

实验准备

第一章 CCS 介绍

1.1 CCS 功能

CCS 是 TI 公司推出的功能强大的软件开发环境,现在该集成软件环境可以用于 TI 各系列 DSP 系统的软件程序开发。CCS 主要具有以下特性和功能:

- ☆ 集成可视化代码编辑界面,可以直接编写 C/C++、汇编、头文件以及 CMD 文件等;
- ☆ 集成代码生成工具,包括汇编器、C 编译器、C++编译器和链接器等;
- ☆ 集成基本调试工具,可以完成执行代码的装入、寄存器和存储器的查看、反汇编器、变量窗口的显示等功能,同时还支持 C 源代码级的调试;
- ☆ 支持多 DSP 的调试;
- ☆ 集成断点工具,包括设置硬件断点、数据空间读/写断点,条件断点等;
- ☆ 集成探针工具 (Probe Points),可用于算法仿真,数据监视等用途;
- ☆ 提供代码分析工具 (Profile Points),可用于计算某段代码执行时消耗的时钟数,从而能够对代码的执行效率做出评估;
- ☆ 提供数据的图形显示工具,可绘制时域/频域波形等图像;
- ☆ 支持通过 GEL (通用扩展语言) 来扩展 CCS 的功能,可以实现用户自定义的控制面板/菜单、自动修改变量或配置参数等功能;
- ☆ 支持 RTDX (实时数据交换) 技术,可在不中断目标系统运行的情况下,实现 DSP 与其他应用程序 (OLE) 间的数据交换;
- ☆ 提供开放式的 plug-ins 技术,支持其他第三方的 ActiveX 插件,支持包括软件仿真在内的各种仿真器 (需要安装相应的驱动程序);
- ☆ 提供 DSP/BIOS 工具,增强了对代码的实时分析能力,如分析代码的执行效率、调度程序执行的优先级、方便了对系统资源的管理或使用 (代码/数据空间的分配、中断服务程序的调用、定时器的使用等等),减小了开发人员对 DSP 硬件知识的依赖程度,从而缩短了软件系统的开发进程。

1.2 CCS 界面

CCS 的主界面如图 1.1 所示。

工程管理器主要用于统一管理各工程中所包含的文件,在工程管理器窗口中,可以添加、删除、激活和编辑工程中的源文件,同时也可以对编译器、汇编器和链接器的参数进行设置。管理器可以同时打开多个工程。但是,当前只能有一个工程是有效的。

调试工具栏集成了程序员调试 DSP 软件时最常用的调试命令。

输出窗口可以用来输出或者显示编译/汇编/链接过程中的各种信息、输出 C 语言标准输出函数的运行结果以及调试过程中出现的错误信息 (例如断点设置错误等)。

变量观察窗口可以观察程序中变量的地址或者数值,其中 Watch Locals 标签页窗口中会自动显示当前堆栈帧中的所有局部变量。程序员也可以在这个窗口或者其他 Watch 窗口中添加其他需要观察的变量,同时,还能根据需要设置其显示的数据格式。堆栈切换窗口主要用于各个堆栈帧之间的切换,因为当前局部变量的访问涉及当前堆栈帧在堆栈中的位置时,或当调试运行到任意一个被调函数中时,由于其调用函数中的局部变量不在当前堆栈帧中,如

果想访问它就必须要进行堆栈切换。这个窗口能显示系统堆栈中的各级堆栈帧，只要点击对应的函数名，就能访问到对应函数中的局部变量。

CPU 寄存器窗口显示当前 CPU 寄存器中的值，同时也可以对其进行修改。

CCS 工作区中，主要有以下四类窗口：

1. 源代码编辑窗口：可以打开，编辑 C++、C 或者汇编等源代码文件。
2. 反汇编窗口：通过仿真器从目标系统中读取二进制程序代码，将其反汇编为汇编指令后显示出来，同时还显示各种符号信息（如函数名）以及对应的地址和指令的二进制目标代码。
3. 存储器观察窗口：通过指定存储器的起始地址和数据格式，可以读取目标系统存储器中连续区域的数据并显示，同时也可以对其进行修改。
4. 图像显示窗口：根据某段连续存储器中的数据显示特定的图形，具体来说可以显示时/频域波形、眼图等形式的图形，其中时/频域波形的显示在调试信号处理算法的过程中是一个非常有效的工具，不管对于时域的采集信号还是最后计算得到的功率谱，通过这个窗口中的显示波形，都能确定其结果是否正确。

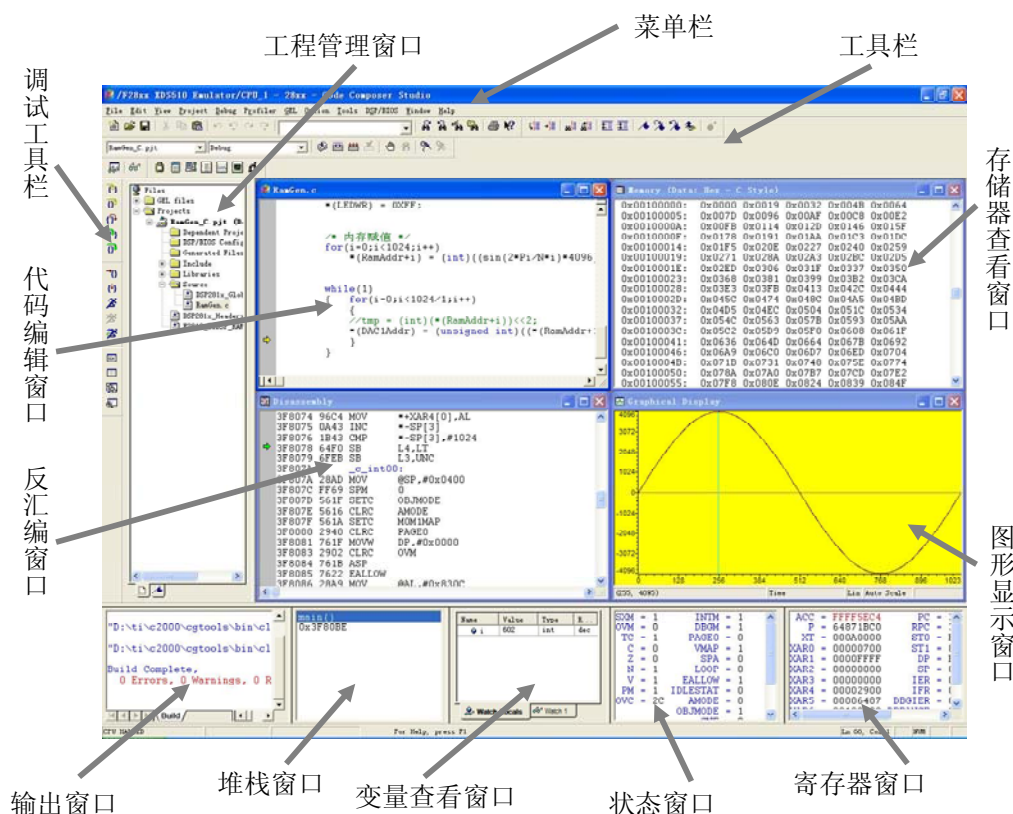


图 1.1 CCS 界面主要组成窗口

1.3 CCS 开发流程

本节介绍基于 TMS320F281x DSP 系统软件程序开发的总体步骤，并对其中比较重要和常用的工具进行介绍。图 1.2 是开发 DSP 程序的整体流程，它可以帮助程序开发人员更好地理解如何使用 CCS 集成开发环境的各功能部件。

由于 CCS 集成开发环境在代码生成工具的基础上，扩展了一系列调试和实时分析功能，因此它能够用于 DSP 系统软件开发的各个阶段，如图 1.2 所示。

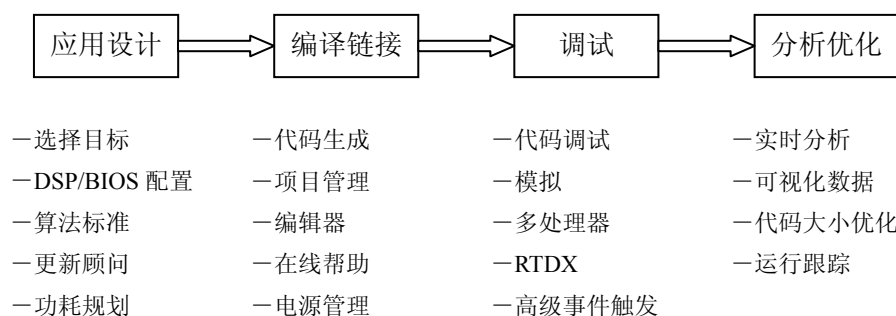


图 1.2 CCS 基本开发流程

一般来说,安装好 CCS 后,首先要正确地对 CCS 进行设置(安装仿真器驱动进行 emulate 或安装虚拟驱动进行 simulate),然后对源程序文件进行规划,建立汇编源代码文件、C 或者 C++源代码文件和定位控制文件(.cmd),把这些文件和必要的库文件(主要针对包含 C 或者 C++源代码文件的工程)都添加到新建的工程中。若采用 DSP/BIOS 工具来开发程序,还需要添加 DSP/BIOS 的 CDB 文件。接下来对此工程的各种汇编、编译和链接选项进行设置,再通过“build all”命令来完成整个工程的编译和链接。如果编译和链接时没有出现错误就能生成一个输出文件(.out),最后用 File 菜单下的 load program 命令将其加载到 DSP 系统的程序存储器(RAM)中,之后就可以开始对 DSP 软件程序进行在线调试,确保软件算法能稳定、可靠地实现目标系统各种功能,另外 CCS 还可以通过强大的分析工具,对代码的执行情况进行统计和分析,并将其作为进一步优化的依据,从而提高代码的执行效率。调试出稳定、可靠、高效的软件程序后,我们就能将程序烧写到 DSP 芯片内部的 Flash 里(如 TMS320F2812)使 DSP 系统能够脱离仿真器独立工作,从而完成 DSP 系统样机的研制。

接下来将以 CCS (C2000) Version2.20 为例,一步一步具体演示如何在 CCS 集成开发环境中通过工程来开发 DSP 软件。

1.3.1 目标代码生成

在完成 CCS 设置基础上,就可以连接目标系统、仿真器及 PC 机。当用 Emulate 仿真方式时,必须首先为仿真器和目标系统上电,然后打开 CCS2 (C2000) 程序,如果 CCS 设置正确,同时目标系统、仿真器和 PC 机间的连接无误,CCS 就能正常启动(启动过程不出现任何错误提示窗口,并在 CCS 窗口的标题栏显示“/F28xx XDS510 Emulator/ CPU_1 - 28xx - Code Composer Studio”)。接下来就能进行具体的软件开发,其一般步骤如下:

1. 选择 Project → New 命令,弹出如图 1.3 所示的创建工程对话框,填入工程名并为其选择工作目录;在 Project Type 中选择工程最终的输出形式(.out 或者.lib),在 Target 中选择目标芯片的类型(这里须选择 TMS320C28XX)。这一过程和目前大多数软件开发环境类似,唯一的区别是必须要指定正确的目标芯片,给定工程的名字后 CCS 会在指定的目录下自动产生一个和工程名相同的子目录。一般情况,我们将工程中所需要的文件都存放在该目录下,以便于项目管理。

2.选择 File → New → Source File,在打开的文本编辑器中输入源代码,然后选择 File → Save as,在出现的文件对话框中输入文件名,并为其选择路径(一般就在对应的工程目录下)。

3. 接下来需要将源代码文件包含到工程中:选择 Project → Add Files to Project,在出现的对话框中选择要包含的源代码文件,然后通过 Project → Compile File 命令就能对当前文本编辑窗口中的源文件进行编译,根据输出窗口的错误提示对源代码进行修改。除了源代码文件外,一个完整的工程还可能需要包括其他源文件(如库文件、链接器命令文件等)才

能生成最终的输出文件。例如，如果工程中存在 C 语言源代码文件，则还要添加 C 实时运行库，库文件可以通过以下两种方式添加：

- 利用 Project → Add files to project 命令添加；
- 选择 Project → Build Options, 在其打开对话框中 Linker 标签页下的 Library Search Path 和 Include Libraries 框中分别输入库文件的路径和名称。

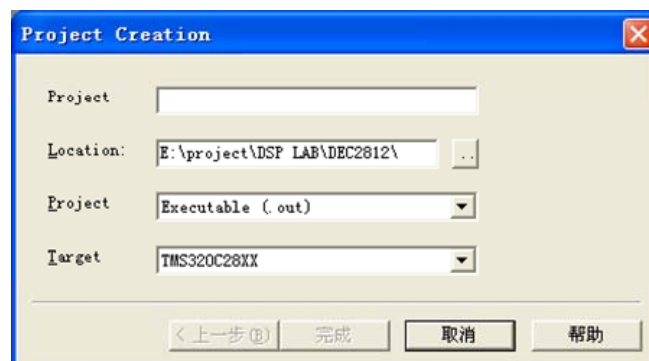


图 1.3 创建工程对话框

4. 添加 C 实时运行库的同时还要将 Build Options 中 Linker 标签页下的 Autoinit Model 设置成 Run-time Autoinitialization (-c)。

5. 接下来用同样的方式，新建链接器命令文件 (.cmd)，用 Project → Add files to project 命令添加链接器命令文件 (.cmd) 文件。


6. 最后选择 Project → build 对整个工程进行编译、汇编和链接。完成上述各步后，CCS 就能生成一个和工程相对应的.out 文件(在工程目录的 Debug 中)，通过 File → Load Program 命令将此.out 文件加载到目标系统中。


1.3.2 目标代码调试


下面以 CCS 安装目录下的教学程序 (...\\TI\\tutorial\\dsk2812\\volume1) 为例来简单介绍一下软件的调试过程：这个工程包含三个源代码文件：两个汇编源代码文件，一个 C 源代码文件（主程序所在文件）。其中 vectors.asm 用于存放中断向量表，这里只有一个复位中断向量，因为没有定义其他中断服务函数。Load.asm 中的代码是一个可以在 C 语言中调用的汇编函数。主程序的作用非常简单，将输入缓冲单元中的数据乘上一个增益后，放到输出缓冲单元中。因为 volume1.pjt 已经存在，可以直接用 build 命令对整个工程进行编译、汇编和链接，生成 volume1.out；也可以将 volume1.pjt 删除，用上面介绍的代码生成流程，重新建立这个工程文件，最后将工程 volume1 生成的 volume1.out 目标代码文件加载到目标系统中，接下来就能利用 CCS 提供的各种调试手段对其进行调试。


1. 调试过程中常用工具命令


在图 1.1 的调试工具栏中，一些常用的调试命令介绍如下：


 **Source - Single step:** 用于 C 源程序里的单步，遇到函数调用会跳到函数中继续单步执行。


 **Source - Step Over:** 用于 C 源程序里的单步，但是，遇到函数调用时不会跳转到函数中执行，而是把整条函数当成一个单步来执行。


 **Step Out:** 当程序执行到被调函数中时（无论是在汇编或者 C 里），如果使用该命令，则程序将执行到该函数返回处才停止。


 **Assembly - Single Step** : 用于汇编程序里的单步，遇到函数调用会跳到函数中继续单步执行。

 **Assembly - Step Over** : 用于汇编程序里的单步，但是，遇到函数调用时不会跳转到函数中执行，而是把整条函数当成一个单步来执行。


 **Run** : 运行程序，直到断点处停止运行，直到再次触发运行命令才恢复程序的运行。

 **Halt** : 停止程序的运行。

 **Animate** : 运行程序，遇到断点后，CCS 先暂停程序的运行，对打开的变量、寄存器、存储器、图像显示等窗口中的数据进行刷新。然后，自动恢复程序运行，这样在图像显示窗口中就可以观察到动画效果。


 **Toggle breakpoint**: 在源程序编辑窗口中提示符所在的行设置断点，如果该行已经存在断点，则取消该断点。


 **Remove all breakpoints**: 取消工程中所有已设置的断点。


 **Toggle Probe Point**: 在源程序编辑窗口中提示符所在的行设置探针点，如果该行已经存在探针点，则取消该探针点。


 **Remove all Probe Points**: 取消工程中所有已设置的探针点。

在调试过程中，可以查看 DSP 相关寄存器以及存储器内的数值，如图 1.1 所示，一些常用的查看指令介绍如下：


 **Registers window**: 用于打开 DSP 内部寄存器窗口，查看各项数值，并可以修改。

 **Memory Window**: 用于打开存储器窗口，包括 DSP 内部以及外部存储器中地址单元的数值，若该单元可写，则支持数据的修改。

 **View Stack**: 用于各个堆栈帧之间的快速切换，查看各帧中局部变量数值。

 **View Disassembly**: 用于查看二进制程序代码所对应的汇编指令，同时还显示各种符号信息及其对应的二进制地址。

 **Watch window**: 用于打开变量窗口，观察程序中变量的地址和数值。

 **Quick Watch**: 快速打开变量窗口。

以上这些快捷方式，在菜单中都可以找到相应的命令。

当把程序加载到 DSP 程序空间以后，首先，可以用 **Debug → Go main** 命令将程序运行到 main 源代码处（实际系统在执行源代码里的 main 函数前，要先调用一个 C 初始化函数即 `_c_int00`，从而完成全局变量的初始化，设置堆栈指针等工作，在这个初始化函数的最后才调用 main 函数，这段初始化代码保存在运行支持库函数中，工程在链接阶段会把它加

到.out 文件，其实现的源代码可以在库文件所在路径下的 rts.src 文件中找到。当把.out 文件加载到目标系统中或者执行 Restart 命令后，反汇编窗口中会显示当前的程序指针处于 _c_int00 函数的入口处，而不是 main 函数的入口处），如图 1.4 所示，其中黄色的箭头（图中上方窗口中）代表当前 PC（程序指针）在对应源代码文件中的位置，而反汇编窗口中绿色箭头（图中下方窗口中）处则是当前 PC 所指向的存储器地址、机器码数据以及其对应的汇编指令，也就是 DSP 下一步要执行的机器指令。

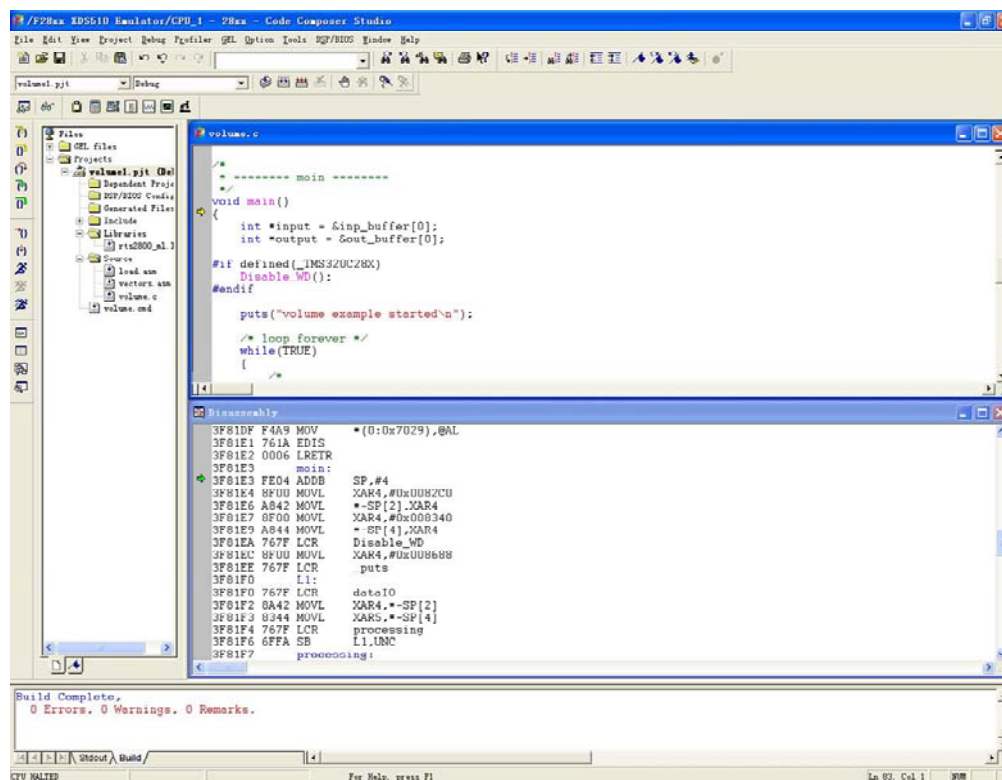


图 1.4 源程序窗口和反汇编窗口

2. 断点的使用

可以通过调试工具栏中的 Toggle Breakpoint 图标或者右键菜单中的 Toggle breakpoint 命令来设置断点，同时用 Debug 菜单下的 BreakPoints 命令能将该断点配置成一般断点、条件断点（特定的表达式为真时才停止程序的运行）或者硬件断点。设置好断点后运行程序（Run）以后，当程序执行到断点处时会停止运行直到下一个运行命令被触发，在程序停止运行期间，CCS 还会对各种调试数据进行更新。在源文件任何有效语句处都可以设置断点，如果断点设置出错，例如将断点设到了无效行，CCS 会给出错误提示，并自动将断点转移到下一有效行处。

