CMT2300A FIFO和包格式使用指南

概要

本文介绍了 CMT2300A 的 FIFO，包格式，以及中断系统的工作原理。在介绍配置寄存器的内容的时候，会跟 RFPDK 上可输入的参数对应起来，方便用户进行配置。

本文档涵盖的产品型号如下表所示。

表 1. 本文档涵盖的产品型号

产品型号	工作频率	调制方式	主要功能	配置方式	封装
CMT2300A	126.33- 1020MHz	(G)FSK/OOK	收发一体机	寄存器	QFN16

阅读此文档之前，建议阅读《AN142-CMT2300A 快速上手指南》以了解 CMT2300A 的基本使用方式。

目录

1. FIFO 工作原理

1.1 FIFO 相关的寄存器

对应的 RFPDK 的界面和参数：

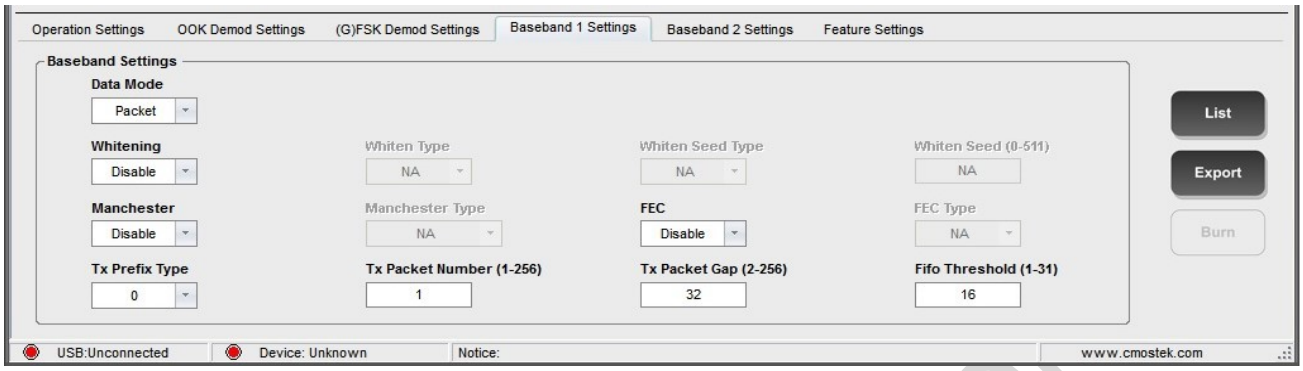

图 1. FIFO 的 RFPDK 界面

表 2. FIFO 相关参数

寄存器比特 RFPDK 参数	寄存器比特
Data Mode	DATA_MODE <1:0>
RFPDK 不显示，由用户在应用程序中灵活配置	FIFO_TH <6:0>
自动根据发射包数量计算，当发射数量大于 1 个包是设置为 1。	FIFO_AUTO_RES_EN

寄存器的内容和解释可看下表。

表 3.位于配置区的寄存器：

寄存器名	位数	R/W	比特名	功能说明
CUS_PKT1 (0x38)	0	RW	DATA_MODE <1:0>	决定数据处理模式： 0：Direct 模式（默认） 1：NA 2：Packet 模式 3：NA
CUS_PKT29 (0x54)	7	RW	FIFO_AUTO_RES_EN	每次发完一个包就自动 restore TX FIFO，如果每次进入 TX 要重复发送超过 1 个包（TX_PKT_NUM> 0），这个比特就必须设成 1。

寄存器名	位数	R/W	比特名	功能说明
	6:0	RW	FIFO_TH <6:0>	FIFO 的填入阈值，单位是 byte，对 RX 来说，当未读数据超过这个阈值时，RX_FIFO_TH_FLG 就会置 1；对 TX 来说，当未发数据小过这个阈值时，TX_FIFO_TH_FLG 就会置 0。 当 FIFO_MERGE_EN = 0 时，有效范围是 1 到 31 当 FIFO_MERGE_EN = 1 时，有效范围是 1 到 63

表 4.位于控制区 1 的寄存器

寄存器名	位数	R/W	比特名	功能说明
CUS_FIFO_CTL (0x69)	4	RW	FIFO_AUTO_CLR_DIS	0：进入 RX 前自动清零 RX FIFO，1：不自动清零
	3	RW	FIFO_TX_RD_EN	0：TX FIFO 只能用 SPI 写，1：RX FIFO 可被 SPI 读取。该比特只对 TX FIFO 有效，除了给用户测试时可以使用，其余时候都应该设成 0。
	2	RW	FIFO_RX_TX_SEL	当作为一个 64-byte FIFO 时可使用。0：用作 RX FIFO，1：用作 TX FIFO。
	1	RW	FIFO_MERGE_EN	0：分成 2 个独立的 32-byte 的 FIFO，1：合并成 1 个 64-byte 的 FIFO。
	0	RW	SPI_FIFO_RD_WR_SEL	0：SPI 的操作是读 FIFO，1：SPI 的操作是写 FIFO。必须在访问 FIFO 前设置好。

表 5.位于控制区 2 的寄存器

寄存器名	位数	R/W	比特名	功能说明
CUS_FIFO_CLR (0x6C)	2	W	FIFO_RESTORE	给用户手动 restore TX FIFO，restore 的意思是复位读指针，维持写指针不变，这样 TX FIFO 又回到未读状态，可以再次重复发射之前填入的数据。
	1	W	FIFO_CLR_RX	0：无效，1：清零 RX FIFO。 用户将这个比特设成 1 之后，无需将它再设回 0，这个比特在内部会自动设回为 0。
	0	W	FIFO_CLR_TX	0:：无效，1：清零 TX FIFO 用户将这个比特设成 1 之后，无需将它再设回 0，这个比特在内部会自动设回为 0。
CUS_FIFO_FLAG (0x6E)	6	R	RX_FIFO_FULL_FLG	指示 RX FIFO 填满的中断。 0：无效

寄存器名	位数	R/W	比特名	功能说明
				1：有效
	5	R	RX_FIFO_NMTY_FLG	指示 RX FIFO 非空的中断标志位。 0：无效 1：有效
	4	R	RX_FIFO_TH_FLG	指示 RX FIFO 未读内容超过 FIFO TH 的中断。 0：无效 1：有效
	3	R	RX_FIFO_OVF_FLG	指示 RX FIFO 溢出的中断。 0：无效 1：有效
	2	R	TX_FIFO_FULL_FLG	指示 TX FIFO 非空的中断。 0：无效 1：有效
	1	R	TX_FIFO_NMTY_FLG	指示 TX FIFO 非空的中断。 0：无效 1：有效
	0	R	TX_FIFO_TH_FLG	指示 TX FIFO 未读内容超过 FIFO TH 的中断。 0：无效 1：有效
注意： 这些中断标志位的极性会受 INT_POLAR 控制，即当 INT_POLAR = 1 时，它们都是 0 有效，1 无效。				

在这些寄存器中，某些与 FIFO 无关的比特，在这里被忽略掉，不做介绍。

1.2 FIFO 的工作模式

CMT2300A 默认提供两个独立的 32-byte 的 FIFO，分别给 RX 和 TX 使用，两者互不相干。用户也可以将 FIFO_MARGE_EN 设成 1，那么两个 FIFO 就合成一个 64-byte 的 FIFO，在 TX 和 RX 下都可以使用，通过配置 FIFO_RX_TX_SEL 来指示目前是用作 TX 还是 RX。

通常我们会建议，在 STBY 状态下将 FIFO 的工作模式进行预配置，然后才开始 TX/RX 的工作。只要是配置区和控制区 1 的寄存器，在 SLEEP 状态都是可以保存内容的，因此，除非要改变工作模式，否则配置一次就够了。

分开的 FIFO 的预配置

1. 将 DATA_MODE <1:0>设置成 2，即将数据处理模式设置成 Packet 模式。
2. 将 FIFO_MERGE_EN 设置成 0。
3. 选择随后要进行的操作：

如果要去 RX：

a) 将 SPI_FIFO_RD_WR_SEL 配置成 0（SPI 读 FIFO 模式）；

b) 如果用户希望每次进入 RX 的时候 RX FIFO 自动清零，就将 FIFO_AUTO_CLR_DIS 设置成 0，否则设置成 1 然后进行手动清零。

如果要去 TX：

a) 将 SPI_FIFO_RD_WR_SEL 配置成 1（SPI 写 FIFO 模式）。

合并的 FIFO 的预配置

1. 将 DATA_MODE <1:0>设置成 2，即将数据处理模式设置成 Packet 模式。
2. 将 FIFO_MERGE_EN 设置成 1。
3. 选择随后要进行的操作：

如果要去 RX：

a) 将 FIFO_RX_TX_SEL 设置成 0（RX 模式）；

b) 将 SPI_FIFO_RD_WR_SEL 配置成 0（SPI 读 FIFO 模式）；

c) 如果用户希望每次进入 RX 的时候 RX FIFO 自动清零，就将 FIFO_AUTO_CLR_DIS 设置成 0，否则设置成 1，即开始使用前进行手动清零。

如果要去 TX：

a) 将 FIFO_RX_TX_SEL 设置成 1（TX 模式）；

b) 将 SPI_FIFO_RD_WR_SEL 配置成 1（SPI 写 FIFO 模式）。

c) 开始使用前进行一次手动清零。

当 FIFO 合并的时候，芯片内部就只有一个 FIFO，由于 RX/TX 是半双工的，一个 FIFO 就够用了。当 FIFO 是分开成两个时，它们相互独立，互不干扰，就可以实现一些特殊的功能。例如，如果发射数据每次都是一样的，而且 RX FIFO 在 SLEEP 下也是可以保存内容的，那么 TX FIFO 就只需要填写一次，可以节省时间和功耗。又例如，在 RX 状态下，RX FIFO 在不停地被写入，用户可以利用接收的时间，并行地先将下次要发射出去的数据填入 TX FIFO，这样既不会打扰 RX FIFO 的工作，也可以节省时间，即不需要腾出特定的时间来填入 TX FIFO，这样就可以达到省电的效果。

需要注意的是，每次在 TX FIFO 和 RX FIFO 之间切换之后，要进行一次手动清零，否则不能正常工作。在 TX FIFO 模式下，只能使用 FIFO_CLR_TX 寄存器位进行清零；在 RX FIFO 模式下，只能使用 FIFO_CLR_RX 寄存器位进行清零。两者不能同时置 1，不可混淆使用。

为适用于 FIFO 合并和不合并两情况，FIFO 读写使能操作中根据合并使用要求均对 FIFO_RX_TX_SEL 和 SPI_FIFO_RD_WR_SEL 进行配置。

```
Cmt2300_GoStby();
Cmt2300_ClearInterruptFlags();
/* Must clear FIFO after enable SPI to read or write the FIFO */
Cmt2300_EnableWriteFifo();
Cmt2300_ClearFifo();
// 当合并 FIFO 使用时，发送字节过长时需增加延时
/* The length need be smaller than 32 */
Cmt2300_WriteFifo(g_pTxBuffer, g_nTxLength);
```

