

CMT2300A 快速上手指南

概要

本应用文档为使用 CMT2300A 进行产品开发的用户提供基本的使用方法和相关寄存器的介绍，以使用户在使用的过程中方便的查阅各个寄存器的说明及用法。

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 | 140 - 1020 MHz | (G)FSK/OOK | 收发一体 | 寄存器 | QFN16 |

目录

概要 1

| | | |
|----------|-----------------------------------|-----------|
| 1 | 芯片架构介绍 | 4 |
| 1.1 | 总体工作原理 | 4 |
| 1.2 | IO 管脚说明 | 5 |
| 2 | SPI 接口时序 | 6 |
| 2.1 | 读/写寄存器操作 | 6 |
| 2.2 | 读/写 FIFO 操作 | 6 |
| 3 | 配置和控制机制 | 8 |
| 3.1 | 寄存器概览 | 8 |
| 3.2 | 工作状态切换 | 10 |
| 3.3 | 软复位 (Softrst) | 13 |
| 3.4 | RFPDK 简介 | 13 |
| 3.5 | 芯片初始化流程 | 17 |
| 3.6 | 配置区的功能划分 | 17 |
| 3.6.1 | CMT 区 (0x00 – 0x0B) | 18 |
| 3.6.2 | 系统区 (0x0C – 0x17) | 19 |
| 3.6.3 | 频率区 (0x18 – 0x1F) | 19 |
| 3.6.4 | 数据率区 (0x20 – 0x37) | 19 |
| 3.6.5 | 基带区 (0x38 – 0x54) | 20 |
| 3.6.6 | 发射区 (0x55 – 0x5F) | 20 |
| 3.7 | 控制区的使用简介 | 21 |
| 3.8 | 流程总结 | 21 |
| 4 | CMT2300A_DemoEasy 简介 | 22 |
| 4.1 | 软件层次结构 | 22 |
| 4.2 | 软件实现以及调用关系 | 22 |
| 4.2.1 | CMT2300A 初始化 | 23 |
| 4.2.2 | CMT2300A 配置 | 24 |
| 4.2.3 | CMT2300 状态处理 | 25 |
| 4.3 | 软件目录结构 | 26 |
| 4.3.1 | 应用层源代码 | 27 |
| 4.3.2 | 模拟 SPI 实现源代码 | 28 |
| 4.3.3 | 抽象硬件层源代码 | 29 |
| 4.3.4 | 芯片驱动层源代码 | 30 |
| 4.3.5 | 芯片处理层源代码 | 31 |
| 5 | 附录 | 32 |

| | | |
|-----|---|----|
| 5.1 | 附录 1 sample code SPI 读写操作代码示例 | 32 |
| 5.2 | 附录 2 Sample code SPI 读写 FIFO 操作代码示例 | 33 |
| 5.3 | 附录 3 Sample code 状态切换库函数代码示例 | 36 |
| 5.4 | 附录 4 Sample code 初始化函数代码示例 | 37 |
| 6 | 文档变更记录 | 38 |
| 7 | 联系方式..... | 39 |

1 芯片架构介绍

1.1 总体工作原理

CMT2300A 是一款数字模拟一体化收发机产品。该产品采用 26 MHz 的晶体提供 PLL 的参考频率和数字时钟，同时支持 OOK 和 (G) FSK 的调制解调模式，并支持 Direct 和 Packet 两种数据处理模式。

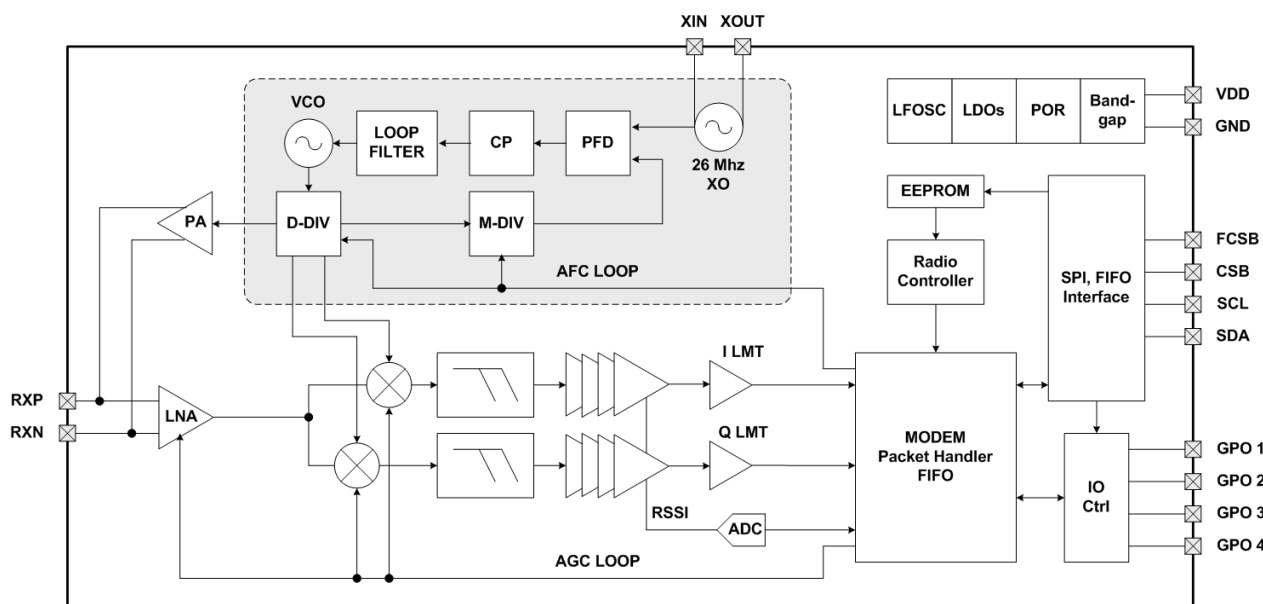


图 1. CMT2300A 系统框图

在接收机部分，该芯片采用 LNA+MIXER+IFFILTER+LIMITTER+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 Limiter 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 (G) FSK 解调。同时，会通过 SARADC 将实时的 RSSI 转换为 8-bit 的数字信号，并送给数字部分做后续的 OOK 解调和其它处理。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 PAD。

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。

芯片提供了 SPI 通讯口，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，控制主控状态机，并访问 FIFO。

1.2 IO 管脚说明

下面以 QFN16 的封装为例，说明 CMT2300A 的管脚分配和功能：

表 2. CMT2300A 管脚描述

| 管脚号 | 名称 | 类型 | I/O | 功能说明 |
|--|-------|----|-----|---|
| 1 | RFIP | 模拟 | I | RF 信号输入 P |
| 2 | RFIN | 模拟 | I | RF 信号输入 N |
| 3 | PA | 模拟 | O | PA 输出 |
| 4 | AVDD | 模拟 | IO | 模拟 VDD |
| 5 | AGND | 模拟 | IO | 模拟 GND |
| 6 | DGND | 模拟 | IO | 数字 GND |
| 7 | DVDD | 模拟 | IO | 数字 VDD |
| 8 | GPIO3 | 数字 | O | 可配置为：CLKO，DOUT/DIN，INT2，DCLK (TX/RX) |
| 9 | SCLK | 数字 | I | SPI 的时钟 |
| 10 | SDIO | 数字 | IO | SPI 的数据输入和输出 |
| 11 | CSB | 数字 | I | SPI 访问寄存器的片选 |
| 12 | FCSB | 数字 | I | SPI 访问 FIFO 的片选 |
| 13 | XI | 模拟 | I | 晶体电路输入 |
| 14 | XO | 模拟 | O | 晶体电路输出 |
| 15 | GPIO2 | 数字 | O | 可配置为：INT1，INT2，DOUT/DIN，DCLK (TX/RX)，RF_SWT |
| 16 | GPIO1 | 数字 | O | 可配置为：DOUT/DIN，INT1，INT2，DCLK (TX/RX)，RF_SWT |
| 备注： INT1 和 INT2 是中断 DOUT 是解调输出 DIN 是调制输入 DCLK 是调制或者解调数据率同步时钟，在 TX/RX 模式切换时自动切换 RSTin 是外部硬复位，跟发送软复位命令一样效果 | | | | |

2 SPI 接口时序

芯片是通过 4-线的 SPI 口与外部进行通信的。低有效的 CSB 是用于访问寄存器的片选信号。低有效的 FCSB 是用于访问 FIFO 的片选信号。两者不能同时设为低。SCL 是串口时钟，最快速度可以到 5MHz。无论对于芯片本身，还是外部的 MCU，都是在 SCL 的下降沿送出数据，在上升沿采集数据。SDA 是一个双向的脚，用于输入和输出数据。地址和数据部分都是从 MSB 开始传送。

2.1 读/写寄存器操作

当访问寄存器的时候，CSB 要拉低。然后首先发送一个 R/W 位，后面跟着 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 SCL 周期，才能开始发送 R/W 位。在 MCU 发送出最后一个 SCL 的下降沿之后，必须等待至少半个 SCL 周期，再把 CSB 拉高。

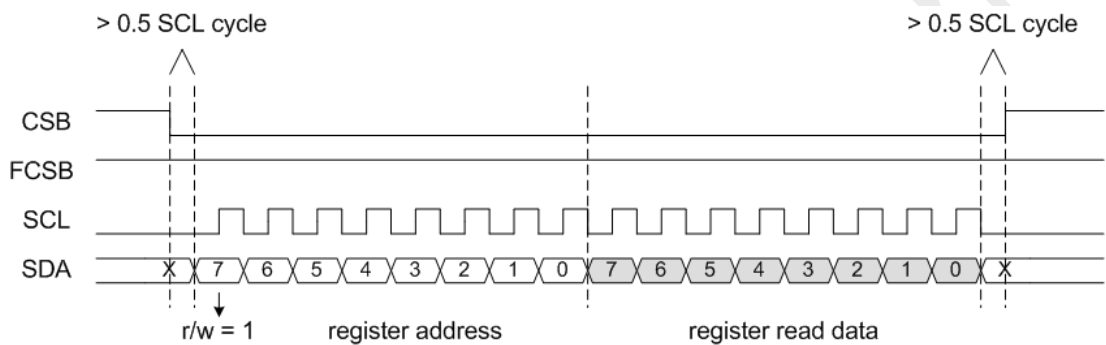


图 2. SPI 读寄存器时序

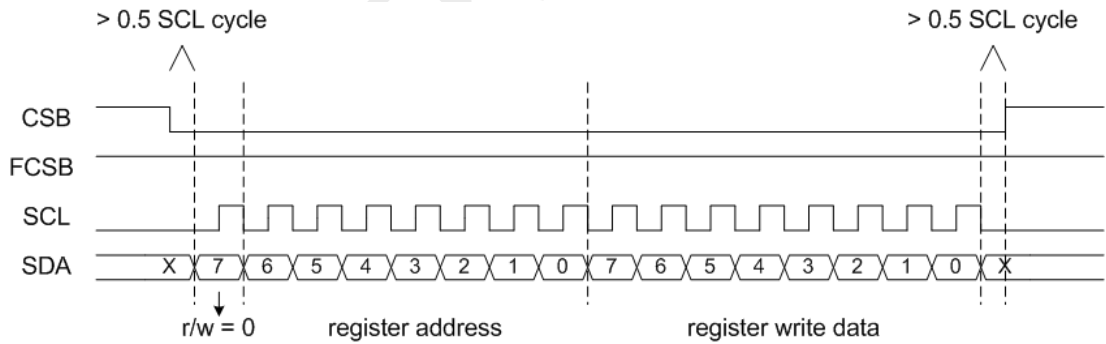


图 3. SPI 写寄存器时序

SPI 读取操作代码示例详见附录 1。

2.2 读/写 FIFO 操作

在 MCU 需要访问 FIFO 的时候，首先要将配置一些寄存器，来设置好 FIFO 的读/写模式，以及其它一些工作模式，这将会在后面的章节介绍，这里给出的是确定模式后，读写的时序图。需要注意的是 FCSB 的控制和访问寄存器时对 CSB 的控制略有差异。开始访问的时候，FCSB 要先拉低 1 个时钟周期后，再送出 SCL 的上升沿。在送出最后一个 SCL 的下降沿后，要过至少 2us 再将 FCSB 拉高。两次连续的读写操作之间，FCSB 必须拉高至少 4us。在进行写 FIFO 时，第一个 bit 的数据必须在第一个 SCL 的上升沿送出前 0.5 个时钟周期准备

