ФЕДЕРАЛЬНОЕ АГЕНТСТВО
ПО ОБРАЗОВАНИЮ

**Государственное образовательное учреждение
высшего профессионального образования**

**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ИНСТИТУТ
РАДИОТЕХНИКИ, ЭЛЕКТРОНИКИ И АВТОМАТИКИ
(ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ)**

Факультет электроники

Методические указания
по выполнению лабораторных работ

ОСНОВЫ ПРОЕКТИРОВАНИЯ СХЕМ С ЦИФРОВЫМИ СИГНАЛЬНЫМИ ПРОЦЕССОРАМИ

Для студентов специальностей

072000, 190400, 190700,
200100, 202100, 200300

МОСКВА

2006

Составители: С.Т. Давитадзе, *к.ф.-м.н., ст. преподаватель физического ф-та МГУ им. М.В. Ломоносова*

Е.Ф. Певцов, *к.т.н., доцент каф. ФКС МИРЭА*

Редактор Е.Ф. Певцов

Учебно-методические указания для выполнения лабораторных работ по основам проектирования систем с сигнальными процессорами предназначены для студентов, обучающихся по направлению 654300 и специальностям 072000, 190400, 190700, 200100, 202100, 200300, специализирующихся в области проектирования и применения систем цифровой обработки данных, а также студентов других специальностей, изучающих общие дисциплины по вычислительной технике.

Печатаются по решению редакционно-издательского совета Университета.

2006

Введение

Интенсивный рост применений цифровой обработки сигналов (ЦОС) в самых разных сферах науки, техники и информационных технологий обусловливает необходимость владения базовыми знаниями в этой области все большего количества инженеров и исследователей. Выполнение двух лабораторных работ, предлагаемых в данном учебном пособии, помогает приобрести начальные навыки работы с интегрированной средой разработки систем на основе цифровых сигнальных процессоров (ЦСП) VisualDSP++. На учебной макетной плате предлагается самостоятельно разработать программу ввода-вывода данных в схеме с ЦСП ADSP 218, а также изучить пример реализации цифрового КИХ-фильтра и исследовать его характеристики.

1. Основные сведения по ЦОС

Основными направлениями ЦОС являются: спектральный анализ, линейная фильтрация, частотно-временной анализ, адаптивная фильтрация, нелинейная обработка и многоскоростная обработка. Математической основой ЦОС служит аппарат, согласно которому каждый сигнал рассматриваются как функция, имеющая эквивалентное представление в виде разложения по ортонормированной базису (представление сигналов обобщенными рядами Фурье). Обычно в качестве базисных функций принимаются гармонические сигналы с частотами, кратными периоду исследуемого сигнала. Соответствующий набор частот определяет спектр сигнала. Энергия сигнала равна сумме квадратов компонент обобщенного ряда Фурье, и можно показать, что такое представление дает минимальную ошибку аппроксимации (см. стр. 35 [1]).

Фундаментальной основой теории обработки сигналов служит теорема Котельникова (теорема Найквиста, теорема отсчетов), которая доказывает, что если f_m – самая высокая частота в спектре функции $x(t)$, то функция полностью определяется последовательностью своих значений, если выборка осуществляется с частотой не меньшей, чем $2f_m$. (см. стр. 143 [1], стр. 67 [2]).

Принципиальной особенностью цифровой обработки сигналов является их дискретизация по времени и по уровню. Процесс дискретизации по времени можно рассматривать как умножение

аналогового сигнала *x(t)* на периодическую последовательность дельта-функций, повторяющихся с периодом выборки $1/F_д$. Как показано на рис. 1, функция спектра такого дискретизированного сигнала представляет собой периодически повторяющийся спектр исходного аналогового сигнала, причем период повторения составляет $1/F_д$. Компоненты частот с центрами в точках, кратных $F_д$, называют зеркальными частотами (подробно с вопросами обработки дискретных сигналов можно ознакомиться, например, в главе 15 учебника [1]).