FIFO 读写操作代码示例详见附录 1。

1.3 FIFO 的中断时序

在这里先给出 RX FIFO 和 TX FIFO 工作时的中断时序示意图，用户可以参考一下方便理解。

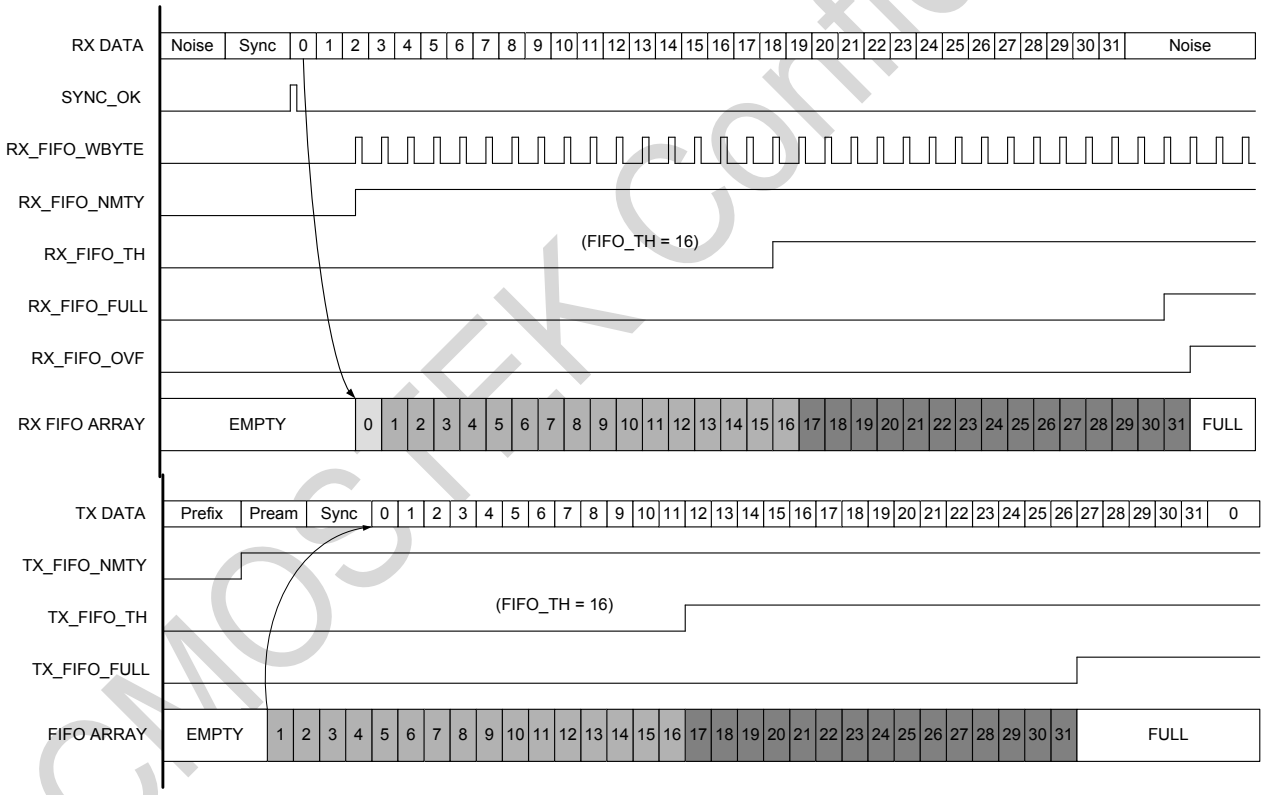

图 2. CMT2300A RX FIFO 中断时序示意图

1.4 FIFO 的应用场景

当 FIFO 配置好之后，就可以开始使用。下面介绍几种经典的应用场景。要完成 TX 和 RX 的整个流程还需要配置和控制其它东西，这些在后面陆续介绍，在这里我们只会介绍与 FIFO 相关的内容，

1.4.1 应用场景一：在 RX 下接收数据

RX FIFO 使用比较直接，每次进入 RX 之前先清零一下，进入 RX 如果有数据接收（成功检测到 Sync Word）就开始填入，MCU 可以结合中断，实现下面这几种操作，收完之后就可以不用管，下次接收之前发送 FIFO_CLR_RX 清零一下就可以了。

1. 检测 RX_FIFO_FULL 中断，一旦有效表示 FIFO 已经被填满，就可以开始读取，这样适合数据包长度刚好等于 FIFO 深度，并且用户要在接收完整数据包之后，才读取 FIFO 的情况。
2. 检测 RX_FIFO_TH 中断，一旦有效表示 FIFO 已经被填入预设的数据长度，就可以开始读取，这样适合数据包长度不等于 FIFO 深度，并且用户要做接收完整数据包之后，才读取 FIFO 的情况。
3. 检测 RX_FIFO_NMTY 中断，一旦有效即立即进行读取，直到该中断无效，又再次等待中断有效时再读取，这样可以实现边收边读，适合数据包长度大于 FIFO 深度的情况，也适合小于和等于的情况。
4. 检测 RX_FIFO_WBYTE 中断，一旦有效即立即进行读取，这样可以实现写一个 byte，读一个 byte 的有规律操作，前提是 SPI 速度要比接受数据的速度要快。

另外，可以通过设置 TX_PKT_NUM <7:0>来定义每次发射发送多少个相同的数据包，如果发送的数量多过 1 个数据包，就需要将 FIFO_AUTO_RES_EN 设置成 1，即让 TX FIFO 在每次发送完一个数据包后，自动将读指针清零，回到未读状态，这样就可以重复发射同一个包，无需 MCU 再次填写数据。用户可以设置 TX_PKT_GAP <7:0>来定义每个数据包之间的时间间隔，以 symbol 为单位。进入发射之后，发射机就会按照这些配置发送 N 个数据包，完成后自动退出 TX，回到指定状态。退出后的状态可以通过 TX_EXIT_STATE <1:0>进行配置。

如果要实现连续不间断的接收，建议使用上面第 4 种操作，这时需要 SPI 的读取速度足够快，至少要比 FIFO 写入一个 byte 的速度快 1.5 倍。例如，如果数据率是 10KHz，接收一个 byte 的时间大概是 800 us，SPI 读取一个 byte 的速度要快 1.5 倍，耗时最多就是 534us。SPI 读取一个 byte 是 8 个 SCL 时钟周期，但是加上前后的时间开销，我们可以按照 10 个 SCL 时钟周期算，那么每个周期就是 53.4us，换算过来 SCL 的时钟频率就是 18.7 kHz 左右，可以粗略认为，SCL 的速率是数据率的 2 倍。

1.4.2 应用场景二：预先填好数据，进入 TX 发射

对很多应用来说，会把要发射的数据包预先填进 TX FIFO 里面，然后再进入 TX 发射。填数据的行为建议在 STBY 状态下执行。这种场景比较适合数据包长度小于或等于 FIFO 深度，且应用时间不需要很紧凑。用户在进行预配置后，就可以直接写入数据，可以通过探测 TX_FIFO_NMTY，TX_FIFO_TH 或者 TX_FIFO_FULL 中断来判断数据是否已经完整写入。在调试阶段，用户在填完数据后，可以按照下面的方法，回读填进去的数据，来确认数据是否填写正确：

1. 将 FIFO_TX_RD_EN 设置成 1，进入 TX FIFO 的回读模式。
2. 将 SPI_FIFO_RD_WR_SEL 设置成 0，进入 SPI 读 FIFO 模式。
3. 读取数据，比较确认正确性。
4. 将 FIFO_TX_RD_EN 设置成 0，退出 TX FIFO 回读模式。
5. 将 SPI_FIFO_RD_WR_SEL 设置成 1，进入 SPI 写 FIFO 模式。
6. 将 FIFO_CLR_TX 设置为 1，清零 FIFO。

7. 重新写入数据，以备发射。

1.4.3 应用场景三：进入 TX 后，一边填数据一边发射

如果用户先进入 TX 状态，但是 TX FIFO 是空的，那么芯片就会一直发射 prefix，prefix 的内容根据 TX_PREFIX_TYPE <1:0>来定，可以是 0，1，或者 preamble。在发射 prefix 的时候，芯片会一直等待用户填入 FIFO，直到用户开始填入第一个数据 byte，芯片就会停止发射 prefix，然后一直按照数据率的节奏去获取 FIFO 的数据进行发射直到结束。所以在这种场景下，SPI 的速度要够快，SCL 的频率至少是数据率的 2 倍。如果发射数据过程中，出现 SPI 填写速度赶不上，即 FIFO 被发射机读空了，发射也不会停止，会一直发 0，直到 FIFO 填入下一个数据，但是这种属于不恰当的操作，用户要尽量避免。

1.4.4 应用场景四：每次都重复发同一个或同一组数据包

由于 TX FIFO 在 SLEEP 状态下会保存内容，所以如果每次发送的数据是一样的，MCU 就只需要填入一次数据，当每次发射完成之后，进入 STBY，将 FIFO_RESTORE 设置成 1（无需设回 0，这个比特会自动清零），就会将 FIFO 的读指针清零，意味着 FIFO 又回到了未读状态，而且数据都保存在里面。下一次进入 TX，又可以重新发射相同的内容，周而复此。如果 FIFO_AUTO_RES_EN 已经设置成 1，就无需手动设置 FIFO_RESTORE，下次直接进入 TX 发射就行了。

1.4.5 应用场景五：一个数据包分开几次发

TX FIFO 在 SLEEP 状态下不仅会保存内容，而且会保存指针状态。例如，用合并的 FIFO，大小是 64-byte，用户要陆续发射四个数据包，每个包是 16-byte。那么，用户可以在 STBY 先把这 4 个数据包按顺序填入 FIFO，确保 FIFO 满了。然后，将数据长度设置为 16（如何设置要根据包格式来定，后续描述），将发射的包个数设置为 1，将 FIFO_AUTO_RES_EN 设置成 0。然后进入 TX，芯片会发送第一个 16-byte 的数据包，完成后退出到 STBY，又再次进入 TX，芯片发送第二个数据包……直到第四次发送完成。这个过程中 MCU 不需要做什么，只负责探测中断，切换状态就可以了。

2. 包格式介绍

CMT2300A 采用了 TX 和 RX 统一配置，比较典型，比较灵活的包格式，结构图如下：

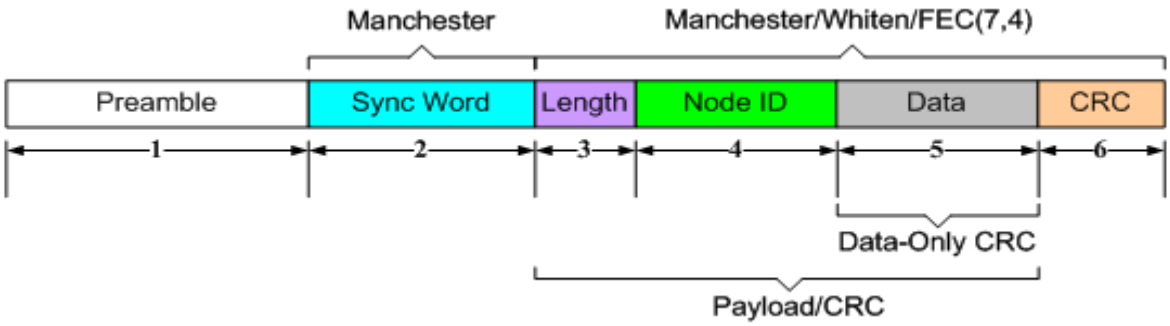

图 3. packet 结构框图

包格式包含了六个可选部分，此结构可满足市场上绝大多数单 Field 结构的需求，不兼容多 Field 结构。包格式可配置的内容都集中放在了“基带区”。下面将结合这些寄存器，解释如何配置的各个部分。

