TMS320F2812混合编程的研究与应用

张 锐

(黑龙江科技学院 电信学院，哈尔滨 150027)

摘 要:F28系列DSP可以用C语言或汇编语言编程,但C语言的效率不高,而汇编的可读性和移植性又差,本文以对实时性要求很严的傅立叶变换为例,采用两种语言结合的编程方法来说明如何对F28系列DSP编程以及在编程中数据的定标问题,为了加快开发的速度,还讨论了TI的IQmath.lib(数学库)的使用方法。

关键词:傅立叶变换; DSP; 数据定标; 函数库

中图分类号:TP311 **文献标识码**:A **文章编号**:1008-8725(2008)01-0126-03

Research and Usage of TMS320F2812 Mixed Programming

ZHANG Rui

(Communication Institute, Heilongjiang Institute of Science and Technology, Harbin 150027, China)

Abstract: DSP of F28 series can be used to program by C language or assemble language. But because of C language is not-high efficiency and assemble language is bad readability and transplantation. This paper takes Fourier Transform which requires very strict real time performance as example and illuminates how to program DSP of F28 series by the method of combining two languages and scaling of data. The usage of IQmath.lib of TI is also disscufed in order to speed up development.

Key words: fourier transform; DSP; data scaling; function library

0 引言

F2812是目前为止控制领域最优秀的芯片。但是在实时性要求严格的场合,如果只用C语言编程也无法满足要求。文中以无功检测的快速傅立叶变换(FFT)为例,说明编写DSP程序的整体思路和应该注意的关键问题,以及一些常用的编程点。

1 C程序的编写

开发DSP程序是主体程序,应该由C程序来完成,而关键代码应由汇编完成。C程序主要由头文件、常量定义和内存开辟、子程序的调用等组成,其中子程序由汇编编写,由主程序负责调用,而调用子程序和子汇编程序的编写要符合一定的规则。

编写头文件,供主程序调用。头文件最主要的是对各个模块的寄存器进行定义,用C语言编写头文件,一定要使用关键词volatile,如果不使用,则在中断函数或非同一线程函数中对寄存器的读写将会无效,导致错误或死循环,这一点是与单片机最大的不同。例如对AD模块中的结果寄存器0定义可用如下语句:

```
volatile * ADCRESULT0 = (volatile * )0x007108;
```

其中,0x 007108为结果寄存器0的物理地址,在程序中用下面的方法使用:

```
unsigned int xx = * ADCRESULT0;
```

即可得到采样结果。其它的寄存器定义与此类似。

为数组开辟内存空间,这是编程中最重要的环节。例如采样后的数据要保存在一块内存数组中(a[64],以64点为例),然后将码位倒置后的数组放在另一块内存数组中(b[64])(下面用到的a,b均指同一数组)进行傅立叶变换,在主程序中,为两个数组声明开辟空间用如下语句(ffta和fftb可以相同):

```
# pragma DATA_SECTION(a,"ffta");
# pragma DATA_SECTION(b,"fftb");
unsigned int a[64];
unsigned int b[64];
```

然后在链接命令文件(.cmd)中把它们定位到指定的地址空间中去,本文把它们放到内部的L0存储器(SARAM)中,命令如下(没有用到的已省略):

```
MEMORY
 {PAGE 1 :
   RAML0ffta : origin = 0x008000,
       length = 0x000040
   RAML0fftb : origin = 0x008040,
       length = 0x000040
 }
 SECTIONS
 { Ffta : > RAML0fftuab PAGE = 1
   Fftb : > RAML0fftia PAGE = 1
 }
```

收稿日期:2007-10-17;**修订日期**:2007-11-16

作者简介:张锐(1977-),男,哈尔滨人,助教,毕业于黑龙江科技学院,现为黑龙江科技学院教师,E-mail: lowbird@qq.com。

这样就把两个数组放在连续的128个字(16位)中,实现了数组的内存开辟。链接命令文件的编写一定要谨慎,否则会出现严重的后果,最好在TI的原有模板上修改。

主程序调用汇编子程序,一定要符合调用规则,这也是编程中的难点。对于本文要用到的主要汇编程序如码位倒置程序和FFT程序,可用如下方法调用:在主程序中,首先声明它们为外部函数,如下所示:

```
extern void FFT(int *);
extern void reverse(unsigned int*, int *);
```

