第四十一章 外部 SRAM 实验

STM32F407ZGT6 自带了 192K 字节的 SRAM，对一般应用来说，已经足够了，不过在一些对内存要求高的场合，STM32F4 自带的这些内存就不够用了。比如跑算法或者跑 GUI 等，就可能不太够用，所以探索者 STM32F4 开发板板载了一颗 1M 字节容量的 SRAM 芯片：IS62WV51216，满足大内存使用的需求。

本章，我们将使用 STM32F4 来驱动 IS62WV51216，实现对 IS62WV51216 的访问控制，并测试其容量。本章分为如下几个部分：

41.1 IS62WV51216 简介
41.2 硬件设计
41.3 软件设计
41.4 下载验证

41.1 IS62WV51216 简介

IS62WV51216 是 ISSI（Integrated Silicon Solution, Inc）公司生产的一颗 16 位宽 512K（512*16，即 1M 字节）容量的 CMOS 静态内存芯片。该芯片具有如下几个特点：

- 高速。具有 45ns/55ns 访问速度。
- 低功耗。
- TTL 电平兼容。
- 全静态操作。不需要刷新和时钟电路。
- 三态输出。
- 字节控制功能。支持高/低字节控制。

IS62WV51216 的功能框图如图 41.1.1 所示：

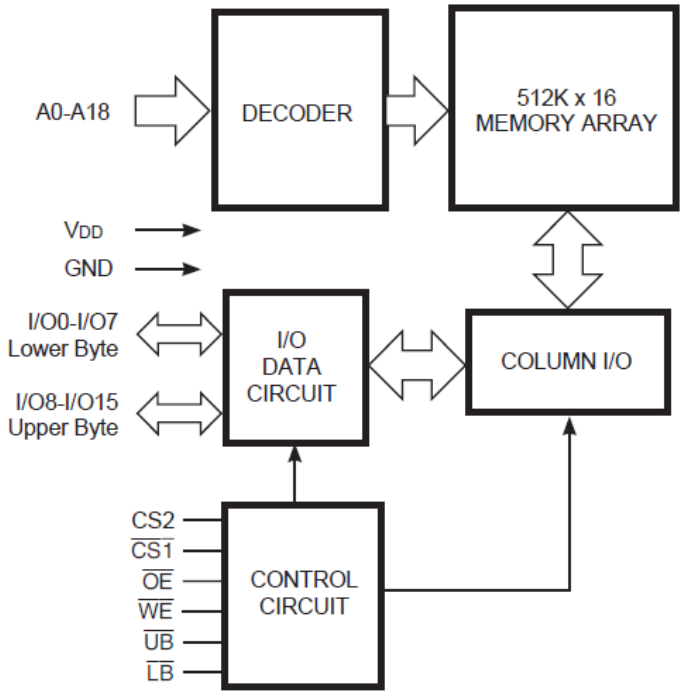

图 41.1.1 IS62WV51216 功能框图

图中 A0~18 为地址线，总共 19 根地址线（即 2^19=512K，1K=1024）；IO0~15 为数据线，

总共 16 根数据线。CS2 和 CS1 都是片选信号，不过 CS2 是高电平有效 CS1 是低电平有效；OE 是输出使能信号（读信号）；WE 为写使能信号；UB 和 LB 分别是高字节控制和低字节控制信号；

探索者 STM32F4 开发板使用的是 TSOP44 封装的 IS62WV51216 芯片，该芯片直接接在 STM32F4 的 FSMC 上，IS62WV51216 原理图如图 41.1.2 所示：

