ASR LoRa Software Training

——徐关平

2021年7月

ASR LoRa Software Training

- 第一部分 LoRa简介
- 第二部分 LoRaWAN
- 第三部分 ASR LoRa
- 第四部分 FreeRTOS移植
- 第五部分 LoRaWAN入网调试
- 第六部分 CAD调试与空中唤醒

第一部分 LoRa简介

LoRa简介

LoRa（LongRange）是一种基于CSS的扩频调制技术，能够显著提升通信距离，适用于低频次、小数据量、长距离的LPWAN低功耗广域网，具有如下优点：

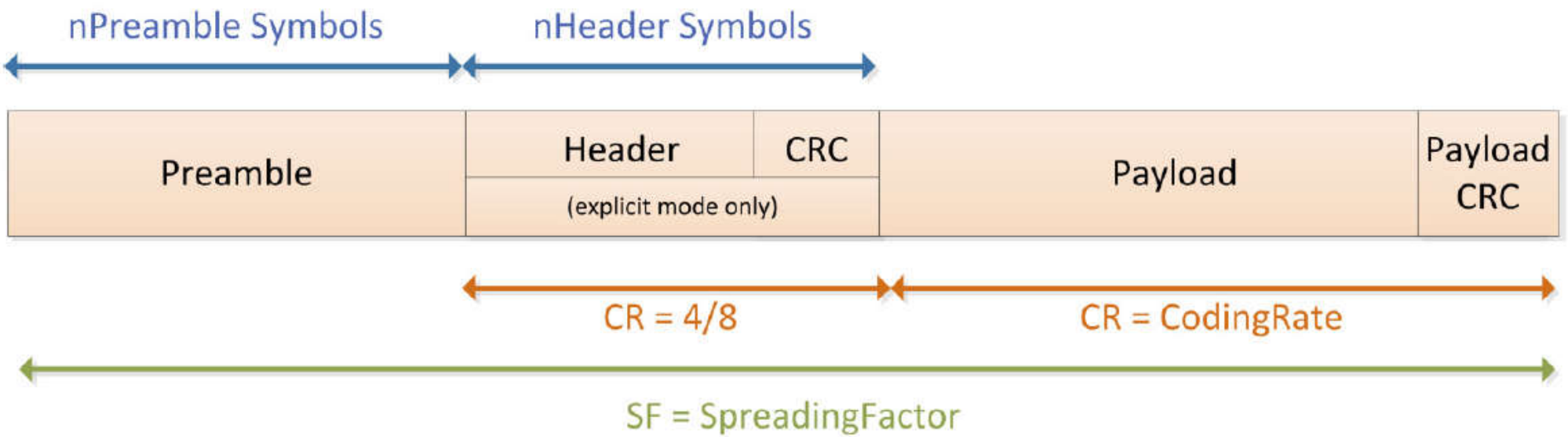
- 信号淹没在噪声中，接收方只需要知道正交的扩频序列即可从噪声中恢复信号
- 比FSK更好的灵敏度（更好的Eb/No），更强的抗干扰、抗噪声和抗阻塞能力
- 不同扩频因子可以使用相同的信道(不同的扩频序列之间是正交的，因此频率可以复用)
- 宽带扩频技术，抗多径、抗衰落能力强

LoRa主要射频参数

使用LoRa通信，发送与接收端需要配置相同的射频参数，详见下表：

参数	描述
SyncWord	同步字，sx127x系列长度1字节，sx126x系列长度2字节，sx126x与sx127x保持兼容
Preamble	前导码，LoRaWAN长度8
Frequency	ASR LoRa产品支持150MHz~960MHz的频率范围
SpreadFactor	确切为扩频因子的Log_2值，取值范围是5~12，实际扩频因子值为2^{SF}
BandWidth	调制带宽，最低可支持7.81kHz，低于62.5kHz必须要使用TCXO以确保精度，LoRaWAN使用125kHz/250kHz/500kHz
CodeRate	LoRa采用前向编码纠错技术，支持4/5、4/6、4/7和4/8的码率
CRC	硬件CRC校验，LoRaWAN上行开启，下行关闭
InvertIQ	LoRaWAN上行为标准IQ，下行为逆置IQ

LoRa报文格式


nPreamble Symbols
nHeader Symbols
Preamble
Header
CRC
(explicit mode only)
Payload
Payload CRC
CR = 4/8
CR = CodingRate
SF = SpreadingFactor

LoRa基本射频性能

通常，影响LoRa模组或者产品通信成功率是以下三项基本射频性能：

- 32M频偏：测试工具为频谱仪。32M晶振的精度在20ppm以内，10ppm以内最好。
- 发射功率：测试工具为频谱仪。接近21dBm为最佳，调节射频匹配网络改善。
- 接收灵敏度：测试工具为信号发生器。@SF12@BW125kHz@CR4/5达到接近-137dBm为最佳，调节射频匹配网络改善。

LoRa参考资料说明

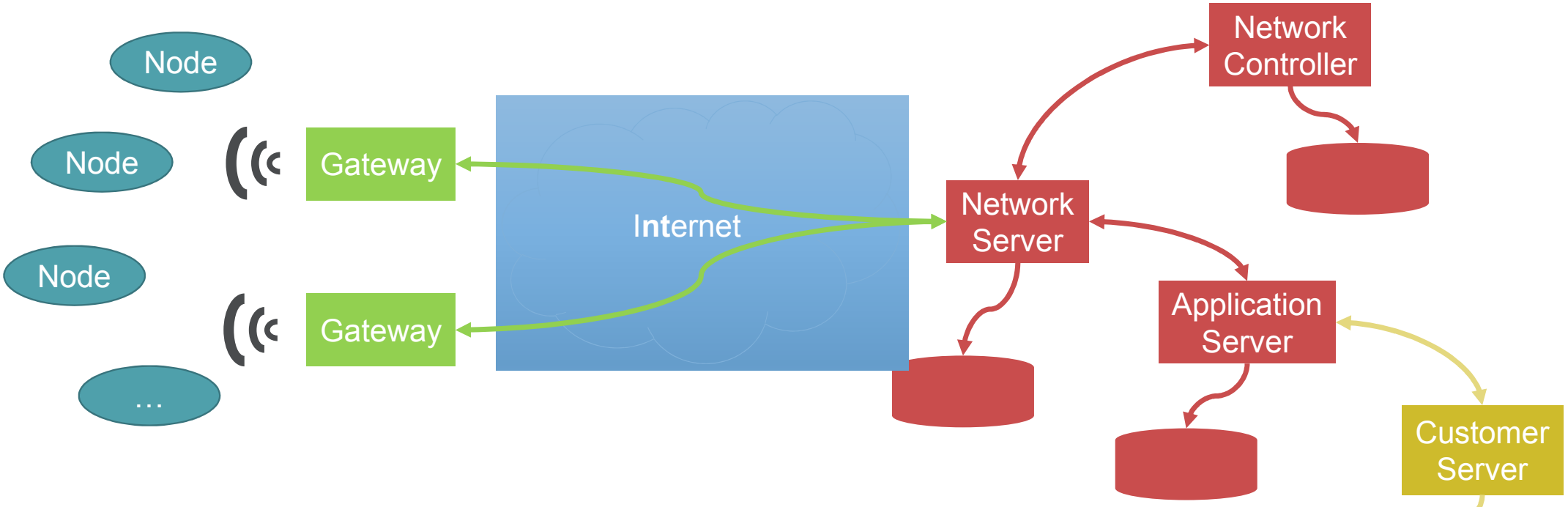
资料名	描述
Introduction to LoRa Technology_for customers	LoRa通信技术介绍
LoRa™Modulation Basics	LoRa调制技术基础
LORA_modulation_whitepaper	LoRa调制技术白皮书
LoRa-FAQs	关于LoRa技术的FAQ
DS_SX1261-2_V1.2	LoRa射频Datasheet

