实验三 基于DSP的系统软硬件协同设计

王亭午

无210

2012011018

1. **实验目的**

**1.1 实践 C 语言主程序调用汇编语言子程序的程序设计方法,重点是用于参数传递的寄存器和堆栈的使用方法。**

**1.2 学习 DSP 实时系统的软硬件协同设计方法,了解数据传输和系统控制的设计方法。**

**1.3 设计前后台工作方式的实时系统,了解中断和中断服务子程序调用过程**

1. **实验材料**

**2.1 实验中使用的 DSP 集成开发系统软件版本为 ADI VisualDSP + + 4.5**

**2.2 软件源程序**

a) 本次实验采用位于安装目录下的\Analog Device\VisualDSP 4.5\Blackfin\Example\Tutorial\fir工程中的 fir\_test. c 和 fir. Asm. b) 第二次实验完成的 fir\_Smac\_asm. Asm. c) Talk\_Through\_533\_I2S 工程文件

**2.3 实验用配件**

a) 评估板 BF533EZ − kit. b) 一对标准 RCA 接口的 2 转 1 和 1 转 2 立体声音频连接线(已配好). c) 耳机. d) 音乐播放器

**2.4 参考资料**

a) 课程课件第三讲《基于数字信号处理器的系统设计 03 —— DSP 系统设计》b) 实验指导书. c) VisualDSP + + 4.5 自带的帮助系统. d) AD1836 Datasheet

1. **实验内容**

**3.1 任务一：子程序调用时的堆栈管理和参数传递**

编译器输出临时文件 .s 文件和源 .c 文件的对比;预处理命令的执行结果;各种寄存器的初始状态;存储器中各 section 的加载情况; C 调用汇编时,存储器中堆栈段的变化; fir\_test.c调用 \_fir.asm时,参数的传递情况。

**3.1.1实验步骤**

1. 建立新工程,加载\Analog\Device\VisualDSP 4.5\Blackfin\Example \Tutorial\fir中的源代码。fir\_test.c 为 C 语言主程序,调用汇编子程序 fir.asm。filter.h 为宏和函数定义头文件;fir\_coeff.h 为系数定义头文件; fir\_input.h 为输入数据头文件; mds\_def.h 为系统常量和数据类型定义头文件; tfilter.h 为函数原型定义和测试错误定义头文件。

(2) 设置工程选项,选择保存编译器输出的临时文件 fir\_test.s在 project option 中的 complie\general 设置中,选中save temporal file执行编译后,即可在 Debug 目录下看到与 C 源程序同名的 .s 文件。虽然新生成的文件在 Windows 环境下的图标中显示为 . asm 文件,然而利用 Visual Studio打开时可以发现该文件为 .s 文件,正如实验指导书中的描述一样。

(3) 对比C源程序和.s文件。Build project完成后,打开\Analog Device \VisualDSP 4.5\Blackfin\Example\Tutorial\fir\Debug\fir\_test. s对照 fir\_test.c ,了解编译器预处理结果和调用汇编程序的接口管理

答案：我们得到的汇编代码如下：

|  |
| --- |
| .file "fir\_test.c";  // Compilation time: Tue Jan 01 01:50:51 2002  // Compiler options: -c -file-attr ProjectName=proj3 -g -save-temps -path-output Debug -structs-do-not-overlap -no-multiline -double-size-32 -decls-strong -warn-protos -proc ADSP-BF533 -o Debug\fir\_test.doj -MM  // Architecture: ADSP-BF533  // Silicon revision: 0.3  .section/DOUBLE32 program;  .epctext:  .align 2;  \_main:  .LN\_main:  .reference \_h;  .reference \_delay;  .reference \_IN;  .reference \_OUT0;  .reference \_OUT1;  .reference \_OUT2;  //---------------------------------------------------------------// Procedure statistics:  //  // Frame size = 52  //  // Scratch registers used:{R0-R3,P0-P2,ASTAT,CC}  //  // Call preserved registers used:{R7}  //  // Registers clobbered by function calls:{R0-R3,P0-P2,I0-I3,B0-B3,M0-M3,ASTAT,SEQSTAT,RETS,CC,A0-A1,LC0-LC1,LT0-LT1,LB0-LB1,SYSCFG,CYCLES,CYCLES2}  //--------------------------------------------------------------- |

