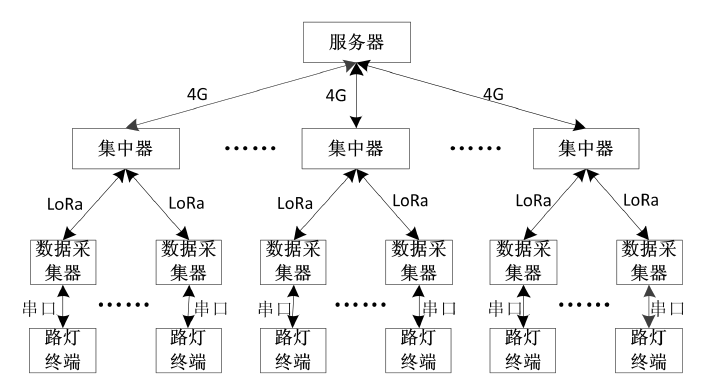
软件详细设计

1. 概述

本文档以路灯控制系统的通信、功能的实现为主进行分析。首先介绍系统软件的整体设计，对系统的 LoRa 组网以及通信协议进行详细说明，着重分析路灯终端以及LoRa模块的软件设计。由于整个系统的代码量非常大，因此本文档只阐述本人负责编写的部分。

本系统的主要软件部分除服务器外，也分为集中器模块、数据采集模块、终端模块 3 个部分。服务器主要是通过运营商的蜂窝网络与集中器建立连接并进行数据和信息的传递，因此两者间的通信主要是通过 4G 模块实现的，集中器和数据采集模块的通信是通过 LoRa 模块实现的，数据采集模块到终端的通信是由串口通信实现，其中集中器模块和数据采集模块上的 GPS 模块可以对集中器和数据采集器的位置进行准确的定位。

LoRa 模块间采用的是星形网络架构，相比于常用的网络架构来说，星形网络具有更低的延迟而且网络结构简单，整个系统的组网形式如下图 所示。系统的集中器能并行接收多个数据采集器发来的路灯终端参数，而所有集中器采集的路灯信息又会全部上传至服务器进行处理展示给管理者，从而有效的扩展了系统的总容量。



1. 平台端

管理者将在web平台上看到所有路灯终端的状态信息，并且可通过网页上的相关设置将控制指令发送给相应的路灯终端，以此实现对路灯终端参数信息的读取和控制。由于这部分接入平台一般会采用华为、阿里提供的平台，或者由网络运营商提供，且于本课程嵌入式系统设计无关，因此就不在此阐述了。实验测试中的平台界面显示如下图所示，该平台是由公司提供的物联网监控平台：



平台下发的指令由网关转换成相应的Modbus或RS485控制码下达到设备终端以实现控制：

**单灯查状态：**

发送： 10 09 00 00 12 34 FF FC DD

帧头 长度 设备地址 命令 CRC校验

接收：11 1D 00 00 12 34 04 10 64 05 04

帧头 长度 设备地址 调光端口 功能码 调光值 开关端口 功能码

00 00 20 0C

开关状态 采集端口 功能码 采集数据长度

57 2B 03 99 08 17 00 11 13 88 37 10 08 5A

电压 电流 有功 无功 频率 温度 CRC校验

**广播查状态**：

发送：10 09 FF FF FF FF FF 1F D8

接收：11 1D 00 00 12 34 04 10 64 05 04 00 00 20 0C 57 6A 03 9D 08 28 00 11 13 88 39 66 21 22

**单灯查询电能**：

发送：10 09 00 00 12 34 FD 7D 1C

帧头 长度 设备地址 命令 CRC校验

接收： 11 17 00 00 12 34 00 21 0C 00 00 00 06

帧头 长度 设备地址 采集端口 功能码 采集数据长度 有功电能

00 00 00 00 00 00 00 06 4F 42

无功电能 视在电能 CRC校验

**广播查询电能**：

发送：10 09 FF FF FF FF FD 9E 19

接收：11 17 00 00 12 34 00 21 0C 00 00 00 0E 00 00 00 00 00 00 00 0E 29 44

**单灯开**：

发送：15 0A 00 00 12 34 05 00 AD 1E

帧头 长度 设备地址 开关端口 开关状态 CRC校验

接收：11 1D 00 00 12 34 04 10 64 05 04

帧头 长度 设备地址 调光端口 功能码 调光值 开关端口 功能码

00 00 20 0C

关状态 采集端口 功能码 采集数据长度

57 34 00 00 00 00 00 00 13 87 37 D7 70 72

电压 电流 有功 无功 频率 温度 CRC校验

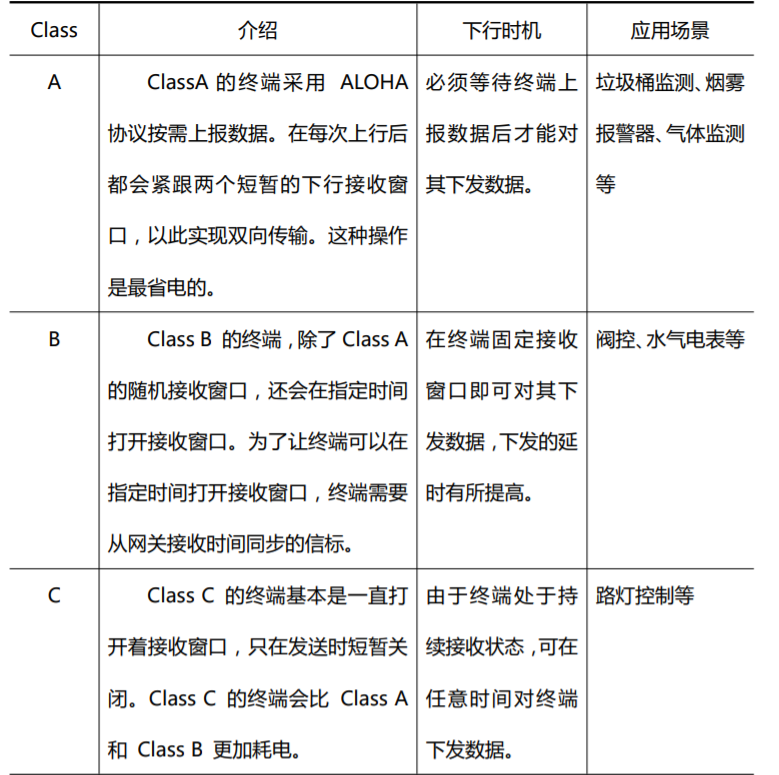
**单灯关**：

发送：15 0A 00 00 12 34 05 01 6C DE

接收：11 1D 00 00 12 34 04 10 64 05 04 01 00 20 0C 57 A8 00 00 00 00 00 00 13 88 38 5C 06 24

