EZRADIOPRO™ 编程指南

Coolbor Xie 翻译自 AN415

1、概述

本文档通过几个简单的软件示例，介绍了如何配置 EZRadioPRO 射频发射器、接收器和收发器操作。

本指南包括以下操作示例：

- 如何使用 EZRadioPRO 发射器或收发器在 FIFO 模式下进行数据包发射；
- 如何使用 EZRadioPRO 接收器或收发器在 FIFO 模式下进行数据包接收；
- 如何使用 EZRadioPRO 收发器进行双向基本通信；
- 如何使用 FIFO 发射和接收超过 64 字节的数据包。

最后一个示例程序源代码可以 Silicon Labs 网站下载：https://www.silabs.com/products/wireless/EZRadioPRO/，或在与评估板工具包一起送出的 WDS 光盘上找到。

2、硬件选项

提供两个 EZRadioPRO 收发器芯片的源代码：Si4431-A0 版和 Si4432-V2 版。

这两个器件之间有几点区别：

- Si4432-V2 要求对一些寄存器进行编程为不同于其默认的值。而 Si4431-A0 则无此需要。
- Si4431-A0 有一个单独的寄存器用于自动频率调整（Auto-Frequency Calibration）限制；而 Si4432-V2 则使用频率偏差（Frequency Deviation）寄存器。
- Si4431 和 Si4432 的调制解调器的参数不同。
- 无效引导码超时的定义不同。

为 Si4431 和 Si4432 所提供的源代码例程分别使用不同的高亮。Si4431 和 Si4432 器件以后的版本将共享 Si4431-A0 相同的寄存器设置。

注意：由于本文档只介绍了 Si4431-A0 和 Si4432-V2 器件，但发射或接收软件支持 EZRadioPRO 单独的发射器和接收器：

- Si4431-A0 发射例程代码无需任何修改即可支持 Si4031-A0。代码也可应用于 Si4430-A0 或 Si4030-A0，但中心频率必须设置为相应的值。
- Si4432-V2 发射示例代码无需任何修改即可支持 Si4032-V2。
- Si4431 接收示例代码无需任何修改即可支持 Si4330-A0。

平台上每个示例都提供了一个单独的 Silicon Labs IDE 工作区。Silicon Labs IDE 文件名代表了所给代码是为以下哪个平台所写的：

- 含有“SDBC_DK3”的工作区文件名是为软件开发板（Software Development board）写的
- 含有“EZLINK”的工作区文件名是为 EZLink 平台写的。

2.1 天线选项

在 EZRadioPRO 器件内部，功放和 LNA 并没有连接。在使用 Si4431 收发器的时候，TX 和 RX 引脚可以直接连接在一起，而无需 RF 开关。在使用 Si4432 最高输出功率设置时，发射和接收引脚通过一个 SPDT（单刀双掷）RF 开关分别连接到天线。EZRadioPRO 器件能进行 RF 开关控制。通过将 RX State 和 TX State 信号送到任意两个 GPIO，即可自动控制 RF 开关。根据工作的模式，GPIO 控制 RF 开关自动将天线连接到接收通道或发射通道。如果芯片未处于激活模式，GPIO 将禁止 RF 开关。

在单天线测试卡和 EZLink SIL 模块上，使用了相同的 RF 开关配置：TX State 信号送到 GPIO1，RX State 信号送到 GPIO2。

EZRadioPRO 一个关键的优点在于内置天线分集支持：连接两个不同极化的天线。在数据包接收的开头，芯片对两个天线的接收信号强度进行评估，并使用较强的一个来接收数据包的剩余部分。通过选择信号最强的天线，可以在多路径衰减和天线极化的影响存在的情况下大大提高接收器的性能。

在使用这个特性的时候，需要有一个 RF 开关将天线连接到接收或发射路径。EZRadioPRO 器件通过任意两个 GPIO 来自动控制 RF 开关。将用于天线分集信号的天线 1 开关和用于天线分集信号的天线 2 开

关送到 GPIO，射频就会在接收模式下自动在两个天线之间切换。用于 RX 数据包接收的天线随后将用于 TX 数据包的发射。在天线分集测试卡（Antenna Diversity Testcard）上，GPIO1 连接到用于天线分集信号的天线 1 开关，而 GPIO2 连接到用于天线分集信号的天线 2 开关，来控制 RF 开关的。

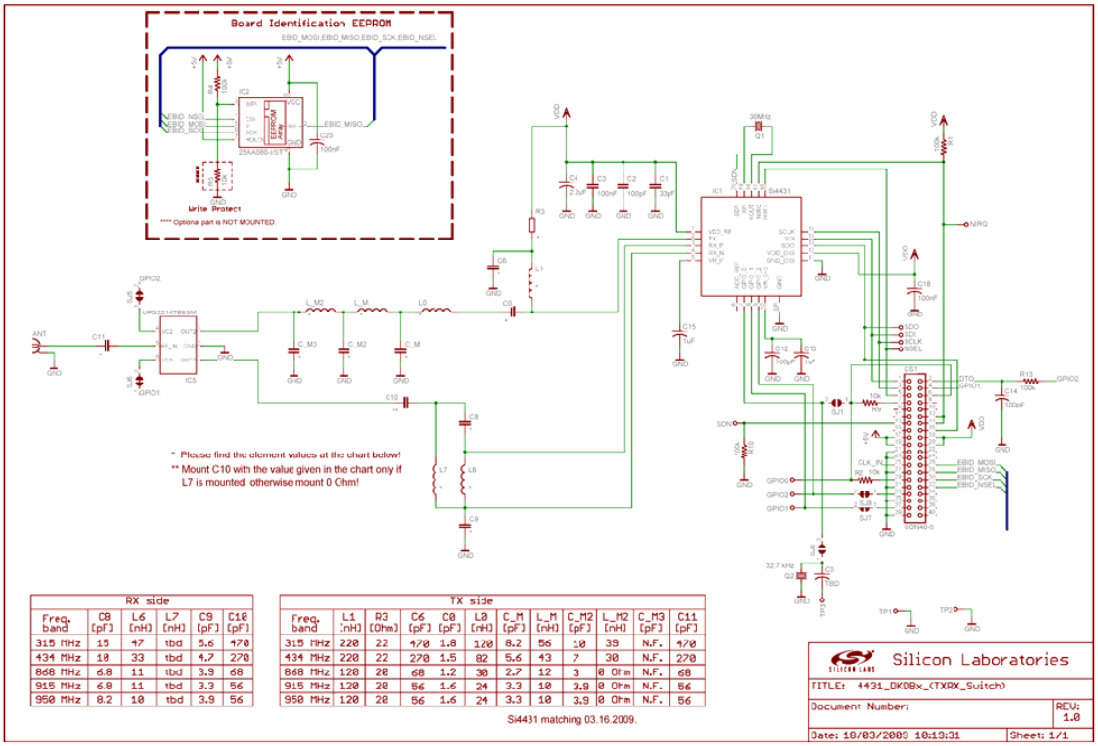

图 1. 单天线 Si4432 测试卡原理图

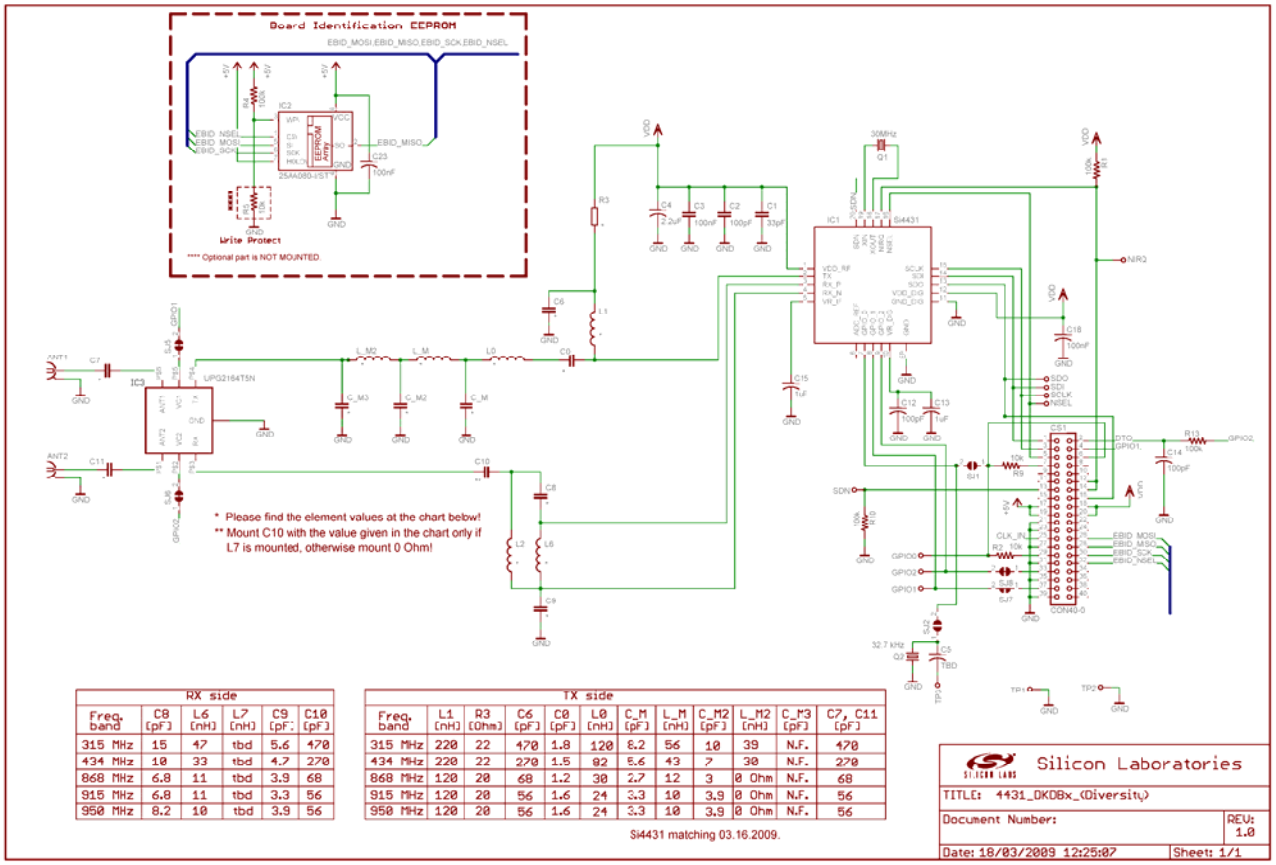

图 2. 天线分集 Si4432 测试卡原理图

用于软件开发（Software Development）板的示例源代码可以编译后用于各种测试卡。以下编译选项，位于 *main_sdbc_dk3.c* 文件的开头，用于选择不同测试卡的正确配置。

- TX/RX Split *#define SEPARATE_RX_TX*
- Single Antenna *#define ONE_SMA_WITH_RF_SWITCH*
- Antenna Diversity *#define ANTENNA_DIVERSITY*

在 EZLink 平台的源代码中没有编译选项，因为这是一个固定的硬件配置。从射频功能的角度来看，相当于一个单天线测试卡。

注意： EZRadioPRO 器件的发射输出必须在输出功率被允许之前终止，通过使用适当的天线或将功率放大器连接到提供 50Ω 终端的 RF 仪器，可以确保正确的操作并保护器件不被损坏。

2.2. 初始化 MCU

软件示例使用了最少的控制器上可用的硬件外围设备来简化源代码的复杂性。提供以下硬件支持：

- 使用按键来启动数据包发射。
- 通过一个 LED 来指示数据包发射与接收。
- EZRadioPRO 器件通过 SPI 连接到 MCU，同时 nIRQ 引脚连接到 MCU 的外部中断。

EZRadioPRO 器件与 MCU 之间的 SPI 和 nIRQ 连接在两个硬件平台上是相同的，但在使用外围设备方面有一些区别：

- 在两个平台上，示例代码只使用按键和 LED；但是，软件开发（Software Development）板有 4 个 LED 和 4 个按键，而 EZLink 分别只有一个。在两个平台上 LED 和按键的 IO 引脚是不同的。
- 在 EZLink 平台上，PWRDN 引脚被连接到 MCU，所以 MCU 可以用它来控制器件。而该连接在软件开发（Software Development）板上是不可用的，PWRDN 引脚被连接到 GND。

2.2.1. 软件开发板（Software Development Board）的初始化

在源代码中包含了与 MCU 有关的头文件，以提供一些宏定义以便代码可以使用几个常用的 C 编译器（Keil、 SDCC、IAR 等）来编译。

```
/* ============================================================ *
* INCLUDE *
* ============================================================ */
#include "C8051F930_defs.h"
#include "compiler_defs.h"
/* ============================================================================ *
* C8051F930 Pin definitions for Software Development Board *
* (using compiler_def.h macros) *
* ============================================================================ */
```

以下宏为给定的 IO 端口指定了一个标签，这样在源代码中可以很容易地引用这些端口。例如：第一行为 Port1.3 指定“NSS”标签，该 nSEL 引脚被连接到该端口：

```
SBIT(NSS, SFR_P1, 3);
SBIT(NIRQ, SFR_P0, 6);
SBIT(PB1, SFR_P0, 0);
SBIT(PB2, SFR_P0, 1);
SBIT(PB3, SFR_P2, 0);
SBIT(PB4, SFR_P2, 1);
SBIT(LED1, SFR_P1, 4);
SBIT(LED2, SFR_P1, 5);
SBIT(LED3, SFR_P1, 6);
SBIT(LED4, SFR_P1, 7);
```

以下定义用于配置插在软件开发（Software Development）板上的测试卡的类型：

```
//One out of these definitions has to be uncommented which tells to the compiler what kind
```

```
//of Testcard is plugged into the Software Development board
#define SEPARATE_RX_TX
//#define ANTENNA_DIVERSITY
//#define ONE_SMA_WITH_RF_SWITCH
```

有几个函数原型用于 MCU 初始化和 SPI 函数。

```
/* ============================================================ *
* Function PROTOTYPES *
* ============================================================ */
//MCU initialization
void MCU_Init(void);
//SPI functions
void SpiWriteRegister (U8, U8);
U8 SpiReadRegister (U8);
```

void MCU_Init(void)函数对软件开发（Software Development）板上的所有必要的外围设备进行初始化：

```
void MCU_Init(void)
{
  //Disable the Watch Dog timer of the MCU
  PCA0MD &= ~0x40;
  // Set the clock source of the MCU: 10MHz, using the internal RC osc.
  CLKSEL = 0x14;
  // Initialize the IO ports and the cross bar
  P0SKIP |= 0xCF; // skip P0.0-3 & 0.6-7
  XBR1 |= 0x40; // Enable SPI1 (3 wire mode)
  P1MDOUT |= 0x01; // Enable SCK push pull
  P1MDOUT |= 0x04; // Enable MOSI push pull
  P1SKIP |= 0x08; // skip NSS
  P1MDOUT |= 0x08; // Enable NSS push pull
  P1SKIP |= 0xF0; // skip LEDs
  P1MDOUT |= 0xF0; // Enable LEDS push pull
  P2SKIP |= 0x03; // skip PB3 & 4
  SFRPAGE = CONFIG_PAGE;
  P1DRV = 0xFD; // MOSI, SCK, NSS, LEDs high current mode
  SFRPAGE = LEGACY_PAGE;
  XBR2 |= 0x40; // enable Crossbar
  // For the SPI communication the hardware peripheral of the MCU is used
  //in 3 wires Single Master Mode. The select pin of the radio is controlled
  //from software
  SPI1CFG = 0x40; //Master SPI, CKPHA=0, CKPOL=0
  SPI1CN = 0x00; //3-wire Single Master, SPI enabled
  SPI1CKR = 0x00;
  SPI1EN = 1; // Enable SPI interrupt
  NSS = 1;
  // Turn off the LEDs
  LED1 = 0;
  LED2 = 0;
  LED3 = 0;
```

```
    LED4 = 0;
}
```

void SpiWriteRegister(U8 reg, U8 value)函数将一个新的值写入器件的寄存器。如果只写一个寄存器，则必须写入一个 16 位的值：一个为寄存器的 8 位地址，后面跟一个 8 位的寄存器值。要写寄存器，地址的最高位必须为 1，以指示为写操作。

```
void SpiWriteRegister (U8 reg, U8 value)
{
   // Send SPI data using double buffered write
   //Select the radio by pulling the nSEL pin to low
   NSS = 0;
   //write the address of the register into the SPI buffer of the MCU
   //(important to set the MSB bit)
   SPI1DAT = (reg|0x80); //write data into the SPI register
   //wait until the MCU finishes sending the byte
   while( SPIF1 == 0);
   SPIF1 = 0;
   //write the new value of the radio register into the SPI buffer of the MCU
   SPI1DAT = value; //write data into the SPI register
   //wait until the MCU finishes sending the byte
   while( SPIF1 == 0); //wait for sending the data
   SPIF1 = 0;
   //Deselect the radio by pulling high the nSEL pin
   NSS = 1;
}
```

