CMT2300A FIFO and Packet Format Usage Guideline

Summary

This article describes the CMT2300A FIFO, packet format and the working principle of the interrupt system. When the article introduces the contents of the configuration register, it will correspond to the parameters that can be entered on the RFPDK to facilitate the user configuration.

The part numbers covered by this document are as shown below.

Table1. Part Numbers Covered by This Document

Part No.	Frequency	Modem	Function	Configuration	Package
CMT2300A	127 - 1020MHz	(G)FSK/OOK	Transceiver	Register	QFN16

Before reading this document, it is recommended that reading the《AN142 - CMT2300A Quick Start Guideline》, that will be make it easy to understand.

Table of contents

Summary 1
1. FIFO Working Principle 3
1.1 FIFO Related Register 3
1.2 FIFO Working Mode 5
1.3 FIFO Interrupt Timing 7
1.4 FIFO Application Scenes 7
1.4.1 Application scene 1: Receive the data at RX 8
1.4.2 Application scene 2: Fill in the data beforehand and enter the TX transmitting 8
1.4.3 Application scene 3: After entering the TX, fill in the data while transmitting them. 9
1.4.4 Application scene 4: Send repeatedly the same or same set of packets each time 9
1.4.5 Application scene 5: A packet is sent several times apart 9
2. Packet Format Introduction 10
2.1 Data Mode Configuration 10
2.2 Preamble Configuration 11
2.3 Sync Word Configuration 12
2.4 Packet Overall Configuration 15
2.5 Node ID Configuration 17
2.6 FEC Configuration 21
2.7 CRC Configuration 22
2.8 Codec Configuration 24
2.9 Tx Packet Specific Configuration 26
2.10 Direct Tx Mode 27
3. GPIO and Interrupt 28
3.1 GPIO Configuration 28
3.2 Interrupt Configuration and Mapping 29
3.3 Antenna TX / RX Switching Control 36
4. Document Modification Record 37
5. Contact Information 40

1. FIFO Working Principle

1.1 FIFO Related Register

The corresponding RFPDK interface and parameters are as below:

Figure 1. FIFO RFPDK Interface

Table 2. FIFO Related Parameter

Register Bit RFPDK Parameter	Register Bit
Data Mode	DATA_MODE <1:0>
RFPDK does not display the parameter and it is flexibly configured by the user in the application.	FIFO_TH <6:0>
The parameter is automatically calculated based on the TX packet number. When the TX packet number is greater than 1, the parameter is set to 1.	FIFO_AUTO_RES_EN

The contents and explanations of the register can be seen in the following table.

Table 3. Register Located in the Configuration Bank:

Register Name	Bits	R/W	Bit Name	Function Description
CUS_PKT1 (0x38)	1:0	RW	DATA_MODE <1:0>	Data processing mode: 0：Direct Mode（Default） 1：NA 2：Packet Mode 3：NA
CUS_PKT29 (0x54)	7	RW	FIFO_AUTO_RES_EN	Each time it sends a package, it automatically restores TX FIFO. If you enter TX each time to send repeatedly more than 1 packet (TX_PKT_NUM> 0), this bit must be set to 1.
	6:0	RW	FIFO_TH <6:0>	FIFO threshold, the unit is byte. For RX, when the unread data exceeds this threshold, the RX_FIFO_TH_FLG will be set to 1. For TX, when no sending data is less than this threshold, TX_FIFO_TH_FLG will be set to 0.

Register Name	Bits	R/W	Bit Name	Function Description
				When FIFO_MERGE_EN = 0, the effective range is 1 to 31. When FIFO_MERGE_EN = 1, the effective range is 1 to 63.

Table 4. Register Located in the Control Bank 1

Register Name	Bits	R/W	Bit Name	Function Description
CUS_FIFO_CTL (0x69)	4	RW	FIFO_AUTO_CLR_DIS	0：Automatically clear RX FIFO before entering RX. 1：No auto clear.
	3	RW	FIFO_TX_RD_EN	0：TX FIFO can only be written by SPI. 1：RX FIFO can be read by SPI. This bit is valid only for TX FIFO, Except that it is available for user testing, the bit should be set to 0 at the rest.
	2	RW	FIFO_RX_TX_SEL	As a 64-byte FIFO, it can be used. 0：Used for RX FIFO; 1：Used for TX FIFO。
	1	RW	FIFO_MERGE_EN	0：The FIFO is divided into 2 separate 32-byte; 1：Merge into one 64-byte FIFO.
	0	RW	SPI_FIFO_RD_WR_SEL	0：The operation of SPI is to read FIFO; 1：The operation of SPI is to write FIFO. You must set it up before you visit FIFO.

Table 5. Register Located in the Control Bank 2

Register Name	Bits	R/W	Bit Name	Function Description
CUS_FIFO_CLR (0x6C)	2	W	FIFO_RESTORE	Users restore TX FIFO manually. Restore means resetting the read pointer. Keep the write pointer unchanged, so that TX FIFO is returned to unread state, and you can repeatedly send the data filled before.
	1	W	FIFO_CLR_RX	0：Invalid， 1：Clear RX FIFO。 After the user has set this bit to 1, it does not need to be set to 0 again, and this bit is automatically set back to 0 inside.
	0	W	FIFO_CLR_TX	0:：Invalid， 1：Clear TX FIFO After the user has set this bit to 1, it does not need to be set to 0 again, and this bit is

				automatically set back to 0 inside.
CUS_FIFO_FLAG (0x6E)	6	R	RX_FIFO_FULL_FLG	Indicates the interrupt flag bit that RX FIFO is full. 0：Invalid 1：Valid
	5	R	RX_FIFO_NMTY_FLG	Indicates the interrupt that the unread contents of RX FIFO exceed the FIFO TH. 0：Invalid 1：Valid
	4	R	RX_FIFO_TH_FLG	Indicates the interrupt that RX FIFO is filled. 0：Invalid 1：Valid
	3	R	RX_FIFO_OVF_FLG	Indicates the interrupt that RX FIFO overflows. 0：Invalid 1：Valid
	2	R	TX_FIFO_FULL_FLG	Indicates the interrupt that TX FIFO is full. 0：Invalid 1：Valid
	1	R	TX_FIFO_NMTY_FLG	Indicates the interrupt that the unread contents of TX FIFO exceed the FIFO TH. 0：Invalid 1：Valid
	0	R	TX_FIFO_TH_FLG	Indicates the interrupt that TX FIFO is filled. 0：Invalid 1：Valid
Note： The polarity of these interrupt flag bits is controlled by the INT_POLAR. That is, when INT_POLAR = 1, they are all 0 valid and 1 invalid.				

In these registers, some bits that are not associated with the FIFO are ignored here and are not introduced.

1.2 FIFO Working Mode

CMT2300A provides two separated 32-byte FIFO by default. They are used for RX and TX respectively, both are separated each other. Users can also set FIFO_MARGE_EN to 1, and then the two FIFO are merged into a 64-byte FIFO. It can be used both under TX and RX. By configuring the FIFO_RX_TX_SEL to indicate whether it is currently used as TX or RX.

In general, we would suggest that the FIFO working mode is pre configured in the STBY state, and then starting the work of TX/RX. As long as the registers locate in the configuration bank and the control bank 1, the contents can be saved in the SLEEP state. Therefore, unless you change your working mode, configuring once is enough.

Separate FIFO pre configuration

1. Set DATA_MODE <1:0> to 2 and set the data processing mode to Packet mode.
2. Set FIFO_MERGE_EN to 0.
3. Select the operation to be performed afterwards:

If you want to go to RX:

a) Configure SPI_FIFO_RD_WR_SEL to 0 (SPI read FIFO mode);
b) If the user wants RX FIFO to be automatically cleared every time he enters the RX, the FIFO_AUTO_CLR_DIS is set to 0, otherwise it is set to 1, and then manually cleared.

If you want to go to TX:

a) Configure SPI_FIFO_RD_WR_SEL to 1 (SPI write FIFO mode)

Merged FIFO pre configuration

1. Set DATA_MODE <1:0> to 2 and set the data processing mode to Packet mode.
2. Set FIFO_MERGE_EN to 1.
3. Select the operation to be performed afterwards:

If you want to go to RX:

a) Set FIFO_RX_TX_SEL to 0 (RX mode);
b) Configure SPI_FIFO_RD_WR_SEL to 0 (SPI read FIFO mode);
c) If the user wants RX FIFO to be automatically cleared every time he enters the RX, the FIFO_AUTO_CLR_DIS is set to 0, otherwise it is set to 1, and then manually cleared before starting.

If you want to go to TX:

a) Set FIFO_RX_TX_SEL to 1 (TX mode);
b) Configure SPI_FIFO_RD_WR_SEL to 1 (SPI write FIFO mode)..
c) Clear manually before starting.

When FIFO is merged, there is only one FIFO inside the chip. Because RX/TX is half duplex, one FIFO is enough. When FIFO is divided into two, they are independent of each other, and without interference, some special functions can be realized. For example, if the transmitting data is the same every time, and the RX FIFO can save the contents under SLEEP, then the TX FIFO will only need to be filled once to save time and power. Another example is that the RX FIFO is written continuously in the RX state, and the user can use the received time to fill the data to be transmitted next time into the TX FIFO in parallel. This will not disturb the work of RX FIFO, but also save time. And also this will not leave the special time to fill in the TX FIFO, but also save the power.

It is important to note that you must clear manually after you switch between TX FIFO and RX FIFO each time, otherwise it will not work properly.

In order to apply to the FIFO merging and not merging the two cases, FIFO read-write enable operations configure FIFO_RX_TX_SEL and SPI_FIFO_RD_WR_SEL according to the merging usage requirements.

```
Cmt2300_GoStby();
Cmt2300_ClearInterruptFlags();
/* Must clear FIFO after enable SPI to read or write the FIFO */
Cmt2300_EnableWriteFifo();
Cmt2300_ClearFifo();
// When merged FIFO is used, additional delay is required when the sending byte is too long.
/* The length need be smaller than 32 */
Cmt2300_WriteFifo(g_pTxBuffer, g_nTxLength);
```

FIFO read-write operation code examples see Appendix 1.

1.3 FIFO Interrupt Timing

Here, we first give the interrupt timing diagram of RX FIFO and TX FIFO. Users can refer to them to understand easily.

