

CMT2310A FIFO和包格式使用指南

概要

本文介绍了 CMT2310A 的 FIFO，包格式，以及中断系统的工作原理。本文中配置寄存器的相关内容介绍与 RFPDK 上可输入的参数相对应，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
CMT2310A	117 - 1050 MHz	(4)(G)FSK/OOK	收发一体机	寄存器	QFN24

阅读此文档之前，建议阅读《AN237 CMT2310A 快速上手指南》以了解 CMT2310A 的基本使用方式。

目 录

1. FIFO 工作原理.....	3
1.1 FIFO 相关的寄存器.....	3
1.2 FIFO 的工作模式	5
1.3 FIFO 的中断时序	7
1.4 FIFO 的应用场景	8
1.4.1 应用场景一：在 RX 下接收数据	8
1.4.2 应用场景二：预先填好数据，进入 TX 发射	8
1.4.3 应用场景三：进入 TX 后，一边填数据一边发射	9
1.4.4 应用场景四：每次都重复发同一个或同一组数据包.....	9
1.4.5 应用场景五：一个数据包分开几次发	9
2. 包格式介绍	10
2.1 数据模式配置	10
2.2 Preamble 配置	11
2.3 Sync Word 配置	13
2.4 数据包总体配置	16
2.5 Address 配置	18
2.6 FCS1 配置	20
2.7 FCS2 配置	22
2.8 CRC 配置.....	28
2.9 编解码配置	32
2.10 TX 数据包专用配置.....	35
2.11 Direct 发射模式.....	36
3. GPIO 和中断	38
3.1 GPIO 的配置.....	38
3.2 中断的配置和映射.....	39
3.3 天线 TX/RX 切换控制	46
4. 附录.....	47
4.1 附录 1 Sample code FIFO 读写操作代码示例	47
4.2 附录 2 Sample code GPIO 输出中断配置函数示例	47
5. 文档变更记录	49
6. 联系方式.....	50

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

寄存器的内容和描述如下表。

表 3. 位于 Page 0 的控制寄存器

寄存器名	位数	R/W	比特名	功能说明
CTL_REG_19 (0x13)	6	RW	PD_FIFO	0: 在 SLEEP 状态下保存 FIFO 内容 1: 在 SLEEP 状态下不保存 FIFO 内容
	5	RW	FIFO_TH<8>	FIFO_TH 的第 8 位。
	4	RW	FIFO_AUTO_CLR_RX_EN	配置进入 RX 前是否自动清除 RX FIFO 的内容： 0: 不清除 1: 清除
	3	RW	FIFO_AUTO_RES_TX_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)，这个比特必须设成 1。
	2	RW	FIFO_TX_TEST_EN	0: TX FIFO 只能用 SPI 写；1: TX FIFO 可被 SPI 读取。该比特只对 TX FIFO 有

寄存器名	位数	R/W	比特名	功能说明
				效，除用于用户测试使用外，其余情况均应设置为 0。
	1	RW	FIFO_MERGE_EN	0: 分为 2 个独立的 128-byte 的 FIFO， 1: 合并成 1 个 256-byte 的 FIFO。
	0	RW	FIFO_TX_RX_SEL	当 FIFO 为合并模式时， 1: FIFO 用作 TX FIFO 0: FIFO 用作 RX FIFO
CTL_REG_20 (0x14)	7:0	RW	FIFO_TH<7:0>	FIFO 的填入阈值，单位为 byte，对 RX 来说，当未读数据超过此阈值时，RX_FIFO_TH_FLG 会置 1；对 TX 来说，当未发数据小于该阈值时，TX_FIFO_TH_FLG 会置 0。 当 FIFO_MERGE_EN = 0 时，有效范围为 1 到 127。 当 FIFO_MERGE_EN = 1 时，有效范围为 1 到 255。
CTL_REG_27 (0x1B)	2	W	TX_FIFO_RESTORE	用于用户手动 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	RX_FIFO_CLR	0: 无效；1: 清零 RX FIFO。 用户将此比特设为 1 之后，无需将其再设回 0，这个比特在内部会自动设回为 0。
	0	W	TX_FIFO_CLR	0: 无效；1: 清零 TX FIFO。 用户将此比特设为 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。
CTL_REG_28 (0x1C)	7	R	RX_FIFO_FULL_FLG	指示 RX FIFO 填满的中断。 0: 无效 1: 有效
	6	R	RX_FIFO_NMTY_FLG	指示 RX FIFO 非空的中断标志位。 0: 无效 1: 有效
	5	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。

寄存器名	位数	R/W	比特名	功能说明
				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

表 4. 位于 Page 0 的配置寄存器

寄存器名	位数	R/W	比特名	功能说明
CTL_REG_40 (0x28)	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

1.2 FIFO 的工作模式

CMT2310A 默认提供两个独立的 128-byte 的 FIFO，分别给 RX 和 TX 使用，两者互不相干。用户也可以将 FIFO_MERGE_EN 设为 1，那么两个 FIFO 就可合成一个 256-byte 的 FIFO，在 TX 和 RX 下都可以使用，通过配置 FIFO_TX_RX_SEL 来指示目前是用作 TX 还是 RX。

通常我们建议，在 STBY 状态下将 FIFO 的工作模式进行预配置，然后才开始 TX/RX 工作。CTL_REG_19, CTL_REG_20, CTL_REG_40 寄存器在 SLEEP 状态均可配置和保存内容，因此，除非要改变工作模式，否则配置一次即可。而 CTL_REG_27 和 CTL_REG_28 寄存器在 SLEEP 状态下不可访问。

用户可以选择在 SLEEP 状态下，FIFO 的内容是否需要保存，通过 PD_FIFO 比特进行配置。在 SLEEP 状态下不保存 FIFO 的内容，可进一步降低漏电指标。

分开的 FIFO 的预配置

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

如果要去 RX:

- a) 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_RX_EN 设置为 1, 否则设置为 0, 然后进行将 RX_FIFO_CLR 置 1 进行手动清零。

如果要去 TX:

- a) 将 TX_FIFO_CLR 置 1 进行手动清零, 然后直接写入 FIFO 即可。

合并的 FIFO 的预配置

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

如果要去 RX:

- a) 将 FIFO_TX_RX_SEL 设置为 0 (RX 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_RX_EN 设置为 1, 否则设置为 0, 然后进行将 RX_FIFO_CLR 置 1 进行手动清零。

如果要去 TX:

- a) 将 FIFO_TX_RX_SEL 设置为 1 (TX 模式);
- b) 将 TX_FIFO_CLR 置 1 进行手动清零, 然后直接写入 FIFO 即可。

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 TX_FIFO_CLR 寄存器位进行清零; 在 RX FIFO 模式下, 只能使用 RX_FIFO_CLR 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对 FIFO_RX_TX_SEL 进行配置。

```
cmt2310a_go_ready();
cmt2310a_delay_ms(2);
cmt2310a_clear_interrupt_flag_0(M_TX_DONE_CLR);
/* if fifo merge is enable, enable spi to write the FIFO */
//cmt2310a_fifo_tx_rx_sel(1);
cmt2310a_clear_tx_fifo();

/* The length need be smaller than 128 */
cmt2310a_write_fifo(g_ptr_tx_buffer, g_tx_length);
```

FIFO 读写操作代码示例详见附录 1。

1.3 FIFO 的中断时序

下图为 RX FIFO 和 TX FIFO 工作时的中断时序示意图，方便用户参理解。

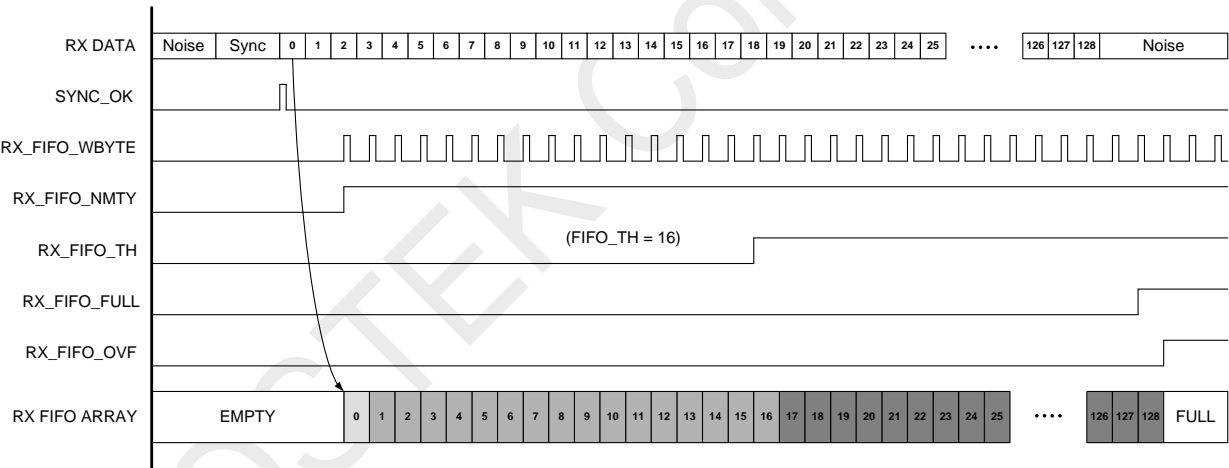


图 2. CMT2310A RX FIFO 中断时序示意图

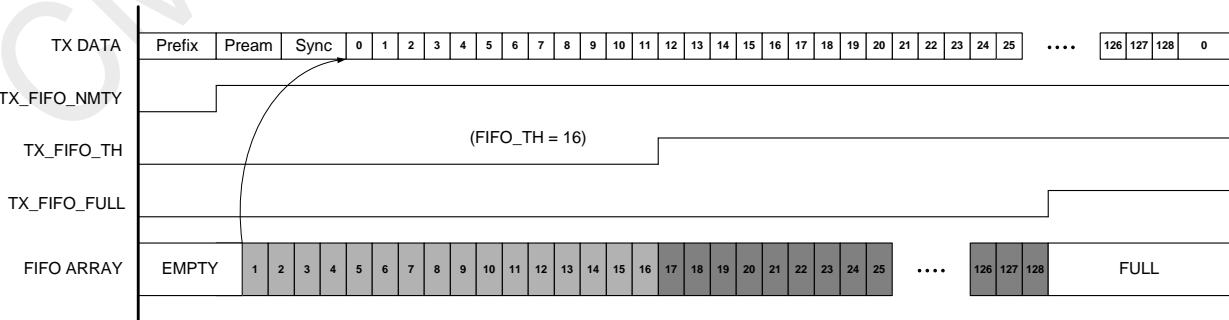


图 3. CMT2310A TX FIFO 中断时序示意图

1.4 FIFO 的应用场景

FIFO 配置好后即可开始使用。下面介绍几种经典的应用场景。要完成 TX 和 RX 的整个流程还需配置和控制其他部分，相关信息将在下文陆续介绍，此处仅介绍与 FIFO 相关的内容。

1.4.1 应用场景一：在 RX 下接收数据

RX FIFO 使用方式比较直接，每次进入 RX 之前先清零，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，接收完之无需其他处理，下次接收之前发送 RX_FIFO_CLR 清零即可。

1. 检测 RX_FIFO_FULL 中断，一旦有效表示 FIFO 已经被填满，即可以开始读取。该选择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，即可以开始读取。该选择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，一旦有效立即进行读取，直到该中断无效，接着再次等待该中断有效时再读取，这样可以实现边收边读。该选择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，一旦有效立即进行读取，这样可以实现写一个 byte，读一个 byte 的规律性操作，前提是 SPI 速度比接受数据的速度要快。

另外，可以通过设置 TX_PKT_NUM <7:0> 来定义每次发送多少个相同的数据包，如果发送的数量多于 1 个数据包，就需要将 FIFO_AUTO_RES_TX_EN 设置为 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。

如果要想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，耗时最多为 534 us。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，可以按照 10 个 SCL 时钟周期算，那么每个周期就是 53.4 us，换算过来 SCL 的时钟频率即为 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入 TX FIFO，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_TEST_EN 设置为 1，进入 TX FIFO 的回读模式。
2. 读取数据，比较确认正确性。
3. 将 FIFO_TX_TEST_EN 设置为 0，退出 TX FIFO 回读模式。
4. 将 TX_FIFO_CLR 设置为 1，清零 FIFO。
5. 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 READY，将 FIFO_RESTORE_TX 设置为 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，周而复始。如果 FIFO_AUTO_RES_TX_EN 已经设置为 1，则无需手动设置 FIFO_RESTORE_TX，下次直接进入 TX 发射即可。