**3.1.2 实验问题**

**问题 1.1 请找出程序中几个参数的存放位置和存储器地址**

**答案：**我们只需要对照我们的汇编中的几行代码就可以了，如下：

|  |
| --- |
| // "fir\_test.c" line 32 col 5  R3 = 260 (X);  [FP+ -8] = R3;  .LN2:  // "fir\_test.c" line 33 col 5  R2 = 8;  [FP+ -4] = R2;  .LN3:  // "fir\_test.c" line 35 col 2  R1.L = \_h; R1.H = \_h;  [FP+ -24] = R1;  R0.L = \_delay; R0.H = \_delay;  [FP+ -20] = R0;  [FP+ -16] = R2;  .LN4:  // "fir\_test.c" line 37 col 2  R7 = 0;  [FP+ -12] = R7; |

可以看出，程序中共6个参数，依次是nsamples, taplength, h, delay, IN, OUT。对照我们的寄存器窗口，它们依次存放在 R3，R2，R1（首地址FF900004），R0（首地址FF900618）。返回函数的时候IN在R0 中，首地址为 FF900014。OUT数组的首地址存放在 R1 中，首地址为FF900628。

**问题 1.2 FP 和 SP 有什么不同？函数调用时，有 4 个参数需要通过主程序传给汇编子程序，它们是如何传递的？R7 中存放的是什么地址?**

**答案：**根据课件我们可以知道：FP 是帧指针，是当前堆栈空间的基地址，在堆栈空间未被一般没有发生调用的情况下是不会发生变化的。大多数情况下，不受程序中参数压栈、出栈的影响（见课件），可以用于参数传递的跟踪。而SP是栈顶指针，有时我们通过它来传递参数，存储的则是当前的堆栈地址。

在汇编代码中我们可以看见，IN, OUT, s依次存入 R0, R1, R2 寄存器中。nsamples被放入堆栈中，以数值的形式进行传递的。同时我们可以看到，R7 中存放s的首地址。

**3.2 任务二：子程序调用时的堆栈管理和参数传递**

在 DSP 的地址发生器中,通过模运算控制存储器指针,用以减少 FIR 运算时同时对存储器访问的请求次数。等效于在线性排序的地址空间中,开设环形缓存器。 Blackfin 利用变址寻址寄存器( Index registers ) I0 − I3, B0 − B3, L0 − L3, M0 − M3 实现模寻址过程。

**3.2.1实验步骤**

(1) 继续任务一中的程序运行,进入 fir.asm 子程序后,打开DAG register, Data register, P register等观察窗口。

(2) 单步执行程序至 L 寄存器赋值完毕( L3 = R2 ),观察上述寄存器的变化继续单步运行至 fir. asm循环体,增加观察窗口 Loop counters 和 memory。观察循环过程中:存储器 delay 存储区和 IN 存储区的对应关系,理解延迟线的概念；同时对比观察 h 存储区，I2 和 R2 寄存器的内容，理解模寻址和环形缓存器的概念。

**答案：**延迟线是FIR中的一个概念，它保存了输入数据的信息，作为延迟单元。R2寄存器中保存了相邻的滤波器系数h。每次运算都会根据循环寻址结果，读入一个滤波器的系数。**3.2.2实验问题**

**问题 2.1 L 寄存器中的内容是什么参数？其数值和主程序中传递过来的参数有什么对应关系?这段子程序中开设了几个环形缓存器？**

**答案：**L寄存器中的内容是进行模循环寻址Length。其数值是主程序中传递过来的参数个数的X2，这是因为这个数字是采用定点小数来存储的，因此实际上有一个系数差别。这段子程序，使用了4个环形寄存器。

**问题 2.2 计算 MAC 的循环起始和结束的计算指令是什么?**

**答案：**分别是：

|  |
| --- |
| A1+=R0.L\*R2.H, A0+=R0.L\*R2.L || R2.L=W[I2++] || R0=[I1--];  A1+=R0.H\*R2.L, A0+=R0.H\*R2.H || R2.H=W[I2++]; |

