

# 江 苏 大 学

JIANGSU UNIVERSITY

## 《无线传感网与识别技术》

### 课程设计



学院名称：\_\_\_\_\_ 计算机学院

专业班级：\_\_\_\_\_ 物联网工程 2303

学生姓名：\_\_\_\_\_ 邱佳亮

学生学号：\_\_\_\_\_ 3230611072

指导教师姓名：\_\_\_\_\_ 熊书明

2025 年 7 月

# 目录

1. 设计目的 .....	2
2. 设计要求 .....	2
3. 系统整体结构 .....	2
4. 模块的设计和实现 .....	3
4.1 无线 Ad Hoc Flooding 实验例程 .....	3
4.1.1 功能描述 .....	3
4.1.2 主要函数 .....	3
4.1.3 程序注释 .....	4
4.1.4 运行结果 .....	14
4.1.5 问题分析 .....	15
4.2 传感器节点与射频识别的实验例程 .....	15
4.2.1 功能描述 .....	15
4.2.2 主要函数 .....	16
4.2.3 程序注释 .....	17
4.2.4 运行结果 .....	21
4.2.5 问题分析 .....	23
4.3 数据采集和标签信息的洪泛传输 .....	23
4.3.1 设计思路 .....	23
4.3.2 流程图 .....	24
4.3.3 运行结果 .....	26
4.3.4 问题分析 .....	28
4.3.5 改进方向 .....	30
5. 收获和心得体会 .....	31
6. 关键代码 .....	32
6.1 FloodingM.nc .....	32
6.2 Adhoc_APPM.nc .....	37
6.3 Server.py .....	42

# 节点数据采集和标签信息的远程洪泛传输

## 1. 设计目的

1) 完成韩博 PDF 文档-无线 Ad Hoc Flooding 实验例程【实习 13, P257, 271】中那几个 nc 文件代码实现的理解。

2) 精读韩博 PDF 文档-传感器节点与射频识别的实验例程【实习 17, P350, 364】中那几个 nc 文件。

3) 综合 1 和 2 两个例程, 完成节点数据采集和标签信息的远程洪泛传输, 在与 Sink 节点相连接的 PC 上能看到结果(标签信息、感知的数据)。其中, 节点感知数据采集包括光照值、温湿度值, 采集节点每 4s 采集一次温湿度、每 2s 采集一次光照。标签数据读写控制周期 3s。

## 2. 设计要求

1) 对主要函数(不少于 50%)的功能完成文字描述

2) 对函数中的语句写出注释(不少于 30%) 3) 完成该例程的运行, 做好实验结果的分析说明、并将运行结果截图以备将来实验报告

3) 记录编译、运行过程中出现的错误, 及解决方法, 以及心得体会。

4) 应在 PC 上, 采用 Python 或 Java 或 C 语言编写串口读写程序, 实现串口数据收发功能; 前期可直接选用相关串口工具访问串口数据。

5) 在本地 PC 上串口数据的读写完成后, 将数据远程传输到云服务器, 然后, 远程 PC 能访问到数据。

## 3. 系统整体结构

本次课程设计构建了一个完整的无线传感网络数据采集与传输系统, 系统主要由三个核心部分组成: Adhoc\_Flooding 模块负责实现基于洪泛协议的多跳无线通信功能, RFIDwithUSN 模块完成 RFID 标签的识别与数据读写操作, 以及环境传感器模块实现温湿度与光照数据的采集。在具体实现上, 我们首先对 Adhoc\_Flooding 和 RFIDwithUSN 的源代码进行了深入分析, 通过添加详细的注释来理解每个函数的功能实现细节, 特别是消息处理流程和状态转换机制。然后将指导书中提供的温湿度传感器(SHT11)和光照传感器的采集代码进行整合, 统一集

成到 `Adhoc_FloodingM.nc` 文件中，构建了包含节点 ID、时间戳、传感器类型和测量值等字段的标准化数据包格式。系统采用分层定时器调度策略，分别为温湿度（4 秒）、光照（2 秒）和 RFID（3 秒）设置了不同的采集周期。所有采集到的数据通过洪泛协议传输至 Sink 节点，再经串口上传至 PC 端。在 PC 端，我们使用 Python 语言开发了图形用户界面，基于 PyQt5 框架实现了数据的实时可视化展示，完整实现了从数据采集、无线传输到可视化展示的全流程功能。。

## 4. 模块的设计和实现

### 4.1 无线 Ad Hoc Flooding 实验例程

#### 4.1.1 功能描述

这个例程实现了一个基于无线自组织网络（Adhoc）的应用系统，包含节点间通信、路由转发和汇聚节点处理等功能。其每隔 3 秒将自身生成的数据通过 flooding 算法传送给 0 号节点(Sink 节点:与 PC 连接的传感器节点)。

#### 4.1.2 主要函数

**StdControl.start:** 这是路由模块的初始化函数，在系统启动时执行核心准备工作。它通过调用 `SeedInit.init()` 初始化随机数种子，生成随机序列号标识本节点，原子化地初始化路由表（将所有条目设为无效状态），并设置广播地址为默认下一跳。最后启动底层通信模块，为后续数据收发建立基础。

**RecvFromMAC.receive:** 作为 MAC 层消息的入口处理函数，实现了智能的消息分发逻辑。它首先解析消息头判断目标节点：若是本节点且未处理过则提交给应用层（`RecvToAPP_task`）；若非本节点则执行多跳转发，检查 TTL 和重复包后，递减 TTL 并修改路由头（添加当前节点到路径记录），最后将消息存入转发缓冲区并触发 `Forwarding_task`。

**Timer.fired:** 周期性数据生成函数，定义了普通节点的核心行为。每当时钟触发时，构造包含节点 ID、递增序列号和测试数据的应用层报文，通过 `Rout_Send.send()` 提交给路由层。配合 LED 状态指示：切换 LED1 表示数据生成事件，点亮 LED2 指示发送请求已提交。

**Rout\_Send.sendDone:** 发送结果回调函数，提供关键的状态同步。当路由层完成发送操作后，根据 `error` 参数判断结果：若 SUCCESS 则熄灭 LED2，表示无线发送过程结束。

### 4.1.3 程序注释

#### Adhoc\_APPM

```

1. module Adhoc_APPM {
2.   uses {
3.     interface Boot;
4.     interface Timer<TMilli> as Timer1;
5.     interface Timer<TMilli> as Timer2;
6.     interface Timer<TMilli> as Timer3;
7.     interface Timer<TMilli> as Timer4;
8.     interface Leds;
9.     interface BusyWait<TMicro, uint16_t>;
10.    interface RFID_Control;
11.    interface StdControl as RControl;
12.    interface AMSend as Rout_Send;
13.    interface Receive as Rout_Receive;
14.    interface SplitControl as CommControl;
15.  }
16.
17.  // Sensor components
18.  uses {
19.    interface Read<uint16_t> as Read_Humi;
20.    interface Read<uint16_t> as Read_Temp;
21.    interface Read<uint16_t> as Read_Photo;
22.  }
23.
24.  // Uart component
25.  uses {
26.    interface StdControl as SCSuartSTD;
27.    interface SCSuartDBG;
28.  }
29.
30. }
31. implementation {
32.
33.   message_t TXData;
34.
35.   //// sensor data ////
36.   uint16_t myTemp=0xFFFF;
37.   uint16_t myHumi=0xFFFF;
38.   uint16_t myPhoto=0xFFFF;
39.   uint16_t Raw_Temp=0xFFFF; // Raw temp info
40.   uint8_t FUN = 0x00;
41.   uint8_t Data[5] = {0x00,0x00,0x00,0x00,0x00,};

```

```

42.  //////////////////////////////////
43. //  uint8_t OutputUartMsg[64]; // 串口输出消息 buff
44. void calc_SHT_(uint16_t p_humidity ,uint16_t p_temperature);
45.
46.
47. event void Boot.booted() {
48.     uint8_t mote_id = (uint8_t) TOS_NODE_ID;
49.     if (TOS_NODE_ID != Sink_Addr) // 不是 sink 节点
50.     {
51.         call Timer3.startOneShot(1000);
52.         call Timer1.startPeriodic(SHT_Interval); // 每隔 20s 读取一次温湿度数据
53.         call Timer4.startPeriodic(3000); // 每隔 7s 读取一次标签
54.     }
55.     call RControl.start();
56.     call CommControl.start();
57. }
58.
59. event void CommControl.startDone(error_t error) {
60.     call SCSuartSTD.start();
61.     call RFID_Control.start();
62. }
63.
64. event void CommControl.stopDone(error_t error) {}
65.
66. event void Timer3.fired() {
67.     call Timer2.startPeriodic(light_Interval); // 每隔 10s 读取一次光照数据
68. }
69. // get sensor data as photo, temp., humi.
70. // 定时器溢出读取光照数据
71. event void Timer2.fired() {
72.     call Leds.led2Toggle();
73.     call Read_Photo.read();
74. }
75. // 定时器溢出读取温湿度
76. event void Timer1.fired() {
77.     call Leds.led2Toggle();
78.     call Read_Temp.read();
79. }
80.
81. // 一般节点发送给 Sink 节点采集到的数据
82. task void transmit_frame(){
83.
84.     DataFrameStruct DFS;
85.

```

```

86.     call Leds.led1On();
87.
88.     DFS.Temp = myTemp;
89.     DFS.Humi = myHumi;
90.     DFS.Photo = myPhoto;
91.     atomic DFS.FUN = FUN;
92.     // memcpy (DFS.ID, ID, sizeof(ID));
93.     memcpy (DFS.Data, Data, sizeof(Data));
94.     memcpy (call Rout_Send.getPayload(&TXData), &DFS, sizeof(DataFrameStruct));
95.
96.     call Rout_Send.send(Sink_Addr, &TXData, sizeof(DataFrameStruct));
97. }
98.
99. // 一般节点数据发送完成事件
100. event void Rout_Send.sendDone(message_t* m, error_t err) {
101.     if (err == SUCCESS)
102.         call Leds.led1Off();
103. }
104.
105. // sink 节点接收到一般节点发送的数据事件
106. event message_t* Rout_Receive.receive(message_t* msg, void* payload, uint8_
t len) {
107.     if (TOS_NODE_ID == Sink_Addr)
108.     {
109.         uint8_t UART_Buff[65], *UART_Buff_p;
110.         uint8_t UART_Buff_len = 0, i;
111.         Route_Msg NWKF;
112.         DataFrameStruct DFS;
113.         UartFrameStruct UFS;
114.
115.         memcpy(&NWKF, call Rout_Send.getPayload(msg), sizeof(Route_Msg));
116.         memcpy(&DFS, NWKF.AppData, sizeof(DataFrameStruct));
117.         UART_Buff_p = (uint8_t *)&UFS;
118.
119.         {
120.             uint32_t Packet_Seq = (uint32_t) NWKF.Sequence;
121.             int16_t OrigSrcAddr = NWKF.OrigSrcAddr;
122.             //call SCSuartDBG.UARTSend(UART_Buff, 6);
123.
124.             memcpy (UART_Buff_p+6, (void *)&OrigSrcAddr, 2);
125.             memcpy (UART_Buff_p+8, (void *)&TOS_NODE_ID, 2);
126.             memcpy (UART_Buff_p+10, (void *)&NWKF.Dst2_for_multihop, 2);
127.             memcpy (UART_Buff_p+12, (void *)&NWKF.Dst3_for_multihop, 2);
128.             memcpy (UART_Buff_p+14, (void *)&Packet_Seq, 4);

```

```

129.     memcpy (UART_Buff_p+18,(void *)&DFS.Temp, 2);
130.     memcpy (UART_Buff_p+20,(void *)&DFS.Humi, 2);
131.     memcpy (UART_Buff_p+22,(void *)&DFS.Photo, 2);
132.     memcpy (UART_Buff_p+24,(void *)&DFS.FUN, 1);
133.     // memcpy (UART_Buff_p+25,(void *)&DFS.ID, 8);
134.     memcpy (UART_Buff_p+25,(void *)&DFS.Data, 5);
135.     }
136.     UART_Buff_len = 0;
137.     for ( i=6; i<sizeof(UartFrameStruct) ; i++)
138.     {
139.         UART_Buff[UART_Buff_len++] = UART_Buff_p[i];
140.     }
141.     // call SCSuartDBG.UARTSend(DFS.Data, sizeof(DFS.Data));
142.     // 串口数据发送
143.     call SCSuartDBG.UARTSend(UART_Buff, UART_Buff_len -13);
144.
145.     call Leds.led0Toggle();
146.     }
147.     return msg;
148. }
149.
150. // 读取光照完成事件, 发送数据至 sink 节点
151. event void Read_Photo.readDone(error_t err, uint16_t val) {
152. if (err == SUCCESS)
153. {
154.     myPhoto = val;
155.     atomic FUN = 1;
156. }
157. post transmit_frame();
158. }
159.
160. // 读取温度完成事件, 准备读取湿度
161. event void Read_Temp.readDone(error_t err, uint16_t val) {
162. if (err == SUCCESS)
163.     Raw_Temp = val;
164. call Read_Humi.read();
165. }
166.
167. // 读取湿度完成事件, 发送数据至 sink 节点
168. event void Read_Humi.readDone(error_t err, uint16_t val) {
169. if (err == SUCCESS && Raw_Temp!=0xFFFF)
170. {
171.     calc_SHT_(val, Raw_Temp);
172.     atomic FUN = 2;

```



```

173. }
174. post transmit_frame();
175. }
176.
177. // 对温湿度循环冗余, 得到真实数值
178. void calc_SHT_(uint16_t p_humidity ,uint16_t p_temperature)
179. //-----
-----
180. // calculates temperature [C] and humidity [%RH]
181. // input : humi [Ticks] (12 bit)
182. // temp [Ticks] (14 bit)
183. // output: humi [%RH]
184. // temp [C]
185. {
186. const float C1=-4.0; // for 12 Bit
187. const float C2= 0.0405; // for 12 Bit
188. const float C3=-0.0000028; // for 12 Bit
189. const float T_1=0.01; // for 14 Bit @ 5V
190. const float T_2=0.00008; // for 14 Bit @ 5V
191.
192. float rh_lin; // rh_lin: Humidity linear
193. float rh_true; // rh_true: Temperature compensated humidity
194. float t_C; // t_C : Temperature [C]
195. float rh=(float)p_humidity; // rh: Humidity [Ticks] 12 Bit
196. float t=(float)p_temperature; // t: Temperature [Ticks] 14 Bit
197.
198. t_C=t*0.01 - 40; //calc. Temperature from ticks to [C]
199. rh_lin=C3*rh*rh + C2*rh + C1; //calc. Humidity from ticks to [%RH]
200. rh_true=(t_C-25)*(T_1+T_2*rh)+rh_lin; //calc. Temperature compensated humidity [%RH]
201. if(rh_true>100)rh_true=100; //cut if the value is outside of
202. if(rh_true<0.1)rh_true=0.1; //the physical possible range
203. atomic myTemp=(uint16_t)t_C; //return temperature [C]
204. atomic myHumi=(uint16_t)rh_true; //return humidity[%RH]
205. }
206. //////////////////////////////////////
207. // 获取 15693ID 完成后发送至 Sink 节点
208. async event void RFID_Control.GetID_15693_Done (char status, uint8_t *buff, char size){
209. // memcpy(ID, buff, 8);
210. // atomic FUN = 3;
211. // post transmit_frame();
212. // call RFID_Control.RData_15693(block);
213. }

```

```

214. // 获取 15693 标签数据完成后发送至 Sink 节点
215. async event void RFID_Control.RData_15693_Done (char status, uint8_t *buff, c
char size){
216.     call SCSuartDBG.UARTSend(buff, 5);
217.     memcpy(Data, buff, 5);
218.     atomic FUN = 3;
219.     post transmit_frame();
220. }
221. // 读取标签数据定时器
222. event void Timer4.fired() {
223.     call RFID_Control.RData_15693 (5);
224. }
225.
226. async event void RFID_Control.GetID_14443A_Done(char status, uint8_t *buff, c
char size) {}
227. async event void RFID_Control.WData_15693_Done(char status){}
228. }

```

### FloodingM

```

1. module FloodingM {
2.
3.     provides {
4.         interface StdControl;
5.         interface AMSend as SendFromAPP;
6.         interface Receive as RecvToAPP;
7.     }
8.     uses {
9.         interface Timer<TMilli>;
10.        interface ParameterInit<uint16_t> as SeedInit;
11.        interface Random;
12.        interface SplitControl as CommControl;
13.        interface AMSend as SendToMAC;
14.        interface Receive as RecvFromMAC;
15.    }
16.
17. }implementation{
18.
19.     message_t SendMsg, RecvMsg, ForwardMsg[MAX_Forward_Buff];
20.     Route_Msg NWKF;
21.     uint16_t Next_Addr;
22.     uint8_t Forward_Buff_Index;
23.     uint8_t RTable_Index;
24.     uint8_t mySequence;
25.     Route_Table RTable[MAX_RTABLE];
26.

