华清远见 FARSIGHT 始于2004 | 嵌入式培训专家

10年口碑积累，成功培养50000多名研发工程师，铸就专业品牌形象

华清远见的企业理念是不仅要做良心教育、做专业教育，更要做受人尊敬的职业教育。

《ARM 嵌入式系统开发典型模块》

作者：华清远见

专业始于专注 卓识源于远见

第 2 章 Flash 存储器模块

专业始于专注 卓识源于远见

2.1 Flash 模块功能简介

Flash 存储器是一种可在系统（In-System）进行电擦写，掉电后信息不丢失的存储器。它具有低功耗、大容量、擦写速度快、可整片或分扇区在系统编程（烧写）、擦除等特点，在体积、抗震性方面都有很大的优势，并且可由内部嵌入的算法完成对芯片的操作，因而在各种嵌入式系统中得到了广泛的应用。作为一种非易失性存储器，Flash 在系统中通常用于存放程序代码、常量表以及一些在系统掉电后需要保存的用户数据等。常用的 Flash 为 8 位或 16 位的数据宽度，编程电压为单 3.3V。主要的生产厂商为 ATMEL、AMD、HYUNDAI 等，他们生产的同型器件一般具有相同的电气特性和封装形式，可通用。Flash 闪存是非易失存储器，可以对存储器单元块进行擦写和再编程。任何 Flash 器件进行写入操作前必须先执行擦除。NAND 器件执行擦除操作十分简单，而 NOR 则要求在进行擦除前，先将目标块内所有的位都写为 0。擦除 NOR 器件时是以 64～128KB 的块进行的，执行一个写入/擦除操作的时间为 1～5s；擦除 NAND 器件是以 8～32KB 的块进行的，执行相同的操作最多只需要 4ms。执行擦除时，块尺寸的不同进一步拉大了 NOR 和 NAND 之间的性能差距。统计表明，对于给定的一套写入操作（尤其是更新小文件时），更多的擦除操作必须在基于 NOR 的单元中进行。因此，当选择存储解决方案时，设计者必须权衡以下的各项因素。

Flash 的一般结构如图 2.1 所示。

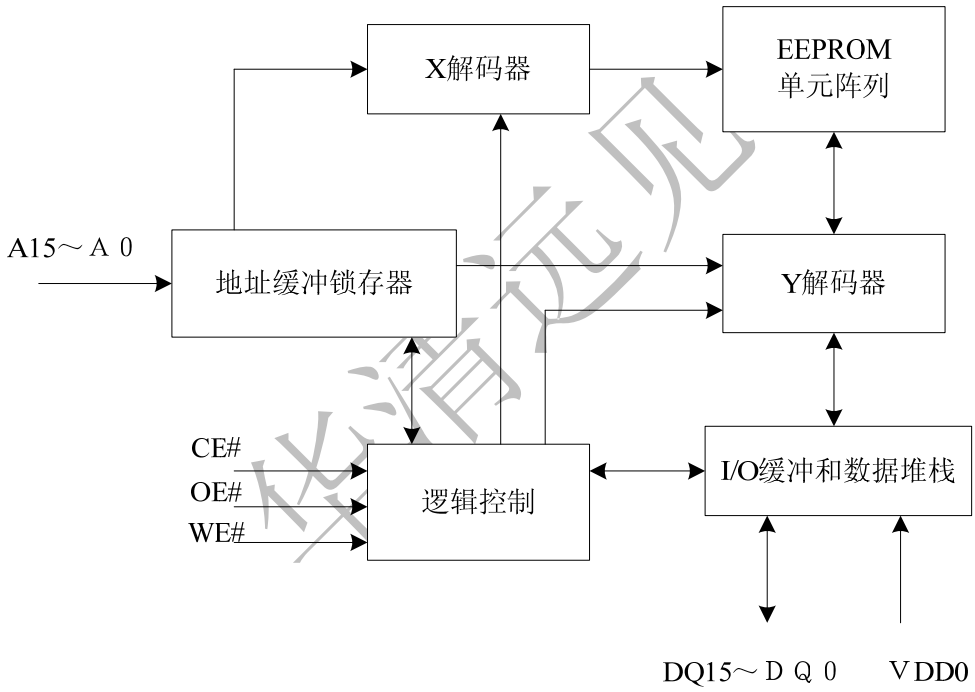

图 2.1 Flash 的结构图

2.2 Flash 器件的分类

Flash 按结构可分为 NOR 和 NAND 两大类。Intel 于 1988 年首先开发出 NOR Flash 技术，彻底改变了原先由 EPROM 和 EEPROM 一统天下的局面。紧接着，1989 年，东芝公司发表了 NAND Flash 结构，强调降低每比特的成本、更高的性能，并且像磁盘一样可以通过接口轻松升级。但是经过了十多年之后，仍然有相当多的硬件工程师分不清 NOR 和 NAND 闪存。NOR 的特点是芯片内执行（XIP，eXecute In Place），这样应用程序可以直接在 Flash 闪存内运行，不必再把代码读到系统 RAM 中。NOR 的传输效率很高，在 1～4MB 的小容量时具有很高的成本效益，但是很低的写入和擦除速度大大影响了它的性能。NAND 结构能提供极高的单元密度，可以达到高存储密度，并且写入和擦除的速度也很快。应用 NAND 的困难在于 Flash 的管理和需要特殊的接口。NOR 的 NAND 的差别主要有以下方面。

（1）性能差别

Flash 闪存是非易失存储器，可以对称为块的存储器单元块进行擦写和再编程。任何 Flash 器件的写入操作只能在空或已擦除的单元内进行，所以大多数情况下，在进行写入操作之前必须先执行擦除。NAND 器件执行擦除操作是十分简单的，而 NOR 则要求在进行擦除前先要将目标块内所有的位都写为 0。由于擦除 NOR 器件时是以 64～128KB 的块进行的，执行一个写入/擦除操作的时间为 5s，与此相反，擦除 NAND 器件是以 8～32KB 的块进行的，执行相同的操作最多只需要 4ms。