**问题 2.3 在 fir.asm 中两层嵌套循环结束后,为什么要对 L 寄存器清零?**

**答案：**因为我们不希望这个子程序的循环设置影响到外面汇编程序的正常进行，因此我们把这几个循环设置都清0，保证外面的程序可以顺心所欲进行。

**3.3 任务三 观察指令流水线控制**

将指令分解成一个基本的任务序列,序列中的每个任务由相应的功能单元执行,构成指令流水线。每个时刻,可以有一个新的指令进入流水线,同时有一个指令被执行完毕,提高了指令的执行效率。但是,由于结构、数据和控制的冒险情况,将引起流水线冲突发生,从而妨碍后续指令的执行。当邻近汇编指令的执行发生冒险时,为防止流水线执行错误,硬件设计上采用互锁( inter − locking )技术,产生有条件的延迟,控制对下一条指令进入流水线的时刻。集成开发环境中的 pipeline viewer 提供了流水线事件的观察窗口。

**3.3.1实验步骤**

1. 将工程文件复位(reset)，打开 view\Debug windows\pipeline viewer 观察窗口。重新build project,单步运行,观察函数调用时的流水线事件,通过帮助文件,理解该事件发生的原因。

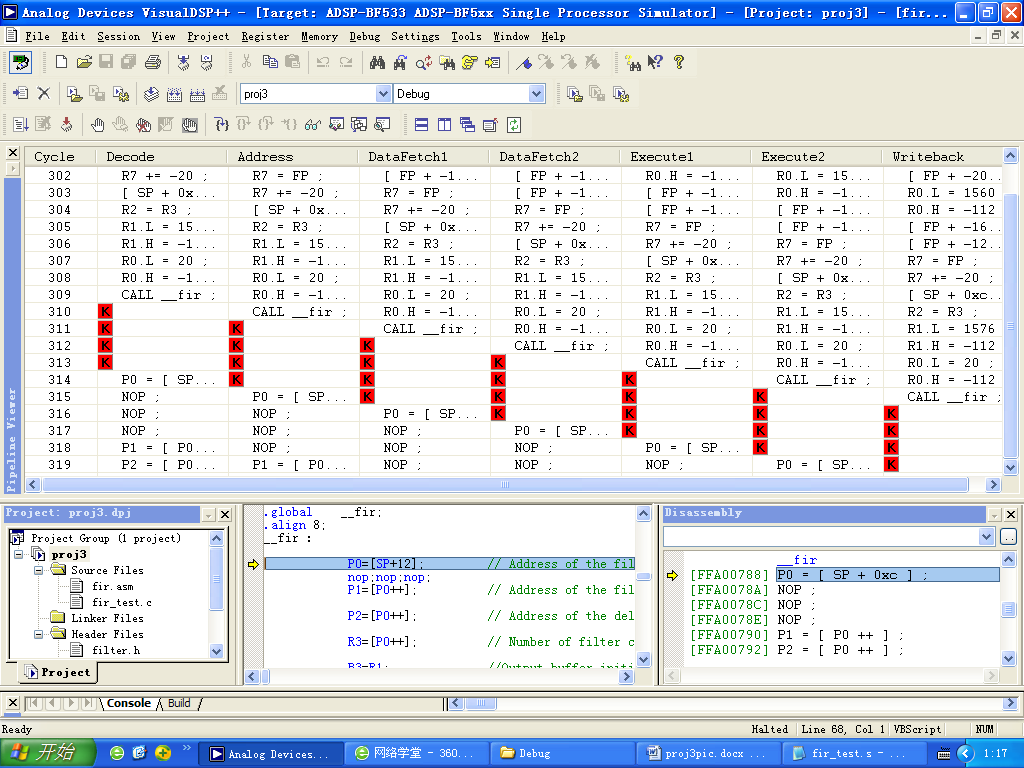


图1

(2)修改汇编子程序中的 NOP 指令,重新 build project,观察修改处的流水线时间,并通过帮助文件理解事件发生的原因。

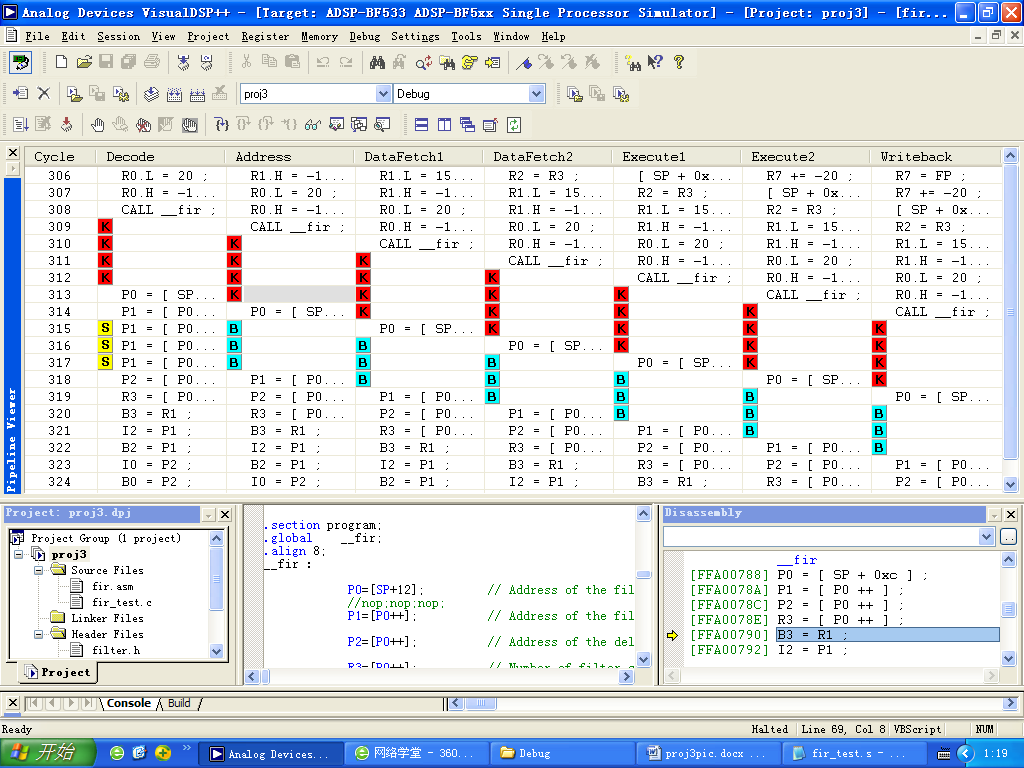


图2