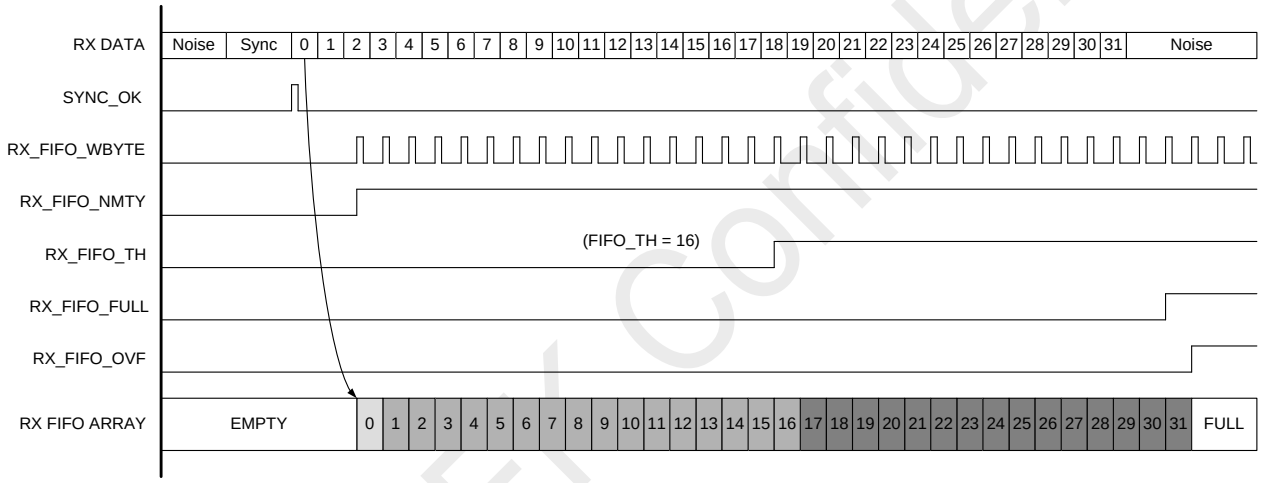

Figure2. CMT2300A RX FIFO Interrupt Timing Diagram

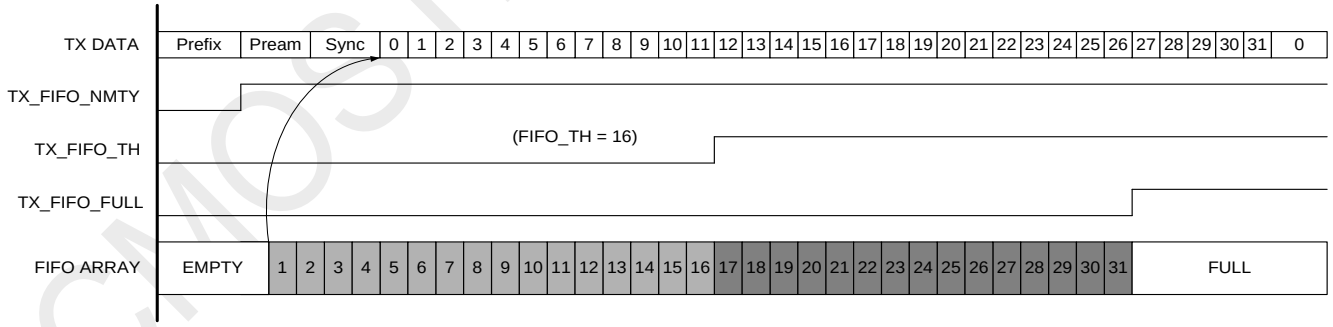

Figure3. CMT2300A TX FIFO Interrupt Timing Diagram

1.4 FIFO Application Scenes

When the FIFO is configured, you can start using it. Here are some classic application scenes. To complete the entire process of TX and RX, you also need to configure and control other things. These are introduced later. Here we will only introduce the contents related to FIFO.

1.4.1 Application scene 1: Receive the data at RX

RX FIFO is used more directly. Clear it when you enter RX every time. Enter RX and fill them if the data is received (Detect the Sync Word successfully). MCU can achieve the following several operations according to the interrupt. After you have finished, you cannot use it. Send FIFO_CLR_RX to clear before you receive it next time.

1. Detect the RX_FIFO_FULL interrupt. Once valid indicates that the FIFO has been filled, you can start reading. The appropriate packet length is just equal to the FIFO depth. Also, the user does not read the FIFO until the full packet is received.
2. Detect the RX_FIFO_TH interrupt. Once valid indicates that the FIFO has been filled in the preset data length, you can start reading. The appropriate packet length is not equal to the FIFO depth. Also, the user does not read the FIFO until the full packet is received.
3. Detect the RX_FIFO_NMTY interrupt. Once valid, read immediately until the interrupt is invalid. Read again when the interrupt is valid again. This allows you to read them while you receive them. It is suitable for the case where the packet length is greater than the FIFO depth and is suitable for the case where the packet length is less than or equal to the FIFO depth.
4. Detect the X_FIFO_WBYTE interrupt. Read immediately once it is valid. This allows you to achieve the regular operation to write a byte and then read a byte. The premise is that SPI is faster than receiving the data.

In addition, you can define how many packets are sent per sending by setting the TX_PKT_NUM <7:0>. If sending the packet is more than one, the FIFO_AUTO_RES_EN is set to 1. That is, the TX FIFO automatically clears the read pointer and goes back the unread state after each packet is sent. This allows you to repeatedly send the same packet. You don't need MCU to fill in the data again. The user can set the TX_PKT_GAP <7:0> to define the time gap between each packet, the unit is the symbol. After entering the transmitting state, the transmitter sends N packets according to these configurations. After completing, the transmitter automatically exits the TX and returns to the specified state. The status after exiting can be configured by TX_EXIT_STATE <1:0>.

If you want to achieve continuous reception, it is recommended to use the above fourth operation. This needs SPI to read quickly enough. At least it is 1.5 times faster than the rate that FIFO writes a byte. For example, if the data rate is 10KHz, the time it takes to receive a byte is about 800 us. The rate SPI reads a byte is 1.5 times faster than the rate FIFO writes a byte. The most time-consuming is 534us. SPI reads a byte for 8 SCL clock cycles, but with the added time cost before and after, we can count it according to the 10 SCL clock cycles, then each cycle is 53.4us, converted to SCL clock frequency is about 18.7 kHz. The rate of SCL is roughly 2 times the rate of data.

1.4.2 Application scene 2: Fill in the data beforehand and enter the TX transmitting

For many applications, pre fill the data packets to be transmitted into TX FIFO, and then enter the TX transmitting. The behavior of filling data is suggested to be executed in the STBY state. This scene is suitable for packet length less than or equal to FIFO depth, and the application time does not need to be very compact. After the user pre configured, he can write the data directly and judge whether the data has been fully written by detecting the interrupt of TX_FIFO_NMTY, TX_FIFO_TH or TX_FIFO_FULL. During the debugging stage, after completing the data, the user can read back the filling data according to the following methods, and

confirm whether the filling data is the correct:

1. Set FIFO_TX_RD_EN to 1 and enter the read back mode of TX FIFO.
2. Set SPI_FIFO_RD_WR_SEL to 0 and enter the read FIFO mode of SPI.
3. Read the data and confirm whether it is correct.
4. Set FIFO_TX_RD_EN to 0 and exit the read back mode of TX FIFO.
5. Set SPI_FIFO_RD_WR_SEL to 1 and enter the write FIFO mode of SPI.
6. Set FIFO_CLR_TX to 1 and clear FIFO.
7. Re write the data to be ready for transmitting.

1.4.3 Application scene 3: After entering the TX, fill in the data while transmitting them.

If the user first enters the TX state, but the TX FIFO is empty, the chip will always transmit the prefix. The content of the prefix is determined by TX_PREFIX_TYPE <1:0>, which can be 0, 1, or preamble. At the time of transmitting the prefix, the chip will have been waiting for the user to fill FIFO, until the user begin to fill in the first data byte, the chip will stop transmitting the prefix, and has been according to the rhythm of the data rate to obtain FIFO data to transmit them until the end. So in this scene, SPI is fast enough, and SCL is at least 2 times as fast as the data rate. If the speed of SPI writing cannot catch up during the transmitting data, that is, the FIFO is read out by the transmitter, transmitting will not stop, and will always send 0 until FIFO fill in the next data. But this is an improper operation, and the user should try to avoid it.

1.4.4 Application scene 4: Send repeatedly the same or same set of packets each time

Since TX FIFO saves the content in the SLEEP state, MCU just needs to fill in the data once if the data sent each time is the same. After completing the sending each time, enter STBY and set FIFO_RESTORE to 1 (Without setting 0, this bit will be cleared automatically), clear the FIFO read pointer. This means that FIFO is back in the unread state, and the data is stored in it. Entry TX next time and re send the same content, go round and begin again. If FIFO_AUTO_RES_EN has been set to 1, you don't have to manually set FIFO_RESTORE, and next time you go directly to the TX to send them.

1.4.5 Application scene 5: A packet is sent several times apart

TX FIFO not only saves the content in the SLEEP state, but also saves the pointer state. For example, with the merged FIFO, the size is 64-byte, and the user will send the four packets one after another, each packet is 16-byte. Then, users can first fill these 4 packets in FIFO in STBY, and make sure that FIFO is full, then set the data length to 16 (how to set it according to the packet format, the following will introduce.), set the TX packet number to 1, set the FIFO_AUTO_RES_EN to 0, and then enter the TX. The chip sends the first 16-byte packet, exit to STBY after completing, and then re-enter the TX, the chip sends the second packet..... until the fourth sending is completed. In this process, MCU does not need to do anything. It is only responsible for detecting the interrupt and switching the state.

2. Packet Format Introduction

CMT2300A uses the TX and RX unified configuration. It is more typical and more flexible packet format. The structure diagram is as follows:

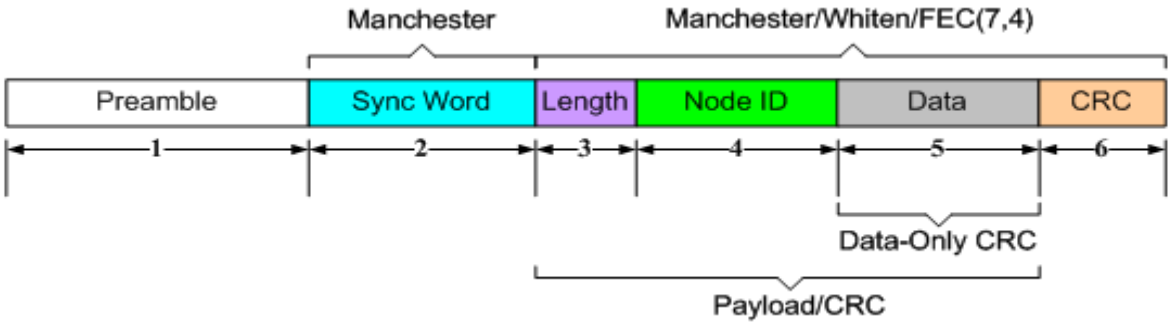

Figure 4. Packet Structure Diagram

The packet format contains six optional parts. This structure can meet the demand of most single Field structures in the market, and is not compatible with multi Field structure.

