UART 单线 LED 显示接口驱动库

技术说明书

索引

综述

BC759x 系列 LED 显示驱动+键盘接口芯片提供统一的 UART 单线 LED 显示接口，本驱动库可适用于所有该系列芯片。本驱动库可以适应串口使用中断和不使用中断两种方式。不使用中断，对应于所用 MCU 没有硬件 UART 口使用软件模拟 UART，或者虽然有硬件的 UART 口但不使用其中断功能的情况。而当 UART 接口采用中断驱动方式，驱动库的内部提供了可由用户自定义大小的 FIFO 缓冲区，串口发送在驱动库控制下在后台完成，可令显示操作所占用的处理器时间减至最少，节省更多资源供处理更重要的用户任务。

驱动库使用标准 C 语言编写，可适用于所有使用 C 语言的目标环境。同时也是轻量型的驱动库，以 Cortex-M3 目标环境为例，本库编译后仅占用约 560 个字节的程序空间和 36 个字节的 RAM(使用中断方式，FIFO 缓冲区为 8 个字节时).

BC759x 芯片每个指令由 2 个字节组成，第一个字节为指令，第二个字节为数据。驱动库提供了一个基本的操作函数 send_cmd()，可以用来向 BC759x 发送任意指令。同时，驱动库提供了几个上层函数，包装了使用中最常用的几个功能。上层函数如下:

`clear()` – 清除显示和闪烁状态

`display_dec()` – 按 10 进制显示数值

`display_hex()` – 按 16 进制显示数值

`digit_blink()` – 按数码管位控制闪烁显示

有些功能虽然很常用——比如单个 LED 的亮/灭控制，但可以用单个 BC759x 指令完成的，也不再另外提供上层函数，因为这样做并不能简化用户的使用。

函数库的安装

函数库的安装非常简单，最简单的使用方法，是将驱动库中的头文件*.h 和源文件 led_disp.c 全部拷贝到项目目录中和主程序文件放在一起，只需将 led_disp.c 加入到项目编译清单，并在用户源文件中 include led_disp.h 头文件即可。

如果希望目录结构更有条理，可以先建立项目，然后将/led_disp/目录拷贝到项目目录中，并将 led_disp.c 加入到项目的源文件清单中，只要 led_disp.h 头文件加入用户的 c 源文件，即可在用户程序中使用驱动库中的函数。（注：可能有些开发环境，还会需要将/led_disp/目录加入到项目的 include 查找目录中。）

驱动库的配置

驱动库的所有可由用户控制的配置，均在 disp_config.h 文件中。可控制的选项共有 3 项:

LOW_DIG_NUM_ON_RIGHT :

该项影响 10 进制和 16 进制显示函数的数字显示的方向。因为电路板设计的不同，数码管在电路板上排列的方式可能有所不同，如果 DIG0、DIG1 这样编号低的数码管位于一排数码管的右侧，则设置此选项为 1（默认值）；如果低编号数码管位于左侧，则此项应设为 0.

UART_MODE_INTERRUPT :

根据是否串口应用于中断方式，分别设置此项为 1(使用中断，此为默认值)和 0(不使用中断)。当使用中断时，驱动库提供了内置的串口 FIFO 缓冲区，可以让串口数据发送工作于后台，提高程序效率。

UART_FIFO_SIZE :

只有在中断模式下，此项才有意义。此项定义串口 FIFO 缓冲区的大小，数值必须是 2 的整次方，如 2, 4, 8, 16, 32 ... 等。

函数参考手册

set_write_func() 设置串口写入函数

函数声明样式:

```
void set_write_func(void (*pUartWriteFunc)(uint8_t));
```

用户必须将串口的写入操作封装为一个函数，该函数必须以一个 uint8_t 类型参数为输入，输入参数即要发送的数据。返回值必须为 void。在使用驱动库前，用户必须呼叫此函数通知驱动库哪个函数是串口写入函数。