执行擦除时块尺寸的不同进一步拉大了 NOR 和 NAND 之间的性能差距，统计表明，对于给定的一套写入操作（尤其是更新小文件时），更多的擦除操作必须在基于 NOR 的单元中进行。这样，当选择存储解决方案时，设计师必须权衡以下的各项因素。

- NOR 的读速度比 NAND 稍快一些。
- NAND 的写入速度比 NOR 快很多。
- NAND 的 4ms 擦除速度远比 NOR 的 5s 快。
- 大多数写入操作需要先进行擦除操作。
- NAND 的擦除单元更小，相应的擦除电路就更少。

（2）接口差别

NOR Flash 带有 SRAM 接口，有足够的地址引脚来寻址，可以很容易地存取其内部的每一个字节。

NAND 器件使用复杂的 I/O 口来串行地存取数据，各个产品或厂商的方法可能各不相同。8 个引脚用来传送控制、地址和数据信息。

NAND 读和写操作采用 512 字节的块，这一点有点像硬盘管理此类操作，很自然地，基于 NAND 的存储器就可以取代硬盘或其他块设备。

（3）容量和成本

NAND Flash 的单元尺寸几乎是 NOR 器件的一半，由于生产过程更为简单，NAND 结构可以在给定的模具尺寸内提供更高的容量，也就相应地降低了价格。

NOR Flash 占据了容量为 1～16MB 闪存市场的大部分，而 NAND Flash 只是用在 8～128MB 的产品当中，这也说明 NOR 主要应用在代码存储介质中，NAND 适合于数据存储。

（4）易用性

可以非常直接地使用基于 NOR 的闪存，可以像其他存储器那样连接，并可以在上面直接运行代码。

由于需要 I/O 接口，NAND 要复杂得多。各种 NAND 器件的存取方法因厂家而异。

在使用 NAND 器件时，必须先写入驱动程序，才能继续执行其他操作。向 NAND 器件写入信息需要相当的技巧，因为设计师绝不能向坏块写入，这就意味着在 NAND 器件上自始至终都必须进行虚拟映射。

（5）耐用性

在 NAND 闪存中每个块的最大擦写次数是 100 万次，而 NOR 的擦写次数是 10 万次。NAND 存储器除了具有 10：1 的块擦除周期优势外，典型的 NAND 块尺寸要比 NOR 器件小 8 倍，每个 NAND 存储器块在给定的时间内的删除次数也要少一些。

（6）位交换

所有 Flash 器件都受位交换现象的困扰。在某些情况下（很少见，NAND 发生的次数要比 NOR 多）一个比特位会发生反转或被报告反转了。

一位的变化可能不很明显，但是如果发生在一个关键文件上，这个小小的故障却可能导致系统停机。如果只是报告有问题，多读几次就可能解决了。

当然，如果这个位真的改变了，就必须采用错误探测/错误更正（EDC/ECC）算法。位反转的问题更多见于 NAND 闪存，NAND 的供应商建议使用 NAND 闪存的时候，同时使用 EDC/ECC 算法。

这个问题对于用 NAND 存储多媒体信息时倒不是致命的。当然，如果用本地存储设备来存储操作系统、配置文件或其他敏感信息时，必须使用 EDC/ECC 系统以确保可靠性。

（7）坏块处理

NAND 器件中的坏块是随机分布的。

NAND 器件需要对介质进行初始化扫描以发现坏块，并将坏块标记为不可用。在已制成的器件中，如果通过可靠的方法不能进行这项处理，将导致高故障率。

华清远见

（8）软件支持

当讨论软件支持的时候，应该区别基本的读/写/擦操作和高一级的用于磁盘仿真和闪存管理算法的软件，包括性能优化。

在 NOR 器件上运行代码不需要任何的软件支持，在 NAND 器件上进行同样操作时，通常需要驱动程序，也就是内存技术驱动程序（MTD），NAND 和 NOR 器件在进行写入和擦除操作时都需要 MTD。

使用 NOR 器件时所需要的 MTD 要相对少一些，许多厂商都提供用于 NOR 器件的更高级软件，这其中包括 M-System 的 TrueFfs 驱动，该驱动被 Wind River System、Microsoft、QNX Software System、Symbian 和 Intel 等厂商所采用。

驱动还用于对 DiskOnChip 产品进行仿真和 NAND 闪存的管理，包括纠错、坏块处理和损耗平衡。

2.3 Flash 存储器的硬件设计

2.3.1 常见 Flash 存储器简介

下面介绍 2 种目前市场常见的 Flash 器件，即 HY29LV160 和 SST39LF160。

HY29LV160 的工作电压为 2 .7～3.6V，单片存储容量为 2 M 字节，采用 48 脚 TSOP 封装，如图 2.2 所示。16 位数据宽度，可以以 8 位（字节模式）或 16 位（字模式）数据宽度的方式工作。

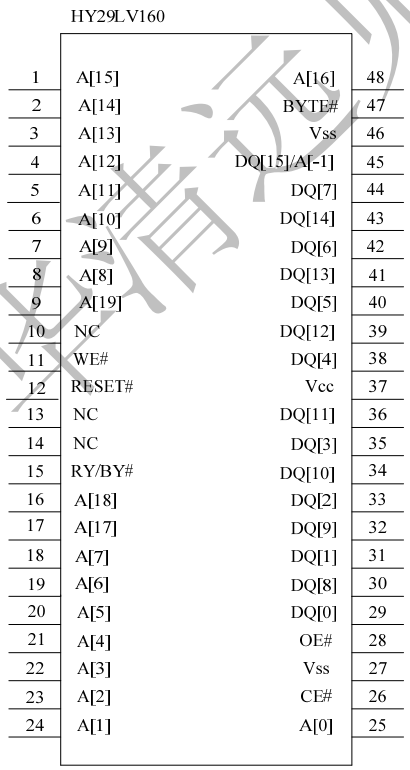

图 2.2 HY29LV160 引脚图

HY29LV160 仅需 3V 电压即可完成在线系统的编程与擦除操作，通过对其内部的命令寄存器写入标准的命令序列，可对 Flash 完成编程（烧写）、整片擦除、按扇区擦除以及其他操作。HV29LV160 的引脚功能描述如表 2.1 所示。