The configurable content of packet format is centered on “the baseband zone". The following will combine these registers to explain how to configure each part.

2.1 Data Mode Configuration

Data Mode refers to the external MCU to input the sending data or obtain the received data by which mode.

The corresponding RFPDK interface and parameters are as below:

Figure 5. Data Mode RFPDK Interface

Table 6. Data Mode Related Parameter

Register Bit RFPDK Parameter	Register Bit
Data Mode	DATA_MODE <1:0>
Automatically select according to Data Mode. When Data Mode is Direct, input directly the data from GPIO, and when Data Mode is Packet, obtain the data from TX FIFO.	TX_DIN_SOURCE

Table 7. Register Located in the Configuration Bank

Register Name	Bits	R/W	Bit Name	Function Description
CUS_PKT1 (0x38)	1:0	RW	DATA_MODE<1:0>	Select the Tx/Rx data mode： 0：Direct Mode（default） 1：NA 2：Packet Mode 3：NA
CUS_TX1 (0x55)	2	RW	TX_DIN_SOURCE	Select the Tx data source location： 0：TX data is obtained from TX FIFO 1：TX data is input directly from the GPIO

The difference between data mode is that:

- **Direct** –Direct mode. The RX mode only supports the preamble and sync detection. FIFO doesn't work. RX does not support any packet formats.
- **Packet** –Packet format mode. It supports all packet format configurations, FIFO works.

2.2 Preamble Configuration

The corresponding RFPDK interface and parameters are as below:

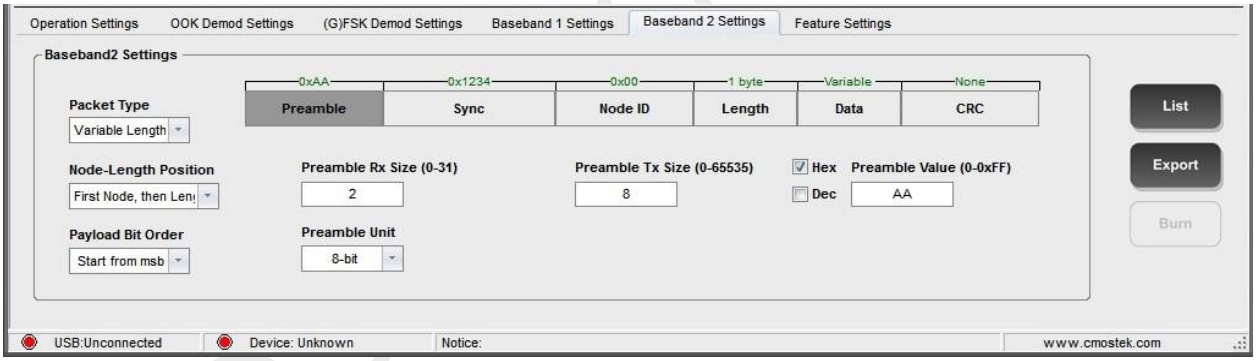

Figure 6. Preamble RFPDK Interface

Table 8. Preamble Related Parameter

Register Bit RFPDK Parameter	Register Bit
Preamble Rx Size	RX_PREAM_SIZE<4:0>
Preamble Tx Size	TX_PREAM_SIZE<15:0>
Preamble Unit	PREAM_LENG_UNIT
Preamble Value	PREAM_VALUE<7:0>

Table9. Register Located in Configuration Bank

Register Name	Bits	R/W	Bit Name	Function Description
CUS_PKT1 (0x38)	7:3	RW	RX_PREAM_SIZE<4:0>	RX mode Preamble length，can be configured to be 0-31 units in length. 0: Indicates not to detect Preamble. 1 Indicates to detect Preamble of 1 unit of length. And so on.
	2	RW	PREAM_LENG_UNIT	The unit of length of Preamble, shared by TX and RX: 0：The unit is 8bits. 1：The unit is 4 bits.
CUS_PKT2 (0x39)	7:0	RW	TX_PREAM_SIZE<7:0>	TX mode Preamble length，can be configured to be 0-65535 units in length. 0 Indicates not to send Preamble， 1 Indicates to send Preamble of 1 unit of length. And so on..
CUS_PKT3 (0x3A)	7:0	RW	TX_PREAM_SIZE<15:8>	
CUS_PKT4 (0x3B)	7:0	RW	PREAM_VALUE<7:0>	Preamble value，shared by TX and RX: 8bit is valid when PREAM_LEN_UNIT =0. Only <3:0> is valid when PREAM_LEN_UNIT =1.

For RX, the Preamble detection success will generate PREAM_OK interrupt. In addition, Preamble detection will continue throughout the receiving phase, and it is recommended that the user only detect the interrupt when necessary.

2.3 Sync Word Configuration

The corresponding RFPDK interface and parameters are as below:

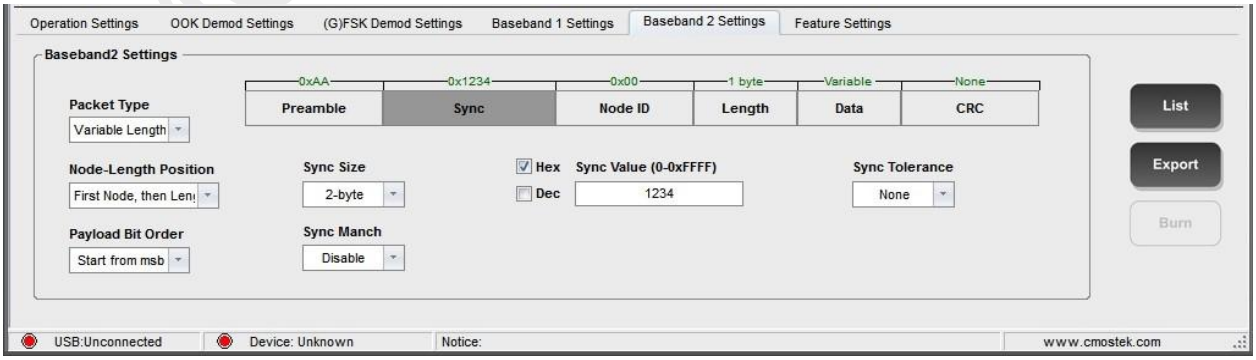

Figure 7. Sync Word RFPDK Interface

Table10. Sync Word Related Parameter

Register Bit RFPDK Parameter	Register Bit
Sync Size	SYNC_SIZE<2:0>
Sync Value	SYNC_VALUE<63:0>
Sync Tolerance	SYNC_TOL<2:0>
Sync Manch	SYNC_MAN_EN

Table11. Register Located in Configuration Bank

Register Name	Bits	R/W	Bit Name	Function Description
CUS_PKT5 (0x3C)	6:4	RW	SYNC_TOL<2:0>	Fault tolerant bits of detecting Sync Word in RX mode： 0: No permit the error. 1: Permit 1bit received error. 2: Permit 2bits received error. 3: Permit 3bits received error. 4: Permit 4bits received error. 5: Permit 5bits received error. 6: Permit 6bits received error. 7: Permit 7bits received error.
	3:1	RW	SYNC_SIZE<2:0>	Sync Word length: 0: 1 byte 1: 2 bytes 2: 3 bytes 3: 4 bytes 4: 5 bytes 5: 6 bytes 6: 7 bytes 7: 8 bytes
	0	RW	SYNC_MAN_EN	Sync Word Manchester codec enable： 0：Disable 1：Enable
CUS_PKT6 (0x3D)	7:0	RW	SYNC_VALUE<7:0>	Sync Word value, according to different SYNC_SIZE settings to fill in different registers, see the following table.
CUS_PKT7 (0x3E)	7:0	RW	SYNC_VALUE<15:8>	
CUS_PKT8 (0x3F)	7:0	RW	SYNC_VALUE<23:16>	
CUS_PKT9 (0x40)	7:0	RW	SYNC_VALUE<31:24>	
CUS_PKT10 (0x41)	7:0	RW	SYNC_VALUE<39:32>	

Register Name	Bits	R/W	Bit Name	Function Description
CUS_PKT11 (0x42)	7:0	RW	SYNC_VALUE<47:40>	
CUS_PKT12 (0x43)	7:0	RW	SYNC_VALUE<55:48>	
CUS_PKT13 (0x44)	7:0	RW	SYNC_VALUE<63:56>	

	SYNC_VALUE							
SYNC_SIZE	<63:56>	<55:48>	<47:40>	<39:32>	<31:24>	<23:16>	<15:8>	<7:0>
0	√							
1	√	√						
2	√	√	√					
3	√	√	√	√				
4	√	√	√	√	√			
5	√	√	√	√	√	√		
6	√	√	√	√	√	√	√	
7	√	√	√	√	√	√	√	√

The tick in the table indicates the register to be filled in. For example, if SYNC_SIZE is set to 1, the length is 2 bytes, the synchronous word is 0x5678, then the user will fill the value in the two registers of SYNC_VALUE<63:56> and SYNC_VALUE<55:48>. The MSB corresponds to the sixty-third bit. The LSB corresponds to the forty-eighth bit. That is filling 0x56 into SYNC_VALUE<63:56> and filling 0x78 into SYNC_VALUE<55:48>.

In addition, some applications need to Manchester encode the entire packet, but most applications only need to encode Payload. So, design a Manchester encoding enable bit alone for Sync Word.