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 256-byte，用户要陆续发射四个数据包，每个包为 64-byte。那么，用户可以在 READY 先把这 4 个数据包按顺序填入 FIFO，确保 FIFO 满；然后，将数据长度设置为 64（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_TX_EN 设置为 0。然后进入 TX，芯片会发送第一个 64-byte 的数据包，完成后退出到 READY，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 无需额外操作，只负责探测中断，切换状态即可。

2. 包格式介绍

CMT2310A 采用了 TX 和 RX 统一配置，较为典型和较灵活的包格式，其结构图如下。

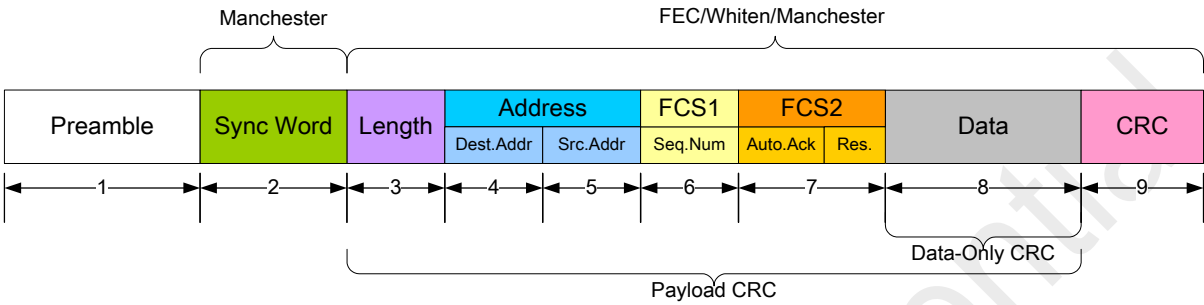


图 4. packet 结构框图

包格式包含了九个可选部分，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”。下面将结合相应寄存器，解释如何配置各个部分。

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 的界面和参数如下图所示。

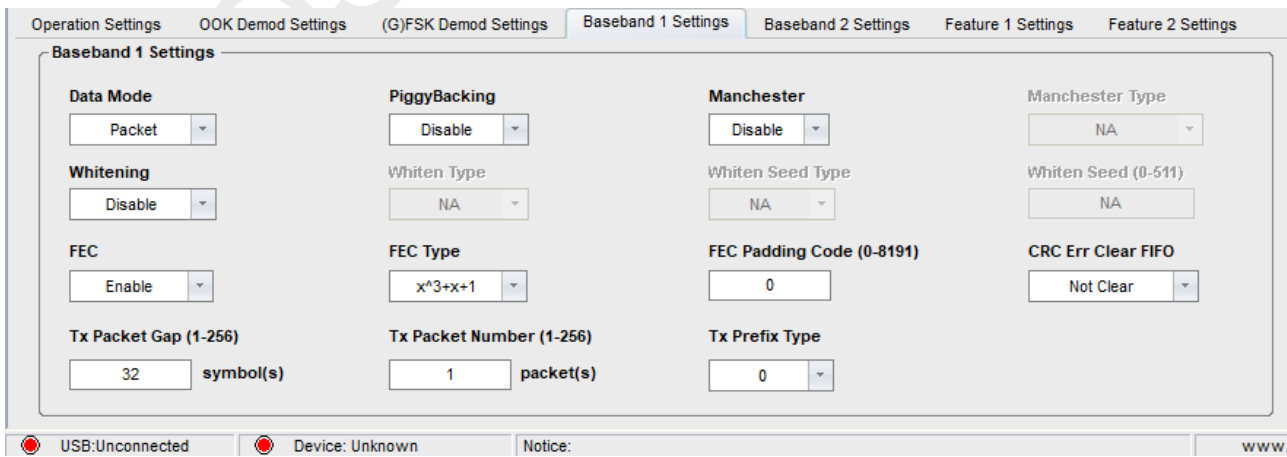


图 5. 数据模式的 RFPDK 界面

表 5. 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_DATA_MODE

表 6. 位于配置区的寄存器

寄存器名	位数	R/W	比特名	功能说明
CTL_REG_40 (0x28)	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
TX_DR_REG_02 (0x97)	7	RW	TX_DATA_MODE	选择 TX 的数据来源位置: 0: TX 数据从 GPIO 直接输入 1: TX 数据从 TX FIFO 获取

不同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; TX 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 的界面和参数如下图。

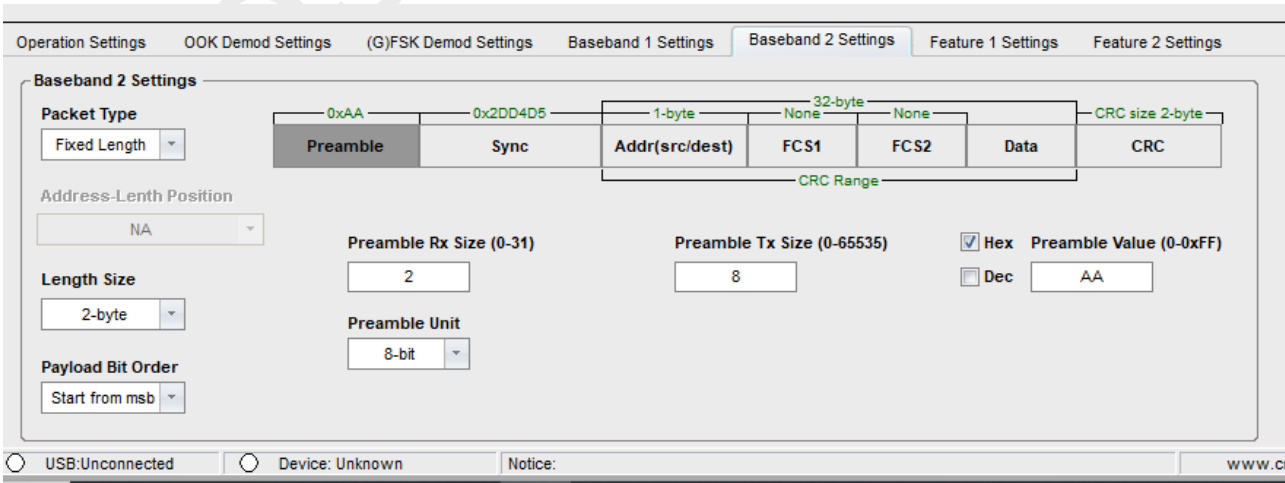


图 6. Preamble 的 RFPDK 界面

表 7. 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_UNIT
Preamble Value	PREAM_VALUE<7:0>

表 8. 位于配置区的寄存器

寄存器名	位数	R/W	比特名	功能说明
CTL_REG_40 (0x28)	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, 以此类推。
	2	RW	PREAM_LEN_UNIT	Preamble 的长度单位, TX 和 RX 共用: 0: 单位为 8bits 1: 单位为 4 bits
CTL_REG_41 (0x29)	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, 以此类推。
CTL_REG_42 (0x2A)	7:0	RW	TX_PREAM_SIZE<15:8>	
CTL_REG_43 (0x2B)	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此中断。

2.3 Sync Word 配置

对应的 RFPDK 的界面和参数如下图。

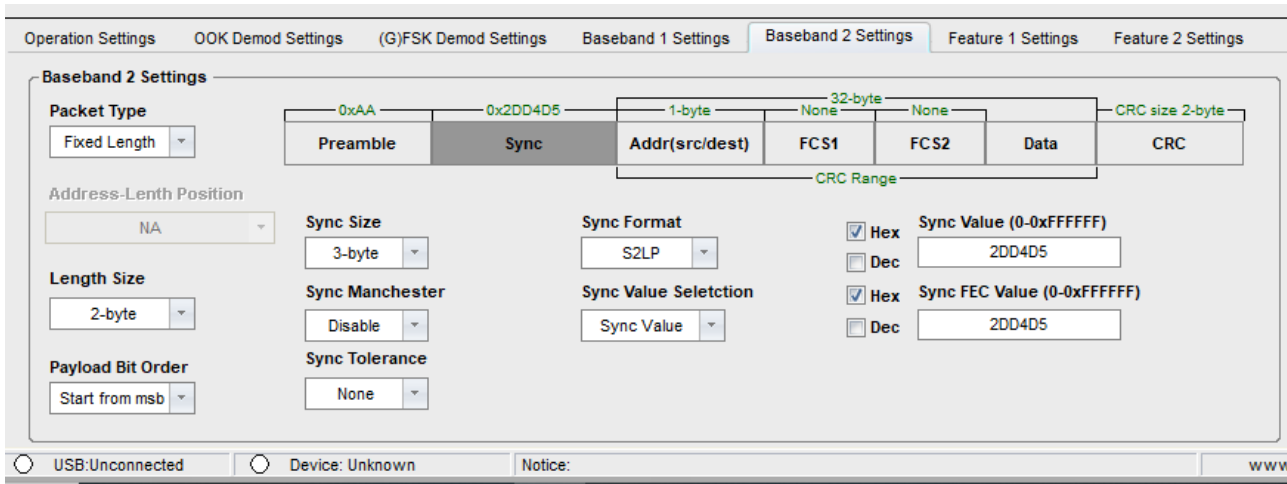


图 7. Sync Word 的 RFPDK 界面

表 9. 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 Format	SYNC_MODE_SEL
Sync Value Selection	SYNC_VALUE_SEL
Sync Value	SYNC_VALUE<63:0>
Sync FEC Value	SYNC_FEC_VALUE<63:0>
Sync Tolerance	SYNC_TOL<2:0>
Sync Manchester	SYNC_MAN_EN

表 10. 位于配置区的寄存器

寄存器名	位数	R/W	比特名	功能说明
CTL_REG_44 (0x2C)	7	RW	SYNC_MODE_SEL	0: S2LP 模式 1: 802.15.4 模式
	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
	0	RW	SYNC_MAN_EN	Sync Word 的曼切斯特编解码使能: 0: 不使能 1: 使能
CTL_REG_45 (0x2D)	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, 详见下表。
CTL_REG_46 (0x2E)	7:0	RW	SYNC_VALUE<15:8>	
CTL_REG_47 (0x2F)	7:0	RW	SYNC_VALUE<23:16>	
CTL_REG_48 (0x30)	7:0	RW	SYNC_VALUE<31:24>	
CTL_REG_49 (0x31)	7:0	RW	SYNC_VALUE<39:32>	
CTL_REG_50 (0x32)	7:0	RW	SYNC_VALUE<47:40>	
CTL_REG_51 (0x33)	7:0	RW	SYNC_VALUE<55:48>	
CTL_REG_52 (0x34)	7:0	RW	SYNC_VALUE<63:56>	

寄存器名	位数	R/W	比特名	功能说明
CTL_REG_53 (0x35)	7:0	RW	SYNC_FEC_VALUE<7:0>	Sync FEC Word 的值, TX 在 FEC_ON 时填充此值到 SYNC 域, RX 在检测到此值时开启 FEC。根据不同的 SYNC_SIZE 设置来填入不同的寄存器, 详见下表。
CTL_REG_54 (0x36)	7:0	RW	SYNC_FEC_VALUE<15:8>	
CTL_REG_55 (0x37)	7:0	RW	SYNC_FEC_VALUE<23:16>	
CTL_REG_56 (0x38)	7:0	RW	SYNC_FEC_VALUE<31:24>	
CTL_REG_57 (0x39)	7:0	RW	SYNC_FEC_VALUE<39:32>	
CTL_REG_58 (0x32A)	7:0	RW	SYNC_FEC_VALUE<47:40>	
CTL_REG_59 (0x3B)	7:0	RW	SYNC_FEC_VALUE<55:48>	
CTL_REG_60 (0x3C)	7:0	RW	SYNC_FEC_VALUE<63:56>	
CTL_REG_64 (0x40)	7	RW	SYNC_VALUE_SEL	当 SYNC_MODE_SEL 为 0 时有效: 0: 选择 SYNC_VALUE 1: 选择 SYNC_FEC_VALUE

