# SSD1306 OLED 驱动

SAM4N 学习笔记

版本号：1.00

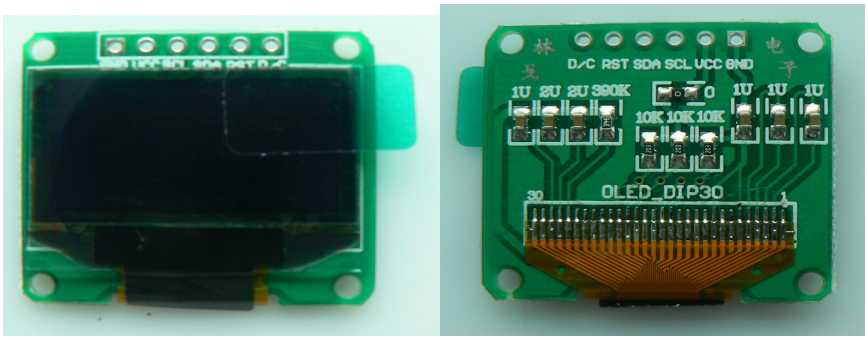
日期：2013/10/25

## 一、准备工作:

将上一节搭建的工程复制一份，命名为“7.ssd1306”。这一节主要讲如何使用 SAM4N 的驱动 SSD1306 OLED 小液晶屏，实现简单的字符显示。

## 二、硬件说明:

笔者手上正好有个 OLED 小屏幕，所以现在正好派上用场了。官方的例子中也有个 OLED 的驱动,用的也是 SSD1306 的控制器,分辨率是 128*32。我手上这个 OLED 也是 SSD1306 的控制器，不同的是分辨率为 128*64。液晶屏模块如下图：




这个 OLED 的接口是 SPI 的，上面的图看到 SDA 和 SCL 的字样，但实际并不是 I2C 接口，以前买这个小屏的时候我也奇怪了，而且 spi 还没有引出 cs 线，默认就下拉的。

这个屏正好可以插在板子旁边的排针上，如下图：


硬件连接如下：

SCL----PIOC12->PIO_SODR=(0x01<<12)

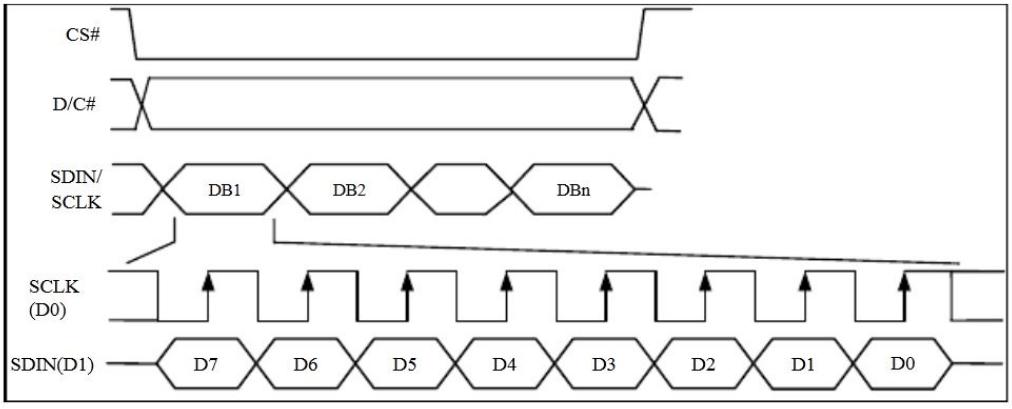
SDA----PIOA25

RST ---- PIOC24

DC ---- PIOC23

下面是 SSD1306 的四线串行接口时序：

**Figure 8-5 : Write procedure in 4-wire Serial interface mode**




因为这个 OLED 默认就把 CS 拉低了，所以每次传输可以不用去管 CS，但没 CS 这几个 IO 就被独占了。从图上可以看到数据是在 SCLK 上升沿时被传入，高位在前。DC 用于表示写入的是数据还是指令，DC 为低时是指令写入，DC 为高时是数据写入。