```

```

27.  command error_t StdControl.start() {
28.      uint8_t i;
29.      uint16_t random_num;
30.
31.      call SeedInit.init(TOS_NODE_ID);
32.      random_num = call Random.rand16();
33.
34.      atomic {
35.          Next_Addr = AM_BROADCAST_ADDR;
36.          Forward_Buff_Index = 0;
37.          RTable_Index = 0;
38.          mySequence = (uint8_t) (random_num%0xFF);
39.          for (i=0;i<MAX_RTABLE;i++) {
40.              RTable[i].FinalDstAddr = UnknownAddr;
41.              RTable[i].OrigSrcAddr = UnknownAddr;
42.              RTable[i].Sequence = 0xFF;
43.          }
44.      }
45.      call CommControl.start();
46.      return SUCCESS;
47.  }
48.
49.  command error_t StdControl.stop() {return SUCCESS;}
50.  event void CommControl.startDone(error_t error) {}
51.  event void CommControl.stopDone(error_t error) {}
52.
53.  //////////////////////////////////////////
54.  void insertMSGtoRTable(message_t* msg) {
55.      Route_Msg pack;
56.      memcpy(&pack, call SendToMAC.getPayload(msg), sizeof(Route_Msg));
57.
58.      atomic{
59.          RTable[RTable_Index].FinalDstAddr = pack.FinalDstAddr;
60.          RTable[RTable_Index].OrigSrcAddr = pack.OrigSrcAddr;
61.          RTable[RTable_Index].Sequence = pack.Sequence;
62.
63.          RTable_Index++;
64.          RTable_Index %= MAX_RTABLE;
65.      }
66.  }
67.
68.  bool isRecvPrevious (message_t* msg) {
69.      Route_Msg pack;
70.      bool return_status = 0;

```

```

71. uint8_t i;
72.
73. memcpy(&pack, call SendToMAC.getPayload(msg), sizeof(Route_Msg));
74. for (i=0;i<MAX_RTABLE;i++) {
75.     if (RTable[i].FinalDstAddr == pack.FinalDstAddr &&
76.         RTable[i].OrigSrcAddr == pack.OrigSrcAddr &&
77.         RTable[i].Sequence == pack.Sequence)
78.     {
79.         return_status = 1;
80.         break;
81.     }
82. }
83. return return_status;
84. }
85.
86. //////////////////////////////////////
87.
88. command error_t SendFromAPP.send(am_addr_t addr, message_t* msg, uint8_t len)
89. {
90.     Route_Msg Route_M;
91.     void *DataPayLoad = call SendToMAC.getPayload(msg);
92.     error_t return_status;
93.
94.     Route_M.FrameControl = GeneralDataFrame;
95.     Route_M.FinalDstAddr = addr;
96.     Route_M.OrigSrcAddr = TOS_NODE_ID;
97.     Route_M.Sequence = mySequence;
98.     Route_M.TTL = Default_TTL;
99.     Route_M.Dst2_for_multihop = UnknownAddr;
100.    Route_M.Dst3_for_multihop = UnknownAddr;
101.    memcpy((void *)&(Route_M.AppData), DataPayLoad, len);
102.    memcpy(call SendToMAC.getPayload(&SendMsg), (uint8_t *)&Route_M, sizeof(Route_Msg));
103.
104.    mySequence++;
105.    mySequence %= 0xFF;
106.
107.    return_status = call SendToMAC.send(Next_Addr, &SendMsg, sizeof(Route_Msg));
108.    if (return_status == SUCCESS)
109.        insertMSGtoRTable(&SendMsg);
110.
111.    return return_status;
112. }

```

```

113.
114.  command error_t SendFromAPP.cancel(message_t* msg){
115.  return call SendToMAC.cancel(msg);
116.  }
117.
118.  command uint8_t SendFromAPP.maxPayloadLength(){
119.  return call SendToMAC.maxPayloadLength();
120.  }
121.
122.  command void* SendFromAPP.getPayload(message_t* msg){
123.  return call SendToMAC.getPayload(msg);
124.  }
125.
126.  event void SendToMAC.sendDone(message_t* msg, error_t error) {
127.  signal SendFromAPP.sendDone(msg, error);
128.  }
129.
130.  ////////////////////////////////////////////
131.
132.  task void RecvToAPP_task(){
133.  memcpy(&NWKF, call SendToMAC.getPayload(&RecvMsg), sizeof(Route_Msg));
134.  signal RecvToAPP.receive(&RecvMsg, (void *)&(NWKF.AppData), sizeof(Route_Msg));
135.  }
136.
137.  task void Forwarding_task(){
138.  if (call SendToMAC.send(Next_Addr, &ForwardMsg[Forward_Buff_Index], sizeof(Route_Msg)) == SUCCESS)
139.  insertMSGtoRTTable(&ForwardMsg[Forward_Buff_Index]);
140.  }
141.
142.  event message_t* RecvFromMAC.receive(message_t* msg, void* payload, uint8_t len) {
143.
144.  Route_Msg *pack = (Route_Msg *) call SendToMAC.getPayload(msg);
145.
146.  if (pack->FinalDstAddr == TOS_NODE_ID) {
147.
148.  #ifndef SHOW_OVERLAP_PACKET
149.  if (!isRecvPrevious(msg))
150.  #endif
151.  {
152.  memcpy((void*)&RecvMsg, (void*)msg, sizeof(message_t));
153.  insertMSGtoRTTable(msg);

```

```

154.     post RecvToAPP_task();
155. }
156.
157. }else{
158.     if (!isRecvPrevious(msg)){
159.         if (pack->TTL>0) {
160.             pack->TTL--;
161.
162.             Forward_Buff_Index++;
163.             Forward_Buff_Index %= MAX_Forward_Buff;
164.             memcpy((void*)&ForwardMsg[Forward_Buff_Index], (void*)msg, sizeof(message_
t));
165.
166.             // Change Route Field
167.             {
168.                 Route_Msg Forward_NWKF;
169.
170.                 memcpy (&Forward_NWKF, call SendToMAC.getPayload(&ForwardMsg[Forward_Buff
_Index]), sizeof(Route_Msg));
171.                 Forward_NWKF.FrameControl = ForwardDataFrame;
172.                 if (Forward_NWKF.Dst2_for_multihop == UnknownAddr){
173.                     Forward_NWKF.OrigSrcAddr = TOS_NODE_ID;
174.                 }else{
175.                     Forward_NWKF.Dst3_for_multihop = Forward_NWKF.Dst2_for_multihop;
176.                     Forward_NWKF.OrigSrcAddr = TOS_NODE_ID;
177.                 }
178.                 memcpy (call SendToMAC.getPayload(&ForwardMsg[Forward_Buff_Index]), &Forw
ard_NWKF, sizeof(Route_Msg));
179.             }
180.
181.             post Forwarding_task();
182.         }
183.     }
184. }
185. return msg;
186. }
187.
188. command void* RecvToAPP.getPayload(message_t* msg, uint8_t* len){
189. return call RecvFromMAC.getPayload(msg, len);
190. }
191. command uint8_t RecvToAPP.payloadLength(message_t* msg){
192. return call RecvFromMAC.payloadLength(msg);
193. }
194.

```

```

195.  event void Timer.fired() {}
196. }

```

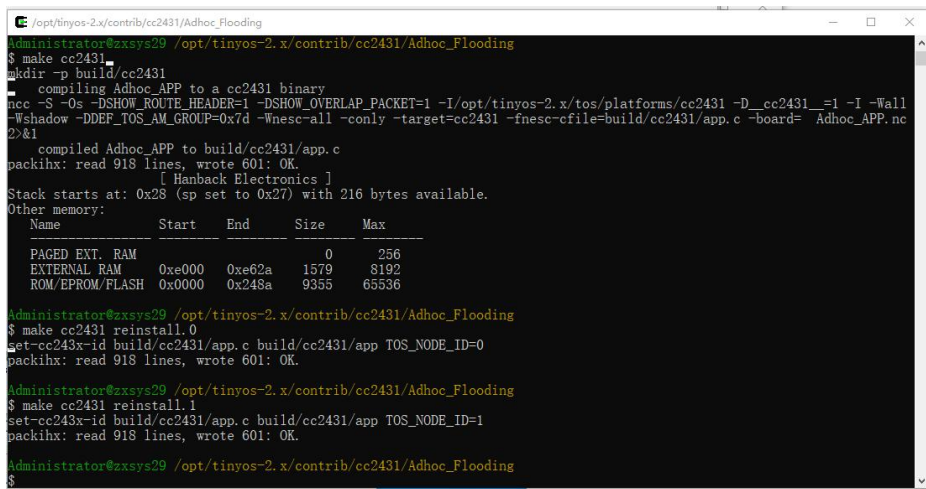
#### 4.1.4 运行结果

首先运行 `cygwin`。如下进行输入，移动到例题文件夹中。

```
cd /opt/tinyos-2.x/contrib/cc2431cd Adhoc_Flooding
```

输入 `make cc2431` 进行编译。

使用 `make cc2431 reinstall.X` 命令生成从 0 号到想要编号的 hex 文件：



```

Administrator@zxsys29 /opt/tinyos-2.x/contrib/cc2431/Adhoc_Flooding
$ make cc2431
mkdir -p build/cc2431
compiling Adhoc_APP to a cc2431 binary
ncc -S -Os -DSHOW_ROUTE_HEADER=1 -DSHOW_OVERLAP_PACKET=1 -I/opt/tinyos-2.x/tos/platforms/cc2431 -D_cc2431_=1 -I-Wall -Wshadow -DDEF_TOS_AM_GROUP=0x7d -Wnesc-all -conly -target=cc2431 -fnesc-cfile=build/cc2431/app.c -board= Adhoc_APP.nc 2>&1
compiling Adhoc_APP to build/cc2431/app.c
packihx: read 918 lines, wrote 601: OK.
[ Hanback Electronics ]
Stack starts at: 0x28 (sp set to 0x27) with 216 bytes available.
Other memory:
  Name          Start      End      Size      Max
  -----
  PAGED EXT. RAM 0xe000    0xe62a    1579    8192
  EXTERNAL RAM   0xe000    0xe62a    1579    8192
  ROM/EPROM/FLASH 0x0000    0x248a    9355   65536

Administrator@zxsys29 /opt/tinyos-2.x/contrib/cc2431/Adhoc_Flooding
$ make cc2431 reinstall.0
set-cc243x-id build/cc2431/app.c build/cc2431/app TOS_NODE_ID=0
packihx: read 918 lines, wrote 601: OK.

Administrator@zxsys29 /opt/tinyos-2.x/contrib/cc2431/Adhoc_Flooding
$ make cc2431 reinstall.1
set-cc243x-id build/cc2431/app.c build/cc2431/app TOS_NODE_ID=1
packihx: read 918 lines, wrote 601: OK.

Administrator@zxsys29 /opt/tinyos-2.x/contrib/cc2431/Adhoc_Flooding
$

```

图 1 编译

将编译后生成的 hex 文件下载到 HBE-Ubi-CC2431 节点中：

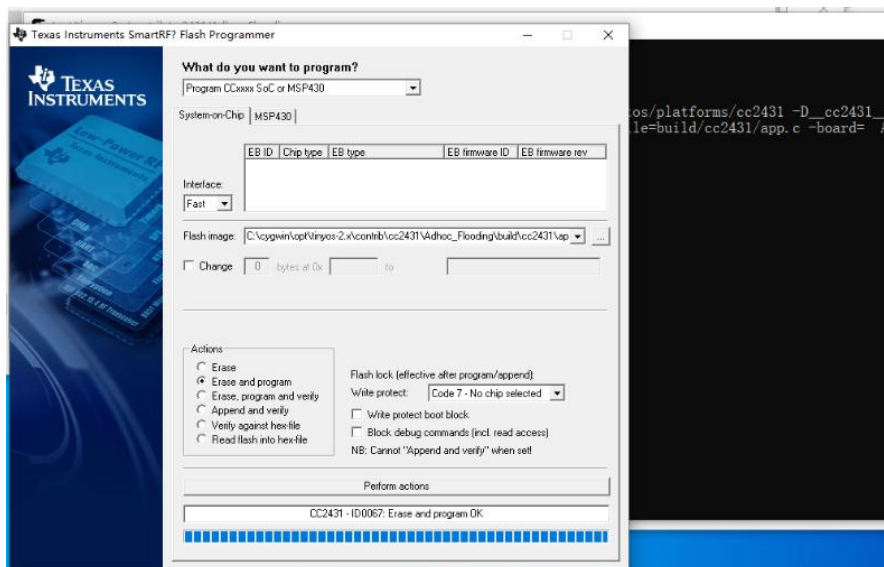


图 2 烧写程序

可以看到串口通信工具已经接收到了数据和原地址，说明这一部分的实验已经成功：

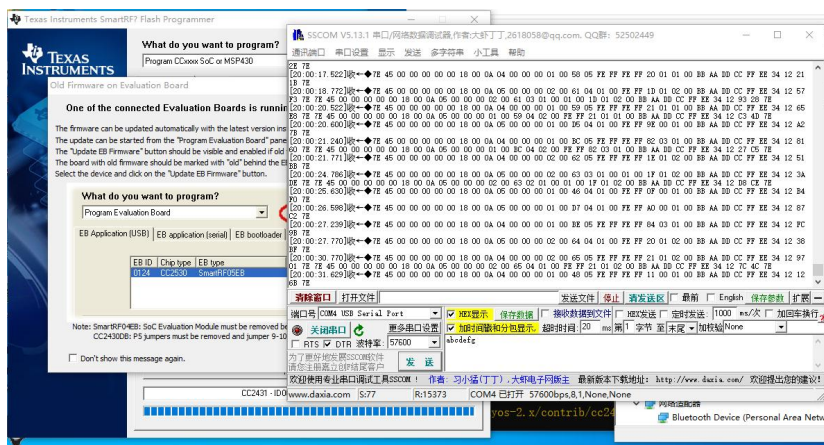


图 3 串口通信

### 4.1.5 问题分析

(1) 在实验过程中，曾经遇到无法接收到串口信号的问题，在烧入过程和接收过程中，都出现过找不到串口的情况。烧录时需将拨动开关置于 MCU 模式并确保烧录目录下的 `app_0.hex`（普通节点）和 `app_1.hex`（汇聚节点）文件正确；接收数据时则需切换至 232 模式。若烧录文件选择错误或拨键位置不当，将导致功能异常。

