常用函数说明

GX09 GX06 常用函数

2020-8-25

深圳市国宇创鑫科技开发有限公司

深圳宝安西乡航城大道航城创新孵化基地 A3 栋 4 楼 3A03

目录

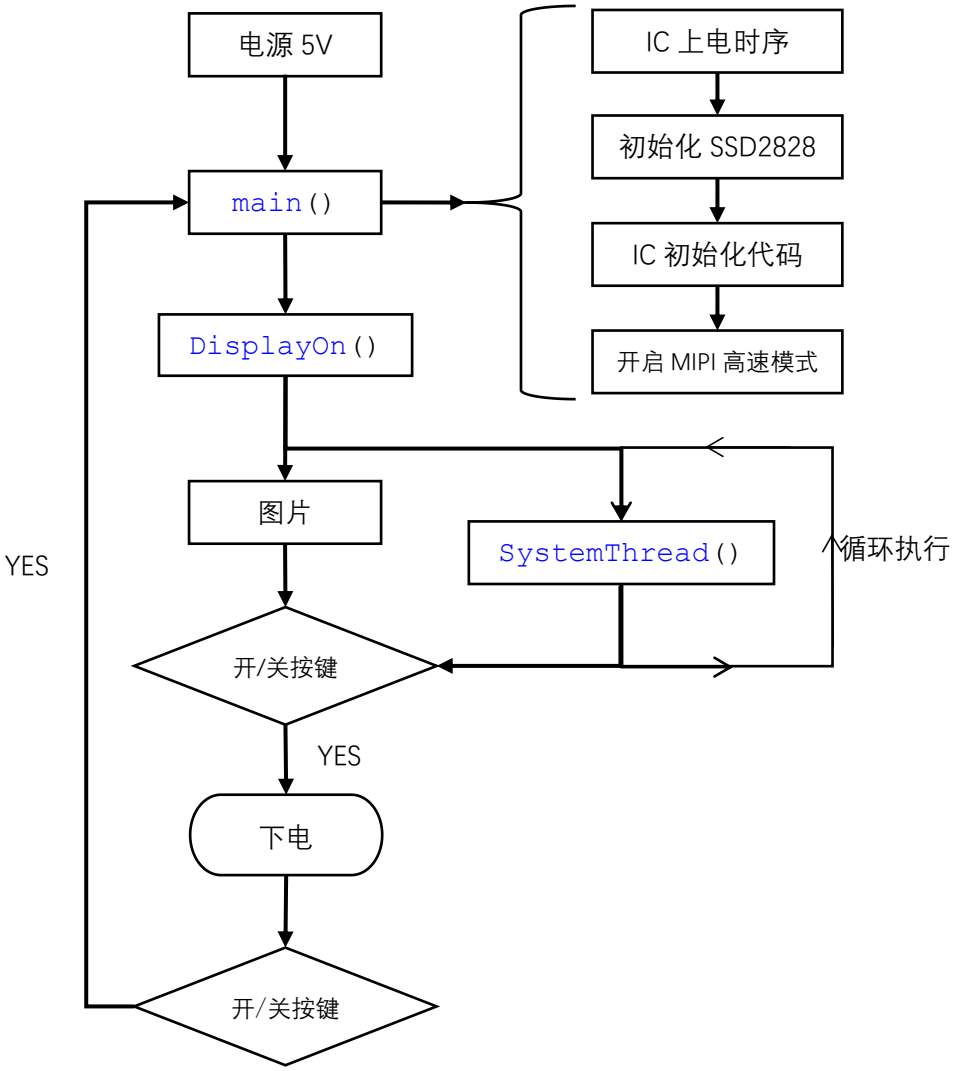
常用函数说明

1. 必要函数

```
void main();//主函数，上电最先执行，包含上电时序、2828 设定、IC 初始化
void DisplayOn();//主函数执行完，执行此函数
void PowerOffSequence();//按“开/关”按键执行，其中包含 IC 下电时序
```

2. 可用函数（非必要）

```
void SystemThread();//在 DisplayOn()函数执行完后，循环执行此函数
void ReadOperation();//可在 main()中调用的函数
```



3. 函数 Set_POWER(1,0,0,0);