2.1 数据模式配置

数据模式（Data Mode）指的外部 MCU 通过什么模式来输入发射数据或获取接收数据。

对应的 RFPDK 的界面和参数：

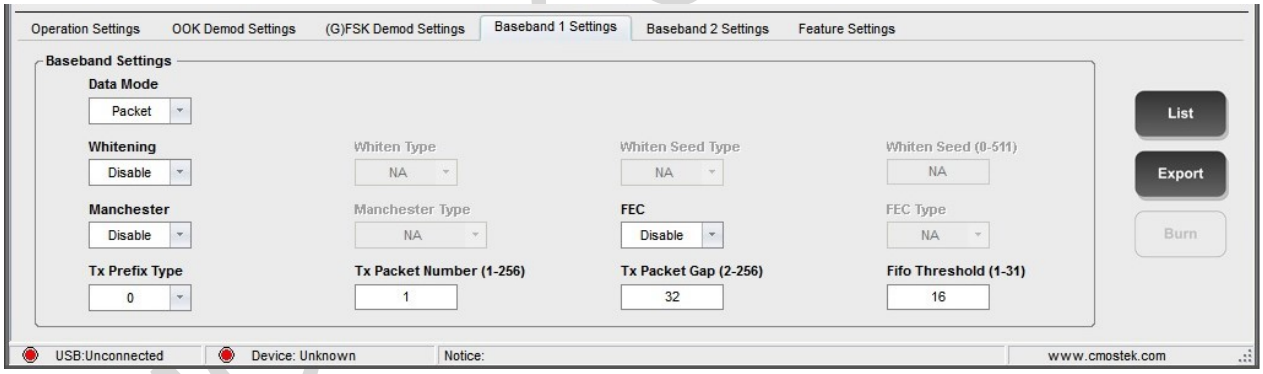

图 4. 数据模式的 RFPDK 界面

表 6. 数据模式相关参数

寄存器比特 RFPDK 参数	寄存器比特
Data Mode	DATA_MODE <1:0>
自动根据 Data Mode 选择，当 Data Mode 是 Direct 时选择数据从 GPIO 直接输入，当 Data Mode 是 Packet 时选择数据从 TX FIFO 获取。	TX_DIN_SOURCE

表 7.位于配置区的寄存器：

寄存器名	位数	R/W	比特名	功能说明
CUS_PKT1 (0x38)	1:0	RW	DATA_MODE<1:0>	选择接收和发射的数据模式： 0：Direct 模式（默认） 1：NA 2：Packet 模式 3：NA
CUS_TX1 (0x55)	2	RW	TX_DIN_SOURCE	选择 TX 的数据来源位置： 0：TX 数据从 TX FIFO 获取 1：TX 数据从 GPIO 直接输入

数据模式的区别在于：

- Direct –直通模式，RX 模式仅支持 preamble 和 sync 检测，FIFO 不工作；RX 不支持任何包格式。
- Packet –包格式模式，支持所有包格式配置，FIFO 工作。

2.2 Preamble 配置

对应的 RFPDK 的界面和参数：

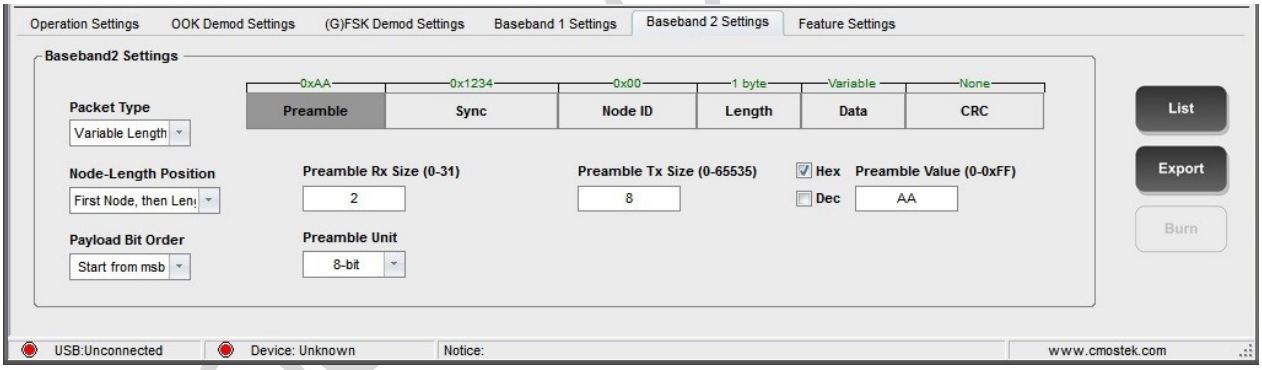

图 5. Preamble 的 RFPDK 界面

表 8. Preamble 的相关参数

寄存器比特 RFPDK 参数	寄存器比特
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>

表 9.位于配置区的寄存器：

寄存器名	位数	R/W	比特名	功能说明
CUS_PKT1 (0x38)	7:3	RW	RX_PREAM_SIZE<4:0>	RX 模式 Preamble 的长度，可配置为 0-31 个单位长度，0 表示不检测 Preamble，1 表示检测 1 个长度单位的 Preamble，如此类推。
	2	RW	PREAM_LENG_UNIT	Preamble 的长度单位，TX 和 RX 共用： 0：单位为 8bits 1：单位为 4 bits
CUS_PKT2 (0x39)	7:0	RW	TX_PREAM_SIZE<7:0>	TX 模式 Preamble 的长度，可配置为 0-65535 个单位长度，0 表示不发送 Preamble，1 表示发送 1 个长度单位的 Preamble，如此类推。
CUS_PKT3 (0x3A)	7:0	RW	TX_PREAM_SIZE<15:8>	
CUS_PKT4 (0x3B)	7:0	RW	PREAM_VALUE<7:0>	Preamble 的值，TX 和 RX 共用： 当 PREAM_LEN_UNIT =0 时 8bit 有效， 当 PREAM_LEN_UNIT =1 时只有<3:0>有效

对于 RX 来说，Preamble 检测成功会有 PREAM_OK 中断产生。另外 Preamble 检测在整个接收阶段会一直进行，建议用户只在有需要的时候去检测这个中断。

2.3 Sync Word 配置

对应的 RFPDK 的界面和参数：

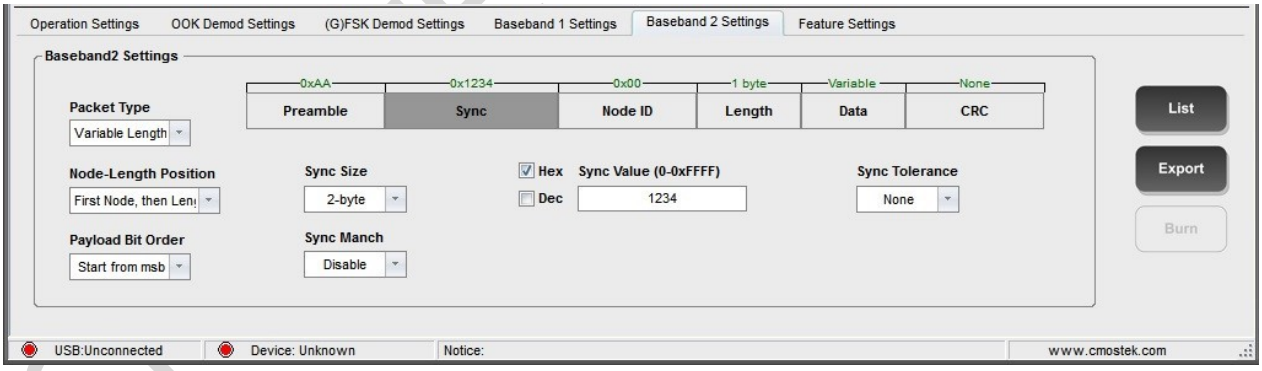

图 6. Sync Word 的 RFPDK 界面

表 10. Sync Word 的相关参数

寄存器比特 RFPDK 参数	寄存器比特
Sync Size	SYNC_SIZE<2:0>
Sync Value	SYNC_VALUE<63:0>

Sync Tolerance	SYNC_TOL<2:0>
Sync Manch	SYNC_MAN_EN

表 11.位于配置区的寄存器：

寄存器名	位数	R/W	比特名	功能说明
CUS_PKT5 (0x3C)	6:4	RW	SYNC_TOL<2:0>	RX 模式对 Sync Word 检测的容错比特数： 0:不允许有错 1:允许 1bit 接收错误 2:允许 2bits 接收错误 3:允许 3bits 接收错误 4:允许 4bits 接收错误 5:允许 5bits 接收错误 6:允许 6bits 接收错误 7:允许 7bits 接收错误
	3:1	RW	SYNC_SIZE<2:0>	Sync Word 长度： 0: 1 byte 1: 2 bytes 2: 3 bytes 3: 4 bytes 4: 5 bytes 5: 6 bytes 6: 7 bytes 7: 8bytes
	1:0	RW	SYNC_MAN_EN	Sync Word 的曼切斯特编解码使能： 0：不使能 1：使能
CUS_PKT6 (0x3D)	7:0	RW	SYNC_VALUE<7:0>	Sync Word 的值，根据不同的 SYNC_SIZE 设置来填入不同的寄存器，请看下表。
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>	
CUS_PKT11 (0x42)	7:0	RW	SYNC_VALUE<47:40>	
CUS_PKT12 (0x43)	7:0	RW	SYNC_VALUE<55:48>	
CUS_PKT13	7:0	RW	SYNC_VALUE<63:56>	

寄存器名	位数	R/W	比特名	功能说明
(0x44)				

	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	√	√	√	√	√	√	√	√

表中打勾的地方表示要填入的寄存器。举例说明，如果 SYNC_SIZE 设置为 1，即长度为 2 个 byte，同步字的值为 0x5678，那么用户就将这个值填入 SYNC_VALUE<63:56>和 SYNC_VALUE<55:48>两个寄存器，msb 对应的是第 63 位，lsb 对应的是第 48 位，即将 0x56 填入 SYNC_VALUE<63:56>，0x78 填入 SYNC_VALUE<55:48>。

另外，有些应用需要整个数据包都是进行曼切斯特编码的，而大部分应用仅需要对 Payload 进行编码，所以给 Sync Word 的部分单独做一个曼切斯特编码使能位。

