5 0000 0101

39 0010 0111

137 1000 1001

OSI 开放系统

ISO 国际标准

协议

~~clien~~

server 斜体

module 加粗

~~echo~~

~~open~~

~~open~~ 删除线

```
int main()
{

    return 0;
}
```

`SKIP`

ANGLE

accleration

$$I = \Delta p$$

$$p = mv$$

$$I =_{\Delta} p = mv_{末} - mv_{初}$$

$$f\sum_{n=1}^{n}$$

LEVEL

LEVEL

$$u(x) = kp(err(t) + \frac{1}{T}\cdot\int err(t)dt + \frac{T_D derr(t)}{dt})$$

充电控制引脚

x，y

x = 1,有USB插入

x = 0,无USB插入

y = 1,充满电

y = 0,充电中

USB	Charge	状态
1	0	USB插入，充电中
1	1	USB插入，充满电
0	0	USB未插入
0	1	USB未插入

baidu

安全灯亮度 60，频率同普闪

智能模式频率更改，周期2s, 亮100ms,灭1900ms

骑行模式抖光问题，优化或缩短爬坡时间为100ms,下坡600ms,高亮100ms,熄灭1200ms

C2滤波电容，VBUS+5V输出，

SMF静电二极管，

R2,R15分压给USB-IN，5V分压后大概2.5V，所以当有USB插入时，检测到高电平

然后作为EMC5755的输入，

CHGB作为检测充满电的引脚，充满电拉低，充电时拉高

PROG作为控制电压输出引脚，通过电阻的大小来改变提供的BAT引脚的电压

BAT引脚接LDO输入，作为SGM2306LDO电压输入，

LDO稳压电路，通过VIN引脚输入的电压，来稳定的输出电压，用以供给单片机供电。

一般输入比输出电压高于0.5~6VLDO即可稳定输出，这里LDO输出2.8V

COB电路

BAT电池供电作为ETA2421的VIN输入，之前用一个C5 10uF来滤波

EN-COB利用单片机PWM波输出，输出到CE引脚来使LX引脚输出正确的电压控制COB灯的亮度

LX引脚接一个2.2UH和一个10uF的电容来进行滤波

电感过滤杂波，仅用一个电感效果并不好。所以多加了一个电容

FB是反馈引脚，LED的启动电压大于2V，压差0.7V，利用FB接一个电阻对其进行分压，来得到适合的电压

RT温敏电阻，温度升高，阻值降低

利用分压电路，一个温敏电阻，一个固定阻值电阻，接一个电容减缓电流，不至于突变

ADC采集电压电路

BAT电压输入，利用分压电路，ADC采集分压值，然后进行计算，同样接一个电容

光敏电阻也是同样的原理

PMOS管，低电平导通，S极接正，D极接负

- 箭头向外

NMOS管，高电平导通，D极接正，S极接负

- 箭头向内
- NMOS管是压控型器件，Vgs电压大于Vth开启电压时，内部沟道在增强的作用下导通，Vgs电压小于Vth开启电压时，内部沟道截止；
- Vgs电压越高，内部电场越大，导通程度越高，导通电阻Ron越小，但需要注意，Vgs电压不能超过芯片允许的极限电压；
- NMOS管一般作为低端驱动器件，源极S接地

PNP

- 箭头向内

NPN

- 箭头向外

结型场效应管

增强型N沟道MOS管

增强型P沟道MOS管

耗尽型N沟道MOS管

增强型P沟道MOS管

滤波算法

1. ANL　ORL XRL CLR SETB　RL　RLC RR　RRL AND　OR　ADD　SUBB MUL　DIV INC DEC DA　#data direct @Ri MOV　MOVC MOVX
2. AJMP　SJMP　LJMP　JMP　ACALL　LCALL　JZ　JNZ　CJNE　CJNZ　JC　JB　NOP

GATT，利用属性协议定义了服务框架，方便client和service之间通过服务请求/响应建立通信。

通过属性来发现，读取，写入这些属性数据，一个服务有多个属性

Attributes（ATT定义的最小数据实体，也是构成GATT的service，characteristic，Descriptor的基本元素）

- handle
- value
- permission
- type

GATT——通用属性配置文件层（Generic Attribute Profile）

利用属性协议来定义服务框架，方便client和service之间实现基于服务请求/响应的通信

GATT通过定义属性来发现，读取和写入这些属性数据，一个服务通常包含有多个属性

Client role: 发送服务，接收服务器的响应数据

Server role:　响应服务，发送给客户端响应数据

- UUID
  通用唯一标识符
- Attributes
  是ATT定义的数据最小实体，也是构成GATT的service，characteristic，descriptor的基本元素，每个Attribute都包含属性本身的信息和实际数据
    - Handle
    - Type

        - Value
        - Permissions
- GATT profile hierarchy
    - Profile
        - GATT定义的Profile由一个或多个Service组成
    - Service
        - 由一个或多个Characteristic构成，每个Service可以看做是完成特定功能或特定的数据或相关行为的集合
    - Characteristic
        - 每个Characteristic一般都包含一个数据或者一个公开行为，以及该数据的单位或者公开行为的的单位
- GATT feature and procedure
  GATT主要定义了11个功能，每个功能都映射到过程和子过程，这些过程和子过程描述了如何使用ATT来完成相应的功能
    -

GATT

UUID

Attributes

handle

type

permission

value

service

client

集显

- 集成在主板上，逐渐已经被淘汰

核显

- 集成在CPU上，一般够用

独显

- 独立的显卡，多用于游戏需求比较大

双显

- 核显和独显都有，自动调配

显卡

链表

typedef struct xxx

{

  struct xxx *next;

  gattInfo_t serviceInfo;

}serviceCBs_t;

结构体

属性封装成一个结构体

```
typedef struct attAttribute_t
{
    gattAttrType_t type;
    uint8 permission;
    uin16 handle;
    uint8 * const pValue;
}gattAttribute_t;
```

osal 操作系统

非抢占，实时操作系统，大while循环

任务ID，

SCL，SDA拉高

位段

```
struct name{
    unsigned int a:1;
    unsigned char b:3;
}Bit_Region;
```

```
srand((unsigned int)time(NULL));
int a = rand()%max+1;
```