声明中,形参都为指针形式,在调用时,将形参变为实参即可,例如FFT(b),reverse(a,b),如果程序中需要其它函数,可仿照上面的两个进行声明和调用,如果是C程序调用C程序,则无需了解调用规则和底层机制,只要互相调用即可。

2　汇编语言的编写

TI说明书[1]中调用规则规定,如果有32位的参数,则第一个放在ACC(累加器)中,其余的32位参数按照反序放在堆栈中;指针放在XAR4和XAR5(辅助寄存器)中,其它的指针参数放在堆栈中。文中所用到的参数全为指针类型,对于reverse()函数,指针a放在XAR4中,指针b放在XAR5中;对于FFT()函数,指针b放在XAR4中。

对于该文的码位倒置函数reverse()中,要充分利用芯片的资源,应用专门的指令(*BR0++)来实现,此指令自动实现反向进位加,节省了时间。对于本文,用以下几条指令即可完成码位倒置:

```
参数以分别放到XAR4和XAR5中
MOV @AR1, #(64-1) ;实现64次循环
MOV @AR0, 64/2 ;采样点数的一半装入AR0寄存器中
LOOP: NOP *, ARP4 ;设置当前的辅助寄存器为AR4
MOV AL, *+++ ;将XAR4指向的值(要码位倒置的数)放到AL(累加器低16位)中
NOP *, ARP5 ;设置当前的辅助寄存器为AR5
MOV *BR0++, AL ;将AL中的数放到AR5+AR0所指向的地址中,即实现了码位倒置
NOP *, ARP1 ;设置当前的辅助寄存器为ARP1
BANZ LOOP, *-- ;如果AR1中的值不为零,则继续循环。
```

这里XAR1寄存器需要保护,在下面详述。

对于DSP而言,FFT程序是考察其性能的一个指标,因为FFT程序涉及到了编程的方方面面及硬件的性能。文中FFT函数的参数为码位倒置后的数据首地址,变换后的数据仍然放在原处,即可以实现同址运算。

FFT运算需要大量的中间结果,F2812中有8个辅助寄存器(XAR0XAR7),1个累加器(ACC),1个被乘数寄存器(XT)和1个乘积寄存器(P),但是这在FFT运算中是不够的,这就需要用到堆栈(SP)寻址模式(*-SP[6-bit数]),此种模式实际上就是起到一个临时变量的作用,其原理如图1所示,由于堆栈指针SP指向下一个空地址,所以如果需要N-1个临时地址,则需要将SP加上N,由于在运算过程中,大部分操作都要经过累加器,所以一般情况下,都是SP与累加器进行数据交换,累加器将运算完的且在下几步中还要用到的数据保存在*-SP[N]中,当需要时再取出来,这样节省了运算的时间。堆栈寻址模式的使用方法就是在子程序法开始将堆栈指针SP加上所需的地址数后再加上1,然后在程序返回前减去所加的数即可。对于其它的寄存器操作,只有XAR1XAR2和XAR3需要保护(当在程序中用到了它们),方法为在程序开始时将其压入堆栈,在程序返回时将其弹出堆栈,其它的寄存器不需要保护,系统会自动进行保护,除此之外,其它的指令只需按照顺序执行即可,具体的使用方法如下所示:

```
.def FFT ;定义符号FFT,注意前面一定要加下划线"-",因为此函数要被C函数调用,如果不被C调用,可以不加
.global -FFT ;定义为全局符号
-FFT: ;一定要顶格写,表示子程序的开始
PUSH XAR1 ;XAR1压入堆栈
PUSH XAR2 ;XAR2压入堆栈
PUSH XAR3 ;XAR3压入堆栈
ADDB SP, #8 ;需要7个临时地址
……
MOV *-SP[1], AL ;保存AL
MOV *-SP[2], AH ;保存AH
……
MOV AL, *-SP[1] ;调出保存的AL
MOV AH, *-SP[2] ;调出保存的AH
……
SUBB SP, #8 ;将堆栈指针还原
POP XAR3 ;从堆栈弹出XAR3
POP XAR2 ;从堆栈弹出XAR2
POP XAR1 ;从堆栈弹出XAR1
LRETR ;子程序返回
```

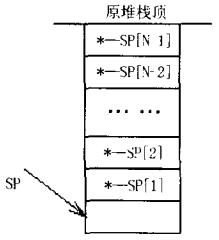

图1 堆栈寻址模式原理图

3 数的定标

在实际的工程应用中,所有的数据都要经过整理和变换之后才能得到所要的结果,而大部分文献都很少涉及这个问题。F2812为定点芯片,而实际的数据为浮点数,所以要将浮点数转化为定点数之后才能运算,这就涉及到数的定标问题。

