14 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

14.1 ОБЗОР

В данной главе дается краткое описание процесса отладки программ для процессоров семейства ADSP-2100. Описание сопровождается примерами программ, которые должны дать читателю представление о том, как ему следует писать его собственные программы для процессоров рассматриваемого семейства.

Представленные в данной главе примеры программ используются во многих операциях цифровой обработки сигналов. Наиболее общими, и применимыми в различных условиях многофункциональными алгоритмами фильтров являются фильтр КИХ и последовательно включенный биквадратный фильтр БИХ. Умножение массивов используется для обработки изображений и в других областях, где требуются операции над векторами. Функция синуса часто требуется во множестве приложений. БПФ (быстрое преобразование Фурье) имеет широкое применение в технике анализа сигналов. Каждый из указанных примеров более подробно описывается в первом томе издания: *"Digital Signal Processing Applications Using ADSP-2100 Family"*. Эти примеры даются здесь для иллюстрации некоторых аспектов наиболее типичных программ для процессоров семейства ADSP-2100.

БПФ проиллюстрирован всей программой, которая состоит из подпрограммы, непосредственно выполняющей БПФ, и главной программы инициализации регистров и вызова подпрограммы БПФ, а также дополнительной программы.

Все другие примеры представлены как подпрограммы в их собственном модуле. Модуль начинается с директивы .MODULE, которая называет модуль, и заканчивается директивой .ENDMOD. Подпрограмма может вызываться из программы, содержащейся в другом модуле и объявляющей начальную метку подпрограммы в качестве внешнего символа. Каждая подпрограмма завершается командой RTS, которая передает управление обратно программе, из которой была вызвана данная подпрограмма. Обратите внимание, что текст программы сопровождается комментарием в фигурных скобках { }.

Перед каждым модулем идет блок комментариев, содержащий следующую информацию:

Параметры вызова	Значения регистров, которые должны быть заданы в главной программе перед вызовом подпрограммы.
Возвращаемые значения	Регистры, в которых содержатся результаты работы подпрограммы

Изменяемые регистры	Регистры, используемые подпрограммой. Вызывающая программа должна сохранить значения этих регистров перед вызовом подпрограммы и восстановить их после возвращения, если требуется сохранить значения этих регистров.
Время вычисления	Число командных циклов необходимых для выполнения подпрограммы.

14.2 ПРОЦЕСС ОТЛАДКИ СИСТЕМЫ

Процессоры семейства ADSP-2100 поддерживаются полным набором средств отладки. Средства программирования и симуляторы процессоров облегчают проектирование и отладку программного обеспечения. Встроенные эмуляторы и демонстрационные платы помогают моделировать аппаратное обеспечение.

Система программных средств отладки включает в себя несколько программ: построитель системы, ассемблер, редактор связей (линкер), подпрограмму разбиения главной программы на блоки (PROM сплиттер), программы-симуляторы и компилятор ***C*** с библиотекой рабочих программ. Все эти программы подробно описываются в следующих изданиях: *"ADSP-2100 Family Assembler Tools&Simulator Manual", "ADSP-2100 Family C Tools Manual", "ADSP-2100 Family C Runtime Library Manual".*

Блок схема процесса отладки системы показана на рис. 14.1.

Процесс отладки начинается с задачи описания аппаратного окружения для программных средств отладки. Здесь вы создаете файл описания системы используя редактор текстов. Этот файл содержит простые директивы, которые описывают адреса памяти и портов ввода/вывода, тип процессора и состояние вывода MMAP в проектируемой конфигурации аппаратных средств. Построитель системы считывает этот файл и генерирует файл описания архитектуры, который передает информацию редактору связей, симулятору и эмулятору.

Генерирование кода программы начинается с создания файлов исходного кода на языке С или ассемблере. Модуль является блоком команд на ассемблере, который образует главную программу, подпрограмму или объявление переменных данных. Программист на языке ***C*** записывает файлы на языке ***C*** и использует компилятор ***C*** для создания из этих файлов модулей кодов ассемблера. Программист на языке ассемблера непосредственно записывает модули кодов ассемблера. Каждый модуль кода программы транслируется ассемблером отдельно.