**3.3.2 实验问题**

**问题 3.1 NOP 的个数由什么决定？程序中不插入NOP会发生什么情况？NOP的作用如**

**何体现?**

**答案：**NOP的个数由我们拥有的流水线的级数和我们实际架构（比如转发和冒险机制）有关。当我们不主动进行NOP操作的时候，我们的汇编器将产生3个bubble防止程序发生错误。如果这个时候没有bubble，那么程序将会数据冒险错误，因为我们有连续的两段代码P0 = [SP + 12]和 P1 = [P0 ++]。

NOP，bubble通过让几个time slot阻断新程序流水，实现防止冒险。

**3.4 任务四 分析计算 FIR 滤波的汇编子程序入口**

将第二次实验任务五中完成的汇编子程序 fir\_Smac\_asm.asm 载入工程,替换 fir.asm ,改变fir\_Smac\_asm.asm 中使用的 DAG 寄存器和数据寄存器,另存 myfir\_Smac\_asm.asm 子程序。

**答案：**我们只需要注意到：DAG 寄存器中每一组 I, M, L, B寄存器是一个整体，然后我们就可以依照此原则对程序进行修改：将I0 M0 L0 B0用I1 M1 L1 B1替换。考虑到我们之使用了I0，实际上我们只需要修改I0为I1就可以了。

**3.4.1 实验问题**

**问题 4.1 如何验证 myfir\_Smac\_asm.asm 可以完全替换 fir.asm?**

**答案：**为了验证结论的一般性，我们只需要验证他们的单位样值响应，就可以确认他们的完全替代性。我们现在假设输入是一个有限长度的单位样值响应，即IN=dirac(x)。我们设这个IN的长度为260，只有IN[0] = 1, 其他的为0，这个时候，两个fir的输出如下：

