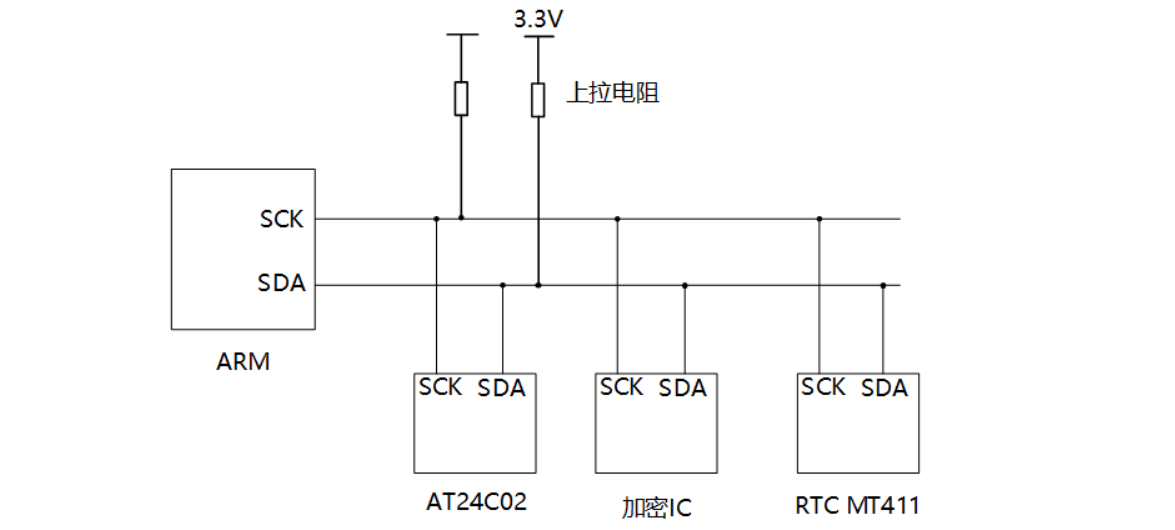
# I2C通信协议

## 物理组成

I2C协议在硬件接线组成上是由两根线SCL时钟线和SDA信号线组成，如图，在总线上可以挂很多设备，任意设备都可以作为主设备，但是同一时刻只有一个主设备。总线上需要接上拉电阻，是因为空闲时SCL和SDA要保证高电平，直接接电源有风险，因此需要接上拉电阻。



