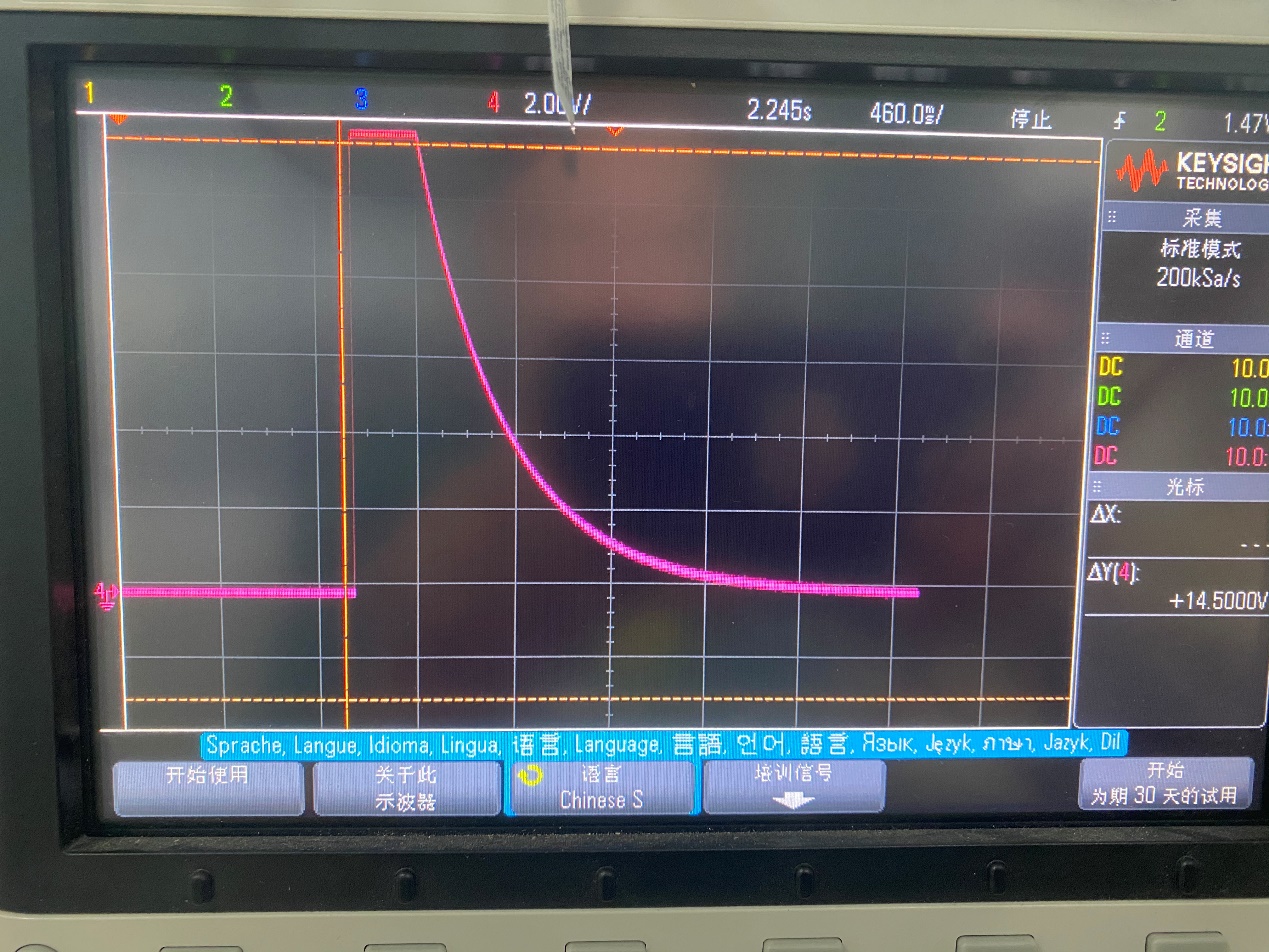
# 9.1

1. 短路的时候关闭总线电压
2. **STM32掉电上电以后不复位的解决办法**——加一个延时，加了20ms的延时。原因是硬件上电比较慢，软件运行比较快，还没来得及初始化就过去了。**Ucosii**中把初始化加在启动任务中，在启动任务中开头加一个延时就可以解决这个问题。在没有ucosii的程序中一般是不会有这个问题的，在main函数中运行就可以了。
3. **Ucosii中的任务该如何排布，为什么有的任务放进去以后其他任务就不能运行了呢。或者说其他任务运行它不运行。**
4. 明天来了以后把后端的电容拆掉试一试，看看有没有问题。

# 9.2

1. 12V的控制电路后端接了三个电容，然后在断路的情况下会自己输出电压。经过检查发现是因为PMOS管后级接了三个电容的原因，电容放不了电，所以检测会有电压。给输出接上回路以后示波器的波形是这样的（从掉电状态到上电）



1. **研究一下OS任务切换系统，尤其是和时间有关的**

# 9.3

1. 先写文档，写完文档去看OS，去看任务切换原理，主要是和时间相关的。带着一个问题，假如某个任务占用时间很长，但对实时性要求很高该怎么处理。
2. 漂移的问题怎么解决
3. 之前检测雷管思路进入了误区，在实现识别第一发的功能上想的是检测到雷管，第一发检测到的就是第一发雷管。但是没有考虑另一个因素，那就是第一发的条件就是，刚开始没有电流，或者说只有绞线的电流
4. 使用定时器定期产生一个中断，然后让判断总线是、否为零的函数把总线电流的状态放在结构体中