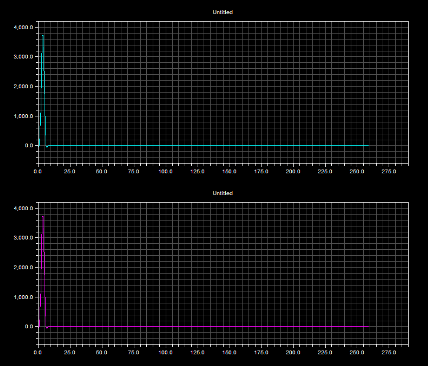


图3，两个汇编代码冲击响应图

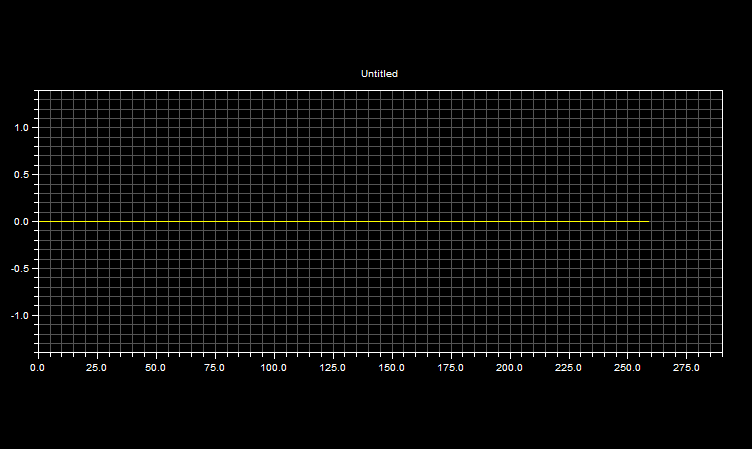


图4，单位样值响应差图

可见，两个的单位样值响应是一样的，同时我们注意到我们的参数长度小于我们的IN长度，我们可以判断，当我们的输入IN的长度是无限长的时候，仍然成立，得证。

**3.5 任务五 实时 DSP 系统数据接口和控制接口配置**

分析第一次实验中的 Talk\_Through\_533\_I2S 工程文件。

**3.4.1 实验步骤**

1. 选择评估板 ADSP-BF533 EZ-kit 或 BF561EZ-kit,按第一次实验指导书描述的方法连接系统,播放音乐,运行 Talk\_Through\_533\_I2S 或Talk\_Through\_561\_I2S工程文件,保证能正常听到播放的音乐。

**答案：**我所选用的评估板的编号不记得了，只记得是一个BF553-EZLITE的板子，REV=1.6。

(2) 观察并阅读理解 AD1836 的控制寄存器(函数 Init1836 ),配置 AD1836 控制寄存器的信号被组织成一个数组,找到这些控制字,并理解每个控制字对 AD/DA 配置的具体内容。

(3) 分析 DMA 通道的初始化函数, 理解系统的 DAM 使用方法。

(4) 通过查阅评估板电路原理图和观察 SPI 的初始化函数,理解 DSP 和 AD1836 的 SPI端口连接和配置

(5) 通过查阅评估板电路原理图和观察 SPORT 的控制寄存器(函数 Init\_Sport0 )，理解 DSP和 AD1836 的 SPORT 端口连接和数据传递过程,运行程序,单步执行至函数 Init\_Sport0内部。 继续单步执行 Init\_Sport0 代码,跟踪系统编译出来的汇编指令。

**3.5.2 实验问题**

**问题 5.1 这里配置的 A/D 采样率和量化级数各是多少?**

**答案：**这里只需要查看课件的相应位置，于是我们可以得到ADC\_CONTROL\_1 = 0xC000 , ADC\_CONTROL\_2 = 0xD000。插标得A/D采样率为48kHz，量化字长为 24 bit。量化级数为 224。

