- **实验目的**

1. 思考嵌入式系统软件特点和合理结构，尝试设计较复杂逻辑关系的程序。

- **实验条件**

1. Keil 、STC-B 学习板
2. Stp-ISP 软件

- **实验要求**

1. 该综合设计针对“STC-B 学习板”或/和“STM32 开发板”现有硬件资源自选设计主题，每位学生通过该自选主题设计展现各自课程学习成效和水平。
2. 所选主题以展示对嵌入式系统相关应用技术理解、掌握、综合运用为主，选题背景、选题意义、选题创新性为辅；通过一对一测试（网上方式）、问辨（网上方式）和实验报告评阅（网下方式）、代码评阅（网下方式），从选题创意、复杂度、工作量、正确性、合理性、实验报告质量、代码质量、总体完成质量等方面考核和评判综合设计成绩。

- **实验内容**

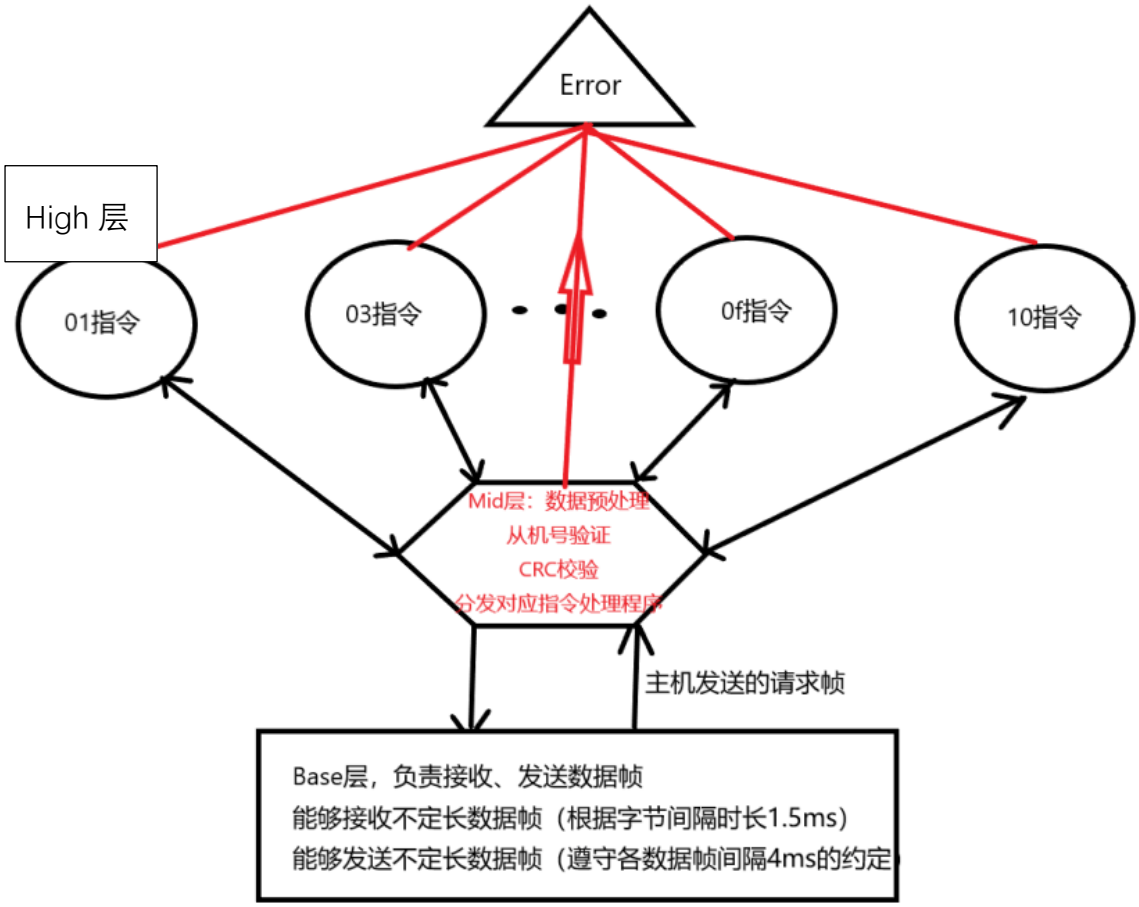
1. **系统整体：**

综合实验完成了以下功能：

ModBus 从机	8 个寄存器、4 个线圈，支持 8 个功能码