2.4 数据包总体配置

对应的 **RFPDK** 的界面和参数：

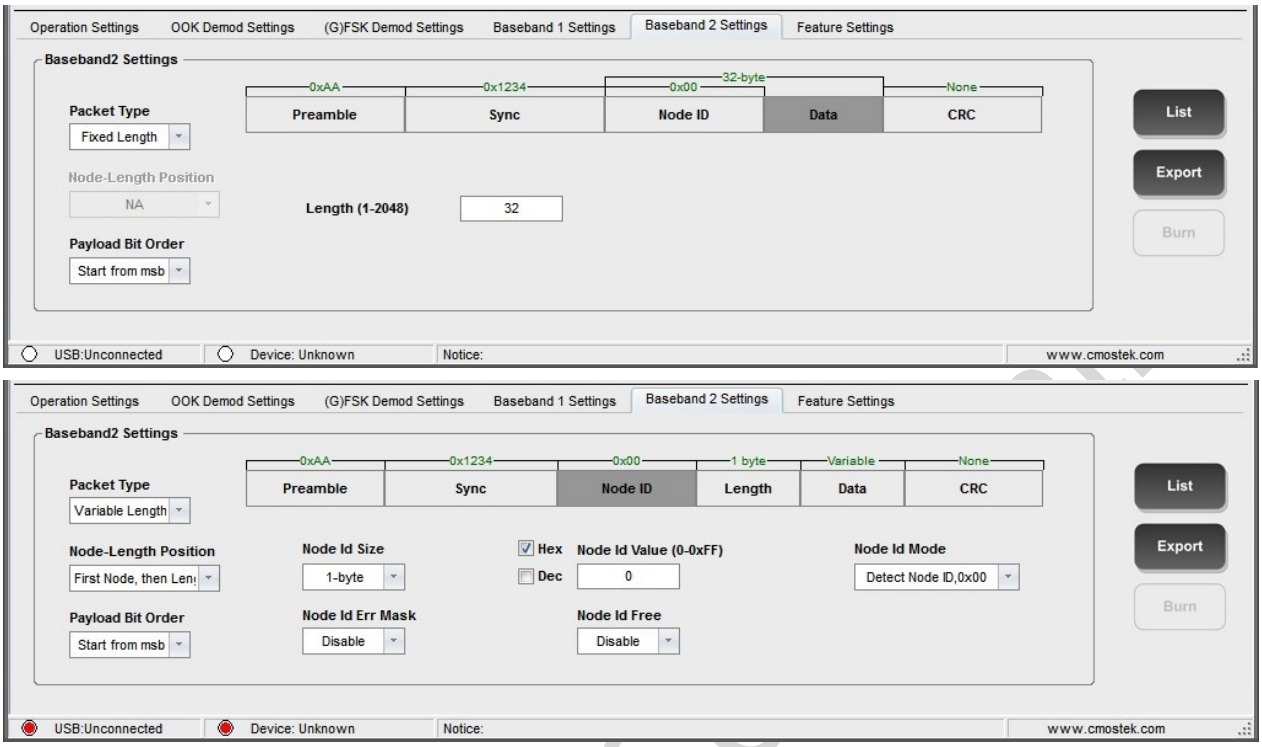

图 7. 数据包的 RFPDK 界面

表 12. 数据包的相关参数

寄存器比特 RFPDK 参数	寄存器比特
Packet Type	PKT_TYPE
根据各个参数综合计算出来，具体方法下文有介绍	PAYLOAD_LENG<10:0>
Node-Length Position	NODE_LENG_POS_SEL
Payload Bit Order	PAYLOAD_BIT_ORDER
RFPDK 中无输入，在应用中灵活使用，下文有介绍	AUTO_ACK_EN

表 13.位于配置区的寄存器：

寄存器名	位数	R/W	比特名	功能说明
CUS_PKT14 (0x45)	6:4	RW	PAYLOAD_LENG<10:8>	11-bit Payload 长度的<10:8>位。 在 PKT_TYPE 设置为定长包时，可配置的内容为 0-2047，指 1-2048 个 bytes 。在 PKT_TYPE 设置为可变包时，只有<7:0>有效，可配置长度是 1-256 个 bytes。
	3	RW	AUTO_ACK_EN	使能自动打包 ACK 数据包的功能 0：不使能 1：使能
	2	RW	NODE_LENG_POS_SEL	在可变包中，Node ID 和 Length Byte 的位置关系 0：Node ID 在 length Byte 之前 1：Node ID 在 length Byte 之后
	1	RW	PAYLOAD_BIT_ORDER	0: 先对 payload+CRC 每个 byte MSB 进行编解码 1: 先对 payload+CRC 每个 byte LSB 进行编解码
	0	RW	PKT_TYPE	包长类型 0：固定包长 1：可变包长
CUS_PKT15 (0x46)	7:0	RW	PAYLOAD_LENG<7:0>	12-bit Payload 长度的<7:0>位 说明同上。

下面详细解释一下 PAYLOAD_BIT_ORDER 的含义。

PAYLOAD_BIT_ORDER = 1 表示发送时把 Payload 和 CRC 的每个 byte 自身从 LSB 到 MSB 的顺序发送或进行 Manchester/Whiten 编码。另一方面，接收时须把解码后的 Payload 和 CRC 的每个 byte 自身的 MSB 和 LSB 顺序调换，再进行 CRC 解码。如果 RX 和 TX 的配置是一样的，那么用户是看不见这个过程的；如果使用我们的产品和别人的产品对接，那么就需要用户理解这个过程并进行合适的配置。

PAYLOAD_BIT_ORDER = 0 表示无此操作。

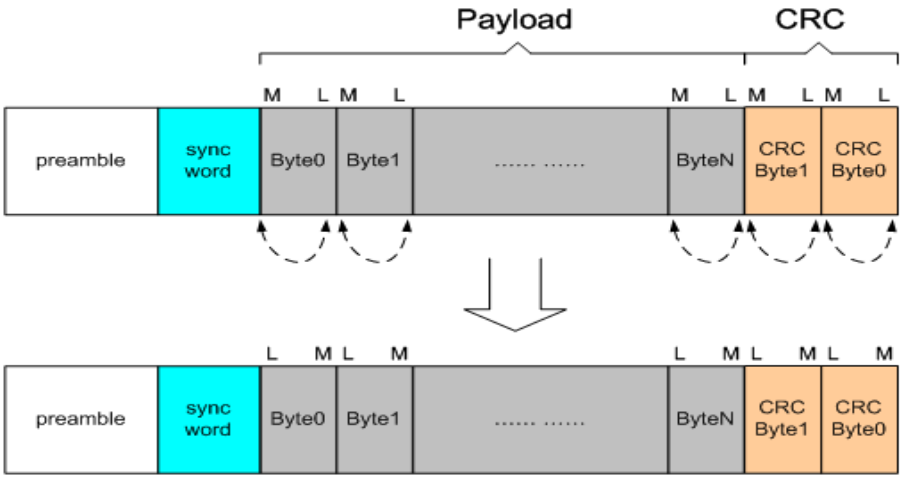

图 8. PAYLOAD_BIT_ORDER 操作

下面详细解释一下 AUTO_ACK_EN 的用法。

CMT2300A 的 AUTO ACK 功能，并不是指，在收到一个数据包后，直接自动切换到发射模式并发回 ACK 包。芯片内部并不支持这种控制模式。实际的用法是，当 MCU 将 AUTO_ACK_EN 设置为 1，那么在芯片内部，数据包的格式会自动配置成 ACK 的包格式，如下：

图 9. ACK 的包格式

这个数据包只包含 Preamble 和 Sync ID。然后外部 MCU 需要将芯片切换到 TX 模式进行发射，发射的内容就是上面这个数据包。当发射完成后，MCU 需要先将 AUTO_ACK_EN 设置为 0，再进行其它操作。

2.5 Node ID 配置

对应的 RFPDK 的界面和参数：

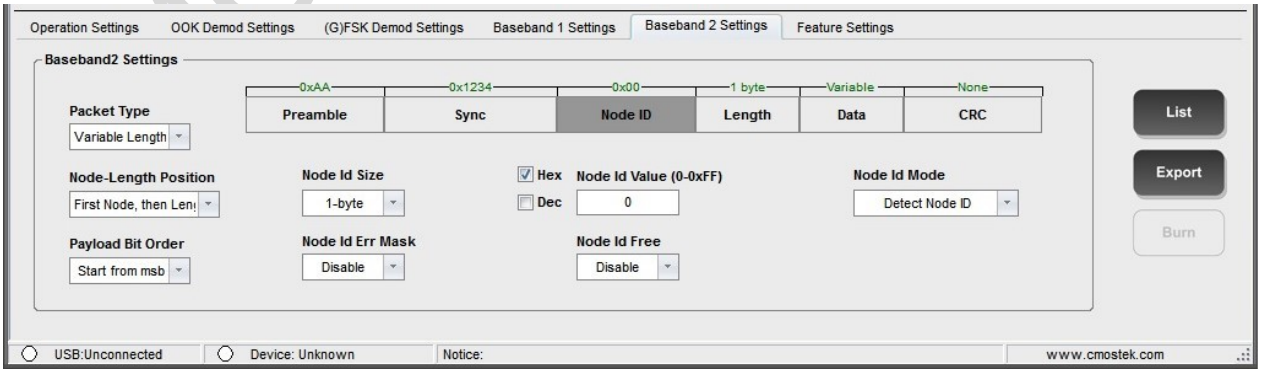

图 10. Node ID 的 RFPDK 界面

表 14. Node ID 的相关参数

寄存器比特 RFPDK 参数	寄存器比特
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

表 15.位于配置区的寄存器：

寄存器名	位数	R/W	比特名	功能说明
CUS_PKT16 (0x47)	5	RW	NODE_FREE_EN	在 RX 模式下，让 Node ID 检测电路独立出来的使能位。 0：不使能 1：使能
	3	RW	NODE_ERR_MASK	Node ID 检测错误，会输出 PKT_ERR 中断，同时可同步复位解码电路，该比特控制是否进行同步复位。 0: 允许同步复位 1: 不同步复位
	2	RW	NODE_SIZE<1:0>	Node ID 的长度： 0：1 byte 1：2 bytes 2：3 bytes 3：4 bytes
	1	RW	NODE_DET_MODE<1:0>	Node ID 的检测模式： 0：不检测 1：TX 模式发送 NODE_VALUE 的内容；RX 模式仅识别 NODE_VALUE 的内容 2：TX 模式发送 NODE_VALUE 的内容；RX 模式仅识别 NODE_VALUE 的内容和全 0 3：TX 模式发送 NODE_VALUE 的内容；RX 模式仅识别 NODE_VALUE 的内容，全 0 和全 1
CUS_PKT17 (0x48)	7:0	RW	NODE_VALUE<7:0>	32-bit Node ID 的值
CUS_PKT18 (0x49)	7:0	RW	NODE_VALUE<15:8>	
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	√	√	√	√

