USMART V2.4 使用说明

USMART 是由 ALIENTEK 开发的一个灵巧的串口调试互交组件，通过它你可以通过串口助手调用程序里面的任何函数，并执行。因此，你可以随意更改函数的输入参数(支持数字(10/16 进制)、字符串、函数入口地址等作为参数)，单个函数最多支持 10 个输入参数，并支持函数返回值显示。V2.1 新增 hex 和 dec 两个指令。他们可以用于设置函数参数的显示格式。也可以用于数据的进制转换。

USMART 的特点如下：

1， 可以调用绝大部分用户直接编写的函数。

2， 资源占用极少（最少情况：FLASH:2.5K；SRAM:72B）。

3， 支持参数类型多（数字（包含 10/16 进制）、字符串、函数指针等）。

4， 支持函数返回值显示。

5， 支持参数及返回值格式设置。

6， 使用方便。

有了 USMART，你可以轻易的修改函数参数、查看函数运行结果，从而快速解决问题。比如你调试一个摄像头模块，需要修改其中的几个参数来得到最佳的效果，普通的做法：写函数->修改参数->下载->看结果->不满意->修改参数->下载->看结果->不满意....不停的循环，直到满意为止。这样做很麻烦不说，单片机也是有寿命的啊，老这样不停的刷，很折寿的。而利用 USMART，则只需要在串口调试助手里面输入函数及参数，然后直接串口发送给单片机，就执行了一次参数调整，不满意的话，你在串口调试助手修改参数在发送就可以了，直到你满意为止。这样，修改参数十分方便，不需要编译、不需要下载、不会让单片机折寿。

USMART 支持的参数类型基本满足任何调试了，支持的类型有：10 或者 16 进制数字、字符串指针（如果该参数是用作参数返回的话，可能会有问题！）、函数指针等。因此绝大部分函数，可以直接被 USMART 调用，对于不能直接调用的，你只需要重写一个函数，把影响调用的参数去掉即可，这个重写后的函数，即可以被 USMART 调用了。

经过以上简单介绍，我们对 USMART 有了个大概了解，接下来我们来看看 USMART 的移植和使用。

一、USMART 移植

USMART 组件总共包含 6 文件如下图所示：

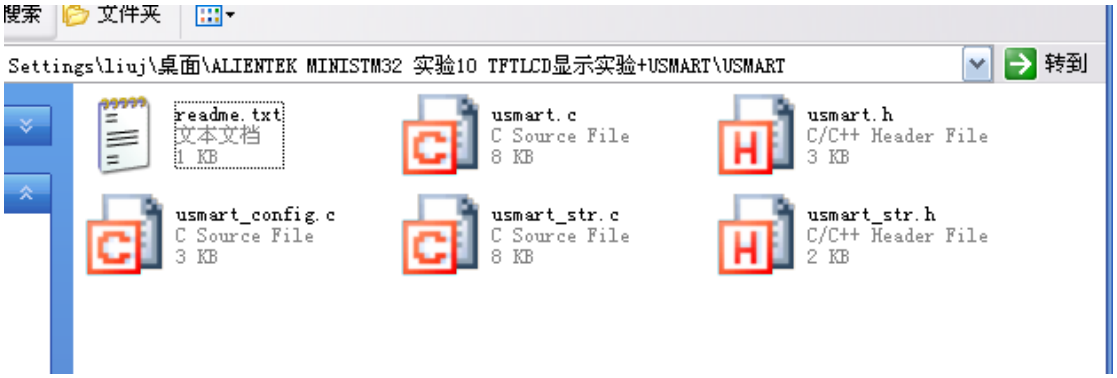


图 1.1 USMART 组件

其中 redeme.txt 是一个说明文件，不参与编译。其他五个文件，usmart.c 负责与外部互交等。usmat_str 主要负责命令和参数解析。usmart_config.c 主要由用户添加需要由 usmart 管理的函数。

usmart.h 和 usmart_str.h 是两个头文件，其中 usmart.h 里面含有几个用户配置宏定义，可以用来配置 usmart 的功能及总参数长度(直接和 SRAM 占用挂钩)。

USMART 的移植，只需要实现两个函数。两个函数都在 usmart.c 里面，第一个是 void usmart_init(void)函数，该函数主要实现串口初始化，如果用中断执行 usmart 的扫描，则可以把中断的初始化代码，也放到这个函数里面。

在 ALIENTEK STM32 开发板上该函数的实现代码如下：

```
//初始化串口控制器
void usmart_init(void)
{
    //必须使能串口中断接收
    uart_init(72,9600);     //串口 1 初始化
#if USMART_ENTIM2_SCAN==1
    Timer2_Init(1000,7199);     //7200 分频,时钟为 10K ,100ms 中断一次
#endif
    usmart_dev.sptype=1; //十六进制显示参数
}
```

上述代码，我们初始化串口波特率为 9600，根据 USMART_ENTIM2_SCAN 的值设定是否开启定时器 2，如果开启，将会每隔 100ms 执行一次 usmart 的扫描。

这里 uart_init 设置还包括开启串口中断接收，并实现利用回车键判别接收是否完成，具体的说明，请参考《STM32 不完全手册 V2.0》2.7.3 节或《例说 STM32》5.3 节。不过我们这里提供的源码 usart.c 文件已经被更新，为 1.4 版本了。主要修改了 USART_RX_STA 这个自定义寄存器，将其由 u8 变为 u16，这样我们的串口一次可以接收的字节数，最大可以是 2 的 14 次方，主要为了增加对长函数名及函数参数的支持。

Timer2 的中断函数如下：

```
//定时器 2 中断服务程序
void TIM2_IRQHandler(void)
{
```

```
    if(TIM2->SR&0X0001)//溢出中断
    {
        usmart_dev.scan();//执行 usmart 扫描
    }
    TIM2->SR&=~(1<<0);//清除中断标志位
}
```

这里主要就是调用了 usmart_dev.scan();这个函数，用于及时处理从串口接收到的数据，即上文所说的 usmart 扫描。这里采用的是中断扫描方式（推荐），你也可以使用死循环里面调用的方式（USMART_ENTIM2_SCAN 为 0），只要保证 usmart_dev.scan();函数每隔一定时间（建议不超过 200ms）被调用一次即可。

第二个要实现的函数就是 void usmart_scan(void)。该函数用于执行 usmart 扫描，该函数需要得到两个参量，第一个是从串口接收到的数组（USART_RX_BUF），第二个是串口接收状态（USART_RX_STA）。接收状态包括接收到的数组大小，以及接收是否完成。

该函数的执行过程：先判断串口接收是否完成（USART_RX_STA 的最高位是否为 1），如果完成，则取得串口接收到的数据长度（USART_RX_STA 的低 14 位），并在末尾增加结束符，再执行解析，解析完之后清空接收标记（USART_RX_STA 置零）。如果没执行完成，则直接跳过，不进行任何处理。

在 ALIENTEK STM32 开发板上该函数实现代码如下：

```
//usmart 扫描函数
//通过调用该函数,实现 usmart 的各个控制.该函数需要每隔一定时间被调用一次
//以及时执行从串口发过来的各个函数.
//本函数可以在中断里面调用,从而实现自动管理.
//如果非 ALIENTEK 用户,则 USART_RX_STA 和 USART_RX_BUF[]需要用户自己实现
void usmart_scan(void)
{
    u8 sta,len;
    if(USART_RX_STA&0x8000)//串口接收完成?
    {
        len=USART_RX_STA&0x3fff;//得到此次接收到的数据长度
        USART_RX_BUF[len]='\0';    //在末尾加入结束符.
        sta=usmart_dev.cmd_rec(USART_RX_BUF);//得到函数各个信息
        if(sta==0)usmart_dev.exe();//执行函数
        else
        {
            len=usmart_sys_cmd_exe(USART_RX_BUF);
            if(len!=USMART_FUNCERR)sta=len;
            if(sta)
            {
                switch(sta)
                {
                    case USMART_FUNCERR:
                        printf("函数错误!\r\n");
                        break;
```

```
                        case USMART_PARMERR:
                            printf("参数错误!\r\n");
                            break;
                        case USMART_PARMOVER:
                            printf("参数太多!\r\n");
                            break;
                        case USMART_NOFUNCFIND:
                            printf("未找到匹配的函数!\r\n");
                            break;
                    }
                }
            }
            USART_RX_STA=0;//状态寄存器清空
        }
    }
```

完成这两个函数的移植，你就可以使用 USMART 了。

二、USMART 使用

USMART 的使用很简单，下面结合 ALIENTEK MINISTM32 实验 10 TFTLCD 显示实验为例介绍一下 USMART 的使用(移植好的例程为: ALIENTEK MINISTM32 扩展实验 9 USMART 应用)。首先打开实验 10 的工程，然后将 usart.h 的 EN_USART1_RX 和设置为 1，使能串口中断接收。如下图所示（注意，这里我们用的是 V1.4 版本的串口驱动代码，如果你的还是旧版本的，请将 usmart 实验源码的 usart 文件夹覆盖你的 usart 文件夹）：