第二部分 LoRaWAN

LoRaWAN简介

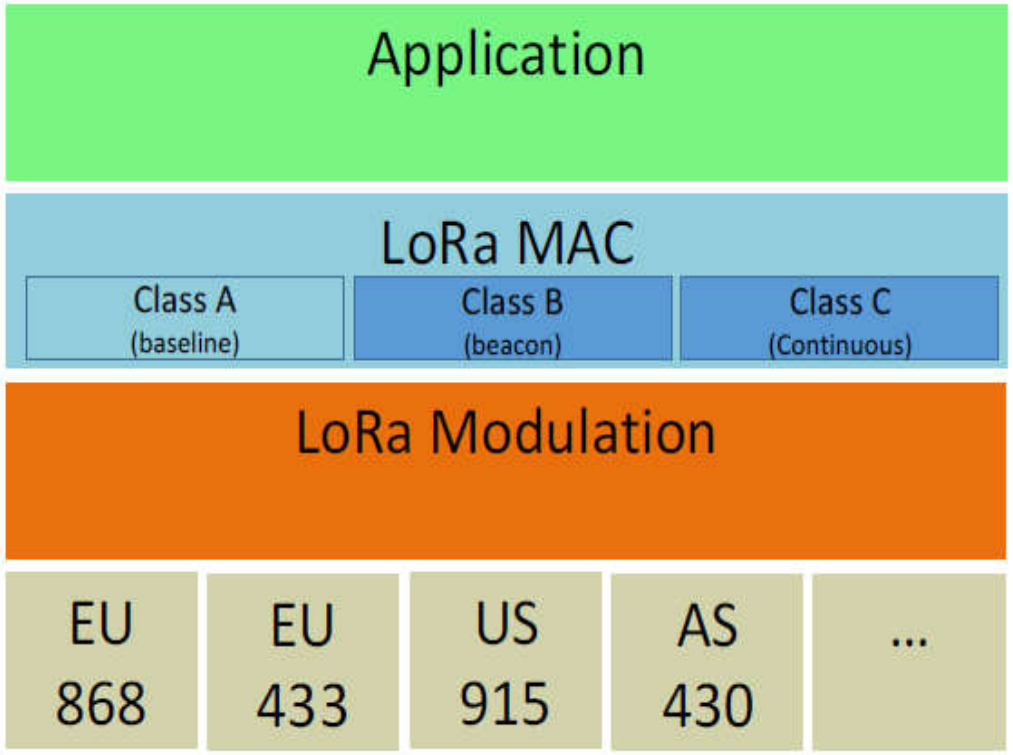
LoRaWAN是基于LoRa调制技术的一个MAC层规范协议，官方机构为LoRa Alliance，LoRaWAN的网络拓扑为星型，具有高容量、长距离、低功耗的优点，特别适合LPWAN。针对依靠电池供电的低成本WSN（Wireless Sensor Network）网络，LoRaWAN协议特别的做了一些优化，在网络延迟与电池寿命间做了平衡。

LoRaWAN网络结构



网关只是负责收发的通道，不可解析报文，所有节点的DevEUI/AppEUI/AppKEY存储在NS。

LoRaWAN协议层次



Application

MAC

MAC options

Modulation

Regional ISM band

LoRaWAN主要国家和地区频段划分

LoRaWAN针对全球不同的国家和地区划分了不同的通信频段，各频段规定了不同的信道划分、数据率等LoRaWAN参数，主要国家和地区LoRaWAN频段划分如下

国家地区	频段	范围	DR(通信速率)											
中国	CN470	470MHz - 510MHz	0	1	2	3	4	5						
			SF12 125kHz	SF11 125kHz	SF10 125kHz	SF9 125kHz	SF8 125kHz	SF7 125kHz						
欧洲	EU868	863MHz - 870MHz	0	1	2	3	4	5	6					
			SF12 125kHz	SF11 125kHz	SF10 125kHz	SF9 125kHz	SF8 125kHz	SF7 125kHz	FSK 50kbps					
北美	US915	902MHz - 928MHz	0	1	2	3	4	5:7	8	9	10	11	12	13
			SF12 125kHz	SF11 125kHz	SF10 125kHz	SF9 125kHz	SF8 500kHz	RFU	SF12 500kHz	SF11 500kHz	SF10 500kHz	SF9 500kHz	SF8 500kHz	SF7 500kHz

LoRaWAN节点类型

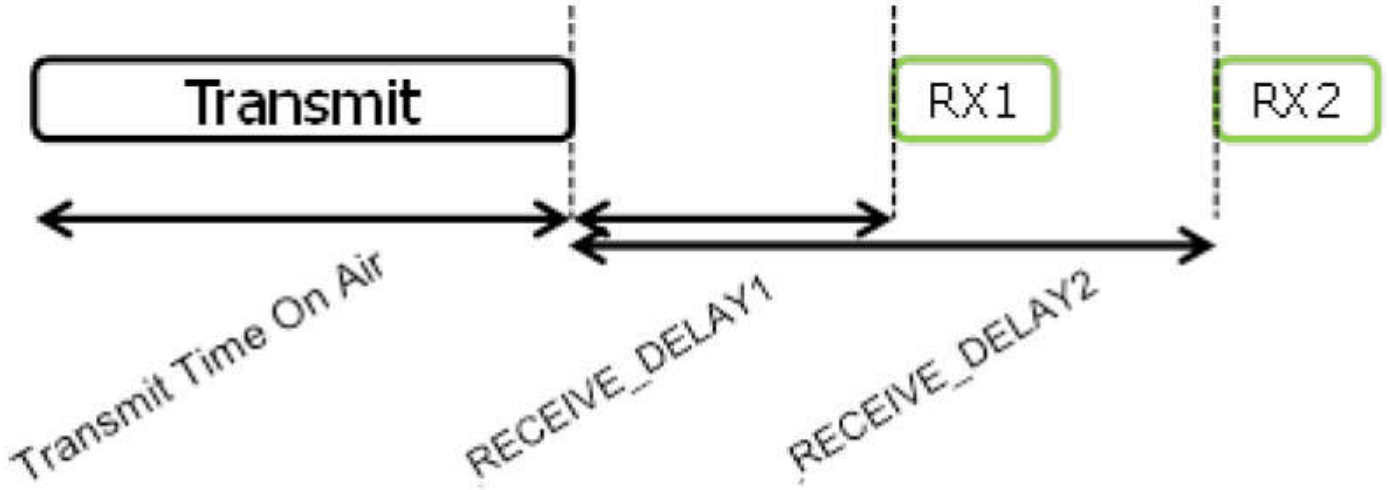
Class name	Intended usage
A (« all »)	**Battery powered sensors**, or actuators with no latency constraint Most energy efficient communication class. Must be supported by all devices
B (« beacon »)	**Battery powered actuators** Energy efficient communication class for latency controlled downlink. Based on slotted communication synchronized with a network beacon.
C (« continuous »)	**Mains powered actuators** Devices which can afford to listen continuously. No latency for downlink communication.

