SDRAM　使用手册

南京博芯电子技术有限公司

2009-04

否认书：
南京博芯电子技术有限公司保留对文档随时进行修改的权利，无须任何申明。南京博芯电子技术有限公司所提供的信息是精确可靠的。对于它的应用以及由于应用而导致违反专利权或是第三方的其他权利，本公司不负任何责任。

版本历史

日期	版本	描述	备注
2009-04	1.0	初稿	Jack

目　　录

一. SDRAM 在 SEP4020 中的位置

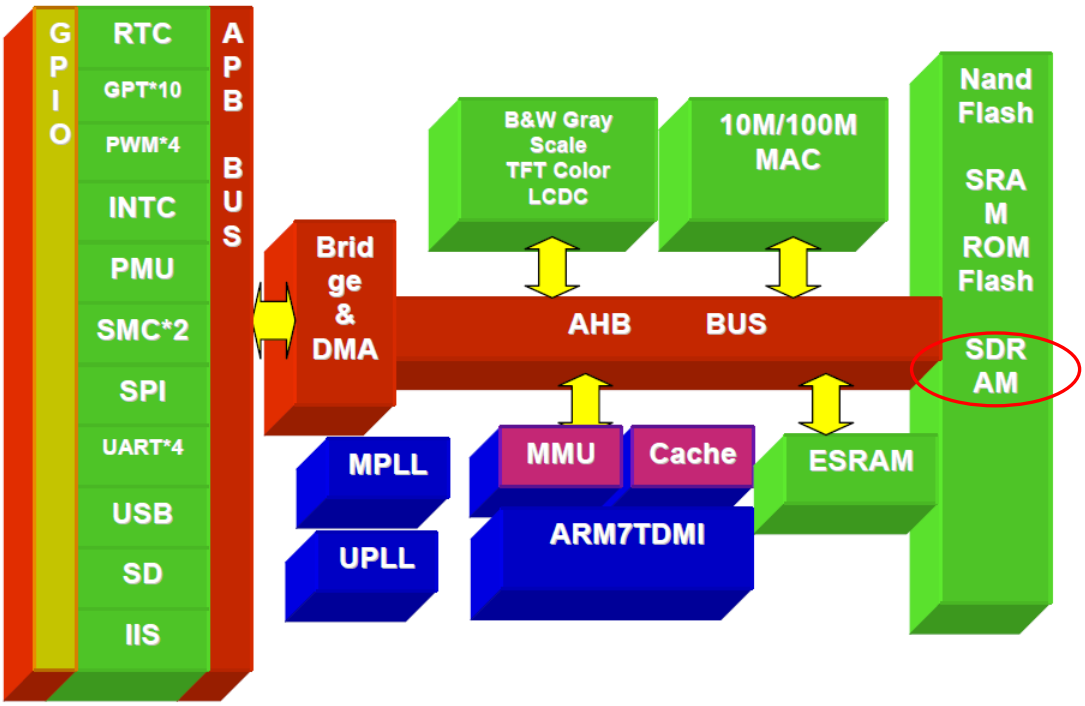


图 1 SDRAM 在 SEP4020 中的位置

二. SDRAM 介绍

2.1 功能介绍

片选 CSA/CSB/CSC/CSD 用于SRAM,CSE/CSF 根据配置用于SRAM 或SDRAM。支持JEDEC 标准的SRAM 和NOR FLASH 的读操作和写操作。

每个片选最大支持16Mbyte 地址空间。

每个片选的数据宽度8 位和16 位可配，其中CSA 仅仅支持16 位存储器。

每个片选的时序参数分别配置,可以使得每个片选工作在不同的速度上。时序参数如下：OE_HOLD、OE_EN、OE_WAIT、CS_HOLD、CS_WAIT、WE_HOLD、WE_EN 和WE_WAIT。（具体配置请参加具体存储器芯片的用户手册）

2.2 寄存器介绍

SDRAM属于外部存储接口模块（EXTERNAL MEMORY INTERFACE，简称EMI）。

EMI模块的基址：0x11000000

名称	偏移地址	复位值	描述
CSECONF	0x010	0x8C656551	CSE 参数配置寄存器
CSFCONF	0x14	0x0D656551	CSF 参数配置寄存器
SDCONF1	0x18	0x1C004077	SDRAM 时序配置寄存器