5. 还有一个想法就是：在检测到高电平以后，就对之后的电平
6. 判断雷管是第一发雷管的方法：
   1. 使用定时器，定期判断总线电流是否为零。然后检测是否有雷管接入。
   2. 因为测到的电流漂移较大，所以在检测雷管接入的时候先检测上限的电流能到多少，然后再检查下限能低到多少。
7. 把延时改成定时器中断。（在OLED显示任务中延时太长了，占用了太长时间的CPU）

# 9.4

1. 
2. 为什么两个任务会冲突，猜想一：一个任务是需要调用AD的底层函数的，而AD底层函数是一种资源，当访问这个资源的时候，优先级会被提高到很高，使用后就会被释放。两个无关的任务看看会不会出错。把使用AD的任务，换成LED4亮灭的任务，观察
3. 两个任务会冲突的原因是因为没有被释放掉。
4. 功耗怎么计算，怎么考虑，考虑芯片，考虑电流
5. 听硬件课程
6. 主机模式，用Bootloader
7. LINEAB输出电压有点低，400发雷管输出电压就变成了6.4V。尝试用DAC输出电压去控制LINEAB的输出电压。
8. 文件系统
9. 休眠模式
10. 中断系统优先级。怎么分配

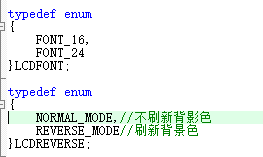
# 9.7

1. 今天任务：把详细设计初稿写完，有时间再研究一下STM的睡眠模式怎么用软件切换。

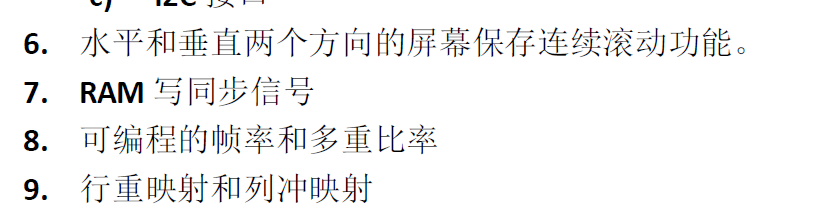
9.8

1. 今天任务：继续写详细设计初稿，有时间看一下C语言方面的书籍，毕竟在这方面有欠缺。
2. 想到一个问题，**运放在长时间满量程情况下会不会坏掉**
3. **数组在源文件中直接这么声明？？？**