表 2.1 HY29LV160 引角功能描述

引脚	类型	描述
RESET#	I	硬件复位，低电平有效，对 HY29LV160 进行硬件复位

RY/BY＃	O	就绪/忙状态指示。用于指示写或擦除操作是否完成。当 HY29LV160 正在进行编程或擦除操作时，该引脚为低电平，操作完成时为高电平，此时可读取内部的数据
A[19:0]	I	地址总线。在字节模式下，DQ[15]/A[-1]用作 21 位字节地址的最低位
DQ[15]/A[-1] SQ[14:0]	I/O 三态	数据总线。在读写操作时提供 8 位或 16 位的数据宽度。在字节模式下，SQ[15]/A[-1]用作 21 位字节地址的最低位，而 DQ[14:8]处于高阻状态
OE#	I	输出使能，低电平有效。在读操作时有效，写操作时无效
WE#	I	写使能，低电平有效。在对 HY29LV160 进行编辑和擦除操作时，控制相应的写命令
CE#	I	片选信号，低电平有效。在对 HY29LV160 进行读写操作时，该引脚必须为低电平，当为高电平时，芯片处于高阻旁路状态
BYTE#	I	模式选择。低电平为字节模式，高电平为字模式
V_{CC}		3.3V 电源
V_{SS}		接地

关于 SST39LF160 的特点及引脚图在本书第一章已经介绍过，这里将介绍 SST39LF160 的各种操作。

1．器件操作

SST39LF160 的存储器操作由命令来启动，命令通过标准微处理器写时序写入器件，将 WE#拉低、CE#保持低电平来写入命令地址。总线上的地址在 WE#或 CE#的下降沿（无论哪一个后产生下降沿）被锁存。数据总线上的数据在 WE#或 CE#的上升沿（无论哪一个先产生上升沿）被锁存。SST39LF160 还包含一种自动低功耗模式，它可在有效读操作将数据读出后使器件进入一个接近等待模式的状态，这样 IDD 有效读电流从 15mA 降低到 4　A。自动低功耗模式可将典型的 IDD 有效读电流降低到读周期内的 1mA/MHz。只要启动另一次读周期的地址或控制信号发生变化，器件就退出自动低功耗模式，不占用访问时间。注意当 CE#保持低电平而使器件上电后器件就不能进入自动低功耗模式，直到出现第一个地址跳变或 CE#变为高电平。

SST39LF160 的读操作由 CE#和 OE#控制，只有两者都为低电平时系统才能从器件的输出管脚获得数据，CE#是器件片选信号，当 CE#为高电平时器件未被选中工作，只消耗等待电流。OE#是输出控制信号，用来控制输出管脚数据的输出。当 CE#或 OE#为高电平时数据总线呈现高阻态。SST39LF160 的读周期时序图如图 2.3 所示。

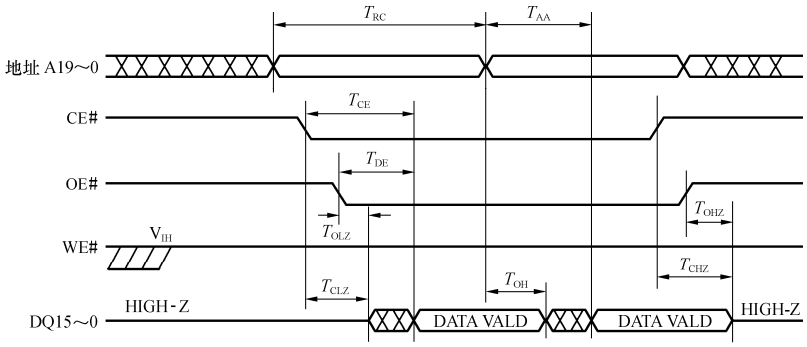

图 2.3　SST39LF160 读周期时序图

2．字编程操作

SST39LF160 以字形式进行编程，编程前包含字的扇区必须完全擦除。编程操作分为 3 步。第一步，执行 3 字节装载时序，用于软件数据保护。第二步，装载字地址和字数据。在字编程操作中地址在 CE#或 WE#的上升沿（不论哪一个后产生下降沿）锁存，数据在 CE#或 WE#的上升沿（不论哪一个先产生上升沿）锁存。第三步，执行内部编程操作该操作。在第 4 个 WE#或 CE#的上升沿出现（不论哪一个先产生上升沿）之后启动编程操作。一旦启动将在 20 s 内完成。

3．扇区/块擦除操作

作扇区操作通过在最新一个总线周期内执行一个 6 字节的命令时序（扇区擦除命令 30H 和扇区地址 SA）来启动。块擦除操作通过在最新一个总线周期内执行一个 6 字节的命令时序（块擦除命令 50H 和块地址 BA）来启动。扇区或块地址在第 6 个 WE#脉冲的下降沿锁存。命令（30H 或 50H）在第 6 个 WE#脉冲的上升沿锁存。内部擦除操作在第 6 个 WE#脉冲后开始执行擦除操作，是否结束由数据查询位或触发位决定。

4．芯片擦除操作

SST39LF160 包含芯片擦除功能，允许用户擦除整个存储器阵列，使其变为“1”状态。这在需要快速擦除整个器件时很有用。芯片擦除操作通过在最新一个总线周期内执行一个 6 字节的命令（5555H 地址处的芯片擦除命令 10H）时序来启动。在第 6 个 WE#或 CE#的上升沿（无论哪一个先出现上升沿）开始执行擦除操作。擦除过程中只有触发位或数据查询位的读操作有效。

5．写操作状态检测

SST39LF160 提供 2 种检测写（编程或擦除）周期结束的软件方法，以便优化系统的写周期。软件检测包括 2 个状态位：数据查询位 DQ7 和触发位 DQ6。写结束检测模式在 WE#的上升沿后使能，WE#的上升沿用来启动内部的编程或擦除操作。

非易失性写操作的结束与系统不同步，因此数据查询位或触发位的读取可能与写周期结束同时发生。如果这样系统就可能得到一个错误的结果（有效数据与 DQ7 或 DQ6 发生冲突）。为了防止错误的情况，当一个错误结果出现时软件程序应当包含一个两次读被访问地址单元的循环，如果两次读取的值均有效，则器件已经完成了写周期，否则拒绝接受数据。