通过对典型应用场景的归纳分类， LoRaWAN定义了三种节点类型：

- Class A（**All End Device**）
- Class B（**Beacon**）
- Class C（**Continuous**）

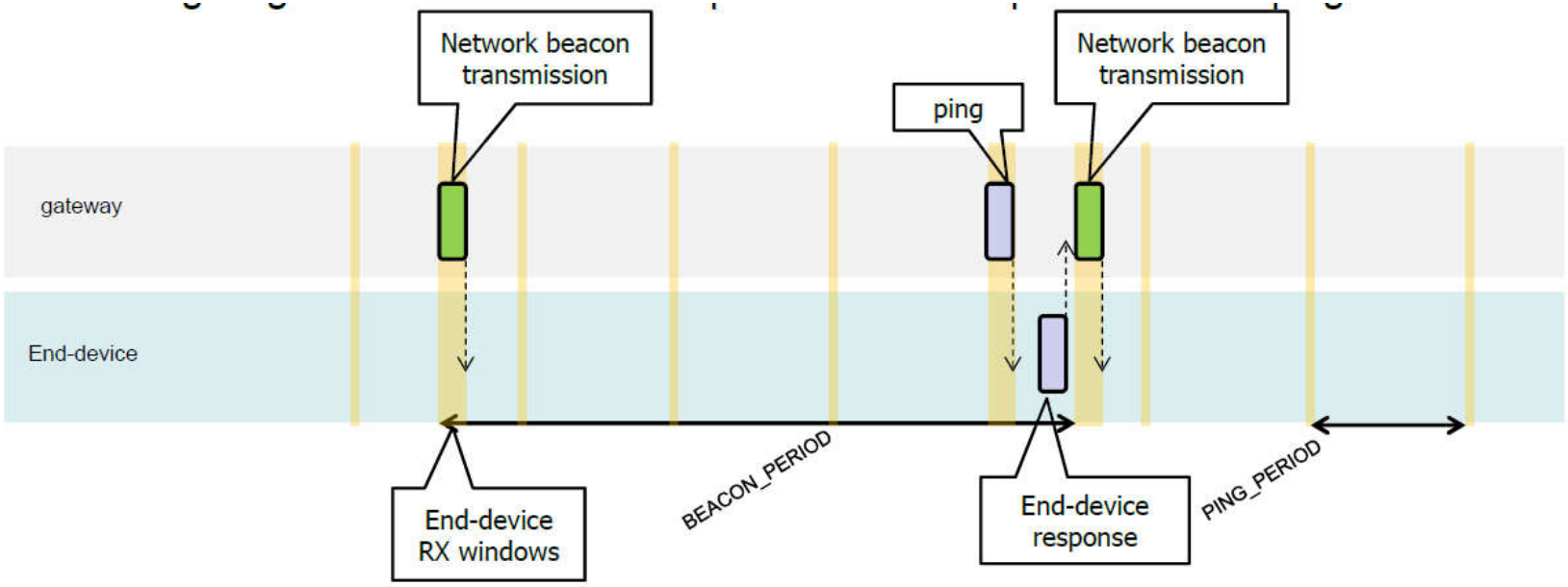
Class A Aloha收发时序

- 节点在发送上行消息后开启RX1接收
- 若RX1无接收则开启RX2



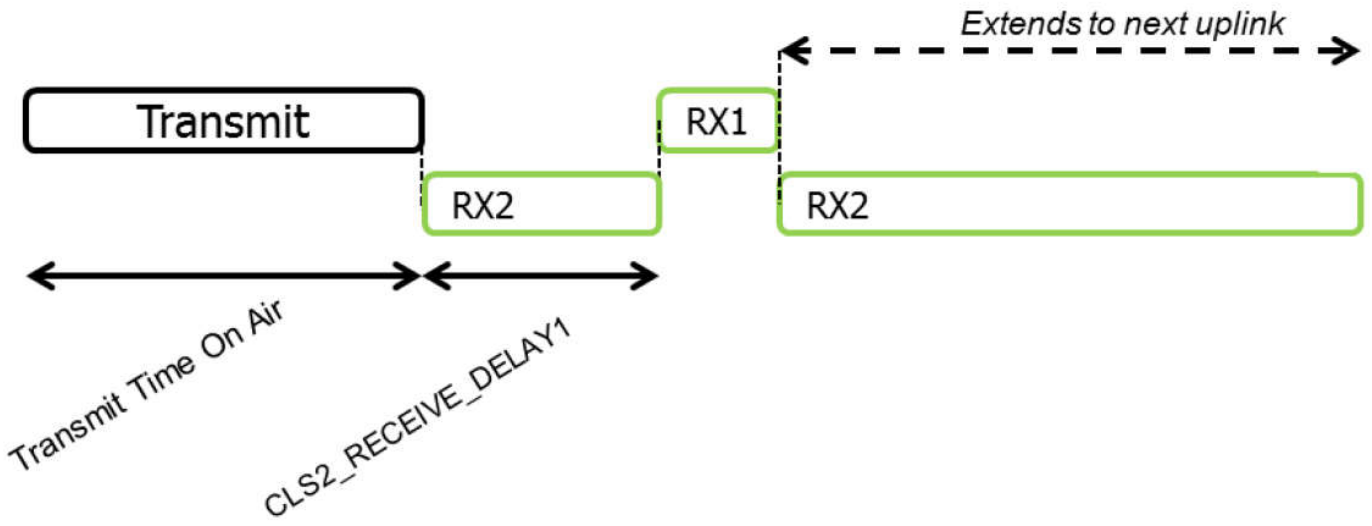
Class B Beacon和Ping时序

- 网关定时发送Beacon帧，Beacon周期为128秒，节点同步接收
- 节点Ping周期可以是1/2/4/8/16/32/64/128秒，节点每个Ping周期开启一次接收
- 节点需要定期上报以通知NS更新消息路径

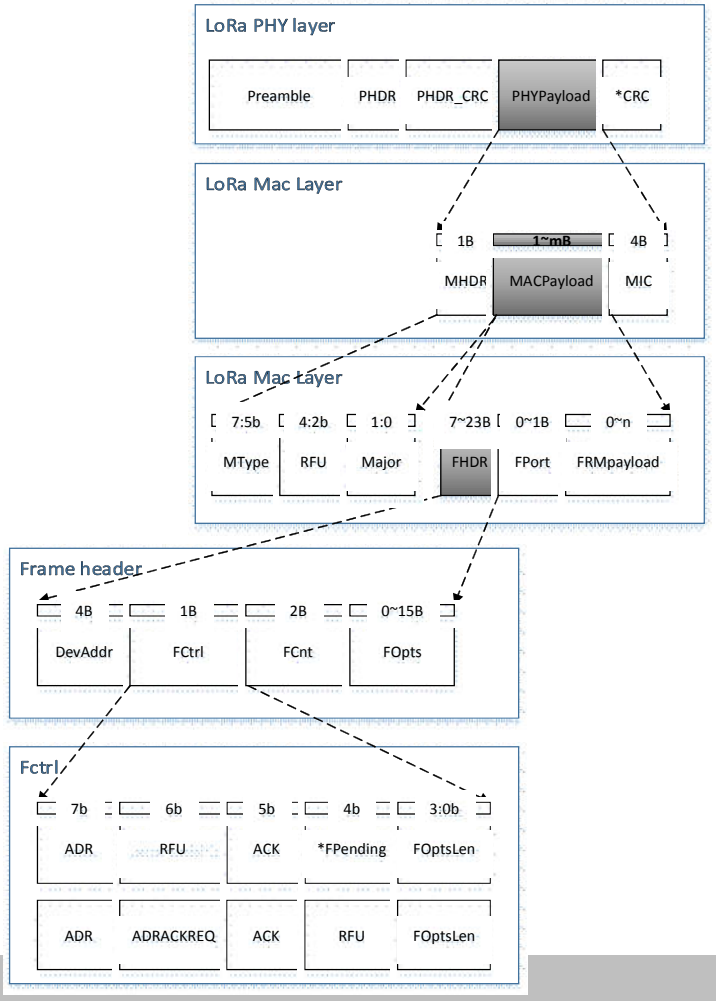


Class C持续接收

- LoRaMAC层无特定指令可以告知NS节点处于Class C模式
- 若有上行，则RX1必须开启
- 除去Uplink和RX1窗口，其他时间都处于RX2接收



LoRaMAC消息格式



- PhyLayer消息上行有CRC校验，下行无CRC校验
- Fopts由FOptsLen决定

MType	Description
000	Join Request
001	Join Accept
010	Unconfirmed Data Up
011	Unconfirmed Data Down
100	Confirmed Data Up
101	Confirmed Data Down
110	RFU
111	Proprietary

LoRaWAN参考资料说明

资料	描述
LoRaWAN102-20161012_1398_1	LoRaWAN 协议规范
LoRaWANRegionalParametersv1.0.2_final_1944_1	LoRaWAN Region参数规范