14 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

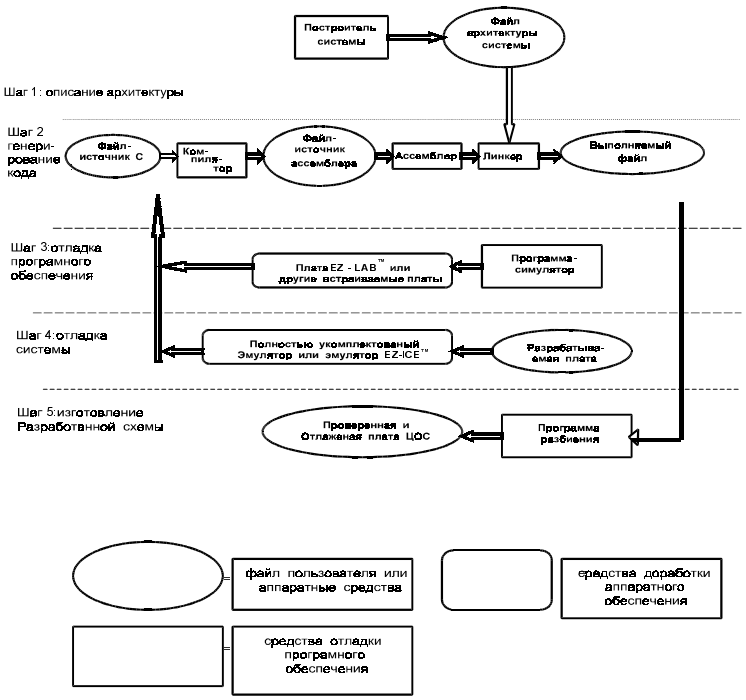

Рис. 14.1 Процесс отладки системы для процессоров семейства ADSP-2100

Редактор связей связывает вместе несколько модулей, формируя, таким образом, выполняемую программу (файл отображения в памяти). Редактор связей считывает информацию о разрабатываемой системе аппаратных средств из файла описания архитектуры и определяет на ее основе оптимальные адреса для кода программы и данных. В модулях программ на ассемблере каждый фрагмент кода/данных может быть задан как полностью перемещаемый, перемещаемый внутри определенного сегмента памяти или неперемещаемый (расположенный по абсолютному адресу). Редактор связей помещает неперемещаемые модули кода программы или данных по заданным адресам в памяти, при условии что область памяти имеет правильные атрибуты. Перемещаемые объекты помещаются по адресам, выбранным редактором связей. Редактор связей генерирует файл отображения памяти, содержащий одну выполняемую программу, которая может для ее проверки загружаться в симулятор или эмулятор.

Программа-симулятор выводит на дисплей окна, в которых можно видеть различные часть аппаратного окружения. Для дублирования проектируемых аппаратных средств симулятор конфигурирует свою память в соответствии с файлом

архитектуры, сгенерированным построителем системы, и моделирует отображенные в карте памяти порты ввода/вывода. Такое моделирование позволяет отладить аппаратные средства системы и проанализировать их рабочие характеристики перед тем, как приступить к их макетированию.

После окончания моделирования системы и ее программного обеспечения в ходе макетирования аппаратных средств может использоваться встроенный эмулятор, с помощью которого проверяются электрические схемы, временные характеристики и выполнение программы в реальном времени.

Программа разбиения (сплиттер) транслирует полученную редактором связей программу в файл стандартного формата для программатора ППЗУ. Как только код программы записан в ППЗУ и инсталлирован (с процессором семейства ADSP-2100) на макете, макет готов к работе.

14.3 ТРАНСВЕРСАЛЬНЫЙ КИХ ФИЛЬТР С ОДИНАРНОЙ ТОЧНОСТЬЮ

Структура трансверсального КИХ фильтра может быть получена непосредственно из уравнения дискретной свертки.

$$y(n) = \sum_{k=0}^{N-1} h_k(n)x(n-k)$$

В данном уравнении x(n) и y(n) представляют вход и выход фильтра в момент времени n. Выход y(n) формируется как взвешенная линейная комбинация текущего и предыдущего входных значений x, и x(n-k). Веса $h_k(n)$ являются коэффициентами трансверсального фильтра в момент времени n. В данном уравнении x(n-k) представляет собой предыдущее значение входного сигнала, "содержащегося" в (k+1)-ом отводе трансверсального фильтра. Например, пусть x(n), текущее значение входного сигнала, соответствует первому отводу, тогда x(n-42) соответствует сорок третьему отводу фильтра.

Ниже показана подпрограмма, которая реализует операцию суммирования произведений, используемую при вычислении отсчетов на выходе трансверсального фильтра.

```
.MODULE fir_sub;
{Подпрограмма трансверсального КИХ фильтра

Параметры вызова
```

```
        I0->Самое старое входное значение данных на
            линии задержки
        L0=Длина фильтра (N)
        I4->Начало таблицы коэффициентов фильтра
        L4=Длина фильтра (N)
        M1,M5=1
        CNTR=Длина фильтра - 1 (N-1)

Возвращаемые значения
        MR1=сумма произведений (округленная и насы-
            щенная)
        I0->Самое старое входное значение данных на
            линии задержки
        I4->Начало таблицы коэффициентов фильтра

Изменяемые регистры
        MX0, MY0, MR

Время вычисления
        N-1+5+2 цикла

Предполагается, что все коэффициенты и значения
данных даны в формате 1.15.}

.ENTRY fir;

fir:    MR=0, MX0=DM(I0,M1), MY0=PM(I4,M5);
        DO sop UNTIL CE;
sop:    MR=MR+MX0*MY0(SS), MX0=DM(I0,M1), MY0=PM(I4,M5);
        MR=MR+MX0*MY0(RND);
        IF MV SAT MR;
        RTS;
.ENDMOD;
```