实时时钟	可实时显示时、分、秒
非易失存储器	可读、可写
串口	接收、发送不定长数据

ADC	光照和温度
按键及消抖	3 个按键
定时器	定时器 0、1、2
指示灯	8 个指示灯
数码管	8 个数码管

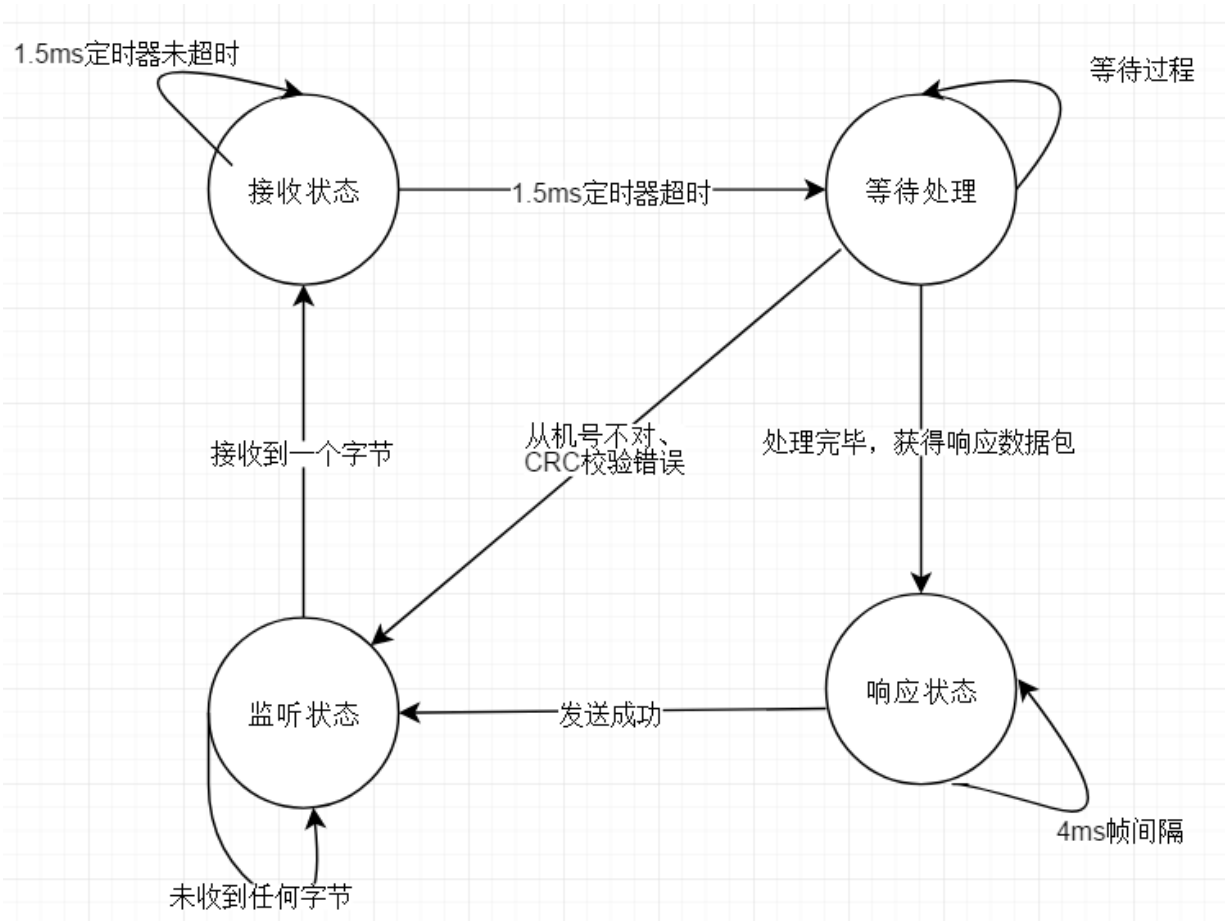
①ModBus 模块整体结构：



ModBus 分为三层：

Base 层：接收串口来的请求报文并传给 Mid 层，发送从 Mid 层穿过来的响应报文。

状态图如下：



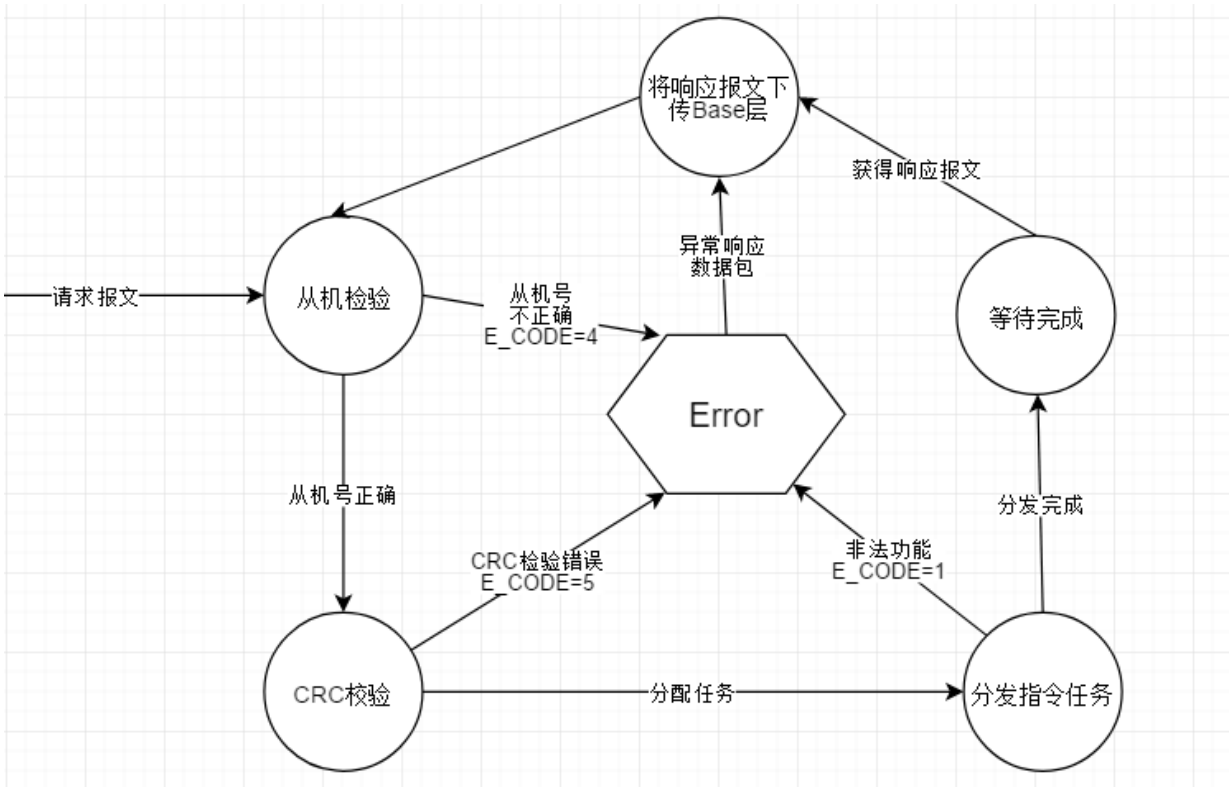
未接收到任何字节时，一直处于监听状态，当收到一个字节，开始进入接收状态，并重启定时器 2 (1.5ms)，如果定时器未超时时又接收一个字节，则接收的字节为同一帧，如果超时，则说明该帧数据包完全接收，等待上层处理，当处理完毕后，无论是正常响应还是异常响应都将数据包发送出去并重新进入监听状态。当然，如果从机号不对或者 CRC 校验错误，则不响应，舍弃该帧并进入监听状态。

