让单片机运行速度更快一些

本文就如何提高单片机的运行速度与读者们展开探讨。

1 问题的提出

1.1 硬件技术背景

单片机的频率越来越高，RAM 的访问速度也来也快，但单片机系统的效率并不一定成比例的提高。

目前，使用的主流单片机有 80386EX(50MHz，外部地址/数据总线 16 位)、MPC860T(66MHz，外部地址/数据总线 32 位)以及 DS80C32(25MHz，外部地址/数据总线 8 位)；使用的 SDRAM 有 HY57 系列、K416 系列(访问速度 100MHz 或 133MHz)；使用的 SRAM 如 IDT71024、IDT7256(50MHz)；使用的 Flash 有 AT29C512、SST39VF040、AT29C010(8MHz 或 15MHz)等。可见，SDRAM，SRAM 的速度和单片机是匹配的，甚至比单片机的速度更快一点，不需要单片机插入等待状态。而 Flash 的访问频率则比单片机慢 2~6 倍，单片机往往要通过插入多个等待状态来和它相匹配，况且 Flash 多为 8 位，而当前单片机多为 16，32 位，更多的降低了单片机的工作性能。

根据上述分析，如果提高 Flash 的访问速度，扩展 Flash 为 16 位或 32 位，那么程序执行的速度就快了，单片机的性能也就提高了。如果能够将这一想法变成事实，而且成本低廉的话，那是最好不过的事情。事实上，可以将 8 位的 Flash 扩展为 16 位、甚至 32 位，但要付出 2~4 倍的成本。由于 Flash 结构及工艺原因，在目前不可能有高达 66MHz 的商用化且价格低廉的 Flash。所以，只能通过其它方式来提升单片机的运行速度。

1.2 软件技术背景

首先，看看传统单片机程序的运行原理，为了便于说明，假定硬件平台为 860T，时钟为 50MHz；SDRAM 空间 4M×32bit，地址范围从 0x00000000~0x00FFFFFF，访问时间 10ns；Flash 空间 512K×8bit，访问时间为 100ns,地址范围从 0x02800000~0x0287FFFF(至于其它单片机,运行原理大致相同,可以类推)。860T 在上电后,PC(Program Counter)=0x2800100，程序从 PC 指定的地方执行，首先执行初始化代码(BootCode)，再执行主程序(AppCode)。程序从 Flash 中读取指令(code)，来完成数据的传输——可能是 SDRAM 和内部寄存器的传输，SDRAM 之间的传输，SDRAM 和外设的传输，中断处理等各项工作。可见在程序运行时，很大一部分时间是从 Flash 中读取指令，而这个过程是很费时间的。以假定的 860T 硬件平台为例，因为 Flash 访问时间为 100ns，所以读一条指令的时间至少是 100ns，也就是说 860T 读一条指令的时候要等待 100ns。(指令 cache 通过预取指令的方式,可以使实际取指令时间短一些,但这种方法的效果并不明显,况且很多单片机还没有指令 cache。)

860T 平台的内存分配如图 1 所示。

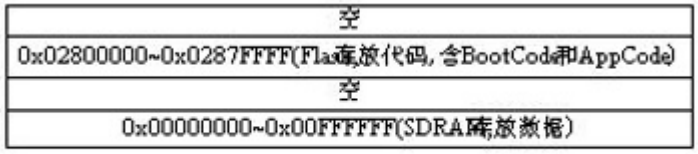

图1 传统单片机的内存分配模式

2 将代码从 Flash 搬运到 SDRAM 中的原理

通过上述分析，初始化代码 BootCode 只在程序启动的时候执行，就是慢一点，也可以接受。真正影响性能的是主程序(AppCode)，因为这里的代码在不停的重复执行，如果可以缩短它的取指令时间，则单片机的空闲时间将大大减少，性能也就提高了很多。SDRAM 的速度比较快，如果将代码搬运到 SDRAM 中，取指令时间就减少了很多；而且 SDRAM 空间大，不会因为代码占用了一部分空间而影响性能。但这不仅仅是简单的搬运过程，有物理存储器地址的变化牵涉在这个过程中。将软件源代码转换成可执行的二进制映像包括三个步骤：首先，每一个源文件都必须被编译或汇编到一个目标文件(object file)；第二步，所有的目标文件要连接成一个目标文件，它叫可重定位程序(relocation program)；最后，在一个称为重定址(relocation)的过程中，要把物理存储器地址指定给可重定位程序里的每个相对偏移处，生成一个可执行的二进制映像文件。如果在 Flash 中运行，则所有的物理存储器地址应该在 Flash 的地址空间中。如果要在 RAM 中运行，则所有的物理存储器地址应该在 Flash 的地址空间之中。也就是说，如果要使从 Flash 中搬运到 SDRAM 中的代码可用，则必须改变被搬运代码的物理存储器地址。

3 搬运代码的实现方法

下面结合假定的硬件平台，详细描述物理存储器地址重定位，代码搬运的原理及过程。我们编写两个 c 文件——RomTool.c、RAMapp.c。

RomTool.c 完成 860T 初始化，SDRAM 的刷新，中断及外设的初始化；Flash 到 SDRAM 的代码搬运驱动模块及跳转模块。对应的二进制映像文件为 RomTool.bin。