链表

```
struct link{
    struct link * next;
    int elem;
}Link_l;
```

回调函数

共用体

ANL ORL XRL CLR SETB RL RLC RR RRC ADD SUBB MUL DIV DV INC DEC

@Ri direct #data A MOV MOVC MOVX AJMP SJMP LJMP JMP RET RETI ACALL LCALL JZ JNZ CJNE CJNZ JCJB NOP

结构体中定义了函数指针，函数指针定义功能，函数参数可由枚举定义，然后利用条件判断函数参数来分别实现对应功能，函数嵌套使用，结构体成员赋值

结构体做函数参数，然后在函数内部对结构体成员分别判断，就是以结构体成员来判断分别执行不同操作。

结构体定义了指针数据成员，然后该结构体作函数参数，

在函数中，又有内存拷贝（osal_memcpy()），定义了一个全局二维数组，将结构体变量的指针数据通过内存拷贝给二维数组，其中长度又是以结构体中定义的长度成员来进行判断

GPIO读写

```
void hal_gpio_write(gpio_pin_e pin, uint8_t en)
{
    if(en)
        AP_GPIO->swporta_dr |= BIT(pin);
    else
        AP_GPIO->swporta_dr &= ~BIT(pin);
    hal_gpio_pin_init(pin,GPIO_OUTPUT);
}
bool hal_gpio_read(gpio_pin_e pin)
{
    uint32_t r;
    if(AP_GPIO->swporta_ddr & BIT(pin))
        r = AP_GPIO->swporta_dr;
    else
        r = AP_GPIO->ext_porta;
    return (int)((r >> pin)) & 1);
}

#define     AP_GPIO     ((AP_GPIO_TypeDef *)AP_GPIOA_BASE)
#define     AP_GPIOA_BASE   (AP_APB0_BASE + 0x8000)//gpio
#define     BIT(n)      (1ul << (n))
typedef struct
{
    __IO uint32_t swporta_dr;       //0x00
    __IO uint32_t swporta_ddr;      //0x04
    ...
    __IO uint32_t ext_porta;        //0x50
```

```
    ...
}AP_GPIO_TypeDef;
```

PWM通道，使能，状态，用结构体定义

```
typedef struct
{
    bool  enable;
    bool  ch_en[6];
    pwm_ch_t    ch[6];
}pwm_Ctx_t;
```

0x04

霍尔效应

导体或半导体位于磁场中，且磁场方向与电流方向垂直。在磁场中，电子受到洛伦兹力的作用，根据左手定则（磁场穿过手掌，大拇指指向电流方向，四指指向的是洛伦兹力的方向）判断洛伦兹力的方向

$$F_L = eBv$$

洛伦兹力会使电子向内偏移，由此形成底层自由电子集合，上层表面空穴集结，产生内电场，产生电动势，称为霍尔电压，与洛伦兹力方向相反。当达到动态平衡后，

$$F_H = F_L$$

$$eE_H = eBv$$

$$E_H = Bv$$

$$U_H = E_H b = Bvb$$

电流等于电子流过单位横截面面积的电荷量，电子浓度为n，电子移动平均速度为v，板宽b，厚度为d，N型半导体自由电子参与导电，与电流方向相反，P型半导体空穴参与导电，与电流方向相同

$$N\text{型半导体}$$

$$I = -nevbd$$

$$P\text{型半导体}$$

$$I = nevbd$$

$$U_H = -\frac{IB}{ned}$$

$$R_H = -\frac{I}{ne}$$

$$R_H\text{霍尔系数}$$

$$K_H = -\frac{I}{ned}$$

K_H霍尔灵敏度

1010 0000 1100

1010 1111 1000

PID

比例，积分，微分

Proportion Integral Differential

闭环控制（有反馈）

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

K_p是比例增益，K_i是积分增益，K_d是微分增益

对上式进行离散化，假设采样系统时间▲t，则将输入e(t)序列化得到

位置式PID

$$u(k) = K_p e_k + K_i \sum_{i=1}^{k} e(i)\Delta t + K_d \frac{e(k) - e(k-1)}{\Delta t}$$

通过$\Delta u(k) = u(k) - u(k-1)$可得到增量式$PID$

增量式PID

$$\Delta u(k) = K_p(e(k) - e(k-1)) + K_i e(k) + K_d(e(k) - 2e(k-1) + e(k-2))$$

GAP

- 通用访问协议
    - 模式和过程
        - Mode
          当一个设备被配置成按照某种方式工作一段较长的时间，则称该设备正处于某种模式
            - Broadcast mode
            - Non-Discoverable mode
            - Limited Discoverable mode
            - General Discoverable mode
            - Non-Connectable mode
            - Directed Connectable mode
            - Undirected Connectable mode
            - Non-Bondable mode
            - Bondable mode
            - Periomic Advertising Synchronizability mode
            - Periomic Advertising mode

        - Procedure
          当一个设备被配置为在某一段有限的时间内执行某种特定的操作时，则称该设备正执行某个过程
            - Observation procedure
            - Limited Discovery procedure
            - General Discovery procedure
            - Name Discovery procedure
            - Auto Connection Establishment procedure
            - General Connection Establishment procedure
            - Selective Connection Establishment procedure
            - Direct Connection Establishment procedure
            - Bonding procedure
            - Periomic Advertising Sychronization Establishment procedure
            - Periomic Advertising Sychronization Transfer procedure
    - role
        - 外围（Peripheral role）
          在GAP规范中，只有Peripheral role才允许被发现
        - 中心（Central role）
          在GAP规范中，Central role执行发现Peripheral role的过程
        - 广播（Broadcaster role）
        - 观察（Observer role）

GATT

ATT

属性协议层（Attribute Protocol）
属性协议规定了属性的发现和读写访问的方法

ATT层定义了属性实体的概念，包括UUID，句柄，属性值，也规定了属性的读写，通知等操作方法和细节，这些与属性操作相关的内容称为属性协议，基于ATT层，可以构建出通用属性规范