(2) 在接收数据的时候，会接收到别人的非 sink 节点传来的消息，根源在于节点地址重复。当多设备使用默认地址（如 0、1）时会产生冲突，解决方案是将节点地址改为非常用值（如 5、9），通过地址隔离避免信号干扰。

(3) 在程序中发出的消息是 AABB,CCDD,EEFF,1234。但是在串口中接收到的数据则使 BBAA,DDCC,FFEE,3412。在韩博节点中 uint16\_t 类型的数据使用小端模式即高字节保存在高地址中，节点（ARM 小端模式）将高字节存高地址（0xAB 存高位），而 PC（大端模式）将高字节存低地址，使得 0xAABB 在传输中被解析为 0xBBAA。

(4) 在串口工具中可能会看到数据为乱码, 需要打开工具的 HEX 显示选项, 以正常显示十六进制, 打开后显示正常。

## 4.2 传感器节点与射频识别的实验例程

### 4.2.1 功能描述

该模块是一个基于无线传感器网络（如 Zigbee）的 RFID 读写控制系统，兼容 ISO 14443A 和 ISO 15693 协议，支持读取标签 ID、读写数据块，通过无线网络与中心节点（Sink）交互指令和上报数据，通过串口接收外部指令或输出调



试信息，通过 LED 灯实时反映系统状态（如数据发送、接收、错误等）。

#### （1）RFID 操作控制

相关命令：

CMD\_GetID\_14443A：读取 ISO 14443A 标签的 ID（5 字节）。

CMD\_GetID\_15693：读取 ISO 15693 标签的 ID（8 字节）。

CMD\_RData\_15693：读取 ISO 15693 标签的指定数据块（4 字节）。

CMD\_WData\_15693：向 ISO 15693 标签的指定数据块写入数据（4 字节）。

触发流程：

通过无线消息（RFID\_COMM\_MSG）或串口指令触发。执行完成后，通过异步事件（如 GetID\_14443A\_Done）返回状态和数据。

#### （2）无线通信

普通节点将 RFID 操作结果封装为 RFID\_DATA\_MSG 数据帧，发送至中心节点（SinkAddress）。数据帧包含节点 ID、操作类型、状态码和实际数据。中心节点通过无线发送 RFID\_COMM\_MSG 指令帧（含操作类型、数据块号、写入数据）。普通节点接收后解析并执行对应 RFID 操作。

#### （3）串口交互

串口数据以 0x7E 作为帧起始符，固定长度为 8 字节。接收完成后，触发无线广播发送（SendToReader 任务）。中心节点通过串口输出格式化信息，如标签 ID、读写状态、错误码等。

#### （4）状态指示

LED0：接收到串口指令或无线消息时 切换状态（led0Toggle）。

LED1：数据发送时 亮起（led1On），发送完成后 关闭（led1Off）。

LED2：无线消息到达时 切换状态（led2Toggle）。

### 4.2.2 主要函数

Print\_MSG\_AccordingTo\_CMD：输入命令类型、状态码（成功/失败）、数据缓冲区（标签 ID 或读取的数据）。根据命令类型和状态码，生成对应的调试信息（如标签 ID、错误提示），并通过串口输出。标签 ID 以十六进制格式显示，如 Recv 14443A ID: [AA BB CC DD EE]。

UARTSend: Print\_MSG\_AccordingTo\_CMD 函数调用了接口 SCSuartDBG 的 UARTSend 函数。SCSuartDBG 是一个串口通信管理模块，其通过 SCSuartDBG.UARTSend 接口发送数据到串口，支持缓冲区管理；通过 HPLUART.get 事件接收串口数据，并触发 SCSuartDBGRecv.UARTRecv 回调；当发送缓冲区为空时，通过 SCSuartDBGIdle.UARTIdle 通知空闲状态。函数功能：将外部传入的

数据按字节存入环形缓冲区 `str_buff`，维护 `buff_start`（发送位置）和 `buff_end`（写入位置）指针，自动处理缓冲区越界。若串口处于空闲状态（`state == IDLE_UART`），立即调用 `HPLUART.put()` 发送缓冲区的第一个字节，并标记为忙碌状态（`BUSY_UART`）。

**UARTRecv:** 以 `0x7E` 作为起始符，触发接收状态。按顺序填充 `RecvBuff`，直到达到 `RecvBuffSize`（8 字节）。完成后切换 `LED0` 状态，指示帧接收完成，将数据复制到无线消息负载，通过 `post` 提交 `SendToReader` 任务广播，重置 `start_flag`，准备接收下一帧。

### 4.2.3 程序注释

#### RFID\_Control

```

1. // 定义 testRFIDwithUSNM 模块
2. module testRFIDwithUSNM
3. {
4.     // 提供的接口
5.     provides interface StdControl; // 标准控制接口，用于启动和停止模块
6.     // 使用的接口
7.     uses {
8.         interface Leds; // LED 控制接口
9.         interface RFID_Control; // RFID 控制接口
10.        interface Boot; // 启动接口
11.        interface StdControl as SCSuartSTD; // 标准串行通信接口
12.        interface SCSuartDBG; // 调试串行通信接口
13.        interface SCSuartDBGRecv; // 调试串行通信接收接口
14.        interface SplitControl as CommControl; // 通信控制接口
15.        interface AMSend as DataMsg; // 数据发送接口
16.        interface Receive as RecvMsg; // 消息接收接口
17.    }
18. }
19. implementation{
20.    // 定义消息缓冲区和相关变量
21.    message_t RF_MSG; // RFID 消息
22.    uint8_t OutputUartMsg[64]; // 输出到串口的消息缓冲区
23.    char Inc_Flag; // 增加标志
24.
25.    // 根据接收到的命令类型打印消息
26.    void Print_MSG_AccordingTo_CMD(uint8_t RecvCMDType, uint8_t status, uint8_t *buff);
27.
28.    // 启动事件，当系统启动时触发
29.    event void Boot.booted() {

```

```

30.     call CommControl.start(); // 启动通信控制
31. }
32.
33. // 通信控制启动完成事件
34. event void CommControl.startDone(error_t error) {
35.     call SCSuartSTD.start(); // 启动标准串行通信
36.     call RFID_Control.start(); // 启动 RFID 控制
37. }
38.
39. // 通信控制停止完成事件
40. event void CommControl.stopDone(error_t error) {
41. }
42.
43. ///////////////////////////////////////////////////
44.
45. // 根据接收到的串口指令调用相应的 RFID 函数
46. void Control_RFID(uint8_t comm, uint8_t block, uint8_t* buff) {
47.     if (comm == 1) { // 如果指令是获取 14443A 标签的 ID
48.         call RFID_Control.GetID_14443A(); // 调用获取 14443A 标签 ID 的函数
49.     } else if (comm == 2) { // 如果指令是获取 15693 标签的 ID
50.         call RFID_Control.GetID_15693(); // 调用获取 15693 标签 ID 的函数
51.     } else if (comm == 3) { // 如果指令是读取 15693 标签的数据
52.         call RFID_Control.RData_15693(block); // 调用读取 15693 标签数据的函数
53.     } else if (comm == 4) { // 如果指令是写入 15693 标签的数据
54.         call RFID_Control.WData_15693(block, buff, 4); // 调用写入 15693 标签数据的函数
55.     }
56. }
57.
58. ///////////////////////////////////////////////////
59.
60. // 与串行通信相关的变量
61. #define RecvBuffSize 8 // 接收缓冲区大小
62. uint8_t RecvBuff[RecvBuffSize]; // 接收缓冲区
63. uint8_t recv_num = 0; // 接收到的字节数
64. bool start_flag = 0; // 开始标志
65.
66. // 数据发送完成事件
67. event void DataMsg.sendDone(message_t* msg, error_t error) {
68.     if (error == SUCCESS) // 如果发送成功
69.         call Leds.led10ff(); // 关闭 LED1
70. }
71.

```

```

72. // 消息接收事件
73. event message_t* RecvMsg.receive(message_t* msg, void* payload, uint8_t len) {
74.     call Leds.led2Toggle(); // 切换 LED2 状态
75.     if (TOS_NODE_ID == SinkAddress) { // 如果当前节点是汇聚节点
76.         RFID_DATA_MSG *pack = (RFID_DATA_MSG *) payload; // 获取数据消息结构
77.         Print_MSG_AccordingTo_CMD(...); // 根据命令类型打印消息
78.     } else {
79.         RFID_COMM_MSG *pack = (RFID_COMM_MSG *) payload; // 获取通信消息结构
80.         Control_RFID(pack->comm, pack->block, pack->wbuf); // 调用相应的 RFID 函数
81.     }
82.     return msg; // 返回消息
83. }
84.
85. ///////////////////////////////////////////////////
86.
87. // 将接收到的串口数据发送到读卡器的任务
88. task void SendToReader() {
89.     if (call DataMsg.send(AM_BROADCAST_ADDR, &RF_MSG, recv_num) == SUCCESS) { // 尝试发送数据
90.         call Leds.led1On(); // 如果发送成功, 打开 LED1
91.     }
92. }
93.
94. // 串行通信接收事件
95. async event void SCSuartDBGRecv.UARTRecv(uint8_t recv_Char) {
96.     if (recv_Char == 0x7E && start_flag == 0) { // 如果接收到帧头且未开始接收
97.         recv_num = 0; // 重置接收字节数
98.         start_flag = 1; // 设置开始标志
99.     } else if (recv_num < RecvBuffSize && start_flag == 1) { // 如果接收字节数未超过缓冲区大小且已开始接收
100.         RecvBuff[recv_num] = recv_Char; // 将接收到的字节存储到缓冲区
101.         recv_num++; // 增加接收字节数
102.         if (recv_num == RecvBuffSize) { // 如果接收完成
103.             call Leds.led0Toggle(); // 切换 LED0 状态
104.             memcpy(call DataMsg.getPayload(&RF_MSG), ...); // 将接收到的数据复制到消息负载
105.             post SendToReader(); // 触发发送任务
106.             start_flag = 0; // 清除开始标志
107.         }
108.     }
109. }
110.

```

```

111.  //////////////////////////////////////
112.
113.  // 将数据发送到汇聚节点的任务
114.  task void SendToSinkAddress() {
115.      if (call DataMsg.send(SinkAddress, &RF_MSG, sizeof(RFID_DATA_MSG)) =
= SUCCESS) { // 尝试发送数据
116.          call Leds.led1On(); // 如果发送成功, 打开 LED1
117.      }
118.  }
119.
120.  // RFID 相关的异步事件
121.  async event void RFID_Control.GetID_14443A_Done(char status, uint8_t *buf
f, char size) {
122.      // ...
123.  }
124.
125.  async event void RFID_Control.GetID_15693_Done(char status, uint8_t *buf
f, char size) {
126.      // ...
127.  }
128.
129.  async event void RFID_Control.RData_15693_Done(char status, uint8_t *buf
f, char size) {
130.      // ...
131.  }
132.
133.  async event void RFID_Control.WData_15693_Done(char status) {
134.      // ...
135.  }
136.
137.  //////////////////////////////////////
138.
139.  // 根据接收到的命令类型打印消息的函数
140.  void Print_MSG_AccordingTo_CMD(uint8_t RecvCMDType, uint8_t status, uint8_
t *buff) {
141.      uint8_t i;
142.      switch (RecvCMDType) {
143.          case CMD_GetID_14443A: // 如果命令类型是获取 14443A 标签 ID
144.              if (status == 0) { // 如果操作成功
145.                  sprintf(OutputUartMsg, "Recv 14443A ID: ["); // 打印接收到
的 14443A 标签 ID
146.                  // ...
147.              } else { // 如果操作失败

```

```

148.             sprintf(OutputUartMsg, "14443A GetID Error: %d\r\n", statu
s); // 打印错误信息
149.             // ...
150.         }
151.         break;
152.         case CMD_GetID_15693: // 如果命令类型是获取 15693 标签 ID
153.             if (status == 0) { // 如果操作成功
154.                 sprintf(OutputUartMsg, "Recv 15693 ID: ["); // 打印接收到
的 15693 标签 ID
155.                 // ...
156.             } else { // 如果操作失败
157.                 sprintf(OutputUartMsg, "15693 GetID Error: %d\r\n", statu
s); // 打印错误信息
158.                 // ...
159.             }
160.             break;
161.         case CMD_RData_15693: // 如果命令类型是读取 15693 标签数据
162.             if (status == SUCCESS) { // 如果操作成功
163.                 sprintf(OutputUartMsg, "Read data from 15693: Data["); /
/ 打印读取到的数据
164.                 // ...
165.             } else { // 如果操作失败
166.                 sprintf(OutputUartMsg, "Recv RData 15693 Error: %d\r\n", s
tatus); // 打印错误信息
167.                 // ...
168.             }
169.             break;
170.         case CMD_WData_15693: // 如果命令类型是写入 15693 标签数据
171.             if (status == SUCCESS) { // 如果操作成功
172.                 sprintf(OutputUartMsg, "Write data to 15693 SUCCESS!!!\r\n
"); // 打印成功信息
173.                 // ...
174.             } else { // 如果操作失败
175.                 sprintf(OutputUartMsg, "Write data to 15693 Error: %d\r\n
", status); // 打印错误信息
176.                 // ...
177.             }
178.             break;
179.         }
180.     }
181. }

```

## 4.2.4 运行结果

首先运行 cygwin。如下进行输入，移动到例题文件夹中。

```
cd /opt/tinyos-2.x/contrib/cc2431
```

```
cd TestRFIDwithRFID
```

