1. ­um202 发送过来的是 22个字节(ASCII码) 如：2220163332100200000B\r\n. (\r = 0x0A, \r = 0x0D)
2. 异常RFID 标签EPC 过滤

**———————————————————命令队列————————————————————**

1. um202 命令接收（EPC信息接收）

um202 命令接收队列结构体定义:

typedef struct{

pointer Head; // 队列头指针

pointer Tail; // 队列尾指针 指向当前入队列数据的下一个位置

pointer Current; // 当前队列数据指针

ElemType Data[CMD\_QUEUE\_MAX\_SIZE]; // 队列 512

}Cmd\_Queue\_Type

1. 命令队列初始化：void Cmd\_Queue\_Init(Cmd\_Queue\_Type \*Cmd\_Queue)；//队列内部各项指针置 0
2. 入队列：void Cmd\_Queue\_Push(Cmd\_Queue\_Type \*Cmd\_Queue, ElemType \*data)

首先检测队列是否已经存满

（(Cmd\_Queue->Tail - Cmd\_Queue->Head + 1) == CMD\_QUEUE\_MAX\_SIZE），

那么Tail 置0，即回到队列的首部

然后存入数据到Tail位置，然后Tail++(即指向当前入队列数据的下一个位置)

1. 出队列：void Cmd\_Queue\_Pop(Cmd\_Queue\_Type \*Cmd\_Queue, ElemType \*data)

首先检测队列Current指针的位置，如果已经指向了队列数组的最后一个数据，就置 0 Current

Cmd\_Queue->Current == (CMD\_QUEUE\_MAX\_SIZE - 1)

然后取出 Current 位置的队列数据，最后Current 移动到下一个队列位置

1. 从队列取一条 EPC 信息，并且检测是不是合法RFID EPC信息:

uint8\_t Cmd\_Queue\_Return\_And\_Check\_String(Cmd\_Queue\_Type \*Cmd\_Queue, char \*data\_string)

为什么要检测？答：因为可能天线可能会检测到一些不合法的EPC信息，并将其存入了Cmd\_Queue,所以取出信息的时候要检测

首先：准备一个较长的（64Byte）的字符数组，然后从Cmd\_Queue中取出一条命令信息。

然后：将这条信息与事先定义好的EPC 前缀进行比较

char \*EPC\_INFO\_PREFIX\_1 = "222016333210018000";

char \*EPC\_INFO\_PREFIX\_2 = "222016333210019000";

char \*EPC\_INFO\_PREFIX\_3 = "222016333210020000";

(strncmp(string\_temp, EPC\_INFO\_PREFIX\_1, strlen(EPC\_INFO\_PREFIX\_1)) == 0)

最后：如果相等strncpy(data\_string, string\_temp, DATA\_BYTE\_LENGTH);并且return 1；

**————————————————数据发送记录链表———————————————————**

1. 数据发送记录类型（链表结点）：

typedef struct EPC\_Node{

uint8\_t EPC\_String\_Count; // 相同 EPC 信息的发送次数

char EPC\_String[DATA\_BYTE\_LENGTH]; // 数据发送记录

struct EPC\_Node \*next;

}EPC\_Node, \*Data\_Send\_Record\_Type;

为什么需要这个链表？UM202发过来的字符串是有可能重复的，它检测到什么标签就发送什么标签信息，所以我们需要查询重复的标签信息，简单的思路是每当单片机接收到新的EPC信息时就发送到屏幕，如果是已经发送过的数据信息就更新对应信息的检测次数。

1. 创建发送记录链表头结点：Data\_Send\_Record\_Type Create\_Data\_Send\_Record\_LinkList(void)；

首先执行创建头结点：

Data\_Send\_Record\_Type head = (Data\_Send\_Record\_Type)malloc(sizeof(EPC\_Node));

然后：如果malloc 分配内存成功 就让 next = NULL， EPC\_String\_Count = 0;

最后：return head;

1. 创建链表内部结点：Data\_Send\_Record\_Type Create\_Data\_Send\_Record\_Node(void)

同9.

注意：/\*在启动文件 starup\_stm32f10x\_hd.s 中 更改 Heap\_Size 为 EQU 0x0000F000 堆空间分配不足会导致 malloc 分配空间失败\*/

1. 链表插入结点(尾部)：

void Data\_Record\_Insert\_Node(char \*data\_string, Data\_Send\_Record\_Type Data\_Send\_Record\_LinkList)

首先：创建一个指向链表头结点的指针p, 新建一个结点Node

然后：由于创建结点的时候在我的算法内部肯定是因为进入了新数据，所以此时可以将此数据的EPC\_String\_Count 置1，将data\_string 赋给 EPC\_String.