1. LoRa 网络组网部分代码

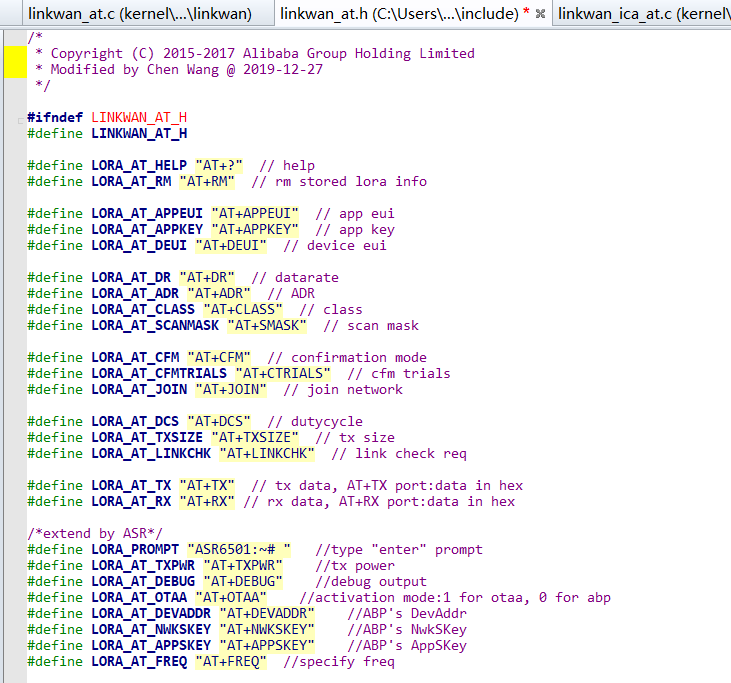
LoRaWAN是LoRa联盟为LoRa远距离通信网络设计的一套通讯协议和系统架构。 在一个 LoRaWAN 的网络架构中包含了终端、基站、网络服务器（NS）、应用服务器这 四个部分。基站和终端之间采用了星形网络拓扑结构，由于 LoRa 具有的长距离特性， 终端之间可以使用单跳传输。基站则对网络服务器和终端之间的 LoRaWAN 协议数据仅做转发处理，将 LoRaWAN 数据分别承载在了 LoRa 射频传输和 TCP/IP 上。 在 LoRaWAN 协议中，终端类型分为 ClassA、ClassB 和 ClassC 三类。具体区别如下表所示，本系统需要将LoRa终端设置为C类。所有终端设备在正式加入LoRaWAN网络之前必须先进行初始化并激活。有两种激活方式：无线激活（Over-The-Air Activation (OTAA)），设备部署和重置时使用；手动激活（Activation By Personalization (ABP)），此时初始化和激活一步完成。以下信息在激活成功后回存储在终端设备：设备地址（DevAddr）、应用ID（AppEUI）、网络会话密钥（NwkSKey）和应用会话密钥（AppKey）。



具体的入网流程比较复杂，详细过程和指令请查询协议资料文件夹中的《LoRaWAN 规范》、《AT指令说明》文件。由于整个系统的代码量非常大，因此本文档只阐述本人负责编写的部分。

以下是LoRa入网AT指令，这部分的AT指令都需要提前在头文件中根据入网需求进行配置添加，如下图所示：

1. AT指令配置



1. AT指令的处理代码：

void process\_linkwan\_at(void)

{

bool ret = false;

int value;

MibRequestConfirm\_t mibReq;

LoRaMacStatus\_t status;

uint8\_t length;

uint8\_t buf[16];

if (atcmd\_index == 0 && atcmd[atcmd\_index] == '\0') {

PRINTF\_RAW("\r\n%s", LORA\_PROMPT);

atcmd\_index = 0;

memset(atcmd, 0xff, ATCMD\_SIZE);

return;

}

if (atcmd\_index == 0 || atcmd[atcmd\_index] != '\0') {

return;

}

if (strncmp(atcmd, LORA\_AT\_HELP, strlen(LORA\_AT\_HELP)) == 0) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s\r\n", LORA\_AT\_HELP);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_RM);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_APPEUI);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_APPKEY);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_DEUI);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_DR);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_ADR);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_CLASS);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_SCANMASK);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_CFM);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_CFMTRIALS);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_JOIN);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_TX);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_RX);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_DCS);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_TXSIZE);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_LINKCHK);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_TXPWR);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_DEBUG);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_OTAA);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_DEVADDR);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_NWKSKEY);

linkwan\_serial\_output(atcmd, strlen(atcmd));

snprintf(atcmd, ATCMD\_SIZE, "%s\r\n", LORA\_AT\_APPSKEY);

linkwan\_serial\_output(atcmd, strlen(atcmd));

linkwan\_serial\_output(LORA\_PROMPT, strlen(LORA\_PROMPT));

atcmd\_index = 0;

memset(atcmd, 0xff, ATCMD\_SIZE);

return;

} else if (strncmp(atcmd, LORA\_AT\_APPEUI, strlen(LORA\_AT\_APPEUI)) == 0) {

if (atcmd\_index == strlen(LORA\_AT\_APPEUI)) {

uint8\_t \*eui = get\_lora\_app\_eui();

ret = true;

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %02x%02x%02x%02x%02x%02x%02x%02x\r\n", \

LORA\_AT\_APPEUI, eui[0], eui[1], eui[2], eui[3], eui[4], eui[5], eui[6], eui[7]);

} else if (atcmd\_index == (strlen(LORA\_AT\_APPEUI) + 16)) {

length = hex2bin(&atcmd[strlen(LORA\_AT\_APPEUI)], buf, 8);

if (length == 8) {

ret = set\_lora\_app\_eui(buf);

if (ret == true) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %02x%02x%02x%02x%02x%02x%02x%02x\r\n", \

LORA\_AT\_APPEUI, buf[0], buf[1], buf[2], buf[3], buf[4], buf[5], buf[6], buf[7]);

}

}

}

if (ret == false) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s ERROR\r\n", LORA\_AT\_APPEUI);

}

} else if (strncmp(atcmd, LORA\_AT\_APPKEY, strlen(LORA\_AT\_APPKEY)) == 0) {

if (atcmd\_index == strlen(LORA\_AT\_APPKEY)) {

uint8\_t \*key = get\_lora\_app\_key();

ret = true;

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x\r\n", \

LORA\_AT\_APPKEY, key[0], key[1], key[2], key[3], key[4], key[5], key[6], key[7],

key[8], key[9], key[10], key[11], key[12], key[13], key[14], key[15]);