- 属性
    - UUID（Universally Unique Identifier）
        - 典型16字节UUID格式：XXXX-XX-XX-XX-XXXXXX
    - Handle
      属性句柄犹如指向属性实体的指针，对端设备通过句柄来访问该属性
      属性句柄为有序排列，是一个 2 字节数
    - Type
    - Value
      属性值可以是一个数字或一个字符串
    - Permission
      属性的读写权限由ATT层之上的协议层规定，有效的读写权限包括：可读，可写，读写
- ATT FDU Format
    - 1 octet指令操作码
    - 可变长度的属性参数
        - Handle

        - Type
        - Value
        - Permission
    - 可选的12 octets的认证签名信息
- ATT protocol methods
    - Command
    - Request
    - Response
    - Indication
    - Confirmation
    - Notification

```
/**
    GATT Attribute Type format.
*/
typedef struct
{
    uint8 len;         //!< Length of UUID
    const uint8* uuid; //!< Pointer to UUID
} gattAttrType_t;

/**
    GATT Attribute format.
*/
typedef struct attAttribute_t
{
    gattAttrType_t type; //!< Attribute type (2 or 16 octet UUIDs)
    uint8 permissions;   //!< Attribute permissions
    uint16 handle;       //!< Attribute handle - assigned internally by attribute server
    uint8* const pValue; //!< Attribute value - encoding of the octet array is defined in
    //!< the applicable profile. The maximum length of an attribute
    //!< value shall be 512 octets.
} gattAttribute_t;
```

L2CAP

LL

```
hal_pwm_init();
hal_pwm_set_count_val();
hal_pwm_open_channel();
hal_pwm_close_channel();
hal_pwm_start();
hal_pwm_destroy();
hal_pwm_stop();
hal_pwm_module_init();
hal_pwm_module_deinit();
hal_pwm_ch_start();
hal_pwm_ch_stop();
hal_pwm_ch_enable();
hal_pwm_ch_reg();
```

```
hal_gpio_init();
hao_gpio_pin_init();
hal_gpioin_enable();
hal_gpioin_register();
hal_gpioin_unregister();
hao_gpio_pull_set();
hal_gpio_write();
hal_gpio_read();
hal_gpio_fmux_set();
hal_gpio_fmux();
hal_gpio_wakeup_set();
```

```
hal_uart_init();
hal_uart_deinit();
hal_uart_set_tx_buf();
hal_uart_get_tx_ready();
hal_uart_send_buff();
hal_uart_send_byte();
uart_hw_init();
uart_hw_deinit();
txmit_buf_polling();
txmit_buf_use_tx_buf();
```

放大：按比例放大，变化前后能量守恒

放大倍数：Xo : Xi

通频带：对不同放大电路的放大作用范围

上限截止频率FH，下限截止频率FL Fbw = FH - FL

开漏：NMOS管 0 ， PMOS管高阻态

推挽：NMOS管 0，PMOS管 1

更新广播数据

1. 获取新的设置从GATT属性
2. 更新广播数据存储
    1. 设置UUID
    2. 设置major
    3. 设置minor
    4. 断开所有连接
    5. 关闭广播
    6. 更新广播数据
        1. 更新广播类型
        2. 更新广告播出
        3. 设置发射功率，RF PHY TX POWER
    7. 设置复位广播实践，提示GAP/LL 会处理关闭广播事件 `osal_start_timerEx(simpleBLEPeripheral_TaskID, SBP_RESET_ADV_EVT,5000)`

1

dio:人类的能力是有极限的，人越是工于计谋，计谋就越会因为意想不到的事情而失败，除非成为超越人类的存在。

jo:你到底想说什么？dio

dio：おれは人间をやめるぞ！　ジョジョ——ッ！！

jump

1.1

PWM占空比精度

1：100

1：1000

占空比：高电平占整个周期的时间比值

PWM周期与PWM频率

边沿对齐方式

（PWMPH，PWMPL）周期设定寄存器

$$F_{pwm} = \frac{F_{clk}}{PWMDIV}$$

$$PWM频率 = \frac{F_{pwm}}{(PWMPH, PWMPL) + 1}$$

位置1

ADC

pwm

点击跳转

点击到位置1

click

click

函数指针数组

函数指针

函数多个参数

结构体做参数，结构体嵌套结构体

结构体成员是函数指针变量

结构体定义变量并赋值

函数指针函数定义

函数里包含多个参数，其中也有结构体变量

结构体变量包含多个成员，还嵌套了另一个结构体

宏定义来实现对指针变量的高低位读取

```
#define BUILD_UINT16(*loByte*, *hiByte*)  ((uint16)(((loByte) & 0x00FF) + (((hiByte) & 0x00FF) << 8)))

#define HI_UINT16(a) (((a) >> 8) & 0xFF)
#define LO_UINT16(a) ((a) & 0xFF)
```

在一个函数中，同过对参数的判断，然后来将全局数据（可以变量，数组，结构体等）的状态来进行操作改变。然后在其他地方通过函数来对这个数据执行对应的功能

```
/* sbpProfile_ota.c  */
CONST uint8 simpleProfileServUUID[ATT_BT_UUID_SIZE] =
{
 LO_UINT16(SIMPLEPROFILE_SERV_UUID), HI_UINT16(SIMPLEPROFILE_SERV_UUID)
};

// Simple Profile Service attribute
static CONST gattAttrType_t simpleProfileService = { ATT_BT_UUID_SIZE, simpleProfileServUUID };

static gattAttribute_t simpleProfileAttrTbl[] =
{
 // Simple Profile Service
 {
     { ATT_BT_UUID_SIZE, primaryServiceUUID }, /* type */
     GATT_PERMIT_READ,                         /* permissions */
     0,                                        /* handle */
     (uint8*)& simpleProfileService            /* pValue */
 },

 ...
};

static uint8 simpleProfile_ReadAttrCB( uint16 connHandle, gattAttribute_t* pAttr,uint8* pValue, uint16* pLen, uint16 offset, uint8 maxLen );