好。

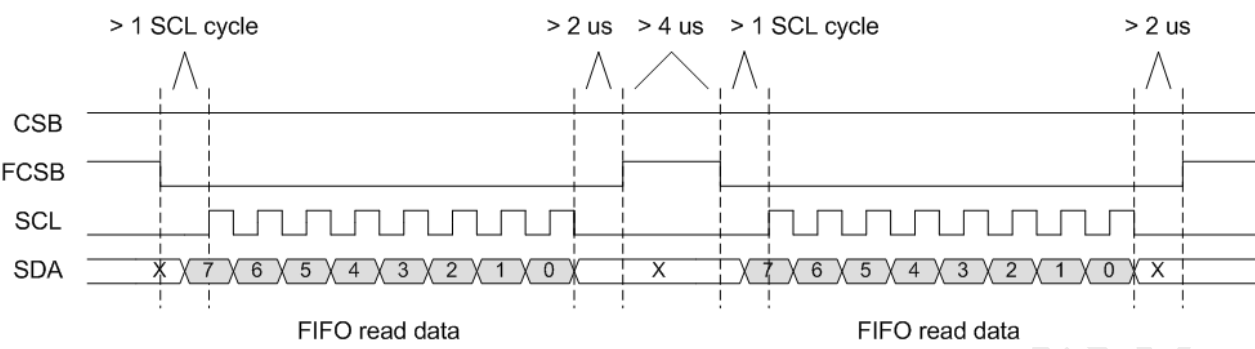


图 4. SPI 读 FIFO 时序

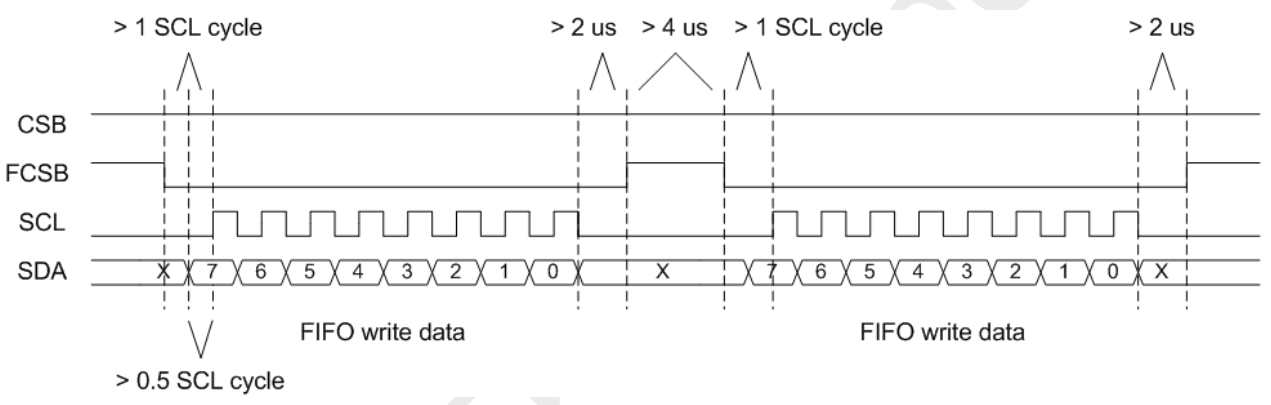


图 5. SPI 写 FIFO 时序

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

表 3. CMT2300A 寄存器概览表

| Addr | R/W | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Function |
|------|-----|----------------|---------------------|-------|-------|-------|-------|-------|-------|-------|----------|
| 0x00 | RW | CUS_CMT1 | 用户无须理解，直接用RFPDK生成导入 | | | | | | | | CMT区 |
| 0x01 | RW | CUS_CMT2 | | | | | | | | | |
| 0x02 | RW | CUS_CMT3 | | | | | | | | | |
| 0x03 | RW | CUS_CMT4 | | | | | | | | | |
| 0x04 | RW | CUS_CMT5 | | | | | | | | | |
| 0x05 | RW | CUS_CMT6 | | | | | | | | | |
| 0x06 | RW | CUS_CMT7 | | | | | | | | | |
| 0x07 | RW | CUS_CMT8 | | | | | | | | | |
| 0x08 | RW | CUS_CMT9 | 系统区 | | | | | | | | |
| 0x09 | RW | CUS_CMT10 | | | | | | | | | |
| 0x0A | RW | CUS_CMT11 | | | | | | | | | |
| 0x0B | RW | CUS_RSSI | | | | | | | | | |
| 0x0C | RW | CUS_SYS1 | | | | | | | | | |
| 0x0D | RW | CUS_SYS2 | | | | | | | | | |
| 0x0E | RW | CUS_SYS3 | | | | | | | | | |
| 0x0F | RW | CUS_SYS4 | | | | | | | | | |
| 0x10 | RW | CUS_SYS5 | 系统区 | | | | | | | | |
| 0x11 | RW | CUS_SYS6 | | | | | | | | | |
| 0x12 | RW | CUS_SYS7 | | | | | | | | | |
| 0x13 | RW | CUS_SYS8 | | | | | | | | | |
| 0x14 | RW | CUS_SYS9 | | | | | | | | | |
| 0x15 | RW | CUS_SYS10 | | | | | | | | | |
| 0x16 | RW | CUS_SYS11 | | | | | | | | | |
| 0x17 | RW | CUS_SYS12 | | | | | | | | | |
| 0x18 | RW | CUS_RF1 | 用户无须理解，直接用RFPDK生成导入 | | | | | | | | 频率区 |
| 0x19 | RW | CUS_RF2 | | | | | | | | | |
| 0x1A | RW | CUS_RF3 | | | | | | | | | |
| 0x1B | RW | CUS_RF4 | | | | | | | | | |
| 0x1C | RW | CUS_RF5 | | | | | | | | | |
| 0x1D | RW | CUS_RF6 | | | | | | | | | |
| 0x1E | RW | CUS_RF7 | | | | | | | | | |
| 0x1F | RW | CUS_RF8 | | | | | | | | | |
| 0x20 | RW | CUS_RF9 | 用户无须理解，直接用RFPDK生成导入 | | | | | | | | 数据率区 |
| 0x21 | RW | CUS_RF10 | | | | | | | | | |
| 0x22 | RW | CUS_RF11 | | | | | | | | | |
| 0x23 | RW | CUS_RF12 | | | | | | | | | |
| 0x24 | RW | CUS_FSK1 | | | | | | | | | |
| 0x25 | RW | CUS_FSK2 | | | | | | | | | |
| 0x26 | RW | CUS_FSK3 | | | | | | | | | |
| 0x27 | RW | CUS_FSK4 | | | | | | | | | |
| 0x28 | RW | CUS_FSK5 | 用户无须理解，直接用RFPDK生成导入 | | | | | | | | 基带区 |
| 0x29 | RW | CUS_FSK6 | | | | | | | | | |
| 0x2A | RW | CUS_FSK7 | | | | | | | | | |
| 0x2B | RW | CUS_CDR1 | | | | | | | | | |
| 0x2C | RW | CUS_CDR2 | | | | | | | | | |
| 0x2D | RW | CUS_CDR3 | | | | | | | | | |
| 0x2E | RW | CUS_CDR4 | | | | | | | | | |
| 0x2F | RW | CUS_AGC1 | | | | | | | | | |
| 0x30 | RW | CUS_AGC2 | 基带区 | | | | | | | | |
| 0x31 | RW | CUS_AGC3 | | | | | | | | | |
| 0x32 | RW | CUS_AGC4 | | | | | | | | | |
| 0x33 | RW | CUS_OOK1 | | | | | | | | | |
| 0x34 | RW | CUS_OOK2 | | | | | | | | | |
| 0x35 | RW | CUS_OOK3 | | | | | | | | | |
| 0x36 | RW | CUS_OOK4 | | | | | | | | | |
| 0x37 | RW | CUS_OOK5 | | | | | | | | | |
| 0x38 | RW | CUS_PKT1 | 基带区 | | | | | | | | |
| 0x39 | RW | CUS_PKT2 | | | | | | | | | |
| 0x3A | RW | CUS_PKT3 | | | | | | | | | |
| 0x3B | RW | CUS_PKT4 | | | | | | | | | |
| 0x3C | RW | CUS_PKT5 | | | | | | | | | |
| 0x3D | RW | CUS_PKT6 | | | | | | | | | |
| 0x3E | RW | CUS_PKT7 | | | | | | | | | |
| 0x3F | RW | CUS_PKT8 | | | | | | | | | |
| 0x40 | RW | CUS_PKT9 | 基带区 | | | | | | | | |
| 0x41 | RW | CUS_PKT10 | | | | | | | | | |
| 0x42 | RW | CUS_PKT11 | | | | | | | | | |
| 0x43 | RW | CUS_PKT12 | | | | | | | | | |
| 0x44 | RW | CUS_PKT13 | | | | | | | | | |
| 0x45 | RW | CUS_PKT14 | | | | | | | | | |
| 0x46 | RW | CUS_PKT15 | | | | | | | | | |
| 0x47 | RW | CUS_PKT16 | | | | | | | | | |
| 0x48 | RW | CUS_PKT17 | 基带区 | | | | | | | | |
| 0x49 | RW | CUS_PKT18 | | | | | | | | | |
| 0x4A | RW | CUS_PKT19 | | | | | | | | | |
| 0x4B | RW | CUS_PKT20 | | | | | | | | | |
| 0x4C | RW | CUS_PKT21 | | | | | | | | | |
| 0x4D | RW | CUS_PKT22 | | | | | | | | | |
| 0x4E | RW | CUS_PKT23 | | | | | | | | | |
| 0x4F | RW | CUS_PKT24 | | | | | | | | | |
| 0x50 | RW | CUS_PKT25 | 基带区 | | | | | | | | |
| 0x51 | RW | CUS_PKT26 | | | | | | | | | |
| 0x52 | RW | CUS_PKT27 | | | | | | | | | |
| 0x53 | RW | CUS_PKT28 | | | | | | | | | |
| 0x54 | RW | CUS_PKT29 | | | | | | | | | |
| 0x55 | RW | CUS_TX1 | | | | | | | | | |
| 0x56 | RW | CUS_TX2 | | | | | | | | | |
| 0x57 | RW | CUS_TX3 | | | | | | | | | |
| 0x58 | RW | CUS_TX4 | 用户无须理解，直接用RFPDK生成导入 | | | | | | | | 发射区 |
| 0x59 | RW | CUS_TX5 | | | | | | | | | |
| 0x5A | RW | CUS_TX6 | | | | | | | | | |
| 0x5B | RW | CUS_TX7 | | | | | | | | | |
| 0x5C | RW | CUS_TX8 | | | | | | | | | |
| 0x5D | RW | CUS_TX9 | | | | | | | | | |
| 0x5E | RW | CUS_TX10 | | | | | | | | | |
| 0x5F | RW | CUS_LBD | | | | | | | | | |
| 0x60 | RW | CUS_MODE_CTL | 控制区1 | | | | | | | | |
| 0x61 | RW | CUS_MODE_STA | | | | | | | | | |
| 0x62 | RW | CUS_EN_CTL | | | | | | | | | |
| 0x63 | RW | CUS_FREQ_CHNL | | | | | | | | | |
| 0x64 | RW | CUS_FREQ_OFS | | | | | | | | | |
| 0x65 | RW | CUS_IO_SEL | | | | | | | | | |
| 0x66 | RW | CUS_INT1_CTL | | | | | | | | | |
| 0x67 | RW | CUS_INT2_CTL | | | | | | | | | |
| 0x68 | RW | CUS_INT_EN | 控制区2 | | | | | | | | |
| 0x69 | RW | CUS_FIFO_CTL | | | | | | | | | |
| 0x6A | RW | CUS_INT CLR1 | | | | | | | | | |
| 0x6B | W | CUS_INT CLR2 | | | | | | | | | |
| 0x6C | W | CUS_FIFO CLR | | | | | | | | | |
| 0x6D | R | CUS_INT FLAG | | | | | | | | | |
| 0x6E | R | CUS_FIFO FLAG | | | | | | | | | |
| 0x6F | R | CUS_RSSI_CODE | | | | | | | | | |
| 0x70 | R | CUS_RSSI_DBM | 控制区2 | | | | | | | | |
| 0x71 | R | CUS_LBD_RESULT | | | | | | | | | |
| 0x72 | R | CUS_LBD_RESULT | | | | | | | | | |
| 0x73 | R | CUS_LBD_RESULT | | | | | | | | | |
| 0x74 | R | CUS_LBD_RESULT | | | | | | | | | |
| 0x75 | R | CUS_LBD_RESULT | | | | | | | | | |
| 0x76 | R | CUS_LBD_RESULT | | | | | | | | | |
| 0x77 | R | CUS_LBD_RESULT | | | | | | | | | |

外部 MCU 对芯片的配置和控制，都是基于使用 SPI 来访问一系列的寄存器完成的。从表格可以看到，地址从 0x00 到 0x7F，其中可以分成 3 个大区去理解，分别是：配置区（其中包括 6 个子区），控制区 1，还有控制区 2。下面做详细的介绍。