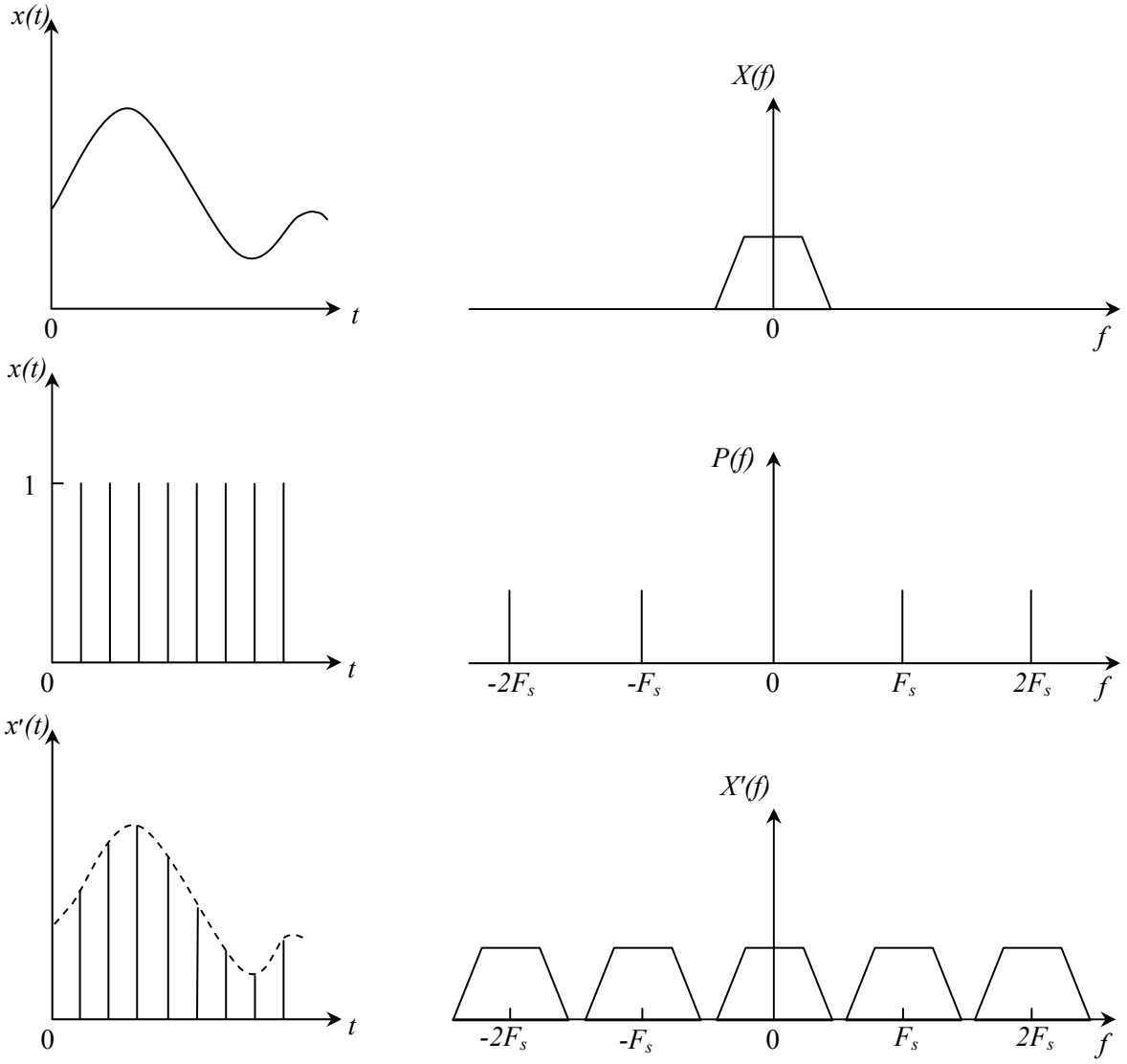

Рис. 1. Частотный спектр дискретизированного сигнала.

Из рис. 1. видно, что если половина частоты дискретизации (это значение называют частотой Найквиста) меньше максимальной частоты сигнала, то зеркальные частоты будут накладываться

на частоты основной полосы. Это приведет к тому, что спектр сигнала будет искажен и по нему будет невозможно восстановить исходный непрерывный сигнал (эффект наложения или перекрытия спектров).