```

```
static bStatus_t simpleProfile_WriteAttrCB( uint16 connHandle,
gattAttribute_t* pAttr,uint8* pValue, uint16 len, uint16 offset );

// Simple Profile Service Callbacks
CONST gattServiceCBs_t simpleProfileCBs =
{
 simpleProfile_ReadAttrCB,  // Read callback function pointer
 simpleProfile_WriteAttrCB, // Write callback function pointer
 NULL                       // Authorization callback function pointer
};


status = GATTServApp_RegisterService(
simpleProfileAttrTbl,GATT_NUM_ATTRS( simpleProfileAttrTbl
),&simpleProfileCBs );
```

Operating System Obstruction Layer

任务注册，任务间互斥，中断管理，内存分配和管理，定时器功能

链表结构

任务

任务调度，时间管理，原语通信

低耦合

软件功能是由任务时间来形成的

1. 创建ID
2. 初始化进程
3. 任务处理时间
4. 消息处理服务

消息机制通信

消息即为数据

MOV MOVX MOVC ANL ORL XRL SETB CLR RR RL RRC RLC AND OR ADD SUBB MUL DIV DV INC DEC SJMP LJMP AJMP JMP ACALL ICALL RET RETI

JZ JNZ CJNE CJNZ NOP JC JB #data direct @Ri Rn

基准

《黑天鹅》 值得一读

inductive演绎，deductive归纳

演绎推理

每个推理都需要信息的帮助，这里我们称之为前提，又可以划分为大前提，小前提

> 前提1 下雨需要带伞
>
> 前提2 明天会下雨
>
> 结论：明天要带伞

这个推理的过程，被称之为演绎推理（Deductive reasoning）

定义：人们以一定反应客观规律的理论认识为依据，从服从该事物的已知部分，推理得到事物的未知部分的思维方法

客观规律为依据，从已知部分，推得未知部分

三段论

> 大前提：所有人都会死
>
> 小前提：苏格拉底是人
>
> 结论：苏格拉底会死

归纳推理

> 前提：A死了，B死了， C死了...我们观察到所有人，最后都会走向死亡
>
> 结论：是人总有一死

这个推理的过程就是归纳推理

定义：人们以一系列经验事物或知识素材为依据，寻找出其服从的基本规律或共同规律，并假设同类事物中的其他事物也服从这些规律

经验为依据，寻找基本规律。

归纳推理其实就是通过看到的经历过的各种事情，抽象出一些方法论出来，用以找到事物的规律从而预测事物未来的发展

区别：

思维的起点不同：归纳推理是从特殊到一般的认识过程；演绎推理是从一般到特殊性的认识过程

准确性不同；归纳推理的结论超出了前提所断定的范围，其结论并不一定能保证完全正确；

而演绎推理其结论不超出前提所断定的范围，一个演绎推理只要前提真实并且推理形式正确，那么，其结论就必然真实

联系：

演绎推理的大前提来自归纳推理概括和总结，从这个意义上来说，没有归纳推理也就没有演绎推理

归纳推理的每一个具体的案例观察结论，来自于演绎推理，从这个意义上来说没有演绎推理也就没有归纳推理

1. 归纳推理在概括事物的共性时，把事物的属性看作为某种既成的东西，静态的东西，它所概括的是事物的过去，难以概括它的发展和未来。
2. 它未必把握住事物的本质

确定演绎法隐藏的前提是否正确

从结论反推前提不正确

串行波特率

$$Baud = \frac{1}{t} * \frac{2^{SMOD}}{32}$$
$$16分频SMOD = 0$$
$$32分频SMOD = 1$$

t 是定时时长

$$t = (2^n - a) * \frac{12}{f_{osc}}$$

联立

$$Baud = \frac{f_{osc} * 2^{SMOD}}{(2^n - a) * 12 * 32}$$

物理地址 = 段地址*16 + 偏移地址　　（乘16相当于二进制左移4位，16进制左移1位）

物理地址 = 基础地址 + 偏移地址

CPU是16位结构，物理地址是20位结构

而物理地址的20位结构是由16位加上另一个16位来的

CPU访问内存空间存储，其实就是访问这些内存空间的物理地址，通过地址来找到具体的内存，然后操作它。

CPU有控制，地址，数据三条总线

本质含义：CPU访问内存空间地址，是通过一个基础地址（段地址*16）和一个相对于基础地址的偏移地址相加，然后形成的物理地址。

段地址：16位，最大访问64KB，

物理地址：20位，最大访问空间1MB

输入信息，大脑消化理解，判断好坏，形成观点

永远保持好奇心是永远进步的人

多问几个为什么

遇到事情，不急于定下结论，多问几个为什么。

锻炼自己的「批判性思维」，对每件事情持有「审视想法」，用合理的质疑态度，来判断事情的真伪与是非，有理有据地分析出真正原因。

只有这样全面去追问、思考、解构问题，才能减少偏见和认知盲区，扩展思考边界，不断逼近事情的本质，形成个人独立见解。

好奇引起兴趣，兴趣激发学习，学习产生思考

任何事物都有两面性，正向思维，逆向思维

思考已经发现的事情因果，从果溯因，也可从因到果

思想深刻的人，才能抓住问题的本质和困难的要害。

提升思想深度与宽度，是一个长期的过程，需要不间断的输入、思考和总结

他们既懂得如何驱动自己持续地努力和积累，也懂得借助社会和科技驱使放大自己努力的收益。

《学习之道》——乔希.维茨金

读完一本书，却不记得讲什么？

美剧看了一箩筐，英语还是没长进？

公众号、指导书、牛人帖，一个都没落，却还是没有形成自己的学习体系？

这些都是典型的低质量学习症状。

不了解学习的原理，就是在无效重复；缺乏有效的学习方法，就是在消耗天赋。

《谁动了我的奶酪》——斯宾塞.约翰逊

UART

通用异步收发器

Universal Asynchronous Receiver/Transmitter

TXD，

RXD

两根线，异步，没有时钟线，按数据帧格式，起始位，数据位，校验位，停止位

发送器