14 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

14.4 ПОСЛЕДОВАТЕЛЬНО ВКЛЮЧЕННЫЙ БИКВАДРАТНЫЙ БИХ ФИЛЬТР

Биквадратный БИХ фильтр второго порядка представлен функцией z-преобразования.

$$H(z) = Y(z)/X(z) = (B_0 + B_1 z^{-1} + B_2 z^{-2})/(1 + A_1 z^{-1} + A_2 z^{-2})$$

где A_1, A_2, B_0, B_1 и B_2 являются коэффициентами, которые определяют требуемую импульсную характеристику системы H(z). Соответствующее разностное уравнение для биквадратной секции:

$$Y(n) = B_0 X(n) + B_1 X(n-1) + B_2 X(n-2) - A_1 Y(n-1) - A_2 Y(n-2)$$

За счет каскадирования (последовательного включения) нескольких биквадратных секций с соответствующими коэффициентами можно получить фильтры более высокого порядка. При этом биквадратные секции могут масштабироваться отдельно друг от друга и затем каскадируются для получения минимального квантования коэффициентов и минимальных накапливающихся ошибок.

Ниже показана подпрограмма фильтра высокого порядка. Отмасштабированные коэффициенты биквадратных секций содержатся в циклическом буфере в памяти программы в следующем порядке: B_2, B_1, B_0, A_2 и A_1 для каждой секции. Группы коэффициентов отдельных биквадратных секций должны храниться в порядке каскадирования этих секций.

```
.MODULE biquad_sub;
{Подпрограмма последовательно включенного
биквадратного фильтра N-го порядка

Параметры вызова:
        SR1=входное значение X(n)
        I0->Буфер линии задержки для X(n-2), X(n-1),
            Y(n-2), Y(n-1)
        L0=0
        I1->Масштабирующие коэффициенты для каждой
            биквадратной секции
        L1=0 (в случае одной биквадратной секции)
        L1=число биквадратных секций (если их
            несколько)
```

```
        I4->коэффициенты масштабированных
              биквадратных секций
        L4=5x[число биквадратных секций]
        M0,M4=1
        M1=-3
        M2=1 (в случае нескольких биквадратных
              секций)
        M2=0 (в случае одной биквадратной секции)
        M3= (1-длина буфера линии задержки)

Возвращаемые значения:
        SR1=выходная выборка Y(n)

Изменяемые регистры:
        SE, MX0, MX1, MY0, MR, SR

Время вычисления
        ADSP-2101/2102: (8xN/2)+5 циклов
        ADSP-2100/2100A: (8xN/2)+5+5 циклов

Предполагается, что все коэффициенты и значения
данных даны в формате 1.15.}

.ENTRY      biquad;

biquad:     CNTR=number_of_biquads;
                DO sections UNTIL CE;          {один цикл для каждой секции}
                SE=DM(I1,M2);                  {коэффициент масштабирования}
                MX0=DM(I0,M0), MY0=PM(I4,M4);
                MR=MX0*MY0(SS), MX1=DM(I0,M0); MY0=PM(I4,M4);
                MR=MR+MX1*MY0(SS), MY0=PM(I4,M4);
                MR=MR+SR1*MY0(SS), MX0=DM(I0,M0), MY0=PM(I4,M4);
                MR=MR+MX0*MY0(SS), MX0=DM(I0,M1), MY0=PM(I4,M4);
                DM(I0,M0)=MX1, MR=MR+MX0*MY0(RND);
sections:       DM(I0,M0)=SR1, SR=ASHIFT MR1 (HI);
        DM(I0,M0)=MX0;
        DM(I0,M3)=SR1;
        RTS;
.ENDMOD;
```

14 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

14.5 АППРОКСИМАЦИЯ СИНУСА

Следующая формула аппроксимирует синус входной переменной x:

$$\sin(x)=3{,}140625x + 0{,}02026367x^2 - 5{,}325196x^3 + 0{,}5446778x^4 + 1{,}800293x^5$$

Аппроксимация верна для любого значения x от 0° до 90° (первый квадрант). Однако, так как sin(-x)=-sin(x), а sin(x)=sin(180° - x), синус любого угла может быть выведен из синуса угла первого квадранта.

Ниже показана программа, которая выполняет аппроксимацию синуса с точностью до двух младших разрядов. Эта программа работает со входными значениями в формате 1.15. Коэффициенты, инициализированные в памяти данных в формате 4.12, подбираются так, чтобы отражать входные значения насколько позволяет этот формат. В данной шкале 180° равняется максимальному положительному значению, 0x7FFF, а -180° равняется максимальному (по абсолютной величине) отрицательному значению, 0x8000.