} else if (atcmd\_index == (strlen(LORA\_AT\_APPKEY) + 32)) {

length = hex2bin(&atcmd[strlen(LORA\_AT\_APPKEY)], buf, 16);

if (length == 16) {

ret = set\_lora\_app\_key(buf);

if (ret == true) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x\r\n", \

LORA\_AT\_APPKEY, buf[0], buf[1], buf[2], buf[3], buf[4], buf[5], buf[6], buf[7],

buf[8], buf[9], buf[10], buf[11], buf[12], buf[13], buf[14], buf[15]);

}

}

}

if (ret == false) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s ERROR\r\n", LORA\_AT\_APPKEY);

}

} else if (strncmp(atcmd, LORA\_AT\_DEUI, strlen(LORA\_AT\_DEUI)) == 0) {

if (atcmd\_index == strlen(LORA\_AT\_DEUI)) {

uint8\_t \*eui = get\_lora\_dev\_eui();

ret = true;

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %02x%02x%02x%02x%02x%02x%02x%02x\r\n", \

LORA\_AT\_DEUI, eui[0], eui[1], eui[2], eui[3], eui[4], eui[5], eui[6], eui[7]);

} else if (atcmd\_index == (strlen(LORA\_AT\_DEUI) + 16)) {

length = hex2bin(&atcmd[strlen(LORA\_AT\_DEUI)], buf, 8);

if (length == 8) {

ret = set\_lora\_dev\_eui(buf);

if (ret == true) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %02x%02x%02x%02x%02x%02x%02x%02x\r\n", \

LORA\_AT\_DEUI, buf[0], buf[1], buf[2], buf[3], buf[4], buf[5], buf[6], buf[7]);

}

}

}

if (ret == false) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s ERROR\r\n", LORA\_AT\_DEUI);

}

} else if (strncmp(atcmd, LORA\_AT\_RM, strlen(LORA\_AT\_RM)) == 0) {

aos\_kv\_del("lora");

aos\_kv\_del("lora\_dev");

ret = true;

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s OK\r\n", LORA\_AT\_RM);

} else if (strncmp(atcmd, LORA\_AT\_DR, strlen(LORA\_AT\_DR)) == 0) {

int8\_t datarate;

if (atcmd\_index == strlen(LORA\_AT\_DR)) {

ret = true;

datarate = get\_lora\_tx\_datarate();

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %d\r\n", LORA\_AT\_DR, datarate);

} else if (atcmd\_index == (strlen(LORA\_AT\_DR) + 1)) {

datarate = strtol(atcmd + strlen(LORA\_AT\_DR), NULL, 0);

ret = set\_lora\_tx\_datarate(datarate);

if (ret == true) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %d\r\n", LORA\_AT\_DR, datarate);

}

}

if (ret == false) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s ERROR\r\n", LORA\_AT\_DR);

}

} else if (strncmp(atcmd, LORA\_AT\_ADR, strlen(LORA\_AT\_ADR)) == 0) {

int adr;

if (atcmd\_index == strlen(LORA\_AT\_ADR)) {

ret = true;

adr = get\_lora\_adr();

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %d\r\n", LORA\_AT\_ADR, adr);

} else if (atcmd\_index == (strlen(LORA\_AT\_ADR) + 1)) {

adr = strtol(atcmd + strlen(LORA\_AT\_ADR), NULL, 0);

ret = set\_lora\_adr(adr);

if (ret == true) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %d\r\n", LORA\_AT\_ADR, adr);

}

}

if (ret == false) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s ERROR\r\n", LORA\_AT\_ADR);

}

} else if (strncmp(atcmd, LORA\_AT\_CLASS, strlen(LORA\_AT\_CLASS)) == 0) {

int8\_t class;

if (atcmd\_index == strlen(LORA\_AT\_CLASS)) {

ret = true;

class = get\_lora\_class();

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %d\r\n", LORA\_AT\_CLASS, class);

} else if (atcmd\_index == (strlen(LORA\_AT\_CLASS) + 1)) {

class = strtol(atcmd + strlen(LORA\_AT\_CLASS), NULL, 0);

if (class == 0) {

ret = set\_lora\_class(CLASS\_A);

} else if (class == 2) {

ret = set\_lora\_class(CLASS\_C);

}

if (ret == true) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %d\r\n", LORA\_AT\_CLASS, class);

atcmd\_index = strlen(atcmd);

}

}

if (ret == false) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s ERROR\r\n", LORA\_AT\_CLASS);

}

} else if (strncmp(atcmd, LORA\_AT\_CFM, strlen(LORA\_AT\_CFM)) == 0) {

int cfm;

if (atcmd\_index == strlen(LORA\_AT\_CFM)) {

ret = true;

cfm = get\_lora\_tx\_cfm\_flag();

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %d\r\n", LORA\_AT\_CFM, cfm);

} else if (atcmd\_index == (strlen(LORA\_AT\_CFM) + 1)) {

cfm = strtol(atcmd + strlen(LORA\_AT\_CFM), NULL, 0);

ret = set\_lora\_tx\_cfm\_flag(cfm);

if (ret == true) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %d\r\n", LORA\_AT\_CFM, cfm);

}

}

if (ret == false) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s ERROR\r\n", LORA\_AT\_CFM);

}

} else if (strncmp(atcmd, LORA\_AT\_CFMTRIALS, strlen(LORA\_AT\_CFMTRIALS)) == 0) {

uint8\_t trials;

if (atcmd\_index == strlen(LORA\_AT\_CFMTRIALS)) {

ret = true;

trials = get\_lora\_tx\_cfm\_trials();

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %d\r\n", LORA\_AT\_CFMTRIALS, trials);

} else if (atcmd\_index > strlen(LORA\_AT\_CFMTRIALS)) {

trials = strtol(atcmd + strlen(LORA\_AT\_CFMTRIALS), NULL, 0);

ret = set\_lora\_tx\_cfm\_trials(trials);

if (ret == true) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %d\r\n", LORA\_AT\_CFMTRIALS, trials);

}

}

if (ret == false) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s ERROR\r\n", LORA\_AT\_CFMTRIALS);

}

} else if (strncmp(atcmd, LORA\_AT\_JOIN, strlen(LORA\_AT\_JOIN)) == 0) {

ret = set\_lora\_state(DEVICE\_STATE\_JOIN);

if (ret == true) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s OK\r\n", LORA\_AT\_JOIN);

}

if (ret == false) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s ERROR\r\n", LORA\_AT\_JOIN);

}

} else if (strncmp(atcmd, LORA\_AT\_DCS, strlen(LORA\_AT\_DCS)) == 0) {

uint32\_t dutycycle;

if (atcmd\_index == strlen(LORA\_AT\_DCS)) {

ret = true;

dutycycle = get\_lora\_tx\_dutycycle();