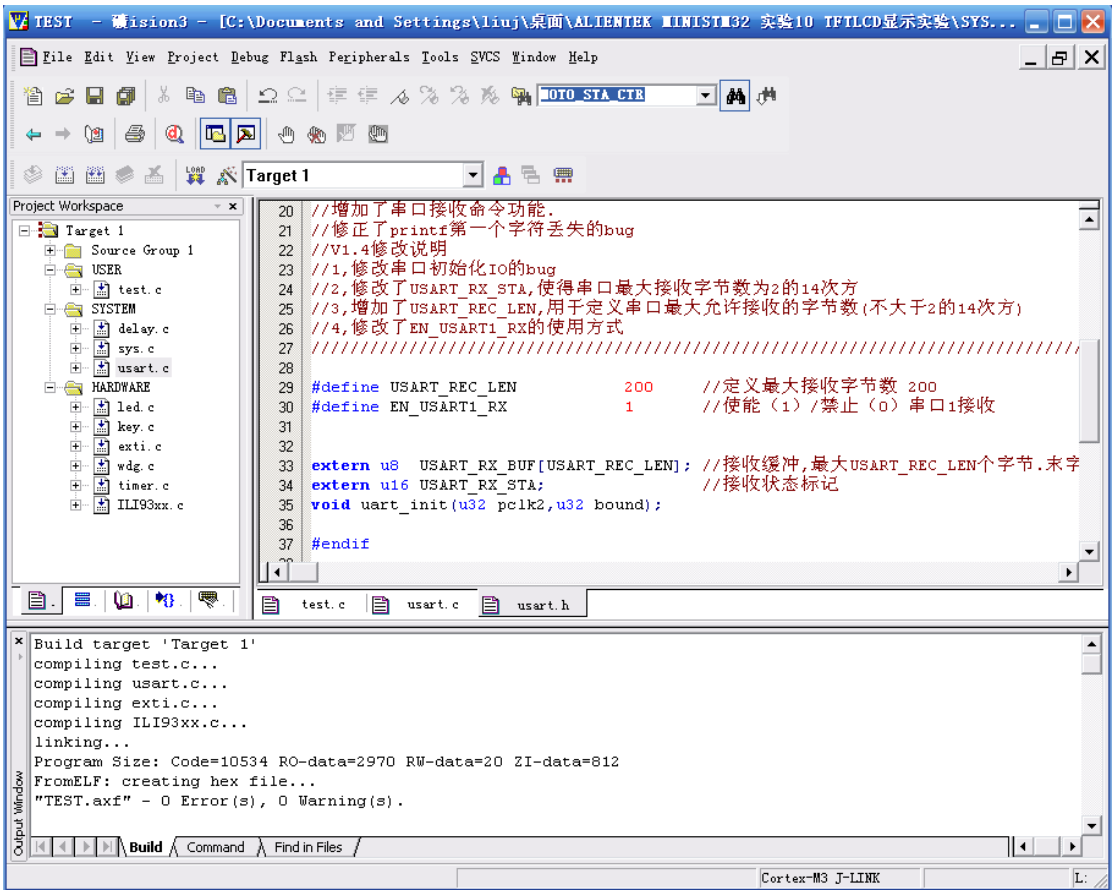


图 2.1 修改 EN_USART1_RX 为 1

然后，我们复制 USMART 文件夹到工程文件夹下面，如下图所示：

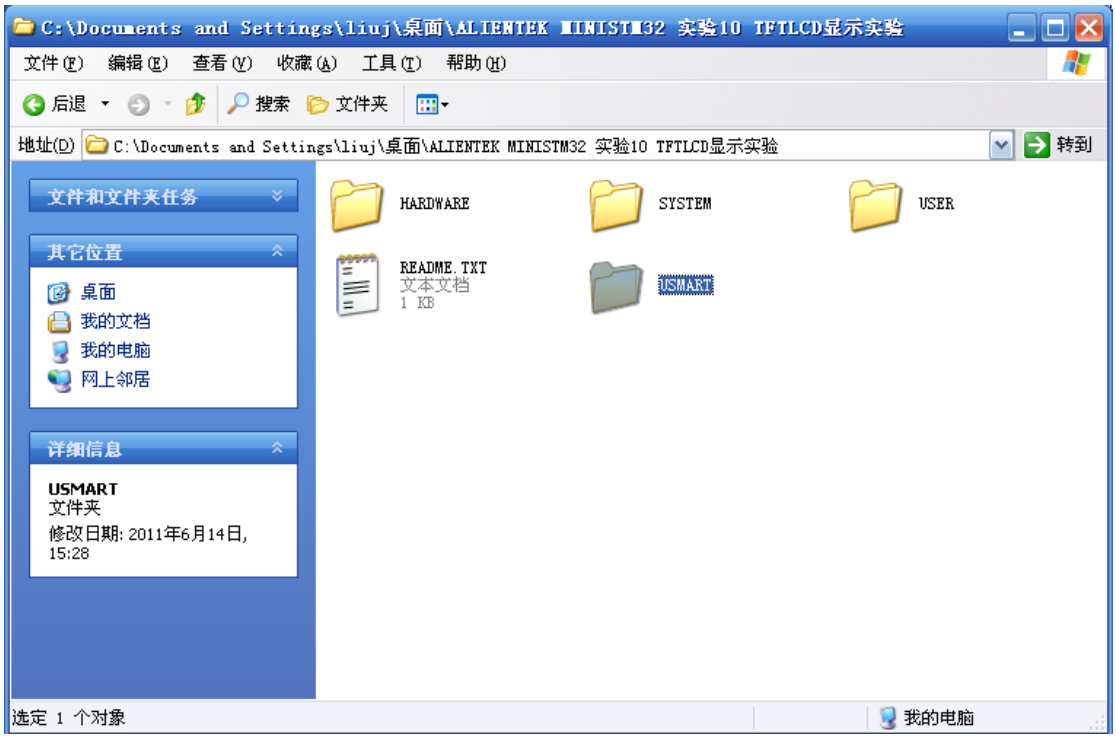


图 2.2 复制 USMART 文件夹到工程文件夹下

接着，我们在工程里面添加 USMART 组件代码，并把 USMART 文件夹添加到头文件包含路径，在主函数里面加入 include “usmart.h” 如下图所示：

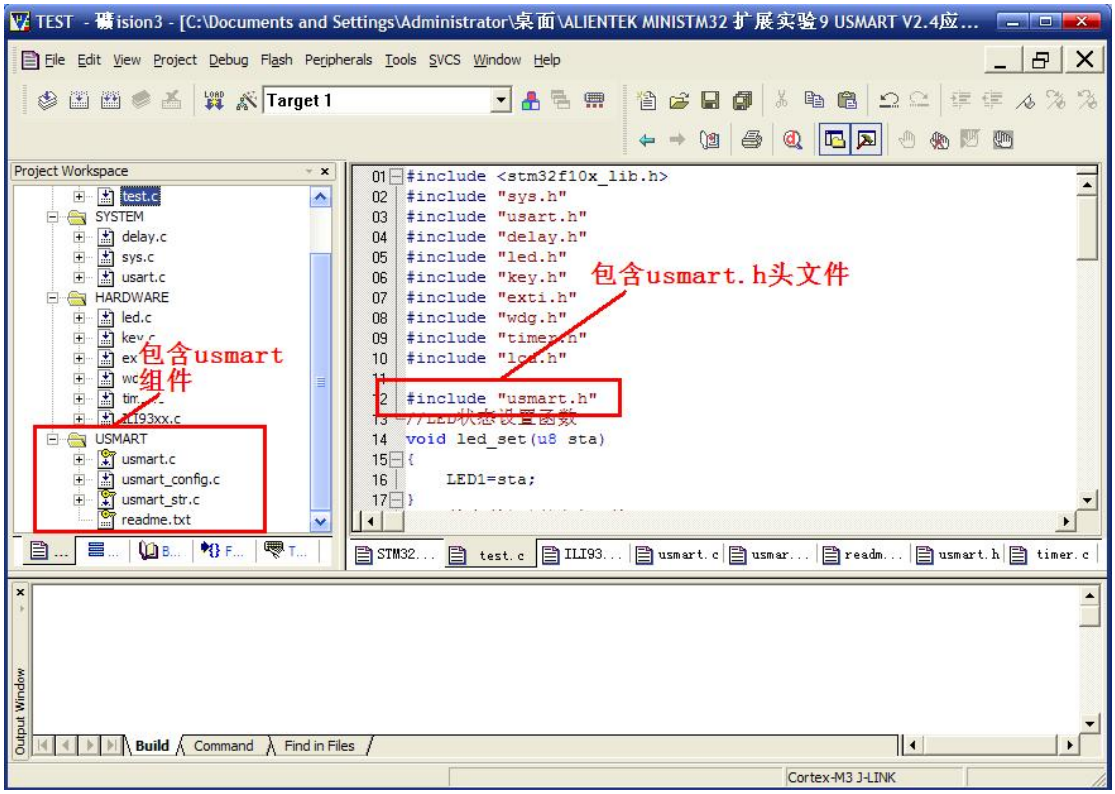


图 2.3 添加 USMART 组件代码

由于 USMART 提供了 STM32 的 TIM2 初始化设置代码，我们只需要在 usmart.h 里面设置 USMART_ENTIM2_SCAN 为 1，即可完成 TIM2 的设置。此部分代码如下：

//

```
#if USMART_ENTIM2_SCAN==1
//下面这两个函数,非 USMART 函数,放到这里,仅仅方便移植.
//定时器 2 中断服务程序
void TIM2_IRQHandler(void)
{
    if(TIM2->SR&0X0001)//溢出中断
    {
        usmart_dev.scan();//执行 usmart 扫描

    }
    TIM2->SR&=~(1<<0);//清除中断标志位
}
//使能定时器 2,使能中断.
void Timer2_Init(u16 arr,u16 psc)
{
    RCC->APB1ENR|=1<<0;//TIM2 时钟使能
    TIM2->ARR=arr;  //设定计数器自动重装值
    TIM2->PSC=psc;  //预分频器 7200,得到 10Khz 的计数时钟
    //这两个东东要同时设置才可以使用中断
    TIM2->DIER|=1<<0;    //允许更新中断
    TIM2->DIER|=1<<6;    //允许触发中断

    TIM2->CR1|=0x01;     //使能定时器 2
    MY_NVIC_Init(3,3,TIM2_IRQChannel,2);//抢占 3，子优先级 3，组 2(组 2 中优先级最低
的)
}
#endif
```