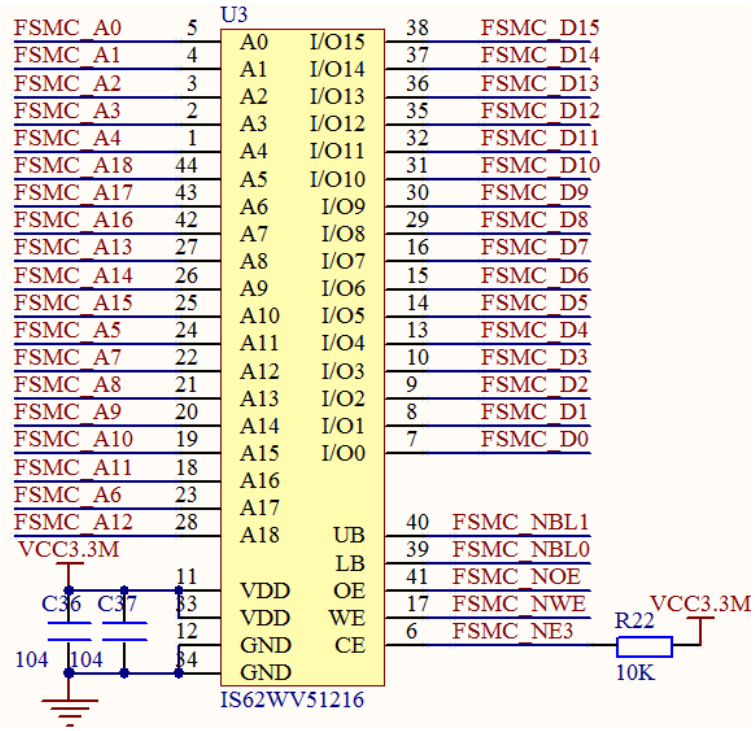

图 41.1.2 IS62WV51216 原理图

从原理图可以看出，IS62WV51216 同 STM32F4 的连接关系：

A[0:18]接 FMSC_A[0:18]（不过顺序错乱了）

D[0:15]接 FSMC_D[0:15]

UB 接 FSMC_NBL1

LB 接 FSMC_NBL0

OE 接 FSMC_OE

WE 接 FSMC_WE

CS 接 FSMC_NE3

上面的连接关系，IS62WV51216 的 A[0:18]并不是按顺序连接 STM32F4 的 FMSC_A[0:18]，不过这并不影响我们正常使用外部 SRAM，因为地址具有唯一性。所以，只要地址线不和数据线混淆，就可以正常使用外部 SRAM。这样设计的好处，就是可以方便我们的 PCB 布线。

本章，我们使用 FSMC 的 BANK1 区域 3 来控制 IS62WV51216，关于 FSMC 的详细介绍，我们在第十八章已经介绍过，在第十八章，我们采用的是读写不同的时序来操作 TFTLCD 模块（因为 TFTLCD 模块读的速度比写的速度慢很多），但是在本章，因为 IS62WV51216 的读写时间基本一致，所以，我们设置读写相同的时序来访问 FSMC。关于 FSMC 的详细介绍，请大家看第十八章和《STM32F4xx 中文参考手册》。

IS62WV51216 就介绍到这，最后，我们来看看实现 IS62WV51216 的访问，需要对 FSMC 进行哪些配置。FSMC 的详细配置介绍在之前的 LCD 实验章节已经有详细讲解，这里就做一个概括性的讲解。步骤如下：

1）使能 FSMC 时钟，并配置 FSMC 相关的 IO 及其时钟使能。

要使用 FSMC，当然首先得开启其时钟。然后需要把 FSMC_D0~15，FSMCA0~18 等相关 IO 口，全部配置为复用输出，并使能各 IO 组的时钟。

使能 FSMC 时钟的方法前面 LCD 实验已经讲解过，方法为：

```
RCC_AHB3PeriphClockCmd(RCC_AHB3Periph_FSMC,ENABLE);//使能 FSMC 时钟
```

配置 IO 口为复用输出的关键行代码为：

```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;//复用输出
```

关于引脚复用映射配置，这在 LCD 实验章节也讲解非常详细，调用函数为：

```
void GPIO_PinAFConfig(GPIO_TypeDef* GPIOx, uint16_t GPIO_PinSource, uint8_t GPIO_AF);
```

针对每个复用引脚调用这个函数即可,例如 GPIOD.0 引脚复用映射配置方法为：

```
GPIO_PinAFConfig(GPIOD,GPIO_PinSource0,GPIO_AF_FSMC);//PD0,AF12
```

2）设置 FSMC BANK1 区域 3 的相关寄存器。

此部分包括设置区域 3 的存储器的工作模式、位宽和读写时序等。本章我们使用模式 A、16 位宽，读写共用一个时序寄存器。这个是通过调用函数 FSMC_NORSRAMInit 来实现的，函数原型为：

```
void FSMC_NORSRAMInit(FSMC_NORSRAMInitTypeDef* FSMC_NORSRAMInitStruct);
```