			1
SDCONF2	0x1C	0x00001860	SDRAM 时序配置寄存器2，SDRAM 初始化用到的配置信息
REMAPCONF	0x020	0x00000003	片选空间及地址映射REMAP 配置寄存器

三. 实现原理

3.1 硬件原理

3.1.1 接口定义

CSE，CSF 可以根据配置用于SDRAM。

支持JEDEC 标准的SDRAM。

SDRAM 片选最大支持64Mbyte 地址空间。

SDRAM 片选支持的数据宽度为16 位。

SDRAM 行地址宽度和列地址宽度可配。行地址范围11－13 位，列地址范围8－11 位。

Bank 地址位置可配，可以配置成row 地址的高位，也可以配置成row 地址的低位。

时序参数可配。时序参数和各自意义如下

tRCD 从一行active 到read/write 命令等待的周期数。

tCAS 从发出read 命令到收到数据等待的周期数。

tRP 从 prechage 命令到下一次active/refresh 命令需要等待的周期。

tRFC 从auto refresh 命令到后续命令需要等待的周期。

tRC 从 active 一行到active 另一行需要等待的周期数。

tXSR 从self refresh 退出时，从CKE 的上升沿到后续auto refresh 命令需要等待的周期数。

一般情况下，其他可以直接满足不需要配置的参数：

tRRD active 从本次命令的时钟上升沿到发出下个命令的时钟上升沿之间的时间

tRASactive 到 precharge 之间的周期数。

等待周期数指的是从发出本次命令的时钟上升沿到发出下个命令的时钟上升沿之间的时间。

SDRAM 的刷新时间可配，每次刷新的行数可配。

SDRAM 的precharge 不使用auto precharge 模式。使用 delay precharge 模式。

支持SDRAM 的powerdown mode，一段时间不工作，SDRAM 控制器可以控制SDRAM 进入powerdown 模式，进入powerdown mode 的时间可配。在powerdown 的情况下，SDRAM 控制

器

仍然保持对SDRAM 的auto refresh。如果power down 期间有读写访问，SDRAM 控制器控制SDRAM 退出power down 模式。

支持SDRAM 的self refresh 模式，由PMU 发出selfref 请求。如果有读写请求，自动退出self refresh 状态。

SDRAM 的Burst 固定为1 拍。由于每次只发出一拍传输，SDRAM 控制器不会发出terminate 命令。在AHB 的incr 模式下，SDRAM 控制器会预取下一个数据。如果incr 传输停止，或者预取到了行边界，预取停止。

对应的框图如下：

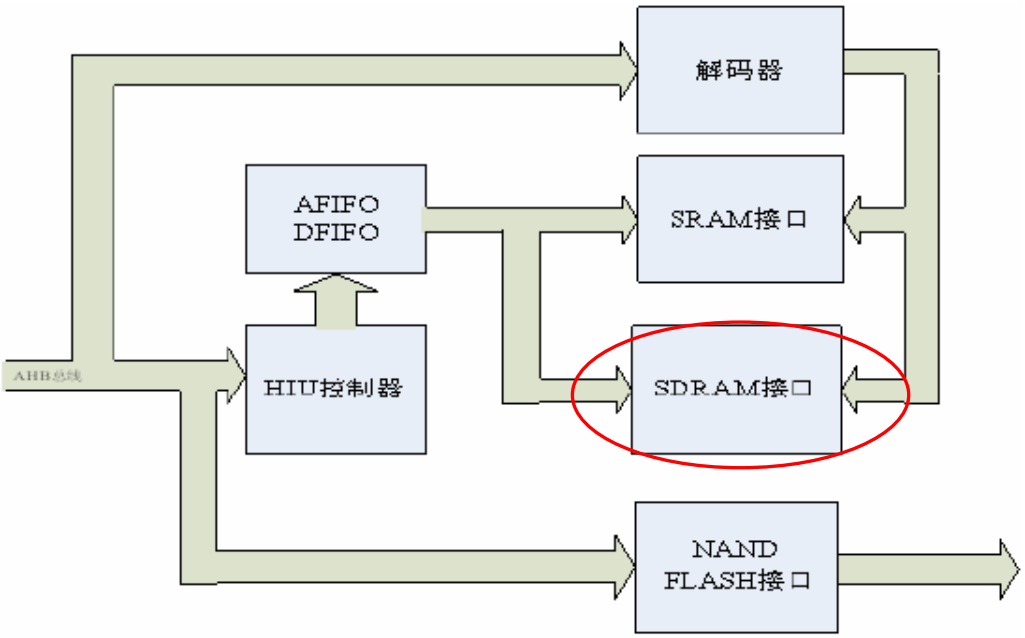


图2 EMI整体框图

3.1.1.1 SEP4020 nandflash部分管脚定义

序号	管脚名	方向	描述	驱动电流（mA）	属性	复位值
156	nSDCSE	O	SRAM/NOR FLASH/SDRAM片选，缺省SDRAM片选	4		1,H1
155	nSDCSF	O	SRAM/NOR FLASH/SDRAM片选，缺省SRAM/NOR FLASH片选	4		1,H1
163	SDCLK	O	SDRAM时钟	4		1,H1

