<http://www.openedv.com/posts/list/43374.htm>

# 第四十一章 外部SRAM实验

[**view source**](http://www.openedv.com/posts/list/43374.htm#viewSource)[**print**](http://www.openedv.com/posts/list/43374.htm#printSource)[**?**](http://www.openedv.com/posts/list/43374.htm#about)

|  |  |  |
| --- | --- | --- |
| 1 | 1.硬件平台：正点原子探索者STM32F407开发板 | |
| 2 | 2.软件平台：MDK5.1 |

|  |  |
| --- | --- |
| 3 | 3.固件库版本：V1.4.0 |

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：

l 高速。具有45ns/55ns访问速度。

l 低功耗。

l TTL电平兼容。

l 全静态操作。不需要刷新和时钟电路。

l 三态输出。

l 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：

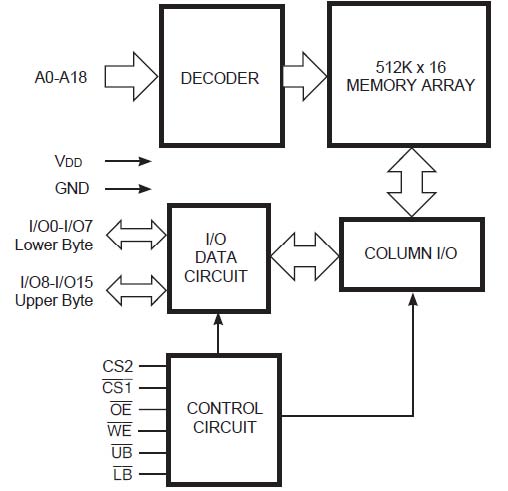