3）使能 BANK1 区域 3。

最后，只需要通过 FSMC_BCR 寄存器使能 BANK1，区域 3 即可。使能方法为：

```
FSMC_NORSRAMCmd(FSMC_Bank1_NORSRAM3, ENABLE);  // 使能 BANK3
```

通过以上几个步骤，我们就完成了 FSMC 的配置，可以访问 IS62WV51216 了，这里还需要注意，因为我们使用的是 BANK1 的区域 3，所以 HADDR[27:26]=10，故外部内存的首地址为 0X68000000。

41.2 硬件设计

本章实验功能简介：开机后，显示提示信息，然后按下 KEY0 按键，即测试外部 SRAM 容量大小并显示在 LCD 上。按下 KEY1 按键，即显示预存在外部 SRAM 的数据。DS0 指示程序运行状态。

本实验用到的硬件资源有：

1） 指示灯 DS0

2） KEY0 和 KEY1 按键

3） 串口

4） TFTLCD 模块

5） IS62WV51216

这些我们都已经介绍过（IS62WV51216 与 STM32F4 的各 IO 对应关系，请参考光盘原理图），接下来我们开始软件设计。

41.3 软件设计

打开外部 SRAM 实验工程，可以看到，我们增加了 sram.c 文件以及头文件 sram.h，FSMC 初始化相关配置和定义都在这两个文件中。同时还引入了 FSMC 固件库文件 stm32f4xx_fsmc.c 和 stm32f4xx_fsmc.h 文件。