161	nSDCKE	O	SDRAM时钟使能	12		1,H1
157	nSDRAS	O	SDRAM行地址	4		1,H1
158	nSDCAS	O	SDRAM列地址	4		1,H1
159	nSDWE	O	SDRAM写使能	4		1,H1
160	SD_A10	O	SDRAM专用地址线，用于输出命令	4		1,H1
165/166	DQM[1:0]	O	SRAM/SDRAM字节使能	4		1’H0

3.1.2 nandflash 驱动信号时序

SDRAM 的读写操作都只使用Burst 方式。在多个读数据或写数据操作时使用random access 方式连续读写。

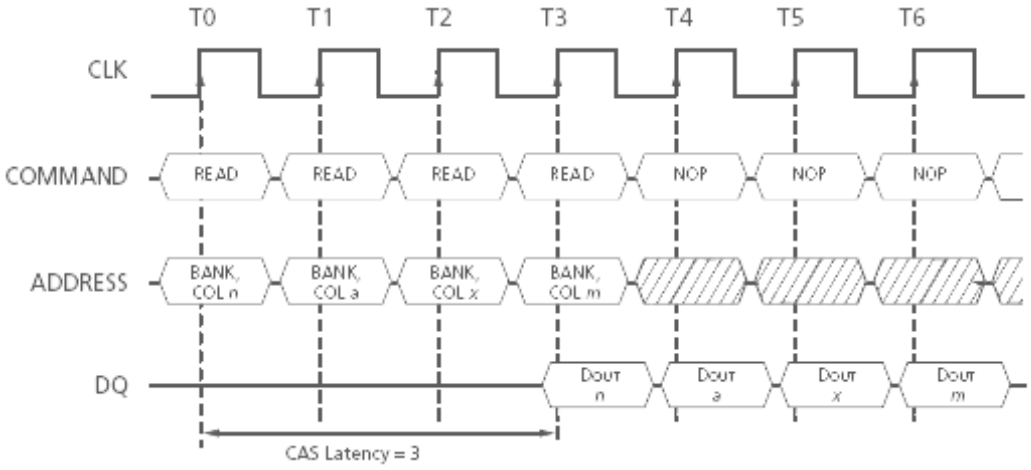


图 3 随机读操作

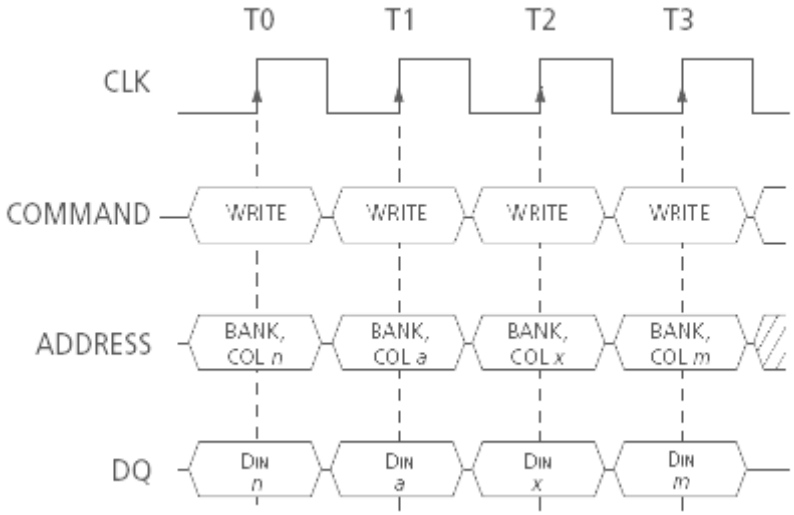


图 4 随机写操作

3.2 软件原理

3.2.1 头文件定义说明

#include　〈stdio.h〉　　标准输入输出函数库