2.4 Packet Overall Configuration

The corresponding RFPDK interface and parameters are as below:

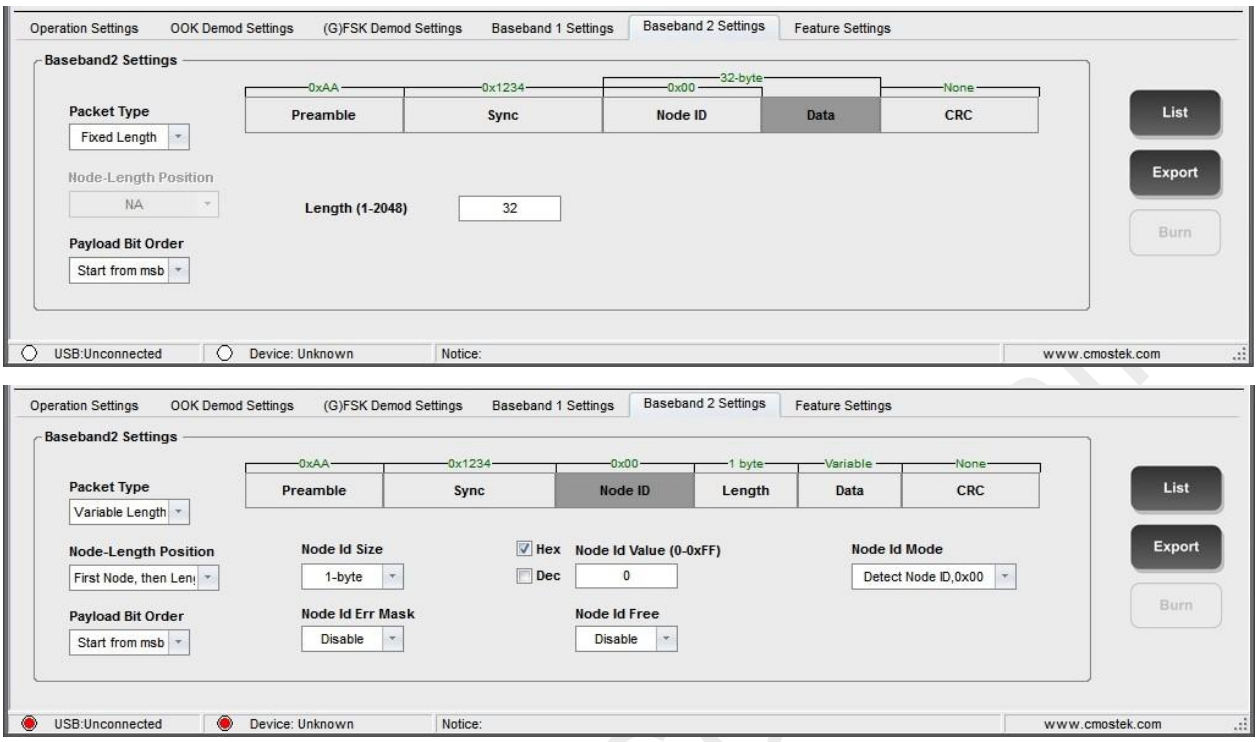

Figure 8. Packet RFPDK Interface

Table 12. Packet Related Parameter

Register Bit RFPDK Parameter	Register Bit
Packet Type	PKT_TYPE
Calculate synthetically according to each parameter. The specific method is described below.	PAYLOAD_LENG<10:0>
Node-Length Position	NODE_LENG_POS_SEL
Payload Bit Order	PAYLOAD_BIT_ORDER
No input in RFPDK. You can use it flexibly in applications, The introduction is below.	AUTO_ACK_EN

Table 13. Register Located in Configuration Bank

Register Name	Bits	R/W	Bit Name	Function Description
CUS_PKT14 (0x45)	6:4	RW	PAYLOAD_LENG<10:8>	The <10:8> bit of the 11-bit Payload length. When PKT_TYPE is set as a fixed length packet, the configurable content is 0-2047, referring to 1-2048 bytes. When PKT_TYPE is set to a variable packet, only <7:0> is valid, and the configurable length is 1-256 bytes.
	3	RW	AUTO_ACK_EN	Automatic packaging the ACK data packets enable. 0：Disable 1：Enable
	2	RW	NODE_LENG_POS_SEL	In variable packets, the position relationship between Node ID and Length Byte 0：Node ID is before length Byte. 1：Node ID is after length Byte.
	1	RW	PAYLOAD_BIT_ORDER	0: First code and decode each byte MSB of the payload+CRC. 1: First code and decode each byte LSB of the payload+CRC
	0	RW	PKT_TYPE	Packet length type 0：Fixed packet length 1：Variable packet length
CUS_PKT15 (0x46)	7:0	RW	PAYLOAD_LENG<7:0>	The <7:0> bit of the 12-bit Payload length. The explain is as above.。

The following explains in detail the meaning of PAYLOAD_BIT_ORDER.

PAYLOAD_BIT_ORDER = 1 indicates that when sending, each byte itself of Payload and CRC is sent from LSB to MSB in sequence or Manchester/Whiten encoded. On the other hand, the MSB and LSB sequence of each byte itself of the decoded Payload and the CRC must be changed when receiving, and then encode CRC again. If the configuration of RX and TX is the same, the user is unable to see the process. If users use our product to connect with other people's product, they need to understand the process and configure it properly.

PAYLOAD_BIT_ORDER = 0 indicates that this operation is not available.

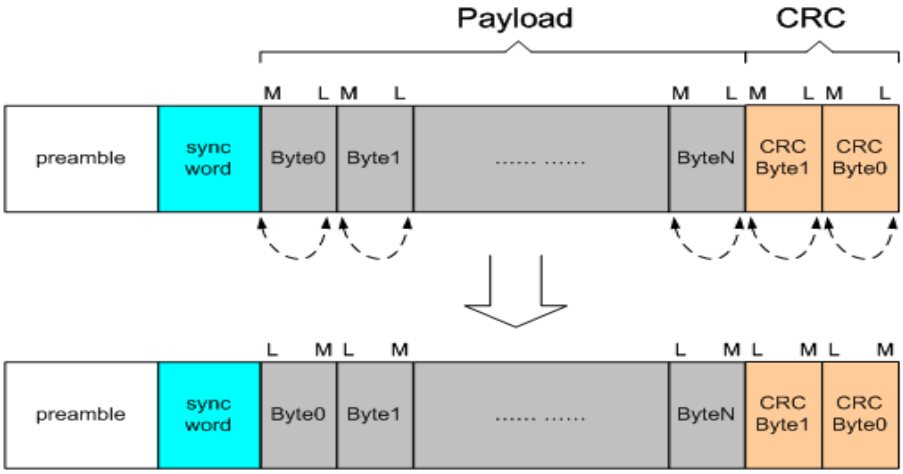

Figure 9. PAYLOAD_BIT_ORDER Operation

Explain AUTO_ACK_EN usage in detail below.

CMT2300A's AUTO ACK function does not mean that after receiving a packet, it automatically switches back to the TX mode and send back the ACK packet. This control mode is not supported within the chip. The actual usage is that when MCU sets AUTO_ACK_EN to 1, the data packet format is automatically configured as the ACK packet format within the chip, as follows:

Figure 10. ACK Packet Format

This packet only contains Preamble and Sync ID. Then, the external MCU needs to switch the chip to TX mode for sending. The sending content is the packet above. After the sending is completed, MCU needs to set AUTO_ACK_EN to 0 first and then perform other operations.

2.5 Node ID Configuration

The corresponding RFPDK interface and parameters are as below:

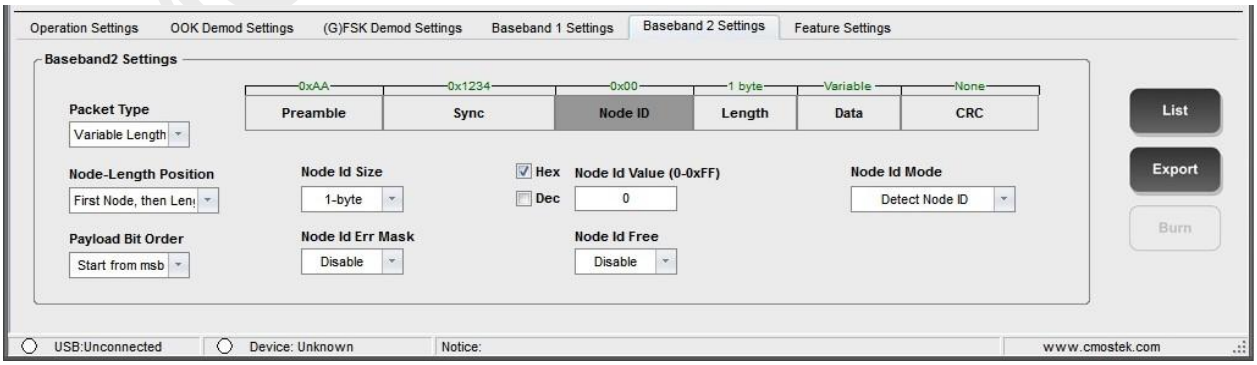

Figure 11. Node ID RFPDK Interface

Table 14. Node ID Related Parameter

Register Bit RFPDK Parameter	Register Bit
Node Id Size	NODE_SIZE<1:0>
Node Id Mode	NODE_DET_MODE<1:0>
Node Id Value	NODE_VALUE<31:0>
Node Id Err Mask	NODE_ERR_MASK
Node Id Free	NODE_FREE_EN

Table 15. Register Located in Configuration Bank

Register Name	Bits	R/W	Bit Name	Function Description
CUS_PKT16 (0x47)	5	RW	NODE_FREE_EN	In RX mode, the enable bit that makes the Node ID detection circuit independent. 0：Disable 1：Enable
	4	RW	NODE_ERR_MASK	The Node ID detection error will output a PKT_ERR interrupt and simultaneously synchronize the reset decoding circuit. The bit controls whether the synchronous reset is performed. 0: Permit the synchronous reset. 1: No permit the synchronous reset.
	3:2	RW	NODE_SIZE<1:0>	Node ID length： 0：1 byte 1：2 bytes 2：3 bytes 3：4 bytes
	1:0	RW	NODE_DET_MODE<1:0>	Node ID detection mode： 0：No detection 1：The TX mode sends the contents of the NODE_VALUE; the RX mode only recognizes the content of the NODE_VALUE. 2：The TX mode sends the contents of the NODE_VALUE; the RX mode only recognizes the content of the NODE_VALUE and all 0. 3：The TX mode sends the contents of the NODE_VALUE; the RX mode only recognizes the content of the NODE_VALUE, all 0 and all 1.
CUS_PKT17 (0x48)	7:0	RW	NODE_VALUE<7:0>	32-bit Node ID value
CUS_PKT18	7:0	RW	NODE_VALUE<15:8>	

(0x49)				
CUS_PKT19 (0x4A)	7:0	RW	NODE_VALUE<23:16>	
CUS_PKT20 (0x4B)	7:0	RW	NODE_VALUE<31:24>	

	NODE_VALUE			
NODE_SIZE	<31:24>	<23:16>	<15:8>	<7:0>
0	√			
1	√	√		
2	√	√	√	
3	√	√	√	√

The tick in the table indicates the register to be filled in. For example, if the NODE_SIZE is set to 1, that is, the length is 2 byte and the value is 0x5678, the user will fill the value in the two registers of SYNC_VALUE<31:24> and SYNC_VALUE<23:16>. The MSB corresponds to the thirty-first bit. The LSB corresponds to the sixteenth bit. That is filling 0x56 into SYNC_VALUE<31:24> and filling 0x78 into SYNC_VALUE<23:16>.