# 9.9

1. **想明白了就是自己所加的开机提醒和自动关机由于AD采集到的数据浮动太大，当电池的电量消耗在10.5V左右 的时候，这时候判断就不准确了。**
2. FPC怎么画
3. 为什么枚举不用写数据类型

而且可以直接作为函数返回类型使用？？？

1. SSD1306内嵌对比度控制器，显示RAM和晶振
2. 

# 9.11

当字符串输入的是汉字的时候，编译器怎么处理

字符串——以\0结尾

**汉字字符串在计算机中如何存储**——“计算机”，这样的话怎么存储

# 9.15

今天完成了详细报告设计，但是最后的不完美就是，关机那里。把自己的想法写出来。

**想法1**：通过删除任务来实现关机（是否能够真正地关掉任务，看门狗怎么解决，看门狗会一直复位）

**想法2**：通过简单的关闭总线电压来实现。（但是问题就在于如何解决资源访问冲突）

**想法3**：老老实实采用待机模式

**想法4**：关机只关掉报警，其他的不管

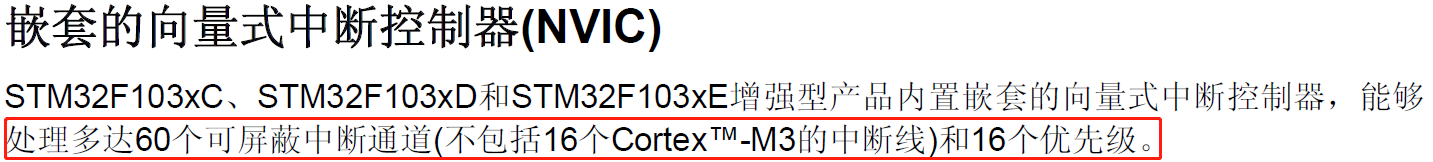
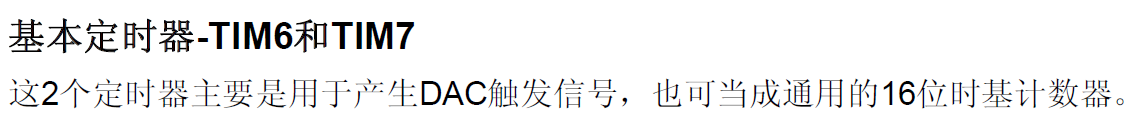
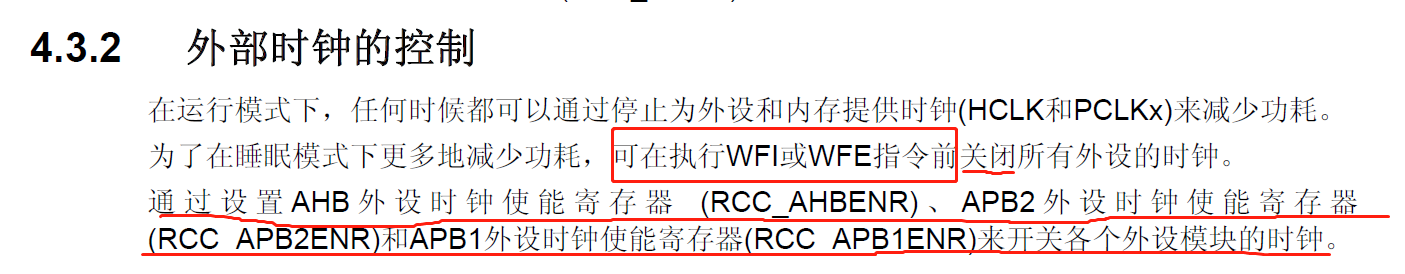
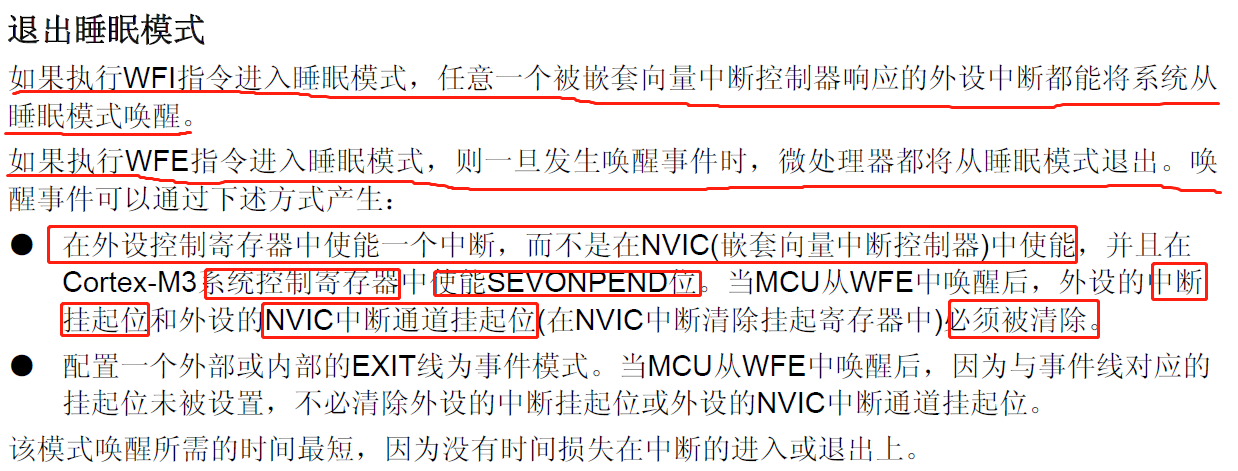
POR,PDR,PVD这是寄存器还是说是模块的名称