	SYNC_VALUE/SYNC_FEC_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 的界面和参数如下。

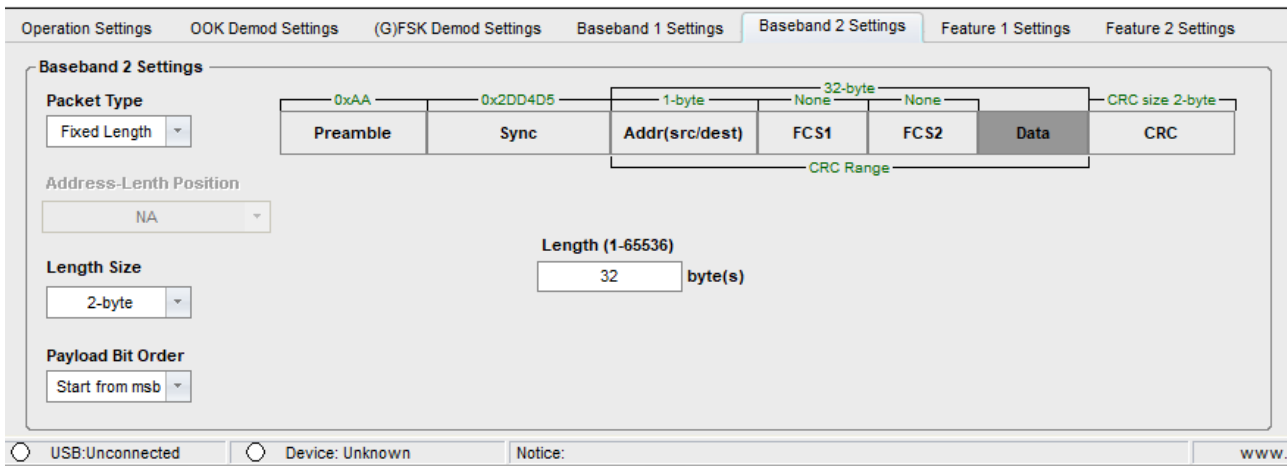


图 8. 数据包的 RFPDK 界面

表 11. 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
Length, 根据各个参数综合计算出来，具体方法下文有介绍	PAYLOAD_LEN<15:0>
Address-Length Position	ADDR_LENGTH_POS_SEL
Payload Bit Order	PAYLOAD_BIT_ORDER
Length Size	LENGTH_SIZE

表 12.位于配置区的寄存器：

寄存器名	位数	R/W	比特名	功能说明
CTL_REG_61 (0x3D)	7:0	RW	PAYLOAD_LENGTH<7:0>	16-bit Payload 长度的<7:0>位。
CTL_REG_62 (0x3E)	7:0	RW	PAYLOAD_LENGTH<15:8>	16-bit Payload 长度的<15:8>位。
CTL_REG_63 (0x3F)	5	RW	LENGTH_SIZE	0: 1 字节, 支持 0 - 255Bytes 的可变包长 1: 2 字节, 支持 0 - 65535Bytes 的可变包长
	2	RW	NODE_LENGTH_POS_SEL	在可变包中, Address 和 Length Byte 的位置关系 0: Address 在 length Byte 之前 1: Address 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

固定包长模式下, 无 Length 域, 收发过程通过 PAYLOAD_LENGTH 配置确定收发包长度; 可变包长模式下, Length 域的值根据 PAYLOAD_LENGTH 确定, Length=PAYLOAD_LENGTH, 表示 payload bytes 数, 长度信息含自身占用。同时, LENGTH_SIZE 为 0 时, Length 域的值取 PAYLOAD_LEN[7:0], LENGTH_SIZE 为 1 时, Length 域的值取 PAYLOAD_LEN[15:0]。

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, 那么用户是看不到这个过程的; 如果使用 CMOSTEK 产品与其他厂商产品对接, 则需要用户理解此过程并进行合适的配置。

PAYLOAD_BIT_ORDER = 0 表示无此操作。

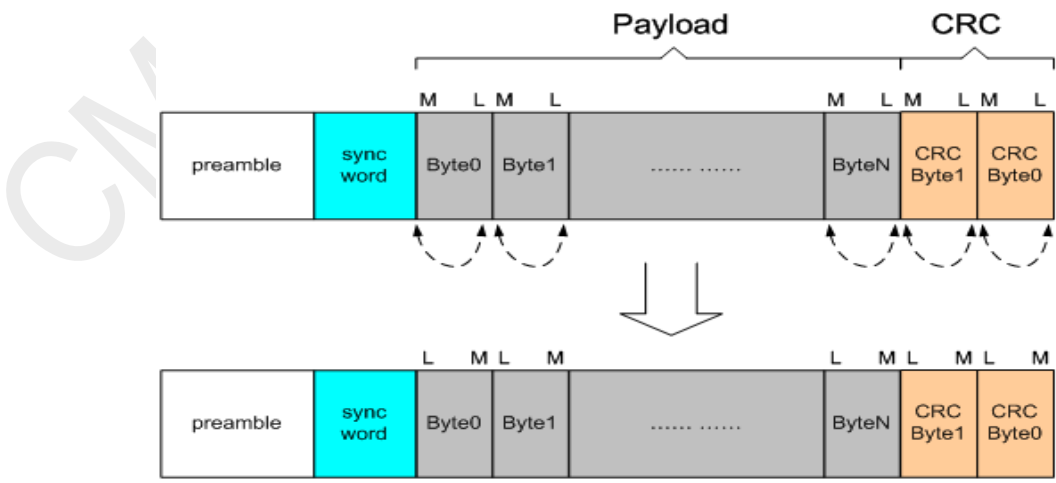


图 9. PAYLOAD_BIT_ORDER 操作

2.5 Address 配置

对应的 RFPDK 的界面和参数如下。

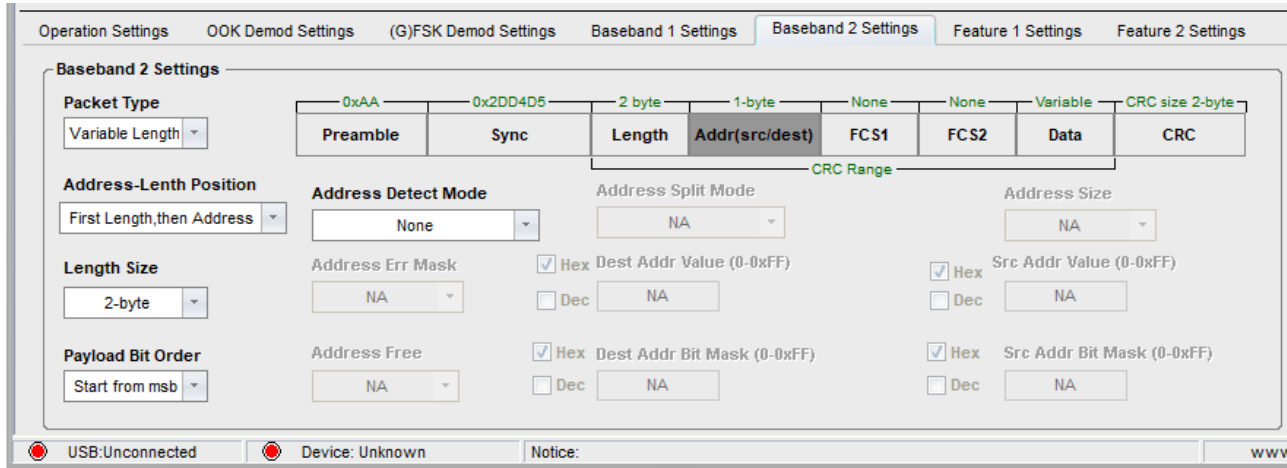


图 10. Address 的 RFPDK 界面

表 13. Address 的相关参数

寄存器比特 RFPDK 参数	寄存器比特
Address Detect Mode	ADDR_DET_MODE<1:0>
Address Spilt Mode	ADDR_SPILT_MODE
Address Size	ADDR_SIZE<1:0>
Address Free	ADDR_FREE_EN
Address Value	ADDR_VALUE<31:0>
Address Err Mask	ADDR_ERR_MASK
Address Bit Mask	ADDR_BITMASK<31:0>

表 14. 位于配置区的寄存器

寄存器名	位数	R/W	比特名	功能说明
CTL_REG_64 (0x40)	6	RW	ADDR_SPILT_MODE	地址分离模式配置。 0: 仅 DEST ADDR 域。这时 ADDR_VALUE 仅用于 DEST ADDR 1: DEST ADDR 域 + SRC ADDR 域。这时 ADDR_VALUE 高 16 比特用于配置 DEST ADDR, 低 16 比特用于配置 SRC ADDR。
	5	RW	ADDR_FREE_EN	在 RX 模式下, 让 ADDRESS 检测电路独立出来的使能位, 即接收期间一直检测地址是否匹配。

寄存器名	位数	R/W	比特名	功能说明
				0: 不使能 1: 使能
	4	RW	ADDR_ERR_MASK	ADDRESS 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
	3:2	RW	ADDR_SIZE<1:0>	ADDRESS 的长度。 0: 1 byte 1: 2 bytes 2: 3 bytes 3: 4 bytes 注: ADDR_SPLIT_MODE 为 1 时, 表示 DEST ADDR 域+SRC ADDR 域各占 1 或 2bytes
	1:0	RW	ADDR_DET_MODE<1:0>	ADDRESS 的检测模式。 0: 不检测。 1: TX 模式发送 ADDR_VALUE 的内容; RX 模式仅识别 ADDR_VALUE 的内容。 2: TX 模式发送 ADDR_VALUE 的内容; RX 模式识别 ADDR_VALUE 的内容和全 0。 3: TX 模式发送 ADDR_VALUE 的内容; RX 模式识别 ADDR_VALUE 的内容, 全 0 和全 1。
CTL_REG_65 (0x41)	7:0	RW	ADDR_VALUE<7:0>	32 bit ADDRESS 的值。 ADDR_SPLIT_MODE 为 0 时, 仅用于 DEST ADDR 域 ADDR_SPLIT_MODE 为 1 时, 高 16 比特用于配置 DEST ADDR; 低 16 比特用于配置 SRC ADDR。
CTL_REG_66 (0x42)	7:0	RW	ADDR_VALUE<15:8>	
CTL_REG_67 (0x43)	7:0	RW	ADDR_VALUE<23:16>	
CTL_REG_68 (0x44)	7:0	RW	ADDR_VALUE<31:24>	
CTL_REG_69 (0x45)	7:0	RW	ADDR_BITMASK<7:0>	地址比特掩码。设置的掩码比特, 对应的 ADDR 不做比较。 ADDR_SPLIT_MODE 为 0 时, 仅用于 DEST ADDR 域 ADDR_SPLIT_MODE 为 1 时, 高 16 比特用于配置 DEST ADDR; 低 16 比特用于配置 SRC ADDR。
CTL_REG_70 (0x46)	7:0	RW	ADDR_BITMASK<15:8>	
CTL_REG_71 (0x47)	7:0	RW	ADDR_BITMASK<23:16>	
CTL_REG_72 (0x48)	7:0	RW	ADDR_BITMASK<31:24>	

	ADDR_VALUE			
ADDR_SIZE	<31:24>	<23:16>	<15:8>	<7:0>
0	√			
1	√	√		
2	√	√	√	
3	√	√	√	√

表中打勾的位置表示要填入的寄存器，要根据 ADDR_SPLIT_MODE 进行选择。

举例说明，如果 ADDR_SPLIT_MODE 设置为 0，ADDR_SIZE 设置为 1，即只有 DEST ADDR 域，且长度为 2 个 byte，值为 0x5678，那么用户就将这个值填入 ADDR_VALUE<31:24>和 ADDR_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>。

如果 ADDR_SPLIT_MODE 设置为 1，ADDR_SIZE 设置为 1，则有 DEST ADDR 域 + SRC ADDR 域，且长度各为 2 个 byte，DEST ADDR 值为 0x5678，SRC ADDR 值为 0x1234，那么用户要将 DEST ADDR 值填入 ADDR_VALUE<31:24>和 ADDR_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 ADDR_VALUE<31:24>，0x78 填入 ADDR_VALUE<23:16>；将 SRC ADDR 值填入 ADDR_VALUE<15:8>和 ADDR_VALUE<7:0>两个寄存器，msb 对应的是第 15 位，lsb 对应的是第 0 位，即将 0x12 填入 ADDR_VALUE<15:8>，0x34 填入 ADDR_VALUE<7:0>。