#include　"sep4020.h"　　所有程序中用到的Typedef，Error Codes，PMU 模块时钟

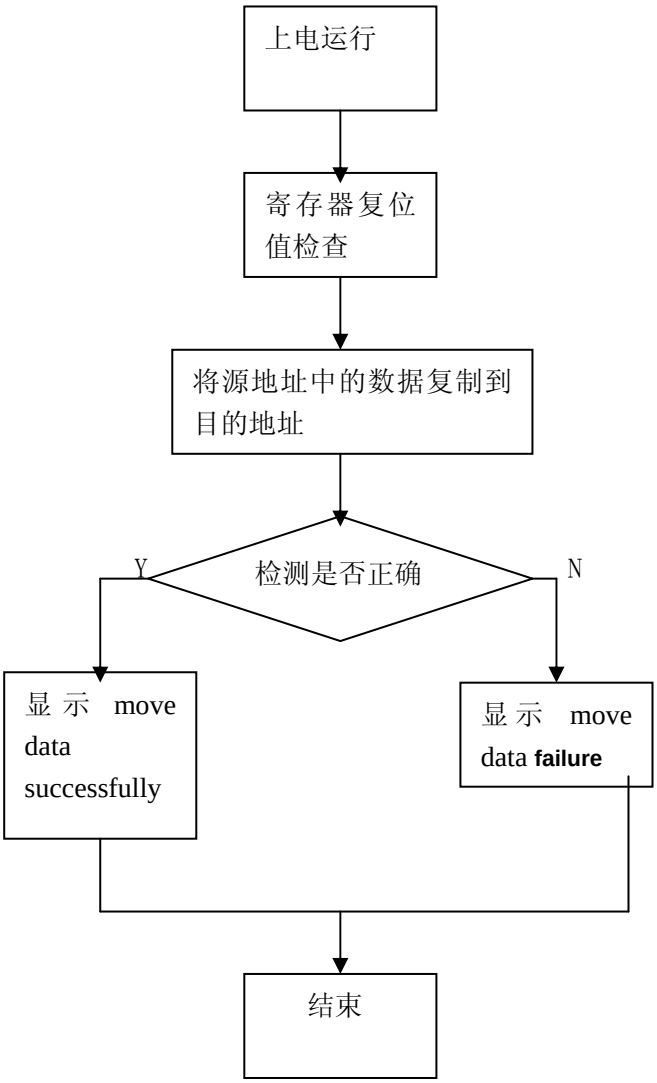
#include　"intc.h"　　INTC 模块的中断源，中断处理，定义中断的向量结构体

#include　"emi.h"　　对Nandflash的操作函数

3.2.2 核心数据结构声明

INT_VECTOR　中断向量结构体包括：中断号和中断处理函数

3.2.3 代码实现流程图



3.2.4 主要函数及参数，返回值介绍

(1) void BasicDataTran(U32 src_addr, U32 dest_addr, U32 length)

描述：数据的传输

输入：src_addr　源地址

dest_addr　目的地址

length　所搬运数据的长度

输出　无

使用位置：初始化后的任意位置

(2) U32 COMPARE_MEM(U32 src_addr, U32 dest_addr, U32 length)