The following will introduce, in different PKT_TYPE, how to set up PAYLOAD_LENG and NODE_SIZE, how to confirm the DATA length inside Payload, how to explain the Length Byte meaning. Users can find the parts they need to understand in the following table as needed:

Fixed packet format:

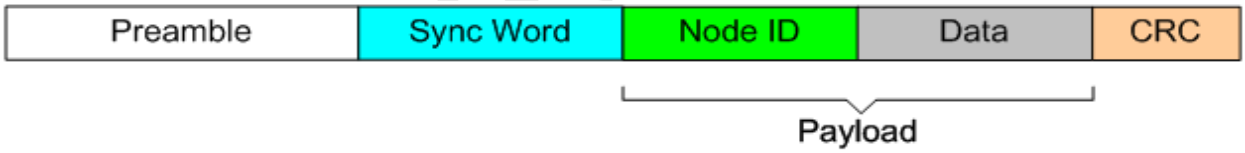

Figure 12. Fixed Length Packet Format

TX Mode	RX Mode
Payload structure： Node ID + Data, where Node ID does not exist, the TX and RX configurations are exactly the same. For example： PAYLOAD_LENG<10:0> = 11 NODE_SIZE<1:0> = 2 Then the total length of Payload is 11 + 1 = 12, the length of Node ID is 2 + 1 = 3. So, Data's length is 12 - 3 = 8 bytes. If Node ID does not exist, then the length of Data is 12.	

Variable packet format:

Condition 1: Node ID does not exist

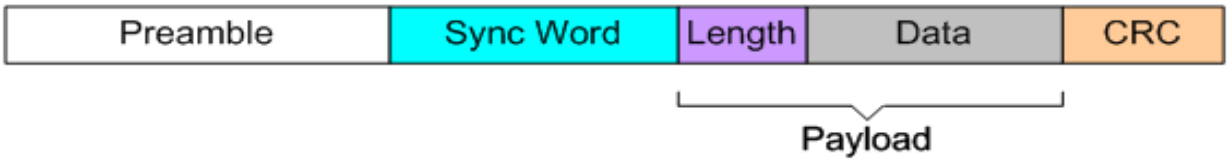

Figure13. Variable Packet Format, Node ID does not exist

TX Mode	RX Mode
Payload structure: Length Byte + Data Payload length: 1 + PAYLOAD_LENG<7:0>，where the content of PAYLOAD_LENG<7:0> is equal to the content of Length Byte, which refers to the Data length followed, and therefore can only fill 8 bits, the maximum value is 255. The previous 1 represents the Length Byte itself is the length of the 1 byte. For example： PAYLOAD_LENG<7:0> = 11， then the total length of Payload is 1 + 11 = 12, so the length of Data is 11.	Payload structure: Length Byte + Data Length Byte content meaning： Represents that the followed Data length corresponds to the TX configuration.

Condition 2: Node ID exists, and NODE_POSITION = 0 (Node ID is before Length Byte.)

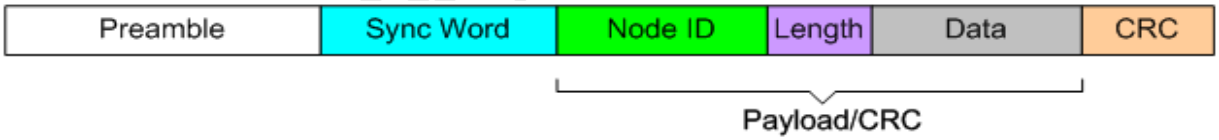

Figure 14. Variable packet format, Node ID exists, Node ID is before Length Byte

TX Mode	RX Mode
Payload structure: Node ID + Length Byte + Data Payload length: (NODE_SIZE+1) + 1 + PAYLOAD_LENG<7:0>，where the content of PAYLOAD_LENG<7:0> is equal to the content of Length Byte, which refers to the length of the Data followed, and therefore can only fill 8 bits, the maximum value is 255. The middle plus 1 means the Length Byte itself is the length of the 1 byte.	Payload structure: Node ID + Length Byte + Data Length Byte content meaning： Represents that the followed Data length corresponds to the TX configuration.

For example： PAYLOAD_LENG<7:0> = 11，NODE_SIZE<1:0> = 2， then the total length of Payload is (2 + 1) + 1 + 11 = 15, so the length of Data is 11.	

Condition 3: Node ID exists, and NODE_POSITION = 1 (Node ID is after Length Byte.)

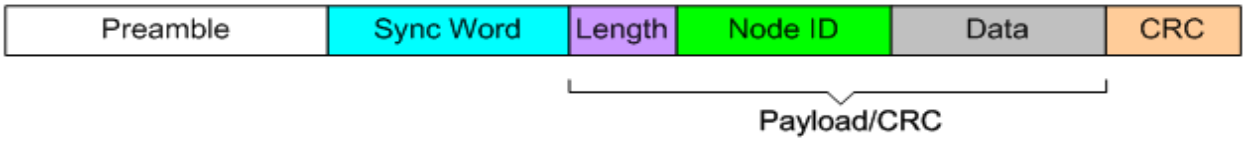

Figure 15. Variable Packet Format, Node ID Exists, Node ID is after Length Byte

TX Mode	RX Mode
Payload structure: Length Byte + Node ID + Data Payload length: 1 + PAYLOAD_LENG<7:0>，where the content of PAYLOAD_LENG<7:0> is equal to the content of Length Byte, which refers to the following Node ID plus the length of Data, so it can only fill 8 bits, the maximum value is 255. The previous 1 means the Length Byte itself is the length of the 1 byte. For example： PAYLOAD_LENG<7:0> = 11，NODE_SIZE<1:0> = 2，then the total length of Payload is 1 + (2 + 1) + 8 = 12, so the length of Data is 8.	Payload structure: Length Byte + Node ID + Data Length Byte content meaning： Represents that the followed length of Node ID + Data corresponds to the TX configuration.

2.6 FEC Configuration

The corresponding RFPDK interface and parameters are as below:

Table 16. FEC Related Parameter

Register Bit RFPDK Parameter	Register Bit
FEC	FEC_EN
FEC_Type	FEC_TYPE

Table 17. Register Located in Configuration Bank

Register Name	Bits	R/W	Bit Name	Function Description
CUS_PKT21 (0x4C)	7	RW	FEC_TYPE	Polynomial selection of FEC (7,4) encoding and decoding： 0: the polynomial is x^3+x+1 1: the polynomial is x^3+x^2+1
	6	RW	FEC_EN	FEC (7,4) encoding and decoding enable: 0: Disable 1: Enable

The main function of FEC is to correct an erroneous data in the packet, so it can reduce the packet error rate.

2.7 CRC Configuration

The corresponding RFPDK interface and parameters are as below:

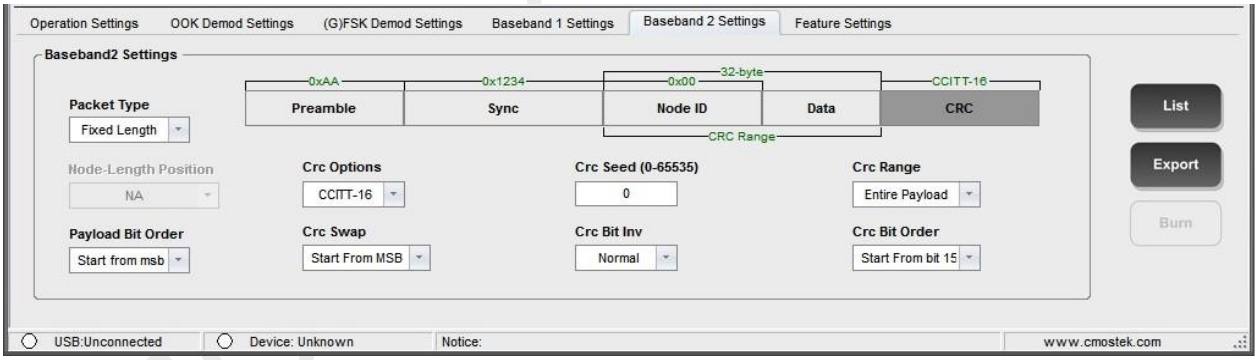

Figure 17. CRC RFPDK Interface

Table18. CRC Related Parameter

Register Bit RFPDK Parameter	Register Bit
Crc Options	CRC_TYPE<1:0>
Crc Seed	CRC_SEED<15:0>
Crc Options 为 None 时，CRC_EN = 0，否则 CRC_EN =1	CRC_EN
Crc Range	CRC_RANGE
Crc Swap	CRC_BYTE_SWAP
Crc Bit Inv	CRC_BIT_INV
Crc Bit Order	CRC_BIT_ORDER

Table 19. Register Located in Configuration Bank

Register Name	Bits	R/W	Bit Name	Function Description
CUS_PKT21 (0x4C)	5	RW	CRC_BYTE_SWAP	CRC send-receive order: 0: First send and receive the high byte 1: First send and receive the low byte
	4	RW	CRC_BIT_INV	Whether or not CRC code is reversed. 0: CRC code is not reversed. 1: CRC code is reversed bit by bit.
	3	RW	CRC_RANGE	CRC calculation range: 0: the whole payload 1: only for data
	2:1	RW	CRC_TYPE<1:0>	CRC polynomial type: 0:CCITT-16 1:IBM-16 2:ITU-16(equals the reversed CCITT-16) 3: NA
	0	RW	CRC_EN	CRC enable 0：Disable 1：Enable
CUS_PKT22 (0x4D)	7:0	RW	CRC_SEED<7:0>	Initial values of CRC polynomial
CUS_PKT23 (0x4E)	7:0	RW	CRC_SEED<15:8>	
CUS_PKT24 (0x4F)	7	RW	CRC_BIT_ORDER	CRC Big-end and Little-end order configuration: 0：CRC bytes send and receive in order from bit15 to bit0. 1：CRC bytes send and receive in order from bit0 to bit15.

The following explains in detail the principles of several configurations of CRC.

CRC_RANGE

This function is the codec checksum object of the specified CRC. It could be the whole Payload, or the Data part.