## 传输协议

I2C协议中数据传输的单位是字节，也就是8位。但是要用到9个时钟：前面8个时钟用来传输8数据，第9个时钟用来传输回应信号。传输时，先传输最高位(MSB)。

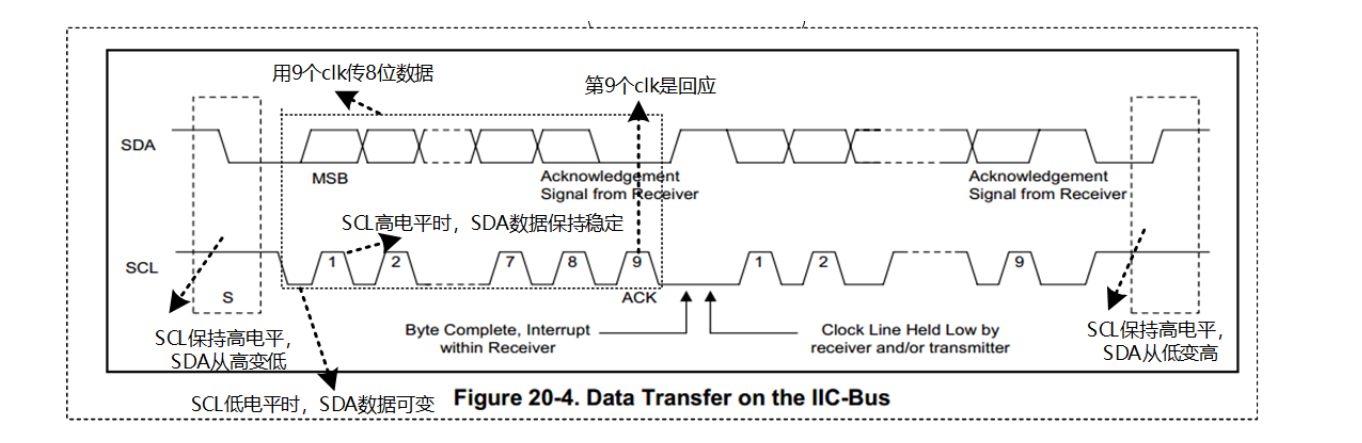
开始信号（S）：SCL为高电平时，SDA山高电平向低电平跳变，开始传送数据。

结束信号（P）：SCL为高电平时，SDA由低电平向高电平跳变，结束传送数据。

响应信号(ACK)：接收器在接收到8位数据后，在第9个时钟周期，拉低SDA

SDA上传输的数据必须在SCL为高电平期间保持稳定，SDA上的数据只能在SCL为低电平期间变化，也就是SDA在低电平期间准备好数据（拉高（0b1）或拉低（0b2））

I2C协议信号如下：



## 信号格式

### 写操作

流程如下：

主芯片要发出一个start信号

然后发出一个设备地址(用来确定是往哪一个芯片写数据)，方向(读/写，0表示写，1表示读)

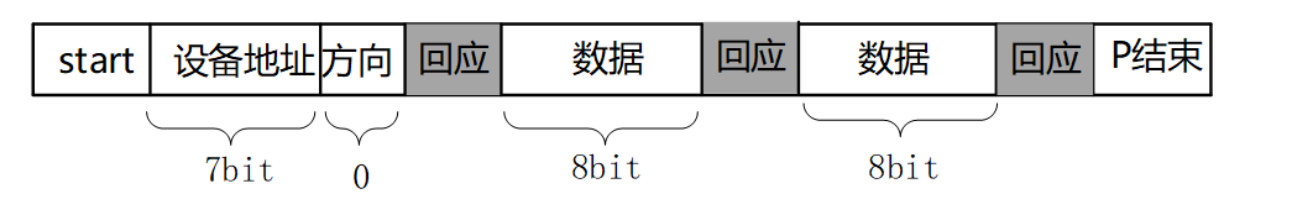
从设备回应(用来确定这个设备是否存在)，然后就可以传输数据

主设备发送一个字节数据给从设备，并等待回应

每传输一字节数据，接收方要有一个回应信号（确定数据是否接受完成)，然后再传输下一个数据。

数据发送完之后，主芯片就会发送一个停止信号。

下图：白色背景表示"主→从"，灰色背景表示"从→主"



### 读操作

流程如下：

主芯片要发出一个start信号

然后发出一个设备地址(用来确定是往哪一个芯片写数据)，方向(读/写，0表示写，1表示读)

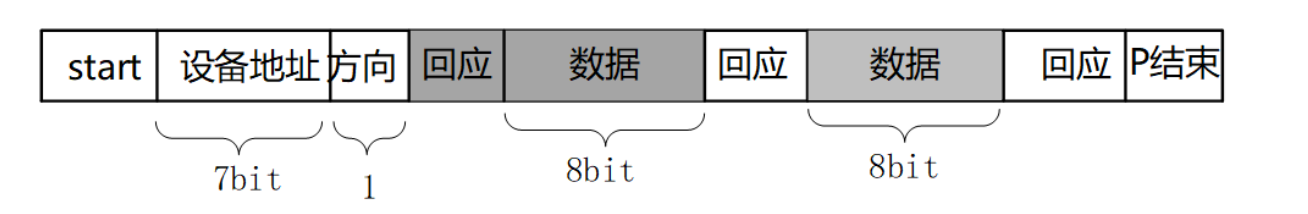
从设备回应(用来确定这个设备是否存在)，然后就可以传输数据

从设备发送一个字节数据给主设备，并等待回应

每传输一字节数据，接收方要有一个回应信号（确定数据是否接受完成)，然后再传输下一个数据。

数据发送完之后，主芯片就会发送一个停止信号。

下图：白色背景表示"主→从"，灰色背景表示"从→主"