表中打勾的地方表示要填入的寄存器。举例说明，如果 NODE_SIZE 设置为 1，即长度为 2 个 byte，值为 0x5678，那么用户就将这个值填入 SYNC_VALUE<31:24>和 SYNC_VALUE<23:16>两个寄存器，msb 对应的是第 31 位，lsb 对应的是第 16 位，即将 0x56 填入 SYNC_VALUE<31:24>，0x78 填入 SYNC_VALUE<23:16>。

下面介绍一下，在不同的 PKT_TYPE 下，要怎样设置 PAYLOAD_LENG 和 NODE_SIZE，怎样确定 Payload 里面的 DATA 的长度，怎样解释 Length Byte 的含义。用户可以根据需要，下列表格中查找自己需要的理解的部分：

固定包格式：

图 11. 固定长度包格式

TX 模式	RX 模式
Payload 结构： Node ID + Data，其中 Node ID 可以不存在，TX 和 RX 配置完全一样。 举例： PAYLOAD_LENG<10:0> = 11 NODE_SIZE<1:0> = 2 那么 Payload 的总长度是 11 + 1 = 12，Node ID 的长度是 2 + 1 = 3，所以，Data 的长度是 12 – 3 = 8 个 byte。 假如 Node ID 不存在，那么 Data 的长度就是 12。	

可变包格式：

情况一：Node ID 不存在

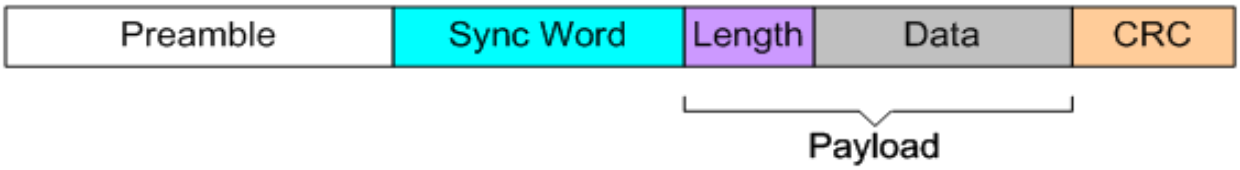

图 12. 可变长度包格式，Node ID 不存在

X 模式	RX 模式
Payload 结构: Length Byte + Data Payload 长度: 1 + PAYLOAD_LENG<7:0>，其中 PAYLOAD_LENG<7:0>的内容就等于 Length Byte 的内容，指的就是后面跟随的 Data 的长度，因此只能填 8 位，最大值为 255。前面的 1 表示 Length Byte 本身是 1 个 byte 的长度。 举例： PAYLOAD_LENG<7:0> = 11， 那么 Payload 的总长度是 1 + 11 = 12，所以 Data 的长度就是 11。	Payload 结构: Length Byte + Data Length Byte 内容含义： 表示随后 Data 的长度，与 TX 配置对应。

情况二：Node ID 存在，且 NODE_POSITION = 0（Node ID 在 Length Byte 之前）

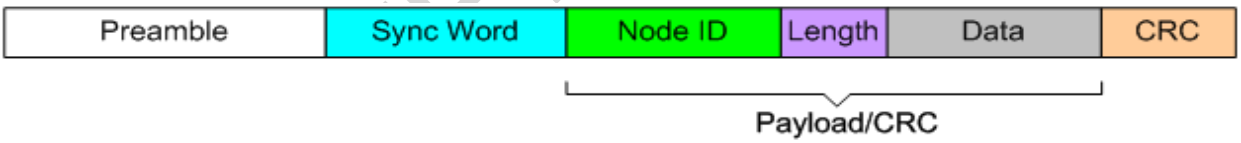

图 13. 可变长度包格式，Node ID 存在，Node ID 在 Length Byte 之前

TX 模式	RX 模式
Payload 结构: Node ID + Length Byte + Data Payload 长度: (NODE_SIZE+1) + 1 + PAYLOAD_LENG<7:0>，其中 PAYLOAD_LENG<7:0>的内容就等于 Length Byte 的内容，指的就是后面跟随的 Data 的长度，因此只能填 8 位，最大值为 255。中间的加 1 表示 Length	Payload 结构: Node ID + Length Byte + Data Length Byte 内容含义： 表示随后 Data 的长度，与 TX 配置对应。

Byte 本身是 1 个 byte 的长度。 举例： PAYLOAD_LENG<7:0> = 11， NODE_SIZE<1:0> = 2， 那么 Payload 的总长度是(2 + 1) + 1 + 11 = 15，所以 Data 的长度就是 11。	

情况三：Node ID 存在，且 NODE_POSITION = 1（Node ID 在 Length Byte 之后）

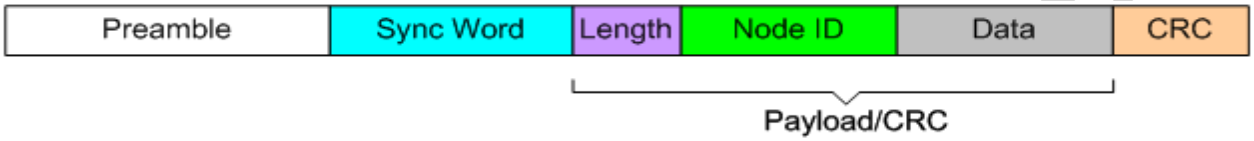

图 14. 可变长度包格式，Node ID 存在，Node ID 在 Length Byte 之后

TX 模式	RX 模式
Payload 结构: Length Byte + Node ID + Data Payload 长度: 1 + PAYLOAD_LENG<7:0>，其中 PAYLOAD_LENG<7:0>的内容就等于 Length Byte 的内容，指的就是后面跟随的 Node ID 加上 Data 的长度，因此只能填 8 位，最大值为 255。前面的 1 表示 Length Byte 本身是 1 个 byte 的长度。 举例： PAYLOAD_LENG<7:0> = 11， NODE_SIZE<1:0> = 2， 那么 Payload 的总长度是 1 + (2 + 1) + 8 = 12，所以 Data 的长度就是 8。	Payload 结构: Length Byte + Node ID + Data Length Byte 内容含义： 表示随后 Node ID + Data 的长度，与 TX 配置对应。

2.6 FEC 配置

对应的 RFPDK 的界面和参数：

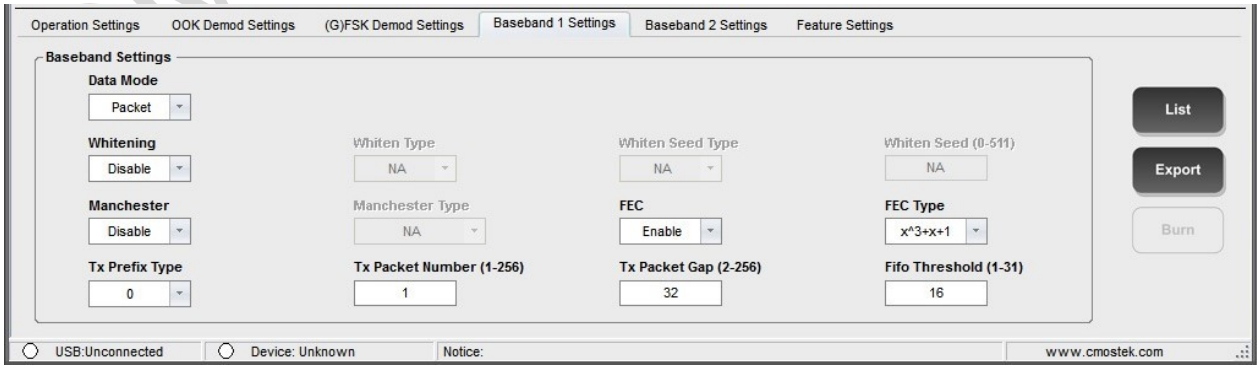

图 15.FEC 的 RFPDK 界面

表 16. FEC 的相关参数

寄存器比特 RFPDK 参数	寄存器比特
FEC	FEC_EN
FEC_Type	FEC_TYPE

表 17.位于配置区的寄存器：

寄存器名	位数	R/W	比特名	功能说明
CUS_PKT21 (0x4C)	7	RW	FEC_TYPE	FEC(7,4)编解码的多项式选择： 0: 多项式为 x^3+x+1 1: 多项式为 x^3+x^2+1
	6	RW	FEC_EN	FEC(7,4)编解码的使能： 0: 不使能 1: 使能

FEC 的主要作用是可以纠正一个包里面的一个错误数据，因此从现象来看，可以减低误包率。

2.7 CRC 配置

对应的 RFPDK 的界面和参数：

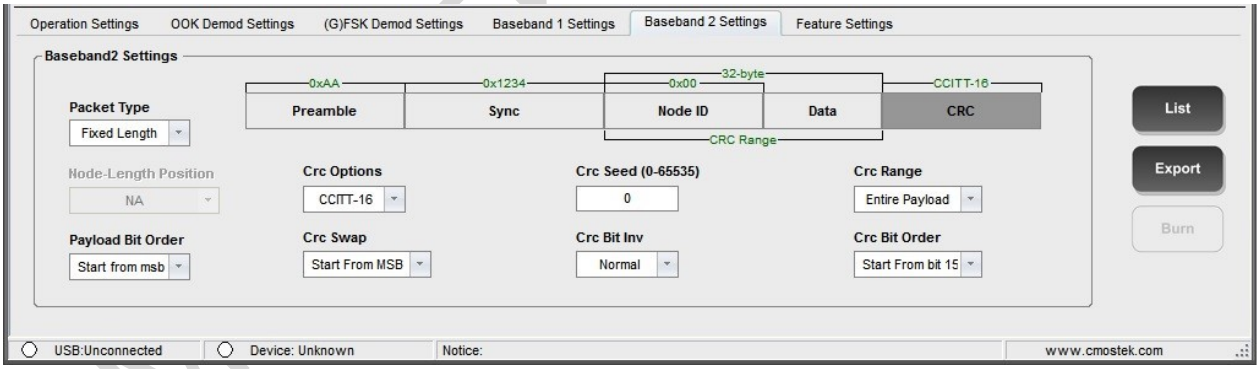

图 16. CRC 的 RFPDK 界面

表 18. CRC 的相关参数

寄存器比特 RFPDK 参数	寄存器比特
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

表 19.位于配置区的寄存器：

寄存器名	位数	R/W	比特名	功能说明
CUS_PKT21 (0x4C)	5	RW	CRC_BYTE_SWAP	CRC 的收发顺序： 0: 先收发高字节 1: 先收发低字节
	4	RW	CRC_BIT_INV	CRC 码是否取反： 0:CRC code 不取反 1:CRC code 逐位取反
	3	RW	CRC_RANGE	CRC 的计算范围： 0: 整个 payload 1: 仅为 data
	2:1	RW	CRC_TYPE<1:0>	CRC 多项式类型： 0:CCITT-16 1:IBM-16 2:ITU-16(相当于倒序 CCITT-16) 3: NA
	0	RW	CRC_EN	CRC 使能 0：不使能 1：使能
CUS_PKT22 (0x4D)	7:0	RW	CRC_SEED<7:0>	CRC 多项式的初始值
CUS_PKT23 (0x4E)	7:0	RW	CRC_SEED<15:8>	
CUS_PKT24 (0x4F)	7	RW	CRC_BIT_ORDER	CRC 大小端顺序配置： 0：CRC bytes 按 bit15 到 bit0 顺序收发 1：CRC bytes 按 bit 0 到 bit 15 顺序收发