第三部分 ASR LoRa

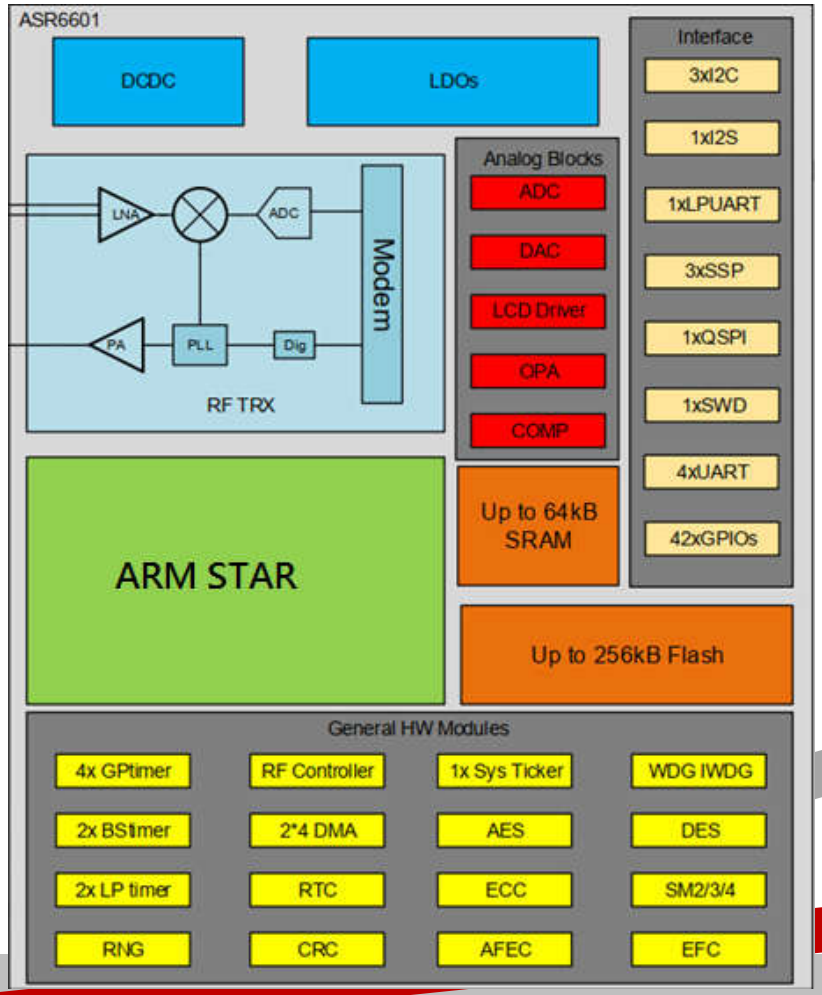
ASR LoRa系列产品

形态	型号	描述
MCU+LoRa SiP	ASR6501	Cortex-M0+ MCU + SX1262, 128KB Flash+16KB SRAM, 150MHz~960MHz, QFN48
	ASR6502	Cortex-M0+ MCU + SX1262, 128KB Flash+16KB SRAM, 150MHz~960MHz, QFN60
	ASR6505	STM8L152 MCU + SX1262, 64KB Flash+4KB SRAM+2KB E2P, 150MHz~960MHz, QFN68
LoRa RF SiP	ASR6500SL	SX1262 + 32M XO + RF Switch + Matching Network，433MHz~510MHz, LGA12
	ASR6500SLT	SX1262 + 32M TCXO + RF Switch + Matching Network，433MHz~510MHz, LGA12
	ASR6500SHT	SX1262 + 32M TCXO + RF Switch + Matching Network，868MHz~928MHz, LGA12
	ASR6500SLC	LLCC68 + 32M XO + RF Switch + Matching Network，433MHz~510MHz, LGA12
LoRa SoC	ASR6601CB	ARM STAR MCU + Sub 1G TRX, 128KB Flash+16KB SRAM, 150MHz~960MHz, QFN48
	ASR6601SE	ARM STAR MCU + Sub 1G TRX, 256KB Flash+64KB SRAM, 150MHz~960MHz, QFN68

ASR6601简介

ASR6601是由翱捷科技股份有限公司研发的首颗支持LoRa调制方式的国产低功耗广域网无线通信SoC，内部集成了Sub 1GHz射频收发机、ARM STAR微处理器。

- 支持150MHz~960MHz的频率范围
- 最大输出功率+22dBm
- 最高灵敏度达-148dBm
- 最大可支持256KB的Flash和64KB的SRAM
- 具有出色的低功耗性能
- 集成了丰富的外设，以及国际和国密的加密算法硬件加速器。



ASR6601 源代码目录结构

- projects目录下提供了各个外设使用的demo工程
- ASR6601CB与ASR6601SE的Flash空间、SRAM空间不同，demo工程分开
- 典型LoRa应用demo工程参考projects/ASR6601XX-EVAL/lorawan下的class_a、class_c和lorawan_at三个工程，其中lorawan_at支持ClassB

ASR6601 ClassA工程解构

Makefile

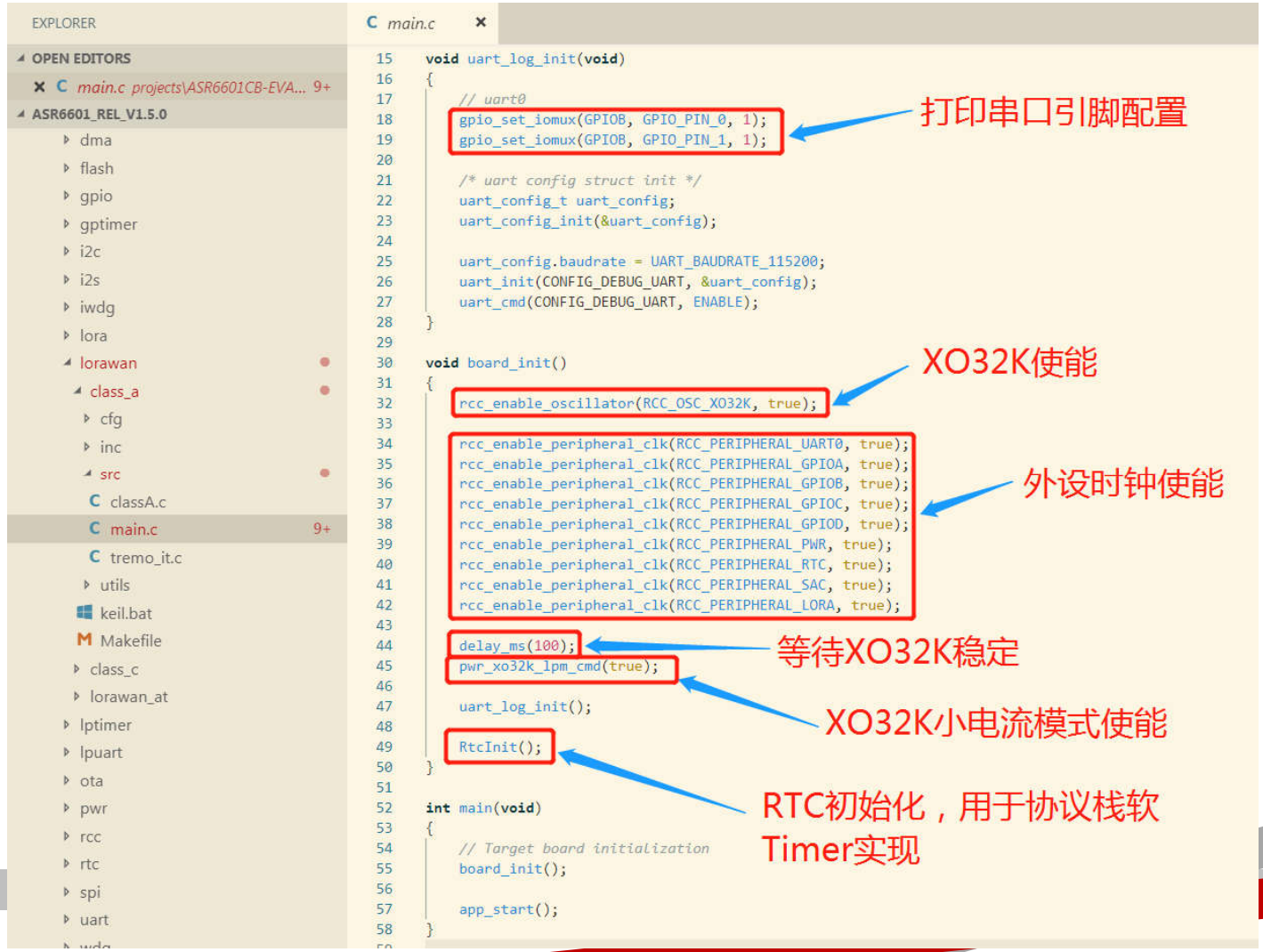
```
1
2  PROJECT := $(notdir $(CURDIR))
3
4  $(PROJECT)_SOURCE := $(wildcard src/*.c)  \
5      $(TREMO_SDK_PATH)/platform/system/printf-stdarg.c  \
6      $(TREMO_SDK_PATH)/platform/system/system_cm4.c  \
7      $(TREMO_SDK_PATH)/platform/system/startup_cm4.S \
8      $(wildcard $(TREMO_SDK_PATH)/drivers/peripheral/src/*.c)  \
9      $(wildcard $(TREMO_SDK_PATH)/lora/system/*.c)  \
10     $(wildcard $(TREMO_SDK_PATH)/lora/system/crypto/*.c)  \
11     $(wildcard $(TREMO_SDK_PATH)/lora/radio/sx126x/*.c)  \
12     $(wildcard $(TREMO_SDK_PATH)/lora/driver/*.c) \
13     $(wildcard $(TREMO_SDK_PATH)/lora/mac/*.c) \
14     $(TREMO_SDK_PATH)/lora/mac/region/Region.c \
15     $(TREMO_SDK_PATH)/lora/mac/region/RegionCommon.c \
16     $(TREMO_SDK_PATH)/lora/mac/region/RegionCN470.c
17
18 $(PROJECT)_INC_PATH := inc \
19     $(TREMO_SDK_PATH)/platform/CMSIS \
20     $(TREMO_SDK_PATH)/platform/common \
21     $(TREMO_SDK_PATH)/platform/system \
22     $(TREMO_SDK_PATH)/drivers/crypto/inc \
23     $(TREMO_SDK_PATH)/drivers/peripheral/inc \
24     $(TREMO_SDK_PATH)/lora/driver \
25     $(TREMO_SDK_PATH)/lora/system \
26     $(TREMO_SDK_PATH)/lora/system/crypto \
27     $(TREMO_SDK_PATH)/lora/radio \
28     $(TREMO_SDK_PATH)/lora/radio/sx126x \
29     $(TREMO_SDK_PATH)/lora/mac \
30     $(TREMO_SDK_PATH)/lora/mac/region
31
32 $(PROJECT)_CFLAGS  := -Wall -Os -ffunction-sections -mfpu=fpv4-sp-d16 -mfloat-abi=softfp -fsingle-precision-constant -std=gnu
33 $(PROJECT)_DEFINES := -DCONFIG_DEBUG_UART=UART0 -DREGION_CN470
34
35 $(PROJECT)_LDFLAGS := -Wl,--gc-sections -Wl,--wrap=printf -Wl,--wrap=sprintf -Wl,--wrap=snprintf
36
37 $(PROJECT)_LIBS := $(TREMO_SDK_PATH)/drivers/crypto/lib/libcrypto.a
38
39 $(PROJECT)_LINK_LD := cfg/gcc.ld
```