纽扣电池是为RTC和后备寄存器供电的

SDIO是什么

修改PCB文档

9.15

1. 把睡眠模式，停机模式和待机模式搞清楚
2. FSMC 静态存储器控制器
3. 定时器分为通用，高级控制，基本这三个怎么理解
4. CAN总线——控制器区域网络
5. DAC
6. ARM是32位的RISC处理器
7. 怎么理解
8. 
9. 
10. 待机模式，停机模式，睡眠模式
11. 睡眠模式
12. **进入睡眠模式前可以使用预分频器，降低外设的时钟——见时钟配置寄存器（RCC\_CFGR）**
13. 单片机里面有什么，电源、时钟、存储
14. 
15. 
16. **RTC 在我的工程里面有开启么。**
17. 复位——系统复位，上电复位和备份区域复位
18. 更改启动任务程序，把循环改为删除任务
19. 这几天一直在考虑一个事情，就是电池没电怎么处理。
20. 找不到为什么电量低于阈值以后不能关闭总线电压的原因，大概率猜测是因为其他程序又开启了总线电压。
21. 任务能不能减少。去看其他工程里面的任务是怎么建立的。任务之间的资源如何访问才不会出错
22. 待机模式，停机模式和睡眠模式都是怎么操作的。去理解什么是全局中断，
23. 写一个进入待机模式的代码。电压不够的时候直接进入待机模式，然后重新上电，电池电量还是不够的话再次进入待机模式。

# 9.16

Flash什么时候擦除。写入的时候擦除什么地方。读取应该是最简单的。假如说指向某一个地址写入数据，能不能不动其他地方的情况下就写好数据。

增益和放大倍数有什么关系。

# 9.17

编程日志：在实时系统永远要安排好每一个使用某一个常用任务的是优先级。不然会打架。而且打得很凶，打后的结果莫民奇妙。

今日关键词：控制权——在系统中对于小白来说一个难题就是如何处理好每一个任务对资源的使用权，分配好他们之间的前后顺序。一个规划师的精妙之处就是在于提前安排好每个任务。