下面详细解释一下 CRC 的几项配置的原理。

CRC_RANGE

这个功能就是指定 CRC 的编解码校验对象，可以是整个 Payload，也可以只是 Data 部分。

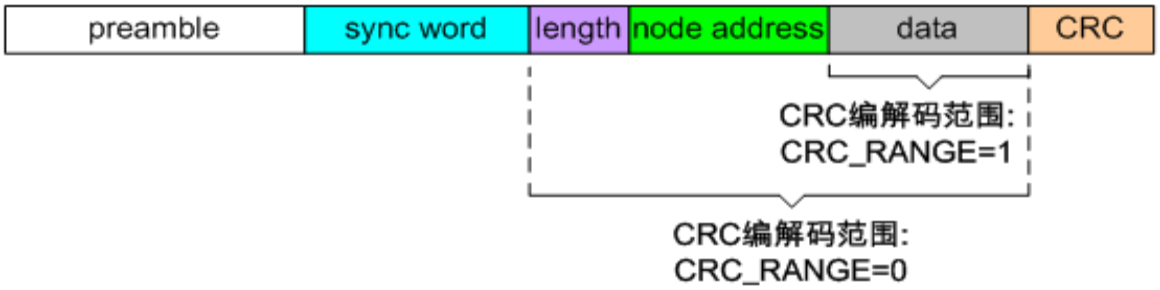

图 17. CRC 编码范围

CRC_BIT_INV

这个功能就是将 CRC 的每一位都进行逻辑取反，原来是 0 就变成 1，原来是 1 就变成 0。

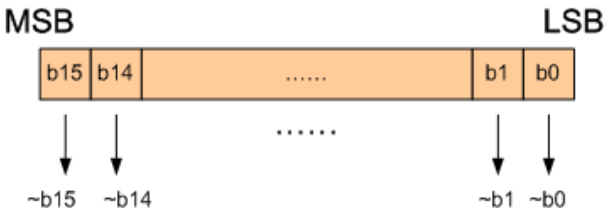

图 18.CRC_BIT_INV

CRC_BYTE_SWAP

这个功能就是将两个 Byte 的位置倒过来，但是不改变每个 Byte 里面的 Bit 的顺序。

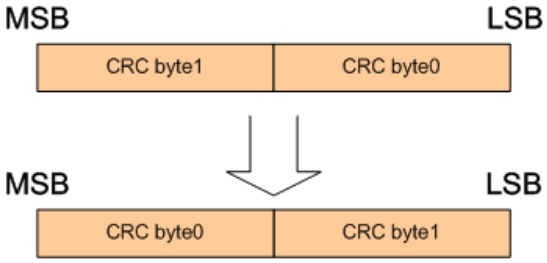

图 19.CRC_BYTE_SWAP

CRC_BIT_ORDER

这个功能就是将整个 CRC 部分的高地位顺序倒过来，如果两个 BYTE 的位置调换了，就会按照调换后的的高低位顺序来倒置。

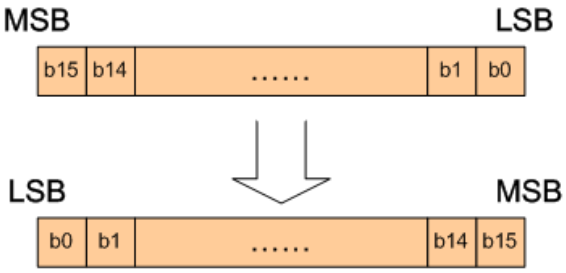

图 20.CRC_BIT_ORDER

2.8 编解码配置

对应的 RFPDK 的界面和参数：

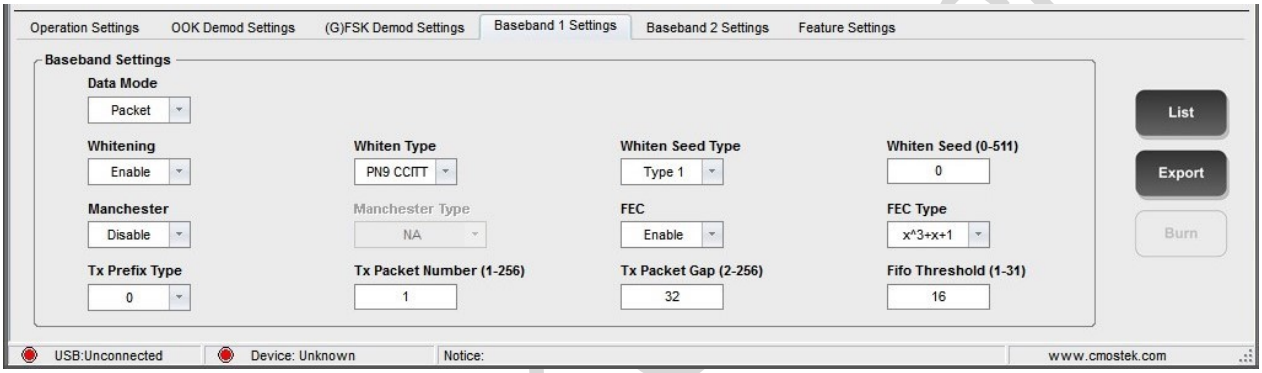

图 21.编解码的 RFPDK 界面

表 20. 编解码的相关参数

寄存器比特 RFPDK 参数	寄存器比特
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

表 21.位于配置区的寄存器：

寄存器名	位数	R/W	比特名	功能说明
CUS_PKT24 (0x4F)	6	RW	WHITEN_SEED<8>	白化编解码多项式的种子的第 8 位。 当白化编解码的方式为 PN9 时，种子取全 9bits；为 PN7 时，种子取低 7bits。

寄存器名	位数	R/W	比特名	功能说明
	5	RW	WHITEN_SEED_TYPE	白化编解码多项式为 PN7 时的种子类型： 0：按 A7139 的方式计算 PN7 seed 1：PN7 seed 为 whiten_seed 定义的值
	4:3	RW	WHITEN_TYPE<1:0>	白化编解码的方式： 0：PN9 CCITT 编解码 1：PN9 IBM 编解码 2：PN7 编解码 3：无效
	2	RW	WHITEN_EN	白化编解码的使能： 0: 无 whiten 编解码 1: 有 whiten 编解码
	1	RW	MANCH_TYPE	曼切斯特编解码的方式： 0：01 表示 1；10 表示 0 1：10 表示 1；01 表示 0
	0	RW	MANCH_EN	曼切斯特编解码的使能： 0：不使能 1：使能
CUS_PKT25 (0x50)	7:0	RW	WHITEN_SEED<7:0>	白化编解码多项式的种子的 7:0 位。

2.9 TX 数据包专用配置

对应的 **RFPDK** 的界面和参数：

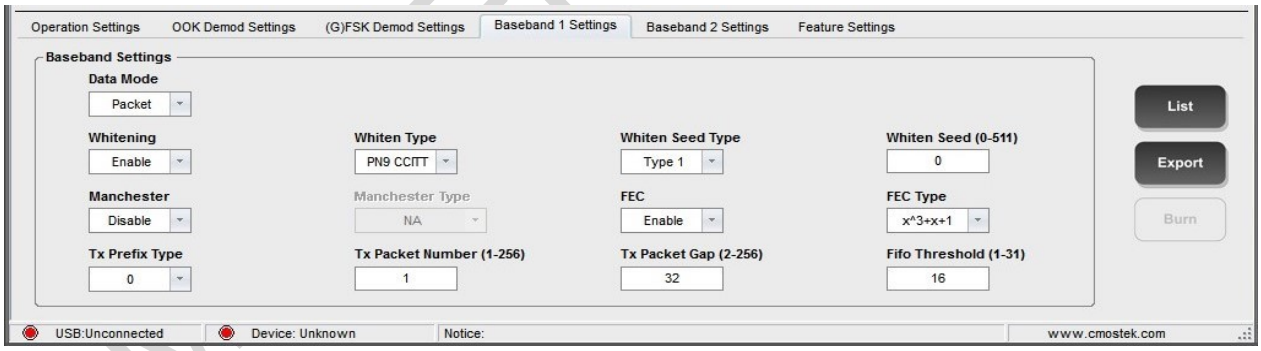

图 22. TX 数据包的 RFPDK 界面

表 22. TX 数据包的相关参数

寄存器比特 RFPDK 参数	寄存器比特
Tx Prefix Type	TX_PREFIX_TYPE<1:0>
Tx Packet Number	TX_PKT_NUM<7:0>
Tx Packet Gap	TX_PKT_GAP<7:0>

表 23.位于配置区的寄存器：

寄存器名	位数	R/W	比特名	功能说明
CUS_PKT26 (0x51)	1:0	RW	TX_PREFIX_TYPE<1:0>	TX Prefix 是指在 Packet 模式下，进入发射状态后，由于 FIFO 数据还没有准备好，但是 PA 已经开始发射了，就需要定义预发射的内容，可以定义为： 0：发送 0 1：发送 1 2：发送 Preamble 3：NA
CUS_PKT27 (0x52)	7:0	RW	TX_PKT_NUM<7:0>	TX 模式下每次重复发的包个数： 0-255 表示发送 1-256 个包
CUS_PKT28 (0x53)	7:0	RW	TX_PKT_GAP<7:0>	TX 模式下重复发包时，包与包之间的间隔： 0-255 表示包与包之间的发送间隔为 1-256 个 Symbol

2.10 Direct 发射模式

如前面的章节介绍，在发射模式中，当 DATA_MODE 配置成 Direct 的情况下，发射的数据会直接来源于 GPIO。下面是与 Direct 发射模式相关的寄存器:

配置寄存器

寄存器名	位数	R/W	比特名	功能说明
CUS_PKT1 (0x38)	1:0	RW	DATA_MODE<1:0>	选择接收和发射的数据模式： 0：Direct 模式（默认） 1：NA 2：Packet 模式 3：NA
CUS_TX1 (0x55)	2	RW	TX_DIN_SOURCE	选择 TX 的数据来源位置： 0：TX 数据从 TX FIFO 获取 1：TX 数据从 GPIO 直接输入

控制寄存器

寄存器名	位数	R/W	比特名	功能说明
CUS_IO_SEL (0x65)	5:4	RW	GPIO3_SEL<1:0>	GPIO3 的选项： 0：CLKO 1：DOUT/DIN 2：INT2 3：DCLK (TX/RX)

	3:2	RW	GPIO2_SEL<1:0>	GPIO2 的选项： 0：INT1 1：INT2 2：DOUT/DIN 3：DCLK (TX/RX)
	1:0	RW	GPIO1_SEL<1:0>	GPIO1 的选项： 0：DOUT/DIN 1：INT1 2：INT2 3：DCLK (TX/RX)
CUS_INT2_CTL (0x67)	5	RW	TX_DIN_INV	发射数据反转控制，无论发射数据来源于 TX FIFO 还是 GPIO，该比特都有效： 0：不反转发射数据 1：反转发射数据
CUS_FIFO_CTL (0x69)	7	RW	TX_DIN_EN	用于选择 GPIO 中 DOUT/DIN 的选项： 0：使能 DOUT 1：使能 DIN
	6:5	RW	TX_DIN_SEL<1:0>	选择发射数据从哪个 GPIO 输入： 0：GPIO1 1：GPIO2 2：GPIO3 3：一直发数据 1

用户必须将 DATA_MODE 配置成 0，将 TX_DIN_SOURCE 配置成 1.