接收器

缓冲区

TI，RI

SPI

串行外围设备接口

Serial Peripheral Interface

串行通信，同步，全双工，主从机

MOSI

MISO

SCLK

CS

主机发送时钟信号

四种工作模式：SPI0，SPI1，SPI2，SPI3，由CPHA（时钟相位），CPOL（时钟极性）决定

CPOL = 0 , SCK 空闲电平为低，CPOL = 1，SCK空闲电平为高

CPHA = 0，在SCK第一个跳变沿采样，CPHA = 1，在SCK第二个跳变沿采样

- CPOL = 0
- CPHA = 0
  SPI0
- CPOL = 0
  CPHA = 1
  SPI1

- CPOL = 1
  CPHA = 0
  SPI2
- CPOL = 1
  CPHA = 1
  SPI3

IIC

内部集成电路总线

Inter-Integrated Circuit Bus

同步，串行，两线，主从机

SCL

SDA

总体特征

SDA 和SCL 都是双向线路，都通过一个电流源或上拉电阻连接到正的电源电压（见图3）当总线空闲时，这两条线路都是高电平。连接到总线的器件输出级必须是漏极开路或集电极开路才能执行线与的功能

标准模式：100Kbit/s

快速模式：400Kbit/s

高速模式：3.4Mbit/s

数据的有效性

SDA 线上的数据必须在时钟的高电平周期保持稳定。数据线的高或低电平状态只有在SCL线的时钟信号是低电平时才能改变

起始信号

SCL 线是高电平时，SDA 线从高电平向低电平切换表示**起始条件**

停止信号

当SCL 是高电平时，SDA 线由低电平向高电平切换表示**停止条件**

高位先发（MSB）

数据传输必须带响应

堆栈操作

设置堆栈，进栈和出栈三种

①

```
    ...
STACK1 SEGMENT PARA STACK
    DW 100 DUP(0)
STACK1 ENDS
    ...
```

②

```
PUSH    AX          ;将AX内容压入栈
PUSH    DS          ;将DS内容压入栈
PUSH    DATA_WORD   ;将存储单元DATA_WORD字数据压入栈
PUSHF               ;将标志寄存器内容压入栈
```

③

```
POP     AX          ;将堆栈顶一个字内容弹出给AX
POP     DS          ;将堆栈顶一个字内容弹出给DS
POP     DATA_WORD   ;将堆栈顶一个字内容弹出给存储单元DATA_WORD
POPF                ;将堆栈顶一个字内容弹给标志寄存器
```

进栈出栈均在栈顶进行，严格执行FILO原则

进栈操作过程：

```
SP-2->SP          ;堆栈指针减2修改
数据->SP          ;数据压入SP所指向的字单元中
```

出栈的操作过程：

```
SP->寄存器/存储器 ;SP所指的字数据送入寄存器
SP+2->SP          ;堆栈指针加2修改
```

数据寻址方式

汇编语言的指令格式分两部分：操作码和操作数。

操作码是指令完成的具体操作

操作数是指令操作的对象数据

寻找指令中所需操作数的方法称为数据寻址方式

80x86指令的操作数主要有3类：

- 立即数（操作数在指令代码中）
- 寄存器操作数（操作数在寄存器中）
- 存储器操作数（操作数在内存中）

对应有数据寻址方式有3类：

- 立即数寻址
- 寄存器寻址
- 存储器寻址

立即数寻址（Immediate Addressing）

立即数可以是8位，16位，32位

```
MOV AH,8
MOV AX,1234H
MOV EAX,0ABCD1234H
```

立即数寻址方式常用于给寄存器赋初值，直接操作不占用总线周期，执行速度快

立即数只能作为源操作数，不能作目的操作数

源操作数长度必须与目的操作数一致

寄存器寻址（Register Addressing）

操作数在指令指定的寄存器中，存取操作完全在CPU内部进行，不占用总线周期，执行速度快

```
MOV AX,DX
ADD BL,CH
SUB SI,0
```

两操作数的长度必须一致

约定：()代表内容，括号内如果是寄存器，也可以不加括号

[]代表地址

[()]代表地址中的内容

([])代表内容作为地址

存储器寻址（Memory Addressing）

寄存器寻址速度快，但数目有限。大多数情况下，操作数还是存放在内存中

多种内存寻址方式

1. 直接寻址（Direct Addressing）
   最简单的一种存储器寻址方式，EA直接由位移量给出。即物理地址=DSX16+EA（偏移量）
2. 寄存器间接寻址（Register Indirect Addressing）
3. 变址寻址/寄存器相对寻址（Indexed Addressing/Register Relative Addressing）
4. 基址变址寻址（Based Indexed Addressing）
5. 相对基址变址寻址（Relative Based Indexed Addressing）
6.

```
自上而下金字塔结构
1.提出主题思想
2.设想受众的主要疑问
想清楚要解决谁的什么问题。
3.写序言：背景-冲突-疑问-回答
背景是问题产生的前提条件，冲突是背景中发生了哪些能使读者产生疑问的“冲突”，疑问是我们要解决的问题，回答就是主题思想。
4.与受众进行疑问、回答式对话
疑问、回答式对话，就是自上而下的金字塔结构。从上一层思想到下一层思想。
5.对受众的新疑问，重复进行疑问/回答式对话
思想慢慢开展，层级慢慢丰富。

```

自下而上法

```
自下而上金字塔结构
1.列出你想表达的所有思想要点
2.找出各要点之间的逻辑关系
3.得出结论
```

思考的逻辑

1. 时间顺序
2. 结构顺序
3. 程度顺序
4. 概括各组思想

解决问题的逻辑

1. 界定问题
2. 结构化分析问题
3. 分析/找到解决方案
4. 组成金字塔与他人交流

```
解决问题的流程
1.是否有/是否可能有问题（或机会）？
判断是否有问题，需要对比期望结果和非期望结果。一般来说，如果期望结果低于非期望结果，则说明存在问题。
2.问题在哪里？
寻找问题的位置，需要从背景入手。
3.为什么存在？
寻找问题的原因，寻找流程上或者结构中潜在的问题。
4.我们能做什么？
其实就是根据问题，罗列可能的解决方案。
5.我们应该做什么？
根据所有罗列的解决方案，选择一个最适合的方案去执行。
```