此时，我们就可以使用 USMART 了，不过在主程序里面还得执行 usmart 的初始化，另外还需要针对你自己想要被 USMART 调用的函数在 usmart_config.c 里面进行添加。下面先介绍如何添加自己想要被 USMART 调用的函数，打开 usmart_config.c，如下图所示：

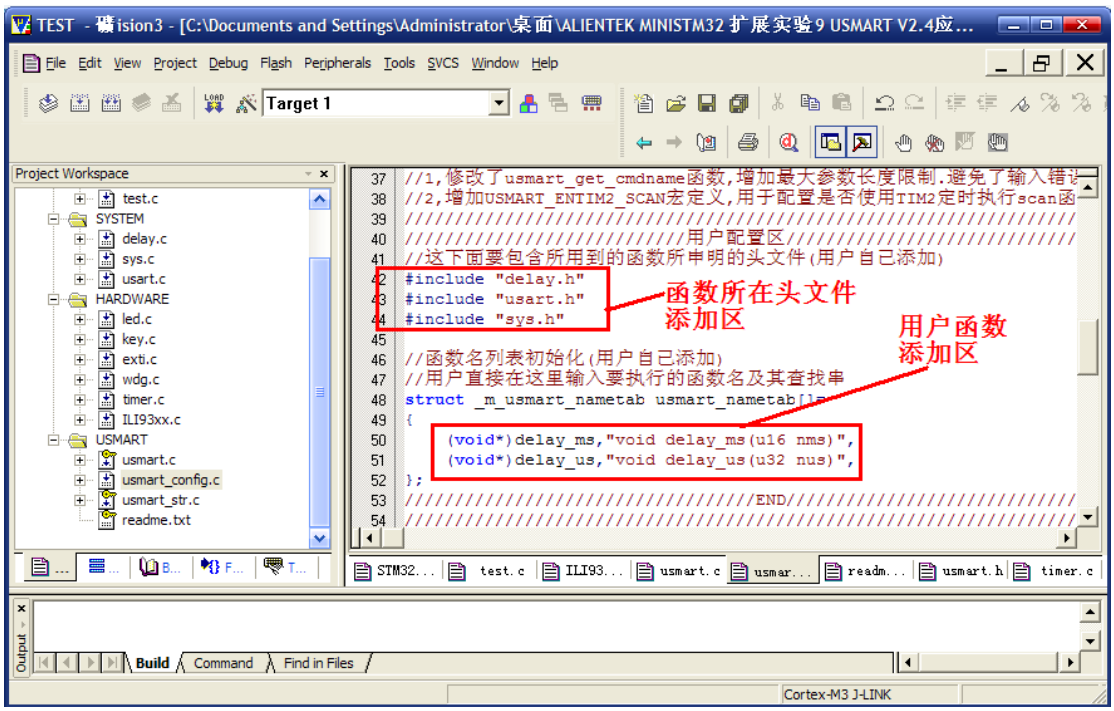


图 2.4 添加需要被 USMART 调用的函数

这里的添加函数很简单，只要把函数所在头文件添加进来，并把函数名按上图所示的方式增加即可，默认我们添加了两个函数：delay_ms 和 delay_us。这里我们根据自己的需要按上图的格式添加其他函数，添加完之后如下图所示：

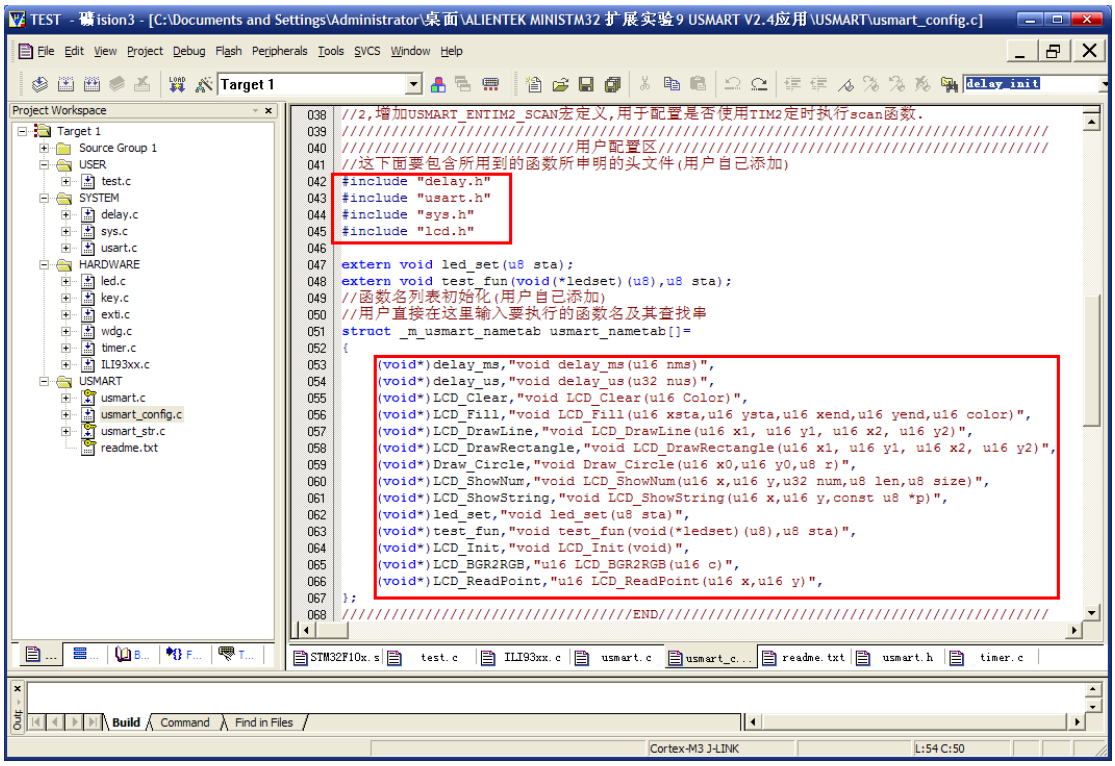


图 2.5 添加需要被 USMART 调用的函数

上图中，我们添加了 lcd.h，并添加了很多 LCD 函数，最后我们还添加了 led_set 和 test_fun 两个函数，这两个函数在 test.c 里面实现，代码如下：

//LED 状态设置函数

```
void led_set(u8 sta)
{
    LED1=sta;
}
//函数参数调用测试函数
void test_fun(void(*ledset)(u8),u8 sta)
{
    ledset(sta);
}
```

led_set 函数，用于设置 LED1 的状态，而第二个函数 test_fun 则是测试 USMART 对函数参数的支持的，test_fun 的第一个参数是函数，在 USMART 里面也是可以被调用的。

在添加完函数之后，我们修改主函数，如下：

```
//Mini STM32 开发板扩展实验
//USMART 测试 实验
//正点原子@ALIENTEK
//技术论坛:www.openedv.com
int main(void)
{
    Stm32_Clock_Init(9);//系统时钟设置
    delay_init(72);          //延时初始化
    uart_init(72,9600); //串口 1 初始化
    LED_Init();
    LCD_Init();
    usmart_dev.init();  //初始化 USMART
    POINT_COLOR=RED;
    LCD_ShowString(30,50,"Mini STM32 ^_^");
    LCD_ShowString(30,70,"USMART TEST");
    LCD_ShowString(30,90,"ATOM@ALIENTEK");
    LCD_ShowString(30,110,"2011/6/18");
    while(1)
    {
        LED0=!LED0;
        delay_ms(500);
    }
}
```

编译之后，我们下下载代码到 ALIENTEK MiniSTM32 开发板上，就可以通过串口来调用我们在图 2.5 中所添加的函数了。下面简单介绍一下，下载完代码，我们可以看到 DS0 不停闪烁，然后屏幕上显示了一些字符（就是主函数里面要显示的字符）。

我们打开串口调试助手（由于我写的助手在 WIN7 上有兼容问题，这里使用丁丁的助手），选择正确的串口号，并选择发送新行（即发送回车键）选项。如下图所示（点击扩展->隐藏）：