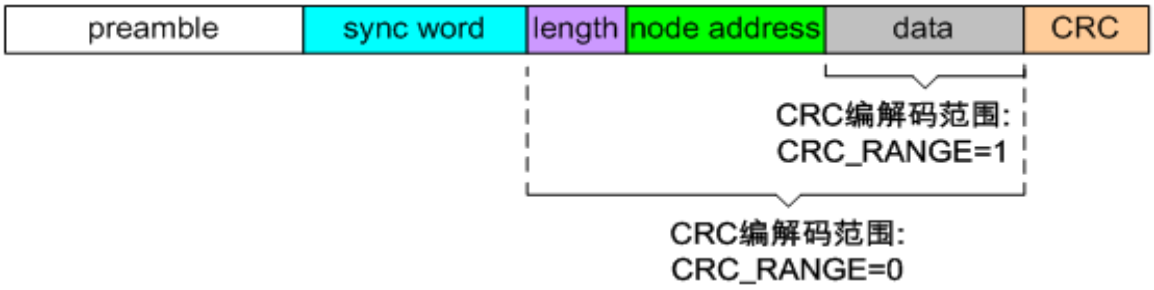

Figure 18. CRC Coding Range

CRC_BIT_INV

This function is to inverse every bit of CRC, which turns 0 into 1, and turns 1 into 0.

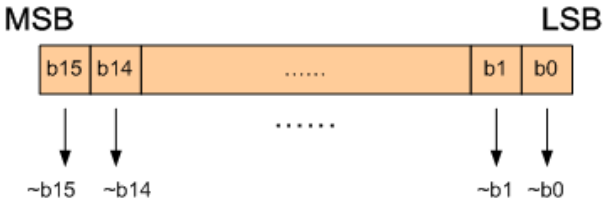

Figure 19. CRC_BIT_INV

CRC_BYTE_SWAP

This function is to swap the position of the two Byte, but not change the Bit order in each Byte.

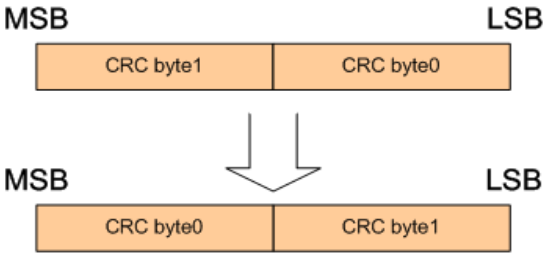

Figure 20. CRC_BYTE_SWAP

CRC_BIT_ORDER

This function is to reverse the high and low order of the whole CRC. If the position of the two BYTE is changed, it will be inverted according to the high and low order after the change.

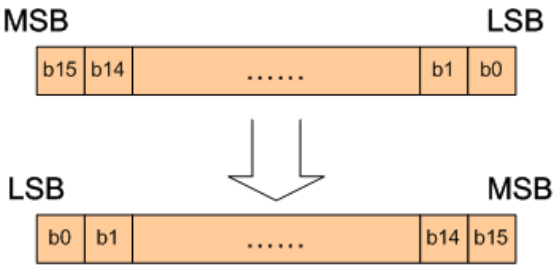

Figure 21. CRC_BIT_ORDER

2.8 Codec Configuration

The corresponding RFPDK interface and parameters are as below:

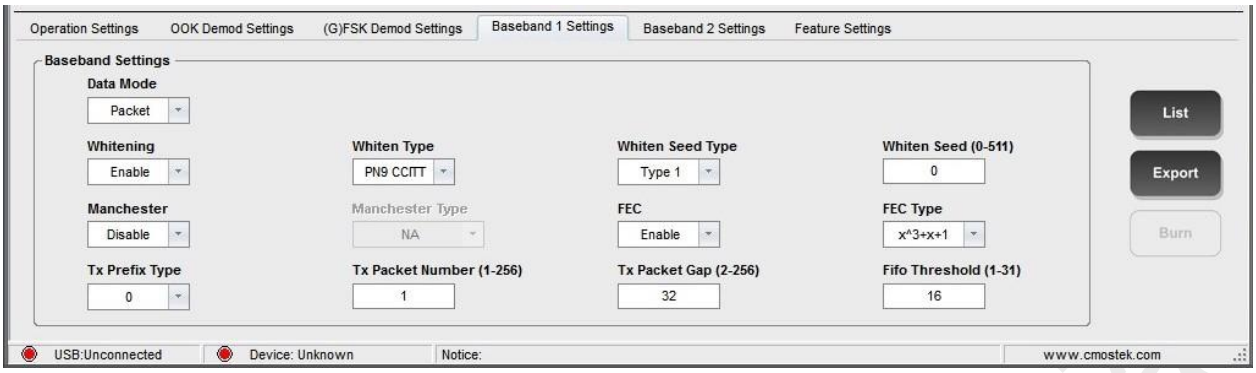

Figure 22. Codec RFPDK Interface

Table 20. Codec Related Parameter

Register Bit RFPDK Parameter	Register Bit
Whitening	WHITEN_EN
Whiten Type	WHITEN_TYPE<1:0>
Whiten Seed Type	WHITEN_SEED_TYPE
Whiten Seed	WHITEN_SEED<8:0>
Manchester	MANCH_EN
Manchester Type	MANCH_TYPE

Table 21. Register Located in Configuration Bank

Register Name	Bits	R/W	Bit Name	Function Description
CUS_PKT24 (0x4F)	6	RW	WHITEN_SEED<8>	The eighth bit of the seed of the whitening codec polynomial. When the whitening codec method is PN9, the seed takes all 9bits. When it is PN7, the seed takes the lower 7bits.
	5	RW	WHITEN_SEED_TYPE	The seed type that the whitening codec polynomial is PN7: 0：Calculate PN7 seed in A7139 way 1：PN7 seed is the value defined for whiten seed.
	4:3	RW	WHITEN_TYPE<1:0>	Whitening codec way: 0：PN9 CCITT Codec 1：PN9 IBM Codec 2：PN7 Codec 3：NA
	2	RW	WHITEN_EN	Whitening codec enable: 0: Disable whitening codec

Register Name	Bits	R/W	Bit Name	Function Description
				1: Enable whitening codec
	1	RW	MANCH_TYPE	Manchester codec way: 0：01 means 1; 10 means 0 1：10 means 1; 01 means 0
	0	RW	MANCH_EN	Manchester codec enable: 0：Disable 1：Enable
CUS_PKT25 (0x50)	7:0	RW	WHITEN_SEED<7:0>	The 7:0 bit of the seed of the whitening codec polynomial.

2.9 Tx Packet Specific Configuration

The corresponding RFPDK interface and parameters are as below:

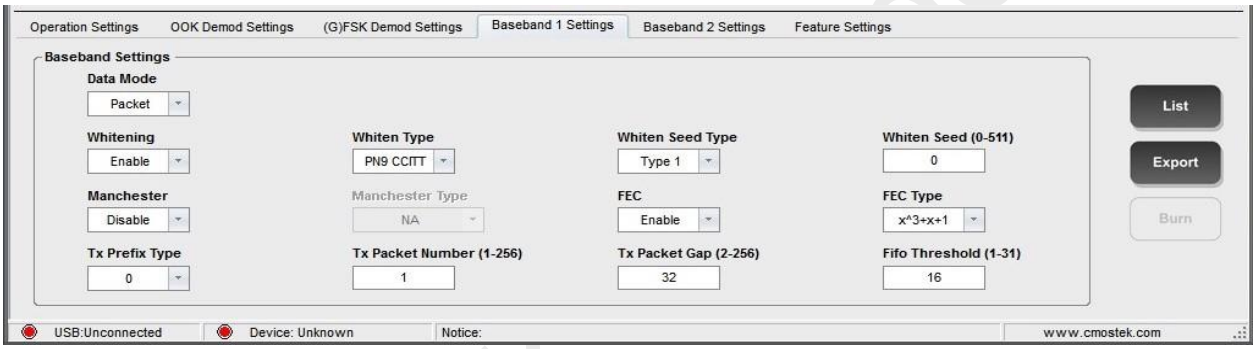

Figure 23. TX Packet RFPDK Interface

Table 22. TX Packet Related Parameter

Register Bit RFPDK Parameter	Register Bit
Tx Prefix Type	TX_PREFIX_TYPE<1:0>
Tx Packet Number	TX_PKT_NUM<7:0>
Tx Packet Gap	TX_PKT_GAP<7:0>

Table 23. Register Located in Configuration Bank

Register Name	Bits	R/W	Bit Name	Function Description
CUS_PKT26 (0x51)	1:0	RW	TX_PREFIX_TYPE<1:0>	"TX Prefix" means that in Packet mode, after entering the sending state, because the FIFO data is not ready yet, PA has already started sending, it is necessary to define the content of the pre sending, which can be defined as:

Register Name	Bits	R/W	Bit Name	Function Description
				0：Send 0 1：Send 1 2：Send Preamble 3：NA
CUS_PKT27 (0x52)	7:0	RW	TX_PKT_NUM<7:0>	The packet number sent repeatedly each time in TX mode: 0-255 means sending 1-256 packets.
CUS_PKT28 (0x53)	7:0	RW	TX_PKT_GAP<7:0>	The gap between packets and packets when sending repeatedly the packet in TX mode,: 0-255 indicates the sending gap between packets and packets is 1-256 Symbol

2.10 Direct Tx Mode

As the previous chapters, in the Tx mode, when DATA_MODE configured to Direct Mode, the Tx data can be directly from the GPIO. The following is associated with Direct Mode register：

Register Name	Bits	R/W	Bit Name	Function Description
CUS_PKT1 (0x38)	1:0	RW	DATA_MODE<1:0>	Data mode option： 0：Direct Mode（Default） 1：NA 2：Packet Mode 3：NA
CUS_TX1 (0x55)	2	RW	TX_DIN_SOURCE	Tx_Data_In option： 0：TX Data from TX FIFO 1：TX Data from GPIO
CUS_IO_SEL (0x65)	7:6	RW	GPIO4_SEL<1:0>	GPIO4 option： 0：RST In 1：INT1 2：DOUT 3：DCLK（Tx / Rx）
	5:4	RW	GPIO3_SEL<1:0>	GPIO3 option： 0：CLKO 1：DOUT / DIN 2：INT2 3：DCLK（Tx / Rx）
	3:2	RW	GPIO2_SEL<1:0>	GPIO2 option： 0：INT1 1：INT2 2：DOUT / DIN 3：DCLK（Tx / Rx）

	1:0	RW	GPIO1_SEL<1:0>	GPIO1 option： 0：DOUT / DIN 1：INT1 2：INT2 3：DCLK（Tx / Rx）
CUS_INT2_CTL (0x67)	5	RW	TX_DIN_INV	Tx data invert control（include FIFO mode）： 0：invert Tx data 1：non-invert Tx data
CUS_FIFO_CTL (0x69)	7	RW	TX_DIN_EN	DOUT / DIN Direction option： 0：Enable DOUT mode 1：Enable DIN mode
	6:5	RW	TX_DIN_SEL<1:0>	Tx Data source option： 0：GPIO1 1：GPIO2 2：GPIO3 3：Always “1”

User must be configured DATA_MODE to 0, and then set the TX_DIN_SOURCE to 1

The user must be configured these registers before GO_TX. And then sending GO_TX command, Tx_Data should be control by the MCU_IO, which connect with the GPIO. It is important to note that in the Direct mode, the data of data rate is controlled by MCU, should closely to the target.

