成绩

中国农业大学

课程设计

（2021-2022 学年 秋季学期）

论文题目： 基于物联网的家庭智能鸡舍监测与控制系统

课程名称： 控制系统综合设计 I

任课教师： 董乔雪

班　　级： 自动 191

学　　号： 2019308130215
2019319010404
2019308130108

姓　　名： 孟令昶、金政祥、詹金

目录

基于物联网的家庭智能鸡舍监测与控制系统

一、需求分析

1. 鸡舍环境因素分析及现状

根绝网上查阅的资料，鸡舍的环境因素包括鸡舍内的温度、湿度值、有害气体的含量、粉尘含量、风速、光照强度等。而这些环境因素的变化对经济效应的影响及对应控制是禽畜养殖者最为关注的问题。

温度是鸡舍环境控制中仅次于通风重要的因素，尤其在育雏前期，保温的重要性甚至超过通风。家禽的最适温度在不同的饲养阶段有所不同，在雏鸡前期根据雏鸡的体质温度控制在34℃～36℃之间，以后根据季节每周降2℃~3℃；到青年鸡和成年鸡的最适环境温度是18℃～24℃，在这个范围内鸡的基础代谢最低。饲料的利用率最高。然而在实际生产中很难达到如此理想的状态。家禽没有汗腺，环境温度升高时，鸡体温度也会随之增加，鸡只能通过喘吸增加蒸发散热和大量饮水来调节体温，采食量降低，同时体内还会发生一系列的生理应激反应，导致生产性能下降，抵抗力低下，从而引起发病；环境温度超过 39℃时就可引起中暑死亡。而当环境温度低下时，尤其在冬季，寒冷是造成生产性能下降及抗病力降低的主要原因。

湿度是鸡舍环境控制中的另一重要因素，在育雏前10天，湿度要求达到60%～65%，以利于卵黄的吸收，之后在温度舒适范围内，最适宜的相对湿度应控制在50%~55%。如果舍内湿度太低，就会造成鸡舍内粉尘飞扬，引起鸡群发生呼吸道疾病，也会引起啄肛等现象。但湿度太高也会产生不利的影响,潮湿空气中的导热性为干燥空气的10倍,冬季如果舍内湿度过高，就会使鸡体的散热量增加，使鸡更加寒冷，影响生产性能，夏季舍内湿度过高，就会使鸡通过呼吸蒸发散热的效率降低，自身调节能力下降，造成鸡体体质下降，同时高温高湿的环境又非常适合病菌的大量繁殖，易引发各种疾病，引起增重减慢和产蛋量下降。

光照是在现代蛋鸡养殖生产过程中必备的一个饲养条件。光照时间的长短、光照强度的强弱等对雏鸡生长发育的达标与否，成活率的高低及对产蛋鸡生产性能的发挥起着至关重要的作用。适宜的光照时间与光照强度能使家禽发挥其良好的生产性能。光照对家禽的影响，育雏期是通过延长光照时间，增加雏鸡的采食量，以促进雏鸡的生长发育，育成期延长（缩短）光照时间会使鸡性成熟时间提前（退后），都对生产不利，所以在育成期及产蛋期都需要稳定的光照时间不可忽长忽短，育成期的光照时间应保持在 8~10 小时为宜，产蛋期应保持在14~16小时。光照强度亦不可忽强忽弱。光照强度过强或过弱都会对鸡产生不良的影响。光照强度太大，不仅浪费电能，而且会使鸡烦躁不安，造成神经质，易惊群，易发生斗殴、啄癖和脱肛。光照强度过小，对产蛋鸡起不到刺激的作用，从而影响产蛋量。在蛋鸡生产过程中

必须按照 1~3 日龄、14~18 周龄、18 周龄以后分别按照 20 勒克斯、5 勒克斯、10 勒克斯的标准严格控制好光照强度。

2. 实际需求

对于以上环境变量，传统的检测方法多依靠人工采样或仪器进行测量，只能得到在某一时间点上舍内环境的数据，不能全面了解某一时间段内环境的变化；同时测量时仪器频繁的移动会使测量结果产生误差，测量人员在进入鸡舍测数时，也会引起蛋鸡群的应激反应，从而影响了蛋鸡的正常养殖。因此，需要一套能实时检测鸡舍各环境因素的智能系统。

此外，目前许多养殖户还采取"粗放式"的养殖模式，许多养殖户等到鸡生病的时候，才知道鸡舍内的环境有问题，除了一些大型的养殖场舍得采用自动化程度比较高的实时监控系统，许多小的养殖场为了节约成本，就不采用自动监控系统，受环境因素的影响比较大；并且，根据相关资料的调研，小型养殖场、家庭养殖环境等都有鸡舍智能化发展的意愿和趋势。

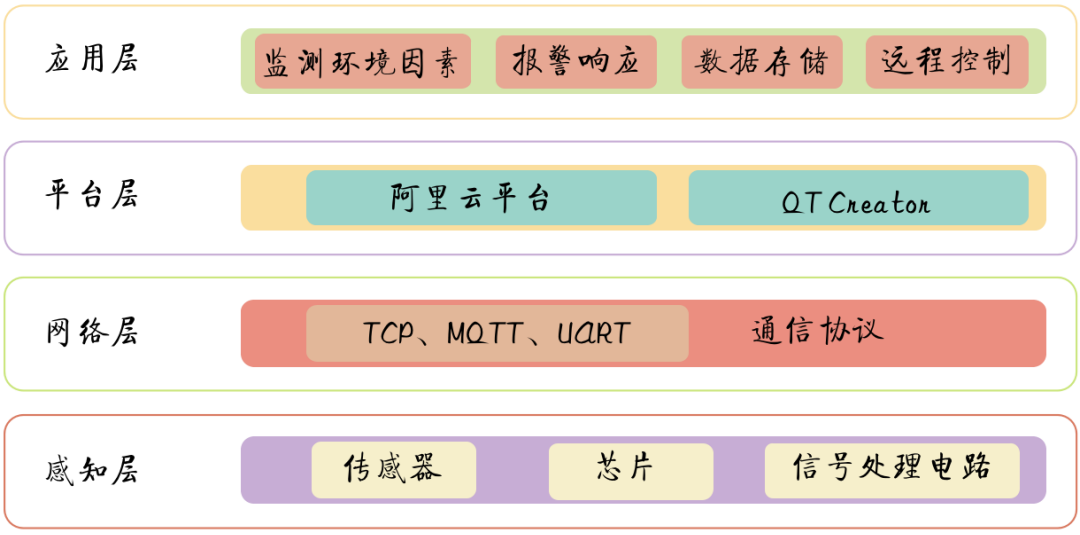
3. 针对需求的解决分析

基于此问题，我们计划基于 MSP430G2553 和物联网架构，研究建立一套家庭鸡舍环境自动监测和控制系统。该系统可以对环境因素进行综合性、连续性的监测，检测到的数据可以实时显示，同时也可以写入本地存储、阿里云平台或者计算机端；分析采集到的环境因素在不同时段的变化，实现禽舍环境的智能控制。

初代设计的应用场景为小型的鸡舍或家用笼舍。

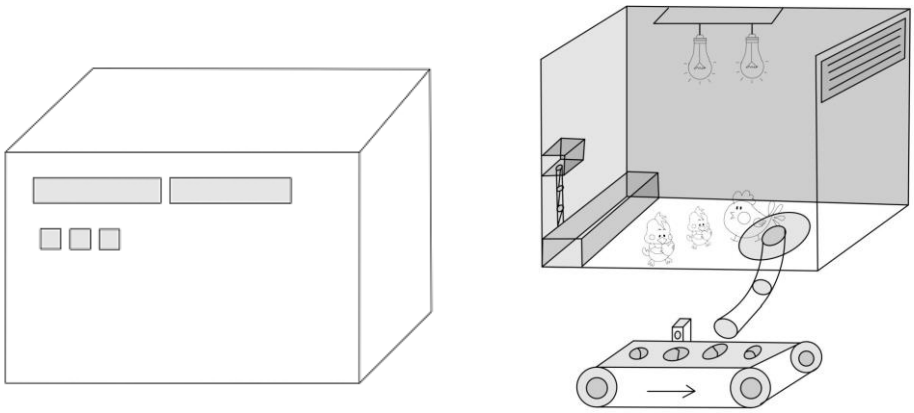
二、概念设计

本项目的智能鸡舍系统物联网架构分为三层，即数据感知层、数据传输层、智能控制层，如下图所示：

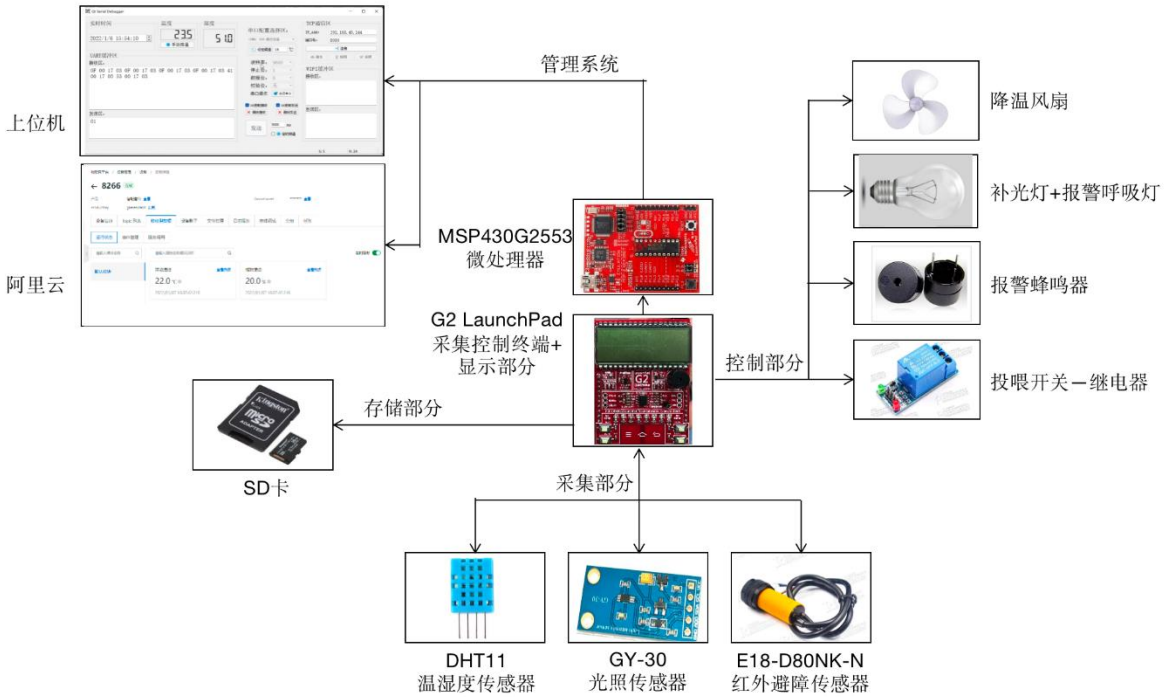


其中下位机部分是以 MSP430G2553 为核心的数据感知、传输及控制系统。感知部分主要实现对禽舍内环境参数的检测，检测的内容包括温度、湿度、光照度、鸡蛋数量、饲料量、饮用水量。控制部分可根据禽舍内空气温湿度、光照度的不同要求，对降温风机、补光灯等设备进行控制；并定时或远程控制补充鸡饲料与水。传输部分主要采取多输入、多输出的多路径传输方式，除了具有信息输入、输出、转发功能外，还须具有利用 SD 卡进行信息暂存功能。Wifi 无线传输模块可实现信息的远程传输与控制。并设计上位机软件，属于物联网应用层，在分析全面感知信息的基础上，实现鸡舍环境因素的自动控制。

初代智能鸡舍的概念模拟图如下所示，其中左图为主视图，右图为透视图：

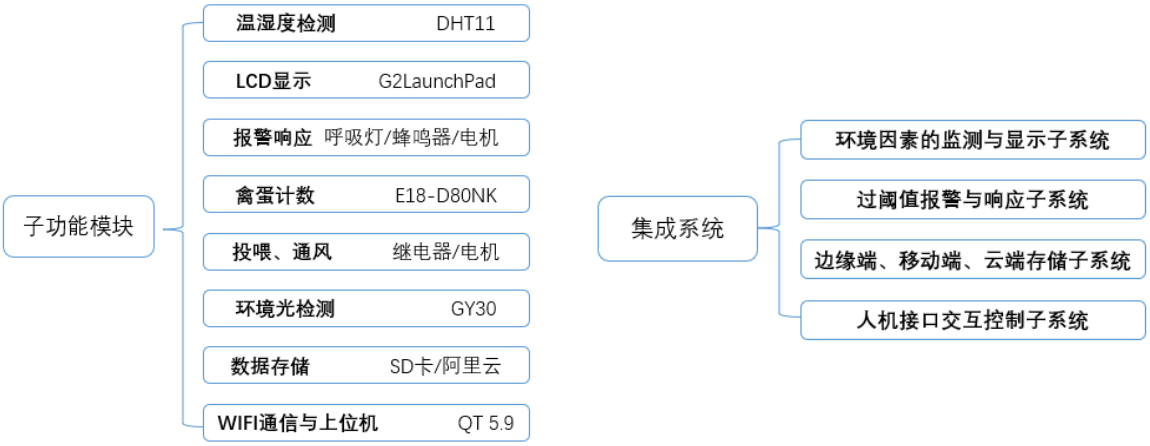


最终初代系统的结构图如下所示：



三、详细设计

该部分设计思路遵循“从子功能模块调试到系统集成设计”的原则，故分为子功能模块设计和分功能集成系统设计两部分，各部分总体展示如下图所示：



以下进行具体介绍。

（一）子功能模块设计

（1）温湿度检测模块

1.传感器选型

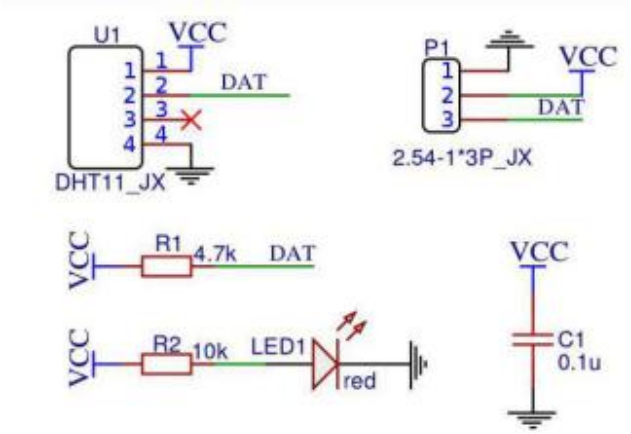
采用 DHT11 数字温湿度传感器。

它是一款含有已校准数字信号输出的温湿度复合传感器。它包括一个电阻式感湿元件和一个 NTC 测温元件。采用单线制串行接口，系统集成建简易快捷、体积小功耗低。可测相对湿度范围$5\sim95\%RH(\pm5\%RH)$，可测温度范围为$-20\sim60℃(\pm2℃)$，采样周期大于 2s/次，满足禽舍温湿度测量场景的需求。

2.模块原理

2.1 接线方式

工作时 VCC 引脚接 3.3V 工作电压，GND 引脚接电源地，DOUT 引脚为串行总线接口接 MSP430G2553 的 I/O 引脚。接线示意图如下。

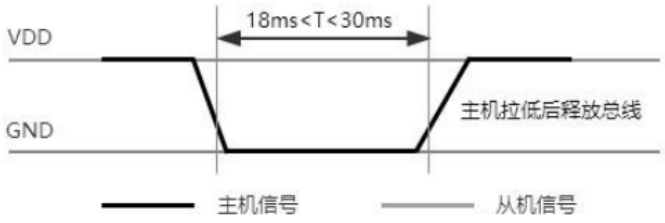


2.2 通信协议

DHT11 采用单总线通信协议，与单片机（主机）通信过程包括：主机发送起始信号，DHT11 检测并发送响应信号，DHT11 发送 40 位数据（高位先出），DHT11 发送结束信号等几个流程。