The ***U8 SpiReadRegister(U8 reg)*** 函数从芯片读取一个寄存器。在读取单个寄存器时，必须发送一个 16 位的值给芯片：一个为寄存器的 8 位地址，后面跟一个 8 位的寄存器值。在 SPI 处理的第二个字节期间，芯片将提供寄存器的值。注意必须将寄存器地址的 MSB 清零以指示为读周期。

```
U8 SpiReadRegister (U8 reg)
{
   //Select the radio by pulling the nSEL pin to low
   NSS = 0;
   //Write the address of the register into the SPI buffer of the MCU
   //(important to clear the MSB bit)
   SPI1DAT = reg; //write data into the SPI register
   //Wait until the MCU finishes sending the byte
   while( SPIF1 == 0);
   SPIF1 = 0;
   //Write a dummy data byte into the SPI buffer of the MCU. During sending
   //this byte the MCU will read the value of the radio register and save it
   //in its SPI buffer.
   SPI1DAT = 0xFF; //write dummy data into the SPI register
   //Wait until the MCU finishes sending the byte
   while( SPIF1 == 0);
   SPIF1 = 0;
   //Deselect the radio by pulling high the nSEL pin
   NSS = 1;
```

```
    //Read the received radio register value and return with it
    return SPI1DAT;
}
```

2.2.2. EZLink 平台的初始化

EZLink 平台的初始化与软件开发（Software Development）板的初始化非常相似，只有一点区别：EZRadioPRO PWRDN 引脚的控制。将该引脚拉高，器件将完全关闭，从而达到最小功耗，但寄存器的值不被保留。将该引脚拉低将全能器件，并执行一个持续约 15ms 的上电复位周期。器件在掉电模式（power down mode）以及上电复位周期不会接收任何 SPI 命令。这就是主函数中 MCU 初始化及 PWRDN 被拉低以后有一个延时的原因。

```
//Initialize the MCU:
// - set IO ports for the Software Development board
// - set MCU clock source
// - initialize the SPI port
// - turn off LEDs
MCU_Init();
/* ============================================================ *
* Initialize the Si4432 ISM chip *
* ============================================================ */
//Turn on the radio by pulling down the PWRDN pin
SDN = 0;
//Wait at least 15ms before any initialization SPI commands are sent to the radio
// (wait for the power on reset sequence)
for (temp8=0;temp8<15;temp8++)
{
   for(delay=0;delay<10000;delay++);
}
//read interrupt status registers to clear the interrupt flags and release NIRQ pin
ItStatus1 = SpiReadRegister(0x03); //read the Interrupt Status1 register
ItStatus2 = SpiReadRegister(0x04); //read the Interrupt Status2 register
```

3. 使用发射数据包处理器（packet handler）的数据包发射

本软件示例使用硬件平台上的按键来发起一次数据包发射。在上电复位后，固件初始化 MCU 和 EZRadioPRO 芯片。芯片被设置进入空闲模式（IDLE mode），该模式下只有晶体振荡器在工作，然后软件等待按键按下。如果软件开发板（Software Development board）上的任意一个按键按下，演示程序将发送一个数据包，然后返回继续轮询按键。用于发射的数据包配置如下：

表 1 演示代码数据包配置

Preamble					Synchron pattern		Payload length	Payload: “Button1+CR”								CRC	
AA	AA	AA	AA	AA	2D	D4	08	42	75	74	74	6F	6E	31	0D	7A	B1
5 bytes (10 bytes if Ant. Diversity is used)					2 bytes		1 byte	8 bytes								2 bytes	

数据包的有效载荷是根据被按下的按键号来设置的。

注意：由于在 EZLink 平台上只有一个按键，所以它总是发送“Button1”有效载荷。

在数据包中唯一真正的应用信息是代表哪一个按键被按下的字符串，而实际的数据包需要几个附加字节来正确操作。这些附加的数据包字段包括：引导码（preamble）、同步字（sync word）、有效载荷长度（payload

length）和 CRC。

在允许接收器后，将开始接收数据，即使没有信号存在。为了使接收器能侦测到可用数据包存在，发射的数据包应该以引导码和同步字开始。

- 引导码（*preamble*）是一串连续的 0101，用于同步发射器和接收器。引导码是数据中具有连续边沿变化的一串已知的序列，所以接收器节点的解调器和时钟恢复电路能够被正确地处理。引导码最小长度与应用有关。详细内容见“4.使用接收数据包处理器（packet handler）的数据包接收”。
- 在发射器和接收器同步之后，接收器必须找出数据包中的有效载荷数据（payload data）从什么地方开始。发射器包含了一个已知位字段来帮助识别有效载荷数据（payload data）：同步字（*synchron word*）。

接收器还需要知道发送的数据包有多长。有几种可能性可以用于指示数据包的长度：

- 在数据包的结尾发送一个特殊的字符；
- 总是发射固定长度的数据包；
- 在发射的数据包中包含长度信息。

在示例代码中，有效载荷的长度包含在数据包里。

在 RF 物理信道中，发射的数据包会由于噪声或其他 RF 发射器在同一时间发送数据包而破坏，所以强烈建议使用一些错误检查，以便确认接收的数据是否可用或包含错误。

如果通信受到干扰，将引起一系列的位错误，这可以用 CRC 来识别。简单的检验可以用于识别单个位错误，大多数链接中断则会引起突发错误，这就需要 CRC 来做健壮检测。

3.1. 器件的初始化

EZRadioPRO 器件有一个内置的数据包处理器（packet handler）来自动进行数据包发射。在基本的数据包参数被初始化之后，有效载荷数据（payload data）只需要送到 FIFO，数据包发射就会自动开始。整个数据包的建立和发送都由器件完成，而不用微控制器。在数据包发射完成后，器件将为微控制器产生一个中断，并返回以前的激活状态。

本节描述如何配置 EZRadioPRO 发射器或收发器来发送表 1 所示的数据包。*tx_SDBC_DK3* 或 *tx_EZlink* 示例项目用于演示数据包发射。本节使用的代码段拷贝至 *main_sdbc_dk3.c* 或 *main_ezlink* 文件。

器件初始化的代码部分必须在下列情况执行：

- 一个复位事件发生。
- 器件的 power down 引脚被拉低（器件允许）。

注意：EZRadioPRO 器件的上电复位周期持续 15ms。在这个复位期间，器件不能接收任何 SPI 命令。有两种方法可以确定在复位事件后芯片是否准备好接收 SPI 命令。

- 在微控制器中使用一个定时器来等待 15ms。
- 允许微控制器的外部中断，等待来自芯片的一个中断。在 MCU ISR 程序中检查 ipor 中断状态位，该状态位在 POR 事件正确结束后会被置 1。

3.1.1. 器件的软件复位

如果微控制器要处理整个应用的上电复位，那么建议在每次微控制器执行一系列上电复位时为芯片提供一个软件复位。

以下的代码部分给出了如何执行软件复位，并确定复位过程什么时候结束，以允许接收来自微控制器的 SPI 命令：

```
//SW reset
SpiWriteRegister(0x07, 0x80); //write 0x80 to the Operating & Function Control1 register
//wait for chip ready interrupt from the radio (while the nIRQ pin is high)
while ( NIRQ == 1);
//read interrupt status registers to clear the interrupt flags and release NIRQ pin
ItStatus1 = SpiReadRegister(0x03); //read the Interrupt Status1 register
ItStatus2 = SpiReadRegister(0x04); //read the Interrupt Status2 register
```

3.1.2. 设置 RF 参数

以下部分设置需要进行 OOK、FSK 或 GFSK 调制数据包发射的基本 RF 参数。这些参数包括：中断频

率（center frequency）、发射数据速率（transmit data rate）和发射偏差（transmit deviation）。

```
/*set the physical parameters*/
//set the center frequency to 915 MHz
SpiWriteRegister(0x75, 0x75); //write 0x75 to the Frequency Band Select register
SpiWriteRegister(0x76, 0xBB); //write 0xBB to the Nominal Carrier Frequency1 register
SpiWriteRegister(0x77, 0x80); //write 0x80 to the Nominal Carrier Frequency0 register
//set the desired TX data rate (9.6kbps)
SpiWriteRegister(0x6E, 0x4E); //write 0x4E to the TXDataRate 1 register
SpiWriteRegister(0x6F, 0xA5); //write 0xA5 to the TXDataRate 0 register
SpiWriteRegister(0x70, 0x2C); //write 0x2C to the Modulation Mode Control 1 register
//set the desired TX deviation (+-45 kHz)
SpiWriteRegister(0x72, 0x48); //write 0x48 to the Frequency Deviation register
```

注意：对于 OOK 调制，不需要配置发射偏差（transmit deviation）。

除了 TX Data Rate 0 和 TX Data Rate 1 寄存器以外，还有一位用于定义数据速率："txdtrtscale"（Modulation Mode Control1 寄存器的 bit5）。如果希望数据速率低于 30kbps，"txdtrtscale"位必须设置为 1。

偏差（deviation）的最高位（MSB）设置在 Modulation Mode Control register 2 (fd[8] – bit2)。

3.1.3. 设置数据包配置（Packet Configuration）

EZRadioPRO 器件提供了一个非常灵活的处理器来支持宽范围的数据配置，包括以下内容：

- 可编程引导码（最多 512 字节）
- 可编程同步字长度和字段（最多 4 字节）
- 自动桢头（header）发生和 qualification（最多 4 字节）
- 最多 64 字节的有效载荷数据（payload data）（使用内置 FIFO）
- 自动 CRC 计算和检验（支持 3 种不同的 CRC 多项式）

如果用户的应用使用符合以下选项的数据包配置，就可以使用 EZRadioPRO 数据包处理器（packet handler）。使用数据包处理器（packet handler）允许 MCU 在配置期间对发射数据包格式和结构进行配置。在发射数据包时，MCU 只需要在每个数据包发射之前将有效载荷数据（payload data）放到 TX FIFO 中，数据包处理器（packet handler）将自动构造和发射该数据包。

EZRadioPRO 器件另一个强大的功能是自动天线分集（Automatic Antenna Diversity）。天线分集（Antenna diversity）允许接收器使用两个独立的天线，以帮助解决多路径衰减或天线极化的影响。接收器将自动评估两个天线的信号强度，并选择功率最大的天线。在接收模式中使用天线分集（Antenna diversity），发射器需要被正确配置以便系统支持分集（diversity）。对天线分集（Antenna diversity）需要考虑以下内容：

- 需要较长的引导码以便接收器评估两个天线的信号强度等级，从而确定使用哪一个天线来接收数据包。
- 内置天线分集算法（antenna diversity algorithm）使用 GPIO 引脚来控制 RF 开关（用于选择正确的天线）。两个 GPIO 必须配置成 RF 开关的控制信号。
- RF 开关控制信号的极性必须配置正确。

下面给出的示例代码演示了如何为表 1 所示的简单数据包配置数据包处理器（packet handler）。

该示例可以被编译用于天线分集（Antenna diversity）或非天线分集（Antenna diversity）操作。这可以通过以下配置选项（见"2. 硬件选项"）来选择：SEPARATE_RX_TX、ANTENNA_DIVERSITY、ONE_SMA_WITH_RF_SWITCH。

引导码（preamble）长度要根据是否允许天线分集算法（antenna diversity algorithm）来设置：

- 在天线分集（Antenna diversity）被禁止时，需要发送 5 个字节的引导码（preamble）。较少的引导码会使得接收器使能自动频率调整（AFC），引导码检测阀值（preamble detection threshold）为 2 个字节。
- 在天线分集（Antenna diversity）被允许时，必须发射一个较长的引导码。较长的引导码可让接收器有充足的时间来评估两个天线的信号强度、正确处理时钟恢复（clock recovery）电路等。

前导码长度，如果不用天线分集器，设置为5

```
/*set the packet structure and the modulation type*/
//set the preamble length to 10bytes if the antenna diversity is used and set to 5bytes if not
#ifdef ANTENNA_DIVERSITY
  SpiWriteRegister(0x34, 0x14); //write 0x14 to the Preamble Length register
#else
  SpiWriteRegister(0x34, 0x09); //write 0x09 to the Preamble Length register
#endif
```

禁止桢头字节（本示例中未用），设置同步字长度为两个字节。

注意：引导码（preamble）长度设置的最高位（MSB）位于这个寄存器中（“prealen8”— Header Control2 寄存器的 bit0）。帧头控制

```
//Disable header bytes; set variable packet length (the length of the payload is defined by the
//received packet length field of the packet); set the synch word to two bytes long
SpiWriteRegister(0x33, 0x02); //write 0x02 to the Header Control2 register
```

设置同步字样式（pattern）为 0x2DD4。建议使用最少两个字节的同步字以增加通信的健壮性。同步字样式（pattern）是出于系统级设计考虑的。出于 EZRadio 器件通用性的考虑，同步字必须设置为 0x2DD4。通过在不同的应用或节点类型中使用不同的值，同步字也可以用作数据包过滤器。同步字的值也能被接收器用于过滤和只从希望的节点接收通信。同步字样式

```
//Set the sync word pattern to 0x2DD4
SpiWriteRegister(0x36, 0x2D); //write 0x2D to the Sync Word 3 register
SpiWriteRegister(0x37, 0xD4); //write 0xD4 to the Sync Word 2 register
```

使能发射数据包处理器和 CRC6 算法：数据存取控制

```
//enable the TX packet handler and CRC-16 (IBM) check
SpiWriteRegister(0x30, 0x0D); //write 0x0D to the Data Access Control register
```
使能数据包处理，应该是8D

本示例将使用 EZRadioPRO 64 字节的 TX FIFO 来为 TX 数据包提供数据。通过使用 FIFO，MCU 只要简单地提供数据填充到 FIFO，芯片内部就会产生相应的位时序，并调制输出信号。FIFO 可以和 TX 数据包处理器（packet handler）一起使用来构造数据包格式，或整个数据包在 MCU 中计算，然后放入 TX FIFO。

当未使用 TX 数据包处理器（packet handler）时，芯片是不会提供形成的数据包的。这时 MCU 需要构造整个数据包（包括

引导码（preamble）、同步字（sync word）、桢头字段（header fields）、有效载荷（payload）等），并且将形成的数据包填充到发射 FIFO 中。