当 SST39LF160 正在执行内部编程操作时，任何读 DQ7 的动作都将得到真实数据的补码，一旦编程操作结束，DQ7 为真实的数据，即使在内部写操作结束后紧接着出现在 DQ7 上的数据可能有效，其余的数据输出管脚上的数据也无效。只有在 1 s 的时间间隔后执行了连续读周期所得的整个数据总线上的数据才有效。在内部擦除操作过程中读出的 DQ7 值为‘0’，一旦内部擦除操作完成 DQ7 的值为‘1’。

编程操作的第 4 个 WE#或 CE#脉冲的上升沿出现后数据查询位有效。对于扇区/块擦除或芯片擦除，数据查询位在第 6 个 WE#或 CE#脉冲的上升沿出现后有效。

6．软件数据保护

SST39LF160 为数据变化操作，即编程和擦除操作提供了符合 JEDEC 标准的数据保护机制。编程操作要求包含一个 3 字节的时序。这个 3 字节装载时序用来启动编程操作。为防止意外写（例如系统上电或下电过程中出现的写操作）提供最佳保护。擦除操作要求包含一个 6 字节的时序，器件出厂时，软件数据保护特性就永久使能。

华清远见

7．通用 Flash 接口

SST39LF160 还包含 CFI 信息，用来描述器件特性。为了进入 CFI 查询模式，系统必须执行装载 3 字节的时序（类似于软件 ID 入口命令），该命令的最后一个周期将 98H（CFI 查询命令）装入地址 5555H。一旦器件进入 CFI 查询模式，系统就会读出 CFI 数据（地址信息表参见 SST39LF160 技术手册）。系统必须通过写入 CFI 退出命令，从 CFI 查询命令的执行过程中退出，返回到读模式。

表 2.2　　SST39LF160 编程擦写周期时序参数

符号	参数	最小	最大	单位
T_{BP}	字编程时间		20	μs
T_{AS}	地址建立时间	0		ns
T_{AH}	地址保持时间	30		ns
T_{CS}	WE#和 CE#建立时间	0		ns
T_{CH}	WE#和 CE#保持时间	0		ns
T_{OES}	OE#高建立时间	0		ns
T_{OEH}	OE#高电平保持时间	10		ns
T_{CP}	CE#脉冲宽度	40		ns
T_{WP}	WE#脉冲宽度	40		ns
T_{WPH1}	WE#脉宽高电平	30		ns
*T*CPH1	CE#脉宽高电平	30		ns
T_{DS}	数据建立时间	30		ns
*T*DH1	数据保持时间	0		ns
*T*IDAL	软件 ID 访问和退出时间	150	150	ns
*T*SE	扇区擦除	25	25	ms
*T*BE	块擦除	25	25	ms
*T*SCE	芯片擦除	100		ms

华清远见

2.3.2　Flash 存储器接口电路

我们以 S3C4510B 和 HY29LV160 芯片为例说明 Flash 存储器与 ARM 芯片的接口电路。图 2.4 为选用一片 Flash 存储器芯片 HY29LV160 构建 16 位的 Flash 存储器系统，其存储容量为 2MB。HY29LV160 仅需要 3V 电压即可完成在系统的编程与擦除操作，通过对其内部的命令寄存器写入标准的命令序列，可对 Flash 进行编程（烧写）、整片擦除、按扇区擦除以及其他操作。Flash 存储器在系统中通常用于存放程序代码，系统上电复位后从此获取指令并开始执行。因此，应将存有程序代码的 Flash 存储器配置到 ROM/SRAM/FLASH BANK0 中。

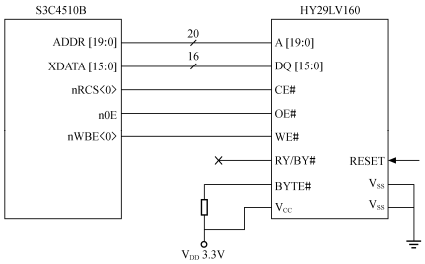

图 2.4　S3C4510B 的一片 Flash 扩展电路

S3C4510B 与 HY29LV160 的接口线

数据线： S3C4510B 的低 16 位数据总线［XDATAI5～XDATA0］与 HY29LV160 的[DQ15～DQ0]相连。

地址线：S3C4510B 的[ADDR19～ADDR0]与 HY29LV160 的[A19～A0]相连。S3C4510B 的 nRCS<0>接至 HY29LV160 的片选端 CE#。

控制线：HY29LV160 的 RESET#接系统复位信号。BYTE#上拉，使 HY29LV160 工作在字模式；RY/BY#指示 HY29LV160 编程或擦除操作的工作状态，其工作状态也可通过查询片内的相关寄存器来判断，因此可将该引脚悬空；读写线 OE#和 WE#分别与 S3C4510B 的 nOE, nWBE＜0＞相连。还应注意的一点是，此时应将 S3C4510B 的 BOSIZE[l: 0]置为‘10’，选择 ROM/SRAM/FLASH Bank0 为 16 位数据总线宽度工作方式。

2.4　Flash 存储器的操作

Flash 存储器的操作包括对 Flash 的擦除和烧写编程，针对不同的厂商和不同型号的 Flash 存储器，程序应作相应的修改。向 Flash 存储器的特定寄存器写入地址和数据命令，就可对 Flash 存储器进行烧写、擦除等操作，但操作必须按照一定的顺序进行，否则就会导致 Flash 存储器复位而使操作命令无法完成。编程指令只能使‘1’变为‘0’，而擦除命令可使‘0’变为‘1’，因此正确的操作顺序是先擦除，后编程，当 Flash 存储器被擦除以后读出的内容应全为 0xFF。下面以 HY29LV160 芯片为例介绍对 Flash 存储器的操作。

2.4.1　Flash 器件的编程操作

HY29LV160 的编程操作需要 4 个总线周期，编成指令时序从 2 个写周期开始，接着是编程开始指令，随后是编程地址和数据，并按照顺序启动相应的内嵌算法。芯片自动产生编成脉冲并校验被编程的单元。系统通过 DQ7、DQ6、RY/BY 来获知擦除操作的状态。当内嵌算法完成后，芯片将返回到读队列，数据和地址被释放。图 2.5 是 Flash 芯片的编程操作时序图。