2.2.1 起始信号：

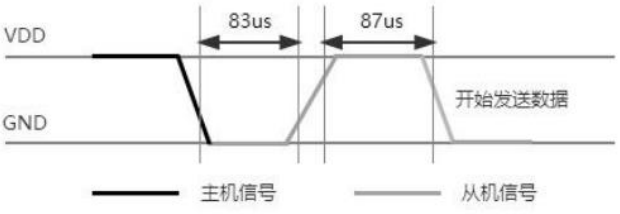
主机把数据线拉低 T（18ms<T<30ms），通知传感器准备数据。



主机发送起始信号

2.2.2 响应信号：

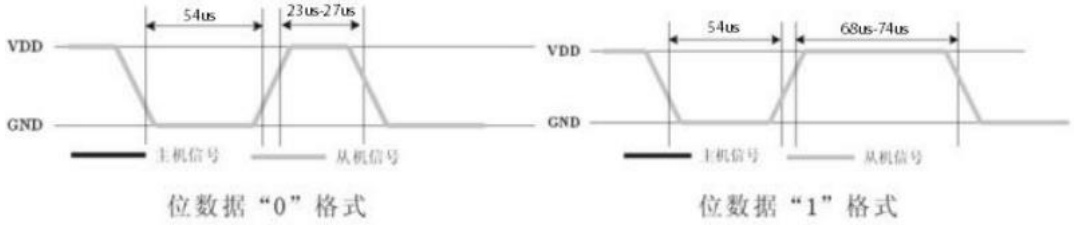
DHT11 把数据总线拉低 83us 以响应起始信号，再拉高 87us 通知主机准备接收信号。



DHT11发送响应信号

2.2.3 数据信号：

DHT11 在拉高总线 87us 后紧接着发送 40Bit 数据（高位先出）。位数据“0”：54us 的低电平和 23-27us 的高电平，位数据“1”54 微秒的低电平加 68-74 微秒的高电平。



2.2.4 结束信号：

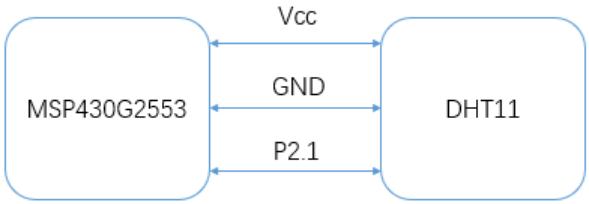
DHT11 在发送完 40Bit 数据后接着输出 54us 低电平作为结束信号。

2.3 数据解析

接收到的 40Bit 数据为：湿度高 8 位即测量湿度的整数部分，湿度低 8 位即测量湿度的小数部分；温度高 8 位测量温度的整数部分，温度低 8 位即测量温度的小数部分，温度低于 0℃的时候低八位最高位置 1；8 位校验位，若校验位＝湿度高位+湿度低位+温度高位+温度低位时，数据传输正确。

3.具体实现

根据传感器的通信协议和数据解析方法,写出 DHT11 驱动函数(DHT11.h 和 DHT11.c)，数据引脚连接 MSP430G2553 的 P2.1 引脚并且上拉一个 3KΩ 的电阻，保持数据引脚在不传输数据的时候为高电平。



4.参考资料：

DHT11-产品手册.pdf

DHT11 电容式传感器-芯片手册.pdf

DHT11 温湿度传感器模块学习手册（STM8S 版）.pdf

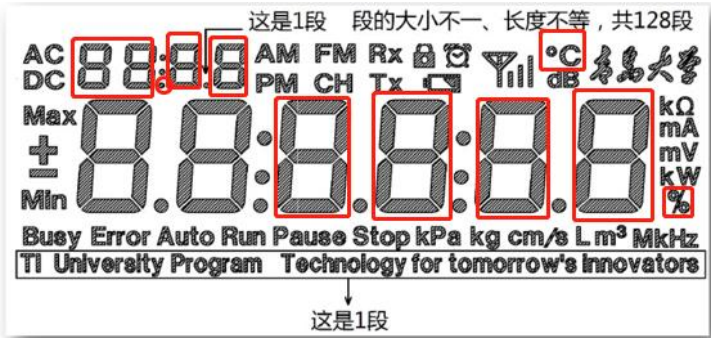
（2）LCD 显示模块

1.模块选型

使用艾研口袋拓展板 G2 LaunchPad 上的 LCD 显示屏，通过 HT1621 芯片驱动 LCD 显示屏，显示内容丰富，电压驱动功耗比较低，符合低功耗设计要求。而拓展板与 MSP430G2553 之间通过 TCA6416 芯片连接，通信协议为I^2C协议，实现对核心板的拓展。

2. 具体实现

通过查询手册，了解 LCD 的段码显示规则，根据本次设计需要采用标号为 3~10 的 8 个数字、52 和 123 两个小数点段、68 和 80 两个单位段进行本地显示温湿度，引入I²C，TCA6416 和 LCD 相关的驱动函数，对 LCD 进行控制来本地显示温湿度。



3. 参考资料：

艾研拓展板手册；

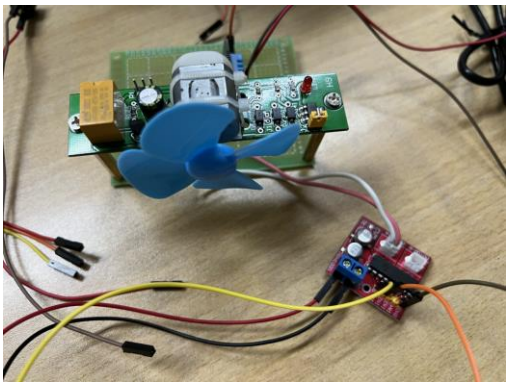
课堂 PPT：T12 基于串行总线的嵌入式系统扩展技术；

（3）电机/蜂鸣器/呼吸灯报警模块

1. 器件选择

选择 MSP430G2553 为处理芯片，MX1515 为电机驱动芯片，电机采用直流电机带动风扇作为负载，蜂鸣器采用无源蜂鸣器，呼吸灯用 LED 灯。

总体接线电路图如下：



2.模块原理

2.1 电机驱动：

MX1515 的 VCC 接电源正极，GND 接电源的负极，输入通道 IN1 和 IN2 接控制电机的驱动信号，直流电机连接到 MOTOR1 输出引脚。

查阅芯片手册，当 IN1 接高电平，IN2 接低电平的时候电机反转；当 IN1 接低电平，IN2 接高电平的时候电机正转。根据设计场景的需求，需要电机正转且可以调速，所以将 IN1 引脚接低电平，IN2 引脚接 MSP430G255 输出的控制电机转速的 PWM 信号。控制电机的 PWM 要相对稳定，可以由 MSP430G2553 的定时器产生。

2.2 蜂鸣器驱动：

无源蜂鸣器需要一定频率的方波信号输入才能引起振动发声，因而将无源蜂鸣器的一端接地，另一端 MSP430G255 输出的响应频率的方波信号。

2.3 呼吸灯：

LED 灯的一端接低电平，另一端接 MSP430G2553 输出的 PWM 信号，通过连续控制 PWM 信号的占空比大小（由小变大，再由大变小）实现呼吸效果。

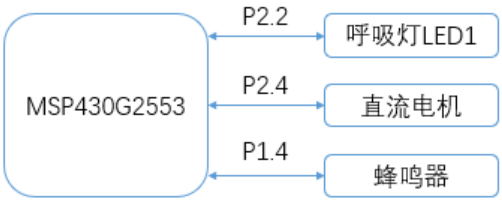
3.功能设计

实现温度预警与预警响应。

通过获取 DHT11 数字温湿度传感器传到 MSP430G2553 的温度值，与上位机或者程序内设定的参考温度作比较，设定不同的预警等级：在实际禽舍养殖中的适宜温度为 25℃，当高于适宜温度 5℃左右时设为第一级预警，高于适宜温度 10℃的时候设为第二级预警。实际温度未达第一级预警时，呼吸灯、降温风扇、蜂鸣器都不工作；温度达到第一级预警的时候启动第一级预警响应，呼吸灯慢闪，蜂鸣器以较低频率报警，降温风扇以较低转速工作来降温；温度达到第二级预警的时候启动第二级预警响应，呼吸灯快闪，蜂鸣器以较高频率报警，降温风扇以较高转速工作来降温。

4.具体实现

根据模块的实现原理，写出相应的初始化与驱动函数（APP.h 和 APP.c），MSP430G2553 的 P2.2 引脚输出占空比连续变化的 PWM 信号，连接呼吸灯，P2.4 引脚输出占空比较稳定的 PWM 信号，连接 MX1515 的 IN2 引脚驱动电机，P1.4 引脚输出不同频率的方波信号驱动无源蜂鸣器工作。其中 PWM 信号由定时器 A1 产生，方波信号由定时器 A0 产生。



5.参考资料：

msp430mcu(G2 系列）结构及原理（中文手册）.pdf

msp430g2553 datasheet.pdf

MX1515 芯片手册.pdf

（4）禽蛋计数模块

1.传感器选型

采用 E18-D80NK-N 红外避障传感器，这是一种集发射与接收于一体的光电传感器，发射光经过调制后发出，接收头对反射光进行解调输出。有效的避免了可见光的干扰。

该传感器具有探测距离远、受可见光干扰小、价格便宜、易于装配、使用方便等特点，可以广泛应用于机器人避障、流水线计件等众多场合，满足禽舍的传送带鸡蛋计数场景的需求。

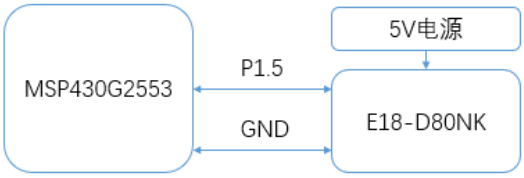
2.模块原理

E18-D80NK-N 传感器内部有一个调制发射器和一个接收器，原理图如下。

当传感器面前没有障碍物的时候，接收器不能接收到反射光，输出为高电平，当有障碍物的时候，接收器可以接收到反射光，输出为低电平，通过改变传感器上面的阈值旋钮可以改变传感器的测量范围，MSP430G2553 通过对读入的低电平出现次数进行计数，可以得到经过传感器面前的鸡蛋个数。

3.具体实现

传感器的棕色线接 5V 电源的正极，蓝色线接 5V 电源的负极，黑色线接 MSP430G2553 的 P1.5 引脚并设置为输入，每当有低电平到来计数器增加 1。



4.参考资料：

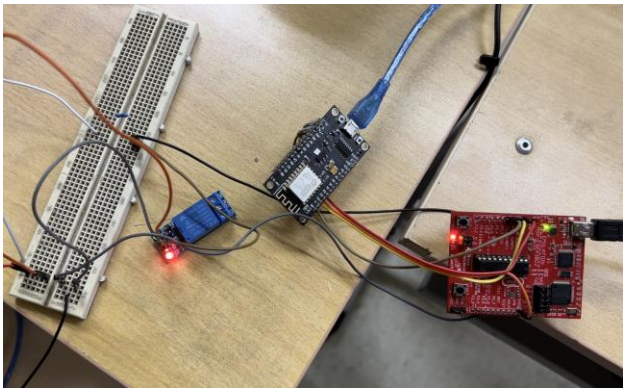
E18-D80NK 说明书.pdf

（5）投喂/通风控制模块

1.器件选择

选择 1 路 5V 继电器模块（低电平触发），该继电器具有电源和继电器动作指示，吸合亮，断开不亮。当有低电平信号的时候巩固端与常开端会导通，可以直接控制各种设备和负载。继电器的尺寸小，质量轻，供电电压为 5V，触发电压为 3.5V，可以作为模拟投喂模块的开关。

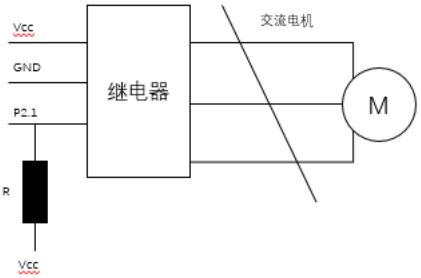
实物连接图如下所示：



2.模块原理

继电器线圈加电压，动触点与静触点吸合，电路导通；继电器线圈断电，动触点与静触点释放，电路切断，从而达到低压控制高压的目的。

投喂模块的原理图如下：



信号线通过上拉电阻接到 VCC（5V），控制 MSP430G2553 的引脚的状态来控制继电器的吸合与释放：当引脚设置为输出且输出为低电平的时候继电器 IN 引脚获得低电位，触点吸引，电路导通；当引脚设置为输入的时候为高阻态，继电器的 IN 引脚获得高电位，触点释放，电路断开。通过控制引脚状态的维持时间进而控制投喂时机。

3.具体实现

继电器的 IN 引脚与 MSP430G2553 的 P2.1 引脚连接，继电器 VCC 和 GND 引脚分别接 5V 直流电源的正负极，设置定时器的定时间隔以及分中断位的状态来控制电路导通与断开。共实现了两种控制模式：

（I）方式 1：模拟每隔一定时间（程序设计为 7s）进行自动投喂操作，循环往复；

（II）方式 2：远程控制投喂电机的开机关闭，实现手动投喂。

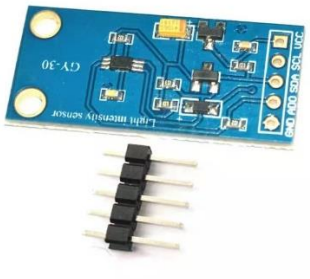
4.参考资料：

单路 5V 继电器模块使用说明.pdf

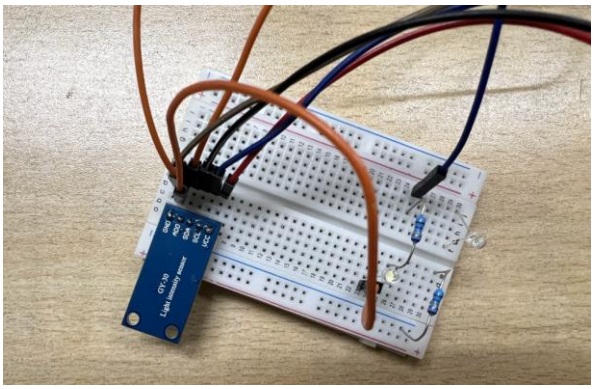
（6）环境光测量及补光模块

1.器件选择：

采用的是 GY-30 光敏传感器，它是一款基于 I²C 通信的 16 位的数字型传感器，该模块主要是以 BH1750 数字型光强感应芯片为核心及一些外围驱动电路。BH1750 是一款内部集成了光电转换、ADC 转换、IIC 信号转换等电路的芯片，省去了复杂信号处理电路，即能保持良好的稳定型又节省空间。具有灵敏性高、稳定性强等特点，满足鸡舍内环境光照测量的要求。



实物连接如下所示：

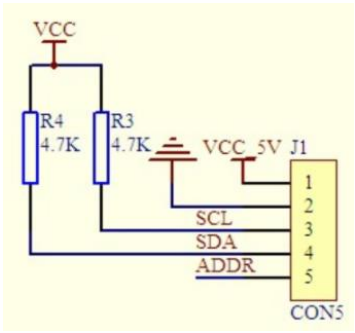


2.模块原理：

2.1 接线方式：

ADDR 在模块中接了 GND，可接可不接，默认地址是 0x46。VCC 接 msp430g2553 上的 3.3V 电源 VCC 即可，时钟线 SCL、数据线 SDA 分别与 MCU 的 IIC 总线引脚相连，两条总线都用上拉电阻拉高电平。

电路连接原理图如下所示：

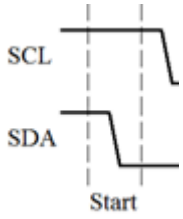


2.2 通信协议：

遵循 IIC 总线通信协议，具体信号定义如下所示：

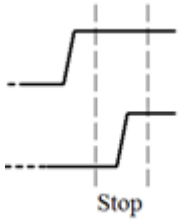
2.2.1 起始信号：

SCL 线为高电平期间，SDA 线由高电平向低电平的变化表示起始信号。



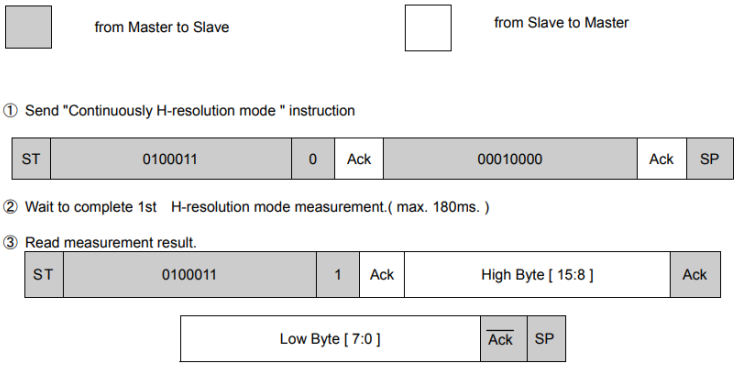
2.2.2 终止信号：

SCL 线为高电平期间，SDA 线由低电平向高电平的变化表示终止信号



2.2.3 读写协议：

GY30 内部的 IIC 通信读写协议有自己的规定，查询手册可知。



①模式配置：

当总线上发送起始信号后，向从机发送写指令 0x46，等待从机应答。当应答有效时再发送模式配置信号（0x10 配置成连续 H-resolution 工作模式），等待到有效应答信号后向从机发送终止信号。