打开 sram.c 文件，代码如下：

```
//使用 NOR/SRAM 的  Bank1.sector3,地址位 HADDR[27,26]=10
//对 IS61LV25616/IS62WV25616,地址线范围为 A0~A17
```

```
//对 IS61LV51216/IS62WV51216,地址线范围为 A0~A18
#define Bank1_SRAM3_ADDR       ((u32)(0x68000000))
//初始化外部 SRAM
void FSMC_SRAM_Init(void)
{
   GPIO_InitTypeDef    GPIO_InitStructure;
   FSMC_NORSRAMInitTypeDef    FSMC_NORSRAMInitStructure;
   FSMC_NORSRAMTimingInitTypeDef    readWriteTiming;

   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB|RCC_AHB1Periph_GPIOD|
            RCC_AHB1Periph_GPIOE|RCC_AHB1Periph_GPIOF|RCC_AHB1Periph_GPIOG,
                                                    ENABLE);//使能 PD,PE,PF,PG 时钟
   RCC_AHB3PeriphClockCmd(RCC_AHB3Periph_FSMC,ENABLE);//使能 FSMC 时钟

   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15;//PB15  推挽输出,控制背光
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;//普通输出模式
   GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;//推挽输出
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//100MHz
   GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;//上拉
   GPIO_Init(GPIOB, &GPIO_InitStructure);//初始化  //PB15  推挽输出,控制背光

   GPIO_InitStructure.GPIO_Pin = (3<<0)|(3<<4)|(0XFF<<8);//PD0,1,4,5,8~15 AF OUT
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;//复用输出
   GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;//推挽输出
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;//100MHz
   GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;//上拉
   GPIO_Init(GPIOD, &GPIO_InitStructure);//初始化

   ……//省略部分 GPIO 初始化设置

   GPIO_PinAFConfig(GPIOD,GPIO_PinSource0,GPIO_AF_FSMC);//PD0,AF12
   GPIO_PinAFConfig(GPIOD,GPIO_PinSource1,GPIO_AF_FSMC);//PD1,AF12
   ……//省略部分 GPIO AF 映射设置
   GPIO_PinAFConfig(GPIOG,GPIO_PinSource5,GPIO_AF_FSMC);
   GPIO_PinAFConfig(GPIOG,GPIO_PinSource10,GPIO_AF_FSMC);


   readWriteTiming.FSMC_AddressSetupTime = 0x00;       //地址建立时间为 1 个 HCLK
   readWriteTiming.FSMC_AddressHoldTime = 0x00;  //地址保持时间模式 A 未用到
   readWriteTiming.FSMC_DataSetupTime = 0x08;    //数据保持时间为 9 个 HCLK
   readWriteTiming.FSMC_BusTurnAroundDuration = 0x00;
   readWriteTiming.FSMC_CLKDivision = 0x00;
   readWriteTiming.FSMC_DataLatency = 0x00;
```

```
    readWriteTiming.FSMC_AccessMode = FSMC_AccessMode_A;    //模式 A

    FSMC_NORSRAMInitStructure.FSMC_Bank = FSMC_Bank1_NORSRAM3;// NE3
    FSMC_NORSRAMInitStructure.FSMC_DataAddressMux =
                                                    FSMC_DataAddressMux_Disable;
    FSMC_NORSRAMInitStructure.FSMC_MemoryType=FSMC_MemoryType_SRAM;
    FSMC_NORSRAMInitStructure.FSMC_MemoryDataWidth =
                                    FSMC_MemoryDataWidth_16b;//存储器数据宽度为 16bit
    FSMC_NORSRAMInitStructure.FSMC_BurstAccessMode =
                                    FSMC_BurstAccessMode_Disable;
    FSMC_NORSRAMInitStructure.FSMC_WaitSignalPolarity =
                                    FSMC_WaitSignalPolarity_Low;
    FSMC_NORSRAMInitStructure.FSMC_AsynchronousWait=
                                    FSMC_AsynchronousWait_Disable;
    FSMC_NORSRAMInitStructure.FSMC_WrapMode = FSMC_WrapMode_Disable;
    FSMC_NORSRAMInitStructure.FSMC_WaitSignalActive =
                                    FSMC_WaitSignalActive_BeforeWaitState;
    FSMC_NORSRAMInitStructure.FSMC_WriteOperation =
                                    FSMC_WriteOperation_Enable;//存储器写使能
    FSMC_NORSRAMInitStructure.FSMC_WaitSignal = FSMC_WaitSignal_Disable;
    FSMC_NORSRAMInitStructure.FSMC_ExtendedMode =
                                    FSMC_ExtendedMode_Disable; //  读写使用相同的时序
    FSMC_NORSRAMInitStructure.FSMC_WriteBurst = FSMC_WriteBurst_Disable;
    FSMC_NORSRAMInitStructure.FSMC_ReadWriteTimingStruct = &readWriteTiming;
    FSMC_NORSRAMInitStructure.FSMC_WriteTimingStruct =
                                    &readWriteTiming; //读写同样时序

    FSMC_NORSRAMInit(&FSMC_NORSRAMInitStructure);    //初始化 FSMC 配置
    FSMC_NORSRAMCmd(FSMC_Bank1_NORSRAM3, ENABLE);    //  使能 BANK1 区域 3
}
//在指定地址(WriteAddr+Bank1_SRAM3_ADDR)开始,连续写入 n 个字节.
//pBuffer:字节指针
//WriteAddr:要写入的地址
//n:要写入的字节数
void FSMC_SRAM_WriteBuffer(u8* pBuffer,u32 WriteAddr,u32 n)
{
    for(;n!=0;n--)
    {
        *(vu8*)(Bank1_SRAM3_ADDR+WriteAddr)=*pBuffer;
        WriteAddr++;
        pBuffer++;
    }
}
```

```
//在指定地址((WriteAddr+Bank1_SRAM3_ADDR))开始,连续读出 n 个字节.
//pBuffer:字节指针
//ReadAddr:要读出的起始地址
//n:要写入的字节数
void FSMC_SRAM_ReadBuffer(u8* pBuffer,u32 ReadAddr,u32 n)
{
    for(;n!=0;n--)
    {
        *pBuffer++=*(vu8*)(Bank1_SRAM3_ADDR+ReadAddr);
        ReadAddr++;
    }
}
```

此部分代码包含 3 个函数，FSMC_SRAM_Init 函数用于初始化，包括 FSMC 相关 IO 口的初始化以及 FSMC 配置；FSMC_SRAM_WriteBuffer 和 FSMC_SRAM_ReadBuffer 这两个函数分别用于在外部 SRAM 的指定地址写入和读取指定长度的数据（字节数）。

这里需要注意的是：FSMC 当位宽为 16 位的时候，HADDR 右移一位同地址对其，但是 ReadAddr 我们这里却没有加 2，而是加 1，是因为我们这里用的数据为宽是 8 位，通过 UB 和 LB 来控制高低字节位，所以地址在这里是可以只加 1 的。另外，因为我们使用的是 BANK1，区域 3，所以外部 SRAM 的基址为：0x68000000。