输入 make cc2431 编译后，通过 make cc2431 reinstall.X 指令生成具有 0 号和 1 号 ID 的 hex 文件，将编译后生成的 hex 文件下载到 HBE-Ubi-CC2431 节点中：

```
Administrator@zxsys16 ~
$ cd /opt/tinyos-2.x/contrib/cc2431
Administrator@zxsys16 /opt/tinyos-2.x/contrib/cc2431
$ cd TestRFIDwithUSNC
Administrator@zxsys16 /opt/tinyos-2.x/contrib/cc2431/TestRFIDwithUSNC
$ make cc2431
mkdir -p build/cc2431
compiling testRFIDwithUSNC to a cc2431 binary
gcc -S -Os -I/opt/tinyos-2.x/tos/platforms/cc2431 -I. -D__cc2431__=1 -I -Wall -Wshadow -DDEF_TOS_AM_GROUP=0x77 -Wnesc-all -only -target=cc2431 -fnesc-cfile=build/cc2431/app.c -board= testRFIDwithUSNC.nc 2>&1
testRFIDwithUSNC.nc:182: warning: 'DataMsg.getPayload' called asynchronously from 'RFID_Control.WData_15693_Done'
testRFIDwithUSNC.nc:171: warning: 'DataMsg.getPayload' called asynchronously from 'RFID_Control.RData_15693_Done'
testRFIDwithUSNC.nc:159: warning: 'DataMsg.getPayload' called asynchronously from 'RFID_Control.GetID_15693_Done'
testRFIDwithUSNC.nc:147: warning: 'DataMsg.getPayload' called asynchronously from 'RFID_Control.GetID_14443A_Done'
testRFIDwithUSNC.nc:124: warning: 'DataMsg.getPayload' called asynchronously from 'SCSuartDBGRecv.UARTRecv'
compiled testRFIDwithUSNC to build/cc2431/app.c
packihx: read 1389 lines, wrote 876: OK.
[ Hanback Electronics ]
Stack starts at: 0x2f (sp set to 0x2e) with 209 bytes available.
Other memory:
  Name          Start      End      Size      Max
  -----
  PAGED EXT. RAM          0          0          256
  EXTERNAL RAM    0xe000    0xe4d2    1235    8192
  ROM/EPROM/FLASH 0x0000    0x35a9   13738   65536
Administrator@zxsys16 /opt/tinyos-2.x/contrib/cc2431/TestRFIDwithUSNC
```

图 4 烧写程序

为了控制 RFID，需要利用位于相同文件夹中的 Serial\_Parsing.c。如下进行输入，编译 Serial\_Parsing.c。(编译前，一定要检查 Serial\_Parsing.c 文件中的 #define MODEMDEVICE "/dev/ttyS0" 语句是否与当前连接的 COM 编号相符。)

```
Administrator@zxsys16 /opt/tinyos-2.x/contrib/cc2431/TestRFIDwithUSNC
$ make cc2431 reinstall.0
set-cc243x-id build/cc2431/app.c build/cc2431/app TOS_NODE_ID=0
packihx: read 1389 lines, wrote 876: OK.
Administrator@zxsys16 /opt/tinyos-2.x/contrib/cc2431/TestRFIDwithUSNC
$ make cc2431 reinstall.1
set-cc243x-id build/cc2431/app.c build/cc2431/app TOS_NODE_ID=1
packihx: read 1389 lines, wrote 876: OK.
Administrator@zxsys16 /opt/tinyos-2.x/contrib/cc2431/TestRFIDwithUSNC
$ gcc -o run Serial_Parsing.c
Administrator@zxsys16 /opt/tinyos-2.x/contrib/cc2431/TestRFIDwithUSNC
$ ./run.exe
Start serial program [Changsu Suh]
/dev/ttyS1: No such file or directory
Administrator@zxsys16 /opt/tinyos-2.x/contrib/cc2431/TestRFIDwithUSNC
$ gcc -o run Serial_Parsing.c
Administrator@zxsys16 /opt/tinyos-2.x/contrib/cc2431/TestRFIDwithUSNC
$ ./run.exe
Start serial program [Changsu Suh]
insert RFID card TYPE ('a'=14443A, 'b'=15693): b
your choice is 15693 true
```

图 5 编译

成功编译后，执行 ./run.exe:



```

/opt/tinyos-2.x/contrib/cc2431/TestRFIDwithUSN
administrator@zxsys29 /opt/tinyos-2.x/contrib/cc2431/TestRFIDwithUSN
./run.exe
start serial program [Changsu Suh]
dev/ttyS3: No such file or directory

administrator@zxsys29 /opt/tinyos-2.x/contrib/cc2431/TestRFIDwithUSN
./run.exe
start serial program [Changsu Suh]
dev/ttyS3: Permission denied

administrator@zxsys29 /opt/tinyos-2.x/contrib/cc2431/TestRFIDwithUSN
./run.exe
start serial program [Changsu Suh]

insert RFID card TYPE ('a'=14443A, 'b'=15693): 14443A
our choice is 15693 type
insert CMD ('i'=getID, 'r'=readData, 'w'=writeData, 'x'=exit): i
our choice: i

insert RFID card TYPE ('a'=14443A, 'b'=15693): a
our choice is 14443A type, This card support getID only
insert CMD ('any key'=getID, 'x'=exit): i

insert RFID card TYPE ('a'=14443A, 'b'=15693): 14443A GetID Error: 1
5693 GetID Error: 1
recv 15693 ID: [E0 4 1 0 38 BE 7F A ]

```

图 6 显示 RFID 信息

## 4.2.5 问题分析

(1) 实验操作需严格执行以下流程：烧录 `app_0.hex`（普通节点）和 `app_1.hex`（汇聚节点）后，将 Sink 节点连接 PC，读卡器接入 5V 独立电源。必须在 `Boot.booted()` 事件函数中显式添加 `call RControl.start()` 启动射频通信模块，否则节点间无法建立无线连接。缺少此关键步骤将导致整个通信链路中断。

(2) 编译前必须验证 `Serial_Parsing.c` 中的设备定义：`#define MODEMDEV ICE "/dev/ttyS0"` 需对应实际 COM 端口。转换规则：COM 编号减 1 得 ttyS 索引值（如 COM4→ttyS3）。若配置错误（如 COM4 仍用 ttyS0），执行 `./run` 时将报错“串口设备不存在”，导致数据接收功能完全失效。

(3) 在接有 RFID 读卡器的节点烧写 hex 程序时需要在断电时将 RFID 读卡器取下，否则会因为供电不足导致无法正常烧写，烧写完成后断电再连接读卡器。

## 4.3 数据采集和标签信息的洪泛传输

### 4.3.1 设计思路

本次课程设计的核心需求是采集温度、湿度、光照度以及 RFID 标签的读写数据。通过参考实习 13 和 17 的内容，我们已掌握数据采集的基本流程及 RFID 标签的读写方法。对于温湿度和光照值的读取，起初并不熟悉具体实现方式，但在查阅指导书的前段部分后，发现可直接复用相关代码片段，仅需进行少量参数调整即可适配需求。值得注意的是，题目要求采用不同频率采集各类数据：温湿度每 4 秒一次、光照值每 2 秒一次、RFID 标签每 3 秒一次。为此我们设计了一



个定时器与计数器的方案，通过计数器对时间取模运算实现精确的定时采集功能。

在代码实现层面，我们引入了头文件 `_ADHOC_APP_H_`，设置感知节点地址为 `Sink_Addr 0`（节点号 5），并包含 `RFID_Control` 等必要组件。具体配置如下：

温湿度组件： `new SensirionSht11C()`

光照组件： `new PhotoSensorC()`

RFID 组件： `RFID_ControlC`

同时声明了三个读取接口：

`interface Read<uint16_t> as Read_Humi;`

`interface Read<uint16_t> as Read_Temp;`

`interface Read<uint16_t> as Read_Photo;`

并编写了相应的数据处理函数。

Python 端采用串口通信模板进行改造，界面开发使用 PySide6 库的 Qt Designer 工具。得益于前期课程设计积累的串口通信经验，这部分实现较为顺利。相比之下，基于 nesC 语言的底层代码修改更具挑战性，主要修改集中在 `Adhoc_FloodingM` 和 `Adhoc_FloodingC` 两个模块中，新增了温湿度、光照值读取以及 RFID 标签读写功能。

### 4.3.2 流程图

#### (1) FloodingM

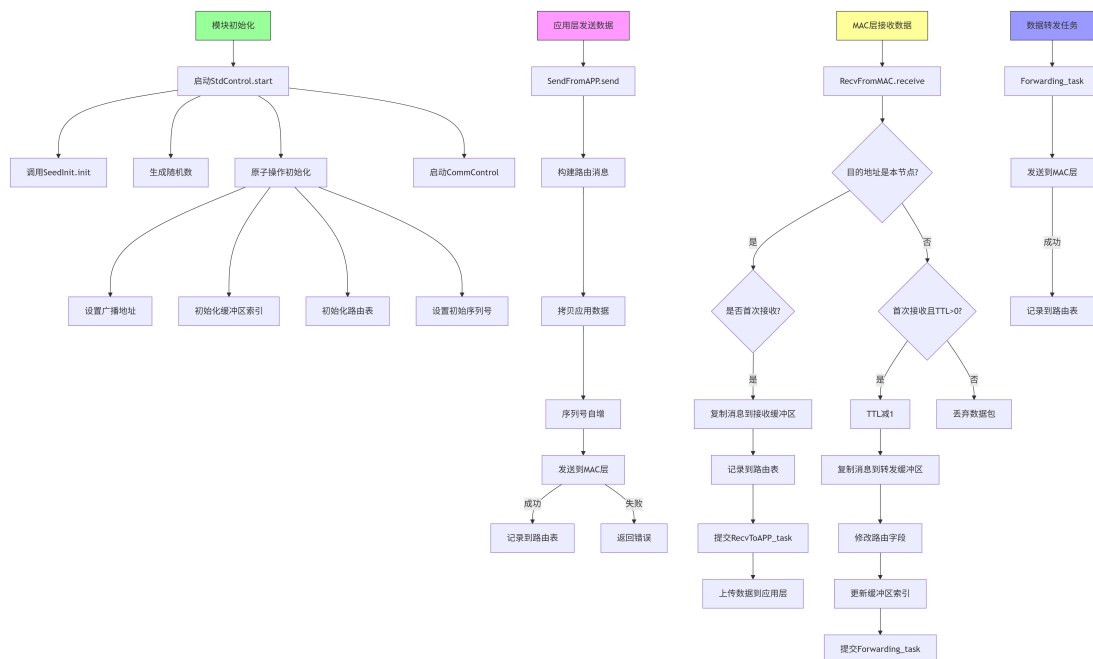


图 7 模块流程图

## (2) Adhoc\_APPM

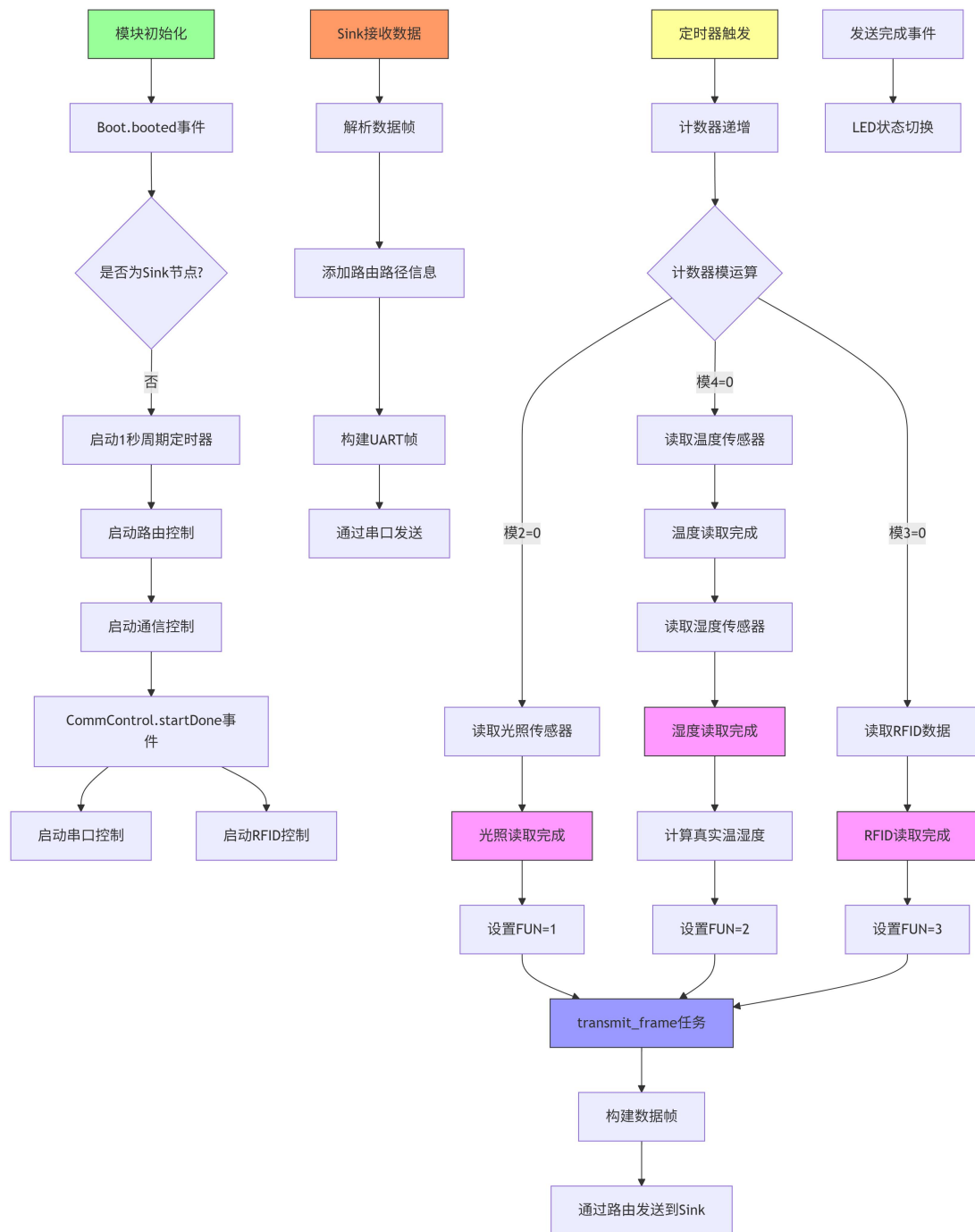


图 8 模块流程图

## (3) Server

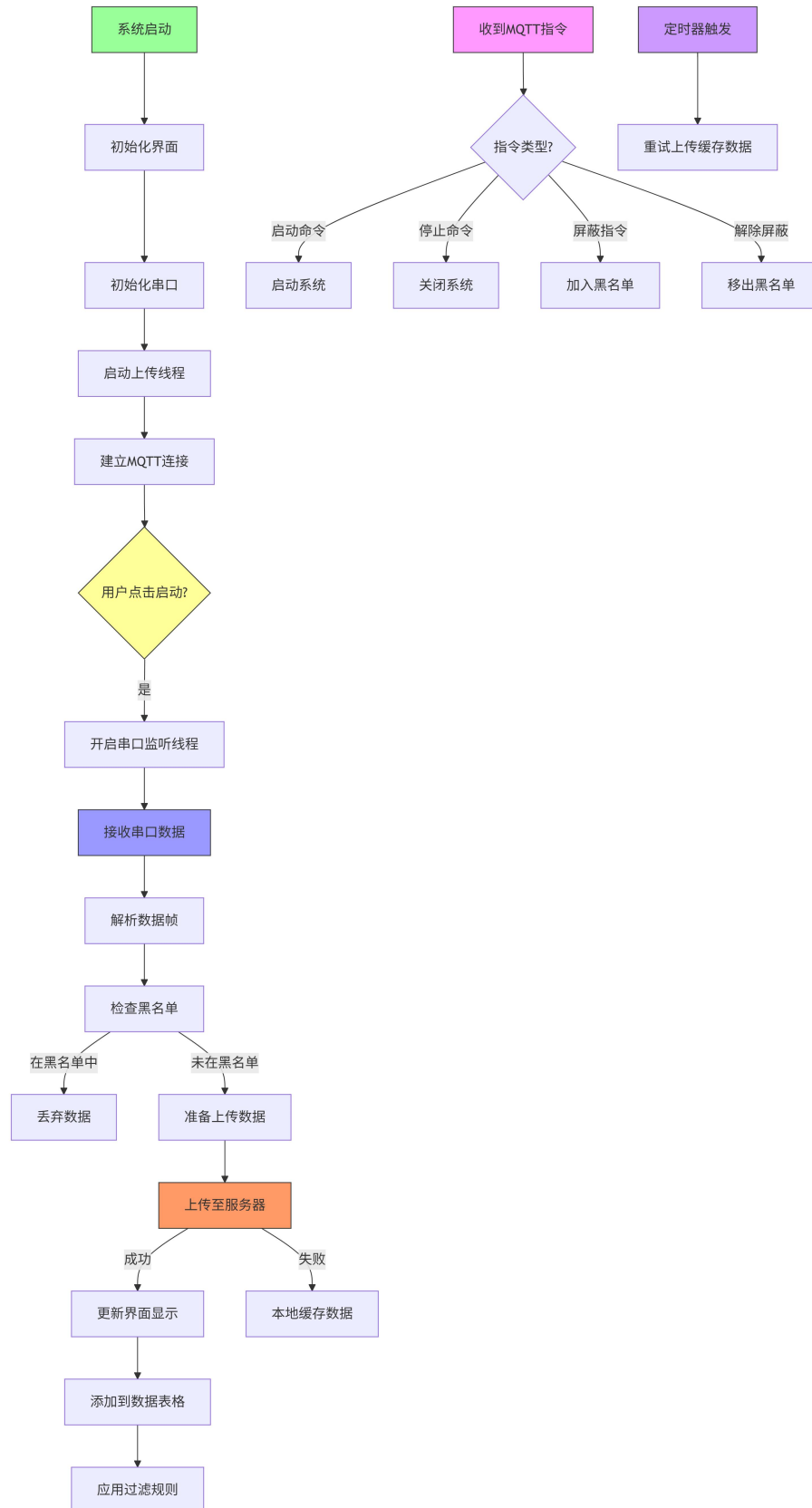


图 9 PC 端流程图

### 4.3.3 运行结果

我们用 Qt-Designer 绘制的 UI 界面如下，可以通过串口实时更新温湿度、光照、标签数据，每个数据帧的原地址、目的地址、Seq、Cardata 和类型：