**问题 5.2 CD 标准被称为 Red Book ,早在 1980 年由 Philips 和 Sony 公司联合制定的数字音频标准( Compact Disc Digital Audio system )，该标准规定的音频指标为2-channel signed 16-bit Linear PCM sampled at 44.100 Hz。如果要使用 ADSP − BF533 EZ − kit 获得满足 CD 音频质量的信号，应该如何配置相关寄存器参数？按照你选择的参数配置系统参数,播发音乐,运行程序,保证听到正常的音乐。**

**答案**：同理，通过查表我们知道我现在需要考虑一个AD的设置，由输出采样率44.100 Hz，量化字长为 16 bit我们查表可以知道De\_emphasis = 01，Data\_Word\_Width = 10，所以我们得到DAC\_CONTROL\_1 = 0x0110。

**问题 5.3 High Definiton Compatible Digital 是由微软公司在 CD 标准基础上扩展的专利技术,音频质量可以达到 120dB 的动态范围。在 PC 机上,用 Windows Media Player software 配合 24-bit 声卡可以播放 HDCD 音频信号。如果采用过采样技术,能获得更好的 HDCD 音质。如何配置 ADSP - BF533 EZ - kit的相关寄存器以便获得满足 HDCD 音频质量的信号？按照你选择的参数配置系统参数,播发音乐,运行程序,保证听到正常的音乐。**

**答案**：现在我们考虑的是ADC的问题，不过原理是一样的，查表得：De\_emphasis = 01 , Data\_Word\_Width = 00，ADC\_CONTROL\_1 = 0x0100。

**问题 5.4 在 Talk\_Through\_533\_I2S 工程文件中, DMA 控制器是被怎样使用的?用到了几个 DMA 通道?各是怎样被使用?**

**答案：**控制器实际上就是一个被传递的数组，名字为Codec1836TxRegs。这个数组通过DMA总线和 DMA 控制器被传递到SPI 端口。见课件我们知道，一共用到了 3 个 DMA 通道DMA2 Channel0负责SPORT 端口的接收，DMA2 Channel1负责SPROT端口的传输而DMA2 Channel4负责SPI 端口的传输。同时根据我们的实验文档我们知道，在 BF533 的板子上,DMA5 被用于 SPI 的控制传输\*pDMA5\_PERIPHERAL\_MAP = 0x5000。

**问题 5.5:请描述 DSP 和 AD1836 的 SPI 端口硬件连接关系**

**答案：**我们翻到课件的39页，可以看到这个图。

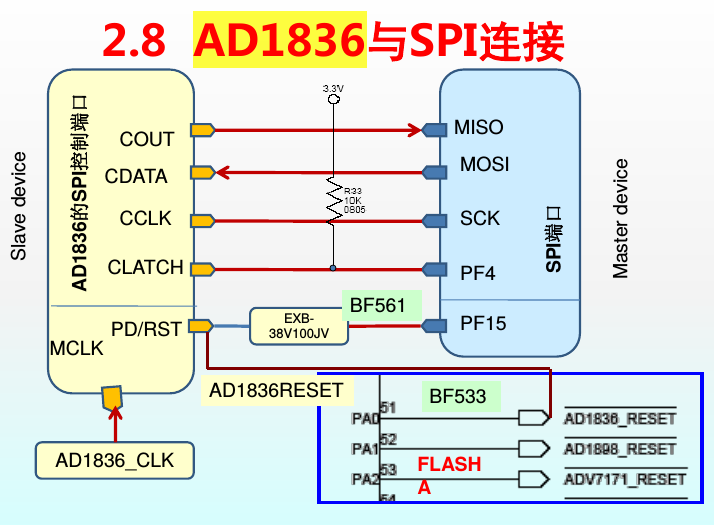


图5

其中 AD1836为从设备，而我们的DSP为主设备。端口COUT输出连接的是数据线 MISO，CDATA 输入连接的是数据线MOSI，CCLK 和 SCK 是两个时钟线。

**问题 5.6:该系统中配置的 SPI 端口的工作参数是什么?各是怎样配置的?**

**答案：**SPI\_CTL:使能、选择主/从、确定传输模式和传输字长(8 or 16bit)。SPI\_FLG:从机选择标志。SPI\_STAT:检测传输完成或发生错误。SPI\_TDBR:传送数据缓存。SPI\_RDBR:接收数据缓存。SPI\_SHADOW:接收缓存的影子,调试软件使用。