3. 观察窗口的使用

在 volume.c 中，选中任意一个变量，在右键菜单中选择 Quickwatch 或者 Add to watch Window，CCS 将打开 Quickwatch 窗口的 Watch1 子窗口并显示选中的变量。在观察窗口中变量名后面的 Value 一栏中可以直接修改被观察变量的取值，在 Radix 一栏中还能设定数据的显示格式（十六进制、八进制、十进制、二进制、浮点数、无符号整型等等）。另外 Quickwatch 窗口的 Watch Locals 子窗口中会自动显示当前函数作用域中所有局部变量的值。

把全局变量 str 加到观察窗口中，执行程序后，点击变量左边的“+”，观察窗口会将结构变量展开，同时显示结构变量中的每个成员的值，该显示方式同样适用于数组的显示。

把 main 函数中的局部变量 input 添加到观察窗口中，执行程序进入 dataIO 函数（通过断点或者单步都可以），此时 input 变量超出了起作用范围（main 函数），所以变量 Value 一栏显示 “identifier not found: input”，但是可以利用 Stack Call 窗口来察看不同作用域中的变量，如图 1.5 所示，执行 View → Call Stack 打开堆栈切换窗口，双击窗口中的 main()项，即可在观察窗口中查看 input 变量的当前值了。

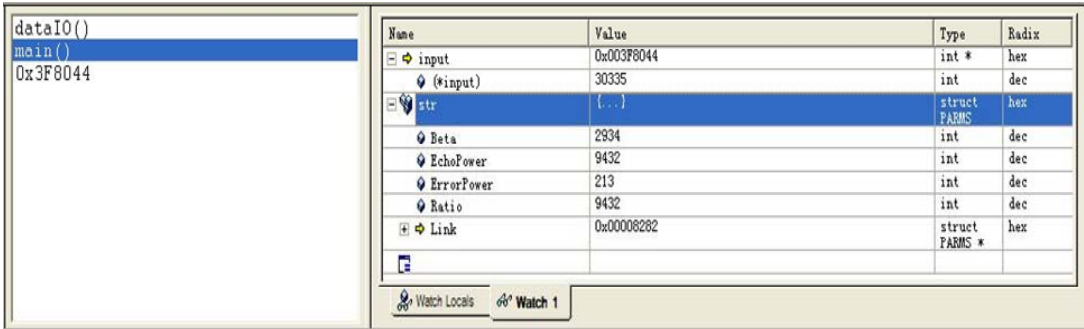


图 1.5 堆栈切换窗口及变量观察窗口

4. 使用探针实现文件输入/输出

探针点是实际上是一种特殊的断点。通过这个功能可以实现：当程序运行到探针点时，CCS 中断目标系统中 DSP 的运行，从与该探针点关联的数据文件中（PC 机中的.dat 文件）读出数据并写到目标系统的存储器中，或者从目标系统的存储器中读出数据并写到数据文件中。完成数据的交换后，CCS 自动恢复目标系统 DSP 的运行。探针点特别适合用于算法的仿真，在目标系统的输入/输出硬件电路完成前，可以使用该工具来模拟数据的输入/输出。另外，它还可以在软件仿真时为虚拟目标系统提供数据的输入/输出功能。它是用已知的数据流测试算法正确性的必备工具。

下面将在工程 volume1 的基础上，利用探针工具来模拟数据的输入。C 源代码程序中有一个 dataIO()函数，由于没有硬件支持，该函数仅仅是个空函数，所以我们就用探针点来模拟其数据输入的功能，具体步骤如下：

- (1) 假设.out 文件已经加载，在光标移到 main()函数中 dataIO()函数的调用语句行上，用右键菜单或者工具栏上的 Toggle Probe Point 命令设置探针点，设置好后，该行语句前面会出现一个蓝色的菱形标识。
- (2) 为探针点建立关联的数据文件：在 File 菜单中选择 File I/O，出现如图 1.6 所示的数据文件 I/O 配置对话框，用 Add File 按钮选择要关联的数据文件，这里可以选择 volume1 目录下的 sine.dat 文件；在 Address 文本框中输入相应的目标系统存储器的起始地址，如果此时已经加载了.out 文件，那么也可以直接输入符号来表示该地址，如_inp_buffer（C 源文件中变量 inp_buffer 在汇编文件中的引用形式）或者 inp_buffer；在 Length 文本框中输入数据的长度（如 100），设置好后运行程序，当程序运行到探针点处时，会从对应的数据文件中读取 100 个数据到 inp_buffer 缓冲区中。

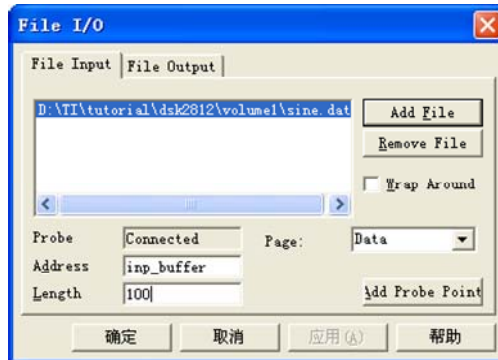


图 1.6 数据文件 I/O 设置对话框

(3) 在图 1.6 中单击 Add Probe Print 按钮，弹出如图 1.7 所示的对话框，在对话框中可以实现探针点和特定 I/O 文件的关联。这个过程和断点的设置一样，先在已有探针点窗口中选定需要设置的探针点，接着选择探针点的类型（一般探针点、条件探针点还是硬件探针点）。如果是条件探针，还要输入代表条件的表达式；然后在 Connect 中选择刚才在 File I/O 中设置好的输入文件并单击 RePlace，此后可以从探针点窗口观察到探针点属性的变化，确定后就完成了 I/O 文件的关联。

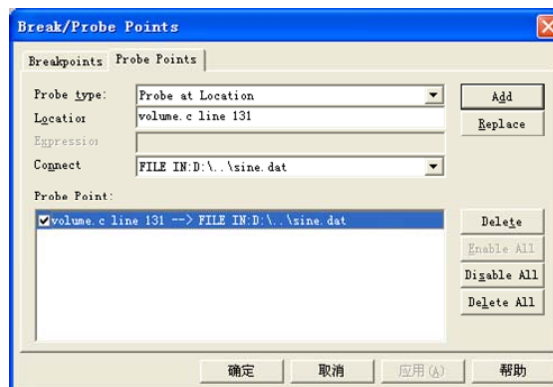


图 1.7 探针设置对话框

(4) 完成上述设置工作后，即可运行程序，当程序运行至 dataIO 处时，CCS 就会自动从 sine.dat 文件中读取 100 个数据并写入到 inp_buffer 数组中，从而实现输入功能的模拟。如果在数据文件 I/O 配置窗口中勾选了 Wrap Around 功能，则 CCS 会自动重复使用该文件中的数据。

5. 图形功能简介

上文中，利用探针工具和外部数据文件模拟实现了数据的输入，下面在此基础上将通过波形显示的方式来观察输入/输出的结果。

(1) 在 View 菜单项中选 Graph，然后选 Timer/Frequency，进入图形属性设置对话框，如图 1.8 所示。

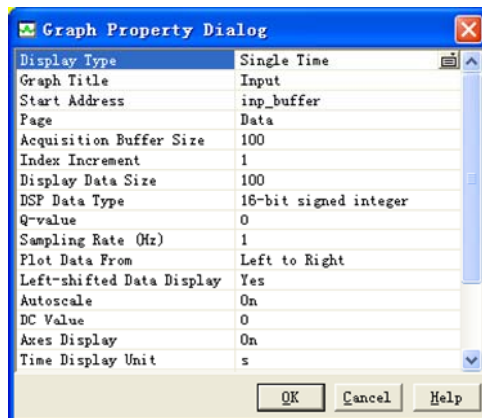


图 1.8 图形属性设置对话框

(2) 在 Graph Title 中输入图形窗口的名称: input, 在 Start Address 中输入数据 (用于绘制图形) 在存储器中的起始地址。如果此时已经加载.out 文件, 则可以直接输入变量符号 inp_buffer; 在 Acquisition Buffer Size 和 Display Data Size 中输入要显示的数据长度 100, 这里前者表示是数据缓冲区的大小, 后者表示用于绘制图形的实际数据大小; 最后在 DSP Data Type 中选择相应的数据类型, 此处我们选 16-bit signed integer; 其他属性都取默认值, 点击确认后自动打开图形显示窗口。重复上述过程, 在 Graph Title 中输入 output, 将起始地址改为 out_buffer, 再打开一个图形显示窗口用来显示输出数据。

(3) 在主函数中 dataIO() 函数调用语句处设置一个断点。

(4) 连续单击 Run 运行程序, 程序每次都会在断点处停止, 同时在图形显示窗口中显示输入信号的波形, 这个信号就是从探针点输入的正弦波形数据, 所以将看到一个正弦波。

(5) 使用 Animate 命令运行程序, 会看到连续变化的输入/输出波形, 如图 1.9。另外, 还可以通过图形属性设置对话框中的 Display Type 中选择显示的波形类型, 如 FFT 幅值图, FFT 相位图等; 通过 Autoscale 选项来开启或者关闭显示比例的自动调整功能, 如果该功能被关闭, 还可以设置显示波形的最大、最小值。具体其他功能不在此细述, 可参考 CCS 是使用手册。

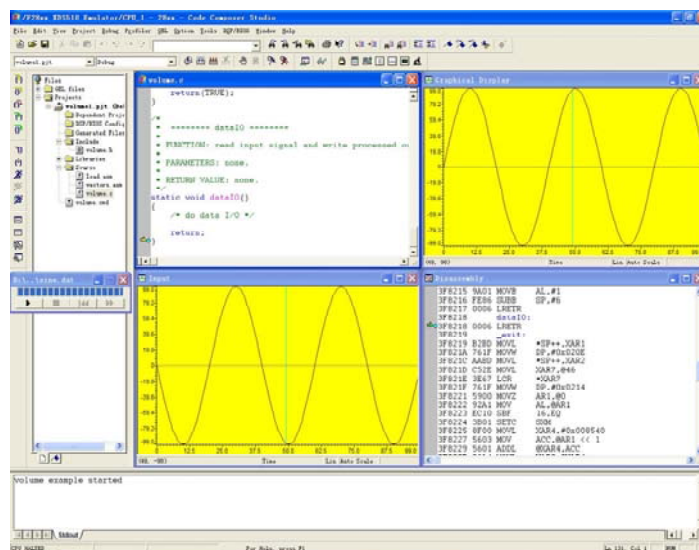


图 1.9 图形显示功能

6. 分析工具 (Profiler) 的使用

在 CCS 中，还集成了代码分析工具 **Profiler**，可以用来计算某段代码共执行了多少个机器周期。这样不但能为程序代码的进一步优化提供依据，也可以为程序的实时性提供一个较精确的度量。下面还是以工程 **volume1** 为例简单介绍一下这个工具的使用。

- (1) 加载生成的 **volume1.out** 文件，将程序执行到 **main()** 处。
- (2) 选择 **Profiler**→**start New session** 命令，新建一个分析任务，并为其指定任务名。
- (3) 打开主程序源文件，如图 1.10 所示，在 **Processing()** 函数定义的首行处选择右键菜单中的 **Profile Function**，将该函数添加到分析窗口中。

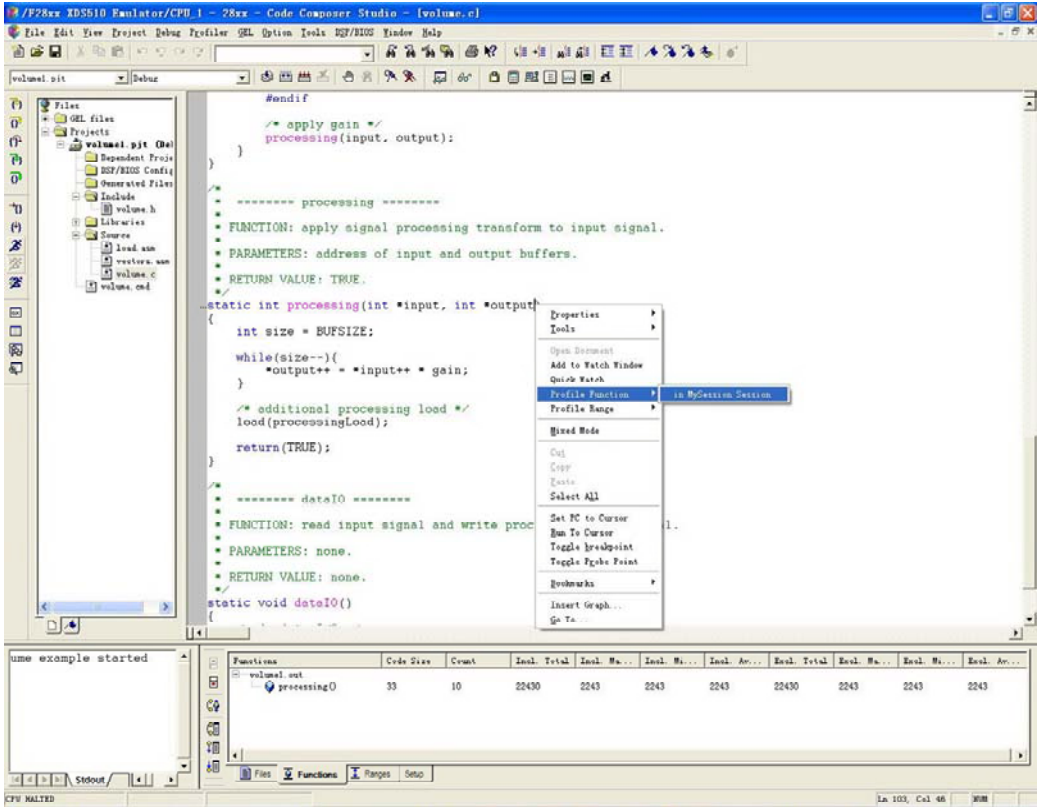


图 1.10 代码分析工具的使用

(4) 运行程序，每次程序停止时（遇到断点、单步执行或者使用 **Halt** 命令）或者运行到探针点时，分析窗口的统计数据就会被刷新。统计数据主要包含三个部分，一是被分析代码的长度（**Code Size**）以及该段代码已经被执行的次数，比如这里 **Processing()** 函数的代码长度就是 33，其执行次数随着运行时间的增加而递增；二是整个被分析的代码区间（包含其中子函数调用的执行情况）中以机器周期数为单位的统计数据，如累计执行时间，最大、最小及平均执行时间等；三是整个被分析的代码区间（不包含其子函数调用的开销）中以机器周期数为单位的统计数据，如累计执行时间，最大、最小及平均执行时间等。

Profiler 工具其他更深入的使用方法请参考 CCS 的使用手册。

第二章 C 语言开发文件说明

DSP 的软件可以使用汇编 (asm)、C、C++等语言进行开发。下面将介绍以 C 语言开发 DSP 过程中的文件说明。

2.1 实验例程中的相关文件

C 语言开发过程中需要一些包含 C281x 寄存器声明和定义的 C 头文件、源文件以及库文件。这些头文件（主要是片内各外设寄存器对应的结构体及共用体类型的声明）、DSP281xGlobalVariableDefs.c（寄存器变量的定义）和 DSP281x_Headers_nonBIOS.cmd（连接器命令文件）主要用于片内系统及外设寄存器变量的声明、定义和定位，同时一些通用的系统或者外设初始化源代码文件（比如 DSP281x_DefaultIsr.c、DSP281x_PieCtrl.c、DSP281x_PieVect.c 等）也会在一些程序用到。

一般来说，工程中除了主程序源文件外，还包括如下文件：

- 前面提到的用于声明寄存器变量结构的头文件（每部分外设或者系统功能寄存器组都对应一个头文件），使用时只要在程序中包含 DSP281x_Device.h 就能包含其他所有的系统及外设寄存器头文件。注意：所有的头文件都不是手工添加的工程中的，只要在源代码文件中加入头文件包含命令，编译连接时会自动添加这些头文件到工程中。

- DSP281x_Headers_nonBIOS.cmd：由于同一片内外设模块中的寄存器地址基本上都是连续的，这样这些寄存器变量就能以外设模块为单位配置到一系列输出段，该文件的作用就是根据各寄存器的实际地址将这些段映射到实际的存储器空间中。

- DSP281x_GlobalVariableDefs.c：将所有存储器映射的系统及外设寄存器定义成全局变量（这些变量的数据结构已经在对应的系统及外设头文件中声明过），并将这些变量分配到.cmd 文件中对应的输出段中。

- F2812_EzDSP_RAM_Ink.cmd：针对在 RAM 中运行的程序而编写的连接器命令文件，可以自己根据存储器的扩展情况重新编写一个。

- rts2800_ml.lib：C 语言实时运行库文件。

2.2 CMD 文件

下面以 F2812_EzDSP_RAM_Ink.cmd 为例，简单介绍一下 CMD 文件的组成、基本伪指令的含义和用法：

CMD 文件的内容主要分为以下两部分：

1. MEMORY

以伪指令 MEMORY 开始的部分是用来定义目标板上存储器资源的分布，即有哪些存储器可以用，该文件中的这部分内容如下所示：

```
MEMORY
{
PAGE 0 :
    /* 片内的 H0 被分割成了 PAGE 0 和 PAGE 1 两部分          */
    /* BEGIN 区域在“H0 引导”模式下使用                      */
    /* 当且仅当从 XINTF 区域 7 中引导时，                      */
    /*                      复位向量加载到 RESET 区域中        */
    /* 否则，复位向量从 Boot ROM 中取得                        */

    RAMM0      : origin = 0x000000, length = 0x000400
```

```

BEGIN      : origin = 0x3F8000, length = 0x000002
PRAMH0     : origin = 0x3F8002, length = 0x000FFE
RESET      : origin = 0x3FFFC0, length = 0x000002

PAGE 1 :
/* 片内的 H0 被分割成了 PAGE 0 和 PAGE 1 两部分      */

RAMM1      : origin = 0x000400, length = 0x000400
DRAMH0     : origin = 0x3f9000, length = 0x001000
}

```

其中，PAGE 0 代表的是程序存储区，PAGE 1 指数据存储器，RAMM0 和 BEGIN 等都是程序存储器中各个自定义子区域的名称，数据存储器同理。每个子区域内的空间是连续的，后面的参数分别指示其起始地址和长度。区域间可以是离散或者连续（有时为了编程思路的清晰化，对实现不同功能的连续存储区域分别独立取名）。如果某一段物理存储器没有在 MEMORY 伪指令后进行配置，则链接器不会将任何程序或者变量定位到那里。

2. SECTIONS

而以伪指令 SECTIONS 开始的部分则用来控制程序文件中代码和数据输出段在存储器区域（必须是在 MEMORY 部分定义好的子区域）中的定位，该部分内容如下：

```

SECTIONS
{
    .text          :> PRAMH0,          PAGE = 0
    .cinit          :> PRAMH0,          PAGE = 0
    .pinit          :> PRAMH0,          PAGE = 0
    .switch         :> RAMM0,           PAGE = 0
    .reset          :> RESET,           PAGE = 0, TYPE = DSECT /* not used, */

    .stack          :> RAMM1,           PAGE = 1
    .ebss           :> DRAMH0,          PAGE = 1
    .econst         :> DRAMH0,          PAGE = 1
    .esysmem        :> DRAMH0,          PAGE = 1
}