头文件 sram.h 内容比较简洁，主要是一些函数申明，这里我们不做过多讲解。

最后我们来看看 main.c 文件代码如下：

```
u32 testsram[250000] __attribute__((at(0X68000000)));//测试用数组
//外部内存测试(最大支持 1M 字节内存测试)
void fsmc_sram_test(u16 x,u16 y)
{
    u32 i=0; u8 temp=0;
    u8 sval=0;      //在地址 0 读到的数据
    LCD_ShowString(x,y,239,y+16,16,"Ex Memory Test:     0KB");
    //每隔 4K 字节,写入一个数据,总共写入 256 个数据,刚好是 1M 字节
    for(i=0;i<1024*1024;i+=4096) { FSMC_SRAM_WriteBuffer(&temp,i,1); temp++;}
    //依次读出之前写入的数据,进行校验
    for(i=0;i<1024*1024;i+=4096)
    {
        FSMC_SRAM_ReadBuffer(&temp,i,1);
        if(i==0)sval=temp;
        else if(temp<=sval)break;//后面读出的数据一定要比第一次读到的数据大.
        LCD_ShowxNum(x+15*8,y,(u16)(temp-sval+1)*4,4,16,0);//显示内存容量
    }
}
int main(void)
{
    u8 key; u8 i=0; u32 ts=0;
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);//设置系统中断优先级分组 2
```

```
    delay_init(168);  //初始化延时函数
    uart_init(115200);      //初始化串口波特率为 115200
    LED_Init();                     //初始化 LED
    LCD_Init();             //LCD 初始化
    KEY_Init();                     //按键初始化
    FSMC_SRAM_Init();               //初始化外部 SRAM
    POINT_COLOR=RED;//设置字体为红色
    LCD_ShowString(30,50,200,16,16,"Explorer STM32F4");
    LCD_ShowString(30,70,200,16,16,"SRAM TEST");
    LCD_ShowString(30,90,200,16,16,"ATOM@ALIENTEK");
    LCD_ShowString(30,110,200,16,16,"2014/5/14");
    LCD_ShowString(30,130,200,16,16,"KEY0:Test Sram");
    LCD_ShowString(30,150,200,16,16,"KEY1:TEST Data");
    POINT_COLOR=BLUE;//设置字体为蓝色
    for(ts=0;ts<250000;ts++)testsram[ts]=ts;//预存测试数据
    while(1)
    {
        key=KEY_Scan(0);//不支持连按
        if(key==KEY0_PRES)fsmc_sram_test(60,170);//测试 SRAM 容量
        else if(key==KEY1_PRES)//打印预存测试数据
        {
    for(ts=0;ts<250000;ts++)LCD_ShowxNum(60,190,testsram[ts],6,16,0);//显示测试数据
        }else delay_ms(10);
        i++;
        if(i==20)//DS0 闪烁.
        {
            i=0;LED0=!LED0;
        }
    }
}
```

此部分代码除了 mian 函数，还有一个 fsmc_sram_test 函数，该函数用于测试外部 SRAM 的容量大小，并显示其容量。main 函数则比较简单，我们就不细说了。

此段代码，我们定义了一个超大数组 testsram，我们指定该数组定义在外部 sram 起始地址（__attribute__((at(0X68000000)))），该数组用来测试外部 SRAM 数据的读写。注意该数组的定义方法，是我们推荐的使用外部 SRAM 的方法。如果想用 MDK 自动分配，那么需要用到分散加载还需要添加汇编的 FSMC 初始化代码，相对来说比较麻烦。而且外部 SRAM 访问速度又远不如内部 SRAM，如果将一些需要快速访问的 SRAM 定义到了外部 SRAM，将会严重拖慢程序运行速度。而如果以我们推荐的方式来分配外部 SRAM，那么就可以控制 SRAM 的分配，可以针对性的选择放外部还是放内部，有利于提高程序运行速度，使用起来也比较方便。

41.4 下载验证

在代码编译成功之后，我们通过下载代码到 ALIENTEK 探索者 STM32F4 开发板上，得到如图 41 在代码编译成功之后，我们通过下载代码到 ALIENTEK 探索者 STM32F4 开发板上，