2.6 FCS1 配置

对应的 RFPDK 的界面和参数如下。

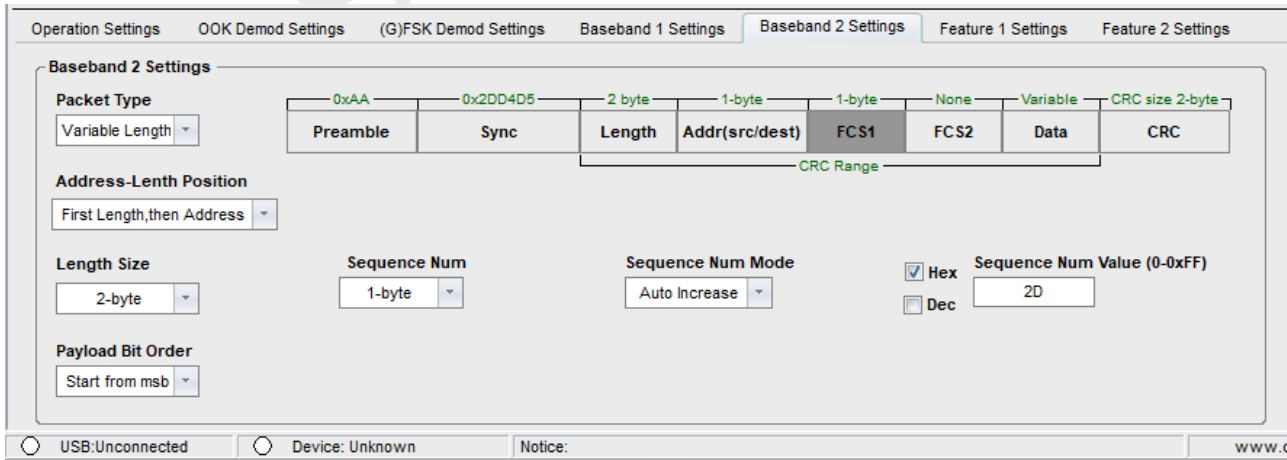


图 11. FCS1 的 RFPDK 界面

表 15. FCS1 的相关参数

寄存器比特 RFPDK 参数	寄存器比特
Sequence Num 为 None 时, SEQNUM_EN = 0, 否则 SEQNUM_EN = 1	SEQNUM_EN
Sequence Num != None	SEQNUM_SIZE
Sequence Num Mode	SEQNUM_AUTO_INC
Sequence Num Value	SEQNUM_TX_IN<15:0>
Sequence Num Match Detect	SEQNUM_MATCH_EN

表 16. 位于配置区的寄存器:

寄存器名	位数	R/W	比特名	功能说明
CTL_REG_84 (0x54)	5	RW	SEQNUM_MATCH_EN	自动应答开启时, Master 端判断 Slave 端是否成功接收包使能。 0: 不判断 1: 使能判断
	4	RW	SEQNUM_SIZE	Sequence Num 域的大小。 0: 1 字节 1: 2 字节
	3	RW	SEQNUM_AUTO_INC	TX 侧 Sequence Num 是否自动递增。 0: 不累加 1: 每包自动加 1
	2	RW	SEQNUM_EN	0: 无 Sequence Num 域 (即 FCS1 域) 1: 使能 Sequence Num 域 (即 FCS1 域)
CTL_REG_87 (0x57)	7:0	RW	SEQNUM_TX_IN<7:0>	TX 侧 Sequence Num 初始值。 SEQNUM_SIZE 为 0 时, SEQNUM_TX_IN<15:8>
CTL_REG_88 (0x58)	7:0	RW	SEQNUM_TX_IN<15:8>	有效 SEQNUM_SIZE 为 1 时, SEQNUM_TX_IN<15:0> 有效

对于发射而言, Sequence Num 作为包结构内容一并发出; 对于接收而言, 自动填入到映射的寄存器中, 用户可以直接读取寄存器值。

对于自动应答包, 下文有具体介绍。

2.7 FCS2 配置

对应的 RFPDK 的界面和参数如下。

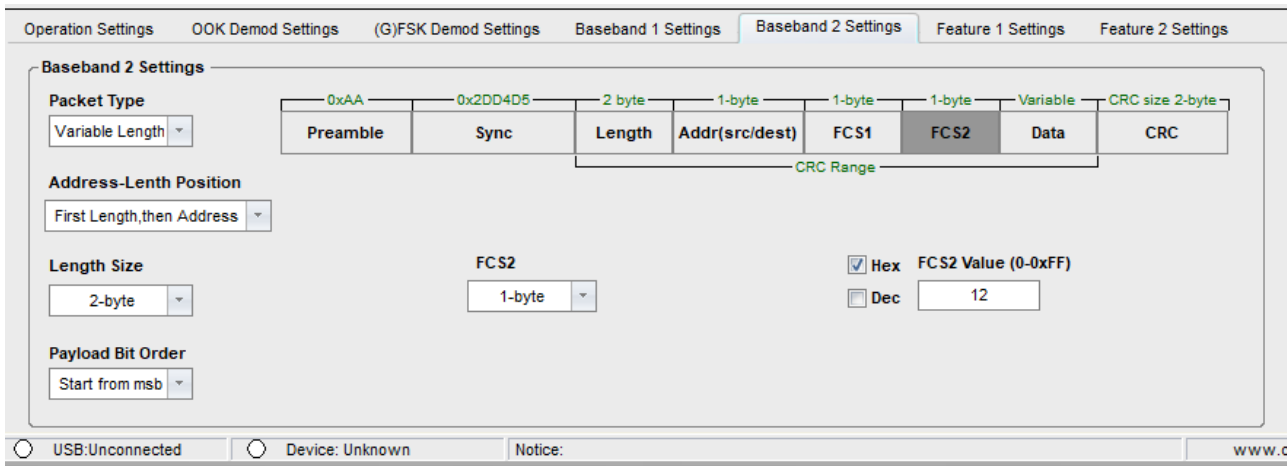


图 12. FCS2 的 RFPDK 界面

表 17. FCS2 的相关参数

寄存器比特 RFPDK 参数	寄存器比特
FCS2	FCS2_EN
FCS2 Value	FCS2_TX_IN

表 18.位于配置区的寄存器

寄存器名	位数	R/W	比特名	功能说明
CTL_REG_84 (0x54)	6	RW	FCS2_EN	0: 无 FCS2 1: 使能 FCS2 域
CTL_REG_91 (0x5B)	7:0	RW	FCS2_TX_IN <7:0>	TX 侧 FCS2 域发送值。 这里定义 FCS2_TX_IN[7]为发送端 Master 要求接收端 Slave 返回 ACK 应答包的使能位(Auto.Ack),即如果接收端 Slave 收到的 FCS2 域 bit7 为 1, 则要发送应答包给 Master。 FCS2_TX_IN[6:0]为保留位。

这里简要说明一下自动应答，即 AUTO ACK 功能。接收端 Slave 具有自动应答功能，在收到一个数据包后，如果该包要求回复 ACK 包（即接收到的 FCS2_TX_IN[7]为 1），则芯片直接自动切换到发射模

式并发回 ACK 包。ACK 的包格式如下：

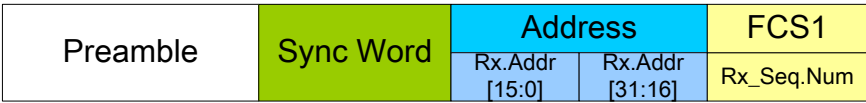


图 13. ACK 的包格式

这个数据包包含 Preamble 和 Sync ID，如果接收到的包结构包含 Address 和 FCS1，则 ACK 包会将这两部分发送出去，其中，Address 域将接收到的 DEST ADDR 和 SRC ADDR 位置交换，FCS1 域将接收到的 Sequence Num 直接返回。

对于 Master 端，如果要求 Slave 发送应答包，需要将 FCS2_TX_IN[7]设置 1，发送完当前包后自动进入接收状态，接收 Slave 发回的 ACK 包，如果 SEQNUM_MATCH_EN 设置为 1，则比对收到的 Sequence Num 与本地发出的 Sequence Num 是否一致，一致则产生 SEQNUM_OK，用户可以回读该信号。

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
CTL_REG_96 (0x60)	2	RW	TX_ACK_EN	发送端 Master 自动应答使能： 0：不使能 1：使能
CTL_REG_97 (0x61)	2	RW	RX_ACK_EN	接收端 Slave 自动应答使能： 0：不使能 1：使能

TX_ACK_EN 和 RX_ACK_EN 是指 Master 和 Slave 端具有自动应答功能，具体是否发送和接收 ACK 包，取决于 FCS2 域的 bit7。

下文将介绍，在不同的 PKT_TYPE 下，如何设置 PAYLOAD_LENGTH 和 ADDR_SIZE，如何确定 Payload 里面的 DATA 的长度，如何解释 Length Byte 的含义。用户可根据需要，在下列表格中查找需要的理解的部分。

固定包格式：

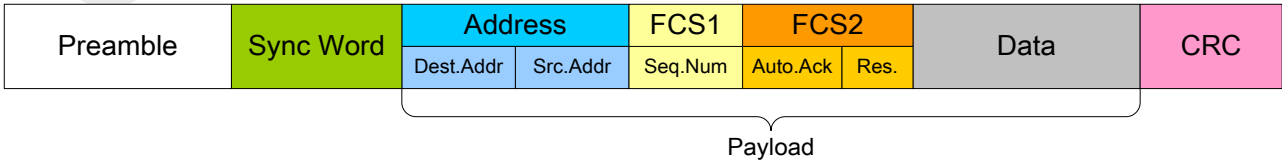


图 14. 固定长度包格式

TX 模式	RX 模式
<p>Payload 结构:</p> <p>Address + FCS1 + FCS2 + Data, 其中 Address / FCS1 / FCS2 可以不存在, TX 和 RX 配置完全相同。</p> <p>举例 1:</p> <p>PAYLOAD_LENGTH<15:0> = 17</p> <p>ADDR_DET_MODE<1:0> != 0</p> <p>ADDR_SIZE<1:0> = 2</p> <p>ADDR_SPILT_MODE = 0</p> <p>FCS1_EN = 1</p> <p>SEQNUM_SIZE = 1</p> <p>FCS2_EN = 1</p> <p>此处, Payload 包含 Address + FCS1 + FCS2 + Data 共 4 部分, Payload 的总长度是 17 + 1 = 18, Address 的长度是 2 (Dest.Addr) + 1 = 3, Sequence Num 的长度是 1 + 1 = 2, FCS2 长度固定为 1, 所以, Data 的长度是 18 - 3 - 2 - 1 = 12 个 byte。</p> <p>假如 ADDR_SPILT_MODE = 1, 即 Address 的长度是 0 (Dest.Addr) + 1 + 0 (Scr.Addr) + 1 = 2,那么 Data 的长度是 18 - 2 - 2 - 1 = 13 个 byte。</p> <p>举例 2:</p> <p>PAYLOAD LENG<15:0> = 17</p> <p>ADDR_DET_MODE<1:0> = 0</p> <p>ADDR_SIZE<1:0> = 2</p> <p>ADDR_SPILT_MODE = 0</p> <p>FCS1_EN = 0</p> <p>SEQNUM_SIZE = 1</p> <p>FCS2_EN = 0</p> <p>此处, Address + FCS1 + FCS2 不存在, 那么 Data 的长度就是 18。</p>	

可变包格式:

情况一: Address 不存在

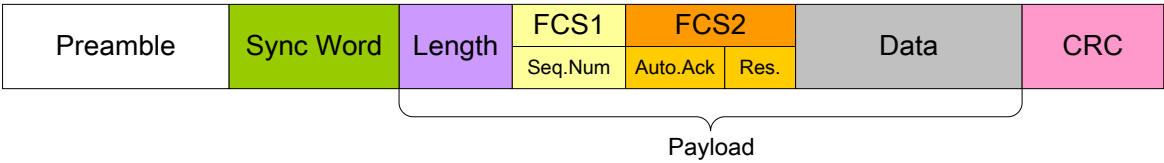


图 15. 可变长度包格式, Address 不存在

TX 模式	RX 模式
<p>Payload 结构:</p> <p>Length Byte + FCS1 + FCS2 + Data</p> <p>Payload 长度:</p> <p>$1 + \text{PAYLOAD_LENGTH}<15:0>$, 其中 $\text{PAYLOAD_LENGTH}<15:0>$ 的内容就等于 Length Byte 的内容, 指的是后面跟随的 Payload 的长度。</p> <p>如果 $\text{LENGTH_SIZE} = 0$, 即 $\text{PAYLOAD_LENGTH}<15:8>$ 有效, 最大值为 255, Length Byte 本身是 1 个 byte 的长度。</p> <p>如果 $\text{LENGTH_SIZE} = 1$, 即 $\text{PAYLOAD_LENGTH}<15:0>$ 有效, 最大值为 65535, Length Byte 本身是 2 个 byte 的长度。</p> <p>举例:</p> <p>$\text{PAYLOAD_LENGTH}<15:0> = 17$</p> <p>$\text{LENGTH_SIZE} = 0$</p> <p>$\text{FCS1_EN} = 1$</p> <p>$\text{SEQNUM_SIZE} = 1$</p> <p>$\text{FCS2_EN} = 1$</p> <p>此处, Payload 包含 Length + FCS1 + FCS2 + Data 共 4 部分, Payload 的总长度是 $17 + 1 = 18$, Length 的长度是 1, Sequence Num 的长度是 $1 + 1 = 2$, FCS2 长度固定为 1, 所以, Data 的长度是 $18 - 1 - 2 - 1 = 14$ 个 byte。如果 FCS1 + FCS2 不存在, 那么 Data 的长度就是 $18 - 1 = 17$ 个 byte。</p> <p>如果 $\text{LENGTH_SIZE} = 1$, 即 Length 的长度是 2, 所以, Data 的长度是 $18 - 2 - 2 - 1 = 13$ 个 byte。如果 FCS1 + FCS2 不存在, 那么 Data 的长度就是 $18 - 2 = 16$ 个 byte。</p>	<p>Payload 结构:</p> <p>Length Byte + FCS1 + FCS2 + Data</p> <p>Length Byte 内容含义:</p> <p>表示随后 Payload 的长度, 与 TX 配置对应。</p>

情况二: Address 存在, 且 $\text{NODE_LENGTH_POS_SEL} = 0$ (Address 在 Length Byte 之前)

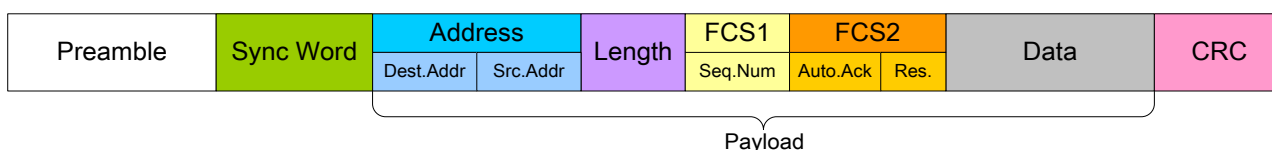


图 16. 可变长度包格式, Address 存在, Address 在 Length Byte 之前

TX 模式	RX 模式
<p>Payload 结构:</p> <p>Address + Length Byte + FCS1 + FCS2 + Data</p> <p>Payload 长度:</p> <p>(ADDR_SIZE<1:0> + 1) + 1 + PAYLOAD_LENGTH<15:0>, 其中 Address 的长度根据 ADDR_SPILT_MODE + ADDR_SIZE<1:0> 确定 (参见前文), PAYLOAD_LENGTH<15:0>的内容就等于 Length Byte 的内容, 指的是后面跟随的除 Address 外的 Payload 的长度。</p> <p>如果 LENGTH_SIZE = 0, 即 PAYLOAD_LENGTH<15:8>有效, 最大值为 255, Length Byte 本身是 1 个 byte 的长度。</p> <p>如果 LENGTH_SIZE = 1, 即 PAYLOAD_LENGTH<15:0>有效, 最大值为 65535, Length Byte 本身是 2 个 byte 的长度。</p> <p>举例:</p> <p>PAYLOAD_LENGTH<15:0> = 17</p> <p>ADDR_DET_MODE<1:0> != 0</p> <p>ADDR_SIZE<1:0> = 2</p> <p>ADDR_SPILT_MODE = 0</p> <p>LENGTH_SIZE = 0</p> <p>FCS1_EN = 1</p> <p>SEQNUM_SIZE = 1</p> <p>FCS2_EN = 1</p> <p>此处, Payload 包含 Address + (Length + FCS1 + FCS2 + Data) 共 5 部分, 其中, Address 的长度是 2 (Dest.Addr) + 1 = 3, Length 的长度是 1, Sequence Num 的长度是 1 + 1 = 2, FCS2 长度固定为 1, 那么 Payload 的总长度是 3 + (17 + 1) = 21, 所以, Data 的长度是 18 - 1 - 2 - 1 = 14 个 byte。如果 FCS1 + FCS2 不存在, 那么 Data 的长度就是 18 - 1 = 17 个 byte。</p> <p>如果 LENGTH_SIZE = 1, 即 Length 的长度是 2, 所以, Data 的长度是 18 - 2 - 2 - 1 = 13 个 byte。如果 FCS1 + FCS2 不存在, 那么 Data 的长度就是 18 - 2 = 16 个 byte。</p>	<p>Payload 结构:</p> <p>Address + Length Byte + FCS1 + FCS2 + Data</p> <p>Length Byte 内容含义:</p> <p>表示随后除 Address 外的 Payload 的长度, 与 TX 配置对应。</p>

情况三：Address 存在，且 NODE_LENGTH_POS_SEL = 1（Address 在 Length Byte 之后）

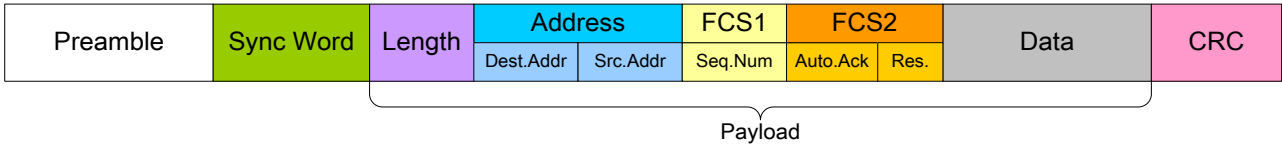


图 17. 可变长度包格式，Address 存在，Address 在 Length Byte 之后

TX 模式	RX 模式
<p>Payload 结构: Length Byte + Address +FCS1 + FCS2 + Data</p> <p>Payload 长度: 1 + PAYLOAD_LENGTH<15:0>, 其中 PAYLOAD_LENGTH<15:0>的内容就等于 Length Byte 的内容, 指的是后面跟随的 Payload 的长度。 如果 LENGTH_SIZE = 0, 即 PAYLOAD_LENGTH<15:8>有效, 最大值为 255, Length Byte 本身是 1 个 byte 的长度。 如果 LENGTH_SIZE = 1, 即 PAYLOAD_LENGTH<15:0>有效, 最大值为 65535, Length Byte 本身是 2 个 byte 的长度。</p> <p>举例: PAYLOAD_LENGTH<15:0> = 17 ADDR_DET_MODE<1:0> != 0 ADDR_SIZE<1:0> = 2 ADDR_SPILT_MODE = 0 LENGTH_SIZE = 0 FCS1_EN = 1 SEQNUM_SIZE = 1 FCS2_EN = 1 此处, Payload 包含 Length + Address + FCS1 + FCS2 + Data 共 5 部分, Payload 的总长度是 17 + 1 = 18, Length 的长度是 1, Address 的长度是 2 (Dest.Addr) + 1 = 3, Sequence Num 的长度是 1 + 1 = 2, FCS2 长度固定为 1, 所以, Data 的长度是 18 - 1 - 3 - 2 - 1 = 11 个 byte。如果 FCS1 + FCS2 不存在, 那么 Data 的长度就是 18 - 1 - 3 = 14 个 byte。 如果 LENGTH_SIZE = 1, 即 Length 的长度是 2, 所以, Data 的长度是 18 - 2 - 3 - 2 - 1 = 10 个 byte。如果 FCS1 + FCS2 不存在, 那么 Data 的长度就是 18 - 2 - 3 = 13 个 byte。</p>	<p>Payload 结构: Length Byte + Address +FCS1 + FCS2 + Data</p> <p>Length Byte 内容含义: 表示随后 Payload 的长度, 与 TX 配置对应。</p>

2.8 CRC 配置

对应的 RFPDK 的界面和参数如下。

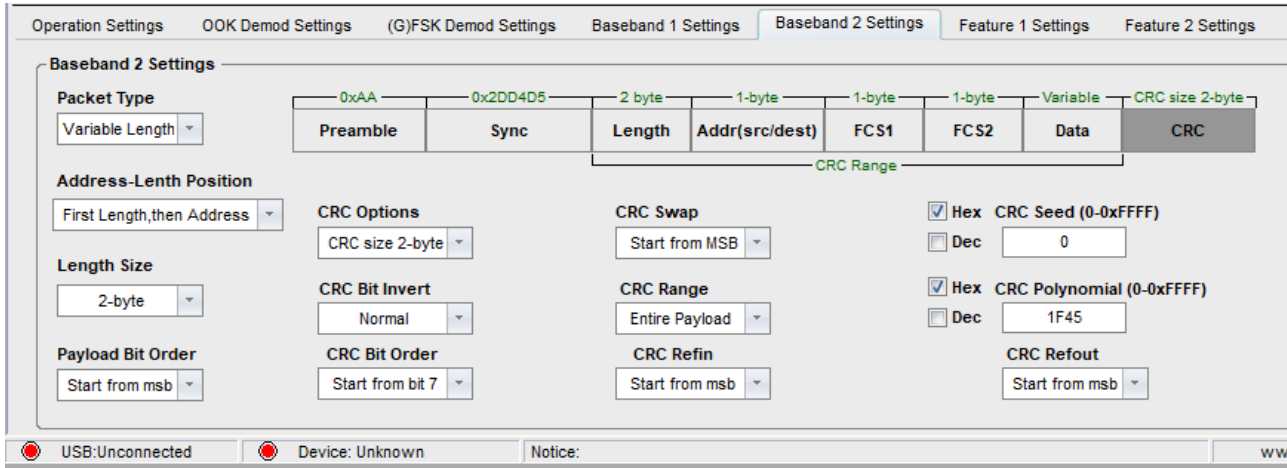


图 18. CRC 的 RFPDK 界面

表 20. CRC 的相关参数

寄存器比特 RFPDK 参数	寄存器比特
Crc Options != None	CRC_SIZE<1:0>
Crc Options 为 None 时，CRC_EN = 0，否则 CRC_EN =1	CRC_EN
Crc Seed	CRC_SEED<31:0>
Crc Polynomial	CRC_POLY<31:0>
Crc Range	CRC_RANGE
Crc Byte Swap	CRC_BYTE_SWAP
Crc Bit Invert	CRC_BIT_INV
Crc Bit Order of Byte	CRC_ORDER
Crc Refin	CRC_REFIN
Crc Refout	CRC_REFOUT

表 1.位于配置区的寄存器:

寄存器名	位数	R/W	比特名	功能说明
CTL_REG_73 (0x49)	7:6	RW	CRC_SIZE<1:0>	CRC 域校验码字节数。 0: 1 字节 1: 2 字节 2: 3 字节 3: 4 字节
	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	RW	CRC_REFIN	CRC 计算时输入字节的比特顺序反转: 0: 从 MSB 到 LSB 1: 从 LSB 到 MSB
	1	RW	CRC_ORDER	CRC 字节内大小端顺序配置: 0: 先收发高 bit 1: 先收发低 bit
	0	RW	CRC_EN	CRC 使能 0: 不使能 1: 使能
CTL_REG_74 (0x4A)	7:0	RW	CRC_SEED<7:0>	CRC 多项式的初始值
CTL_REG_75 (0x4B)	7:0	RW	CRC_SEED<15:8>	
CTL_REG_76 (0x4C)	7:0	RW	CRC_SEED<23:16>	
CTL_REG_77 (0x4D)	7:0	RW	CRC_SEED<31:24>	
CTL_REG_78 (0x4E)	7:0	RW	CRC_POLY<7:0>	CRC 计算的多项式, 支持 32 位以内的任意多项式配置。
CTL_REG_79 (0x4F)	7:0	RW	CRC_POLY<15:8>	
CTL_REG_80 (0x50)	7:0	RW	CRC_POLY<23:16>	
CTL_REG_81 (0x51)	7:0	RW	CRC_POLY<31:24>	
CTL_REG_82 (0x52)	7	RW	CRC_REFOUT	CRC 计算时输出所有字节的比特顺序反转: 0: 从 MSB 到 LSB 1: 从 LSB 到 MSB

