CMOSTEK AN142

CMT2300AW Quick Start Guide

Introduction

This document provides the basic method of use and the introduction of the relevant registers for the users developing the product with CMT2300AW, so that the users refer to the description and usage of each register in the process of using.

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

Table 1. Part Numbers Covered in this Document

Product	Frequency	Modem	Function	Configuration	Package
CMT2300AW	140 - 1020MHz	(G)FSK/OOK	Transceiver	Register	QFN16

Table of Contents

1. Chip Architecture Introduction

1.1 General Working Principle

CMT2300AW is a mixed-signal integrated transceiver. The product uses the 26 MHz crystal to provide the PLL reference frequency and the digital clock, while supporting the OOK and (G) FSK modulation and demodulation mode, and supports the Direct and Packet data processing mode.

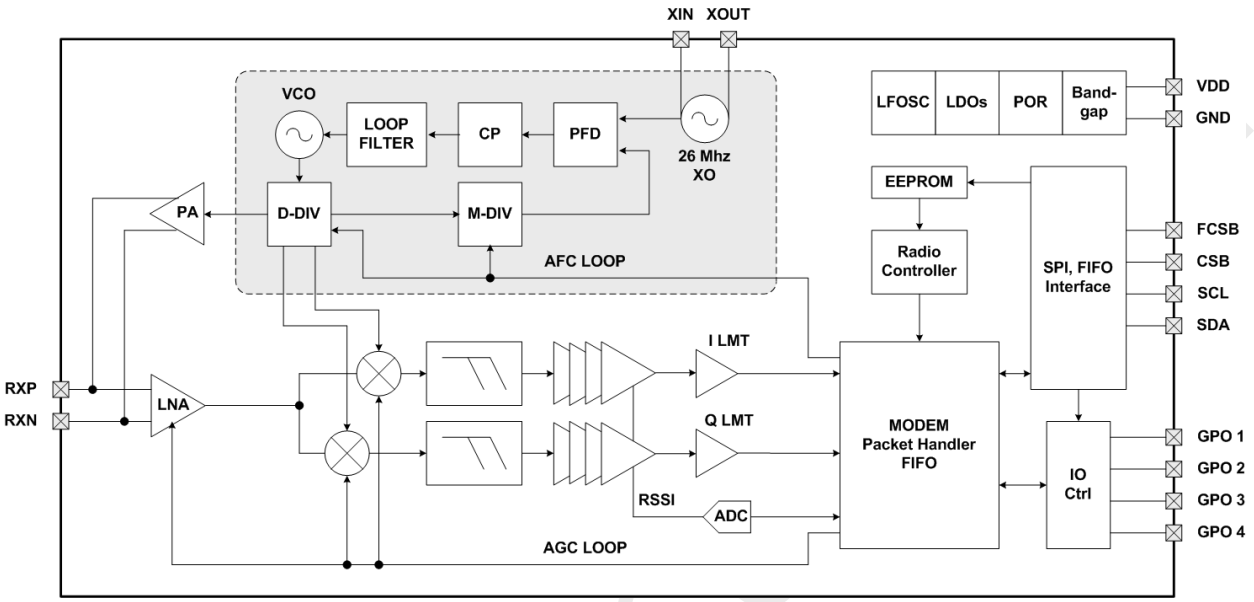

Figure 1. CMT2300AW System Block Diagram

In the receiver part, the chip uses LNA+MIXER+IFFILTER+LIMITTER+PLL low-IF architecture to achieve the Sub-GHz wireless reception function. The chip uses PLL+PA architecture to achieve the Sub-GHz wireless transmitting function.

In the receiver system, the analog circuit mixes the RF signal to IF and converts the signal from analog to digital through the Limiter module, then outputs I/Q two single bit signals to the digital circuit for (G) FSK demodulation. At the same time, SARADC will convert the real-time RSSI signal to 8-bit digital signal, and sent them to the digital part for OOK demodulation and other processing. The digital circuit is responsible for mixing the intermediate frequency to zero frequency (Baseband) and performing a series of filtering and decision processing, while AFC and AGC control the analog circuit dynamically, finally the 1-bit original signal is demodulated. After demodulation, the signal will be sent to the decoder to decode and fill in the FIFO, or output to the PAD directly.

In the transmitter system, the digital circuitry will encode the data and then send them to the modulator (or send them to the modulator directly without encoding). The modulator will directly control the PLL and PA, modulate the data by (G) FSK or OOK and transmit them.

The chip provides the SPI communication port. The external MCU can configure the various functions by accessing to the register, control the main state machine, and access to the FIFO.

1.2 Pin Descriptions

Now take the QFN16 package as an example to introduce the CMT2300AW pin distribution and function:

Table 2. CMT2300AW Pin Descriptions

Pin No.	Name	I/O	Descriptions
1	RFIP	I	RF signal input P
2	RFIN	I	RF signal input N
3	PA	O	PA output
4	AVDD	IO	Analog VDD
5	AGND	IO	Analog GND
6	DGND	IO	Digital GND
7	DVDD	IO	Digital VDD
8[1]	GPIO3	IO	Configured as CLKO, DOUT/DIN, INT2 and DCLK (TX/RX)
9	SCLK	I	SPI clock
10	SDIO	IO	SPI data input and output
11	CSB	I	SPI chip selection bar for register access, active low
12	FCSB	I	SPI chip selection bar for FIFO access ,active low
13	XI	I	Crystal circuit input
14	XO	O	Crystal circuit output
15[1]	GPIO2	IO	Configured as INT1, INT2, DOUT/DIN, DCLK (TX/RX) and RF_SWT
16[1]	GPIO1	IO	Configured as DOUT/DIN, INT1, INT2, DCLK (TX/RX) and RF_SWT
17	GND	I	Analog GND. It must be grounded.

Note:

[1]. INT1 and INT2 are interrupts. DOUT is demodulated output. DIN is a modulation input. DCLK is a modulation or demodulation data rate synchronization clock, automatic switching in TX/RX mode.

2. SPI Interface Timing

The chip communicates with the outside through the 4-wire SPI interface. The CSB is the selection bar for accessing to the register, active low. The FCSB is the selection bar for accessing to the FIFO, active low. Both cannot be set to low at the same time. The SCL is the serial clock that the fastest speed can be up to 5 MHz. The chip itself and the external MCU send the data at the falling edge of SCL and collect the data at the rising edge of SCL. The SDA is the bidirectional pin for input and output data. The address and data are transferred starting from MSB.

2.1 Read / Write Register Operation

When accessing the register, CSB is pulled low. Then send a R/W bit first, followed by a 7 bit register address. After the external MCU pulls down the CSB, it must wait for at least half a SCL cycle, and then send the R/W bit. After the MCU sends out the last falling edge of SCL, it must wait for at least half a SCL cycle, and then pull the CSB high.

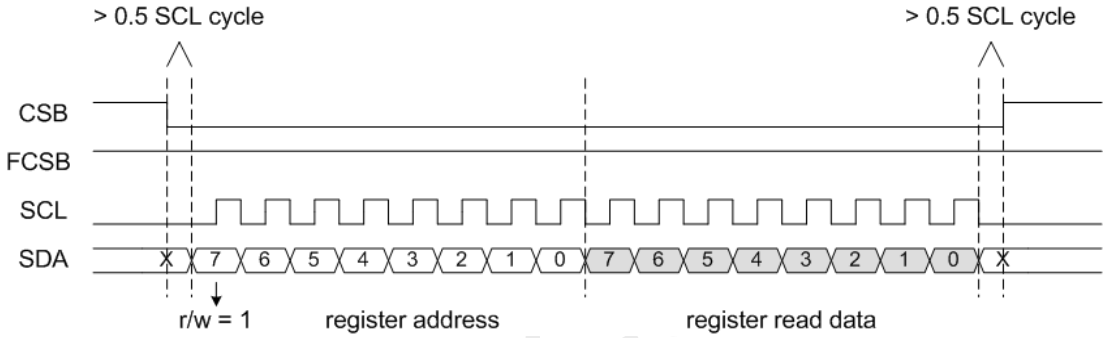

Figure 2. SPI Read Register Timing

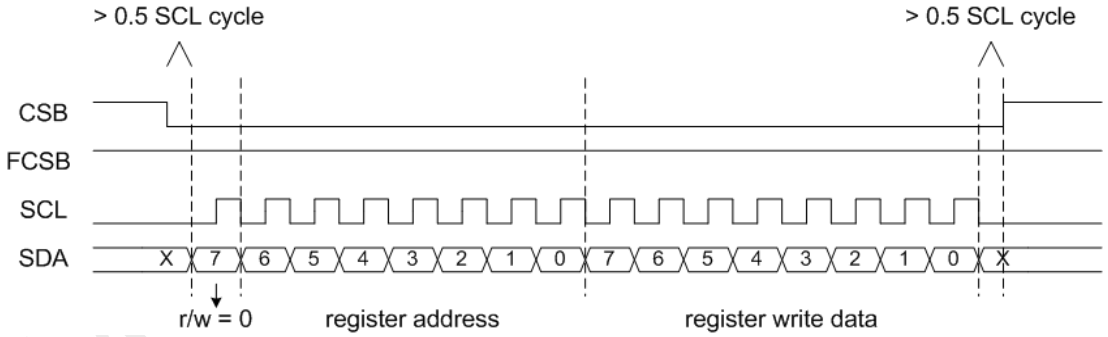

Figure 3. SPI Write Register Timing

SPI read-write register operation code examples see Appendix 1.