RAMapp.c 是实际的应用程序，对应的二进制映像文件为 RAMapp .bin。RAMapp.bin 被搬运后在 SDRAM 中运行。

3.1 物理存储器地址映射规则

RomTool.c 的物理地址映射规则为：数据放在起始为 0x3000，大小为 0xf0000 的 SDRAM 空间里；代码被烧结在起始为 0x02800000，大小为 0x10000 的 Flash 空间里，不会被搬运，也在该空间里运行。
所以在 RomTool.lnx 中指定的定位规则也应该是这个地址范围，如下：

```
MEMORY
{
ram1: ORIGIN = 0x00003000, LENGTH = 0xf000
flash: ORIGIN = 0x02800000, LENGTH = 0x1000
}
SECTIONS
{
.data : {} > ram1
.text : {} > flash
}
```

RamApp.c 的物理地址映射规则为：

数据放在起始为 0x3000，大小为 0xf0000 的空间里；代码被烧结在起始为 0x02810000，大小为 0x70000 的 Flash 中，它要被搬运到起始为 0x00F00000，大小为 0x70000 的 SDRAM 空间里，即 RamApp.Bin 实际在 SDRAM 中运行。

所以，在 RamApp.lnx 中指定的定位规则应该在 SDRAM 中，如下：

```
MEMORY
{
ram1: ORIGIN = 0x00003000, LENGTH = 0xf000
ram: ORIGIN = 0x00F00000, LENGTH = 0x7000
}
SECTIONS
{
.data : {} > ram1
.text : {} > ram
}
```

最后，在 860 单片机系统的地址映射规则如图 2 所示。对照图 1，可以观察到这和传统的程序地址映射有很大不同。

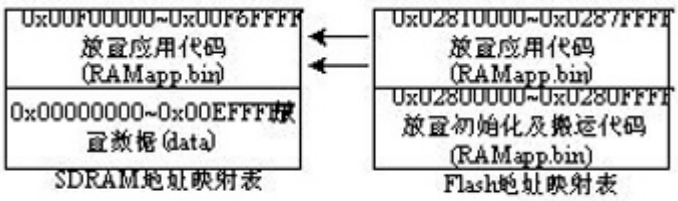

图2 地址映射表

3.2 搬运的过程

860T 上电复位，RomTool.bin 首先被执行，完成初始化工作后，运行代码搬运函数，将 RAMapp.bin 搬运到 SDRAM 中，随后改变 PC(Program Counter)的值，无条件转移到 SDRAM 中运行 RAMapp.bin，如图 3 所示。

图3 从Flash到SDRAM搬运代码的过程

3.3 搬运代码的驱动模块及跳转模块源代码

(1)搬运代码驱动模块的代码

```
void MoveCodeF_to_RAM(UWORD *FlashCode_Add, UWORD *RamCode_Add,UWORD CodeLen) {
do{
*RamCode_Add++ = *FlashCode_Add++;
CodeLen?
} while ( CodeLen!=0)
}
```

该段代码是将开始地址为FlashCode_Add,长度为CodeLen的Flash代码搬运到开始地址为RamCode_Add,长度为CodeLen的SDRAM 中。

(2)主函数及跳转模块

```
#define FlashCode_Add_V 0x02810000
#define RamCode_Add_V 0x00f00000
#define CodeLen_V 0x00070000/4
void main(){
UWORD *I=(UWORD *) FlashCode_Add_v;
UWORD *j= (UWORD *) RamCode_Add_v;
UWORD *k= (UWORD *) CodeLen_V;
MoveCodeF_to_RAM( (UWORD *) i, (UWORD *) j, (UWORD *)k );
# 跳转模块
asm(“addis r2,0,0x00f0”);
asm(“ori r2,r2,0x0000”); # 代码起始地址 0x00f00000
asm(“mtspr LR,r2”);
asm(“bclr 20,0”); # 无条件转跳到链接寄存器
# (LR)中的地址
}
```

FlashCode_Add_V：被搬运代码的首地址，在 Flash 中。
RamCode_Add_V：被搬运代码的目标地址，在 RAM 中。
CodeLen_V：被搬运代码的长度，按 32 位计算。

该函数在调用代码搬运 MoveCodeF_to_RAM 函数，将代码从 Flash 搬运到 SDRAM 中后，将程序指针转移到 SDRAM 中。注意跳转的地址一定要和 RamCode_Add_V 一致。

4 小 结

可见，正确完成代码搬运的关键在于：

① 确定被搬运代码的物理地址映射规则， 物理地址一定是在 SDRAM 中；
② 被搬运代码是被烧结在 Flash 中，后来又被搬运到 SDRAM 中；
③ 无条件转移到 SDRAM 中，运行被搬运代码(应用程序代码)。

对于其它型号的单片机，可以根据该原理类推，方法是一样的，只是具体的代码不同而已。相信你的单片机系统在经过这样的处理后，效率一定会高很多。