Обобщение представления для реально обрабатываемых сигналов, ограниченных во времени, и потому не являющихся периодическими, приводит к тому, что набор базисных частот (спектр) становиться бесконечным. Это является непреодолимым препятствием для дискретизации и восстановления сигнала без искажений. Однако физическая природа любого измеряемого сигнала (соотношение неопределенности) обуславливает тот факт, что в его спектре есть такие высшие составляющие, энергия которых, начиная с некоторой верхней частоты $f_в$ становится малой настолько, что всеми последующими высшими составляющими в рамках решаемой конкретной задачи можно пренебречь. Типовая схема цифровой обработки аналоговых сигналов приведена на рис. 2 (см. подробнее стр. 72–134 [2] и стр. 5–11 [3]).

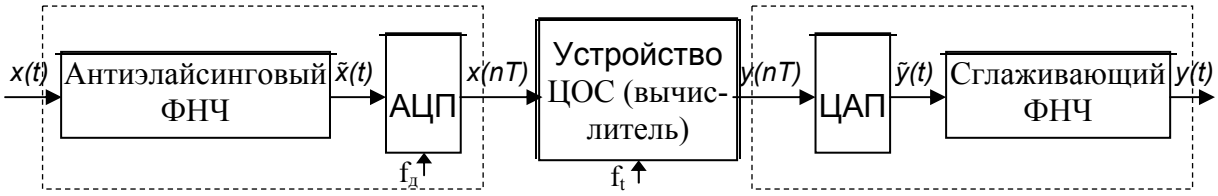

Рис. 2. Обобщенная схема устройства ЦОС.

Для устранения эффекта наложения частот необходимо ограничивать верхнюю частоту работы устройства обработки частотой F, определяемой частотой дискретизации с тем, чтобы одновременно выполнялись условия $F=f_в$ и $f_д \geq 2F$. Ограничение спектра сверху частотой реализуется с помощью обязательного в системе ЦОС аналогового фильтра низких частот (ФНЧ), называемого еще антиэлайсинговым (предотвращающим наложение частот, от англ. *aliasing*).

Следующий за ФНЧ аналого-цифровой преобразователь (АЦП) формирует последовательности чисел, описывающих отсчеты сигналов $x(nT)$, взятые через равные промежутки времени $T=1/F$. Ясно, что для наилучшего представления сигнала следует стремиться к увеличению частоты дискретизации, но это требование приходит в противоречие с ресурсами системы, поскольку

обработка сигналов в реальном времени, требует выполнения преобразований и вычислений в темпе поступления отсчетов.

Дискретизация сигналов по уровню является следствием формирования цифровых отсчетов, поскольку весь диапазон изменения сигналов разбивается на счетное количество дискретных уровней, определяемое разрядностью преобразования (обычно 8–20 разрядов).

Последовательность $x(nT)=x(n)$ поступает на вход вычислителя, который по заданному алгоритму формирует последовательность выходных отсчетов $y(n)$. Количество операций обработки (умножений, сложений, пересылок и т.п.) для получения может исчисляться тысячами, поэтому вычислитель должен работать с частотой f_t, синхронизирующей такты операций вычислений, значительно превышающей частоту дискретизации $f_д$. Полученные выходные отсчеты подаются на цифро-аналоговый преобразователь, формирующий ступенчатый аналоговый сигнал $\hat{y}(t)$, который затем с помощью сглаживающего ФНЧ преобразуется в выходной аналоговый сигнал $y(t)$.

Таким образом, предметом ЦОС являются последовательности чисел, представляющие собой N равноотстоящих отсчетов $x(n)$, для которых по формулам дискретного преобразования Фурье (ДПФ) ставятся во взаимнооднозначное соответствие значения комплексных составляющих спектра $X(k)$:

прямое ДПФ: $$X(k)=\frac{1}{N}\sum_{n=0}^{N-1}x(n)W_N^{nk},\ 0\le k\le N-1, \qquad (1)$$

где W_N^{nk} – ядро преобразования (поворачивающий множитель):

$$W_N^{nk}=e^{-j\frac{2\pi}{N}nk}; \qquad (2)$$

Составляющие спектра $X(k)$ имеют период N и располагаются по частотной оси с интервалом, называемым частотой преобразования:

$$\Delta\omega=2\pi/NT, \qquad T=1/f_д \qquad (3)$$

Выходной спектр ДПФ $X(k)$ является результатом вычисления свертки между выборкой, состоящей из входных отсчетов во временной области и набором из N пар гармонических базисных