标准打印
芯片初始化
中断向量和复位处理
加密算法
指定打印串口为UART0
指定CN470频段
引入加密算法库
指定链接脚本

gcc.ld

```
1  /*
2  ******************************************************************************
3  */
4
5  /* Entry Point */
6  ENTRY(Reset_Handler)
7
8  /* Highest address of the user mode stack */
9  _estack = 0x20004000;    /* end of RAM */
10
11 /* Generate a link error if heap and stack don't fit into RAM */
12 _HEAP_SIZE = 0x0000;      /* required amount of heap  */
13 _STACK_SIZE = 0x1000; /* required amount of stack */
14
15 /* Specify the memory areas */
16 MEMORY
17 {
18     FLASH (rx)      :  ORIGIN = 0x08000000, LENGTH = 128K
19     RAM (xrw)       :  ORIGIN = 0x20000000, LENGTH = 16K
20 }
21
```

ASR6601 ClassA工程初始化



ASR6601 ClassA工程LoRaMAC初始化

classA.c

```
431 int app_start( void )
432 {
433     LoRaMacPrimitives_t LoRaMacPrimitives;
434     LoRaMacCallback_t LoRaMacCallbacks;
435     MibRequestConfirm_t mibReq;
436
437     DeviceState = DEVICE_STATE_INIT;
438
439     printf("ClassA app start\r\n");
440     while( 1 )
441     {
442         switch( DeviceState )
443         {
444             case DEVICE_STATE_INIT:
445             {
446                 LoRaMacPrimitives.MacMcpsConfirm = McpsConfirm;
447                 LoRaMacPrimitives.MacMcpsIndication = McpsIndication;
448                 LoRaMacPrimitives.MacMlmeConfirm = MlmeConfirm;
449                 LoRaMacPrimitives.MacMlmeIndication = MlmeIndication;
450                 LoRaMacCallbacks.GetBatteryLevel = BoardGetBatteryLevel;
451                 LoRaMacInitialization( &LoRaMacPrimitives, &LoRaMacCallbacks, ACTIVE_REGION );
452
453                 TimerInit( &TxNextPacketTimer, OnTxNextPacketTimerEvent );
454
455                 mibReq.Type = MIB_ADR;
456                 mibReq.Param.AdrEnable = LORAWAN_ADR_ON;
457                 LoRaMacMibSetRequestConfirm( &mibReq );
458
459                 mibReq.Type = MIB_PUBLIC_NETWORK;
460                 mibReq.Param.EnablePublicNetwork = LORAWAN_PUBLIC_NETWORK;
461                 LoRaMacMibSetRequestConfirm( &mibReq );
462
463                 lwan_dev_params_update();
464
465                 DeviceState = DEVICE_STATE_JOIN;
466                 break;
467             }
```

配置回调

初始化LoRaMac

初始化循环发送Timer

配置为公共网络

classA.c

```
402
403 static void lwan_dev_params_update( void )
404 {
405     MibRequestConfirm_t mibReq;
406     uint16_t channelsMaskTemp[6];
407     channelsMaskTemp[0] = 0x00FF;
408     channelsMaskTemp[1] = 0x0000;
409     channelsMaskTemp[2] = 0x0000;
410     channelsMaskTemp[3] = 0x0000;
411     channelsMaskTemp[4] = 0x0000;
412     channelsMaskTemp[5] = 0x0000;
413
414     mibReq.Type = MIB_CHANNELS_DEFAULT_MASK;
415     mibReq.Param.ChannelsDefaultMask = channelsMaskTemp;
416     LoRaMacMibSetRequestConfirm(&mibReq);
417     mibReq.Type = MIB_CHANNELS_MASK;
418     mibReq.Param.ChannelsMask = channelsMaskTemp;
419     LoRaMacMibSetRequestConfirm(&mibReq);
420 }
421
```

信道掩码配置，CN470频段共支持96个上行信道，用一个长度为6的16位数组表示，mask每1位对应1个信道，1开启，0关闭

ASR6601 ClassA工程软件流程

classA.c

```
468             case DEVICE_STATE_JOIN:
469             {
470 #if( OVER_THE_AIR_ACTIVATION != 0 )
471                 MlmeReq_t mlmeReq;
472
473                 // Initialize LoRaMac device unique ID
474                 //BoardGetUniqueId( DevEui );
475
476                 mlmeReq.Type = MLME_JOIN;
477
478                 mlmeReq.Req.Join.DevEui = DevEui;
479                 mlmeReq.Req.Join.AppEui = AppEui;
480                 mlmeReq.Req.Join.AppKey = AppKey;
481                 mlmeReq.Req.Join.NbTrials = 8;
482
483                 if( LoRaMacMlmeRequest( &mlmeReq ) == LORAMAC_STATUS_OK )
484                 {
485                     DeviceState = DEVICE_STATE_SLEEP;
486                 }
487                 else
488                 {
489                     DeviceState = DEVICE_STATE_CYCLE;
490                 }
491 #else
492
493                 mibReq.Type = MIB_NET_ID;
494                 mibReq.Param.NetID = LORAWAN_NETWORK_ID;
495                 LoRaMacMibSetRequestConfirm( &mibReq );
496
497                 mibReq.Type = MIB_DEV_ADDR;
498                 mibReq.Param.DevAddr = DevAddr;
499                 LoRaMacMibSetRequestConfirm( &mibReq );
500
501                 mibReq.Type = MIB_NWK_SKEY;
502                 mibReq.Param.NwkSKey = NwkSKey;
503                 LoRaMacMibSetRequestConfirm( &mibReq );
504
505                 mibReq.Type = MIB_APP_SKEY;
506                 mibReq.Param.AppSKey = AppSKey;
507                 LoRaMacMibSetRequestConfirm( &mibReq );
508
509                 mibReq.Type = MIB_NETWORK_JOINED;
510                 mibReq.Param.IsNetworkJoined = true;
511                 LoRaMacMibSetRequestConfirm( &mibReq );
512
513                 DeviceState = DEVICE_STATE_SEND;
```

OTAA入网分支

配置入网三元组

请求成功则进入休眠，等待回复

否则循环发送

ABP入网分支

配置网络ID

配置设备地址

配置网络安全秘钥

配置应用安全秘钥

配置为已入网

进入发送

classA.c

```
516             }
517             case DEVICE_STATE_SEND:
518             {
519                 if( NextTx == true )
520                 {
521                     PrepareTxFrame( AppPort );
522
523                     NextTx = SendFrame( );
524                 }
525
526                 // Schedule next packet transmission
527                 TxDutyCycleTime = APP_TX_DUTYCYCLE + randr( 0, APP_TX_DUTYCYCLE_RND );
528                 DeviceState = DEVICE_STATE_CYCLE;
529                 break;
530             }
531             case DEVICE_STATE_CYCLE:
532             {
533                 DeviceState = DEVICE_STATE_SLEEP;
534
535                 // Schedule next packet transmission
536                 TimerSetValue( &TxNextPacketTimer, TxDutyCycleTime );
537                 TimerStart( &TxNextPacketTimer );
538                 break;
539             }
540             case DEVICE_STATE_SLEEP:
541             {
542                 // Wake up through events
543                 TimerLowPowerHandler( );
544
545                 // Process Radio IRQ
546                 Radio.IrqProcess( );
547                 break;
548             }
549             default:
550             {
551                 DeviceState = DEVICE_STATE_INIT;
552                 break;
553             }
554         }
555     }
556 }
557
```