首先，对 3 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，3 个区有不同的作用，如下所示：

表 4. CMT2300A 寄存器区域划分表

| 区域 | 地址 | 功能描述 |
|-------|-------------|---|
| 配置区 | 0x00– 0x5F | <p>这个区域用于对芯片进行配置，可配置的内容包括了：</p> <ol style="list-style-type: none"> 1. CMT 内部使用的隐藏属性 2. 系统运作 3. RF 特性 4. FSK 解调 5. 时钟恢复 6. AGC 特性 7. OOK 解调 8. 包格式和编解码方式 9. TX 的调制特性 <p>寄存器的值既可以来自于 RFPDK，也可以由客户自己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就可以了。</p> |
| 控制区 1 | 0x60 – 0x6A | <p>这个区域用于在应用过程当中实时控制芯片。可控制的内容包括了：</p> <ol style="list-style-type: none"> 1. 芯片状态切换 2. 特殊功能的使能 3. 手动跳频的频道切换 4. IO 控制 5. 中断使能 6. FIFO 的工作模式配置 7. 芯片状态相关中断的控制 <p>控制区 1 在 SLEEP 状态下都是可以保存的，只要电池不断电，芯片不进行复位操作，配置好的内容不会丢失。</p> |
| 控制区 2 | 0x6B – 0x71 | <p>这个区域是提供给客户用来控制芯片的。可控制的内容包括了：</p> <ol style="list-style-type: none"> 1. 包和 FIFO 先关中断的控制 2. FIFO 的控制 3. RSSI 值的读取 4. LBD 功能的控制 <p>控制区 2 在 SLEEP 状态下是不可保存的，所有配置好的值，以及可读取的值在 SLEEP 时都会消失，用户需要注意。</p> |

3.2 工作状态切换

外部 MCU 可以通过访问下面这些控制区 1 的寄存器来切换和查询芯片的工作状态：

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 |
|------------------------|-----|-----|--------------------|---|
| CUS_MODE_CTL (0x60) | 7:0 | RW | CHIP_MODE_SWT<7:0> | 状态切换的命令： 00000010: go_stby 00000100: go_rfs 00001000: go_rx 00010000: go_sleep 00100000: go_tfs 01000000: go_tx 10000000: go_switch 其余值：不允许发送。 |
| CUS_MODE_STA (0x61) | 3:0 | RW | CHIP_MODE_STA<3:0> | 芯片状态： 0000: IDLE 0001: SLEEP 0010: STBY 0011: RFS 0100: TFS 0101: RX 0110: TX 1000: UNLOCKED/LOW_VDD 1001: CAL 其余值：无效 UNLOCKED 状态是指由于芯片受到严重干扰导致了 PLL 失锁，LOW_VDD 是指 LBD 检测出 VDD 过低，不宜进行发射和接收，用户可以选择性地让芯片停留在这个状态，后续有章节详细介绍。CAL 状态就是校正状态，芯片不会长期停留在校正状态，因此用户通常是查询不到 CAL 状态的。 |

状态的切换图如下所示，UNLOCKED 和 CAL 状态都不会出现在下面的图中，因为客户是不可见的。

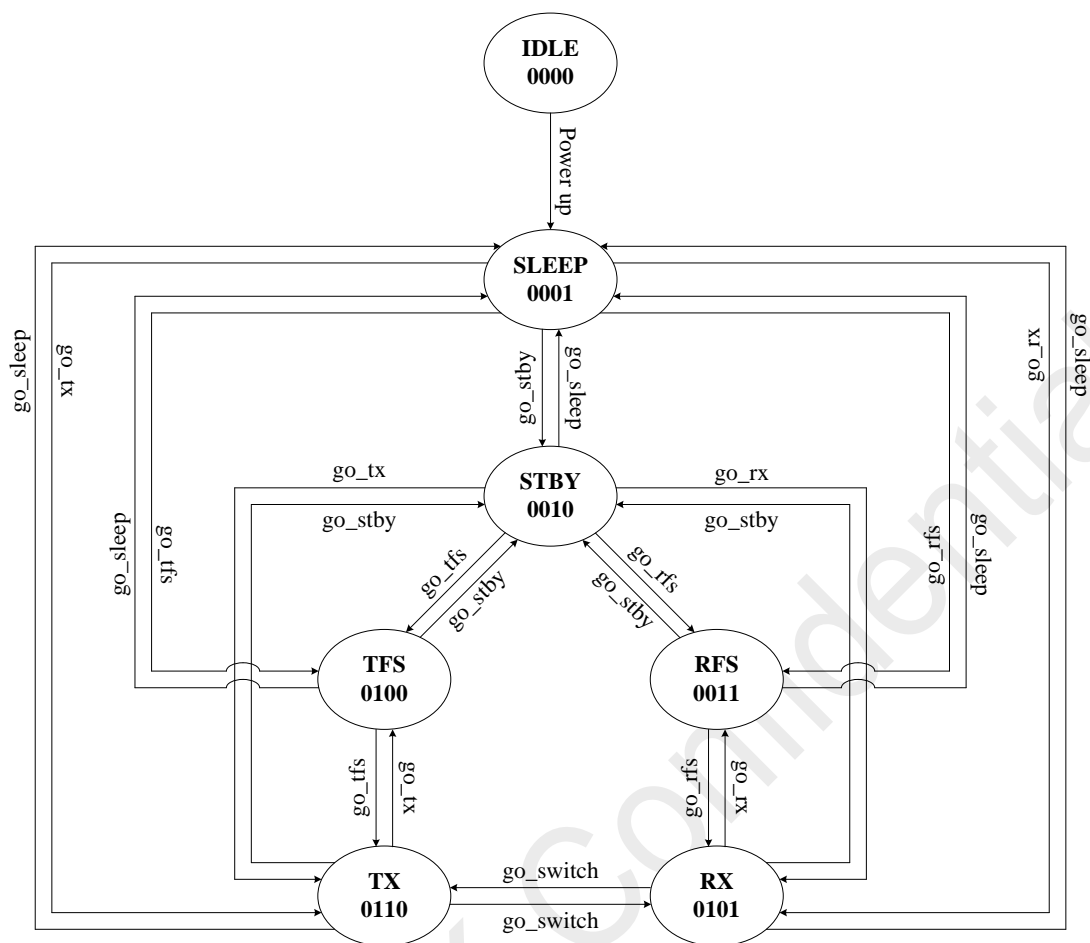


图 6. CMT2300A 状态切换图

表 6. CMT2300A 状态和模块开启表

| 状态 | 二进制码 | 切换命令 | 开启模块 | 可选择开启模块 |
|-------|------|----------|---|--------------------|
| IDLE | 0000 | soft_rst | SPI, POR | 无 |
| SLEEP | 0001 | go_sleep | SPI, POR, FIFO | LFOSC, Sleep Timer |
| STBY | 0010 | go_stby | SPI, POR, XTAL, FIFO | CLKO |
| RFS | 0011 | go_rfs | SPI, POR, XTAL, PLL, FIFO | CLKO |
| TFS | 0100 | go_tfs | SPI, POR, XTAL, PLL, FIFO | CLKO |
| RX | 0101 | go_rx | SPI, POR, XTAL, PLL, LNA+MIXER+IF, FIFO | CLKO, RX Timer |
| TX | 0110 | go_tx | SPI, POR, XTAL, PLL, PA, FIFO | CLKO |

MCU 在发送 go_*命令之后，芯片有时需要等待一定的时间才能给成功切换状态，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 上电后，通常需要等待大概 1ms 的时间，POR 才会释放。POR 释放之后，晶体也会启动，启动时间默认为 N ms，根据晶体本身特性而定；启动后需要等待晶体稳定系统才能开始工作，默认设置的稳定时间是 2.48ms，这个时间在后面可以通过写入 XTAL_STB_TIME <2:0> 进行修改。在晶体稳定之前，芯片都会停留在 IDLE 状态。在晶体的稳定之后，芯片就会离开 IDLE，开始做各个模块的校正。芯片完成校正后就会停留在 SLEEP，等待用户进行初始化配置。在任何时候，只要进行外部硬复位或者软复位，芯片就会回到 IDLE 并重新进行一次上电流程。

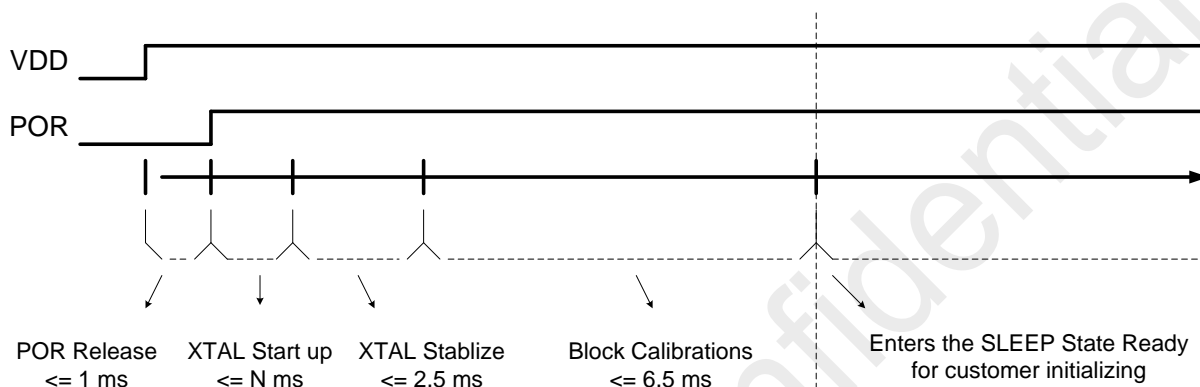


图 7. CMT2300A 上电流程图

Block Calibrations 里面包含了 LFOSC 的校正，可保证 LFOSC 的频率误差小于 1%。LFOSC 主要用于驱动 Sleep Timer，如果不需要使用 Sleep Timer，又或者不介意 Sleep Timer 的精度低一点，就可以将 LFOSC 的校正关闭。关掉 LFOSC 校正的方法是，用户需要在配置阶段（下面有介绍）将 LFSOC_CAL1_EN 和 LFOSC_CAL2_EN 配置成 0。那么下一次软复位之后，LFOSC 校正功能就会关闭，可以节省大约 5ms 的校正时间。但是无论如何，第一次上电还是会默认做一次 LFOSC 校正。

■ SLEEP 状态

在 SLEEP 下芯片的功耗是最低的，几乎所有的模块都关闭了。SPI 是开启的，配置区和控制区 1 的寄存器会被保存，FIFO 之前被填入的内容，也会保持不变。但是，用户不能操作 FIFO，不能更改寄存器内容。如果用户打开了定时唤醒的功能，那么 LFOSC 和睡眠计数器就会开启并工作。从 IDLE 切换到 SLEEP 所需要的时间就是上面介绍的上电流程时间。从其余状态切换到 SLEEP 都会立即完成。

■ STBY 状态

在 STBY 下，晶体开启了，数字电路的 LDO 也会开启，电流会稍微增加，FIFO 可以被操作。用户可以选择是否输出 CLK0（系统时钟）到 PIN 上。由于晶体以及 LDO 的开启，所以相比起 SLEEP，从 STBY 切换到发射或者接收所需要的时间都会比较短。从 SLEEP 切换到 STBY 需要等待晶体开启和稳定的时间后才能完成。从其他状态切换到 STBY 会立即完成。

■ RFS 状态

RFS 是切换到 RX 之前的一个过渡状态，除了接收机的 RF 模块是关闭之外，其它模块都开启了，电流会比 STBY 大。由于在 RFS 的时候，PLL 已经锁定在 RX 的频点了，所以不能切换到 TX。从 STBY 切换到 RFS 大概需要 350us 的 PLL 校正和稳定时间，从 SLEEP 切换到 RFS 就需要加上晶体启动和稳定的时间，从其它

状态切换到 RFS 会立即完成。

■ TFS 状态

TFS 是切换到 TX 之前的一个过渡状态，除了发射机的 RF 模块是关闭之外，其它模块都开启了，电流会比 STBY 大。由于在 TFS 的时候，PLL 已经锁定在 TX 的频点了，所以不能切换到 RX。从 STBY 切换到 TFS 大概需要 350us 的 PLL 校正和稳定时间，从 SLEEP 切换到 TFS 就需要加上晶体启动和稳定的时间，从其它状态切换到 TFS 会立即完成。

■ RX 状态

在 RX 所有关于接收机的模块都会打开。从 RFS 切换到 RX 只需要 20us。从 STBY 切换到 RX 需要加上 350us 的 PLL 校正和稳定时间。从 SLEEP 切换到 RX 需要加上晶体启动和稳定的时间。在 TX 可以通过发送 go_switch 命令来快速切换到 RX，无论 TX 和 RX 设置的频点是否相同，都需要等待 350us 的 PLL 重新校正和稳定时间才能切换成功。

■ TX 状态

在 TX 所有关于发射机的模块都会打开。从 TFS 切换到 TX 只需要 20us。从 STBY 切换到 TX 需要加上 350us 的 PLL 校正和稳定时间。从 SLEEP 切换到 TX 需要加上晶体启动和稳定的时间。在 RX 可以通过发送 go_switch 命令来快速切换到 TX，无论 RX 和 TX 设置的频点是否相同，都需要等待 350us 的 PLL 重新校正和稳定时间才能切换成功。

CMT2300A 状态切换库函数代码示例详见附录 3。

3.3 软复位 (Softrst)

CMT2300A 的软复位，是通过用 SPI 将 0xFF 写入地址 0x7F 实现的。芯片收到这个命令后，会立即进行复位操作，回到 IDLE 状态，并立即重新进行芯片初始化流程。所以用户发送软复位命令后，是查询不到 IDLE 状态的，因为这个状态一闪而过。

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 CMT2300A 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。

CMT2300A 的配置思路是，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，用户只需要懂得操作 RFPDK，以及选择性地去理解他们需要理解的寄存器就行了。