### I2C传输细节

\*如何在SDA上实现双向传输？

主芯片通过一根SDA线既可以把数据发给从设备，也可以从SDA上读取数据，连接SDA线的引脚里面必然有两个引脚（发送引脚/接受引脚）。

主、从设备都可以通过SDA发送数据，肯定不能同时发送数据，怎么错开时间？

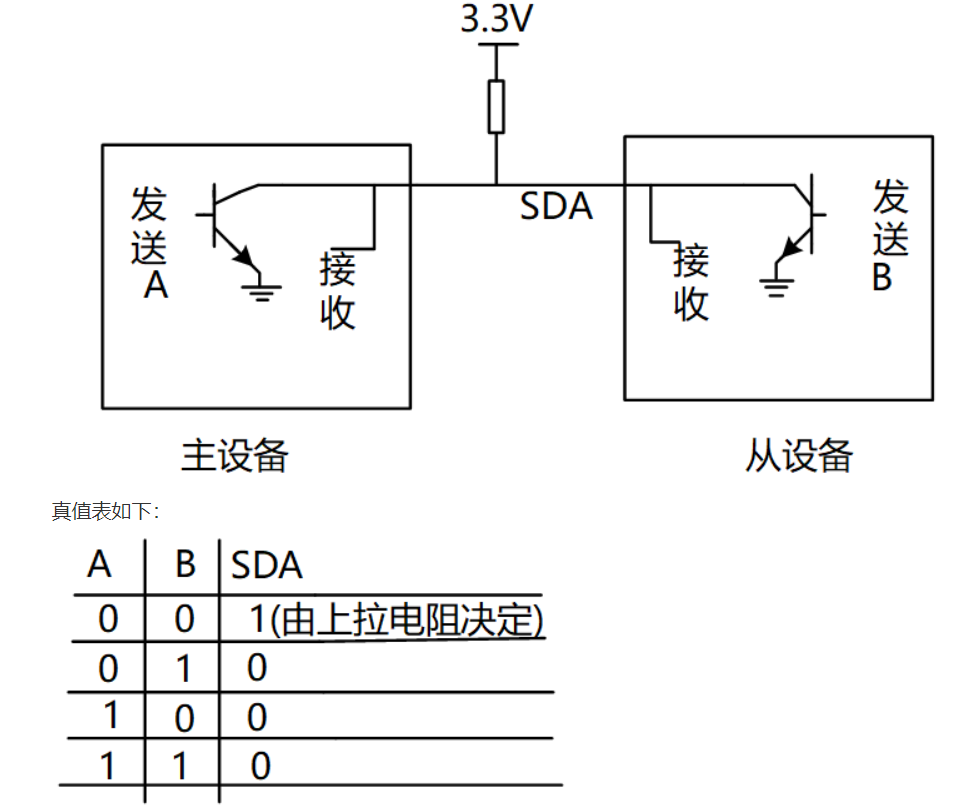
在9个时钟里，

前8个时钟由主设备发送数据的话，第9个时钟就由从设备发送数据；

前8个时钟由从设备发送数据的话，第9个时钟就由主设备发送数据。

双方设备中，某个设备发送数据时，另一方怎样才能不影响SDA上的数据？

设备的SDA中有一个三极管，使用开极/开漏电路(三极管是开极，CMOS管是开漏，作用一样)，如下图：



从I2C的硬件可以看出，所谓的主机释放总线就是不驱动总线，如果这个时候从机回应，驱动总线，那么SDA就为高电平，主机通过读取引脚的电平，即可获得回应信号。因此上拉电阻的作用既是维持了总线空闲时的高电平，同时避免了总线冲突（主从都驱动SDA时）

# 软件模拟I2C

## GPIO引脚设置

使用GPIO模拟I2C协议就是，就是通过两个GPIO引脚来模拟SCL和SDA两根线，通过程序控制这两个引脚的按照I2C的时序协议动作，既可以视为I2C的通讯协议。

GPIO模拟I2C时，首先输出模式需要设置为开漏输出模式，开漏输出模式下，只能输出低电平状态，是否输出高电平，由外部电路决定，由上节可知道，外部电路上接上拉电阻，便可控制输出高电平。关于STM32的输入输出模式，可以参考：https://blog.csdn.net/helaisun/article/details/90738803

## 软件模拟时序

### 起始信号

起始信号是SCL高电平时，SDA被由高拉低，程序编写可以为：

/\*

\*I2c开始信号：SCL为高电平时，SDA由高拉低

\*/

void I2c\_Start(void)