128*64 的 OLED 在 SSD1306 的控制下，被分成 8 个页，每个页有 8 行 128 列，每个列对应一个字节。

## 三、程序编写:

首先定义 OLED 控制器对应的 GPIO 以及 OLED 的分辨率、缓冲区等。

```
#define SCL_SET() PIOC->PIO_SODR=(0x01<<12)
#define SCL_CLR() PIOC->PIO_CODR=(0x01<<12)
#define SDA_SET() PIOA->PIO_SODR=(0x01<<25)
#define SDA_CLR() PIOA->PIO_CODR=(0x01<<25)
#define RST_SET() PIOC->PIO_SODR=(0x01<<24)
#define RST_CLR() PIOC->PIO_CODR=(0x01<<24)
#define DC_SET() PIOC->PIO_SODR=(0x01<<23)
#define DC_CLR() PIOC->PIO_CODR=(0x01<<23)

#define  xsize 128
#define ysize 64
  /*液晶屏的字体*/
static sFONT *LCD_Currentfonts;
static uint8_t framebuffer[xsize * ysize / 8];
```

这里我们采用了 framebuffer 的方式，开辟一个对应屏的每个像素点的缓冲区。这种方式需要消耗一定的 ram 内存空间，但这点对于 SAM4N 片内 80K 的 ram 来说，微不足道。这样做的好处是可以在任意地方绘制各种图形，而且绘制完以后才刷新到 lcd 上，这在一定程度上避免了闪烁。 而且还可以对所显示的内容进行各种操作，比较灵活，但同时也需要付出点代价。

接下来是两个很重要的函数，那就是向 SSD1306 写指令和写数据函数，这两个是驱动的核心，基本所有对 OLED 的操作都要通过这两个函数来完成。

```
static void ssd1306_write_command(uint8_t command)
{
     uint8_t i;
     DC_CLR();
  for(i=0;i<8;i++)
  {
   if(command&0x80)
   {
    SDA_SET();
   }else
   {
    SDA_CLR();
   }
   SCL_CLR();
   SCL_SET();
   command<<=1;
  }
```

```
}
static  void ssd1306_write_data(uint8_t data)
{
    uint8_t i;
    DC_SET();
  for(i=0;i<8;i++)
  {
   if(data&0x80)
   {
    SDA_SET();
   }else
   {
    SDA_CLR();
   }
   SCL_CLR();
   SCL_SET();
   data<<=1;
  }
}
```