Приведенная ниже программа сперва находит угол первого квадранта эквивалентный входному значению. Синус модифицированного угла вычисляется при умножении возрастающих степеней угла на соответствующие коэффициенты. Затем результат может изменяться по знаку в соответствии со знаком синуса в квадранте, которому первоначально принадлежало введенное значение угла.

```
.MODULE Sin_Approximation;

{Аппроксимация синуса
        Y=Sin(x)

Параметры вызова
        AX0=x в масштабированном формате 1.15
        M3=1
        L3=0

Возвращаемые значения
        AR=y в формате 1.15

Изменяемые регистры
        AY0, AF, AR, MY1, MX1, MF, MR, SR, I3

Время вычисления
        25 циклов                }
```

14 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

```
.VAR/DM   sin_coeff(5);
.INIT     sin_coeff:0x3240,0x0053,0xAACC,0x08B7,0x1CCE;
.ENTRY    sin;
sin:      I3=^sin_coeff;            {указатель на буфер}
                                    {коэффициентов}
          AY0=0x4000;
          AR=AX0, AF=AX0 AND AY0;   {проверка 2-го и 4-го}
                                    {квадрантов}
          IF NE AR=-AX0;            {если да, то отрицание
                                    входного значения}
          AY0=0x7FFF;
          AR=AR AND AYO;            {удалить знаковый бит}
          MY1=AR;
          MF=AR*MY1(RND), MX1=DM(I3,M3);    {MF=x2}
          MR=MX1*MY1(SS), MX1=DM(I3,M3);    {MR=C1x}
          CNTR=3;
          DO approx UNTIL CE;
            MR=MR+MX1*MF(SS);
approx:     MF=AR*MF(RND), MX1=DM(I3,M3);
          MR=MR+MX1*MF(SS);
          SR=ASHIFT MR1 BY 3 (HI);
          SR=SR OR LSHIFT MR0 BY 3 (LO);    {Преобразование
                                            в формат 1.15}
          AR=PASS SR1;
          IF LT AR=PASS AY0;        {насыщение, если}
                                    {требуется}
          AF=PASS AX0;
          IF LT AR=-AR;             {Отрицание результата, если
                                    требуется}
          RTS;
.ENDMOD;
```

14 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

14.6 УМНОЖЕНИЕ МАССИВОВ С ОДИНАРНОЙ ТОЧНОСТЬЮ

В данном разделе представлена программа, которая перемножает два входных массива: X размерностью RxS (R строк, S столбцов), хранящийся в памяти данных, и Y разрядностью SxT (S строк, T столбцов), хранящийся в памяти программ. Выходной массив Z разрядностью RxT (R строк, T столбцов) записывается в память данных.

Для данной программы требуется инициализировать несколько регистров, что показано в разделе "Параметры вызова" в начальном комментарии. Регистр SE должен содержать значение, необходимое для сдвига результата каждого умножения в требуемый формат. Например, чтобы получить массив значений в формате 1.31 в результате перемножения двух массивов в формате 1.15, SE должен содержать значение ноль.

```
.MODULE matmul;
{Умножение массивов с одинарной точностью

        Z(i,j) = Σ (k=0 to S) [X(i,k)xY(k,j)]      i=0...R; j=0...T
        X - массив RxS
        Y - массив SxT
        Z - массив RxT

Параметры вызова
        I1-> буфер Z в памяти данных                    L1=0
        I2-> X, хранится по рядам в памяти данных       L2=0
        I6-> Y, хранится по рядам в памяти программы    L6=0
        M0=1        M1=S
        M4=1        M5=T
        L0, L4, L5=0
        SE=масштабирующее значение
        CNTR=R

Возвращаемые значения
        Буфер Z, заполненный строками

Изменяемые регистры
        I0, I1, I2, I4, I5, MR, MX0, MY0, SR
```

```
Время вычисления
      ((S + 8) x T + 4) x R + 2 + 2 цикла

.ENTRY      spmm;

spmm:       DO row_loop UNTIL CE;
            I5=I6;                          {I5 - начало Y}
            CNTR=M5;
            DO column_loop UNTIL CE;
            I0=I2;                          {I0 указывает на}
                                            {текущую строку X}
            I4=I5;                          {I4 указывает на}
                                            {текущий столбец Y}
            CNTR=M1;
            MR=0, MX0=DM(I0,M0), MY0=PM(I4,M5);
                                            {получение первых}
                                            {данных}
            Do element_loop UNTIL CE;
element_loop: MR=MR+MX0*MY0(SS), MX0=DM(I0,M0), MY0=PM(I4,M4);
            SR=ASHIFT MR1(HI), MY0=DM(I5,M4);     {обновле-}
                                                  {ние I5}
            SR=SR OR LSHIFT MR0(LO);      {окончание сдвига}
column_loop: DM(I1,M0)=SR;                {запись результата}
row_loop:   MODIFY(I2,M1);                {обновление I2, чтобы}
                                          {он указывал на следу-}
                                          {ющий ряд X}
            RTS;
.ENDMOD;
```