CRC_BIT_ORDER

此功能将整个 CRC 部分以 Byte 为单位，Byte 内的高低位顺序倒过来。

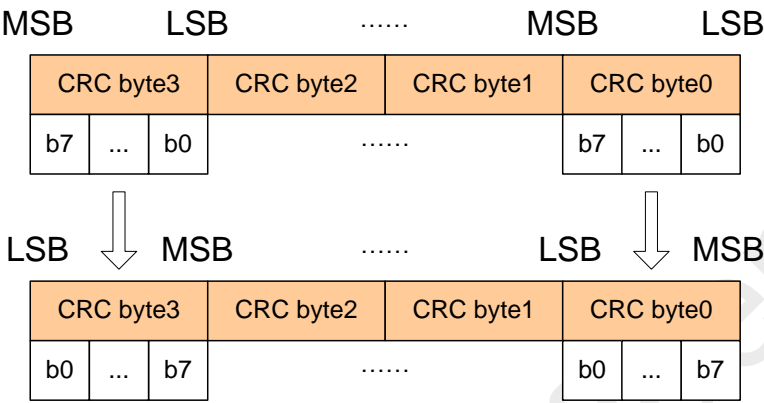


图 22. CRC_BIT_ORDER

2.9 编解码配置

对应的 RFPDK 的界面和参数如下。

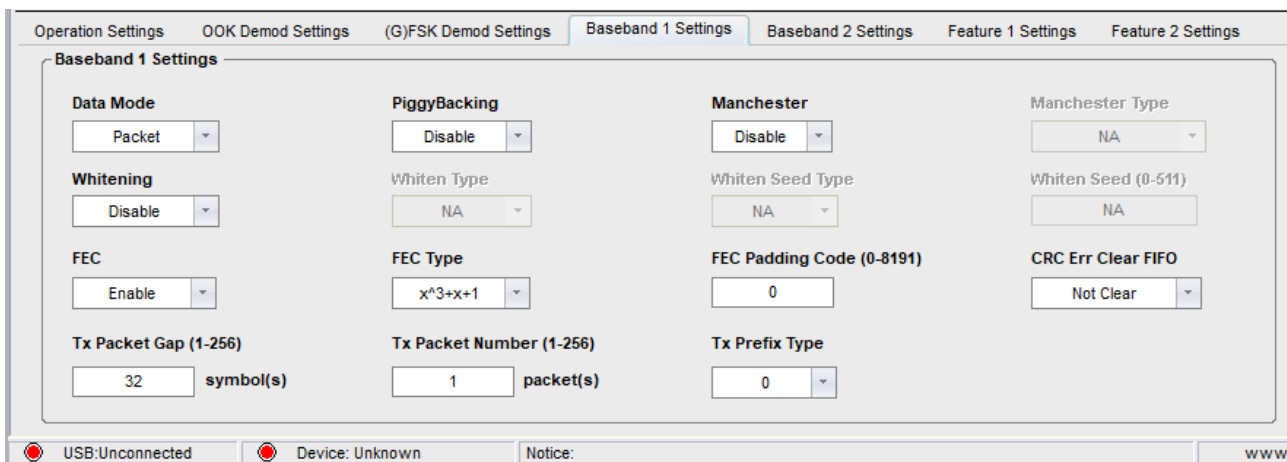


图 23. 编解码的 RFPDK 界面

表 21. 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
FEC	FEC_EN
FEC Type	FEC_RSC_NRNSC_SEL
FEC Padding Code	FEC_PAD_CODE
PaggyBacking	PAGGYBACKING_EN

表 22. 位于配置区的寄存器

寄存器名	位数	R/W	比特名	功能说明
CTL_REG_82 (0x52)	6	RW	WHITEN_SEED<8>	白化编解码多项式种子的第 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。
	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
CTL_REG_83 (0x53)	7:0	RW	WHITEN_SEED<7:0>	白化编解码多项式的种子的 7:0 位。
CTL_REG_93 (0x5D)	6:2	RW	FEC_PAD_CODE<12: 8>	FEC 的 Padding 码的高 12:8 位
	1	RW	FEC_RSC_NRNSC_SEL	FEC 多项式选择： 0：RSC 1：NRNSC
	0	RW	FEC_EN	FEC 编解码使能： 0：无 FEC 编解码 1：有 FEC 编解码
CTL_REG_94 (0x5E)	7:0	RW	FEC_PAD_CODE<7:0>	FEC 的 Padding 码的低 7:0 位
CTL_REG_63 (0x3F)	4	RW	PAGGYBACKING_EN	ACK 包是否搭载 payload： 0：无 payload 1：有 payload

下面解释 FEC 编解码的使用。

FEC 计算的范围是(Length + Address + FCS1 + FCS2 + Data + CRC) + Tail + Padding，其中，Tail + Padding 是计算 FEC 时插入的，Tail 保证 FEC 编码结束时状态归 0，Padding 保证了交织运算的完整性；当 Length + Address + FCS1 + FCS2 + Data + CRC 总长度为奇数时，取 FEC_PAD_CODE[4:0]；为偶数时，取 FEC_PAD_CODE[12:0]。

FEC 有如下 2 种多项式供选择：

RSC:

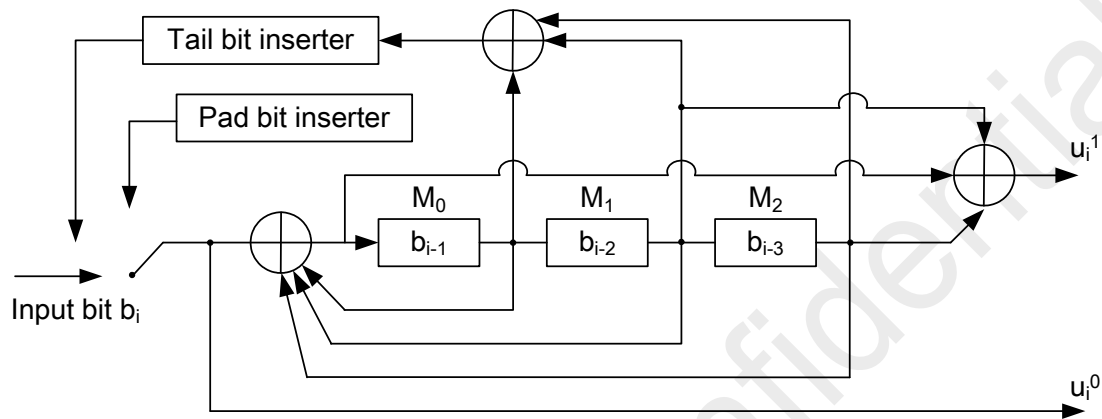


图 24. RSC 编码

其中， $u_i^1 = b_i \oplus (b_{i-1} \oplus b_{i-2} \oplus b_{i-3}) \oplus b_{i-2} \oplus b_{i-3}$

$u_i^0 = b_i$

NRNSC:

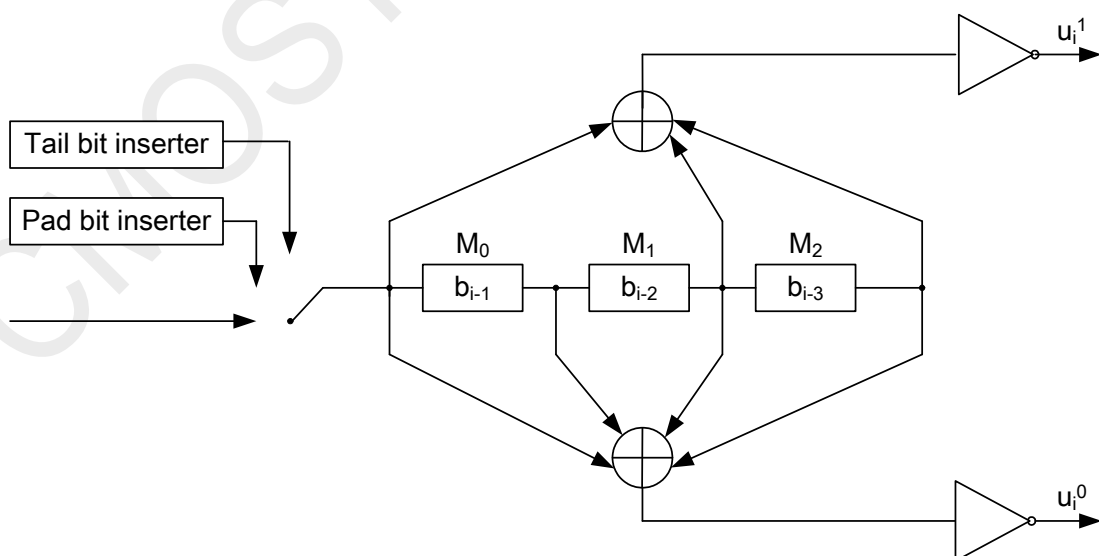


图 25. NRNSC 编码

其中， $u_i^1 = \sim(b_i \oplus b_{i-2} \oplus b_{i-3})$

$u_i^0 = \sim(b_i \oplus b_{i-1} \oplus b_{i-2} \oplus b_{i-3})$

RSC 和 NRNSC 编码后都会进行交织处理，可选择进行白化，但曼切斯特编码与 FEC 和白化编码不能同时存在，FEC 编码后接收端采用维特比解码。

2.10TX 数据包专用配置

对应的 RFPDK 的界面和参数如下。

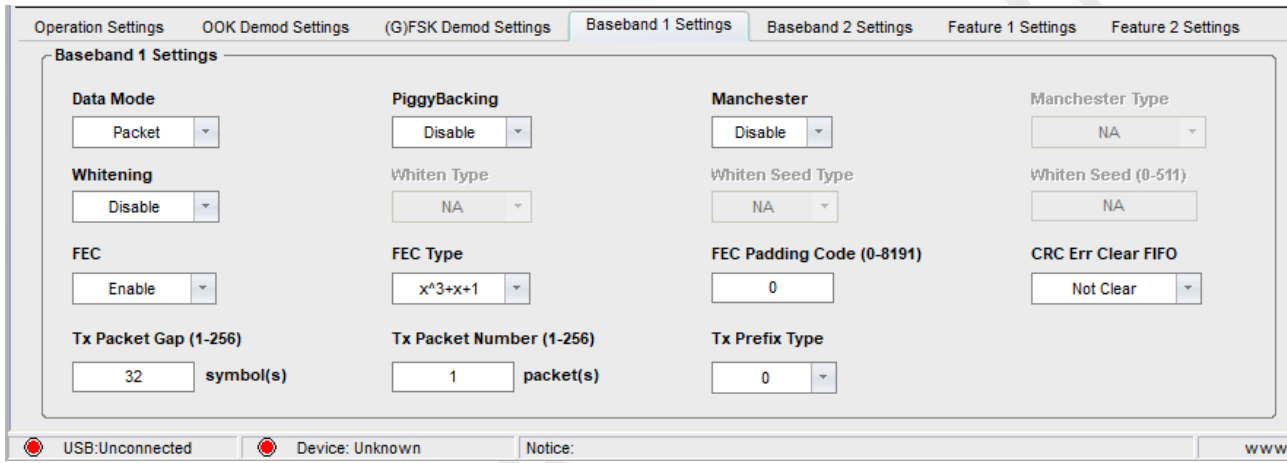


图 26. TX 数据包的 RFPDK 界面

表 23.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 24. 位于配置区的寄存器：

寄存器名	位数	R/W	比特名	功能说明
CTL_REG_84 (0x54)	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，就需要定义预发射的内容，可以定义为： 0: 发送 2'b00 1: 发送 2'b11 2: 发送 2'b01 3: 发送 2'b10
CTL_REG_85 (0x55)	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。
CTL_REG_86 (0x56)	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。