配置如下：

|  |
| --- |
| // Enable PF4  \*pSPI\_FLG = FLS4;  // Set baud rate SCK = HCLK/(2\*SPIBAUD) SCK = 2MHz  \*pSPI\_BAUD = 16;  // configure spi port  // SPI DMA write, 16-bit data, MSB first, SPI Master  \*pSPI\_CTL = TIMOD\_DMA\_TX | SIZE | MSTR;    // Set up DMA5 to transmit  // Map DMA5 to SPI  \*pDMA5\_PERIPHERAL\_MAP = 0x5000;    // Configure DMA5  // 16-bit transfers  \*pDMA5\_CONFIG = WDSIZE\_16;  // Start address of data buffer  \*pDMA5\_START\_ADDR = (void\*)sCodec1836TxRegs;  // DMA inner loop count  \*pDMA5\_X\_COUNT = CODEC\_1836\_REGS\_LENGTH;  // Inner loop address increment  \*pDMA5\_X\_MODIFY = 2;    // enable DMAs  \*pDMA5\_CONFIG = (\*pDMA5\_CONFIG | DMAEN);  // enable spi  \*pSPI\_CTL = (\*pSPI\_CTL | SPE);    // wait until dma transfers for spi are finished  for (j=0; j<0xaff; j++);    // disable spi  \*pSPI\_CTL = 0x0000; |

**问题 5.7 跟踪 Init\_Sport0 中每个寄存器是在什么内存地址被修改的,通过观察对应内存地址,看最终被修改成了什么内容?其具体含义是什么?**

**答案：**通过mixed功能查看系统编译出来的汇编指令，如下（可放大）。

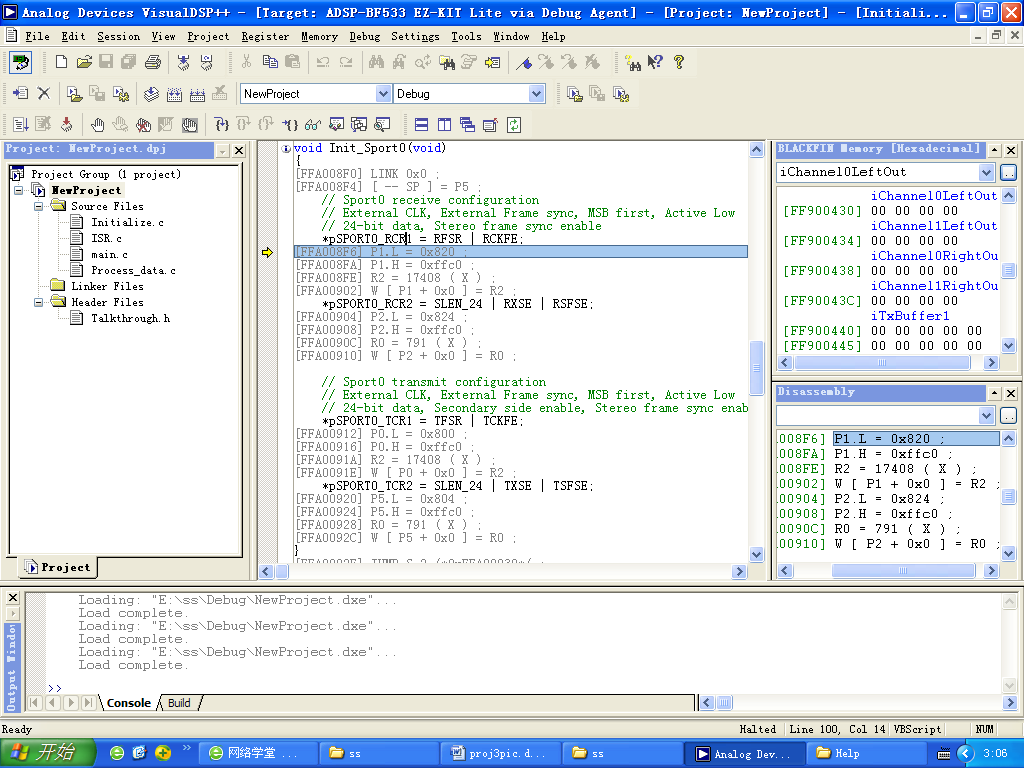


图6，可放大

可以发现,这些操作是将 17408 和791等控制数据写入到0xffc00820，0xffc00800 和0xffc00824 与 0xffc00804 中。通过这样的写操作，将 sport 的控制字传入进行设置。