这是根据上面的时序图写的对 OLED 的写数据写指令两个函数，高位在前，在一个时钟上升沿时完成一个 bit 数据的传入。

接下来是对相应 GPIO 的初始化配置和 OLED 的初始化。

```
static void ssd1306_gpio_init(void)
{
   /*使能 PIOC 和 PIOA 外设时钟*/
 REG_PMC_PCER0=(0x01<<ID_PIOC)|(0x01<<ID_PIOA);
 /*使能 PC23、PC24 和 PC12 管脚*/
 PIOC->PIO_PER|=(0x03<<23|0x01<<12);
    /*使能 PC23、PC24 管脚输出*/
 PIOC->PIO_OER|=(0x03<<23|0x01<<12);
     /*使能 PA25 管脚*/
 PIOA->PIO_PER|=(0x01<<25);
    /*使能 PA25 管脚输出*/
 PIOA->PIO_OER|=(0x01<<25);
 SCL_SET();
 SDA_SET();
 DC_CLR();
 RST_SET();
}
void ssd1306_hw_init(void)
{
    ssd1306_gpio_init();
    // Do a hard reset of the OLED display controller
```

```
      ssd1306_hard_reset();

      // Initialize the interface
      ssd1306_gpio_init();

      // 1/32 Duty (0x0F~0x3F)
      ssd1306_write_command(SET_MULTIPLEX_RATIO);
      ssd1306_write_command(0x3F);

      // Shift Mapping RAM Counter (0x00~0x3F)
      ssd1306_write_command(SET_DISPLAY_OFFSET);
      ssd1306_write_command(0x00);

      // Set Mapping RAM Display Start Line (0x00~0x3F)
      ssd1306_write_command(SET_START_LINE(0x00));

      // Set Column Address 0 Mapped to SEG0
      ssd1306_write_command(SET_SEGMENT_RE_MAP_COL127_SEG0);

      // Set COM/Row Scan Scan from COM63 to 0
      ssd1306_write_command(SET_COM_OUTPUT_SCAN_DOWN);

      // Set COM Pins hardware configuration
      ssd1306_write_command(SET_COM_PINS);
      ssd1306_write_command(0x12);

      ssd1306_set_contrast(0xCF);

      // Disable Entire display On
      ssd1306_write_command(ENTIRE_DISPLAY_AND_GDDRAM_ON);

      ssd1306_display_invert_disable();

      // Set Display Clock Divide Ratio / Oscillator Frequency
(Default => 0x80)
      ssd1306_write_command(SET_DISPLAY_CLOCK_DIVIDE_RATIO);
      ssd1306_write_command(0x80);

      // Enable charge pump regulator
      ssd1306_write_command(SET_CHARGE_PUMP_SETTING);
      ssd1306_write_command(0x14);

      // Set VCOMH Deselect Level
      ssd1306_write_command(SET_VCOMH_DESELECT_LEVEL);
```

```
      ssd1306_write_command(0x40); // Default => 0x20 (0.77*VCC)

      // Set Pre-Charge as 15 Clocks & Discharge as 1 Clock
      ssd1306_write_command(SET_PRE_CHARGE_PERIOD);
      ssd1306_write_command(0xF1);

      ssd1306_display_on();
      ssd1306_clear(BLACK);
  }
```