# 9.21

二极管的压降，在正常电流中测。

栅极引出来，用示波测

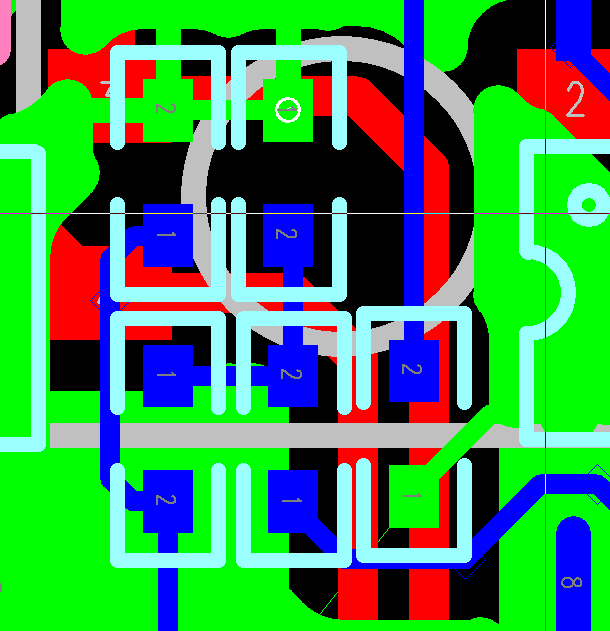
空闲的运放怎么处理

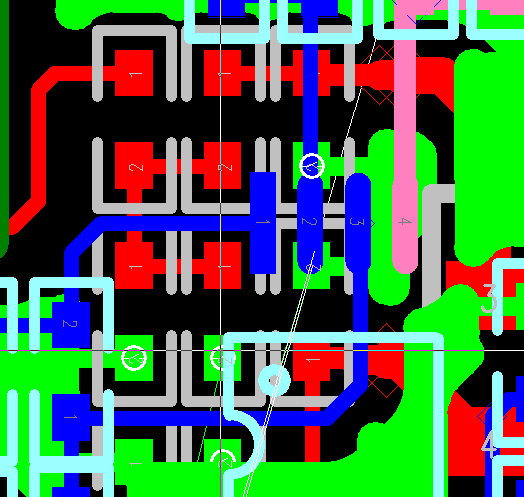
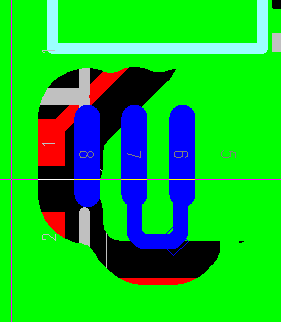
# 9.22