参数 1：控制 IOVCC(1.8V)上下电，1→1.8V ON， 0→1.8V OFF
参数 2：控制 VCC(2.8V)上下电，1→2.8V ON， 0→2.8V OFF
参数 3：控制 SSD2828 的 5V 供电，1→5.0V ON， 0→5.0V OFF
参数 4：控制背光供电，1→BL ON， 0→BL OFF

4. 函数 Delay(X)；

参数 1：延时函数，表示延时 X 个单位，每个单位 200us（***注意每个单位不是 1ms***）
快速换算 ms = X / 5;

5. 函数 Set_BOOST(5.5, 5.5, 0x01, 50);

参数 1：VSP(或 AVDD)电压值，步进 0.05V
参数 2：VSN(或 AVEE)电压值，步进 0.05V
参数 3：

上电：	0x01	VSP ON	→	Delay	→	VSN ON
	0x02	VSN ON	→	Delay	→	VSP ON
	0x03	VSP VSN 同时上电				
下电：	0x81	VSP OFF	→	Delay	→	VSN OFF
	0x82	VSN OFF	→	Delay	→	VSP OFF
	0x83	VSP VSN 同时下电				

参数 4：Delay 延时，单位 200us

6. 函数 Set_RESET(1,1)；

参数 1：控制 SSD2828 复位，1→MIPI RESET 1 0→MIPI RESET 0
参数 2：控制 LCM IC 复位， 1→LCM RESET 1 0→LCM RESET 0

7. 函数 Set_GPIO(1)；

参数 1： **GX09** -- AVDD 固定输出 7.5V
GX06 -- VGH、VGL、AVDD、VCOM 四路电压开关，1→ON 0→OFF
(注：此四路电压的电压值，通过按键旁边的蓝色可调电位器调节)

8. 函数 Set_DC2DC(10.1, 18.6, 3.5)； //仅 GX09B GX09C 可用

此方法需要编译器版本 3.8.0 以上才支持，编译器版本过低无法编译
版本查看：打开 LCD Studio，点击右上角"About",即可查看
参数 1：设置 GX09C 的 AVDD 电压值（V=1.2V ~ 16V A=200mA(max)）
参数 2：设置 GX09C 的 VGH 电压值（V=1.5V ~ 26V A=200mA(max)）
参数 3：设置 GX09C 的 VCOM 电压值（V=0.1V ~ 4.5V A=50mA(max)）
注： VGL = VSN， 共用设置函数 Set_BOOST(0, 8.9, 0x01, 50); **（V=-1.2V ~ -16.0V A=200mA(max)）**

9. 函数 Write_SSPI_REG(0xB7,0x015B);

参数 1： SSD2828 寄存器
参数 2： 配置 SSD2828 寄存器的参数
注：用于配置 SSD2828 的参数，以上写法是往 SSD2828 的 0xB7 寄存器写入参数 0x015B
详细参数设置，请阅读 SSD2828 DataSheet 自行配置

10. 函数 SSD_LANE(4， 0);

参数 1： MIPI LANE 选择，1→1 Lane　　2→2 Lane　　3→3 Lane　　4→4 Lane
参数 2： 设置 MIPI 速率，0→自动设置(仅 Non-burst mode 有效)　　X→ X Mbps/Lane

11. 函数 Wait_Key();

程序执行到 Wait_Key()处，停止往下执行，直到"开/关"按键按下才往下执行

12. MIPI 写入函数

写法①

DCS_Short_Write_NP(U16 DCS);//DCS 写寄存器，无参数
DCS_Short_Write_1P(U16 DCS,U16 P);//DCS 写寄存器，1 个参数
DCS_Long_Write_2P(U16 DCS,U16 P1,U16 P2);//DCS 写寄存器，2 个参数
DCS_Long_Write_3P(U16 DCS,U16 P1,U16 P2,U16 P3);//DCS 写寄存器，3 个参数
DCS_Long_Write_4P(U16 DCS,U16 P1,U16 P2,U16 P3,U16 P4);//DCS 写寄存器，6 个参数
DCS_Long_Write_5P(U16 DCS,U16 P1,U16 P2,U16 P3,U16 P4,U16 P5);//DCS 写寄存器，5 个参数
DCS_Long_Write_6P(U16 DCS,U16 P1,U16 P2,U16 P3,U16 P4,U16 P5,U16 P6);//DCS 写寄存器，6 个参数
DCS_Long_Write_7P(U16 DCS,U16 P1,U16 P2,U16 P3,U16 P4,U16 P5,U16 P6,U16 P7);//DCS 写寄存器，7 个参数
DCS_Long_Write_FIFO(U16 NUM,U16 *P);//大于 7 个参数，使用数组写入
注：数组写入最大支持 256 个数据

