following standard UART signals are used:

- TX: Transmit data.
- RX: Receive data.
- CTS: Clear to send.
- RTS: Ready to send.
- The MRDY and SRDY signals are not used with UART transport.

Figure below shows the RTS/CTS flow control connections to the host processor. On the CC2538, RTS and CTS are active-low signals. The RT output is driven low when the receive register is empty and reception is enabled. Transmission of a byte does not occur before the CTS input goes low.

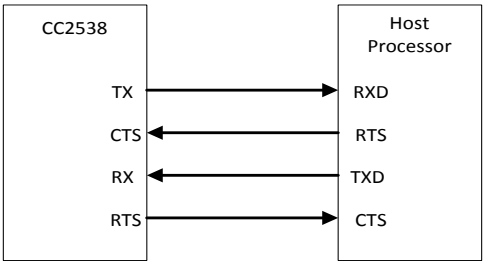


Figure 11 RTS/CTS Flow Control Connections

2.2.4.5 Signal Operation

UART transport sends and receives data asynchronously. Data can be sent and received simultaneously and the transfer of a frame can be initiated at any time by either the application processor or the CC2538.

2.2.5 General Frame Format

Please refer to section 2.1.3.4.

2.2.6 Initialization Procedures

2.2.6.1 CC2538-ZNP power-up procedure

The recommended power-up procedure is as follows:

1. Application processor and CC2538 power up.
2. For ZNP_ALT configuration, the application processor sets CC2538 EM_RESET pin low, holding CC2538 in reset. Please note that step 2 is only for ZNP Alternate configuration.
3. The application processor initializes its UART or SPI interface.
4. For ZNP_ALT configuration, the application processor sets CC2538 EM_RESETpin high and CC2538 starts operation. Please ignore this step if not using ZNP_ALT preprocessor define.
5. Application processor receives the SYS_RESET_IND message using the POLL command. When SPI transport is used CC2538 will set SRDY low to indicate the message is available and the application processor should retrieve the message.
6. The application processor receives the `SYS_RESET_IND` message.

The CC2538-ZNP can be reset when the application processor sends a `SYS_RESET_REQ` message.

2.3 CC2530

2.3.1 Network processor signals

The figure below shows how an application processor interfaces with the CC2530.