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %d\r\n", LORA\_AT\_DCS, dutycycle);

} else if (atcmd\_index > strlen(LORA\_AT\_DCS)) {

dutycycle = strtol(atcmd + strlen(LORA\_AT\_DCS), NULL, 0);

ret = set\_lora\_tx\_dutycycle(dutycycle);

if (ret == true) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %d\r\n", LORA\_AT\_DCS, dutycycle);

}

}

if (ret == false) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s ERROR\r\n", LORA\_AT\_DCS);

}

} else if (strncmp(atcmd, LORA\_AT\_TXSIZE, strlen(LORA\_AT\_TXSIZE)) == 0) {

uint8\_t len;

if (atcmd\_index == strlen(LORA\_AT\_TXSIZE)) {

ret = true;

len = get\_lora\_tx\_len();

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %d\r\n", LORA\_AT\_TXSIZE, len);

} else if (atcmd\_index > strlen(LORA\_AT\_TXSIZE)) {

len = strtol(atcmd + strlen(LORA\_AT\_TXSIZE), NULL, 0);

ret = set\_lora\_tx\_len(len);

if (ret == true) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %d\r\n", LORA\_AT\_TXSIZE, len);

}

}

if (ret == false) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s ERROR\r\n", LORA\_AT\_TXSIZE);

}

} else if (strncmp(atcmd, LORA\_AT\_LINKCHK, strlen(LORA\_AT\_LINKCHK)) == 0) {

ret = send\_lora\_link\_check();

if (ret == true) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s OK\r\n", LORA\_AT\_LINKCHK);

}

if (ret == false) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s ERROR\r\n", LORA\_AT\_LINKCHK);

}

} else if (strncmp(atcmd, LORA\_AT\_SCANMASK, strlen(LORA\_AT\_SCANMASK)) == 0) {

uint8\_t mask[2];

int length;

if (atcmd\_index == strlen(LORA\_AT\_SCANMASK)) {

ret = true;

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s 0x%04x\r\n", LORA\_AT\_SCANMASK, get\_lora\_freqband\_mask());

} else if (atcmd\_index > strlen(LORA\_AT\_SCANMASK)) {

length = hex2bin(&atcmd[strlen(LORA\_AT\_SCANMASK)], (uint8\_t \*)mask, 2);

if (length == 2) {

ret = set\_lora\_freqband\_mask(mask[1] | (mask[0] << 8));

if (ret == true) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s 0x%04x\r\n", LORA\_AT\_SCANMASK, get\_lora\_freqband\_mask());

}

}

}

if (ret == false) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s ERROR\r\n", LORA\_AT\_SCANMASK);

}

} else if (strncmp(atcmd, LORA\_AT\_TXPWR, strlen(LORA\_AT\_TXPWR)) == 0) {

if (atcmd\_index == strlen(LORA\_AT\_TXPWR)) {

ret = true;

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s 0x%04x\r\n", LORA\_AT\_TXPWR, get\_lora\_tx\_pwr());

} else if (atcmd\_index > strlen(LORA\_AT\_TXPWR)) {

uint8\_t tx\_pwr;

length = hex2bin(&atcmd[strlen(LORA\_AT\_TXPWR)], (uint8\_t \*)&tx\_pwr, 1);

if (length == 1) {

ret = set\_lora\_tx\_pwr(tx\_pwr);

if (ret == true) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s 0x%04x\r\n", LORA\_AT\_TXPWR, get\_lora\_tx\_pwr());

}

}

}

if (ret == false) {

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s ERROR\r\n", LORA\_AT\_TXPWR);

}

} else if (strncmp(atcmd, LORA\_AT\_TX, strlen(LORA\_AT\_TX)) == 0) {

uint8\_t port\_num[PORT\_LEN];

uint8\_t \*cmd = &atcmd[strlen(LORA\_AT\_TX)];

uint8\_t index = 0;

lora\_AppData\_t \*tx\_data;

tx\_data = get\_lora\_tx\_data();

if (tx\_data == NULL || atcmd\_index == strlen(LORA\_AT\_TX) + 1) {

ret = false;

} else {

for (index = 0; index < PORT\_LEN && cmd[index] != ':'; index++) {

port\_num[index] = cmd[index];

}

if (cmd[index] != ':') {

ret = false;

} else {

port\_num[index] = '\0';

ret = true;

}

if (ret == true) {

tx\_data->Port = (uint8\_t)atoi((char \*)port\_num);

tx\_data->BuffSize = atcmd\_index - strlen(LORA\_AT\_TX) - index - 1;

memcpy(tx\_data->Buff, &atcmd[strlen(LORA\_AT\_TX) + index + 1], tx\_data->BuffSize);

ret = tx\_lora\_data();

}

}

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %s\r\n", LORA\_AT\_TX, ret == true ? "OK" : "ERROR");

} else if (strncmp(atcmd, LORA\_AT\_RX, strlen(LORA\_AT\_RX)) == 0) {

lora\_AppData\_t \*rx\_data;

int16\_t len = 0;

len = snprintf(atcmd, ATCMD\_SIZE, "\r\n%s ", LORA\_AT\_RX);

rx\_data = get\_lora\_rx\_data();

if (rx\_data->BuffSize > 0) {

len += snprintf(atcmd + len, ATCMD\_SIZE, "%02x:", rx\_data->Port);

memcpy(atcmd + len, rx\_data->Buff, rx\_data->BuffSize);

len += rx\_data->BuffSize;

len += snprintf(atcmd + len, ATCMD\_SIZE, " ");

}

snprintf(atcmd + len, ATCMD\_SIZE, "%s\r\n", "OK");

} else if (strncmp(atcmd, LORA\_AT\_DEBUG, strlen(LORA\_AT\_DEBUG)) == 0) {

int32 debug;

if (atcmd\_index == strlen(LORA\_AT\_DEBUG)) {

ret = true;

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %d\r\n", LORA\_AT\_DEBUG, get\_lora\_debug());

} else if (atcmd\_index > strlen(LORA\_AT\_DEBUG)) {

debug = strtol(atcmd + strlen(LORA\_AT\_DEBUG), NULL, 0);

set\_lora\_debug(debug != 0);

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %s\r\n", LORA\_AT\_DEBUG, "OK");

}

} else if (strncmp(atcmd, LORA\_AT\_OTAA, strlen(LORA\_AT\_OTAA)) == 0) {

int32 otaa;

if (atcmd\_index == strlen(LORA\_AT\_OTAA)) {

ret = true;

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %d\r\n", LORA\_AT\_OTAA, get\_lora\_active\_mode());