用户必须在发射前将这些寄存器配置好。在发送 GO_TX 命令之后，外部 MCU 在相应的 GPIO 送入发射数据。需要注意的是，在 Direct 模式下，发射数据的数据率受控于 MCU，应该尽量与芯片配置的数据率接近。

另外，用户可以使用 DCLK 来辅助数据的同步，建议 MCU 在 DCLK 的下降沿发出数据。当发射完成后，可发送 GO_*命令来切换模式，需要注意的是，一般情况下，当芯片接收到 GO_*命令后，需要 2-3 个 symbol 的时间来进行 PA RAMP DOWN 的操作，完成后才会切换到目标状态。

3. GPIO 和中断

CMT2300A 有 3 个 GPIO，每个 GPIO 都可以配置成不同的输入或者输出；CMT2300A 有 2 个中断口，可以配置到不同的 GPIO 输出，下面逐一介绍。

3.1 GPIO 的配置

下面是跟 GPIO 配置相关的寄存器，在配置区 1 里：

表 24. GPIO 相关寄存器

寄存器名	位数	R/W	比特名	功能说明
CUS_IO_SEL (0x65)	5:4	RW	GPIO3_SEL<1:0>	GPIO3 的选项： 0：CLKO 1：DOUT/DIN 2：INT2 3：DCLK (TX/RX)
	3:2	RW	GPIO2_SEL<1:0>	GPIO2 的选项： 0：INT1 1：INT2 2：DOUT/DIN 3：DCLK (TX/RX)
	1:0	RW	GPIO1_SEL<1:0>	GPIO1 的选项： 0：DOUT/DIN 1：INT1 2：INT2 3：DCLK (TX/RX)

上表中有几点需要注意的：DCLK 是在 direct 模式下给 MCU 用作发射或者接收数据的同步时钟，进入 RX 或者 TX 状态后，会自动切换成为接收或者发射同步。

3.2 中断的配置和映射

CMT2300A 有两个中断口，分别是 INT1 和 INT2，可以分配到不同的 GPIO 上。下面是中断相关的寄存器

表 25.控制区 1

寄存器名	位数	R/W	比特名	功能说明
CUS_INT1_CTL (0x66)	5	RW	INT_POLAR	中断输出极性选择： 0：0 无效，1 有效 1：0 有效，1 无效
	4:0	RW	INT1_SEL<4:0>	INT1 的映射选项，请参考下面的中断映射表。
CUS_INT2_CTL (0x67)	4:0	RW	INT2_SEL<4:0>	INT2 的映射选项，请参考下面的中断映射表。
CUS_INT_EN (0x68)	7	RW	SL_TMO_EN	睡眠超时中断使能 0：不使能 1：使能
	6	RW	RX_TMO_EN	接收超时中断使能 0：不使能 1：使能
	5	RW	TX_DONE_EN	发射完成中断使能 0：不使能 1：使能
	4	RW	PREAM_OK_EN	Preamble 检测成功中断使能 0：不使能 1：使能
	3	RW	SYNC_OK_EN	Sync Word 检测成功中断使能 0：不使能 1：使能
	2	RW	NODE_OK_EN	Node ID 检测成功中断使能 0：不使能 1：使能
	1	RW	CRC_OK_EN	CRC 检测成功中断使能 0：不使能 1：使能
	0	RW	PKT_DONE_EN	Packet 接收完成（不管对错）中断使能 0：不使能 1：使能
CUS_INT_CLR1 (0x6A)	5	RW	SL_TMO_FLG	SL_TMO 中断标志
	4	RW	RX_TMO_FLG	RX_TMO 中断标志
	3	RW	TX_DONE_FLG	TX_DONE 中断标志
	2	RW	TX_DONE_CLR	TX_DONE 中断清零 0：无动作 1：清零
	1	RW	SL_TMO_CLR	SL_TMO 中断清零 0：无动作 1：清零
	0	RW	RX_TMO_CLR	RX_TMO 中断清零 0：无动作 1：清零

表 26.控制区 2

寄存器名	位数	R/W	比特名	功能说明
CUS_INT_CLR2 (0x6B)	5	RW	LBD_CLR	LBD 有效（成功检测到低电压）中断清零 0：无动作 1：清零
	4	RW	PREAM_OK_CLR	PREAM_OK 中断清零 0：无动作 1：清零
	3	RW	SYNC_OK_CLR	SYNC_OK 中断清零 0：无动作 1：清零
	2	RW	NODE_OK_CLR	NODE_OK 中断清零 0：无动作 1：清零
	1	RW	CRC_OK_CLR	CRC_OK 中断清零 0：无动作 1：清零
	0	RW	PKT_DONE_CLR	PKT_DONE 中断清零 0：无动作 1：清零
CUS_INT_FLAG (0x6D)	7	RW	LBD_FLG	LBD 有效（成功检测到低电压）中断标志
	6	RW	COL_ERR_FLG	COL_ERR 中断标志
	5	RW	PKT_ERR_FLG	PKT_ERR 中断标志
	4	RW	PREAM_OK_FLG	PREAM_OK 中断标志
	3	RW	SYNC_OK_FLG	SYNC_OK 中断标志
	2	RW	NODE_OK_FLG	NODE_OK 中断标志
	1	RW	CRC_OK_FLG	CRC_OK 中断标志
	0	RW	PKT_OK_FLG	PKT_OK 中断标志

下面给出中断映射表，INT1 和 INT2 的映射是一样的，下面以 INT1 为例说明：

表 27.CMT2300A 中断映射表

名称	INT1_SEL	描述	清除方式
RX_ACTIVE	00000	指示准备进入 RX 和已经进入 RX 的中断，在 PLL 校正和 RX 状态下为 1，其余时候为 0。	Auto
TX_ACTIVE	00001	指示准备进入 TX 和已经进入 TX 的中断，在 PLL 校正和 TX 状态下为 1，其余时候为 0。	Auto
RSSI_VLD	00010	指示 RSSI 是否有效的中断	Auto
PREAM_OK	00011	指示成功收到 Preamble 的中断	by MCU
SYNC_OK	00100	指示成功收到 Sync Word 的中断	by MCU
NODE_OK	00101	指示成功收到 Node ID 的中断	by MCU

名称	INT1_SEL	描述	清除方式
CRC_OK	00110	指示成功收到并通过 CRC 校验的中断	by MCU
PKT_OK	00111	指示完整收到一个数据包的中断	by MCU
SL_TMO	01000	指示 SLEEP 计数器超时的中断	by MCU
RX_TMO	01001	指示 RX 计数器超时的中断	by MCU
TX_DONE	01010	指示 TX 完成的中断	by MCU
RX_FIFO_NMTY	01011	指示 RX FIFO 非空的中断	Auto
RX_FIFO_TH	01100	指示 RX FIFO 未读内容超过 FIFO TH 的中断	Auto
RX_FIFO_FULL	01101	指示 RX FIFO 填满的中断	Auto
RX_FIFO_WBYTE	01110	指示 RX FIFO 每写入一个 BYTE 的中断，是脉冲	Auto
RX_FIFO_OVF	01111	指示 RX FIFO 溢出的中断	Auto
TX_FIFO_NMTY	10000	指示 TX FIFO 非空的中断	Auto
TX_FIFO_TH	10001	指示 TX FIFO 未读内容超过 FIFO TH 的中断	Auto
TX_FIFO_FULL	10010	指示 TX FIFO 满的中断	Auto
STATE_IS_STBY	10011	指示当前状态是 STBY 的中断	Auto
STATE_IS_FS	10100	指示当前状态是 RFS 或 TFS 的中断	Auto
STATE_IS_RX	10101	指示当前状态是 RX 的中断	Auto
STATE_IS_TX	10110	指示当前状态是 TX 的中断	Auto
LBD	10111	指示低电压检测有效（VDD 低于设置的 TH）的中断	Auto
TRX_ACTIVE	11000	指示准备进入 RX 或者 RX 和已经进入 RX 或者 TX 的中断，在 PLL 校正，RX 状态，或 TX 状态下为 1，其余时候为 0。	Auto
PKT_DONE	11001	指示当前的数据包已经接收完成，会有下面 4 种情况： 1. 完整地接收到整个数据包 2. 曼切斯特解码错误，解码电路自动重启 3. NODE ID 接收错误，解码电路自动重启 4. 发现信号冲突，解码电路不自动重启，等待 MCU 处理	by MCU

中断默认是 1 有效，但是可以通过将 INT_POLAR 这个寄存器比特设置成 1，使所有中断都变成 0 有效。下面还是以 INT1 为例，画出了所有中断源的控制和选择图。对于控制和映射来说，INT1 和 INT2 也是一样的。