EZRadioPRO 也支持几种模式来提供作为数据位的包数据给芯片：

使用其中一个 GPIO 来直接调制。在这种情况下，MCU 必须形成数据包并以正确的位时序将数据位提供给所选的 GPIO。

使用 SDI 引脚来直接调制。在这种情况下，MCU 必须形成数据包，并以正确的位时序将数据位提供给所选的 SDI 引脚。

使用内部 PN9 随机数据发生器。这是一个测试模式，可以用于测量调制输出频谱的形状。

在当前示例中，FIFO 是调制源，TX 数据包处理器（packet handler）用于形成和发射数据包，相应地 Modulation Mode Control2 寄存器设置为：调制模式控制 2

```
//enable FIFO mode and GFSK modulation
SpiWriteRegister(0x71, 0x63);//write 0x63 to the Modulation Mode Control 2 register
```

如果使用了天线分集（Antenna diversity），必须正确配置 GPIO。对于 Si4432-DKDBx 测试卡，RF 开关使用以下配置：

```
#ifdef ANTENNA_DIVERSITY
  SpiWriteRegister(0x0C, 0x17); //write 0x17 to the GPIO1 Configuration(set the Antenna 1 Switch used for
  antenna diversity )
```

```
    SpiWriteRegister(0x0D, 0x18); //write 0x18 to the GPIO2 Configuration(set the Antenna 2 Switch used for
    antenna diversity )
#endif
#ifdef ANTENNA_DIVERSITY
    //enable the antenna diversity mode
    SpiWriteRegister(0x08, 0x80); //write 0x80 to the Operating Function Control 2 register
#endif
```

对于单天线测试卡（Single Antenna Testcard），对 GPIO1 和 GPIO2 采用以下配置来控制 RF 开关：

```
#ifdef ONE_SMA_WITH_RF_SWITCH
    SpiWriteRegister(0x0C, 0x12);//write 0x12 to the GPIO1 Configuration(set the TX state)
    SpiWriteRegister(0x0D, 0x15);//write 0x15 to the GPIO2 Configuration(set the RX state)
#endif
```

3.1.4. 选择调制

EZRadioPRO 支持 3 种调制类型：

- 频移键控（FSK）
- 高斯频移键控（GFSK）
- 开关键控（OOK）

本示例使用 GFSK 调制，但以下部分对各种不同的调制类型作了一个概述。

注意：EZRadioPRO 器件可以被设置来提供一个未调制的载波信号以便测试，这可以设置“modtyp[1:0]”位为 0（Modulation mode Control2 寄存器中的 bit[1:0]）。

3.1.4.1. 频移键控（Frequency Shift Keying）

FSK 调制使用信号频率的变化来发射数字数据。

未调制时，在中心频率点发射一个连续波信号（称做 CW 信号）。

要发送一个 0 位，CW 信号的频率将降低与频率偏差（deviation）相等的值，最后频率为 $f_0 - \Delta f_{FSK}$，

其中 f_0 为中心频率，Δf_{FSK} 为频率偏差（deviation）。

要发送一个 1 位，CW 信号的频率将提高与频率偏差（deviation）相等的值，最后频率为 $f_0 + \Delta f_{FSK}$。

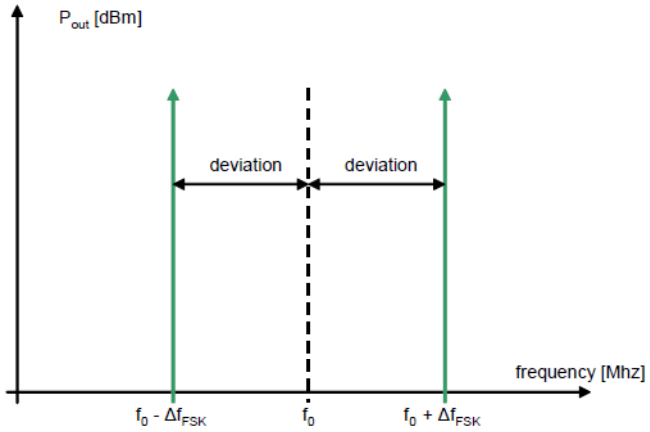

图 3. FSK 调制

要使能 FSK，Modulation Mode Control2 寄存器的“modtyp[1:0]”位应设置为 0x2。

FSK 调制可以有效抵抗干扰；但，性能取决于频率参考的精度，也就是取决于所使用的晶体：建议使用 10~20ppm 精度的晶体。

可以使用具有损耗误差的晶体，但需要提高 TX 频率偏差（deviation）以及接收器的带宽，以确保信号能量落在接收器的滤波器带宽以内。提高信号带宽将引起系统灵敏度的降低。

3.1.4.2. 高斯频移键控（Gaussian Frequency Shift Keying）

GFSK 调制与 FSK 相似，只不过数据位要使用高斯滤波器滤波。这种滤波降低了 TX 位的尖峰沿，从而降低了频谱散射，以及占用更窄的带宽。

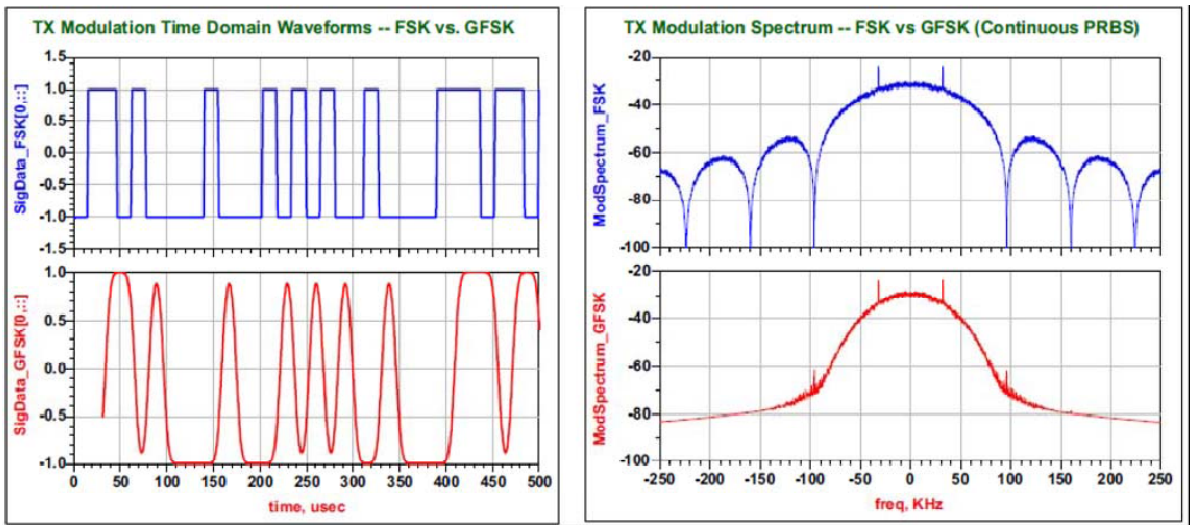

图 4. FSK 和 GFSK 调制的区别（时域和频谱）

GFSK 调制通过设置 Modulation Mode Control2 寄存器的“modtyp[1:0]”为 0x3 来使能的。

GFSK 调制通过提供最窄占用带宽方面的最好性能来提供了更强大的链接性能。象 FSK 一样，性能与链接参数取决与晶体或 TCXO。

3.1.4.3. 开关键控（On-Off Keying）

OOK 调制通过开通和关闭功率放大器来对数据进行编码。

- 在没有数据时，功率放大器关闭。
- 要发送 0，功率放大器在位期间关闭。
- 要发送 1，功率放大器在位期间开通。

OOK 调制通过设置 Modulation Mode Control2 寄存器的“modtyp[1:0]”为 0x1 来使能的。

相比于 FSK 和 GFSK 调制，OOK 调制消耗的电话更低，因为功率放大器在发射 0 时被关闭。OOK 调制提供的链接键壮性比 FSK 或 GFSK 调制要弱，并且所需的频率带宽更宽。因此，OOK 调制通常只有在需要与现有产品兼容时才使用。

3.1.5. Si4432 Revision V2 的寄存器配置

Si4432 revision V2 需要对一些寄存器进行编程，而不使用这些寄存器的默认值：

PLL 和 VCO 必须被编程为以下设置：

```
/*set the non-default Si4432 registers*/
//set VCO and PLL
SpiWriteRegister(0x5A, 0x7F); //write 0x7F to the VCO Current Trimming register
SpiWriteRegister(0x59, 0x40); //write 0x40 to the Divider Current Trimming register
```

注意：这些设置在以后的版本中是不需要的。

3.1.6. Si4431 Revision A0 的寄存器配置

The Si4431 revision A0 需要对一些寄存器进行编程，而不使用这些寄存器的默认值。PLL 和 VCO 必须被编程为以下设置，以优化电流消耗：

```
//set VCO and PLL
SpiWriteRegister(0x57, 0x01);//write 0x01 to the Chargepump Test register
SpiWriteRegister(0x59, 0x00);//write 0x40 to the Divider Current Trimming register
SpiWriteRegister(0x5A, 0x01); //write 0x01 to the VCO Current Trimming register
```

注意：这些设置在以后的版本中是不需要的。

3.1.7. 晶体振荡器调谐电容（Crystal Oscillator Tuning Capacitor）

中心频率的精度取决于所用晶体的几个参数（如：负载电容、晶体精度等），以及与晶体电路有关的 PCB 的寄生电容。

EZRadioPRO 提供了几个特性来减少这些晶体参数的影响：

- 通过使用较宽的发射偏差和较宽的接收带宽，链接将对任何频率偏移的灵敏度降低，但链接预算将降低，相对比较理想。（By using a wide transmit deviation and wide receiver bandwidth, the link will be less sensitive to any frequency offset, but the link budget will be reduced compared to an ideal case.）
- 通过使用内置的自动频率调整（AFC），调整中心频率以配合发射器的中心频率。这种方法有需要较长引导码的限制。
- 通过使用晶体振荡器负载电容（Crystal Oscillator Load Capacitance）寄存器来补偿晶体的不准确度。通过调节到适当的电容值，由晶体不匹配引起的频率不准确度可以得到补偿。假设在产品的整个生命周期使用相同的 PCB 和同类型的晶体，晶体振荡器负载电容值可以测量一次并编程。

在 Silicon Labs 演示板上就使用了同类型的晶体，并测量正确的晶体负载电容：

```
//set Crystal Oscillator Load Capacitance register
SpiWriteRegister(0x09, 0xD7); //write 0xD7 to the CrystalOscillatorLoadCapacitance register
```

注意：可以用两个简单的方法来确定正确的晶体负载电容值：

使用频谱分析仪测量中心频率，并调整晶体振荡器负载电容（Crystal Oscillator Load Capacitance）寄存器，直到中心频率达到希望的频率。

选择微控制器时钟输出到 GPIO2，并且频率计测量时钟。调整晶体振荡器负载电容（Crystal Oscillator Load Capacitance）寄存器直到所测量的时钟达到希望的频率。

3.2. 发送数据包（Packet）

在 MCU 和芯片为数据包发射初始化后，软件进入一个持续的循环并查看按键是否被按下。只要有按键按下，软件就将相应的有效载荷（payload）填充到 TX FIFO，启动数据包发射并等待“enpksent”中断。在数据包发射期间，会点亮一个 LED。

下面的代码部分给出了如何捕捉按键按下，以及如何发送数据包。

源代码的主循环如下：

```
/*MAIN Loop*/
while(1)
{
  //Poll the port pins of the MCU to figure out whether the push button is pressed or not
  if(PB1 == 0)
  {
    //Wait for releasing the push button
    while(PB1 == 0);
    //turn on the LED to show the packet transmission
    LED1 = 1;
    /*SET THE CONTENT OF THE PACKET*/
    //set the length of the payload to 8bytes
    SpiWriteRegister(0x3E, 8); //write 8 to the Transmit Packet Length register
    //fill the payload into the transmit FIFO
    SpiWriteRegister(0x7F, 0x42); //write 0x42 ('B') to the FIFO Access register
    SpiWriteRegister(0x7F, 0x55); //write 0x55 ('U') to the FIFO Access register
    SpiWriteRegister(0x7F, 0x54); //write 0x54 ('T') to the FIFO Access register
    SpiWriteRegister(0x7F, 0x54); //write 0x54 ('T') to the FIFO Access register
    SpiWriteRegister(0x7F, 0x4F); //write 0x4F ('O') to the FIFO Access register
    SpiWriteRegister(0x7F, 0x4E); //write 0x4E ('N') to the FIFO Access register
    SpiWriteRegister(0x7F, 0x31); //write 0x31 ('1') to the FIFO Access register
```

```
    SpiWriteRegister(0x7F, 0x0D); //write 0x0D (CR) to the FIFO Access register
    //Disable all other interrupts and enable the packet sent interrupt only.
    //This will be used for indicating the successful packet transmission for the MCU
    SpiWriteRegister(0x05, 0x04); //write 0x04 to the Interrupt Enable 1 register
    SpiWriteRegister(0x06, 0x00); //write 0x03 to the Interrupt Enable 2 register
    //Read interrupt status registers. It clear all pending interrupts and the nIRQ pin goes back to high.
    ItStatus1 = SpiReadRegister(0x03); //read the Interrupt Status1 register
    ItStatus2 = SpiReadRegister(0x04); //read the Interrupt Status2 register
    /*enable transmitter*/
    //The radio forms the packet and send it automatically.
    SpiWriteRegister(0x07, 0x09); //write 0x09 to the Operating Function Control 1 register
    /*wait for the packet sent interrupt*/
    //The MCU just needs to wait for the 'ipksent' interrupt.
    while(NIRQ == 1);
    //read interrupt status registers to release the interrupt flags
    ItStatus1 = SpiReadRegister(0x03); //read the Interrupt Status1 register
    ItStatus2 = SpiReadRegister(0x04); //read the Interrupt Status2 register
    //wait a bit for showing the LED a bit longer
    for(delay = 0; delay < 10000;delay++);
    //turn off the LED
    LED1 = 0;
}
```

4. 使用接收数据包处理器（packet handler）的数据包接收

软件示例作为一个连续的接收节点工作。在 MCU 和 EZRadioPRO 接收器或收发器器件配置后，固件设置芯片进入连续的接收模式，并等待数据包。只要与表 1 中数据包配置相同的数据包达到，相应的 LED 就会闪烁。如果接收到的数据包的有效载荷（payload）为“Button1”，则软件开发板（Software Development board）上的 LED1 就会闪烁（其余类似）。在一次成功的数据包接收之后，软件会重新启动接收器并等待下一个数据包。

软件示例可以用作“3. 使用发射数据包处理器（packet handler）的数据包发射”中软件示例的一个接收器节点。

注意：EZLink 平台只接收含有“Button1” 有效载荷（payload）的数据包。

4.1.芯片初始化

EZRadioPRO 接收器和收发器器件有一个内置数据调制解调器，可以被配置为不同的设置。它支持 OOK、FSK 和 GFSK 调制，支持可变基带滤波器带宽以实现窄带或宽带应用。数字调制解调器对接收到的数据位进行解调，并提供给应用。有几种方法可以用来从芯片获取接收到的数据位。