写法②

Generic_Short_Write_NP(U16 Generic);//Generic 写寄存器，无参数
Generic_Short_Write_1P(U16 Generic,U16 P);//Generic 写寄存器，1 个参数
Generic_Long_Write_2P(U16 Generic,U16 P1,U16 P2);//Generic 写寄存器，2 个参数
Generic_Long_Write_3P(U16 Generic,U16 P1,U16 P2,U16 P3);//Generic 写寄存器，3 个参数
Generic_Long_Write_4P(U16 Generic,U16 P1,U16 P2,U16 P3,U16 P4);//Generic 写寄存器，6 个参数
Generic_Long_Write_5P(U16 Generic,U16 P1,U16 P2,U16 P3,U16 P4,U16 P5);//Generic 写寄存器，5 个参数
Generic_Long_Write_6P(U16 Generic,U16 P1,U16 P2,U16 P3,U16 P4,U16 P5,U16 P6);//Generic 写寄存器，6 个参数
Generic_Long_Write_7P(U16 Generic,U16 P1,U16 P2,U16 P3,U16 P4,U16 P5,U16 P6,U16 P7);//Generic 写寄存器，7 个参数
Generic_Long_Write_FIFO(U16 NUM,U16 *P);//大于 7 个参数，使用数组写入
注：数组写入最大支持 256 个数据

写法③

```
SSD_SEND(0x11,0xB0,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08);
```

此方法需要编译器版本 3.8.0 以上才支持，编译器版本过低无法编译

版本查看：打开 LCD Studio，点击右上角"About",即可查看

参数 1： 0x00→ DCS_Short_Write

0x01→ DCS_Long_Write

0x10→ Generic_Short_Write

0x11→ Generic_Long_Write

参数 2~参数 N：

寄存器和参数，总数最大为 256 个

13. MIPI 回读函数

IC 寄存器回读值常用以下函数回读:

```
Generic_Short_Read_1P(0x04,3,BUFFER+0);
DCS_Short_Read_NP(0x04,3,BUFFER+0);
```

个别 IC 寄存器回读使用以下函数回读:

```
Generic_Long_Read_1P(0x04,3,BUFFER+0);
DCS_Long_Read_NP(0x04,3,BUFFER+0);
```

以上 4 个函数参数设置如下:

参数 1：回读的寄存器地址（示例为 0x04）

参数 2：回读寄存器参数的个数（示例为 3 个）

参数 3：将回读到的值存放到 BUFFER 中（示例为存放到 BUFFER[0] BUFFER[1] BUFFER[2]）

(注：示例解读为，读 0x04 寄存器，读 3 个值，将值存放在从 BUFFER+0 开始的 BUFFER[0]、BUFFER[1]、BUFFER[2]中，可在 TEXT 画面查看 BUFFER 存储的数值)

BUFFER 只有 8 个，回读的个数大于 8，会从 BUFFER[0]开始循环覆盖

即：第 9 个存于 BUFFER[0]，第 10 个存于 BUFFER[1]，第 11 个存于 BUFFER[2] ……

```
DCS_Short_Read_NP(0xCC,16,BUFFER+0);
```

--BUFFER[0]~BUFFER[7]保存的是第 9 个到第 16 个参数，1~8 被覆盖

```
DCS_Short_Read_NP(0xCC,24,BUFFER+0);
```

--BUFFER[0]~BUFFER[7]保存的是第 17 个到第 24 个参数，1~8 被覆盖,9~16 再次被覆盖

14. 函数 SSD_MODE(0, 1);