接收不定长的请求报文，在串口每成功接收一个字节，我就会将重启定时器 2（1.5ms）。如果在定时器 2 未完成一个中断时又收到一个数据，则重启定时器。如果定时器 2 发送定时中断，则本次请求报文完全接收，将请求报文传给 Mid 层。该层不进行任何检查。

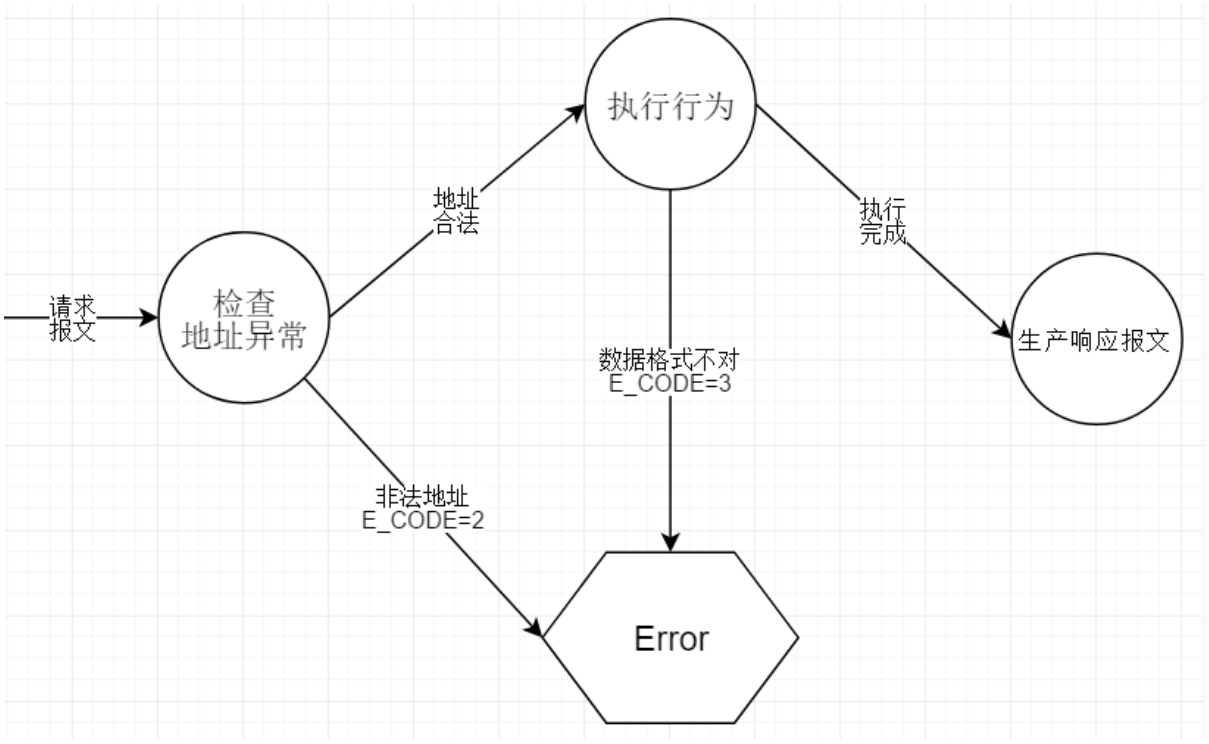
发送补不定长响应报文，响应报文虽不定长，但长度是可知的，当 Mid 层将响应报文下传后，串口将根据响应报文长度将字节依次发送出去。另外，请求报文的发送并非是即时的，当 Base 层会以 100Hz 的频率检查是否有响应报文发送，如果有，发送请求报文。当发送一个响

应报文后，会设置一个 wait 变量为 8，改变量非零时每隔 500us 减 1，当 wait 为 0 时才能继续发送下一响应报文，这就满足了前后数据帧发送要相隔 4ms 的要求。

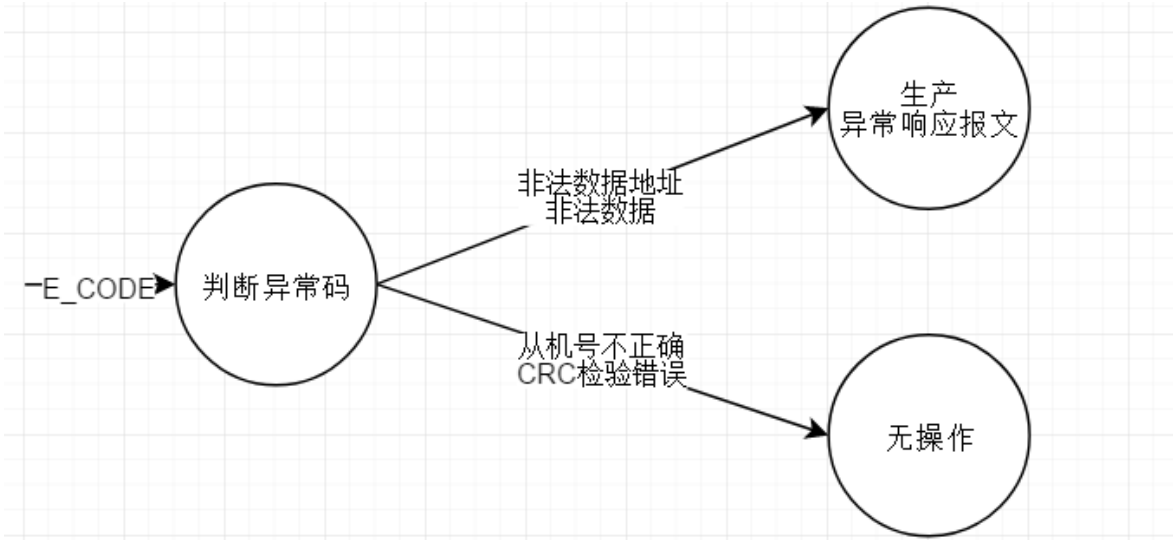
Mid 层：对请求报文进行预处理，包括从机号检验、CRC 校验、根据功能码将请求报文分给对应处理程序。如果非本机号或 CRC 校验错误，则忽略该报文。如果不支持功能码，则返回 01 异常码 llegal lnstruct。