2.11 Direct 发射模式

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

表 25. Direct 发射模式配置寄存器

寄存器名	位数	R/W	比特名	功能说明
CTL_REG_40 (0x28)	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
TX_DR_REG_02 (0x97)	7	RW	TX_DATA_MODE	选择 TX 的数据来源位置： 0: TX 数据从 GPIO 直接输入 1: TX 数据从 TX FIFO 获取

表 26. Direct 发射模式控制寄存器

寄存器名	位数	R/W	比特名	功能说明
CTL_REG_4 (0x04)	6	RW	TX_DIN_EN	0: 屏蔽发射数据从 GPIO 输入 1: 使能发射数据从 GPIO 输入
CTL_REG_5 (0x05)	7:6	RW	TX_DIN_SEL<1:0>	选择发射数据从哪个 GPIO 输入： 00: GPIO3 01: GPIO4 10: NIRQ

寄存器名	位数	R/W	比特名	功能说明
				11: NA
	5:3	RW	GPIO3_SEL<2:0>	选择 GPIO3 的功能: 000: INT2 001: INT1 010: DCLK 011: DOUT 101: DIN 其余选项: NA
CTL_REG_6 (0x06)	2:0	RW	GPIO4_SEL<2:0>	选择 GPIO4 的功能: 000: DOUT 001: INT1 010: INT2 011: DCLK 101: DIN 其余选项: NA
CTL_REG_7 (0x07)	2:0	RW	NIRQ_SEL<2:0>	选择 NIRQ 的功能: 000: INT1 001: INT2 010: DCLK 011: DOUT 101: DIN 其余选项: NA
CTL_REG_22 (0x16)	4	RW	TX_DATA_INV	0: GPIO 发射数据输入不取反 1: GPIO 发射数据输入取反

用户必须将 DATA_MODE 配置为 0，将 TX_DATA_MODE 配置为 0。

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

CMT2310A 有 6 个 GPIO，每个 GPIO 都可以配置成不同的输入或者输出；CMT2310A 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

以下为与 GPIO 配置相关的寄存器，属于配置区 1。

表 27. GPIO 相关寄存器

寄存器名	位数	R/W	比特名	功能说明
CTL_REG_4 (0x04)	5:3	RW	GPIO1_SEL<2:0>	选择 GPIO1 的功能： 000: DCLK 001: INT1 010: INT2 011: DOUT 其余选项: NA
	2:0	RW	GPIO0_SEL<2:0>	选择 GPIO0 的功能： 000: DOUT 001: INT1 010: INT2 011: DCLK 其余选项: NA
CTL_REG_5 (0x05)	5:3	RW	GPIO3_SEL<2:0>	选择 GPIO3 的功能： 000: INT2 001: INT1 010: DCLK 011: DOUT 101: DIN 其余选项: NA
	2:0	RW	GPIO2_SEL<2:0>	选择 GPIO2 的功能： 000: INT1 001: INT2 010: DCLK 011: DOUT 其余选项: NA
CTL_REG_6 (0x06)	5:3	RW	GPIO5_SEL<2:0>	选择 GPIO5 的功能： 000: RSTn 001: INT1 010: INT2 011: DOUT

寄存器名	位数	R/W	比特名	功能说明
				100: DCLK 其余选项: NA
	2:0	RW	GPIO4_SEL<2:0>	选择 GPIO4 的功能: 000: DOUT 001: INT1 010: INT2 011: DCLK 101: DIN 其余选项: NA
CTL_REG_7 (0x07)	2:0	RW	NIRQ_SEL<2:0>	选择 NIRQ 的功能: 000: INT1 001: INT2 010: DCLK 011: DOUT 101: DIN 其余选项: NA

上表中有点需要注意: 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

CMT2310A 有两个中断口, 分别是 INT1 和 INT2, 可以分配到不同的 GPIO 上。以下为中断相关的寄存器。

表 28. 中断相关寄存器

寄存器名	位数	R/W	比特名	功能说明
CTL_REG_7 (0x07)	7	RW	INT1_TYPE_SEL	选择从 GPIO 输出 INT1 的模式: 0: 模式 1, 输出某一个中断源 1: 模式 2, 输出全部中断源相或的结果
	6	RW	INT2_TYPE_SEL	选择从 GPIO 输出 INT2 的模式: 0: 模式 1, 输出某一个中断源 1: 模式 2, 输出全部中断源相或的结果
	2:0	RW	NIRQ_SEL<2:0>	选择 NIRQ 的功能: 000: INT1 001: INT2 010: DCLK 011: DOUT 101: DIN 其余选项: NA
CTL_REG_16 (0x10)	5:0	RW	INT1_SEL<5:0>	INT1 的映射选项, 请参考下面的中断映射表。

寄存器名	位数	R/W	比特名	功能说明
CTL_REG_17 (0x11)	7	RW	INT1_POLAR	中断 1 的极性: 0: 高有效 1: 低有效
	6	RW	INT2_POLAR	中断 2 的极性: 0: 高有效 1: 低有效
	5:0	RW	INT2_SEL<5:0>	INT2 的映射选项, 请参考下面的中断映射表。
CTL_REG_18 (0x12)	7	RW	SLEEP_TMO_EN	0: 屏蔽 SLEEP_TMO 中断 1: 使能 SLEEP_TMO 中断
	6	RW	RX_TMO_EN	0: 屏蔽 RX_TMO 中断 1: 使能 RX_TMO 中断
	5	RW	TX_DONE_EN	0: 屏蔽 TX_DONE 中断 1: 使能 TX_DONE 中断
	4	RW	PREAM_PASS_EN	0: 屏蔽 PREAM_PASS 中断 1: 使能 PREAM_PASS 中断
	3	RW	SYNC_PASS_EN	0: 屏蔽 SYNC_PASS 中断 1: 使能 SYNC_PASS 中断
	2	RW	ADDR_PASS_EN	0: 屏蔽 ADDR_PASS 中断 1: 使能 ADDR_PASS 中断
	1	RW	CRC_PASS_EN	0: 屏蔽 CRC_PASS 中断 1: 使能 CRC_PASS 中断
	0	RW	PKT_DONE_EN	0: 屏蔽 PKT_DONE 中断 1: 使能 PKT_DONE 中断
CTL_REG_21 (0x15)	6	RW	RSSI_PJD_VALID_EN	0: 屏蔽 RSSI_PJD_VALID 中断 1: 使能 RSSI_PJD_VALID 中断
	5	RW	OP_CMD_FAILED_EN	0: 屏蔽 API_CMD_FAILED 中断 1: 使能 API_CMD_FAILED 中断
	4	RW	RSSI_COLL_EN	0: 屏蔽 RSSI_COLL 中断 1: 使能 RSSI_COLL 中断
	3	RW	DC_FREE_ERR_EN	0: 屏蔽 DC_FREE_ERR 中断 1: 使能 DC_FREE_ERR 中断
	2	RW	LBD_STATUS_EN	0: 屏蔽 LBD_STATUS 中断 1: 使能 LBD_STATUS 中断
	1	RW	LBD_STOP_EN	0: 屏蔽 LBD_STOP 中断 1: 使能 LBD_STOP 中断
	0	RW	LD_STOP_EN	0: 屏蔽 LD_STOP 中断 1: 使能 LD_STOP 中断
CTL_REG_23 (0x17)	7	RW	API_DONE_EN	0: 屏蔽 API_DONE 中断 1: 使能 API_DONE 中断
	6	RW	CCA_STATUS_EN	0: 屏蔽 CCA_STATUS 中断 1: 使能 CCA_STATUS 中断

寄存器名	位数	R/W	比特名	功能说明
	5	RW	CSMA_MAX_EN	0: 屏蔽 CSMA_MAX 中断 1: 使能 CSMA_MAX 中断
	4	RW	TX_DC_MAX_EN	0: 屏蔽 TX_DC_MAX 中断 1: 使能 TX_DC_MAX 中断
	3	RW	ACK_RECV_FAILED_EN	0: 屏蔽 ACK_RECV_FAILED 中断 1: 使能 ACK_RECV_FAILED 中断
	2	RW	TX_RESEND_MAX_EN	0: 屏蔽 TX_RESEND_MAX 中断 1: 使能 TX_RESEND_MAX 中断
	1	RW	NACK_RECV_EN	0: 屏蔽 NACK_RECV 中断 1: 使能 NACK_RECV 中断
	0	RW	SEQ_MATCH_EN	0: 屏蔽 SEQ_MATCH 中断 1: 使能 SEQ_MATCH 中断
CTL_REG_24 (0x18)	5	R	SLEEP_TMO_FLG	SLEEP_TMO 中断标志
	4	R	RX_TMO_FLG	RX_TMO 中断标志
	3	R	TX_DONE_FLG	TX_DONE 中断标志
	2	W	SLEEP_TMO_CLR	SLEEP_TMO 中断清零 0: 无动作 1: 清零
	1	W	RX_TMO_CLR	RX_TMO 中断清零 0: 无动作 1: 清零
	0	W	TX_DONE_CLR	TX_DONE 中断清零 0: 无动作 1: 清零
CTL_REG_25 (0x19)	4	W	PREAM_PASS_CLR	PREAM_PASS 中断清零 0: 无动作 1: 清零
	3	W	SYNC_PASS_CLR	SYNC_PASS 中断清零 0: 无动作 1: 清零
	2	W	ADDR_PASS_CLR	ADDR_PASS 中断清零 0: 无动作 1: 清零
	1	W	CRC_PASS_CLR	CRC_PASS 中断清零 0: 无动作 1: 清零
	0	W	PKT_DONE_CLR	PKT_DONE 中断清零 0: 无动作 1: 清零
CTL_REG_26 (0x1A)	5	R	SYNC1_PASS_FLG	SYNC1_PASS 中断标志
	4	R	PREAM_PASS_FLG	PREAM_PASS 中断标志
	3	R	SYNC_PASS_FLG	SYNC_PASS 中断标志

寄存器名	位数	R/W	比特名	功能说明
	2	R	ADDR_PASS_FLG	ADDR_PASS 中断标志
	1	R	CRC_PASS_FLG	CRC_PASS 中断标志
	0	R	PKT_DONE_FLG	PKT_DONE 中断标志
CTL_REG_28 (0x1C)	7	R	RX_FIFO_FULL_FLG	指示 RX FIFO 填满的中断。 0: 无效 1: 有效
	6	R	RX_FIFO_NMTY_FLG	指示 RX FIFO 非空的中断标志位。 0: 无效 1: 有效
	5	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
CTL_REG_29 (0x1D)	3	W	OP_CMD_FAILED_CLR	OP_CMD_FAILED 中断清零 0: 无动作 1: 清零
	2	W	RSSI_COLL_CLR	RSSI_COLL 中断清零 0: 无动作 1: 清零
	1	W	DC_FREE_ERR_CLR	DC_FREE_ERR 中断清零 0: 无动作 1: 清零
	0	W	LBD_STATUS_CLR	LBD_STATUS 中断清零 0: 无动作 1: 清零
CTL_REG_30 (0x1E)	3	R	OP_CMD_FAILED_FLG	OP_CMD_FAILED 中断标志
	1	R	DC_FREE_ERR_FLG	DC_FREE_ERR 中断标志
	0	R	LBD_STATUS_FLG	LBD_STATUS 中断标志
CTL_REG_31	7	W	API_DONE_CLR	API_DONE 中断清零