```

这里.text 代表程序中的可执行代码段，后面的指令参数表示此段代码程序将被装载到程序存储器的 PRAMH0 区域中，而.cinit 段的存储器区域定位将紧接着.text 段后面。同理，以.stack 和.ebss 为首的指令参数表示的是堆栈和未初始化变量在数据存储器 DRAMH0 区域中的定位。

MEMORY 部分描述的是用户如何给目标存储器进行分类、分区，其描述和定义的对象必须是实际存在的物理存储器；而 SECTIONS 部分就是规定目标程序代码、变量将被装载或是定位到存储器的哪个区域，其控制的对象是源代码程序的各个输出段，其定位的范围只能是 MEMORY 部分中定义好的存储器区域。

注意：从 CCS2.20 开始才允许向一个工程里添加多个 CMD 文件。

这里仅仅给出了一个 CMD 文件最简单的应用，并介绍了其中最基本和最常用伪指令的用法，如果读者需要进一步了解 CMD 文件中的其他伪指令及应用，请参考 28x 的汇编语言工具使用手册中有关链接器的章节。

2.3 寄存器变量的声明和定义文件

下面以通用 I/O 口数据寄存器变量为例，通过其寄存器变量的声明(.h)和定义文件(.c)简单介绍一下寄存器变量型数据结构的声明、寄存器变量对象的定义、输出段的映射以及寄存器变量成员的访问方法。在 DSP281x_Gpio.h 中有如下声明：

```
struct GPAMUX_BITS {           // bits   description
    Uint16 PWM1_GPIOA0:1;      // 0
    Uint16 PWM2_GPIOA1:1;      // 1
    Uint16 PWM3_GPIOA2:1;      // 2
    Uint16 PWM4_GPIOA3:1;      // 3
    Uint16 PWM5_GPIOA4:1;      // 4
    Uint16 PWM6_GPIOA5:1;      // 5
    Uint16 T1PWM_GPIOA6:1;      // 6
    Uint16 T2PWM_GPIOA7:1;      // 7
    Uint16 CAP1Q1_GPIOA8:1;     // 8
    Uint16 CAP2Q2_GPIOA9:1;     // 9
    Uint16 CAP3QI1_GPIOA10:1;   // 10
    Uint16 TDIRA_GPIOA11:1;     // 11
    Uint16 TCLKINA_GPIOA12:1;   // 12
    Uint16 C1TRIP_GPIOA13:1;    // 13
    Uint16 C2TRIP_GPIOA14:1;    // 14
    Uint16 C3TRIP_GPIOA15:1;    // 15
};
```

上面的代码声明了一个叫 GPADAT_BITS 的结构体，这个 16 位结构体中包含 16 个二进制位成员，这些成员的名称从低到高各位分别对应 GPIOA0~GPIOA15。

```
union GPAMUX_REG {
    Uint16          all;
    struct GPAMUX_BITS  bit;
};
```

上面的代码声明了一个叫 GPADAT_REG 的共用体，这个共用体既可以当成一个 16 位无符号整型数据来用，也可以当成 GPADAT_BITS 结构体形式的数据。如果需要当成前者来引用，就要使用 all 这个成员名，如果是后者，则要用成员名 bit。

```
struct GPIO_DATA_REGS {
    union  GPADAT_REG      GPADAT;
    union  GPASET_REG      GPASET;
    union  GPACLEAR_REG    GPACLEAR;
    union  GPATOGGLE_REG    GPATOGGLE;
    union  GPBDAT_REG      GPBDAT;
    union  GPBSET_REG      GPBSET;
    union  GPBCLEAR_REG    GPBCLEAR;
    union  GPBTOGGLE_REG    GPBTOGGLE;
    Uint16                    rsvd1[4];
    union  GPDDAT_REG      GPDDAT;
    union  GPDSET_REG      GPDSET;
    union  GPDCLR_REG      GPDCLR;
    union  GPDTOGGLE_REG    GPDTOGGLE;
```

```

union  GPEDAT_REG      GPEDAT;
union  GPESET_REG      GPESET;
union  GPECLEAR_REG    GPECLEAR;
union  GPETOGGLE_REG   GPETOGGLE;
union  GPFDAT_REG      GPFDAT;
union  GPFSET_REG      GPFSET;
union  GPFCLEAR_REG    GPFCLEAR;
union  GPFTOGGLE_REG   GPFTOGGLE;
union  GPGDAT_REG      GPGDAT;
union  GPGSET_REG      GPGSET;
union  GPGCLEAR_REG    GPGCLEAR;
union  GPGTOGGLE_REG   GPGTOGGLE;
Uint16                                rsvd2[4];
};

```

上面的代码声明了一个叫 GPIO_DATA_REGS 的结构体，它是根据通用 I/O 口各数据寄存器的地址分布（总体上讲，他们在存储器空间中是连续分布的），为这块存储区域声明一个结构体。在这块存储区域中，如果遇到保留区域，就根据其大小将这片保留区域声明成 16 位无符号整型数组，如上面 GPIO_DATA_REGS 结构体中的 rsvd1 数组成员就表示 GPBTOGGLE 寄存器后面 4 个字大小的存储区域是被保留的区域。

```

#ifdef __cplusplus
#pragma DATA_SECTION("GpioDataRegsFile")
#else
#pragma DATA_SECTION(GpioDataRegs,"GpioDataRegsFile");
#endif
volatile struct GPIO_DATA_REGS GpioDataRegs;

```

编译指示符 DATA_SECTION 的作用是通知编译器将某变量分配到指定的输出段里，上面这段代码的功能就是定义 GPIO_DATA_REGS 结构体型的变量 GpioDataRegs（寄存器组变量），并将 GpioDataRegs 变量定位到 GpioDataRegsFile 输出数据段中。

注意：此处变量定义时使用的 volatile 关键词非常重要，它能够告诉编译器此变量的内容可能会被硬件修改，因此，编译器就不会对其进行优化。如果不使用这个关键词，那么编译器有可能将这个变量优化到 CPU 寄存器中，从而导致不可预见的错误。

在 DSP281x_Headers_nonBIOS.cmd 中有如下伪指令代码：

```

MEMORY
{
PAGE 0:      /* Program Memory */

PAGE 1:      /* Data Memory */
    // 省略
    GPIODAT    : origin = 0x0070E0, length = 0x000020    /* GPIO 数据寄存器 */
    // 省略
}

SECTIONS
{

```



```
// 省略
GpioDataRegsFile :> GPIODAT    PAGE = 1
// 省略
}
```

通过上面 cmd 文件中的伪指令，能将数据输出段 GpioDataRegsFile 指定到实际的数据存储器区域 GPIODAT 里（该区域起始地址为 0x0070E0，长度是 0x000020）。这样，当在主程序中需要访问 GPADAT 寄存器时，就能通过下面的形式直接实现：

```
GpioDataRegs.GPADAT.bit.GPIOA4 = ....
GpioDataRegs.GPADAT.all = ....
```

前者表示的是单独访问 GPADAT 寄存器的第 4 位 GPIOA4，同理也可以单独访问 16 位数据中的任何一位；后者表示将 GPADAT 整体当成一个 16 位无符号整型数据来访问。

一般来说，在声明寄存器变量结构时，都会根据功能将各个位域描述成整体的结构成员（一些寄存器中可能需要几个位组合起来使用，如果是这样就将这几个位作为一个局部整体声明成寄存器的一个成员），因此就能以结构体成员的形式来访问寄存器的某一位或者某几位：寄存器组.寄存器.bit.功能位。其中寄存器组是外设寄存器组结构体类型（包含多个寄存器，可能还有保留区域）的变量名，寄存器是寄存器组数据结构声明中的成员名（结构体），bit 是寄存器以位域结构体形式进行访问时使用的成员名（共用体），功能位是该位域结构体中的具体成员名（结构体）；同理，用寄存器组.寄存器.all 就能以 16 位无符号数整体的形式访问寄存器，这里的 all 是寄存器以 16 位无符号数整体形式被引用时所使用的成员名（共用体）。

相比使用的宏定义方式，这种方式通过结构体、共用体以及全局变量实现了对寄存器位域的独立访问，为寄存器提供了更加灵活和高效的访问手段，也大大提高了代码的可读性、可靠性和可维护性。

第三章 C2000 DSP 教学实验箱介绍

在实验前，必须仔细阅读，熟悉实验平台硬件。

3.1 概述

在 C2000 DSP 实验箱系统中主要集成了 DSP、SRAM、ADC、DAC、UART、SPI、CAN、USB、LCD、LED 以及键盘电路等，其主要结构框图如图 3.1 所示。

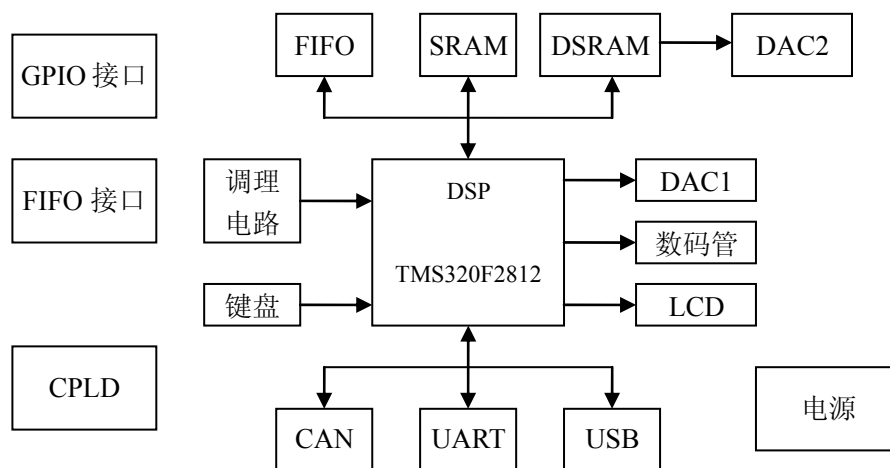


图 3.1 C2000 实验箱原理框图

实验箱主要芯片说明：

- DSP: TMS320F2812 (32bit, 150MHz)

片上存储器：

FLASH	128K×16位
SRAM	18K×16位
BOOTROM	4K×16位
OPTROM	1K×16位

其中 FLASHOPTROM 和 8K×16 位的 SRAM 受口令保护，用以保护程序。

片上外设：

PWM	12路
QEP	6通道
ADC	2×8通道，12位，80ns转换时间，0~3V输入量程
SCI	2通道
McBSP	1通道
CAN	1通道
SPI	1通道

- 外扩 SRAM: CY7C1041 (256K×16bit, 15ns)
- 外扩双端口 SRAM: IDT70V24 (4K×16bit, 15ns)
- 外扩 FIFO: IDT7206 (16K×8bit)
- 外扩 USB 芯片: CY7C68001
- 外扩两路 DAC: AD768 (16bit, 30ns 转换速率)
- 外扩 LCD 显示芯片: T6963C

- 外扩串行 EEPROM 芯片：AT24C04 (4Kb)
- 实时时钟芯片：X1226 (4Kb EEPROM)

实验箱主要接口说明：

- 模拟输入 INPUT1, INPUT2: $\pm 1V$
- 模拟输出 OUT1, OUT2, OUT3: $\pm 1V$
- USB 接口 (JP3): 符合 USB2.0 协议, 最高传输速率 480Mbps
- CAN 接口 (JP5): 符合 CAN2.0 协议, 最高传输速率 1Mbps
- RS232/485 接口 (JP4): 标准 232/485 接口
- 键盘接口 (JP12): 19 键非标准自定义键盘
- LCD 接口 (JP11): 240 \times 128 点阵 LCD
- GPIO 接口 (JP10): 10 路输出, 8 路输入
- FIFO 接口 (JP9): 外扩 FIFO 接口
- DSP_JTAG 接口 (JP1): DSP 仿真接口
- CPLD_JTAG 接口 (JP2): CPLD 程序下载口

实物图及接口说明如图 3.2 所示：

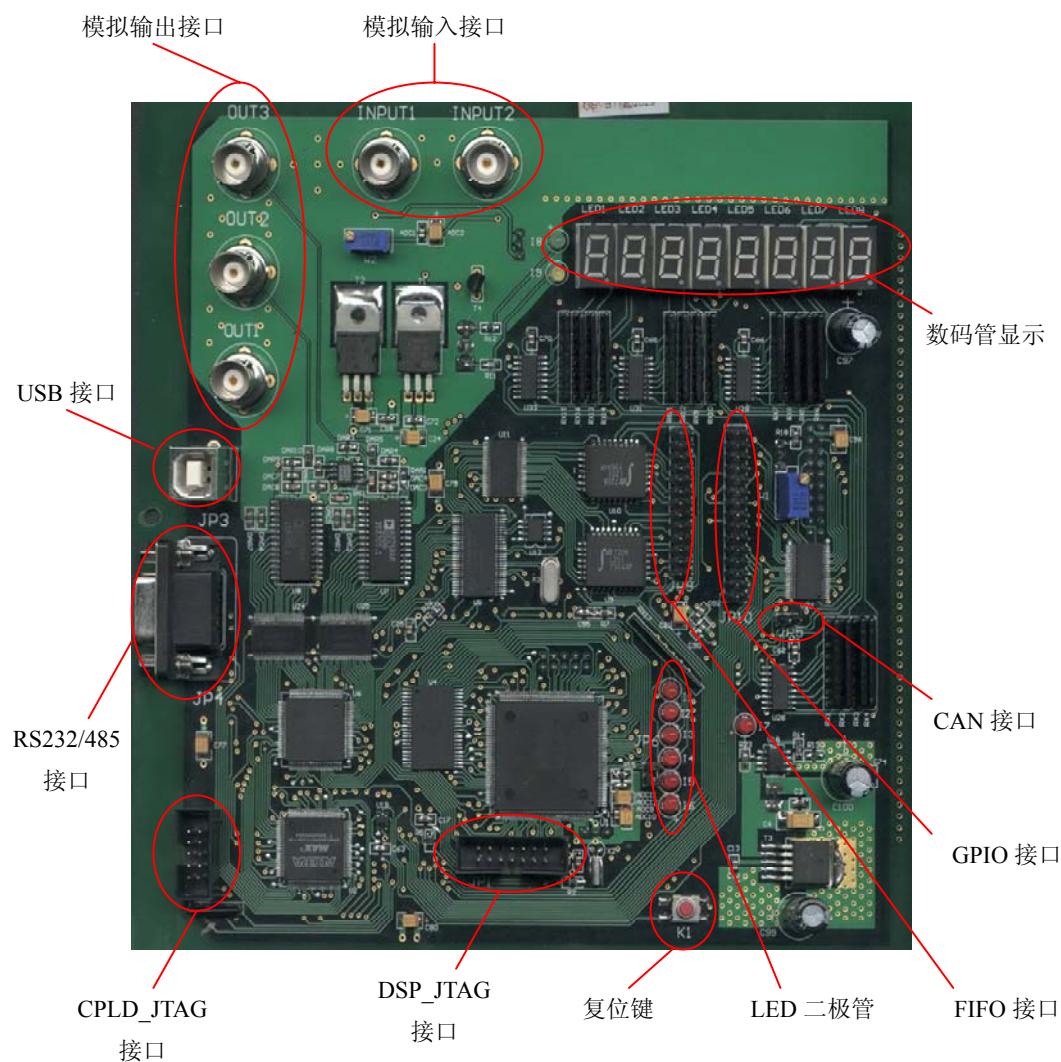


图 3.2 C2000 实验箱接口

3.2 实验箱分系统介绍

3.2.1 时钟电路

实验箱中的 TMS320F2812 用 30MHz 外部晶体给 DSP 提供时钟,并使能 F2812 片上 PLL 电路。PLL 倍频系数由 PLL 控制寄存器 PLLCR 的低 4 位控制,可由软件动态的修改。外部复位信号 (RS) 可将此 4 位清零 (CCS 中的复位命令将不能对这 4 位清零)。F2812 的 CPU 最高可工作在 150MHz 的主频下,也即是对 30MHz 输入频率进行 5 倍频。

3.2.2 存储空间的配置

TMS320F2812 为哈佛结构的 DSP,在逻辑上有 $4\text{M} \times 16$ 位的程序空间和 $4\text{M} \times 16$ 位数据空间,但在物理上已将程序空间和数据空间统一成一个 $4\text{M} \times 16$ 位的空间。其地址映射表如图 3.3 所示。

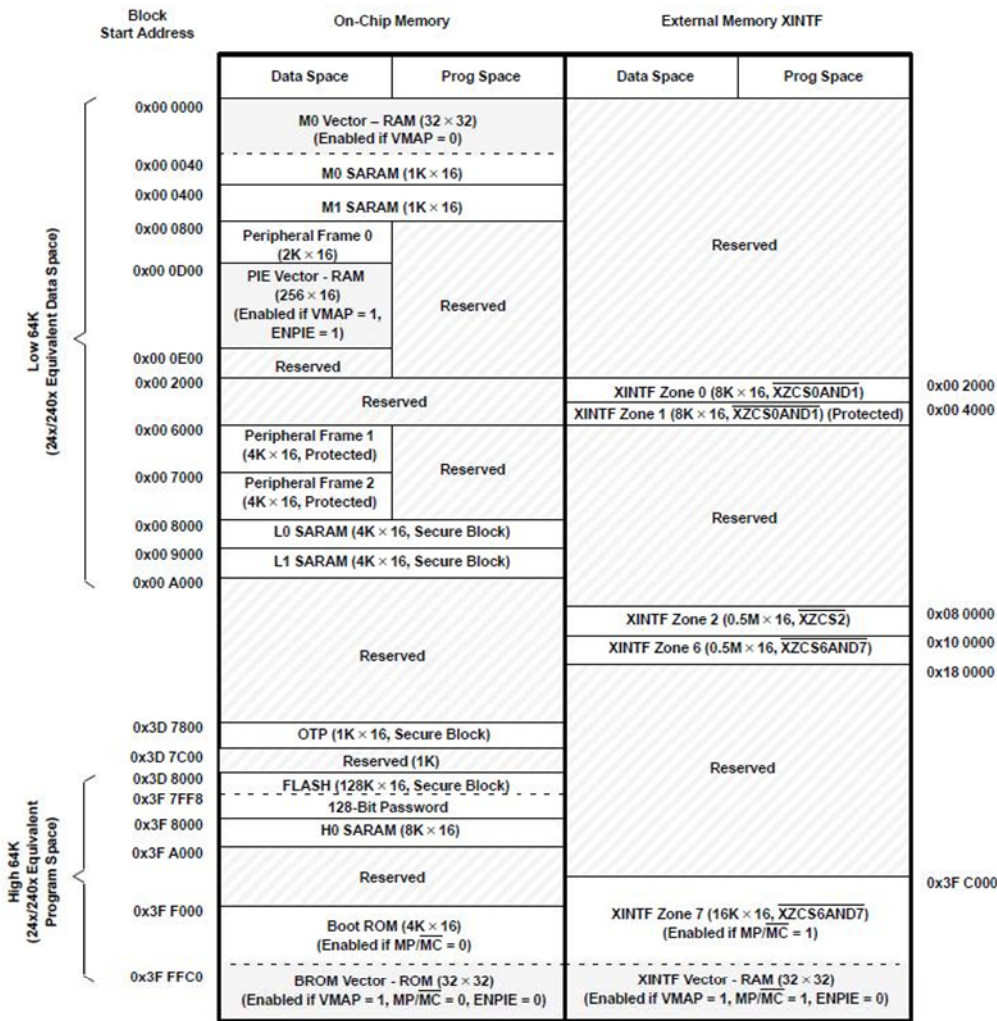


图 3.3 TMS320F2812 地址映射表

TMS320F2812 片上有 $128\text{K} \times 16$ 位的 FLASH, $18\text{K} \times 16$ 位的 SRAM, $4\text{K} \times 16$ 位的 BOOT ROM, $1\text{K} \times 16$ 位的 OTP ROM。

实验箱上还外扩了 $256\text{K} \times 16$ 位 SRAM, $4\text{K} \times 16$ 位双端口 SRAM 和 $16\text{K} \times 8$ 位 FIFO 存储器,另外还外扩了若干个控制状态接口。这些存储体在存储空间中的映射如下表 3.1 所示。