文中采样之后的数的范围为0:3,经过整理之后的码位倒置的数的范围为±1.5。在用Q形式表示定点数时,Q值越大,表示的数的精确度就越高,如果为了编程的简单化而只用一种Q形式表示,那么只有牺牲精度,用$Q0$表示时精度为1,而用$Q15$表示时,精度为$1/2^{15}$,可见相差很大。本文可用$Q14$的形式表示码位倒置的数,因为$Q14$表示的范围为-2:1.999 939 0,满足要求。当运算时,这个数和正弦值或余弦值相乘,因为正弦和余弦值的范围在±1之间,可以把它们用$Q15$格式表示(表示范围在-1:0.9999695),这样为$Q14\times Q15$形式,众所周知,经过傅立叶变换后的数的最大范围不会超过原来数的2.414倍,在每一步中可以用2进行归一化,这样得到的数的最大范围为$1.5\times1.414=2.121$,得到的数仍用$Q14$格式表示,基本上不会溢出。

由定点数乘法公式,当X的定标为QX,Y的定标为QY,Z的定标为QZ,则Z的结果应该右移($QX+QY-QZ$)位,本文中$QX=QZ=14$,$QY=15$,所以需要右移15位,而先前得到的乘积为32位(放在ACC中),已经进行了一次算术左移,去掉了一位符号位,这样只要再取高16位即可,无需在进行移位即可完成数据的定标。在主程序中,如果想得到浮点数,用下面的方法即可(假设Z为得到的$Q14$格式的定点数,Y为要得到的浮点数):

$Y=(float)\ Z>>14$

这样最大限度的保证了计算的精度。

4 应用函数库加快开发

TI公司推出的几个标准库可以缩短用户的开发周期,文中使用了其数学函数库(IQmath.lib),用它来进行数的运算,因为这个函数库是高精度、高优化的函数库,它提供了标准的C/C++语言的无缝接口,用在计算密集型的应用程序中。

假如已经得到了电流和电压的变换后的傅立叶值,分别为U[64]和I[64](Q14形式),以C语言的形式调用库中的函数,易于编程,而函数全由汇编完成,效率高,这就使原来的问题得以解决。使用方法是在程序中加入头文件IQmathlib.h,然后将IQmath.lib加入到工程中,因为库中用到查找表IQmathTable和代码段IQmath,所以相应的命令文件中应加入下面的代码:

```
MEMORY
  { PAGE 0:
    BOOTROM : origin = 0x3ff000,
      length = 0x000fc0
    RAMH0 : origin = 0x3f8000, length = 0x002000
  }
SECTIONS
  {
    IQmathTables : load = BOOTROM,
      type = NOLOAD, PAGE = 0
    IQmath : load = RAMH0, PAGE = 0
  }
```

本例中计算电压的实部和电流的实部(实部在前,虚部在后)相乘,然后求其和,虚部和虚部相乘,然后求其和,使用方法如下:

在程序前面加入代码:#define GLOBAL_Q 14,即定义Q形式为Q14,_iq y1,y2;

```
for ( i=0; i<64/2+1; i++)
{ y1 += _IQrsmpy( U[2*i] , I[2*i] );
y2 += _IQrsmpy( U[2*i+1] , I[2*i+1] );
}
```

函数_IQrsmpy实现两个Q值相同的定点数相乘,得到Q值相同的定点数,其它的函数使用也大致相同,易于应用,不再叙述。

5 结论

通过FFT变换实例论述了如何对F2812进行C语言和汇编语言的编程,以及C语言调用汇编语言的编程,最后又说明了怎样对定点数进行定标,如何应用TI数学库进行定点数的相乘运算,使之最大限度的保证精度和速度。

参考文献:

[1] Texas Instruments Incorporated ,TMS320C28x DSP CPU and Instruction Set reference Guide[Z]. August,2001,(7-14).
[2] Texas Instruments Incorporated ,TMS320C28x Optimizing C/C++ Compiler User's Guide[Z]. August,2001.
[3] Texas Instruments Incorporated , TMS320C28x Assembly Language Tools User's Guide[Z]. August, 2001.
[4] Texas Instruments Incorporated ,Module user's Guide C28x Foundation Software[z]. June, 2002.