High 层：由 Mid 层请求报文，完成相应操作，支持包括 01、02、03、04、05、06、0f、10 八条指令，对寄存器或者线圈进行相应操作。执行过程中，对功能码指定的地址、数量进行检查，如果出现错误，则返回相应的异常码。最后将得到的响应报文返回给 Mid 层。



Error 层：根据不同的异常码，封装异常报文帧，并传给 Mid 层。



②ADC 模块：

对于 ADC，此处 ADC 的转换频率大概为 12M/90=130KHz（SPEED=00），不同 ADC 之间的测量是串行的，所以轮流测量每个 ADC 端口，以时间片轮转的形式“并发”得到每个 ADC 端口数据。

ADC 框架如下：

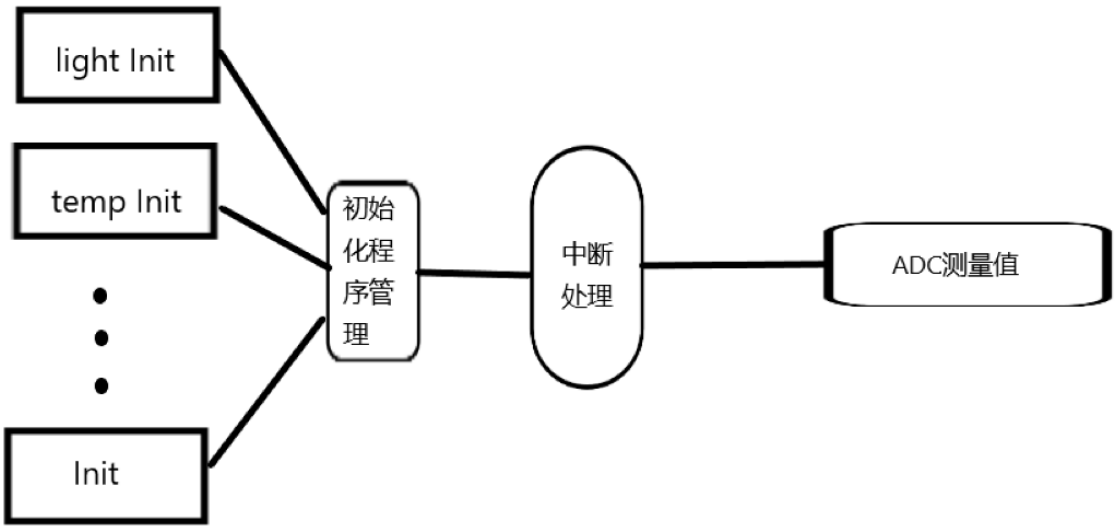
我们初始化不同的 ADC，得到每个 ADC 的初始化函数。

接下来，需要将这些 ADC 初始化函数注册到初始化管理模块中

初始化管理模块以 10ms 的时间片轮流初始化不同 ADC

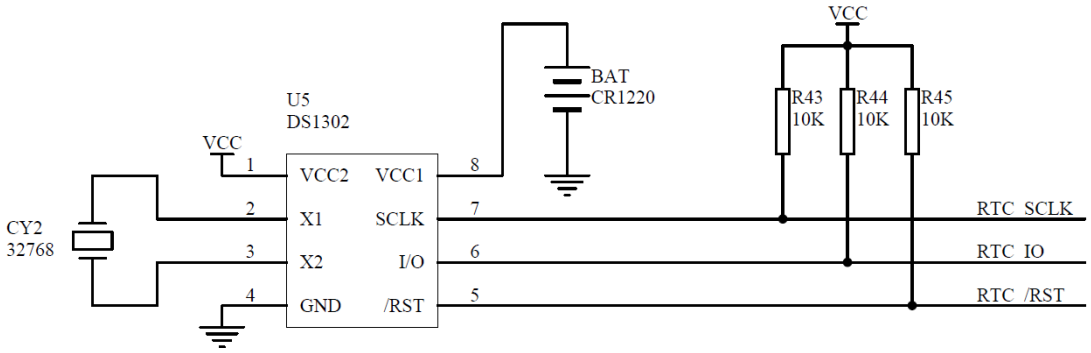
ADC 中断处理会根据不同的初始化函数获得不同 ADC 端口的测量值，并将测量的原始数据存放在每个 ADC 端口对应的测量值中。

比如，初始化管理程序在 0-10ms 初始化光强 ADC 端口，然后得到光强值存入 ADC_Num[0].初始化管理程序在 11-20ms 调度初始化温度 ADC 端口，然后得到温度值存入 ADC_Num[1].



③实时时钟模块

引脚编号	引脚名称	引脚功能
1	Vcc2	主电源引脚，当Vcc2比Vcc1高0.2V以上时，DS1302由VCC2供电，当Vcc2低于Vcc1时，由Vcc1供电。
2	X1	这两个引脚需要接一个32.768K的晶振，给DS1302提供一个基准。特别注意，要求这个晶振的引脚负载电容必须是6pF，而不是要加6pF的电容。如果使用有源晶振的话，接到X1上即可，X2悬空。
3	X2	
4	GND	接地。
5	CE	DS1302的输入引脚。当读写DS1302的时候，这个引脚必须是高电平，DS1302这个引脚内部有一个40k的下拉电阻。
6	I/O	这个引脚是一个双向通信引脚，读写数据都是通过这个引脚完成。DS1302这个引脚的内部含有一个40k的下拉电阻。
7	SCLK	输入引脚。SCLK是用来作为通信的时钟信号。DS1302这个引脚的内部含有一个40k的下拉电阻。
8	Vcc1	备用电源引脚。