表 3.1 C2000 实验箱 TMS320F2812 DSP 存储空间配置表

地址范围	数据空间	程序空间	等待时间	备注
0x00,0000~ 0x00,03FF	1K×16 位 片上 M0 SARAM	1K×16 位 片上 M0 SRAM	0 等待	
0x00,0400~ 0x0007FF	1K×16 位 片上 M1 SARAM	1K×16 位 片上 M1 SRAM	0 等待	
0x00,0800~ 0x00,0FFF	2K×16 位 片上外设寄存器块 0	保留	0 等待	
0x00,1000~ 0x00,1FFF	4K×16 位 保留	保留		
0x00,2000~ 0x00,5FFF	16K×16 位 外扩控制/状态寄存器	保留	1 个等待	占 ZONE0 和 ZONE1
0x00,6000~ 0x00,6FFF	4K×16 位 片上外设寄存器块 1	保留	0 等待	
0x00,7000~ 0x00,7FFF	4K×16 位 片上外设寄存器块 2	保留	0 等待	
0x00,8000~ 0x00,8FFF	4K×16 位 片上 L0 SARAM	4K×16 位 片上 L0 SARAM	0 等待	受 CSM 保护
0x00,9000~ 0x00,9FFF	4K×16 位 片上 L1 SARAM	4K×16 位 片上 L1 SARAM	0 等待	受 CSM 保护
0x00,A000~ 0x07,FFFF	472K×16 位 保留	保留		
0x08,0000~ 0x08,0FFF	4K×16 位 外扩双口 SRAM	4K×16 位 外扩双口 SRAM	2 个等待	占 ZONE2
0x10,0000~ 0x13,FFFF	256K×16 位 外扩 SRAM	256K×16 位 外扩 SRAM	2 个等待	占 ZONE6
0x14,0000~ 0x14,FFFF	16K×8 位 外扩 FIFO	16K×8 位 外扩 FIFO	4 个等待	占 ZONE6
0x18,0000~ 0x3D,77FF	2398K×16 位 保留	2398K×16 位 保留		
0x3D,7800~ 0x3D,7BFF	1K×16 位 片上 OTP ROM	1K×16 位 片上 OTP ROM	1 个等待	受 CSM 保护
0x3D,7C00~ 0x3D,7FFF	1K×16 位 保留	1K×16 位 保留		
0x3D,8000~ 0x3F,7FFF	128K×16 位 片上 FLASH	128K×16 位 片上 FLASH	0 等待	受 CSM 保护
0x3F,8000~ 0x3F,9FFF	8K×16 位 片上 H0 SARAM	8K×16 位 片上 H0 SARAM	0 等待	
0x3F,A000~ 0x3F,EFFF	20K×16bit 保留	20K×16bit 保留		
0x3F,F000~ 0x3F,FFFF	4K×16 位 片上 BOOT ROM	4K×16 位 片上 BOOT ROM	1 等待	MP/MC=0

3.2.3 外部扩展控制/状态寄存器

C2000 实验箱上配置有 LED、USB 接口、LCD、DAC、控制寄存器和状态寄存器等，它们都映射在 F2812 的 Zone0 或 Zone1 存储空间中，具体的定义如下表 3.2 所示：

表 3.2 外部扩展控制/状态寄存器地址

名称	地址	操作	访问周期
LED8	0x00,2000	8 位，只写	Ts=6.67ns Tw=10ns Th=0ns
LED7	0x00,2100		
LED6	0x00,2200		
LED5	0x00,2300		
LED4	0x00,2400		
LED3	0x00,2500		
LED2	0x00,2600		
LED1	0x00,2700		
LED 数据更新	0x00,2C00	写任意数	
USB	0x00,2800 ~0x00,28FF	16 位，读/写	Ts=10ns, Tw=50ns, Th=70ns
DAC1	0x00,2900	16 位，只写	Ts=10ns, Tw=10ns, Th=5ns
LCD	0x00,2A00 0x00,2A01	8 位，读/写	Ts=10ns, Tw=50ns, Th=70ns
状态寄存器	0x00,2D00	16 位，只读	Ts=6.67ns, Tw=10ns, Th=0ns
控制寄存器	0x00,2D00	16 位，只写	
FIFO 复位	0x00,2E00	写任意数	Ts=6.67ns, Tw=10ns, Th=0ns

注：

- (1) Ts 为建立时间，Tw 为读写宽度，Th 为保持时间
- (2) LED 为 8 段数码显示管
- (3) DAC1 为 16 位高速 DAC，与输出口 OUT3 相连

状态寄存器地址为 0x00,2D00，只读，各位含义如下图 3.4 所示：

D7	D6	D5	D4	D3	D2	D1	D0
保留	保留	FIFO_EF	USB_INT	USB_RDY	FLAGC	FLAGB	FLAGA

图 3.4 状态寄存器（0x00,2D00）各位含义

FIFO_EF：外扩 FIFO 标志，低电平表示有数

USB_INT：USB 中断信号，低电平有效

USB_RDY：USB 的 READY 的状态

0：USB 未就绪

1：USB 已就绪

FLAGC：USB 的 FLAGC 的状态

0：有效

1：无效

FLAGB：USB 的 FLAGB 的状态

0：有效

1：无效

FLAGA：USB 的 FLAGA 的状态

- 0: 有效
1: 无效

控制寄存器地位也为 0x00,2D00，只写，各位含义如下图 3.5 所示：

D7	D6	D5	D4	D3	D2	D1	D0
保留	保留	USB_WKUP	LCD_LIGHT	保留	FFT_P1	FFT_P2	DA11

图 3.5 控制寄存器（0x00,2D00）各位含义

USB_WKUP: CY7C68001 唤醒，复位为 0

- 0: 休眠
1: 唤醒

LCD_LIGHT: LCD 背光灯控制

- 0: 关
1: 开

FFT_P1, FFT_P0: FFT 运算点控制

- 00: 256 点
01: 512 点
10: 1024 点
11: 2048 点

DA11: 双端口 SRAM 上、下两部分切换

- 0: 0x08,0000~0x08,7FFF 内容送 DA2
1: 0x08,8000~0x08,FFFF 内容送 DA2

3.2.4 外部扩展输入输出

1. 外扩 10 路数字输出和 8 路数字输入（GPIO）

DOUT 为输出 IO，DIN 为输入 IO。外扩的 GPIO 口对应 C2000 实验箱 JP10 插座，其对应关系如下表 3.3 所示。

表 3.3 JP10 对应的数字输入输出端口

DSP_GPIO	信号名称	JP10 管脚号	JP10 管脚号	信号名称	DSP_GPIO
	VCC	30	29	VCC	
	GND	28	27	GND	
	GND	26	25	DOUT9	GPIOB5
	GND	24	23	DOUT8	GPIOB4
GPIOB3	DOUT7	22	21	DOUT6	GPIOB2
	GND	20	19	DOUT5	GPIOB1
GPIOB0	DOUT4	18	17	DOUT3	GPIOA03
	GND	16	15	DOUT2	GPIOA02
GPIOA01	DOUT1	14	13	DOUT0	GPIOA00
	GND	12	11	GND	
GPIOA15	DIN7	10	9	DIN6	GPIOA14
	GND	8	7	DIN7	GPIOA15
GPIOA12	DIN4	6	5	DIN3	GPIOA11
	GND	4	3	DIN2	GPIOA10
GPIOA09	DIN1	2	1	DIN0	GPIOA08

2. 发光二极管

6 只发光二极管（低电平有效）对应的 DSP 的 GPIO 口为：

表 3.4 发光二极管对应的 GPIO 接口

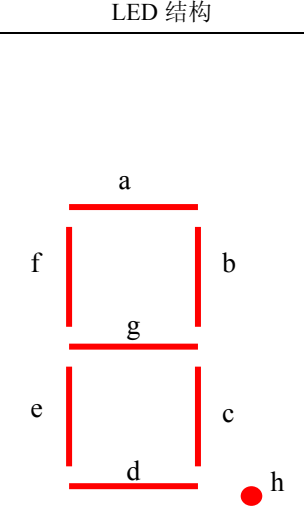
I1	I2	I3	I4	I5	I6
GPIOF11	GPIOF8	GPIOF10	GPIOF12	GPIOF9	GPIOF13

3. LED 数码显示管

8 个 LED 为共阴极显示管，其端口地址如表 3.2 所示。将字符的显示码送入相应地址后，发出更新命令（即在 0x00,2C00 写任意数）即可使 LED 显示相应字符。

LED 的数符对应码字如下表 3.5 所示

表 3.5 LED 结构及对应码字

LED 结构	数符	码字
 <p>数码管编码格式 hgfe,dcba</p>	0	0x3F
	1	0x06
	2	0x5B
	3	0x4F
	4	0x66
	5	0x6D
	6	0x7D
	7	0x07
	8	0x7F
	9	0x6F
	A	0x77
	B	0x7C
	C	0x39
	D	0x5E
	E	0x79
	F	0x71

4. 存储空间的扩展

C2000 实验箱中的 TMS320F2812 DSP 扩展了三个外扩存储器，它们分别是 256K×16 位的 SRAM、4K×16 位的双端口 SRAM 和 16K×8 位的 FIFO，其地址映射空间如表 3.1 所示。

DSP 外扩了两片 FIFO，对 DSP 而言，一片只读，另一片只写，映射的物理地址相同。可通过外部 FIFO 接口（JP9）分别对两片 FIFO 进行读写，交换数据。

5. SPI（串行外设接口）

串行外设接口（SPI）是一个高速同步串行输入输出接口，允许可编程位长的串行位流（1～16 位）以可编程的位传输速率移入或移出期间。SPI 通常用于 DSP 控制器和外设或另一个处理器之间的通信。

TMS320F2812 有一个 SPI 同步串口，在 C2000 实验箱上的串行 EEPROM AT25080 与 DSP 无缝连接。

6. SCI（异步串口）

TMS320F2812 有两个异步串口（SCI），在实验箱中分别经 RS485 和 RS232 电平转换芯片连接到 JP4 插座，提供外部使用。其中 RS232 电平可直接与计算机实现通信。

RS485 与 TMS320F2812 的 SCI_B 相连接，收发有 GPIOB6 引脚控制：GPIOB6 为低电平时，数据发送；GPIOB6 为高电平时，数据接收。

RS232 与 TMS320F2812 的 SCI_A 相连接。

具体接口如下表 3.6 所示。

表 3.6 JP4 对应的 SCI 引脚

JP4 管脚号	引脚名称	
2	TXD	RS232
3	RXD	
5	GND	
7	A	RS485
8	B	
1、4、6、9	NULL	

7. CAN（控制局域网络）

TMS320F2812 内部有增强型控制局域网络（eCAN）控制器，它完全兼容 CAN 2.0B 标准，数据传输速率最高可搭 1Mbps，经 CAN 收发器驱动后连接到 JP5 接口，可供外部使用。具体接口如下表 3.7 所示。

表 3.7 JP5 对应的 CAN 引脚

信号名称	JP5 管脚号	JP5 管脚号	信号名称
CAN+	1	2	CAN-

8. JTAG

C2000 实验箱中有两个 JTAG 引脚，分别对应着 DSP 与 CPLD。

DSP（TMS320F2812）的 JTAG 插座为 JP1。

CPLD（EPM7256）的 JTAG 插座为 JP2。

9. USB 接口

C2000 实验箱采用了 Cypress 公司的 CY7C68001 芯片实现 USB 2.0 接口。CY7C68001 上集成了 USB 2.0 收发器（物理层）、USB 2.0 串行接口引擎 SIE（链路层，实现底层通信协议）。CY7C68001 则作为 TMS320F2812 的外设，USB 的应用层协议由 TMS320F2812 编程实现。

CY7C68001 采用并行异步存储器接口与 TMS320F2812 相连接，主机可以唤醒 TMS320F2812，亦可以配置 USB。USB 接口芯片占用 F2812 的 Zone0 空间，地址为 0x002800～0x0028FF。地址分配如下表 3.8 所示：

表 3.8 USB 芯片接口在 DSP 中的映射地址

访问空间类型	FIFOADR[2:0]	TMS320F2812 映射地址
FIFO2	000	0x00,2800
FIFO4	001	0x00,2801
FIFO6	010	0x00,2802
FIFO8	011	0x00,2803
命令端口	100	0x00,2804

保留	101	0x00,2805
保留	110	0x00,2806
保留	111	0x00,2807

CY7C68001 除了存储器接口外,还有 1 个中断信号 USB_INT 和 4 个状态信号(READY、FLAGA、FLAGB 和 FLAGC)。中断信号 USB_INT 占用 TMS320F2812 的外部中断 XINT2,状态信号 READY、FLAGA、FLAGB 和 FLAGC 配置在状态寄存器(USB_STS)中,可由 TMS320F2812 查询。TMS320F2812 还可以通过控制寄存器将 USB 唤醒,具体参见外扩状态、控制寄存器。

CY7C68001 有 EEPROM 和 DSP 两种自举方式,均可以使用,一般采用 EEPROM 来进行 USB 的初始化。

10. 实时时钟

C2000 实验箱上配置有 I²C 串口的实时时钟 RTC+NvSRAM,可以产生年、月、日、星期、时、分、秒等实时时间信息,片上集成了 512×8 位 NvSRAM,可以用来存储数据。I²C 串口有 2 个信号 SDA 和 SCL,分别接到 GPIOE2/NMI_INT13 和 GPIOF14/XF_PLDIS 引脚上。详细资料和有关技术指标可阅读“X1226 Datasheet”。

11. 模拟输入

C2000 实验箱上有 2 路模拟量输入,由 BNC 座(INPUT1、INPUT2)输入,此模拟电压信号经过前端的低通滤波器,滤除不必要的高频噪声信号,以及将模拟输入信号范围由 ±3V_{pp} 变换成 TMS320F2812 的 ADC 所能接受的信号范围 0~3V。前端每一路的模拟调理电路均带有保护电路,防止烧毁 DSP。INPUT1 通过模拟调理电路送到 F2812 的 ADCINA0; INPUT2 通过模拟调理电路送到 F2812 的 ADCINB0。

12. 模拟输出

C2000 实验箱上有 2 路模拟量输出,通过 BNC 座(OUT2、OUT3)输出。实验 2 板上有 2 路 ADI 公司的 16bit/30ns DAC 芯片(AD768),1 路 DSP 通过端口地址(0x002900H)直接输出,主要用于 FIR、IIR 运算数据流输出,对应 OUT3 模拟输出口。

另 1 路 DAC 通过双端口存储器(0x080000H~0x080FFFH,4K×16 位)输出,主要用于 FFT 等运算数据块输出,对应 OUT2 模拟输出口,该双端口存储器组成乒乓结构,由外扩控制寄存器控制切换。也就是说,DSP 将最新运算结果送到双端口存储器上半部分(地址范围 0x08,0000~0x08,7FFF),然后切换地址(控制寄存器 0x002D00 中的 D0=1),将该存储器中下半部分(地址范围 0x08,8000~0x08,FFFF)数据,通过 DAC 输出;在下一时刻,然后再切换地址(控制寄存器 0x002D00 中的 D0=0),循环操作。实验版上还有模拟输出的同步信号(BNC 座,OUT1),可接至示波器外触发端。

13. 人机接口

C2000 实验箱上提供一个 LCD 点阵液晶显示接口,用于连接符合 Intel 读/写时序的 240×128 点阵 LCD 模块,以显示图形和字符等信息。另外,还提供 4×5 键盘接口,可接受 19 个按键输入。

对于点阵 LCD 液晶(采用 T6963C 控制器)显示接口来说,对其的读/写访问通过 1 个 8 位的命令口和 1 个 8 位的数据口进行。LCD 液晶显示接口分配在 TMS320F2812 的 Zone0 空间中,占用 2 个地址单元:0x002A00H, LCD 液晶显示接口的数据口;0x002A01H, LCD 液晶显示接口的命令口。其 LCD 接口连接如下表 3.9 所示。

表 3.9 LCD 接口说明

信号线	功能	备注
LCD_D[7:0]	8 位数据线	DSP 的最低 8 位数据
LCD_CE	片选信号	0x2A00 (数据), 0x2A01 (命令)
LCD_I/D	命令/数据选择信号	0: 数据; 1: 命令
LCD_RD	读信号	与 DSP 的 RD 信号相连
LCD_WE	写信号	与 DSP 的 WE 信号相连
LCD_RST	复位信号	与 DSP 的 RESET 信号相连
LCD_LIGHT	背光控制信号	控制寄存器的 D4 位

C2000 实验箱上, 还设计有 4×5 键盘接口。TMS320F2812 的 GPIOB[15: 12]作为 4 根键盘扫描输出线, TMS320F2812 的 GPIOB[11: 7]作为 5 根键盘扫描回读线, 最多可组成 20 个 (在此用 19 键) 按键的键盘。键盘的扫描、回读、去抖等时序由软件编程实现。键盘的连线和定义如下图 3.6、表 3.10 所示。

表 3.10 键盘与 DSP 接口引脚

KS0	KS1	KS2	KS3	KD0	KD1	KD2	KD3	KD4
GPIOB 12	GPIOB 13	GPIOB 14	GPIOB 16	GPIOB 07	GPIOB 08	GPIOB 09	GPIOB 10	GPIOB 11

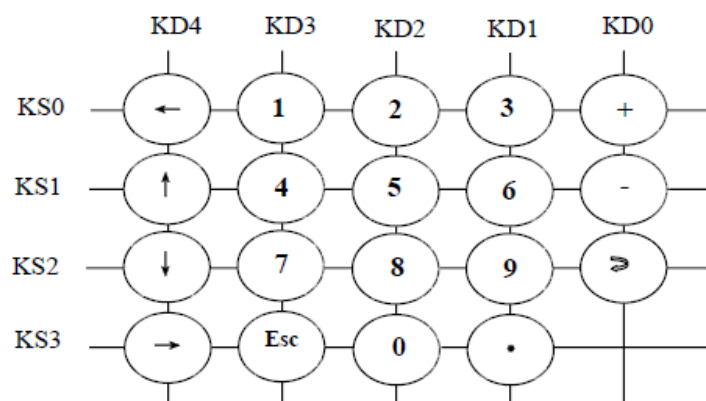


图 3.6 键盘接口连接电路

14. 复位与中断向量表

TMS320F2812 复位时, 将终止所有的当前操作, 使 CPU 进入已知的初始状态, 刷新流水线操作, 复位所有的 CPU 寄存器, 复位相关的信号的状态。复位完成后, CPU 从 0x3FFFC0H 处取复位向量到 PC 寄存器中, 然后开始执行程序。

如果 XMP/MC 引脚为低电平, 则程序空间高 16K×16 位映射为片上 Boot ROM, 片上 Boot ROM 在 0x3FFFC0H 单元中存放 0x3FFC00H, 也即程序将从片上 Boot ROM 的 0x3FFC00H 处开始执行, 片上 Boot ROM 中 0x3FFC00H 起始的 1K×16 位存储空间中存放的是 BootLoader 程序。如果 XMP/MC 引脚为高电平, 则程序空间高 16K×16 位映射为片外 Zone7, 用户应该在 Zone7 的 0x3FFFC0H 处存放 CPU 中断向量表。

在 JP6 上插上短路块, F2812 则工作在 MC 方式, 上电从片内 Boot ROM 中执行程序, GPIOF04/SCI_TXDA 片内已上拉, 则选中 Jump to Flash Boot 方式, 程序将跳转到片内 Flash 的 0x3F7FF6H 处, 在 0x3F7FF6H 存放跳转指令, 跳转到实际的应用程序中, 应用程序首先

初始化 PIE 中断向量表，然后使能 PIE。注意用 Jump to Flash Boot 方式时，没有停止片内看门狗电路，所以应在规定的时间内刷新看门狗电路，否则将导致看门狗溢出，产生复位。

实验板上有四个复位源：上电复位、手动复位（按键 K1）、看门狗电路与电源监测。任何一个复位有效，将导致整个系统复位。

TMS320F2812 有 3 个中断引脚：XINT1，XINT2 和 XNMI_INT13，每个中断可配置为上升沿或下降沿触发，也可以被使能或禁止。实验板上只用到 XINT2，为 USB 中断。

15. TMS320F2812 工作方式配置

实验箱由 JP6 选择 DSP 工作模式，JP6 按上“短路块”，将 MP/MC 接地，TMS320F2812 工作在微计算机 MC 方式；去掉“短路块”，MP/MC 上拉电阻，TMS320F2812 工作在微处理器 MP 方式。