得到如图 41.4.1 所示界面：

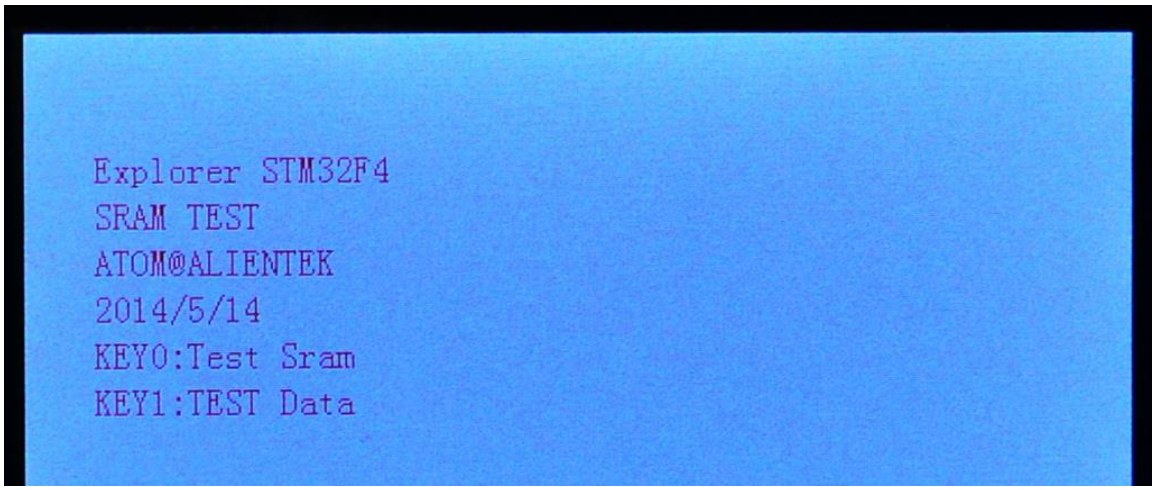

图 41.4.1 程序运行效果图

此时，我们按下 KEY0，就可以在 LCD 上看到内存测试的画面，同样，按下 KEY1，就可以看到 LCD 显示存放在数组 testsram 里面的测试数据，如图 41.4.2 所示：

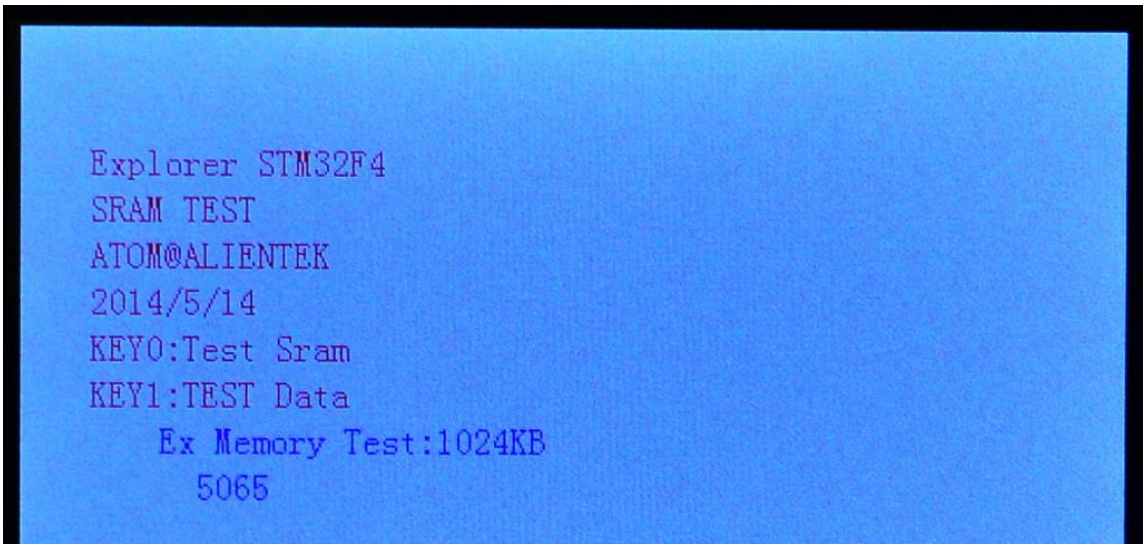

图 41.4.2 外部 SRAM 测试界面