在 SSD1306 初始化时需要进行一些配置，比如显示方向，分辨率大小，时钟等。
有了这些东西还不够，这只是完成了初始化功能，要进行一些显示，还需要去扩展一下显示函数，下面是笔者实现的一些常用的图形显示函数：

```
  void glcd_init(void)
  {
      ssd1306_hw_init();
      glcd_set_font(&Font8x16);
  }
  /*************************************************************
   * 函数名: glcd_set_font()
   * 输入  : sFONT *fonts 要设置的字体
   * 输出  : void
   * 描述  : 设置 LCD 的字体
   * 调用  : 外部调用

*************************************************************/
  void glcd_set_font(sFONT *fonts)
  {
    LCD_Currentfonts = fonts;
  }
  /*************************************************************
   * 函数名: glcd_get_font()
   * 输入  : void
   * 输出  : sFONT * 获取字体
   * 描述  : 设置 LCD 的字体
   * 调用  : 外部调用

*************************************************************/
  sFONT* glcd_get_font(void)
  {
    return LCD_Currentfonts;
  }
```

```
  void glcd_draw_pixel(uint16_t x, uint16_t y,color_t color)
  {
      if( x > xsize ||  y > ysize)
      {
             return;
      }
      {
       uint8_t page=y/8;
   uint8_t row=y%8;
   if(color==BLACK)
       {
        framebuffer[page*128+x]&=~(0x01<<row);
     }else
       {
      framebuffer[page*128+x]|=(0x01<<row);
     }
    }
  }
  void glcd_clear(color_t color)
  {
  ssd1306_clear(color);
  }

  /*************************************************************
   * 函数名: glcd_draw_hline()
   * 输入  : uint16_t Xpos, uint16_t Ypos, uint16_t Length 起点 X 和
Y 坐标及长度
   * 输出  : void
   * 描述  : 画水平线
   * 调用  : 外部调用

*************************************************************/
  void glcd_draw_hline(uint16_t Xpos, uint16_t Ypos, uint16_t
Length,color_t color)
  {
       uint8_t page=Ypos/8;
   uint8_t row=Ypos%8;
   while(Length--){
   if(color==BLACK)
       {
        framebuffer[page*128+Xpos]&=~(0x01<<row);
     }else
       {
      framebuffer[page*128+Xpos]|=(0x01<<row);
```

```
    }
       Xpos++;
   }
  }
  /*************************************************************
* 函数名：glcd_draw_vline()
   * 输入  ：uint16_t Xpos, uint16_t Ypos, uint16_t Length 起点 X 和
Y 坐标及长度
   * 输出  ：void
   * 描述  ：画垂直线
   * 调用  ：外部调用

*************************************************************/
  void glcd_draw_vline(uint16_t Xpos, uint16_t Ypos, uint16_t
Length,color_t color)
  {
   while(Length--){
        uint8_t page=Ypos/8;
   uint8_t row=Ypos%8;
   if(color==BLACK)
       {
        framebuffer[page*128+Xpos]&=~(0x01<<row);
     }else
       {
      framebuffer[page*128+Xpos]|=(0x01<<row);
     }
       Ypos++;
   }
  }
  /*************************************************************
   * 函数名：glcd_draw_rect()
   * 输入  ：uint16_t Xpos, uint16_t Ypos, uint16_t Width, uint8_t
Height 矩形左上角点的坐标及宽和高
   * 输出  ：void
   * 描述  ：画矩形函数
   * 调用  ：外部调用

*************************************************************/
  void glcd_draw_rect(uint16_t Xpos, uint16_t Ypos, uint16_t Width,
uint8_t Height,color_t color)
  {
    glcd_draw_hline(Xpos, Ypos, Width,color);
    glcd_draw_hline(Xpos, Ypos+ Height, Width,color);
    glcd_draw_vline(Xpos, Ypos, Height,color);
```

```
    glcd_draw_vline(Xpos+ Width,Ypos, Height,color);

  }
  /*************************************************************
* 函数名: glcd_draw_circle()
   * 输入  : uint16_t Xpos, uint16_t Ypos, uint16_t Radius 圆心坐标
点及半径
   * 输出  : void
   * 描述  : 画圆函数
   * 调用  : 外部调用

*************************************************************/
  void glcd_draw_circle(uint16_t Xpos, uint16_t Ypos, uint16_t
Radius,color_t color)
  {
    int32_t  D;/* Decision Variable */
    uint32_t  CurX;/* Current X Value */
    uint32_t  CurY;/* Current Y Value */

    D = 3 - (Radius << 1);
    CurX = 0;
    CurY = Radius;

    while (CurX <= CurY)
    {
        glcd_draw_pixel(Xpos + CurX, Ypos + CurY,color);
      glcd_draw_pixel(Xpos + CurX, Ypos - CurY,color);
       glcd_draw_pixel(Xpos - CurX, Ypos + CurY,color);
       glcd_draw_pixel(Xpos - CurX, Ypos - CurY,color);