2.2 Read / Write FIFO Operation

When the MCU accesses to the FIFO, the user must first configure a few registers to set the FIFO read-write mode, as well as some other working mode. This will be introduced in the following chapters. Here is the read-write timing diagram after determining the mode. Note that there is a slight difference in the control of the FCSB and the control of the CSB when accessing the register. When the MCU starts to access to the FIFO, FCSB must be pulled down 1-clock cycle at first, and then send the rising edge of SCL. After the last falling edge of SCL is sent, the FCSB must be pulled high at least 2 us later. Between the two continuous read-write operations, the FCSB must be pulled high for 4 us at least. When writing the FIFO, the first bit data must be

ready 0.5 clock cycles before sending the first rising edge of SCL.

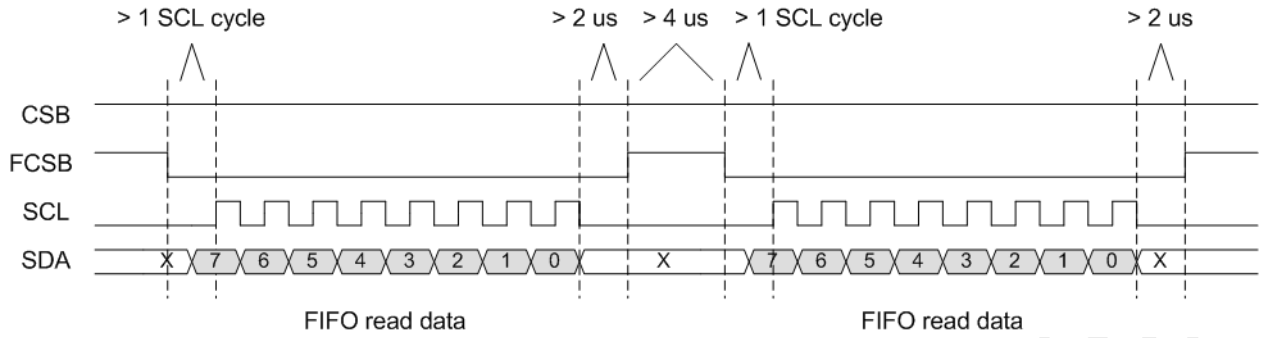

Figure 4. SPI Read FIFO Timing

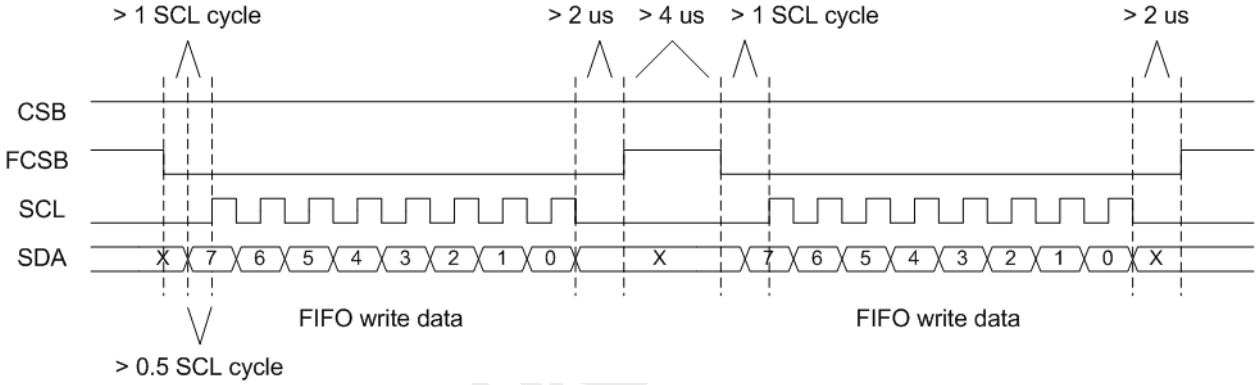

Figure 5. SPI Write FIFO Timing

SPI read-write FIFO operation code examples see Appendix 2.

3. Configuration and Control Mechanism

3.1 Register Overview

Table 3. CMT2300AW Register Overview

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	User does not need to understand the detail, just directly export the register contents from the RFPDK								CMT Bank
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									
Addr	**R/W**	**Name**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**	**Function**
0x0C	RW	CUS_SYS1	LMT_VTR [1:0]		MIXER_BIAS [1:0]		LNA_MODE [1:0]		LNA_BIAS [1:0]		System Bank
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				
Addr	**R/W**	**Name**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**	**Function**
0x18	RW	CUS_RF1	User does not need to understand the detail, just directly export the register contents from the RFPDK								Freq Bank
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									
Addr	**R/W**	**Name**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**	**Function**
0x20	RW	CUS_RF9	User does not need to understand the detail, just directly export the register contents from the RFPDK								Data Rate Bank
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									
Addr	**R/W**	**Name**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**	**Function**
0x38	RW	CUS_PKT1	RX_PREAM_SIZE [4:0]					PREAM_LENG_UNIT	DATA_MODE [1:0]		Baseband Bank
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	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]							
Addr	**R/W**	**Name**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**	**Function**
0x55	RW	CUS_TX1	User does not need to understand the detail, just directly export the register contents from the RFPDK								Tx Bank
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									
Addr	**R/W**	**Name**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**	**Function**
0x60	RW	CUS_MODE_CTL	CHIP_MODE_SWT [7:0]								Control Bank 1
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	
Addr	**R/W**	**Name**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**	**Function**
0x6B	W	CUS_INT_CLR2	RESV	RESV	LBD_CLR	PREAM_OK_CLR	SYNC_OK_CLR	NODE_OK_CLR	CRC_OK_CLR	PKT_DONE_CLR	Control Bank 2
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]								

The external MCU configures and controls the chip by accessing the series of registers through the SPI interface. The address from 0x00 to 0x7F can be divided into 3 major banks, as shown above. They are the configuration bank (including 6 sub banks), the control bank 1 and the control bank 2. The details are as below.

For the 3 banks, the address is continuous and the operation is no essential difference. They all directly read and write by accessing the register through the SPI interface. However, from the function and operation mode, the 3 banks have different roles, as shown below:

Table 4. CMT2300AW Register Partition Table

Bank	Address	Function descriptions
Configuration Bank	0x00– 0x5F	This bank is used to configure the chip. They include： **1. Reserved attributes for internal use** **2. System operation** **3. RF characteristics** **4. FSK demodulation** **5. Clock recovery** **6. AGC characteristics** **7. OOK demodulation** **8. Packet format and codec mode** **9. Modulation characteristics of TX** The value of the register can either come from the RFPDK, or can be changed by the customer in accordance with their own needs in the application process. In general, except that the individual register on the RF frequency or data rate may be configured multiple times in the application, most of the rest of the register only need to be configured one time in the initialization process.
Control Bank 1	0x60 – 0x6A	The bank is used to real-time control the chip. They include: **1. Chip state switch** **2. Special function enable** **3. Channel switching by hopping manually** **4. IO control** **5. Interrupt enable** **6. FIFO mode configuration** **7. Related interrupt control** Control bank 1 can be saved in the SLEEP state. As long as the battery is not powered down and the chip is not reset, the configuration content will not be lost.
Control Bank2	0x6B – 0x71	This bank is provided to the user to control the chip. They include: **1. Packet and FIFO interrupt control** **2. FIFO control** **3. RSSI read** **4. LBD function control** The control bank 2 is not saved in the SLEEP state. All the configured values and the values that can be read will disappear. Users need to pay attention to it.