In addition, user can use DCLK to secondary data synchronization, suggest MCU send data on the falling edge of DCLK. When after tx, it can send GO_ * command to switch mode, it is important to note that in general, after the chip receives the GO_ * command, it need 2-3 symbol time for PA_RAMP_DOWN after the completion of the will switch to the target state.

3. GPIO and Interrupt

CMT2300A has 3 GPIO ports; each GPIO can be configured into different input or output. CMT2300A has 2 interrupt ports and can be configured to the different GPIO output. The following will introduce each of them.

3.1 GPIO Configuration

Here is the register associated with the GPIO configuration, in the configuration bank 1:

Table 24. GPIO Related Register

Register Name	Bits	R/W	Bit Name	Function Description
CUS_IO_SEL (0x65)	5:4	RW	GPIO3_SEL<1:0>	GPIO3 option： 0：CLKO 1：DOUT/DIN 2：INT2

				3：DCLK (TX/RX)
	3:2	RW	GPIO2_SEL<1:0>	GPIO2 option： 0：INT1 1：INT2 2：DOUT/DIN 3：DCLK (TX/RX)
	1:0	RW	GPIO1_SEL<1:0>	GPIO1 option： 0：DOUT/DIN 1：INT1 2：INT2 3：DCLK (TX/RX)

There are several points to note above: DCLK is the synchronous clock for MCU transmitting or receiving data in direct mode. After entering the RX or TX status, it will automatically switch to the synchronization for transmitting or receiving.

3.2 Interrupt Configuration and Mapping

CMT2300A has two interrupt ports, INT1 and INT2, which can be assigned to different GPIO. The following are the register related with the interrupt.

Table 25. Control Bank 1

Register Name	Bits	R/W	Bit Name	Function Description
CUS_INT1_CTL (0x66)	5	RW	INT_POLAR	Interrupt output polarity option: 0：0 invalid, 1 valid 0：0 valid, 1 invalid
	4:0	RW	INT1_SEL<4:0>	INT1 mapping option. Please refer to the interrupt mapping table below.
CUS_INT2_CTL (0x67)	4:0	RW	INT2_SEL<4:0>	INT2 mapping option. Please refer to the interrupt mapping table below.
CUS_INT_EN (0x68)	7	RW	SL_TMO_EN	Sleep timeout interrupt enable 0：Disable 1：Enable
	6	RW	RX_TMO_EN	Receiving timeout interrupt enable 0：Disable 1：Enable
	5	RW	TX_DONE_EN	Transmitting done interrupt enable 0：Disable 1：Enable
	4	RW	PREAM_OK_EN	Preamble detection OK interrupt enable: 0：Disable 1：Enable
	3	RW	SYNC_OK_EN	Sync Word detection OK interrupt enable: 0：Disable

				1：Enable
	2	RW	NODE_OK_EN	Node ID detection OK interrupt enable: 0：Disable 1：Enable
	1	RW	CRC_OK_EN	CRC detection OK interrupt enable: 0：Disable 1：Enable
	0	RW	PKT_DONE_EN	Packet receiving done（Right or wrong）interrupt enable: 0：Disable 1：Enable
CUS_INT_CLR1 (0x6A)	5	RW	SL_TMO_FLG	SL_TMO interrupt flag
	4	RW	RX_TMO_FLG	RX_TMO interrupt flag
	3	RW	TX_DONE_FLG	TX_DONE interrupt flag
	2	RW	TX_DONE_CLR	TX_DONE interrupt clear 0：No action 1：Clear
	1	RW	SL_TMO_CLR	SL_TMO interrupt clear 0：No action 1：Clear
	0	RW	RX_TMO_CLR	RX_TMO interrupt clear 0：No action 1：Clear

Table 26. Control Bank 2

Register Name	Bits	R/W	Bit Name	Function Description
CUS_INT_CLR2 (0x6B)	5	RW	LBD_CLR	LBD is valid（detected successfully the low voltage）. Interrupt clear 0：No action 1：Clear
	4	RW	PREAM_OK_CLR	PREAM_OK interrupt clear 0：No action 1：Clear
	3	RW	SYNC_OK_CLR	SYNC_OK interrupt clear 0：No action 1：Clear
	2	RW	NODE_OK_CLR	NODE_OK interrupt clear 0：No action 1：Clear
	1	RW	CRC_OK_CLR	CRC_OK interrupt clear 0：No action 1：Clear
	0	RW	PKT_DONE_CLR	PKT_DONE interrupt clear 0：No action 1：Clear
CUS_INT_FLAG (0x6D)	7	RW	LBD_FLG	The interrupt flag LBD is valid（detected successfully the low voltage）.
	6	RW	COL_ERR_FLG	COL_ERR interrupt flag
	5	RW	PKT_ERR_FLG	PKT_ERR interrupt flag
	4	RW	PREAM_OK_FLG	PREAM_OK interrupt flag
	3	RW	SYNC_OK_FLG	SYNC_OK interrupt flag
	2	RW	NODE_OK_FLG	NODE_OK interrupt flag
	1	RW	CRC_OK_FLG	CRC_OK interrupt flag
	0	RW	PKT_OK_FLG	PKT_OK interrupt flag

The following is the interrupt mapping table. The mappings of INT1 and INT2 are the same. Take INT1 as an example:

Table 27. CMT2300A Interrupt Mapping Table

Name	INT1_SEL	Description	Cleanup Way
RX_ACTIVE	00000	Indicates the interrupt ready to enter RX and has entered RX is 1 in the PLL correction and RX States, and it is 0 at the rest.	Auto
TX_ACTIVE	00001	Indicates the interrupt ready to enter TX and has entered TX is 1 in the PLL correction and TX States, and it is 0 at the rest.	Auto
RSSI_VLD	00010	Indicates the interrupt whether RSSI is valid.	Auto
PREAM_OK	00011	Indicates the interrupt successfully received Preamble.	by MCU
SYNC_OK	00100	Indicates the interrupt successfully received Sync Word.	by MCU
NODE_OK	00101	Indicates the interrupt successfully received Node ID.	by MCU
CRC_OK	00110	Indicates the interrupt successfully received and passed through the CRC check.	by MCU
PKT_OK	00111	Indicates the interrupt received a full packet.	by MCU
SL_TMO	01000	Indicates the interrupt of the SLEEP counter timeout.	by MCU
RX_TMO	01001	Indicates the interrupt of the RX counter timeout.	by MCU
TX_DONE	01010	Indicates the interrupt of TX completion.	by MCU
RX_FIFO_NMTY	01011	Indicates the non empty interrupt of RX FIFO	Auto
RX_FIFO_TH	01100	Indicates the interrupt of RX FIFO unread content over FIFO TH	Auto
RX_FIFO_FULL	01101	Indicates the interrupt of RX FIFO full	Auto
RX_FIFO_WBYTE	01110	Indicates the interrupt RX FIFO writes a BYTE each time. It is a pulse.	Auto
RX_FIFO_OVF	01111	Indicates the interrupt of the RX FIFO overflow	Auto
TX_FIFO_NMTY	10000	Indicates the non empty interrupt of TX FIFO	Auto
TX_FIFO_TH	10001	Indicates the interrupt of TX FIFO unread content over FIFO TH.	Auto
TX_FIFO_FULL	10010	Indicates the interrupt of TX FIFO full.	Auto
STATE_IS_STBY	10011	Indicates the interrupt the current state is STBY.	Auto
STATE_IS_FS	10100	Indicates the interrupt the current state is RFS or TFS.	Auto
STATE_IS_RX	10101	Indicates the interrupt the current state is RX.	Auto
STATE_IS_TX	10110	Indicates the interrupt the current state is TX.	Auto
LBD	10111	Indicates the interrupt the low voltage detection is valid (VDD is below the set TH).	Auto
TRX_ACTIVE	11000	Indicates the interrupt ready to enter RX or TX and has entered RX or TX is 1 in the PLL correction, RX and TX States, and it is 0 at the rest.	Auto
PKT_DONE	11001	Indicates that the current packet has been received, and there will be 4 cases: 1. Received completely the full packet.	by MCU

Name	INT1_SEL	Description	Cleanup Way
		2. Manchester decoding is error, decoding circuit will automatically restart 3. NODE ID receiving is error, decoding circuit will automatically restart. 4. Detect the signal conflict, the decoding circuit does not automatically restart, waits for MCU processing.	

Interrupt default is 1 valid. However, by setting the INT_POLAR register bit to 1, all interrupts will become 0 valid. Take INT1 as an example, the control and selection diagrams for all interrupt sources are drawn below. For control and mapping, INT1 and INT2 are the same.