       glcd_draw_pixel(Xpos + CurY, Ypos + CurX,color);
       glcd_draw_pixel(Xpos + CurY, Ypos - CurX,color);
       glcd_draw_pixel(Xpos - CurY, Ypos + CurX,color);
       glcd_draw_pixel(Xpos - CurY, Ypos - CurX,color);
      if (D < 0)
      {
        D += (CurX << 2) + 6;
      }
      else
      {
        D += ((CurX - CurY) << 2) + 10;
        CurY--;
      }
      CurX++;
    }
```

```
  }
  /*************************************************************
* 函数名：glcd_fill_rect()
   * 输入  ：uint16_t Xpos, uint16_t Ypos, uint16_t Width, uint16_t
Height 填充矩形左上角点、宽和高
   * 输出  ：void
   * 描述  ：画一个填充的矩形
   * 调用  ：外部调用

*************************************************************/
  void glcd_fill_rect(uint16_t Xpos, uint16_t Ypos, uint16_t Width,
uint16_t Height,color_t color)
  {
    glcd_draw_hline(Xpos, Ypos, Width,color);
    glcd_draw_hline(Xpos, Ypos+ Height, Width,color);

    glcd_draw_vline(Xpos, Ypos, Height,color);
    glcd_draw_vline(Xpos+Width, Ypos, Height,color);
    Width --;
    Height--;
    Xpos++;
    while(Height--)
    {
      glcd_draw_hline(Xpos, ++Ypos, Width,color);
    }
  }
  /*************************************************************
   * 函数名：glcd_fill_circle()
   * 输入  ：uint16_t Xpos, uint16_t Ypos, uint16_t Radius 填充圆的
圆心和半径
   * 输出  ：void
   * 描述  ：画一个填充圆
   * 调用  ：外部调用

*************************************************************/
  void glcd_fill_circle(uint16_t Xpos, uint16_t Ypos, uint16_t
Radius,color_t color)
  {
    int32_t  D;    /* Decision Variable */
    uint32_t  CurX;/* Current X Value */
    uint32_t  CurY;/* Current Y Value */
    uint32_t tempcolor;
    D = 3 - (Radius << 1);
```

```
    CurX = 0;
    CurY = Radius;

    while (CurX <= CurY)
    {
      if(CurY > 0)
      {
        glcd_draw_hline(Xpos - CurY, Ypos - CurX, 2*CurY,color);
        glcd_draw_hline(Xpos - CurY, Ypos + CurX, 2*CurY,color);
      }

      if(CurX > 0)
      {
        glcd_draw_hline(Xpos - CurX, Ypos -CurY, 2*CurX,color);
        glcd_draw_hline(Xpos - CurX, Ypos + CurY, 2*CurX,color);
      }
      if (D < 0)
      {
        D += (CurX << 2) + 6;
      }
      else
      {
        D += ((CurX - CurY) << 2) + 10;
        CurY--;
      }
      CurX++;
    }
    glcd_draw_circle(Xpos, Ypos, Radius,color);
  }
  /*************************************************************
   * 函数名: glcd_draw_uniline()
   * 输入  : uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2 起
始点坐标和终点坐标
   * 输出  : void
   * 描述  : 画任意方向的直线
   * 调用  : 外部调用

*************************************************************/
  void glcd_draw_uniline(uint16_t x1, uint16_t y1, uint16_t x2,
uint16_t y2,color_t color)
  {
    int16_t deltax = 0, deltay = 0, x = 0, y = 0, xinc1 = 0, xinc2
= 0,
```

```
    yinc1 = 0, yinc2 = 0, den = 0, num = 0, numadd = 0, numpixels
= 0,
    curpixel = 0;

    deltax = ABS(x2 - x1);        /* The difference between the x's
*/
    deltay = ABS(y2 - y1);        /* The difference between the y's
*/
    x = x1;                      /* Start x off at the first pixel */
    y = y1;                      /* Start y off at the first pixel */

    if (x2 >= x1)                 /* The x-values are increasing */
    {
      xinc1 = 1;
      xinc2 = 1;
    }
    else                         /* The x-values are decreasing */
    {
      xinc1 = -1;
      xinc2 = -1;
    }

    if (y2 >= y1)                 /* The y-values are increasing */
    {
      yinc1 = 1;
      yinc2 = 1;
    }
    else                         /* The y-values are decreasing */
    {
      yinc1 = -1;
      yinc2 = -1;
    }

    if (deltax >= deltay)         /* There is at least one x-value for
every y-value */
    {
      xinc1 = 0;                 /* Don't change the x when numerator
>= denominator */
      yinc2 = 0;                 /* Don't change the y for every
iteration */
      den = deltax;