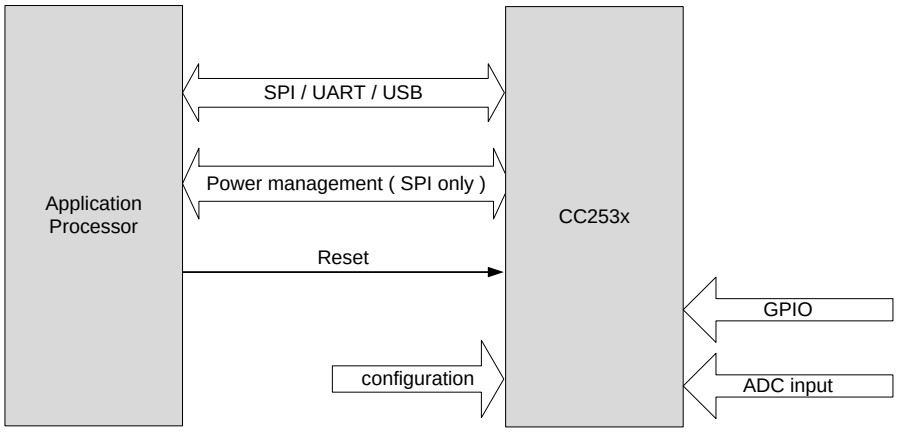


Figure 12 CC2530 Interface

The CC2530-ZNP uses the following signals for the hardware interface

- **MI/MO/C/SS for SPI, and RX/TX/RT/CT for UART:** These are the standard signals used for SPI or UART communication. See sections 2.3.3.3 (for SPI) and 2.3.4.4 (for UART) for details.
- **SRDY:** This signal is asserted by the CC2530 for power management and transaction control when using SPI transport. The application processor can use a regular GPIO pin to poll the status of this signal, or connect it to a GPIO with edge configurable interrupt capability. See section 2.3.3.3 for details.
- **MRDY:** This signal is asserted by the application processor for power management and transaction control when using SPI transport. This is typically hardwired to the SS pin and does not have to be controlled by a separate GPIO from the application processor. See section 2.3.3.3 for details.
- **RESET:** This signal is used by the application processor to reset the CC2530.
- **CFG0, CFG1:** These two signals are used to configure the CC2530-ZNP. The CC2530-ZNP reads these signals at power up and configures its operation accordingly. See section 2.3.1.1.1 for details.

2.3.1.1 Pin Configurations

2.3.1.1.1 Configuration pins

The CC2530-ZNP project reads the two hardware configuration pins at powerup and configures itself accordingly.

The CFG0 pin is used to indicate the presence (if pin is high) or absence of the 32kHz crystal connected to the CC2530-ZNP. This is the sleep crystal that is used to maintain accurate timing when the device is in sleep mode. The advantage of using this instead of the internal 32kHz oscillator is that it typically provides faster wakeup time for sleep and a lower power consumption during this time. If this crystal is not populated, then the CC2530 can use the internal RC oscillator.

If the CFG1 pin is high, the CC2530-ZNP will use the SPI transport mode in the main pin configuration listed below. Otherwise, it will use the UART transport mode in the alternate pin

configuration listed below. The ZNP Kit pin configuration is used by the ZNP kit target board. The pin-out diagram of the CC2530 can be found in [R5].

2.3.1.1.2 Main pin configuration

CC2530-ZNP signal	CC2530 PIN	CC2530 NAME	*Direction (on C2530)*
SS / CT	6	P1_4	*In*
C / RT	5	P1_5	*In / Out*
MO / TX	38	P1_6	*In / Out*
MI / RX	37	P1_7	*Out / In*
RESET	20	RESET_N	*In*
MRDY	16	P0_3	*In*
SRDY	15	P0_4	*Out*
PAEN	9	P1_1	*Out*
EN	7	P1_3	*Out*
HGM	12	P0_7	*Out*
CFG0	8	P1_2	*In*
CFG1	36	P2_0	*In*
GPIO0/AIN0	19	P0_0	*Configurable*
GPIO1/AIN1	18	P0_1	*Configurable*
GPIO2	13	P0_6	*Configurable*
GPIO3	*11*	*P1_0*	*Configurable*

2.3.1.1.3 Alternate pin configuration

CC2530-ZNP signal	CC2530 PIN	CC2530 NAME	*Direction (on C2530)*
SS / CT	15	P0_4	*In*
C / RT	14	P0_5	*In / Out*
MO / TX	16	P0_3	*In / Out*
MI / RX	17	P0_2	*Out / In*
RESET	20	RESET_N	*In*
MRDY	38	P1_6	*In*
SRDY	37	P1_7	*Out*
PAEN	9	P1_1	*Out*
EN	6	P1_4	*Out*
HGM	12	P0_7	*Out*
CFG0	8	P1_2	*In*
CFG1	36	P2_0	*In*
GPIO0/AIN0	19	P0_0	*Configurable*
GPIO1/AIN1	18	P0_1	*Configurable*
GPIO2	13	P0_6	*Configurable*
GPIO3	*11*	*P1_0*	*Configurable*

2.3.1.1.4 ZNP Kit pin configuration

CC2530-ZNP signal	CC2530 PIN	CC2530 NAME	*Direction (on C2530)*
SS / CT	15	P0_4	*In*
C / RT	14	P0_5	*In / Out*
MO / TX	16	P0_3	*In / Out*
MI / RX	17	P0_2	*Out / In*
RESET	20	RESET_N	*In*
MRDY	36	P2_0	*In*
SRDY	11	P1_0	*Out*
PAEN	9	P1_1	*Out*
EN	6	P1_4	*Out*
HGM	12	P0_7	*Out*
CFG0	19	P0_0	*In*
CFG1	18	P0_1	*In*
GPIO0	13	P0_6	*Configurable*
GPIO1	12	P0_7	*Configurable*
GPIO2	38	P1_6	*Configurable*
GPIO3	*37*	*P1_7*	*Configurable*