14.7 БЫСТРОЕ ПРЕОБРАЗОВАНИЕ ФУРЬЕ (БПФ) С ПРОРЕЖИВАНИЕМ ПО АЛГОРИТМУ RADIX-2

Программа БПФ состоит из трех подпрограмм. Первая из них скремблирует входные данные (помещает их по порядку их инвертированных адресов), для того чтобы выходные данные БПФ были в нормальном последовательном порядке.

Следующая подпрограмма вычисляет БПФ, а третья преобразует масштаб выходных данных, чтобы они соответствовали формату с блочной плавающей точкой.

Программа состоит из четырех модулей. В главном модуле объявляются и инициализируются буферы данных и вызываются подпрограммы. В остальных трех модулях содержатся подпрограммы БПФ, инвертирования бит, и приведения данных к формату с блочной плавающей точкой. Подпрограммы БПФ и инвертирования бит вызываются из главного модуля. Подпрограмма масштабирования данных вызывается модулем БПФ.

БПФ выполняется на месте, т.е. выходные значения записываются в тот же буфер, с которого считывались входные значения.

14.7.1 Главный модуль

Ниже приводится главный модуль dit_fft. N является числом точек БПФ (в данном примере N = 1024), а N_div_2 используется для определения длины буферов. Для изменения числа точек БПФ следует изменить значения этих констант и множителей (значения cos и sin).

Буферы данных twid_real и twid_imag в памяти программы содержат значения синуса и косинуса. Буферы inplacereal, inplaceimag, inputreal и inputimag в памяти данных содержат реальные и мнимые значения данных. Расположенные последовательно входные данные хранятся в inputreal и inputimag. Буфер с четырьмя ячейками, называемый padding, расположен в конце inplaceimag для того, чтобы обеспечить обращение к данным за границей этого буфера. Этот буфер помогает на этапе отладки , но не нужен в реальной системе. Последними объявляются переменные, называемые groups, bflys_per_groups, node_space и blk_exponent.

Реальные части (значения косинусов) коэффициентов хранятся в буфере twid_real. Этот буфер инициализируется из файла twid_real.dat. Подобным образом файл twid_imag.sat инициализирует буфер twid_imag, в котором содержатся мнимые части коэффициентов (значения синусов). В реальной системе, аппаратные средства должны быть установлены соответствующим образом для инициализации этих ячеек памяти.

Переменная groups получает первоначальное значение N-div_2, а переменные bflys_per_group и node_space получают первоначальное значение 2, так как на втором этапе БПФ для каждой группы требуется дважды выполнить алгоритм "бабочка". Во время масштабирования выходных данных это значение порядка обновляется.

14 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

После завершения инициализации вызываются две подпрограммы, первая из которых инвертирует биты входной последовательности значений, а вторая выполняет БПФ и вызывает подпрограмму приведения данных к формату с блочной плавающей точкой.

```
.MODULE/ABS=4              dit_fft_main;
.CONST                     N=1024, N_div_2=512; {для 1024
                           точек}
.VAR/PM/RAM/CIRC           twid_real [N_div_2];
.VAR/PM/RAM/CIRC           twid_imag [N_div_2];
.VAR/DM/RAM/ABS=0          inplacereal [N], inplaceimag [N],
                           padding [4];
.VARDM/RAM/ABS=H#1000      inputreal [N], inputimag [N];
.VAR/DM/RAM                groups, bflys_per_group, node_space,
                           blk_exponent;

.INIT   twid_real: <twid_real.dat>;
.INIT   twid_imag: <twid_imag.dat>;
.INIT   inputreal: <inputreal.dat>;
.INIT   inputimag: <inputimag.dat>;
.INIT   inplaceimag: <inputimag.dat>;
.INIT   groups: N_div_2;
.INIT   bflys_per_group: 2;
.INIT   node_space: 2;
.INIT   blk_exponent: 0;
.INIT   padding: 0,0,0,0;                  {нули после inplaceimag}

.GLOBAL  twid_real, twid_imag;
.GLOBAL  inplacereal, inplaceimag;
.GLOBAL  inputreal, inputimag;
.GLOBAL  groups, bflys_per_group, node_space, blk_exponent;

.EXTERNAL  scramble, fft_strt;

        CALL scramble;                     {вызов подпрограммы}
        CALL fft_strt;
        TRAP;                              {остановка программы}
.ENDMOD;
```

14.7.2 Подпрограмма БПФ с прореживанием

Подпрограмма БПФ с прореживанием по алгоритму radix-2 приводится ниже. Постоянные N и log2N являются числом точек и числом стадий БПФ, соответственно. Для изменения числа точек БПФ следует модифицировать значения этих постоянных.