Node->EPC\_String\_Count = 1;

strncpy(Node->EPC\_String, data\_string, DATA\_BYTE\_LENGTH);

最后：将新结点插入到链表尾部

1. 查找此EPC信息在Data\_Send\_Record.EPC\_String是否是否已经发送过

Data\_Send\_Record\_Type EPC\_Info\_Is\_Exist(char \*data\_string, Data\_Send\_Record\_Type Data\_Send\_Record\_LinkList)

首先：定义一个指向链表头结点的结构体指针 p

然后：SCREEN\_MODIFY\_ROW这个全局变量用来记录将要修改的行数，因为链表中的数据存储循序和屏幕显示的顺序是一样的，所以当在链表中查找时 执行SCREEN\_MODIFY\_ROW ++，就获得了如果需要修改屏幕内容的所需要的行数。

while(p)

{

p = p->next;

if(strcmp(p->EPC\_String, data\_string) == 0)

{

return p;

}

SCREEN\_MODIFY\_ROW ++; // 记录将要修改的行数

}

return NULL;

**————————————————异常设备查询———————————————————**

1. \*Exception\_History\_Array; // 查询离线设备历史记录

\*Exception\_Current\_Array; // 查询离线设备当前记录

1. 复制数据记录表内容到当前异常查询数组

void Exception\_Copy\_To\_Current\_Array(Data\_Send\_Record\_Type Data\_Send\_Record\_LinkList)

首先：定义一个指针 \*current\_pointer，定义一个 p用来指向链表头结点的下一个结点。

然后：给Exception\_Current\_Array 重新分配（realloc）ROW\_COUNT个空间，然后把current\_pointer指向Exception\_Current\_Array，

最后：数据记录表中的 EPC\_String\_Count 到Exception\_Current\_Array，

1. 复制数据记录表内容到历史异常查询数组

void Exception\_Copy\_To\_History\_Array(Data\_Send\_Record\_Type Data\_Send\_Record\_LinkList)

首先：定义一个指针 \*history\_pointer，定义一个 p用来指向链表头结点的下一个结点。

然后：给Exception\_ History \_Array 重新分配（realloc）ROW\_COUNT个空间，然后把history \_pointer指向Exception\_ History \_Array，

最后：数据记录表中的 EPC\_String\_Count 到Exception\_History \_Array, 同时History\_Array\_Element\_Count++; （// 用来记录 Exception\_History\_Array 数组的长度 初始值为1）

1. 查询设备是否发生离线异常

void Exception\_Check()

首先：定义一个offline\_count 用来记录有多少个离线设备，定义一个定义一个指针 \*current\_pointer用来指向Exception\_Current\_Array, 定义一个定义一个指针 \*history \_pointer用来指向Exception\_History \_Array,

然后：清除异常设备数据记录Screen\_Exception\_Data\_Clear()，用来清除屏幕异常设备表

其次：发送异常数据，比较Exception\_Current\_Array和 Exception\_History\_Array内部查询次数是否一致，如果一致则说明就过来多次检测这个数据都没有更新，可以认为其对应的设备离线

最后：更新在线设备数，离线设备数，蜂鸣器报警，Exception\_History\_Array 数组的长度（History\_Array\_Element\_Count）重新置零

**————————————————算法总体说明———————————————————**

① 连接在 USART1 上的 Zigbee2 接收到来自 连接在 UM202 上的 Zigbee1 发送来的 22字节数据，然后 Zigbee2 向 USART1 发送接收到的数据，此时 USART1触发 USART1接收数据中断，在中断内部将接收到的数据push 到 Cmd\_Queue队列中，当接收到最后一个字符“\n”（0x0D）时将中断接收标志INTERRUT\_END置为1，此时 main 函数中的 while 语句将会进入 if语句。

② main 函数中的 while 语句将会进入 if语句。此时已经一次USART1串口中断接收数据操作已经结束

则从 Cmd\_Queue队列中pop 出一条EPC 信息，由于Cmd\_Queue中可能会包含不合法的EPC信息，所以要对 pop 出来的EPC 信息进行检测，检测方法是：与实现定义好的EPC前缀字符串进行比较，如果相同就是合法的EPC,

③ 当 pop出一条合法的EPC信息后，还要检测这条信息是不是新EPC数据或者已经发送过到屏幕显示，所以要进行查重，如果数据发送记录链表Data\_Send\_Record\_LinkList中某个结点中有这个EPC信息，那么返回链表中这个结点，并且记录要修改屏幕的行数。如果链表中没有结点包含这个EPC信息，那么表明这个是新数据，则发送新数据到屏幕，并且用新数据建立一个结点插入到链表的尾部。一次这样的操作结束后，INTERRUT\_END置为0，等待下一次中断发生。

④ 异常设备检测。当执行一定次数的while循环则对设备是否在线进行检测。算法是：将历史设备查询记录次数与一定次数while循环后的当前设备查询次数进行比较，如果相同则认为设备离线，不同则说明数据在线。

论文框架：

摘要

关键字

英文摘要

英文关键字

1. 导论
   1. 引言

简述一下作品的应用背景。简介说明设计此作品原因

简述一下市场上的代表性产品。

引入毕业设计，简述系统设计。

* 1. STM32F103ZET6微控制器简介
  2. RFID技术简介
  3. Zigbee技术简介

1. 系统分析及方案设计

2.1 系统概述

2.2 系统分解

2.3 系统总体方案设计

2.4 功能模块的选择

1. 系统硬件设计

3.1元件清单

3.2 MCU——STM32F103VET6

3.2.1 STM32F103VET6单片机引脚介绍

3.2.2 STM32F103VET6单片机主要功能

3.3天线

3.3.1 天线主要功能介绍

3.4 RFID读写器——UM202

3.4.1 UM202 RFID读写器引脚介绍

3.4.2 UM202 RFID读写器主要功能

3.5显示屏

1. 软件设计
2. 联机调试
3. 结论

参考文献

致谢