3.2 Working State Switch

The external MCU can switch and query the chip working state by accessing the registers in the following control bank 1.

Table 5. Switch State Register

Register name	Bits	R/W	Bit name	Function descriptions
CUS_MODE_CTL (0x60)	7:0	RW	CHIP_MODE_SWT<7:0>	State switching command： 00000010：go_stby 00000100：go_rfs 00001000：go_rx 00010000：go_sleep 00100000：go_tfs 01000000：go_tx 10000000：go_switch Others：Not allowed to send.
CUS_MODE_STA (0x61)	3:0	RW	CHIP_MODE_STA<3:0>	Chip status： 0000：IDLE 0001：SLEEP 0010：STBY 0011：RFS 0100：TFS 0101：RX 0110：TX 1000：UNLOCKED/LOW_VDD 1001：CAL Others：Invalid UNLOCKED means that the PLL is not locked because the chip is disturbed seriously. LOW_VDD means that the LBD detects the VDD too low, not suitable for transmitting and receiving, the user can selectively allow the chip to stay in this state, followed by a detailed introduction. CAL is the calibration state. The chip will not stay in the state for a long time, so the user could not usually query the CAL state.

The state switching diagram is as follows. The UNLOCKED and CAL states are not shown in the following diagram because the user could not query them.

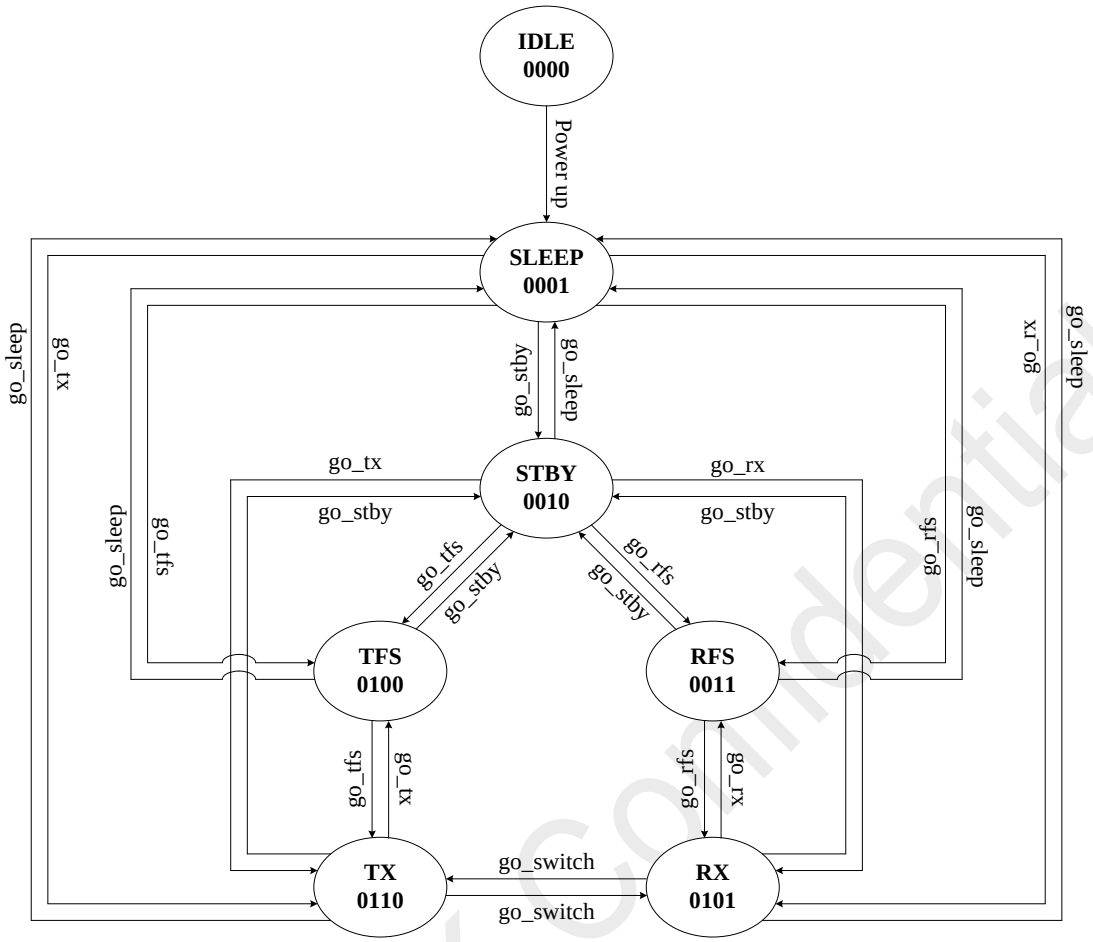

Figure 6. CMT2300AW State Switching Diagram

Table 6. CMT2300AW State and Module Opening Table

State	Binary code	Switching command	Opening module	Optional opening module
IDLE	0000	soft_rst	SPI, POR	None
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

After the MCU sends the go_* command, the chip sometimes needs to wait for a certain time to switch the state successfully. The following will explain each state and the waiting time for switching.

■ IDLE State and Power Up Flow

After the chip VDD is powered up, the chip usually needs to wait about 1ms, then POR will release. After the release of the POR,

the crystal will start, the start time default is N ms, according to the characteristics of the crystal itself. After starting, the user need to wait for the crystal settled, then the system start working. The default setting is 2.48ms. This time can be modified by writing XTAL_STB_TIME <2:0> afterword. The chip remains in the IDLE status until the crystal is settled. After the crystal is settled, the chip will leave the IDLE state and begin to do the calibration of each module. After the calibration is completed, the chip will stay in the SLEEP and wait until the user to initialize the configuration. At any time, as long as the external hard reset or soft reset is performed, the chip will go back to the IDLE and be powered up again.

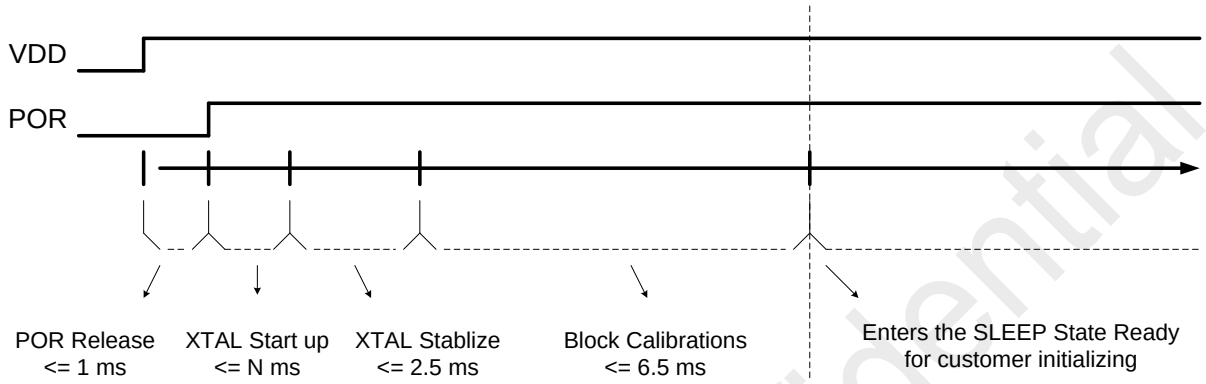

Figure 7. CMT2300AW Power-Up Flow Diagram

Block Calibrations contain the LFOSC calibration, to ensure that the frequency error of LFOSC is less than 1%. LFOSC is mainly used to drive the Sleep Timer. If the user does not need to use the Sleep Timer, or does not mind the low precision of the Sleep Timer, the user can turn off the LFOSC calibration. The way to turn off the LFOSC calibration is to configure the LFSOC_CAL1_EN and LFOSC_CAL2_EN to 0 in the configuration phase (introduced as below). After the soft reset next time, LFOSC calibration function will be closed, the user can save about 5ms. But in any case, the chip will be default to do a LFOSC calibration at the first power-up.

■ SLEEP State

The chip power consumption is the lowest in SLEEP state, and almost all the modules are turned off. SPI is open, the registers of the configuration bank and control bank 1 will be saved, and the contents filled in the FIFO before will remain unchanged. However, the user cannot operate the FIFO and cannot change the contents of the register. If the user opens the wake-up function, the LFOSC and the sleep counter will turn on and start working. The time required to switch from IDLE to SLEEP is the power up time. Switch from other state to SLEEP will be completed immediately.

