CMT2310A快速上手指南

概要

本应用文档为用户提供使用 CMT2310A 进行产品开发的关键功能和关键信息介绍，以使用户能够快速入门进行该产品的应用开发。

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	113 -960MHz	(4)(G)FSK/OOK	收发一体	寄存器	QFN24

用户需要结合阅读以下的应用文档，以了解全部的信息来辅助软硬件开发：

《AN238 CMT2310A 射频参数配置指南》

《AN235 CMT2310A FIFO 和包格式使用指南》

《AN236 CMT2310A 寄存器说明》

《AN239 CMT2310A 自动收发功能使用指南》

目录

1 芯片架构介绍 3
1.1 基本介绍 3
1.2 IO 管脚说明 4
2 SPI 接口时序 6
2.1 读/写寄存器操作 6
2.2 批量（BURST）读写寄存器操作 7
2.3 读/写 FIFO 操作 9
3 配置和控制机制 11
3.1 寄存器概览 11
3.2 工作状态切换 12
3.3 芯片复位机制 14
3.4 RFPDK 简介 14
3.5 芯片初始化流程 18
3.6 流程总结 19
4 cmt2310a_easy_demo 简介 20
4.1 软件层次结构 20
4.2 软件实现以及调用关系 21
4.2.1 CMT2310A 初始化 22
4.2.2 CMT2310A 配置 23
4.2.3 CMT2310A 状态处理 24
4.3 软件目录结构 25
4.3.1 应用层源代码 26
4.3.2 模拟 SPI 实现源代码 27
4.3.3 抽象硬件层源代码 27
4.3.4 芯片驱动层源代码 29
4.3.5 芯片处理层源代码 30
5 附录 31
5.1 附录 1：Sample Code -SPI 读写操作代码示例 31
5.2 附录 2：Sample Code -状态切换库函数代码示例 34
5.3 附录 3：SampleCode - 初始化函数代码示例 35
6 文档变更记录 38
7 联系方式 39

1 芯片架构介绍

1.1 基本介绍

CMT2310A 是一款数字模拟一体化收发机产品。该产品采用 32MHz 晶体提供 PLL 的参考频率和数字时钟，同时支持 OOK、2(G)FSK 和 4(G)FSK 的调制解调模式，并支持 Direct 和 Packet 两种数据处理模式。

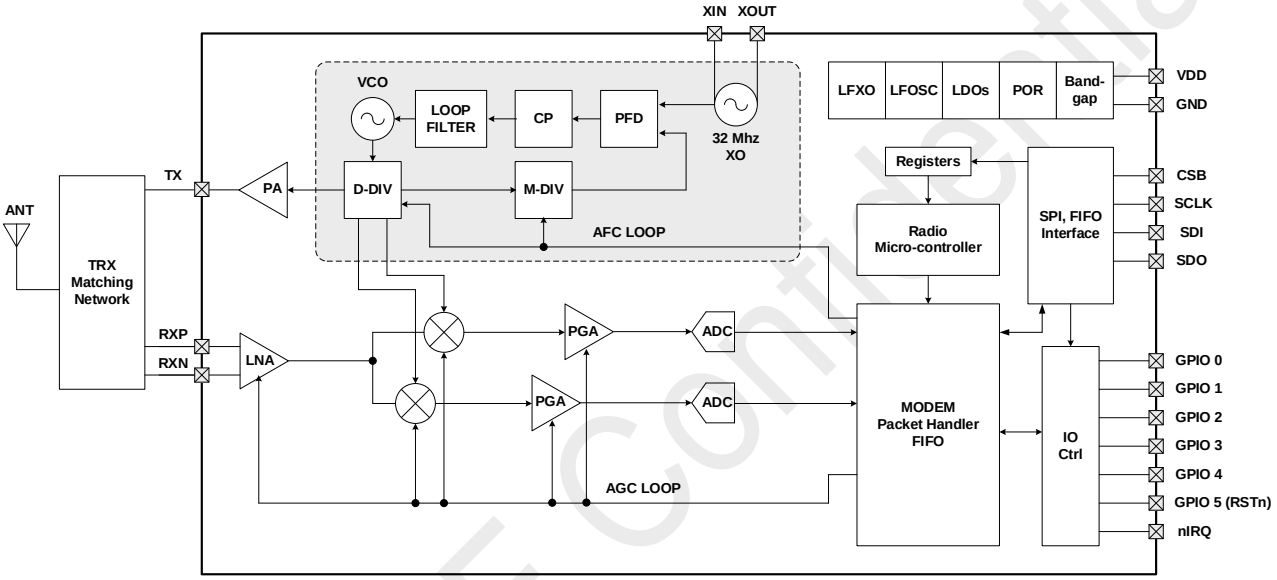

图 1. CMT2310A 系统框图

该芯片采用 LNA+PGA+ADC+PLL 的低中频结构实现 1G 以下频率的无线接收功能；采用 PLL+PA 结构实现 1G 以下频率的无线发射功能。

在接收机系统内，模拟电路负责将射频信号下混频至中频，并通过 ADC 模块做对中频信号数模转换处理，输出 I/Q 两路单比特信号到数字电路做后续的数字解调。数字电路负责将中频信号下混频到零频（基带）并进行一系列滤波和判决处理，同时进行 AFC 和 AGC 动态地控制模拟电路，最后将 1-bit 的原始的信号解调出来。信号解调出来之后，会送到解码器里面进行解码并填入 FIFO，或者直接输出到 GPIO。

在发射机系统内，数字电路会对数据进行编码打包处理，并将处理后的数据送到调制器（也可不经过编码打包，直接送到调制器），调制器会直接控制 PLL 和 PA，对数据进行（G）FSK 或者 OOK 调制并发射出去。

芯片内部有一个小型的微控制器（只限于原厂内部编程），该控制器负责调度芯片的各种运作，包括支持丰富的无线收发处理，例如自动 ACK，自动跳频，DUTY-CYCLE 低功耗收发，CSMA 等特色功能。

芯片提供了 SPI 通讯接口，外部的 MCU 可以通过访问寄存器的方式来对芯片的各种功能进行配置，

控制主芯片，并访问 FIFO。

1.2 IO 管脚说明

下面以 QFN24 的封装为例，说明 CMT2310A 的管脚分配和功能：

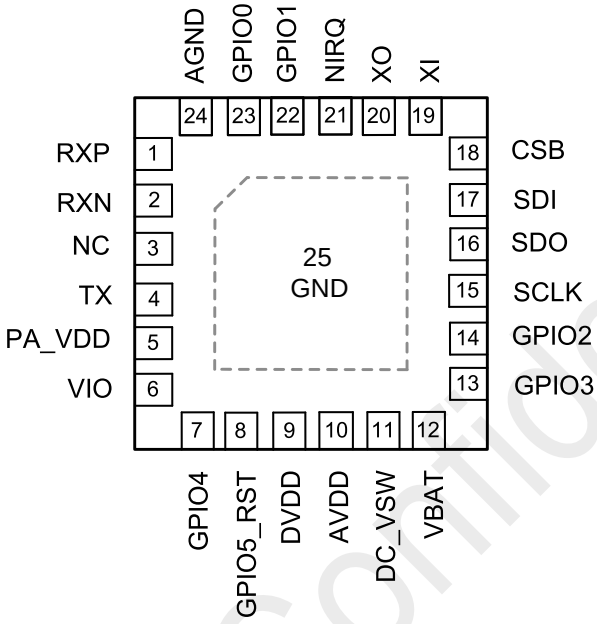

图 2. CMT2310A 芯片管脚图

表 2. CMT2310A 管脚描述

管脚号	名称	I/O	功能说明
1	RXP	I	RF 信号输入 P
2	RXN	I	RF 信号输入 N
3	NC	--	悬空
4	TX	O	PA 输出
5	PA_VDD	IO	PA VDD
6	VIO	IO	IO VDD
7	GPIO4	IO	可配置为：DOUT，INT1，INT2，DCLK，CLKO，LFCLKO。
8	GPIO5_RST	IO	可配置为：RSTn，INT1，INT2，DOUT，DCLK。
9	DVDD	I	数字 VDD
10	AVDD	I	模拟 VDD
11	DC_VSW	I	DCDC
12	VBAT	I	模拟 VDD
13	GPIO3	IO	可配置为：INT1，INT2，DCLK，DOUT，LFXO2，ANTD2。
14	GPIO2	IO	可配置为：INT1，INT2，INT3，DCLK，DOUT，LFXO1，ANTD1。
15	SCLK	I	SPI 的时钟
16	SDO	O	SPI 的数据输出
17	SDI	I	SPI 的数据输入

管脚号	名称	I/O	功能说明
18	CSB	I	SPI 的片选输入
19	XI	I	晶体电路输入
20	XO	O	晶体电路输出
21	NIRQ	I	可配置为：INT1，INT2，DCLK，DOUT，TCXO。
22	GPIO1	IO	可配置为：DCLK，INT1，INT2，DOUT，TRX_SWT。
23	GPIO0	IO	可配置为：DOUT，INT1，INT2，INT3，DCLK，TRX_SWT。
24	AGND	I	模拟 GND
25	GND	I	芯片衬底 GND

2 SPI 接口时序

芯片通过 4 线 SPI 接口与外部进行通信，SPI 默认为 4 线，上电后可配置为 3 线。低有效的 CSB 为用于访问寄存器的片选信号。SCLK 为串口时钟，最快速度可达 10MHz。无论对芯片本身，还是外部的 MCU，都是在 SCLK 下降沿送出数据，在上升沿采集数据。SDI 用于数据输入，SDO 用于数据输出。在 3 线模式下，SDI 同时用于数据输入和输出，SDO 闲置。地址和数据部分均从 MSB 开始传送。

2.1 读/写寄存器操作

当访问寄存器时，CSB 要拉低，之后首先发送一个 R/W 位，接着为 7 位的寄存器地址。外部 MCU 在拉低 CSB 之后，必须等待至少半个 SCLK 周期，才能开始发送 R/W 位。在 MCU 发送出最后一个 SCLK 下降沿之后，必须等待至少半个 SCLK 周期，再把 CSB 拉高。

需注意的是，对于 4 线写寄存器操作，在 SDI 输入写数据的同时，SDO 会输出该寄存器当前的值（old register read data），MCU 可根据需要决定是否读取该值。