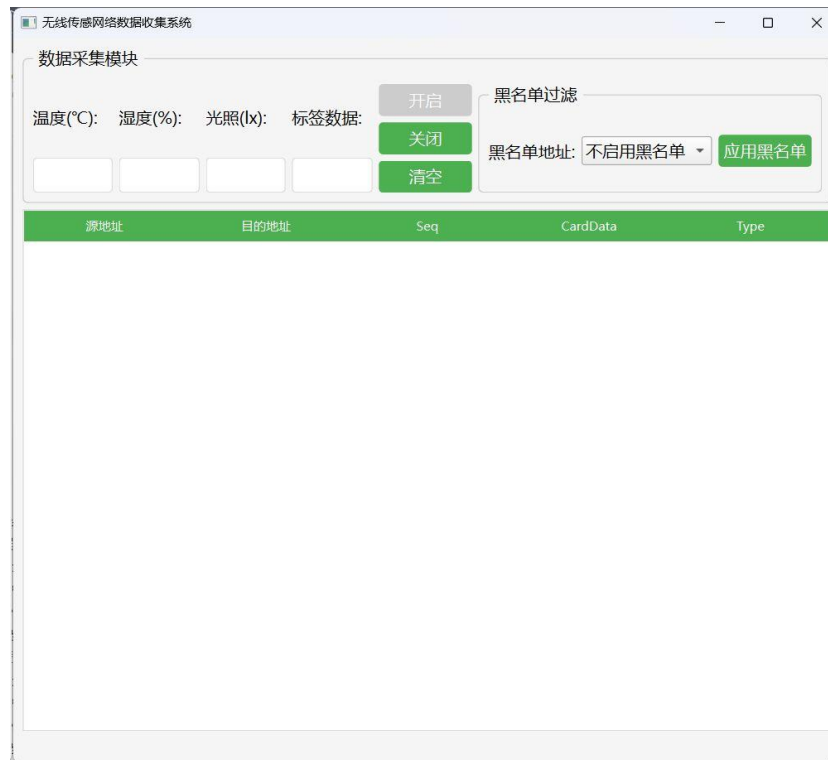


图 10 UI 界面

从串口接收到数据帧后，通过 API 接口将数据包上传至远程服务器数据库：

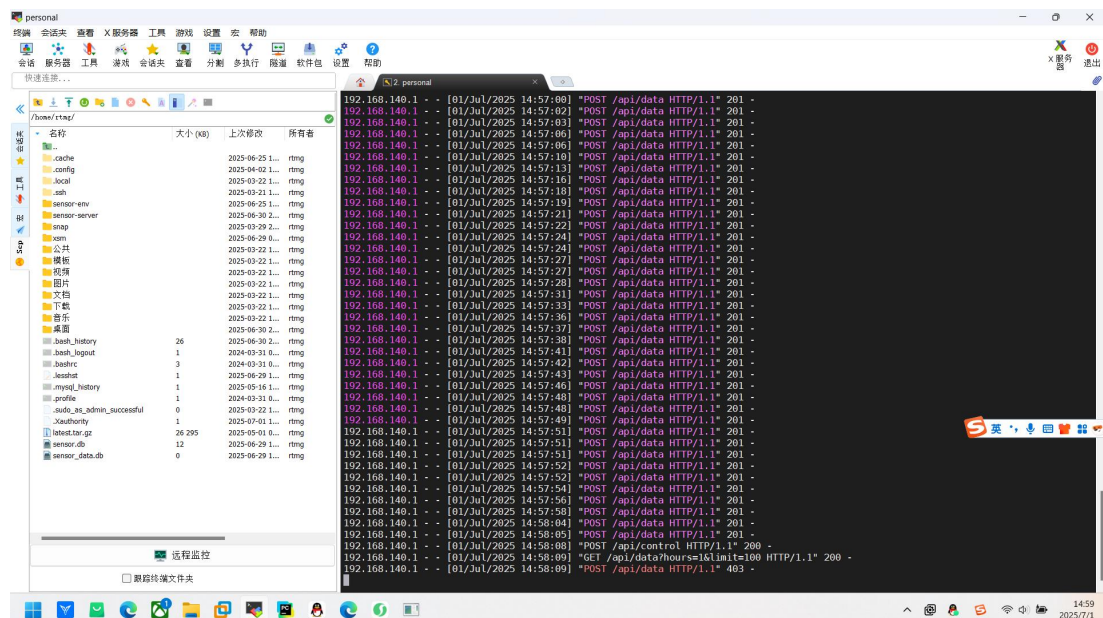


图 11 与远程 PC 通信

此时在远程 Web 端可以实时更新采集的数据:

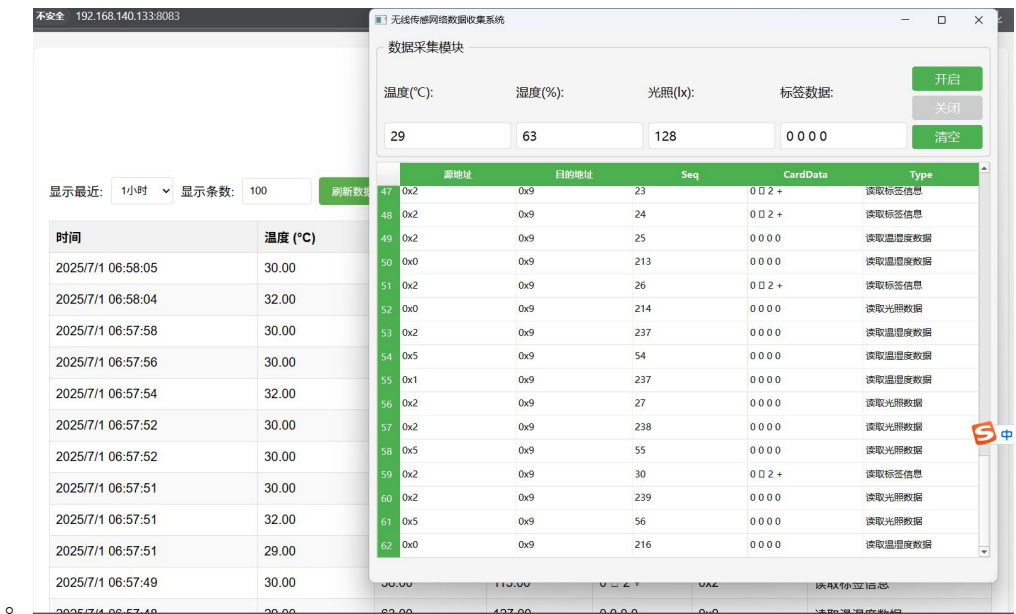


图 12 数据采集

4.3.4 问题分析

- (1) 在设计过程中，我们遇到了不少技术挑战。最初在硬件操作方面就遇到了困难，由于不熟悉设备结构，不知道如何将程序烧录进读卡器，经过多次尝试才发现需要先移除上盖板才能进行烧录操作。
- (2) 在实验步骤安排上，我们采取了由易到难的顺序：首先实现温湿度传感器的数据读取，接着完成光照传感器的数据采集，最后攻克 RFID 卡的数据读写。在初期数据采集时，我们发现传感器返回的数值存在异常，出现了负数或明显不合理的数值。通过深入研究相关技术文档，我们发现这些原始数据需要经过循环冗余校验（CRC）处理才能得到准确值。其中，温湿度和光照传感器的数据读取相对简单，参考技术手册就能快速实现；而 RFID 读写则较为复杂，需要仔细分析 TestRFIDwithsUSN 示例代码。最终我们发现，只要按照示例中的数据结构构造和数据分析方法来处理发送的数据，就能成功实现 RFID 功能。
- (3) 在组网测试环节，我们搭建了由三个节点组成的测试环境：Sink 节点作为数据接收端，节点 1 作为数据发送端，节点 2 作为中继节点。通过实验验证了网络洪泛机制：当节点 1 和 Sink 节点之间加入节点 2 时，数据可以通过中继跳转传输；若移除节点 2，数据则直接由节点 1 洪泛至 Sink 节点。我们还进行了控制变量实验：当移除节点 1 的天线并同时移除节点 2 时，由于信号覆盖范围减小，Sink 节点无法接收数据；但如果将改装后的节点 1 与节点 2 靠近布置，数据仍可通过节点 2 中继传输至 Sink 节点。
- (4) 在远程功能开发时，我们遇到了如下这些问题：

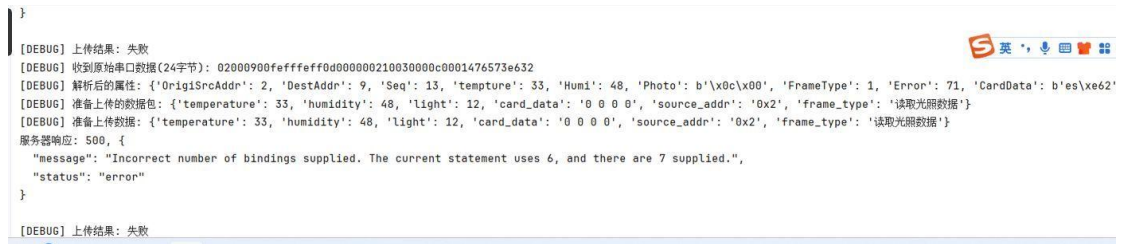


图 13 遇到的问题

日志显示数据上传失败，服务器返回 500 错误，明确提示："Incorrect number of bindings supplied. The current statement uses 6, and there are 7 supplied."，排查后发现客户端提交的数据字段数量（7 个）与服务器 SQL 语句的绑定参数数量（6 个）不匹配。客户端将 card\_data 从二进制 b'es\xe62'转换为字符串'0 0 0'时，未正确处理格式。后续我们统一了二进制数据转换规则，即使用 `return ''.join(f'{b}' for b in raw_bytes[:4])`，将数据限制为 4 字节。

(5) 此外，我们的数据库存储数据的方式是"data": [

```
{
  "card_data": "7 8 9 6",
  "frame_type": "\u8bf6\u53d6\u6e29\u6e7f\u5ea6\u6570\u636e",
  "humidity": 40.0,
  "id": 537,
  "light": 138.0,
  "source_addr": "0x1",
  "temperature": 38.0,
  "timestamp": "2025-06-30 11:49:19"
},
```

在远程 Web 中发现其中 timestamp 的日期是对的，但时间不对。排查后发现 timestamp 字段在数据库中被定义为 DATETIME DEFAULT CURRENT\_TIMESTAMP，这应该使用 SQLite 的当前时间，当数据通过 POST 请求插入时，没有显式设置时间戳，而是依赖数据库的默认值，后续我们添加了时区转换逻辑，问题解决。

(6) 在运行后没有数据显示，而且有报错：

[DEBUG] 收到原始串口数据(24 字节): 02000900feffff7f00000021003100180001476573e632

[DEBUG] 解析后的属性: {'OrigSrcAddr': 2, 'DestAddr': 9, 'Seq': 127, 'temperature': 33, 'Humi': 49, 'Photo': b'\x18\x00', 'FrameType': 1, 'Error': 71, 'CardData': b'es\xe62'}

[DEBUG] 准备上传的数据包: {'temperature': 33, 'humidity': 49, 'light': 24, 'card\_data': '0 0 0 0', 'source\_addr': '0x2', 'frame\_type': '读取光照数据'}

[DEBUG] 准备上传数据: {'temperature': 33, 'humidity': 49, 'light': 24, 'card\_data': '0 0 0 0', 'source\_addr': '0x2', 'frame\_type': '读取光照数据'}

[ERROR] 数据接收处理异常: 'UploadThread' object has no attribute 'blocked\_addresses'

我们发现问题出在 UploadThread 类中缺少 blocked\_addresses 属性的初始化。在 UploadThread 类初始化方法中添加 blocked\_addresses 属性后问题解决。

(7) 当 type 不是读取标签信息时, carddata 会是乱码:

	源地址	目的地址	Seq	CardData	Type
170	0x1	0x9	118	8 8 9 9	读取标签信息
171	0x1	0x9	52	b @ # D	读取光照数据
172	0x1	0x9	119	8 8 9 9	读取标签信息
173	0x1	0x9	53	b @ # D	读取光照数据
174	0x1	0x9	120	8 8 9 9	读取标签信息

图 14 CardData 乱码

后续我们增加了一个判断逻辑, 如过不是读取标签信息, 就将 CardData 设置为 '0000'。

(8) 我们发现关闭再打开应用程序时出现大量数据输出, 这是因为虽然关闭了串口连接, 但串口设备本身可能仍然在接收数据并存储在缓冲区中。当重新打开应用程序时, 这些缓冲的数据会被一次性读取出来。于是我们在关闭串口前, 先清空输入缓冲区:

```
if self.serial_port and self.serial_port.is_open:
    self.serial_port.reset_input_buffer() # 清空输入缓冲区
    self.serial_port.close()
```