} else if (atcmd\_index > strlen(LORA\_AT\_OTAA)) {

otaa = atoi(atcmd + strlen(LORA\_AT\_OTAA), NULL, 0);

set\_lora\_active\_mode(otaa != 0);

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %s\r\n", LORA\_AT\_OTAA, "OK");

}

} else if (strncmp(atcmd, LORA\_AT\_DEVADDR, strlen(LORA\_AT\_DEVADDR)) == 0) {

uint32 dev\_addr;

if (atcmd\_index == strlen(LORA\_AT\_DEVADDR)) {

ret = true;

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s 0x%x\r\n", LORA\_AT\_DEVADDR, get\_lora\_abp\_devaddr());

} else if (atcmd\_index > strlen(LORA\_AT\_DEVADDR)) {

dev\_addr = (uint32\_t)strtol(atcmd + strlen(LORA\_AT\_DEVADDR), NULL, 16);

set\_lora\_abp\_devaddr(dev\_addr);

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %s\r\n", LORA\_AT\_DEVADDR, "OK");

}

} else if (strncmp(atcmd, LORA\_AT\_NWKSKEY, strlen(LORA\_AT\_NWKSKEY)) == 0) {

uint8\_t \*nwkskey;

if (atcmd\_index == strlen(LORA\_AT\_NWKSKEY)) {

ret = true;

int i, len;

nwkskey = get\_lora\_abp\_nwkskey();

len = snprintf(atcmd, ATCMD\_SIZE, "\r\n%s:", LORA\_AT\_NWKSKEY);

for (i = 0; i < 16; i++) {

len += snprintf(atcmd + len, ATCMD\_SIZE, "0x%02x ", nwkskey[i]);

}

snprintf(atcmd + len, ATCMD\_SIZE, "\r\n");

} else if (atcmd\_index > strlen(LORA\_AT\_NWKSKEY)) {

length = hex2bin(&atcmd[strlen(LORA\_AT\_NWKSKEY)], buf, 16);

if (length == 16) {

set\_lora\_abp\_nwkskey(buf, 16);

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %s\r\n", LORA\_AT\_NWKSKEY, "OK");

}

}

} else if (strncmp(atcmd, LORA\_AT\_APPSKEY, strlen(LORA\_AT\_APPSKEY)) == 0) {

uint8\_t \*appskey;

if (atcmd\_index == strlen(LORA\_AT\_APPSKEY)) {

ret = true;

int i, len;

appskey = get\_lora\_abp\_appskey();

len = snprintf(atcmd, ATCMD\_SIZE, "\r\n%s:", LORA\_AT\_APPSKEY);

for (i = 0; i < 16; i++) {

len += snprintf(atcmd + len, ATCMD\_SIZE, "0x%02x ", appskey[i]);

}

snprintf(atcmd + len, ATCMD\_SIZE, "\r\n");

} else if (atcmd\_index > strlen(LORA\_AT\_APPSKEY)) {

length = hex2bin(&atcmd[strlen(LORA\_AT\_APPSKEY)], buf, 16);

if (length == 16) {

set\_lora\_abp\_appskey(buf, 16);

snprintf(atcmd, ATCMD\_SIZE, "\r\n%s %s\r\n", LORA\_AT\_APPSKEY, "OK");

}

}

} else {

snprintf(atcmd, ATCMD\_SIZE, "ERROR\r\n");

}

exit:

linkwan\_serial\_output(atcmd, strlen(atcmd));

PRINTF\_RAW("%s", LORA\_PROMPT);

atcmd\_index = 0;

memset(atcmd, 0xff, ATCMD\_SIZE);

}

1. lora class C state machine(入口函数)

void lora\_fsm( void )

{

switch ( DeviceState )

{

case DEVICE\_STATE\_INIT:

{

LoRaMacPrimitives.MacMcpsConfirm = McpsConfirm;

LoRaMacPrimitives.MacMcpsIndication = McpsIndication;

LoRaMacPrimitives.MacMlmeConfirm = MlmeConfirm;

LoRaMacCallbacks.GetBatteryLevel = LoRaMainCallbacks->BoardGetBatteryLevel;

#if defined(REGION\_AS923)

LoRaMacInitialization(&LoRaMacPrimitives, &LoRaMacCallbacks, LORAMAC\_REGION\_AS923);

#elif defined(REGION\_AS923)

LoRaMacInitialization(&LoRaMacPrimitives, &LoRaMacCallbacks, LORAMAC\_REGION\_AU915);

#elif defined(REGION\_CN470)

LoRaMacInitialization(&LoRaMacPrimitives, &LoRaMacCallbacks, LORAMAC\_REGION\_CN470);

#elif defined(REGION\_CN779)

LoRaMacInitialization(&LoRaMacPrimitives, &LoRaMacCallbacks, LORAMAC\_REGION\_CN779);

#elif defined(REGION\_EU433)

LoRaMacInitialization(&LoRaMacPrimitives, &LoRaMacCallbacks, LORAMAC\_REGION\_EU433);

#elif defined(REGION\_IN865)

LoRaMacInitialization(&LoRaMacPrimitives, &LoRaMacCallbacks, LORAMAC\_REGION\_IN865);

#elif defined(REGION\_EU868)

LoRaMacInitialization(&LoRaMacPrimitives, &LoRaMacCallbacks, LORAMAC\_REGION\_EU868);

#elif defined(REGION\_KR920)

LoRaMacInitialization(&LoRaMacPrimitives, &LoRaMacCallbacks, LORAMAC\_REGION\_KR920);

#elif defined(REGION\_US915)

LoRaMacInitialization(&LoRaMacPrimitives, &LoRaMacCallbacks, LORAMAC\_REGION\_US915);

#elif defined(REGION\_US915\_HYBRID)

LoRaMacInitialization(&LoRaMacPrimitives, &LoRaMacCallbacks, LORAMAC\_REGION\_US915\_HYBRID);

#else

#error "Please define a region in the compiler options."