②模式转换延时：

配置完模式后，程序延时 120-180ms 的时间。

③读取 16 位光强测量值：

当总线上发送起始信号后，再发送读指令 0x47，等待从机应答。当应答有效时，接收高 8 位数据，向从机发送应答回复，再接收低 8 位数据，向从机发送无应答信号后发送终止信号。

2.3 数据解析：

读取到的 16 位光强测量值是一个数字量,需要转化成模拟量,具体的转换方法以“1000 0011”“1001 0000”为例，实际光强值：

$$(2^{15}+2^{9}+2^{8}+2^{7}+2^{4})/1.2 \fallingdotseq 28067\,[\mathrm{lx}]$$

3.具体实现：

根据传感器的通信协议和数据解析方法，写出 GY30 驱动函数（init.h），时钟线 SCL 引脚连接 MSP430G2553 的 P2.3 引脚，数据线 SDA 引脚连接 MSP430G2553 的 P2.4 引脚并且各上拉一个 3kΩ的电阻，保持数据引脚在不传输数据的时候为高电平。此外在此模块的后续应用设计中，采用了 P2.2 引脚控制 LED 进行补光，电路中串联一个 56Ω的电阻进行限流。

4.参考资料：

传感器技术手册 BH1750FVI.pdf

（7）SD 卡读写模块

1.器件选择：

利用艾研拓展板 G2 LaunchPad 上的 TF 卡拓展功能区进行 SD 卡扇区的读写操作，但只能选择 2GB 以下的 SD 卡。

2.模块原理：

SD 卡 SPI 模式需要 SD 卡作为从机，MCU 作为主机进行通信。可以采用三线 SPI(SCLK,MISO,MOSI)外加一个片选信号线 CS 来进行控制和通信。其各动作信号的定义如下介绍。

2.2.1 SD 卡复位：

SD 卡在上电后是处于 SD 模式，RESET 命令使其进入 SPI 模式。具体流程为:

1、拉高 CS，发送至少 74 个时钟周期使 SD 卡稳定；

2、拉低 CS，发送 CMD0；

3、由第一节知 CMD0 返回 R1，需要收到回应 0x01 表示 in idle state；

4、拉高 CS，附加发送 8 个时钟；

2.2.2 SD 卡初始化：

SD 卡初始化可以使用 CMD1 和 ACMD41 两种方式。具体流程为：

1、发送 CMD55，收到 0x01 表示 in idle state；

2、发送 ACMD41，返回 R1，在这里 R1 的 in idle state 用来表示 SD 卡是否处于初始化状态，因此需要循环读取直到 0x00；

2.2.3 SD 卡读动作：

1、发送 CMD17，收到 0x00 表示发送成功；

2、连续读取知道读到 0xFE；

3、读一个 block (通常为 512 字节)；

4、读两个 CRC 字节；

2.2.4 SD 卡写动作：

1、发送 CMD24，收到 0x00 表示发送成功；

2、发送若干时钟；

3、发送写开始标志 0xFE；

4、发送一个 block (通常为 512 字节)；

5、发送两个 CRC 字节；

6、连续读直到读到二进制 xxx00101 表示数据写入成功；

7、连续读忙检测，直到读到 0xFF 表示操作完成；

3.具体实现：

利用艾研拓展板附带的程序“11_SPI_SD”改写，SD 卡与 SPI 协议之间的“接口”写在了 SD_HardWare 文件中，相当于是把 G2 的 SPI 和 SD 卡联系起来，具体实现过程如下：
①配置 DCO 频率—>设置 GPIO 方向—>调用库函数 USCI_A0_init()实现 SPI 初始化—>调用库函数 SD_Init()实现 SD 卡初始化。
②调用写 SD 函数 My_Write_Data()，将温湿度数值写入 SD 指定扇区“SD_SECTOR_ADDR”，为了演示方便此处程序只读写 16 个字节的数据。
③调用 SD 读函数 My_Read_Data()，从 SD 卡中指定扇区“SD_SECTOR_ADDR”读取数据存入全局变量数组 DATA[16]。

此外为了可视化程序执行时的顺序，利用 P1.0 控制的 LED 灯的不同动作来观察：初始化时 LED1 熄灭，闪烁三下（每次延时 1s）开始接收 DHT11 的数据，接收过程中闪烁一次；再进行 SD 卡初始化，初始化成功后 LED1 常亮，写完后再闪烁一下，开始读取数据，读完后闪烁一下，最后保持 LED1 常量代表整个过程完成。

4.参考资料：

拓展板手册：AY-G2PL KIT_用户手册.pdf

（8）WIFI 通信模块

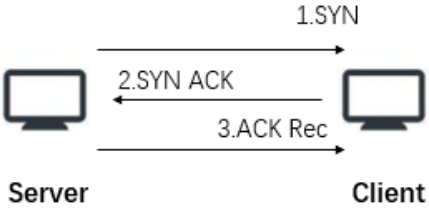
1.芯片选择

ESP8266，型号：ESP-12F。优点是 ESP8266 的价格实惠，功能强大，将处理器和 WiFi 芯片集成在一起，具有 GPIO、PWM、I2C、UART、1-Wire、ADC 等功能，开发体系成熟，可以满足大多数无线开发场景需求，也适用于禽舍的智能监控系统的无线通信需求。

2.模块原理

本系统实现上位机与硬件之间的数据交换，可以采用 TCP 通信协议或者 UDP 服务，虽然 UDP 服务有较好的实时性，工作效率比较高，适用于实时性和高速传输有较高要求的通信或广播通信，但是 TCP 相对来说连接稳定，数据传输不会出错，服务可靠。结合系统的应用场景，选择连接稳定的 TCP 协议。

TCP 协议中包含两个角色：TCP client（客户端）和 TCP server（服务器），概念图如下。



在此系统中，选择将 ESP8266 设置为 TCP server，将上位机设置为 TCP client，进行通信。ESP8266 建立局域内的 server，上位机作为 client 通过路由热点向 ESP8266 请求数据，8266 解析收到的数据返回响应的数据。由于 ESP8266 通过 UART 串口与 MSP430G2553 进行通信，因而可以做到通过上位机发送指令，ESP8266 解析指令并传到 MSP430G2553，然后执行机构做出响应的应答。

3.模块实现

ESP8266 通过以下一系列操作，设置成为 sever：

1.ESP8266 以 STA 模式连接至路由热点。

2.创建并启用一个 TCP sever，并设置最大接入 TCP client 的数量限制。

3.进行通信，检测并解析 TCP client 传来的指令，通过串口传送到 MSP430G2553，并将 MSP430G2553 的响应信号传输至 TCP client。

上位机通过以下一系列操作，设置成为 client：

1.建立 TCPSocket 对象 socket；

2.利用 connecthost(ip，port)函数连接至服务器。

3.通过 TCP 通信进行动作的设定。

（9）上位机模块

1.构思

独立开发一个上位机，功能类似于常用的上位机软件，在此基础上进行功能拓展。

2.设计流程

需求分析->ui 界面设计->按钮动作编程设定

3.可选方案及选择

（1）MATLAB GUI：

MATLAB 采用拖拽模块布局界面的方式比较方便，但是内部可调用的库比较少，可实现的功能比较有限，仅可以在 MATLAB 内部运行，跨平台应用适配性较差。

（2）QT Creator/C++：

新版本的 QT Creator 兼顾了 MATLAB 拖拽模块设计的优点，运用 C++从底层进行开发，封装机制使得 Qt 的模块化程度非常高，效率高，可以做到任何需要的功能，并且跨平台适配特性极强。

（3）java:

用起来比较方便，但效率略低于 QT,要做好看的界面不容易，并且语言需要重新学习。

（4）Python Tkinter:

纯代码设计界面，效率低，界面简单，难以实现复杂的功能。

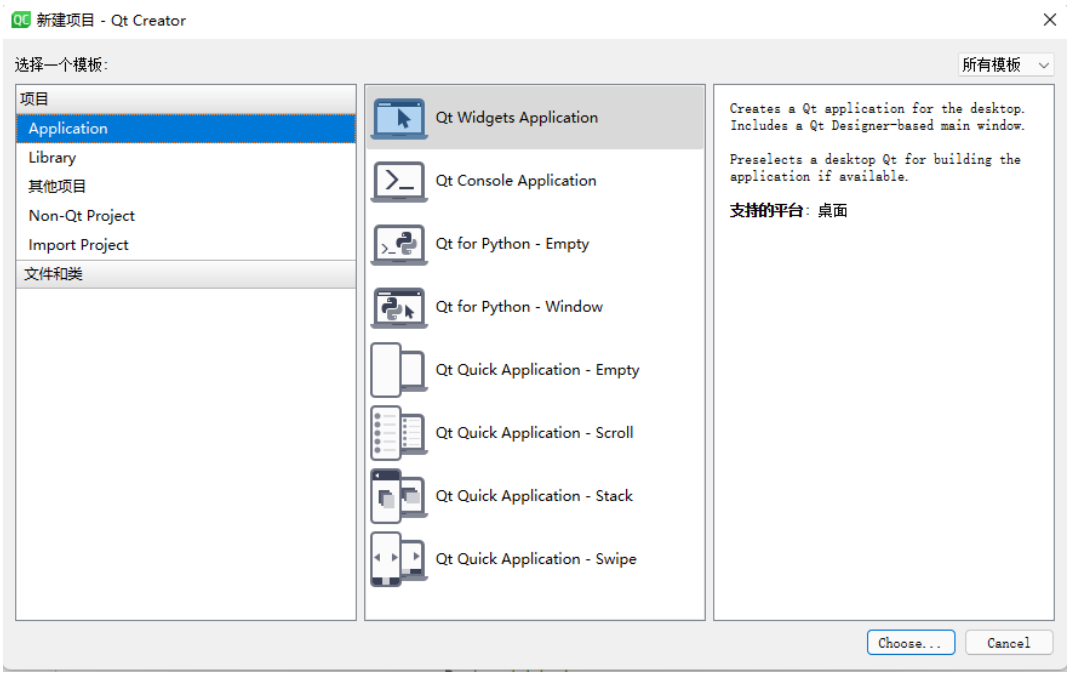
综上考虑最终使用 QT Creator5.9 进行上位机的设计。

4.详细过程

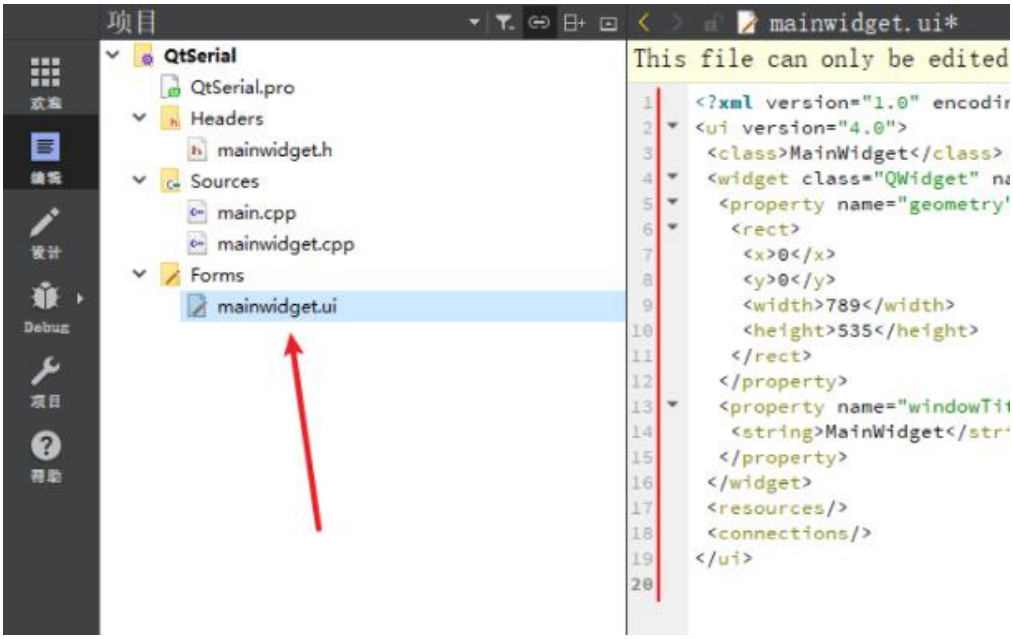
（I）UI 界面设计

（1）布局 UI 界面：

创建 QMainWindow 工程，进行初始化配置。

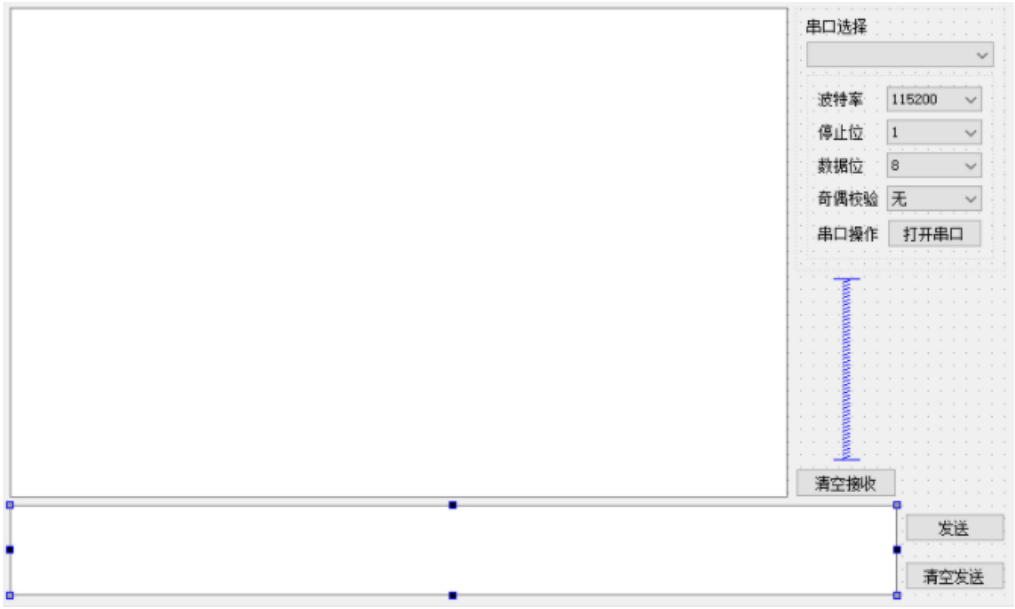


在 Qt 的 UI 界面编辑器来设计界面，**不以纯代码的形式来进行界面的设计**。



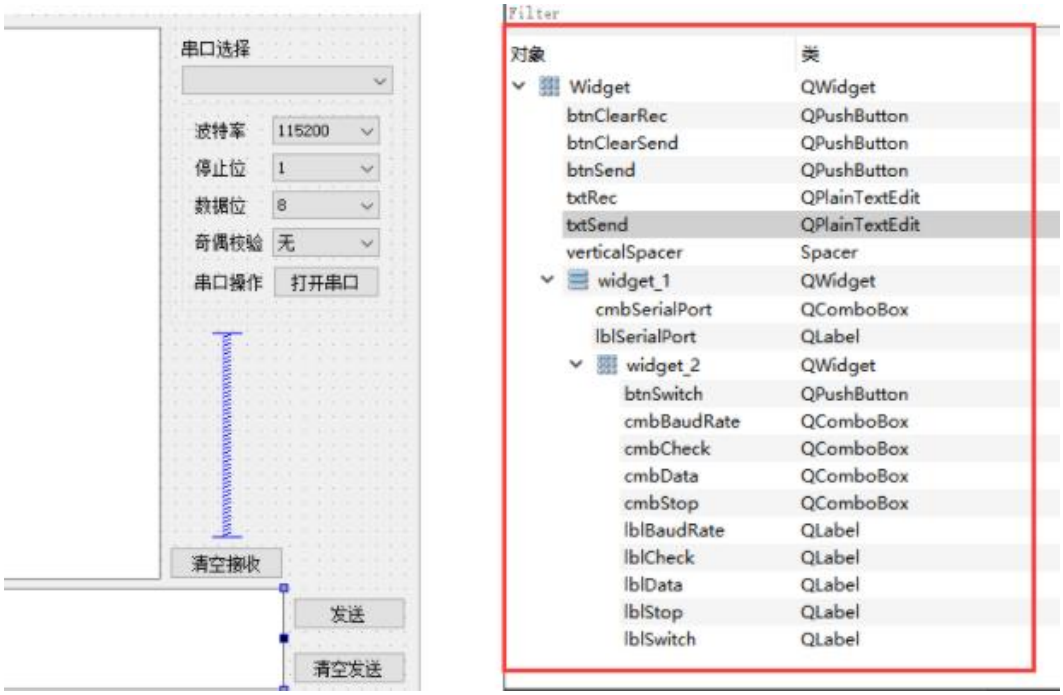
（2）界面布局：

拖入可编辑文本框（Plain Text Edit），作为串口数据的接收显示和发送框。再拖入下拉菜单（Combo Box）、文本标签（Label）和按钮（Push Button）等进行功能设计，初代界面布局如下所示。