之前的章节对寄存器做了一个简单介绍。这个章节会对 RFPDK 做一个简单介绍，然后在后续的章节介绍如何结合两者开展配置工作。

下面是 RFPDK 的截图：

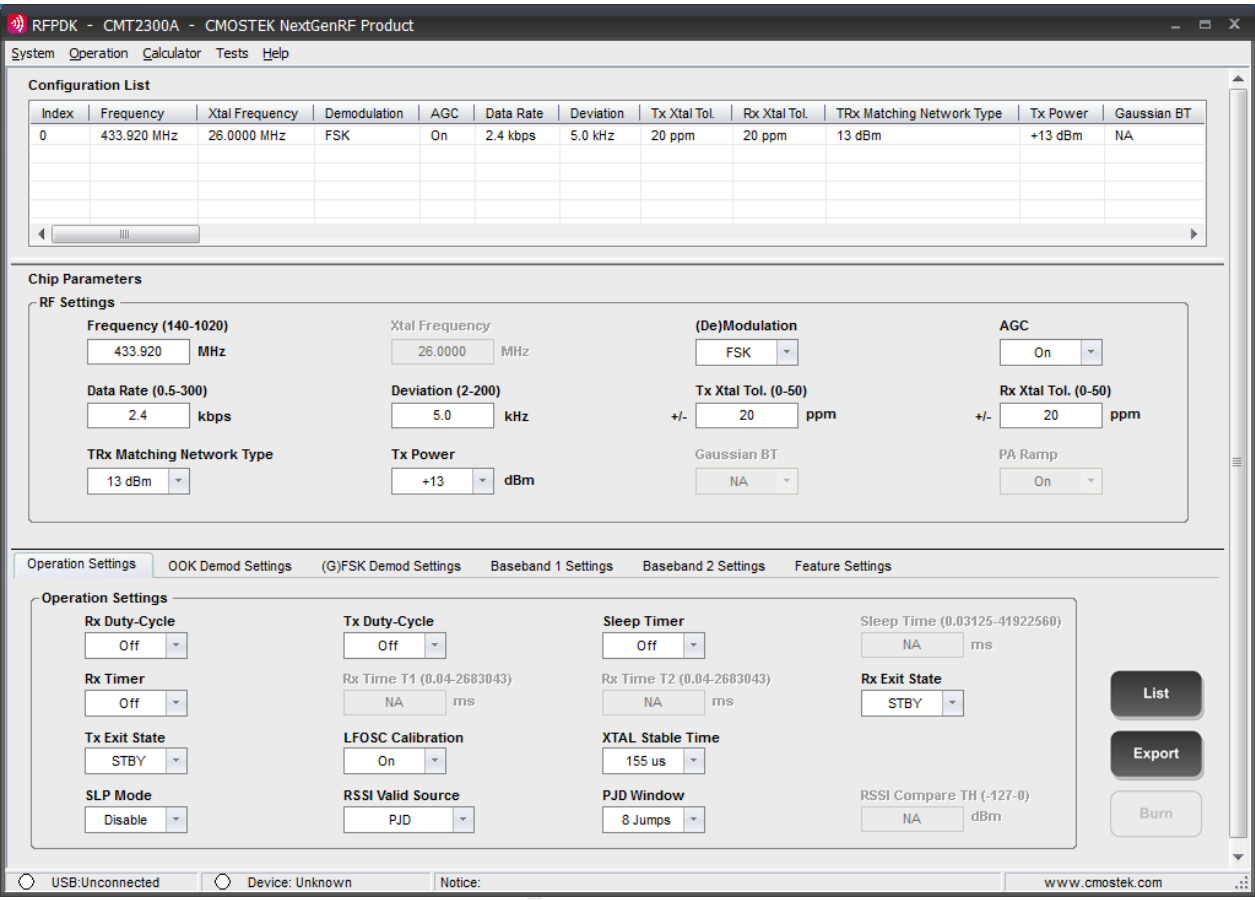


图 8. CMT2300A RFPDK

界面上，需要用户输入的有 7 个主要板块：

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 |
| 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 Settings | 其余功能参数 | 输入参数与寄存器一一对应，用户可以理解 |

下面是生成的配置文件的实例。

首先在文件的上部的注释，用于确认当前成功生成的配置信息。这部分通常是查错时使用，一般情况下用户不需要太关注。

```

-----
; CMT2300A Configuration File
; Generated by CMOSTEK RFPDK 1.41 Beta
; 2017.02.13 16:26
-----

; Mode = Advanced
; Part Number = CMT2300A
; Frequency = 433.920 MHz
; Xtal Frequency = 26.0000 MHz
; Demodulation = FSK
; AGC = On
; Data Rate = 2.4 kbps
; Deviation = 4.8 kHz
; TxXtalTol. = 20 ppm
; Rx XtalTol. = 20 ppm
; Tx Power = +16 dBm
; Gaussian BT = NA
; Bandwidth = Auto-Select kHz
; CDR Type = Counting
; CDR DR Range = NA
; AFC = On
; AFC Method = Auto-Select
; Data Representation = 0:F-low 1:F-high
; 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 = SLEEP
; Tx Exit State = SLEEP
; SLP Mode = Disable
; RSSI Valid Source = PJD
; PJD Window = 8 Jumps
; LFOSC Calibration = On
; Xtal Stable Time = 155 us
; RSSI Compare TH = NA
; Data Mode = Packet
; Whitening = Disable
; Whiten Type = NA
; Whiten Seed Type = NA
; Whiten Seed = NA
; Manchester = Disable
; Manchester Type = NA
; FEC = Disable
; FEC Type = NA
; Tx Prefix Type = 0
; Tx Packet Number = 1
; Tx Packet Gap = 32
; Fifo Threshold = 16
; Packet Type = Fixed Length
; Node-Length Position = NA
; Payload Bit Order = Start from msb

```

```

; Preamble Rx Size           = 2
; Preamble Tx Size          = 8
; Preamble Value             = 170
; Preamble Unit              = 8-bit
; Sync Size                  = 2-byte
; Sync Value                  = 4660
; Sync Tolerance              = None
; Sync Manch                 = Disable
; Node ID Size                = NA
; Node ID Value               = NA
; Node ID Mode                = None
; Node ID Err Mask            = NA
; Node ID Free                = NA
; Payload Length              = 32
; CRC Options                 = None
; CRC Seed                    = NA
; CRC Range                   = NA
; CRC Swap                    = NA
; CRC Bit Inv                 = NA
; CRC Bit Order               = NA
; Dout Mute                   = Off
; Dout Adjust Mode            = Disable
; Dout Adjust Percentage      = NA
; Collision Detect             = Off
; Collision Detect Offset      = NA
; RSSI Detect Mode             = Always
; RSSI Filter Setting          = No Filtering
; RF Performance               = Low
; LBD Threshold                = 2.4 V
; System Clock Output          = Off
; System Clock Frequency      = NA
; RSSI Offset                  = 26
; RSSI Offset Sign             = 1

```

文件的后半部分，就是用户需要导入 MCU 程序的寄存器配置内容，用 16 进制表示。为了用户使用方便，工具会自动将所有寄存器的内容划分成 6 个区，地址范围是 0x00 – 0x5F。为了方便阅读，下面用列表的方式显示 6 个区的内容。

```

;-----
; The following are the Register contents
;-----

```

| [CMT Bank] | | [System Bank] | | [Frequency Bank] | | [Data Rate Bank] | | [Baseband Bank] | | [TX Bank] | |
|------------|-------|---------------|-------|------------------|-------|------------------|-------|-----------------|-------|-----------|-------|
| Addr | Value | Addr | Value | Addr | Value | Addr | Value | Addr | Value | Addr | Value |
| 0x00 | 0x00 | 0x0C | 0xA5 | 0x18 | 0x42 | 0x20 | 0x32 | 0x38 | 0x12 | 0x55 | 0x50 |
| 0x01 | 0x66 | 0x0D | 0xE0 | 0x19 | 0x71 | 0x21 | 0x18 | 0x39 | 0x08 | 0x56 | 0x06 |
| 0x02 | 0x6C | 0x0E | 0x30 | 0x1A | 0xCE | 0x22 | 0x00 | 0x3A | 0x00 | 0x57 | 0x03 |
| 0x03 | 0x7C | 0x0F | 0x00 | 0x1B | 0x1C | 0x23 | 0x99 | 0x3B | 0xAA | 0x58 | 0x00 |
| 0x04 | 0xF0 | 0x10 | 0x00 | 0x1C | 0x42 | 0x24 | 0xC1 | 0x3C | 0x02 | 0x59 | 0x44 |
| 0x05 | 0x80 | 0x11 | 0xF4 | 0x1D | 0x5B | 0x25 | 0x9B | 0x3D | 0x00 | 0x5A | 0xD0 |
| 0x06 | 0x14 | 0x12 | 0x10 | 0x1E | 0x1C | 0x26 | 0x06 | 0x3E | 0x00 | 0x5B | 0x00 |
| 0x07 | 0x08 | 0x13 | 0xE2 | 0x1F | 0x1C | 0x27 | 0x0A | 0x3F | 0x00 | 0x5C | 0x72 |
| 0x08 | 0xB1 | 0x14 | 0x42 | | | 0x28 | 0x9F | 0x40 | 0x00 | 0x5D | 0x0C |
| 0x09 | 0x03 | 0x15 | 0x20 | | | 0x29 | 0x39 | 0x41 | 0x00 | 0x5E | 0x3F |
| 0x0A | 0x00 | 0x16 | 0x00 | | | 0x2A | 0x29 | 0x42 | 0x00 | 0x5F | 0x7F |
| 0x0B | 0xD0 | 0x17 | 0x81 | | | 0x2B | 0x29 | 0x43 | 0x34 | | |
| | | | | | | 0x2C | 0xC0 | 0x44 | 0x12 | | |
| | | | | | | 0x2D | 0x51 | 0x45 | 0x00 | | |
| | | | | | | 0x2E | 0x2A | 0x46 | 0x1F | | |
| | | | | | | 0x2F | 0x53 | 0x47 | 0x00 | | |
| | | | | | | 0x30 | 0x00 | 0x48 | 0x00 | | |

| [CMT Bank] | [System Bank] | [Frequency Bank] | [Data Rate Bank] | [Baseband Bank] | [TX Bank] |
|------------|---------------|------------------|------------------|-----------------|-----------|
| | | | 0x31 0x00 | 0x49 0x00 | |
| | | | 0x32 0xB4 | 0x4A 0x00 | |
| | | | 0x33 0x00 | 0x4B 0x00 | |
| | | | 0x34 0x00 | 0x4C 0x00 | |
| | | | 0x35 0x01 | 0x4D 0x00 | |
| | | | 0x36 0x00 | 0x4E 0x00 | |
| | | | 0x37 0x00 | 0x4F 0x00 | |
| | | | | 0x50 0x00 | |
| | | | | 0x51 0x00 | |
| | | | | 0x52 0x00 | |
| | | | | 0x53 0x20 | |
| | | | | 0x54 0x10 | |

最后一部分是用于工具自身校验的 CRC 值，用户可以忽略。

```

-----
; The following is the CRC result for
; the above Register contents
-----

```

0xDE67

3.5 芯片初始化流程

客户需要用 RFPDK 生成配置区（0x00 – 0x5F）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 可以按照以下的初始化流程来操作：

1. 确认芯片完成了上电或者复位，停留在 SLEEP 状态。
2. 发送 go_stby 命令并确认芯片进入了 STBY 状态。
3. 将 RFPDK 生成的寄存器内容写入配置区，地址是 0x00 – 0x5F。
4. 如果不需要使用 SLEEP TIMER，将 RECAL_LFOSC_EN, LFSOC_CAL1_EN 和 LFOSC_CAL2_EN 配置成 0。
5. 按照实际需要，设置好控制区 1（0x60 – 0x6A）的寄存器，将 RSTN_IN_EN 设置为 0。
6. 将 CUS_MODE_STA（0x61）寄存器的第 4 个比特 CFG_RETAIN 配置成 1，这个比特会让 0x00 – 0x5F 整个配置区的值不会被软复位擦除掉。
7. 发送 go_sleep 命令，这个动作让寄存器配置生效。

初始化完成后，就可以开始工作了。往后要更改全部或者部分的寄存器，也要让芯片安全地切换并停留在 STBY 状态，才可以配置，完成后要切换到 SLEEP 让其生效。无论任何时候要更改配置区的寄存器，都要按照这个步骤进行配置。

另外，如何当前芯片处于全自动的 duty cycle 工作模式，MCU 必须先安全地让芯片退出 Duty Cycle 模式，然后按上面介绍的流程去配置。关于进入和退出 Duty Cycle 模式的操作细节，在 AN 有详细介绍。

CMT2300A 初始化代码操作详见附录 4。

3.6 配置区的功能划分

如前面提到，整个配置区按照功能被划分成 6 个片区，与 RFPDK 生成的寄存器文件划分的区域一一对应。当用户需要更改某一项功能的时候，就根据使用指引，直接将对应的区域修改就可以了，不需要牵涉到别的寄存器，也不需去研究明白具体某个寄存器的内容，因为这些内容都是用 RFPDK 配置生成的。