#endif

TimerInit( &TxNextPacketTimer, OnTxNextPacketTimerEvent );

mibReq.Type = MIB\_ADR;

mibReq.Param.AdrEnable = LoRaParamInit->AdrEnable;

LoRaMacMibSetRequestConfirm( &mibReq );

mibReq.Type = MIB\_PUBLIC\_NETWORK;

mibReq.Param.EnablePublicNetwork = LoRaParamInit->EnablePublicNetwork;

LoRaMacMibSetRequestConfirm( &mibReq );

mibReq.Type = MIB\_DEVICE\_CLASS;

mibReq.Param.Class = LoRaParamInit->Class;

LoRaMacMibSetRequestConfirm( &mibReq );

#if defined(REGION\_EU868)

lora\_config\_duty\_cycle\_set(LORAWAN\_DUTYCYCLE\_ON ? ENABLE : DISABLE);

#if (USE\_SEMTECH\_DEFAULT\_CHANNEL\_LINEUP == 1)

LoRaMacChannelAdd(3, (ChannelParams\_t)LC4);

LoRaMacChannelAdd(4, (ChannelParams\_t)LC5);

LoRaMacChannelAdd(5, (ChannelParams\_t)LC6);

LoRaMacChannelAdd(6, (ChannelParams\_t)LC7);

LoRaMacChannelAdd(7, (ChannelParams\_t)LC8);

LoRaMacChannelAdd(8, (ChannelParams\_t)LC9);

LoRaMacChannelAdd(9, (ChannelParams\_t)LC10);

mibReq.Type = MIB\_RX2\_DEFAULT\_CHANNEL;

mibReq.Param.Rx2DefaultChannel = (Rx2ChannelParams\_t)

{ 869525000, DR\_3};

LoRaMacMibSetRequestConfirm(&mibReq);

mibReq.Type = MIB\_RX2\_CHANNEL;

mibReq.Param.Rx2Channel = (Rx2ChannelParams\_t)

{ 869525000, DR\_3};

LoRaMacMibSetRequestConfirm(&mibReq);

#endif

#endif

DeviceState = DEVICE\_STATE\_JOIN;

break;

}

case DEVICE\_STATE\_JOIN:

{

#if (OVER\_THE\_AIR\_ACTIVATION != 0)

MlmeReq\_t mlmeReq;

mlmeReq.Type = MLME\_JOIN;

mlmeReq.Req.Join.DevEui = DevEui;

mlmeReq.Req.Join.AppEui = AppEui;

mlmeReq.Req.Join.AppKey = AppKey;

mlmeReq.Req.Join.NbTrials = LoRaParamInit->NbTrials;

if ( NextTx == true )

{

LoRaMacMlmeRequest( &mlmeReq );

}

DeviceState = DEVICE\_STATE\_SLEEP;

#else

mibReq.Type = MIB\_NET\_ID;

mibReq.Param.NetID = LORAWAN\_NETWORK\_ID;

LoRaMacMibSetRequestConfirm(&mibReq);

mibReq.Type = MIB\_DEV\_ADDR;

mibReq.Param.DevAddr = DevAddr;

LoRaMacMibSetRequestConfirm(&mibReq);

mibReq.Type = MIB\_NWK\_SKEY;

mibReq.Param.NwkSKey = NwkSKey;

LoRaMacMibSetRequestConfirm(&mibReq);

mibReq.Type = MIB\_APP\_SKEY;

mibReq.Param.AppSKey = AppSKey;

LoRaMacMibSetRequestConfirm(&mibReq);

mibReq.Type = MIB\_NETWORK\_JOINED;

mibReq.Param.IsNetworkJoined = true;

LoRaMacMibSetRequestConfirm(&mibReq);

DeviceState = DEVICE\_STATE\_SEND;

#endif

break;

}

case DEVICE\_STATE\_JOINED:

{

PRINTF("JOINED\n\r")

;

DeviceState = DEVICE\_STATE\_SEND;

break;

}

case DEVICE\_STATE\_SEND:

{

if ( NextTx == true )

{

PrepareTxFrame( );

NextTx = SendFrame( );

}

if ( ComplianceTest.Running == true )

{

// Schedule next packet transmission as soon as possible

TimerSetValue( &TxNextPacketTimer, 5000 ); /\* 5s \*/

TimerStart( &TxNextPacketTimer );

}

else if ( LoRaParamInit->TxEvent == TX\_ON\_TIMER )

{

// Schedule next packet transmission

TimerSetValue( &TxNextPacketTimer, LoRaParamInit->TxDutyCycleTime );

TimerStart( &TxNextPacketTimer );

}

DeviceState = DEVICE\_STATE\_SLEEP;

break;

}

case DEVICE\_STATE\_SLEEP:

{

// Wake up through events

break;

}

default:

{

DeviceState = DEVICE\_STATE\_INIT;

break;

}

}

}

1. 心跳时间函数

ATprocess函数在将AT指令处理完毕后会转入相应的功能函数，我们以设置心跳时间功能为例，该功能的AT指令为LORA\_AT\_BEATTIME=XXX，其函数如下所示，心跳的目的是定时要求路灯返回相应的状态信息，以便及时发现排查故障：

static int at\_beattime\_func(int opt, int argc, char \*argv[])

{

int ret = LWAN\_ERROR;

uint8\_t beatflag;

uint32\_t beattime;//单位秒

switch(opt) {

case QUERY\_CMD: {

ret = LWAN\_SUCCESS;

//lwan\_mac\_config\_get(MAC\_CONFIG\_APP\_PORT, &port);

beattime=lora\_dev\_para.Beattime;

snprintf((char \*)atcmd, ATCMD\_SIZE, "\r\n%s:%d,%ds\r\nOK\r\n", LORA\_AT\_BEATTIME,lora\_dev\_para.Beatflag,lora\_dev\_para.Beattime/1000);

break;

}

case DESC\_CMD: {

ret = LWAN\_SUCCESS;

snprintf((char \*)atcmd, ATCMD\_SIZE, "\r\n%s:\"value,value\"\r\nOK\r\n", LORA\_AT\_BEATTIME);

break;

}

case SET\_CMD: {

if(argc < 1) break;

lora\_dev\_para.Beatflag = (u\_int8\_t)strtol((const char \*)argv[0], NULL, 0);

beattime = (uint32\_t)strtol((const char \*)argv[1], NULL, 0);

lora\_dev\_para.Beattime=beattime\*1000; //ms

// if (set\_lora\_freq() == LWAN\_SUCCESS) {

ret = LWAN\_SUCCESS;

snprintf((char \*)atcmd, ATCMD\_SIZE, "\r\nOK\r\n");

//}

break;

}

default: break;

}

return ret;

}

void Heart\_Beat\_EVENT(void)

{

TimerStop( &Heartbeat );

AT\_cmd\_send = true;

dev\_state = DEVICE\_STATE\_NUESTATS;

// Uart2PutBuffer( &appConcentrator\_Uart2,"1001\r\n", 6 );

if( TRUE == RegisterPacketEnable )

{

TimerSetValue( &Heartbeat, (Hearttime \* 1000) );

TimerStart( &Heartbeat );

}

osal\_set\_event(appConcentrator\_TaskID, TRANSPARENT\_MAIN\_LOOP\_EVT);