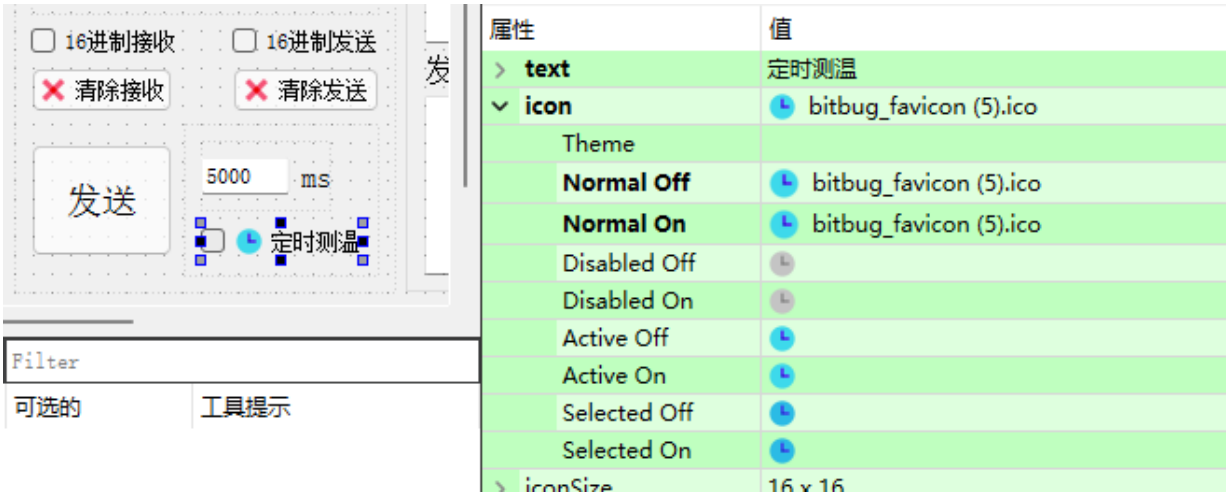


（3）修改控件名称和配置控件属性：

为了后续编程调用和调试，在右侧编辑栏将控件的命名统一，方便管理和程序书写。



在属性配置栏配置字体、控件属性、图标等。



（4）控件分区：

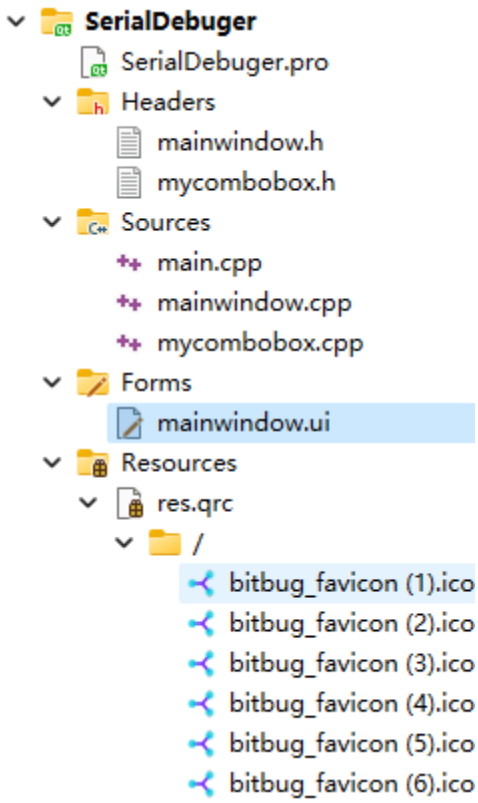
为了人机交互的便捷性，利用窗口组(Group Box)将功能控件分区，共划分成左右两个界面区，分别是 UART 通信区和 WIFI/TCP 通信区。此外，在界面的最下方还设计了 Lable 区用于展示收发的字符数和作者个人信息。

其中，UART 通信区内进行 UART 通讯有关动作，包括实时时间显示、温湿度显示、串口配置选择、上位机收发缓存区、各子功能部件等。WIFI 通信区包括了网关配置区，WIFI 收发缓存区和各子功能部件等。

最终实现的界面布局如下图所示。

（II）控件动作代码实现

QT UI 内的每一个控件都相当于一个槽，它们的动作由槽函数来实现。项目 workspace 中包含了项目管理文件（.pro），用于记录项目的一些设置，以及项目包含文件的组织管理；头文件(.h)、源文件(.cpp)，分别实现类和对象的构建；界面绘制文件(.ui)，用于实现窗口内控件的布局；资源文件（Resources），用于储存设计界面所需的图标等文件。



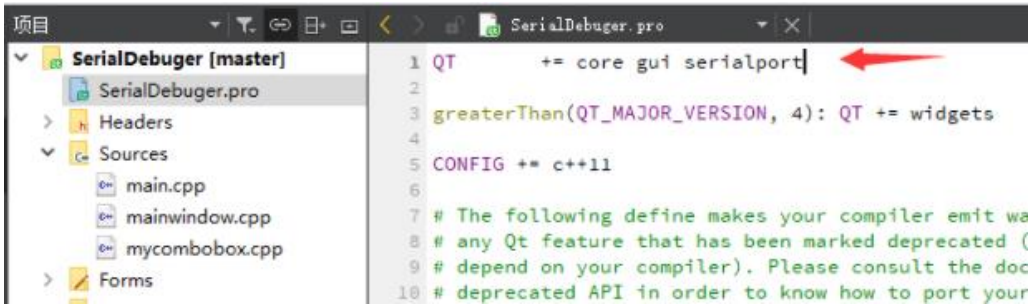
C++类的构造函数中添加的代码仅在初始化界面时调用，其他子函数在检测到槽信号时调用，以下对每个动作的进行实现进行详细的介绍。

（1）添加串口类，自动扫描可用串口

功能：扫描可用的串口

实现过程：

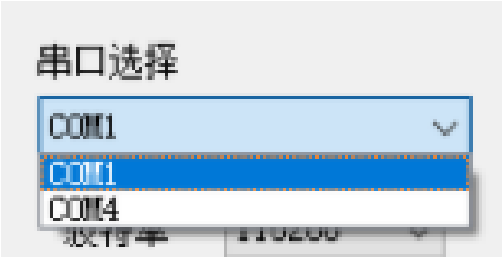
在工程 .pro 文件中，添加 "QT += core gui serialport"，以支持串口功能。



在窗口的构造函数中，添加自动扫描可用串口代码，由于代码位于窗口构造函数中，故仅在应用刚打开时扫描串口。以后会增加点击下拉框时扫描，需要对鼠标点击事件进行重写。

```cpp
// 扫描可用串口
void myComboBox::scanActivePort()
{
    // 先清空列表项，防止多次刷新后重叠
    clear();
    // 串口端口号列表
    QStringList serialPortName;
    // 自动扫描当前可用串口，返回值追加到字符数组中
    foreach(const QSerialPortInfo &info, QSerialPortInfo::availablePorts())
    {
        //serialPortName << info.portName();// 不携带有串口设备信息的文本
        // 携带有串口设备信息的文本
        QString serialPortInfo = info.portName() + ": " + info.description();// 串口设备信息，芯片/驱动名称
        //QString serialPortInfo = info.portName() + ": " + info.manufacturer();// 串口设备制造商
        //QString serialPortInfo = info.portName() + ": " + info.serialNumber();// 串口设备的序列号，没什么用
        //QString serialPortInfo = info.portName() + ": " + info.systemLocation();// 串口设备的系统位置，没什么用
        serialPortName << serialPortInfo;
    }
    // 可用串口号，显示到串口选择下拉框中
    this->addItems(serialPortName);
}
```

如果电脑连接了串口设备，运行 Qt 应用，点击下拉框，会看到可用的串口号。



（2）串口初始化配置：

功能：点击 ui 中的 ″打开串口″ 按钮，初始化串口参数，并按照设置好的参数打开串口。

实现过程：

在″打开串口″按钮位置，右键，转到槽....。编写串口配置代码。利用语句 ui->“控件名称”->currentText()将字符串，转换为可以代码设置的枚举，根据当前值，设置参数。之后打开对应串口端口，以波特率设定为例：

```
void MainWindow::on_btnSwitch_clicked()
{

    QSerialPort::BaudRate baudRate;
    QSerialPort::DataBits dataBits;
    QSerialPort::StopBits stopBits;
    QSerialPort::Parity   checkBits;

    // 获取串口波特率
    // 有没有直接字符串转换为 int的方法？？？
    //baudRate = ui->cmbBaudRate->currentText().toInt();

    /*如果主窗口的下拉菜单值是9600，则设置波特率9600*/

    if(ui->cmbBaudRate->currentText() == "9600")
    {
        baudRate = QSerialPort::Baud9600;
    }
    else if(ui->cmbBaudRate->currentText() == "38400")
    {
        baudRate = QSerialPort::Baud38400;
    }
    else if(ui->cmbBaudRate->currentText() == "115200")
    {
        baudRate = QSerialPort::Baud115200;
    }
```

为了避免串口被占用，或者其他错误，我们对“打开串口”做相应的处理：如果串口被占用，会弹出错误提示框。

为了防止端口在打开时，被篡改参数，避免改动串口号，做相应的处理：端口在打开后，下拉框是灰色不可选的。最终实现打开串口/关闭串口的功能转换与翻转显示。

```
if(ui->btnSwitch->text() == "打开串口")
{
    //如果打开成功，反转打开按钮显示和功能
    if(mySerialPort->open(QIODevice::ReadWrite) == true)
    {
        ui->btnSwitch->setText("关闭串口");
        // 让端口号下拉框不可选setEnabled，避免误操作（选择功能不可用，控件背景为灰色）
        ui->cmbSerialPort->setEnabled(false);
        ui->cmbBaudRate->setEnabled(false);
        ui->cmbStop->setEnabled(false);
        ui->cmbData->setEnabled(false);
        ui->cmbCheck->setEnabled(false);
    }
    //打开失败，无变化，并且弹出错误对话框。
    else
    {
        QMessageBox::critical(this, "Error!", "串口打开失败");
    }
}
```

（3）接收串口数据：

功能：接收到串口数据后，将其显示到接收文本框中。

实现过程：

构造函数中建立信号槽，将串口读准备信号，关联接收区显示处理槽函数。

```
// 串口接收，信号槽关联
connect(mySerialPort, SIGNAL(readyRead()), this, SLOT(serialPortRead_Slot()));
```

为了使文本全部显示在缓冲区，且每次读到的数追加显示，文本框内容使用插入显示 insertPlainText()，并在每次追加后移动光标。

```
QByteArray str1 = QByteArray::fromHex(txtBuf.toUtf8());
// 文本控件清屏，显示新文本
ui->txtRec->clear();
ui->txtRec->insertPlainText(str1);
// 移动光标到文本结尾
ui->txtRec->moveCursor(QTextCursor::End);
```

（4）发送串口数据：

功能：通过上位机发送串口数据给 MCU。

实现过程：

利用串口库的 write 函数将 QByteArray 型数据写入串口，在其中可以进行例如进制、编码类型等的配置，后续会进行一些逻辑上的功能改进。

```
// 字符串形式发送
sendData = ui->txtSend->toPlainText().toLocal8Bit().data();

// 16进制发送
sendData = QByteArray::fromHex(ui->txtSend->toPlainText().toUtf8()).data();
```

（5）清空收区、发区文本内容

功能：清空 ui 上 uart 接收和发送缓冲区的内容

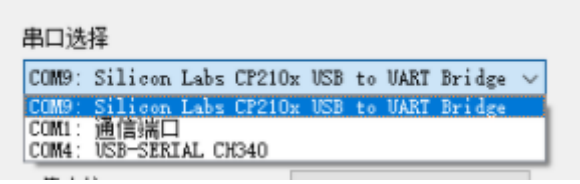
实现过程：

利用功能库的 clear 清空可编辑文本(QPlainTextEdit)内的内容。此处以发送区为例，由于后续包括了 Label 区的字符数统计，因此清空缓冲区时连带着清空了计数值。

```
//清空发送区
void MainWindow::on_btnClearSend_clicked()
{
    ui->txtSend->clear();
    // 清除发送字节计数
    sendNum = 0;
    // 状态栏显示计数值
    setNumOnLabel(lblSendNum, "S: ", sendNum);
}
```

（6）点击下拉框扫描可用串口

功能：重写下拉框 ComboBox 的鼠标点击事件，实现点击下拉框扫描可用串口。



实现过程：

新建类，继承 QComboBox，重写鼠标点击事件。

代码中，对一些方法和属性的说明：

scanActivePort：扫描可用串口函数。由构造函数和鼠标点击事件使用。

QSerialPortInfo::portName：串口端口名称。

QSerialPortInfo::description：串口端口硬件/驱动信息。

showPopup：ComboBox 弹出下拉框动作。

```
myComboBox::myComboBox(QWidget *parent) : QComboBox(parent)
{
    // 扫描可用串口
    scanActivePort();
}

// 扫描可用串口
void myComboBox::scanActivePort()
{
    // 先清空列表项，防止多次刷新后重叠
    clear();
    // 串口端口号列表
    QStringList serialPortName;
    // 自动扫描当前可用串口，返回值追加到字符数组中
    foreach(const QSerialPortInfo &info, QSerialPortInfo::availablePorts())
    {
        //serialPortName << info.portName();// 不携带有串口设备信息的文本
        // 携带有串口设备信息的文本
        QString serialPortInfo = info.portName() + ": " + info.description();// 串口设备信息，芯片/驱动名称
        //QString serialPortInfo = info.portName() + ": " + info.manufacturer();// 串口设备制造商
        //QString serialPortInfo = info.portName() + ": " + info.serialNumber();// 串口设备的序列号，没什么用
        //QString serialPortInfo = info.portName() + ": " + info.systemLocation();// 串口设备的系统位置，没什么用
        serialPortName << serialPortInfo;
    }
    // 可用串口号，显示到串口选择下拉框中
    this->addItems(serialPortName);
}
```

```
// 重写鼠标点击事件
void myComboBox::mousePressEvent(QMouseEvent *event)
{
    if(event->button() == Qt::LeftButton)
    {
        // 扫描可用串口
        scanActivePort();

        // 弹出下拉框
        showPopup();
    }
}
```

（7）16 进制接收、16 进制发送

功能：实现接收和发送的数据在 UART 缓冲区内转化成 16 进制显示。

实现过程：

以 16 进制接收为例，根据"16 进制接收"复选框的值，选择性的将每次新接收到的数据，进行 16 进制显示转换。

```
// 串口接收显示，槽函数
void Widget::serialPortRead_Slot()
{
    /*QString recBuf;
    recBuf = QString(mySerialPort->readAll());*/

    QByteArray recBuf;
    recBuf = mySerialPort->readAll();

    // 判断是否为16进制接收，将以后接收的数据全部转换为16进制显示（先前接收的部分在多选框槽函数中进行转
    if(ui->chkRec->checkState() == false)
    {
        // 在当前位置插入文本，不会发生换行。如果没有移动光标到文件结尾，会导致文件超出当前界面显示范围
        ui->txtRec->insertPlainText(recBuf);
    }else{
        // 16进制显示，并转换为大写
        QString str1 = recBuf.toHex().toUpper();//.data();
        // 添加空格
        QString str2;
        for(int i = 0; i<str1.length (); i+=2)
        {
            str2 += str1.mid (i,2);
            str2 += " ";
        }
        ui->txtRec->insertPlainText(str2);
        //ui->txtRec->insertPlainText(recBuf.toHex());
    }
```

（红框）勾选 "16 进制接收" 复选框时，将先前接收到的 asc2 显示部分，全部转换为 16 进制显示。

（蓝框）取消勾选 "16 进制接收" 复选框时，将接收区的 16 进制显示，全部转换为 asc2 显示。

```
void MainWindow::on_chkRec_stateChanged(int arg1)
{
    // 获取文本字符串
    QString txtBuf = ui->txtRec->toPlainText();

    // 获取多选框状态，未选为0，选中为2
    // 为0时，多选框未被勾选，接收区先前接收的16进制数据转换为asc2字符串格式
    if(arg1 == 0){
        QByteArray str1 = QByteArray::fromHex(txtBuf.toUtf8());
        // 文本控件清屏，显示新文本
        ui->txtRec->clear();
        ui->txtRec->insertPlainText(str1);
        // 移动光标到文本结尾
        ui->txtRec->moveCursor(QTextCursor::End);

    }else{// 不为0时，多选框被勾选，接收区先前接收asc2字符串转换为16进制显示
        QByteArray str1 = txtBuf.toUtf8().toHex().toUpper();
        // 添加空格
        QByteArray str2;
        for(int i = 0; i<str1.length (); i+=2)
        {
            str2 += str1.mid (i,2);
            str2 += " ";
        }
        // 文本控件清屏，显示新文本
        ui->txtRec->clear();
        ui->txtRec->insertPlainText(str2);
        // 移动光标到文本结尾
        ui->txtRec->moveCursor(QTextCursor::End);

    }
```