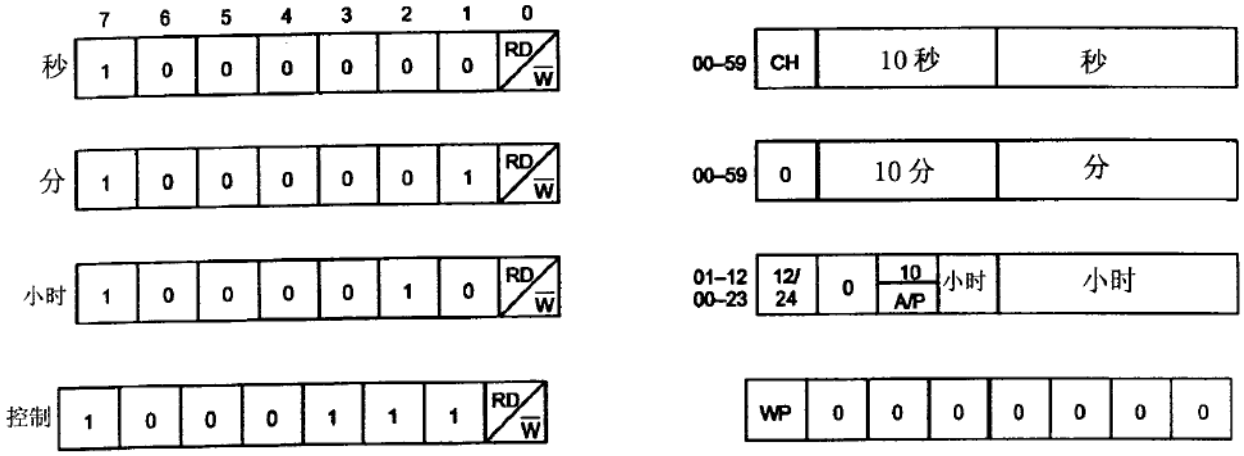


先了解 DS1302 的工作机制：

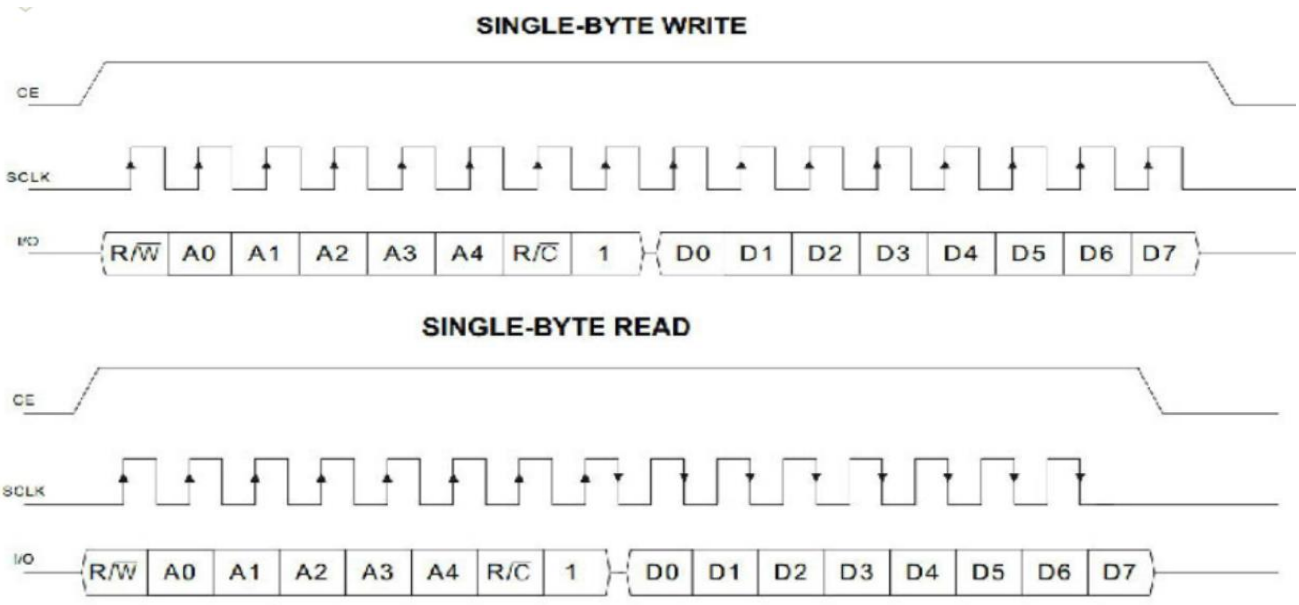
DS1302 内部有多个寄存器来对应存储年月日时分秒，如下图，左边是对应读写对应寄存器时所需要的命令，后面继命令后读写的实际寄存器。

比如，当我们要读秒，那么需要先写入 0x81（1000 0001），然后在才可以读取秒寄存器。控制寄存器，当 WP=0 时，才可写这些寄存器，否则不可写。

A. 时钟



DS1302 在进行读写操作时最少读写两个字节，第一个是控制字节，就是一个命令，说明是读还是写操作，第二个时需要读写的数据。对于单字节写，只有在 SCLK 为低电平时才能将 CE 置高电平，所以刚开始将 SCLK 置低，CE 置高，然后把需要写入的字节送入 IO 口，然后跳变 SCLK，在 SCLK 下降沿时，写入数据。