首先我们要学会界定问题，界定问题是解决问题的第一步，界定问题不是找到具体的问题，而是规定问题所在的集合范围。界定问题可以采用连续分析法。

判断问题是否存在，通常要看经过努力得到的结果（现状），与希望得到的结果（目标）之间是否有差距。

其次是结构化分析问题

收集信息，需要在锁定的问题所涉及的背景范围内，进行收集。

描述发现，其实是尽可能找出产生问题的原因。找出原因有三种方法：呈现有形结构、寻找因果关系、归类分组。

呈现有形结构：需要画一幅系统的现况或理想状况的图，帮助决定是否要回答这些问题，并找到和分析产生问题的原因。

寻找因果关系：寻找具有因果关系的要素、行为或任务，得出结果。

归类分组：将所有可能的原因按相似性分类，前提是这种预先的分类有助于综合分析各种事实。

归类分组中的要素，需要遵循相互独立，完全穷尽的原则。

这件事存在问题吗？

如果存在，存在哪些问题？

为什么会出现这些问题？

这些问题可以被解决吗？

如果可以应该怎么做？如果不可以又该怎么办？

十六进制转十进制

```
0x80  = 16^1 * 8 + 16^0 * 0 = 128;
0x808 = 16^2 * 8 + 16^1 * 0 + 16^0 * 8 = 2056;
0xaba = 16^2 * a + 16^1 * b + 16^0 * a = ...;
...
第N位的的权值(16^N)乘于该N位上的数X(X=[0,15])，即
```

$$X * 16^N$$

两大要素：场景和形象

三大过程：创建，编码和连接

机器周期 = 12 * 晶振周期

定时器 / 计数器：一个机器周期才能产生一个计数（计数脉冲），计数器才加1.

1. 对常识的理解和把握

2. 切换视角的能力
3. 敢于承受孤独的心志
4. 对生活的感悟

拆分，重组，层次化，结构化，无穷无限

- 防止一个头文件重复包含

```
#ifndef COMDEF_H
#define COMDEF_H


#endif
```

- 重定义一些类型，防止由于各种平台和编译器的不同，而产生的类型字节数差异，方便移植

```
typedef unsigned long int uint32;
```

- 得到指定地址上的一个字节或字

```
#define MEM_B(x)     (*((byte *)(x)))
#define MEM_W(x)     (*((word *)(x)))
```

- 求最大值和最小值

```
#define MAX(x,y)     (((x) > (y))?(x):(y))
#define MIN(x,y)     (((x) < (y))?(x):(y))
```

- 得到一个field在结构体（struct）中的偏移量

```
#define FPOS(type, field)   ((dword) & ((type *)0)->field)
```

- 得到一个结构体所占用的字节数

```
#define FSIZ(type, field)   sizeof(((type *)0)->field)
```

- 按照LSB格式把两个字转化为一个word

```
#define FLIPW(ray) ((((word)(ray)[0]) * 256) + (ray)[1])
```

- 按照LSB格式把一个word转化为两个字节

```
#define FLOPW(ray,val1)  \
    {   \
        (ray)[0] = ((val) / 256); \
        (ray)[1] = ((val) & 0xFF) \
    }
```

- 得到一个变量的地址（word宽度）

```
#define B_PTR(var)  ((byte *)(void *)&(var))
#define W_PTR(var)  ((word *)(void *)&(var))
```

- 得到一个字的高位和低位字节

```
#define WORD_LO(xxx)    ((byte)((word)(var) & 255))
#define WORD_HI(xxx)    ((byte)((word)(var) >> 8))
```

学习不是为了找到答案，而是为了找到方法

合理的学习安排

成果具象化

保持心态平和

走出舒适区，尽量去做难题

长时间保持专注（todolist +番茄工作法）

学习趁早，把握当下

时钟周期，机器周期与指令周期

时钟周期

CPU从存储器取出并执行一条指令所需的全部时间称之为指令周期

CPU执行一条指令的过程

分解：

```
1. Instruction Fetch（取指令）
2. Instruction Decode（译码）
3. Execute（执行）
```

演绎：

- 阐述世界上已存在的一些情况。
- 阐述世界上存在的相关情况，如果第二个表述是针对第一个表述的主语或谓语，则说明这两个表述相关
- 说明两种情况同时存在的意义

segmentation and reassembly 分割和重组

BR

EDR

颜料三原色：红黄蓝（RYB）

光学三原色：红绿蓝（RGB）

间色：橙（红+黄）绿（黄+蓝）紫（蓝+红）

补色：橙和蓝，绿和红，紫和黄互为补色

```
/* 结构体定义
 * struct multiScan: 结构体名
 * GAPMultiRolScanner_t：结构体别名
 * 结构体成员：rssi, addrtype, advDatalen, advData(指针), scanRsplen
 *           rspData(指针), next(该结构体指针，指向下一个结构体)
*/
typedef struct multiScan
{
    uint8 rssi;
    uint8 addrtype;
    uint8 advDatalen;
    uint8 *advData;
    uint8 scanRsplen;
    uint8 *rspData;
    struct multiScan *next;
}GAPMultiRolScanner_t;


#define B_ADDR_LEN      6

/*
 * 函数指针，typedef定义函数指针别名，用来定义结构体成员
*/
```

```
typedef void (*gapRolesEachScan_t)(gapDeviceInfoEvent_t * pPkt);
typedef void (*gapRolesScanDone_t)(GAPMultiRolScanner_t * pPkt);
typedef void (*gapRolesEstablish_t)(uint8 status, uint16 connHandle,
GAPMultiRole_State_t role, uint8 perIdx, uint8 * addr);
typedef void (*gapRolesTerminate_t)(uint16 connHandle, GAPMultiRole_State_t
role, uint8 perIdx, uint8 reason);
...

typedef struct
{
    osal_evnet_hdr_t hdr;
    uint8 opcode;
    uint8 eventType;
    uint8 addrType;
    uint8 addr[B_ADDR_LEN];
    int8 rssi;
    uint8 dataLen;
    uint8 * pEvtData;
}gapDeviceInfoEvent_t;

typedef struct
{
    uint8 event;
    uint8 status;
}osal_evnet_hdr_t;

typedef struct
{
    gapRolesEachScan_t  pfnEachScan;
    gapRolesScanDone_t  pfnScanDone;
    gapRolesSDPNotify_t SDPNotify;
    gapRolesDataNotify_t    dataNotify;
    gapRolesEstablish_t pfnEstablish;
    gapRolesTerminate_t pfnTerminate;
}gapMultiRolesCBs_t;