布线，和电源相关的线，一律不画，只先画信号线

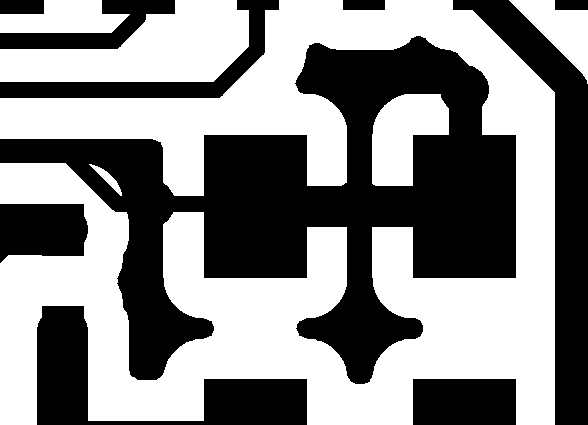
时序要匹配

为什么有的地方灌铜灌注不了呢





# 9.24

YOU

铺铜皮铺成这样能行不

# 9.25



# 9.27

Output 用PMOS管做控制开关的10R电阻要去掉。

蓝牙模块电源加一个开关用单片机去控制

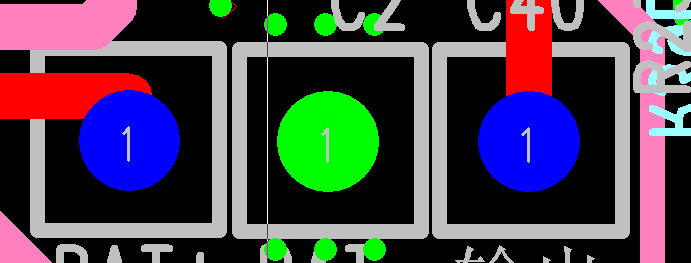
蓝牙模块天线所在的地方不要铺铜

要么加大封装 要么减小容值 33uF/16V

33uF/10V

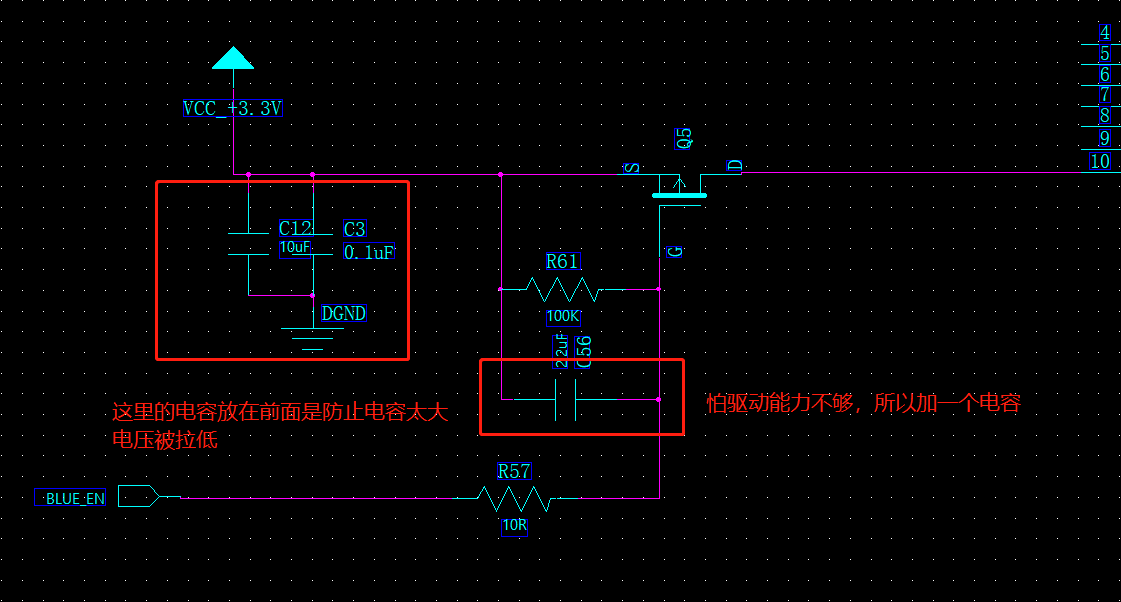
低电压的开关电源可以用一个PMOS管就可以搞定

若用低电压去控制高电压的话就用一个PMOS管和一个NMOS管就可以搞定



封装太近了

# 9.28



电容的用处：

写一下新板子的代码:

新板子改动的地方：去掉了拨码开关；去掉了量程切换；增加了蓝牙模块；增加了UART通讯；接口改成了FPC。代码中需要改动到的地方，有三个地方，把量程切换和读取拨码开关的相关代码删掉。增加UART通讯代码，把之前USB和上位机通讯的命令部分转移到蓝牙部分。USB部分仍然保留。

**之前想得用电流队列去识别雷管上线的功能是这样的**：定时地去采样数据并存储在队列中，队列的长度是自己设定的，假如说100个数据。队列是不断地循环去写。然后之前是读取完一次数据以后等500ms，这样的话就存在一个问题，时间间隔太长识别就不准了。然后使用队列的方法是缩短读取时间。然后又陷入在之前的死区里面，**间隔多长时间去读**。这样的话还是和之前一样的毛病，有没有队列都无所谓。

**正确的思路应该是这样**：同样是间隔一定的时间去读取数据，但是设计队列的长度的时候就考虑到 长度\*采样时间=实际间隔的时间（300ms），然后判断的时候就去读取队头和队尾。这样就消除了硬性地暂停程序，去等待一定时间然后过了这个时间以后再去读取电流。