функций. Можно показать, также, что запись (1) означает вычисление скалярного произведения вектора *x(n)* и вектора ортонормированного базиса в *N*-мерном пространстве (преобразование Фурье означает просто разложение исходной функции по ортонормированному базису). Поскольку представления сигналов во как функции времени и функции частоты эквивалентны, то сигнал также полностью определяет обратное ДПФ (ОДПФ):

$$x(n) = \sum_{k=0}^{N-1} X(k) W_N^{-nk},\ 0 \le n \le N-1. \qquad (4)$$

Прямое вычисление ДПФ и ОДПФ для больших *N* неэффективно, т.к. требует порядка N^2 операций умножений и сложений комплексных чисел. Для уменьшения объема вычислений используются свойства периодичности ядра преобразования W_N^{nk}. Идея алгоритма быстрого преобразования Фурье (БПФ) состоит в том, чтобы разделить *N*-точечную (*N* выбирается кратным степеням 2) последовательность на две, из ДПФ которых можно получить ДПФ исходной последовательности и продолжать такое разделение до момента, когда останется только два элемента последовательности.

2. Применение ЦСП для реализации цифровых фильтров

Цифровая фильтрация является одним из наиболее мощных инструментальных средств ЦОС. Очевидны преимущества устранения ошибок в фильтре, связанных с флуктуациями параметров пассивных компонентов во времени и по температуре, дрейфом ОУ (в активных фильтрах) и т.д. Характеристики цифрового фильтра могут быть легко изменены программно. По ряду параметров цифровые фильтры способны удовлетворять таким требованиям, которых невозможно достичь в аналоговом исполнении. Поэтому они широко используются в телекоммуникациях, в приложениях адаптивной фильтрации, таких как подавление эха в модемах, подавление шума и распознавание речи.

Существует два основных типа цифровых фильтров: фильтры с конечной импульсной характеристикой (КИХ) и фильтры с бесконечной импульсной характеристикой (БИХ). Высокоэффективные КИХ-фильтры (FIR-фильтры) строятся с большим числом операций умножения с накоплением и поэтому тре-

буют использования быстрых и эффективных процессоров ЦСП. Благодаря использованию обратной связи, БИХ-фильтры могут быть реализованы с меньшим количеством коэффициентов, чем КИХ-фильтры.

Схема КИХ-фильтра представлена на рис. 3 (см. гл. 6. [5]). Идея построения КИХ-фильтра основана на методе устранения высокочастотных шумов методом скользящего среднего [5, 6], в котором для определения текущего значения отфильтрованного сигнала используются предыдущие значения сигнала с разными весовыми коэффициентами.

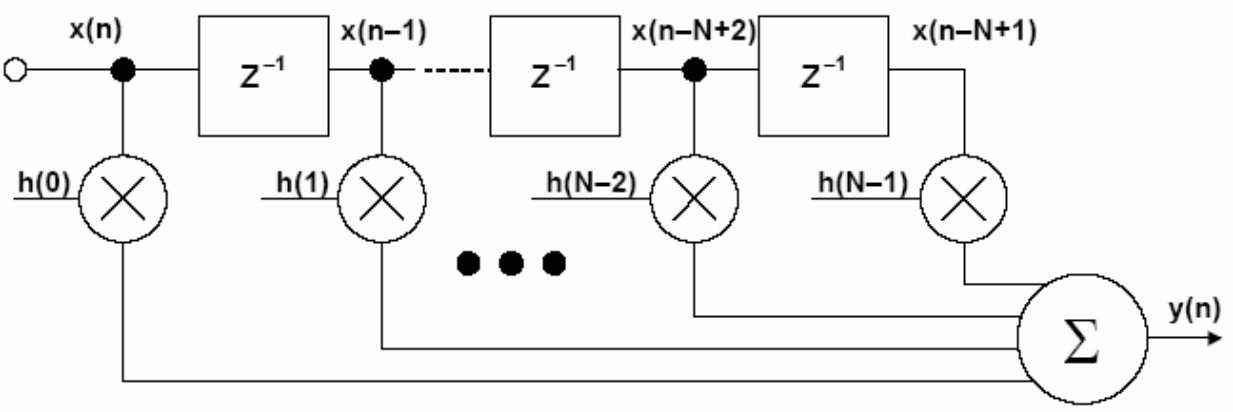

Рис. 3. Фильтр с конечной импульсной характеристикой порядка N. Через Z^{-1} обозначены элементы задержки.