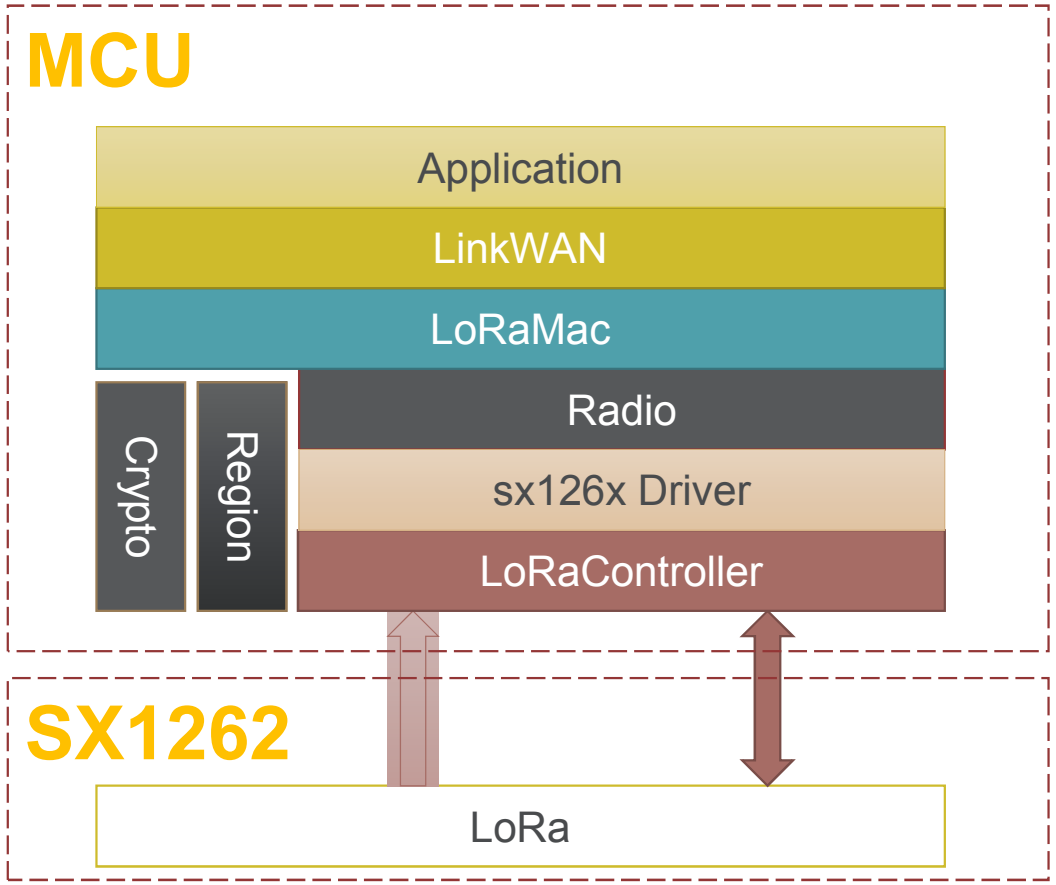
填充发送数据

发送

启动下一次发送Timer

休眠

ASR6601 LoRa协议栈框架



- MCU与LoRa通过LoRaController连接
- sx126x Driver实现了sx1262寄存器读写和指令操作封装，并实现了一些基本的驱动接口函数
- Radio是协议栈的中间层，对LoRa芯片的驱动接口进行了封装，屏蔽了LoRa射频芯片差异
- LoRaMac处理MAC层事务
- Region根据不同的频段提供相应的配置参数，以及相应配置接口
- Crypto的主要作用是加解密及MIC计算

ASR6601低功耗处理

- 低功耗模式，MCU配置为Stop3模式，LoRa配置为Sleep模式
- MCU的大部分模块在Stop3模式自动关闭，唤醒后自动恢复，但是与外部引脚相连接的模块进入Stop3前需要将引脚电平与外部配平以防止漏电
- LCDCTRL、RTC、LPTIM、LPUART和IWDG等模块在Stop3模式默认开启
- 由于RTC时钟源为32.768K，而MCU的时钟源为RC048M，RTC在进入Stop3前需要同步

```
classA.c
540             case DEVICE_STATE_SLEEP:
541             {
542                 // Wake up through events
543                 TimerLowPowerHandler( );
```

timer.c D:\Working\LoRa\ASR6601\src\sdk\ASR6601_rel_v1.5.0\lora\system – 2 definitions

```
407 void TimerLowPowerHandler( void )
408 {
409     //if( ( TimerListHead != NULL ) && ( TimerListHead->IsRunning == true ) )
410     {
411         if( HasLoopedThroughMain < 5 )
412         {
413             HasLoopedThroughMain++;
414         }
415         else
416         {
417             HasLoopedThroughMain = 0;
418             RtcEnterLowPowerStopMode( );
```

rtc-board.c D:\Working\LoRa\ASR6601\src\sdk\ASR6601_rel_v1.5.0\lora\driver – 2 definitions

```
234
235 void RtcEnterLowPowerStopMode( void )
236 {
237     if(Radio.GetStatus() != RF_IDLE)
238         return;
239
240     rtc_check_syn();
241     pwr_deepsleep_wfi(PWR_LP_MODE_STOP3);
242 }
243
```

确保RF已经休眠

休眠前同步RTC

MCU进入Stop3（wfi）模式

ASR6601开发环境搭建与编译下载

- **安装ARM GNU编译器**，下载链接：https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads，版本号建议：gcc-arm-none-eabi-9-2020-q2-update-win32.zip
- **安装MSYS2**，下载链接：https://www.msys2.org/
- **安装工具链**，指令：pacman -S git vim make python python-pip，安装pyserial，指令：pip install pyserial
- **配置环境变量**，进入SDK目录，指令：source build/envsetup.sh
- **进入烧录模式**，将Demo板通过USB线连接到电脑，按下SW3，然后按下SW2，松开SW3
- **编译下载**，进入工程目录projects/ASR6601CB-EVAL/examples/uart/uart_printf，指令：make flash SERIAL_PORT=/dev/ttyS3

ASR6601软件相关资料说明

资料	描述
ASR6601_程序开发快速入门指南	开发环境搭建、快速开始
ASR6601_烧录工具使用说明	烧录工具使用说明
ASR6601_参考手册	技术参考手册
ASR6601_User_Manual	API参考手册
ASR6601_AT命令说明	AT指令手册
ASR6601_参考板测试指南	Demo板测试指南
ASR6601_OTA升级说明	OTA Demo说明

第四部分 FreeRTOS移植

ASR6601 FreeRTOS移植思路

- Cortex-M系列对OS的支持主要是通过Systick、PSP、SVC、PendSV来实现，基于Context实现任务调度。其中Systick主要用于OS计时，PSP主要用于App栈以独立于OS栈，SVC主要用于App访问OS资源，PendSV主要用于产生OS请求。
- 本例将在ASR6601上实现FreeRTOS的三个简单的功能：
  - 1.)创建一个Task，在该Task中利用Delay闪烁LED。
  - 2.)创建一个Timer，在Timer超时回调里循环打印信息。
  - 3.)创建一个Task，利用Semaphore响应GPIO中断。
- 移植FreeRTOS的主要工作：
  - a).移植Context代码。
  - b).移植Kernel代码。
  - c).实现最小功能Demo。

ASR6601 FreeRTOS工程创建

- 以FreeRTOSv10.1.1为例，从官网下载FreeRTOS代码，并解压到到ASR6601 SDK的目录下
- 在FreeRTOSv10.1.1\FreeRTOS\Demo目录下创建ARM_STAR_ASR6601_GCC目录，拷贝uart_printf工程文件至该目录下
- 从FreeRTOS的Demo工程拷贝FreeRTOSConfig.h到inc目录，拷贝ParTest.c文件到src目录。

ASR6601 FreeRTOS工程Makefile修改

修改Makefile，添加源文件和头文件包含路径

ASR6601 FreeRTOS工程配置

修改FreeRTOSConfig.h文件，配置主要参数

- 配置configCPU_CLOCK_HZ为24000000，此处为CPU系统时钟
- 配置configTOTAL_HEAP_SIZE为1024*1000，此处为堆空间总量
- 配置configPRIO_BITS为3，此处为中断优先级位数
- 配置configLIBRARY_LOWEST_INTERRUPT_PRIORITY为7，此处为最低中断优先级
- 配置configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY为2，此处为系统调用最大中断优先级

ASR6601 FreeRTOS工程main文件头文件

main.c

```
80  /* standard includes. */
81  #include <stdio.h>
82  #include <stdarg.h>
83
84  /* Kernel includes. */
85  #include "FreeRTOS.h"
86  #include "task.h"
87  #include "timers.h"
88  #include "semphr.h"
89
90  /* Demo application includes. */
91  #include "partest.h"
92  #include "flash.h"
93  #include "flop.h"
94  #include "integer.h"
95  #include "PollQ.h"
96  #include "semtest.h"
97  #include "dynamic.h"
98  #include "BlockQ.h"
99  #include "blocktim.h"
100 #include "countsem.h"
101 #include "GenQTest.h"
102 #include "recmutex.h"
103 #include "death.h"
104
105 /* Hardware and starter kit includes. */
106 #include "tremo_uart.h"
107 #include "tremo_gpio.h"
108 #include "tremo_rcc.h"
109
```