- 直接模式（**Direct Mode**）：EZRadioPRO 器件可以在其中一个 GPIO 上提供原始数据位。在这种情况下，微控制器必须逐位地处理接收到的数据，就象是从芯片发送的一样。这个过程要求 MCU 消耗时间来打包、解码和 qualifying 数据。直接模式可以异步或同步地提供数据。在同步直接数据模式下，芯片为原始数据提供位时钟以便减轻 MCU 的采样。
- 先入先出模式（**FIFO Mode**）：在 EZRadioPRO 器件中有 64 个可用的接收 FIFO。在成功的引导码和同步字识别后，芯片直接将接收到的数据位填充到 RX FIFO。微控制器可以通过使用标准 SPI 命令（读 FIFO Access 寄存器）来访问 FIFO。
- 还有另外的内置功能可用于更容易的 FIFO 处理。RX FIFO 几乎满（almost full）状态信号可以提醒微控制器什么时候从 FIFO 读取数据。其阀值(threshold)可通过 RF FIFO Almost Full Threshold 寄存器来编程为 1~64 字节。
- 带 RX 数据包处理器（packet handler）的 FIFO 模式（**FIFO Mode with RX Packet Handler**）：如

果应用中使用的数据包格式符合“3.1.3. 设置数据包配置”中列出的选项，芯片就可以设置为自动接收数据包。这是最好的选项，因为在数据包接收期间不会消耗 MCU 的资源。如果数据包被正确接收，芯片还提供了一个中断给 MCU，以便唤醒 MCU 或执行其他的功能直到来自芯片的数据可用。

在当前软件示例中，使用了 RX 数据包处理器（packet handler）来接收数据包。以下部分给出了如何配置芯片为这种模式。

4.1.1. 引导码（Preamble）侦测

用一个引导码（preamble，“1010”位样式）开始每个数据包是强制性的。引导码的作用是使得接收器在数据有效载荷（payload）到达之前正确地锁定接收信号。引导码最小需要的长度取决于接收器稳定时间（settling time）和引导码侦测阀值的总和。

在使用 FSK 或 GFSK 调制时接收器稳定（settling）取决于自动频率调整（AFC）允许与否。最小接收器稳定时间（settling time）为 1 个字节；但如果 AFC 被允许，那么接收器稳定时间（settling time）为 2 个字节。另外，在 OOK 调制的情况下，接收器稳定时间（settling time）为 2 个字节。

EZRadioPRO 接收器有一个内置引导码侦测器（Preamble Detector）。在稳定接收器后，芯片会将接收到的位与引导码位样式进行比较。如果引导码侦测器在接收到的数据中找到了预定义的连续引导码位长度，然后芯片会报告一个有效的引导码接收，并产生一个“ipreavalid”中断给微控制器。

引导码侦测阀值（引导码的长度）是可编程的。需要的引导码长度取决于几个因素，包括：芯片开启时间（the duration of the radio on-time）、天线分集（antenna diversity）和 AFC。如果接收器开启时间（on time）相对于数据包长度要长，那么较短的引导码阀值可能会导致错误的引导码侦测。这个条件可能会发生，因为接收器会在数据包之间接收到长时间的随机噪声，有可能噪声（随机数据）可能会获得一个较短的引导码位样式。在这种情况下，建议采用较长的引导码阀值，如 20 位。如果接收器只在数据包发射之前被同步允许，可以使用较短的引导码侦测阀值，如 8 位。

在允许天线分集（antenna diversity）时，接收器必须在引导码阶段进行额外的测量，这样就必须使用较长的引导码。建议在使用天线分集（antenna diversity）时设置引导码侦测阀值（preamble detection threshold）为 20 位。同样在接收器中使用 AFC 时，也需要在引导码阶段另外进行额外的测量，结果也发较长的引导码长度。

表 2 归纳了不同情况下建议的发射引导码长度和引导码侦测阀值：

表 2 推荐的引导码长度

Mode	Approximate Receiver Settling Time	Recommended Preamble Length with 8-Bit Detection Threshold	Recommended Preamble Length with 20-Bit Detection Threshold
(G)FSK AFC Disabled	1 byte	20 bits	32 bits
(G)FSK AFC Enabled	2 byte	28 bits	40 bits
(G)FSK AFC Disabled + Antenna Diversity Enabled	1 byte	—	64 bits
(G)FSK AFC Enabled + Antenna Diversity Enabled	2 byte	—	8 byte
OOK	2 byte	3 byte	4 byte
OOK + Antenna Diversity Enabled	8 byte	—	8 byte

注：如果自动频率校正（AFC）被允许，Frequency Offset 1 和 Frequency Offset 2 寄存器将保持所测量的发射器和接收器之间的偏差。该偏差值可以被读取和保存，以用于调整接收器和发射器之间的超出任何系统的频率偏差。例如，在发射数据包的长度为临界值的应用中，引导码长度可以通过使用 Offset 寄存器来缩短，而不用 AFC 来调整 TX 和 RX 节点之间的频率变化。

芯片包含一个内置的无效引导码（Invalid Preamble）侦测器，用于在最短的时间内验证信道：只要接收器被允许，无效引导码定时器就会启动。如果引导码在无效引导码超时前没有接收到引导码，芯片就会

产生一个中断给微控制器，以指示没有数据包发射。

超时取决于 EZRadioPRO 芯片的版本：

- 如果使用 Si4431-A0，那阀值设置为

$$Timeout = Invalid\_Preamble\_Threshold[5:2] \bullet 4 \bullet T_{bit}$$

- 如果使用 Si4432-V2，那阀值设置为

$$Timeout = 4 \bullet T_{bit} + 16 \bullet (Preath[4:0] + 1) \bullet T_{bit}$$

该特性对以下情况很有用：

- 执行一个跳频协议以验证在给定的信道是否有数据包发射，或软件必须跳到下一个频率点。
- 对地时间同步协议：在唤醒后，节点可以快速决定是否有人在发射或返回睡眠模式。

4.1.2. 同步字侦测（Synchron Word Detection）

在成功的引导码侦测后，芯片将等待同步字（synchron word）。将接收到的数据位与同步字样式进行比较。只要同步字样式与接收到的数据位匹配，就会用数据包有效载荷数据（payload data）填充 FIFO。EZRadioPRO 器件在同步字接收后可以提供一个“iswdet”中断给 MCU。

同步字的长度和样式可编程为 1~4 个字节。建议最少使用两个字节的同步字以免错误侦测。

4.1.3. 选择调制类型

有几个寄存器必须设置，以配置数字调制解调器的特性。这些寄存器必须配置为要使用的 RF 参数：调制（modulation）、晶体误差（crystal tolerance）、数据速率（data rate）、发射频率偏差（transmit deviation）、和 AFC 状态。不需要知道这些寄存器和 RF 参数的之间的关系，因为使用 Excel 计算器工具就可以轻松地定义基于 RF 设置的调制解调器参数。在 OOK 或 FSK/GFSK 调制类型之间的调制解调器参数是不同的。

4.1.3.1. OOK 调制的调制解调器配置

对 OOK 调制，必须配置以下寄存器：

- IF Filter Bandwidth
- Clock Recovery Oversampling Ratio
- Clock Recovery Offset 2
- Clock Recovery Offset 1
- Clock Recovery Offset 0
- Clock Recovery Timing Loop Gain 1
- Clock Recovery Timing Loop Gain 0

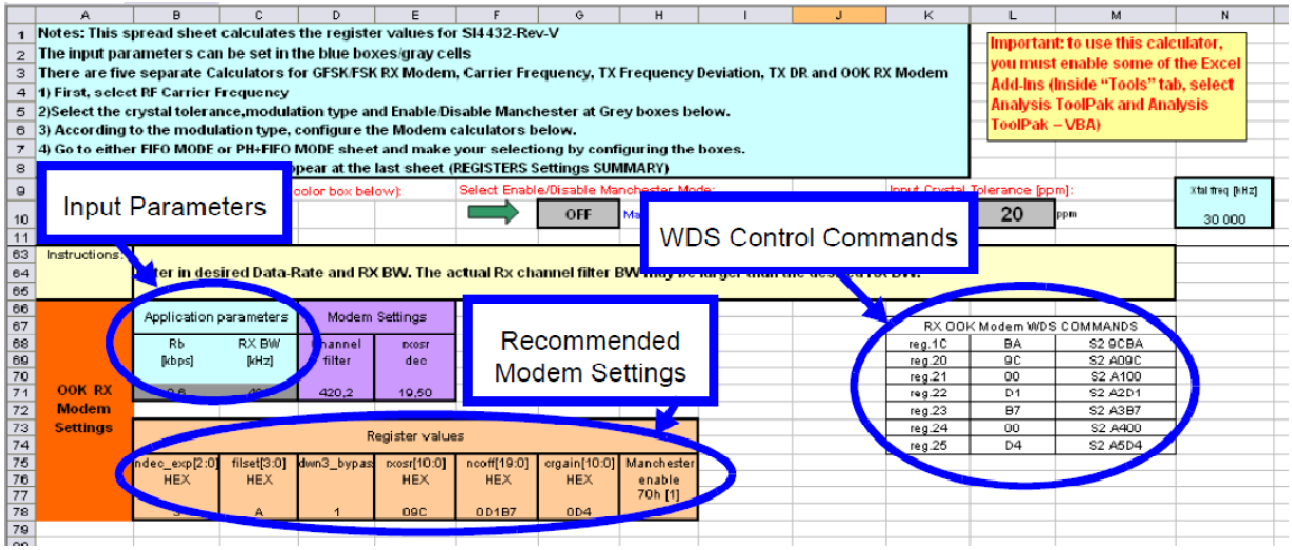

图 5. OOK 调制解调器参数计算

注意：Si4431-A0 和 Si4432-V2 的调制解调器参数是不同的；所以分别提供了 Excel 计算器。

Excel 计算器的输入参数如下：

- 调制类型（Modulation type）为 OOK
- 希望的数据速率（data rate）
- 希望的接收器基带带宽（receiver baseband bandwidth）
- 曼彻斯特编码的使用（Manchester coding）

如图 5 所示，Excel 计算器以十六进制格式提供了推荐的调制解调器参数（在橙色的单元格里显示设置），同时还提供了必要的设置数字调制解调器的 WDS 控制命令（“RX OOK Modem WDS COMMANDS”部分）。

4.1.3.2. FSK/GFSK 调制的调制解调器配置

对于 FSK/GFSK 调制，必须配置以下寄存器：

- IF Filter Bandwidth
- Clock Recovery Oversampling Ratio
- Clock Recovery Offset 2
- Clock Recovery Offset 1
- Clock Recovery Offset 0
- Clock Recovery Timing Loop Gain 1
- Clock Recovery Timing Loop Gain 0
- AFC Loop Gearshift Override

Excel 计算器的输入参数如下：

- 调制类型为（Modulation type）为 FSK 或 GFSK
- 曼彻斯特编码（Manchester coding）允许/禁止
- 晶体误差（Crystal tolerance）
- 希望的数据速率（data rate）
- 发射频率偏差（Transmit deviation）

在输入参数的基础上，Excel 计算器定义了建议的信道滤波器带宽和估计的接收灵敏度。Excel 计算器还以十六进制格式提供了建议的调制解调器参数（在橙色单元格中显示这些设置），以及必要的设置数字调制解调器的 WDS 控制命令（“RX GFSK/FSK Modem WDS COMMANDS”部分）。

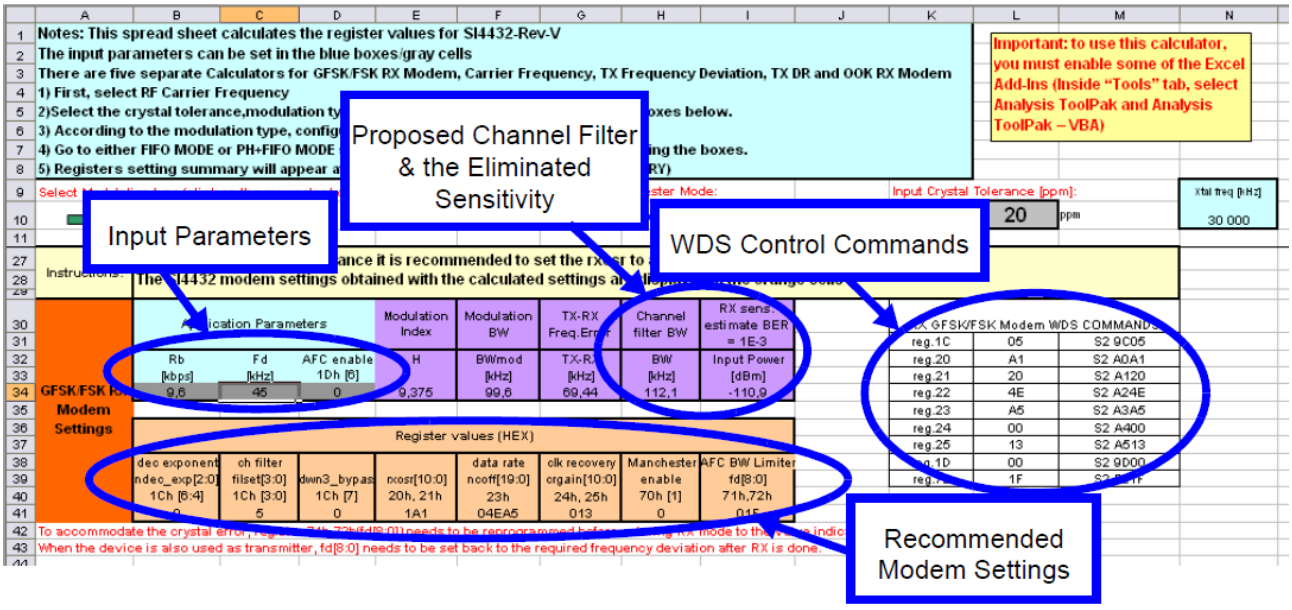

Figure 6. FSK/GFSK 调制解调器参数计算

注意：Si4431-A0 和 Si4432-V2 的调制解调器参数是不同的；所以分别提供了 Excel 计算器。

4.1.4. 示例源代码中的初始化芯片

以下代码部分摘自 rx_SDBC_CK3 或 rx_EZLink 项目的 main.c 文件 。对于 MCU 的初始化和测试卡

选择见“2.硬件选项”。

该示例源代码使用 GFSK 调制、9.6kbps 数据速率，以及±45 kHz 调制，以下代码为 Si4432 芯片的配置。

4.1.4.1. 软件复位（Software Reset）

为芯片提供软件复位。（芯片复位的必要性在“3.1.1. 芯片的软件复位”已经讨论过。）

```
/* ============================================================ *
* Initialize the Si4432 ISM chip *
* ============================================================ */
//SW reset
SpiWriteRegister(0x07, 0x80); //write 0x80 to the Operating & Function Control1 register
//wait for chip ready interrupt from the radio (while the nIRQ pin is high)
while ( NIRQ == 1);
//read interrupt status registers to clear the interrupt flags and release NIRQ pin
ItStatus1 = SpiReadRegister(0x03); //read the Interrupt Status1 register
ItStatus2 = SpiReadRegister(0x04); //read the Interrupt Status2 register
```

4.1.4.2. 设置 RF 参数

以下代码依据希望的 RF 设置来配置芯片工作的物理层参数。中心频率配置为 915MHz。

```
/*set the physical signal parameters*/
//set the center frequency to 915 MHz
SpiWriteRegister(0x75, 0x75); //write 0x73 to the Frequency Band Select register
SpiWriteRegister(0x76, 0xBB); //write 0x67 to the Nominal Carrier Frequency1 register
SpiWriteRegister(0x77, 0x80); //write 0xC0 to the Nominal Carrier Frequency0 register
```