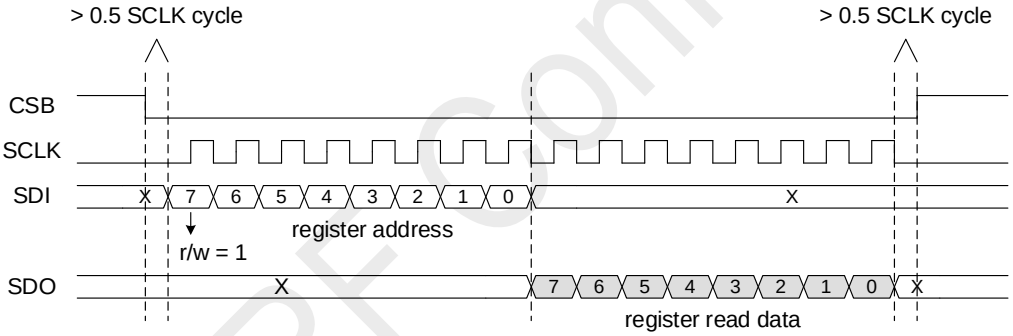

图 3. SPI（4 线）读寄存器时序

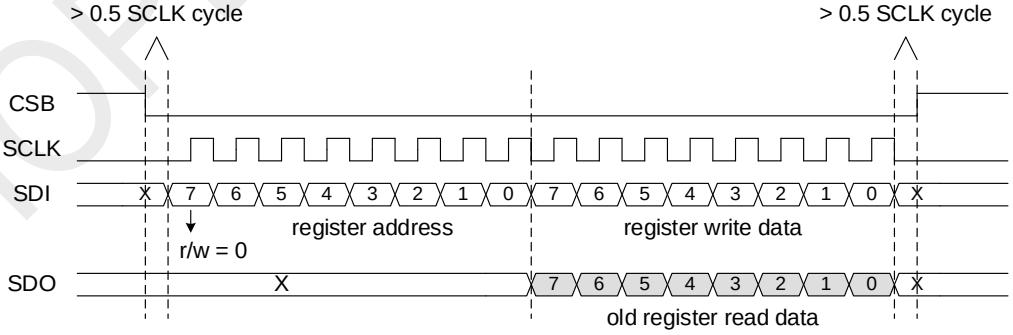

图 4. SPI（4 线）写寄存器时序

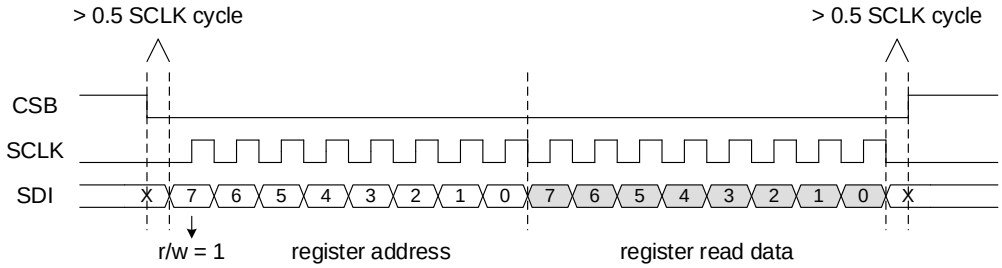

图 5. SPI（3 线）读寄存器时序

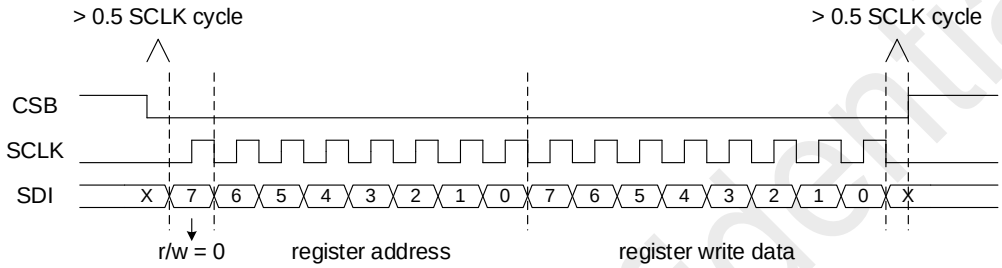

图 6. SPI（3 线）写寄存器时序

对于图 6 的读寄存器操作，MCU 和 CMT2310A 都会在地址 0 和数据 7 之间产生切换 IO（SDI）口的行为。此时 CMT2310A 会将 IO 口从输入切换到输出，MCU 会将 IO 口从输出切换到输入。请注意中间虚线的位置，此时强烈建议 MCU 在送出 SCLK 的下降沿前，先将 IO 口切换为输入；CMT2310A 在看到下降沿之后，才会将 IO 切换为输出。这样可避免两者同时将 SDIO 设为输出导致电气冲突的情况。对于某些 MCU 来说，这种情况可能会导致其复位或出现其它异常行为。

SPI 读取操作代码示例详见附录 1。

2.2 批量（BURST）读写寄存器操作

除了上面陈述的单字节读写寄存器操作，SPI 还可以支持 Burst 读写寄存器的操作。BURST 读写操作以写入 Page 0 的 0x7B 地址 BRW_PORT 来触发，当 r/w 比特为 0 时，会进行写寄存器操作，当为 1 时，会进行读寄存器操作。

BURST 读写也可使用 3 线 SPI 来操作；当使用 3 线时，读数据的输出和写数据的输入，都是在 SDI 管脚上进行。当使用 4 线时，写数据从 SDI 输入，读数据从 SDO 输出。BURST 读写的操作流程为，先访问 0x7B 地址的 BRW 操作端口，其中包含的读写位决定后面是写数据还是读数据操作。后面一直为读或者写的数据阶段，用户决定何时完成操作。

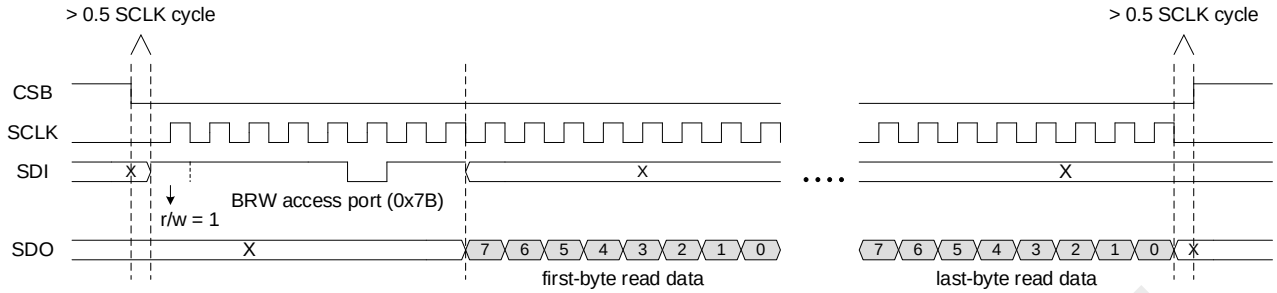

图 7. SPI（4 线）BURST 读时序

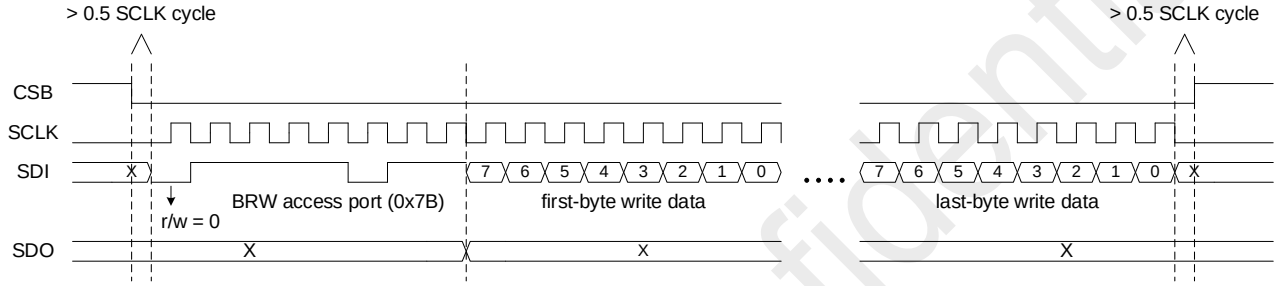

图 8. SPI（4 线）BURST 写时序

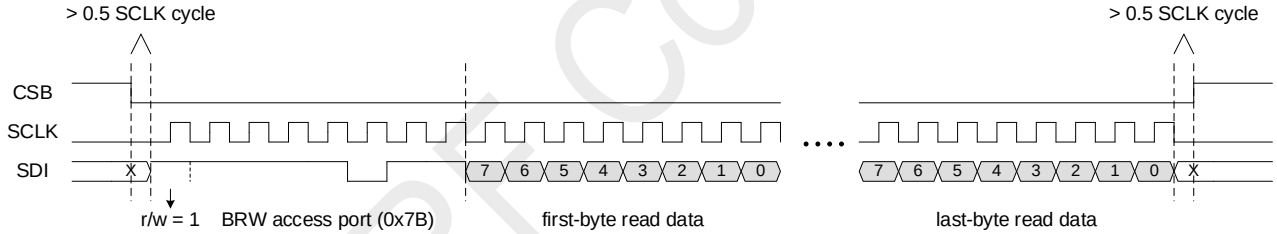

图 9. SPI（3 线）BURST 读时序

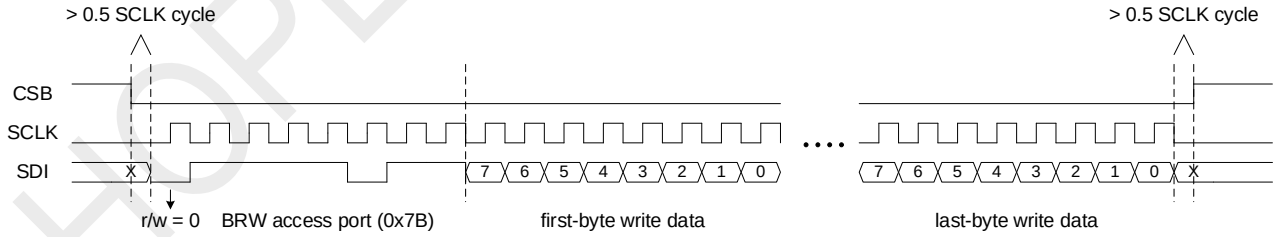

图 10. SPI（3 线）BURST 写时序

需要注意的是 BURST 读写不能跨越 Page，即一次 BURST 只能在一个 Page 中完成。下面列出了每个 Page 的可以进行 BURST 读写的最大地址范围：

PAGE	起始地址	结束地址	地址个数	说明
0	0x28	0x77	80	Page0 的配置寄存器
1	0x00	0x6F	112	Page1 的配置寄存器
2	0x00	0x3F	64	Page2 的跳频频道表

用户可以在每一个 Page 的地址范围内，选择任意起始地址，进行 N 个地址的连续 BURST 读写操作，N 的取值不能让结束地址超出地址范围。BURST 读写的常用方式，是上电之前让用户进行快速的一次性配置，配置寄存器内容来源于 RFPDK 工具，跳频频道表的内容由用户自己设计。