参数 1： **0** → Video Mode - Non burst mode with sync pulses

1 → Video Mode - Non burst mode with sync events

2 → Video Mode - Burst mode

3 → Command mode(需要外接我司 Command 转接板)

参数 2: **0** → 无操作

1 → 开启 MIPI HS Mode

15. 函数 Beep(1);

参数 1: 0 → 关闭治具蜂鸣器
1 → 开启治具蜂鸣器

16. 函数 Set_BIT(X);

参数 1: 可选填 0、1 、2,设置内部的固定存取地址为 0/1/2

17. 函数 Get_BIT();

无参数，获取内部的固定存取地址的值到 BUFFER[0]

18. 函数 Wait_Done();

等待程序及图片加载完毕再显示画面，添加在 main()函数最后，用于去掉开机感叹号“!”

19. 函数 Set_TEXT(0, 0, 0x00);

参数 1: 显示标题 0 → 显示 “ID”
1 → 显示 “OTP”
2 → 显示 “ADC”
3 → 显示 “TE”
4 → 显示 “CABC”
0x1_ _ → 显示高低 4 位组合标题

参数 2: 锁屏状态 0→不锁定屏幕，可切换画面
1→锁定屏幕，不可切换画面

参数 3: 显示字符 0x0_ → 黑色背景
0x1_ → 绿色背景
0x2_ → 红色背景
0x3_ → 黄色背景

0x_0 → 显示 BUFFER 中的数据
0x_1 → 显示“PASS”
0x_2 → 显示“FAIL”
0x_3 → 显示“OLD”
0x_4 → 显示“TEST”
0x_5 → 显示“OFF”

20. 函数 memcmp("0x01,0x02,0x03,…")

用于比对 BUFFER 中的数值，具体使用如下：

```
DCS_Short_Read_NP(0x04, 3, BUFFER+0);
//读0x04寄存器的前3个值，数据存到BUFFER中，从BUFFER[0]开始，依次存放到BUFFER[0]、BUFFER[1]、BUFFER[2]
if(memcmp("0x01,0x02,0x03")) //如果BUFFER[0]等于0x01
                            //同时BUFFER[1]等于0x02
                            //同时BUFFER[2] 等于0x03
                            //那么，条件为真
{
    Set_TEXT(0,0,0x01);//显示ID PASS，不锁屏幕
}
else
{
    Set_TEXT(0,1,0x02);//显示 ID FAIL，锁住屏幕
}
```

21. 函数 if(memcmp"0x01"){ } else{ }

此函数仅支持两层嵌套操作，不支持 3 层以上嵌套

```
if(memcmp("0x01"))
{
    if(memcmp("0x01,0x02"))
    {
        //User code T2
    }
    else
    {
        //User code T2
    }
}
else
{
    //User code T1
}
```

正确

```
if(memcmp("0x01"))
{
    if(memcmp("0x01,0x02"))
    {
        //User code T2
        if(memcmp("0x01,0x02,0x03"))
        {
            //User code T3
        }
        else
        {
            //User code T3
        }
    }
    else
    {
        //User code T2
    }
}
else
{
    //User code T1
}
```

错误

22. 函数 Read_ADC(1.8, 0, 0.1);

参数 1： ID 电阻电压值（0.0V~3.3V）

参数 2： 0 → 当 ID 电压值不在设定范围内，背光闪烁

1 → 返回比对结果到 BUFFER[0]；0-范围外 1-范围内

2 → 返回电压值到 BUFFER[1]-[0]，例如 1.82V，在 TEXT 画面显示 0182

参数 3： 误差范围。（本示例为 1.8V±0.1V）

23. 函数 Read_ADC2(1.2, 0, 0.1);

参数 1： ID2 电阻电压值（0.0V~3.3V）

参数 2： 0 → 当 ID 电压值不在设定范围内，背光闪烁

1 → 返回比对结果到 BUFFER[0]；0-范围外 1-范围内

2 → 返回电压值到 BUFFER[1]-[0]，例如 1.22V，在 TEXT 画面显示 0122

参数 3： 误差范围。（本示例为 1.2V±0.1V）

24. 函数 Read_CABC(X, 0x00, 50);

参数 1： CABC 频率 **X**(10Hz~50KHz)

占空比=(**X**/10)%

电压有效值 **X**(0.0V~3.3V)