```

```
typedef enum
{
    advertiser = 1,
    scanner,
    initiator
}GAPMultiRole_type;

typedef struct multiList
{
    GAPMultiRole_type role;
    uint8 busy;
    union
    {
        struct
        {
            uint8 perIdx;
            uint8 DatConfUpd;
            GAPMultiRole_states_t state;
```

```
        }adv;
        struct
        {
            uint8 scanning;
        }scan;
        struct
        {
            uint8 initiating;
        }initiate;
    }roleScd;
    uint32 nextScdTime;
    struct multiList* next;
}multiScehdule_t;
```

苏格拉底：“承认自己的无知乃是开启智慧的大门”

欠压：**不够电压”。 对所有的电器设备而言，都有一个额定电压，但在实际中，不能完全保证在额定电压下工作，是在额定电压附近的一个范围，一般要求在±15%。 为了保护电器设备和工艺质量，如果低于 - 15%这个电压，就是“欠压”，当工作电压下降到这个电压以下，保护动作，切断电源。 相反，如果高于+15%这个电压，就是“过压”，保护也动作切断电源**

SW6007 集成了最高效率高达 96%的开关充电模块，其支持 4.2V/4.35V/4.4V/4.5V 等多种电池类型，
开关频率 500KHz，可以使用小体积的 1uH 电感

充电流程分为如下三个过程：涓流模式、恒流模式、恒压模式。当电池电压低于 3V 时，充电模块处于涓流模式，充电电流为涓流充电电流；当电池电压大于 3V 时，充电模块进入恒流模式，此时按照设定的目标电流全速充电；当电池电压上升到充电目标电压（比如 4.2V）时，充电模块进入恒压模式，此时电流逐渐减小，而电池端电压保持不变；当充电电流减小到充电截止电流，充电结束。充满后如果电池电压降低到比目标电压低 0.1V，则自动重新开始充电。

> 系统属性命令：sysdm.cpl

1度=1千瓦.时(1000瓦 x 1小时)

$$用电量（度）=功率（千瓦）x使用小时小时）$$

其中瓦特是功率的单位。如果在功率上再乘以一个时间，那么这个结果就是功。功的单位有焦耳和千瓦时，它们之间的关系如下

1焦=1瓦秒

1千瓦时=1千瓦小时=1000瓦小时=(1000*3600)瓦秒=3600000焦

即：1J=1Ws

$$1kWh = 1kWh = 1000Wh = 1000W * 3600s = 3600000J$$

因此度和千瓦时和焦耳之间的关系为：1度=1千瓦·时=3600000J。

即1千瓦时等于3600000焦耳

导体中的电子形成定向移动而产生电流

电荷的定向移动产生电流

电流方向与负电荷移动方向相反

发电机，电池本身并不带电，它的两极分别有正负电荷，由正负电荷产生电压（电流是电荷[带电粒子或者电子]在电压的作用下定向移动而形成的）

电源内部电流方向与外部电流方向恰好相反

电源外部，电流方向是从电源正极流向负极，即势能高的一级流向势能低的一级。

而在电源内部，是有一种动能（比如电池内部是化学能）将阳离子从负极输送到正极，所以电源内部电流方向是从负极到正极的

阴离子：得电子 带负电

阳离子：失电子 带正电

阴极：得电子的极，也就是发生还原反应的极

阳极：失电子的极，使电解质发生氧化反应的电极

最大充满电压(4.35V)

电流是电池容量1C(1* 800mAh)