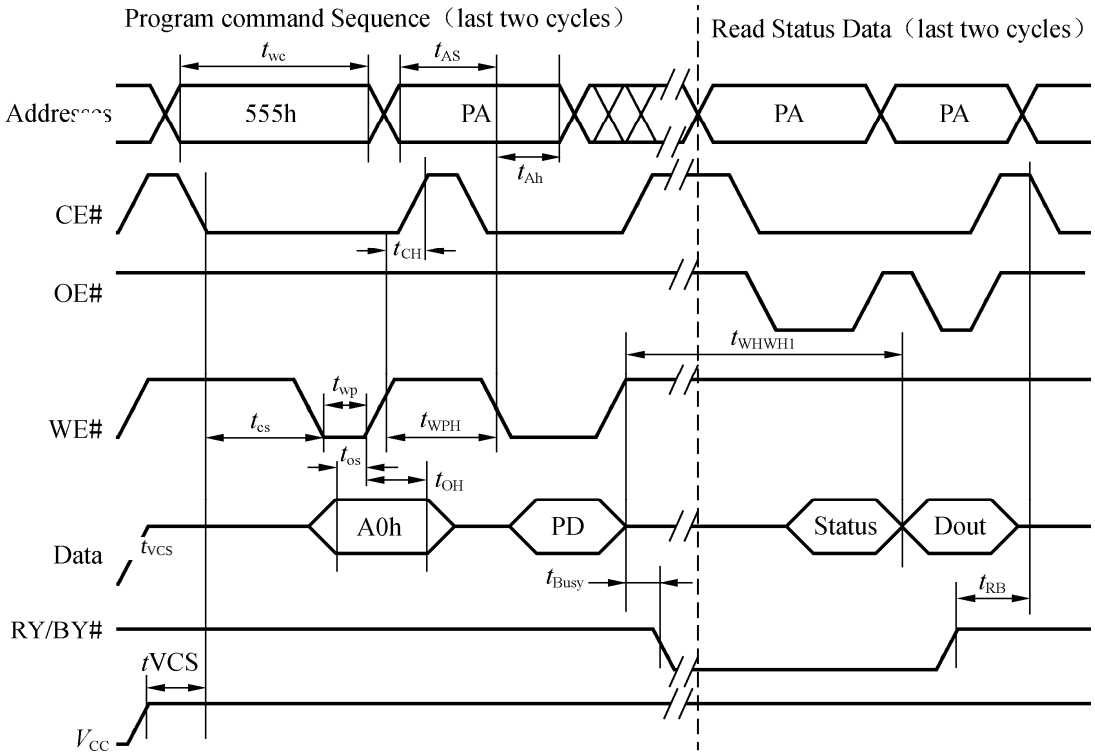

图 2.5　编程操作时序图

表 2.3　　正常编程命令

		第一步		第二步		第三步		第四步	
		地址	数据	地址	数据	地址	数据	地址	数据
正常	字	0x555	0xAA	0x2AA		0x555	0xA0	待编程的地址	
编程	字节	0xAAA		0x555		0xAAA		等编程的数据	

如上表所示，向 HY29LV160 指定地址中写入数据，可按（16 位）或字节（ 8 位）操作，共需要 4 个总线周期，分 4 步完成，前 2 个周期是解锁周期，第 3 个是建立编程命令，最后 1 个周期完成向待编程地址写入特定的数据。

2.4.2　Flash 器件的擦除操作

片擦除操作需要 6 个时序来操作。片擦除命令由 2 个非锁定周期引导，接下来是一个开始命令。跟在片擦除命令后的 2 个额外的非锁定写周期用来触发内嵌的擦除算法。内嵌擦除算法自动修正整个非零数据区。在擦除操作期间，系统不需要提供任何控制和它时序，但擦除过程中需要片擦除的数据和地址命令序列。在内嵌算法执行期间，任何写芯片的命令都被忽略。当芯片返回到读取数据队列时，片擦除命令序列被重新引导，来确保数据的完整。系统通过 DQ7、DQ6、DQ2、RY/BY 来获知擦除操作的状态。当内嵌擦除算法完成后，芯片返回到队列，数据和地址被释放。图 2.6 描述了 FLash 芯片的擦除操作时序。

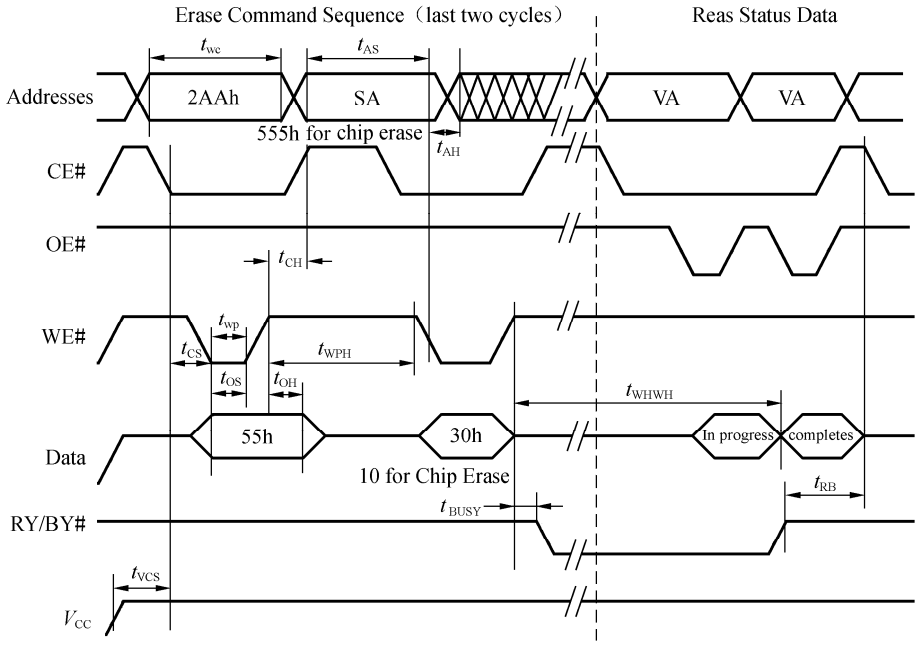

图 2.6　擦除操作时序图

结合图 2.6 的 HY29LV160 擦除时序图，表 2.4 给出了擦除命令及其地址及数据。将 HY29LV160 整片擦除，也可按字（16 位）或字节操作，共需要 6 个总线周期，分 6 步完成，前 2 个是解锁周期，第 3 个是建立编程命令，第 4、第 5 是解锁周期，最后 1 个周期是整片擦除周期。