（8）字符串接收统计：

功能：接收和发送的字节数计数

实现过程：

在 mainwindow.h 文件中创建收发数属性，再创建两个标签指针，用于显示收发数目。

```
44  private:
45      Ui::MainWindow *ui;
46
47      QSerialPort *mySerialPort;
48
49      // 发送、接收字节计数
50      long sendNum, recvNum;
51      QLabel *lblSendNum;
52      QLabel *lblRecvNum;
53      void setNumOnLabel(QLabel *lbl, QString strS, long num);
54
```

在 mainwindow.c 构造函数中创建两个标签，用于显示收发数量，并将标签添加到底部状态栏中。

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    setWindowTitle("Qt Serial Debugger");

    // 发送、接收计数清零
    sendNum = 0;
    recvNum = 0;
    // 状态栏
    QStatusBar *sBar = statusBar();
    // 状态栏的收、发计数标签
    lblSendNum = new QLabel(this);
    lblRecvNum = new QLabel(this);
    // 设置标签最小大小
    lblSendNum->setMinimumSize(100, 20);
    lblRecvNum->setMinimumSize(100, 20);
    //statusBar()->showMessage("留言", 5000);// 留言显示，过期时间单位为ms，过期后不再有显示
    //statusBar()->setSizeGripEnabled(false); // 是否显示右下角拖放控制点，默认显示
    //statusBar()->setStyleSheet(QString("QStatusBar::item{border: 0px}")); // 设置不显示label的边框
    //lblSendNum->setAlignment(Qt::AlignHCenter);// 设置label属性
    //sBar->addPermanentWidget();//addSeparator();// 添加分割线，不能用
    // 状态栏显示计数值
    //lblSendNum->setText("S: 0");
    //lblRecvNum->setText("R: 0");
    setNumOnLabel(lblSendNum, "S: ", sendNum);
    setNumOnLabel(lblRecvNum, "R: ", recvNum);
    // 从右往左依次添加
    sBar->addPermanentWidget(lblSendNum);
    sBar->addPermanentWidget(lblRecvNum);
```

在串口接收槽函数中，添加接收数目统计代码，并将数量同步显示到底部状态栏。

```
void MainWindow::serialPortRead_Slot()
{
    /*QString recBuf;
    recBuf = QString(mySerialPort->readAll());*/
    QByteArray recBuf;

    recBuf = mySerialPort->readAll();
    // 接收字节计数
    recvNum += recBuf.size();
```

发送计数在串口发送槽函数中，对成功发送出的字节数量进行累加。

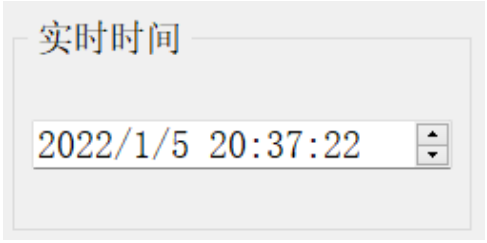
```
// 发送按键槽函数
// 如果勾选16进制发送，按照asc2的16进制发送
void MainWindow::on_btnSend_clicked()
{
    QByteArray sendData;
    // 判断是否为16进制发送，将发送区全部的asc2转换为16进制字符串显示，发送的时候转换为16进制发送
    if(ui->chkSend->checkState() == false){
        // 字符串形式发送
        sendData = ui->txtSend->toPlainText().toLocal8Bit().data();
    }else{
        // 16进制发送
        sendData = QByteArray::fromHex(ui->txtSend->toPlainText().toUtf8()).data();
    }

    // 如发送成功，会返回发送的字节长度。失败，返回-1。
    int a = mySerialPort->write(sendData);
    // 发送字节计数并显示
    if(a > 0)
    {
        // 发送字节计数
        sendNum += a;
        // 状态栏显示计数值
        setNumOnLabel(lblSendNum, "S: ", sendNum);
    }
```

清空发送和接收计数则写在清空发送区和接收区的槽函数中，直接将计数变量清零即可。

（9）实时时间显示：

功能：显示北京时间的年月日时分秒



实现过程：

利用内置的 currentDatetime()实现读取实时日期和时间，在 QDateTimeEdit 窗口中显示读取的值。

```
//显示时间
void MainWindow::timerEvent(QTimerEvent *event)
{
    //获取当前时间
    QDate currentdate = dateTime.currentDateTime().date();
    QTime currenttime = dateTime.currentDateTime().time();
    if(event->timerId() == timerid){
        ui->dateTimeEdit->setDate(currentdate);
        ui->dateTimeEdit->setTime(currenttime);
    }
}
```

（10）温湿度 LCD 显示

功能：实现从串口读取到的温湿度数据在 LcdNumber 控件中显示。

实现过程：

在接收串口数据的槽函数 MainWindow::serialPortRead_Slot()中，将 16 进制接收时收到的温湿度（整数位、小数位）数据转换成整数类型，再进行到十进制转换，转换完成后进行 QString 类型拼接，利用 display()函数进行显示。

```
ui->lcdTemp->display(temp);
ui->lcdHumi->display(humi);
```

由于接收到的数据长度不定，有时候是一个字节，有时候是多个字节（最多四个字节），如下所示：

```
"0A "
"00 "
"18 07 "
"0C "
"00 18 "
"07 "
```

因此，在此处还设计了算法对收到的字符串进行拆分，放到数组 reclist 中缓存。

```
//设置算法分字符接收
if(str1.length()==2)
{
    reclist.append(str1);

}
else if(str1.length()==4)
{
    reclist.append(str1.mid(0,2));
    reclist.append(str1.mid(2,2));

}
else if(str1.length()==6)
{
    reclist.append(str1.mid(0,2));
    reclist.append(str1.mid(2,2));
    reclist.append(str1.mid(4,2));
}
else if(str1.length()==8)
{
    reclist.append(str1.mid(0,2));
    reclist.append(str1.mid(2,2));
    reclist.append(str1.mid(4,2));
    reclist.append(str1.mid(6,2));
}
if(recvNum%4==0)
{
    QString humi_h = reclist.mid (0,2);
    QString humi_l = reclist.mid (2,2);
    QString temp_h = reclist.mid (4,2);
    QString temp_l = reclist.mid (6,2);
```

```
//每位转换成int
bool ok1;
bool ok2;
bool ok3;
bool ok4;
int dec1 = humi_h.toInt(&ok1,16);//前一个参数表示
int dec2 = humi_l.toInt(&ok2,16);
int dec3 = temp_h.toInt(&ok3,16);
int dec4 = temp_l.toInt(&ok4,16);
//int再变成10进制qstring
QString num1 = QString("%1").arg(dec1,0,10);
QString num2 = QString("%1").arg(dec2,0,10);
QString num3 = QString("%1").arg(dec3,0,10);
QString num4 = QString("%1").arg(dec4,0,10);
//字符串拼接
humi = num1+'.'+num2;
temp = num3+'.'+num4;
//显示
ui->lcdTemp->display(temp);
ui->lcdHumi->display(humi);
qDebug("温度:");
qDebug()<<temp;
qDebug("湿度: ");
qDebug()<<humi;
reclist.clear();
}
```

（11）定时测温：

功能：实现对温度数据的定时测定，可以通过上位机调节定时间隔。

实现过程：

构造函数中创建定时器，使用信号槽，与按键关联。

```
// 定时发送-定时器
timSend = new QTimer;
timSend->setInterval(1000);// 设置默认定时时长1000ms
//connect(timSend, &QTimer::timeout, this, [=](){on_btnSend_clicked();});//只有发送框写上03才能定时测温，此处要改写
connect(timSend,&QTimer::timeout,this,[=](){sampleTemp();});//配套app2函数，temp+uart要调一下中断
```

"定时发送"复选框，转到槽，添加相应代码。根据输入框的值设置定时器间隔，控制开关。当定时器值满时，调用 write 向串口中写入一个指令，通过 MCU 接收中断的判断，实现温度可编辑间隔的定时读取。

```
// 定时发送开关 选择复选框---定时采集温度，实时显示温度---2022/1/2
void MainWindow::on_chkTimSend_stateChanged(int arg1)
{
    QByteArray sendTempTime;
    QString sampleTemp ="03";
    // 获取复选框状态，未选为0，选中为2
    if(arg1 == 0){
        timSend->stop();//关闭定时器
        // 时间输入框恢复可选
        ui->txtSendMs->setEnabled(true);//定时采集温度可以使用
    }
    else
    {
        // 对输入的值做限幅，小于5000ms会弹出对话框提示
        if(ui->txtSendMs->text().toInt() >= 5000)
        {
            timSend->start(ui->txtSendMs->text().toInt());// 设置定时时长，重新计数
            ui->txtSendMs->setEnabled(false);// 让时间输入框不可选，避免误操作（输入功能不可用，控件背景为灰色）
            sendTempTime = QByteArray::fromHex(sampleTemp.toUtf8()).data();
            mySerialPort->write(sendTempTime);
        }
        else
        {
            ui->chkTimSend->setCheckState(Qt::Unchecked);
            QMessageBox::critical(this, "错误提示", "定时采集温度的最小间隔为5000ms\r\n请确保输入的值 >=10");
        }
    }
}
```

```
//定时测温子函数，定时器记完时调用，Line 59 connect连接
void MainWindow::sampleTemp()
{
    QByteArray sendTempTime;
    QString sampleTemp ="03";
    sendTempTime = QByteArray::fromHex(sampleTemp.toUtf8()).data();
    mySerialPort->write(sendTempTime);
}
```

注意：此处由于 DHT11 时序的限制，读取温度不能过于频繁，一般在 3s 以上比较稳定，否则会造成总线时序错乱。

（12）手动降温：

功能：实现上位机手动开关电机，模拟对智能鸡舍的手动降温。

实现过程：

设置槽函数，检测按键文本，如果按键文本是“手动降温”时按下按键，则反转文本为“停止降温”，同时通过 UART 发送 02 命令给 MCU；如果按键文本是“停止降温”时按下按键，则反转文本为“手动降温”，同时通过 UART 发送 01 命令给 MCU。

MCU 通过读取命令，在接收中断中控制电机不同的转速，实现上位机对电机的控制。

```
//手动降温函数--------2022/1/2
void MainWindow::on_HandReduceTemp_clicked()
{
    QByteArray sendLower;
    QByteArray sendHigher;
    QString lower ="01";
    QString higher = "02";

    if(ui->HandReduceTemp->text() == "手动降温")
    {
        ui->HandReduceTemp->setText("停止降温");
        sendHigher = QByteArray::fromHex(higher.toUtf8()).data();
        mySerialPort->write(sendHigher);
    }
    else//停止降温动作
    {
        ui->HandReduceTemp->setText("手动降温");
        sendLower = QByteArray::fromHex(lower.toUtf8()).data();
        mySerialPort->write(sendLower);
    }
}
```

（13）设定阈值：

功能：实现上位机设定温度报警的阈值，MCU 检测到温度超过阈值时，连接的电机转动降温，蜂鸣器报警，呼吸灯闪烁。

实现过程：

读取可编辑文本框 QLineEdit 的值，在检测到“设定阈值”按键按下时，上位机将这个值通过 UART 发送给 MCU。

```
void MainWindow::on_SetThreshold_clicked()
{
    QByteArray sendThreshold;
    QString cmd ="04";//控制字，第一个字符
    //获取int温度阈值+16进制转换
    int threshold_int;
    threshold_int=ui->TempThreshold->text().toInt();
    QString Tempthreshold = QString::number(threshold_int, 16);
    QString Tempthreshold1 = cmd+Tempthreshold;
    sendThreshold = QByteArray::fromHex(Tempthreshold1.toUtf8()).data();
    mySerialPort->write(sendThreshold);
    qDebug()<<sendThreshold;
}
```

（14）建立 WIFI 连接与 TCO 配置：

功能：实现 TCP 通信 IP 地址和端口号的配置

实现过程：

在项目配置文件(.pro)中添加“QT+= network”引入 Tcp 通信有关库，利用 QTcpSocket 类在构造函数中构建一个新的客户端对象，

```
//wifi通信客户端有关的
socket = new QTcpSocket();
```

配置按键的槽函数，按下后文本翻转为断开连接，获取 QLineEdit 的值得到 IP 地址和端口号，利用 Socket 库中的 connectToHost()函数建立连接。

此处还添加了，连接检测响应，如果连接时间超过 3s 判断为连接错误，弹窗警告。建立连接之前与 WIFI 通信有关的控件动作都设置为不可用。

```
void MainWindow::on_pushButton_Connect_clicked()
{
    if(ui->pushButton_Connect->text() == tr("连接"))
    {
        QString IP;
        int port;

        //获取IP地址
        IP = ui->lineEdit_IP->text();
        //获取端口号
        port = ui->lineEdit_Port->text().toInt();

        //取消已有的连接
        socket->abort();
        //连接服务器
        socket->connectToHost(IP, port);

        //等待连接成功发送“连接成功”，否则发送“连接失败”
        if(!socket->waitForConnected(3000))//等待3s
        {
            QMessageBox::critical(this, "Error!", "WIFI通讯连接失败！\n请检查IP地址或端口号");
            //qDebug() << "Connection failed!";
            //return;
        }
        else
        {
            qDebug() << "Connect successfully!";
            //发送按键使能
            ui->pushButton_Send->setEnabled(true);
            ui->pushButton_Light->setEnabled(true);
            ui->pushButton_Feed->setEnabled(true);
            //修改按键文字

              ui->pushButton_Connect->setText("断开连接");
         }
    }
    else
    {
         //断开连接
         socket->disconnectFromHost();
         //修改按键文字
         ui->pushButton_Connect->setText("连接");
         ui->pushButton_Send->setEnabled(false);
         ui->pushButton_Light->setEnabled(false);
         ui->pushButton_Feed->setEnabled(false);
    }
}
```

（15）WIFI 接收和发送缓冲区

功能：实现局域网内数据的收发的缓存显示。

实现过程：

在构造函数中使用信号槽与函数相关联。

```
//连接信号槽
QObject::connect(socket, &QTcpSocket::readyRead, this, &MainWindow::socket_Read_Data);
QObject::connect(socket, &QTcpSocket::disconnected, this, &MainWindow::socket_Disconnected);
```

槽函数动作与UART接收发送缓冲区类似，按下“通讯”按键时，调用socket_Read_Data()函数，将发送区的数据利用 socket 库的 write 函数发送出去。

```
void MainWindow::on_pushButton_Send_clicked()
{
    qDebug() << "Send: " << ui->textEdit_Send->toPlainText();
     //获取文本框内容并以ASCII码形式发送
    //socket->write(ui->textEdit_Send->toPlainText().toLatin1());
    //以十六进制发送
    socket->write(QByteArray::fromHex(ui->textEdit_Send->toPlainText().toUtf8().data()));
    socket->flush();
}
```

缓冲区内显示内容的编程与 UART 部分类似，在此只附代码，不做赘述。

```
void MainWindow::socket_Read_Data()
{
    QByteArray buffer;
    //读取wifi缓冲区数据
    buffer = socket->readAll();
    // 16进制显示，并转换为大写
    QString str11 = buffer.toHex().toUpper();//.data();
    // 添加空格
    QString str22;
    for(int i = 0; i<str11.length (); i+=2)
    {
        str22 += str11.mid (i,2);
        str22 += " ";
    }
    //在缓冲区显示
    ui->textEdit_Recv->insertPlainText(str22);//不能settext，否则会刷新接收区显示的内容，不能把所有的都显示出来
    ui->textEdit_Recv->moveCursor(QTextCursor::End);
}
void MainWindow::socket_Disconnected()
{
    //发送按键失能
    ui->pushButton_Send->setEnabled(false);
    ui->pushButton_Light->setEnabled(false);
    ui->pushButton_Feed->setEnabled(false);
    //修改按键文字
    ui->pushButton_Connect->setText("连接");
    qDebug() << "Disconnected!";
}
```

（16）WIFI 远程控制鸡舍内部动作

功能：实现上位机通过 WIFI 局域网无线控制鸡舍照明灯开启、定时投喂。

实现过程：

定义两个按键和槽函数，实现的思想与测温功能类似。