}

路灯控制系统上电后，终端的路灯由下至上开始自动组网，组网成功后数据采集模块将终端的编号存储到内部 flash，集中器将数据采集模块的编号存储到外部 flash。服务器可接收到组网成功的信息，则可终端进行命令控制。其中心跳机制用来监视入网的模块是否在线。组网的时候需要对路灯的工作模式和 LoRa 模块的参数进行设置，数据采集程序中的定时器用于心跳组网轮询计时，如果已入网的模块失联，则系统将会将附近的入网的数据采集模块转为中继，将 LoRa 模式由定点模式变为广播模式，重新写入 LoRa 地址为 0xFFFF 和更新配置，重新进行组网。如果多次连接还未能入网则要入网的数据采集模块未在组网范围内，需要重新进行组网。组网成功后，客户可以通过网页对路灯终端进行指令操作，服务器对终端上传的数据进行分析。

下面是 LoRa 模块初始化时所需配置的 LoRa 参数：

head =C0; //控制命令，LoRa 具有掉电保存功能

address =0xFFFF; //写入地址,设置为广播模式

check =8N1; //设置串口校验位

baud = 115200; //设置初始化波特率

transferSpeed = 2.4kbps; //设置空中传输速率

channel = 04; //写入信道

transferMode = TRANSPARENT; //传输模式

driverMode = LORA\_DRIVE\_MODE\_1; //IO驱动方式

wakeupTime = 250ms; //设置唤醒时间

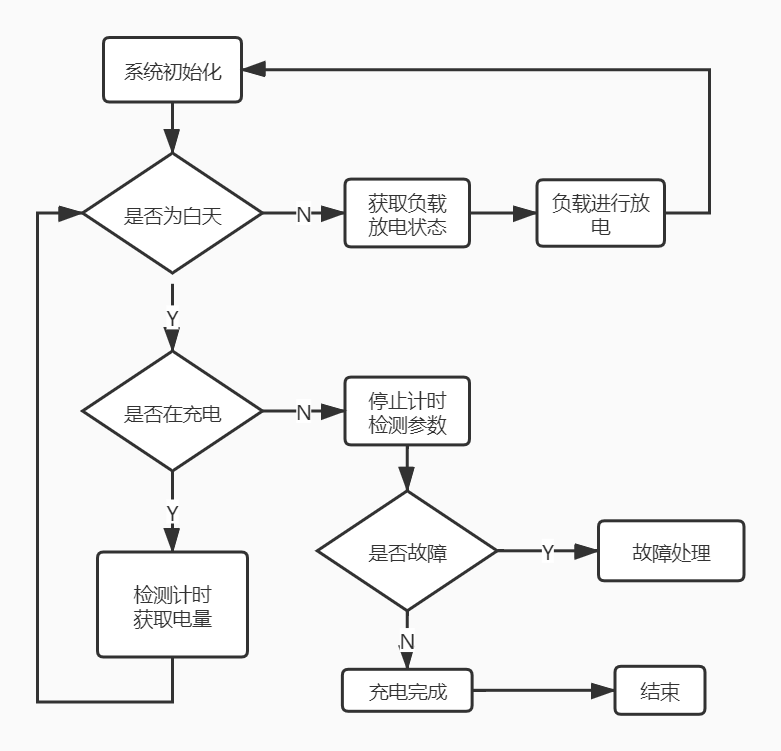
FEC = LORA\_FEC\_ON; //开启 FEC 纠错功能

transferPower = 20dBm; //设置发射功率

workMode = NORMAL\_MODE; //设置 LoRa 模块工作模式

1. 路灯终端的程序设计

系统上电后，首先进行初始化：配置定时器、配置 GPIO、初始化 LoRa模块等，然后判断时间为白天还是晚上；如果当前时间为晚上，系统则控制电池对负载供电；如果当前时间为白天，系统首先判断电池是否在充电；如果电池正在充电，系统则会开始计时，每隔 60 秒检测一次电量，并根据此电量对电池采用不同的充电方式，直到电池电量达到满程值，停止对蓄电池充电，充电完成；如果当前时间为白天，电池没在充电，系统则会停止计时，并检测参数，判断是否出现故障；如果是由于出现故障而导致电池不在充电，系统则采用相应的故障处理方式；如果没有发生故障，系统则进入休眠状态。整个流程如下图所示：



当太阳能电池放电时，需要对其可能出现的故障进行检测并做出及时处理。在系统运行时，首先会判断电路是否发生故障；如果发生故障系统会及时关闭负载，并保存此时的工作状态，并根据不同的故障类型，通过负载指示灯和电池指示灯的不同状态显示出来；如果没有发生故障，系统则会判断电路之前是否发生过故障；如果之前发生了故障，系统则会恢复当时的工作状态；如果之前没有发生故障，系统则会判断负载是否在工作，如果在工作，则负载指示灯亮；如果负载没有工作，系统则会判断电池是否在充电，如果在充电，则充电指示灯亮；如没有在充电，系统则会判断电池是否充满，如果充满，则充电指示灯灭；如果没有充满，则充电指示灯绿灯快闪。系统会根据太阳能板电压与蓄电池电压的大小关系实现对蓄电池自动充电；针对电池的不同工作阶段以及故障类型，系统设置了不同强度的充放电方法和故障处理方式，以此来增加蓄电池的使用周期和系统的稳定性；同时系统也会将运行状态和故障类型以不同的字符串通过 LoRa 无线模块传给上位机，实现对系统的远程实时监控。

系统会在蓄电池对负载进行供电时启动计时，并会将夜晚分为 6 个时间阶段，随着夜晚向白天的推移，供电也会向下一个阶段移动，最终到达程序所设定的晨亮时间时，系统会停止蓄电池向负载供电，模块放电工作部分代码如下所示：

if(Get\_LoadWork\_Count\_Flag() == 0)

Start\_LoadWork\_Count(ENABLE);//启动工作计时

if(Get\_LoadWorkRecord\_Count\_Flag()==0)

Start\_LoadWork\_CountkRecord(ENABLE);

//启动工作时间记录计时

if((h\_Count\_LoadWork\*3600+s\_Count\_LoadWork)>=GetConfigParameterTime(TIME\_1))

{

//计时完成 转换工作状态

Start\_LoadWork\_Count(DISABLE);

TIM3\_Count\_LoadWork\_Reset();

SetLoadWorkStatus(TIME\_2\_STATUS);

ReadConfigParameter();

}

else {

RegulateCurrent(GetConfigParameterCurrent(CURRENT\_1));