图41.1.1 IS62WV51216功能框图

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：

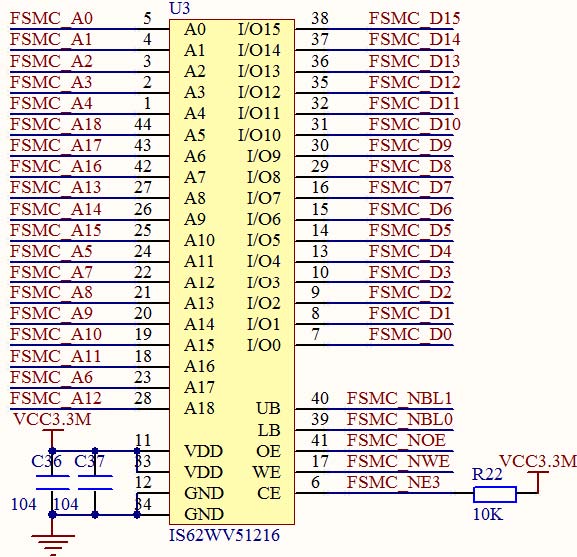


图41.1.2 IS62WV51216原理图

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：

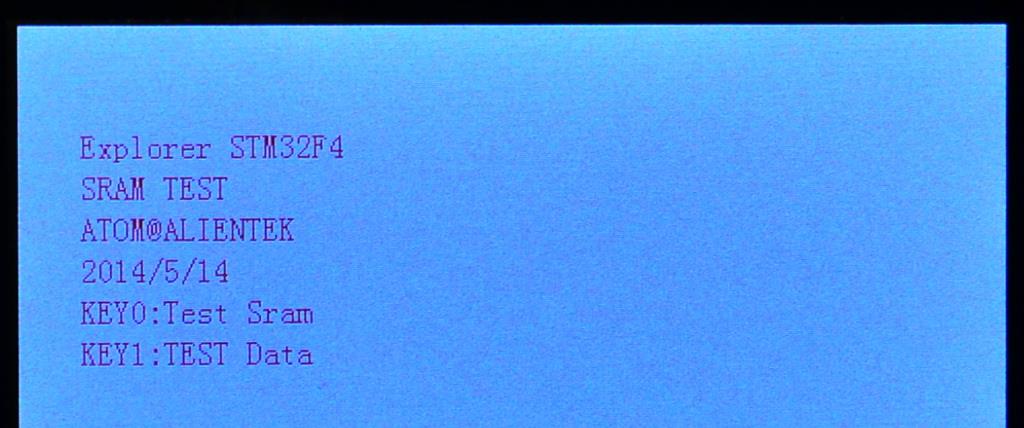


图41.4.1 程序运行效果图

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：

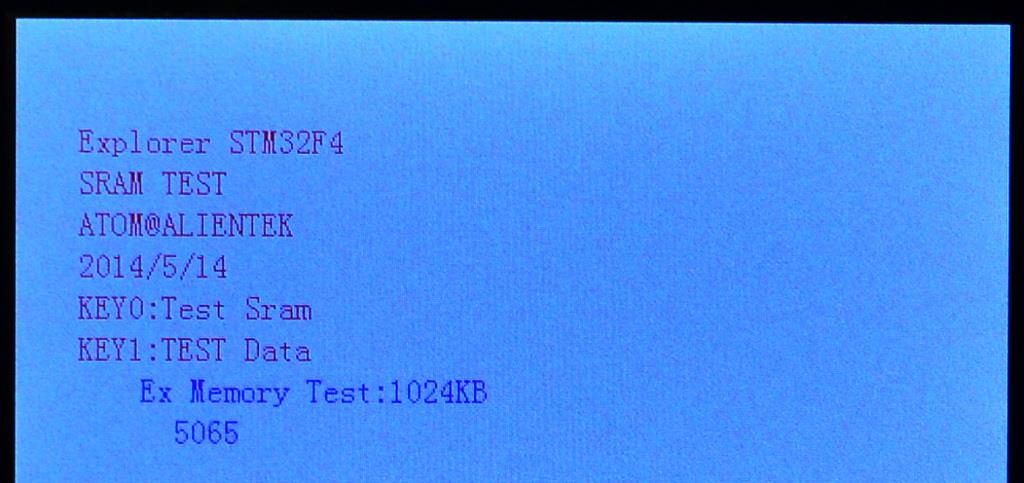
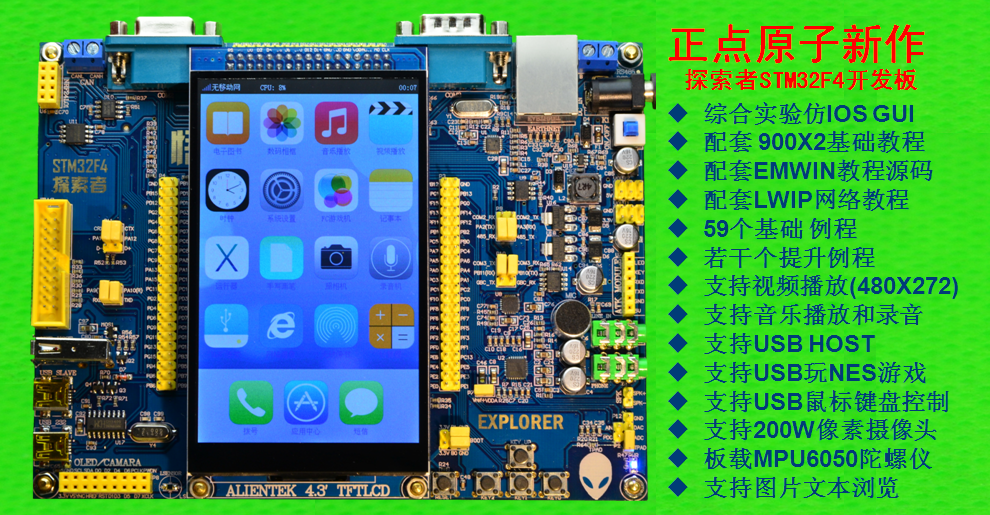


图41.4.2 外部SRAM测试界面

**实验详细手册和源码下载地址：**<http://www.openedv.com/posts/list/41586.htm>

**正点原子探索者STM32F407开发板购买地址：**[**http://item.taobao.com/item.htm?id=41855882779**](http://item.taobao.com/item.htm?id=41855882779)

