13XX 2.4G收发效率调试

目录

**[前言 2](#_Toc27315)**

**[1 rx window 2](#_Toc26833)**

**[2 2.4G收发时序 2](#_Toc3128)**

[2.1 收发时序概述 2](#_Toc10609)

[2.2 收发测试 4](#_Toc22889)

**[3 软件优化 6](#_Toc6341)**

[3.1 发射端 6](#_Toc31317)

[3.2 接收端 7](#_Toc3694)

[3.3 信号线debug方式 8](#_Toc8375)

# 前言

此文档，提供2.4G收发效率的调试方式，主要解决在应用中发现收包会概率性丢包问题。

# 1 rx window

rx window是相对**接收端**而言的。处于rx window的芯片会以阻塞的方式等待空口数据包。如果收到数据包，则关闭rx window，开始接收数据，数据接收完成后，这一次收包就结束，跳出收包函数。如果没收到数据包，则一直阻塞监听，直到rx window timeout后，跳出收包函数。rx window一般在2.5ms左右。客户无需更改，软件中有配置（变量：mem\_24g\_rx\_window）。

当芯片在rx window的时刻，是不会处理软件业务的。芯片在执行应用代码的时候，芯片也没有处于rx window状态，称为**空闲态**。

# 2 2.4G收发时序

## 2.1 收发时序概述

2.4G通信遵循的规则：2.4G发射端（TX）发送数据时，2.4G接收端（RX）刚好处于监听状态（rx window），这时RX才能正确收到TX发送的数据。如果，TX发送数据的时刻，RX没有在监听状态，则RX无法送到这笔数据包，称为**丢包**。

为了方便观察，我们通过拉gpio信号线的方式，来观察TX和RX的收发时序。

channel0：TX的发射信号线，一个高电平脉冲，即为一次发包。

channel1：RX的接收信号线，高电平为rx window，低电平为空闲态。

channel2：高电平脉冲表示收到一笔正确的数据包。

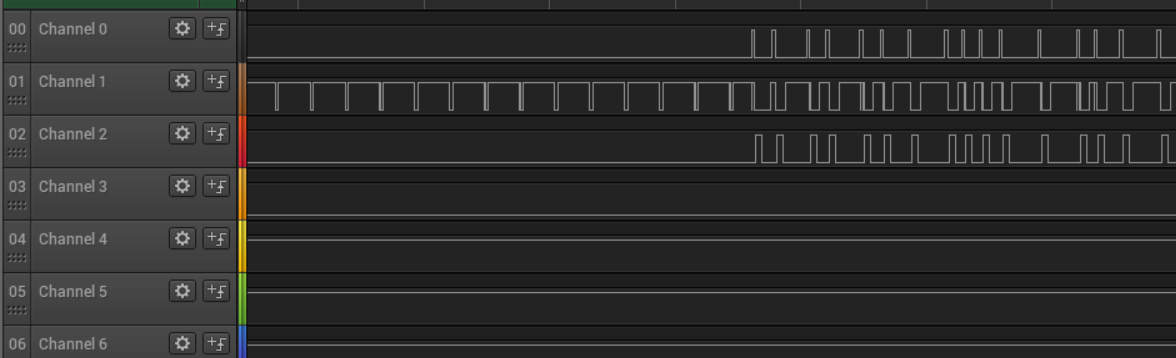


图1

我们对图1中信号线进行分析，channel1处于高电平时，RX处于rx window状态。图中TX的发射密度和RX的监听密度都相对较高，所以RX几乎能收到每一笔包。