例如，用户希望能在应用中修改数据率，就先用 RFPDK 配置好要使用的数据率，然后导出整个寄存器配置内容，再将数据率区对应的内容摘取出来作为一个数组写在 MCU 程序里，到程序需要修改数据率的时候，就将这部分内容直接写入芯片对应的寄存器地址就可以了。

下面的列表，分别给出 FSK 和 OOK 模式下 6 个区是如何划分的。两者的主要区别在于数据率区的划分。

表 8. CMT2300A FSK 模式下寄存器配置区划分表

| 区域 | 地址 | RFPDK 生成文件的区域名称 |
|-------|------------------|-----------------|
| CMT 区 | 0x00 – 0x0B | CMT Bank |
| 系统区 | 0x0C – 0x17 | System Bank |
| 频率区 | 0x18 – 0x1F | Frequency Bank |
| 数据率区 | 需填写区：0x20 – 0x32 | Data Rate Bank |
| | 可忽略区：0x33– 0x37 | |
| 基带区 | 0x38 – 0x54 | Baseband Bank |
| 发射区 | 0x55 – 0x5F | TX Bank |

表 9. CMT2300A OOK 模式下寄存器配置区划分表

| 区域 | 地址 | RFPDK 生成文件的区域名称 |
|-------|------------------|-----------------|
| CMT 区 | 0x00 – 0x0B | CMT Bank |
| 系统区 | 0x0C – 0x17 | System Bank |
| 频率区 | 0x18 – 0x1F | Frequency Bank |
| 数据率区 | 需填写区：0x20 – 0x23 | Data Rate Bank |
| | 可忽略区：0x24– 0x2A | |
| | 需填写区：0x2B – 0x37 | |
| 基带区 | 0x38 – 0x54 | Baseband Bank |
| 发射区 | 0x55 – 0x5F | TX Bank |

表中的可忽略区，指的是用户无需进行配置的区域。下面简单介绍一下每一个区的内容，其中 CMT 区，频率区，数据率区，发射区的寄存器用户无须理解，直接导入 RFPDK 生成的参数即可。

3.6.1 CMT 区（0x00 – 0x0B）

CMT 区是给 CMOSTEK 内部使用的，用户可使用 RFPDK 生成的内容直接填写。

表 10. CMT 区

| Addr | R/W | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|------|-----|-----------|---------------------|-------|-------|-------|-------|-------|-------|-------|
| 0x00 | RW | CUS_CMT1 | 用户无须理解，直接用RFPDK生成导入 | | | | | | | |
| 0x01 | RW | CUS_CMT2 | | | | | | | | |
| 0x02 | RW | CUS_CMT3 | | | | | | | | |
| 0x03 | RW | CUS_CMT4 | | | | | | | | |
| 0x04 | RW | CUS_CMT5 | | | | | | | | |
| 0x05 | RW | CUS_CMT6 | | | | | | | | |
| 0x06 | RW | CUS_CMT7 | | | | | | | | |
| 0x07 | RW | CUS_CMT8 | | | | | | | | |
| 0x08 | RW | CUS_CMT9 | | | | | | | | |
| 0x09 | RW | CUS_CMT10 | | | | | | | | |
| 0x0A | RW | CUS_CMT11 | | | | | | | | |
| 0x0B | RW | CUS_RSSI | | | | | | | | |

3.6.2 系统区（0x0C – 0x17）

表 11. 系统区

| Addr | R/W | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|------|-----|-----------|----------------------|---------------|---------------------|-------------|---------------------|----------|----------------------|----------|
| 0x0C | RW | CUS_SYS1 | LMT_VTR [1:0] | | MIXER_BIAS [1:0] | | LNA_MODE [1:0] | | LNA_BIAS [1:0] | |
| 0x0D | RW | CUS_SYS2 | LFOSC_RECAL_EN | LFOSC_CAL1_EN | LFOSC_CAL2_EN | RX_TIMER_EN | SLEEP_TIMER_EN | TX_DC_EN | RX_DC_EN | DC_PAUSE |
| 0x0E | RW | CUS_SYS3 | SLEEP_BYPASS_EN | | XTAL_STB_TIME [2:0] | | TX_EXIT_STATE [1:0] | | RX_EXIT_STATE [1:0] | |
| 0x0F | RW | CUS_SYS4 | | | | | SLEEP_TIMER_M [7:0] | | | |
| 0x10 | RW | CUS_SYS5 | SLEEP_TIMER_M [10:8] | | | | SLEEP_TIMER_R [3:0] | | | |
| 0x11 | RW | CUS_SYS6 | | | | | RX_TIMER_T1_M [7:0] | | | |
| 0x12 | RW | CUS_SYS7 | RX_TIMER_T1_M [10:8] | | | | RX_TIMER_T1_R [3:0] | | | |
| 0x13 | RW | CUS_SYS8 | | | | | RX_TIMER_T2_M [7:0] | | | |
| 0x14 | RW | CUS_SYS9 | RX_TIMER_T2_M [10:8] | | | | RX_TIMER_T2_R [3:0] | | | |
| 0x15 | RW | CUS_SYS10 | COL_DET_EN | COL_OFS_SEL | RX_AUTO_EXIT_DIS | | DOUT_MUTE | | RX_EXTEND_MODE [3:0] | |
| 0x16 | RW | CUS_SYS11 | PJD_TH_SEL | | RSSI_VLD_SRC [1:0] | | RSSI_DET_SEL [1:0] | | RSSI_AVG_MODE [2:0] | |
| 0x17 | RW | CUS_SYS12 | PJD_WIN_SEL [1:0] | | RESV | | RESV | | | |

系统区的参数负责对系统运转模式以及一些系统 Feature 进行控制，例如是否开启 SLEEP TIMER，是否开启 DUTY CYCLE，如何利用 RSSI 或者 PJD（相位跳变检测）来进行超低功耗接收。

3.6.3 频率区（0x18 – 0x1F）

频率区负责配置 RX 和 TX 频点。用户可使用 RFPDK 生成的内容直接填写。如果用户希望在实际应用过程中改变频点，有两种做法：

1. 使用 RFPDK 生成对应各频点的参数，更改频点时重新写入整个频率区的寄存器。
2. 以初始化时频率区设置的频点作为一个基础频点，然后使用快速手动调频机制去切换新的频点。

表 12. 频率区

| Addr | R/W | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|------|-----|---------|---------------------|-------|-------|-------|-------|-------|-------|-------|
| 0x18 | RW | CUS_RF1 | 用户无须理解，直接用RFPDK生成导入 | | | | | | | |
| 0x19 | RW | CUS_RF2 | | | | | | | | |
| 0x1A | RW | CUS_RF3 | | | | | | | | |
| 0x1B | RW | CUS_RF4 | | | | | | | | |
| 0x1C | RW | CUS_RF5 | | | | | | | | |
| 0x1D | RW | CUS_RF6 | | | | | | | | |
| 0x1E | RW | CUS_RF7 | | | | | | | | |
| 0x1F | RW | CUS_RF8 | | | | | | | | |

对于某些特殊应用，TX 和 RX 频点需要分开配置，需要操作特定的寄存器，详情请咨询 CMOSTEK 的技术支持。

3.6.4 数据率区（0x20 – 0x37）

如前面介绍，数据率区的地址划分要分 FSK 和 OOK 两种情况，可忽略的部分为用户无须写入的部分。

数据率区要配置的内容很多，因为不同的数据率会影响到 FSK 或 OOK 的解调参数，还有时钟恢复，以及 AGC 的配置，因此覆盖了比较多的地址。如果用户想在应用程序中改变数据率，需要预先储存 RFPDK 生成的对应数据率区的参数，然后写入整个数据率区的内容（可忽略区除外）。

TX 个 RX 的数据率是统一配置的。如果用户在应用中，需要使用不同的 TX 和 RX 数据率，就必须在切换 TX/RX 之前重新配置一下数据率区。但是，对 TX 数据率起作用的寄存器地址就只有 0x20, 0x21, 0x22，如果用户从 RX 切换到 TX 并且要改变数据率，就只需要改变这 3 个寄存器就可以了；反之，用户从 TX 切换到 RX 并且要改变数据率，就要写入整个数据率区。

表 13. 数据率区

| Addr | R/W | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|------|-----|----------|---------------------|-------|-------|-------|-------|-------|-------|-------|
| 0x20 | RW | CUS_RF9 | 用户无须理解，直接用RFPDK生成导入 | | | | | | | |
| 0x21 | RW | CUS_RF10 | | | | | | | | |
| 0x22 | RW | CUS_RF11 | | | | | | | | |
| 0x23 | RW | CUS_RF12 | | | | | | | | |
| 0x24 | RW | CUS_FSK1 | | | | | | | | |
| 0x25 | RW | CUS_FSK2 | | | | | | | | |
| 0x26 | RW | CUS_FSK3 | | | | | | | | |
| 0x27 | RW | CUS_FSK4 | | | | | | | | |
| 0x28 | RW | CUS_FSK5 | | | | | | | | |
| 0x29 | RW | CUS_FSK6 | | | | | | | | |
| 0x2A | RW | CUS_FSK7 | | | | | | | | |
| 0x2B | RW | CUS_CDR1 | | | | | | | | |
| 0x2C | RW | CUS_CDR2 | | | | | | | | |
| 0x2D | RW | CUS_CDR3 | | | | | | | | |
| 0x2E | RW | CUS_CDR4 | | | | | | | | |
| 0x2F | RW | CUS_AGC1 | | | | | | | | |
| 0x30 | RW | CUS_AGC2 | | | | | | | | |
| 0x31 | RW | CUS_AGC3 | | | | | | | | |
| 0x32 | RW | CUS_AGC4 | | | | | | | | |
| 0x33 | RW | CUS_OOK1 | | | | | | | | |
| 0x34 | RW | CUS_OOK2 | | | | | | | | |
| 0x35 | RW | CUS_OOK3 | | | | | | | | |
| 0x36 | RW | CUS_OOK4 | | | | | | | | |
| 0x37 | RW | CUS_OOK5 | | | | | | | | |

3.6.5 基带区（0x38 – 0x54）

表 14. 基带区

| Addr | R/W | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|------|-----|-----------|---------------------|---------------------|------------------|-------------------|-----------------|----------------------|---------------------|-------------|
| 0x38 | RW | CUS_PKT1 | RX_PREAM_SIZE [4:0] | | | | | PREAM_LENG_UNIT | DATA_MODE [1:0] | |
| 0x39 | RW | CUS_PKT2 | | | | | | TX_PREAM_SIZE [7:0] | | |
| 0x3A | RW | CUS_PKT3 | | | | | | TX_PREAM_SIZE [15:8] | | |
| 0x3B | RW | CUS_PKT4 | | | | | | PREAM_VALUE [7:0] | | |
| 0x3C | RW | CUS_PKT5 | RESV | SYNC_TOL [2:0] | | | SYNC_SIZE [2:0] | | | SYNC_MAN_EN |
| 0x3D | RW | CUS_PKT6 | | | | | | SYNC_VALUE [7:0] | | |
| 0x3E | RW | CUS_PKT7 | | | | | | SYNC_VALUE [15:8] | | |
| 0x3F | RW | CUS_PKT8 | | | | | | SYNC_VALUE [23:16] | | |
| 0x40 | RW | CUS_PKT9 | | | | | | SYNC_VALUE [31:24] | | |
| 0x41 | RW | CUS_PKT10 | | | | | | SYNC_VALUE [39:32] | | |
| 0x42 | RW | CUS_PKT11 | | | | | | SYNC_VALUE [47:40] | | |
| 0x43 | RW | CUS_PKT12 | | | | | | | | |
| 0x44 | RW | CUS_PKT13 | | | | | | SYNC_VALUE [63:56] | | |
| 0x45 | RW | CUS_PKT14 | RESV | PAYLOAD_LENG [10:8] | | | AUTO_ACK_EN | NODE_LENG_POS_SEL | PAYLOAD_BIT_ORDER | PKT_TYPE |
| 0x46 | RW | CUS_PKT15 | | | | | | PAYLOAD_LENG [7:0] | | |
| 0x47 | RW | CUS_PKT16 | RESV | RESV | NODE_FREE_EN | NODE_ERR_MASK | NODE_SIZE [1:0] | | NODE_DET_MODE [1:0] | |
| 0x48 | RW | CUS_PKT17 | | | | | | NODE_VALUE [7:0] | | |
| 0x49 | RW | CUS_PKT18 | | | | | | NODE_VALUE [15:8] | | |
| 0x4A | RW | CUS_PKT19 | | | | | | NODE_VALUE [23:16] | | |
| 0x4B | RW | CUS_PKT20 | | | | | | NODE_VALUE [31:24] | | |
| 0x4C | RW | CUS_PKT21 | FEC_TYPE | FEC_EN | CRC_BYTE_SWAP | CRC_BIT_INV | CRC_RANGE | CRC_TYPE [1:0] | | CRC_EN |
| 0x4D | RW | CUS_PKT22 | | | | | | CRC_SEED [7:0] | | |
| 0x4E | RW | CUS_PKT23 | | | | | | CRC_SEED [15:8] | | |
| 0x4F | RW | CUS_PKT24 | CRC_BIT_ORDER | WHITEN_SEED [8] | WHITEN_SEED_TYPE | WHITEN_TYPE [1:0] | | WHITEN_EN | MANCH_TYPE | MANCH_EN |
| 0x50 | RW | CUS_PKT25 | | | | | | WHITEN_SEED [7:0] | | |
| 0x51 | RW | CUS_PKT26 | RESV | RESV | RESV | RESV | RESV | TX_PREFIX_TYPE [1:0] | | |
| 0x52 | RW | CUS_PKT27 | | | | | | TX_PKT_NUM [7:0] | | |
| 0x53 | RW | CUS_PKT28 | | | | | | TX_PKT_GAP [7:0] | | |
| 0x54 | RW | CUS_PKT29 | FIFO_AUTO_RES_EN | | | | FIFO_TH [6:0] | | | |