队列有**静态**和**动态**之分

动态演示队列，栈等数据结构的基本操作

<https://www.sohu.com/a/276607433_478315>

为什么队列 栈等数据结构要规定好进出顺序。就不能随意放置么。

我自己想了一个数据结构。用一维数组实现：建立100个数组（设立成全局变量，在结构体中声明），循环顺序地去往这100个数组里面放数据。然后存完这100个数组所需要的时间是300ms，然后如何判断雷管上线呢，当需要判断的时候去读取这100个数组的头和尾。或者说去取头部十个数据，尾部十个数据。取平均，然后根据平均值去判断。这样设计有弊端。弊端是这样的，假如放满数据以后重新去从头到尾放数据的时候，就会造成这样的现象。假如每隔1s放一个数据，原来数组里面放的是从1s到10s的数据，然后重新放数据的时候，数据里面的顺序是这样的，第11s的数据 第2s的数据 第3s的数据 ……这样他就不是顺序的了。所以这个数据结构不对。应该是这样的数据结构，放满以后，第一个数据顶出去，然后第二个数据顶上去，新数据放在最后一个位置上。可实现的方式是这样，用指针，当满了以后放入新数据的时候，放在原来第一个数据的位置，此时，原来的第2个位置变成头，第1个位置变成尾。在静态存储区里面不是按照顺序放的，但是在抽象出来的层面上（也就是在外部表现上是顺序排列的）。也就是说刚刚设想的那个数据结构仍然是最底层层面的操作。不是显示给外部看的层面。外部看的层面应该再建立一个变量，从0-100，对应的数据永远都是顺序的。而他的下一层就不是顺序的了。

问：需不需要建立两个变量，局部变量和全局变量。局部变量的100个数组是顺序显示的，而全局变量不是。

答：不用，虽说这样很直观，但是多了一步操作就浪费了一步的时间，而且还多暂用了一倍的存储空间。最好的方式是用指针去指向。

实现方式：建立两个指针头指针front和尾指针rear。刚开始头和尾指针都初始化为0。接着放进存数据，放一个数据头指针front还是0，尾指针rear加1。还需要设立一个计数器，当计数器到内存存放最大值的时候计数器归零。然后每放入一个数据计数器加1。当计数器碰到内存存放最大值的时候，也就是数据放满了。此时尾指针指向最后一个数组。再存入新的数据的时候，头指针front增加1，尾指针指向新存入的数据。然后就这样存。当尾碰到最大值的时候，尾rear变成0，头碰到最大值的时候，头front变成0。就这样循环。

typedef int datatype 这句话是什么含义。

datatype queue\_array[MAX\_QUEUE\_SIZE]这句话是什么含义

# 9.29

1. **struct** Link
2. {
3. **int** data;
4. **struct** Link\* next;//这是干什么，是指针型变量么
5. };

这种写法和在KEIL中32的写法不一样，放在32中应该这么写

1. **typedef** **struct**
2. {
3. **int** data;
4. }Link;
6. Link\* next;
7. **extern** Link\* next;

这两种写法应该都一样，都能够被编译

问：结构体在声明的时候后面有一个变量表，这个变量表的作用机理是什么，是给这个结构起名字还是建立同样结构的结构体？

1. **struct** Queue
2. {
3. **struct** Link\* front;//在结构体里面只有一个数据，有两个结构共用了这个结构体，数据会不会冲突，为什么说冲突呢，
4. **struct** Link\* rear; // A和B都在使用这个结构体，A往这个结构体里面放数据，B也往这个结构体里面放数据，然后读取数据的时候读的是A的数据还是B得数据。还是说A和B在使用结构体的时候同时复制了这个结构体的结构，存取数据都是在不同的存储位置上面，不是在同一个位置上存取。
5. **int** size;
6. };

## 结构体的定义格式

1. **struct** tag {
2. member-list
3. member-list
4. member-list
5. ...
6. } variable-list ;