2.3 读/写 FIFO 操作

CMT2310A 默认提供两个独立的 128-byte 的 FIFO，分别给 RX 和 TX 使用，两者相互独立。RX FIFO 用来在 RX 模式中存储接收数据，TX FIFO 用于 TX 模式中存储即将发射的数据。用户也可以将 FIFO_MARGE_EN 设成 1，那么两个 FIFO 就合成一个 256-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。如果没有使用合并，当 128 字节 RX FIFO 被填入时，用户可以同时为下一次发射填入 128 字节的 TX FIFO，以节省系统操作时间。

FIFO 可以通过 SPI 接口访问。用户可以通过设置 TX_FIFO_CLR/RX_FIFO_CLR 位来清空 FIFO。并且，用户可以通过设置 FIFO_RESTORE 来重复发射之前填入的数据，无需重新填入数据。

用户可以通过配置 PD_FIFO 来控制 FIFO 是否在 SLEEP 状态下保存内容。PD_FIFO = 0 指 FIFO 可以在 SLEEP 状态下保存内容，但会消耗 200 nA 左右的漏电电流。

当 MCU 需要访问 FIFO 的时候，首先需配置某些寄存器，来设置好 FIFO 的读/写模式，以及其它工作模式，详见 AN235《CMT2310A FIFO 和包格式使用指南》，此处给出的是确定模式后的读写时序图。FIFO 的操作以写入 Page 0 的 0x7A 地址来触发，当 r/w 比特为 0 时，会进行写 FIFO 操作，当为 1 时，会进行读 FIFO 操作。

FIFO 的读写也可使用 3 线 SPI 来操作；当使用 3 线时，读数据的输出和写数据的输入，都是在 SDI 管脚上进行。当使用 4 线时，写数据从 SDI 输入，读数据从 SDO 输出。FIFO 的操作流程为，先访问 0x7A 地址的 FIFO 操作端口，其中包含的读写位决定后面是写数据还是读数据操作。后面一直为读或者写的数据阶段，用户决定何时完成操作。

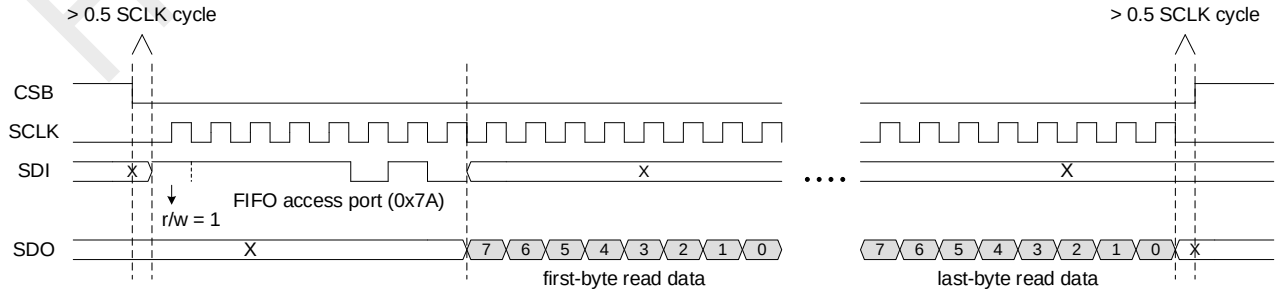

图 11. SPI（4 线）读 FIFO 时序

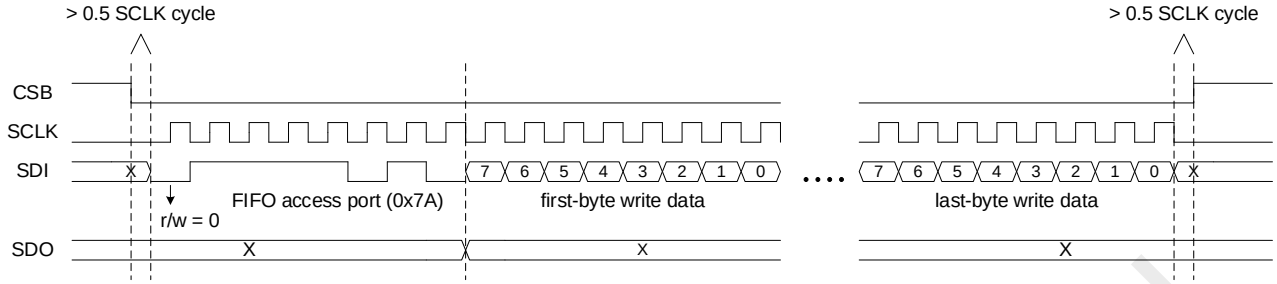

图 12. SPI（4 线）写 FIFO 时序

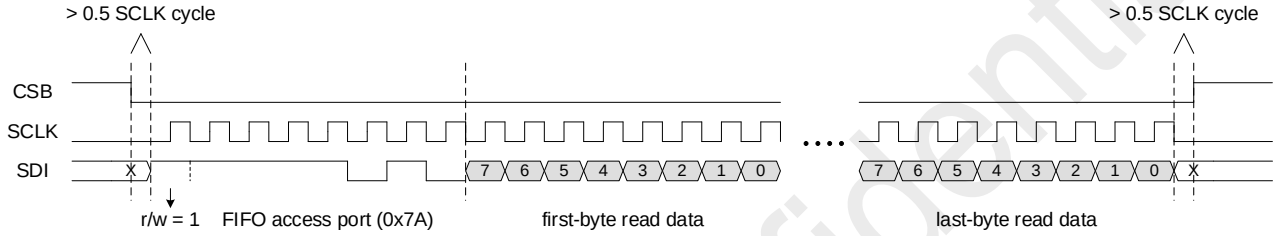

图 13. SPI（3 线）读 FIFO 时序

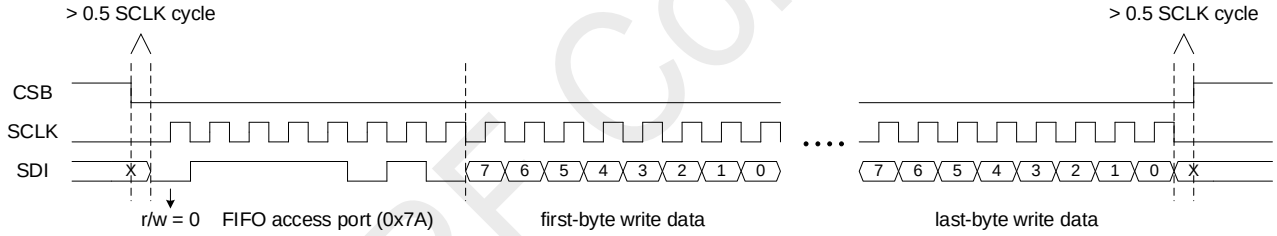

图 14. SPI（3 线）写 FIFO 时序

SPI 读写 FIFO 操作代码示例详见附录 2。

3 配置和控制机制

3.1 寄存器概览

外部 MCU 对芯片的配置和控制，全部基于使用 SPI 来访问 3 页的寄存器完成。三个页之间的切换通过写入 0x7E 地址完成。如下表所示，地址从 0x00 到 0x77，可以分成两个大区去理解，分别是：配置区（其中包括 6 个子区）和控制区（包含 2 个子区）。下面将进行详细介绍。

首先，对两个区来说，地址是连续的，操作方式无本质区别，都是使用 SPI 按照访问寄存器的时序进行直接读写操作。但是从功能和使用方式来说，两个区有不同的作用，如下表所示：

表 3. CMT2310A 寄存器区域划分表

页	地址	区域	功能描述
0	0x00 - 0x17	系统控制区	用于操控芯片，例如状态切换，状态查询等。以及用于使能芯片的某些功能。
	0x18 - 0x27	中断控制区	用于读取和清除中断标志位，在 SLEEP 状态不可访问。
	0x28 - 0x5F	配置区（包格式）	用于配置包格式和 FIFO 相关功能
	0x60 - 0x77	配置区（系统特性）	用于配置系统运行相关功能
1	0x00 - 0x0F	配置区（CMT 专用）	仅限 CMT 内部使用
	0x10 - 0x27	配置区（TX）	用于配置发射机特性
	0x30 - 0x68	配置区（RX）	用于配置接收机特性
2	0x00 - 0x3F	配置区（自动跳频）	用于存放自动跳频的频道跳转表。

备注：

1. 在页 0 中，0x7A 地址为 FIFO 读写的端口地址
2. 在所有页中，0x7B 地址为寄存器批量读写的端口地址。
3. 在所有页中，0x7E 是页切换地址。
4. 在所有页中，0x7F 是软复位地址。
5. 除此之外，未列入上表中的地址，禁止访问。

除“中断配置区”在 SLEEP 状态下不可访问，其余所有区域均可在 SLEEP 状态下访问，只要电池不断电，芯片不进行 POR 复位操作，配置好的内容不会丢失。

页 0 和 1 中的寄存器的值，既可以来自于 RFPDK，也可以由客户在应用过程中根据自己的需求去更改。一般来说，除了个别关于 RF 频率或者数据率的配置，有可能需要在应用中进行多次配置，其余大部分寄存器，只需要在初始化过程中配置一次即可。

页 2 中的跳频表格，需要用户根据 AN 文档指引进行设计和配置。

3.2 工作状态切换

芯片的工作状态切换图如下图所示，共有 7 个状态，用户通过配置 CTL_REG_0 和 CTL_REG_1 寄存器来发送切换命令，通过读取 CTL_REG_10 寄存器来查询各个状态的对应码值。