{

SCL\_H;

SDA\_H;

I2c\_Delay(100);

SDA\_L;

I2c\_Delay(100);

}

### 结束信号

结束信号为，SCL为高时，SDA由低被拉高

/\*

\*I2c停止信号：SCL为高电平时，SDA由低拉高

\*/

void I2c\_Stop(void)

{

SDA\_L;

SCL\_H;

I2c\_Delay(100);

SDA\_H;

I2c\_Delay(100);

}

### 主机获取应答信号

主机获取应答信号就是在传输完一个字节后，第9个时钟周期，主机释放对SDA的控制权，因为有上拉电阻的存在，SDA为高电平，SCL会在第9个周期将SDA拉低，如果主机引脚读到了SDA为低电平，则视为获取了应答信号。应答信号成功获取的标志表示从机成功收到数据，主机可进行下一字节数据的传输。

/\*

\*I2C获取应答信号：在传输完一个字节后，接收应答信号

\*传输完一个字节后，将SDA拉高，然后通过检测SDA 是否被拉低

\*来确认是否有应答

\*输入参数：无

\*输出参数：无

\*返回值：成功收到应答：0 未应答：-1

\*/

int GetAck(void)

{

SCL\_L;

SDA\_H;

I2c\_Delay(100);

SCL\_H;

int i=0;

while(SDA\_IN!=0)

{

i++;

if(i==100)

{

SCL\_L;

return -1;

}

I2c\_Delay(10);

}

SCL\_L;

return 0;

}

### 发出应答信号

如果主机是读数据时，那么在接收到一个字节的数据后，同样需要给从机一个应答信号。

/\*

\*发出应答信号: 主机读数据时，一个字节结束后，主机将SDA由高拉低，发出应答

\*输入：无

\*输出：无

\*返回值：无

\*/

void I2c\_Ack(void)

{

SCL\_L;

SDA\_H; //SCL为低电平时，将SDA准备为高电平

I2c\_Delay(100);

SCL\_H; //开启SCL，从机可读SDA

SDA\_L; //从机读到SDA为低电平，即获得应答

}

### 写一个字节

void I2c\_WriteByte(uint8\_t date)

{

int i=0;

for(i=0;i<8;i++)

{

SCL\_L; //拉低时钟，准备数据

I2c\_Delay(100);

if(0x80&date)

{

SDA\_H;

}

else

{

SDA\_L;

}

date<<=1;

SCL\_H; //数据准备好了，拉高时钟，传输数据

I2c\_Delay(100);

}

GetAck();

}

以上是通过I2C写一个字节的程序，I2C传输是先传MSB（最高有效位），因此软件模拟时序发送一个字节的逻辑是：

首先将SCL拉低，然后依次读取发送数据date的最高有效位，并在SDA上准备好数据，相应的bit为1，则SDA拉高；为0，则SDA拉低，准备好之后，将SCL拉高，等待从机读取，然后再重复以上过程，直至一个字节发送完成。

一个字节发送完成后，主机获取从机应答。

### 读一个字节

void I2c\_ReadByte(uint8\_t \*pdate)

{

int i=0;

uint8\_t tmp=0;

for(i=0;i<8;i++)

{

SCL\_L;

I2c\_Delay(100); //SCL为低电平时，SDA准备数据

SCL\_H;

I2c\_Delay(100); //SCL为高电平时，SDA读取数据

if(SDA\_IN)

{

tmp++;

}

else

{

tmp=tmp;

}

tmp<<=1;

}

//接收完成一个Byte，发出应答信号

I2c\_Ack();