2.3.1.1.5 USB pin configuration

This is only available when used with the CC2531 chip. In this configuration, the CC2530-ZNP will use the USB transport with the alternate pin configuration. The pin-out of the CC2531 can be found in the datasheet [R6]. The USB transport exposes the CDC (communication device class) class USB interface and exposes a virtual COM port to the host. The host processor would then access this device as a regular COM port device and communicate with the ZNP using the UART Transport.

2.3.2 Interface Configuration

The CC2530-ZNP supports SPI, UART, or USB interface to the application processor.

2.3.2.1 IAR project configuration

The CC2530-ZNP IAR project that is included in the ZStack software package has two project configurations – CC2530-ZNP and CC2531-ZNP. As the name indicates, the configurations are intended for use with the CC2530 and CC2531 (USB) chips.

2.3.3 SPI Transport

2.3.3.1 Configuration

The following SPI configuration is supported:

- SPI slave.
- Clock speed up to 4 MHz.
- Clock polarity 0 and clock phase 0 on CC2530.
- Bit order MSB first.

2.3.3.2 Frame Format

SPI transport uses the general frame format described in 2.2.5.

2.3.3.3 Signal Description

The following standard SPI signals are used:

- C: Serial clock.
- SS: Slave select.
- MO: Master-output slave-input data.
- MI: Master-input slave-output data.

Two additional signals are required for SPI transaction handling and power management:

- MRDY: Master ready, an active low signal. This signal is set by the application processor when it has data ready to send to the CC2530. This signal can either be controlled independently or it can be hardwired to the slave select signal. The RPC sequence diagrams in this document assume MRDY is hardwired to SS.
- SRDY: Slave ready, a bi-modal signal. This signal is set by the CC2530 when it is ready to receive or send data. When set low, it indicates the CC2530 is ready to receive data. When set high during an SPI POLL or SREQ transaction it indicates the CC2530 is ready to send data. When set high during an SPI AREQ transaction it indicates the CC2530 is done receiving data.

2.3.3.4 Signal Operation

The signals operate according to the following rules:

1. The application processor initiates a transaction by setting MRDY low and then waits for SRDY to go low.
2. The application processor shall never set MRDY high to end a transaction before all bytes of the frame have been transferred.
3. When receiving a POLL or SREQ, the CC2530 shall set SRDY high when it has data ready for the application processor.
4. When receiving an AREQ, the CC2530 shall set SRDY high when all bytes of the frame have been received.

2.3.3.5 Protocol Scenarios

2.3.3.5.1 AREQ Command

The following figure shows an AREQ command sent from the application processor to the CC2530.

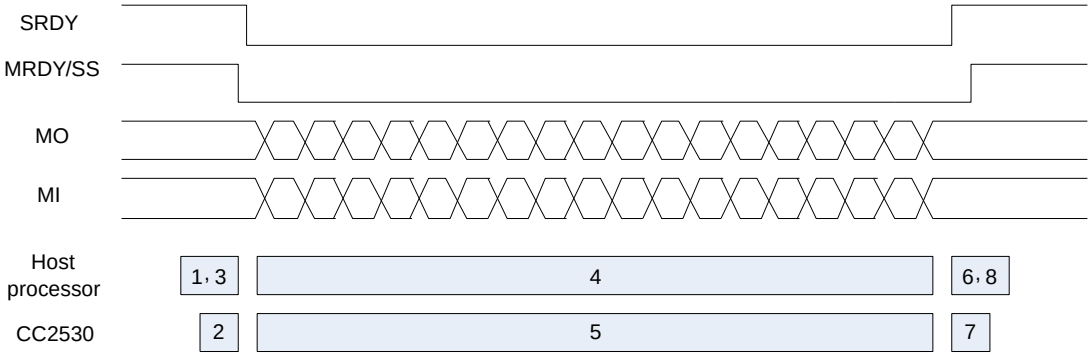


Figure 13 AREQ Command

The following sequence of events occurs on the application processor and CC2530:

1. Application processor has an AREQ frame to send. Set MRDY low and wait for SRDY to go low.
2. CC2530 receives falling edge of MRDY. When ready to receive data set SRDY low.
3. Application processor reads SRDY low. Start data transmission.
4. Application processor transmits data until frame is complete.
5. CC2530 receives data until frame is complete.
6. Application processor waits for SRDY to go high.
7. CC2530 receives complete frame and sets SRDY high.
8. Application processor reads SRDY high. Set MRDY high.

2.3.3.5.2 POLL Command

The following figure shows a POLL command sent from the application processor to the CC2530-ZNP.

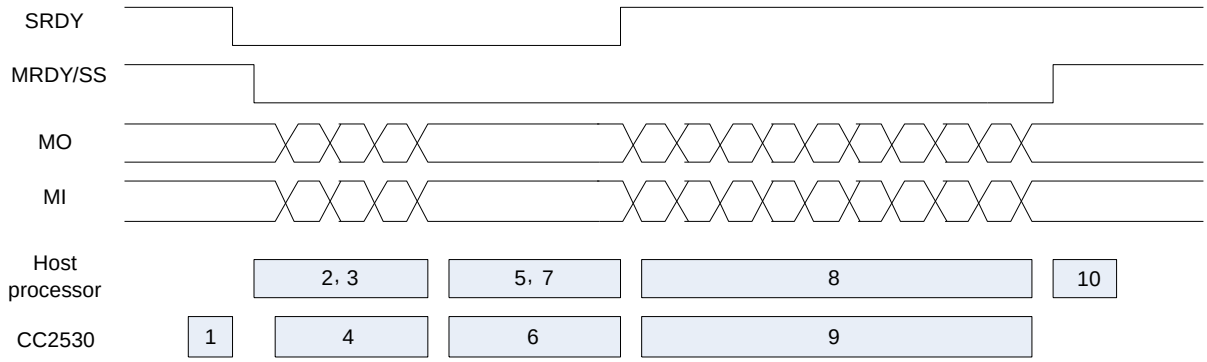


Figure 14 POLL command

The following sequence of events occurs on the application processor and CC2530:

1. CC2530 has an AREQ frame to send. When ready to receive data set SRDY low.
2. Application processor detects SRDY low and sets MRDY low. Prepare POLL command and start data transmission.
3. Application processor transmits data until frame is complete.
4. CC2530 receives data until frame is complete.
5. Application processor waits for SRDY to go high.
6. CC2530 prepares AREQ frame for transmission. When ready to transmit set SRDY high.
7. Application processor reads SRDY high. Start data reception.
8. Application processor receives data until frame is complete.
9. CC2530 transmits data until frame is complete.
10. Application processor receives complete frame. Set MRDY high.

2.3.3.5.3 SREQ Command

The following figure shows a SREQ command sent from the application processor to the CC2530-ZNP.