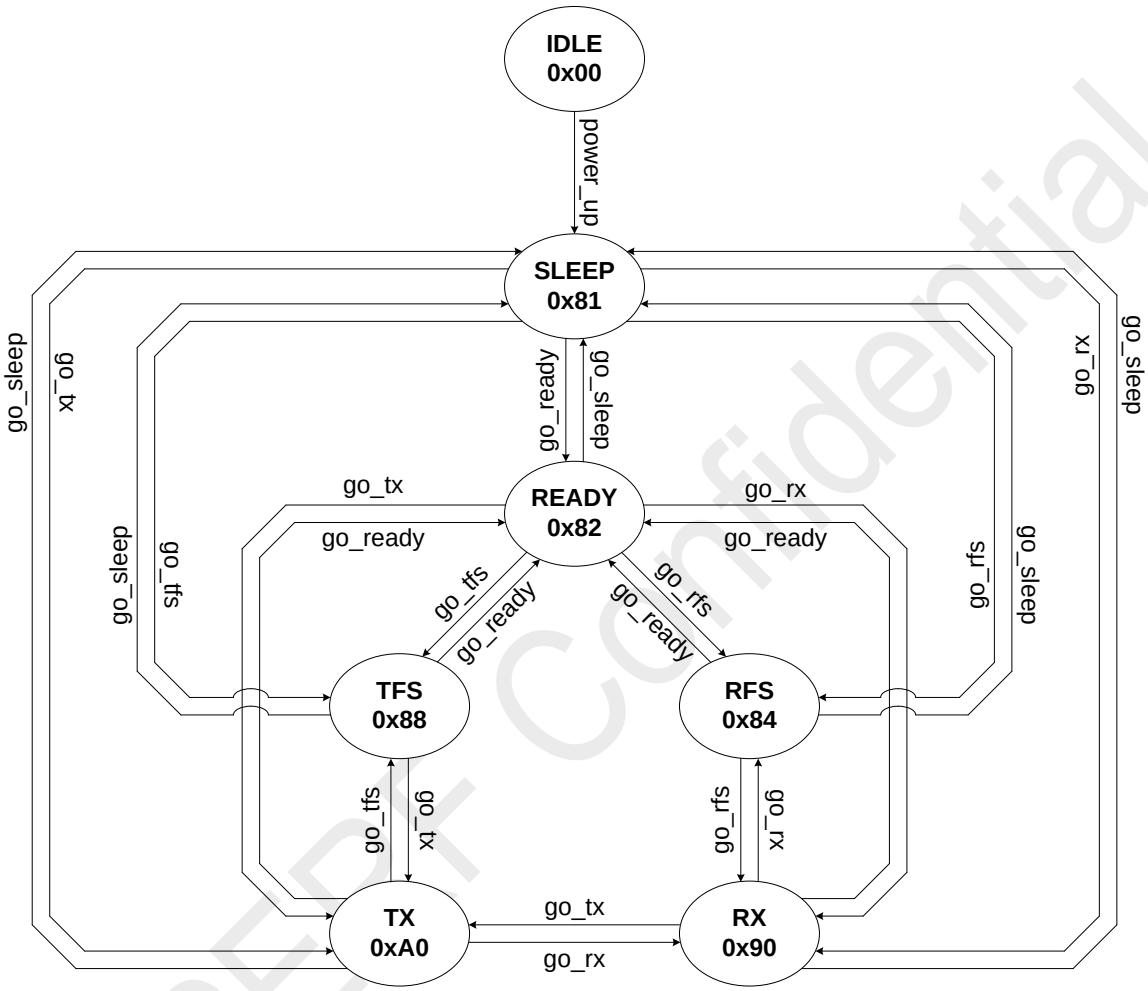

图 15. CMT2310A 状态切换图

表 4. CMT2310A 状态和模块开启表

状态	状态码	切换命令	开启模块	可选择开启模块
IDLE	0x00	soft_rst	SPI, POR	无
SLEEP	0x81	go_sleep	SPI, POR	LFOSC/LFXO, FIFO, Sleep Timer
READY	0x82	go_ready	SPI, POR, XTAL, FIFO	无
RFS	0x84	go_rfs	SPI, POR, XTAL, PLL, FIFO	无
TFS	0x88	go_tfs	SPI, POR, XTAL, PLL, FIFO	无
RX	0x90	go_rx	SPI, POR, XTAL, PLL, LNA+MIXER+ADC, FIFO	RX Timer
TX	0xA0	go_tx	SPI, POR, XTAL, PLL,PA, FIFO	无

外部 MCU 可以通过访问以下控制区寄存器来切换和查询芯片工作状态。

表 5.切换状态的寄存器

寄存器名	位数	R/W	比特名	功能说明
CTL_REG_00 (0x00)	7:0	W	POWERUP<7:0>	上电命令： 0x03：power_up 禁止写入其余值。
CTL_REG_01 (0x01)	7:0	W	CHIP_MODE_SWT<7:0>	状态切换的命令： 0x01：go_sleep 0x02：go_ready 0x20：go_rfs 0x10：go_tfs 0x08：go_rx 0x04：go_tx 禁止写入其余值。
CTL_REG_10 (0x0A)	7:0	R	CHIP_MODE_STA<3:0>	芯片的状态码： 0x00：IDLE 0x81：SLEEP 0x82：READY 0x84：RFS 0x88：TFS 0x90：RX 0xA0：TX 其余值：无效 如果收到其余值，证明芯片工作不正常，建议复位芯片。

MCU 在发送 go_*命令之后，芯片有时需要等待一定的时间才能成功切换状态，下面会分别说明每个状态以及切换需要等待的时间。

■ IDLE 状态与上电流程

芯片在 VDD 接通后，等待约 1ms 的时间后 POR 就会释放，但芯片会停留在 IDLE 状态，不做任何动作。用户发送 power_up 命令后，芯片启动上电流程，进行各模块校正。芯片完成校正后会停留在 SLEEP。任何时候，只要进行复位（包括 POR，硬复位，和软复位），芯片会回到 IDLE 状态，等待用户重新发送 power_up 命令。

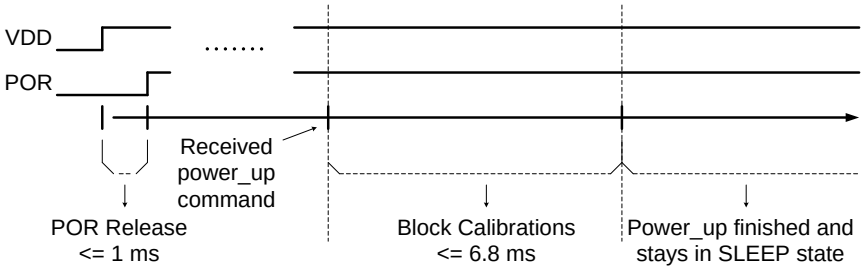

图 16. CMT2310A 上电流程图

下表列出了状态切换所需要的时间，表格中左边列出的为起始状态。

表 6.状态切换时间表

	目标状态					
起始状态	**SLEEP**	**READY**	**RFS**	**RX**	**TFS**	**TX**
SLEEP		660 us	770 us	820 us	770 us	820 us
READY	立即		110 us	160 us	110 us	160 us
RFS	立即	立即		20 us	不可切换	不可切换
RX	立即	立即	立即		不可切换	160 us
TFS	立即	立即	不可切换	不可切换		20 us
TX	立即	立即	不可切换	160 us	立即	

备注：

在 Direct 模式下，如芯片正在发射，收到切换命令会立即退出 TX 状态。

在 Packet 模式下，如芯片正在发射，必须完成了当前发射才能退出 TX 状态。

3.3 芯片复位机制

CMT2310A 支持 3 种复位：上电复位（POR），硬复位（GPIO5），软复位（SOFTRST）。

上电复位顾名思义就是芯片通电时会进行上电复位。硬复位是通过将 GPIO5 拉高来实现的，硬复位生效并释放后，芯片会回到 IDLE 状态。软复位是通过 SPI 将 0xFF 写入地址 0x7F 来实现的，芯片收到此命令后，会立即进行复位操作，回到 IDLE 状态。软复位不会复位 Page 0, 1 和 2 的所有配置区寄存器，而上电复位和硬复位会复位所有寄存器。

3.4 RFPDK 简介

RFPDK 为 CMOSTEK 提供的用于配置或烧录 RF 芯片的软件工具，在 Windows 环境安装运行。对于 CMT2310A 来说，RFPDK 的作用是通过用户在界面上输入的参数，生成寄存器配置文件。用户可将该

文件导入 MCU 程序来进行寄存器配置。

CMT2310A 的配置思路为：在芯片初始化过程中，用户先通过使用 RFPDK 来生成寄存器配置文件，对芯片进行初始化配置；然后，根据应用需要，用户可以在应用程序中对芯片的某些特定的寄存器进行灵活的配置和控制。因此，用户只需要掌握 RFPDK 操作，并选择性地去理解部分需要使用的寄存器即可。

之前的章节对寄存器进行了简单介绍。本章节会对 RFPDK 进行简单介绍，后续章节将介绍如何结合两者开展配置工作。

RFPDK 主界面如下图所示。

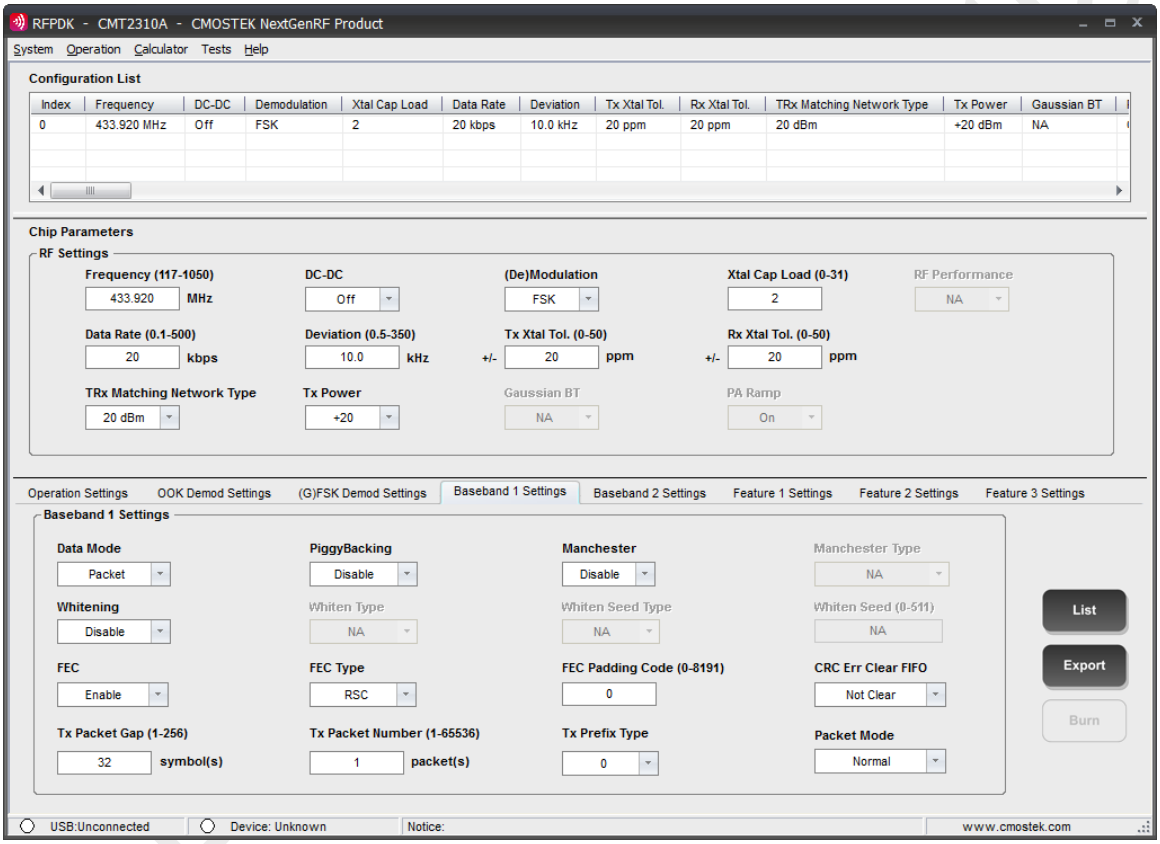

图 17. CMT2310A RFPDK

在此界面上，需要用户输入主要包括 9 个板块：

表 7. RFPDK 的主要板块

RFPDK 板块	配置参数内容	RFPDK 输入参数与寄存器的关系
RF Settings	频率，数据率，功率等 RF 参数	输入参数进行复杂的计算后生成寄存器内容，用户无需理解寄存器内容
Operation Settings	系统运行参数	输入参数与寄存器一一对应，用户可以理解
OOK Demod Settings	OOK 解调参数	输入参数进行复杂的计算后生成寄存器内容，用户无需理解寄存器内容，一般情况用户可使用默认参数