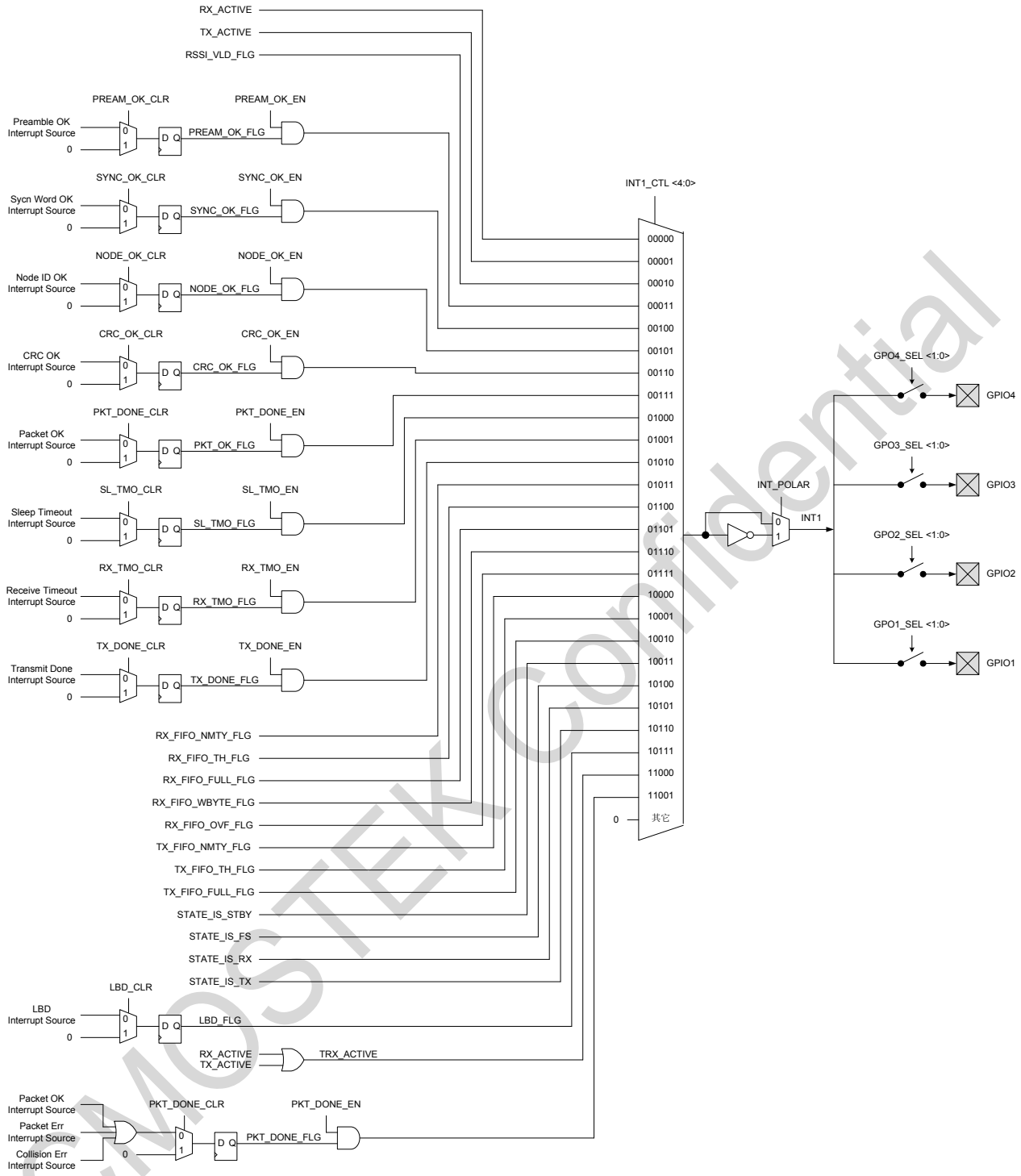

图 23. CMT2300A INT1 中断映射图

对于那些需要让 MCU 清零的中断源来说，每一个都配有一个 EN 使能和一个 CLR 清除比特，除了 LBD 有点特殊，它没有 EN 比特。例如，SYNC_OK 的中断源只有在 SYNC_OK_EN 这个比特设为 1 时才会产生，当这个中断产生后，只有将 SYNC_OK_CLR 设置为 1 就会将它清零。当设置 CLR 比特的时候，MCU 只需要将它设为 1 就可以了，随后不需要将它设置回 0，因为在芯片内部这个 CLR 比特在对应的中断清零之后，也会会自动清零。

PKT_OK 和 PKT_DONE 两个中断有本质区别，PKT_OK 是 PKT_DONE 的其中一种情况，即包已经完整地接收完成了。但是在实际应用中，在成功检测到 SYNC WORD 之后，会有 3 种意外情况发生：

1. 如果包里面有 NODE ID，有可能会发生 NODE ID 检测出错。这时 PKT_ERR_FLG 标志位会置起，芯片会停止接收包，并自动重启解码器，等待下一个 SYNC WORD 来临。这种情况下，PKT_OK 是不会产生的。

2. 如果使能了曼切斯特解码，有可能会发生解码出错。这时 PKT_ERR_FLG 标志位会置起，芯片会停止接收包，自动重启解码器，等待下一个 SYNC WORD 来临。这种情况下，PKT_OK 是不会产生的。

3. 如果使能了信号冲突检测（后面有介绍具体用法），有可能会发现有信号冲突，会导致后面接收的包的内容出错。这时 COL_ERR_FLG 标志位会置起，同时芯片会继续接收包直到完成，解码器不会自动重启。这种情况下，PKT_OK 还是会产生的，但是收到的数据是错的。

无论是发生哪一种情况，都需要告诉外部的 MCU，否则 MCU 会一直等待中断，会发生与芯片交互失败的情况。所以，我们就给出中断信号 PKT_DONE = PKT_OK | PKT_ERR_FLG | COL_ERR_FLG，即无论哪一种情况发生，都会通知 MCU。MCU 在收到这个中断后，就可以查询相关的 3 个标志位，得知究竟发生了什么事，再进行后续处理。

另外一种处理方式，就是 MCU 可以只等待 PKT_OK，再检查 CRC（如果有）的标志位，但是要注意与 SYNC_OK 中断的配合。例如，第一个包出现解码出错，解码器立即重新再接收下一个包，对于 MCU 来说，会出现连续出现两次 SYNC_OK 中断的情况，如果 MCU 没有处理好，就会以为第二个 SYNC_OK 中断是 PKT_OK 中断，就会处理错误。

总之，使用编解码中断，会碰到比较多变的情况，不同的用户的理解不一样，建议应用程序尽量简单，但要将稳定性排在第一位，最好自带超时机制。如果发生 MCU 与芯片交互失败（失联），就会出现死机状况了。

Cmt2300GPIO 输出中断配置代码示例详见附录 2。

关于 TX_DONE 中断的使用：

当用户使用 Packet 模式进行发射时，CMT2300 在发射完当前数据包（或者用户设置的 N 个数据包）后，会自动退出 TX 状态。安全退出 TX 状态时，TX_DONE 中断就会产生。

因此，用户只需要预先通过设置 TX_EXIT_STATE<1:0>这个寄存器，来设定芯片自动退出 TX 状态之后，是切换到 SLEEP/STBY/TFS 状态。然后，MCU 只需要检测 TX_DONE 中断来判断是否退出 TX，而无须通过发送状态切换命令来进行退出 TX 的操作。

3.3 天线 TX/RX 切换控制

下面是控制外部天线进行 TX/RX 切换控制相关的寄存器：

表 28. TX/RX 切换控制相关的寄存器

寄存器名	位数	R/W	比特名	功能说明
CUS_INT1_CTL (0x66)	7	RW	RF_SWT1_EN	天线开关信号使能 1 0：不使能 1：使能 不能和 RF_SWT2_EN 配置成同时有效
	6	RW	RF_SWT2_EN	天线开关信号使能 2 0：不使能 1：使能 不能和 RF_SWT1_EN 配置成同时有效

芯片可以在 GPIO1 和 GPIO2 输出一组（2 个不同的）信号，来控制外部天线的 RX/TX 切换。我们可以选择输出 2 组具有不同时序特性的控制信号，分别由两个使能信号 RF_SWT1_EN 和 RF_SWT2_EN 来使能：

1. 当 RF_SWT1_EN 使能的时候，GPIO1 会输出 RX_ACTIVE，GPIO2 会输出 TX_ACTIVE。
2. 当 RF_SWT2_EN 使能的时候，GPIO1 会输出 RX_ACTIVE，GPIO2 会输出 RX_ACTIVE 取反，即完全差分。

用户需要注意的是，只要 RF_SWT1_EN 或者 RF_SWT2_EN 任意一个打开，另外一个就不能打开，而且 GPIO1_SEL 和 GPIO2_SEL 就会失效，即天线切换控制具有最高的优先级。

用户可以先测试观察两种控制方式的时序区别，根据实际情况决定使用哪一种。

4. 附录

4.1 附录 1 Sample code FIFO 读写操作代码示例

Sample code FIFO 读取使能操作子函数

```
/*! ********************************************************
* @name    Cmt2300A_EnableReadFifo
* @desc    Enable SPI to read the FIFO.
* ********************************************************/
void Cmt2300A_EnableReadFifo(void)
{
    u8 tmp = Cmt2300A_ReadReg(CMT2300A_CUS_FIFO_CTL);
    tmp &= ~CMT2300A_MASK_SPI_FIFO_RD_WR_SEL;
    tmp &= ~CMT2300A_MASK_FIFO_RX_TX_SEL;
    Cmt2300A_WriteReg(CMT2300A_CUS_FIFO_CTL, tmp);
}

/*! ********************************************************
* @name    Cmt2300A_EnableWriteFifo
* @desc    Enable SPI to write the FIFO.
* ********************************************************/
void Cmt2300A_EnableWriteFifo(void)
{
    u8 tmp = Cmt2300A_ReadReg(CMT2300A_CUS_FIFO_CTL);
    tmp |= CMT2300A_MASK_SPI_FIFO_RD_WR_SEL;
    tmp |= CMT2300A_MASK_FIFO_RX_TX_SEL;
    Cmt2300A_WriteReg(CMT2300A_CUS_FIFO_CTL, tmp);
}
```

4.2 附录 2 Sample code GPIO 输出中断配置函数示例

```
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 */
    Cmt2300A_EnableAntennaSwitch(0);
#else
```

```
    /* Config GPIOs */
    Cmt2300A_ConfigGpio(
        CMT2300A_GPIO1_SEL_INT1 |  /* INT1 > GPIO1 */
        CMT2300A_GPIO2_SEL_INT2 |  /* INT2 > GPIO2 */
        CMT2300A_GPIO3_SEL_DOUT  /*DOUT>GPIO3*/
        );

    /* Config interrupt */
    Cmt2300A_ConfigInterrupt(
        CMT2300A_INT_SEL_TX_DONE, /* Config INT1 */
        CMT2300A_INT_SEL_PKT_OK   /* Config INT2 */
        );
#endif

    /* Enable interrupt */
    Cmt2300A_EnableInterrupt(
        CMT2300A_MASK_TX_DONE_EN  |
        CMT2300A_MASK_PREAM_OK_EN |
        CMT2300A_MASK_SYNC_OK_EN  |
        CMT2300A_MASK_NODE_OK_EN  |
        CMT2300A_MASK_CRC_OK_EN   |
        CMT2300A_MASK_PKT_DONE_EN
        );
    Cmt2300A_EnableLfosc(FALSE);
    /* Use a single 64-byte FIFO for either Tx or Rx */
    //Cmt2300A_EnableFifoMerge(TRUE);
    //Cmt2300A_SetFifoThreshold(16);

    /* Go to sleep for configuration to take effect */
    Cmt2300A_GoSleep();
}
```

5. 文档变更记录

表 29.文档变更记录表

版本号	章节	变更描述	日期
0.8	所有	初始版本发布	2017-03-29
0.9	概要	加入阅读 AN142 的建议	2017-07-12
	所有	修改个别错别字和文字格式	
1.0	表 25	INT_POLAR 纰漏改正	2018-05-09
	表 5	FIFO FULL、NMTY、TH 等三个描述顺序颠倒了	
	Page39	描述 TX FIFO 操作时，应该为 FIFO_CLR_TX，而不是 FIFO_CLR_RX	

6. 联系方式

无锡泽太微电子有限公司深圳分公司

中国广东省深圳市南山区前海路鸿海大厦 203 室

邮编： 518000

电话： +86 - 755 - 83235017

传真： +86 - 755 - 82761326

销售： sales@cmostek.com

技术支持： support@cmostek.com

网址： www.cmostek.com