Первая и последняя стадии БПФ выполняются за пределами цикла, в котором выполняются все остальные стадии. Отдельное выполнение первой и последней стадии обеспечивает выигрыш в быстроте. В первой стадии на группу выполняется только один алгоритм "бабочка", что делает цикл "бабочка" ненужным, коэффициенты равны 0 или 1, что исключает необходимость умножения. В последней стадии имеется только одна группа, поэтому цикл групп становится ненужным, как и операции установки для следующей стадии.

```
{БПФ с прореживанием по алгоритму radix-2 на 1024 точки}
{Преобразование к масштабу с блочной плавающей точкой}

.MODULE   fft;

{       Параметры вызова
            inplacereal = реальные входные данные в
скремблированном порядке
            inplaceimag= все нули (предполагается, что
входные             данные реальны)
            twid_real=значения косинуса
            twid_imag=значения синуса
            groups=N/2
            bflys_per_group=1
            node_space=1

        Возвращаемые значения
            inplacereal=реальные результаты БПФ по порядку
            inplaceimag=мнимые результаты БПФ по порядку

        Изменяемые регистры
            I0, I1, I2, I3, I4, I5, L0, L1, L2, L3, L4, L5
            M0,M1, M2, M3, M4, M5
```

```
            AX0, AX1, AY0, AY1, AR, AF
            MX0, MX1, MY0, MY1, MR, SB, SE, SR, SI

      Изменяемые области памяти
            inplacereal, inplaceimag, groups, node_space,
                  bflys_per_group, blk_exponent
}
.CONST      log2N=10, N=1024, nover2=512, nover4=256;

.EXTERNAL  twid_real, twid_imag;
.EXTERNAL  inplacereal, inplaceimag;
.EXTERNAL  groups, bflys_per_group, node_space;
.EXTERNAL  bfp_adj;
.ENTRY     fft_strt;

fft_strt:      CNTR=log2N - 2; {инициализация счетчика стадий}
               M0=0;
               M1=1;
               L1=0;
               L1=0;
               L3=0;
               L4=%twid_real;
               L5=%twid_imag;
               L6=0;
               SB=-2;

{-------- С Т А Д И Я  1 -------}
               I0=^inplacereal;
               I1=^inplacereal + 1;
               I2=^inplaceimag;
               I3=^inplaceimag + 1;
               M2=2;

               CNTR=nover2;
               AX0=DM(I0,M0);
               AY0=DM(I1,M0);
               AY1=DM(I3,M0);
```

```
                DO group_lp UNTIL CE;
                  AR=AX0+AY0, AX1=DM(I2,M0);
                  SB=EXPADJ AR, DM(I0,M2)=AR;
                  AR=AX0-AY0;
                  SB=EXPADJ AR;
                  DM(I1,M2)=AR, AR=AX1+AY1;
                  SB=EXPADJ AR, DM(I2,M2)=AR;
                  AR=AX1-AY1, AX0=DM(I0,M0);
                  SB=EXPADJ AR, DM(I3,M2)=AR;
                  AY0=DM(I1,M0);
group_lp:         AY1=DM(I3,M0);
                CALL bfp_adj;

{--------  С Т А Д И И  С  2  ДО  N - 1 -------}
-
          DO stage_loop UNTIL CE;      {вычисление всех стадий БПФ}
            I0=^inplacereal;           {I0->x0 в 1-ой группе стадии}
            I2=^inplaceimag;           {I2->y0 в 1-ой группе стадии}
            SI=DM(groups);
            SR=ASHIFT SI BY -1 (LO);    {groups/2}
            DM(groups)=SR0;            {groups=groups/2}
            CNTR=SR0;                  {CNTR=счетчик групп}
            M4=SR0;                    {M4=модификатор
                                       коэффициента с тильдой}
            M2=DM(node_space);         {M2=модификатор
                                       пересекающейся области}
            I1=I0;
            MODIFY(I1,M2);             {I1->y0 1-ой группы стадии}
            I3=I2;
            MODIFY(I3,M2);             {I3->y1 1-ой группы стадии}
          DO group_loop UNTIL CE;
            I4=^twid_real;             {I4->C от W0}
            I5=^twid_imag;             {I5->(-S) от W0}
            CNTR=DM(bflys_per_group);  CNTR=счетчик bfly
           MY0=PM(I4,M4), MX0=DM(I1,M0);      {MY)=C, MX)=x1}
          MY1=PM(I5,M4), MX1=DM(I3,M0);     {MY1=x1(-S), MX1=y1}
```