RFPDK 板块	配置参数内容	RFPDK 输入参数与寄存器的关系
FSK Demod Settings	FSK 解调参数	输入参数进行复杂的计算后生成寄存器内容，用户无需理解寄存器内容，一般情况用户可使用默认参数
Baseband 1 Settings	基带编解码和 FIFO 等参数	输入参数与寄存器一一对应，用户可以理解
Baseband 2 Settings	基带包格式参数	输入参数与寄存器一一对应，用户可以理解
Feature 1 Settings	特色功能参数	输入参数与寄存器一一对应，用户可以理解
Feature 2 Settings	特色功能参数	输入参数与寄存器一一对应，用户可以理解
Feature 3 Settings	特色功能参数	输入参数与寄存器一一对应，用户可以理解

以下为生成的配置文件实例。

文件的上部为注释部分，用于确认当前成功生成的配置信息，此部分内容通常用于查错使用，一般情况下用户无需太多关注。

```
;------------------------------------------------------------------------------
;   CMT2310A Configuration File
;   Generated by CMOSTEK RFPDK 1.53_Beta_15
;   2022.01.18 16:54
;------------------------------------------------------------------------------
; Mode                          = Advanced
; Part Number                   = CMT2310A
; Frequency                     = 433.920 MHz
; DC-DC                         = Off
; Demodulation                  = FSK
; Xtal Cap Load                 = 2
; Data Rate                     = 20.0 kbps
; Deviation                     = 10.0 kHz
; Tx Xtal Tol.                  = 20 ppm
; Rx Xtal Tol.                  = 20 ppm
; TRx Matching Network Type     = 20 dBm
; Tx Power                      = +20 dBm
; Gaussian BT                   = NA
; PA Ramp                       = On
; RF Performance                = NA
; Rx Duty-Cycle                 = Off
; Tx Duty-Cycle                 = Off
; Sleep Timer                   = Off
; Sleep Time                    = NA
; Rx Timer                      = Off
; Rx Time T1                    = NA
; Rx Time T2                    = NA
; Rx Exit State                 = STBY
; Tx Exit State                 = STBY
; TX Duty Cycle Persist         = Off
; Packet Done Exit              = On
; TX Duty-Cycle Times           = 0
; SLP Mode                      = Mode 0
; RSSI Valid Source             = RSSI Compare
; PJD Window                    = NA
; RSSI Compare TH               = -127 dBm
; CDR Type                      = Counting
; AFC                           = autosel
; FSK2 Data Map                 = 0:F-low 1:F-high
```

; FSK4 Data Map = NA
; CDR Type = Counting
; Channel BW = autosel
; Baseband BW FSK = autosel
; Data Mode = Packet
; Packet Mode = Normal
; PiggyBacking = Disable
; Manchester = Disable
; Manchester Type = NA
; Whitening = Disable
; Whiten Type = NA
; Whiten Seed Type = NA
; Whiten Seed = NA
; FEC = Disable
; FEC Type = NA
; FEC Padding Code = NA
; crc err clear fifo = Not Clear
; Tx Packet Gap = 32 symbol(s)
; Tx Packet Number = 1 packet(s)
; Tx Prefix Type = 0
; Packet Type = Fixed Length
; Address-Length Position = NA
; Length Size = 1-byte
; Payload Bit Order = Start from msb
; Address Field = Disable
; Preamble Rx Size = 2
; Preamble Tx Size = 8
; Preamble Value = 170
; Preamble Unit = 8-bit
; Sync Size = 3-byte
; Sync Format = S2LP
; Sync Value = 3003605
; Sync Manchester = Disable
; Sync Value Selection = Sync Value
; Sync FEC Value = 3003605
; Sync Tolerance = None
; Address Detect Mode = None
; Address Split Mode = NA
; Address Size = NA
; Address Err Mask = NA
; Address Free = NA
; Dest Addr Value = NA
; Src Addr Value = NA
; Dest Addr Bit Mask = NA
; Src Addr Bit Mask = NA
; Sequence Num = None
; Sequence Num Match = off
; Sequence Num Mode = NA
; Sequence Num Value = NA
; FCS2 = None
; FCS2 Value = NA
; Payload Length = 32
; CRC Options = None
; CRC Swap = NA
; CRC Seed = NA
; CRC Bit Invert = NA
; CRC Range = NA
; CRC Polynomial = NA
; CRC Bit Order = NA

```
; CRC Refin                    = NA
; CRC_Refout                   = NA
; Frequency Hopping Mode       = Mode 2
; Freq Hopping Space           = 10 kHz
; Hopping Channels             = 10
; CSMA Mode                    = Disable
; CSMA RSSI Detection          = NA
; Hopping Persist              = Disable
; Hopping Intermediate State   = TRFS
; CSMA Sleep Timer Random      = NA
; CSMA Rx Time                 = NA
; CSMA Sleep Time M            = NA
; CSMA Sleep Time R            = NA
; CSMA Persist                 = NA
; CSMA Detect Times            = NA
; Tx Auto Hopping              = Disable
; Rx Auto Hopping              = Disable
; Auto Acknowledge             = off
; Auto Resend                  = off
; Maximum Resend Times         = 1
; RSSI Detect Mode             = always
; LFOSC LFXO Sel               = LFOSC(32kHz)
; LF Clock Out                 = off
; Dout Mute                    = disable
; Dout Mute Sel                = NA
; dout adjust mode             = disable
; Dout Adjust Percentage       = NA
; LBD Threshold                = 2.4 v
; Antenna Diversity            = off
; Antenna Switch Mode          = NA
; Collision Detect             = off
; Collision Step               = NA
; RSSI Offset dB               = NA
; RSSI Offset Sel              = autosel
```

文件的后半部分，是用户需要导入 MCU 程序的寄存器配置内容，用 16 进制表示。为用户使用方便，工具会自动将所有寄存器的内容划分成 3 个页，覆盖了第 0，1 和 2 页寄存器的配置区。

3.5 芯片初始化流程

用户需要使用 RFPDK 生成配置区的内容，以备 MCU 用来做初始配置。芯片上电或者复位之后会自动进入 SLEEP 等待 MCU 操作。MCU 可以按照以下初始化流程来操作：

1. 在 MCU 准备好工作后，对 CMT2310 发送一次硬复位或软复位。如果不使用 GPIO5 上的硬复位，在进行软复位后，将 GPIO5_SEL 寄存器配置为 101，让 GPIO5 切换为输出并一直输出低电平。
2. 查询 CHIP_MODE_STA<7:0>寄存器，确认芯片停留在 IDLE 状态（0x00）。
3. 发送 power_up 命令让芯片上电，等待 10ms 后，确认芯片进入了 SLEEP 状态（0x81）。
4. 将 RFPDK 生成的寄存器内容，写入 Page 0 和 1 的配置区。
5. 如需要使用自动跳频，将设计好的跳频表写入 Page 2 的配置区。

初始化完成后，就可以开始工作了。此后无论任何时候要更改全部或者部分的配置寄存器，都需将芯片安全地切换并停留在 SLEEP 或 READY 状态，才可进行配置。

3.6 流程总结

总结上文，用户操作芯片的主要流程包含 3 步：

1. 阅读相关的 AN 文档，使用 RFPDK 生成想要的寄存器文件。
2. 对芯片进行初始化配置流程。
3. 执行应用程序，在应用程序中有两大类操作：
   a) 进行配置区的更改 - 明确需要更改的配置属于配置区中的哪个功能区，用 RFPDK 生成新的配置表，将内容写入对应的寄存器地址。
   b) 控制芯片进行工作 - 明确控制区和中断控制区的寄存器含义，阅读 AN 文档中详解介绍的每一项芯片特性和操作规则，包括 FIFO 控制，IO 和中断控制，RSSI 读取，自动跳频，等等，再对芯片进行符合应用要求的操作。

4 cmt2310a_easy_demo 简介

本章节会详细介绍 CMT2310A 的 Demo 程序，该程序与上文介绍的种种操作紧密结合。程序中调用的函数，其源代码会在第 5 章给出。

4.1 软件层次结构

为了规范 CMT2310A 操作流程，加强可移植性，Demo 程序做了分层处理，每个模块都向下调用相应的 API 函数。整个程序主要分为下面 5 个层：

1. Application：应用层，Demo 中为简单的收发数据包
2. Radio handlers：芯片处理层，包含芯片的初始化，配置，状态控制等流程
3. CMT2310A drivers：芯片驱动层，提供给上层调用
4. Hardware abstraction layer：抽象硬件层，实现芯片的寄存器，FIFO，GPIO 的访问和控制
5. Hardware：硬件层，包括 MCU 提供的 LED，按键，SPI 通讯等资源

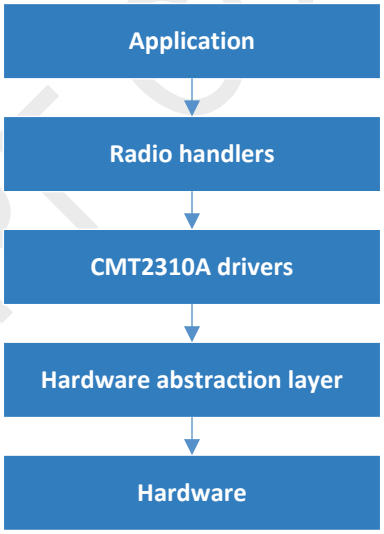

图 18. 软件层次结构图

4.2 软件实现以及调用关系

这里介绍模块相互间的调用关系，系统初始化流程，以及工作流程，其中包括了下面 5 个主要文件：

1. main.c： 应用层实现
2. radio.c： 芯片处理层实现
3. cmt2310a.c： 芯片驱动层实现
4. cmt2310a_hal.c： 抽象硬件层实现
5. cmt_spi4.c： 芯片 SPI 模拟时序实现，分为寄存器和 FIFO 访问时序