表 2.4　　擦除命令

		第一步		第二步		第三步		第四步		第五步		第六步	
		地址	数据	地址	数据	地址	数据	地址	数据	地址	数据	地址	数据
整片擦除	字	0x555	0xAA	0x2AA	0x55	0x555	0x80	0x555	0xAA	0x2AA	0x55	0x555	0x10
	字节	0xAAA		0x555		0xAAA		0xAAA		0x555		0xAAA	

2.4.3 Flash 存储器的操作检测

当按照规定的命令序列向 Flash 存储器发出命令时，其内部的编程或擦除算法就可以自动完成编程或擦除操作，但无论是编程或擦除都需要一定的操作时间，同时用户还应了解其内部的操作检测机制，以便知道操作是否完成或操作是否正确。常用检测的状态位有：跳变位（DQ6），超时标志位（DQ5），数据查询位（DQ7）和 Ready/Busy（RY/BY#）。

常用的检测方法有 3 种。第 1 种是判断引脚 RY/BY#的状态，在编程或擦除操作过程中，RY/BY#引脚一直为低电平，操作完成后变为高电平。第 2 种就是检测跳变位 DQ6，在编程或擦除时对任何地址进行连续的读均引起 DQ6 连续跳变，直至操作结束才停止跳变。第 3 种方法是使用数据线的 DQ7 和 DQ5 位，DQ7 位在编程或擦除过程中输出的数是写入该位数据的反码，当操作完成时输出才变为写入该位的数据；DQ5 的状态为“1”时表示操作超时，此时应该再读一次 DQ7 的状态，若 DQ7 输出仍不是写入的数据，则操作失败，复位 Flash 存储器。其流程如图 2.7 所示。

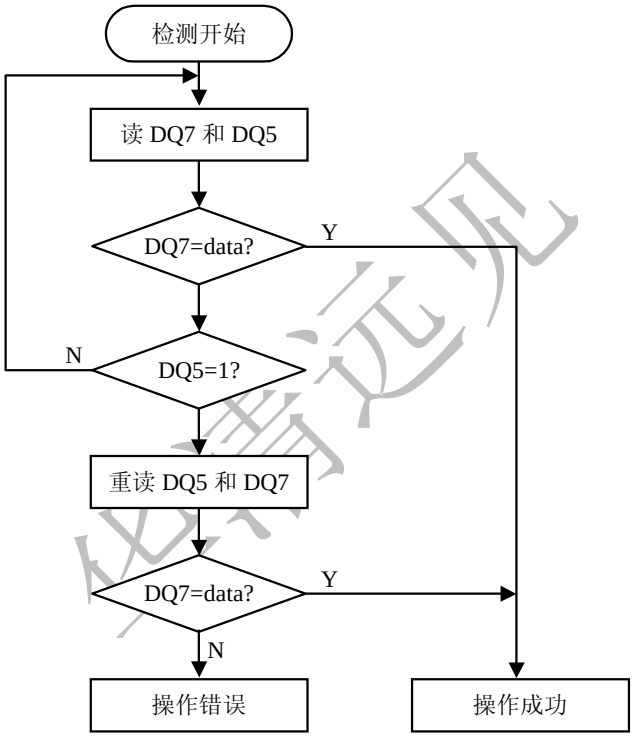

图 2.7 操作检测流程图

2.4.4 Flash 存储器的编程方法

对 Flash 存储器的烧写、整片擦除、按扇区擦除以及其他操作，都可以通过编程实现。现在以 HY23LV160 Flash 存储器的整片擦除为例，介绍 Flash 编程的方法。其参考程序如下。

```
# define FLASH_START_ADDR          0x0000
# define FLASH_ADDR_UNLOCK1        0x555
# define FLASH_ADDR_UNLOCK2        0x2aa
# define FLASH_DATA_UNLOCK1        0xaaaa
# define FLASH_DATA_UNLOCK2        0x5555
# define FLASH_SETUP_ERASE         0x8080
# define FLASH_CHIP_ERASE          0x1010
# define UNIT16 unsigned short
```

```
int FlashStatusDetect(UINT16 *ptr, UINT16 Data, int TimeCounter);
int    Main()
{
        int i;
        Volatile UINT16 *flashptr=(UINT16*) FLASH_START_ADDR;
        //按 HY29LV160 整片擦除命令要求写入命令序列,以 16 位方式操作;
        *((volatile UINT16 *) FLASH_START_ADDR+FLASH_ADDR_UNLOCK1)=FLASH_DATA_UNLOCK1;
        *((volatile UINT16 *) FLASH_START_ADDR+FLASH_ADDR_UNLOCK2)=FLASH_DATA_UNLOCK2;
        *((volatile UINT16 *) FLASH_START_ADDR+FLASH_ADDR_UNLOCK1)= FLASH_SETUP_ERASE;
        *((volatile UINT16 *) FLASH_START_ADDR+FLASH_ADDR_UNLOCK1)=FLASH_DATA_UNLOCK1;
        *((volatile UINT16 *) FLASH_START_ADDR+FLASH_ADDR_UNLOCK2)=FLASH_DATA_UNLOCK2;
        *((volatile UINT16 *) FLASH_START_ADDR+FLASH_ADDR_UNLOCK1)= FLASH_CHIP_ERASE;
//判断擦除操作是否正确完成
        if(FlashStatusDetect((UINT16 *)flashptr, 0xffff, 0x1000000)!=1)
        printf("ERROR!");
        return(0);
}
int FlashStatusDetect(UINT16 *ptr, UINT16 Data, int TimeCounter)
{
        int tmp=TimeCounter;
        volatile UINT16 *p=ptr;
        UINT16 data1, data2, current_data;
        current_data=Data & 0x8080;
        while((*p & 0x8080;) != current_data)
        {
                if(tmp--<= 0)
                return 0;
        }
        return 1;
}
```