注：如果应用使用多频率来通信，并且这些频率被信道化，EZRadioPRO 器件提供了一个简单的方法来快速修改中心频率。第一个信道的频率由 Frequency Band Select 、Nominal Frequency1 和 Nominal Frequency2 寄存器来定义。信道之间的频率步进值由 Frequency Hopping Step Size 寄存器以 10kHz 的增量来定义。配置这些设置后，实际的信道可以通过 Frequency Hopping Channel Select 寄存器来设置。在这种情况下，只有需要写一个 SPI 寄存器即可修改频率。

为接收期望的 GFSK 调制数据配置调制解调器参数：

```
/*set the modem parameters according to the excel calculator (parameters: 9.6 kbps, deviation: 45 kHz, channel filter BW: 112.1 kHz*/
SpiWriteRegister(0x1C, 0x05); //write 0x05 to the IF Filter Bandwidth register
SpiWriteRegister(0x20, 0xA1); //write 0xA1 to the Clock Recovery Oversampling Ratio register
SpiWriteRegister(0x21, 0x20); //write 0x20 to the Clock Recovery Offset 2 register
SpiWriteRegister(0x22, 0x4E); //write 0x4E to the Clock Recovery Offset 1 register
SpiWriteRegister(0x23, 0xA5); //write 0xA5 to the Clock Recovery Offset 0 register
SpiWriteRegister(0x24, 0x00); //write 0x00 to the Clock Recovery Timing Loop Gain 1 register
SpiWriteRegister(0x25, 0x13);/ /write 0x13 to the Clock Recovery Timing Loop Gain 0 register
SpiWriteRegister(0x1D, 0x40); //write 0x40 to the AFC Loop Gearshift Override register
SpiWriteRegister(0x72, 0x1F); //write 0x1F to the Frequency Deviation register
```

注意：Si4432-V2 芯片频率偏差（Frequency Deviation）寄存器来定义自动频率校正（AFC）的最大允许频率偏移。该寄存器的值必须在进入接收模式前作为 AFC 限制设置；在进入发射模式前根据频率偏差（frequency deviation）修改这个寄存器的值。Si4431- A0 芯片使用一个单独的寄存器：AFC Limiter。

注意相同的 RF 设置（data rate, deviation），Si4431-A0 需要与 Si4432-V2 不同的调制解调器配置。

```
/*set the modem parameters according to the exel calculator(parameters: 9.6 kbps, deviation: 45 kHz, channel filter BW: 102.2 kHz*/
SpiWriteRegister(0x1C, 0x1E); //write 0x1E to the IF Filter Bandwidth register
```

```
SpiWriteRegister(0x20, 0xD0); //write 0xD0 to the Clock Recovery Oversampling Ratio register
SpiWriteRegister(0x21, 0x00); //write 0x00 to the Clock Recovery Offset 2 register
SpiWriteRegister(0x22, 0x9D); //write 0x9D to the Clock Recovery Offset 1 register
SpiWriteRegister(0x23, 0x49); //write 0x49 to the Clock Recovery Offset 0 register
SpiWriteRegister(0x24, 0x00); //write 0x00 to the Clock Recovery Timing Loop Gain 1 register
SpiWriteRegister(0x25, 0x24); //write 0x24 to the Clock Recovery Timing Loop Gain 0 register
SpiWriteRegister(0x1D, 0x40); /write 0x40 to the AFC Loop Gearshift Override register
```

在 Si4431-A0 中有一个专门的寄存器设置 AFC Limit。这在早期的芯片中使用；不需要象 Si4432 那样去修改 Frequency Deviation 寄存器：

```
SpiWriteRegister(0x2A, 0x20); //write 0x20 to the AFC Limiter register
```

如果使用 Si4431，并且应用不是时间同步（例如接收器持续被允许，因为不知道数据包什么时候会达到），那么 Modem Test 寄存器必须被编程为与默认值不同的值。如果不用推荐的寄存器修改，那么接收器将停止接收，并停留在接收模式，直到 MCU 对芯片复位。

```
//set the Modem test register -- IMPORTANT for Si4431 revA0!
SpiWriteRegister(0x56, 0xC1); //write 0xC1 to the Modem test register
```

注意：这个设置在以后的版本中不需要。

4.1.4.3. 设置数据包配置

EZRadioPRO 器件支持各种数据包配置：

- 引导码（preamble）以半字节可编程，最多 512 字节
- 同步字的长度和样式可编程，最多 4 字节
- 最多 4 字节的帧头可用，在接收端使用帧头过滤器
- 可接收固定和可变长度的数据包
- 最多可接收 64 字节的有效载荷
- 芯片自动计算并校验 CRC-16，并且支持三种 16 位 CRC 多项式

如果应用使用符合以上选项的数据包配置，那么接收数据处理器就可配置自动接收数据包。本节的代码部分给出了如何按表 1 所示的数据包来配置数据包处理器（packet handler）。

注：如果数据包有帧头字段（最多 4 字节），那么接收数据包处理器（packet handler）可进行帧头过滤。如果执行网络操作，这是一个很有用的特性，例如，一个节点只要接收来自专门的节点或节点组的数据。

有几个预期的接收帧头（Check Header 3 … 0 寄存器）和帧头屏蔽寄存器（Header Enable 3 … 0）用来单独为每一个帧头字节定义。每个帧头字节也可以与广播地址（0xFF）进行比较。如果任何接收到的帧头字节在过滤检查进失败，则 Device Status 寄存器的“headerr”位被置 1。下图显示了各个寄存器之间的关系以及帧头过滤是如何进行的：

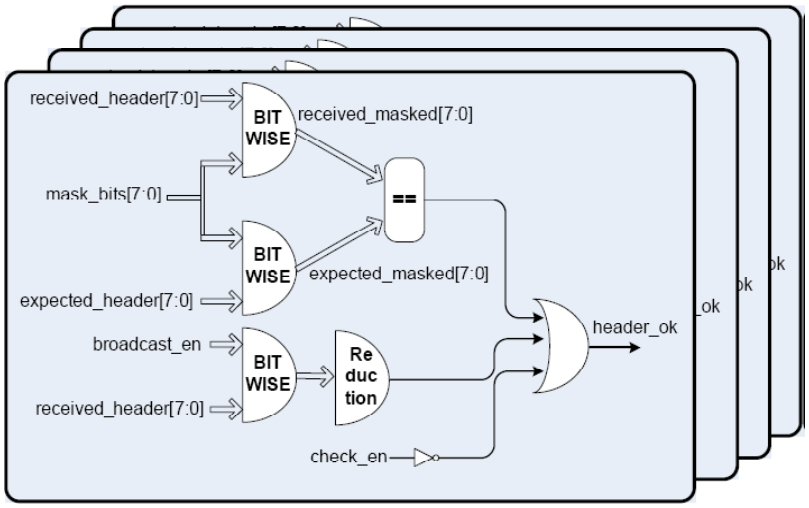

图 7. 接收帧头过滤器（Receive Header Filter）

本示例代码支持天线分集（Antenna Diversity）和非天线分集（non-Antenna diversity）模式。天线分集（Antenna Diversity）选择通过预编译定义来实现：在“2.硬件选项”描述的 SEPARATE_RX_TX、ANTENNA_DIVERSITY、ONE_SMA_WITH_RF_SWITCH。以下代码部分按正确的数据包格式配置接收数据包处理器（packet handler）：

```
/*Configure the receive packet handler*/
//Disable header bytes; set variable packet length (the length of the payload is defined by the
//received packet length field of the packet); set the synch word to two bytes long
SpiWriteRegister(0x33, 0x02 ); //write 0x02 to the Header Control2 register
//Disable the receive header filters
SpiWriteRegister(0x32, 0x00 ); //write 0x00 to the Header Control1 register
//Set the sync word pattern to 0x2DD4
SpiWriteRegister(0x36, 0x2D); //write 0x2D to the Sync Word 3 register
SpiWriteRegister(0x37, 0xD4); //write 0xD4 to the Sync Word 2 register
//Enable the receive packet handler and CRC-16 (IBM) check
SpiWriteRegister(0x30, 0x85); //write 0x0D to the Data Access Control register
//Enable FIFO mode and GFSK modulation
SpiWriteRegister(0x71, 0x63); //write 0x63 to the Modulation Mode Control 2 register
//set preamble detection threshold to 20bits
SpiWriteRegister(0x35, 0x28); //write 0x30 to the Preamble Detection Control register
#ifdef ANTENNA_DIVERSITY
   //Enable antenna diversity mode
   SpiWriteRegister(0x08, 0x80); //write 0x80 to the Operating Function Control 2 register
#endif
/*set the GPIO's according the testcard type*/
#ifdef ANTENNA_DIVERSITY
   SpiWriteRegister(0x0C, 0x17); //write 0x17 to the GPIO1 Configuration(set the Antenna 1 Switch used for antenna
   diversity )
   SpiWriteRegister(0x0D, 0x18); //write 0x18 to the GPIO2 Configuration(set the Antenna 2 Switch used for antenna
   diversity )
#endif
#ifdef ONE_SMA_WITH_RF_SWITCH
   SpiWriteRegister(0x0C, 0x12); //write 0x12 to the GPIO1 Configuration(set the TX state)
   SpiWriteRegister(0x0D, 0x15); //write 0x15 to the GPIO2 Configuration(set the RX state)
#endif
```

注：RX 和 TX 数据包处理器（packet handler）可以设置为发射和接收固定长度的数据包。在这种情况下，有效载荷（payload）的长度不包括在发射的数据包内，而是由 Transmit Packet Length 寄存器来定义。注意这时 Header Control 2 寄存器中的“fixpklen”位必须设置为 1。

4.1.4.4. Si4432-V2 的特殊考虑

必须为 Si4432-V2 器件配置以下寄存器以确保最优的操作。

VCO 寄存器必须使用下面的值来配置：

```
/*set the non-default Si4432 registers*/
//set the VCO and PLL
SpiWriteRegister(0x5A, 0x7F); //write 0x7F to the VCO Current Trimming register
SpiWriteRegister(0x58, 0x80); //write 0xD7 to the ChargepumpCurrentTrimmingOverride register
SpiWriteRegister(0x59, 0x40); //write 0x40 to the Divider Current Trimming register
```

为获得最好的接收器性能，强烈建议按下面的内容来设置 AGC 和 ADC 参数：

```
//set the AGC
SpiWriteRegister(0x6A, 0x0B); //write 0x0B to the AGC Override 2 register
//set ADC reference voltage to 0.9V
SpiWriteRegister(0x68, 0x04); //write 0x04 to the Deltasigma ADC Tuning 2 register
SpiWriteRegister(0x1F, 0x03); //write 0x03 to the Clock Recovery Gearshift Override register
```

注意：这些设置对较早的芯片版本（如 Si4431-A0）是不需要的。

设置晶体负载电容寄存器（Crystal Load Capacitance）：

```
//set Crystal Oscillator Load Capacitance register
SpiWriteRegister(0x09, 0xD7); //write 0xD7 to the Crystal Oscillator Load Capacitance register
```

4.1.4.5. Si4431-A0 的特殊考虑

必须为 Si4431-A0 器件配置以下寄存器，以确保最优的电流消耗：

```
//set VCO and PLL
SpiWriteRegister(0x57, 0x01); //write 0x01 to the Chargepump Test register
SpiWriteRegister(0x59, 0x00); //write 0x40 to the Divider Current Trimming register
SpiWriteRegister(0x5A, 0x01); //write 0x01 to the VCO Current Trimming register
```

注意：这些设置在早期的芯片版本中不需要。

设置晶体负载电容（Crystal Load Capacitance）寄存器（详细内容见“3.1.6 晶体振荡器调谐电容（Crystal Oscillator Tuning Capacitor）”）：

```
//set Crystal Oscillator Load Capacitance register
SpiWriteRegister(0x09, 0xD7); //write 0xD7 to the Crystal Oscillator Load Capacitance register
```

4.2. 接收数据包

在初始化 MCU 和芯片之后，演示程序切换到接收模式并等待来自发射节点的数据包。芯片使用内置的接收数据包处理器（receive packet handler）来接收数据包，只有在接收到的是有效的数据包或数据包没有错误 CRC 时才会产生一个中断给 MCU：

```
/*enable receiver chain*/
SpiWriteRegister(0x07, 0x05);//write 0x05 to the Operating Function Control 1 register
//Enable two interrupts:
// a) one which shows that a valid packet received: 'ipkval'
// b) second shows if the packet received with incorrect CRC: 'icrcerror'
SpiWriteRegister(0x05, 0x03); //write 0x03 to the Interrupt Enable 1 register
SpiWriteRegister(0x06, 0x00); //write 0x00 to the Interrupt Enable 2 register
//read interrupt status registers to release all pending interrupts
ItStatus1 = SpiReadRegister(0x03);//read the Interrupt Status1 register
ItStatus2 = SpiReadRegister(0x04);//read the Interrupt Status2 register
/*MAIN Loop*/
while(1)
{
   //wait for the interrupt event
   if( NIRQ == 0 )
   {
```

在芯片产生中断后，MCU 必须通过读取中断状态（interrupt status）寄存器以确定中断原因。如果接收到的数据包有错误的 CRC，演示程序就会停止接收器，并闪烁所有的 LED（提示用户发生错误），然后重启接收器链：

```
//read interrupt status registers
ItStatus1 = SpiReadRegister(0x03);//read the Interrupt Status1 register
ItStatus2 = SpiReadRegister(0x04);//read the Interrupt Status2 register
```

```
/*CRC Error interrupt occured*/
if( (ItStatus1 & 0x01) == 0x01 )
{
   //disable the receiver chain
   SpiWriteRegister(0x07, 0x01);//write 0x01 to the Operating Function Control 1 register
   //reset the RX FIFO
   SpiWriteRegister(0x08, 0x02);//write 0x02 to the Operating Function Control 2 register
   SpiWriteRegister(0x08, 0x00);//write 0x00 to the Operating Function Control 2 register
   //blink all LEDs to show the error
   LED1 = 1;
   LED2 = 1;
   LED3 = 1;
   LED4 = 1;
   for(delay = 0; delay < 10000;delay++);
   LED1 = 0;
   LED2 = 0;
   LED3 = 0;
   LED4 = 0;
   //enable the receiver chain again
   SpiWriteRegister(0x07, 0x05);//write 0x05 to the Operating Function Control 1 register
}
```

如果接收到的是 CRC 正确的有效数据包，MCU 将读取接收到的有效载荷（payload）的长度，检查 MCU 中分配的缓冲器是否足够贮存数据包，然后读取 RX FIFO 的内容。然后根据数据包的内容闪烁相应的 LED。