## 2.2 收发测试

### 2.2.1 rx window密度对比测试

测试条件：

TX**以随机小于10ms**的间隔发包；

RX收包密度较高：



图2

RX适当降低收包密度（在软件中加了一些delay）：

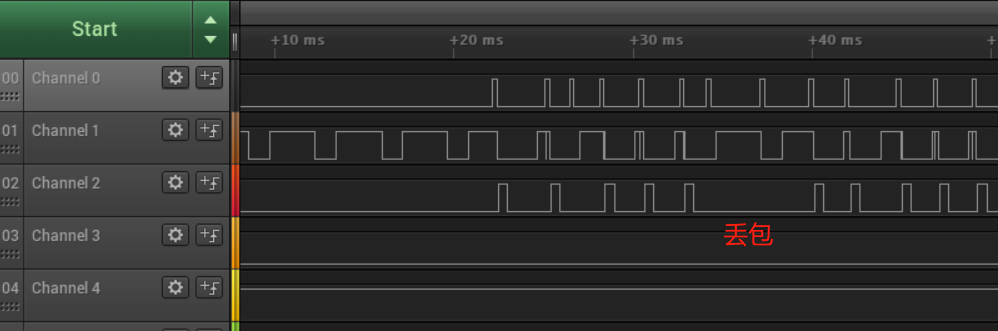


图3

对比图2和图3的channel1信号线，可以发现，图3中的rx window间的间隔更大，也就是处于空闲状态的时间变多了，当TX刚好在RX的空闲态时发包，就会造成丢包。

### 2.2.2 100ms发一次数据包：

下图中，channel2的缺口处，即为一次丢包。我们把缺口处放大，如图5所示。

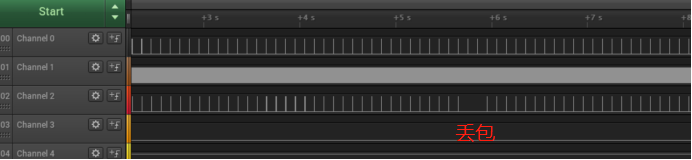


图4

图5中可以清晰的看到，TX的发包时刻，RX正处于空闲状态，即在处理其他的软件业务，导致了丢包。



图5

### 2.2.3 小节

综上所述，不管是rx window密度低，还是发包密度低都会导致丢包。但这是不可避免的，因为TX无法保证每次发包都在RX的rx window期间。所以，我们需要对数据包进行**重传**，同时接收端的软件可以设计过滤规则过滤掉重复收到的相同的数据包。

# 3 软件优化

## 3.1 发射端

发射的接口——g24\_sim\_tx，SDK已经在程序的固定位置对其进行调用，**无需再反复调用**。**flag**——mem\_le\_24g\_tx\_enable作为控制发包的开关。置1，允许发包；置0，停止发包。

在应用代码中，没有固定delay的情况下，每5ms不到（一般在2~3ms）程序就会访问一次发包函数。所以在使能发包的情况下，每5ms不到，都会发一次包。

数据的发送，**一定要做重传**，否则一定会有概率会丢包。当有事件触发需要发包时，只需组好包，enable flag，重传结束后再关掉flag即可。

提供两种重传的方式供参考：

1. 可以发送固定的时间，比如重传100ms，100ms超时后，disable flag即可；
2. 可以自己写一个计数的标志，比如每次固定重传10次，10次 结束后，disable flag。

## 3.2 接收端

接收端接口——g24\_sim\_rx，已经在程序的固定位置对其进行调用，**无需再反复调用。**接收端要注意，应用代码不要有过多的delay阻塞，否则会导致芯片处于rx window的密度降低，增大丢包的概率。