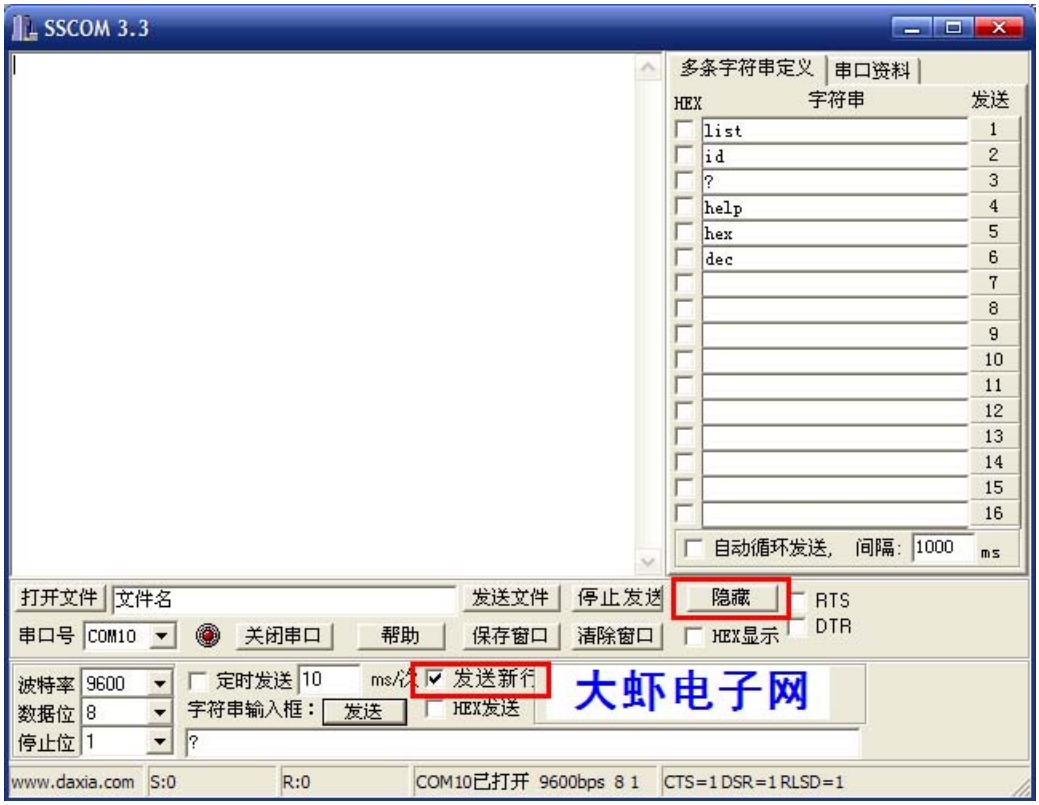


图 2.6 启动串口助手

此时我们在右边的栏里面输入各种指令或者函数，就可以得到执行结果如下图：

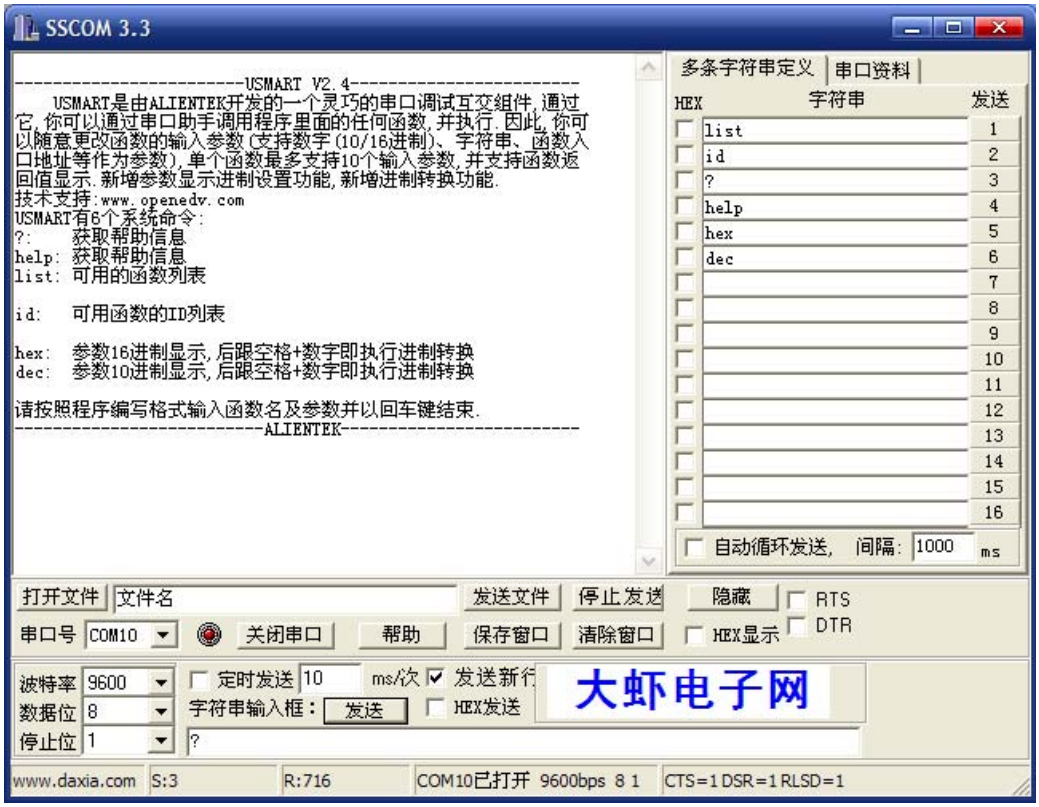


图 2.7 输入“？/help”指令

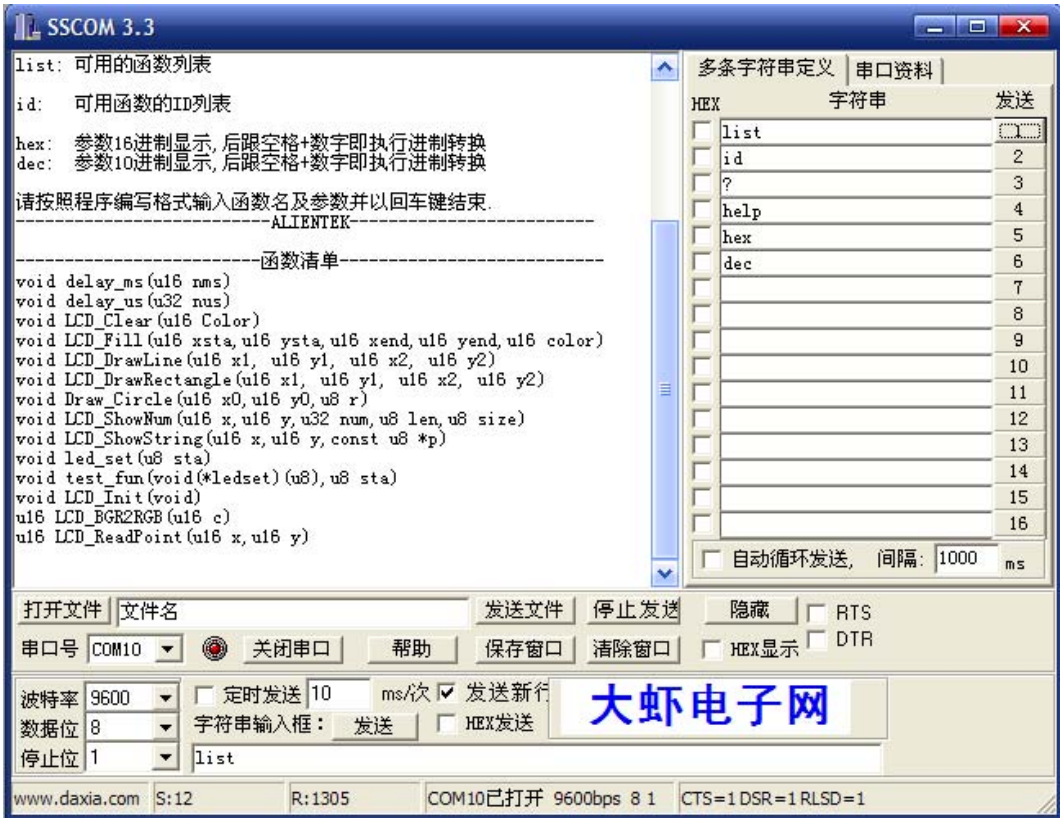


图 2.8 输入“list”指令

通过“list”指令，我们可以获得当前 USMART 所管理的全部函数。

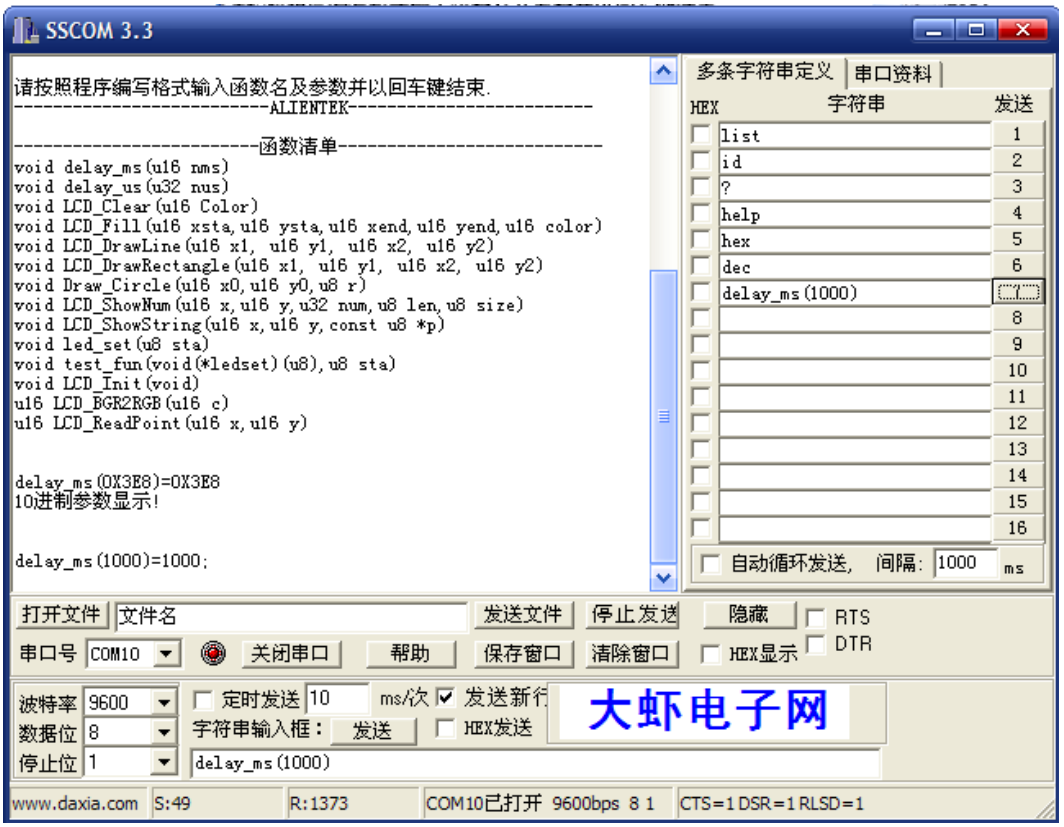


图 2.9 执行：delay_ms(1000)函数

通过输入 delay_ms(1000)函数，观察板子上的 DS0，可以看到会停 1 秒钟之后再闪烁。

该函数没有返回值，所以输出的返回值 1000 对我们来说没有意义，忽略之，下同。

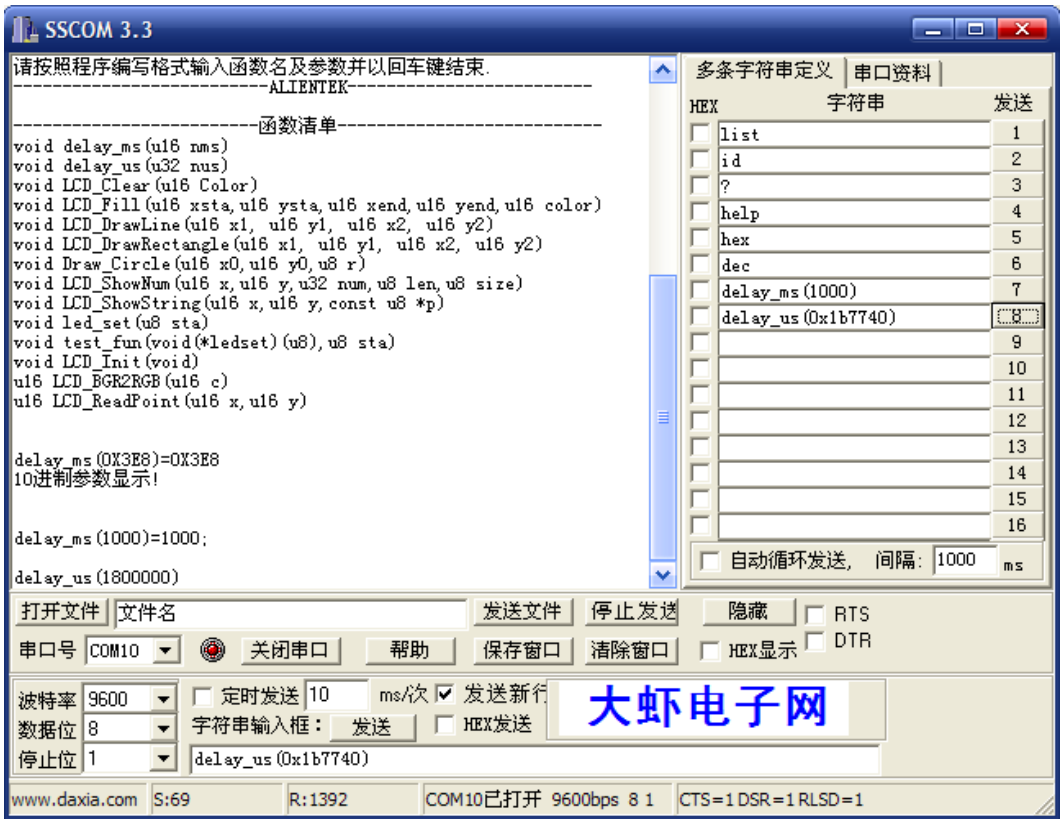


图 2.10 执行：delay_us(0x1B7740)函数

这里，我们修改输入参数为 16 进制，调用 delay_us 函数延时（注意 delay_us 同样有延时范围哦！）0x1b7740=1800000us=1800ms=1.8s，可以看到 DS0 的闪烁会停的更久一些。这里输入参数 0x1b7740，其大小写是自动转换的，你可以输入 0X1b7740 也可以输入 0x1B7740 都是一样的。