如果有效载荷（payload）不适合 MCU 中分配的缓冲器，或有效载荷（payload）与预期的数据包不同，MCU 就会丢弃该数据包并重启接收器链：

```
/*packet received interrupt occurred*/
if( (ItStatus1 & 0x02) == 0x02 )
{
   //disable the receiver chain
   SpiWriteRegister(0x07, 0x01);//write 0x01 to the Operating Function Control 1 register
   //Read the length of the received payload
   length = SpiReadRegister(0x4B);//read the Received Packet Length register
   //check whether the received payload is not longer than the allocated buffer in the MCU
   if(length < 11)
   {
      //Get the received payload from the RX FIFO
      for(temp8=0;temp8 < length;temp8++)
      {
         payload[temp8] = SpiReadRegister(0x7F);//read the FIFO Access register
      }
      //check whether the content of the packet is what the demo expects
      if( length == 8 )
      {
         if( memcmp(&payload[0], "BUTTON1", 7) == 0 )
         {
```

```
                    //blink LED1 to show that the packet received
                    LED1 = 1;
                    for(delay = 0; delay < 10000;delay++);
                    LED1 = 0;
                }
                if( memcmp(&payload[0], "BUTTON2", 7) == 0 )
                {
                    //blink LED2 to show that the packet received
                    LED2 = 1;
                    for(delay = 0; delay < 10000;delay++);
                    LED2 = 0;
                }
                if( memcmp(&payload[0], "BUTTON3", 7) == 0 )
                {
                    //blink LED3 to show that the packet received
                    LED3 = 1;
                    for(delay = 0; delay < 10000;delay++);
                    LED3 = 0;
                }
                if( memcmp(&payload[0], "BUTTON4", 7) == 0 )
                {
                    //blink LED4 to show that the packet received
                    LED4 = 1;
                    for(delay = 0; delay < 10000;delay++);
                    LED4 = 0;
                }
            }
        }
        //reset the RX FIFO
        SpiWriteRegister(0x08, 0x02);//write 0x02 to the Operating Function Control 2 register
        SpiWriteRegister(0x08, 0x00);//write 0x00 to the Operating Function Control 2 register
        //enable the receiver chain again
        SpiWriteRegister(0x07, 0x05);//write 0x05 to the Operating Function Control 1 register
    }
  }
}
```

注意：对于 EZLink 快速原型平台编译的源代码只接收含有“Button1”有效载荷（payload）的数据包。

4.2.1. 使用 Si4431-A0 接收数据包

如果使用 Si4431-A0，如果发生了一个错误的引导码侦测，会有一个能使得芯片停留在同步字侦测状态的错误。当芯片停留在同步字侦测状态时，将拉架数据包错误率。这个问题即可通过增加引导码侦测阀值、降低错误引导码侦测的可能性来纠正，也可以通过应用以下的软件解决办法来纠正：

- 允许 Valid Preamble Detect 和 Synch Word Detect 中断。芯片就会开始等待侦测引导码。
- 如果芯片侦测到引导码，就会产生一个中断给 MCU。在芯片等待同步字时，在 MCU 中运行一个同步字超时定时器。超时时间应该为（引导码-引导码侦测阀值+同步字）×字节间隔时间。如果定时时间到，而没有接收到同步字，那就禁止再允许接收器。

- 在成功地侦测到同步字后，程序就和 Si4432 相同了：数据包有效和 CRC 错误中断应该被允许，MCU 必须等待这些中断。

与 Si4432 不同的代码部分如下所示：

```
/*enable receiver chain*/
SpiWriteRegister(0x07, 0x05);//write 0x05 to the Operating Function Control 1 register
//Enable two interrupts:
// a) one wich shows that a valid preamble received:'ipreaval'
// b) second shows if a sync word received:'iswdet'
SpiWriteRegister(0x05, 0x00); //write 0x00 to the Interrupt Enable 1 register
SpiWriteRegister(0x06, 0xC0); //write 0xC0 to the Interrupt Enable 2 register
//read interrupt status registers to release all pending interrupts
ItStatus1 = SpiReadRegister(0x03);//read the Interrupt Status1 register
ItStatus2 = SpiReadRegister(0x04);//read the Interrupt Status2 register
/*MAIN Loop*/
while(1)
{
   //wait for the interrupt event
   if( NIRQ == 0 )
   {
      //read interrupt status registers
      ItStatus1 = SpiReadRegister(0x03);//read the Interrupt Status1 register
      ItStatus2 = SpiReadRegister(0x04);//read the Interrupt Status2 register
      //check whether preamble is detected
      if( (ItStatus2 & 0x40) == 0x40 )
      {//preamble detected
         //wait for the synch word interrupt with timeout -- THIS is the proposed SW workaround
         //start a timer in the MCU and during timeout check whether synch word interrupt
         happened or not
         delay = 0;
         do {
            delay++;
         } while((delay < 20000) && (NIRQ == 1));
         //check whether the synch word interrupt is detected
         if( NIRQ == 0)
         {//synch word detected correctly
            //read interrupt status registers
            ItStatus1 = SpiReadRegister(0x03); //read the Interrupt Status1 register
            ItStatus2 = SpiReadRegister(0x04); //read the Interrupt Status2 register
            //Enable two interrupts:
            // a) one which shows that a valid packet received: 'ipkval'
            // b) second shows if the packet received with incorrect CRC: 'icrcerror'
            SpiWriteRegister(0x05, 0x03); //write 0x03 to the Interrupt Enable 1 register
            SpiWriteRegister(0x06, 0x00); //write 0x00 to the Interrupt Enable 2 register
            //read interrupt status registers to release all pending interrupts
            ItStatus1 = SpiReadRegister(0x03); //read the Interrupt Status1 register
            ItStatus2 = SpiReadRegister(0x04); //read the Interrupt Status2 register
```

```
//wait for the interrupt event
while(NIRQ == 1);
//read interrupt status registers
ItStatus1 = SpiReadRegister(0x03); //read the Interrupt Status1 register
ItStatus2 = SpiReadRegister(0x04); //read the Interrupt Status2 register
/*CRC Error interrupt occured*/
if( (ItStatus1 & 0x01) == 0x01 )
{
   //disable the receiver chain
   SpiWriteRegister(0x07, 0x01); //write 0x01 to the Operating Function
   Control 1 register
   //blink all LEDs to show the error
   RX_LED = 1;
   TX_LED = 1;
   for(delay = 0; delay < 10000;delay++);
   RX_LED = 0;
   TX_LED = 0;
}
/*packet received interrupt occured*/
if( (ItStatus1 & 0x02) == 0x02 )
{
   //disable the receiver chain
   SpiWriteRegister(0x07, 0x01); //write 0x01 to the Operating Function
   Control 1 register
   //Read the length of the received payload
   length = SpiReadRegister(0x4B); //read the Received Packet Length
   register
   //check whether the received payload is not longer than the allocated
   buffer in the MCU
   if(length < 11)
   {
      //Get the reeived payload from the RX FIFO
      for(temp8=0;temp8 < length;temp8++)
      {
         payload[temp8] = SpiReadRegister(0x7F);
      }
      //check whether the content of the packet is what the demo
      expects
      if( length == 8 )
      {
         if(memcmp(&payload[0], "BUTTON1", 7) == 0 )
         {//blink LED1 to show that the packet received
            RX_LED = 1;
            for(delay = 0; delay < 10000;delay++);
            RX_LED = 0;
         }
```

```
                }
              }
            }
            //Enable two interrupts agin:
            // a) one wich shows that a valid preamble received:'ipreaval'
            // b) second shows if a sync word received:'iswdet'
            SpiWriteRegister(0x05, 0x00); //write 0x00 to the Interrupt Enable 1 register
            SpiWriteRegister(0x06, 0xC0); //write 0xC0 to the Interrupt Enable 2 register
            //read interrupt status registers to release all pending interrupts
            ItStatus1 = SpiReadRegister(0x03); //read the Interrupt Status1 register
            ItStatus2 = SpiReadRegister(0x04); //read the Interrupt Status2 register
            //enable the receiver chain again
            SpiWriteRegister(0x07, 0x05); //write 0x05 to the Operating Function Control 1 register
          }
          else
          {//synch word is not detected within timeout --> reset RX chain to workaround the Si4431-A0 errata
            //disable receiver chain
            SpiWriteRegister(0x07, 0x01); //write 0x01 to the Operating Function Control 1 register
            //read interrupt status
            ItStatus2 = SpiReadRegister(0x04); //read the Interrupt Status2 register //enable RX chain
            SpiWriteRegister(0x07, 0x05); //write 0x05 to the Operating Function Control 1 register
          }
        }
      }
}
}
```

5. 双向通信——发射应答

软件示例在两个节点之间实现一个双向、半工通信协议。必须加载相同的固件到两个器件中。

在初始化 MCU 和 EZRadioPRO 收发器后，软件进入持续接收模式。等待接收数据包或用户按下按键。

如果接收到一个含有“Button1”有效载荷（payload）的有效数据包，软件就会闪烁软件开发板（software development board）上的 LED1，或闪烁 EZLink 快速原型平台上的 RX LED，以指示成功的数据包接收。然后产生并发送一个应答数据给发起者。该（应答）数据包成功地发射后，返回持续接收模式。

如果按键 1 被按下，软件将禁止接收模式，产生一个含有“Button1”有效载荷（payload）的有效数据包，并发射该数据包。在数据包发射期间，软件开发板（software development board）上的 LED2 闪烁，而 EZLink 平台上的 TX LED 会闪烁。在成功地发射数据包后，返回持续接收模式，并等待应答。如果该节点接收到应答数据包，就会闪烁 LED1 或 RX LED，并返回接收模式。

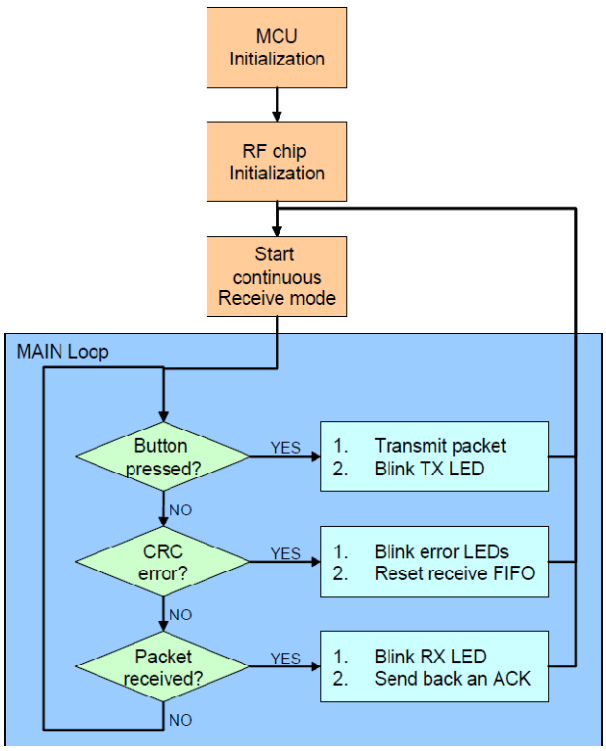

图 8. 流程图

5.1. 软件流程图

图 8 给出演示程序的流程图：

5.2. 软件实现

软件示例使用内置接收和发射数据包处理器（packet handler）和收发器芯片的 FIFO，该程序代码发射和接收具有表 1 所示的数据包格式的数据包。另外，调制方式为 GFSK，数据速率为 9.6kbps，发射频率偏差为±45 kHz。这些与前面的示例中使用的 RF 参数相同。对于双向通信，收发器芯片的发射器和接收器部分都必须配置。发射器和接收器的初始化代码在前面的示例讨论过（“3.使用发射数据包处理器的数据包发射”和“4.使用接收数据包处理器的数据包接收”）。本章只突出那些需要双向通信的描述。

5.2.1. 中断标志

EZRadioPRO 器件能提供几个事件的中断（如：复位发生、晶体稳定、数据包发射、数据包接收等）。当 MCU 接收到一个中断时，它必须读取中断状态（Interrupt Status）寄存器来判断是什么原因引起中断的。要实现最快的反应时间，建议对每个操作模式只允许有关的中断。这样，软件就不需要总是检查所有的状态标志来识别中断源，而只需要检查少数几个有关的位。

例如：

- 如果软件处于持续接收模式，建议允许数据包有效（Interrupt Enable 1 寄存器的“enpkvalid”位）和 CRC（Interrupt Enable 1 寄存器的“encrcerror”位）错误中断。
- 如果软件要发射数据包，允许数据包发送中断（Interrupt Enable 1 寄存器的“enpksent”位）就足够了。

5.2.2. 频率偏差（Frequency Deviation）寄存器

在使用 Si4432-V2 收发器芯片的情况下，频率偏差（Frequency Deviation）寄存器有两个作用：

在数据包发射期间，用于定义 RF 链接的发射频率偏差（Frequency Deviation）。

在接收模式期间，如果 AFC 被允许，该寄存器定义自动频率校正（AFC）的最大频率偏差。

在进入接收或发射模式前根据相应的功能来设置该寄存器是很重要的。

5.2.3. EZRadioPRO 收发器的初始化

以下代码部分给出了收发器芯片的初始化。它对接收或发射操作的所有寄存器进行设置：

```
/* ============================================================ *
* Initialize the Si4432 ISM chip *
* ============================================================ */
//SW reset
SpiWriteRegister(0x07, 0x80);//write 0x80 to the Operating & Function Control1 register
//wait for chip ready interrupt from the radio (while the nIRQ pin is high)
while ( NIRQ == 1);
//read interrupt status registers to clear the interrupt flags and release NIRQ pin
ItStatus1 = SpiReadRegister(0x03); //read the Interrupt Status1 register
ItStatus2 = SpiReadRegister(0x04); //read the Interrupt Status2 register
/*set the physical parameters*/
//set the center frequency to 915 MHz
SpiWriteRegister(0x75, 0x75); //write 0x75 to the Frequency Band Select register
SpiWriteRegister(0x76, 0xBB); //write 0xBB to the Nominal Carrier Frequency1 register
SpiWriteRegister(0x77, 0x80); //write 0x80 to the Nominal Carrier Frequency0 register
//set the desired TX data rate (9.6kbps)
SpiWriteRegister(0x6E, 0x4E); //write 0x4E to the TXDataRate 1 register
SpiWriteRegister(0x6F, 0xA5); //write 0xA5 to the TXDataRate 0 register
SpiWriteRegister(0x70, 0x2C); //write 0x2C to the Modulation Mode Control 1 register
/*set the modem parameters according to the excel calculator(parameters: 9.6 kbps, deviation: 45 kHz, channel filter BW:
```

```
112.1 kHz*/
SpiWriteRegister(0x1C, 0x05); //write 0x05 to the IF Filter Bandwidth register
SpiWriteRegister(0x20, 0xA1); //write 0xA1 to the Clock Recovery Oversampling Ratio register
SpiWriteRegister(0x21, 0x20); //write 0x20 to the Clock Recovery Offset 2 register
SpiWriteRegister(0x22, 0x4E); //write 0x4E to the Clock Recovery Offset 1 register
SpiWriteRegister(0x23, 0xA5); //write 0xA5 to the Clock Recovery Offset 0 register
SpiWriteRegister(0x24, 0x00); //write 0x00 to the Clock Recovery Timing Loop Gain 1 register
SpiWriteRegister(0x25, 0x13); //write 0x13 to the Clock Recovery Timing Loop Gain 0 register
SpiWriteRegister(0x1D, 0x40); //write 0x40 to the AFC Loop Gearshift Override register
SpiWriteRegister(0x72, 0x1F); //write 0x1F to the Frequency Deviation register
/*set the packet structure and the modulation type*/
//set the preamble length to 10bytes if the antenna diversity is used and set to 5bytes if not
#ifdef ANTENNA_DIVERSITY
   SpiWriteRegister(0x34, 0x18); //write 0x18 to the Preamble Length register
#else
   SpiWriteRegister(0x34, 0x0C); //write 0x0C to the Preamble Length register
#endif
//set preamble detection threshold to 20bits
SpiWriteRegister(0x35, 0x28); //write 0x28 to the Preamble Detection Control register
//Disable header bytes; set variable packet length (the length of the payload is defined by the
//received packet length field of the packet); set the synch word to two bytes long
SpiWriteRegister(0x33, 0x02); //write 0x02 to the Header Control2 register
//Set the sync word pattern to 0x2DD4
SpiWriteRegister(0x36, 0x2D); //write 0x2D to the Sync Word 3 register
SpiWriteRegister(0x37, 0xD4); //write 0xD4 to the Sync Word 2 register
//enable the TX & RX packet handler and CRC-16 (IBM) check
SpiWriteRegister(0x30, 0x8D); //write 0x8D to the Data Access Control register
//Disable the receive header filters
SpiWriteRegister(0x32, 0x00 ); //write 0x00 to the Header Control1 register
//enable FIFO mode and GFSK modulation
SpiWriteRegister(0x71, 0x63); //write 0x63 to the Modulation Mode Control 2 register
#ifdef ANTENNA_DIVERSITY
   //enable the antenna diversity mode
   SpiWriteRegister(0x08, 0x80);//write 0x80 to the Operating Function Control 2 register
#endif
/*set the GPIO's according the testcard type*/
   #ifdef ANTENNA_DIVERSITY
   SpiWriteRegister(0x0C, 0x17); //write 0x17 to the GPIO1 Configuration(set the Antenna 1 Switch used for antenna
   diversity )
   SpiWriteRegister(0x0D, 0x18); //write 0x18 to the GPIO2 Configuration(set the Antenna 2 Switch used for antenna
   diversity )
#endif
   #ifdef ONE_SMA_WITH_RF_SWITCH
   SpiWriteRegister(0x0C, 0x12); //write 0x12 to the GPIO1 Configuration(set the TX state)
   SpiWriteRegister(0x0D, 0x15); //write 0x15 to the GPIO2 Configuration(set the RX state)
#endif
```

```
/*set the non-default Si4432 registers*/
//set the VCO and PLL
SpiWriteRegister(0x5A, 0x7F); //write 0x7F to the VCO Current Trimming register
SpiWriteRegister(0x58, 0x80); //write 0x80 to the ChargepumpCurrentTrimmingOverride register
SpiWriteRegister(0x59, 0x40); //write 0x40 to the Divider Current Trimming register
//set the AGC
SpiWriteRegister(0x6A, 0x0B); //write 0x0B to the AGC Override 2 register
//set ADC reference voltage to 0.9V
SpiWriteRegister(0x68, 0x04); //write 0x04 to the Deltasigma ADC Tuning 2 register
SpiWriteRegister(0x1F, 0x03); //write 0x03 to the Clock Recovery Gearshift Override register
//set Crystal Oscillator Load Capacitance register
SpiWriteRegister(0x09, 0xD7); //write 0xD7 to the Crystal Oscillator Load Capacitance register
```