下图中画出了前 4 层的文件，并列出了每层执行的函数，cmt_sp4.c 文件负责提供底层 SPI 的函数。

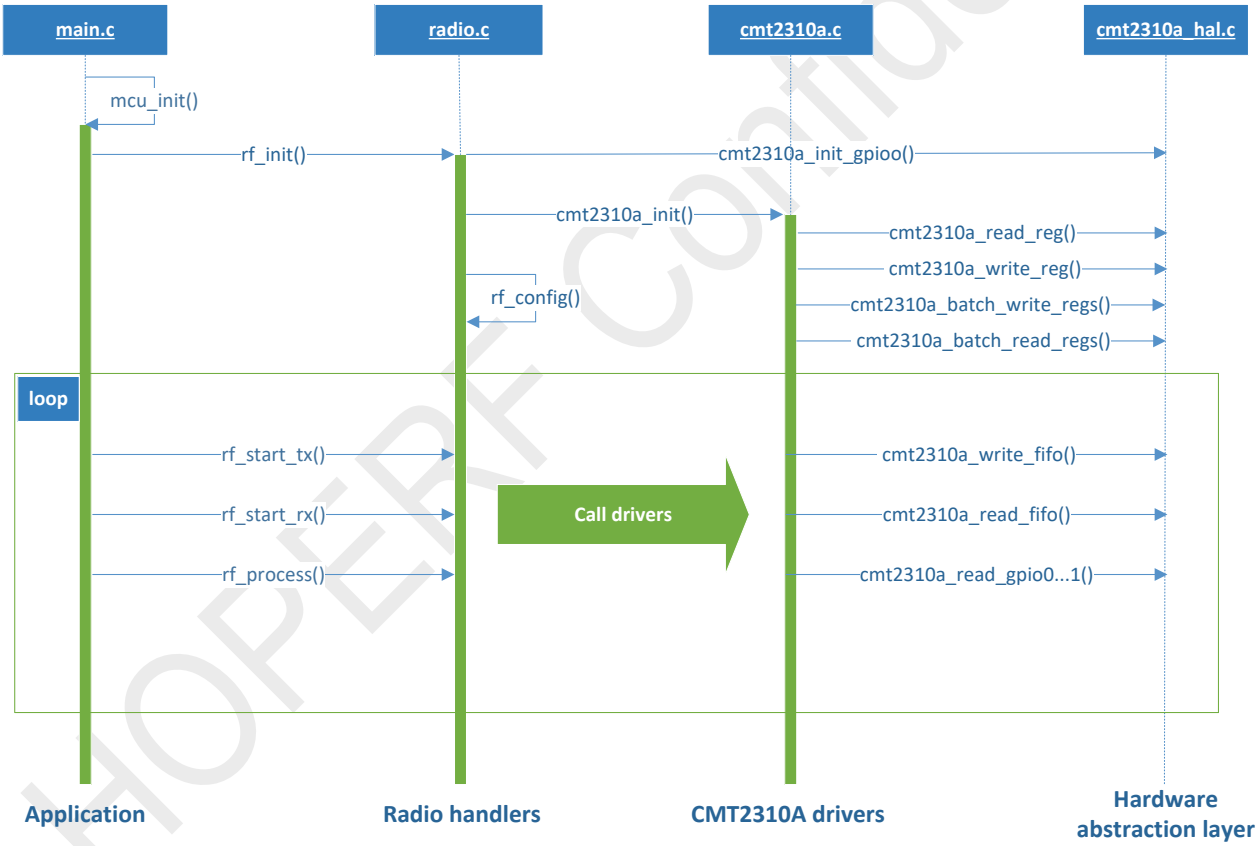

图 19. 软件实现以及调用关系图

4.2.1 CMT2310A 初始化

芯片上电之后必须调用 rf_init()函数，做一次初始化，包括初始化 SPI，GPIO0/1，NIRQ，选择 SPI 模式(3pin 或 4pin)软复位，使能以及关闭一些寄存器。下面是初始化流程图：

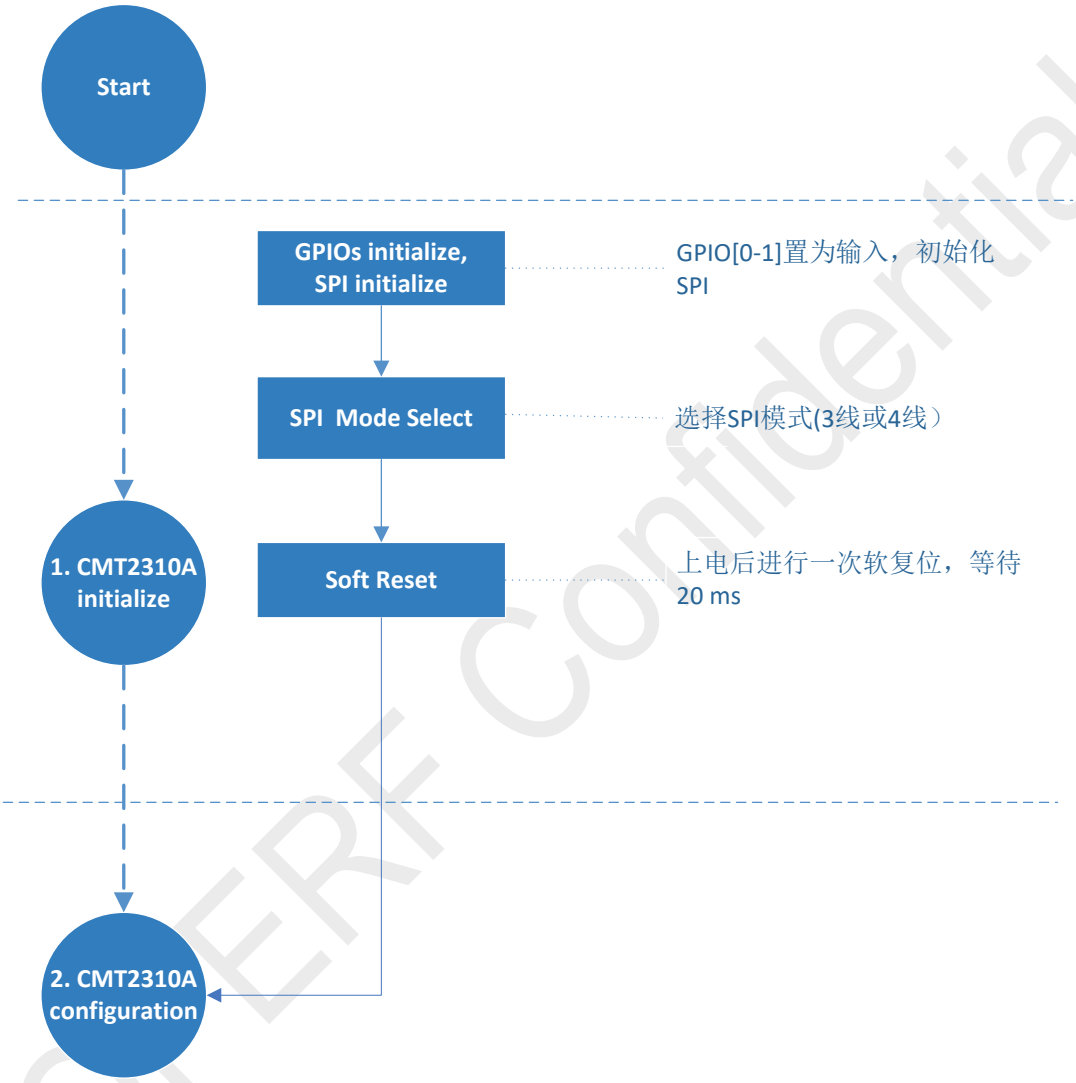

图 20. CMT2310A 初始化流程图

4.2.2 CMT2310A 配置

芯片初始化完成之后，需要调用 rf_config 函数，在 IDLE 状态下，对芯片寄存器配置，再引导固件，然后进入 READY 状态，做 IR 校正，对中断、GPIO 进行配置。下面是配置流程图：

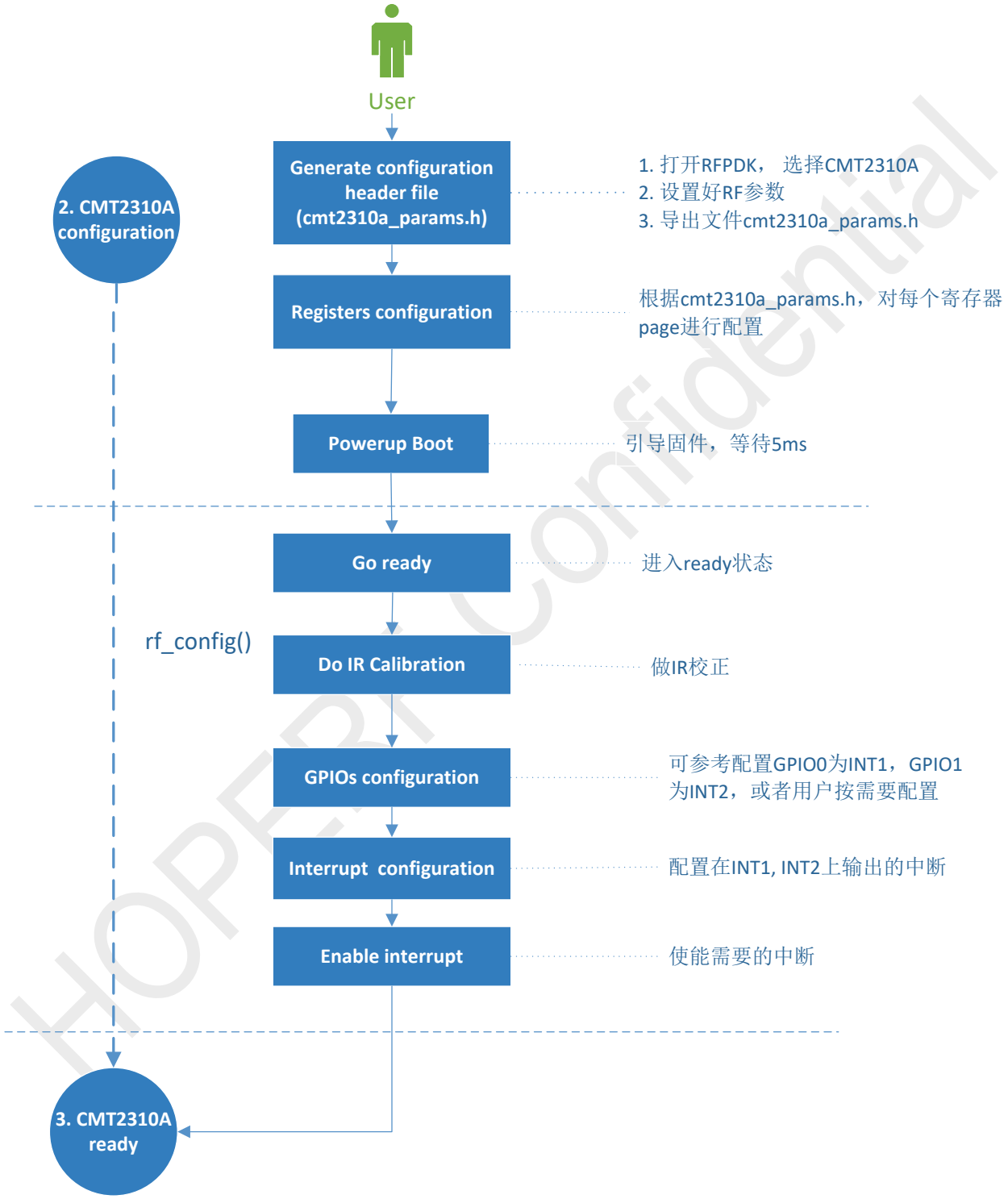

图 21. CMT2310A 配置流程图

4.2.3 CMT2310A 状态处理

芯片配置完成之后，可调用 rf_start_tx()或 rf_start_rx()函数，进入发射或者接收状态，之后循环调用 rf_process()函数来进行通讯处理。rf_process()中使用一个状态机对芯片进行控制，并返回状态结果给应用层进行处理。