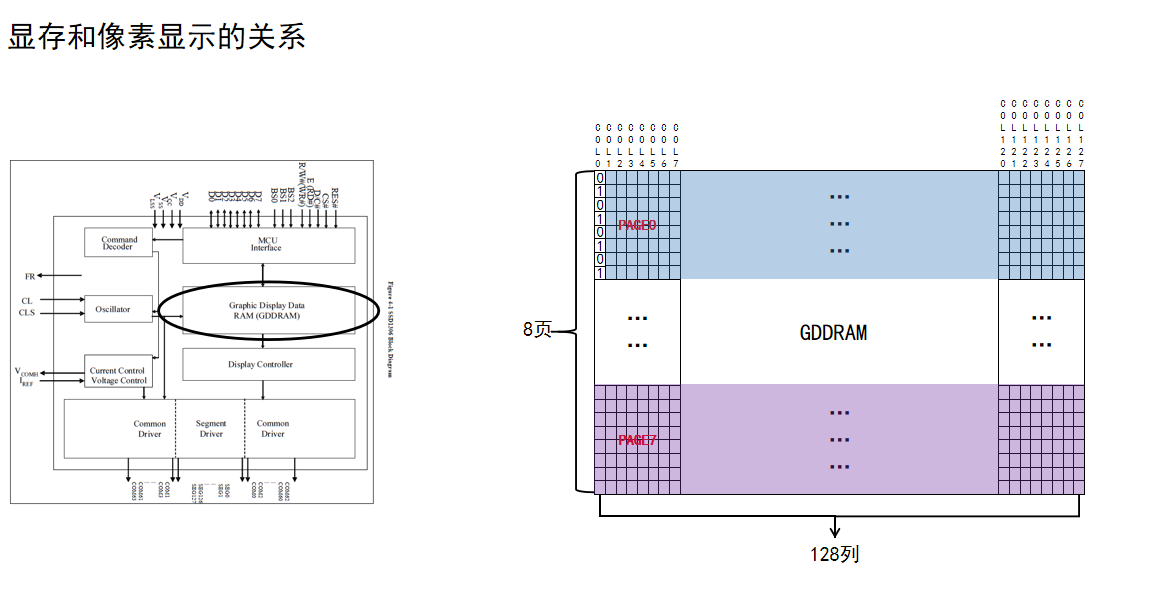
\*pdate=tmp;

}

以上是主机读取从机一个字节程序，与写操作相反，读取数据是在SCL为高时，读取SDA上的电平信号，然后根据电平信号（0b1或0b1），将其写入接收数据的数据结构里，注意I2C接收的也是从MSB到LSB，因此接收时，每接收一个bit，需要左移1位。

# OLED

## 显示概述

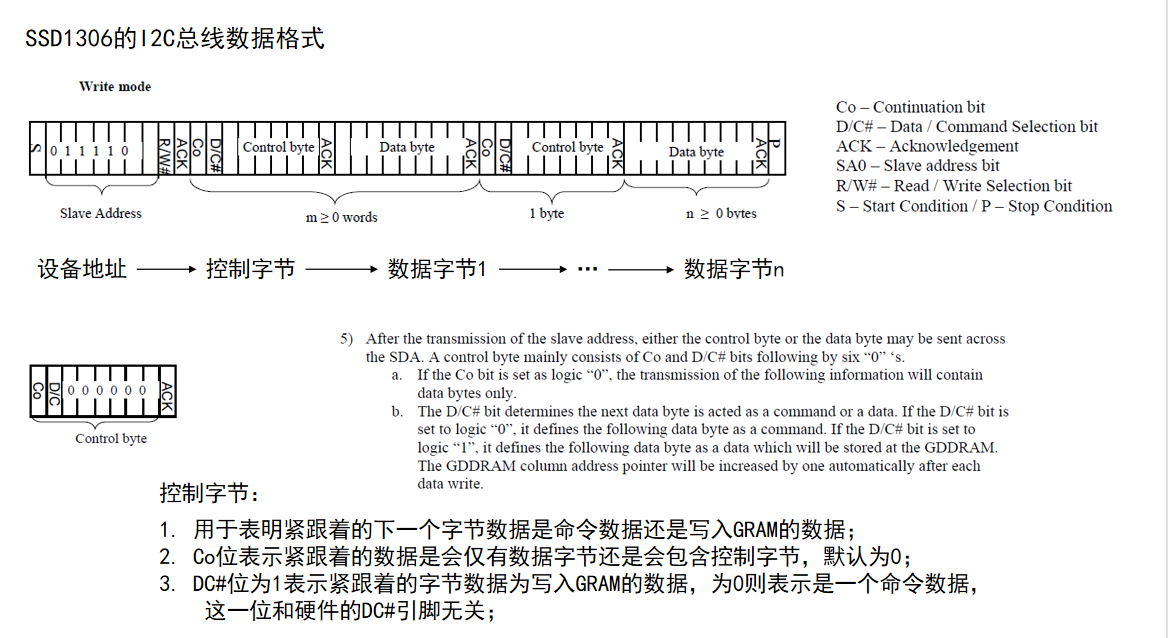


如图，SSD1306的显存是64\*128的，也就是这块显示屏的分辨率为64\*128，可以理解为64行，128列，其中64行被分成8页（page），也就是说每页的大小为8\*128，每页的一列共8个bit，可以存放一个字节数据，共128列。