5.2.4. 接收数据包

在配置了 MCU 和芯片之后，演示程序进入持续接收模式，等待来自一个节点的数据包或等待按键输入。

```
/*enable receiver chain*/
SpiWriteRegister(0x07, 0x05); //write 0x05 to the Operating Function Control 1 register
//Enable two interrupts:
// a) one which shows that a valid packet received: 'ipkval'
// b) second shows if the packet received with incorrect CRC: 'icrcerror'
SpiWriteRegister(0x05, 0x03); //write 0x03 to the Interrupt Enable 1 register
SpiWriteRegister(0x06, 0x00); //write 0x00 to the Interrupt Enable 2 register
//read interrupt status registers to release all pending interrupts
ItStatus1 = SpiReadRegister(0x03); //read the Interrupt Status1 register
ItStatus2 = SpiReadRegister(0x04); //read the Interrupt Status2 register
```

MCU 在主循环轮询按键和芯片的中断引脚。

如果按键被按下，演示程序禁止接收器模式，产生一个数据包（包含“Button1”的有效载荷（payload））并将其发射到其他的节点：

```
/*MAIN Loop*/
while(1)
{
  //Poll the port pins of the MCU to figure out whether the push button is pressed or not
  if(PB1 == 0)
  {
    //Wait for releasing the push button
    while( PB1 == 0 );
    //disable the receiver chain (but keep the XTAL running to have shorter TX on time!)
    SpiWriteRegister(0x07, 0x01); //write 0x01 to the Operating Function Control 1 register
    //turn on the LED to show the packet transmission
    LED1 = 1;
    //The Tx deviation register has to set according to the deviation before every transmission (+-45kHz)
    SpiWriteRegister(0x72, 0x48); //write 0x48 to the Frequency Deviation register
    /*SET THE CONTENT OF THE PACKET*/
    //set the length of the payload to 8bytes
    SpiWriteRegister(0x3E, 8); //write 8 to the Transmit Packet Length register
    //fill the payload into the transmit FIFO
```

```
    SpiWriteRegister(0x7F, 0x42); //write 0x42 ('B') to the FIFO Access register
    SpiWriteRegister(0x7F, 0x55); //write 0x55 ('U') to the FIFO Access register
    SpiWriteRegister(0x7F, 0x54); //write 0x54 ('T') to the FIFO Access register
    SpiWriteRegister(0x7F, 0x54); //write 0x54 ('T') to the FIFO Access register
    SpiWriteRegister(0x7F, 0x4F); //write 0x4F ('O') to the FIFO Access register
    SpiWriteRegister(0x7F, 0x4E); //write 0x4E ('N') to the FIFO Access register
    SpiWriteRegister(0x7F, 0x31); //write 0x31 ('1') to the FIFO Access register
    SpiWriteRegister(0x7F, 0x0D); //write 0x0D (CR) to the FIFO Access register
    //Disable all other interrupts and enable the packet sent interrupt only.
    //This will be used for indicating the successfull packet transmission for the MCU
    SpiWriteRegister(0x05, 0x04); //write 0x04 to the Interrupt Enable 1 register
    SpiWriteRegister(0x06, 0x00); //write 0x03 to the Interrupt Enable 2 register
    //Read interrupt status regsiters. It clear all pending interrupts and the nIRQ pin goes back to high.
    ItStatus1 = SpiReadRegister(0x03);//read the Interrupt Status1 register
    ItStatus2 = SpiReadRegister(0x04);//read the Interrupt Status2 register
    /*enable transmitter*/
    //The radio forms the packet and send it automatically.
    SpiWriteRegister(0x07, 0x09);//write 0x09 to the Operating Function Control 1 register
    /*wait for the packet sent interrupt*/
    //The MCU just needs to wait for the 'ipksent' interrupt.
    while(NIRQ == 1);
    //read interrupt status registers to release the interrupt flags
    ItStatus1 = SpiReadRegister(0x03); //read the Interrupt Status1 register
    ItStatus2 = SpiReadRegister(0x04); //read the Interrupt Status2 register
    //wait a bit for showing the LED a bit longer
    for(delay = 0; delay < 10000;delay++);
    //turn off the LED
    LED1 = 0;
    //after packet transmission set the interrupt enable bits according receiving mode
    //Enable two interrupts:
    // a) one which shows that a valid packet received: 'ipkval'
    // b) second shows if the packet received with incorrect CRC: 'icrcerror'
    SpiWriteRegister(0x05, 0x03); //write 0x03 to the Interrupt Enable 1 register
    SpiWriteRegister(0x06, 0x00); //write 0x00 to the Interrupt Enable 2 register
    //read interrupt status registers to release all pending interrupts
    ItStatus1 = SpiReadRegister(0x03); //read the Interrupt Status1 register
    ItStatus2 = SpiReadRegister(0x04); //read the Interrupt Status2 register
    //set the Frequency Deviation register according to the AFC limiter
    SpiWriteRegister(0x72, 0x1F); //write 0x1F to the Frequency Deviation register
    /*enable receiver chain again*/
    SpiWriteRegister(0x07, 0x05); //write 0x05 to the Operating Function Control 1 register
}
```

芯片使用内置的接收数据包处理器（receive packet handler）来接收数据包，只有在接收到的是有效的数据包或数据包有错误 CRC 时才会产生一个中断给 MCU。在接收到一个中断后，MCU 通过读取中断状态（Interrupt States）寄存器检查中断原因：

如果接收到的数据包有错误 CRC，演示程序将丢弃该数据包，然后复位接收 FIFO，并返回接收模式。

如果接收到的数据包含有“Button1”有效载荷（payload），演示程序将闪烁 RX LED，并发送一个应答数据包给发起者。

如果接收到一个应答数据包，演示程序就闪烁 RX LED 指示成功的双向数据包发射。

```
//wait for the interrupt event
//If it occurs, then it means a packet received or CRC error happened
if( NIRQ == 0 )
{
  //disable the receiver chain
  SpiWriteRegister(0x07, 0x01); //write 0x01 to the Operating Function Control 1 register
  //read interrupt status registers
  ItStatus1 = SpiReadRegister(0x03); //read the Interrupt Status1 register
  ItStatus2 = SpiReadRegister(0x04);//read the Interrupt Status2 register
  /*CRC Error interrupt occured*/
  if( (ItStatus1 & 0x01) == 0x01 )
  { //reset the RX FIFO
    SpiWriteRegister(0x08, 0x02);//write 0x02 to the Operating Function Control 2 register
    SpiWriteRegister(0x08, 0x00);//write 0x00 to the Operating Function Control 2 register
    //blink all LEDs to show the error
    LED1 = 1;
    LED2 = 1;
    LED3 = 1;
    LED4 = 1;
    for(delay = 0; delay < 10000;delay++);
    LED1 = 0;
    LED2 = 0;
    LED3 = 0;
    LED4 = 0;
  }
  /*packet received interrupt occured*/
  if( (ItStatus1 & 0x02) == 0x02 )
  {
    //Read the length of the received payload
    length = SpiReadRegister(0x4B); //read the Received Packet Length register
    //check whether the received payload is not longer than the allocated buffer in the MCU
    if(length < 11)
    {
      //Get the reeived payload from the RX FIFO
      for(temp8=0;temp8 < length;temp8++)
      {
        payload[temp8] = SpiReadRegister(0x7F);//read the FIFO Access register
      }
      //check whether the acknowledgement packet received
      if( length == 4 )
      {
        if( memcmp(&payload[0], "ACK", 3) == 0 )
        {
```

```
            //blink LED2 to show that ACK received
            LED2 = 1;
            for(delay = 0; delay < 10000;delay++);
            LED2 = 0;
        }
    }
    //check whether an expected packet received, this should be acknowledged
    if( length == 8 )
    {
        if( memcmp(&payload[0], "BUTTON1", 7) == 0 )
        {
            //blink LED2 to show that the packet received
            LED2 = 1;
            for(delay = 0; delay < 10000;delay++);
            LED2 = 0;
            /*send back an acknowledgement*/
            //turn on LED1 to show packet transmission
            LED1 = 1;
            //The Tx deviation register has to set according to the deviation before
            //every transmission (+-45kHz)
            //write 0x48 to the Frequency Deviation register
            SpiWriteRegister(0x72, 0x48);
            /*set packet content*/
            //set the length of the payload to 4bytes
            //write 4 to the Transmit Packet Length register
            SpiWriteRegister(0x3E, 4);
            //fill the payload into the transmit FIFO
            //write 0x42 ('A') to the FIFO Access register
            SpiWriteRegister(0x7F, 0x41);
            //write 0x55 ('C') to the FIFO Access register
            SpiWriteRegister(0x7F, 0x43);
            //write 0x54 ('K') to the FIFO Access register
            SpiWriteRegister(0x7F, 0x4B);
            //write 0x0D (CR) to the FIFO Access register
            SpiWriteRegister(0x7F, 0x0D);
            //Disable all other interrupts and enable the packet sent interrupt only.
            //This will be used for indicating the successfull packet transmission for
            //the MCU
            //write 0x04 to the Interrupt Enable 1 register
            SpiWriteRegister(0x05, 0x04);
            //write 0x03 to the Interrupt Enable 2 register
            SpiWriteRegister(0x06, 0x00);
            //Read interrupt status regsiters. It clear all pending interrupts and the
            //nIRQ pin goes back to high.
            //read the Interrupt Status1 register
            ItStatus1 = SpiReadRegister(0x03);
```

```
                //read the Interrupt Status2 register
                ItStatus2 = SpiReadRegister(0x04);
                /*enable transmitter*/
                //The radio forms the packet and send it automatically.
                //write 0x09 to the Operating Function Control 1 register
                SpiWriteRegister(0x07, 0x09);
                /*wait for the packet sent interrupt*/
                //The MCU just needs to wait for the 'ipksent' interrupt.
                while(NIRQ == 1);
                //read interrupt status registers to release the interrupt flags
                //read the Interrupt Status1 register
                ItStatus1 = SpiReadRegister(0x03);
                //read the Interrupt Status2 register
                ItStatus2 = SpiReadRegister(0x04);
                //wait a bit for showing the LED a bit longer
                for(delay = 0; delay < 10000;delay++);
                //turn off the LED
                LED1 = 0;
                //after packet transmission set the interrupt enable bits according
                //receiving mode
                //Enable two interrupts:
                // a) one which shows that a valid packet received: 'ipkval'
                // b) second shows if the packet received with incorrect CRC: 'icrcerror'
                //write 0x03 to the Interrupt Enable 1 register
                SpiWriteRegister(0x05, 0x03);
                //write 0x00 to the Interrupt Enable 2 register
                SpiWriteRegister(0x06, 0x00);
                //read interrupt status registers to release all pending interrupts
                //read the Interrupt Status1 register
                ItStatus1 = SpiReadRegister(0x03);
                //read the Interrupt Status2 register
                ItStatus2 = SpiReadRegister(0x04);
                //set the Frequency Deviation register according to the AFC limiter
                //write 0x1F to the Frequency Deviation register
                SpiWriteRegister(0x72, 0x1F);
            }
        }
    }
  }
  /*enable receiver chain again*/
  SpiWriteRegister(0x07, 0x05);//write 0x05 to the Operating Function Control 1 register
}
}
}
```

注意：Si4431 的接收功能稍微不同；详细内容见“4.2.1.使用 Si4431-A0 接收数据包”。

6. 使用 FIFO 发射和接收长数据包

EZRadioPRO 器件有 64 字节发射和接收 FIFO；但有可能发射和接收的数据包有多达 255 字节的较长的有效载荷（payload）。以下示例源代码通过发射和接收有 128 字节有效载荷（payload）的数据包实现双向通信。

本示例使用的数据包格式使用以下较长数据包格式：

表 3 数据包配置

Preamble					Synchron pattern		Payload length	Payload				CRC	
AA	AA	AA	AA	AA	2D	D4	80	00	01	…	7F	16	1B
5 bytes					2 bytes		1 byte	128 bytes				2 bytes	

(10 bytes if Ant. Diversity is used)

6.1. 如何发送多于 64 字节的有效载荷（Payload）

内置发射和接收数据包处理器（packet handler）提供了三个不同的 FIFO 状态信号：发射 FIFO 几乎满（Transmit FIFO Almost Full）、发射 FIFO 几乎空（Transmit FIFO Almost Empty）和接收 FIFO 几乎满（Receive FIFO Almost Full）。这些信号可以由 MCU 用来发送和接收长度超过内置 FIFO 的数据包。

发射 FIFO 有两个可编程阀值，当发射 FIFO 中的数据达到这些阀值时，芯片将产生中断。第一个阀值是发射 FIFO 几乎满（Transmit FIFO Almost Full）（由 TX FIFO Control 1 寄存器设置）。如果填充到 FIFO 的字节数达到这个值时，芯片就会提供一个中断给 MCU 开始数据包发射。第二个阀值是发射 FIFO 几乎空（Transmit FIFO Almost Empty）（由 TX FIFO Control 2 寄存器设置）.如果发射 FIFO 的字节数达到这个值时，芯片将提供一个中断给 MCU。MCU 必须退出发射模式或填充更多的数据到发射 FIFO。