在 MC 方式，DSP 的 SCITXDA 连接了上拉电阻，使 DSP 工作在“Jump to Flash”上电自举方式。

实 验 内 容

1. DSP 基础实验

1.1 实验目的

1. 了解 DSP 开发系统的基本配置
2. 熟悉 DSP 集成开发环境（CCS）
3. 掌握 C 语言开发的基本流程
4. 熟悉代码调试的基本方法

1.2 实验仪器

计算机，C2000 DSP 教学实验箱，XDS510 USB 仿真器

1.3 实验内容

建立工程，对工程进行编译、链接，载入可执行程序，在 DSP 硬件平台上进行实时调试，利用代码调试工具，查看程序运行结果。

1.4 实验准备

CCS 2（C2000）这一集成开发环境，不仅支持汇编的编译、链接，还支持对 C/C++ 汇编、编译、链接以及优化。同时强大的 IDE 开发环境也为代码的调试提供了强大的功能支持，已经成为 TI 各 DSP 系列的程序设计、制作、调试、优化的主流工具。

TMS320C28x 软件开发流程如图 1.1 所示。其中 C 语言程序开发流程用阴影部分标出，其他部分是代码开发过程中的一些增强功能。

下面简单介绍各主要模块功能。

- C/C++ Compiler（C/C++ 编译器）

C/C++ 编译器把 C/C++ 程序自动转换成 C28x 的汇编语言源程序。这种转换并非一一对应，甚至会产生冗余的汇编代码，在某些场合需要使用优化器（Optimizer）来提高转换的效率，使得汇编代码长度尽可能的短小，程序所使用的资源尽可能的少。优化器是编译器的一部分。

- Assembler（汇编器）

汇编器负责将汇编源程序转换为符合公共目标格式（COFF）的机器目标代码，这种转换是一一对应的，每一条汇编指令都对应了唯一的机器代码。源文件中还包括汇编指令、伪指令和宏指令。

- Linker（链接器）

链接器负责把可重定位的多个目标文件和目标库文件转换为一个 DSP 可执行程序。链接器必须依赖配置命令文件（CMD）的指令，实现对目标文件中各段的定位。

- Run-time-support library（运行支持库）

对于用 C/C++ 语言中编写 DSP 程序中的某些功能（例如存储器的寻址定位、字符串转换等）并不属于 C/C++ 语言所能描述对象，包含在 C/C++ 编译器中的运行支持库却可以很好的支持这些算法的标准 ANSI/ISO C 函数描述。函数运行支持库包含有 ANSI/ISO C 的标准运行支持库函数、编译器功能函数、浮点算术函数和系统初始化子程序（这些函数都集成在汇编源文件 rts.src 中）。当对 C/C++ 编写的 DSP 程序进行链接时，必须根据不同型号的 DSP 芯片添加相应的运行支持库到工程中。除此之外，在使用运行支持库中的函数时，必须在程序起始用 `include` 语句包含相应的头文件（如使用数学运算 `sin`、`cos` 时，必须包含 `math.h`）。而采用汇编语言编写程序时，却不需要这个运行支持库。因此 C 语言编写的 DSP 程序链接后，会产生大量的“冗余”汇编程序。

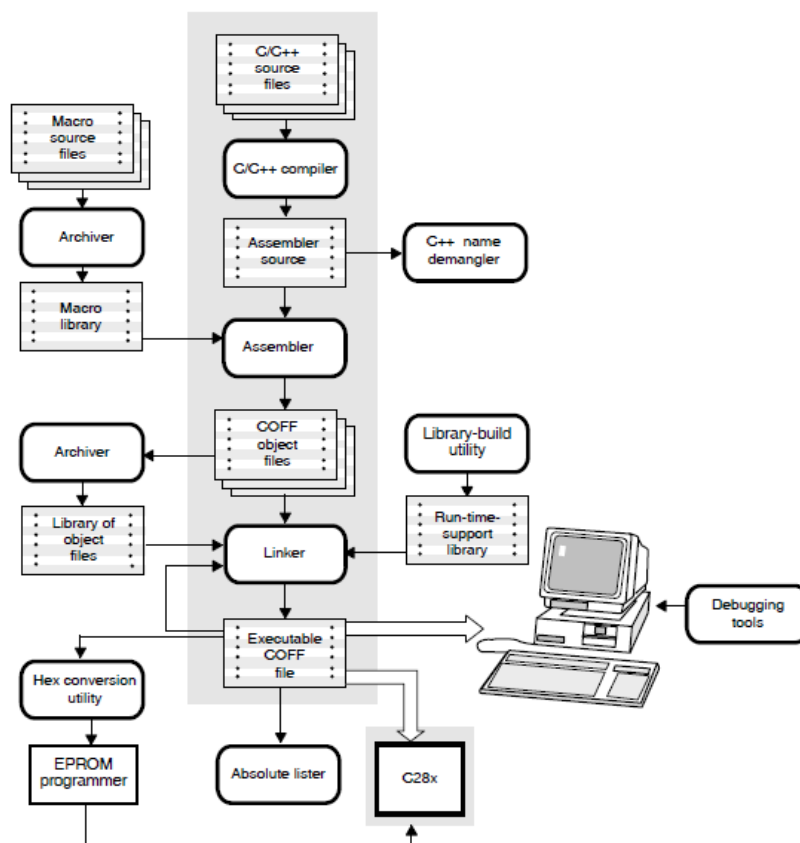


图 1.1 TMS320C28x 软件开发流程

由此可见，用 C/C++ 语言来开发 DSP 程序，一般在工程中必须包含以下文件：

- `.c` 或者 `.cpp`：C 或 C++ 程序，是主程序或函数，用于描述用户特定的算法功能；
- `.cmd`：配置命令文件，用于对编译生成的 COFF 格式目标文件（`.obj`）定位，安排各段的物理存储空间；
- `.lib`：运行支持库文件，不同芯片有不同的运行支持库，必须根据具体芯片加以选择，例如 TMS320C28x 的运行支持库文件命名为 `rts2800.lib` 或 `rts2800_ml.lib`。（后缀 `ml` 含义是 `large memory model`，由于 C/C++ 默认的寻址空间为 64K，而 TMS320C28x 地址范围可达 4M，由此如果要访问高于 64K 的空间，必须在工程中添加 `rts2800_ml.lib`）。

至于头文件（`.h`），只有当使用了运行支持库中相应的函数时，才需要在 C 文件的主程序中用 `include` 语句指定相应的头文件（`math.h`、`stdlib.h`、`float.h` 等）。具体内容参见 TI 公司的 TMS320C28x Optimizing C/C++ Compiler User's Guide。

其次用户自定义函数、寄存器地址、常量定义等信息也可以编制到头文件中，使用时也同样需要在 C 主程序中指定。

例如本实验中，需要的文件：

- sine.c: C 语言主程序。
- sinewave.cmd: 配置命令文件。
- rts2800_ml.lib: 运行支持库。
- Sine.h: 常量定义头文件。
- sine.dat: 实验中需要的数据文件。

对于使用 CCS 以工程为单位进行 DSP 程序的项目开发时，一般为每个工程建立一个独立的目录，将项目中所需要的文件都存放在该工程目录下，便于程序的管理。rts2800_ml.lib 在 TI 的安装目录...\\TI\\c2000\\cgtools\\lib 中可以找到。

1.5 实验步骤

1. 设备检查

检查仿真器、C2000 DSP 实验箱、计算机之间的连接正确，打开计算机和实验箱电源。

2. 启动集成开发环境

点击桌面 CCS 2 (C2000) 快捷方式，或者点击开始菜单中“[开始] 菜单\\程序\\Texas Instruments\\Code Composer Studio 2 (C2000)”，启动 CCS。若硬件连接正确，实验箱电源被正确加载，则可以进入集成开发环境 CCS，并在 CC 软件的标题栏显示“/F28xx XDS510 Emulator / CPU_1 – 28xx – Code Composer Studio”。如果 CCS2 SETUP 设置错误或者 DSP 电源没有正确加载，则会显示错误对话框，无法进入 CCS Emulate 模式。

3. 新建工程

在主菜单中单击“Project → New”命令，弹出“Project Creation”对话框。在第一项 Project Name 中输入新建的工程名称，在第二项 Location 中选择工程所在目录，第三项 Project Type 中选择输出文件格式“Executable (.out)”，在第四项 Target Family 中选择与当前 DSP 芯片吻合的 TMS320C28XX。单击“完成”按钮确定。则在工程指定的目录中，建立了一个以工程命名的工程文件 (.pj1)，它会存储有关该工程的所有设置。

4. 添加工程文件

在主菜单中单击“Project → Add Files to Project”命令，在弹出的对话框中依次选择当前工程目录下 sine.c、sinewave.cmd 以及 rts2800_ml.lib 文件，添加到当前工程中。在工程浏览窗口中，展开工程文件列表，可看到刚刚所添加的文件。

头文件 (.h) 不用专门人工指定添加，在后续的 build 工程时，CCS 会自动完成关联文件的扫描，自动添加到工程中。

如果错误的添加了文件，可以在工程浏览窗口中的文件名中单击鼠标右键，在弹出的菜单中选择“Remove from project”。

当然，CCS 也支持文件编辑功能，可以在主菜单选择“File → New”新建一个文件，编辑完成保存为所需要相应格式的 C 语言程序、汇编程序、cmd 配置命令或者头文件，然后添加到工程中。

5. 查阅代码

在 build 工程之前，先阅读一下源代码，明白各文件的内容：鼠标双击工程浏览窗口里的“sine.c”文件，即可在 CCS 的编辑窗口看到 c 程序的源代码，代码中有以下四个部分：

- Disable_WD() 函数，禁止 TMS320F2812 内部的看门狗工作；

- 在主函数输出消息“SineWave example started”之后，进入一个无限循环，在循环体内调用了两个函数 `dataIO()` 和 `processing()`。

- 函数 `dataIO()` 在本实验中，DSP 不作任何实际操作而直接返回。

- 函数 `processing()` 对输入缓冲区的每个数据进行增益控制，并将结果存入输出缓冲区中。

6. 建立工程 (Build 工程)

建造工程 (build) 是指对 `asm`、`c` 源程序文件进行编译 (Compile)、汇编 (Assemble)，并结合配置命令文件对工程进行链接 (Link)，输出可执行程序 (`.out`)。该命令在主菜单“Project → Build (Build All)”中。生成的可执行 `.out` 程序位于工程目录的 `debug` 子目录下。

这三个步骤也可以手工分步执行，比如对源程序 `Compile` 会生成 COFF 格式的目标文件 (`.obj`)，然后用 `build` 命令完成链接。

对工程文件中的语法或是链接错误，CCS 会终止当前的 `build`，在底部消息窗口指示出程序包含的编译链接错误，或是警告信息。根据错误提示修改源程序文件或者配置命令文件，直至编译链接正确。

以上的工作称为目标代码生成。

9. 加载程序

当工程被正确建立以后，只有将程序通过仿真器下载到 DSP 芯片上，才能够进行实时的代码调试。

在主菜单下，选择“File → Load Program”，选中该工程的输出可执行 `.out` 程序即可。

CCS 装载程序完毕以后，会自动弹出“Disassembly”窗口，显示构成源代码的反汇编指令。

10. 程序的运行

对 C 语言编写的程序，载入程序以后，DSP 的程序计数器 (PC) 会自动指向 `_c_int00`，这是 C 程序链接以后的入口地址（在反汇编窗口上有一个绿色箭头标识）。它是运行支持库自动在用户 C 程序之前添加的一段初始化程序。如果想让程序计数器指向 C 语言的起始处，在菜单中选择“Debug → Go Main”命令，让程序从主函数开始执行（在源程序窗口有一个黄色箭头标识）。

倘若想同时看到 C 语言代码和对应编译生成的编译代码，在主菜单“View → Mixed Source/ASM”，浅灰色部分显示的就是汇编指令。可见一条 C 语言命令对应了很多条汇编指令，这是由于 C 语言的每一条语句并不是对应了 DSP 内部的硬件的动作，C 语言不适合描述 DSP 底层的硬件机制。

在主菜单中选择“Debug → Run”，可以让 DSP 全速运行，由于 DSP 程序输出并不具备 GUI 界面，由此执行结果只有依赖外部硬件或者查看寄存器、存储器的数值加以验证。在主菜单选择“Debug → Halt”命令，可以停止程序的执行。DSP 指令的执行严格按照指令流的顺序。

当想再次运行程序，可以执行菜单命令“Debug → Restart”，使 PC 重新指向 `_c_int00`；也可以重新加载程序。

当执行菜单命令“Debug → Reset CPU”时，DSP 复位，内部寄存器恢复默认值，PC 指向中断矢量表的复位向量处。

11. 程序的调试

在程序的开发与测试过程中，常常需要检查某个变量、或者是存储器的数值在程序运行过程中变化情况，这就需要停止程序执行，用断点与观察窗口等方式来验证数值的正确性。这就是 DSP 目标代码的调试。

添加结构体变量 `currentBuffer` 到变量观察窗口，观察 `currentBuffer.output` 和 `currentBuffer.input` 的地址以及数值。添加 `dataIO()` 到变量窗口，查看该子程序的入口地址。

在 `dataIO()` 处设立探针，关联输入文件 `sine.dat`，设置数据加载的起始地址为 `currentBuffer.input`，长度为 128。

打开图形显示功能，查看存储空间 `currentBuffer.input` 和 `currentBuffer.output` 的时域波形。全速运行程序，或者动画执行程序，查看以上存储空间的数值变化。

在 `processing()` 子程序中设置断点，分别执行主菜单命令“Debug → Step into”和“Debug → Step over”单步执行程序，以及“Debug → Assembly/Source Stepping”中的各项命令，查看并比较这些单步执行方式的区别。

具体方法参见实验准备中相关内容。

1.6 实验要求

1. 独立完成项目编译、链接、调试的全过程。
2. 记录 `dataIO()`、`processing()` 子程序的入口地址，记录 `currentBuffer.input` 和 `currentBuffer.output` 所在存储器地址。
3. 记录增益控制处理后，以图形方式显示数据空间 `currentBuffer.input` 和 `currentBuffer.output` 缓冲存储器中的波形。
4. 打开工程的 `.map` 文件，查看 `.text`、`.data`、`.bss` 段在存储空间的地址和长度，指出分别位于 TMS320F2812 的什么存储空间以及物理存储块名称。
5. 查看 `.cmd` 命令文件，比较其与上述 `.map` 中的映射关系。试图修改 `.cmd` 文件，再次编译链接，查看配置命令与各段的映射关系。

1.7 注意事项

运行 CCS 集成开发软件后，务必确保 DSP 实验箱电源加载正常。

2. 任意信号发生器

2.1 实验目的

1. 熟悉 DSP 硬件开发平台
2. 熟悉 DSP 集成开发环境（CCS）
3. 掌握 TMS320F2812 的存储器配置表
4. 学习 TMS320F2812 的编程开发
5. 熟悉代码调试的基本方法

2.2 实验仪器

计算机，C2000 DSP 教学实验箱，XDS510 USB 仿真器，示波器

2.3 实验内容

建立工程，编写 DSP 的主程序，并对工程进行编译、链接，利用现有 DSP 平台实现任意波的产生，通过示波器观察结果。

2.4 实验准备

2.4.1 程序流程

在 DSP2000 实验平台上实现任意波形的产生，可通过 DSP 实时运算得到相应波形的数据，随后通过 DAC 完成模拟输出。在该实验中，我们利用 DSP 的运算能力，首先计算出波形的数值信息，存储到相应的数据空间中，通过查表的方式读取该波形的数值并写入到 DAC 端口，实现任意波形的生成。由此可得程序流程如图 2.1 所示。

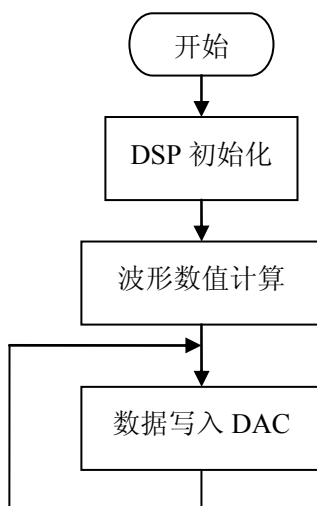


图 2.1 任意波形发生程序流程

该实现方式属于查表法，类似与直接数字频率合成 DDS 的数字产生部分原理，可以改变相位控制字来改变输出信号的频率。

实现程序可参照工程 RamGen_C（C 语言格式）或 RamGen_Asm（汇编语言格式）。

2.4.2 数据的定标

TMS320C28xx 是定点 DSP 芯片，采用定点数进行数值的运算，其操作数一般采用整型或长整型数据。数据最大表示范围取决于 DSP 芯片给定的字长，字长越长，所能表示数的范围就越大，精度也越高。DSP 芯片的数以 2 进制补码格式表征，最高位是符号位，其余 15 位表示数值的大小。

而在实际中，数值的大小、数据的运算都会带来小数，用定点数格式表示小数，确定小数点的位置，称之为数据的定标。数据的定标一般有 Q 表示法，即 Q15 表示在定点数格式中有 15 位小数。由此 16 位定点数有 16 种 Q 表示形式，对应了 16 种十进制数据范围。例如 16 位定点的 Q0 表示没有小数，数据范围[-32768,32767]；Q4 表示有 4 位小数，数据范围[-2048,2047.9375]；Q15 表示有 15 位小数，数据范围[-1,0.9999695]。可见，不同 Q 所表示的数据范围和精度都有所不同，精度与范围是一对矛盾，在实际定点算法中，为了达到最佳性能，必须对数据进行合理的定标。

浮点数 X_F 与定点数 X_D 的转换关系可表示为：

$$\text{定点数 } X_D = \lfloor X_F \times 2^Q \rfloor$$

$$\text{浮点数 } X_F = X_D \times 2^{-Q}$$

在程序中，根据数据的动态范围来确定 Q 值，分析程序中的数据可能的绝对值最大值 $|\max|$ ，使下式成立：

$$2^{n-1} < |\max| < 2^n$$

则 $Q = 15 - n$ 。

例如某变量的值在 -1 到 +1 之间，即 $|\max| < 1$ ，因此 $n = 0$ ， $Q = 15$ 。

2.4.3 相关实验硬件资源

TMS320F2812 内部采用哈佛结构总线，与 TMS320F24xx 以及 TMS320F206 系列 DSP 不同，内部的程序空间、数据空间采用统一的编址方式。其 memory map 参见《实验准备》中图 3.3。其次 TMS320F2812 支持 32 位格式的数据访问，32 位的数据访问必须从偶地址开始。由于 TMS320C28xx 绝大部分指令采用 32 位，因此，当程序存放到程序空间时，必须分配到偶数地址空间。

除了 TMS320F2812 片上集成的存储器，在 DSP2000 实验箱上还扩展了 RAM、FIFO、双端口存储器等资源，供实验者使用。其地址分配如下表 2.1 所示。

表 2.1 外扩存储器地址映射

地址范围	存储体	等待时间	备注
0x08,0000~0x08,0FFF	双端口 RAM	至少 2 等待	占 ZONE2
0x10,0000~0x13,FFFF	SRAM	至少 2 等待	占 ZONE6
0x14,0000~0x14,FFFF	FIFO	至少 4 等待	占 ZONE6

DSP 处理器的外部接口（XINTF）负责完成对外扩设备的连接管理。TMS320F2812 的 XINTF 映射到 5 个独立的存储空间，分别是 ZONE0、ZONE1、ZONE2、ZONE6 和 ZONE7。每一个空间都有一个内部的片选信号，并可以通过编程来独立的配置访问等待、选择、建立以及保持时间，以实现 TMS320F2812 与各种外部存储器或设备的无缝连接。

实验箱上的 DAC1 采用的是 AD768，位宽 16bit，数据以无符号数表示，转换速度 30ns，通过 OUT3 端口输出，在 TMS320F2812 的地址映射为 0x2900（只写）。即 DSP 只要将数字信号写到该端口，DAC1 自动完成模拟的转换。

实验箱中的 8 个数码显示管为共阴极显示管，即只要对相应的显示位写 1，就可点亮该位，写 0，则熄灭该显示位。数码显示管的结构以及对应字符表见《实验准备》中实验箱外部扩展输入输出相关内容。