■ STBY State

In STBY state, the crystal is turned on, the LDO of the digital circuit will also be turned on, the current will be slightly increased, and the FIFO can be operated. The user can choose whether to output CLKO (system clock) to PIN. Because the crystal and LDO is turned on, compared to the SLEEP, the time switching from the STBY to transmitting or receiving will be relatively short. Switching from SLEEP to STBY will be completed after the crystal is turned on and settled. Switching from other state to STBY will be completed immediately.

■ RFS State

RFS is a transition state before switching to RX. Except that the receiver RF module is off, the other modules are turned on, and the current will be larger than STBY. Because PLL has been locked in the RX frequency, RFS cannot switch to TX. Switching from STBY to RFS probably requires PLL calibration and stability time of 350us. Switching from SLEEP to RFS needs to add the crystal start-up and stability time. Switching from other state to RFS will be completed immediately.

■ TFS State

TFS is a transition state before switching to TX. Except that the transmitter RF module is off, the other modules are turned on, and the current will be larger than STBY. Because PLL has been locked in the TX frequency, TFS cannot switch to RX. Switching from STBY to TFS probably requires PLL calibration and stability time of 350us. Switching from SLEEP to TFS needs to add the crystal start-up and settled time. Switching from other state to TFS will be completed immediately.

■ RX State

All modules on the receiver will be opened in RX state. Switching from RFS to RX requires only 20us. Switching from STBY to RX needs to add the PLL calibration and settled time of 350us. Switching from SLEEP to RX needs to add the crystal start-up and settled time. TX can be quickly switched to RX by sending go_switch command. Whether the TX and RX setting frequency is the same, the user need to wait for the PLL re-calibration and settled time of 350us to switch successfully.

■ TX State

All modules on the transmitter will be opened in TX state. Switching from TFS to TX requires only 20us. Switching from STBY to TX needs to add the PLL calibration and settled time of 350us. Switching from SLEEP to TX needs to add the crystal start-up and settled time. RX can be quickly switched to TX by sending go_switch command. Whether the RX and TX setting frequency is the same, the user need to wait for the PLL re-calibration and settled time of 350us to switch successfully.

CMT2300AW state switching library function code examples see Appendix 3

3.3 Softrst

CMT2300AW soft reset is achieved by writing 0xFF to the address 0x7F through the SPI interface. After receiving this command, the chip will immediately reset and return to the IDLE status, and be immediately initialized again. So after the user sends the soft reset command, he could not query the IDLE status because the status is flash across.

3.4 RFPDK Profile

RFPDK is the software tool provided by CMOSTEK, so as to configure or burn the RF chip. It is installed in the Windows. For CMT2300AW, the role of RFPDK is to generate the register configuration file by users input parameters on the interface. Users can import the file into the MCU to configure the register of the CMT2300AW.

CMT2300AW configuration idea is, in the process of chip initialization, users first generate the register configuration file through RFPDK to initial the chip, and then according to the applications, users can flexibly configure and control some of the specific registers. Therefore, users only need to know how to operate the RFPDK, and to selectively understand the registers they need to understand.

The previous chapter introduces the registers. This chapter will introduce the RFPDK. And then the subsequent chapters will introduce how to combine the both to carry out the configuration work.

Here is the screenshot of RFPDK:

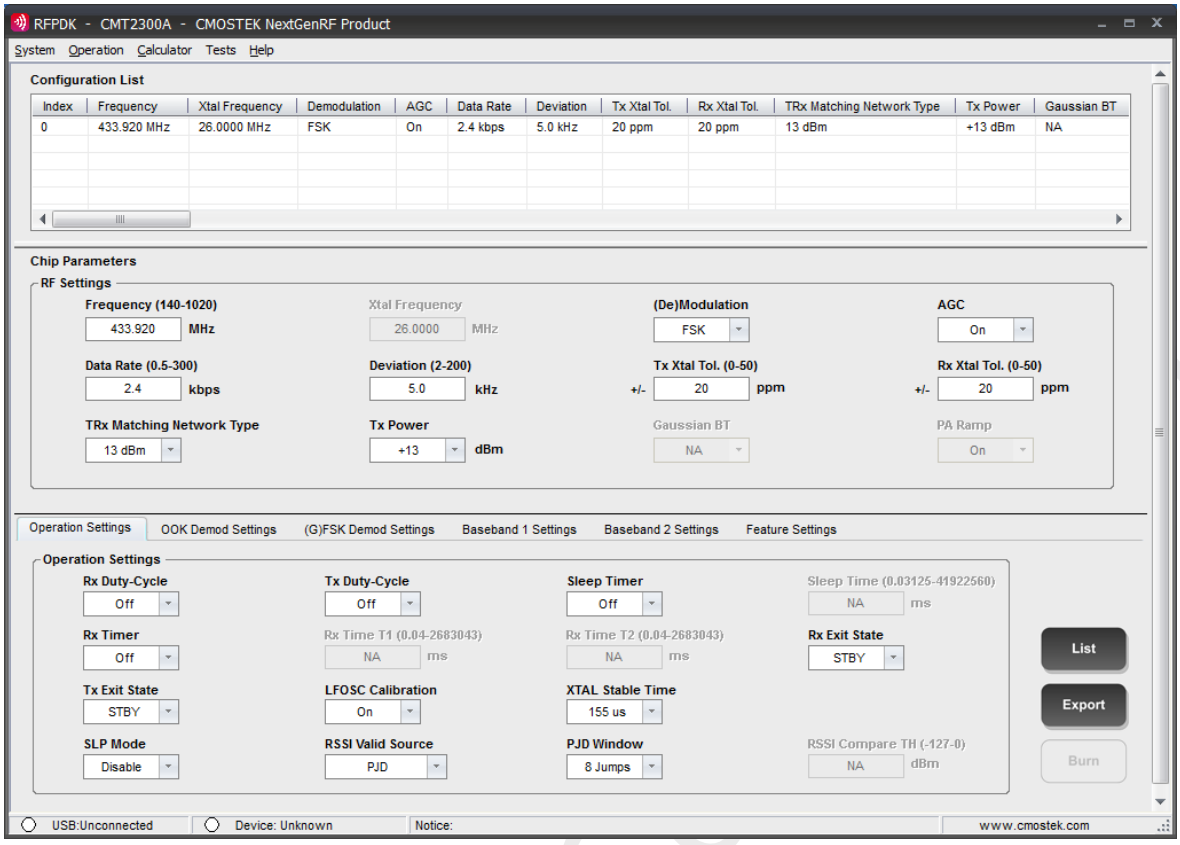

Figure 8. CMT2300AW RFPDK

There are seven main plates in the interface.

Table 7. RFPDK Main Plates

RFPDK plates	Configuration parameters	Relationship between RFPDK input parameters and registers
RF Settings	Frequency, data rate, Tx power and other RF parameters	The input parameters are calculated to generate the register contents. Users do not need to understand them.
Operation Settings	System operating parameters	Input parameters correspond to registers, users can comprehend.
OOK Demod Settings	OOK demodulation parameters	The input parameters are calculated to generate the register contents. Users do not need to understand them. In general, users can use the default parameters.
FSK Demod Settings	FSK demodulation parameters	The input parameters are calculated to generate the register contents. Users do not need to understand them. In general, users can use the default parameters.
Baseband 1 Settings	Baseband codec and FIFO parameters	The input parameters correspond to registers, users can comprehend.
Baseband 2 Settings	Baseband packet format parameters	The input parameters correspond to registers, users can comprehend.
Feature Settings	Other function parameters	The input parameters correspond to registers, users can comprehend.

The following is an example of generating the configuration file.