图 2.11 执行：LCD_Clear(0X7d7C)函数

可以看到整个 LCD 颜色变为了浅蓝色。

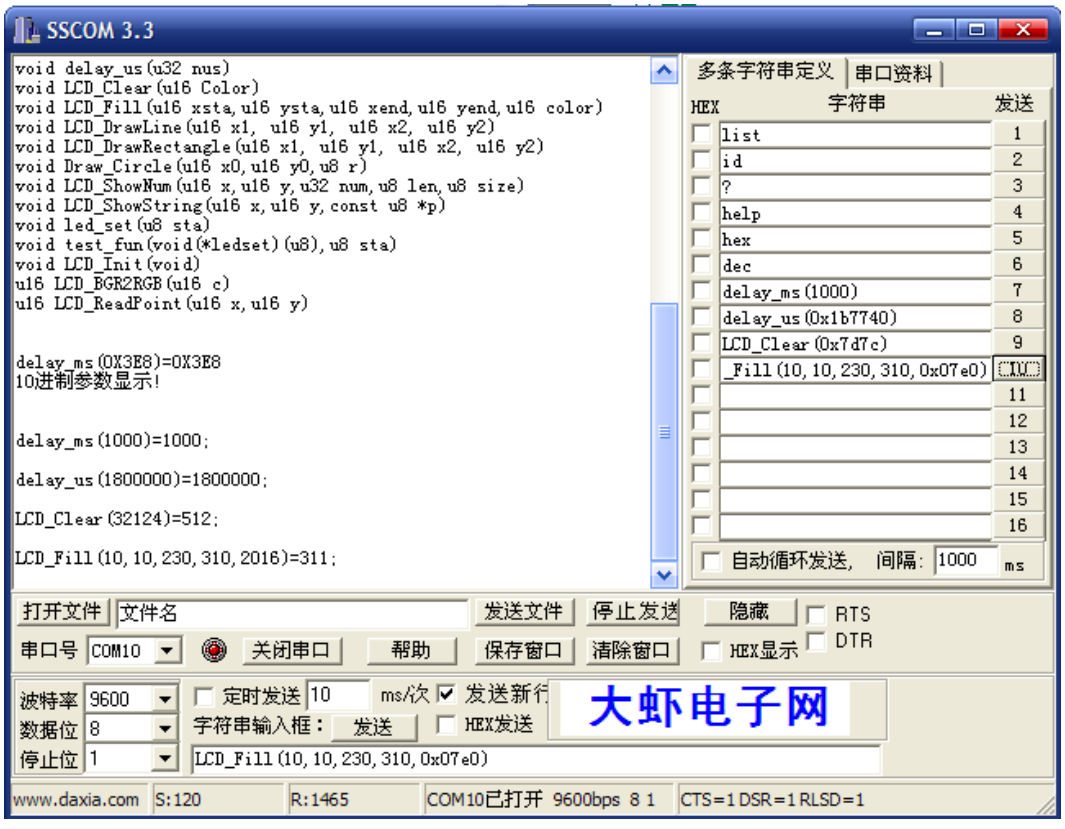


图 2.12 执行：LCD_Fill(10,10,230,310,0x07E0)函数

可以看到 LCD 内出现一个绿色的正方形。

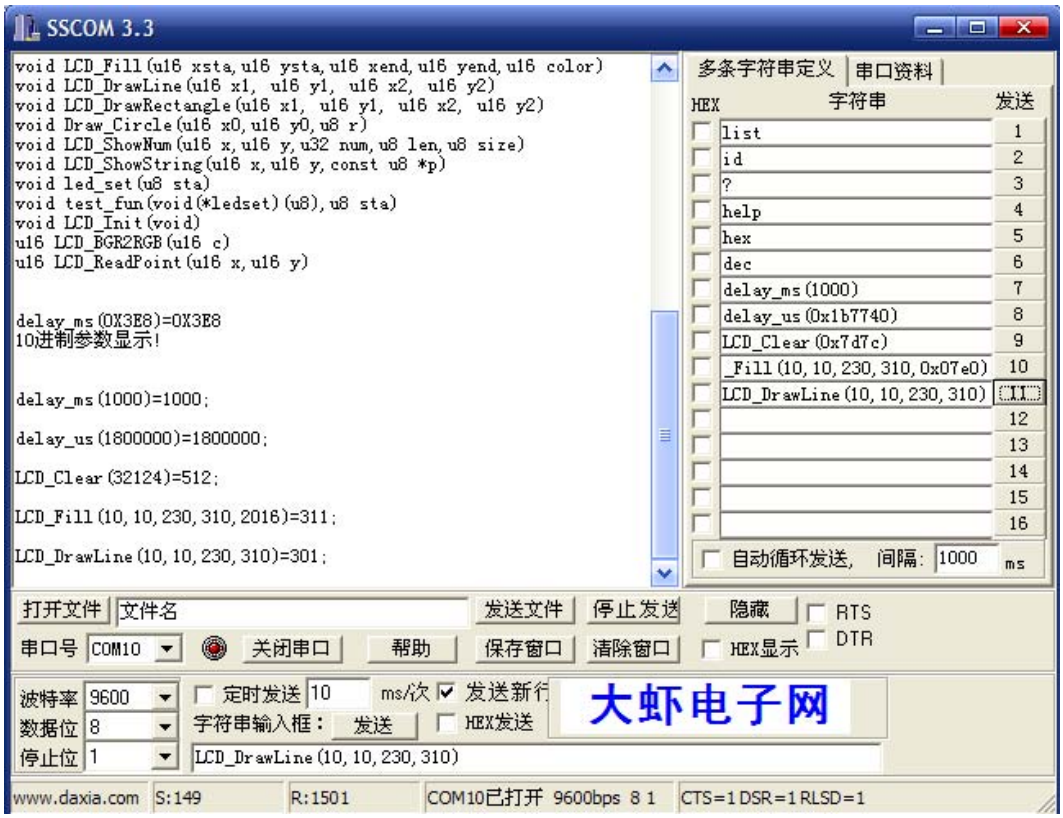


图 2.13 执行：LCD_DrawLine(10,10,230,310)函数

可以看到绿色正方形内出现一条红色的对角线。

图 2.14 执行：LCD_DrawRectangle(10,10,230,310)函数

可以看到绿色正方形边框变为红色的了。

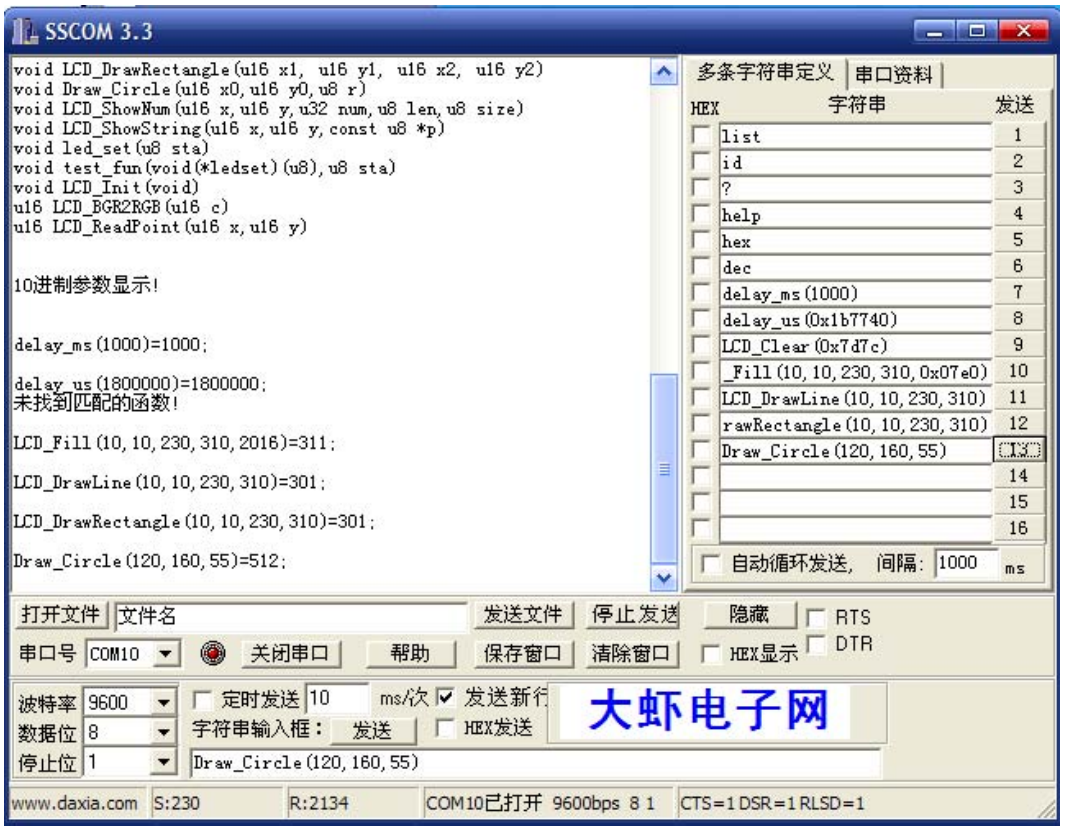


图 2.15 执行：Draw_Circle(120,160,55)函数

可以看到绿色正方形内出现了一个红色的圆圈。

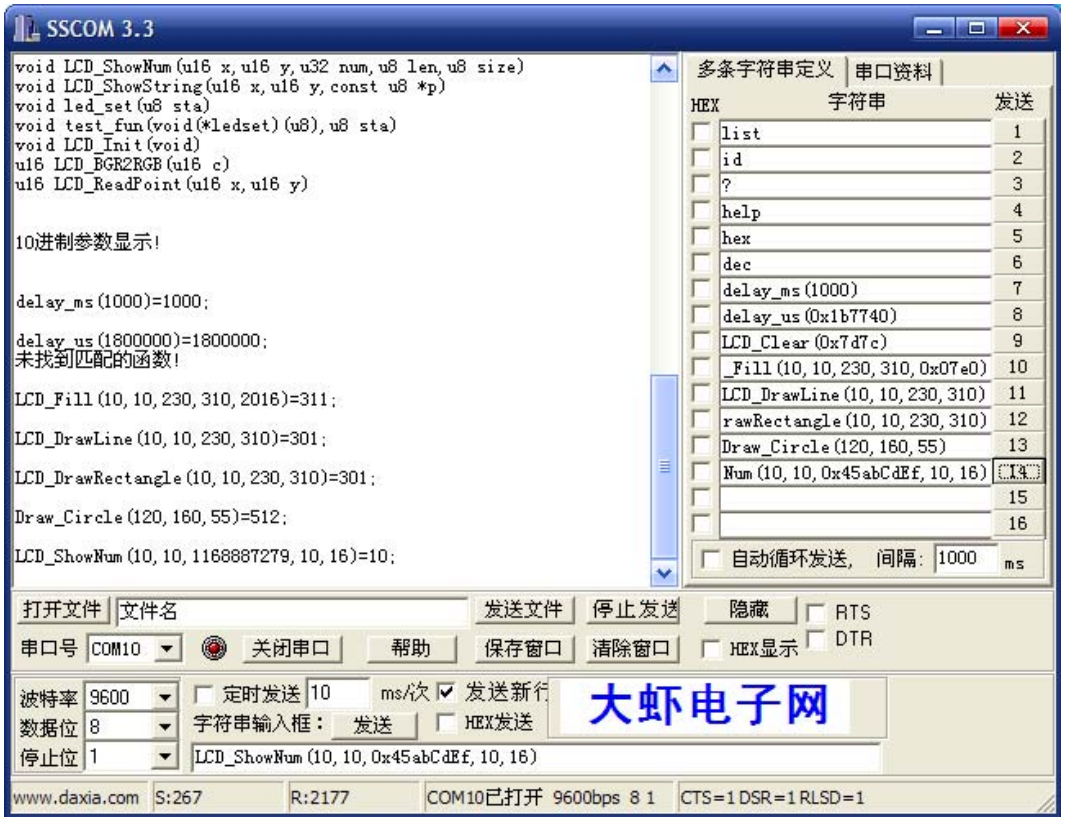