**[](http://item.taobao.com/item.htm?id=41855882779)**

|  |
| --- |
| **附件1：** [**实验36 外部SRAM实验.zip (文件大小: 522 KB 下载次数:30次)** [AttachmentIcon](http://www.openedv.com/posts/downloadAttach/8409.htm)](http://www.openedv.com/posts/downloadAttach/8409.htm)  **附件2：** [**第四十一章 外部SRAM实验-STM32F4开发指南-正点原子探索者STM32开发板.pdf (文件大小: 669 KB 下载次数:29次)** [AttachmentIcon](http://www.openedv.com/posts/downloadAttach/8410.htm)](http://www.openedv.com/posts/downloadAttach/8410.htm) |

<http://blog.163.com/tianjunqiang666@126/blog/static/87259119201311272520135/>

### STM32在MDK中使用外部RAM浅解

2013-12-27 14:54:50| 分类： [STM32](http://blog.163.com/tianjunqiang666@126/blog/#m=0&t=1&c=fks_084068081084083070085084083095092081083070093087087074) | 标签： |举报 |字号大中小 订阅

使用简介：

开发板 神舟王STM32F207开发板

问题描述：

在使用emwin的时候由于占用的变量较多，出现了STM32F207内部RAM不够用的尴尬局面，开发板自带了4M的外部SRAM，因此考虑速度和存储的平衡，使用芯片内部RAM作为堆栈区使用，外部RAM则用来存储其他变量

修改内容：

启动文件即startup\_stm32f2xx.s文件

1.添加标志量：DATA\_IN\_ExtSRAM EQU 1 ;主要是方便控制切换使用内部和外部RAM

2.修改栈区： IF DATA\_IN\_ExtSRAM == 1

\_\_initial\_sp EQU 0x20000000 + Stack\_Size

ELSE

\_\_initial\_sp

ENDIF

3.在进入main函数之前初始化外部RAM

有两种初始化方式

第一种：在SystemInit内初始化

SystemInit函数位于system\_stm32f2xx.c文件内

定义宏变量DATA\_IN\_ExtSRAM

这样系统初始化时就会调用函数 SystemInit\_ExtMemCtl

修改 SystemInit\_ExtMemCtl函数内容，即初始化FSMC RAM

第二种

在startup\_stm32f2xx.s内 调用main函数前先调用RAM初始化函数完成RAM的初始化

IMPORT \_\_main

IMPORT SystemInit

IMPORT SRAM\_Init

LDR R0, =SystemInit

BLX R0

LDR R0, =SRAM\_Init

BLX R0

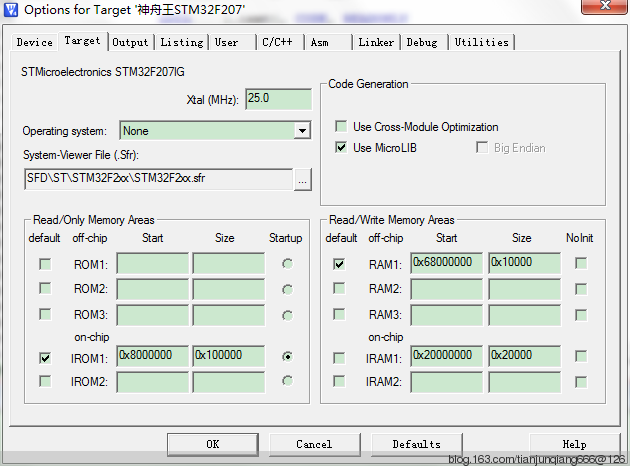
LDR R0, =\_\_main

BX R0

ENDP

当然还有最蛋疼的一种方式就是在startup\_stm32f2xx.s初始化外部RAM 不过需要修改成汇编语言

最后就是修改MDK的配置 在option->target勾选上外部RAM



之后就可以尝试定义一个大数组试试了