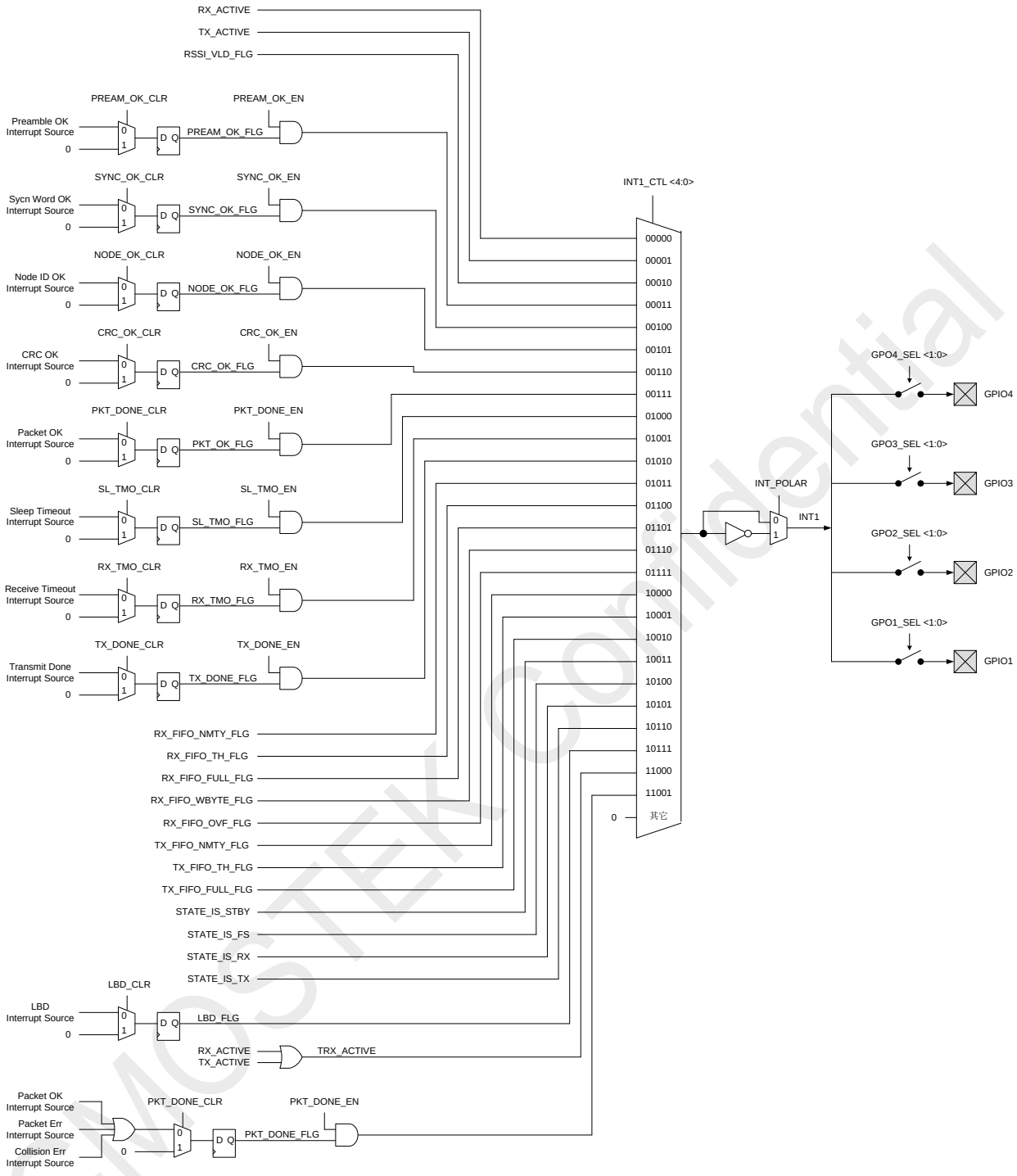

Figure 24. CMT2300A INT1 Interrupt Mapping

For those interrupt sources that require MCU clearing, each one is equipped with an EN enable and a CLR clear bit, except that LBD is special and has no EN enable bit. For example, the interrupt source of SYNC_OK will only be generated when the bit SYNC_OK_EN is set to 1. When this interrupt occurs, it is cleared only if the SYNC_OK_CLR is set to 1. When you set the CLR bit, MCU just needs to set it to 1, and then does not need to set it back to 0, because in the chip, the CLR bit will be cleared automatically after the corresponding interrupt is cleared.

1. If there is a NODE ID in the packet, there may be an error in the NODE ID check. At this point, the PKT_ERR_FLG flag will be set up and the chip will stop receiving the packet, and automatically restart the decoder and wait for the next SYNC WORD coming. In this case, PKT_OK is not generated.

2. If you enable the Manchester decoding, there may be an error in decoding. At this point, the PKT_ERR_FLG flag will be set up and the chip will stop receiving the packet, and automatically restart the decoder and wait for the next SYNC WORD coming. In this case, PKT_OK is not generated.

3. If you enable the signal conflict detection (The following will introduce the specific usage), it is possible to detect a signal conflict that results in errors in the contents of the received packet later. At this point, the COL_ERR_FLG flag will be set up and the chip will continue to receive the packet until it is finished and the decoder will not restart automatically. In this case, PKT_OK will still occur, but the received data is wrong.

No matter what happened, we need to tell the external MCU, or the MCU will always wait for the interrupt, which will cause the failure to interact with the chip. So, we will give the interrupt signal PKT_DONE = PKT_OK | PKT_ERR_FLG | COL_ERR_FLG, that is no matter what happened, we will notify the MCU. After the MCU received the interrupt, it can check the related 3 flag bits and know what happened, and then deal with the follow-up.

Another approach is that MCU can only wait for PKT_OK and check CRC flag bit (if there is) , but be careful with the SYNC_OK interrupt. For example, the first packet appears wrong with decoding, and the decoder immediately receives the next packet again. For MCU, there will be two consecutive SYNC_OK interrupts. If the MCU doesn't handle it well, it will be assumed that the second SYNC_OK interrupt is the PKT_OK interrupt and that it will be handled incorrectly.

In short, using the codec interrupt will run into a variety of situations. Different users have the different understanding. It is recommended that the program should be as simple as possible, but the stability is at the top of the list, it is better to bring the timeout mechanism with itself. If the MCU fails to interact with the chip (losing contact), a crash occurs.

The Cmt2300GPIO output interrupt configuration code is illustrated in Appendix 2.

3.3 Antenna TX / RX Switching Control

The following are the registers that control the external antenna for TX/RX switching:

Table 28. TX/RX Switch Control Related Register

Register Name	Bits	R/W	Bit Name	Function Description
CUS_INT1_CTL (0x66)	7	RW	RF_SWT1_EN	Antenna switch signal enable 1: 0：Disable 1：Enable At the same time, RF_SWT1_EN and RF_SWT2_EN cannot be configured to be valid.
	6	RW	RF_SWT2_EN	Antenna switch signal enable 2: 0：Disable 1：Enable At the same time, RF_SWT2_EN and RF_SWT1_EN cannot be configured to be valid.

The chip can output a set of signals (two different) on the GPIO1 and GPIO2 to control the RX/TX switching of the external antenna. . We can choose to output two sets of control signals with different timing characteristics, which are respectively enabled by two enabling signals RF_SWT1_EN and RF_SWT2_EN:

1. When RF_SWT1_EN enables, GPIO1 will output RX_ACTIVE, and GPIO2 will output TX_ACTIVE.
2. When RF_SWT2_EN enables, GPIO1 will output RX_ACTIVE, and GPIO2 will output RX_ACTIVE reversed, that is the complete difference.

Users need to note, either RF_SWT1_EN or RF_SWT2_EN is opened, another one cannot be opened, meanwhile GPIO1_SEL and GPIO2_SEL will be invalid, that is the antenna switching control has the highest priority.

Users can test the timing difference between the two control modes and decide which one to use according to the actual situation.

4. Document Modification Record

Table29. Document Modification Record Sheet

Version	Chapter	Modification descriptions	Date
0.9	All	Initial release	2017-08-11
1.1	Table 3	Change bit of CUS_PKT1	2019-03-15
1.1	Table 5	Change TX_FIFO_FULL_FLG description from indicating FIFO not empty to FIFO full.	2019-03-15
1.1	Table 11	Change bit of SYNC_MAN_EN.	2019-03-15
1.1	Table 15	Change bit of NODE_ERR_MASK, NODE_SIZE<1:0>, NODE_DET_MODE<1:0>	2019-03-15

Appendix 1: Sample code FIFO read-write operation code examples

Sample code FIFO read enable operation sub function

```
/*! ********************************************************
* @name       Cmt2300_EnableReadFifo
* @desc       Enable SPI to read the FIFO.
* ********************************************************/
void Cmt2300_EnableReadFifo(void)
{
    u8 tmp = Cmt2300_ReadReg(CMT2300_CUS_FIFO_CTL);
    tmp &= ~CMT2300_MASK_SPI_FIFO_RD_WR_SEL;
    tmp &= ~CMT2300_MASK_FIFO_RX_TX_SEL;
    Cmt2300_WriteReg(CMT2300_CUS_FIFO_CTL, tmp);
}

/*! ********************************************************
* @name       Cmt2300_EnableWriteFifo
* @desc       Enable SPI to write the FIFO.
* ********************************************************/
void Cmt2300_EnableWriteFifo(void)
{
    u8 tmp = Cmt2300_ReadReg(CMT2300_CUS_FIFO_CTL);
    tmp |= CMT2300_MASK_SPI_FIFO_RD_WR_SEL;
    tmp |= CMT2300_MASK_FIFO_RX_TX_SEL;
    Cmt2300_WriteReg(CMT2300_CUS_FIFO_CTL, tmp);
}
```

Appendix 2: Sample code GPIO outputs interrupt configuration function examples

```
void RF_Config(void)
{
#ifdef ENABLE_ANTENNA_SWITCH
    /* If you enable antenna switch, GPIO1/GPIO2 will output RX_ACTIVE/TX_ACTIVE,
       and it can't output INT1/INT2 via GPIO1/GPIO2 */
    Cmt2300_EnableAntennaSwitch(0);
#else
    /* Config GPIOs */
    Cmt2300_ConfigGpio(
        CMT2300_GPIO1_SEL_INT1 | /* INT1 > GPIO1 */
        CMT2300_GPIO2_SEL_INT2 | /* INT2 > GPIO2 */
        CMT2300_GPIO3_SEL_DOUT/*DOUT>GPIO3*/
        );

    /* Config interrupt */
    Cmt2300_ConfigInterrupt(
        CMT2300_INT_SEL_TX_DONE, /* Config INT1 */
```

```
            CMT2300_INT_SEL_PKT_OK      /* Config INT2 */
            );
#endif

    /* Enable interrupt */
    Cmt2300_EnableInterrupt(
        CMT2300_MASK_TX_DONE_EN    |
        CMT2300_MASK_PREAM_OK_EN |
        CMT2300_MASK_SYNC_OK_EN    |
        CMT2300_MASK_NODE_OK_EN    |
        CMT2300_MASK_CRC_OK_EN     |
        CMT2300_MASK_PKT_DONE_EN
        );
    Cmt2300_EnableLfosc(FALSE);
    /* Use a single 64-byte FIFO for either Tx or Rx */
    //Cmt2300_EnableFifoMerge(TRUE);
    //Cmt2300_SetFifoThreshold(16);

    /* Go to sleep for configuration to take effect */
    Cmt2300_GoSleep();
}
```

5. Contact Information

Wuxi CMOSTEK Microelectronics Co., Ltd. Shenzhen branch
Room 203, Honghai Building, Qianhai Road, Nanshan District, Shenzhen, Guangdong, China

Zip Code： 518000
Tel： +86 - 755 - 83235017
Fax： +86 - 755 - 82761326
Sales： sales@cmostek.com
Technical support： support@cmostek.com
Website： www.cmostek.com