图 2.16 执行：LCD_ShowNum(10,10,0x45abCdEf,10,16)函数

可以看到在 LCD 的绿色区域内显示了 1168887279（与 0X45ABCDEF 相等）。

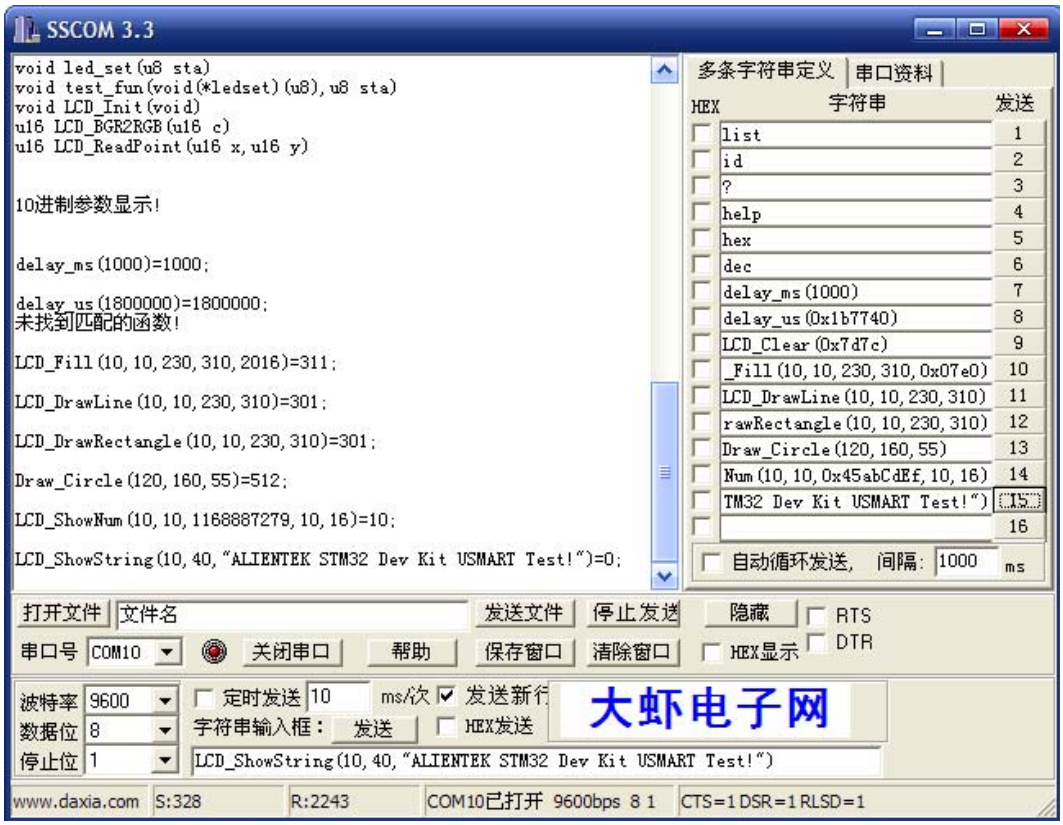


图 2.17 执行：LCD_ShowString(10,40,"ALIENTEK STM32 Dev Kit USMART Test!")函数

可以看到在 LCD 的屏幕对应位置显示了 ALIENTEK STM32 Dev Kit USMART Test!字样。

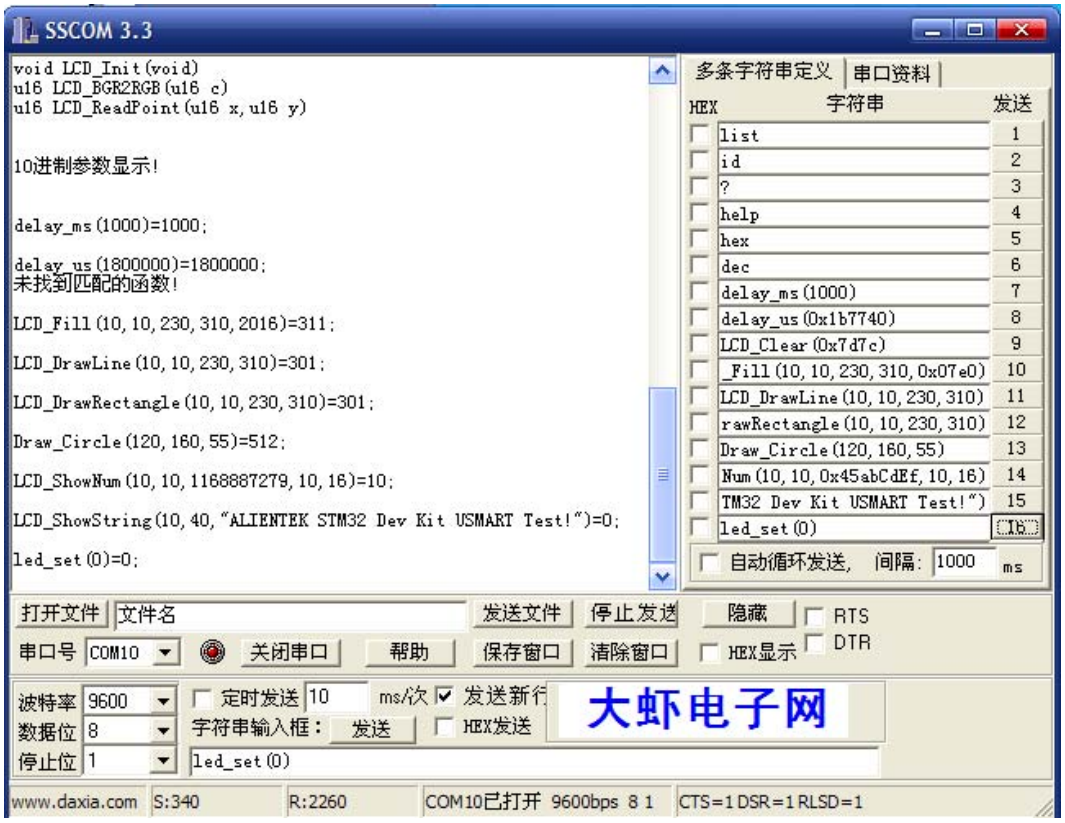


图 2.18 执行：led_set(0)函数

可以看到 DS1 点亮了。

最后一个函数，其参数为函数指针，需要我们先获得函数的地址，这里通过 id 指令获得，发送 id 指令，得到所有函数的 id，如下图：