rf_process()内部状态：

1. RF_STATE_IDLE：空闲状态
2. RF_STATE_RX_START：接收启动状态，使能读 FIFO，进行接收
3. RF_STATE_RX_WAIT：接收等待状态，不停检测接收完成中断
4. RF_STATE_RX_DONE：接收完成状态，读 FIFO，检查并清除中断
5. RF_STATE_RX_TIMEOUT：接收超时状态，让芯片退出接收
6. RF_STATE_TX_START：发射启动状态，填写 FIFO 数据，进入发射
7. RF_STATE_TX_WAIT：发射等待状态，不停检测发射完成中断
8. RF_STATE_TX_DONE：发射完成状态，检查并清除中断
9. RF_STATE_TX_TIMEOUT：发射超时状态，让芯片退出发射
10. RF_STATE_ERROR：错误状态，芯片无法进入接收或发射，对芯片进行软复位，并重新配置

rf_process()返回结果：

1. RF_IDLE：芯片空闲，可让其进入接收或者发射
2. RF_BUSY：芯片忙碌，当前正在发射或者接收数据
3. RF_RX_DONE：芯片接收完成，上层可处理接收数据
4. RF_RX_TIMEOUT：芯片接收超时
5. RF_TX_DONE：芯片发射完成
6. RF_TX_TIMEOUT：芯片发射超时
7. RF_ERROR：芯片接收或者发射错误

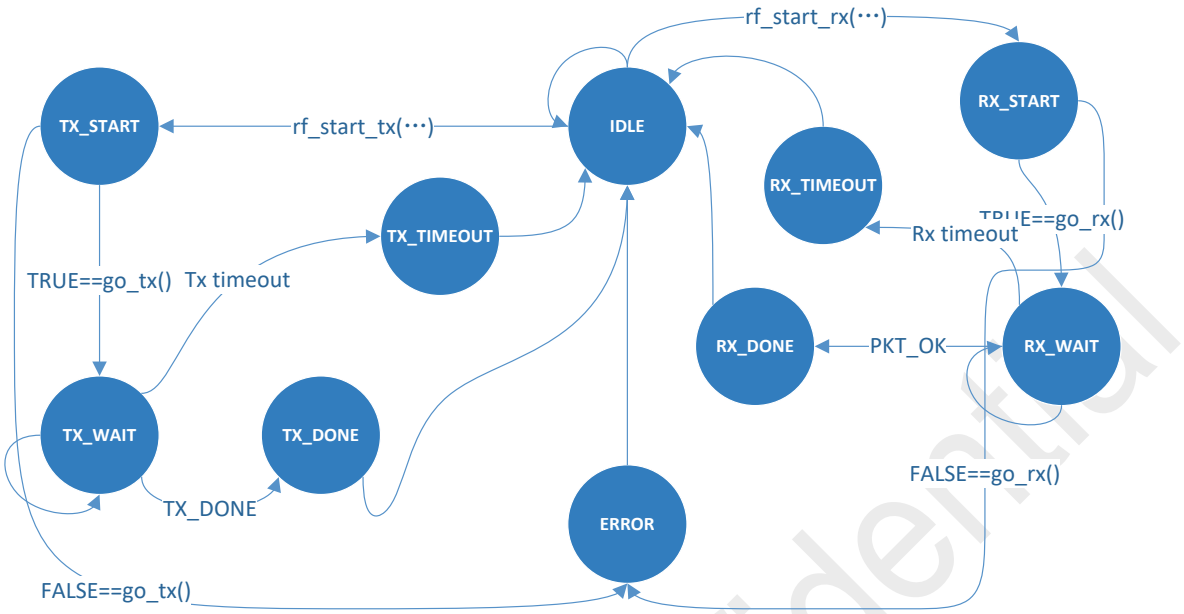

图 22. CMT2310A 状态处理图

4.3 软件目录结构

Demo 程序基于 RFEB 平台，使用 Keil5 IDE 进行开发，有一个完整的工程目录：

1. Libraries： STM32F103 相关库文件
2. MDK-ARM： Keil5 相关工程和编译文件
3. USER： Demo 源程序
4. clear.bat： 执行可清空所有编译中间文件
5. services： 基于 MCU 实现的时间，中断等服务
6. platform： 平台相关的配置，控制等文件
7. periph： LED，按键，LCD，SPI 等外设资源
8. radio： CMT2310A 全部 API 接口文件
9. cmt2310a_params.h： RFPDK 导出的头文件
10. cmt2310a_defs.h： CMT2310A 芯片的寄存器地址宏定义

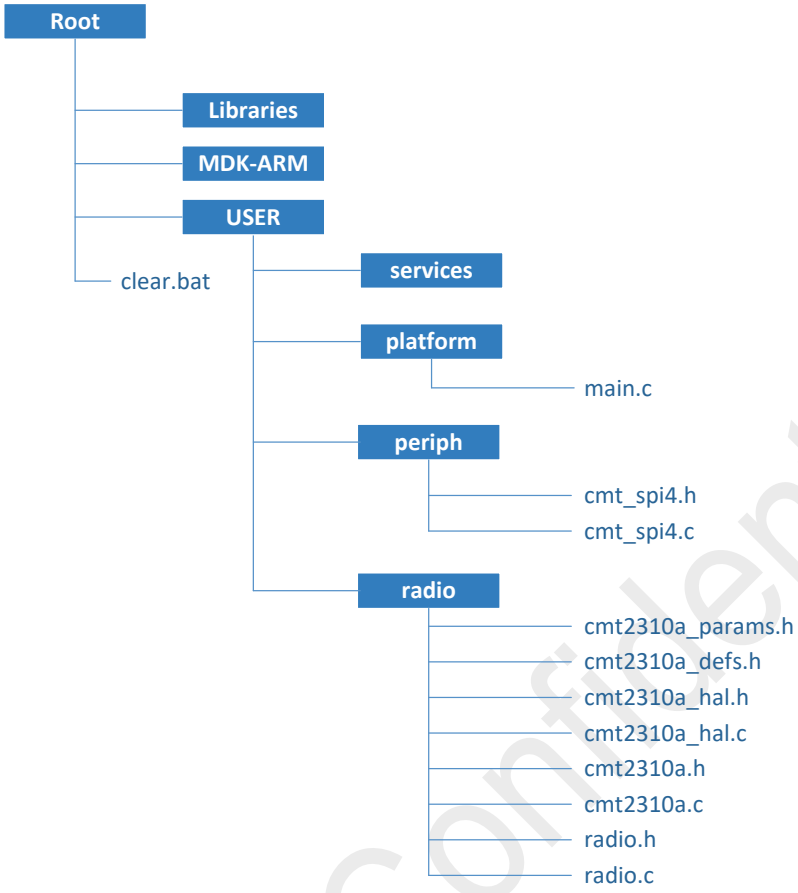

图 23. 软件目录结构图

4.3.1 应用层源代码

main.c 为应用层源代码，配置 g_enable_master 选择主机或者从机，分别调用 on_master()，on_slave() 来进行 CMT2310A 收发通讯。

```
#define RF_PACKET_SIZE 128            /* Define the payload size here */

static u8 g_rx_buffer[RF_PACKET_SIZE];   /* RF Rx buffer */
static u8 g_tx_buffer[RF_PACKET_SIZE];   /* RF Tx buffer */

static BOOL g_enable_master = FALSE;    /* Master/Slave selection */

void mcu_init(void);

/* Manages the master operation */
void on_master(void);

/* Manages the slave operation */
void on_slave(void);
```

4.3.2 模拟 SPI 实现源代码

cmt_spi4.c 是针对 CMT2310A 实现的 SPI 通讯时序，如果需要移植到其他 MCU 平台，要修改下列宏定义。

```
/* ******************************************************************
*  The following need to be modified by user
*  ****************************************************************** */
#define cmt_spi4_csb_out()     SET_GPIO_OUT(CMT_CSB_GPIO)
#define cmt_spi4_miso_out()    SET_GPIO_OUT(CMT_MISO_GPIO)
#define cmt_spi4_miso_in()     SET_GPIO_IN(CMT_MISO_GPIO)
#define cmt_spi4_scl_out()     SET_GPIO_OUT(CMT_SCL_GPIO)
#define cmt_spi4_mosi_out()    SET_GPIO_OUT(CMT_MOSI_GPIO)
#define cmt_spi4_mosi_in()     SET_GPIO_IN(CMT_MOSI_GPIO)

#define cmt_spi4_csb_1()       SET_GPIO_H(CMT_CSB_GPIO)
#define cmt_spi4_csb_0()       SET_GPIO_L(CMT_CSB_GPIO)

#define cmt_spi4_miso_1()      SET_GPIO_H(CMT_MISO_GPIO)
#define cmt_spi4_miso_0()      SET_GPIO_L(CMT_MISO_GPIO)
#define cmt_spi4_miso_read()   READ_GPIO_PIN(CMT_MISO_GPIO)

#define cmt_spi4_scl_1()       SET_GPIO_H(CMT_SCL_GPIO)
#define cmt_spi4_scl_0()       SET_GPIO_L(CMT_SCL_GPIO)

#define cmt_spi4_mosi_1()      SET_GPIO_H(CMT_MOSI_GPIO)
#define cmt_spi4_mosi_0()      SET_GPIO_L(CMT_MOSI_GPIO)
#define cmt_spi4_mosi_read()   READ_GPIO_PIN(CMT_MOSI_GPIO)
/* ****************************************************************** */
```