      num = deltax / 2;
      numadd = deltay;
```

```
     numpixels = deltax;        /* There are more x-values than
y-values */
    }
    else                       /* There is at least one y-value for
every x-value */
    {
      xinc2 = 0;                /* Don't change the x for every
iteration */
      yinc1 = 0;                /* Don't change the y when numerator
>= denominator */
      den = deltay;
      num = deltay / 2;
      numadd = deltax;
      numpixels = deltay;        /* There are more y-values than
x-values */
    }

    for (curpixel = 0; curpixel <= numpixels; curpixel++)
    {
      glcd_draw_pixel(x, y,color);           /* Draw the current
pixel */
      num += numadd;             /* Increase the numerator by the top
of the fraction */
      if (num >= den)            /* Check if numerator >= denominator
*/
      {
        num -= den;              /* Calculate the new numerator value
*/
        x += xinc1;              /* Change the x as appropriate */
        y += yinc1;              /* Change the y as appropriate */
      }
      x += xinc2;                /* Change the x as appropriate */
      y += yinc2;                /* Change the y as appropriate */
    }
  }

  /*************************************************************
* 函数名: glcd_draw_char()
   * 输入  : const uint16_t *c   字符编码
   * 输出  : void
   * 描述  : LCD 画一个字符
   * 调用  : 外部调用

*************************************************************/
```

```
  void glcd_draw_char(uint16_t Xpos, uint16_t Ypos, const uint16_t
*c,color_t color)
  {
    uint32_t index = 0, i = 0;
    uint16_t  x = 0,y=0;
    y = Ypos;

    for(index = 0; index < LCD_Currentfonts->Height; index++)
    {
        x=Xpos;
      for(i = 0; i < LCD_Currentfonts->Width; i++)
      {

       if((((c[index] & ((0x80 << ((LCD_Currentfonts->Width / 12 )
* 8 ) ) >> i)) == 0x00) &&(LCD_Currentfonts->Width <= 12))||
         (((c[index] & (0x1 << i)) ==
0x00)&&(LCD_Currentfonts->Width > 12 )))

       {
                glcd_draw_pixel(x++,y,!color);
       }
       else
       {
        glcd_draw_pixel(x++,y,color);
       }
      }
      y++;

    }

  }
  /*************************************************************
   * 函数名：glcd_display_char()
   * 输入  ：uint16_t Xpos, uint16_t Ypos, uint8_t Ascii 显示的位置
和字符
   * 输出  ：void
   * 描述  ：LCD 显示一个字符
   * 调用  ：外部调用

*************************************************************/
  void glcd_display_char(uint16_t Xpos, uint16_t Ypos, uint8_t
Ascii,color_t color)
  {
    Ascii -= 32;
```

```
   glcd_draw_char(Xpos, Ypos, &LCD_Currentfonts->table[Ascii *
LCD_Currentfonts->Height],color);
  }
  /*************************************************************
   * 函数名: glcd_draw_string()
   * 输入  : u16 xpos, u16 ypos, u8 *ptr 显示的位置和字符串
   * 输出  : void
   * 描述  : LCD 显示一串字符
   * 调用  : 外部调用

*************************************************************/
  void glcd_draw_string(uint16_t xpos, uint16_t ypos, uint8_t
*ptr,color_t color)
  {
      uint16_t refypos=xpos;
      while(*ptr!=0)
      {
             glcd_display_char(refypos,ypos,*ptr,color);
      refypos+=LCD_Currentfonts->Width;
      ptr++;
      }
  }
```

有了这些基本的图形绘制函数，就可以在 OLED 上面绘制我们需要显示的一下图形了，下面是写的一个小例子：

```
  int main(void)
  {
   systick_hw_init();
   led_hw_init();
   USART0_Init(115200);
   glcd_init();
   glcd_draw_string(0,0,"<<SSD1306 OLED>>",WHITE);
   glcd_draw_string(0,16,"Hello,SAM4N",WHITE);
   glcd_draw_string(0,32,"www.eeboard.com",WHITE);
   glcd_draw_string(0,48,"oled font test",WHITE);
   glcd_update();
   while(1){
       USART0_Init(115200);
  USART0_SendString("hello\r\n");
   PIOB->PIO_CODR=(0x01<<LED0_PIN);
   delay_ms(100);
   PIOB->PIO_SODR=(0x01<<LED0_PIN);
   delay_ms(100);
   }
```

```
}
```

这里主要显示 4 行字符，8*16 大小点阵，这是在初始化时默认设置的字体。

这里要主要的是，每次绘制完需要显示时都要调用 glcd_update()这个函数去更新，不然绘制的数据都是在 framebuffer 里面。

上面的例子显示效果如下：