}

系统在运作过程中需要保证路灯的供电电流的稳定，所以在电路里面加入了恒流功能，主要是通过 GetLoadCurrent 和 GetLoadVoltage 函数获取电路中的实际运行的电流、电压和程序中设定的阀值进行比较，遵循超减，少增的原则来进行电路中电流的平衡以保证路灯稳定工作，如果获取的当前电流值远远大于程序中设定的阈值 LOAD\_CURRENT\_DIFF\_MAX，不进行比较直接增减电流，路灯恒流程序设计部分代码如下所示：

if(setcurrent <= 0.4){

comp = LOAD\_CURRENT\_DIFF - 0.01;

offset = LOAD\_CURRENT\_OFFSET - 0.01;

}

else{

comp = LOAD\_CURRENT\_DIFF;

offset = LOAD\_CURRENT\_OFFSET; // 如 果 电 流 小 于 0.4 减 少

LOAD\_CURRENT\_DIFF 和 LOAD\_CURRENT\_OFFSET 的值

}

if(curcurrent <= setcurrent) {//当前电流小于设置电流 增加电流输出

diff = setcurrent - curcurrent;//取差值

if(diff > comp) {//差值较大则 增加工作电流

RegulateOrientation\_flag=INCREASE\_FLAG;//增大工作电流

}

//若差值大于 MAX 则判定为当前工作电流与设置电流差距较大 则进行增大

工作电流操作 跳过 cache\_flag 判断

if(diff > LOAD\_CURRENT\_DIFF\_MAX && current\_cache == setcurrent)

cache\_flag =REGULATE\_FLAG;

}

else{

//当前电流大于设置电流 减小电池输出

diff = curcurrent - setcurrent;//取差值

if(diff > comp) {//差值较大则 减小工作电流

RegulateOrientation\_flag=DECREASE\_FLAG;//减小工作电流

}

//若差值大于 MAX 则判定为当前工作电流与设置电流差距较大 则进行减小

工作电流操作 跳过 cache\_flag 判断

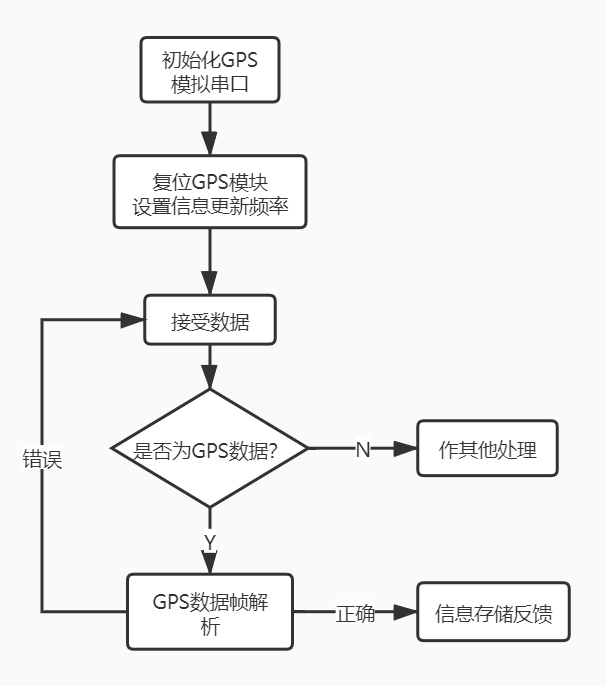
if(diff > LOAD\_CURRENT\_DIFF\_MAX && current\_cache ==setcurrent)

cache\_flag = REGULATE\_FLAG;

}

1. GPS功能模块的程序设计

GPS 功能模块程序流程图如下所示：



由于系统运行时路灯的位置只靠 LoRa 获取是不准确的，因此本系统采用GPS 来提高定位的精度，及时获取故障路灯的位置，以便于系统的维护。数据采集模块的第二部分工作是接收卫星传递过来的信息，其采用的是 SKG09BL 导航模块，定位精度达到 3m 左右，速度精度是 0.1m/s，它的外围电路主要是由于RF 射频芯片，基带芯片和核心 CPU 组成。支持无源、有缘天线，其连接方式是通过串口和主芯片连接。由于选用的 STM32 芯片的串口有限，所以在设计过程中需用模拟串口接收数据，定时器 TIM3 模拟接收 GPS 信息所需要的波特率，GPIO 口 PA4 与 PA5 模拟串口用来接收卫星数据。传输的数据遵循由美国国家海事电子协会于 1983 年制定的 NMEA-0183 协议，输出采用 ASCII 码形式，数据传输以“语句”的方式进行，GPS 信息的每个语句均以“$”开头，然后是两个字母的“识别符”和三个字母的“语句名”，接着就是以逗号分割的数据体，语句末尾为校验和，整条语句以回车换行符结束。由于 GPS 有三种启动模式热启动、温启动、冷启动，考虑到系统实际应用过程中不会出现断电和持续传输数据，程序初始化时候选用的是温启动模式。

温启动代码：

case WARM\_RESTART\_MODE://温启动

len = createGpsCommand(data,"%s","102");

status = GPS\_START\_FEEDBACK\_ARRAY.len;

gpsUartCommunicationReceive.mode = GPS\_CONFIG\_MODE;

resetGpsUartCommunicationReceive(0);

Sim\_UART\_Send((uint8\_t\*)data,len);

指令输出格式如下所示：

GPGLL：大地坐标信息

$GPGLL,,,,,235943.800,V,N\*78

GPGSA：当前卫星信息

$GPGSA,A,1,,,,,,,,,,,,,,,\*1E

GPGSV：输出可见的卫星信息

$GPGSV,1,1,00\*79

GPRMC: 输出 GPS 推荐的最短数据信息

$GPRMC,000503.800,V,,,,,0.00,0.00,060180,,,N\*4C

GPVTG： 地面速度信息

$GPVTG,0.00,T,,M,0.00,N,0.00,K,N\*32

GPGGA：输出 GPS 的定位信息

$GPGGA,004603.800,,,,,0,0,,,M,,M,,\*41

程序通过 Sim\_UART\_Init 函数模拟串口接收卫星发送的数据，再通过函数gpsRestart(WARM\_RESTART\_MODE,12000)进入温启动模式，GPS 的更新频率5s 一次通过 gpsUpdateRate 函数进行设定;然后进入 GpsUartDataCheckAndDeal函数对数据进行解析，其中解析主要作用是将 GPS 的接收到的数据进行分类，以便管理者区分接收到的数据是哪一类 GPS 数据，并将解析后的信息通过memcpy 函数存储到内部 flash，最后向上一级反馈收到的信息。其中部分代码:

if(strstr((char\*)gpsUartCommunicationReceive.dataMode,"TG")!=NULL){

GPVTG.status = 0;

clearWithLen(GPVTG.data,96);

memcpy(GPVTG.data,gpsUartCommunicationReceive.data,gpsUartCommunicationReceive.len);

GPVTG.status = 1;

}