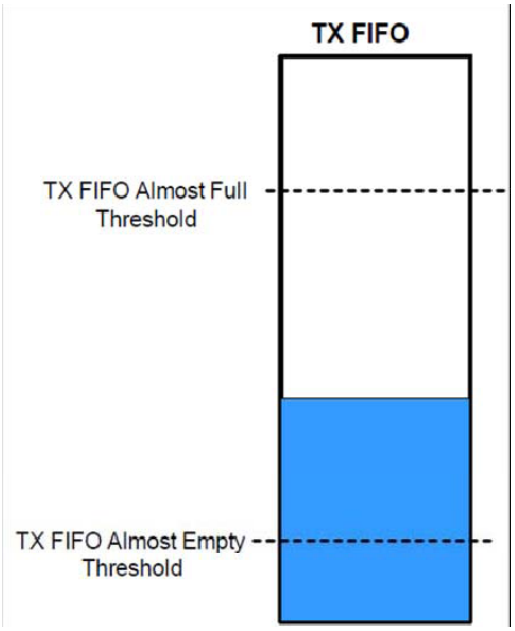

图 9. 发射 FIFO 状态信号

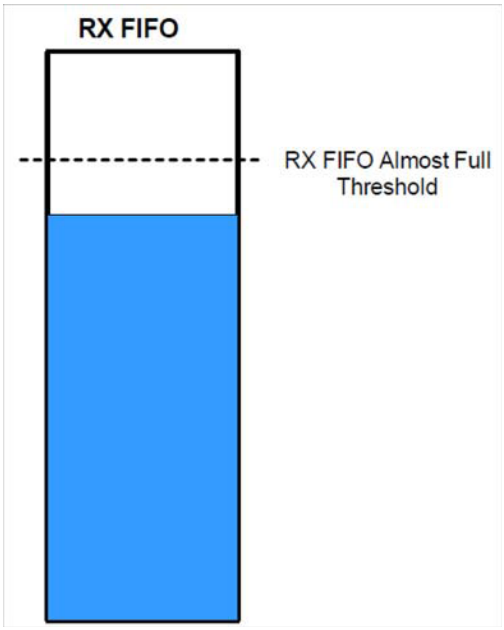

图 10. 接收 FIFO

接收 FIFO 有一个阀值：接收 FIFO 几乎满（Receive FIFO Almost Full）。如果存贮到接收 FIFO 的接收到的字节数达到这个阀值，芯片将产生一个中断给 MCU 来从 FIFO 读取数据。

6.2. 软件流程图

软件示例使用发射 FIFO 几乎空（Transmit FIFO Almost Empty）和接收 FIFO 几乎满（Receive FIFO Almost Full）状态标志通过下面的方法来发射和接收有 128 字节有效载荷（payload）的数据包：

发射数据包：演示程序设置发射 FIFO 几乎空（Transmit FIFO Almost Empty）阀值为 10 字节。程序将第一个 64 字节的有效载荷（payload）填充到发射 FIFO 并启动数据包发射。然后等待发射 FIFO 几乎空（Transmit FIFO Almost Empty）中断。当中断发生后，再填充 32 字节到 FIFO 并等待发射 FIFO 几乎空（Transmit FIFO Almost Empty）中断。当中断发生时，填充最后的 32 字节到 FIFO 并等待数据发送中断。

接收数据包：演示程序设置接收 FIFO 几乎满（Receive FIFO Almost Full）阀值为 54 字节。在接收到一个有效数据包中断后，MCU 等待接收 FIFO 几乎满（Receive FIFO Almost Full）中断，然后从接收 FIFO 读取 32 字节。芯片重复上术过程直到全部 128 字节有效载荷（payload）被接收。在 CRC 错误的情况下，MCU 将复位接收 FIFO 并丢弃数据包。

图 11 给出了软件流程图。

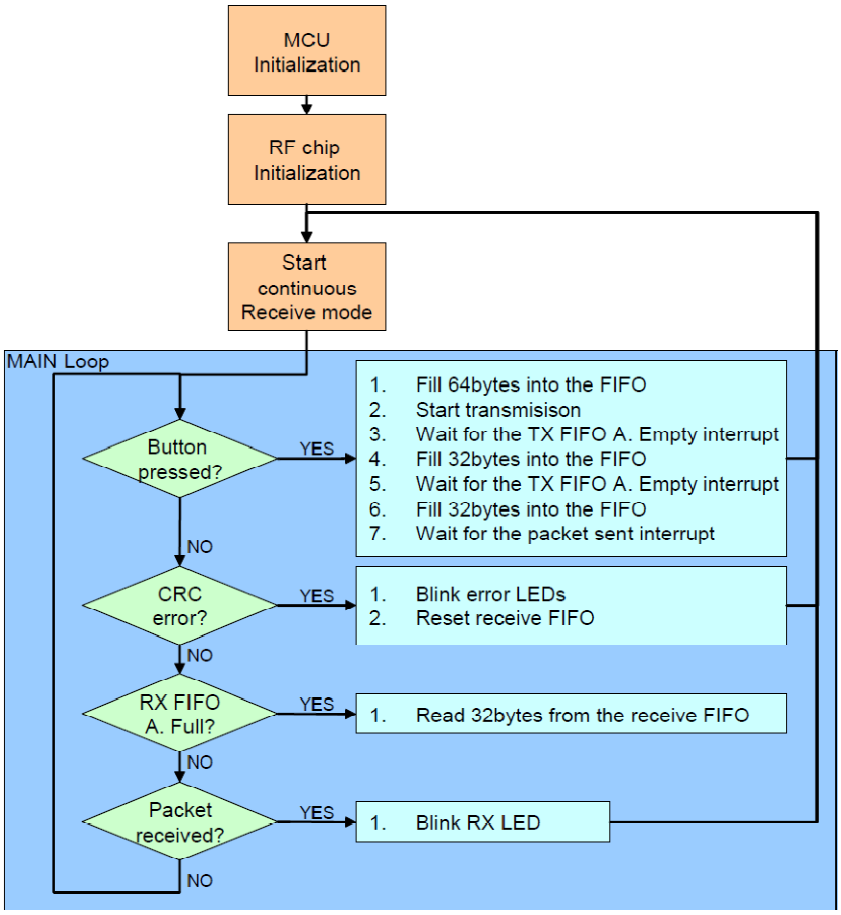

图 11. 演示程序软件流程图

6.3. 软件实现

6.3.1. MCU 和芯片的初始化

“5.2.3. EZRadioPRO 发射器的初始化”介绍了芯片的配置。

6.3.2. 数据包发射

以下代码段给出了如何实现数据包发射。为简化软件，发射数据包的有效载荷（payload）以字节数组形式存贮在 FLASH 中。该数组的声明位 C 代码的开头：

```
/* ============================================================ *
* GLOBAL VARIABLE *
* ============================================================ */
code U8 tx_packet[128] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
                          16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
                          32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47,
                          48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63,
                          64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79,
                          80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95,
                          96, 97, 98, 99, 100,101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111,
                          112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122,123, 124, 125, 126, 127};
```

如果按下按键，软件退出接收模式，将数据包的前 64 字节填充到发射 FIFO。MCU 启动数据包发射

并等待发射 FIFO 几乎空（Transmit FIFO Almost Empty）中断（Interrupt Status 1 寄存器的“itxffafull”位），然后填充接下来的 32 字节到 FIFO，再次重复这个过程，并等待数据包发送中断（Interrupt Status 1 寄存器的“ipksent”位）。在成功地发射数据包后返回持续接收模式。

```
//Poll the port pins of the MCU to figure out whether the push button is pressed or not
if(PB1 == 0)
{
   //Wait for releasing the push button
   while( PB1 == 0 );
   //disable the receiver chain (but keep the XTAL running to have shorter TX on time!)
   SpiWriteRegister(0x07, 0x01);//write 0x01 to the Operating Function Control 1 register
   //turn on the LED to show the packet transmission
   LED1 = 1;
   //The Tx deviation register has to set according to the deviation before every transmission (+-45kHz)
   SpiWriteRegister(0x72, 0x48);//write 0x48 to the Frequency Deviation register
   /*SET THE CONTENT OF THE PACKET*/
   //set the length of the payload to 8bytes
   SpiWriteRegister(0x3E, 128);//write 8 to the Transmit Packet Length register
   //fill the payload into the transmit FIFO
   //write 64bytes into the FIFO
   for(temp8=0;temp8<64;temp8++)
   {
      SpiWriteRegister(0x7F,tx_packet[temp8]);
   }
   //Disable all other interrupts and enable the FIFO almost empty interrupt only.
   SpiWriteRegister(0x05, 0x20); //write 0x20 to the Interrupt Enable 1 register
   SpiWriteRegister(0x06, 0x00); //write 0x00 to the Interrupt Enable 2 register
   //Read interrupt status regsiters. It clear all pending interrupts and the nIRQ pin goes back to high.
   ItStatus1 = SpiReadRegister(0x03);//read the Interrupt Status1 register
   ItStatus2 = SpiReadRegister(0x04);//read the Interrupt Status2 register
   /*enable transmitter*/
   //The radio forms the packet and send it automatically.
   SpiWriteRegister(0x07, 0x09); //write 0x09 to the Operating Function Control 1 register
   /*wait for TX FIFO almost empty interrupt*/
   while(NIRQ == 1);
   //TX FIFO almost empty interrupt occured -->
   //write the next 32bytes into the FIFO
   for(temp8=0;temp8<32;temp8++)
   {
   SpiWriteRegister(0x7F,tx_packet[64+temp8]);
   }
   //Read interrupt status regsiters. It clear all pending interrupts and the nIRQ pin goes back to high.
   ItStatus1 = SpiReadRegister(0x03); //read the Interrupt Status1 register
   ItStatus2 = SpiReadRegister(0x04); //read the Interrupt Status2 register
   /*Wait for TX FIFO almost empty interrupt*/
   while(NIRQ == 1);
   //TX FIFO almost empty interrupt occured -->
```

```
//write the last 32bytes into the FIFO
for(temp8=0;temp8<32;temp8++) //copy the remain 32 byte to the FIFO
{
SpiWriteRegister(0x7F,tx_packet[96+temp8]);
}
//Disable all other interrupts and enable the packet sent interrupt only.
//This will be used for indicating the successfull packet transmission for the MCU
SpiWriteRegister(0x05, 0x04);//write 0x04 to the Interrupt Enable 1 register
//Read interrupt status regsiters. It clear all pending interrupts and the nIRQ pin goes back to high.
ItStatus1 = SpiReadRegister(0x03); //read the Interrupt Status1 register
ItStatus2 = SpiReadRegister(0x04); //read the Interrupt Status2 register
/*wait for the packet sent interrupt*/
//The MCU just needs to wait for the 'ipksent' interrupt.
while(NIRQ == 1);
//Read interrupt status registers. It clear all pending interrupts and the nIRQ pin goes back to high.
ItStatus1 = SpiReadRegister(0x03); //read the Interrupt Status1 register
ItStatus2 = SpiReadRegister(0x04); //read the Interrupt Status2 register
//wait a bit for showing the LED a bit longer
for(delay = 0; delay < 10000;delay++);
//turn off the LED
LED1 = 0;
//after packet transmission set the interrupt enable bits according receiving mode
//Enable two interrupts:
// a) one which shows that a valid packet received: 'ipkval'
// b) second shows if the packet received with incorrect CRC: 'icrcerror'
// c) third shows if the RX fifo almost full: 'irxffafull
SpiWriteRegister(0x05, 0x13); //write 0x03 to the Interrupt Enable 1 register
SpiWriteRegister(0x06, 0x00); //write 0x00 to the Interrupt Enable 2 register
//Read interrupt status registers. It clear all pending interrupts and the nIRQ pin goes back to high.
ItStatus1 = SpiReadRegister(0x03); //read the Interrupt Status1 register
ItStatus2 = SpiReadRegister(0x04); //read the Interrupt Status2 register
//set the Frequency Deviation register according to the AFC limiter
SpiWriteRegister(0x72, 0x1F); //write 0x1F to the Frequency Deviation register
/*enable receiver chain again*/
SpiWriteRegister(0x07, 0x05); //write 0x05 to the Operating Function Control 1 register}
```

6.3.3. 数据包接收

如果发生了一个中断，MCU 检查中断原因：

- 在 CRC 错误的情况下，演示程序复位接收 FIFO，闪烁 LED 提示错误，并返回持续接收模式。
- 在接收 FIFO 几乎满（Receive FIFO Almost Full）中断（Interrupt Status 1 寄存器的“irxffafull”位）发生的情况下，意味着数据包正在被接收，MCU 需要从接收 FIFO 读取数据。每次中断，MCU 从接收 FIFO 读取 32 字节，以便为数据包剩余的字节留出空间来。
- 如果发生一个数据包接收中断，那么 MCU 退出接收模式，检查是否有期望的数据包到达并闪烁 RX LED。然后演示程序返回持续接收模式。

```
//wait for the interrupt event
//If it occurs, then it means a packet received or CRC error happened
if( NIRQ == 0 )
```

```
{
  //read interrupt status registers
  ItStatus1 = SpiReadRegister(0x03); //read the Interrupt Status1 register
  ItStatus2 = SpiReadRegister(0x04); //read the Interrupt Status2 register
  /*CRC Error interrupt occurred*/
  if( (ItStatus1 & 0x01) == 0x01 )
  {
    //reset the RX FIFO
    SpiWriteRegister(0x08, 0x02);
    //write 0x02 to the Operating Function Control 2 register
    SpiWriteRegister(0x08, 0x00);//write 0x00 to the Operating Function Control 2 register
    //blink all LEDs to show the error
    LED1 = 1;
    LED2 = 1;
    LED3 = 1;
    LED4 = 1;
    for(delay = 0; delay < 10000;delay++);
    LED1 = 0;
    LED2 = 0;
    LED3 = 0;
    LED4 = 0;
  }
  /*RX FIFO almost full interrupt occurred*/
  if( (ItStatus1 & 0x10) == 0x10 )
  {
    //read 32bytes from the FIFO
    for(temp8=pointer;temp8<pointer+32;temp8++)
    {
    rx_packet[temp8] = SpiReadRegister(0x7F);
    }
    //update receive buffer pointer
    pointer = pointer + 32;
  }
  /*packet received interrupt occurred*/
  if( (ItStatus1 & 0x02) == 0x02 )
  {
    //disable the receiver chain
    SpiWriteRegister(0x07, 0x01);//write 0x01 to the Operating Function Control 1 register
    //Read the length of the received payload
    length = SpiReadRegister(0x4B); //read the Received Packet Length register
    //get the remaining 32bytes from the RX FIFO
    for(temp8=pointer;temp8<pointer+32;temp8++)
    {
      rx_packet[temp8] = SpiReadRegister(0x7F);
    }
    //clear receive buffer pointer
```

```
        pointer = 0;
        //check whether the content of the packet is valid
        if(( length == 128 ) && ( rx_packet[127] == 127 ))
        {
           //turn on the LED
           LED1 = 1;
           //wait a bit for showing the LED a bit longer
           for(delay = 0; delay < 10000;delay++);
           //turn off the LED
           LED1 = 0;
        }
        /*enable receiver chain again*/
        SpiWriteRegister(0x07, 0x05);
        //write 0x05 to the Operating Function Control 1 register
    }
}
```

注意：Si4431 的接收功能稍微不同；详细内容见“4.2.1.使用 Si4431-A0 接收数据包”。

特别声明：

本文档是根据 Silicon Labs 的应用手册 AN415《EZRADIOPRO™ PROGRAMMING GUIDE》翻译而来。由于本人水平有限，其中难免有错，敬请谅解，如需要更准确的信息请参考英文原文。