实现实时时钟需要下列几个步骤：

初始：将秒寄存器的 CH 位置 0，开启实时时钟

运行：每隔 100ms 重新读取时分秒寄存器的值，并保存在_systemtime_结构体中。

渲染：每次重新获取时分秒后，重新渲染在数码管上。4

④非易失存储器

非非易失存储器和 stc 芯片之间使用的是 IIC 通讯协议。

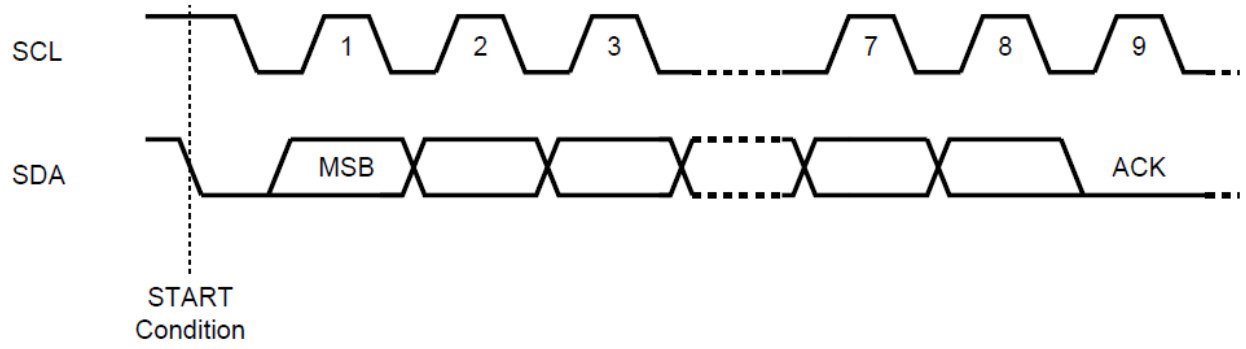
IIC 通讯协议如下：

初始化：IIC 的初始化为 SDA 和 SCL 均为高。

开始信号：处理器让 SCL 时钟保持高电平，然后让 SDA 数据信号由高变低就表示一个开始信号。同时 IIC 总线上的设备检测到这个开始信号它就知道处理器要发送数据了。

数据传输：SDA 上的数据只能在 SCL 为低电平期间翻转变化，在 SCL 为高电平期间必须保持稳定，IIC 设备只在 SCL 为高电平期间采集 SDA 数据。

停止信号：处理器让 SCL 时钟保持高电平，然后让 SDA 数据信号由低变高就表示一个停止信号。同时 IIC 总线上的设备检测到这个停止信号它就知道处理器已经结束了数据传输，我们就可以各忙各个的了，如休眠等。



Operation：

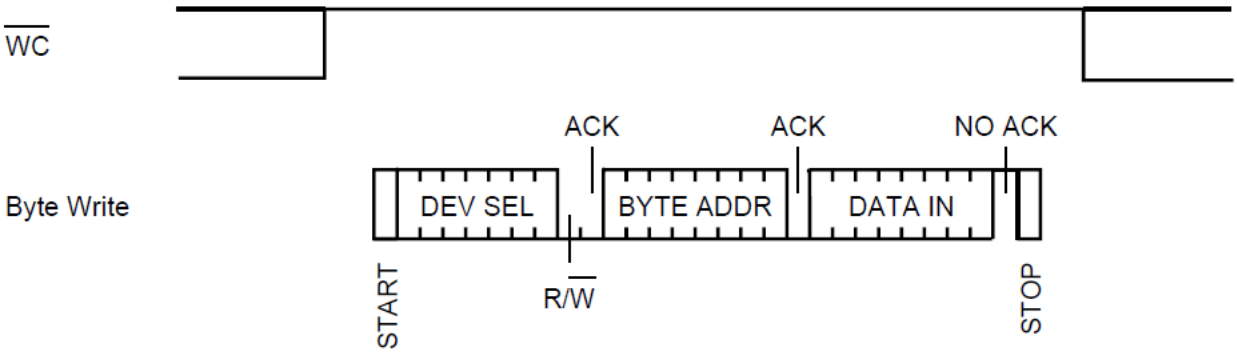
Table 2. Device Select Code

	Device Type Identifier[1]				Chip Enable[2,3]			RW
	b7	b6	b5	b4	b3	b2	b1	b0
M24C01 Select Code	1	0	1	0	E2	E1	E0	RW
M24C02 Select Code	1	0	1	0	E2	E1	E0	RW
M24C04 Select Code	1	0	1	0	E2	E1	A8	RW
M24C08 Select Code	1	0	1	0	E2	A9	A8	RW
M24C16 Select Code	1	0	1	0	A10	A9	A8	RW

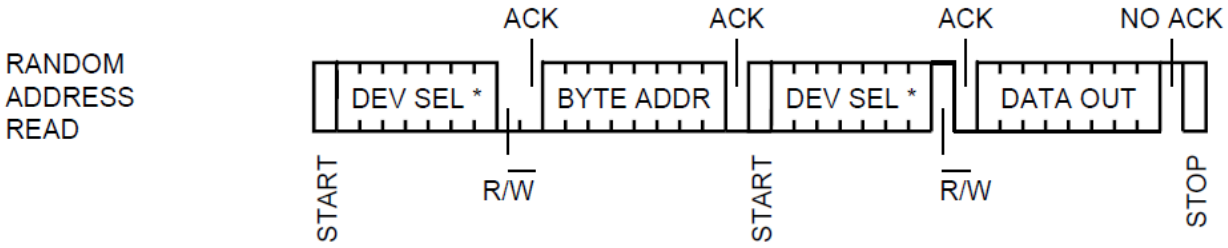
对 M24C08 进行操作时，首先要写入该操作对应的操作码，然后再写或者读数据。