使用中断模式时，函数完成写入 UART 寄存器后，即可返回。如果是非中断方式，写入函数必须在写入之前确认 UART 寄存器已清空或写入后待数据发送完成再返回，否则有可能令 UART 无法正常工作。

set_eni_func() 设置串口中断使能函数

函数声明样式:

```
void set_eni_func(void (*pTxIntEnFunc)(void));
```

此函数仅在使用中断方式时需要。驱动库使用 UART 中断时，需要能够控制串口中断的使能和禁止，如果使用的芯片串口的发送和接收是不同的中断，此处控制的可以是总的串口中断，也可以是串口发送中断，此串口发送中断应该是而且仅是在串口数据发送完成时发生。用户需将中断使能操作封装为函数然后通过此库函数通知驱动库呼叫哪个函数完成使能串口中断的操作。

中断使能函数必须具有 void 的输入，返回值也必须是 void.

set_disi_func() 设置串口中断禁止函数

函数声明样式:

```
void set_disi_func(void (*pTxIntDisFunc)(void));
```

此函数仅在使用中断方式时需要。驱动库使用 UART 中断时，需要能够控制串口中断的使能和禁止，如果使用的芯片串口的发送和接收是不同的中断，此处控制的可以是总的串口中断，也可以是串口发送中断，此串口发送中断应该是而且仅是在串口数据发送完成时发生。用户需将中断禁止操作封装为函数然后通过此库函数通知驱动库呼叫哪个函数完成使能串口中断的操作。

中断禁止函数必须具有 void 的输入，返回值也必须是 void.

tx_ready() 串口发送准备好

函数声明样式:

```
void tx_ready(void);
```

此函数仅在使用中断方式时需要。在中断方式下，串口在数据发送完成后会产生一个中断，此函数即从此中断的服务程序中呼叫，通知驱动库如果 FIFO 不为空可以向串口写入下一个数据。

send_cmd() 发送指令

函数声明样式:

```
void send_cmd(uint8_t Cmd, uint8_t Data);
```

这个函数可以用来向 BC759x 发送任意指令。函数库仅提供了几个上层的常用显示函数，如果用户需要对 BC759x 进行全面的控制，则需要通过调用此函数。参数有两个，Cmd 为待发送的指令，Data 为待发送的数据。led_disp.h 头文件中列出了所有 BC759x 芯片的指令的定义。关于具体 BC759x 芯片的指令的详细说明，请参阅所选用的 BC759x 芯片的技术手册。函数库所提供的所有的其它函数，均通过调用此函数实现功能。

clear() 清除显示和闪烁

函数声明样式:

```
void clear(void);
```

该函数用来清除所有的显示和闪烁属性。

display_dec() 按 10 进制显示数值

函数声明样式:

```
void display_dec(uint32_t Val, uint8_t Pos, uint8_t Width);
```

此函数将输入数值按十进制显示。只接受无符号数值，显示负值需用户自行显示‘-’号。如果高位数字超出了芯片的显示范围，超出部分将不会显示，也不会给出错误信息。

Val 为待显示数值，取值范围是 0～4,294,967,295. Pos 是显示的位置。显示位置以最低位为准，Pos 值即最低位所在的数码管 DIG 序号，取值范围为 0-31。Width 是显示宽度，低 7 位为设置显示宽度，取值范围为 1-32；最高位决定是否显示前导‘0’。当显示宽度小于实际数值的宽度时，超出部分将不予显示，而当显示宽度 Width 大于实际数值的宽度时，如果 Width 最高位为 0, 超出部分将为空白不显示任何内容，如果 Width 最高位为 1, 超出部分将显示‘0’。如实际数值为 123, 设置显示宽度 Width 为 5, 则显示结果为“␣␣123”(␣代表空白显示), 如果 Width 为 0x85，则显示结果为“00123”。

如果较高位的数字超出了实际芯片的显示范围，超出部分将被忽略。

配置文件 disp_config.h 中 LOW_DIG_NUM_ON_RIGHT 项将影响数字显示的方向。当该值为 1 时，第 Pos 位显示最低位，而显示数值的较高位将依次显示在更高序号值的 DIG 位。而当该值为 0 时，显示数值的较高位将依次显示在比 Pos 值更小的 DIG 位上。此设置可以用来适应不同的电路板设计。

display_hex() 按 16 进制显示数值

函数声明样式:

```
void display_hex(uint16_t Val, uint8_t Pos, uint8_t Width);
```

此函数将输入数值按 16 进制显示。此函数最大输入值为 0xffff，不过更大数字可以通过拆解和多次调用本函数完成显示。Pos 和 Width 的含义均与上面 display_dec()中的含义相同，所不同的一点在于，对于 16 进制显示，当 Width 设置超过实际数值的宽度时，超出部分将显示 0, 而不是十进制的空白。例如输入数值为 0xA5, 设置 Width 为 5,则显示的结果将是 000A5

display_float() 显示浮点数

函数声明样式:

```
void display_float(float Val, uint8_t Pos, uint8_t Width, uint8_t Frac);
```