在该程序中，首先以 16 位操作方式，按 HY29LV160 整片擦除命令要求写入命令序列，然后判断 DQ7 的状态是否正确。为稳妥起见，在擦除完成之后还应读取 Flash 存储器的内容与 0xFFFF 相比较，看是否相等，若相等则表明擦除操作完全正确，否则表明擦除操作错误或 Flash 存储器已经损坏，读者可自行添加这段代码。

注意符号常量 FLASH_START_ADDR，程序中用其标识系统中 Flash 存储器的起始物理地址，本例假定系统中 Flash 存储器的起始物理地址为 0x0，在系统复位或上电时，位于 ROM/SRAM/FLASH Bank0 的 Flash 存储器的起始物理地址会被映射到 0x0，可用本例进行整片擦除操作。但当 Flash 中写有应用程序或操作系统时，这些程序在启动并初始化系统后，可能会把 Flash 存储器映射到其他的地址空间，此时应修改该起始，否则就不能正确擦除 Flash 存储器。

2.5 用 16 位 Flash 芯片构成 32 位存储系统

ARM 微处理器的体系结构支持 8 位/16 位/32 位的存储器系统，对应地可以构建 8 位、16 位或 32 位的 Flash 存储器系统。32 位的存储器系统具有较高的性能，而 16 位的存储器系统则在成本及功耗方面占有优势，而 8 位的存储器系统现在已经很少使用。

图 2.8 以 HY29LV160 的 Flash 芯片为例说明如何用 2 片 16 位的 Flash 芯片构造基于 S3C4510B 的 32 位存储系统。采用 2 片 16 位数据宽度的 HY29LV160Flash 存储器芯片并联构建 16/32 位的 Flash 存储系统，其总存储容量可以根据需要灵活配置为 16 位 2MB 和 32 位 4MB，采用 2 片 HY29LV160 并联的方式构建 32 位的 Flash 存储器系统，其中一片为高 16 位，另一片为低 16 位，Flash 存储器在系统中通常用于存放程序代码，系统上电或复位后从此获取指令并开始执行，因此，应将存有程序代码的 Flash 存储器配置到 ROMISRAM/FLASH Bank0，即将 S3C4510B 的 nRCS<0>(Pin75)接至 2 片 HY29LV16 的 CE#端；

2 片 HY29LV160 的 RESET#端接系统复位信号；

2 片 HY29LV160 的 OE#端接 S3C4510B 的 nOE (Pin72)；

低 16 位片的 WE#端接 S3C4510B 的 nWBE<0> (Pin100)，高 16 位片的 WE#端接 S3C4510B 的 nWBE<2> (Pin102)；

2 片 HY29LV160 的 BYTE#均上拉，使之均工作在字模式；

2 片 HY29LV160 的地址总线[A19～A0」均与 S3C4510B 的地址总线[ADDR19～ADDR0]相连；

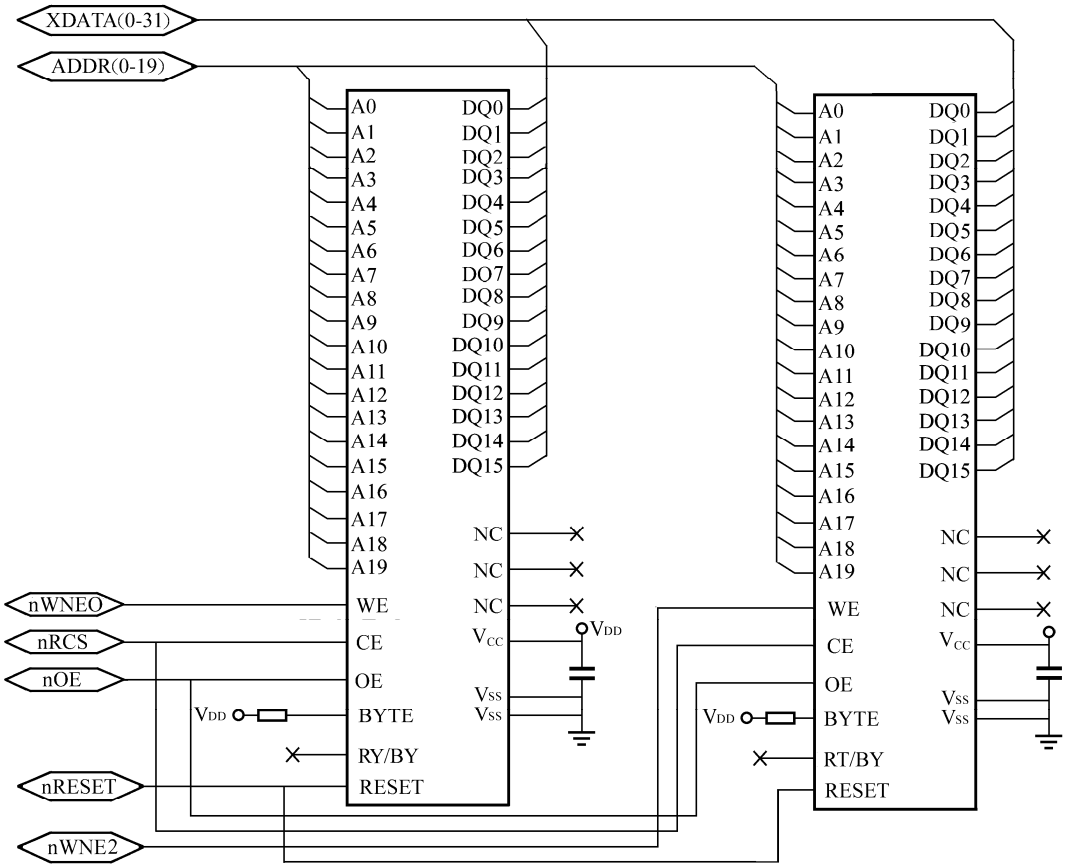

图 2.8　32 位系统接线图

低 16 位片的数据总线与 S3C4510B 的低 16 位数据总线「XDATAl5～XDATA0]相连，高 16 位片的数据总线与 S3C4510B[XDATA31 ～XDATA 16]相连；