Firstly the comment at the top of the file is used to confirm the current successful configuration information. They are usually used to check the error. In general, users do not need too much attention.

```
;------------------------------------------------------------------------------
;   CMT2300AW Configuration File
;   Generated by CMOSTEK RFPDK 1.41 Beta
;   2017.02.13 16:26
;------------------------------------------------------------------------------

; Mode                      = Advanced
; Part Number               = CMT2300AW
; 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

The second part of the file list the register values for the chip initiation, the value are in hex format. For easier read, they are divided into 6 sub-sections, shown as below.

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

[CMT Bank]

Addr	Value
0x00	0x00
0x01	0x66
0x02	0x6C
0x03	0x7C
0x04	0xF0
0x05	0x80
0x06	0x14
0x07	0x08
0x08	0xB1
0x09	0x03
0x0A	0x00
0x0B	0xD0

[System Bank]

Addr	Value
0x0C	0xA5
0x0D	0xE0
0x0E	0x30
0x0F	0x00
0x10	0x00
0x11	0xF4
0x12	0x10
0x13	0xE2
0x14	0x42
0x15	0x20
0x16	0x00
0x17	0x81

[Frequency Bank]

Addr	Value
0x18	0x42
0x19	0x71
0x1A	0xCE
0x1B	0x1C
0x1C	0x42
0x1D	0x5B
0x1E	0x1C
0x1F	0x1C

[Data Rate Bank]

Addr	Value
0x20	0x32
0x21	0x18
0x22	0x00
0x23	0x99
0x24	0xC1
0x25	0x9B
0x26	0x06
0x27	0x0A
0x28	0x9F
0x29	0x39
0x2A	0x29
0x2B	0x29
0x2C	0xC0
0x2D	0x51
0x2E	0x2A
0x2F	0x53
0x30	0x00
0x31	0x00
0x32	0xB4
0x33	0x00

[Baseband Bank]

Addr	Value
0x38	0x12
0x39	0x08
0x3A	0x00
0x3B	0xAA
0x3C	0x02
0x3D	0x00
0x3E	0x00
0x3F	0x00
0x40	0x00
0x41	0x00
0x42	0x00
0x43	0x34
0x44	0x12
0x45	0x00
0x46	0x1F
0x47	0x00
0x48	0x00
0x49	0x00
0x4A	0x00
0x4B	0x00

[TX Bank]

Addr	Value
0x55	0x50
0x56	0x06
0x57	0x03
0x58	0x00
0x59	0x44
0x5A	0xD0
0x5B	0x00
0x5C	0x72
0x5D	0x0C
0x5E	0x3F
0x5F	0x7F

[CMT Bank]	[System Bank]	[Frequency Bank]	[Data Rate Bank]	[Baseband Bank]	[TX Bank]
			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	

The last part is the CRC value for internal use, the user can ignore it.

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

0xDE67

3.5 Chip Initialization Process

Users need to use RFPDK to generate the configuration bank (0x00 - 0x5F) contents for the MCU to initialize the configuration. After the chip is powered up or reset, it will automatically enter the SLEEP to wait for MCU operation. MCU can be operated according to the following initialization process.

1. Confirm that the chip is powered up or reset and remains in the SLEEP state.
2. Send the go_stby command and confirm that the chip is in the STBY state.
3. Write the register contents generated by RFPDK to the configuration bank. The address is 0x00 - 0x5F.
4. If the user does not need to use SLEEP TIMER, configure the RECAL_LFOSC_EN, LFSOC_CAL1_EN and LFOSC_CAL2_EN to 0.
5. According to actual needs, set the register of the control bank 1 (0x60 - 0x6A), the RSTN_IN_EN is set to 0.
6. The 4thbit CFG_RETAIN of the CUS_MODE_STA（0x61） register is configured to 1. This bit will allow the value of the 0x00 - 0x5F configuration bank not to be erased by the soft reset.
7. Send the go_sleep command, which allows the register configuration to take effect.

After the initialization is completed, the chip can start working. Afterwards if the user wants to change the whole or part of the register; the user must safely switch the chip and stay in the STBY state, and then configure them. After finishing the operation above, the user must switch to SLEEP state to allow them to take effect. Whenever the user wants to change the configuration register, he must follow this step.

In addition, if the chip is in a fully automatic duty cycle mode, the MCU must first allow the chip to exit the Duty Cycle mode, and then configure the parameters as described above. Details about entering and exiting the Duty Cycle mode are described in AN document.

CMT2300AW initialization code operations see Appendix 4.

3.6 Function Partitions of Configuration Bank

As mentioned earlier, the entire configuration bank is divided into 6 banks in accordance with the function. They correspond to the partition of the register file generated by RFPDK. When users need to change one function, they can directly modify the corresponding bank according to the use guidelines. This does not involve other registers and does not study and understand the contents of a specific register, because they are generated by the RFPDK.

For example, users want to modify the data rate in the application. Firstly configure the data rate by RFPDK, then export the entire register configuration, and then extract the contents of data rate bank as an array to write in the MCU program. When the program needs to modify the data rate, the user can directly write this part of the content to the register address corresponding to the chip.

The following list will introduce how to divide the 6 sub banks in the FSK and OOK mode. The both main difference is the partition of data rate.

Table 8. CMT2300AW Register Configuration Bank Partition Table in FSK Mode

Bank	Address	Bank name
CMT Bank	0x00 – 0x0B	CMT Bank
System Bank	0x0C – 0x17	System Bank
Frequency Bank	0x18 – 0x1F	Frequency Bank
Data Rate Bank	Filling bank：0x20 – 0x32	Data Rate Bank
	Ignorable bank：0x33– 0x37	
Baseband Bank	0x38 – 0x54	Baseband Bank
TX Bank	0x55 – 0x5F	TX Bank

Table 9. CMT2300AW Register Configuration Bank Partition Table in OOK Mode

Bank	Address	Bank name
CMT Bank	0x00 – 0x0B	CMT Bank
System Bank	0x0C – 0x17	System Bank
Frequency Bank	0x18 – 0x1F	Frequency Bank
Data Rate Bank	Filling bank：0x20 – 0x23	Data Rate Bank
	Ignorable bank：0x24– 0x2A	
	Filling bank：0x2B – 0x37	
Baseband Bank	0x38 – 0x54	Baseband Bank
TX Bank	0x55 – 0x5F	TX Bank