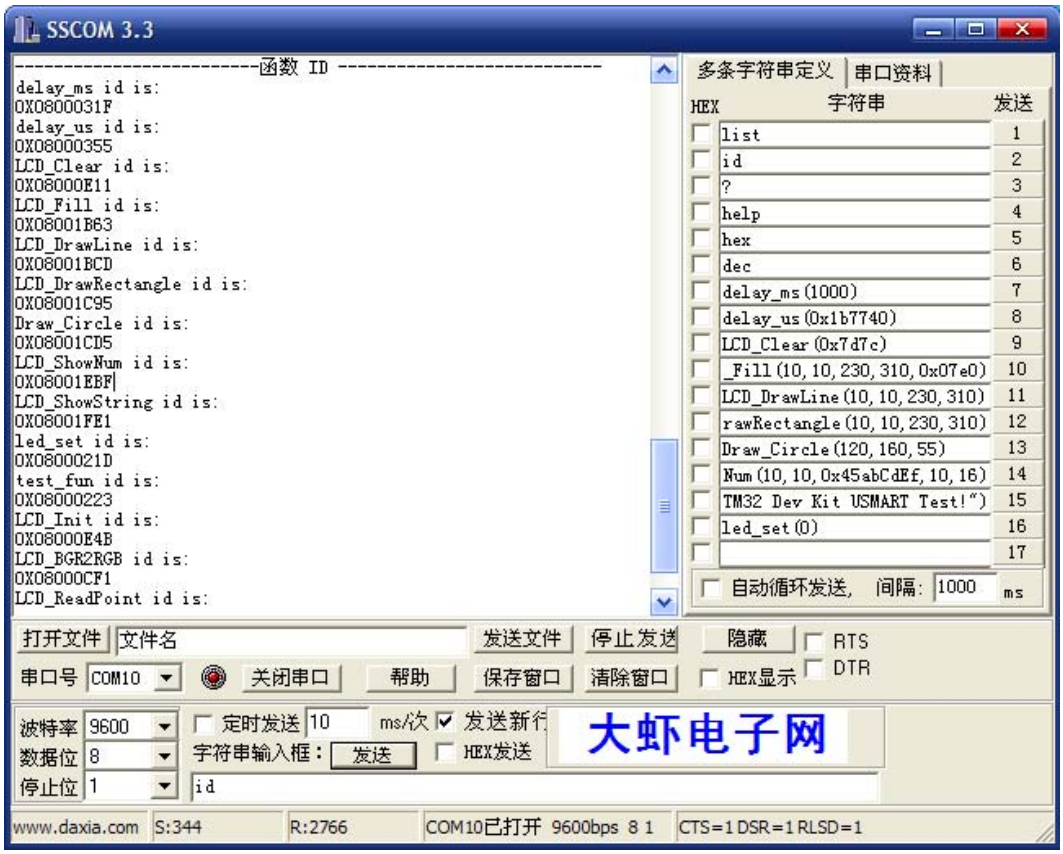


图 2.19 执行“id”指令

由于 test_fun 的函数指针我们设计的时候考虑的是 led_set 函数，所以，我们从上面的列表中选择 led_set 的 ID，为 0x0800021d。再作为 test_fun 的函数参数输入，发送给 ALIENTEK MiniSTM32 开发板，得到如下结果：

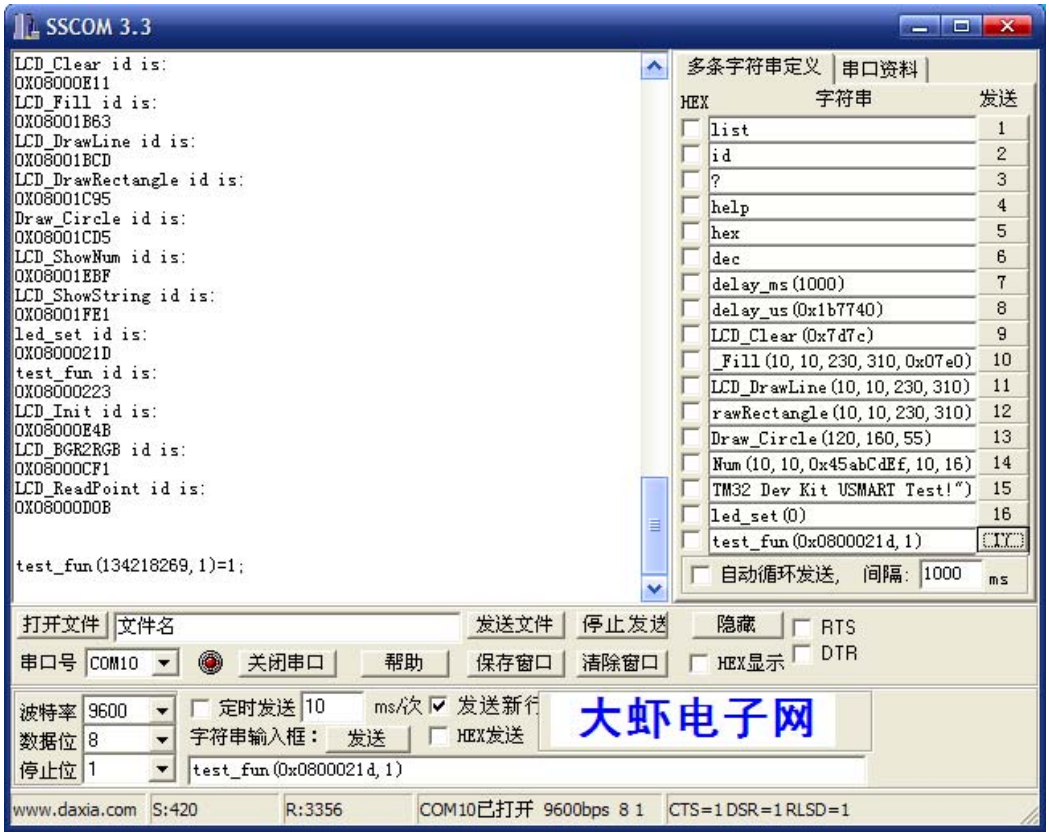


图 2.20 执行：test_fun(0x0800021d,1)函数

此时可以看到 DS1 灯灭了。说明我们确实通过 test_fun 调用了 led_set，并设置了 LED1=1；这里注意函数参数 ID 一定不能错，如果错了，可能导致程序崩溃！

最后，我们介绍一下 hex 和 dec 这两个系统命令的用法，当使用 hex 和 dec 不带参数时，用于设置函数的参数及返回值的显示格式。比如发送 hex，然后再发送 delay_ms(1000)，我们可以看到如下结果：