添加FreeRTOS头文件包含

ASR6601 FreeRTOS初始化

```
/*-----------------------------------------------------------*/

static void prvSetupHardware( void )
{
    rcc_enable_peripheral_clk(RCC_PERIPHERAL_AFEC, true);
    // enable the clk
    rcc_enable_oscillator(RCC_OSC_XO32K, true);

    rcc_enable_peripheral_clk(RCC_PERIPHERAL_UART0, true);
    rcc_enable_peripheral_clk(RCC_PERIPHERAL_GPIOA, true);
    rcc_enable_peripheral_clk(RCC_PERIPHERAL_GPIOB, true);

    // uart0
    gpio_set_iomux(GPIOB, GPIO_PIN_0, 1);
    gpio_set_iomux(GPIOB, GPIO_PIN_1, 1);

    /* uart config struct init */
    uart_config_t uart_config;
    uart_config_init(&uart_config);

    uart_config.baudrate = UART_BAUDRATE_115200;
    uart_init(CONFIG_DEBUG_UART, &uart_config);
    uart_cmd(CONFIG_DEBUG_UART, ENABLE);

    gpio_init(GPIOA, GPIO_PIN_11, GPIO_MODE_INPUT_PULL_UP);
    gpio_config_interrupt(GPIOA, GPIO_PIN_11, GPIO_INTR_FALLING_EDGE);
    /* NVIC config */
    NVIC_EnableIRQ(GPIO_IRQn);
    NVIC_SetPriority(GPIO_IRQn, 3);

    vParTestInitialise();
}
```

添加preSetupHardware函数，开启UART0、GPIOA和GPIOB的时钟，初始化UART0，初始化GPIO11为下降沿中断触发，使能GPIO中断

ASR6601 FreeRTOS工程中断处理

main.c

```
266  void GPIO_IRQHandler(void)
267  {
268      long lHigherPriorityTaskWoken = pdFALSE;
269
270      /* Only line 6 is enabled, so there is no need to test which line generated
271      the interrupt. */
272      if (gpio_get_interrupt_status(GPIOA, GPIO_PIN_11) == SET) {
273          gpio_clear_interrupt(GPIOA, GPIO_PIN_11);
274          /* This interrupt does nothing more than demonstrate how to synchronise a
275          task with an interrupt.  First the handler releases a semaphore.
276          lHigherPriorityTaskWoken has been initialised to zero. */
277          xSemaphoreGiveFromISR( xTestSemaphore, &lHigherPriorityTaskWoken );
278      }
279
280      /* If there was a task that was blocked on the semaphore, and giving the
281      semaphore caused the task to unblock, and the unblocked task has a priority
282      higher than the currently executing task (the task that this interrupt
283      interrupted), then lHigherPriorityTaskWoken will have been set to pdTRUE.
284      Passing pdTRUE into the following macro call will cause this interrupt to
285      return directly to the unblocked, higher priority, task. */
286      portEND_SWITCHING_ISR( lHigherPriorityTaskWoken );
287  }
```

- 添加GPIO中断处理，GPIO11中断则赋信号量
- 在tremo_it.c文件中移除SVC_Handler、PendSV_Handler和SysTick_Handler三个中断处理

ASR6601 FreeRTOS工程Timer回调与按键处理Task

main.c

```
197  static void prvTestTimerCallback( TimerHandle_t xTimer )
198  {
199      printf("%s fired!!!\r\n", *(char **)xTimer);
200  }
201
202  /*-----------------------------------------------------------*/
203
204  static void prvButtonTestTask( void *pvParameters )
205  {
206      configASSERT( xTestSemaphore );
207
208      /* This is the task used as an example of how to synchronise a task with
209      an interrupt.  Each time the button interrupt gives the semaphore, this task
210      will unblock, increment its execution counter, then return to block
211      again. */
212
213      /* Take the semaphore before started to ensure it is in the correct
214      state. */
215      xSemaphoreTake( xTestSemaphore, mainDONT_BLOCK );
216
217      printf("Enter Button Test Task\r\n");
218      for( ;; )
219      {
220          xSemaphoreTake( xTestSemaphore, portMAX_DELAY );
221          ulButtonPressCounts++;
222          printf("Button Click %d\r\n", ulButtonPressCounts);
223      }
224  }
225  /*-----------------------------------------------------------*/
```

- 在测试Timer超时回调里打印log
- 每按下一次SW1，打印一次log

ASR6601 FreeRTOS工程Hook添加

main.c

```
282  /*-----------------------------------------------------------*/
283  void vApplicationTickHook( void )
284  {
285
286  }
287
288  /*-----------------------------------------------------------*/
289
290  void vApplicationMallocFailedHook( void )
291  {
292      /* vApplicationMallocFailedHook() will only be called if
293      configUSE_MALLOC_FAILED_HOOK is set to 1 in FreeRTOSConfig.h.  It is a hook
294      function that will get called if a call to pvPortMalloc() fails.
295      pvPortMalloc() is called internally by the kernel whenever a task, queue,
296      timer or semaphore is created.  It is also called by various parts of the
297      demo application.  If heap_1.c or heap_2.c are used, then the size of the
298      heap available to pvPortMalloc() is defined by configTOTAL_HEAP_SIZE in
299      FreeRTOSConfig.h, and the xPortGetFreeHeapSize() API function can be used
300      to query the size of free heap space that remains (although it does not
301      provide information on how the remaining heap might be fragmented). */
302      taskDISABLE_INTERRUPTS();
303      for( ;; );
304  }
305  /*-----------------------------------------------------------*/
306
307  void vApplicationIdleHook( void )
308  {···
318  }
319  /*-----------------------------------------------------------*/
320
321  void vApplicationStackOverflowHook( TaskHandle_t pxTask, char *pcTaskName )
322  {
323      ( void ) pcTaskName;
324      ( void ) pxTask;
325
326      /* Run time stack overflow checking is performed if
327      configCHECK_FOR_STACK_OVERFLOW is defined to 1 or 2.  This hook
328      function is called if a stack overflow is detected. */
329      taskDISABLE_INTERRUPTS();
330      for( ;; );
331  }
```

- 添加vApplicationTickHook,函数体为空
- 添加vApplicationMallocFailedHook，关闭中断，并进入无限空循环
- 添加vApplicationIdleHook，函数体为空
- 添加vApplicationStackOverflowHook ，关闭中断，并进入无限空循环

ASR6601 FreeRTOS工程LED闪烁处理

ParTest.c

```
/* Standard demo include. */
#include "partest.h"

/* Starter kit includes. */
#include "tremo_gpio.h"

/*-----------------------------------------------------------*/

void vParTestInitialise( void )
{
    /* Initialise all four LEDs that are built onto the starter kit. */
    gpio_init(GPIOA, GPIO_PIN_8, GPIO_MODE_OUTPUT_PP_LOW);
}
/*-----------------------------------------------------------*/

void vParTestSetLED( unsigned long ulLED, signed portBASE_TYPE xValue )
{
    if( xValue == pdTRUE )
    {
        gpio_write(GPIOA, GPIO_PIN_8, GPIO_LEVEL_HIGH);
    }
    else
    {
        gpio_write(GPIOA, GPIO_PIN_8, GPIO_LEVEL_LOW);
    }
}
/*-----------------------------------------------------------*/

void vParTestToggleLED( unsigned long ulLED )
{
    taskENTER_CRITICAL();
    {
        gpio_toggle(GPIOA, GPIO_PIN_8);
    }
    taskEXIT_CRITICAL();
}
/*-----------------------------------------------------------*/
```

包含tremo_gpio.h头文件

修改GPIO初始化代码

修改GPIO写代码

修改GPIO翻转代码

ASR6601 Demo板上的LED为GPIO08，修改ParTest文件，替换GPIO相关操作函数为ASR6601 API

ASR6601 FreeRTOS工程main函数

main.c

```c
int main(void)
{
    TimerHandle_t xTestTimer = NULL;

    /* Configure the hardware ready to run the test. */
    prvSetupHardware();

    /* Start standard demo/test application flash tasks.  See the comments at
    the top of this file.  The LED flash tasks are always created.  See the comments at the top of this file for
    more information. */
    vStartLEDFlashTasks( mainFLASH_TASK_PRIORITY );

    /* Create the semaphore that is used to demonstrate a task being
    synchronised with an interrupt. */
    vSemaphoreCreateBinary( xTestSemaphore );

    /* Create the task that is unblocked by the demonstration interrupt. */
    xTaskCreate( prvButtonTestTask, "BtnTest", configMINIMAL_STACK_SIZE, ( void * ) NULL, tskIDLE_PRIORITY, NULL );

    /* Create the software timer that performs the 'check' functionality,
    as described at the top of this file. */
    xTestTimer = xTimerCreate( "TestTimer",                /* A text name, purely to help debugging. */
                               ( mainCHECK_TEST_PERIOD_MS ),   /* The timer period, in this case 3000ms (3s). */
                               pdTRUE,                         /* This is an auto-reload timer, so xAutoReload is set to pdTRUE. */
                               ( void * ) 0,                   /* The ID is not used, so can be set to anything. */
                               prvTestTimerCallback            /* The callback function that inspects the status of all the other tasks. */
                               );

    if( xTestTimer != NULL )
    {
        xTimerStart( xTestTimer, mainDONT_BLOCK );
    }

    /* Start the scheduler. */
    vTaskStartScheduler();

    /* If all is well, the scheduler will now be running, and the following line
    will never be reached.  If the following line does execute, then there was
    insufficient FreeRTOS heap memory available for the idle and/or timer tasks
    to be created.  See the memory management section on the FreeRTOS web site
    for more details. */
    for( ;; );
}
```