Пусть имеется N звеньев, $x(n–k)$ – входной массив данных ($k=0,1,..N–1$), $h(k)$ – массив коэффициентов, КИХ-фильтр осуществляет взвешивание и суммирование входных отчетов в частотной области. Согласно известной теореме, произведение в частотной области можно заменить операцией свертки во временной и наоборот:

$$Y(n) = \sum_{k=0}^{N-1} h(k)x(n-k) = h(n) * x(n) \qquad (5)$$

Основой проектирования КИХ-фильтров является утверждение, что его коэффициенты $h(n)$ являются квантованными значениями импульсной характеристики этого фильтра. Сущность разработки КИХ-фильтра сводится к выбору соответствующих коэффициентов и необходимого числа звеньев для формирования желаемой частотной характеристики фильтра $H(f)$. Для этого имеются различные алгоритмы и программные пакеты, разработанные для современных ПК и доступные на рынке.

Искомая импульсная характеристика ЦФ является дискретным преобразованием Фурье требуемой *H(f)*.

Схема, реализующая, КИХ-фильтр с 4-мя звеньями приведена на рис. 4. В общем случае в рядах, задаваемых уравнениями КИХ-фильтров, предполагается последовательное обращение к *N* коэффициентам от *h(0)* до *h(N–1)*. Соответствующие точки данных циркулируют в памяти. При этом добавляются новые отсчеты данных, заменяя самые старые, и каждый раз производится вычисление выходного значения фильтра. Как показано на рис. 4, для реализации такого циклического буфера может использоваться фиксированный объем оперативной памяти. Выборка из последних отсчетов данных всегда сохраняется в оперативной памяти.

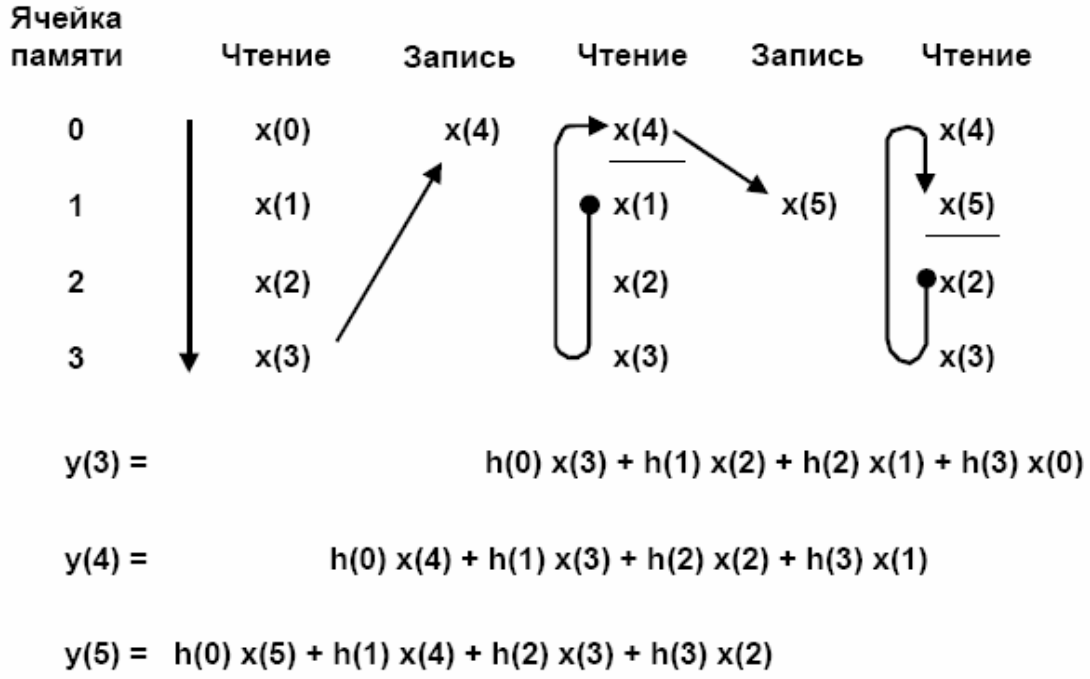

Рис. 4. Вычисление выходного сигнала КИХ-фильтра 4-го порядка с организацией циклического буфера памяти.