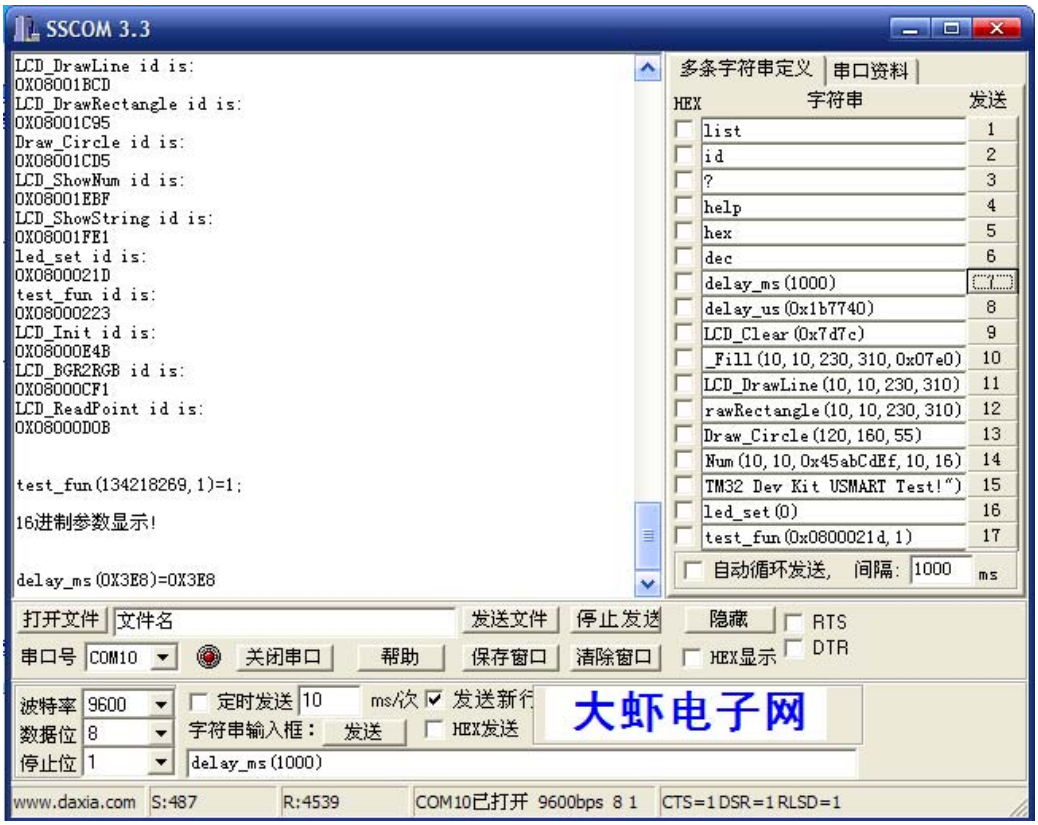


图 2.21 hex 方式显示参数

可以看到参数及返回值的显示都变为 16 进制显示了，如果再输入 dec，则可以让显示格式变为 10 进制，如下图所示：

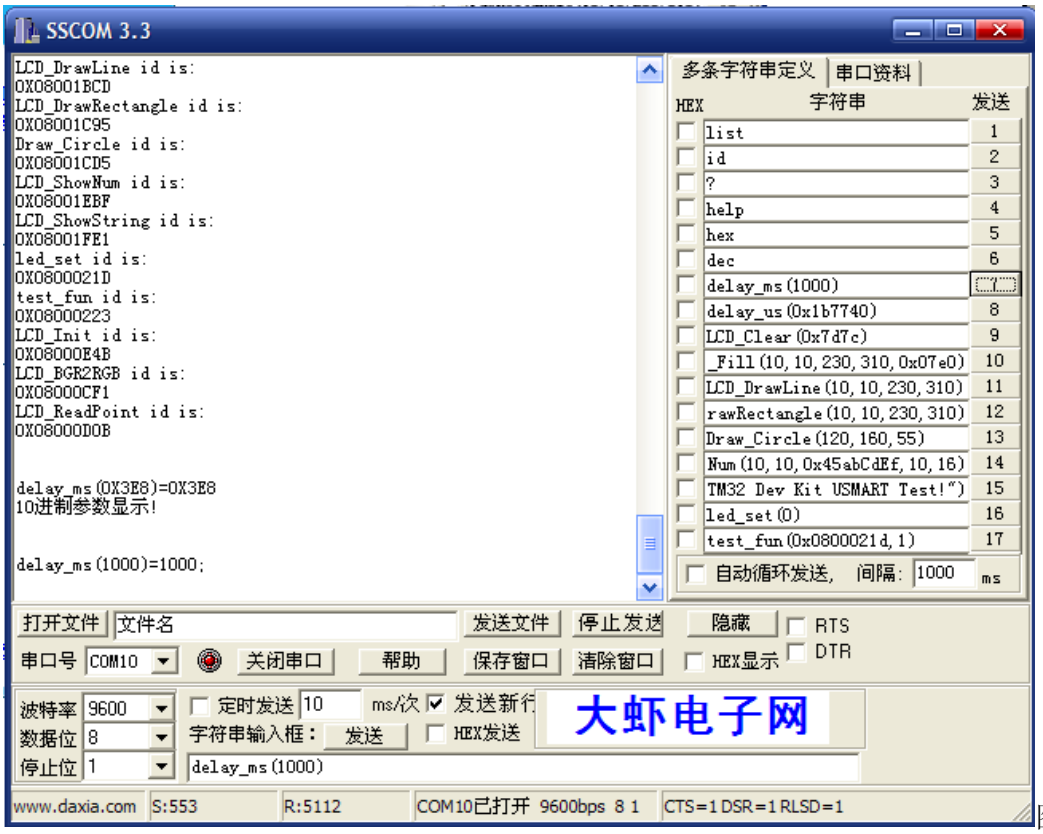
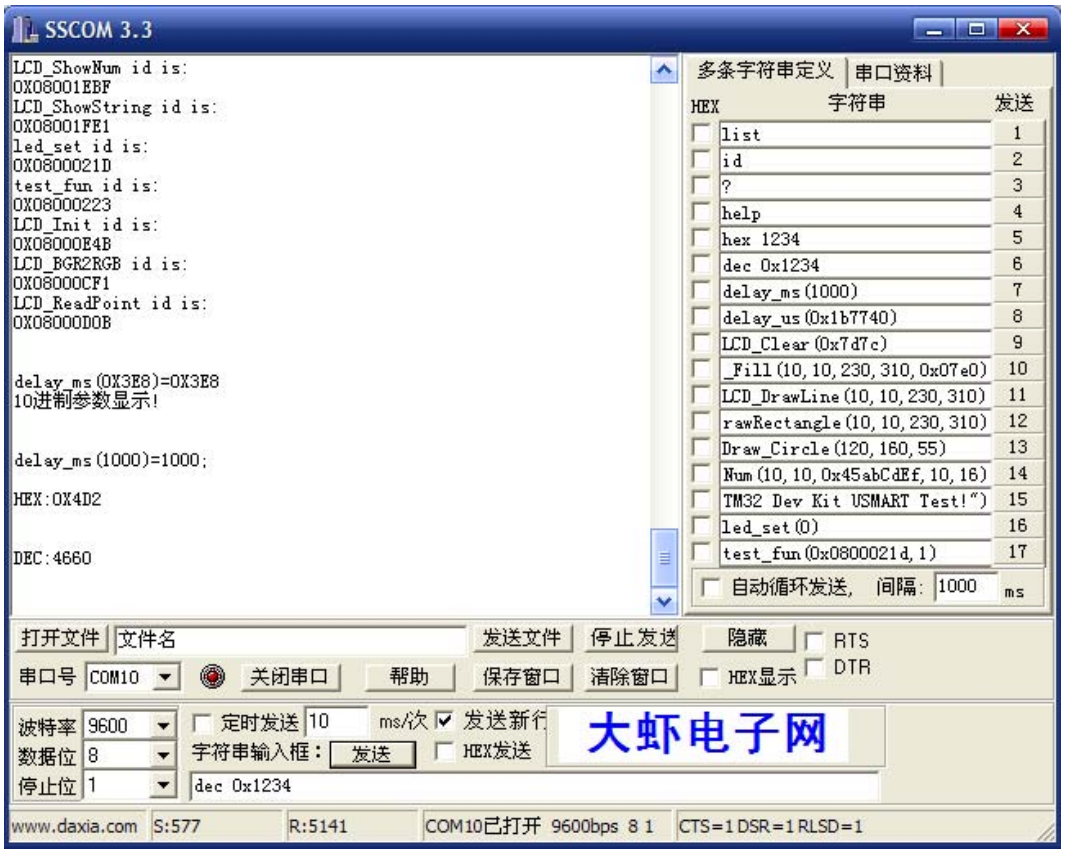


图 2.22 dec 方式显示参数

如果在 dec 和 hex 后加空格再带参数的话，就可以执行进制转换。比如分别输入：“hex 1234”和“dec 1234”就可以得到 HEX:0X4D2 和 DEC:4660，如下图所示：



2.23 hex 和 dec 的进制转换

通过 dec 和 hex 两个指令，就可以方便快速的进行进制转换，调试起来更加方便。

至此整个 USMART 的使用就介绍完了。通过以上实例，我们可以看出 USMART 的使用时非常简单的，我相信如果你真的学会了 usmart 的使用，一定会给你的学习/工作带来很大的方便。

三、USMART 注意事项

1，USMART 的函数在函数名与'('之间不要留空格，否则导致无法匹配。比如 delay_ms(1000)，不要弄成 delay_ms (1000)。

2，在参数为函数指针的时候，其函数 ID 一定不要搞错，否则可能导致程序崩溃。

3，usmart.h 里面，PARM_LEN 的值最少应该为 4，他是用来设定保存函数参数的数组大小的。PARM_LEN 的值将直接影响到 USMART 组件的 SRAM 占用率。计算公式为：sram=PARM_LEN+72-4。当 PARM_LEN 设置为 4 的时候，组件只占用 72 个字节。但此时的参数长度则限制在 4 个字节。PARM_LEN 的值，大家自己根据需要修改。

TO BE ADDED

正点原子@ALIENTEK
2011-6-14