初始化硬件

启动LED闪烁任务

创建信号量

启动按键处理任务

创建Timer

启动Timer

启动任务调度器

- main函数中初始化硬件，并启动LED闪烁Task、按键处理Task和Timer
- 编译、下载、测试，观察Demo板LED闪烁，查看串口Timer log，按下SW1查看log

```
[10:28:19.196]收←◆Enter Button Test Task
[10:28:21.720]收←◆Button Click 1
[10:28:22.197]收←◆TestTimer fired!!!
[10:28:22.238]收←◆Button Click 2
[10:28:23.060]收←◆Button Click 3
[10:28:23.446]收←◆Button Click 4
[10:28:25.198]收←◆TestTimer fired!!!
```

第五部分 LoRaWAN入网调试

LoRaWAN入网必要条件

- 确保节点频段与网关频段一致
- 确保节点信道掩码与网关信道一致
- 确保节点三元组正确
- 确保节点ClassC配置与NS一致

LoRaWAN入网调试

入网失败，最好的debug方法就是查看网关log，a.)收到JoinRequest，b.)下发JoinAccept

- 没收到JoinRequest，频段不一致，或者信道掩码不一致，或者节点发射功率太低，没接天线等
- 收到JoinRequest ，没下发JoinAccept，节点三元组配置错误
- 收到JoinRequest ，也下发了JoinAccept，极可能是节点接收时间窗口与网关没有对齐，作为debug手段，可以通过增大MIB_SYSTEM_MAX_RX_ERROR的值来重新入网来验证，注意：这么做会增加功耗

LoRaMac.c

```
2991    // Init parameters which are not set in function ResetMacParameters
2992    LoRaMacParams.RepeaterSupport = false;
2993    LoRaMacParamsDefaults.ChannelsNbRep = 1;
2994    LoRaMacParamsDefaults.SystemMaxRxError = 10;
2995    LoRaMacParamsDefaults.MinRxSymbols = 6;
2996
2997    LoRaMacParams.SystemMaxRxError = LoRaMacParamsDefaults.SystemMaxRxError;
2998    LoRaMacParams.MinRxSymbols = LoRaMacParamsDefaults.MinRxSymbols;
2999    LoRaMacParams.MaxRxWindow = LoRaMacParamsDefaults.MaxRxWindow;
3000    LoRaMacParams.ReceiveDelay1 = LoRaMacParamsDefaults.ReceiveDelay1;
3001    LoRaMacParams.ReceiveDelay2 = LoRaMacParamsDefaults.ReceiveDelay2;
3002    LoRaMacParams.JoinAcceptDelay1 = LoRaMacParamsDefaults.JoinAcceptDelay1;
3003    LoRaMacParams.JoinAcceptDelay2 = LoRaMacParamsDefaults.JoinAcceptDelay2;
3004    LoRaMacParams.ChannelsNbRep = LoRaMacParamsDefaults.ChannelsNbRep;
3005
3006    ResetMacParameters( );
3007
3008    // Initialize timers
3009    TimerInit( &MacStateCheckTimer, OnMacStateCheckTimerEvent );
3010    TimerSetValue( &MacStateCheckTimer, MAC_STATE_CHECK_TIMEOUT );
3011
3012    TimerInit( &TxDelayedTimer, OnTxDelayedTimerEvent );
3013    TimerInit( &RxWindowTimer1, OnRxWindow1TimerEvent );
3014    TimerInit( &RxWindowTimer2, OnRxWindow2TimerEvent );
3015    TimerInit( &AckTimeoutTimer, OnAckTimeoutTimerEvent );
3016 #ifdef CONFIG_LORA_CAD
3017    TimerInit( &TxImmediateTimer, OnTxImmediateTimerEvent );
3018 #endif
3019
```

1.加大拓宽接收窗口
2.减小缩窄接收窗口

LoRaWAN入网抓包

LoRaWAN节点与网关接收时间窗口不对齐，可能是节点原因，也可能是网关原因：

- 节点时间窗口可以通过一个Timer拉GPIO测量脉冲宽度确定

网关时间窗口可以通过抓包确定，抓包步骤：

- 确定入网上行信道与下行信道，CN470对应关系为RX1_CH=UpLink_CH%48
- 确定上行与下行的射频配置。二者主要差别是CRC和IQInverted两个参数的配置。上行CRC开启，下行CRC关闭。上行IQInverted关闭，下行IQInverted开启。详细参考各频段的Region文件
- 准备两个节点，配置为公共网络，一个接收配置为LoRaWAN上行射频配置，一个接收配置为LoRaWAN下行射频配置，持续接收抓包
- 待收到数据，把收到的下行数据时间减去上行数据时间，差值即为网关的下行时间窗口开启Delay

第六部分 CAD调试与空中唤醒

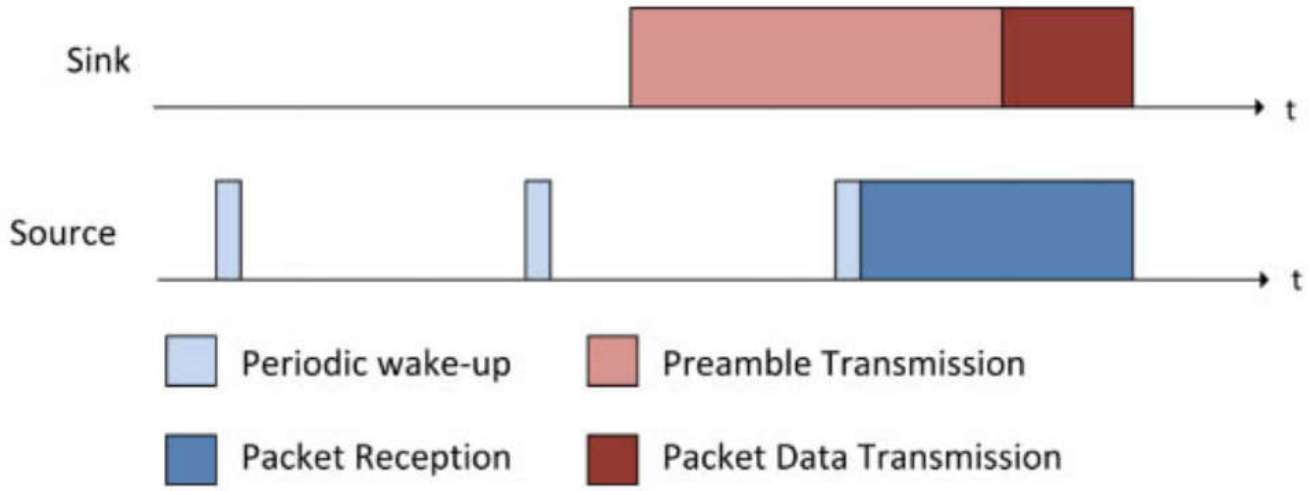
CAD调试

由于LoRa可以在噪声之下传输信号，因此通过RSSI来判断信道是否处于空闲状态不可取，信道活动侦测CAD(Channel Activity Detect) 可以很好解决这个问题，SX127x系列仅支持前导码检测，从SX126x开始可以支持前导码和数据部分检测，同时也支持检测符号长度和灵敏度的配置：

- 检测符号长度cadSymbolNum，支持1、2、4、8和16个符号，越长则过滤功能越强，相应的也会增长检测时间，增加单次检测功耗
- 灵敏度cadDetPeak和cadDetMin，cadDetPeak越大则灵敏度越低，相应的误检测几率也会降低，cadDetMin一般配置为10，不需要更改

cadSymbolNum和cadDetPeak的配置，需要根据实际应用情况在功耗和灵敏度之间做平衡，可以参考《AN1200.48》

空中唤醒



空中唤醒利用了长前导码，接收端以低于前导码时间长度的间隔周期性的唤醒做CAD检测，检测到有效信号则进入接收，否则继续休眠，在实际使用中还有两点需要注意：

- 接收端的计时时钟精度决定了实际的检测间隔
- 接收时的前导码长度配置应大于等于发射端的前导码长度

谢谢!

THANK!

ASR物联