基带区主要就是用于配置发射和接受数据的包格式和编解码，还有一部分 FIFO 的配置（FIFO 的使用模式和控制寄存器主要放在控制区）。这部分的内容与 RFPDK 的参数一一对应，用户需要详细了解后进行配置。

3.6.6 发射区（0x55 – 0x5F）

发射区主要负责配置 TX 的参数，由于 TX 的数据率跟 RX 是统一的，所以被归类到数据率区里面。

TX 的配置相关性比较小，如果用户只想单独改变某一项特性，例如功率，或者是 Deviation，又或者是 LBD 的配置，是不需要整个发射区写一遍的，只需要配置某几个寄存器。因此，对于这些发射区来说，用户不需要去理解每一个寄存器的意义和计算方式，但是需要知道什么时候需要更改哪几个寄存器，详情请参考 AN。

表 15. 发射区

| Addr | R/W | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|------|-----|----------|---------------------|-------|-------|-------|-------|-------|-------|-------|
| 0x55 | RW | CUS_TX1 | 用户无须理解，直接用RFPDK生成导入 | | | | | | | |
| 0x56 | RW | CUS_TX2 | | | | | | | | |
| 0x57 | RW | CUS_TX3 | | | | | | | | |
| 0x58 | RW | CUS_TX4 | | | | | | | | |
| 0x59 | RW | CUS_TX5 | | | | | | | | |
| 0x5A | RW | CUS_TX6 | | | | | | | | |
| 0x5B | RW | CUS_TX7 | | | | | | | | |
| 0x5C | RW | CUS_TX8 | | | | | | | | |
| 0x5D | RW | CUS_TX9 | | | | | | | | |
| 0x5E | RW | CUS_TX10 | | | | | | | | |
| 0x5F | RW | CUS_LBD | | | | | | | | |

3.7 控制区的使用简介

如前面介绍，控制区分为 1 和 2 两个区。控制区 1 是在 SLEEP 状态是可以保存的，控制区 2 在 SLEEP 状态是不能保存的。

控制区 1（0x60 – 0x6A）

表 16. 控制 1 区

| Addr | R/W | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|------|-----|---------------|---------------------|------------------|-----------------|-------------------|---------------------|----------------|-----------------|--------------------|
| 0x60 | RW | CUS_MODE_CTL | CHIP_MODE_SWT [7:0] | | | | | | | |
| 0x61 | RW | CUS_MODE_STA | RESV | RESV | RSTN_IN_EN | CFG_RETAIN | CHIP_MODE_STA [3:0] | | | |
| 0x62 | RW | CUS_EN_CTL | RESV | RESV | UNLOCK_STOP_EN | LBD_STOP_EN | RESV | RESV | RESV | RESV |
| 0x63 | RW | CUS_FREQ_CHNL | FH_CHANNEL [7:0] | | | | | | | |
| 0x64 | RW | CUS_FREQ_OFS | FH_OFFSET [7:0] | | | | | | | |
| 0x65 | RW | CUS_IO_SEL | RESV | RESV | GPIO3_SEL [1:0] | | GPIO2_SEL [1:0] | | GPIO1_SEL [1:0] | |
| 0x66 | RW | CUS_INT1_CTL | RF_SWT1_EN | RF_SWT2_EN | INT_POLAR | INT1_SEL [4:0] | | | | |
| 0x67 | RW | CUS_INT2_CTL | RESV | LFOSC_OUT_EN | TX_DIN_INV | INT2_SEL [4:0] | | | | |
| 0x68 | RW | CUS_INT_EN | SL_TMO_EN | RX_TMO_EN | TX_DONE_EN | PREAM_OK_EN | SYNC_OK_EN | NODE_OK_EN | CRC_OK_EN | PKT_DONE_EN |
| 0x69 | RW | CUS_FIFO_CTL | TX_DIN_EN | TX_DIN_SEL [1:0] | | FIFO_AUTO_CLR_DIS | FIFO_TX_RD_EN | FIFO_RX_TX_SEL | FIFO_MERGE_EN | SPI_FIFO_RD_WR_SEL |
| 0x6A | W | CUS_INT CLR1 | RESV | RESV | SL_TMO_FLG | RX_TMO_FLG | TX_DONE_FLG | TX_DONE_CLR | SL_TMO_CLR | RX_TMO_CLR |

控制区 2（0x6B – 0x71）

表 17. 控制 2 区

| Addr | R/W | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|------|-----|----------------|------------------|------------------|------------------|----------------|-----------------|------------------|------------------|----------------|
| 0x6B | W | CUS_INT_CLR2 | RESV | RESV | LBD_CLR | PREAM_OK_CLR | SYNC_OK_CLR | NODE_OK_CLR | CRC_OK_CLR | PKT_DONE_CLR |
| 0x6C | W | CUS_FIFO_CLR | RESV | RESV | RESV | RESV | RESV | FIFO_RESTORE | FIFO_CLR_RX | FIFO_CLR_TX |
| 0x6D | R | CUS_INT_FLAG | LBD_FLG | COL_ERR_FLG | PKT_ERR_FLG | PREAM_OK_FLG | SYNC_OK_FLG | NODE_OK_FLG | CRC_OK_FLG | PKT_OK_FLG |
| 0x6E | R | CUS_FIFO_FLAG | RESV | RX_FIFO_FULL_FLG | RX_FIFO_NMTY_FLG | RX_FIFO_TH_FLG | RX_FIFO_OVF_FLG | TX_FIFO_FULL_FLG | TX_FIFO_NMTY_FLG | TX_FIFO_TH_FLG |
| 0x6F | R | CUS_RSSI_CODE | RSSI_CODE [7:0] | | | | | | | |
| 0x70 | R | CUS_RSSI_DBM | RSSI_DBM [7:0] | | | | | | | |
| 0x71 | R | CUS_LBD_RESULT | LBD_RESULT [7:0] | | | | | | | |

用户可以通过操作控制区，来实现对芯片的工作模式切换，IO 和中断控制，FIFO 控制，手动跳频，RSSI 读取，等等。因此，控制区的寄存器是用户在应用程序中频繁操作的。

3.8 流程总结

从上面介绍的信息可以总结出，用户操作芯片的主要流程其实就是包含了 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– 需要搞清楚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，将需要更改的区域截取出来并导入程序，在需要的时候将内容写入对应的寄存器地址。
 - b) 控制芯片进行工作– 搞清楚控制区 1 和控制区 2 的寄存器含义，阅读 AN 中详解介绍的每一项芯片特性和操作规则，包括了 FIFO 控制，IO 和中断控制，RSSI 读取，手动跳频，等等，再对芯片进行符合应用要求的操作。

4 CMT2300A_DemoEasy 简介

4.1 软件层次结构

为了规范 CMT2300A 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。主要分为 Application, Radio handlers, CMT2300A drivers, Hardware abstraction layer, Hardware

- i. Application: 应用层，Demo 中为简单的收发数据包
- ii. Radio handlers: 芯片处理层，包含芯片的初始化，配置，状态控制等流程
- iii. CMT2300 drivers: 芯片驱动层，提供给上层调用
- iv. Hardware abstraction layer: 抽象硬件层，实现芯片的寄存器，FIFO，GPIO 的访问和控制
- v. Hardware: 硬件层，包括 MCU 提供的 LED，按键，SPI 通讯等资源

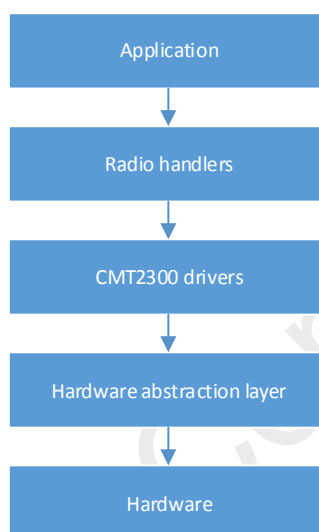


图 9. 软件层次结构

4.2 软件实现以及调用关系

这里介绍模块模块相互间的调用关系，系统初始化流程，工作流程。主要介绍 main.c, radio.c, cmt2300.c, cmt2300_hal.c 等文件

- 1> main.c: 应用层实现
- 2> radio.c: 芯片处理层实现
- 3> cmt2300.c: 芯片驱动层实现
- 4> cmt2300_hal.c: 抽象硬件层实现
- 5> cmt_spi3.c: 芯片 SPI 模拟时序实现，分为寄存器和 FIFO 访问时序