此函数直接接受 float 数据类型的值 Val，并显示出来，小数点后的最低位自动四舍五入。小数点后面的位数，由 Frac 决定，Frac 取值范围为 0-7；整体的显示区域位置和宽度，由 Pos 和 Width 决定，其含义同于 display_dec()函数中的相应参数。Width 必须不小于 Frac。**本函数并不会显示小数点和负号**，有需要时用户需用额外指令完成小数点和负号的显示。这样当小数点位置固定时，用户可以使用固定电路驱动小数点常亮而将芯片输出的小数点驱动用作其它指示灯。

下面通过举例说明控制参数的用法(为了直观，例子中显示结果包含小数点):

例 1：如 Val=3.14，Width=5, Frac=4, 则显示结果为 3.1400

例 2：如 Val=3.14159, Width=7, Frac=4, 则显示结果为 ␣␣3.1416

例 3：如 Val=0.00567, Width=0x88, Frac=3, 则显示结果为 00000.006

例 4：如 Val=476.0056, Width=4, Frac=3, 则显示结果为 6.006 (高位因超出显示宽度未显示)

本函数内部实际是将待显示浮点数转换为 32 位无符号整数然后调用 display_dec()函数实现的，因此必须注意防止 32 位整数溢出的问题。如有浮点数 Val=500000.3，如果使用 Frac=4 为参数显示，则函数内部需先将数据转换为无符号整数 5000003000，但 32 位无符号整数所能表示最大值为 4,294,967,295，因此将产生溢出而无法得到正确结果。

digit_blink() 位闪烁控制

函数声明样式:

```
void digit_blink(uint8_t Digit, uint8_t OnOff);
```

按数码管位控制闪烁。此函数每次控制一个显示位的闪烁属性，第 16 位-第 31 位数码管的闪烁控制如果用户通过 send_cmd()函数直接改变了状态，再使用本函数对 16-31 位的闪烁进行控制时，通过直接命令所做的设置可能会丢失。Digit 为待设置的显示位，取值范围为 0-31；OnOff 为设置的状态，1=闪烁, 0=不闪烁。

使用方法示例

不使用中断方式

此应用做一个计数，然后将计数值按 10 进制显示。请参考以下伪代码(高亮部分为显示相关代码):

```
#include “led_disp/led_disp.h”

uint16_t Counter;

int main(void)
{
    set_write_func(write_uart);
    while(1)      // 主循环
    {
          do_something();     // 程序主任务
          …
          disp_dec(Counter, 0, 5);   // 将计数值按 10 进制显示在从第 0 位开始的位置，宽度为 5 位
          Counter++;
          delay();
    }
}

void write_uart(uint8_t TxData)
{
    TX_REG = TxData;    // 数据写入发送寄存器
    while (!Tx_Ready);  // 确认发送已经完成
}
```

使用中断方式

任务与上例相同，请参考以下伪代码(高亮部分为与不使用中断方式不同部分):

```
#include “led_disp/led_disp.h”

uint16_t Counter;

int main(void)
{
    set_write_func(write_uart);
    set_eni_func(enable_tx_interrupt);
    set_disi_func(disable_tx_interrupt);
    while(1)      // 主循环
    {
          do_something();     // 程序主任务
          …
          disp_dec(Counter, 0, 5);   // 将计数值按 10 进制显示在从第 0 位开始的位置，宽度为 5 位
          Counter++;
          delay();
    }
}

void write_uart(uint8_t TxData)
{
    while (!Tx_Ready); // 确认发送寄存器已清空
    TX_REG = TxData;    // 数据写入发送寄存器
}

void enable_tx_interrupt(void)
{
    TXIE = 1;
}

void disable_tx_interrupt(void)
{
    TXIE = 0;
}
```

```
void UART_ISR(void)      // UART 中断服务程序
{
    if (TX)        // 如果是发送中断
    {
          tx_ready();
    }
}
```