可以解决关闭再打开时输出大量数据的问题。

#### 4.3.5 改进方向

后续, 我们考虑优化节点能耗管理, 通过动态调整传感器采样频率 (如空闲时降低 RFID 读取频率) 延长电池寿命; 增强数据安全性, 在洪泛传输中加入轻



量级加密算法防止数据篡改；简化 PC 端界面，采用更直观的图表展示实时数据趋势；完善异常处理机制，自动过滤异常传感器值并标记故障节点，提升系统稳定性。

## 5. 收获和心得体会

通过这次课程设计的实践，我深刻体会到理论知识与工程实践的紧密联系。在一学期的理论学习中，虽然老师对洪泛协议、RFID 标签技术等内容进行了详细讲解，但仅通过课堂学习很难真正理解这些技术在实际系统中的运作机制。课程设计恰好填补了这一空白，让我有机会将抽象的理论转化为具体的工程实现。

在实验初期，我严格按照指导书步骤完成了多跳通信和 RFID 标签读取两个基础实验。这个过程使我对 NesC 语言的模块化设计有了直观认识——其组件化架构虽然提高了代码复用性，但也带来了理解上的挑战。特别是在跟踪程序执行流程时，需要频繁在上下层组件间跳转，这要求开发者必须清晰掌握每个接口的调用关系。通过反复研读 `Adhoc_FloodingM` 和 `RFIDwithUSN` 等核心模块的代码，我逐渐建立起对系统整体运行机制的理解。遇到难以理解的函数时，除了查阅有限的 NesC 文档外，与同学的讨论和老师的指导成为突破瓶颈的关键，这种协作解决问题的经历让我受益匪浅。

在功能扩展阶段，温湿度与光照传感器的数据采集相对顺利，但 RFID 数据读写功能的实现过程充满波折。其中，精确读取小数温度值的功能花费了大量调试时间。实践中遇到的硬件问题尤为棘手：RFID 接口未初始化导致通信失败、`Boot.booted` 事件中遗漏射频启动代码、Sink 节点因长期运行出现性能下降等。通过系统性地排查——从电路连接检查到固件逻辑验证，最终在 CSDN 等技术论坛的案例启发下逐一解决了这些问题。这些经历让我深刻认识到，嵌入式开发不仅需要扎实的编程能力，更需要严谨的工程思维和故障排查技巧。

在数据处理与可视化方面，基于 PyQt6 的 UI 开发得益于前期项目积累显得较为顺利。利用 `PySerial` 库实现的串口通信模块，通过数据切片和类型转换将原始报文解析为结构化信息，最终以动态图表形式展示环境参数。这种将底层硬件数据转化为用户友好界面的完整链路实践，让我对物联网系统的端到端架构有了更立体的认识。

回顾整个项目，最大的收获是对 NesC 语言特性的深入理解。通过研读 Tiny OS 源码和调试实践，我不仅掌握了组件间接口的交互机制，更体会到事件驱动模型在资源受限设备上的优势。同时，对 Flooding 路由协议的实际测试，修正了我原先对洪泛算法效率的片面认知——在小型传感网络中，其实现简单、可靠



性高的特点具有独特价值。此外，RFID 读写过程中的时序控制、数据校验等细节处理，极大地提升了我的嵌入式调试能力。

这段实践经历让我意识到，优秀工程师的培养需要理论学习和项目实践的螺旋上升。未来我将继续保持这种"学中做、做中学"的态度，通过参与更多实际项目来巩固专业知识，提升工程能力。课程设计中暴露的知识盲区也为我指明了后续学习方向，包括深入理解无线传感网络协议栈、加强嵌入式系统调试技能等。这段宝贵的实践经历，必将为我的专业发展打下坚实基础。

## 6. 关键代码

### 6.1 FloodingM.nc

```

1. module FloodingM {
2.
3.   provides {
4.     interface StdControl;
5.     interface AMSend as SendFromAPP;
6.     interface Receive as RecvToAPP;
7.   }
8.   uses {
9.     interface Timer<TMilli>;
10.    interface ParameterInit<uint16_t> as SeedInit;
11.    interface Random;
12.    interface SplitControl as CommControl;
13.    interface AMSend as SendToMAC;
14.    interface Receive as RecvFromMAC;
15.  }
16.
17. }implementation{
18.
19.   message_t SendMsg, RecvMsg, ForwardMsg[MAX_Forward_Buff]; //定义发送数据，接收数据
20.   Route_Msg NWKF; // 路由信息
21.   uint16_t Next_Addr; // 下一跳节点地址
22.   uint8_t Forward_Buff_Index;
23.   uint8_t RTable_Index; // 路由表下标
24.   uint8_t mySequence; // 序列号
25.   Route_Table RTable[MAX_RTABLE]; // 路由表
26.
27.   command error_t StdControl.start() { // 启动 StdControl

```

```

28.     uint8_t i;
29.     uint16_t random_num;
30.
31.     call SeedInit.init(TOS_NODE_ID);
32.     random_num = call Random.rand16();
33.
34.     atomic {
35. Next_Addr = AM_BROADCAST_ADDR; // 广播地址
36. Forward_Buff_Index = 0;
37. RTable_Index = 0;
38. mySequence = (uint8_t) (random_num%0xFF); // 0~255 之间的随机数
39. for (i=0;i<MAX_RTABLE;i++) { // 路由表赋初值
40. RTable[i].FinalDstAddr = UnknownAddr; // 最终地址未知
41. RTable[i].OrigSrcAddr = UnknownAddr; // 源地址未知
42. RTable[i].Sequence = 0xFF; // 序列号为 255
43. }
44. }
45. call CommControl.start(); // 启动分相控制
46. return SUCCESS;
47. }
48.
49. command error_t StdControl.stop() {return SUCCESS;} // 启动 std 控制结束
50. event void CommControl.startDone(error_t error) {} // 启动分相控制启动完成事件
51. event void CommControl.stopDone(error_t error) {} // 启动分相控制关闭完成事件
52.
53. ///////////////////////////////////////////////////
54. void insertMSGtoRTable(message_t* msg) { // 将接收到的路由信息写入路由表
55. Route_Msg pack;
56. memcpy(&pack, call SendToMAC.getPayload(msg), sizeof(Route_Msg)); // 获取数据载
   荷
57.
58. atomic{
59. RTable[RTable_Index].FinalDstAddr = pack.FinalDstAddr;
60. RTable[RTable_Index].OrigSrcAddr = pack.OrigSrcAddr;
61. RTable[RTable_Index].Sequence = pack.Sequence;
62.
63. RTable_Index++;
64. RTable_Index %= MAX_RTABLE;
65. }
66. }
67.
68. bool isRecvPrevious (message_t* msg) { // 是否以前接收过一样的数据包 (0, 没有, 1,
   有)
69. Route_Msg pack;

```

```

70. bool return_status = 0; // 曾经没有接收过
71. uint8_t i;
72.
73. memcpy(&pack, call SendToMAC.getPayload(msg), sizeof(Route_Msg));
74. for (i=0;i<MAX_RTABLE;i++) { // 查看路由表是否已经记录过
75.     if (RTable[i].FinalDstAddr == pack.FinalDstAddr &&
76.         RTable[i].OrigSrcAddr == pack.OrigSrcAddr &&
77.         RTable[i].Sequence == pack.Sequence)
78.     {
79.         return_status = 1; // 曾经接收过
80.         break;
81.     }
82. }
83. return return_status;
84. }
85.
86. //////////////////////////////////////
87.
88. command error_t SendFromAPP.send(am_addr_t addr, message_t* msg, uint8_t len)
89. {
90.     Route_Msg Route_M;
91.     void *DataPayLoad = call SendToMAC.getPayload(msg); // 获取 MAC 层数据载荷
92.     error_t return_status;
93.     // 路由信息
94.     Route_M.FrameControl = GeneralDataFrame;
95.     Route_M.FinalDstAddr = addr;
96.     Route_M.OrigSrcAddr = TOS_NODE_ID;
97.     Route_M.Sequence = mySequence;
98.     Route_M.TTL = Default_TTL;
99.     Route_M.Dst2_for_multihop = UnknownAddr;
100.    Route_M.Dst3_for_multihop = UnknownAddr;
101.    memcpy((void *)&(Route_M.AppData), DataPayLoad, len); // 将数据载荷拷贝给路由信
    息的数据字段
102.    memcpy(call SendToMAC.getPayload(&SendMsg), (uint8_t *)&Route_M, sizeof(Route
    _Msg)); // 将路由信息拷贝给待发送至 MAC 层数据载荷
103.
104.    mySequence++;
105.    mySequence %= 0xFF; // 序列号 + 1
106.
107.    return_status = call SendToMAC.send(Next_Addr, &SendMsg, sizeof(Route_Ms
    g)); // 发送至 MAC 层
108.    if (return_status == SUCCESS)
109.        insertMSGtoRTable(&SendMsg); // 将发送信息写入到路由表

```

```

110.
111. return return_status;
112. }
113.
114. command error_t SendFromAPP.cancel(message_t* msg){
115. return call SendToMAC.cancel(msg);
116. }
117.
118. command uint8_t SendFromAPP.maxPayloadLength(){
119. return call SendToMAC.maxPayloadLength();
120. }
121.
122. command void* SendFromAPP.getPayload(message_t* msg){
123. return call SendToMAC.getPayload(msg);
124. }
125.
126. event void SendToMAC.sendDone(message_t* msg, error_t error) {
127. signal SendFromAPP.sendDone(msg, error);
128. }
129.
130. //////////////////////////////////////
131.
132. task void RecvToAPP_task(){ // 接收数据包转发给应用层
133. memcpy(&NWKF, call SendToMAC.getPayload(&RecvMsg), sizeof(Route_Msg));
134. signal RecvToAPP.receive(&RecvMsg, (void *)&(NWKF.AppData), sizeof(Route_Msg));
135. }
136.
137. task void Forwarding_task(){ // 转发数据包任务
138. if (call SendToMAC.send(Next_Addr, &ForwardMsg[Forward_Buff_Index], sizeof(Route_Msg))==SUCCESS)
139. insertMSGtoRTTable(&ForwardMsg[Forward_Buff_Index]);
140. }
141.
142. event message_t* RecvFromMAC.receive(message_t* msg, void* payload, uint8_t len) {
143.
144. Route_Msg *pack = (Route_Msg *) call SendToMAC.getPayload(msg);
145.
146. if (pack->FinalDstAddr == TOS_NODE_ID) { // 如果最终目的节点为本节点
147.
148. #ifndef SHOW_OVERLAP_PACKET
149. if (!isRecvPrevious(msg)) // 以前未曾接收过
150. #endif

```

```

151. {
152.     memcpy((void*)&RecvMsg, (void*)msg, sizeof(message_t));
153.     insertMSGtoRTable(msg); // 将数据插入路由表
154.     post RecvToAPP_task(); // 将数据上传给应用层
155. }
156.
157. }else{ // 如果最终目的节点不是本节点
158.     if (!isRecvPrevious(msg)){ // 以前未曾接收过
159.         if (pack->TTL>0) { // 如果 TTL > 0
160.             pack->TTL--; // TTL 减一
161.
162.             Forward_Buff_Index++; // 转发次数加 1
163.             Forward_Buff_Index %= MAX_Forward_Buff;
164.             memcpy((void*)&ForwardMsg[Forward_Buff_Index], (void*)msg, sizeof(message_
t));
165.
166.             // Change Route Field
167.             {
168.                 Route_Msg Forward_NWKF; // 转发路由信息
169.
170.                 memcpy (&Forward_NWKF, call SendToMAC.getPayload(&ForwardMsg[Forward_Buff
_Index]), sizeof(Route_Msg));
171.                 Forward_NWKF.FrameControl = ForwardDataFrame;
172.                 if (Forward_NWKF.Dst2_for_multihop == UnknownAddr){
173.                     Forward_NWKF.Dst2_for_multihop = TOS_NODE_ID;
174.                 }else{
175.                     Forward_NWKF.Dst3_for_multihop = Forward_NWKF.Dst2_for_multihop;
176.                     Forward_NWKF.Dst2_for_multihop = TOS_NODE_ID;
177.                 }
178.                 memcpy (call SendToMAC.getPayload(&ForwardMsg[Forward_Buff_Index]), &Forw
ard_NWKF, sizeof(Route_Msg));
179.             }
180.
181.             post Forwarding_task(); // 抛出转发任务
182.         }
183.     }
184. }
185. return msg;
186. }
187.
188. command void* RecvToAPP.getPayload(message_t* msg, uint8_t* len){
189. return call RecvFromMAC.getPayload(msg, len);
190. }
191. command uint8_t RecvToAPP.payloadLength(message_t* msg){

```

```

192. return call RecvFromMAC.payloadLength(msg);
193. }
194.
195. event void Timer.fired() {}
196. }

```

## 6.2 Adhoc\_APPM.nc

```

1. module Adhoc_APPM {
2.   uses {
3.     interface Boot;
4.     interface Timer<TMilli> as Timer1; // 仅保留一个定时器
5.     interface Leds;
6.     interface BusyWait<TMicro, uint16_t>;
7.     interface RFID_Control;
8.     interface StdControl as RControl;
9.     interface AMSend as Rout_Send;
10.    interface Receive as Rout_Receive;
11.    interface SplitControl as CommControl;
12.  }
13.
14.  // Sensor components
15.  uses {
16.    interface Read<uint16_t> as Read_Humi;
17.    interface Read<uint16_t> as Read_Temp;
18.    interface Read<uint16_t> as Read_Photo;
19.  }
20.
21.  // Uart component
22.  uses {
23.    interface StdControl as SCSuartSTD;
24.    interface SCSuartDBG;
25.  }
26.
27. }
28. implementation {
29.
30.   message_t TXData;
31.
32.   //// sensor data ////
33.   uint16_t myTemp=0xFFFF;
34.   uint16_t myHumi=0xFFFF;
35.   uint16_t myPhoto=0xFFFF;
36.   uint16_t Raw_Temp=0xFFFF; // Raw temp info

```

```

37.  uint8_t FUN = 0x00;
38.  uint8_t Data[5] = {0x00,0x00,0x00,0x00,0x00,};
39.  ///////////////////////////////////////////////////
40.  // 只使用一个计数器
41.  uint16_t counter = 0;
42.  ///////////////////////////////////////////////////
43.  void calc_SHT_(uint16_t p_humidity ,uint16_t p_temperature);
44.
45.
46.  event void Boot.booted() {
47.      uint8_t mote_id = (uint8_t) TOS_NODE_ID;
48.      if (TOS_NODE_ID != Sink_Addr) // 不是sink 节点
49.      {
50.          call Timer1.startPeriodic(1000); // 统一使用1 秒周期的定时器
51.      }
52.      call RControl.start();
53.      call CommControl.start();
54.  }
55.
56.  event void CommControl.startDone(error_t error) {
57.      call SCSuartSTD.start();
58.      call RFID_Control.start();
59.  }
60.
61.  event void CommControl.stopDone(error_t error) {}
62.
63.  // 统一处理所有周期性任务
64.  event void Timer1.fired() {
65.      if (TOS_NODE_ID != Sink_Addr)
66.      {
67.          counter++;
68.
69.          // 光照读取任务（每2 秒）
70.          if (counter % 2 == 0) {
71.              call Leds.led2Toggle();
72.              call Read_Photo.read();
73.          }
74.
75.          // 温湿度读取任务（每4 秒）
76.          if (counter % 4 == 0) {
77.              call Leds.led2Toggle();
78.              call Read_Temp.read();
79.          }
80.