**问题 5.8 请问在头文件 defBF561.h 中,对 Serial port masks 的定义起到什么作用？对SPORT0 Controller 的定义又有什么作用?对 DMA1 Controller registers 的定义有几组？这些不同的组号有什么实际意义吗?**

**答案：**在这个头文件中，实际上就是把一些复杂的地址和参数进行宏定义，成为比较好理解的英文变量形式，比如对Serial port masks 的一系列设置参数进行了宏定义,控制字定义成对应的英文简写，对SPORT0 Controller 中对各个数据寄存器、控制寄存器进行宏定义，将地址定义为英文简写。

我们可以看到：对 DMA1 Controller registers的定义有12组，代表了 DMA 的 12 个通道。

**问题 5.9 请描述 DSP 和 AD1836 的 SPORT 端口硬件连接关系**

**答案：**连接关系有两条路：发送的三个D/A 数据连接为：DT0PRI-DSDATA1，DT0SEC -DSDATA2，DT1PRI-DSDATA3;同时接收的两个立体声 A/D 数据连接为DR0PRI-DSDATA1，DR0SEC-DSDATA2。

**问题 5.10 该系统中配置的 SPORT 端口的工作参数是什么?各是怎样配置的?**

**答案：**在接收端规则接收外部时钟和外部同步帧，同时采用 MSB 优先，低电平有效的设置。

在发送端，转发外部时钟和外部同步帧; 也采用MSB 优先，低电平有效，两侧声立体声均有效。接受和发送都是24 bit 传输。设置代码如下：

|  |
| --- |
| // Sport0 receive configuration  // External CLK, External Frame sync, MSB first, Active Low  // 24-bit data, Stereo frame sync enable  \*pSPORT0\_RCR1 = RFSR | RCKFE;  \*pSPORT0\_RCR2 = SLEN\_24 | RXSE | RSFSE;    // Sport0 transmit configuration  // External CLK, External Frame sync, MSB first, Active Low  // 24-bit data, Secondary side enable, Stereo frame sync enable  \*pSPORT0\_TCR1 = TFSR | TCKFE;  \*pSPORT0\_TCR2 = SLEN\_24 | TXSE | TSFSE; |

1. **实验总结**

**4.1 总结实验中遇到的问题和解决办法**

**答案：**最大的问题是533上面的接受和发送端插反了。这个时候我们的直连去听，是没有问题的，但是一旦让DSP处理，就听不见。当时我们调了非常久的时间，一度怀疑是代码的问题，驱动的问题，最后还换了一个板子。最后终于是老师亲自帮我们调了一段时间，发现了这个问题。个人总结就是自己对输入输出的端口太不了解了，浪费了特别多的时间。

其次就是感觉这个课的课时有点不够，每次上课讲的东西都太多了，很难一次消化，在做实验的时候，总是要摸索很长的时间。

其次就是看汇编代码的时候花了很长的时间，最后经过隔壁组提醒才发现，原来上面的注释有对应的汇编之前C代码的行数。当时瞬间就能够读了，感觉观察应该更加细致一点才是。

**4.2 本次实验体会与建议**

**答案：**感觉几个实验环环相扣，不断深入。不过要提一个意见，很多上一次实验遇到的问题，光是看上一次的课件和指导书是无法知道的，我们花了成倍的时间在上面钻研。比如上一个实验中，我们花了很长的时间来研究为什么汇编代码里面的循环没有复位，而这个知识点确实这个实验里面的重点之一。我觉得应该在上一个实验就讲清楚，至少得提一下这个知识点。

其他的地方我感觉都是非常不错，实验设计的非常好。同时也感谢和我们一起调代码的助教老师和老师！