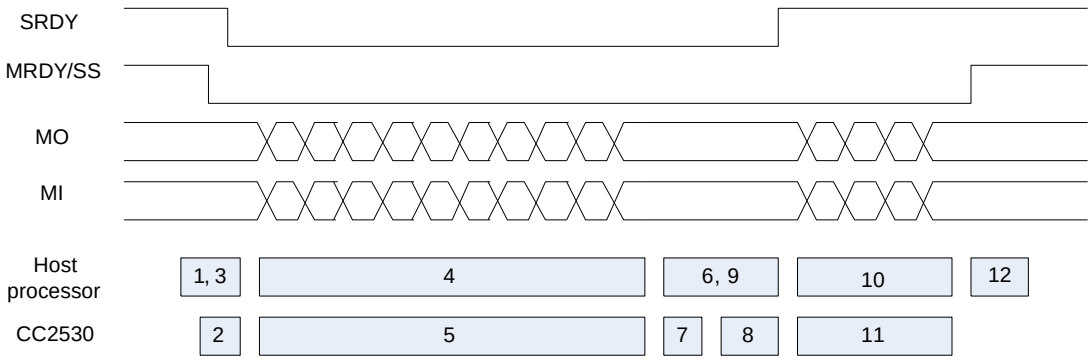


Figure 15 SREQ command

The following sequence of events occurs on the application processor and CC2530-ZNP:

1. Application processor has an SREQ frame to send. Set MRDY low and wait for SRDY to go low.
2. CC2530 receives falling edge of MRDY. When ready to receive data set SRDY low.
3. Application processor reads SRDY low. Start data transmission.
4. Application processor transmits data until frame is complete.
5. CC2530 receives data until frame is complete.
6. Application processor waits for SRDY to go high.
7. CC2530 processes SREQ command and executes function
8. CC2530 prepares SRSP frame. When ready to transmit data set SRDY high.
9. Application processor reads SRDY high. Start data reception.
10. Application processor receives data until frame is complete.
11. CC2530 transmits data until frame is complete.
12. Application processor receives complete frame. Set MRDY high.

2.3.4 UART Transport

2.3.4.1 Configuration

The following UART configuration is supported:

- Baud rate: 115200
- Hardware (RTS/CTS) flow control.
- 8-N-1 byte format.

2.3.4.2 Frame Format

Please refer to section 2.1.3.2.

2.3.4.3 Sample FCS Calculation

Please refer to section 2.1.3.3 for the sample code.

2.3.4.4 Signal Description

The following standard UART signals are used:

- TX: Transmit data.
- RX: Receive data.
- CT: Clear to send.
- RT: Ready to send.
- The MRDY and SRDY signals are not used with UART transport.

Figure 16 shows the RTS/CTS flow control connections to the host processor. On the CC2530, RT and CT are active-low signals. The RT output is driven low when the receive register is empty and reception is enabled. Transmission of a byte does not occur before the CT input goes low.

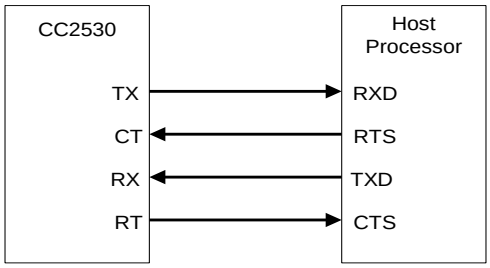


Figure 16 RTS/CTS Flow Control Connections

2.3.4.5 Signal Operation

UART transport sends and receives data asynchronously. Data can be sent and received simultaneously and the transfer of a frame can be initiated at any time by either the application processor or the CC2530.

2.3.5 General Frame Format

Please refer to section 2.1.3.4.

2.3.6 Initialization Procedures

2.3.6.1 CC2530-ZNP power-up procedure

The recommended power-up procedure is as follows:

1. Application processor and CC2530 power up.
2. Application processor sets CC2530 RESET_N pin low, holding CC2530 in reset.
3. The application processor sets the optional CC2530 CFG0 and CFG1 pins (if these pins are controlled by the application processor).
4. The application processor initializes its UART or SPI interface.
5. The application processor sets CC2530 RESET_N pin high and CC2530 starts operation.
6. Application processor receives the SYS_RESET_IND message using the POLL command. When SPI transport is used CC2530 will set SRDY low to indicate the message is available and the application processor should retrieve the message.
7. The application processor receives the `SYS_RESET_IND` message.

If the CC2530-ZNP device was configured as an end-device (and using SPI transport), it will automatically enter low power state after the application processor retrieves the `SYS_RESET_IND` command from the CC2530.