若要使 LED 显示字符，可先往相应的端口写入字符对应的码字，而后往 LED 数据更新端口写任意数，即可刷新 LED 显示的字符。比如 LED1 显示字符“A”，先往端口 0x2700 写入字符 0x77，随后往端口 0x2C00 写入 0x00。

2.5 实验步骤

1. 设备检查

检查仿真器、C2000 DSP 实验箱、计算机之间的连接正确，打开计算机和实验箱电源。

2. 启动集成开发环境

点击桌面 CCS 2 (C2000) 快捷方式，进入集成开发环境 CCS。

3. 新建工程

新建一个 DSP 工程，编辑源程序、配置命令等相关文件，并在工程中添加这些程序文件。

要求产生一个线性调频信号，其数学表达式如式 2.1 所示：

$$s(t) = \cos(\pi K t^2) \quad (2.1)$$

其中调制斜率 K 为 39062，t 为持续时间是[-0.0128,0.0128]，在采样时间内共 1024 个采样点，即有 1024 个离散数值。该波形的离散值计算后存储在 DSP 的数据空间中。

源程序的编写可参照工程 RamGen_C (C 语言格式) 或 RamGen_Asm (汇编语言格式) 中的相关内容。

4. 建立工程 (Build)

建造工程 (build)，若出错，则根据错误提示，修改源程序文件或者配置命令文件，直至编译链接正确，生成可执行的.out 文件。

5. 加载程序

在主菜单下，选择“File → Load Program”，将程序下载到 DSP 内部。

6. 调试程序

在程序中的“波形数值计算”子模块后设置断点，运行程序后 PC 指针会停留在此处，打开图形显示功能，查看存储空间中保存的时域波形，是否为线性调频信号。如果不是，则重新修改程序，直至正确为止。

程序调试时，可以利用各种调试手段，比如打开寄存器窗口、变量窗口等辅助手段，查看数值计算是否满足要求。

7. 运行程序

若第六步正确，可去掉断点，重新全速运行程序。

连接 C2000 实验箱 OUT3 输出口至示波器，调节示波器，观察线性调频信号的输出。

2.6 实验要求

1. 独立完成项目编译、链接、调试的全过程。

2. 利用数码显示管，在 DSP 初始化子模块后添加语句或者编写子程序，使之能够显示实验日期。

3. 记录实验中个子程序包括主程序的入口实际地址，与 memory 比较，指出分别位于什么类型的存储器中。

3. 指出波形数据保存的空间地址，并以图形方式显示线性调频信号的波形，并保存，附在实验报告中。

2.7 注意事项

运行 CCS 集成开发软件后，务必确保 DSP 实验箱电源加载正常。

2.8 实验思考

1. 打开工程的.map 文件，查看除.text、.data、.bss 段之外，还有哪些有实际长度的段，查找相关资料，指出其含义与作用。
2. 在保持源文件功能正确的前提下，仅修改.cmd 配置命令文件，改变段的地址分配，链接工程后，执行程序，如果出现错误，思考原因。
3. 在不修改波形数值计算子模块前提下，即保持波形数值表中的数据，依照 DDS 原理，修改程序，调整线性调频信号的输出周期。

3. DSP 数据采集

3.1 实验目的

1. 熟悉 DSP 的软硬件开发平台
2. 掌握 TMS320F2812 的 ADC 外设的使用
3. 熟悉 TMS320F2812 的中断的设置
4. 掌握代码调试的基本方法

3.2 实验仪器

计算机, C2000 DSP 教学实验箱, XDS510 USB 仿真器, 示波器, 信号源

3.3 实验内容

建立工程, 编写 DSP 的主程序, 并对工程进行编译、链接, 利用现有 DSP 平台实现数据的采集、存储以及模拟还原, 通过图表以及、示波器观察结果。

3.4 实验准备

3.4.1 程序流程

为实现 DSP 的数据采集存储以及模拟的还原, 必须依赖于 ADC、DSP 以及 DAC 三大基本部件, 而 TMS320F2812 芯片上集成了外设 ADC, 因此实现该功能较为简单, 数据采集的工作可以由 DSP 单独完成, 只需要对相关外设进行配置。模拟还原由 DSP2000 实验箱中 DAC1 (AD768) 来完成。TMS320F2812 中的 ADC 外设与 DSP 的通信可以通过查询方式或中断方式, 在此, 我们采用 ADC 的中断功能实现数据的交换。TMS320F2812 中 ADC 的转换频率和采样频率可以独立设置, 分别位于 ADC 外设模块和事件管理器模块中, 因此要使 ADC 工作, 必须掌握 ADC 外设和事件管理器外设中的相关设置。

由此可得程序流程如图 3.1 所示。

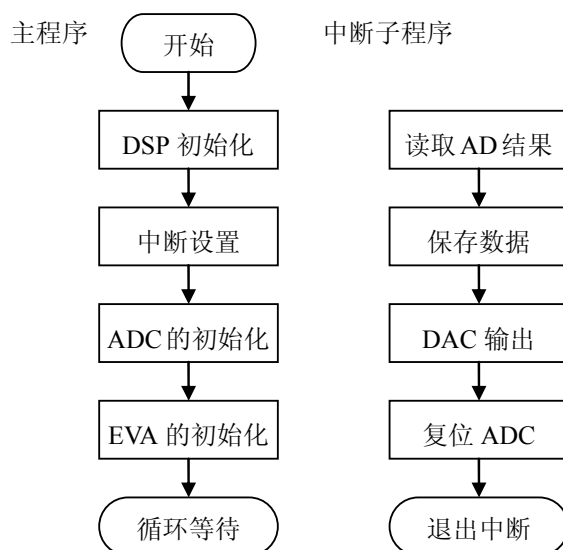


图 3.1 DSP 数据采集程序流程

实现程序可参照工程 AD_C（C 语言格式）或 AD_Asm（汇编语言格式）。

3.4.2 DSP 初始化

一般而言，DSP 要正常工作，必须首先设置时钟，时钟确定了 DSP 工作主频。
TMS320F2812 中时钟设置大致分为三个主要寄存器，它们分别是锁相环控制寄存器（PLLCR）、外设时钟使能控制寄存器（PCLKCR）和外设时钟预定标设置寄存器（HISPCP、LOSPCP）。

1. PLLCR 寄存器（地址@0x7021）

PLLCR 寄存器用于改变 PLL 的锁相环倍频值，输出 CLKIN 用于 DSP 内部的主频，控制 DSP 指令执行周期以及外设输入时钟。

15430			
保留		DIV	
DIV	CLKIN	DIV	CLKIN
0000	OSCCLK/2（PLL 旁路）	0001	(OSCCLK×1) /2
0010	(OSCCLK×2) /2	0011	(OSCCLK×3) /2
0100	(OSCCLK×4) /2	0101	(OSCCLK×5) /2
0110	(OSCCLK×6) /2	0111	(OSCCLK×7) /2
1000	(OSCCLK×8) /2	1001	(OSCCLK×9) /2
1010	(OSCCLK×10) /2	1011~1111	保留

图 3.2 PLLCR 寄存器说明

2. PCLKCR 寄存器（地址@0x701C）

外设时钟使能控制器用于控制片上各种外设时钟的工作状态，禁止或使能外设时钟能够有效降低系统外设。当该位置 1 时，相应的外设时钟被使能。

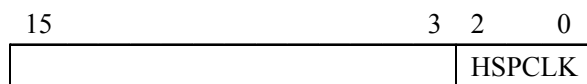
15	14	13	12	11	10	9	8
保留	ECANEN	保留	MCBSPEN	SCIBEN	SCIAEN	保留	SPIEN
7	4			3	2	2	1
保留				ADCEN		EVBEN	EVAEN

名称	说明	名称	说明
ECANEN	CAN 外设时钟使能	MCBSPEN	McBSP 外设时钟使能
SCIBEN	SCI-B 外设时钟使能	SCIAEN	SCI-A 外设时钟使能
SPIEN	SPI 外设时钟使能	ADCEN	ADC 外设时钟使能
EVBEN	EVB 外设时钟使能	EVAEN	EVA 外设时钟使能

图 3.3 时钟控制寄存器

3. 外设时钟预定标设置寄存器

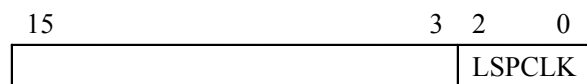
HISPCP（地址@0x701A）和 LOSPCP（地址@0x701B）寄存器分别用来配置高速和低速外设时钟。



HISPCP 寄存器

如果 HISPCP≠0, $HSPCLK = SYSCLKOUT / (HPSPLK(2:0) \times 2)$;

如果 HISPCP=0, $HSPCLK = SYSCLKOUT$ 。



LOSPCP 寄存器

如果 LOSPCP≠0, $LSPCLK = SYSCLKOUT / (LSPLK(2:0) \times 2)$;

如果 LOSPCP=0, $LSPCLK = SYSCLKOUT$ 。

图 3.4 外设时钟预定标设置寄存器

3.4.3 模数转换器 (ADC)

TMS320F2812 内部有一个 16 通道、采样精度为 12bit 的 ADC 模块，分别为事件管理器 A 和事件管理器 B 服务。这 16 通道可配置两个独立的 8 通道模块，具有同步采样和顺序采样模式，模拟输入范围 0~3V，最快转换时间为 80ns，具有多个触发源用于启动 AD 的转换，采用灵活的中断控制。

ADC 工作流程如图 3.5 所示。

ADC 模块的初始化包括设置外设的部件的上电、复位、时钟设定、触发源的设置、中断设置、运行模式的设置以及采样通道的设置。这些设置分别在 ADCTRL1、ADCTRL2、ADCTRL3 和 ADCMAXCONV 寄存器中。下面将会具体介绍。

当启动信号转换信号 SOC 达到后，ADC 启动，首先将 MAX CONV_n 数值自动加载到 SEQ CNTR_n 中，一次启动信号 ADC 转换的次数为 MAX CONV_n+1。转换按照预先设定的顺序进行（由 ADCCHCELSQ_n 确定），转换的结果依次写入到 ADCRESULT_n 寄存器中。

当所有转换完成后（即当 SEQ CNTR_n 值为 0 时），ADC 工作方式取决于 ADCTRL1 寄存器中连续运行模式位

（CONT RUN）。若该位为 1，则转换再次开始。因此，必须保证在下次转换完成之前读取 ADCRESULT_n 的数值。若该位为 0，则 SEQ CNTR_n 继续保持为 0 值，等待下一次启动触发信号的到达。

由于在 SEQ CNTR_n 每次到达 0 时，中断标识位都会被置 1。因此，可以在中

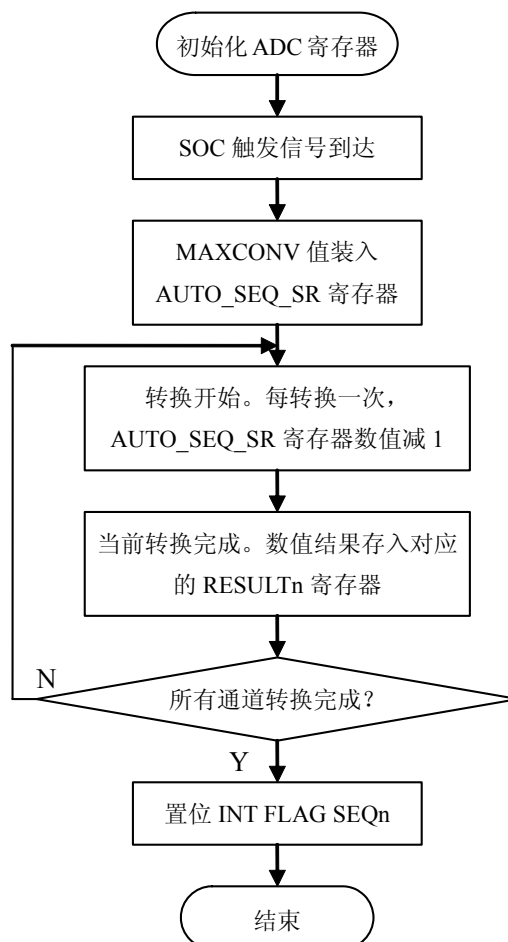


图 3.5 ADC 工作流程

断服务子程序中，复位 ADCTRL2 中的 RST SEQn 位，这将使得下次转换重新开始。

由于 TMS320F2812 的 ADC 为多通道 ADC，因此其保持时间、转换时间和采样间隔不同。采样保持时间和转换时间的时钟产生如下图 3.6 所示。

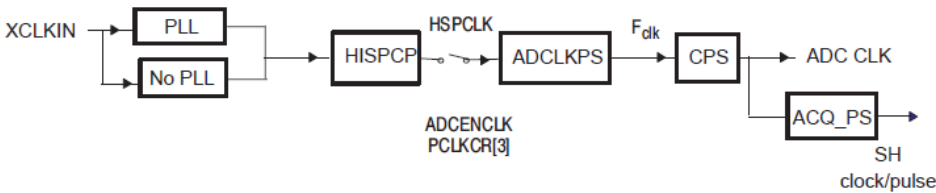


图 3.6 ADC 的时钟链路

ADC CLK 为转换时间，SH clock/pulse 为采样保持时间。图中各模块都是 ADC 相关寄存器中的设置位。

ADC 相关控制寄存器如下所述。

1. ADCTRL1（地址@0x7100）

15	14	13	12	11	10	9	8
保留	RESET	SUSMOD1	SUSMOD0	ACQ PS3	ACQ PS2	ACQ PS1	ACQ PS0
7	6	5	4	3	0		
CPS	CONTRUN	SEQ1OVRD	SEQCASC	保留			

图 3.7 ADCTRL1 寄存器

位	名称	说明
14	RESET	ADC 模块软件复位：写 1 复位
13~12	SUSMOD	仿真模式位
11~8	ACQ PS	采样保持窗口设置位：控制采样脉冲宽度为（ACQ_PS+1）个 ADCLK
7	CPS	核心时钟预分频器：分频系数为 2CPS
6	CONTRUN	运行模式位：1，连续转换模式；0，启动停止模式
5	SEQ1OVRD	排序器覆盖
4	SEQCASC	级联工作模式位：1，级联模式；0，双排序模式

2. ADCTRL2（地址@0x7101）

15	14	13	12	11	10	9	8
EVB SOC SEQ	RST SEQ1	SOC SEQ1	保留	INT ENA SEQ1	INT MOD SEQ1	保留	EVA SOC SEQ1
7	6	5	4	3	保留		0
EXT SOC SEQ1	RST SEQ2	SOC SEQ2	保留	INT ENA SEQ2	INT MOD SEQ2	保留	EVB SOC SEQ2

图 3.8 ADCTRL2 寄存器

位	名称	说明
15	EVB SOC	级联模式下 EVB SOC 使能信号：1，使能
14	RST SEQ1	排序器 1（SEQ1）复位：写 1 复位
13	SOC SEQ1	SEQ1 的启动触发信号（只读），为 1 时表明启动信号产生
11	INT ENA SEQ1	SEQ1 中断使能：1，使能中断；0，禁用中断
10	INT MOD SEQ1	SEQ1 中断模式：0，每个 SEQ1 结束产生中断；1，每隔一个 SEQ1 结束产生中断

8	EVA SOC SEQ1	EVA 的启动 SEQ1 屏蔽位：1，允许 EVA 触发 SEQ1；0，禁止
7	EXT SOC SEQ1	SEQ1 的外部启动转换信号
6	RST SEQ2	排序器 2（SEQ2）复位：写 1 复位
5	SOC SEQ2	SEQ2 的启动触发信号（只读），为 1 时表明启动信号产生
3	INT ENA SEQ2	SEQ2 中断使能：1，使能中断；0，禁用中断
2	INT MOD SEQ2	SEQ2 中断模式：0，每个 SEQ2 结束产生中断；1，每隔一个 SEQ2 结束产生中断
0	EVB SOC SEQ2	EVB 的启动 SEQ2 屏蔽位：1，允许 EVA 触发 SEQ2；0，禁止

3. ADCTRL3（地址@0x7118）

15	9	8
保留		
7	6	5
4	1	0
ADCBGRFDN[1:0]	ADCPWDN	ADCCLKPS[3:0]
SMODE SEL		

图 3.9 ADCTRL3 寄存器

位	名称	说明
8	EXTREF	ADCREFP 和 ADCREFM 引脚输入使能：1，输入；0 输出
7~6	ADCBGRFDN[1:0]	ADC 带隙参考源：11，参考电源开启；00，关闭
5	ADCPWDN	ADC 关闭：0，关闭所有模拟电源；1，开启
4~1	ADCCLKPS[3:0]	核心时钟分频：分频系数为 2 ^{ADCCLKPS[3:0]}
0	SMODE SEL	采样模式选择：1，同步采样；0，顺序采样

ADC 支持三个独立的供电电源，每一个可以通过 ADCTRL3 寄存器的独立位来控制。ADC 外设要正常工作，必须使 ADCBGRFDN[1:0]、ADCPWDN 这三位都置 1。

4. ADC 状态和标志寄存器 ADCST（地址@0x7119）

15							8	
保留								
7		6	5	4	3	2	1	0
EOS BUF2		EOS BUF1	INT SEQ2 CLR	INT SEQ1 CLR	SEQ2 BSY	SEQ1 BSY	INT SEQ2	INT SEQ1

图 3.10 ADC 状态和标志寄存器

位	名称	说明
7	EOS BUF2	SEQ2 的序列缓冲结束位
6	EOS BUF1	SEQ1 的序列缓冲结束位
5	INT SEQ2 CLR	SEQ2 中断清除位：写 1 清除 SEQ2 INT
4	INT SEQ1 CLR	SEQ1 中断清除位：写 1 清除 SEQ1 INT
3	SEQ2 BSY	SEQ2 忙标志位：1，忙；0，空闲
2	SEQ1 BSY	SEQ1 忙标志位：1，忙；0，空闲
1	INT SEQ2	SEQ2 中断标志位：1，发生 SEQ2 转换结束
0	INT SEQ1	SEQ1 中断标志位：1，发生 SEQ2 转换结束

4. 最大转换通道寄存器 ADCMAXCONV（地址@0x7102）



图 3.11 最大转换通道寄存器 ADCMAXCONV

MAX CONVn: 一次启动触发信号 ADC 的最大转换次数为 MAXCONVn+1。

5. ADC 输入通道选择控制寄存器 ADCCHSELSEQn

每一个 4 位域的数值 CONVnn，都可以为自动转换选择 16 个模拟输入通道中的一个，如下表 3.1 所示。

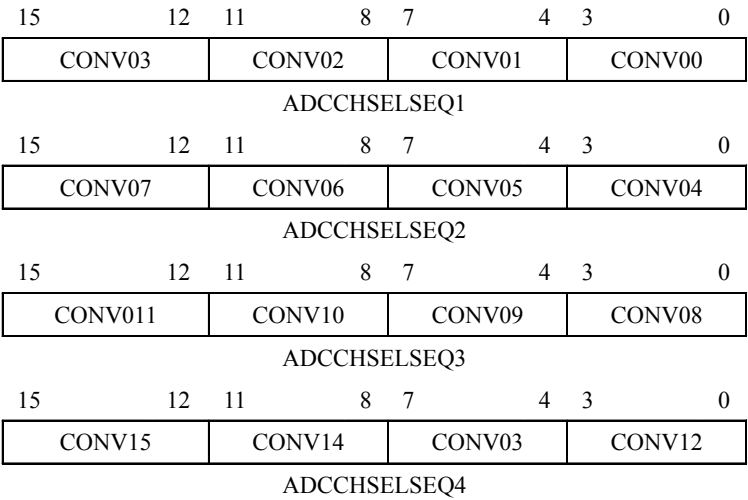


图 3.12 输入通道选择控制寄存器

表 3.1 CONVnn 数值与 ADC 输入通道关系

CONVnn 数值	ADC 输入通道	CONVnn 数值	ADC 输入通道
0000	ADCINA0	0000	ADCINB0
0001	ADCINA1	0001	ADCINB1
0010	ADCINA2	0010	ADCINB2
0011	ADCINA3	0011	ADCINB3
0100	ADCINA4	0100	ADCINB4
0101	ADCINA5	0101	ADCINB5
0110	ADCINA6	0110	ADCINB6
0111	ADCINA7	0111	ADCINB7