描述：比较搬运后,目的地址上的内容与源地址上的内容是否一致

输入：src_addr　源地址

dest_addr　目的地址

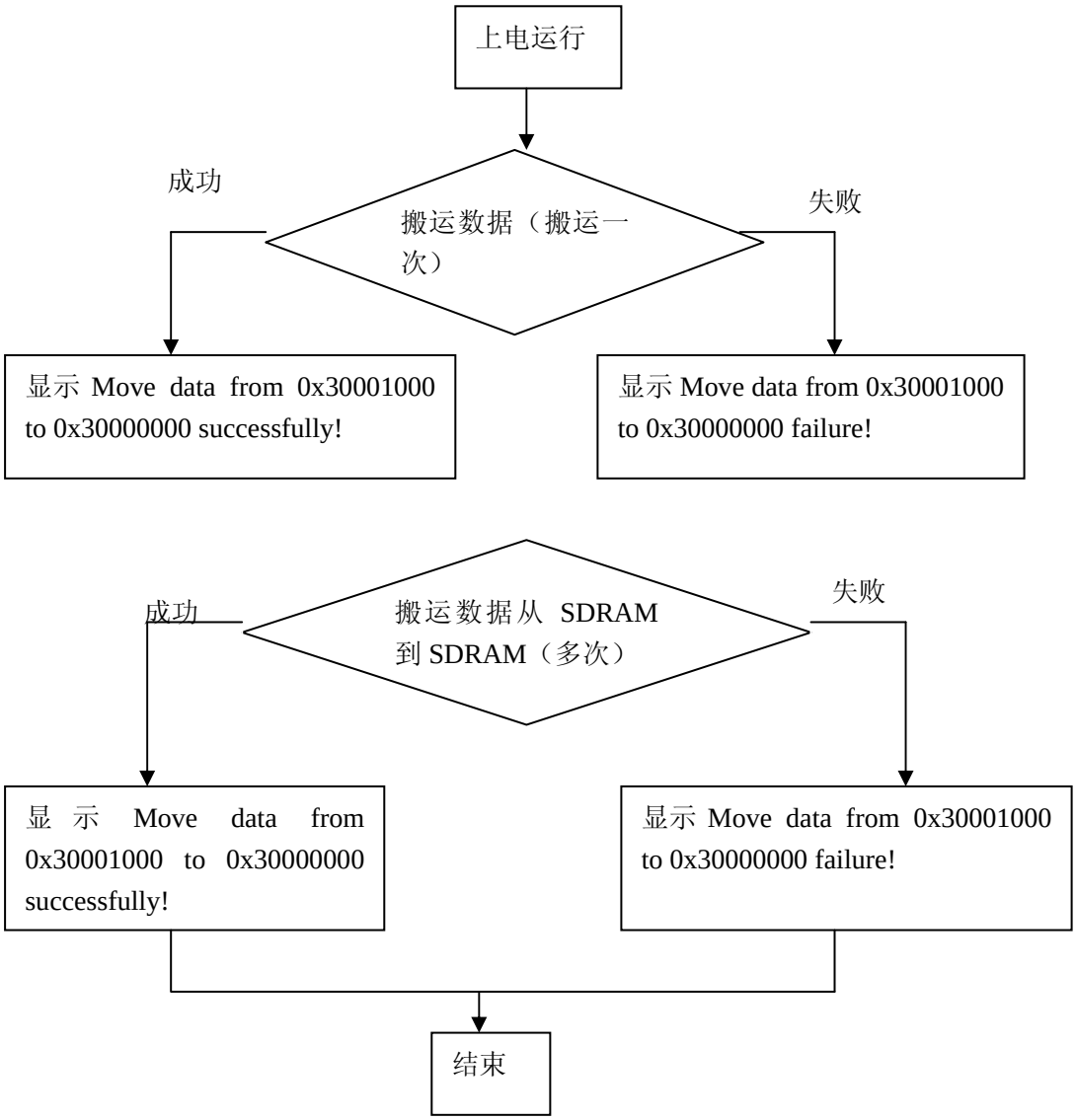
length　所比较的数据的长度

输出：flag=0表示成功

flag=1表示失败

四．测试说明

4.1测试流程图



4.2测试结果

在console窗口中输出的数据

```
Tests begin!
Move data from 0x30001000 to 0x30000000 successfully!
Move data from 0x30001000 to 0x30003000 + 0 *0x1000  successfully!
Move data from 0x30001000 to 0x30003000 + 1 *0x1000  successfully!
Move data from 0x30001000 to 0x30003000 + 2 *0x1000  successfully!
Move data from 0x30001000 to 0x30003000 + 3 *0x1000  successfully!
Move data from 0x30001000 to 0x30003000 + 4 *0x1000  successfully!
Move data from 0x30001000 to 0x30003000 + 5 *0x1000  successfully!
Move data from 0x30001000 to 0x30003000 + 6 *0x1000  successfully!
```

```
Move data from 0x30001000 to 0x30003000 + 7 *0x1000  successfully!
Move data from 0x30001000 to 0x30003000 + 8 *0x1000  successfully!
Move data from 0x30001000 to 0x30003000 + 9 *0x1000  successfully!
Move data from 0x30001000 to 0x30003000 + 10 *0x1000  successfully!
Move data from 0x30001000 to 0x30003000 + 11 *0x1000  successfully!
Move data from 0x30001000 to 0x30003000 + 12 *0x1000  successfully!
Move data from 0x30001000 to 0x30003000 + 13 *0x1000  successfully!
Move data from 0x30001000 to 0x30003000 + 14 *0x1000  successfully!
Move data from 0x30001
```