The CC2530-ZNP can also be reset when the application processor sends a `SYS_RESET_REQ` message. However, resetting CC2530 with the RESET_N pin is recommended because it is faster and more reliable.

3 ZNP software command interface

The ZNP software command interface is sub-divided into the following categories

- The SYS interface (MT_SYS) provides the application processor with a low level interface to the ZNP hardware and software.
- The Simple API interface (MT_SAPI) is a simplified ZigBee interface that can be used to quickly create simple ZigBee compliant networked applications. It allows for easy device configuration, network formation, binding and data transfer. However, a limitation of the Simple API is that it can only be used with one application registered endpoint. Therefore, it is recommended that applications that support multiple endpoints use the AF interface.
- The AF (MT_AF) and ZDO (MT_ZDO) interfaces feature the complete ZigBee interface and can be used to create a full range of ZigBee compliant applications. The AF (Application Framework) interface allows the application processor to register its application with the ZNP and send and receive data. The ZDO (ZigBee Device Object) interface provides various ZigBee management functions like device and service discovery.
- The UTIL (MT_UTIL) interface provides support functionalities such as setting PanId, getting device info, getting NV info, subscribing callbacks…etc.

3.1 ZNP startup procedure

After executing the power-up procedure, the host processor must call some mandatory APIs before executing any APIs that invoke ZigBee over-the-air messaging. Not following this sequence could result in unexpected behaviour. The recommended startup procedure is as follows:

1. The host processor must use the ZB_WRITE_CONFIGURATION command to configure at the minimum the ZCD_NV_LOGICAL_TYPE, ZCD_NV_PAN_ID, and ZCD_NV_CHANLIST configuration items.
2. If the Simple API is used, the ZB_APP_REGISTER_REQUEST command should be sent by the host processor to register the application endpoint.
3. The ZB_START_REQUEST command should be sent by the host processor to either form a network (if the device is a coordinator) or join a network (if the device is a router or end device).
4. The host processor should then wait for the ZB_START_CONFIRM command with a status of ZB_SUCCESS before performing any other API operations.
5. If the Simple API is not used after performing step 1, the AF_REGISTER command should be sent by the host processor to register the application endpoint.
6. The ZDO_STARTUP_FROM_APP command should be sent by the host processor to either form a network (if the device is a coordinator) or join a network (if the device is a router or end device).
7. The host processor should then wait for the ZDO_STATE_CHANGE_IND command with a status of DEV_ZB_COORD, DEV_ROUTER, or DEV_END_DEVICE before performing any other API operations.

3.2 Example Message Exchange

The following sequence chart is provided as a simple example of a message exchange between a Host and ZNP. In this example the following (generalized) events take place:

1. The ZNP is reset.
2. The host writes some configuration data to the ZNP.
3. An endpoint in the host is registered with the ZNP.
4. In this case, the ZNP is a network coordinator and is started via the SAPI “Start” request.
5. Another device joins the network, indicated by the ZDO Device indications.
6. Data is exchanged between the Host+ZNP and joining device through AF Data Req’s and AF Incoming Messages.