因此像素显示就是将显存的数据以2进制的形式读出来，一个bit对应一个led灯，为1时，led灯亮，为0时灭。显存和像素的映射关系可以通过程序命令来设置，如可以将显存的page1的第一列数据显示到屏幕的任意页和任意行。

## OLED命令

OLED的许多设置，如设置起始页、寻址模式等也是通过I2C传输控制数据来进行设置的。



如图时SSD1306的I2C总线数据格式，I2C的通信协议只是约定了数据是如何传输的，至于与设备如何建立通信关系，需要根绝设备的控制器和数据格式来定。

从上图可以看出，与SSD1306的数据格式，首先一个字节，包括设备地址(7bit)和读写操作位（写0；读1,因此，向SSD1306写操作的地址为：0x78，读操作的地址为：0x79）然后紧跟着是一个控制字节，这个控制字节是用来告诉SSD1306后面的传输的信息是命令还是数据；所谓命令就是SSD1306接收后，通过控制器用来设置配置SSD1306的显示的，所谓数据，就是写入显存，并将数据显示出来的。

控制字节通过D/C#（bit6）来控制，如果D/C#=0,控制字节为：0x00，表示后续信息是命令；如果D/C#=1，控制字节为：0x40,表示后续信息是数据。

### OLED写命令

/\*

\*描述：通过I2C给OLED写命令

\*参数：cmd

\*返回值：无

\*/

void OLED\_WriteCmd(uint8\_t cmd)

{

//I2C开始

I2c\_Start();

//写从机地址

I2c\_WriteByte(WRITE\_ADDRESS);

//写控制字节：0x00，后面传输的是命令

I2c\_WriteByte(0x00);

//写命令

I2c\_WriteByte(cmd);

//I2C结束

I2c\_Stop();

}

以上是OLED写命令的程序，首先按照I2C的通信协议，要有起始和结束信号，然后发送从设备地址和写命令，再根据SSD1306的I2C数据格式，先发送控制字节，再发送具体的命令。

### OLED写数据

void OLED\_WriteDate(uint8\_t pdate)

{

//I2C开始

I2c\_Start();

//写从机地址

I2c\_WriteByte(WRITE\_ADDRESS);

//写控制字节：0x40，后面传输的是数据

I2c\_WriteByte(0x40);

//写数据

I2c\_WriteByte(pdate);

//I2C结束

I2c\_Stop();

}

以上是写一个字节数据的程序，与写命令类似，只是控制指令有所不同。

void OLED\_WriteNByte(uint8\_t \*buf,uint8\_t psize)

{

if(NULL==buf)

return;

int i=0;

//I2C开始

I2c\_Start();

//写从机地址

I2c\_WriteByte(WRITE\_ADDRESS);

//写控制字节：0x40，后面传输的是数据

I2c\_WriteByte(0x40);

//写数据

for(i=0;i<psize;i++)

{

I2c\_WriteByte(buf[i]);