参数 2： 0x_0 → 频率/占空比/电压有效值不在范围内，背光闪烁。

0x_1 → 返回比对结果到 BUFFER[0]，0 - 范围外；1 – 范围内。

0x_2 → 返回频率值/占空比/电压值到 BUFFER[2]-[1]-[0]。

0x0_ → 测量 CABC 频率。

0x1_ → 测量 CABC 占空比。（GX06 系列不支持）

0x2_ → 测量 CABC 电压有效值。（GX06 系列不支持）

0x0F → 返回电源板版本号到 BUFFER[0]。

参数 3： 设定频率/占空比/电压有效值误差范围。

示例：

```
Read_CABC(40000, 0x01, 500);  //40KHz±500Hz，返回比对结果到 BUFFER[0]。
Read_CABC(500, 0x11, 50);  //50%±5%，返回比对结果到 BUFFER[0]。
Read_CABC(1.5, 0x21, 0.2);  //1.5V±0.2V，返回比对结果到 BUFFER[0]。
```

(注：测量 CABC 占空比、电压有效值功能，GX08 系列、GX09 系列才支持，GX06 系列不支持；测量 CABC 频率 GX06 系列、GX08 系列、GX09 系列都支持)

25. 函数 Read_TE(X, 0, 5);

参数 1： TE 频率 **X**(10Hz~1KHz)
占空比=(**X**/10)%
电压有效值 **X**(0.0V~3.3V)

参数 2： 0x_0 → 频率/占空比/电压有效值不在范围内，背光闪烁。
0x_1 → 返回比对结果到 BUFFER[0]，0 - 范围外；1 – 范围内。
0x_2 → 返回频率值/占空比/电压值到 BUFFER[2]-[1]-[0]。
0x0_ → 测量 TE 频率。
0x1_ → 测量 TE 占空比。（GX06 系列不支持）
0x2_ → 测量 TE 电压有效值。（GX06 系列不支持）
0x0F → 返回电源板版本号到 BUFFER[0]。

参数 3： 设定频率/占空比/有效值范围。
示例：

```
Read_TE(60, 0x01, 2);  //60Hz±2Hz，返回比对结果到 BUFFER[0]。
Read_TE(10, 0x11, 5);  //1%±0.5%，返回比对结果到 BUFFER[0]。
Read_TE(1.4, 0x21, 0.2);  //1.4V±0.2V，返回比对结果到 BUFFER[0]。
```

(注：测 TE 占空比、电压有效值功能，GX08 系列、GX09 系列才支持，GX06 系列不支持；测量 TE 频率 GX06 系列、GX08 系列、GX09 系列都支持)

26. 函数 Get_BurnKey();

返回烧录按键状态到 BUFFER[0]， **0** - 未按下 **1** - 按下
该函数需要放在 main 函数执行
示例：

```
Get_BurnKey();//等待烧录按键
if(memcmp("0x01"))
{
    //OTP 烧录流程
}
```

27. 函数 Set_STANDBY();

MIPI 停止数据传输

28. 函数 SleepInCurrent(1);

测量睡眠电流使用，需搭配我司 GX07 使用，示例如下：

```
void PowerOffSequence()
{
    DCS_Short_Write_NP(0x28);
    Delay(200);
    DCS_Short_Write_NP(0x10);
    Delay(100);
    Set_STANDBY();//Video transfer stop
    Delay(50);
    SleepInCurrent(1);
    Set_RESET(1,0);//MIPI RESET 1, LCD RESET 0
    Delay(50);
    Set_RESET(0,0);//MIPI RESET 0, LCD RESET 0
    Delay(50);
    Set_POWER(1,1,0,1);//1.8V ON, 2.8V ON, 5V OFF, BL ON
    Delay(150);
    Set_BOOST(5.75, 5.75, 0x82, 50);//VDD, VEE, OFF:VDD->VEE, 10ms
    Delay(50);
    Set_POWER(1,0,0,1);//1.8V ON, 2.8V OFF, 5V OFF, BL ON
    Delay(150);
    Set_POWER(0,0,0,0);//1.8V OFF, 2.8V OFF, 5V OFF, BL OFF
}
```