若要工作在 16 位 2MB 方式时，应将 S3C4510B 的 BOSIZE [1:0]置为‘10’，选择 ROM/ SRAM/FLASH Bank0 为 16 位工作方式；

若要工作在 32 位 4MB 方式时，应将 S3C4510B 的 BOSIZE [1:0]置为‘11’，选择 ROM/ SRAM/FLASH Bank0 为 32 位工作方式。

2.6　应用程序设计

```
/*************************************************
```

```
 * NAME       : 44BLIB.H
 ***********************************************/
#define DebugOut Uart_Printf

#define min(x1,x2) ((x1<x2)? x1:x2)
#define max(x1,x2) ((x1>x2)? x1:x2)
#define ONESEC0 (62500)  //16us resolution, max 1.04 sec
#define ONESEC1 (31250)  //32us resolution, max 2.09 sec
#define ONESEC2 (15625)  //64us resolution, max 4.19 sec
#define ONESEC3 (7812)   //128us resolution, max 8.38 sec
#define ONESEC4 (MCLK/128/(0xff+1))   //@60Mhz, 128*4us resolution, max 32.53 sec
#define NULL 0
#define EnterPWDN(clkcon) ((void (*)(int))0xe0)(clkcon)
#define DOWNLOAD_ADDRESS _RAM_STARTADDRESS
/*44blib.c*/
void Delay(int time); //Watchdog Timer is used.
void *malloc(unsigned nbyte);
void free(void *pt);
void Port_Init(void);
void Cache_Flush(void);
void ChangeMemCon(unsigned *pMemCfg);
void Uart_Select(int ch);
void Uart_TxEmpty(int ch);
void Uart_Init(int mclk,int baud);
char Uart_Getch(void);
char Uart_GetKey(void);
int   Uart_GetIntNum(void);
void Uart_SendByte(int data);
void Uart_Printf(char *fmt,...);
void Uart_SendString(char *pt);
void Timer_Start(int divider); //应用看门狗定时器
int Timer_Stop(void);                //应用看门狗定时器
//void restart(void);
//void run(void);
void Led_Display(int LedStatus);
void Beep(int BeepStatus);
void ChangePllValue(int m,int p,int s);
/********Def.h***********/
#define U32          unsigned int
#define INT32U       unsigned int
#define INT32        int
#define U16          unsigned short
#define INT16U       unsigned short
#define INT16        short int
#define S32          int
#define S16          short int
```

```
#define U8              unsigned char
#define INT8U           unsigned char
#define INT8            char
#define     S8          char
#define TRUE            1
#define FALSE           0
#define OK              1
#define FAIL            0
#define FileEnd         1
#define     NotEnd      0
/*************flashrom.h************/
#include "..\Target\def.h"
#ifndef __ROMDEF_H__
#define __ROMDEF_H__
int     SST39VF160_WordProg(INT32U addr,INT16U dat);
void    SST39VF160_SectorErase(INT32U SAaddr);
int     SST39VF160_CheckBlank(INT32U addr,INT32U WordSize);
#define    Writeflash(addr,dat)       *((volatile INT16U *)(addr<<1))=(INT16U)dat
#define    Readflash(addr)            (*((volatile INT16U *)(addr<<1)))
#endif    // __ROMDEF_H //
/*************flashrom.c************/
#include "flashrom.h"
#include "..\Target\def.h"
int SST39VF160_CheckBlank(INT32U addr,INT32U WordSize)
{
    INT32U i,temp;
    for (i=addr;i<(addr+WordSize);i++)
    {
        temp=*((volatile INT16U *)(i<<1));
        if(temp!=0xffff)
        return 0;
    }
    return 1;
}
int Waitfor_endofprg(void) //Check if the bit6 toggle ends.
{
    volatile INT16U flashStatus,old;
    old=*((volatile INT16U *)0x0);
    while(1)
    {
        flashStatus=*((volatile INT16U *)0x0);
        if( (old&0x40) = = (flashStatus&0x40) )
            break;
        else
        old=flashStatus;
    }
```

```
    return 1;
}
int SST39VF160_WordProg (INT32U addr,INT16U dat)
{
    Writeflash (0x5555,0xAA);
    Writeflash (0x2AAA,0x55);
    Writeflash (0x5555,0xA0);
    Writeflash (addr,dat);
    return(Waitfor_endofprg());
}
void   SST39VF160_SectorErase(INT32U SAaddr)
{
    Writeflash(0x5555,0xAA);
    Writeflash(0x2AAA,0x55);
    Writeflash(0x5555,0x80);
    Writeflash(0x5555,0xAA);
    Writeflash(0x2AAA,0x55);
    Writeflash(SAaddr,0x30);
    Waitfor_endofprg();
}
```

联系方式

集团官网：www.hqyj.com　　嵌入式学院：www.embedu.org　　移动互联网学院：www.3g-edu.org

企业学院：www.farsight.com.cn　　物联网学院：www.topsight.cn　　研发中心：dev.hqyj.com

集团总部地址：北京市海淀区西三旗悦秀路北京明园大学校内 华清远见教育集团

北京地址：北京市海淀区西三旗悦秀路北京明园大学校区，电话：010-82600386/5

上海地址：上海市徐汇区漕溪路 250 号银海大厦 11 层 B 区，电话：021-54485127

深圳地址：深圳市龙华新区人民北路美丽 AAA 大厦 15 层，电话：0755-25590506

成都地址：成都市武侯区科华北路 99 号科华大厦 6 层，电话：028-85405115

南京地址：南京市白下区汉中路 185 号鸿运大厦 10 层，电话：025-86551900

武汉地址：武汉市工程大学卓刀泉校区科技孵化器大楼 8 层，电话：027-87804688

西安地址：西安市高新区高新一路 12 号创业大厦 D3 楼 5 层，电话：029-68785218

广州地址：广州市天河区中山大道 268 号天河广场 3 层，电话：020-28916067