}

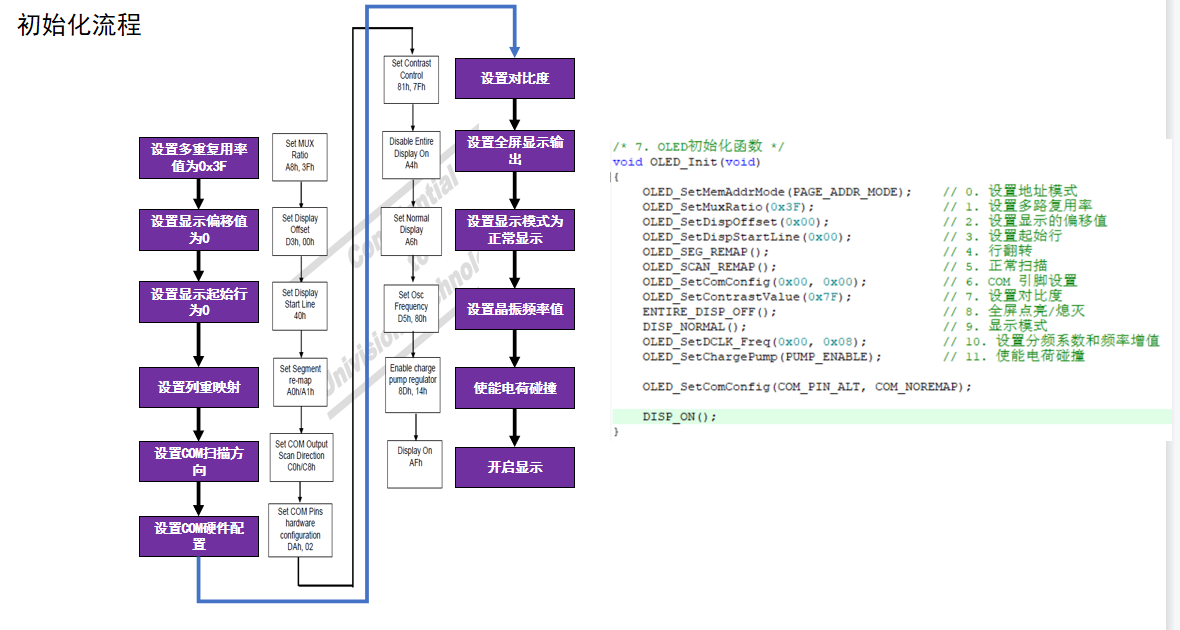
//I2C结束

I2c\_Stop();

}

以上是写多个字节的数据，在写一个字节数据的基础上，根据字节长度，重复写多次一个字节数据。

## OLED初始化流程



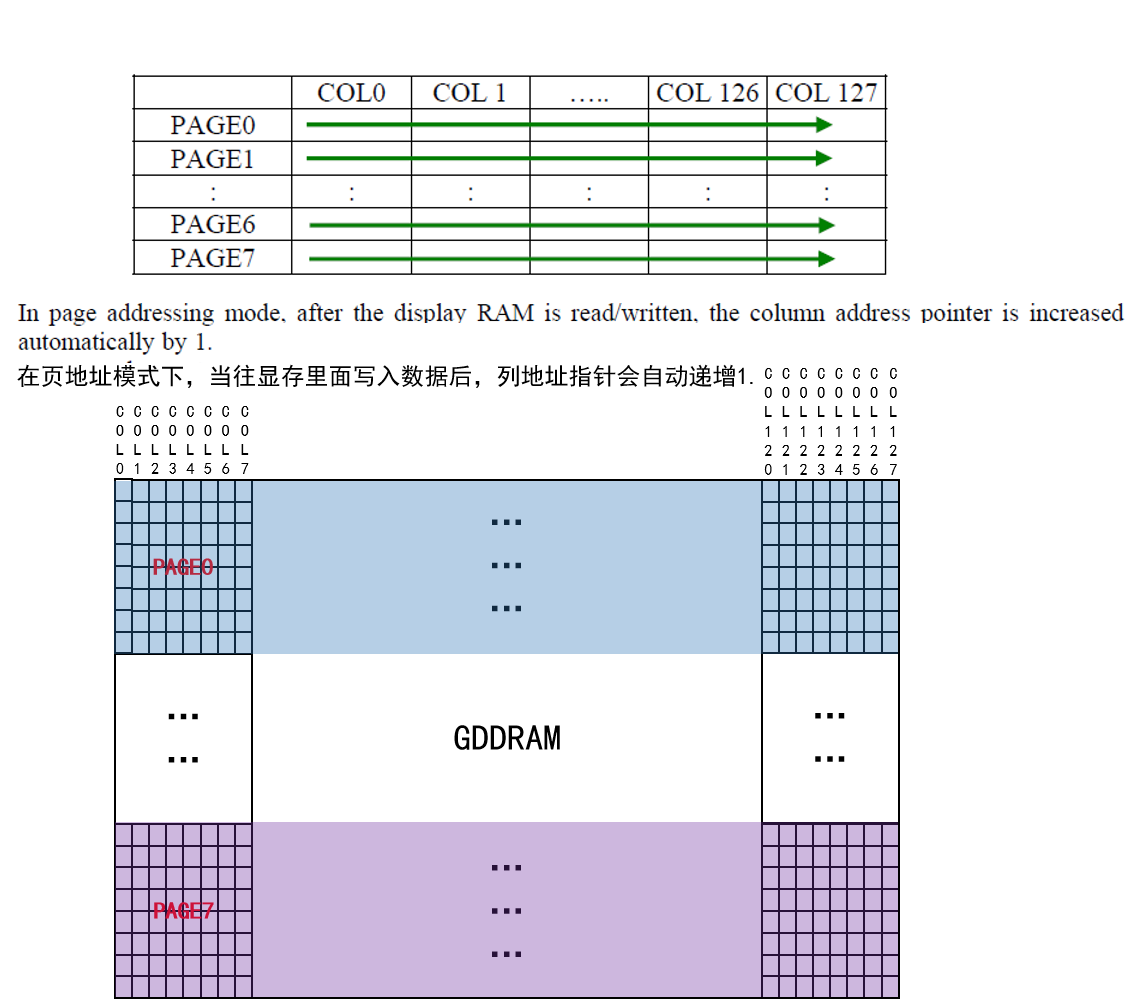
以上是OLED的初始化流程，每一步流程涉及的函数都是根据SSD1306的命令编写，因为没有详细开发研究SSD1306这块芯片，初始化中设置的参数此处不作详细分析，一般需要特殊显示时才会改变参数，这也不是当前学习的重点，只需要按照官方例程设置即可。