The ignorable bank in the table means that users do not need to configure them. The following is a brief introduction to the content of each bank, including the CMT bank, the frequency bank, the data rate bank. Users do not need to understand the TX bank register and directly import the parameter generated by RFPDK.

3.6.1 CMT Bank（0x00 – 0x0B）

CMT bank is used internally for CMOSTEK. Users can use the contents generated by RFPDK and directly fill in.

Table 10. CMT Bank

Addr	R/W	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x00	RW	CUS_CMT1	User does not need to understand the detail, just directly export the register contents from the RFPDK							
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 System Bank（0x0C – 0x17）

Table 11. System Bank

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				

The parameters of System Bank control the system operation mode and some system features, such as whether to turn on the SLEEP TIMER, and whether to open the DUTY CYCL, how to use RSSI or PJD (phase jump detection) for the ultra low power receiving.

3.6.3 Frequency Bank（0x18 – 0x1F）

Frequency Band is used to configure RX and TX frequency. Users can use the contents generated by RFPDK and directly fill in. If users want to change the frequency point in the application, there are two ways:

1. Using RFPDK generate the corresponding parameter of every frequency. If the user wants to change the frequency, the user can write the register of the entire Frequency Bank again.
2. The frequency point set in the initialization Frequency Bank is used as a base frequency. Users can switch the new frequency by the Fast Frequency Hopping mechanism.

Table 12. Frequency Bank

Addr	R/W	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x18	RW	CUS_RF1	User does not need to understand the detail, just directly export the register contents from the RFPDK							
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								

For some special applications, the TX and RX frequency need to be configured separately. The user needs to operate the specific register. For details, please refer to CMOSTEK technical support.

3.6.4 Data Rate Bank（0x20 – 0x37）

As described earlier, the address of Data Rate Bank can be divided into two cases of FSK and OOK. The ignorable part is not filled by the user.

There are a lot of configuration contents in Data Rate Bank. They cover more addresses because different data rates will affect the demodulation parameters of FSK or OOK, the clock recovery and the AGC configuration,. If users want to change the data rate in the application, it is necessary to pre store the corresponding Data Rate Bank parameters generated by RFPDK, and then write the contents of the whole Data Rate Bank. (Except ignorable bank)

TX and RX data rates are configured uniformly. If users need to use the different TX and RX data rate, it is necessary to reconfigure the data rate before switching TX/RX. However, the effective register address for the TX data rate is 0x20, 0x21 and 0x22. If users switch from RX to TX and change the data rate, it is only necessary to change the 3 registers. Instead, users switch from TX to RX and change the data rate, it is necessary to write the contents of the whole Data Rate Bank..

Table 13. Data Rate Bank

Addr	R/W	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x20	RW	CUS_RF9	User does not need to understand the detail, just directly export the register contents from the RFPDK							
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 Baseband Bank（0x38 – 0x54）

Table 14. Baseband Bank

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	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]						

Baseband Bank is mainly used to configure the packet format and codec of the transmitting and receiving data, as well as part of the FIFO configuration (FIFO usage mode and control registers are mainly placed in the Control Bank). This part of the content and the RFPDK parameter is one-to-one match, the user needs to learn more about them and then configure them.

3.6.6 TX Bank（0x55 – 0x5F）

TX Bank is mainly used to configure the TX parameters. Because the data rates for TX and RX are unified, they are classified into Data Rate Bank.

TX configuration correlation is relatively small. If users want to change a single feature, such as Tx Power, Deviation or LBD, they do not need to write the entire TX Bank, only need to configure several registers. Therefore, for TX Bank, users do not need to understand the meaning and calculation method of each register, only need to know when to change which registers. For details, please refer to the AN document.

Table 15. TX Bank

Addr	R/W	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x55	RW	CUS_TX1	User does not need to understand the detail, just directly export the register contents from the RFPDK							
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 Control Bank Introduction

As described earlier, Control Bank is divided into Control Bank 1 and Control Bank 2. Control Bank 1 can be saved in SLEEP state. Control Bank 2 can not to be saved in SLEEP state.

Control Bank 1（0x60 – 0x6A）

Table 16. Control Bank 1

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

Control Bank 2（0x6B – 0x71）

Table 17. Control Bank 2

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]							

Users can operate the Control Bank to achieve the chip working mode switching, IO and interrupt control, FIFO control, Fast Frequency Hopping, RSSI read, etc. The register of Control Bank is used to operate frequently in the application program.

3.8 Process Summary

Summarizing the above information, the main process of the user operating the chip actually includes 3 steps:

1. Read the relevant AN documents, using RFPDK to generate the desired register file.