比如，读数据的时候。首先通过上面可以知道写对于的操作码是 0xa1，所以我们先通过 IIC 向 M24C08 写入一个 0xa1，然后才能读数据。

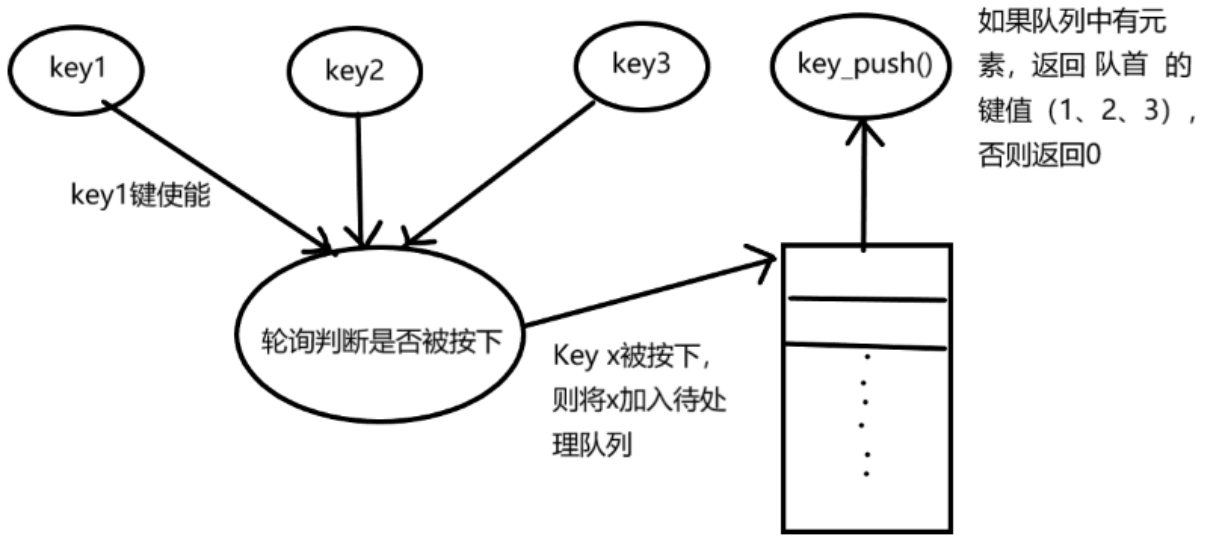
写数据如下图：首先写入操作码，然后写入地址，最后写入数据。



读数据如下图：首先写入写操作码 0xa0，然后写入地址，接着写入读操作码 0xa1，最后读取数据即是给定地址的数据。

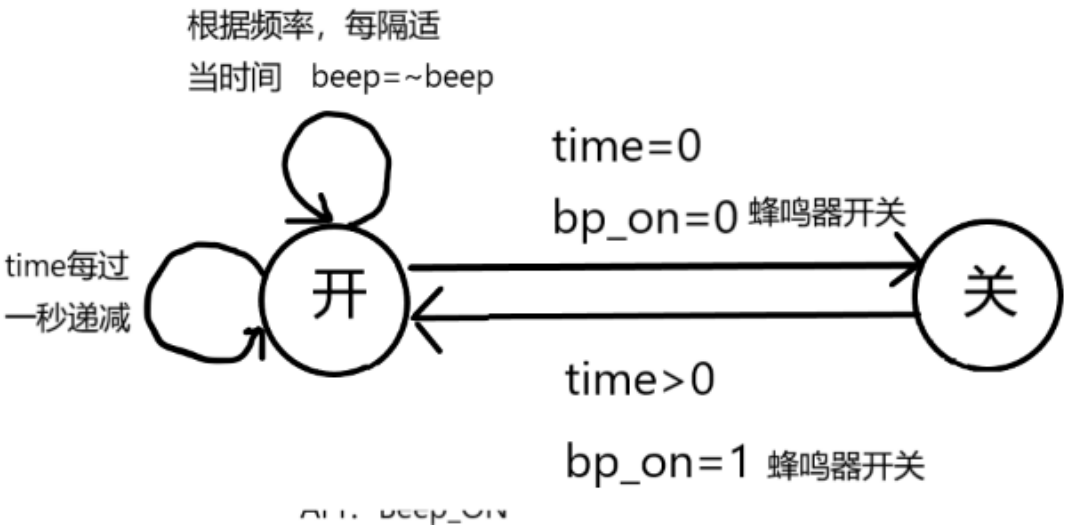


⑤按键模块：



按键模块：当某个键被使能后，我们采用轮询的方式进行消抖检查按键是否被按下秒，如果被按下，则将被按下的键值（1、2、3）加入到循环队列中，等待 CPU 通过 key_push()检查是否有按键的信息，如果有，则做出用户自定义的按键行为，每次检查完就会将队头的按键信息清除。