有三个要素：tag、member-list、variable-list，在定义的时候这三个要素要么三个都有要么选择其中两个。**但是**不能低于两个。

1. **typedef**

作用为类型取新名字

循环队列：数组实现 链表实现

先用数组实现：实现的要求 队列的初始化 入队 没有出队 只有读取队列内的数据 重点是对对头队尾位置顺序规则的制定

最大元素数量

队头

队尾

初始化：队头=队尾=0

入队：

第一阶段：队头不变，放入一个数据，队尾+1；直到队尾到达最大元素数量，然后队尾为0，队头+1。建立一个计数器，去记录队尾从零开始放入数据的次数。计数器满也就是队尾满，然后

先放数据还是先判断，加1放在什么位置？

假设先判断；判断完以后再放数据，当上一次rear指向数组最后一位，计数器count=max-1；然后判断count < max，将这次数据放进去以后rear+1（此时这个是无效的），计数器count=max。然后第一轮完成，进入第二轮判断条件为count=max和第二轮标志位，经过判断后进入Y，rear被强行等于front，front+1，然后数据存入原来front的位置也就是0。

一共需要三个函数还有一个结构体

初始化：Queue\_Init()

放入数据：Queue\_Push()

读取出数据：队列头和队列尾

1. **typedef** **struct**
2. {
3. **int** queue[ARRAY\_MAX];
4. **int** front;
5. **int** rear;
6. **int** count;
7. **int** Queue\_flag;
8. }QUEUE;
10. QUEUE queue;
11. **extern**  QUEUE   queue;
13. **void** Queue\_Init()
14. {
15. queue.front = queue.rear = 0;
16. queue.count = 0;
17. queue.Queue\_flag = 0;
18. }
20. u8 Queue\_push(u8 data)
21. {
22. **if**(Queue\_flag != 0x01)
23. {
24. queue.queue[queue.rear] = data;
25. queue.rear += 1;
26. queue.count += 1;
28. **if**(queue.count = ARRAY\_MAX)
29. {
30. queue.Queue\_flag = 0x01;
31. }
32. }
33. **else**
34. {
36. }
38. }

9.30

写完检测雷管上电代码

1. #define ARRAY\_MAX 100;
3. **typedef** **struct**
4. {
5. **int** queue[ARRAY\_MAX];
6. **int** front;
7. **int** rear;
8. **int** count;
9. **int** Queue\_flag;
10. }QUEUE;
12. QUEUE queue;
13. **extern**  QUEUE   queue;
15. **typedef** **enum** {ERROR = 0, SUCCESS = !ERROR} ErrorStatus;

18. **void** Queue\_Init()
19. {
20. queue.front = queue.rear = 0;
21. queue.count = 0;
22. queue.Queue\_flag = 0;
23. }
25. ErrorStatus Queue\_push(u8 data)
26. {
27. **if**(Queue\_flag != 0x01)
28. {
29. queue.queue[queue.rear] = data;
30. queue.rear += 1;
31. queue.count += 1;
33. **if**(queue.count = ARRAY\_MAX)
34. {
35. queue.Queue\_flag = 0x01;
36. }
37. }
38. **else** **if**(Queue\_flag = 0x01)
39. {
40. **if**(queue.count == ARRAY\_MAX)
41. {
42. queue.rear = queue.front;
43. queue.count = 0;
44. **if**(queue.front == queue.rear == ARRAY\_MAX)
45. {
46. queue.front = 0;
47. queue.queue[ARRAY\_MAX] = data;
48. **return** success;
49. }
50. }
51. **if**(queue.rear = ARRAY\_MAX)
52. {
53. queue.rear = 0;
54. }
55. **else**
56. {
57. queue.rear += 1;
58. }
59. **if**(queue.front = ARRAY\_MAX)
60. {
61. queue.front = 0;
62. }
63. **else**
64. {
65. queue.front += 1;
66. }
67. queue.queue[queue.rear] = data;
68. **return** succuss;
69. }
70. **else**
71. {
72. **return** FAIL;
73. }

76. }

有一个问题：怎么知道程序运行一周用了多长时间？





问题是front=rear