寄存器名	位数	R/W	比特名	功能说明
(0x1F)				0: 无动作 1: 清零
	6	W	CCA_STATUS_CLR	CCA_STATUS 中断清零 0: 无动作 1: 清零
	5	W	CSMA_MAX_CLR	CSMA_MAX 中断清零 0: 无动作 1: 清零
	4	W	TX_DC_MAX_CLR	TX_DC_MAX 中断清零 0: 无动作 1: 清零
	3	W	ACK_RECV_FAILED_CLR	ACK_RECV_FAILED 中断清零 0: 无动作 1: 清零
	2	W	TX_RESEND_MAX_CLR	TX_RESEND_MAX 中断清零 0: 无动作 1: 清零
	1	W	NACK_RECV_CLR	NACK_RECV 中断清零 0: 无动作 1: 清零
	0	W	SEQ_MATCH_CLR	SEQ_MATCH 中断清零 0: 无动作 1: 清零
CTL_REG_32 (0x20)	7	R	API_DONE_FLG	API_DONE 中断标志
	6	R	CCA_STATUS_FLG	CCA_STATUS 中断标志
	5	R	CSMA_MAX_FLG	CSMA_MAX 中断标志
	4	R	TX_DC_MAX_FLG	TX_DC_MAX 中断标志
	3	R	ACK_RECV_FAILED_FLG	ACK_RECV_FAILED 中断标志
	2	R	TX_RESEND_MAX_FLG	TX_RESEND_MAX 中断标志
	1	R	NACK_RECV_FLG	NACK_RECV 中断标志
	0	R	SEQ_MATCH_FLG	SEQ_MATCH 中断标志

下表给出中断映射表，INT1 和 INT2 的映射是相同的，以下以 INT1 为例说明。

表 29. CMT2310A 中断映射表

名称	INT1_SEL	描述	清除方式
RSSI_PJD_VALID	000010	RSSI 和（或）PJD 的组合有效中断	Auto
PREAM_PASS	000011	指示成功收到 Preamble 的中断	By MCU
SYNC_PASS	000100	指示成功收到 Sync Word 的中断	By MCU
ADDR_PASS	000101	指示成功收到 Addr 的中断	By MCU
CRC_PASS	000110	指示成功收到并通过 CRC 校验的中断	By MCU
PKT_DONE	000111	指示完整收到一个数据包的中断	By MCU
SLEEP_TMO	001000	指示 SLEEP 计数器超时的中断	By MCU
RX_TMO	001001	指示 RX 计数器超时的中断	By MCU
RX_FIFO_NMTY	001010	指示 RX FIFO 非空的中断	Auto
RX_FIFO_TH	001011	指示 RX FIFO 未读内容超过 FIFO TH 的中断	Auto
RX_FIFO_FULL	001100	指示 RX FIFO 填满的中断	Auto
RX_FIFO_WBYTE	001101	指示 RX FIFO 每写入一个 BYTE 的中断，是脉冲	Auto
RX_FIFO_OVF	001110	指示 RX FIFO 溢出的中断	Auto
TX_DONE	001111	指示 TX 完成的中断	By MCU
TX_FIFO_NMTY	010000	指示 TX FIFO 非空的中断	Auto
TX_FIFO_TH	010001	指示 TX FIFO 未读内容超过 FIFO TH 的中断	Auto
TX_FIFO_FULL	010010	指示 TX FIFO 满的中断	Auto
STATE_IS_READY	010011	指示当前状态是 READY 的中断	Auto
STATE_IS_FS	010100	指示当前状态是 RFS 或 TFS 的中断	Auto
STATE_IS_RX	010101	指示当前状态是 RX 的中断	Auto
STATE_IS_TX	010110	指示当前状态是 TX 的中断	Auto
DC_FREE_ERR	010111	曼切斯特解码出错的中断	By MCU
LBD_STATUS	011000	指示低电压检测有效（VDD 低于设置的 TH）的中断	By MCU
API_CMD_FAILED	011010	API 命令执行错误中断	By MCU
API_DONE	011011	API 命令完成指示中断	By MCU
TX_DC_MAX	011100	Duty Cycle 发射模式运行达到最大次数中断	By MCU
ACK_RECV_FAILED	011101	ACK 接收失败中断	By MCU
TX_RESEND_MAX	011110	重复发射运行达到最大次数中断	By MCU
NACK_RECV	011111	接收到 NACK 的指示中断	By MCU
SEQ_MATCH	100000	序列号匹配成功中断	By MCU
CSMA_MAX	100001	CSMA 运行达到最大次数中断	By MCU
CCA_STATUS	100010	信道监听状态中断	By MCU

中断默认是 1 有效,但是可通过将 INT_POLAR 这个寄存器比特设置为 1,使所有中断都变成 0 有效。下图以 INT1 为例,给出了两个不同性质的中断源的控制和选择图。对于控制和映射来说,INT1 和 INT2 也是相同的。

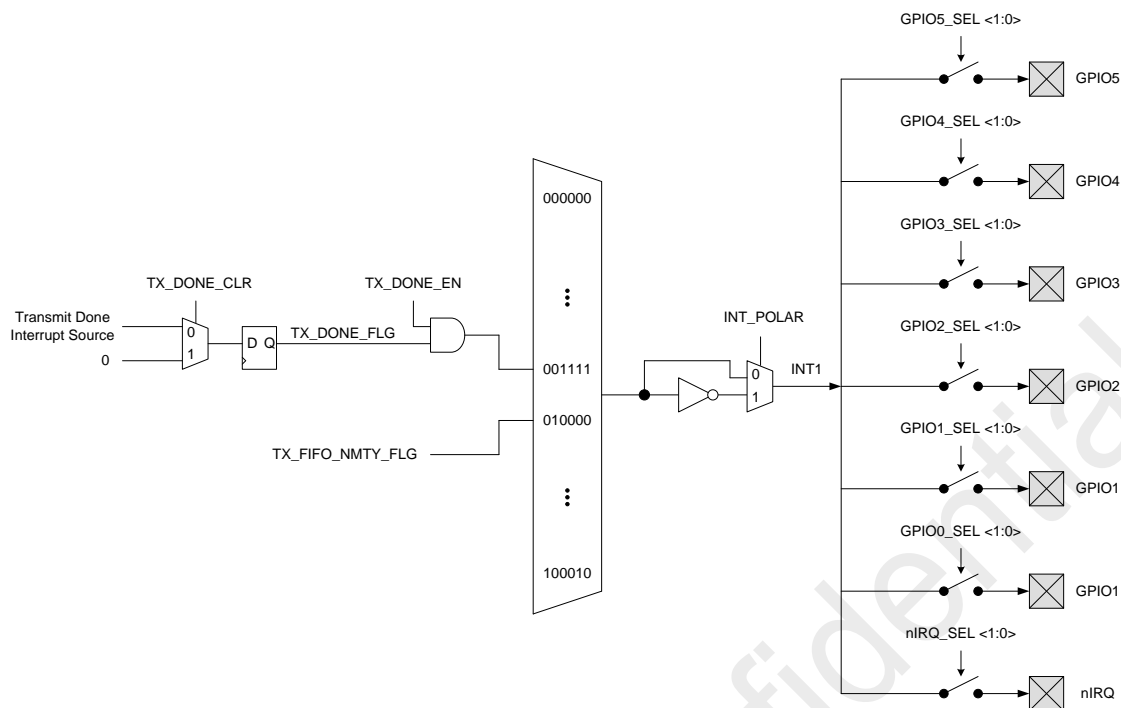


图 27. CMT2310A INT1 中断映射图

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。例如，SYNC_OK 的中断源只有在 SYNC_OK_EN 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 才会将它清零。当设置 CLR 比特时，MCU 只需要将它设为 1 即可，随后不需要将它设置回 0，因为在芯片内部此 CLR 比特在对应的中断清零之后，也会会自动清零。

CMT2310 GPIO 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，CMT2310A 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>寄存器，来设定芯片自动退出 TX 状态之后，将会切换到 SLEEP/READY/TFS 其中之一的状态。之后，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

以下为控制外部天线进行 TX/RX 切换控制相关的寄存器。

表 30. 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
CTL_REG_22 (0x16)	1	RW	TRX_SWT_INV	控制 TX/RX 天线开关的两个输出值： 0: 不取反 1: 取反
	0	RW	TRX_SWT_EN	将 GPIO0 和 GPIO1 设置为 TX/RX 天线开关的控制输出。该功能的优先级比 GPIO0_SEL 和 GPIO1_SEL 要高。

芯片可以在 GPIO0 和 GPIO1 输出一组（2 个不同的）信号，来控制外部天线的 RX/TX 切换。

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      cmt2310a_fifo_rd_wr_sel
* @desc      read or write fifo(only valid by fifo merge enable)
* @param     rw    0: read fifo, 1: write fifo
* *****/
void cmt2310a_fifo_rd_wr_sel(u8 rw)
{
    cmt2310a_set_reg_bits(CMT2310A_CTL_REG_19, (rw << BIT0), M_FIFO_RD_WR_SEL);
}

```

4.2 附录 2 Sample code GPIO 输出中断配置函数示例

```

void rf_config(void)
{
    /* Config registers */
    cmt2310a_config_page_regs(0, g_cmt2310a_page0, CMT2310A_PAGE0_SIZE); /* config page 0 */
    cmt2310a_config_page_regs(1, g_cmt2310a_page1, CMT2310A_PAGE1_SIZE); /* config page 1 */
#ifdef CMT2310A_AUTO_RX_HOP_ENABLE /* cmt2310a_param.h exported by rfpdk */
    cmt2310a_select_reg_page(2);
    for(u8 idx = 0; idx < freq_times_val; idx++)
    {
        cmt2310a_write_reg(idx, g_cmt2310a_page2[idx]); /* config page 2 */
    }
    cmt2310a_select_reg_page(0);
    cmt2310a_write_reg(CMT2310A_CTL_REG_12, freq_space_val);
    cmt2310a_write_reg(CMT2310A_CTL_REG_13, freq_times_val);
    cmt2310a_write_reg(CMT2310A_CTL_REG_14, freq_switch_state_val);
#endif

    cmt2310a_go_ready();
    cmt2310a_delay_ms(2);

    /* do ir calibration */
    cmt2310a_api_command_and_wait(0x01);
}

```

```
cmt2310a_api_command_and_wait(0x01);

/* Config GPIOs */
cmt2310a_config_gpio0(GPIO0_SEL_INT1); /* INT1 -> GPIO0 */
cmt2310a_config_gpio1(GPIO1_SEL_INT2); /* INT2 -> GPIO1 */

/* Config interrupt */
cmt2310a_config_interrupt(
    INT_SRC_TX_DONE,    /* Config TX_DONE -> INT1 */
    INT_SRC_PKT_DONE    /* Config PKT_DONE -> INT2 */
);

/* config interrupt mode */
cmt2310a_config_interrupt_mode(INT1_TYPE_SEL_MODE1, INT2_TYPE_SEL_MODE1);

/* config interrupt polar */
cmt2310a_set_interrupt_polar(0,0); /* INT1: active-high, INT2: active-high */

/* enable interrupt source */
cmt2310a_enable_interrupt_source_0(
    M_TX_DONE_EN      |
    M_PREAM_PASS_EN   |
    M_SYNC_PASS_EN    |
    M_ADDR_PASS_EN    |
    M_CRC_PASS_EN     |
    M_PKT_DONE_EN
);

/* Use a single 256-byte FIFO for either Tx or Rx */
//cmt2310a_enable_fifo_merge(1);

//cmt2310a_set_fifo_threshold(16);

/* save fifo at sleep state or not */
//cmt2310a_fifo_sleep_save(0); /* 0: save, 1: no save */

/* select pa mode, 0: single, 1: difference */
cmt2310a_select_pa_mode(0);
}
```


5. 文档变更记录

表 31. 文档变更记录表

版本号	章节	变更描述	日期
0.5	所有	初始版本发布	2020-09-17

6. 联系方式

无锡泽太微电子有限公司深圳分公司

深圳市南山区西丽街道万科云城 3 期 8 栋 A 座 30 楼

邮编: 518055

电话: +86-755-83231427

销售: sales@cmostek.com

技术支持: support@cmostek.com

网址: www.cmostek.com

版权所有 © 无锡泽太微电子有限公司，保留一切权利。

无锡泽太微电子有限公司（以下简称：“CMOSTEK”）保留随时更改、更正、增强、修改 CMOSTEK 产品和/或本文档的权利，恕不另行通知。非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。由于产品版本升级或其他原因，本文档内容会不定期进行更新。CMOSTEK 的产品不建议应用于生命相关的设备和系统，在使用该器件中因为设备或系统运转失灵而导致的损失，CMOSTEK 不承担任何责任。

CMOSTEK 商标和其他 CMOSTEK 商标为无锡泽太微电子有限公司的商标，本文档提及的其他所有商标或注册商标，由各自的所有人拥有。