14 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

```
            DO bfly_loop UNTIL CE;
              MR=MX0*MY1(SS), AX0=DM(I0,M0);   {MR=x1(-S),}
                                               {AX0=x0}
              MR=MR+MX1*MY0(RND),AX1=DM(I2,M0); {MR=(y1(C)+x1(-S),}
                                               {AX1=y0}
              AY1=MR1, MR=MX0*MY0(SS);         {AY1=y1(C)+x1(-S)),}
                                               {AX1=y0}
              MR=MR-MX1*MY1(RND);                MR=x1(C)-y1(-S)}
              AY0=MR1, AR=AX1-AY1;             {AY0=x1(C)-y1(-S),}
                                               {AR=y0-(y1(C)+x1(-S))}
             SB=EXPADJ AR, DM(I3,M1)=AR;       {проверка увеличения бит,}
                                              {y1=y0-[y1(C)+x1(-S)]}
              AR=AX0-AY0, MX1=DM(I3,M0), MY1=PM(I5,M4);
                {AR=x0-[x1(C)-y1(-S)], MX1=следующее y1, MY1=следующее (-S)}
              SB=EXPADJ AR, DM(I1,M1)=AR;      {проверка увеличения бит,}
                                               {x1=x0-[x1(C)-y1(-S)]}
              AR=AX0+AY0, MX0=DM(I1,M0), MY0=PM(I4,M4);
                  {AR=x0+[x1(C)-y1(-S)], MX0=следующее x1, MY0=следующее C}
              SB=EXPADJ AR,DM(I0,M1)=AR;       {проверка увеличения бит,}
                                               {x0=x0+[x1(C)-y1(-S)]}
              AR=AX1+AY1;                      {AR=y0+[y1(C)+x1(-S)]}
bfly_loop:    SB=EXPADJ AR,DM(I2,M1)=AR;       {проверка увеличения бит,}
                                               {y0=y0+[y1(C)+x1(-S)]}
               MODIFY(I0,M2);                {I0->первое x0 в следующей группе}
               MODIFY(I1,M2);                {I1->первое x1 в следующей группе}
               MODIFY(I2,M2);                {I2->первое y0 в следующей группе}
group_loop:    MODIFY(I3,M2);                {I3->первое y1 в следующей группе}
               CALL bfp_adj;                 {компенсация увеличения бит}
               SI=DM(bflys_per_group);
               SR=ASHIFT SI BY 1(LO);
               DM(node_space)=SR0;           {node_space=node_space/2}
stage_loop:    DM(bflys_per_group)=SR0;      {bflys_per_group=bflys_per_group/2}

{----------    ПОСЛЕДНЯЯ СТАДИЯ-----------}
-
               I0=^inplacereal;
               I1=^inplacereal+nover2;
```

```
                I2=^inplaceimag;
                I3=^inplaceimag+nover2;

                CNTR=nover2;
                M2=DM(node_space);
                M4=1;
                I4=^twid_real;
                I5=^twid_imag;

                MY0=PM(I4,M4), MX0=DM(I1,M0);    {MY0=C,MX0=x1}
                MY1=PM(I5,M4), MX1=DM(I3,M0);    {MY1=-S, MX1=y1}
                DO bfly_lp UNTIL CE;
                  MR=MX0*MY1(SS), AX0=DM(I0,M0);    {MR=x1(-S), AX0=x0}
                  MR=MR+MX1*MY0(RND), AX1=DM(I2,M0);    {MR=(y1(C)+x1}
                                                        {(-S)), AX1=y0}
                   AY1=MR1, MR=MX0*MY0(SS);    {AY1=y1(C)+x1(-S), MR=x1(C)}
                MR=MR-MX1*MY1(RND);    {MR=x1(C)-y1(-S)}
               AY0=MR1, AR=AX1-AY1;   {AY0=x1(C)-y1(-S), AR=y0-[y1(C)+x1(-S)]}
                SB=EXPADJ AR,DM(I3,M1)=AR;    {проверка увеличения бит,}
                                              {y1=y0-[y1(C)+x1(-S)]}
                AR=AX0-AY0, MX1=DM(I3,M0), MY1=PM(I5,M4);
                   {AR=x0-[x1(C)-y1(-S)], MX1 = следующее y1, MY1 = следующее (-S)}
                SB=EXPADJ AR,DM(I1,M1)=AR;    {проверка увеличения бит,}
                                              {x1=x0-[x1(C)-y1(-S)]}
               AR=AX0+AY0, MX0=DM(I1,M0), MY0=PM(I4,M4);
                   {AR=x0+[x1(C)-y1(-S)], MX0 = следующее x1, MY0 = следующее C}
                SB=EXPADJ AR,DM(I0,M1)=AR;    {проверка увеличения бит,}
                                              {x0=x0+[x1(C)-y1(-S)]}
                AR=AX1+AY1;                {AR=y0+[y1(C)+x1(-S)]}
bfly_lp:        SB=EXPADJ AR, DM(I2,M1)=AR;    {проверка увеличения бит}
                CALL bfp_adj;
                RTS;
.ENDMOD;
```