根据实际应用的经验来看，在结合TX的重传，可以做到几乎不会丢包。

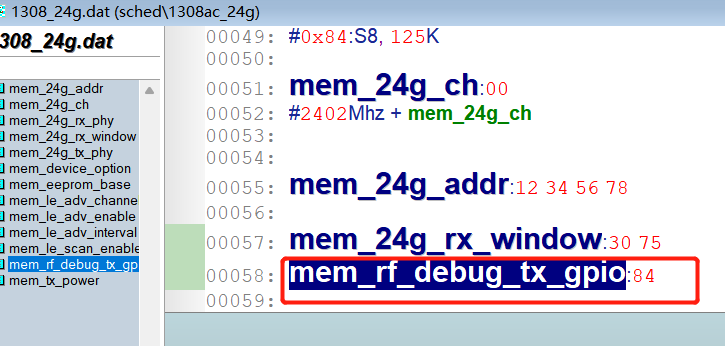
## 3.3 信号线debug方式

在调试优化丢包问题时，通常是通过gpio拉信号线的方式，来debug。调试时，只需指定gpio即可。

**TX端**，开始发包,拉高gpio，发包结束，拉低gpio。所以我们看到的高电平脉冲，就是一次发包的耗时。

变量——mem\_rf\_debug\_tx\_gpio。bit7：始终置1，bit0~6：指定gpio。

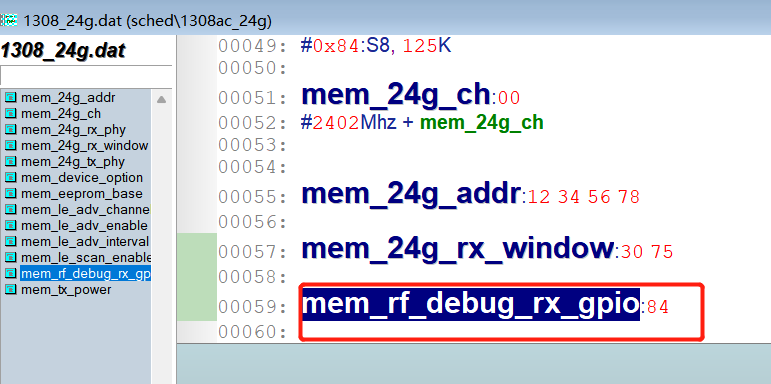
例如，指定gpio4位TX的信号线：



**RX端**，rx window开始，拉高gpio，rx window结束，拉低gpio。一次高电平脉冲，为阻塞收包的耗时。

变量——mem\_rf\_debug\_rx\_gpio。bit7：始终置1，bit0~6：指定gpio

例如，指定gpio4位RX的信号线：



信号线示例：

软件场景：

TX每500ms发一次包，每次重传100ms。

如下图：

channel0：TX的发射信号线，一个高电平脉冲，即为一次发包。

channel1：RX的接收信号线，高电平为rx window，低电平为空闲态。

channel2：高电平脉冲表示收到一笔正确的数据包。

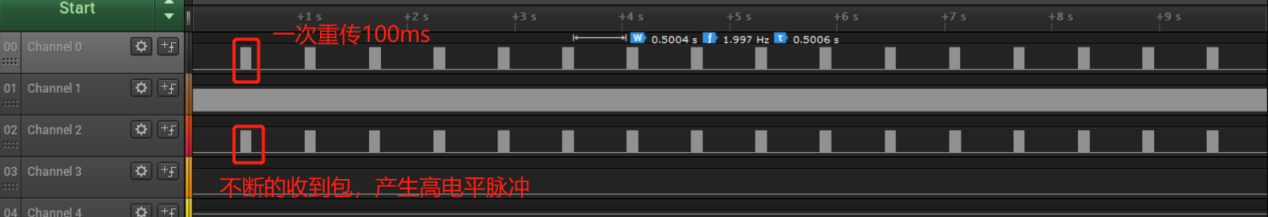


图6

把图6中“一次重传100ms”处放大，如图7所示：

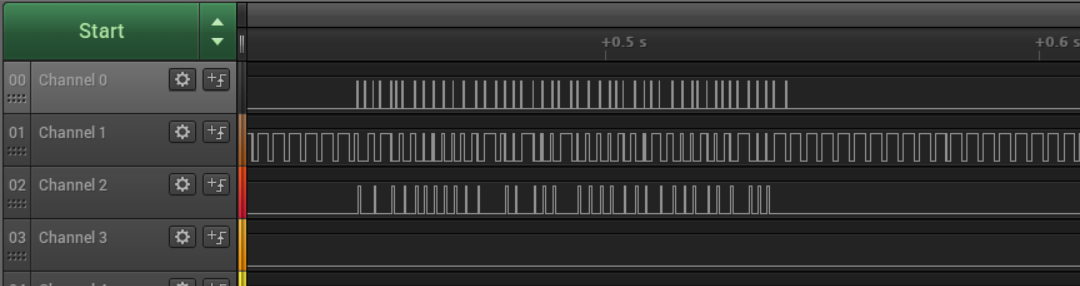


图7

由图6可知，在进行重传的情况下，根据channel2显示，不存在有哪一次”100ms”一个包没收到的情况。由图7可以，100ms内可以发送几十笔包，发包的效率很高。