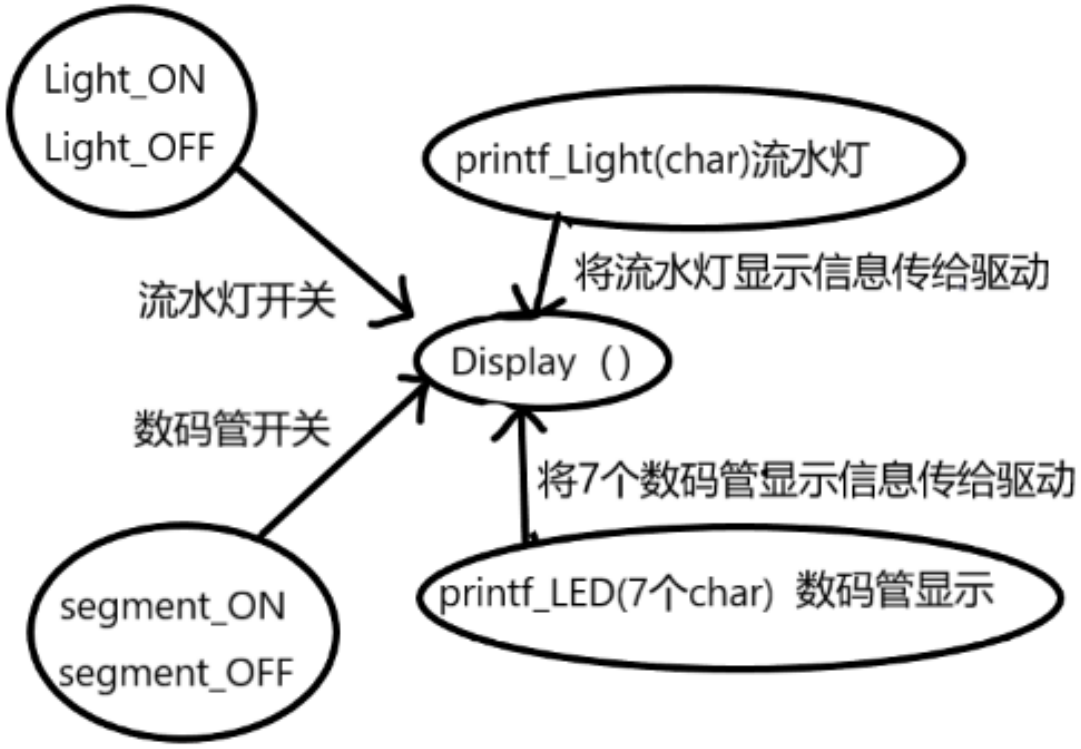
⑥蜂鸣器模块：



蜂鸣器模块：如果蜂鸣器开关被打开，则按照设定的频率发出声音，如果蜂鸣器被关闭，

将蜂鸣器引脚设为低电平后进入关闭状态。

⑦数码管&LED 灯模块



数码管&LED 模块: 这个模块是最经常使用的, 所以要尽可能地让接口更加的常用和方便。提供给用户的接口如上图所示，其中 Display（）是驱动程序，不需要用户调用。

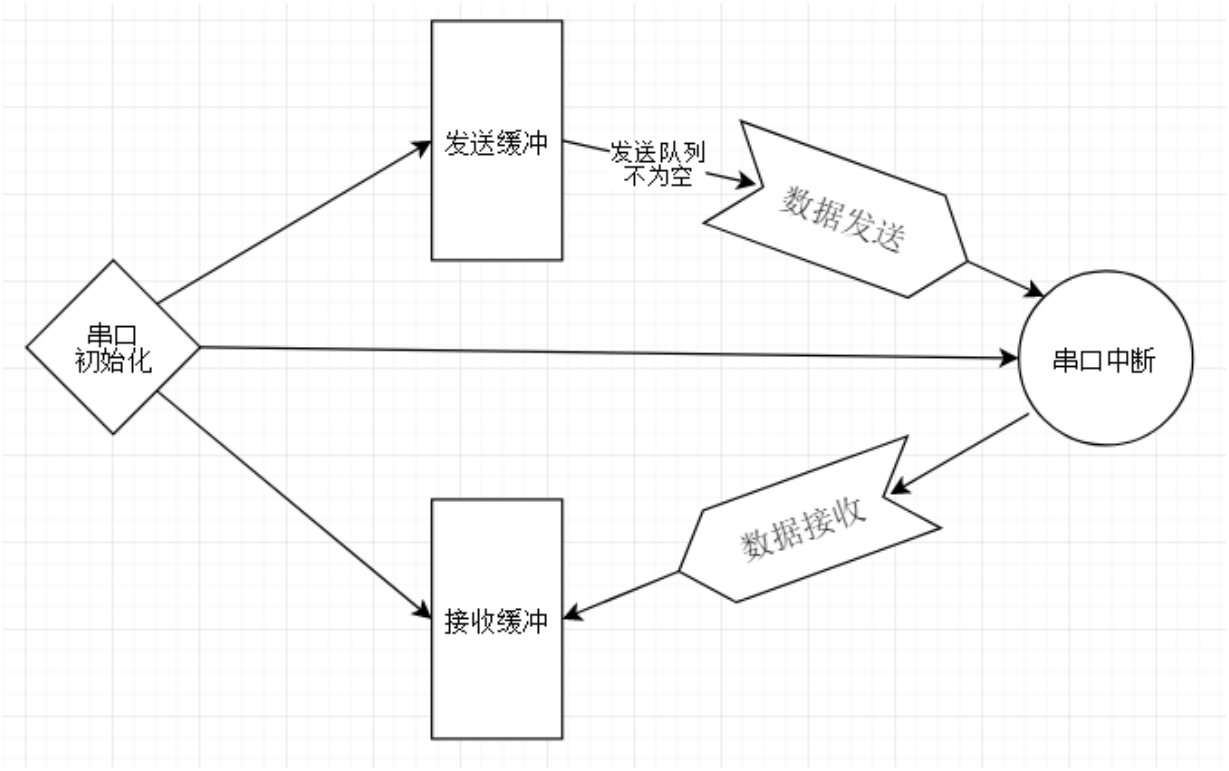
用户接口主要包括 Light_ON、OFF（开启关闭 LED 灯）, Segment_ON（开启关闭数码管）, print_Light()流水的显示内容，print_LED()数码管显示内容。这些接口在实现过程中都是修改该模块中 LED 结构体中的参数。驱动 Display（）则是通过该结构体进行硬件上的实际行为。

⑧串口模块

串口发送：如果发送缓存不为空，则数据发送模块将依次发送发送缓存中的数据。

串口接收：如果接收缓存不满，则数据接收模块将串口接收的数据保存到接收缓存

注明：接收和发送缓冲都是基于循环队列实现



- **实验心得**

本次实验是一个大型的综合实验，在本次实验中，使用了单片机的很多内外部外设，另外还完成了 ModBus 协议下的从机，实现了八种功能码及其部分异常响应。这次实验或许是我有史以来代码长度最长的一个项目，大概有 1000 多行。从按键、数码管、串口、ADC、实时时钟、非易失存储器到 ModBus 协议，最后还需要一些步骤将这些模块有机的结合在一起。

每一个功能的实现，我都要求自己不能够像写流水账一样，要对代码的结构，层次有着逐渐加深的认识，不能够单单为了完成某个功能，图方便去破坏程序的层次性。

在实现 ModBus 的过程中，尝试了其他的思路，最终确定上述的三层结构。当完成之后，我一直使用的是串口助手进行验证，并且地址和数据部分都是字节类型，后来再使用 ModBus 虚拟主机的时候，出现了一些问题，因为 ModBus 标准下地址和数据部分是 16 位，由于分层只需要在每个功能处理函数中修改一下地址和个数占有的字节数即可。

本次报告中不包含代码截图，工程文件内已经添加注释。毕竟代码只是工具，最终留下的只有框架、协议、思想。