```
void MainWindow::on_pushButton_Light_clicked()
{

    if(ui->pushButton_Light->text() == "照明")//按下照明按键
    {
        ui->pushButton_Light->setText("停止");
    }
    else
    {
        ui->pushButton_Light->setText("照明");
    }
    QByteArray SendLight;
    QString SendL ="05";
    SendLight = QByteArray::fromHex(SendL.toUtf8()).data();
    socket->write(SendLight);
    socket->flush();
}
```

按下“照明”按键时翻转文本为“停止”，发送“05”指令，在 MCU 接收中断中控制 IO 引脚输出高电平点亮照明灯；按下“停止”按键时反转文本为“照明”，发送“05”指令，在 MCU 接收中断中控制 IO 引脚翻转电平关闭照明灯。

```
void MainWindow::on_pushButton_Feed_clicked()
{
    QByteArray SendFeed;
    QByteArray SendStopFeed;
    QString SendF = "06";
    QString SendS = "07";
    if(ui->pushButton_Feed->text() == "投喂")//按下投喂按键
    {
        ui->pushButton_Feed->setText("停止");
        SendFeed = QByteArray::fromHex(SendF.toUtf8()).data();
        socket->write(SendFeed);
        socket->flush();
    }
    else//停止投喂
    {
        ui->pushButton_Feed->setText("投喂");
        SendStopFeed = QByteArray::fromHex(SendS.toUtf8()).data();
        socket->write(SendStopFeed);
        socket->flush();
    }

}
```

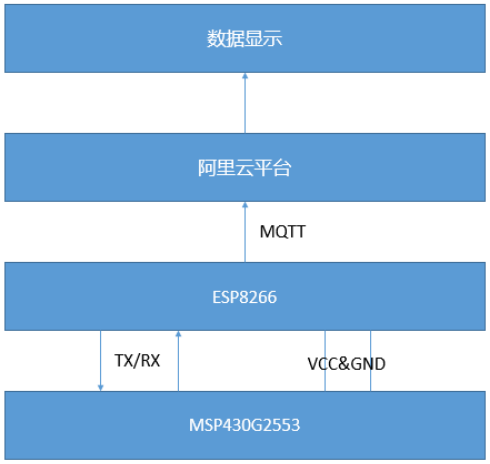
按下“投喂”按键时反转文本为“停止”，发送“06”指令，在 MCU 接收中断中控制打开定时器分中断 CCIE，开始定时，计数到 CCR0 时 P2.1 引脚输出低电平，继电器闭合，投喂电机开始工作；按下“停止”按键时反转文本为“投喂”，发送“07”指令，在 MCU 接收中断中，关闭中断 CCIE，同时 P2.1 引脚输出高电平，继电器断开，投喂电机停止工作。

（10）阿里云平台存储模块：

1. 平台简介

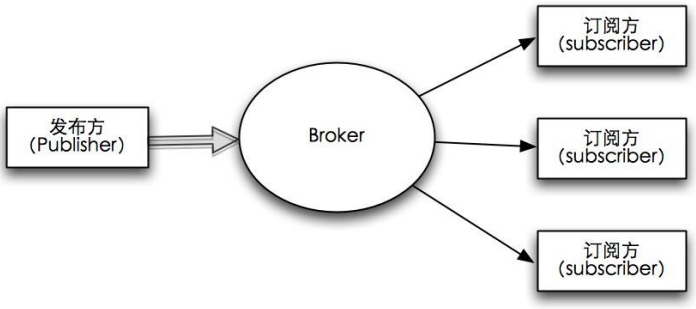
阿里云物联网平台提供了 2G/3G/4G、NB-IoT、LoRaWAN、WIFI 等不同的设备接入方案，提供 MQTT、CoAP、HTTP/S 等多种协议的设备 SDK，既满足长连接实时性需求，也满足短连接低功耗需求，同时 ESP8266 有支持相关功能的库。此外阿里云物联网平台提供一机一密设备认证机制，安全级别高。其概念图如下所示：

2. 本系统的物联网架构



3. 通信协议

MQTT 是一个客户端服务端架构的发布/订阅模式的消息传输协议。它的设计思想是轻巧、开放、简单、规范，易于实现。这些特点使得它对很多场景来说都是很好的选择，特别是对于受限的环境如机器与机器的通信（M2M）以及物联网环境（IoT）。MQTT 控制报文由固定报头、可变报头和有效载荷三个部分组成，发布方发布主题到云端，订阅方从云端获取主题包含的信息。



4. 实现过程

（1）创建产品

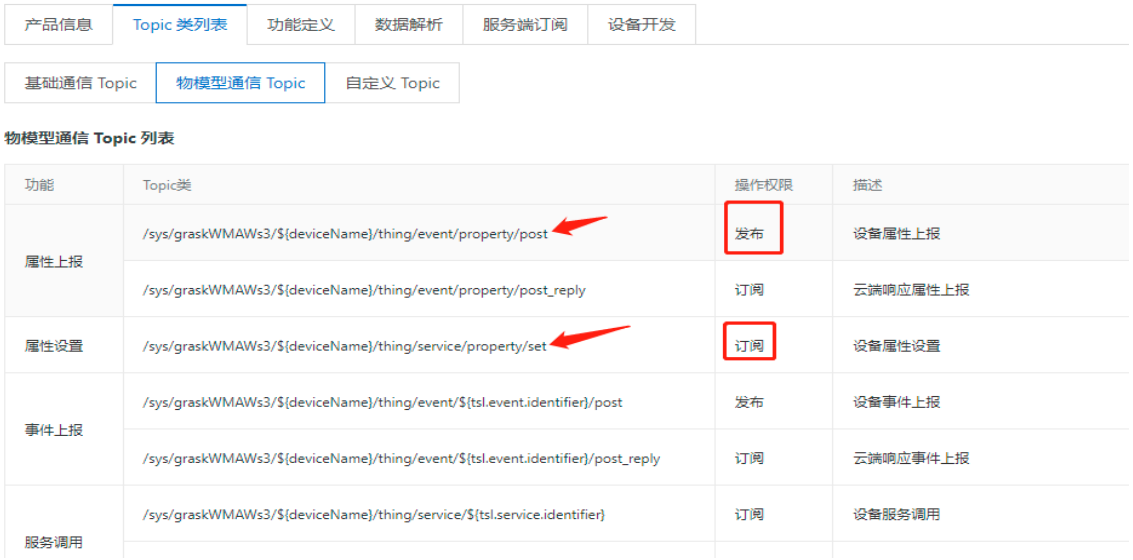
在物联网平台\设备管理界面新建一个产品，选择直连设备节点。

（2）添加功能定义

在物联网平台\设备管理\产品详情\功能定义界面点击编辑草稿添加功能。根据系统需求添加相应的属性例如：温度、湿度等，自定义相关的标识符、取值范围等。确认无误后发布上线，并查看相关的物理模型定义（JSON 格式）。

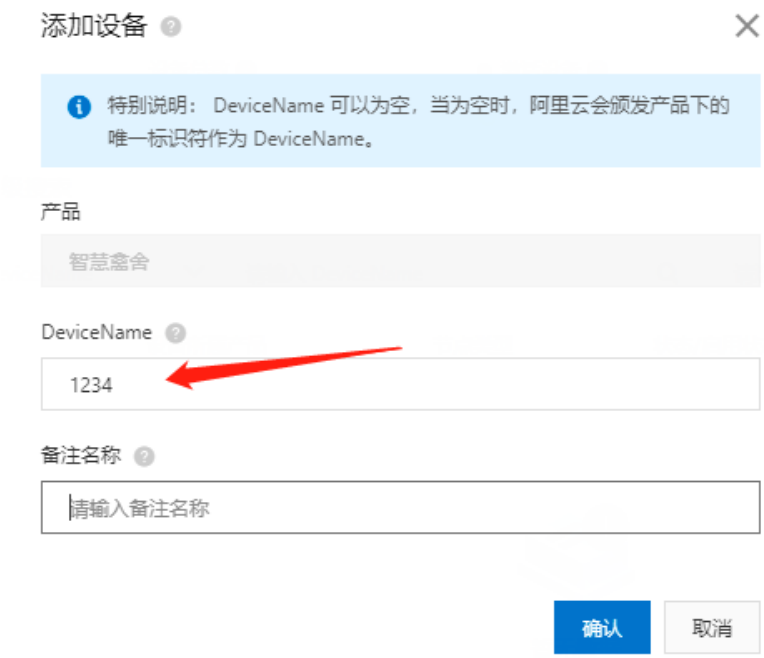
（3）选择相关的主题

在物联网平台\设备管理\产品详情\Topic 类列表，找到系统需要的有关发布和订阅的 Topic，并记录下来备用。



（4）新建设备

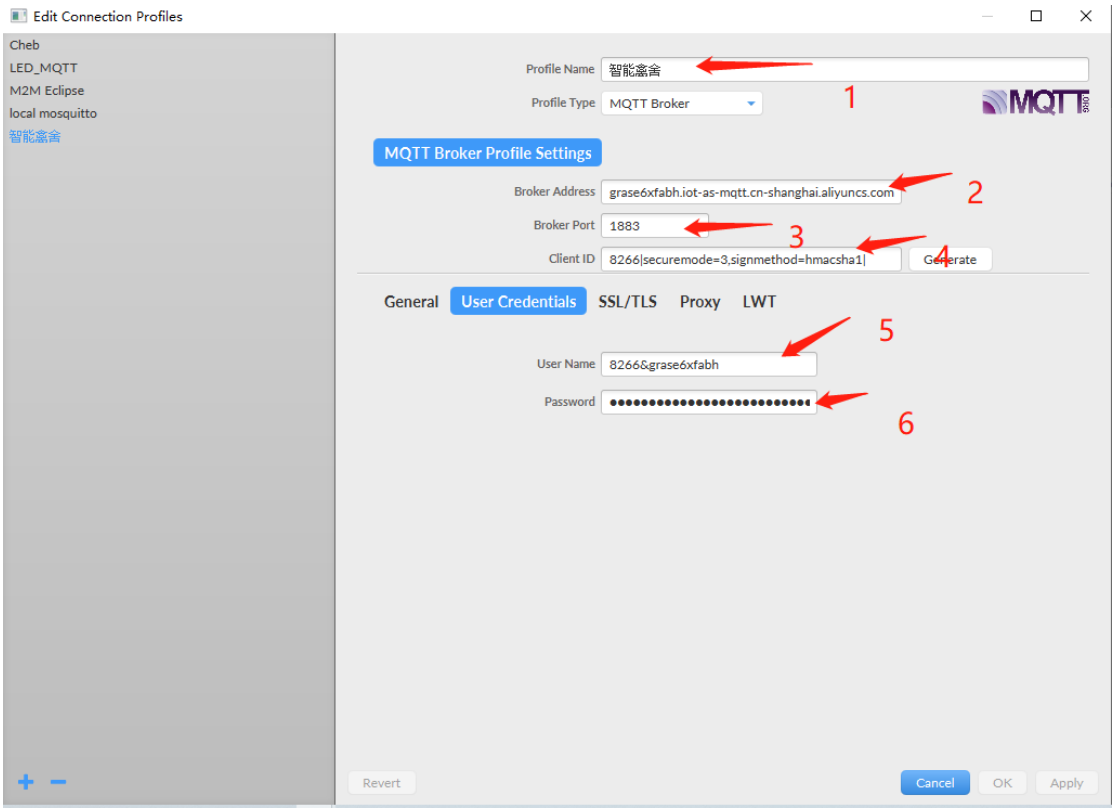
在物联网平台\设备管理\产品详情\设备开发，点击添加/注册设备进行设备的添加。自定义设备的名称。然后一键复制设备证书保存至文档备用。设备证书包括了 ProductKey, DeviceName, DeviceSecret 三个设备信息，将会在 MQTT 通信时用到。至此，云端部署已经完成。



（5）MQTT.fx 虚拟客户端调试

新建一个客户端，自定义文件名称。

严格按照规定，设置 Broker Address，Broker Port，Client ID，User Name，Passward，其他默认。



Broker Address：{ProductKey}.iot-as-mqtt.cn-shanghai.aliyuncs.com

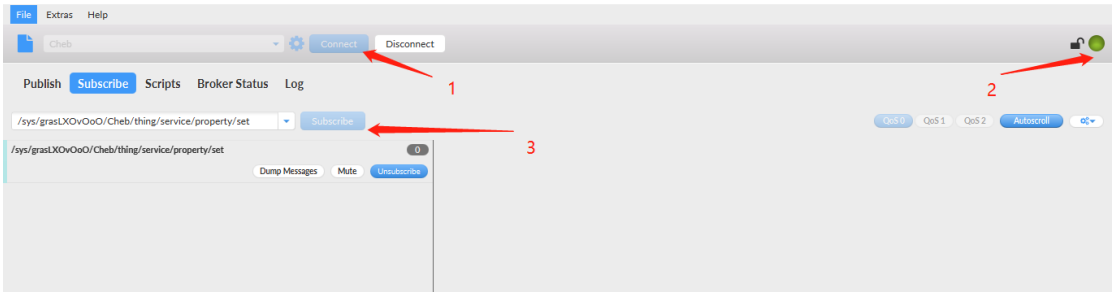
Broker Port：默认 1883 即可。

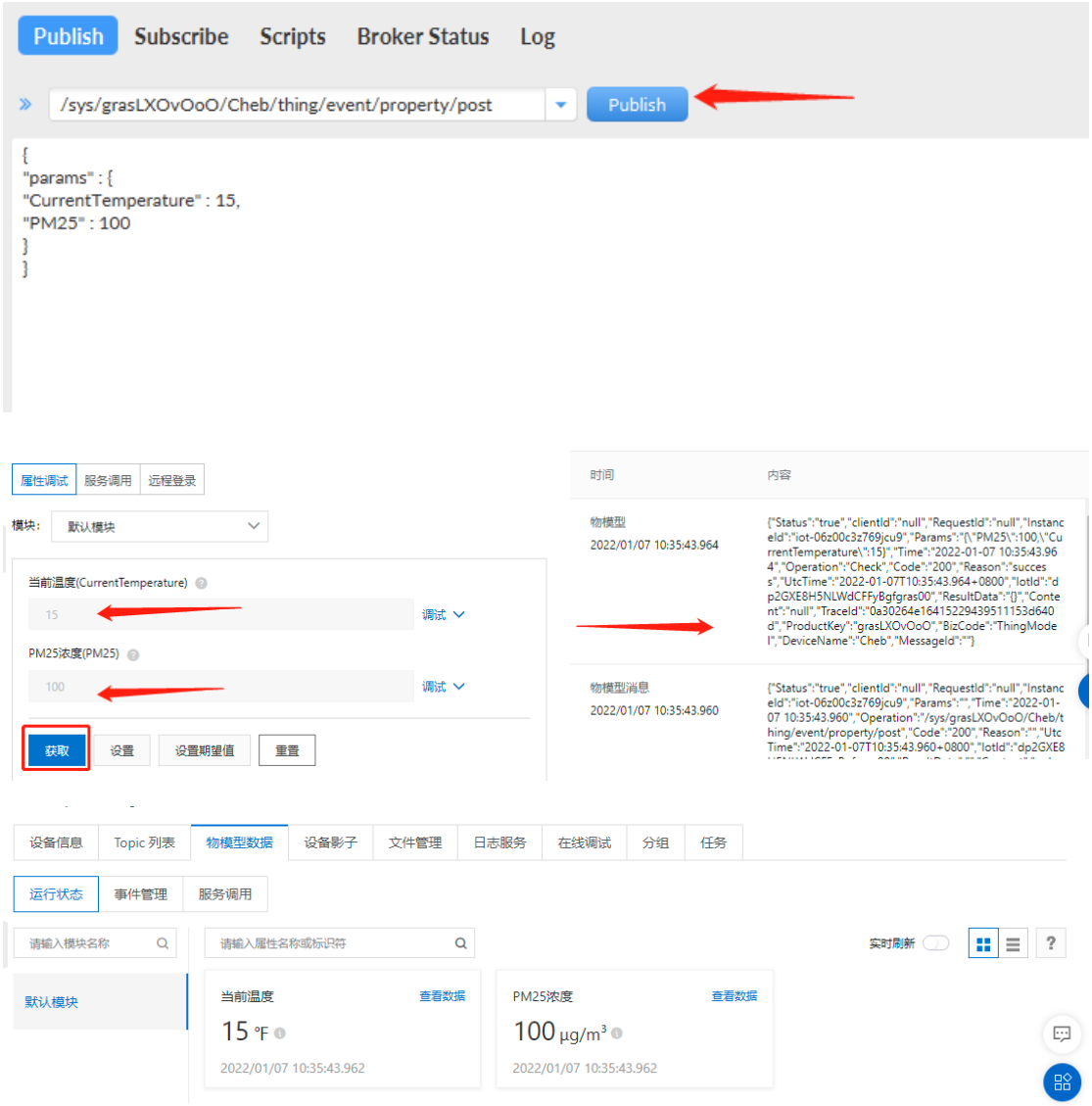
Client ID：{clientId}|securement=3,signmethod=hmacsha1|，clientId 为设备信息，可以为任何值，此处选择 DeviceName。

User Name：{DeviceName} & { ProductKey }。

Password：密码由参数值拼接而成。

之后就可以 connect 连接至阿里云平台，然后在 MQTT.fx 客户端订阅相关的主题，并发布一个包含属性的 JSON 格式主题，然后在物联网平台\监控运维\在线调试就可以接收到由客户端发来的信息，就可以获取到客户端发来的属性信息。最后就可以在物联网平台 \设备管理\设备详情\物模型数据看到数据的实时显示。





（6）ESP8266 开发

类似于 MQTT.fx 客户端的操作，在 ESP8266 中烧写代码，模拟虚拟客户端与阿里云进行通信。在代码中声明之前在 MQTT.fx 新建虚拟客户端时生成的一系列有关 MQTT 协议的信息，然后定义发送相关的操作。

1. 向 MSP430G2553 发送起始命令，开始接收从 MSP430G2553 传输过来的温湿度信息。
2. 解析得到的数据，计算得到相对湿度和环境温度。
3. 构建一个 JSON 格式的 payload 字符串模拟发送的主题信息。
4. 通过响应的主题发布信息至云平台 ，云平台得到数据并显示。