```

```

81. // RFID 读取任务 (每3 秒)
82. if (counter % 3 == 0) {
83.     call RFID_Control.RData_15693 (5);
84. }
85. }
86. }
87.
88. // 一般节点发送给Sink 节点采集到的数据
89. task void transmit_frame(){
90.
91.     DataFrameStruct DFS;
92.
93.     call Leds.led10n();
94.
95.     DFS.Temp = myTemp;
96.     DFS.Humi = myHumi;
97.     DFS.Photo = myPhoto;
98.     atomic DFS.FUN = FUN;
99.     memcpy (DFS.Data, Data, sizeof(Data));
100.    memcpy (call Rout_Send.getPayload(&TXData), &DFS, sizeof(DataFrameStruct));
101.
102.    call Rout_Send.send(Sink_Addr, &TXData, sizeof(DataFrameStruct));
103. }
104.
105. // 一般节点数据发送完成事件
106. event void Rout_Send.sendDone(message_t* m, error_t err) {
107.     if (err == SUCCESS)
108.         call Leds.led10ff();
109. }
110.
111. // sink 节点接收到一般节点发送的数据事件
112. event message_t* Rout_Receive.receive(message_t* msg, void* payload, uint8_t len) {
113.     if (TOS_NODE_ID == Sink_Addr)
114.     {
115.         uint8_t UART_Buff[65], *UART_Buff_p;
116.         uint8_t UART_Buff_len = 0, i;
117.         Route_Msg NWKF;
118.         DataFrameStruct DFS;
119.         UartFrameStruct UFS;
120.
121.         memcpy(&NWKF, call Rout_Send.getPayload(msg), sizeof(Route_Msg));
122.         memcpy(&DFS, NWKF.AppData, sizeof(DataFrameStruct));

```



```

123.     UART_Buff_p = (uint8_t *)&UFS;
124.
125.     {
126.         uint32_t Packet_Seq = (uint32_t) NWKF.Sequence;
127.         int16_t OrigSrcAddr = NWKF.OrigSrcAddr;
128.
129.         memcpy (UART_Buff_p+6, (void *)&OrigSrcAddr, 2);
130.         memcpy (UART_Buff_p+8, (void *)&TOS_NODE_ID, 2);
131.         memcpy (UART_Buff_p+10, (void *)&NWKF.Dst2_for_multihop, 2);
132.         memcpy (UART_Buff_p+12, (void *)&NWKF.Dst3_for_multihop, 2);
133.         memcpy (UART_Buff_p+14, (void *)&Packet_Seq, 4);
134.         memcpy (UART_Buff_p+18, (void *)&DFS.Temp, 2);
135.         memcpy (UART_Buff_p+20, (void *)&DFS.Humi, 2);
136.         memcpy (UART_Buff_p+22, (void *)&DFS.Photo, 2);
137.         memcpy (UART_Buff_p+24, (void *)&DFS.FUN, 1);
138.         memcpy (UART_Buff_p+25, (void *)&DFS.Data, 5);
139.     }
140.     UART_Buff_len = 0;
141.     for ( i=6; i<sizeof(UartFrameStruct) ; i++)
142.     {
143.         UART_Buff[UART_Buff_len++] = UART_Buff_p[i];
144.     }
145.     call SCSuartDBG.UARTSend(UART_Buff, UART_Buff_len -13);
146.
147.     call Leds.led0Toggle();
148. }
149. return msg;
150. }
151.
152. // 读取光照完成事件，发送数据至 sink 节点
153. event void Read_Photo.readDone(error_t err, uint16_t val) {
154. if (err == SUCCESS)
155. {
156.     myPhoto = val;
157.     atomic FUN = 1;
158. }
159. post transmit_frame();
160. }
161.
162. // 读取温度完成事件，准备读取湿度
163. event void Read_Temp.readDone(error_t err, uint16_t val) {
164. if (err == SUCCESS)
165.     Raw_Temp = val;
166. call Read_Humi.read();

```

```

167.     }
168.
169.     // 读取湿度完成事件, 发送数据至 sink 节点
170.     event void Read_Humi.readDone(error_t err, uint16_t val) {
171.         if (err == SUCCESS && Raw_Temp!=0xFFFF)
172.         {
173.             calc_SHT_(val, Raw_Temp);
174.             atomic FUN = 2;
175.         }
176.         post transmit_frame();
177.     }
178.
179.     // 对温湿度循环冗余, 得到真实数值
180.     void calc_SHT_(uint16_t p_humidity ,uint16_t p_temperature)
181.     {
182.         const float C1=-4.0;
183.         const float C2= 0.0405;
184.         const float C3=-0.0000028;
185.         const float T_1=0.01;
186.         const float T_2=0.00008;
187.
188.         float rh_lin;
189.         float rh_true;
190.         float t_C;
191.         float rh=(float)p_humidity;
192.         float t=(float)p_temperature;
193.
194.         t_C=t*0.01 - 40;
195.         rh_lin=C3*rh*rh + C2*rh + C1;
196.         rh_true=(t_C-25)*(T_1+T_2*rh)+rh_lin;
197.         if(rh_true>100)rh_true=100;
198.         if(rh_true<0.1)rh_true=0.1;
199.         atomic myTemp=(uint16_t)t_C;
200.         atomic myHumi=(uint16_t)rh_true;
201.     }
202.
203.     async event void RFID_Control.GetID_15693_Done (char status, uint8_t *buff, c
204.     har size){
205.
206.         // 获取 15693 标签数据完成后发送至 Sink 节点
207.         async event void RFID_Control.RData_15693_Done (char status, uint8_t *buff, c
208.         har size){

```

```

209.     atomic FUN = 3;
210.         post transmit_frame();
211.     }
212.
213.     async event void RFID_Control.GetID_14443A_Done(char status, uint8_t *buff, char size) {}
214.     async event void RFID_Control.WData_15693_Done(char status){}
215. }

```

### 6.3 Server.py

```

1. import os
2. import sys
3. import threading
4. import serial
5. from PySide6.QtWidgets import (QApplication, QMainWindow, QVBoxLayout, QHBoxLayout,
6.                                QWidget, QPushButton, QLabel, QTextEdit, QLineEdit,
7.                                QMessageBox, QTableWidgetItem, QTableWidgetItem, QGroupBox,
8.                                QHeaderView, QStatusBar, QComboBox)
9. from PySide6.QtCore import QFile, QIODevice, Qt, QThread, Signal, QThreadPool, QTimer
10. from datetime import datetime
11. import time
12. import struct
13. from collections import defaultdict
14. import requests
15. import json
16. import paho.mqtt.client as mqtt
17. from collections import defaultdict
18.
19.
20. def Attributes_set(OrigSrcAddr, DestAddr, Seq, tempture, Humi, Photo, FrameType, Error, CardData):
21.     return {
22.         'OrigSrcAddr': OrigSrcAddr,
23.         'DestAddr': DestAddr,
24.         'Seq': Seq,
25.         'tempture': tempture,
26.         'Humi': Humi,
27.         'Photo': Photo,
28.         'FrameType': FrameType,

```

```

29.         'Error': Error,
30.         'CardData': CardData
31.     }
32.
33.
34. def FrameType_rename(FrameType):
35.     return {
36.         1: '读取光照数据',
37.         2: '读取温湿度数据',
38.         3: '读取标签信息'
39.     }.get(FrameType, '未知类型')
40.
41.
42. def CardData_to_string(CardData):
43.     try:
44.         CardData = struct.unpack('4B', CardData)
45.         return ' '.join(str(b) if b <= 9 else chr(b) for b in CardData)
46.     except:
47.         return 'None'
48.
49.
50. class UploadThread(QThread):
51.     upload_status = Signal(bool, str)
52.     data_received = Signal(dict)
53.     block_command_received = Signal(str, str)
54.     def __init__(self, parent=None):
55.         super().__init__(parent)
56.         self.server_url = "http://192.168.140.133:8083/api/data"
57.         self.mqtt_broker = "192.168.140.133"
58.         self._running = True
59.         self.blocked_addresses=set()
60.         self.setup_mqtt()
61.
62.     def setup_mqtt(self):
63.         """MQTT 客户端初始化"""
64.         self.mqtt_client = mqtt.Client()
65.         self.mqtt_client.on_connect = self.on_mqtt_connect
66.         self.mqtt_client.on_message = self.on_mqtt_message
67.         try:
68.             self.mqtt_client.connect(self.mqtt_broker, 1883, 60)
69.             self.mqtt_client.subscribe('sensor/control', qos=1)
70.             self.mqtt_client.subscribe('sensor/block/+', qos=1)
71.             self.mqtt_client.loop_start()
72.         except Exception as e:

```

```

73.         self.upload_status.emit(False, f"MQTT 连接失败: {str(e)}")
74.
75.     def on_mqtt_connect(self, client, userdata, flags, rc):
76.         """MQTT 连接回调"""
77.         if rc == 0:
78.             self.upload_status.emit(True, "MQTT 连接成功")
79.         else:
80.             self.upload_status.emit(False, f"MQTT 连接错误码: {rc}")
81.
82.     def on_mqtt_message(self, client, userdata, message):
83.         """MQTT 消息接收回调"""
84.         try:
85.             topic = message.topic
86.             payload = message.payload.decode()
87.
88.             if topic == 'sensor/control':
89.                 self.data_received.emit({
90.                     'type': 'command',
91.                     'data': payload,
92.                     'timestamp': datetime.now().isoformat()
93.                 })
94.             elif topic.startswith('sensor/block/'):
95.                 # 处理屏蔽命令
96.                 source_addr = topic.split('/')[ -1]
97.                 action = payload # 'block' 或 'unblock'
98.                 self.block_command_received.emit(action, source_addr)
99.
100.                if action == 'block':
101.                    self.blocked_addresses.add(source_addr) # 使用复数形式
102.                    self.upload_status.emit(True, f"已屏蔽节点 {source_addr}")
103.                elif action == 'unblock':
104.                    if source_addr in self.blocked_addresses:
105.                        self.blocked_addresses.remove(source_addr)
106.                        self.upload_status.emit(True, f"已取消屏蔽节点 {source_addr}")
107.
108.            except Exception as e:
109.                self.upload_status.emit(False, f"MQTT 消息处理失败: {str(e)}")
110.
111.     def is_blocked(self, source_addr):
112.         """检查地址是否被屏蔽"""
113.         return str(source_addr) in self.blocked_addresses
114.
115.     def upload_data(self, data):

```

```

116.         """上传数据到服务器"""
117.         print(f"[DEBUG] 准备上传数据: {data}")
118.
119.         # 检查源地址是否被屏蔽
120.         source_addr = data.get('source_addr', '')
121.         if self.is_blocked(source_addr):
122.             print(f"[INFO] 数据来自被屏蔽的节点 {source_addr}, 不上传")
123.             return False
124.
125.         try:
126.             response = requests.post(
127.                 self.server_url,
128.                 json=data,
129.                 headers={'Content-Type': 'application/json'},
130.                 timeout=5
131.             )
132.             print(f"服务器响应: {response.status_code}, {response.text}")
133.             return response.status_code == 201
134.         except Exception as e:
135.             print(f"上传异常: {str(e)}")
136.             return False
137.
138.
139. class SerialControlApp(QMainWindow):
140.     update_ui_signal = Signal(dict)
141.     log_message_signal = Signal(str)
142.
143.     def __init__(self):
144.         super().__init__()
145.         self.serial_port = None
146.         self.upload_thread = UploadThread(self)
147.         self.upload_thread.data_received.connect(self.handle_remote_command)
148.         self.upload_thread.block_command_received.connect(self.handle_block_co
149. mmand) # 新增: 连接屏蔽命令信号
149.         self._running = True
150.         self.init_ui()
151.         self.init_serial()
152.
153.         # 连接信号
154.         self.update_ui_signal.connect(self.update_ui)
155.         self.log_message_signal.connect(self.log_message)
156.
157.         # 初始化数据属性
158.         self.Attributes = {

```

```

159.         'OrigSrcAddr': 0x00,
160.         'DestAddr': 0x00,
161.         'Seq': 0x00,
162.         'tempture': 0xFF,
163.         'Humi': 0xFF,
164.         'Photo': b'\xFF\xFF',
165.         'Error': 0x00,
166.         'CardData': b'\xFF\xFF\xFF\xFF',
167.         'FrameType': 0x00
168.     }
169.
170.     # 添加上传重试定时器
171.     self.upload_retry_timer = QTimer(self)
172.     self.upload_retry_timer.timeout.connect(self.retry_upload_cached)
173.     self.upload_retry_timer.start(60000) # 每60秒尝试重传
174.
175.     # 连接上传状态信号
176.     self.upload_thread.upload_status.connect(self.handle_upload_status)
177.
178.     # 初始化数据表
179.     self.init_data_table()
180.
181.     def init_ui(self):
182.         """初始化UI界面"""
183.         self.setWindowTitle("无线传感网络数据收集系统")
184.         self.resize(800, 700)
185.
186.         # 主窗口中心部件
187.         self.central_widget = QWidget()
188.         self.setCentralWidget(self.central_widget)
189.
190.         # 主布局
191.         main_layout = QVBoxLayout(self.central_widget)
192.
193.         # 数据采集模块
194.         self.data_group = QGroupBox("数据采集模块")
195.         self.data_group.setFont(self.get_font(12))
196.         main_layout.addWidget(self.data_group)
197.
198.         # 数据采集模块布局
199.         data_layout = QHBoxLayout(self.data_group)
200.
201.         # 温度显示
202.         temp_layout = QVBoxLayout()

```

```

203.         self.temp_label = QLabel("温度(℃):")
204.         self.temp_display = QLineEdit()
205.         self.temp_display.setReadOnly(True)
206.         temp_layout.addWidget(self.temp_label)
207.         temp_layout.addWidget(self.temp_display)
208.
209.         # 湿度显示
210.         humi_layout = QVBoxLayout()
211.         self.humi_label = QLabel("湿度(%):")
212.         self.humi_display = QLineEdit()
213.         self.humi_display.setReadOnly(True)
214.         humi_layout.addWidget(self.humi_label)
215.         humi_layout.addWidget(self.humi_display)
216.
217.         # 光照显示
218.         photo_layout = QVBoxLayout()
219.         self.photo_label = QLabel("光照(lx):")
220.         self.photo_display = QLineEdit()
221.         self.photo_display.setReadOnly(True)
222.         photo_layout.addWidget(self.photo_label)
223.         photo_layout.addWidget(self.photo_display)
224.
225.         # 标签数据显示
226.         card_layout = QVBoxLayout()
227.         self.card_label = QLabel("标签数据:")
228.         self.card_display = QLineEdit()
229.         self.card_display.setReadOnly(True)
230.         card_layout.addWidget(self.card_label)
231.         card_layout.addWidget(self.card_display)
232.
233.         # 按钮布局 (垂直布局)
234.         btn_layout = QVBoxLayout()
235.         self.btn_open = QPushButton("开启")
236.         self.btn_close = QPushButton("关闭")
237.         self.btn_clear = QPushButton("清空")
238.         self.btn_close.setEnabled(False)
239.         # 将按钮添加到垂直布局
240.         btn_layout.addWidget(self.btn_open)
241.         btn_layout.addWidget(self.btn_close)
242.         btn_layout.addWidget(self.btn_clear)
243.         # 修改过滤控件标签
244.         filter_group = QGroupBox("黑名单过滤") # 修改标题
245.         filter_layout = QHBoxLayout(filter_group)
246.