P 空穴（可移动）和电离杂质——离子（固定）

N 电子（可移动）和正离子（固定）

空穴和电子结合，

空间电荷区（固定离子形成），P区（负离子），N区（正离子）

数据元素随机存储，并通过指针表示数据之间逻辑关系的存储结构就是链式存储结构。

```
git init                                                    # 初始化本地git仓库（创建新仓库）
git config --global user.name "xxx"                         # 配置用户名
git config --global user.email "xxx@xxx.com"                # 配置邮件
git config --global color.ui true                           # git status等命令自动着色
git config --global color.status auto
git config --global color.diff auto
```

```
git config --global color.branch auto
git config --global color.interactive auto
git config --global --unset http.proxy                          # remove  proxy configuration on git
git clone git+ssh://git@192.168.53.168/VT.git                   # clone远程仓库
git status                                                      # 查看当前版本状态（是否修改）
git add xyz                                                     # 添加xyz文件至index
git add .                                                       # 增加当前子目录下所有更改过的文件至index
git commit -m 'xxx'                                             # 提交
git commit --amend -m 'xxx'                                     # 合并上一次提交（用于反复修改）
git commit -am 'xxx'                                            # 将add和commit合为一步
git rm xxx                                                      # 删除index中的文件
git rm -r *                                                     # 递归删除
git log                                                         # 显示提交日志
git log -1                                                      # 显示1行日志 -n为n行
git log -5
git log --stat                                                  # 显示提交日志及相关变动文件
git log -p -m
git show dfb02e6e4f2f7b573337763e5c0013802e392818               # 显示某个提交的详细内容
git show dfb02                                                  # 可只用commitid的前几位
git show HEAD                                                   # 显示HEAD提交日志
git show HEAD^                                                  # 显示HEAD的父（上一个版本）的提交日志 ^^为上两个版本 ^5为上5个版本
git tag                                                         # 显示已存在的tag
git tag -a v2.0 -m 'xxx'                                        # 增加v2.0的tag
git show v2.0                                                   # 显示v2.0的日志及详细内容
git log v2.0                                                    # 显示v2.0的日志
git diff                                                        # 显示所有未添加至index的变更
git diff --cached                                               # 显示所有已添加index但还未commit的变更
git diff HEAD^                                                  # 比较与上一个版本的差异
git diff HEAD -- ./lib                                          # 比较与HEAD版本lib目录的差异
git diff origin/master..master                                  # 比较远程分支master上有本地分支master上没有的
git diff origin/master..master --stat                           # 只显示差异的文件，不显示具体内容
git remote add origin git+ssh://git@192.168.53.168/VT.git # 增加远程定义（用于push/pull/fetch）
git branch                                                      # 显示本地分支
git branch --contains 50089                                     # 显示包含提交50089的分支
git branch -a                                                   # 显示所有分支
git branch -r                                                   # 显示所有原创分支
```

```
git branch --merged                                          # 显示所有已合并到当前分支的分支
git branch --no-merged                                       # 显示所有未合并到当前分支的分支
git branch -m master master_copy                             # 本地分支改名
git checkout -b master_copy                                  # 从当前分支创建新分支master_copy并检出
git checkout -b master master_copy                           # 上面的完整版
git checkout features/performance                            # 检出已存在的features/performance分支
git checkout --track hotfixes/BJVEP933                       # 检出远程分支hotfixes/BJVEP933并创建本地跟踪分支
git checkout v2.0                                            # 检出版本v2.0
git checkout -b devel origin/develop                         # 从远程分支develop创建新本地分支devel并检出
git checkout -- README                                       # 检出head版本的README文件（可用于修改错误回退）
git merge origin/master                                      # 合并远程master分支至当前分支
git cherry-pick ff44785404a8e                                # 合并提交ff44785404a8e的修改
git push origin master                                       # 将当前分支push到远程master分支
git push origin :hotfixes/BJVEP933                           # 删除远程仓库的hotfixes/BJVEP933分支
git push --tags                                              # 把所有tag推送到远程仓库
git fetch                                                    # 获取所有远程分支（不更新本地分支，另需merge）
git fetch --prune                                            # 获取所有原创分支并清除服务器上已删掉的分支
git pull origin master                                       # 获取远程分支master并merge到当前分支
git mv README README2                                        # 重命名文件README为README2
git reset --hard HEAD                                        # 将当前版本重置为HEAD（通常用于merge失败回退）
git rebase
git branch -d hotfixes/BJVEP933                              # 删除分支hotfixes/BJVEP933（本分支修改已合并到其他分支）
git branch -D hotfixes/BJVEP933                              # 强制删除分支hotfixes/BJVEP933
git ls-files                                                 # 列出git index包含的文件
git show-branch                                              # 图示当前分支历史
git show-branch --all                                        # 图示所有分支历史
git whatchanged                                              # 显示提交历史对应的文件修改
git revert dfb02e6e4f2f7b573337763e5c0013802e392818           # 撤销提交dfb02e6e4f2f7b573337763e5c0013802e392818
git ls-tree HEAD                                             # 内部命令：显示某个git对象
git rev-parse v2.0                                           # 内部命令：显示某个ref对于的SHA1 HASH
```

```
git reflog                                          # 显示所有提交，包括孤立节点
git show HEAD@{5}
git show master@{yesterday}                         # 显示master分支昨天的状态
git log --pretty=format:'%h %s' --graph             # 图示提交日志
git show HEAD~3
git show -s --pretty=raw 2be7fcb476
git stash                                           # 暂存当前修改，将所有至为HEAD状态
git stash list                                      # 查看所有暂存
git stash show -p stash@{0}                         # 参考第一次暂存
git stash apply stash@{0}                           # 应用第一次暂存
git grep "delete from"                              # 文件中搜索文本“delete from”
git grep -e '#define' --and -e SORT_DIRENT
git gc
git fsck
```

```
第一步 克隆代码，
git clone 仓库地址

第二步 首次提交，将项目代码传到你的仓库
git push 你的仓库地址

第三步 切换远程仓库地址
#方式一
$ git remote rm origin
$ git remote add 你的仓库地址

#方式二
$ git remote set-url origin  你的仓库地址

第四步
初次更新提交
git add . （提交你的修改）
git commit -m "初次修改提交"
git push --set-upstream origin master(重新关联分支)

```

https://blog.csdn.net/m0_45234510/article/details/120181503

IIC

集成内部互联总线。主要用于同一电路板内各集成电路模块的连接

双向两线制串行数据传输方式

PHILIPS

20世纪80年代

标准模式

快速模式

高速模式

主机

从机

发送器

接收器

地址

SDA与SCL

IIC总线数据通信协议

串行方式传输数据

从数据字节的最高位开始传送，每个数据位在SCL上都有一个时钟脉冲相对应

在一个时钟周期内，在SCL高电平期间，SDA上必须保持稳定的逻辑电平状态，只有SCL低电平期间，才允许SDA上电平状态变化

`multiRoleApp_Init()`

- 广播参数初始化`multiRoleApp_AdvInit()
    - `GAPMultiRole_SetParameter()`
    - `GAP_SetParamValue()`
    - `multiBle_adv_name_set()`
        - `multiSchedule_advParam_init()`
        - `multiSchedule_advParam_init()`
    - `GGS_SetParameter()`
- `ppsp_serv_add_serv()`
- `MultiProfile_RegisterAppCBs()`
- 扫描参数初始化`multiRoleAPP_ScIn_Init()`
    - `GAP_SetParamValue()`
    - `GAPMultiRole_SetParameter()`
    - `ble_conn_addr_add()`

> 函数作为参数，选择执行普通模式还是ota模式

祖母绿：被称为绿宝石之王，国际公认的四大名贵宝石之一（红蓝绿宝石以及钻石），在古埃及时代就已经用作珠宝；祖母绿象征仁慈，信心，善良，永恒，幸运和幸福，佩戴它会带来一生的平安

乌兰孖努：一种特产于呼伦贝尔草原的红宝石