## OELD的寻址方式

OLED的显存寻址方式，就是按照怎么样的顺序找到显存的地址，并将显存地址的数据内容读到显示屏幕上。共有三种寻址方式：

1. 页地址模式
2. 水平地址模式
3. 垂直地址模式

主要分析页地址模式，其他地址模式，后续再学习。



如图所示，页地址模式，在设置好起始页和起始行后，就会将数据从起始页开始映射显示，一个字节显示所在页的一列，显示完一列后，列指针自增，显示下一列，直至显示到127列。当显示完第127列之后，会回到第0列，但是页指针不会自增，需要程序控制页指针增加。

## OLED显示编程

OLED的显示编程，根据前面的分析，OLED初始化后，只需要通过I2C向SSD1306传输字符对应的二进制数据即可，字符对应的二进制数据可以用工具生成。

从3.4了解到，为OLED编程写入字符前，需要首先指定页地址和列地址

### 显示一个字符

/\*

\* 函数名：OLED\_PutChar

\* 功能描述：显示一个字符

\* 输入参数：page --> 起始页地址

\* col --> 起始列地址

\* c --> 显示的字符

\* 输出参数：无

\* 返回值：无

\*/

void OLED\_PutChar(uint8\_t page, uint8\_t col, char c)

{

OLED\_SetPosition(page, col);

OLED\_WriteNByte((uint8\_t\*)&ascii\_font[c][0], 8);

OLED\_SetPosition(page + 1, col);

OLED\_WriteNByte((uint8\_t\*)&ascii\_font[c][8], 8);

}

以上是显示一个字符的函数，变量ascii\_font是存放相应字符二进制的二维数据。因为英文字符是8\*16的，占16字节，需要2页8列才能显示完，因此在第一页显示完8列后，需要换页，即页指针+1。

### 显示字符串

/\*

\* 函数名：OLED\_PutString

\* 功能描述：显示一个字符串

\* 输入参数：page --> 起始页地址

\* col --> 起始列地址

\* str --> 显示的字符串

\* 输出参数：无

\* 返回值：无

\*/

void OLED\_PutString(uint8\_t page,uint8\_t col,char \*string)

{

if(NULL==string)

return ;

uint32\_t i=0;

while('\0'!=string[i])

{

OLED\_PutChar(page,col,string[i]);

col+=8;

i++;

if(col>127)

{

page+=2;

col=0;

}

if(page>7)

{

return ;

}

}

}

以上是显示字符串的程序，与显示一个字符不同的是，需要控制字符结束和显示完一个字符后的换页与换行，以及显示完一页后的换页。