14.7.3 Подпрограмма инвертирования бит

Подпрограмма инвертирования бит, или иначе, скремблирования, помещает входные данные в обратном порядке таким образом, что результаты выдаются последовательно. Эта программа использует возможности битреверсной адресации процессоров семейства ADSP-2100.

```
.MODULE dit_scramble;

{Параметры вызова
      Последовательно организованные входные данные в
буфере inputreal

Возвращаемые значения
      Скремблированные входные данные в буфере inplacereal

Изменяемые регистры
      I0, I4, M0, M4, AY1

Изменяемые области памяти
      inplacereal}

.CONST     N=1025,mod_value=H#0010  {Инициализация констант}
           ;
.EXTERNAL  inputreal, inplacereal;
.ENTRY     scramble;

scramble:  I4=^inputreal;          {I4->последовательно организован-}
                                   {ные данные}
           I0=^inplacereal;        {I0->скремблированные данные}
           M4=1;
           M0=mod_value;           {M0=модификатор для инвертирова-}
                                   {ния N бит}
           L4=0;
           L0=0;
           CNTR=N;
```

```
                ENA BIT_REV;          {Разрешение вывода результатов с }
                                      {инвертированным порядком бит }
                                      {для DAG1}
                DO brev UNTIL CE;
                                      {Чтение последовательно организо-}
                AY1=DM(I4,M4);        {ванных данных}
brev:                                 {Запись данных в ячейки в обрат-}
                DM(I0,M0)=AY1;        {ном порядке}
                DIS BIT_REV;          {Запрещение инвертирования бит}
                RTS;                {Возвращение в вызывающую программу}
.ENDMOD;
```

14.7.4 Подпрограмма приведения данных к масштабу с блочной плавающей точкой

Подпрограмма bfp_adj проверяет выходные данные БПФ на наличие увеличения бит и при необходимости масштабирует весь набор данных. Такая проверка предотвращает переполнение данных на каждой стадии БПФ. В показанной ниже подпрограмме используется такая характеристика устройства сдвига, как возможность нахождения порядка.

```
.MODULE  dit_radix_2_bfp_adjust;

{Параметры вызова
      Результаты БПФ с прореживанием по алгоритму RADIX-
2 в буферах inplacereal и inplaceimag

Возвращаемые параметры
      Буферы inplacereal и inplaceimag, приспособленные для
увеличения бит

Изменяемые регистры
      I0, I1, AX0, AY0, AR, MX0, MY0, MR, CNTR

Изменяемые области памяти
      inplacereal, inplaceimag, blk_exponent}
.CONST       Ntimes2=2048;
```

14 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

```
.EXTERNAL  inplacereal, blk_exponent;   {начало описания переменных}

.ENTRY     bfp_adj;

bfp_adj:   AY0=CNTR;                    {проверка, не является ли}
                                        {стадия последней}
           AR=AY0-1;
           IF EQ RTS;
             {если стадия последняя, возвращение в главную программу}
           AY0=-2;
           AX0=SB;
           AR=AX0-AY0;                  {проверка SB=-2}
           IF EQ RTS;
             {если SB=-2, увеличения бит нет, возвращение в}
           {главную программу}
           I0=^inplacereal;             {I0=указатель считывания}
           I1=^inplacereal;             {I1=указатель записи}
           AY0=-1;
           MY0=H#4000;                  {сдвиг MY0 на 1 бит вправо}
           AR=AX0-AY0,MX0=DM(I0,M1);    {проверка SB=-1; получение}
                                        {первой выборки}
           IF EQ JUMP strt_shift;     {если SB=-1, сдвиг блока данных }
                                      {на 1 бит}
           AX0=-2;      {установка AX0 на обновление блочного порядка}
           MY0=H#2000;                  {сдвиг MY0 на 2 бита вправо}
strt_shift: CNTR=Ntimes2 - 1;          {инициализация счетчика циклов}
           DO shift_loop UNTIL CE;      {сдвиг блока данных}
            MR=MX0*MY0(RND), MX0=DM(I0,M1);
              {MR=данные после сдвига, MX0=следующее значение}
shift_loop:   DM(I1,M1)=MR1;    {данные, не подвергнувшиеся
                                сдвигу=} {= данные после сдвига}
           MR=MX0*MY0(RND);   {сдвиг последнего слова данных}
           AY0=DM(blk_exponent);        {обновление блочного порядка и}
           DM(I1,M1)=MR1,AR=AY0-AX0;    {запись последней выборки,}
                                        {подвергнувшейся сдвигу}
           DM(blk_exponent)=AR;
           RTS;
.ENDMOD;
```