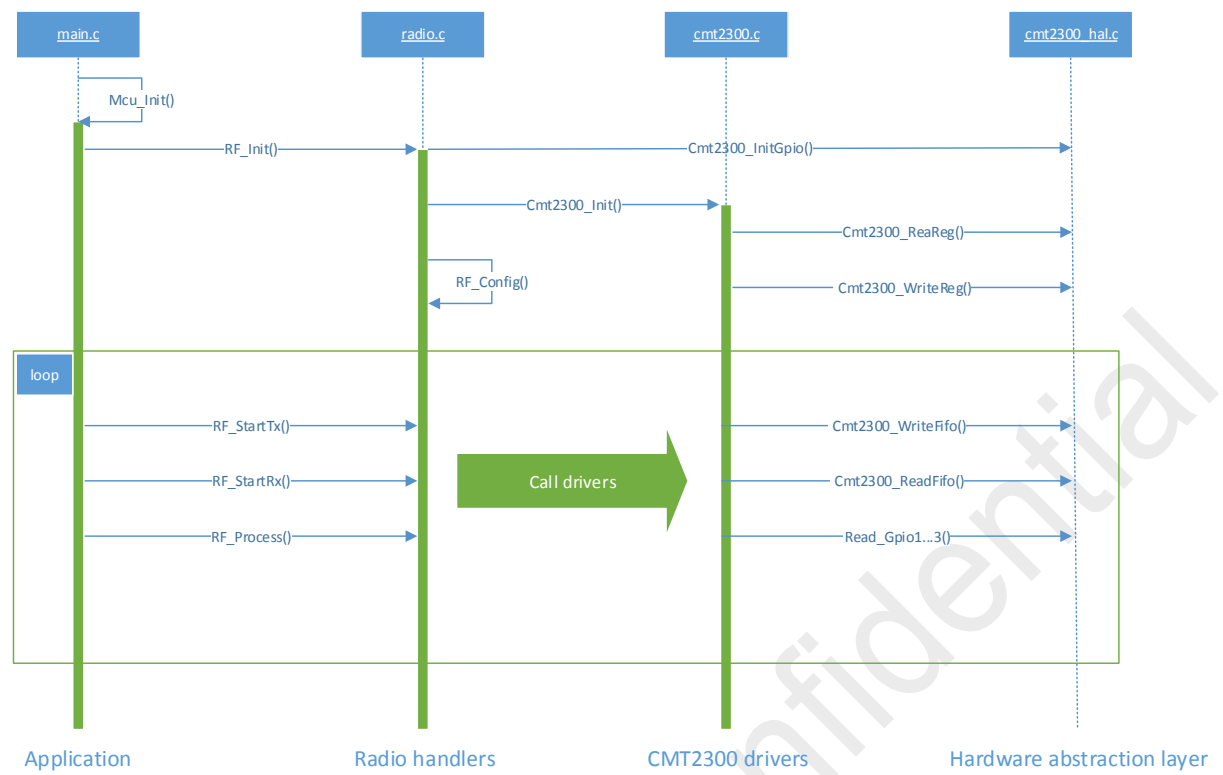


图 10. 软件实现以及调用关系

4.2.1 CMT2300A 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**，**GPIO1/2/3**，软复位，使能或者关闭一些寄存器。

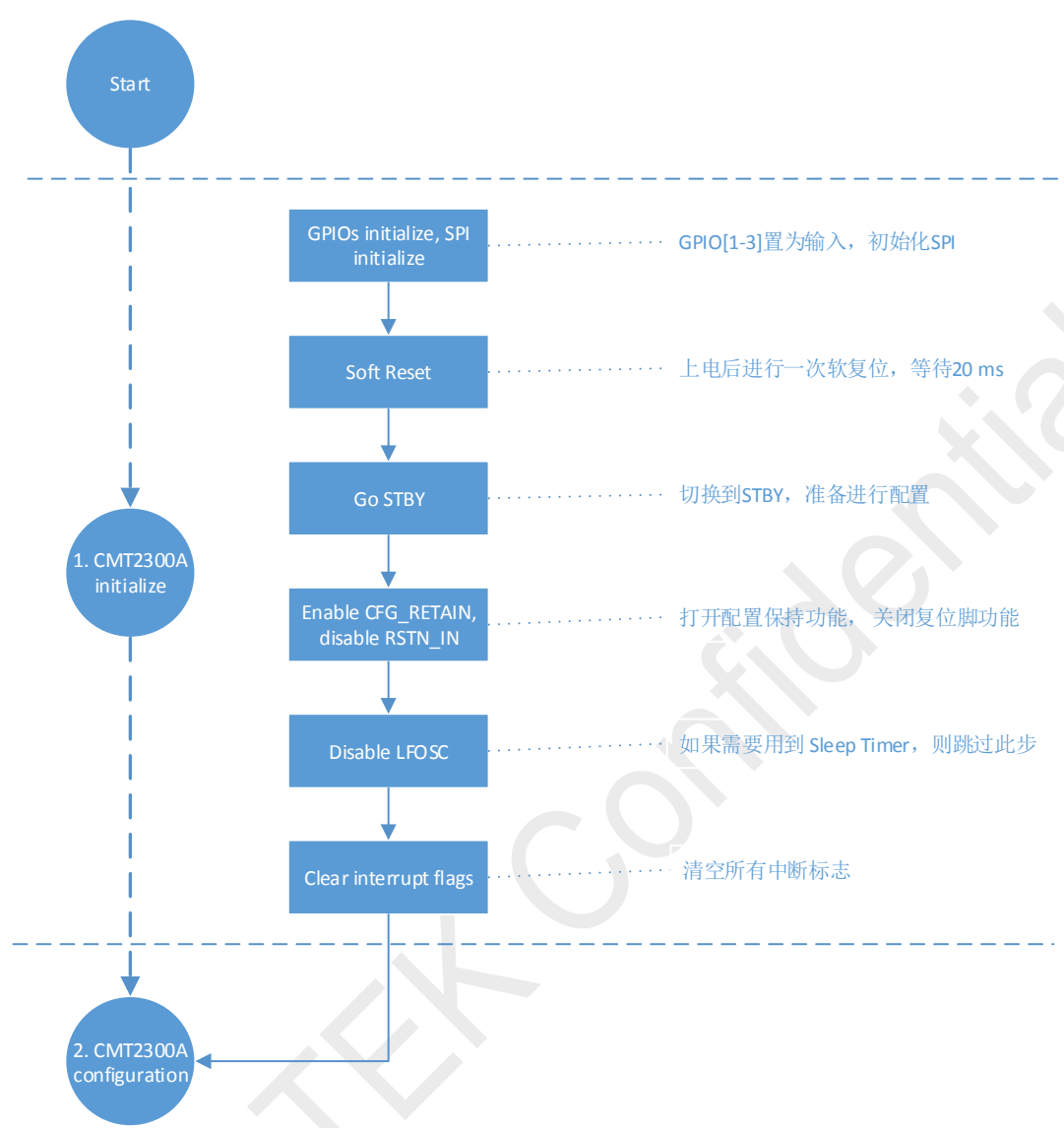


图 11. CMT2300A 初始化

4.2.2 CMT2300A 配置

芯片初始化完成之后，需要调用 RF_Config 函数，在 STBY 状态下，对芯片寄存器，中断，GPIO 进行配置，最后进入 SLEEP 状态让配置生效。

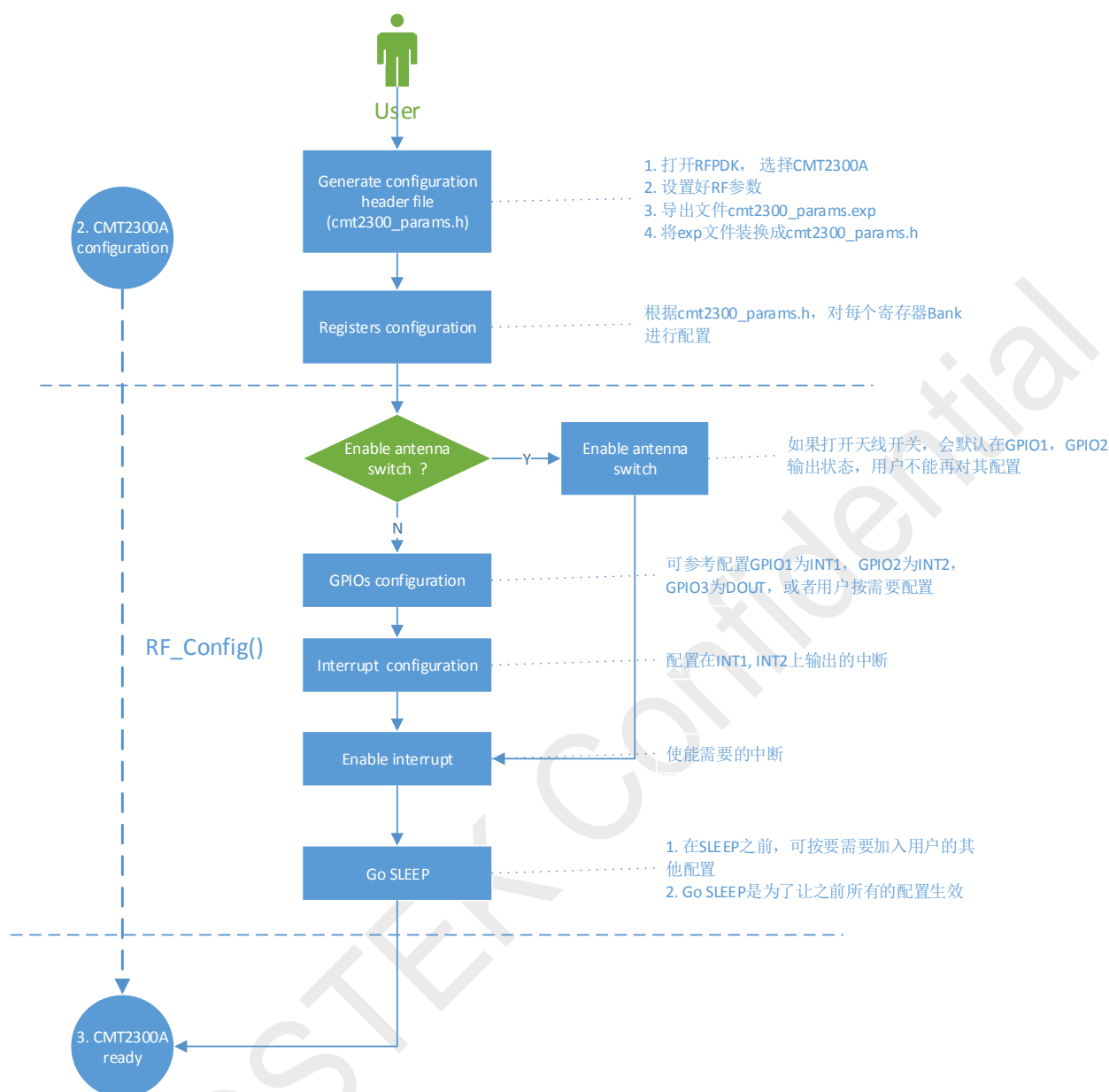


图 12. CMT2300A 配置

4.2.3 CMT2300 状态处理

芯片配置完成之后, 可调用 `RF_StartTx()` 或 `RF_StartRx()` 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者发射, 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

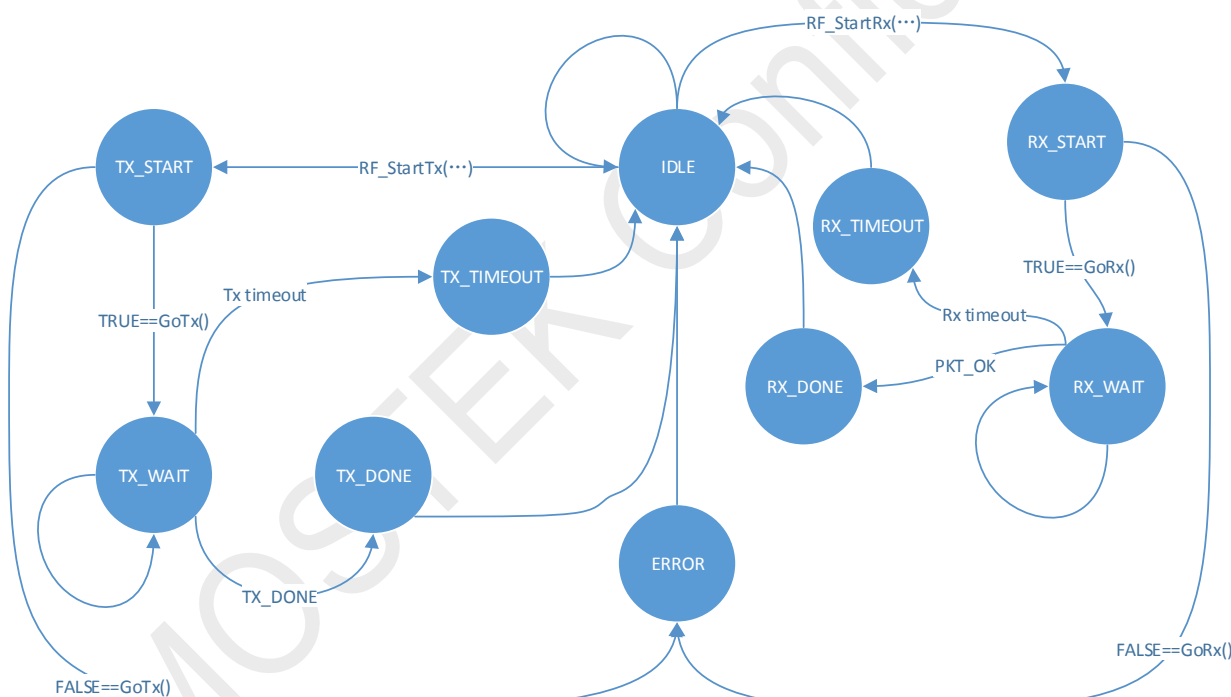


图 13. CMT2300A 状态处理

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: CMT2300 全部 API 接口文件
- 9> cmt2300_params.h: 基于 RFPDK 导出 exp 文件转换的头文件，一一对应 exp 文件的各个寄存器 Bank
- 10> cmt2300_defs.h: CMT2300 芯片的寄存器地址宏定义

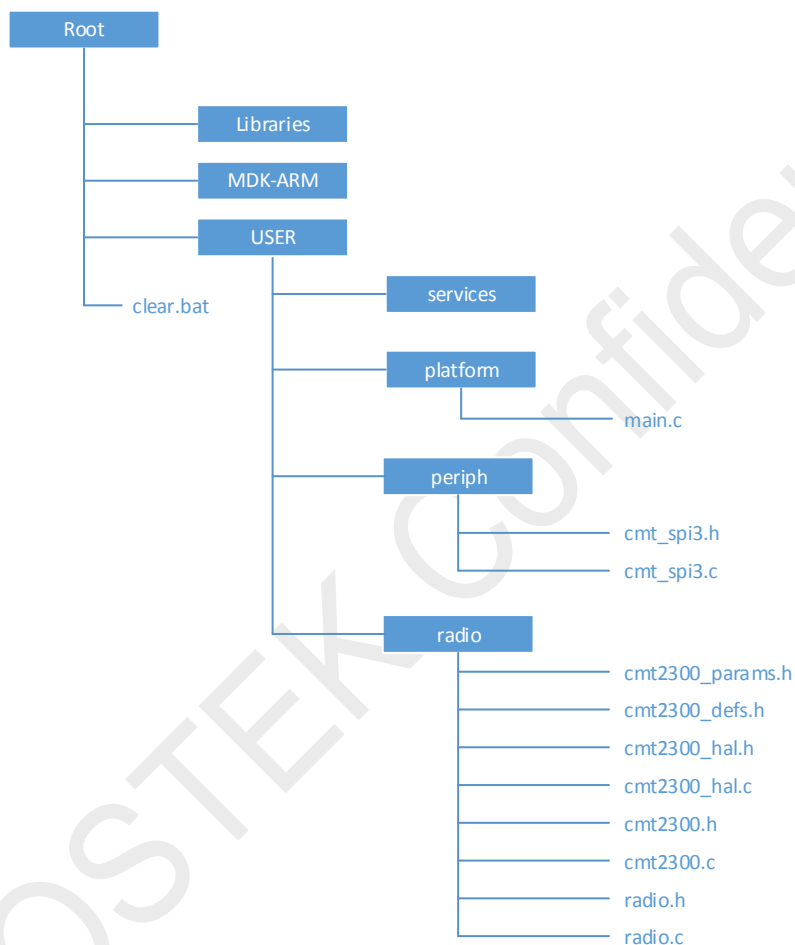


图 14. 软件目录结构

4.3.1 应用层源代码

main.c 为应用层源代码，配置 g_bEnableMaster 选择主机或者从机，分别调用 OnMaster(), OnSlave()来进行 CMT2300 收发通讯。

```

#define RF_PACKET_SIZE 32                /* Define the payload size here */

static u8 g_rxBuffer[RF_PACKET_SIZE];    /* RF Rx buffer */
static u8 g_txBuffer[RF_PACKET_SIZE];    /* RF Tx buffer */
  