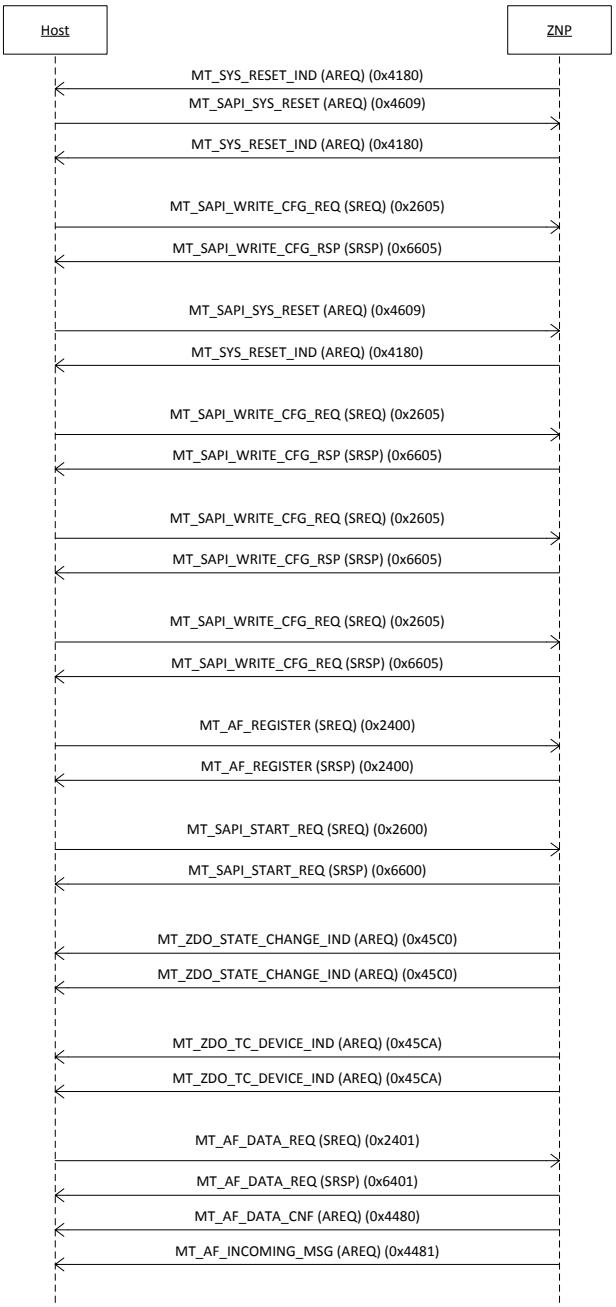


Figure 17 Example Message Sequence Chart

3.3 Return Values

The status parameter that is returned from the ZNP device may take one of the following values:

Name	Value
ZSuccess	*0x00*
ZFailure	*0x01*
ZInvalidParameter	*0x02*
NV_ITEM_UNINIT	*0x09*
NV_OPER_FAILED	*0x0a*
NV_BAD_ITEM_LEN	*0x0c*
ZMemError	*0x10*
ZBufferFull	*0x11*
ZUnsupportedMode	*0x12*
ZMacMemError	*0x13*
zdoInvalidRequestType	*0x80*
zdoInvalidEndpoint	*0x82*
zdoUnsupported	*0x84*
zdoTimeout	*0x85*
zdoNoMatch	*0x86*
zdoTableFull	*0x87*
zdoNoBindEntry	*0x88*
ZSecNoKey	*0xa1*
ZSecMaxFrmCount	*0xa3*
ZApsFail	*0xb1*
ZApsTableFull	*0xb2*
ZApsIllegalRequest	*0xb3*
ZApsInvalidBinding	*0xb4*
ZApsUnsupportedAttrib	*0xb5*
ZApsNotSupported	*0xb6*
ZApsNoAck	*0xb7*
ZApsDuplicateEntry	*0xb8*
ZApsNoBoundDevice	*0xb9*
ZNwkInvalidParam	*0xc1*
ZNwkInvalidRequest	*0xc2*
ZNwkNotPermitted	*0xc3*
ZNwkStartupFailure	*0xc4*
ZNwkTableFull	*0xc7*
ZNwkUnknownDevice	*0xc8*
ZNwkUnsupportedAttribute	*0xc9*
ZNwkNoNetworks	*0xca*
ZNwkLeaveUnconfirmed	*0xcb*
ZNwkNoAck	*0xcc*
ZNwkNoRoute	*0xcd*
ZMacNoACK	*0xe9*

3.4 Additional Information

For additional details of the individual commands, please refer to the Z-Stack Monitor and Test API [R2].

4 General Information

4.1 Document History

Table 1: Document History

Revision	Date	Description/Changes
1.0	02/20/2015	Initial version