```
const char* WIFI_SSID     = "[redacted]"; // WiFi账号密码
const char* WIFI_PASSWORD = "[redacted]"; // WiFi密码

/******************产品/设备配置(每个人需要根据自己的产品设备信息去动态更换)*********************/
#define PRODUCT_KEY "gra[redacted]h"  //产品key  从产品详情获取
#define DEVICE_NAME "8266"  //设备deviceName  从设备详情获取
// 服务端相关
#define MQTT_SERVER   PRODUCT_KEY ".iot-as-mqtt.cn-shanghai.aliyuncs.com" //阿里云MQTT服务地址
#define MQTT_PORT     1883                                                //MQTT服务端口
// 校验三元组
#define MQTT_CLIENT_ID  DEVICE_NAME"|securemode=2,signmethod=hmacsha1|" //mqtt clientid
#define MQTT_USERNAME   DEVICE_NAME"&"  PRODUCT_KEY                    //mqtt username
#define MQTT_PASSWORD   [redacted]                                      //mqtt password 通过生成工具获得
// 相关主题 自定义一个主题
#define TOPIC1 "/"PRODUCT_KEY"/"DEVICE_NAME"/user/setledstatus"
#define TOPIC2 "/sys/"PRODUCT_KEY"/"DEVICE_NAME"/thing/event/property/post"//发布的主题
#define TOPIC3 "/sys/"PRODUCT_KEY"/"DEVICE_NAME"/thing/service/property/set"//订阅的主题
/*************************************************************************************************/

float RelativeHumidity=17.0;
float EnvironmentTemperature=28.3;
Serial.write(0x01);
RelativeHumidity=Serial.read()+Serial.read()/100.0;
EnvironmentTemperature=Serial.read()+Serial.read()/100.0;

// 构建一个 JSON 格式的payload的字符串   {"params" : {"RelativeHumidity" : 15,"EnvironmentTemperature" : 24}}
String payload = "{\"params\":";payload += "{";
payload += "\"RelativeHumidity\":"; payload += RelativeHumidity; payload += ",";
payload += "\"EnvironmentTemperature\":"; payload += EnvironmentTemperature;
payload += "}";payload += "}";
char attributes[100];
payload.toCharArray( attributes, 100 );

mqttclient.publish( TOPIC2, attributes );  //发布TOPIC2的主题
Serial.print("[publish]-->>");
Serial.println( attributes );
delay(5000);
```

4.参考文档

阿里云物联网官方文档（https://help.aliyun.com/product/30520.html）
CSDN 单片机菜鸟哥博客（https://blog.csdn.net/dpjcn1990/article/details/92831918）

（二）系统具体实现设计

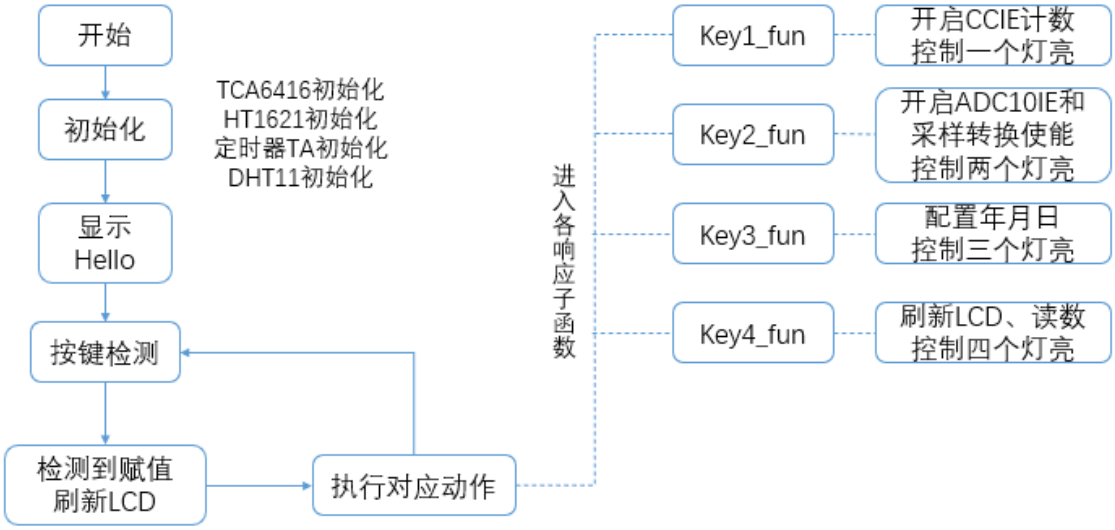
（I）LCD 菜单切换显示 V1：

1. 总体功能：

利用艾研拓展板 G2 LaunchPad 上的 4 个按键 Key1-Key4 按键控制菜单，切换 LCD 屏上显示不同的内容，包括初始化时的 Hello、Menu1 的工作时间、Menu2 的芯片内部温度、Menu3 的年月日和 Menu4 的环境温湿度。

2. 软件设计流程：

总体思路，将每个按键的动作分别调试最后整合到一起，具体伪代码如下图所示：



3. 具体实现：

详细代码见 000menu_V3.0，以下介绍各功能的实现方法。

（1）Menu1 计时：

由于没有外接晶振,因此采用内部时钟定时,配置时钟为低频振荡器LFXT1S的12Khz，经过测试这种条件下显示 1s 时间的最准确，实际大概为 0.92s，利用定时器 TA0 的上数模式进行定时，当 CCR0=12000 时进入中断服务子程序 Timer_A()，计数器+1，完成 1s 定时。

（2）Menu2 内部芯片温度检测：

由于MSP430G2553的ADC10通道有检测内部芯片温度的功能,因此利用ADC10的Inch10来实现。配置基准为 VR+= VREF+和 VR-= VSS,采样时间为 64 个时钟周期,基准电压 1.5v，输入通道 10，连续传输。

当按下按键时，打开 ADC10 中断、采样和保持使能，进入中断，在中断服务子程序 ADC10_IRS()中，执行 ADC10MEM 到实际温度十位、个位、十分位的转换，完成后关闭使能转换。

（3）Menu3 年月日：

由于 MCU 不联网无法获取实时日期，因此这里仅作模拟展示，给年月日三变量赋值，显示变量值的形式进行展示。具体实现只需要更改三变量的值，配置每一位数字的显示位置即可，利用 HT1621_BufferWrite()函数进行显示。

（4）Menu4 环境温湿度显示：

由于 DHT11 的时序有严格的限制，因此在功能函数中配置 DCO 为 1MHz，每次调用，利用子功能“环境温湿度测量”模块的函数从 DHT11 读取温湿度整数、小数位的值存入 4 个变量，每次显示 4 个变量。此处将温度放在数码管 7-10 位置，将湿度放在 3-5 位置显示。

（5）显示各菜单内容：

配置 LCD 各显示位置的显示数组，存放不同数值的 16 进制段码值，利用 HT1621_BufferWrite()函数在固定位置显示不同变量的值，每次切换菜单时可以采用两种方式切屏，一种方式是直接 HT1621_init+HT1621_clear 或者 HT1621_Reflash()，清空显存，另一种是利用 HT1621_BufferWrite(xxxx,0x00)函数将用到的显示位置清空，经过尝试后者效率比较高，在菜单显示 V2 中用到，此处用的是第一种方式。

4. 调试中出现的问题及解决方法：

根据多天的实验调试及分析，主要问题出现在**菜单切换时死机和按键检测跑飞**。

（1）菜单切换时死机：

我们发现艾研提供的驱动函数中的 Write_Frame_to_Tca6416()在对 HT1621 进行初始化、清空动作时，会对总线时序产生干扰，因此有的时候在 while 循环中同时调用 LCD_Displayxx()和 HT1621_BufferWrite()函数时可能出现菜单切换死机的现象。

解决方法是在每次按键检测后添加一个清屏 HT1621_init+HT1621_clear 或者 HT1621_Reflash()。

```
while (1)
{
  input1=scan_key();                //扫描扩展板上的4按键，每个按键代表一个地址
  if((cmp&input1)!=0x0f)            //cmp=0x0f
  {
    input=input1;                  //默认按键1状态
    //HT1621_Reflash(LCD_Buffer);   //按键按下时，刷新一次LCD
    __delay_cycles(20000);
    HT1621_init();      // 初始化lcd_128
    HT1621_clear();     //
    if(input==0xfe)
    {
```

（2）按键检测跑飞：

此处是由于艾研拓展板拓展程序代码逻辑的不完善，最初检测按键时赋值变量 input 是 8 位同时赋值检测，这样电平不稳定时，可能会对高四位产生干扰，导致按键检测失败，但实际只需要检测低四位某一位为 0 即可。此外，这个问题，由于源程序中只用了 else if 检测按键，但 else 函数中为空，因此还可以在 else if 的 else 中时 flag 标志位赋值等于 1，这样跑飞时默认回到首页。

解决方法是按键消抖或者修改检测逻辑，利用与操作检测第四位中的某一位为 0。

（3）其他问题及解决方法：

①由于传感器、ADC 读到的数都是整数，要想得到小数，需要除法运算时设置 float 类型，并且保证运算数有一个是浮点类型，如 1024.0；

②ADC10 温度值跳变太快可能当前显示的不是实际值，可以添加延时或者连续读取 10 个数取平均的方法显示。

③显示位置 1-6 和 7-10 的数字段码值不一样，需要重新编写数值数组；

④如果想让某个位置显示小数点，只需要将数值数组内的段码值+0x08/0x10；