与 ADC 外设模块相关的寄存器及地址如下表 3.2 所示。

表 3.2 ADC 模块寄存器

名称	地址	说明
ADCTRL1	0x7100	ADC 控制寄存器 1
ADCTRL2	0x7101	ADC 控制寄存器 2
ADCMAXCONV	0x7102	ADC 最大转换通道寄存器
ADCCHSELSEQ1	0x7103	ADC 通道选择控制寄存器 1

ADCCHSELSEQ2	0x7104	ADC 通道选择控制寄存器 2
ADCCHSELSEQ3	0x7105	ADC 通道选择控制寄存器 3
ADCCHSELSEQ4	0x7106	ADC 通道选择控制寄存器 4
ADCASEQSR	0x7107	ADC 自动排序状态寄存器
ADCRESULT0	0x7108	ADC 转换结果缓冲寄存器 0
ADCRESULT1	0x7109	ADC 转换结果缓冲寄存器 1
ADCRESULT2	0x710A	ADC 转换结果缓冲寄存器 2
ADCRESULT3	0x710B	ADC 转换结果缓冲寄存器 3
ADCRESULT4	0x710C	ADC 转换结果缓冲寄存器 4
ADCRESULT5	0x710D	ADC 转换结果缓冲寄存器 5
ADCRESULT6	0x710E	ADC 转换结果缓冲寄存器 6
ADCRESULT7	0x710F	ADC 转换结果缓冲寄存器 7
ADCRESULT8	0x7110	ADC 转换结果缓冲寄存器 8
ADCRESULT9	0x7111	ADC 转换结果缓冲寄存器 9
ADCRESULT10	0x7112	ADC 转换结果缓冲寄存器 10
ADCRESULT11	0x7113	ADC 转换结果缓冲寄存器 11
ADCRESULT12	0x7114	ADC 转换结果缓冲寄存器 12
ADCRESULT13	0x7115	ADC 转换结果缓冲寄存器 13
ADCRESULT14	0x7116	ADC 转换结果缓冲寄存器 14
ADCRESULT15	0x7117	ADC 转换结果缓冲寄存器 15
ADCCTRL3	0x7118	ADC 控制寄存器 3
ADCST	0x7119	ADC 状态寄存器

ADC 模块具体工作原理以及设置可参见 TMS320x281x Analog-to-Digital Converter (ADC) Reference Guide 等相关资料。

3.4.3 事件管理器

TMS320F2812 片内集成了两个事件管理器 EVA 和 EVB, 他们具有完全相同的结构和功能, 内部包含通用定时器、全比较/PWM 单元、捕获单元以及这叫编码脉冲 (QEP) 电路。具体内容可参见相关资料, 在此只介绍与 ADC 相关的事件管理器中通用定时器。

每个事件管理器模块有两个通用定时器 (GP Timer), 在 GPTCONA/B 寄存器中可以定义 ADC 的启动触发信号由通用定时器的事件来产生, 比如这些事件可以是下溢、比较匹配或周期匹配, 下溢是指定时器计数器 (TxCNT) 的数值为 0, 比较匹配是指 TxCNT 与比较寄存器 (TxCMPR) 中的数值相等, 周期匹配是指 TxCNT 与周期寄存器 (TxPR) 中的数值相等。这一特性允许在没有 CPU 干涉的情况下, 实现通用定时器事件和模数转换启动操作的同步。

每个通用定时器有 4 种可选的操作模式:

- 停止/保持模式;
- 连续递增计数模式;
- 定向的增/减计数模式;
- 连续增/减计数模式。

定时器控制寄存器 TxCON 中相应的模式位决定了通用定时器的计数模式。定时器的使能位为 TxCON[6], 可以使能或禁止定时器的计数操作。当定时器被禁止时, 定时器的计数器操作将停止, 其预定标器被复位为 X/1。当定时器被使能时, 定时器将按照寄存器 TxCON 中的相应位 (TxCON[12~11]) 设定的计数模式并开始计数。

(1) 停止/保持模式

在这种模式下，通用定时器的操作将停止并保持其当前状态，定时器的计数器、比较输出和预定标计数器都保持不变。

(2) 连续递增计数模式

在这种模式下，通用定时器将按照已定标的输入时钟计数，直到定时器计数器的值和周期寄存器的值匹配为止。在发生匹配之后的下一个输入时钟的上升沿，计数器被复位为 0，并开始下一个计数周期。

定时器计数器与周期寄存器发生匹配后再过一个 CPU 时钟周期，周期中断标志将被置位。如果外设中断没有被屏蔽，将会产生一个外设中断请求。如果 GPTCONA/B 寄存器的相应位将定时器的周期中断定义为 ADC（模/数转换）启动信号，那么在周期中断标志被设置的同时，会向 ADC 模块发出一个 ADC 启动信号。

除了第一个计数周期外，定时器的周期时间为 $(TxPR)+1$ 个定标后的时钟输入周期。

(3) 定向的增/减计数模式

通用定时器在定向的增/减计数模式中，将根据 TDIRA/B 引脚的输入，对定标后的时钟进行递增或递减计数。当引脚 TDIRA/B 保持为高电平时，通用定时器进行递增计数，直到计数值等于周期寄存器的值（或 FFFFh，如果计数器初值大于周期寄存器的值）。当定时器的值等于周期寄存器的值（或 FFFFh）时，如果引脚 TDIRA/B 仍保持为高电平，定时器的计数器将复位为 0 并继续重新递增计数到周期寄存器的值。当引脚 TDIRA/B 保持为低电平时，通用定时器将递减计数直到计数值为 0。当定时器的值递减计数到 0 时，如果引脚 TDIRA/B 仍保持为低电平，那么定时器会重新将周期寄存器的值载入计数器，从而开始下一个递减计数周期。

在定向的增/减计数模式中，周期、下溢和上溢中断标志位，中断以及相关的操作都会根据各自的事件而产生，这与连续递增计数模式是一样的。

(4) 连续增/减计数模式

这种工作模式与定向的增/减计数模式一样，但是，在连续增/减计数模式下，引脚 TDIRA/B 的状态对计数的方向没有影响。定时器的计数方向仅在定时器的值达到周期寄存器的值时（或 FFFFh，如果定时器的初始值大于周期寄存器的值），才从递增计数变为递减计数。当计数器的值递减至 0 时，定时器又从递减计数变为递增计数。

在这种工作模式下，除了第一个周期外，定时器的周期都是 $2 \times (TxPR)$ 个定标的输入时钟周期。如果开始计数时，定时器计数器的初始值为 0，那么第一个计数周期的时间就与其他的周期一样。

在连续增/减计数模式下，周期、下溢和上溢中断标志位，中断以及相关的操作都根据各自的事件产生，这和连续递增计数模式一样。

在连续增/减计数模式下，定时器的计数方向由 GPTCONA/B 寄存器中的相应位确定：1 表示递增计数，0 表示递减计数。

在实验中，我们采用的是连续增计数模式，当发生周期匹配时，触发 ADC 的采样启动信号。

通用定时器中的寄存器及其地址如表 3.3 所示：

表 3.3 EVA/EVB 通用定时器相关寄存器

寄存器	地址	说明	寄存器	地址	说明
GPTCONA	0x7400	定时器全局控制寄存器 A	GPTCONA	0x7500	定时器全局控制寄存器 B
T1CNT	0x7401	定时器 1 计数寄存器	T3CNT	0x7501	定时器 3 计数寄存器
T1CMPR	0x7402	定时器 1 比较寄存器	T3CMPR	0x7502	定时器 3 比较寄存器
T1PR	0x7403	定时器 1 周期寄存器	T3PR	0x7503	定时器 3 周期寄存器

T1CON	0x7404	定时器 1 控制寄存器	T3CON	0x7504	定时器 3 控制寄存器
T1CNT	0x7405	定时器 2 计数寄存器	T4CNT	0x7505	定时器 4 计数寄存器
T1CMPR	0x7406	定时器 2 比较寄存器	T4CMPR	0x7506	定时器 4 比较寄存器
T1PR	0x7407	定时器 2 周期寄存器	T4PR	0x7507	定时器 4 周期寄存器
T1CON	0x7408	定时器 2 控制寄存器	T4CON	0x7508	定时器 4 控制寄存器

- 定时器计数器寄存器 TxCNT (x=1, 2, 3 或 4): 保存当前时刻定时器 x 的计数值, 16bit;
- 定时器的比较寄存器 TxCMPR (x=1, 2, 3 或 4): 存放定时器 x 的比较值, 16bit;
- 定时器的周期寄存器 TxPR (x=1, 2, 3 或 4): 存放定时器的周期值, 16bit;
- 定时器的控制寄存器 TxCON (x=1, 2, 3 或 4): 单个通用定时器的控制寄存器决定一个通用定时器的操作模式。

如下图 3.12 所示, 描述了 TxCON 寄存器各位的含义。

15	14	13	12	11	10	8
Free	Soft	保留	TMODE[1:0]	TPS[2:0]		
7	6	5	4	3	2	1
T2SWT1/ T4SWT3	TENABLE	TCLKS[1:0]	TCLD[1:0]	TECMPR	SELT1PR/ SELT3PR	

图 3.13 TxCON 寄存器 (x=1, 2, 3, 4)

位	名称	说明
15~14	Free Soft	仿真控制位: 可设置为 00
12~11	TMODE[1:0]	计数模式选择: 00, 停止保持; 01, 连续增/减计数模式; 10, 连续增计数模式; 11 定向的增/减计数模式
10~8	TPS[2:0]	输入时钟预分频因子: 分频系数为 2TPS[2:0]
7	T2SWT1/T4SWT3	为 1 时, 使用 T1CON 或 T3CON 的使能位来使能或禁止或定时操作
6	TENABLE	定时器使能位: 1, 允许定时器操作; 0, 禁止
5~4	TCLKS[1:0]	时钟源选择: 00, 内部时钟 (HSPCLK); 01, 外部时钟 (TCLKINx) 10, 保留; 11, 正交编码脉冲电路
3~2	TCLD[1:0]	定时器比较寄存器的重载条件: 00, 计数器为 0; 01, 计数器的值为 0 或等于周期寄存器的值; 10, 立即重载; 11, 保留
1	TECMPR	定时器比较使能: 1, 使能比较操作; 0, 禁止比较操作
0	SELT1PR/SELT3PR	为 1 时, 忽略自身的周期寄存器将 T1PR 或 T3PR 作为周期寄存器; 为 0 时, 使用自身的周期寄存器

- 通用定时器控制寄存器 GPTCONA/B: 全局通用定时器控制寄存器 (GPTCONA/B) 规定了发生各种定时事件时通用定时器所采用的动作及其计数方向。

15	14	13	12	11	10	9	8
保留	T2STAT	T1STAT	T2CTRIPE	T1CTRIPE	T2TOADC[1:0]	T1T0ADC[1]	
7	6	5	4	3	2	1	0
T1TOADC[0]	TCMPOE	T2CMPOE	T1CMPOE	T2PIN[1:0]	T1PIN[1:0]		
全局通用定时器控制寄存器 GPTCONA							
15	14	13	12	11	10	9	8
保留	T4STAT	T3STAT	T4CTRIPE	T3CTRIPE	T4TOADC[1:0]	T3T0ADC[1]	
7	6	5	4	3	2	1	0
T3TOADC[0]	TCMPOE	T4CMPOE	T3CMPOE	T4PIN[1:0]	T3PIN[1:0]		
全局通用定时器控制寄存器 GPTCONB							

图 3.14 全局通用定时器控制寄存器 GPTCONA/B

位	名称	说明
14	T2STAT	通用定时器 2 的状态位（只读）：1，递减计数；0 递增计数
13	T2STAT	通用定时器 1 的状态位（只读）：1，递减计数；0 递增计数
12	T2CTRIPE	定时器 2 输出切断功能使能位：1，使能；0，禁止
11	T1CTRIPE	定时器 1 输出切断功能使能位：1，使能；0，禁止
10~9	T2TOADC[1:0]	通用定时器 2 启动 ADC 方式：00，不启动；01：下溢启动 10：周期中毒启动；11：比较中断启动
8~7	T1TOADC[1:0]	通用定时器 1 启动 ADC 方式：00，不启动；01：下溢启动 10：周期中毒启动；11：比较中断启动
6	TCOMPOE	定时器比较输出使能
5	T2CMPOE	定时器 2 的比较输出使能
4	T1CMPOE	定时器 1 的比较输出使能
3~2	T2PIN[1:0]	定时器 2 比较输出极性：00，强制低；01，低有效； 10：高有效；11：强制高
1~0	T1PIN[1:0]	定时器 1 比较输出极性：00，强制低；01，低有效； 10：高有效；11：强制高

一般以 A 为后缀的寄存器中对应的事件管理 A 的控制位，以 B 为后缀的寄存器中对应的事件管理 B 的控制位，两者的布局往往是一致的，只不过各位对应的定时器不一样，如果在 GPTCONA 中用来控制定时器 1 的，在 GPTCONB 中相应的位就是用来控制定时器 3 的。其余依次类推。

EVA 和 EVB 模块具体工作原理以及设置可参见 TMS320x281x Event Manager (EV) Reference Guide 等相关资料。

3.4.4 TMS320F281x 中断系统.

TMS320F281x 的外设中断扩展 (PIE) 单元通过少量中断输入信号的复用来扩展大量的中断源，PIE 单元支持多达 96 个独立的中断，这些中断以 8 个为一组进行分类，每组中的所有中断共用一个 CPU 级中断 (INT1~INT12)。96 个中断对应的中断向量表存储在专用 RAM 区域中。PIE 向量表用来存储系统中每个中断服务程序 (ISR) 的入口地址。一般来说，在设备初始化时就要设置 PIE 向量表，并可在程序执行期间根据需要对其进行更新。

在实验中，当我们设置 VMAP=1 (ST1 寄存器的 bit3)，ENPIE=1 (PIECTRL 寄存器的 bit0) 后，TMS320F2812 的中断向量表地址范围 0x000D00~0x000DFF。例如外设模块 ADC 使用的中断 ADCINT 向量地址为 0x000D4A。

TMS320F281x 的中断分为三个级别：外设级，PIE 级和 CPU 级。每个片内外设的各个中断信号都具有自己的中断标志寄存器和中断使能寄存器，例如 ADC 中的 INT ENA SEQ1 位就是 ADC 的中断使能位，其标志为在 ADCST 中。PIE 单元将中断分为 12 组，每组 8 个中断。一旦片内外设想 PIE 发出中断请求，对应的外设中断标志寄存器 PIEIFRx.y 就会被置位，如果外设中断使能寄存器 PIEIERx.y 为 1（即被使能），当前外设中断应答寄存器 PIEACKx.y 为 0，PIE 就会向 CPU 级发出中断请求。CPU 级中断接受到 PIE 的中断，会立即置位 CPU 中断标志寄存器 IFR 相应位，并判断 CPU 中断使能寄存器 IER 中相应位是否被使能，以及全局中断 IMTN 是否被允许，若满足条件，则开始跳对应的中断向量表中的地址被加载到程序计数器 PC 中。

在退出中断服务程序时，为确保下次中断服务能够被可靠的执行，务必人工清除相应的中断标志位，其标志位也分为三个级别，分别是外设级中断应答标志，PIE 级中断应答标志 PIEACK 以及 CPU 级中断标志 IFR。

由此可见，要想正确使用中断，首先应该合理设置中断向量表，在对应地址填入中断服务子程序的入口地址。其次，必须对上述三个级别的中断作出正确的设置，以及在相应的中断向量表中填入适当的中断服务程序的入口地址。比如实验中，要想实现 CPU 利用中断方式读取 ADC 的采样数据，必须使能 ADC 外设的中断，其次使能外设使能寄存器 PIEIER1.6，保证中断发生时 PIEACK1.6 位清零，最后使能 CPU 中断使能寄存器 IER 中的 INT1，以及全局中断使能位 INTM。这些工作必须在系统初始化时完成。退出中断服务程序前，清除 ADCST 中的 INT SEQ1 以及相应的 PIEACKx。

PIE 配置控制寄存器如下表 3.4 所示。

表 3.4 PIE 相关寄存器

名称	地址	说明	名称	地址	说明
PIECTRL	0x0CE0	PIE 控制寄存器	PIEIFR6	0x0CED	PIEINT6 标志寄存器
PIEACK	0x0CE1	PIE 应答寄存器	PIEIER7	0x0CEE	PIEINT7 使能寄存器
PIEIER1	0x0CE2	PIEINT1 使能寄存器	PIEIFR7	0x0CEF	PIEINT7 标志寄存器
PIEIFR1	0x0CE3	PIEINT1 标志寄存器	PIEIER8	0x0CF0	PIEINT8 使能寄存器
PIEIER2	0x0CE4	PIEINT2 使能寄存器	PIEIFR8	0x0CF1	PIEINT8 标志寄存器
PIEIFR2	0x0CE5	PIEINT2 标志寄存器	PIEIER9	0x0CF2	PIEINT9 使能寄存器
PIEIER3	0x0CE6	PIEINT3 使能寄存器	PIEIFR9	0x0CF3	PIEINT9 标志寄存器
PIEIFR3	0x0CE7	PIEINT3 标志寄存器	PIEIER10	0x0CF4	PIEINT10 使能寄存器
PIEIER4	0x0CE8	PIEINT4 使能寄存器	PIEIFR10	0x0CF5	PIEINT10 标志寄存器
PIEIFR4	0x0CE9	PIEINT4 标志寄存器	PIEIER11	0x0CF6	PIEINT11 使能寄存器
PIEIER5	0x0CEA	PIEINT5 使能寄存器	PIEIFR11	0x0CF7	PIEINT11 标志寄存器
PIEIFR5	0x0CEB	PIEINT5 标志寄存器	PIEIER12	0x0CF8	PIEINT12 使能寄存器
PIEIER6	0x0CEC	PIEINT6 使能寄存器	PIEIFR12	0x0CF9	PIEINT12 标志寄存器

1. PIE 中断寄存器



图 3.15 PIE 中断寄存器

位	名称	说明
15~1	PIEVECT	提供了发生中断的向量地址（只读）
0	ENPIE	写 1 使能从 PIE 单元获取中断向量

2. PIE 中断应答寄存器 PIEACK

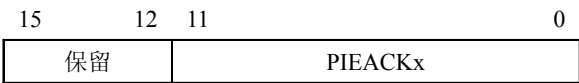


图 3.16 PIE 中断应答寄存器 PIEACK

位	名称	说明
11~0	PIEACKx	向相应的位写 1 可清除该 PIE 应答标志

3. PIE 中断标志寄存器

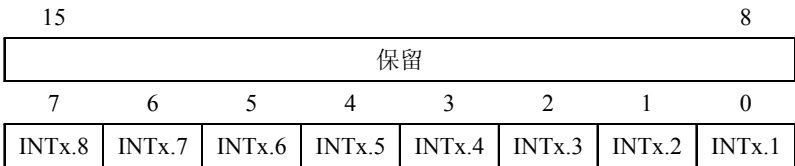


图 3.17 PIE 中断使能寄存器（x=1，…，12）

向相应的位写 1，就可以使能对应的外设中断；写 0 禁止对应外设中断。

4. CPU 级中断使能寄存器 IER

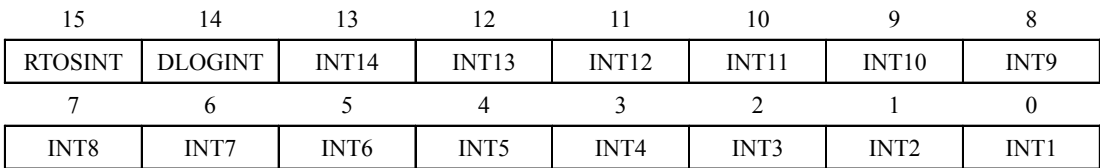


图 3.18 CPU 中断使能寄存器 IER

其中 RTOSINT 为实时操作系统中断使能位，DLOGINT 为数据日志中断使能位，INTx 为 CPU 中断，当位标志是 1 时，相应中断被使能。

关于 CPU 中断设置的详细内容参见 TMS320x281x System Control and Interrupts Reference Guide 等资料。