2. Initial the chip configuration.
3. Execute the application program. There are two major types of operations in the application program.
   a) Modify the configuration bank—need to know the modified configuration bank belongs to which function bank in this configuration bank. Generate a new configuration table with RFPDK, intercept the modified bank and import into the program. Write the contents to the corresponding register address when needed.
   b) Control the chip operation—need to understand the register functions of Control Bank 1 and Control Bank 2. Read each chip characteristics and operating rules in AN document, including the FIFO control, IO and interrupt control, RSSI read, manual frequency hopping, etc. And then operate the chip to meet the application requirement.

4. CMT2300AW_DemoEasy Introduction

4.1 Software Hierarchy Architecture

In order to standardize the CMT2300AW operation process and enhance the portability, the Demo program does the hierarchy processing and each module calls down the corresponding API function. They are divided into Application, Radio handlers, CMT2300AW drivers, Hardware abstraction layer and Hardware.

i. Application: Application layer, simply sending and receiving the data packets in Demo.
ii. Radio handlers: Chip processing layer, including the chip initialization, configuration, state control and other processes.
iii. CMT2300 drivers: Chip driver layer, called by the upper layer.
iv. Hardware abstraction layer: Access and control the chip register, FIFO and GPIO.
v. Hardware: Hardware layer, including LED, buttons, SPI communications and other resources provided by MCU.

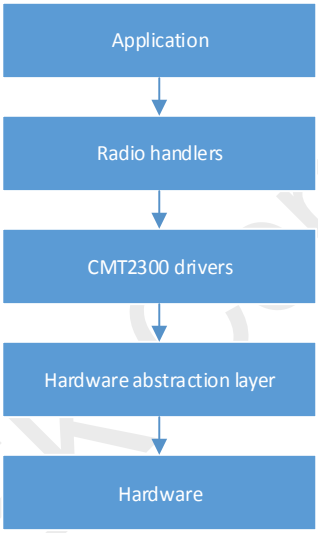

Figure 9. Software Hierarchy Architecture

4.2 Software Realization and Calling Relation

This part introduces the calling relation among modules, the system initialization process and workflow. Mainly introduces main.c, radio.c, cmt2300.c, cmt2300_hal.c and other documents.

1> main.c: Application layer realization
2> radio.c: Chip processing layer realization
3> cmt2300.c: Chip driver layer realization
4> cmt2300_hal.c: Hardware abstraction layer realization
5> cmt_spi3.c: Chip SPI timing realization, including accessing the register and the FIFO timing.

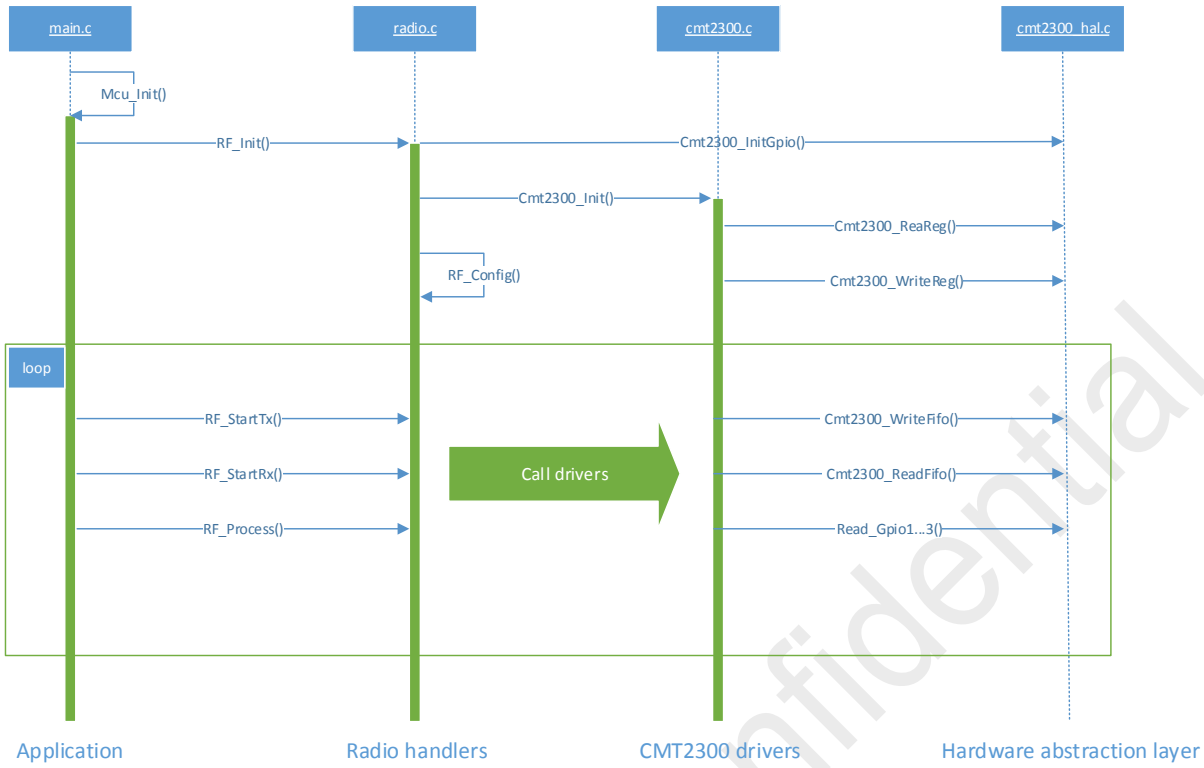

Figure 10. Software Realization and Calling Relation

4.2.1 CMT2300AW Initialization

After the chip is powered up, it must call the RF_Init () function and implement an initialization, including initializing SPI, GPIO1/2/3, soft reset, enable or disable some registers.

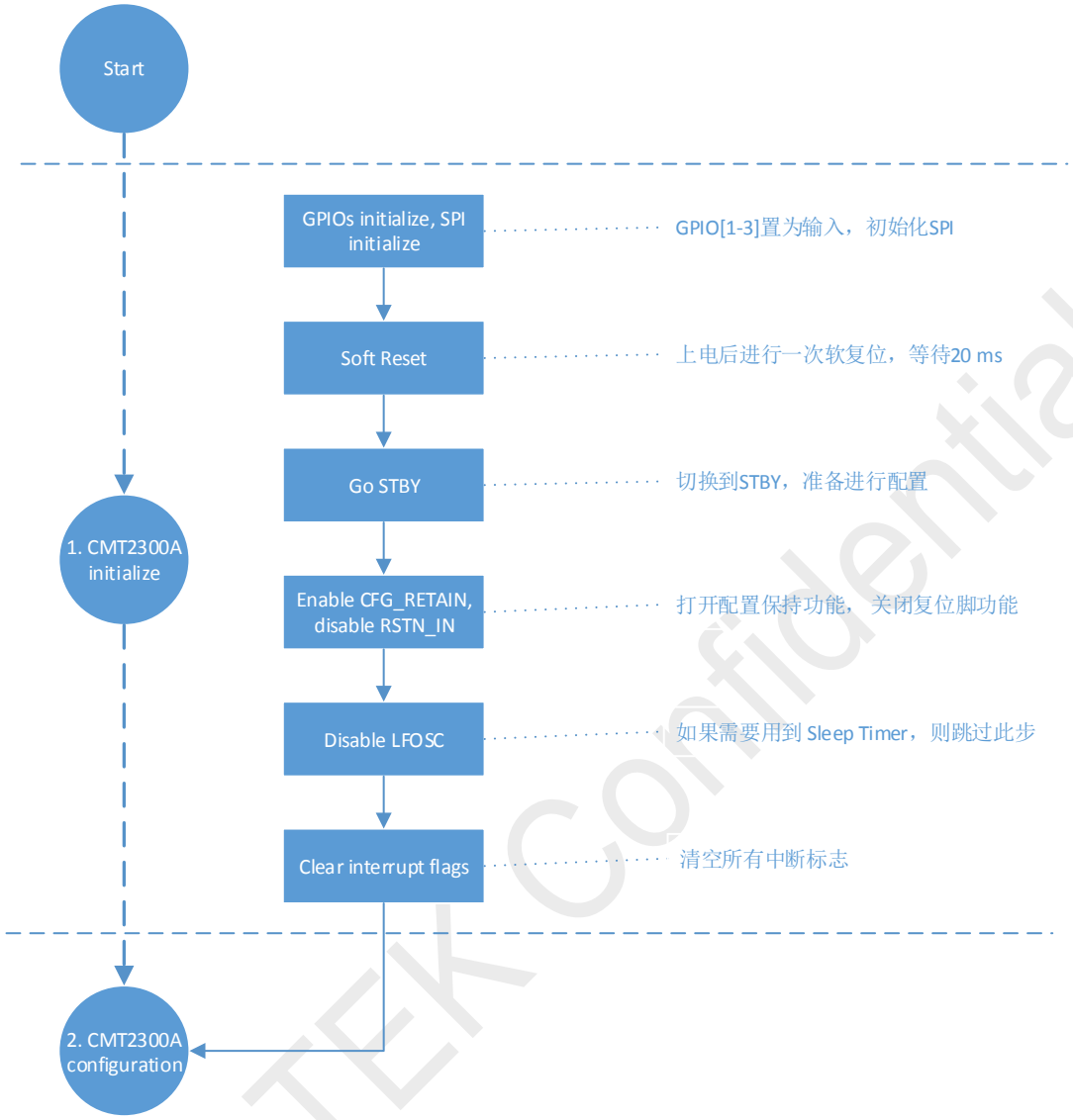

Figure 11. CMT2300AW Initialization

4.2.2 CMT2300AW Configuration

After the chip is initialized, it needs to call the RF_Config function. In the STBY state, configure the register, interrupt and GPIO. Finally enter the SLEEP state to allow the configuration to take effect.

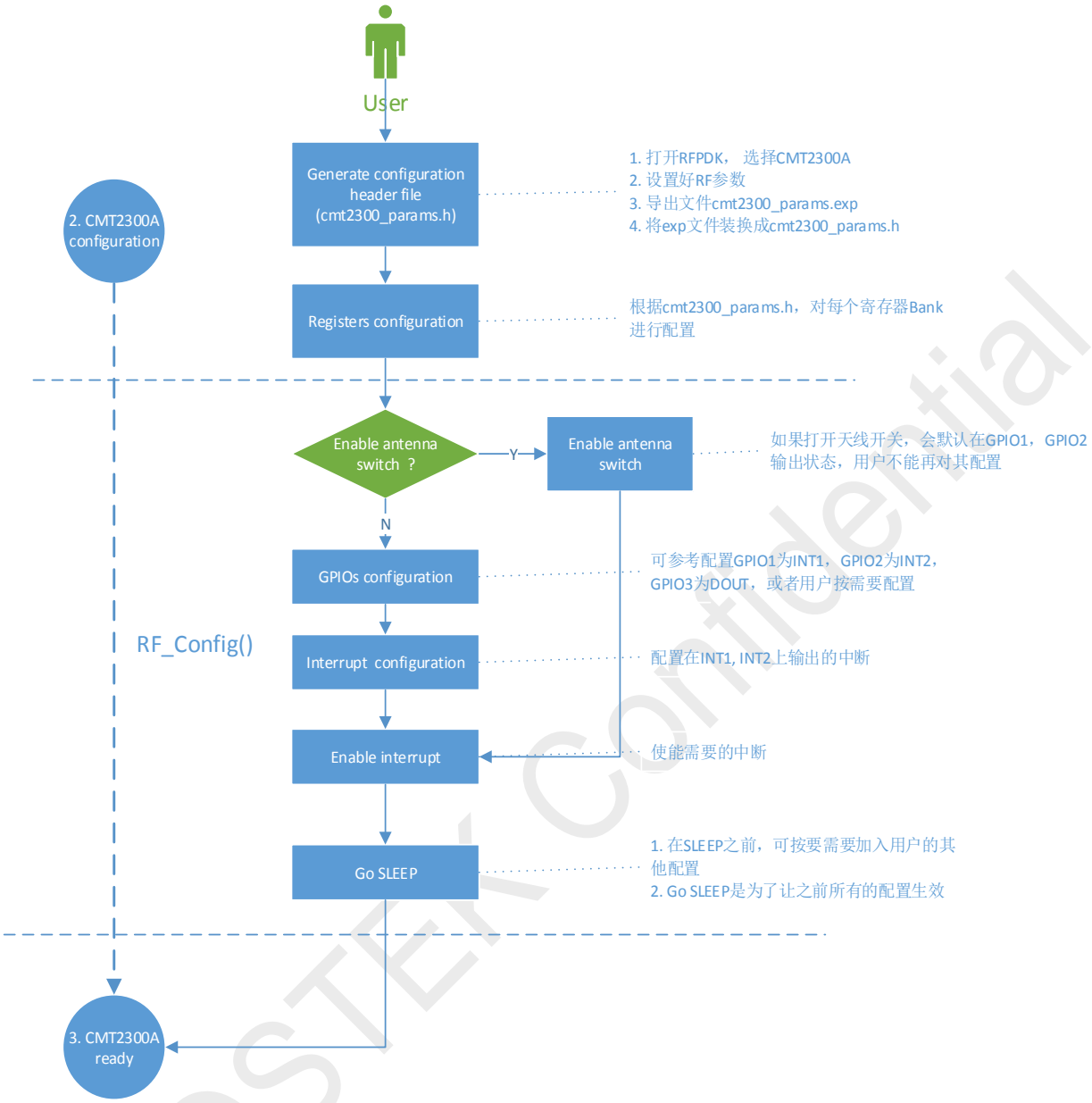

Figure 12. CMT2300AW Configuration

4.2.3 CMT2300AW State Processing

After the chip is configured, it can call the RF_StartTx() or RF_StartRx() function and enter the TX or RX state, then cyclically call RF_Process () function to communicate. In RF_Process() function, a state machine is used to control the chip, and the result is returned to the application layer.

RF_Process () internal state:

1> RF_STATE_IDLE: Idle state

2> RF_STATE_RX_START: RX is in starting state, read FIFO and start receiving.

3> RF_STATE_RX_WAIT: RX is in waiting state, continuously detect the interrupt of completing the reception.

4> RF_STATE_RX_DONE: RX is in completion state, read FIFO, check and clear the interrupt.

5> RF_STATE_RX_TIMEOUT: RX is in timeout state, allow the chip to exit the reception.

6> RF_STATE_TX_START: TX is in starting state, fill the FIFO data and start transmitting.

7> RF_STATE_TX_WAIT: TX is in waiting state, continuously detect the interrupt of completing the transmission.

8> RF_STATE_TX_DONE: TX is in completion state, check and clear the interrupt.

9> RF_STATE_TX_TIMEOUT: TX is in timeout state, allow the chip to exit the transmission.

10> RF_STATE_ERROR: Error state. The chip cannot enter the TX or RX state to soft reset the chip and reconfigure it.

RF_Process () returns the result:

1> RF_IDLE: Chip is idle. Allow it to enter the RX or TX state.

2> RF_BUSY: Chip is busy. It is currently transmitting or receiving the data.

3> RF_RX_DONE: RX is completed. The upper layer can process the received data.

4> RF_RX_TIMEOUT: RX is timeout.

5> RF_TX_DONE: TX is completed.

6> RF_TX_TIMEOUT: TX is timeout.

7> RF_ERROR: TX or RX is error.

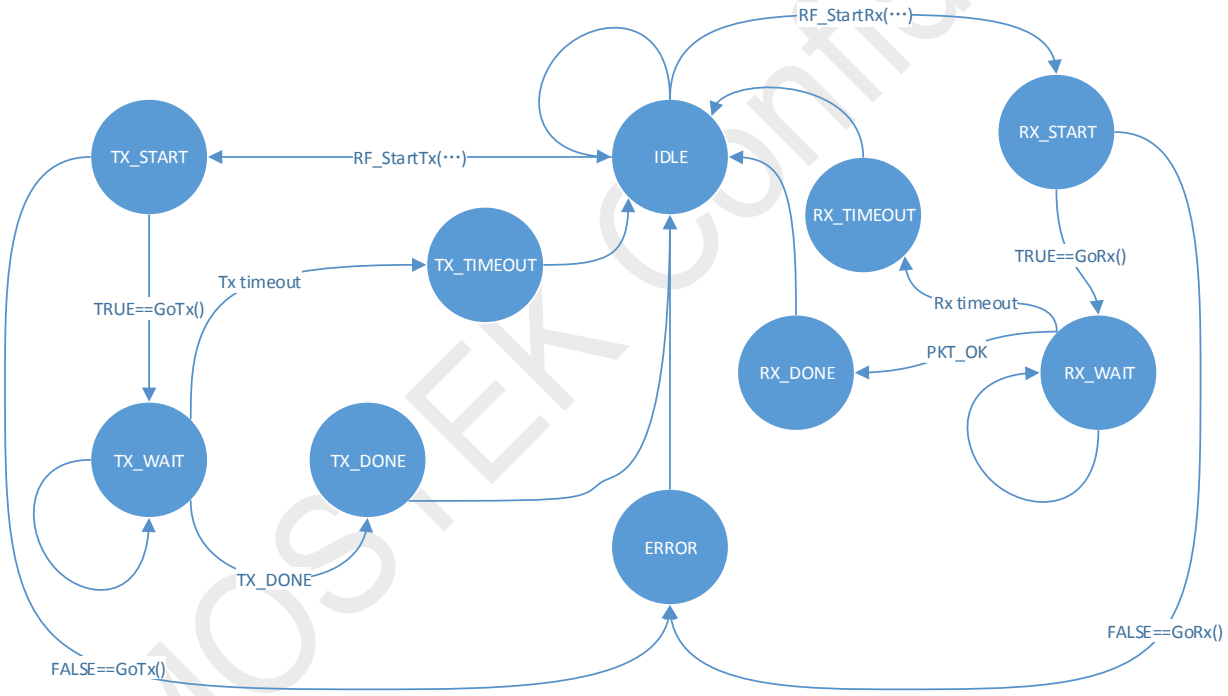

Figure 13. CMT2300 State Processing

4.3 Software Directory Architecture

Demo program is based on RFEB platform. It uses the Keil5 IDE to develop. There is a complete engineering directory

1> Libraries: STM32F103 related library files

2> MDK-ARM: Keil5 related engineering and compiling files

3> USER: Demo source program

4> clear.bat: Clear all compiling intermediate files

5> services: Base on MCU to achieve the time and interrupt services.

6> platform: Platform specific configuration and control files

7> periph: LED, buttons, LCD, SPI and other peripheral resources

8> radio: CMT2300 all API interface files

9> cmt2300_params.h: Export the header file converted by exp file with RFPDK, one by one corresponding to the register Bank in exp file.

10> cmt2300_defs.h: Register address macro definition for CMT2300

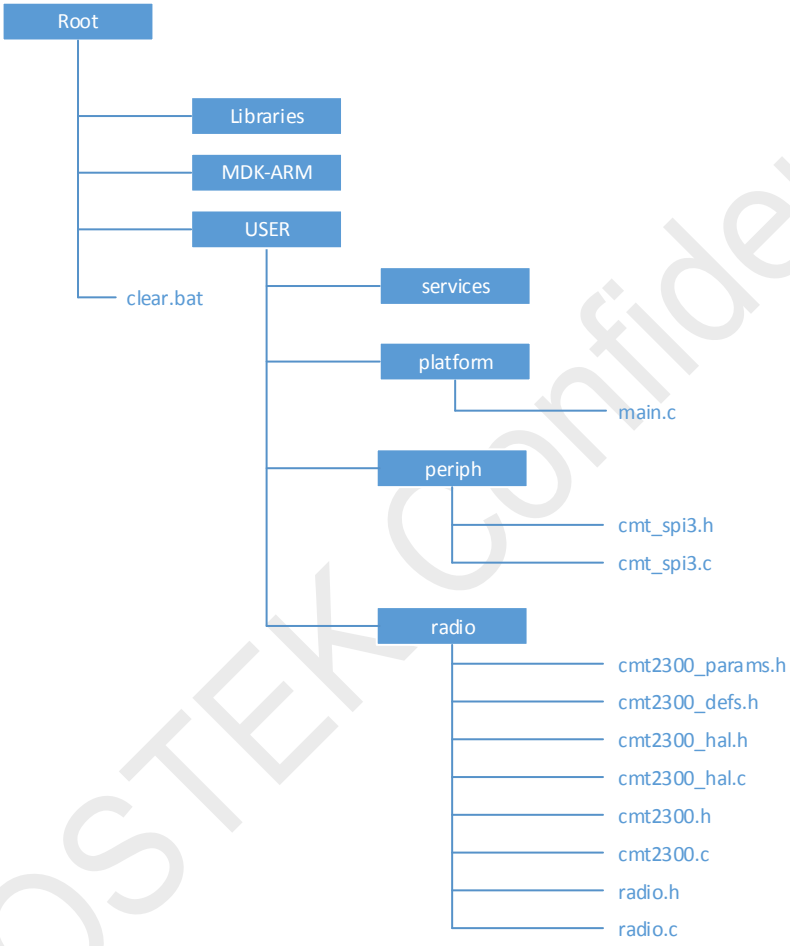

Figure 14. Software Directory Architecture

4.3.1 Application Layer Source Code

main.c is the application layer source code. Configure g_bEnableMaster to select the master or slave. Call OnMaster() or OnSlave() to transmit or receive.