```



```

247.         filter_label = QLabel("黑名单地址:") # 修改标签
248.         filter_layout.addWidget(filter_label)
249.
250.         self.addr_filter = QComboBox()
251.         self.addr_filter.addItem("不启用黑名单", 0xFF) # 修改选项文本
252.         self.addr_filter.addItem("0x00", 0x00)
253.         self.addr_filter.addItem("0x01", 0x01)
254.         self.addr_filter.addItem("0x02", 0x02)
255.         filter_layout.addWidget(self.addr_filter)
256.         self.filter_btn = QPushButton("应用黑名单") # 修改按钮文本
257.         self.filter_btn.clicked.connect(self.apply_filter)
258.         filter_layout.addWidget(self.filter_btn)
259.
260.         # 将过滤布局添加到主布局
261.         btn_layout.addLayout(filter_layout)
262.         # 将按钮布局和过滤组添加到主数据布局
263.         data_layout.addLayout(temp_layout)
264.         data_layout.addLayout(humi_layout)
265.         data_layout.addLayout(photo_layout)
266.         data_layout.addLayout(card_layout)
267.         data_layout.addLayout(btn_layout) # 添加按钮布局
268.         data_layout.addWidget(filter_group) # 添加过滤组
269.
270.         # 数据表格
271.         self.data_table = QTableWidgetItem()
272.         self.data_table.setColumnCount(5)
273.         self.data_table.setHorizontalHeaderLabels(["源地址", "目的地址", "Seq", "CardData", "Type"])
274.         self.data_table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
275.         main_layout.addWidget(self.data_table)
276.
277.         # 状态栏
278.         self.status_bar = QStatusBar()
279.         self.setStatusBar(self.status_bar)
280.         self.status_bar.showMessage("系统准备就绪")
281.
282.         # 连接信号
283.         self.btn_open.clicked.connect(self.start_system)
284.         self.btn_close.clicked.connect(self.stop_system)
285.         self.btn_clear.clicked.connect(self.clear_data)
286.
287.         # 应用样式
288.         self.apply_styles()

```

```

289.     def handle_block_command(self, action, source_addr):
290.         """处理屏蔽/取消屏蔽命令"""
291.         if action == 'block':
292.             self.log_message(f"收到屏蔽命令, 将忽略来自 {source_addr} 的数据")
293.         elif action == 'unblock':
294.             self.log_message(f"收到取消屏蔽命令, 将恢复接收来自 {source_addr} 的数据")
295.     def apply_filter(self):
296.         """应用黑名单过滤(隐藏指定源地址的数据)"""
297.         if not hasattr(self, 'addr_filter'):
298.             return
299.
300.         # 获取选中的黑名单地址
301.         blacklist_addr = self.addr_filter.currentData()
302.
303.         # 如果是"全部地址"选项, 则显示所有数据
304.         if blacklist_addr == 0xFF:
305.             for row in range(self.data_table.rowCount()):
306.                 self.data_table.setRowHidden(row, False)
307.             self.status_bar.showMessage("已显示全部数据", 3000)
308.             return
309.
310.         # 黑名单过滤逻辑
311.         for row in range(self.data_table.rowCount()):
312.             item = self.data_table.item(row, 0) # 源地址在第0列
313.             if item:
314.                 try:
315.                     addr = int(item.text(), 16)
316.                     # 隐藏匹配黑名单地址的行(黑名单模式)
317.                     self.data_table.setRowHidden(row, addr == blacklist_addr)
318.                 except ValueError:
319.                     pass
320.
321.             self.status_bar.showMessage(f"已隐藏源地址 {hex(blacklist_addr)} 的数据", 3000)
322.
323.         # 在接收新数据时自动应用当前过滤
324.     def init_data_table(self):
325.         """初始化数据表格"""
326.         self.data_table.setColumnCount(5)
327.         self.data_table.setHorizontalHeaderLabels(["源地址", "目的地址", "Seq", "CardData", "Type"])
328.         self.data_table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)

```

```

329.         self.data_table.setEditTriggers(QTableWidget.NoEditTriggers)
330.         self.data_table.setSelectionBehavior(QTableWidget.SelectRows)
331.         self.data_table.setSelectionMode(QTableWidget.SingleSelection)
332.
333.     def apply_styles(self):
334.         """应用样式表"""
335.         self.setStyleSheet("""
336.             QMainWindow {
337.                 background-color: #f5f5f5;
338.             }
339.             QGroupBox {
340.                 border: 1px solid #ccc;
341.                 border-radius: 5px;
342.                 margin-top: 10px;
343.                 padding-top: 15px;
344.             }
345.             QGroupBox::title {
346.                 subcontrol-origin: margin;
347.                 left: 10px;
348.                 padding: 0 3px;
349.             }
350.             QPushButton {
351.                 min-width: 80px;
352.                 padding: 5px;
353.                 background-color: #4CAF50;
354.                 color: white;
355.                 border: none;
356.                 border-radius: 4px;
357.             }
358.             QPushButton:disabled {
359.                 background-color: #cccccc;
360.             }
361.             QPushButton:hover {
362.                 background-color: #45a049;
363.             }
364.             QPushButton:pressed {
365.                 background-color: #3d8b40;
366.             }
367.             QLineEdit {
368.                 padding: 5px;
369.                 border: 1px solid #ddd;
370.                 border-radius: 4px;
371.             }
372.             QTableWidget {

```

```

373.             border: 1px solid #ddd;
374.             gridline-color: #eee;
375.         }
376.         QHeaderView::section {
377.             background-color: #4CAF50;
378.             color: white;
379.             padding: 5px;
380.             border: none;
381.         }
382.     """
383.
384.     def get_font(self, size=10, bold=False):
385.         """获取字体对象"""
386.         font = self.font()
387.         font.setPointSize(size)
388.         font.setBold(bold)
389.         return font
390.
391.     def init_serial(self):
392.         """初始化串口连接"""
393.         try:
394.             self.serial_port = serial.Serial(
395.                 port="COM8",
396.                 baudrate=57600,
397.                 timeout=None
398.             )
399.             self.log_message("串口 COM8 初始化成功")
400.         except Exception as e:
401.             self.show_error(f"串口初始化失败: {str(e)}")
402.             self.serial_port = None
403.
404.     def start_system(self):
405.         """启动系统"""
406.         if self.serial_port and self.serial_port.is_open:
407.             self.btn_open.setEnabled(False)
408.             self.btn_close.setEnabled(True)
409.             self._running = True
410.
411.             # 启动数据接收线程
412.             self.receive_thread = threading.Thread(
413.                 target=self.receive_data,
414.                 daemon=True
415.             )
416.             self.receive_thread.start()

```

```

417.         self.log_message("系统启动 - 开始接收数据")
418.     else:
419.         self.show_error("串口未正确初始化，无法启动")
420.
421.     def receive_data(self):
422.         """接收串口数据"""
423.         while self._running and self.serial_port and self.serial_port.is_open:
424.             try:
425.                 Frame = self.serial_port.read(24)
426.                 print(f"[DEBUG] 收到原始串口数据({len(Frame)}字节): {Frame.hex()}")
427.
428.                 if len(Frame) != 24:
429.                     print(f"[WARNING] 数据长度不正确，期望 24 字节，实际收到{len(Frame)}字节")
430.                     continue
431.
432.                 # 解析数据
433.                 attrs = Attributes_set(
434.                     OrigSrcAddr=Frame[0],
435.                     DestAddr=Frame[2],
436.                     Seq=Frame[8],
437.                     tempture=Frame[12],
438.                     Humi=Frame[14],
439.                     Photo=Frame[16:18],
440.                     FrameType=Frame[18],
441.                     Error=Frame[19],
442.                     CardData=Frame[20:]
443.                 )
444.
445.                 print(f"[DEBUG] 解析后的属性: {attrs}")
446.
447.                 # 转换数据格式
448.                 try:
449.                     attrs['Photo'] = struct.unpack('H', attrs['Photo'])[0]
450.                     attrs['FrameType'] = FrameType_rename(attrs['FrameType'])
451.                     # 如果是标签信息类型，使用实际卡数据
452.                     if attrs['FrameType'] == '读取标签信息':
453.                         attrs['CardData'] = CardData_to_string(attrs['CardData'])
454.                     else:
455.                         # 非标签信息类型，卡数据显示为"0000"
456.                         attrs['CardData'] = "0 0 0 0"
457.                 except Exception as e:

```

```

458.             print(f"[ERROR] 数据转换失败: {str(e)}")
459.             continue
460.
461.             # 准备上传数据
462.             upload_data = {
463.                 'temperature': attrs['tempture'],
464.                 'humidity': attrs['Humi'],
465.                 'light': attrs['Photo'],
466.                 'card_data': attrs['CardData'],
467.                 'source_addr': hex(attrs['OrigSrcAddr']),
468.                 'frame_type': attrs['FrameType'],
469.                 'timestamp': datetime.now().strftime('%Y-%m-%d %H:%M:%S
') # 添加时间戳
470.             }
471.             print(f"[DEBUG] 准备上传的数据包: {upload_data}")
472.
473.             # 上传数据
474.             success = self.upload_thread.upload_data(upload_data)
475.             print(f"[DEBUG] 上传结果: {'成功' if success else '失败'}")
476.
477.             # 更新UI
478.             self.update_ui_signal.emit(attrs)
479.
480.         except Exception as e:
481.             print(f"[ERROR] 数据接收处理异常: {str(e)}")
482.             continue
483.
484.     def update_ui(self, attributes):
485.         """更新UI 界面"""
486.         # 更新数据显示
487.         self.temp_display.setText(str(attributes['tempture']))
488.         self.humi_display.setText(str(attributes['Humi']))
489.         self.photo_display.setText(str(attributes['Photo']))
490.         self.card_display.setText(str(attributes['CardData']))
491.
492.         # 更新数据表格
493.         row = self.data_table.rowCount()
494.         self.data_table.insertRow(row)
495.
496.         self.data_table.setItem(row, 0, QTableWidgetItem(hex(attributes['OrigSrcAddr'])))
497.         self.data_table.setItem(row, 1, QTableWidgetItem(hex(attributes['DestAddr'])))

```

```

498.         self.data_table.setItem(row, 2, QTableWidgetItem(str(attributes['Seq
' ])))
499.         self.data_table.setItem(row, 3, QTableWidgetItem(attributes['CardData
' ]))
500.         self.data_table.setItem(row, 4, QTableWidgetItem(attributes['FrameType
' ]))
501.
502.         # 自动应用当前过滤
503.         filter_addr = self.addr_filter.currentData()
504.         if filter_addr != 0xFF: # 如果不是"全部地址"模式
505.             item = self.data_table.item(row, 0)
506.             if item:
507.                 try:
508.                     addr = int(item.text(), 16)
509.                     self.data_table.setRowHidden(row, addr != filter_addr)
510.                 except ValueError:
511.                     self.data_table.setRowHidden(row, True)
512.
513.         # 自动滚动到最后一行
514.         self.data_table.scrollToBottom()
515.
516.         # 记录日志
517.         self.log_message(f"收到数据: 温度{attributes['tempture']}°C, 湿度{attrib
utes['Humi']}%")
518.
519.     def stop_system(self):
520.         """关闭系统"""
521.         self._running = False
522.         self.btn_open.setEnabled(True)
523.         self.btn_close.setEnabled(False)
524.         self.log_message("系统已停止")
525.
526.     def clear_data(self):
527.         """清空数据"""
528.         self.temp_display.clear()
529.         self.humi_display.clear()
530.         self.photo_display.clear()
531.         self.card_display.clear()
532.         self.data_table.setRowCount(0)
533.         self.log_message("已清空所有数据")
534.
535.     def handle_remote_command(self, data):
536.         """处理从MQTT接收到的远程控制命令"""
537.         if data['type'] == 'command':

```

```

538.         if data['data'] == 'start' and self.btn_open.isEnabled():
539.             self.start_system()
540.             self.log_message("收到远程启动命令")
541.         elif data['data'] == 'stop' and self.btn_close.isEnabled():
542.             self.stop_system()
543.             self.log_message("收到远程停止命令")
544.
545.     def handle_upload_status(self, success, message):
546.         """处理上传状态反馈"""
547.         if success:
548.             self.log_message(f"[上传成功] {message}")
549.         else:
550.             self.log_message(f"[上传失败] {message}")
551.             self.status_bar.showMessage(f"上传失败: {message}", 5000)
552.
553.     def upload_in_thread(self, data):
554.         """线程池中执行的上传任务"""
555.         success = self.upload_thread.upload_data(data)
556.         if not success:
557.             self.log_message("数据上传失败, 将尝试缓存数据")
558.             self.cache_data(data)
559.
560.     def cache_data(self, data):
561.         """本地缓存失败的上传数据"""
562.         cache_file = "upload_cache.json"
563.         try:
564.             cached = []
565.             # 读取现有缓存
566.             if os.path.exists(cache_file):
567.                 with open(cache_file, 'r') as f:
568.                     cached = json.load(f)
569.
570.             # 添加新数据
571.             cached.append(data)
572.
573.             # 保存缓存 (限制最大100条)
574.             with open(cache_file, 'w') as f:
575.                 json.dump(cached[-100:], f)
576.
577.         except Exception as e:
578.             print(f"缓存数据失败: {str(e)}")
579.
580.     def retry_upload_cached(self):
581.         """重试上传缓存的数据"""

```



```

582.         cache_file = "upload_cache.json"
583.         if not os.path.exists(cache_file):
584.             return
585.
586.         try:
587.             with open(cache_file, 'r') as f:
588.                 cached = json.load(f)
589.
590.                 success_count = 0
591.                 for data in cached[:]: # 使用副本遍历
592.                     if self.upload_thread.upload_data(data):
593.                         success_count += 1
594.                         cached.remove(data)
595.
596.                 # 更新缓存文件
597.                 with open(cache_file, 'w') as f:
598.                     json.dump(cached, f)
599.
600.                 self.log_message(f"重传完成: 成功{success_count}条")
601.         except Exception as e:
602.             self.log_message(f"重传缓存失败: {str(e)}")
603.
604.     def log_message(self, message):
605.         """记录日志消息"""
606.         timestamp = datetime.now().strftime("%H:%M:%S")
607.         self.status_bar.showMessage(f"[{timestamp}] {message}", 5000)
608.
609.     def show_error(self, message):
610.         """显示错误消息"""
611.         QMessageBox.critical(self, "错误", message)
612.
613.     def closeEvent(self, event):
614.         """关闭窗口时确保资源释放"""
615.         self._running = False
616.
617.         if hasattr(self, 'receive_thread') and self.receive_thread.is_alive():
618.             self.receive_thread.join(timeout=1)
619.
620.         if hasattr(self, 'upload_thread'):
621.             self.upload_thread.mqtt_client.disconnect()
622.             self.upload_thread.quit()
623.
624.         if self.serial_port and self.serial_port.is_open:
625.             self.serial_port.close()

```

```
626.  
627.         super().closeEvent(event)  
628.  
629.  
630. if __name__ == "__main__":  
631.     app = QApplication(sys.argv)  
632.  
633.     # 设置全局样式  
634.     app.setStyle("Fusion")  
635.  
636.     # 创建并显示主窗口  
637.     window = SerialControlApp()  
638.     window.show()  
639.  
640.     sys.exit(app.exec())
```