3.5 实验步骤

1. 设备检查

检查仿真器、C2000 DSP 实验箱、计算机之间的连接正确，打开计算机和实验箱电源。

2. 启动集成开发环境

点击桌面 CCS 2（C2000）快捷方式，进入集成开发环境 CCS。

3. 新建工程

新建一个 DSP 工程，编辑源程序、配置命令等相关文件，并在工程中添加这些程序文件。

在源程序中，通过对中断、ADC 外设以及事件管理通用时钟的设置，利用中断方式读取 ADC 的采样结果，并用 DAC 实现模拟信号的还原。在程序中，开辟一段数据空间，用于保存 ADC 的采样结果，要求保存 1024 点数据，且该空间的数据不断刷新。

源程序的编写可参照工程 AD_C（C 语言格式）或 AD_Asm（汇编语言格式）中的相关内容。

4. 建立工程 (Build)

建造工程 (build)，若出错，则根据错误提示，修改源程序文件或者配置命令文件，直至编译链接正确，生成可执行的.out 文件。

5. 加载程序

在主菜单下，选择“File → Load Program”，将程序下载到 DSP 内部。

6. 连接外部电路

打开信号源，产生一个合适的频率（ADC 的采样频率必须满足奈奎斯特采样定律），信号幅度控制在 $\pm 0.5V$ 以内，验证后将信号通过 INPUT1 接口输入到 DSP 中。

打开示波器，将 C2000 实验箱中的 OUT3 接口输出到示波器上，并正确设置。

7. 调试程序

首先验证中断设置是否正确。可以在 ADC 中断服务程序的入口地址处添加断点，全速或者动画运行程序，检查程序计数器 PC 能否间隔性的停留在中断服务入口地址处。若能，说明中断设置基本正确。

若以上步骤正确，其次，验证数据采集的正确性。程序连续运行一段时间后，暂停程序执行，打开图形显示功能，查看存储空间中保存的时域波形，是否为信号源输出的信号波形。

若上述步骤正确，则调节示波器，观察信号波形，是否为信号源的输入波形。若是，则实验调试结束。

以上步骤如果出错，则可以利用各种调试手段，比如打开寄存器窗口、变量窗口等辅助手段，根据数值以及实验原理，查找错误原因，重新修改程序，直至正确为止。

8. 运行程序

若第七步正确，可去掉断点，重新全速运行程序。

连接 C2000 实验箱 OUT3 输出口至示波器，调节示波器，观察信号的输出。可以实时的改变信号源的输入信号（注意信号幅度不要随意修改，超出输入范围易烧毁实验电路），示波器上显示的波形亦会随之变化。

数据直通通道就是最简单的实时信号处理电路。

3.6 实验要求

1. 独立完成项目编译、链接、调试的全过程。
2. 根据提示程序，给出 ADC 的采样频率计算公式，改写源程序中，修改 ADC 的采样频率。
3. 指出波形数据保存的空间地址，并以图形方式显示采集的信号波形，并保存，附在实验报告中。
4. 利用上述图形，给出验证采样频率的方法。以此验证数据采集程序的正确性。
5. 利用数码显示管，添加语句或者编写子程序，使之能够显示实验者的学号。

3.7 注意事项

1. 运行 CCS 集成开发软件后，务必确保 DSP 实验箱电源加载正常。
2. 信号源在连接实验箱前，务必保证信号幅度控制在 $\pm 0.5V$ 以内。当需改变信号时，可以更改信号波形以及频率。

3.8 实验思考

1. 观察输入信号与示波器显示信号、存储器中存储波形信号幅度的差异，解释差异产生的

原因。

2. 除了上述粗略验证 ADC 采样频率以外，思考其他测试采样频率的方法和手段。
3. 除了中断方式，DSP 内核还可以采用查询方式获取 ADC 外设的采样数据。如果采样查询方式，则需要查询哪些标志位。试图编程实现。

4. FIR 滤波器的 DSP 实现

4.1 实验目的

1. 巩固数字 FIR 滤波器的概念
2. 理解定点 DSP 中数的定标、有限字长、溢出等概念
3. 理解算法实现中实时的概念
4. 掌握 DSP 开发过程以及基本调试方法
5. 理解汇编以及高级语言开发 DSP、实现算法的区别

4.2 实验仪器

计算机, C2000 DSP 教学实验箱, XDS510 USB 仿真器, 示波器, 信号源

4.3 实验内容

针对 FIR 算法, 设计滤波器系数, 完成数据的定标, 查看滤波器特性曲线。

建立工程, 编写 DSP 的主程序, 并对工程进行编译、链接, 利用现有 DSP 平台实现 FIR 滤波器算法, 通过信号源、示波器理解滤波器特性, 验证实现与理论设计的一致性。

4.4 实验准备

4.4.1 实验流程

实验之前首先必须对 FIR 滤波器的设计、首先算法有所了解, 必要时通过计算机算法仿真, 理解 FIR 滤波器特性。由于计算机仿真属于浮点运算, 而 TMS320F2812 是定点 DSP, 因此需要针对所设计的 FIR 滤波器系数进行定标, 随后对定点后的数据再次进行仿真, 以验证定点实现的性能是否满足系统指标。

根据 FIR 滤波器算法, 编写 C 源程序或者汇编程序, 实现算法功能。并验证 DSP 实现时算法的正确性以及精度的要求。这种算法功能上的仿真可以利用 CCS 集成开发环境中数据 IO 输入来模拟信号的输入, 完成算法精度与功能的正确。

验证了算法的功能正确之后, 可以将程序下载到 DSP 上运行, 观察现象。更为重要的是, 在硬件平台上验证系统的实时性, 以及评估资源的使用的情况。若满足实时性要求, 则测试各项指标, 应该与原理设计相吻合。如果实现与理论不一致, 则首先检查算法的实时性, 以及资源使用是否冲突等原

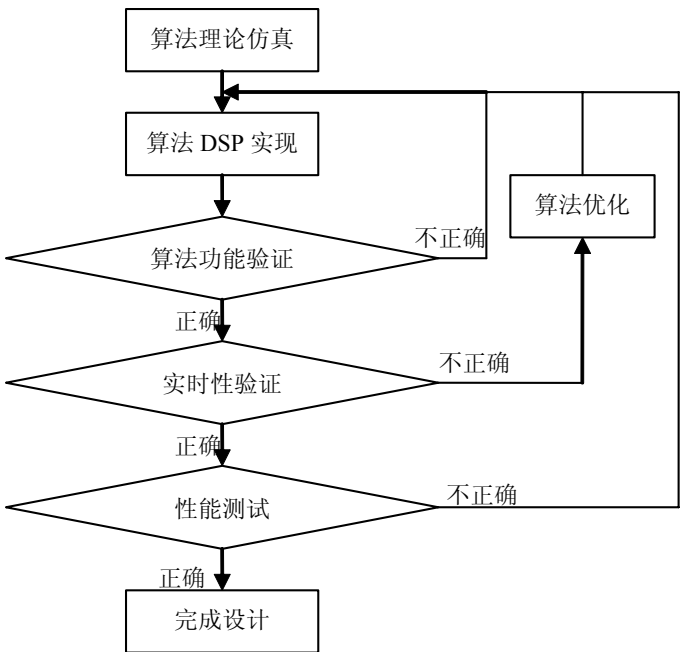


图 4.1 算法实现流程

因，对程序进行优化后再次编译链接，重新验证直至正确。算法的优化有时会贯穿与整个设计之中。

4.4.2 程序流程

FIR 滤波器算法属于典型的数据流处理方式，每到达一个新数据，就必须进行一次计算，更新输出。因此，当一次采样完成之后，就可以进行 FIR 核心算法，并将计算结果输出到 DA 中。

因此，和 DSP 的数据采集实验类似，用 DSP 实现实时的 FIR 信号处理算法必须依赖于 ADC、DSP 以及 DAC 三大基本部件。充分利用 DSP 片上 ADC 外设，实现模拟信号的采样，并由 DSP 完成 FIR 核心算法，由 DSP2000 实验箱中 DAC1（AD768）来完成数字到模拟的还原。在数据采集实验基础上，我们对程序流程稍加改动，就可实现完整数字 FIR 滤波器功能。程序流程如图 4.2 所示。

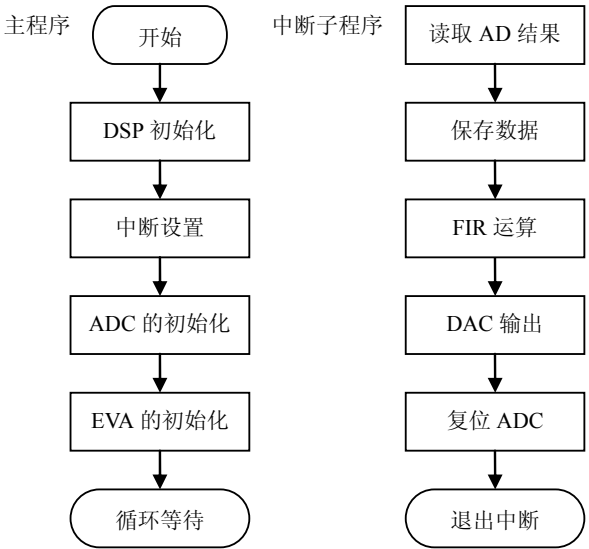


图 4.2 FIR 滤波器程序流程

4.4.3 FIR 滤波器设计

数字滤波器用于完成信号的滤波处理功能，是用有限精度算法实现的离散时间非时变系统。用 DSP 实现数字 FIR 滤波算法，具有稳定性强、精度高、实时性好、灵活性大、实现简单等优点。

有限长的单位冲击相应滤波器（FIR）差分方程可表示为：

$$y(n) = \sum_{k=0}^N h(k) \bullet x(n-k) \quad (4.1)$$

其中， h 是滤波器系数， x 为输入的数字信号， y 为 FIR 滤波器计算输出。 N 为滤波器阶数。由此可得，一个 N 阶的滤波器计算，需要 $N+1$ 个滤波器系数， $N+1$ 个数字输入，每得到一个 y 值，需要 $N+1$ 次乘法以及 N 次加法。另外， N 阶滤波器需要保存当前的 $N+1$ 个输入信号数值，以及事先设计的 $N+1$ 个滤波器系数。

滤波器系数的设计有很多方式，这里我们采样 MATLAB 软件来对 FIR 滤波器算法仿真并验证性能。

MATLAB 是新一代的科学计算软件，能够快速完成其他高级语言只有通过复杂编程才能实现的数值计算功能和图形显示，它可以用直观的数学表达式来描述问题，轻松的完成各项复杂数值计算、数据分析、符号计算等任务，可以解决线性代数、矩阵分析、微积分、微分方程、信号系统、信号分析与处理、系统控制等领域的算法建模分析。MATLAB 中有专

门的滤波器设计命令以及工具,在这里我们的重点不在于如何使用这个数学工具,只是使用,具体内容可参见 MATLAB 中滤波器设计方面的书籍。

在 MATLAB 界面中,利用 FIR1 命令来设计滤波器系数。fir1 的完整命令如下:

$$h = \text{fir1}(n, Wn, 'ftype', window) \quad (4.2)$$

其中, n 为滤波器阶数, Wn 为归一化截止频率(这里的归一化指与采样频率一半进行归一化), Wn 对应了在幅频曲线上-6dB 点的频率数值, $ftype$ 为滤波器类型,可以是低通、带通、高通、带阻等形式。 $window$ 是使用的窗函数,可以是 hamming、hanning、chebwin 等形式。 h 为产生的滤波器系数。详细说明可在 MATLAB 中输入 `help fir1` 或 `doc fir1` 查看。

对产生的滤波器系数可以用 FREQZ 命令查看幅频、相频特性曲线。具体命令如下:

$$\text{freqz}(h) \quad (4.3)$$

其中 h 为设计的滤波器系数。当然也可以使用 `fvtool(h)` 命令,验证滤波器设计是否满足系统指标要求,例如通带范围、阻带衰减、过渡带宽度等。

4.4.4 DSP 的算法实现

TMS320F2812 是定点型 DSP,存储器字长 16bit,可进行 32bit 的运算。而仿真计算中得到的数据大多是浮点型,因此将算法用定点 DSP 实现时,必须进行数据格式的定标。比如对 FIR 滤波器系数的定标可以参照实验二中介绍的方法来完成。对系数定标后,还要进行仿真以验证性能。

另外,由于 TMS320F2812 的数据字长只有 32bit, DAC1 接受的字长为 16bit,因此有限字长带来精度的损失。更为重要的是当加法的结果超过 16 位表示范围时,数据产生了溢出,这是有限字长带来的第二个问题。再者,数据的计算结果存放在 32bit 的寄存器中,但 DAC 却是 16bit,取高位输出还是低位输出,还是取一个合适的范围,这是数据截取的问题。因此在编写程序时,必须考虑定点数据的运算效应,由数据的动态范围来确定截取、定标等问题。定点 DSP 内部一般有溢出保护功能,可以查看溢出标志位及时发现溢出现象,其次用溢出模式位来使 ACC 结果控制在最大值范围之内,以达到防止溢出引起精度严重恶化的目的。

具体的实现编程,可以采用 C 语言或者汇编语言。C 语言描述算法较为简单,在此不作详细叙述。若用汇编语言实现,必须结合算法特点和汇编指令,充分利用片内多功能单元可以执行的特点,提高程序的执行效率,比如在 TMS320F2812 的汇编指令有 XMACD 指令,支持在单周期内完成数据的加、乘以及数据搬移功能,或者使用 DMAC 指令在单周期内实现双乘与双加运算。采用不同的指令,必须辅以不同的数据编排方式,因此需要综合考虑选取最优的实现方案。

4.4.5 算法实时性测试

算法的实时性测试主要指该算法能否在规定的时间内完成 FIR 运算,规定时间在此是指采样周期。FIR 的运算必须在两次采样间隔内完成,否则会造成数据的丢失。这是数据流处理的特点,数据的运算速度必须大于数据的更新速度。

在实验平台中,我们可以利用 GPIO 管脚来实测采样周期和算法执行时间。思想是添加程序,当程序进入断点时,将 GPIO 的某一个引脚输出置高,完成算法退出中断时再置低。由此当全速执行程序时,测量该 GPIO 引脚上的周期,便是采样周期,高电平持续时间即为 FIR 滤波器算法执行时间,由此判断计算法实现是否实时。

TMS320F2812 的通用数字输入输出引脚是复用的,即可以作为通用 IO 口使用,又可以作为外设引脚使用,这是由 GPxMUX 寄存器来控制切换。如果是数字 IO 模式,方向控制寄存器 GPxDIR 用来配置引脚信号的传输方向,另外 GPxDAT 寄存器用来反映当前对应引脚的电平,GPxSET 用来设置引脚的高电平(置高),GPxCLEAR 寄存器用来清除引脚的电平(清零),GPxTOGGLE 寄存器用来反相引脚电平(取反)。C2000 实验箱中可用的 GPIO

口为如《实验准备》中 C2000 DSP 教学实验箱介绍所述，下面以 GPAIO 为例简单介绍一下相关寄存器含义。

1. 通用 IO 口多路切换控制寄存器 GPAMUX（地址@0x70C0）

如果 GPAMUX.y 位=0，则对应引脚配置为通用 IO 口；

如果 GPAMUX.y 位=1，则对应引脚配置为专用功能外设口；

2. 通用 IO 口方向控制寄存器 GPADIR（地址@0x70C1）

如果 GPADIR.y 位=0，则对应通用 IO 口配置为输入；

如果 GPADIR.y 位=1，则对应通用 IO 口配置为输出；

3. 通用 IO 数据寄存器 GPADAT（地址@0x70E0）

这是一个可读可写的寄存器，读访问能返回当前输入 IO 引脚的信号电平状态，写操作能设置对应输出 IO 引脚的电平。

如果 GPADAT.y 位=0，则表示引脚为低电平；

如果 GPADAT.y 位=1，则表示引脚为高电平；

由于不同引脚读写的同时发生，引起冲突，所以一般不使用该寄存器改变输出引脚的电平，而使用下述 GPASET、GPACLEAR、GPATOGGLE 寄存器来解决。

4. 通用 IO 口置位寄存器 GPASET（地址@0x70E1）

这是一个只写寄存器，读操作总是返回 0。

如果 GPASET.y 位=0，则表示引脚没有变化；

如果 GPASET.y 位=1，则表示引脚电平置高（前提是该端口作为输出）；

5. 通用 IO 口清零寄存器 GPACLEAR（地址@0x70E2）

这是一个只写寄存器，读操作总是返回 0。

如果 GPACLEAR.y 位=0，则表示引脚没有变化；

如果 GPACLEAR.y 位=1，则表示引脚电平清零（前提是该端口作为输出）；

6. 通用 IO 口反相寄存器 GPATOGGLE（地址@0x70E3）

这是一个只写寄存器，读操作总是返回 0。

如果 GPATOGGLE.y 位=0，则表示引脚没有变化；

如果 GPATOGGLE.y 位=1，则表示引脚电平反转（前提是该端口作为输出）；

4.5 实验步骤

1. 系数设计

利用 MATLAB 设计滤波器系数，并对系数进行数据定标，完成浮点到定点的转换。分别作出两组系数的幅频、相频特性曲线，看是否满足设计要求。

2. 设备检查

检查仿真器、C2000 DSP 实验箱、计算机之间的连接正确，打开计算机和实验箱电源。

3. 启动集成开发环境

点击桌面 CCS 2（C2000）快捷方式，进入集成开发环境 CCS。

4. 建立工程

新建一个 DSP 工程，编辑源程序、配置命令等相关文件，并在工程中添加这些程序文件。

在上次实验程序的基础上加以修改，在中断程序内添加 FIR 算法模块，完成 FIR 算法程序。以及 GPIO 的输出控制相关设置，使之能够完成算法执行时间测量的工作。

建造工程（build），若出错，则根据错误提示，修改源程序文件或者配置命令文件，直至编译链接正确，生成可执行的.out 文件。

5. 加载程序

在主菜单下，选择“File → Load Program”，将程序下载到 DSP 内部。

6. 算法功能验证

在中断服务子程序恰当的地方，设置探针点，利用文件 IO 的方式，输入 x 数据（具体方法参见实验一）；在 FIR 算法结束处设置断点，验证 FIR 滤波器算法的正确性。输入的数据可具有一定的特殊性，进行特例验证。

若计算结果不正确，则进行程序的调试。调试关键在于现象重现、错误定位，可以利用各种调试手段，比如打开寄存器窗口、变量窗口等辅助手段，根据数值以及实验原理，查找错误原因，修改程序直至正确为止。

解决错误后，再重新恢复原错误程序，观察错误现象是否重现，以确定错误的唯一性。

7. 算法实时性验证

去除程序中的探针点以及断点重新全速运行程序，测量采样频率以及 FIR 算法的核心执行时间，判断该系统是否实时。

若非实时，则优化程序，甚至修改滤波器系数，直至满足实时要求。

8. 连接外部电路

打开信号源，产生一个合适的频率（ADC 的采样频率必须满足奈奎斯特采样定律）的正弦信号，信号幅度控制在 $\pm 0.5V$ 以内，验证后将信号通过 INPUT1 接口输入到 DSP 中。

打开示波器，将 C2000 实验箱中的 OUT3 接口输出到示波器上，并正确设置。

全速运行程序，调节信号源（正弦信号）的输出频率（正弦信号），记录各频点的示波器上输出幅度。描点作图，与理论幅频特性曲线比较，是否满足设计要求。

4.6 实验要求

1. 独立完成项目编译、链接、调试的全过程。
2. 当输入信号为正弦信号时，改变正弦信号频率，观察示波器，记录各频点对应的幅度，并描点作图，与实际理论幅频曲线比对，做误差分析。实际测量幅频曲线与理论曲线均需附在实验报告中，指出 FIR 滤波器系数的设计参数指标。
3. 记录 FIR 核心算法程序执行时间，以及采样时间，判断该系统是否实时。
4. 利用数码显示管，添加语句或者编写子程序，使之能够显示实验完成日期。

4.7 注意事项

1. 运行 CCS 集成开发软件后，务必确保 DSP 实验箱电源加载正常。
2. 信号源在连接实验箱前，务必保证信号幅度控制在 $\pm 0.5V$ 以内。当需改变信号时，可以更改信号波形以及频率。

4.8 实验思考

1. 观察各种输入信号通过数字滤波器系统之后的输出波形，解释信号失真原因。
2. 加载其他格式编写程序所产生的 FIR 滤波器程序，测量运算时间，比较分析 C 语言效率低的原因。
3. 以该 FIR 滤波器系统为例，总结分析系统实时性的取决因素。