```
#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 Simulating SPI Realization Source Code

cmt_spi3.c provides the SPI communication timing for CMT2300AW. If the user wants to migrate to other MCU platforms, the user needs to modify the following macro definitions.

```
/* ****************************************************************************
*  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 Abstract Hardware Layer Source Code

cmt2300_hal.c is the abstract hardware layer source code. It provides the register, FIFO and GPIO access interface.

```
/*! ********************************************************
* @name    Cmt2300_InitGpio
* @desc    Initializes the CMT2300AW 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 CMT2300AW 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 CMT2300AW 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 CMT2300AW 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);
```

If the user wants to transplant the Demo program to other MCU platform, the user needs to modify some macro definitions of the cmt2300_hal.h.

```
/* ****************************************************************************
*  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 Chip Driver Layer Source Code

cmt2300.c is the chip driver layer source code. It provides the chip state switching operations, interrupts, GPIO, FIFO operations, general register configuring and accessing, etc. It belongs to the firmware and is unrelated to the MCU. Users can not modify it.

```
/*! ********************************************************
* @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);           /* Diable LFOSC */

    Cmt2300_ClearInterruptFlags();
}
```

4.3.5 Chip Processing Layer Source Code

```
/* 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. Appendix

5.1 Appendix 1: SPI Read/Write Sample Code

```
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 Appendix 2: SPI Read/Write FIFO Sample Code

Sample code SPI Write 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 Read FIFO

```
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 Appendix 3: State Switching Sample Code

```
/*! ********************************************************
* @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 Appendix 4: Initialization Sample Code

```
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. Document Change List

Table 18. Document Change List

Rev. No.	Chapter	Description of Changes	Date
0.8	All	Initial release	2017-05-05

7. Contact Information

CMOSTEK Microelectronics Co., Ltd.
Room 203, Honghai Building, Qianhai Road. Nanshan District
Shenzhen, Guangdong, China PRC
Zip Code: 518000
Tel: 0755 - 83235017
Fax: 0755 - 82761326
Sales: sales@cmostek.com
Technical support: support@cmostek.com
Website: www.cmostek.com