```

```
static BOOL g_bEnableMaster = TRUE;    /* Master/Slave selection */

void Mcu_Init(void);

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

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

4.3.2 模拟 SPI 实现源代码

cmt_spi3.c 是针对 CMT2300A 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_spi3_csb_out()      SET_GPIO_OUT(CMT_CSB_GPIO)
#define cmt_spi3_fcsb_out()    SET_GPIO_OUT(CMT_FCSB_GPIO)
#define cmt_spi3_scl_out()     SET_GPIO_OUT(CMT_SCL_GPIO)
#define cmt_spi3_sda_out()     SET_GPIO_OUT(CMT_SDA_GPIO)
#define cmt_spi3_sda_in()      SET_GPIO_IN(CMT_SDA_GPIO)

#define cmt_spi3_csb_1()       SET_GPIO_H(CMT_CSB_GPIO)
#define cmt_spi3_csb_0()       SET_GPIO_L(CMT_CSB_GPIO)

#define cmt_spi3_fcsb_1()      SET_GPIO_H(CMT_FCSB_GPIO)
#define cmt_spi3_fcsb_0()      SET_GPIO_L(CMT_FCSB_GPIO)

#define cmt_spi3_scl_1()       SET_GPIO_H(CMT_SCL_GPIO)
#define cmt_spi3_scl_0()       SET_GPIO_L(CMT_SCL_GPIO)

#define cmt_spi3_sda_1()       SET_GPIO_H(CMT_SDA_GPIO)
#define cmt_spi3_sda_0()       SET_GPIO_L(CMT_SDA_GPIO)
#define cmt_spi3_sda_read()    READ_GPIO_PIN(CMT_SDA_GPIO)

/* ***** */
```

4.3.3 抽象硬件层源代码

cmt2300_hal.c 为抽象硬件层源代码，实现 CMT2300A 寄存器，FIFO，GPIO 访问接口。

```

/*! *****
 * @name    Cmt2300_InitGpio
 * @desc    Initializes the CMT2300A interface GPIOs.
 * *****/
void Cmt2300_InitGpio(void);

/*! *****
 * @name    Cmt2300_ReadReg
 * @desc    Read the CMT2300 register at the specified address.
 * @param   addr: register address
 * @return  Register value
 * *****/
u8 Cmt2300_ReadReg(u8 addr);

/*! *****
 * @name    Cmt2300_WriteReg
 * @desc    Write the CMT2300A register at the specified address.
 * @param   addr: register address
 *          dat: register value
 * *****/
void Cmt2300_WriteReg(u8 addr, u8 dat);

/*! *****
 * @name    Cmt2300_ReadFifo
 * @desc    Reads the contents of the CMT2300A FIFO.
 * @param   buf: buffer where to copy the FIFO read data
 *          len: number of bytes to be read from the FIFO
 * *****/
void Cmt2300_ReadFifo(u8 buf[], u16 len);

/*! *****
 * @name    Cmt2300_WriteFifo
 * @desc    Writes the buffer contents to the CMT2300A FIFO.
 * @param   buf: buffer containing data to be put on the FIFO
 *          len: number of bytes to be written to the FIFO
 * *****/
void Cmt2300_WriteFifo(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 cmt2300_hal.h 中的一些宏定义。

```
/* *****
* The following need to be modified by user
* ***** */
#define Cmt2300_SetGpio1In()      SET_GPIO_IN(CMT_GPIO1_GPIO)
#define Cmt2300_SetGpio2In()      SET_GPIO_IN(CMT_GPIO2_GPIO)
#define Cmt2300_SetGpio3In()      SET_GPIO_IN(CMT_GPIO3_GPIO)
#define Cmt2300_ReadGpio1()       READ_GPIO_PIN(CMT_GPIO1_GPIO)
#define Cmt2300_ReadGpio2()       READ_GPIO_PIN(CMT_GPIO2_GPIO)
#define Cmt2300_ReadGpio3()       READ_GPIO_PIN(CMT_GPIO3_GPIO)
#define Cmt2300_DelayMs(ms)        system_delay_ms(ms)
#define Cmt2300_DelayUs(us)        system_delay_us(us)
#define Cmt2300_GetTickCount()     g_nSysTickCount
/* ***** */
```

4.3.4 芯片驱动层源代码

cmt2300.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    Cmt2300_Init
* @desc    Initialize chip status.
* *****/
void Cmt2300_Init(void)
{
    u8 tmp;

    Cmt2300_SoftReset();
    Cmt2300_DelayMs(20);

    Cmt2300_GoStby();

    tmp = Cmt2300_ReadReg(CMT2300_CUS_MODE_STA);
    tmp |= CMT2300_MASK_CFG_RETAIN;          /* Enable CFG_RETAIN */
    tmp &= ~CMT2300_MASK_RSTN_IN_EN;        /* Disable RSTN_IN */
    Cmt2300_WriteReg(CMT2300_CUS_MODE_STA, tmp);

    Cmt2300_EnableLfosc(FALSE);              /* Diabale LFOSC */

    Cmt2300_ClearInterruptFlags();
}
```

4.3.5 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_spi3_write(u8 addr, u8 dat)
{
    cmt_spi3_sda_1();
    cmt_spi3_sda_out();

    cmt_spi3_scl_0();
    cmt_spi3_scl_out();
    cmt_spi3_scl_0();

    cmt_spi3_fcsb_1();
    cmt_spi3_fcsb_out();
    cmt_spi3_fcsb_1();

    cmt_spi3_csb_0();

    cmt_spi3_delay();    /* > 0.5 SCL cycle */
    cmt_spi3_delay();

    cmt_spi3_send(addr&0x7F);    /* r/w = 0 */

    cmt_spi3_send(dat);

    cmt_spi3_scl_0();

    /* > 0.5 SCL cycle */
    cmt_spi3_delay();
    cmt_spi3_delay();

    cmt_spi3_csb_1();

    cmt_spi3_sda_1();
    cmt_spi3_sda_in();

    cmt_spi3_fcsb_1();
}

void cmt_spi3_read(u8 addr, u8* p_dat)
{
    cmt_spi3_sda_1();
    cmt_spi3_sda_out();
```



```
cmt_spi3_scl_0();
cmt_spi3_scl_out();
cmt_spi3_scl_0();

cmt_spi3_fcsb_1();
cmt_spi3_fcsb_out();
cmt_spi3_fcsb_1();

cmt_spi3_csb_0();

cmt_spi3_delay();    /* > 0.5 SCL cycle */
cmt_spi3_delay();

cmt_spi3_send(addr|0x80);    /* r/w = 1 */

cmt_spi3_sda_in();    /* Must set SDA to input before the falling edge of SCL */

*p_dat = cmt_spi3_recv();

cmt_spi3_scl_0();

/* > 0.5 SCL cycle */
cmt_spi3_delay();
cmt_spi3_delay();

cmt_spi3_csb_1();

cmt_spi3_sda_1();
cmt_spi3_sda_in();

cmt_spi3_fcsb_1();
}
```

5.2 附录 2 Sample code SPI 读写 FIFO 操作代码示例

Sample code SPI 写 FIFO

```
void cmt_spi3_write_fifo(const u8* p_buf, u16 len)
{
    u16 i;

    cmt_spi3_fcsb_1();
    cmt_spi3_fcsb_out();
```

```
cmt_spi3_fcsb_1();

cmt_spi3_csb_1();
cmt_spi3_csb_out();
cmt_spi3_csb_1();

cmt_spi3_scl_0();
cmt_spi3_scl_out();
cmt_spi3_scl_0();

cmt_spi3_sda_out();

for(i=0; i<len; i++)
{
    cmt_spi3_fcsb_0();

    /* > 1 SCL cycle */
    cmt_spi3_delay();
    cmt_spi3_delay();

    cmt_spi3_send(p_buf[i]);

    cmt_spi3_scl_0();

    /* > 2 us */
    cmt_spi3_delay_us();
    cmt_spi3_delay_us();
    cmt_spi3_delay_us();

    cmt_spi3_fcsb_1();

    /* > 4 us */
    cmt_spi3_delay_us();
    cmt_spi3_delay_us();
    cmt_spi3_delay_us();
    cmt_spi3_delay_us();
    cmt_spi3_delay_us();
    cmt_spi3_delay_us();
}

cmt_spi3_sda_in();
cmt_spi3_fcsb_1();
}
```

Sample code SPI 读 FIFO 子函数示例:

```
void cmt_spi3_read_fifo(u8* p_buf, u16 len)
{
    u16 i;

    cmt_spi3_fcsb_1();
    cmt_spi3_fcsb_out();
    cmt_spi3_fcsb_1();

    cmt_spi3_csb_1();
    cmt_spi3_csb_out();
    cmt_spi3_csb_1();

    cmt_spi3_scl_0();
    cmt_spi3_scl_out();
    cmt_spi3_scl_0();

    cmt_spi3_sda_in();

    for(i=0; i<len; i++)
    {
        cmt_spi3_fcsb_0();

        /* > 1 SCL cycle */
        cmt_spi3_delay();
        cmt_spi3_delay();

        p_buf[i] = cmt_spi3_recv();

        cmt_spi3_scl_0();

        /* > 2 us */
        cmt_spi3_delay_us();
        cmt_spi3_delay_us();
        cmt_spi3_delay_us();

        cmt_spi3_fcsb_1();

        /* > 4 us */
        cmt_spi3_delay_us();
        cmt_spi3_delay_us();
        cmt_spi3_delay_us();
        cmt_spi3_delay_us();
    }
}
```

```

        cmt_spi3_delay_us();
        cmt_spi3_delay_us();
    }

    cmt_spi3_sda_in();
    cmt_spi3_fcsb_1();
}

```

5.3 附录 3 Sample code 状态切换库函数代码示例

```

/*! *****
* @name    Cmt2300_GoSleep
* @desc    Entry SLEEP mode.
* @return  TRUE or FALSE
* *****/
BOOL Cmt2300_GoSleep(void)
{
    Cmt2300_WriteReg(CMT2300_CUS_MODE_CTL, CMT2300_GO_SLEEP);
    return Cmt2300_WaitChipStatus(CMT2300_STA_SLEEP);
}

/*! *****
* @name    Cmt2300_GoStby
* @desc    Entry Sleep mode.
* @return  TRUE or FALSE
* *****/
BOOL Cmt2300_GoStby(void)
{
    Cmt2300_WriteReg(CMT2300_CUS_MODE_CTL, CMT2300_GO_STBY);
    return Cmt2300_WaitChipStatus(CMT2300_STA_STBY);
}

/*! *****
* @name    Cmt2300_GoTx
* @desc    Entry Tx mode.
* @return  TRUE or FALSE
* *****/
BOOL Cmt2300_GoTx(void)
{
    Cmt2300_WriteReg(CMT2300_CUS_MODE_CTL, CMT2300_GO_TX);
    return Cmt2300_WaitChipStatus(CMT2300_STA_TX);
}

/*! *****
* @name    Cmt2300_GoRx

```

```

* @desc    Entry Rx mode.
* @return  TRUE or FALSE
* *****/
BOOL Cmt2300_GoRx(void)
{
    Cmt2300_WriteReg(CMT2300_CUS_MODE_CTL, CMT2300_GO_RX);
    return Cmt2300_WaitChipStatus(CMT2300_STA_RX);
}

```

5.4 附录 4 Sample code 初始化函数代码示例

```

void RF_Init(void)
{
    Cmt2300_InitGpio();
    Cmt2300_Init();

    /* Config registers */
    Cmt2300_ConfigRegBank(CMT2300_CMT_BANK_ADDR,
g_cmt2300CmtBank,
CMT2300_CMT_BANK_SIZE);
    Cmt2300_ConfigRegBank(CMT2300_SYSTEM_BANK_ADDR,
g_cmt2300SystemBank,
CMT2300_SYSTEM_BANK_SIZE);
    Cmt2300_ConfigRegBank(CMT2300_FREQUENCY_BANK_ADDR, g_cmt2300FrequencyBank,
CMT2300_FREQUENCY_BANK_SIZE );
    Cmt2300_ConfigRegBank(CMT2300_DATA_RATE_BANK_ADDR , g_cmt2300DataRateBank,
CMT2300_DATA_RATE_BANK_SIZE );
    Cmt2300_ConfigRegBank(CMT2300_BASEBAND_BANK_ADDR ,
g_cmt2300BasebandBank ,
CMT2300_BASEBAND_BANK_SIZE );
    Cmt2300_ConfigRegBank(CMT2300_TX_BANK_ADDR,
g_cmt2300TxBank,
CMT2300_TX_BANK_SIZE);

    RF_Config();
}

```

6 文档变更记录

表 18. 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 | 2017-03-22 |

7 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

Copyright. CMOSTEK Microelectronics Co., Ltd. All rights are reserved.

The information furnished by CMOSTEK is believed to be accurate and reliable. However, no responsibility is assumed for inaccuracies and specifications within this document are subject to change without notice. The material contained herein is the exclusive property of CMOSTEK and shall not be distributed, reproduced, or disclosed in whole or in part without prior written permission of CMOSTEK. CMOSTEK products are not authorized for use as critical components in life support devices or systems without express written approval of CMOSTEK. The CMOSTEK logo is a registered trademark of CMOSTEK Microelectronics Co., Ltd. All other names are the property of their respective owners.