```
const char num[10]={0xeb,0x60,0xc7,0xe5,0x6c,0xad,0xaf,0xe0,0xef,0xed};//代表0-9是个数值
const char num_year[10]={0xd7,0x06,0xe3,0xa7,0x36,0xb5,0xf5,0x07,0xf7,0xb7};//代表0-9是个数值
const char num_year1[10]={0xdf,0x0e,0xeb,0xaf,0x3e,0xbd,0xfd,0x0f,0xff,0xbf};//代表0-9是个数值
const char num_minh[10]={0xfb,0x70,0xd7,0xf5,0x7c,0xbd,0xbf,0xf0,0xff,0xfd};//恒显示小数点
const char num_minl[10]={0xfb,0x70,0xd7,0xf5,0x7c,0xbd,0xbf,0xf0,0xff,0xfd};//代表0-9是个数值
```

⑤DHT11 手册中得知，其读取数据的间隔最好在 2s 以上，因此按键不能太频繁，否则会出现时序错乱。

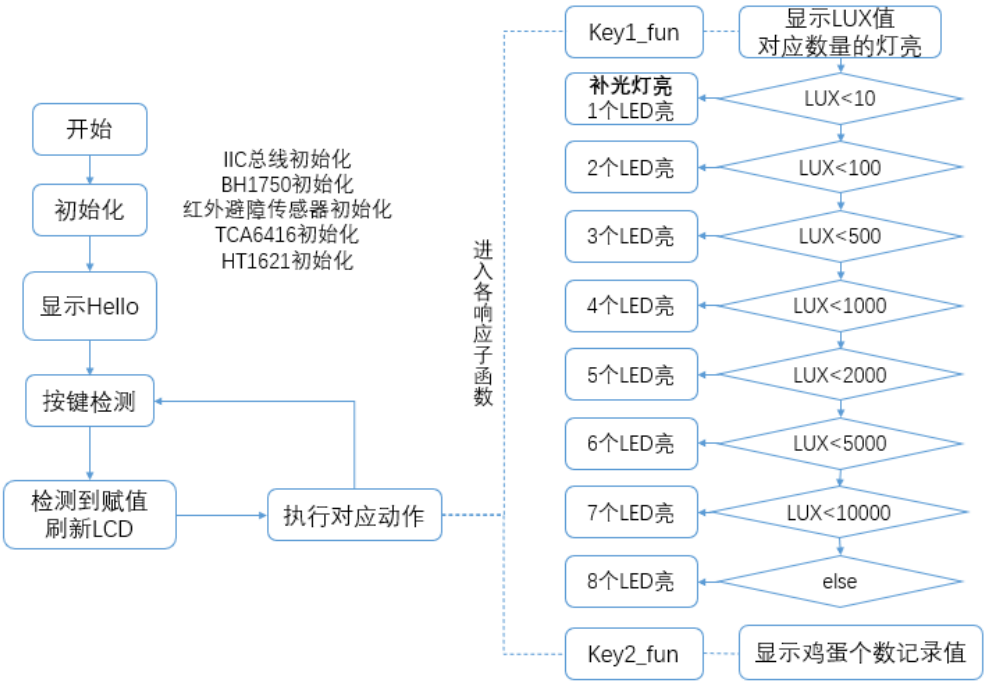
（II）LCD 菜单显示 V2：

1. 总体功能：

跟菜单显示 V1 类似，利用艾研拓展板 G2 LaunchPad 上的按键 Key1-Key2 按键控制菜单，切换 LCD 屏上显示不同的内容，包括初始化时的 Hello、Menu1 的光强值、Menu2 的采集到的鸡蛋数值。

2. 软件设计流程：

总体思路类似于 V1 部分，将每个按键的动作分别调试最后整合到一起，具体伪代码如下图所示：



3. 具体实现：

本部分的功能其实可以整合到 V1 部分，利用按键按下次数的数值来安排多于 4 次的菜单切换，但由于 MSP430G2553 引脚资源的限制，以及循环个数对运行延迟的影响，单独分离出来。详细代码见 000menu=cnt+lx+led，以下介绍各功能的实现方法。

（1）GY30 光强检测：

根据分功能“环境光强检测模块”的分析，编程实现适用于 GY30 的 IIC 总线动作，具体内容见 init.h 文件，由于 G2LaunchPad 占用了 P1.6 和 P1.7 两个 IIC 总线引脚，因此此处配置 SCL、SDA 引脚分别为 P2.3 和 P2.4，按照伪代码流程图的逻辑编写 key1 按键的动作。

（2）鸡蛋计数显示：

由于红外避障传感器 E18-D80NK 利用的是高低电平上升沿/下降沿变化实现对传感器遮挡与否的模拟，因此程序中利用 P1.5 引脚输入电平检测，配置上拉电阻，下降沿触发中断的方式。在遮挡到无遮挡时，引脚下降沿被检测到，进入中断服务子程序，统计值 number+1，最后在按键 key2 的动作函数中进行显示。

（3）各数值的显示：

类似于 V1 部分的思想，但这里采用的切屏方式是 HT6416_BufferWrite()函数清空使用的显示位，这样的效率比较高，而且很少出现切屏时的卡机现象。

4.调试中出现的问题及解决方法：

本部分主要是**各传感器的使用问题**。

（1）光强检测传感器 GY30：

最开始以为跟拓展板的 IIC 总线动作可以用一套代码，但实际 GY30 的 IIC 通信协议与之不同，不能互用，查阅手册根据给出的方案进行修改。

（2）红外避障传感器 E18-D80NK：

该传感器的 VCC 必须接 5v，传感器必须与 MSP430G2553 进行共地，否则会检测不到电平的变化。

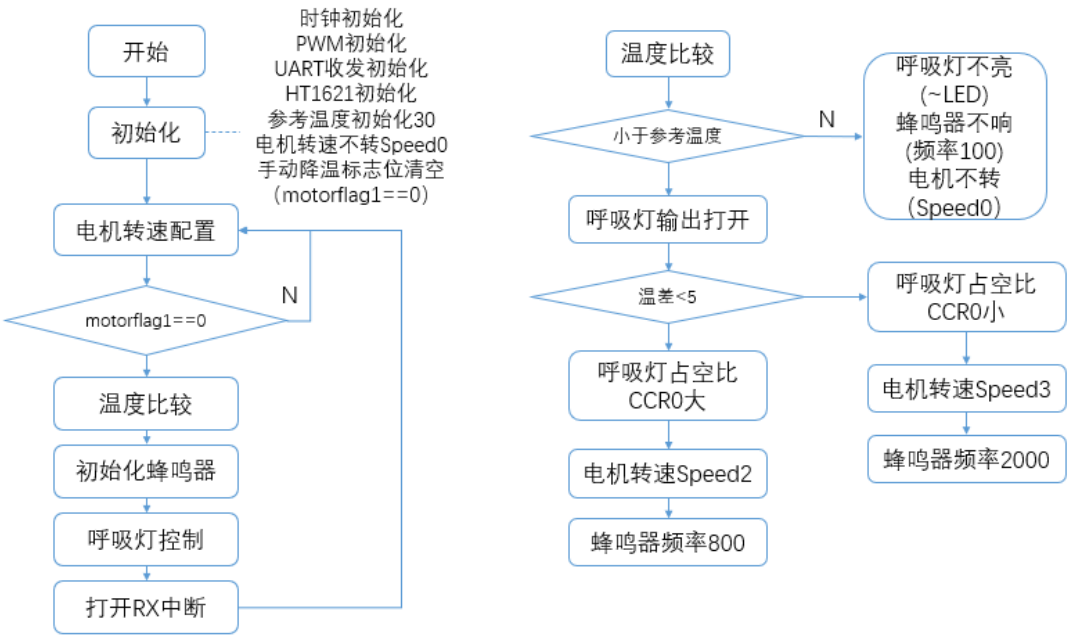
（III）环境变量检测报警与 UART 上位机通讯

1.总体功能：

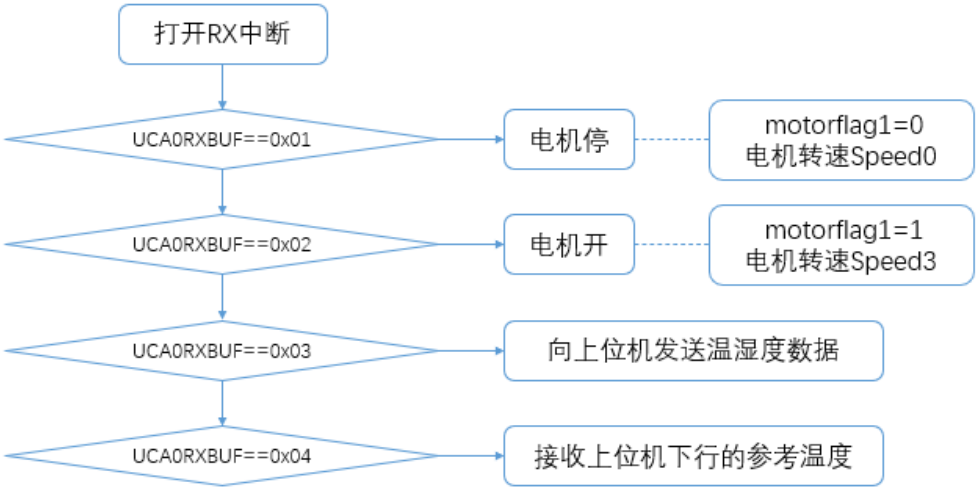
通过上位机或者程序设置参考温度，实现对环境温度检测的同时，进行检测报警及响应。报警及响应部分包括蜂鸣器、呼吸灯、电机的动作配置。利用 RX 中断与上位机实现通信，达到定时采温、手动控制电机、配置参考温度灯功能。

2.软件设计流程：

具体伪代码如下图所示：



RX 中断服务子程序中主要是对从上位机中接收到的指令进行分析，配置电机转速，及温湿度互传。



3. 具体实现：

编程逻辑主要是根据手动降温标志位 motorflag，决定是否进行手动降温。当标志位关闭时，在主程序 while 循环中执行温度比较子函数，根据温差配置报警相应的不同动作。具体各部分的分析见各子模块分析。详细代码见 000temp+motor+uart_v2.0，以下主要介绍上位机与 MCU 的通讯逻辑。

（1）手动降温：

在上位机软件中按下“手动降温”按键，其槽函数会向 uart 串口中写入 0x02，MCU 进入接收中断，手动降温标志位 motorflag 置位，电机转速 speed3，电机开启。此时按键翻转为“停止降温”，再按下时，槽函数会向 uart 串口中写入 0x01，MCU 接收中断中 motorflag 复位，电机转速 speed0，电机停止，按键反转为“手动降温”。

（2）定时测温：

上位机的 Qtime 库中包含一个定时器，只需要在上位机中设定定时器的定时间隔，当定时器达到定时间隔时，槽函数会向 uart 串口中写入 0x03，MCU 进入接收中断子函数，向上位机发送一次温湿度的数据，上位机接收后进行处理（详见上位机部分），显示在上位机的 LCD 段码管上。

（3）上位机设定参考温度：

思想与之前所述类似，当按下“设定阈值”按键时，其槽函数会读取阈值温度窗口内的值，通过 uart 串口发送“0x04”+“0x 阈值”，MCU 进入中断服务子函数，接收阈值温度，赋值给参考温度变量。

4. 调试中出现的问题及解决方法：

qt 上位机不能发 0x00，会默认串口没有数据；

收发温度数据时，缓存温度数值的数组要初始化，否则发送会出错；

定时测温的定时间隔不能太短，要在 2s 以上。

（IV）通过 TCP/WIFI 控制鸡舍照明、通风和投喂

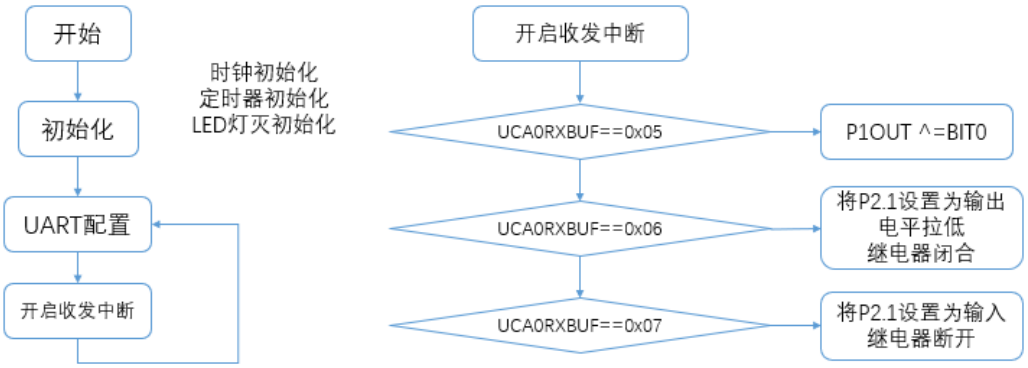
1. 主要功能：

实现利用 ESP8266 与 MSP430G2553 建立局域网实现 WIFI 通信，无线控制鸡舍照明灯开关、投喂和通风电机的继电器开断。其中投喂有两种方案，可以实现定时投喂和手动控制投喂。

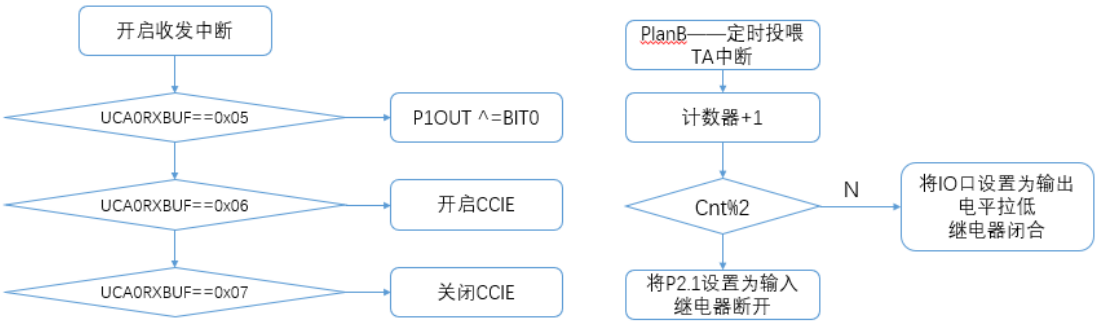
2. 代码编程逻辑：

两种投喂方案主程序逻辑相同，主要是接收中断服务子程序的判断不同，具体伪代码如下图所示：

PlanA：



PlanB：



3. 具体实现：

利用 ESP8266 和局域网，具体方案详见“TCP/WIFI 通信部分”，在上位机的 TCP/WIFI 通讯窗口，先配置 IP 地址和端口号与局域网建立连接。利用 socket 库的读写函数，实现 wifi 通讯。具体代码详见 OOOUART_wifi_led+swich(PlanA)/(PlanB)。

（1）远程照明控制：

点击“照明”按钮或发送“0x05”，上位机通过局域网发送 0x05 指令给 ESP8266，其与 MSP430G2553 进行 UART 通信，当 MCU 接收到命令时，进入接收中断，LED 灯对应引脚翻转，实现开灯，按键翻转为“停止”；当电机“停止”按钮时，再次发送 0x05 指令，重复上述过程，实现关灯。

（2）远程投喂：

如代码逻辑所示，点击“投喂”按钮，上位机通过局域网发送 0x06，MCU 接收后进入中断服务子程序，拉低引脚电平，继电器闭合，电机转动投喂饲料，按键翻转为“停止”。点击“停止”按钮，上位机通过局域网发送 0x07，MCU 接收后进入中断服务子程序，转换引脚为输入，继电器断开，电机停止投喂，按键翻转为“投喂”。

（3）远程通风：

与上述过程类似，不详细介绍。

4. 调试中出现的问题及解决方法：

建立局域网时，每次的 ip 地址会变化，因此建立连接时要修改，如果不修改需要在上位机中添加弹窗报警，否则会程序死机。

继电器必须接 5v 电压才能工作。

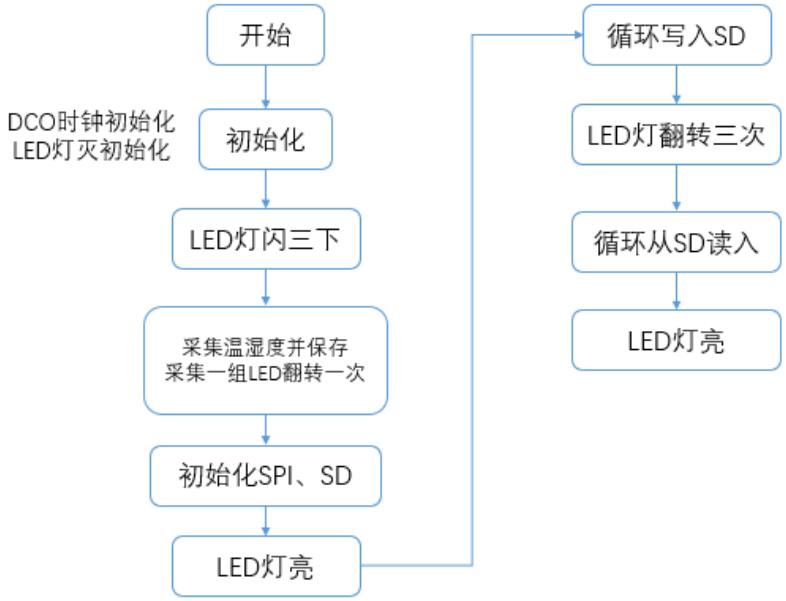
（V）数据储存到 SD 卡

1. 主要功能：

实现 DHT11 读取到的温湿度数据存放到 SD 卡。

2. 代码设计逻辑：

具体伪代码如下所示：



3. 具体实现：

具体代码详见 000SD+temp，实现的逻辑见分功能“环境温度存储模块”。

4. 调试中出现的问题及解决方法：

SD 卡要小于 2GB 容量，否则会报错；

缓存温湿度值的数组不能太大（手册上是 512 字节以内，实际因为 MSP430G2553 RAM 的限制会更小），否则会因内存不够报错。

四、对健康、环境、成本、可持续发展的考虑

由于本项目设计采用的开发板、传感器都是无毒无害的金属、半导体制作而成，并且不会引入过多的高频噪声、红外波等干扰，故不会对人禽健康和造成影响，也不会对禽类的生产、作息造成任何损害。使用到的传感器的稳定性高，成本低，消耗的电能十分有限，因此也不会对环境造成危害，受损、老化的传感器可以回收处理，更新替换，具有可持续发展的优势。

五、设计开展过程中存在的风险

项目进展过程都在实验室和教室进行，采用了先在面包板上调试，再接线焊接的方式。未使用高压设备，仅调试、焊接时用到的烙铁需要注意安全问题，因此实验风险较低。

六、应用前景与未来改进方向

近年来，随着中国城市化发展、新农村建设的深入开展，休闲农业产业发展迅猛，越来越多的市民青睐家禽的家庭养殖，这成为了都市型现代农业的重要组成部分。现阶段，行业内很少有专为家庭和小型养殖场设计的智能禽舍装置，我们将物联网思想融入智能鸡舍，利用端到云、端到端的互联，实现了数据的存储与不同平台上的综合应用，综上所述，我们的初代智能鸡舍解决了传统散养蛋鸡对环境的污染以及环境因素自动监测和调控问题，达到了生态养殖、互动体验的目的，相信其对未来的农事体验将是一个有力的推进。

未来我们希望完善环境因素检测的多样性（如加入环境气体检测等）以及系统功能的复杂性（如传感器组网实现上位机数据的二维、多维展示），并期望加入手机端实现更为完善的物联网控制。

对于阿里云平台部分，目前只实现了从设备到云端的上行数据传输，接下来可以进行从云端到设备的下行数据传输以及设备到云端到设备的控制等功能。

七、收获与体验

通过本次课程设计及本学期课程的学习，我们充分的理解了数模电课本的内容、微机原理的实际应用、以及传感器与检测系统的设计，并且还真正独立的解决了开发过程中遇到的各种问题，可以说是收获满满。

关于课程，印象最深刻的是前几节课入门的介绍，从触发器到寄存器，从晶体管到门电路再到功能电路，真正的从底层领会到了芯片电路的设计思想，并在实验中把所学知识与实际应用相联系。

关于项目，我们应该是最早进行开发设计的，一共在实验室泡了将近两周。从最初调试 lcd 时候的屡屡受挫，到跟着老师一步一步调试学会耐心分析，再到小组成员间分工合作共同推进项目目标的一点点实现，大家都是为了团队和项目完成而努力，并没有为了个人得失计较，真的是我不断努力的动力。这是我们三个人合作的第二个项目，相比于第一次的焦躁，这一次大家有所进步，一如既往的相互鼓励支持，真的是非常 nice 的体验。

当时开发设计的时候觉得很困难，但是最后总结的时候却又感觉设计的很简单，可能这就是设计开发的魅力吧。所谓行百里者半九十，在本次设计中体现的淋漓尽致，虽然开发完成了，但是还有总结汇报需要做，因此最后几天越是想玩，越是逼着自己坚持到最后，好在一切都顺利完成了，希望大家都取得好成绩，假期休息一下，下学期继续努力！

八、小组团队分工及时间安排

小组成员始终为了项目最初设定任务的完成而共同努力，在过程当中相互协助，实现功能后互相学习交流，因此团队分工仅概述主要工作内容：

孟令昶：功能模块开发设计

金政祥：功能模块开发设计

詹　金：功能模块调试，汇报总结材料整理

项目实践周期从 2021 年 12 月 23 号开始一直到 2022 年 1 月 7 日，历时 16 天，具体时间安排如下：

2021 年 12 月 23 日——12 月 25 日：完成初代上位机界面设计、LCD 显示部分框架调试；

2021 年 12 月 26 日——12 月 28 日：完成温湿度检测模块、报警及响应功能模块调试；

2021 年 12 月 29 日——12 月 31 日：完善上位机 UART 通信部分及子功能模块结合设计；

2022 年 1 月 1 日—— 1 月 3 日：光照、禽蛋计数、TCP/WIFI 通信部分的开发设计；

2022 年 1 月 4 日—— 1 月 5 日：功能系统集成整合，上位机 WIFI 通信部分的完善；

2022 年 1 月 5 日—— 1 月 7 日：总结汇报材料的准备、ppt 制作、报告撰写；