4.3.3 抽象硬件层源代码

cmt2310a_hal.c 为抽象硬件层源代码，实现 CMT2310A 寄存器，FIFO，GPIO 访问接口。

```
/*! ********************************************************
* @name    cmt2310a_init_gpio
* @desc    Initializes the cmt2310a interface GPIOs.
* ********************************************************/
void cmt2310a_init_gpio(void);

/*! ********************************************************
```

```
* @name    cmt2310a_read_reg
* @desc    Read the cmt2310a register at the specified address.
* @param   addr: register address
* @return  Register value
* ********************************************************/
u8 cmt2310a_read_reg(u8 addr);

/*! ********************************************************
* @name    cmt2310a_write_reg
* @desc    Write the cmt2310a register at the specified address.
* @param   addr: register address
*         dat: register value
* ********************************************************/
void cmt2310a_write_reg(u8 addr, u8 dat);

/*! ********************************************************
* @name    cmt2310a_batch_read_regs
* @desc    Reads the contents of the cmt2310a regs(
          page0: register address start from 0x28,
          page1: register address start from 0x0).
* @param   buf: buffer where to copy the regs read data
*         len: number of bytes to be read from the regs
* ********************************************************/
void cmt2310a_batch_read_regs(u8 buf[], u16 len);

/*! ********************************************************
* @name    cmt2310a_batch_write_regs
* @desc    Writes the buffer contents to the cmt2310a regs(
          page0: register address start from 0x28,
          page1: register address start from 0x00).
* @param   buf: buffer containing data to be put on the regs
*         len: number of bytes to be written to the regs
* ********************************************************/
void cmt2310a_batch_write_regs(const u8 buf[], u16 len);

/*! ********************************************************
* @name    cmt2310a_read_fifo
* @desc    Reads the contents of the cmt2310a FIFO.
* @param   buf: buffer where to copy the FIFO read data
*         len: number of bytes to be read from the FIFO
* ********************************************************/
void cmt2310a_read_fifo(u8 buf[], u16 len);
```

```
/*! ********************************************************
* @name    cmt2310a_write_fifo
* @desc    Writes the buffer contents to the cmt2310a FIFO.
* @param   buf: buffer containing data to be put on the FIFO
*         len: number of bytes to be written to the FIFO
* ********************************************************/
void cmt2310a_write_fifo(const u8 buf[], u16 len);
```

如果需要将 Demo 程序移植到其他 MCU 平台，需要修改 cmt2310a_hal.h 中的一些宏定义。

```
/* *****************************************************************************
*  The following need to be modified by user
*  ***************************************************************************** */
#define cmt2310a_set_gipo0_in()        SET_GPIO_IN(CMT_GPIO0_GPIO)
#define cmt2310a_set_gipo1_in()        SET_GPIO_IN(CMT_GPIO1_GPIO)
#define cmt2310a_set_nirq_in()         SET_GPIO_IN(CMT_NIRQ_GPIO)
#define cmt2310a_read_gpio0()          READ_GPIO_PIN(CMT_GPIO0_GPIO)
#define cmt2310a_read_gpio1()          READ_GPIO_PIN(CMT_GPIO1_GPIO)
#define cmt2310a_read_nirq()           READ_GPIO_PIN(CMT_NIRQ_GPIO)
#define cmt2310a_delay_ms(ms)          system_delay_ms(ms)
#define cmt2310a_delay_us(us)          system_delay_us(us)
#define cmt2310a_get_tick_count()      g_nSysTickCount
/* ***************************************************************************** */
```

4.3.4 芯片驱动层源代码

cmt2310a.c 为芯片驱动层源代码，提供芯片状态切换操作，中断，GPIO，FIFO 操作，通用寄存器配置和访问等。此部分属于固化的程序，和 MCU 平台无关，用户可不用对其修改。

```
/*! ********************************************************
* @name    cmt2310a_init
* @desc    Initialize chip status.
* ********************************************************/
void cmt2310a_init(void)
{
#ifdef SPI_3W_EN  /* cmt_spi4.h */
    cmt2310a_select_spi_wire_mode(1);
#endif
    cmt2310a_soft_reset();
    cmt2310a_delay_ms(20);
}
```

4.3.5 芯片处理层源代码

radio.c 为芯片处理层源代码。

```
/* RF state machine */
typedef enum {
    RF_STATE_IDLE = 0,
    RF_STATE_RX_START,
    RF_STATE_RX_WAIT,
    RF_STATE_RX_DONE,
    RF_STATE_RX_TIMEOUT,
    RF_STATE_TX_START,
    RF_STATE_TX_WAIT,
    RF_STATE_TX_DONE,
    RF_STATE_TX_TIMEOUT,
    RF_STATE_ERROR,
} EnumRFStatus;

/* RF process function results */
typedef enum {
    RF_IDLE = 0,
    RF_BUSY,
    RF_RX_DONE,
    RF_RX_TIMEOUT,
    RF_TX_DONE,
    RF_TX_TIMEOUT,
    RF_ERROR,
} EnumRFResult;

//#define ENABLE_ANTENNA_SWITCH
```

5 附录

5.1 附录 1：Sample Code -SPI 读写操作代码示例

```
void cmt_spi4_send(u8 data8)
{
    u8 i;

    for(i=0; i<8; i++)
    {
        cmt_spi4_scl_0();

        /* Send byte on the rising edge of SCL */
        if(data8 & 0x80)
            cmt_spi4_mosi_1();
        else
            cmt_spi4_mosi_0();

        cmt_spi4_delay();

        data8 <<= 1;
        cmt_spi4_scl_1();
        cmt_spi4_delay();
    }
}

u8 cmt_spi4_recv(void)
{
    u8 i;
    u8 data8 = 0xFF;

    for(i=0; i<8; i++)
    {
        cmt_spi4_scl_0();
        cmt_spi4_delay();
        data8 <<= 1;

        cmt_spi4_scl_1();

        /* Read byte on the rising edge of SCL */
#ifndef SPI_3W_EN
```

```
        if(cmt_spi4_miso_read())
#else
        if(cmt_spi4_mosi_read())
#endif
            data8 |= 0x01;
        else
            data8 &= ~0x01;

        cmt_spi4_delay();
    }

    return data8;
}

void cmt_spi4_write(u8 addr, const u8* p_buf, u16 len)
{
    u16 i;

    cmt_spi4_mosi_1();
    cmt_spi4_mosi_out();
    cmt_spi4_mosi_1();

    cmt_spi4_scl_0();
    cmt_spi4_scl_out();
    cmt_spi4_scl_0();

    cmt_spi4_csb_0();
    cmt_spi4_csb_out();
    cmt_spi4_csb_0();

    /* > 0.5 SCL cycle */
    cmt_spi4_delay();
    cmt_spi4_delay();

    /* r/w = 0 */
    cmt_spi4_send(addr&0x7F);

    for(i = 0; i < len; i++)
    {
        cmt_spi4_send(p_buf[i]);
    }

    cmt_spi4_scl_0();
```

```
    /* > 0.5 SCL cycle */
    cmt_spi4_delay();
    cmt_spi4_delay();

    cmt_spi4_csb_1();

    cmt_spi4_mosi_1();
#ifdef SPI_3W_EN
    cmt_spi4_mosi_in();
#endif
}

void cmt_spi4_read(u8 addr, u8* p_buf, u16 len)
{
    u16 i;

    cmt_spi4_mosi_1();
    cmt_spi4_mosi_out();
    cmt_spi4_mosi_1();
#ifndef SPI_3W_EN
    cmt_spi4_miso_1();
    cmt_spi4_miso_in();
    cmt_spi4_miso_1();
#endif
    cmt_spi4_scl_0();
    cmt_spi4_scl_out();
    cmt_spi4_scl_0();

    cmt_spi4_csb_0();
    cmt_spi4_csb_out();
    cmt_spi4_csb_0();

    /* > 0.5 SCL cycle */
    cmt_spi4_delay();
    cmt_spi4_delay();

    /* r/w = 1 */
    cmt_spi4_send(addr|0x80);
#ifdef SPI_3W_EN
    cmt_spi4_mosi_in();
    cmt_spi4_delay();
#endif
```

```
    for(i = 0; i < len; i++)
    {
        p_buf[i] = cmt_spi4_recv();
    }

    cmt_spi4_scl_0();

    /* > 0.5 SCL cycle */
    cmt_spi4_delay();
    cmt_spi4_delay();

    cmt_spi4_csb_1();

    cmt_spi4_mosi_1();

}
```

5.2 附录 2：Sample Code -状态切换库函数代码示例

```
/*! ********************************************************
* @name    cmt2310a_go_sleep
* @desc    Entry SLEEP mode.
* @return  TRUE or FALSE
* ********************************************************/
BOOL cmt2310a_go_sleep(void)
{
    return cmt2310a_auto_switch_state(M_GO_SLEEP);
}

/*! ********************************************************
* @name    cmt2310a_go_ready
* @desc    Entry ready mode.
* @return  TRUE or FALSE
* ********************************************************/
BOOL cmt2310a_go_ready(void)
{
    return cmt2310a_auto_switch_state(M_GO_READY);
}

/*! ********************************************************
* @name    cmt2310a_go_txfs
* @desc    Entry TFS mode.
```

```
* @return  TRUE or FALSE
* ******************************************************/
BOOL cmt2310a_go_txfs(void)
{
    return cmt2310a_auto_switch_state(M_GO_TXFS);
}

/*! ******************************************************
* @name    cmt2310a_go_rxfs
* @desc    Entry RFS mode.
* @return  TRUE or FALSE
* ******************************************************/
BOOL cmt2310a_go_rxfs(void)
{
    return cmt2310a_auto_switch_state(M_GO_RXFS);
}

/*! ******************************************************
* @name    cmt2310a_go_tx
* @desc    Entry Tx mode.
* @return  TRUE or FALSE
* ******************************************************/
BOOL cmt2310a_go_tx(void)
{
    return cmt2310a_auto_switch_state(M_GO_TX);
}

/*! ******************************************************
* @name    cmt2310a_go_rx
* @desc    Entry Rx mode.
* @return  TRUE or FALSE
* ******************************************************/
BOOL cmt2310a_go_rx(void)
{
    return cmt2310a_auto_switch_state(M_GO_RX);
}
```


5.3 附录 3：SampleCode - 初始化函数代码示例

```
void rf_init(void)
{
    cmt2310a_init_gpio();
```

```
    cmt2310a_init();

      cmt2310a_go_sleep();
    cmt2310a_delay_ms(2);

    rf_config();
}

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_powerup_boot();
    cmt2310a_delay_ms(5);

    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);
}
```

6 文档变更记录

表 8.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
0.6A	所有	审阅校对	2022-01-09
0.6B	所有	1. SCL 描述改为 SCLK； 2. 1.2 章节补充芯片管脚图； 3. 2.2 章节 0x7B 为 BRW 操作端口； 4. 2.3 章节图例中操作 FIFO 端口为 0x7A； 5. 2 章 SPI 时序图修正；	2022-01-19
0.7	所有	审校	2022-08-12

7 联系方式

深圳市华普微电子股份有限公司
中国广东省深圳市南山区西丽街道万科云城三期 8A 栋 30 层
邮编： 518052
电话： +86 - 755 - 82973805
销售： sales@hoperf.com
网址： www.hoperf.cn