Выборка коэффициентов из памяти осуществляется одновременно с выборкой данных. В соответствии с описанной схемой адресации, самый старый отсчет данных выбирается первым. Поэтому сначала должна осуществляться выборка из памяти последнего коэффициента. При использовании адресного генератора, поддерживающего инкрементную адресацию, коэффициенты могут быть сохранены в памяти в обратном порядке: *h(N–1)* помещается в первую ячейку, а *h(0)* – в последнюю. И наоборот,

коэффициенты могут быть сохранены в порядке возрастания их номеров, если использовать адресный генератор, поддерживающий декрементную адресацию. В примере, показанном на рис. 4, коэффициенты сохранены в обратном порядке.

Алгоритм, реализующий КИХ-фильтр в ЦСП представлен на рис. 5. Один цикл фильтрации выделен стрелкой.

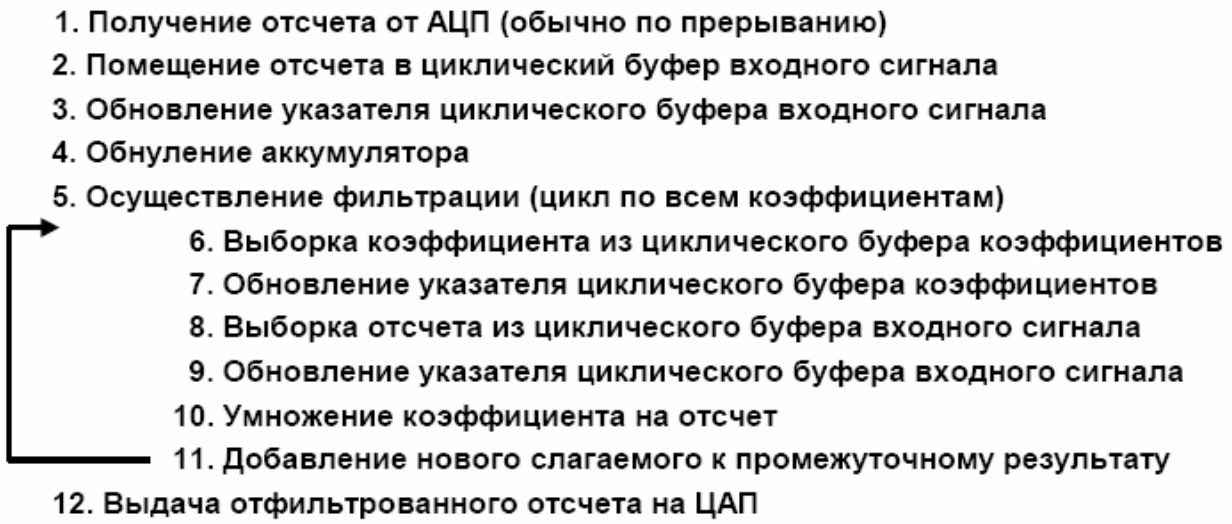

Рис. 5. Алгоритм, реализующий КИХ-фильрацию с циклическим буфером.

ЦСП компании Analog Devices специально разработаны так, что все операции, выполняемые за один цикл фильтра, производятся за один командный цикл процессора, благодаря чему существенно увеличивается эффективность вычислений. Ассемблерный код КИХ-фильтра для семейства процессоров ЦОС ADSP-21XX с фиксированной точкой представлен на рис. 6. Стрелками в тексте помечены исполняемые команды. Первая команда (помеченная меткой `fir:`) инициирует вычисления, очищая регистр MR и загружая первый элемент данных из памяти данных и значение коэффициента из памяти программ соответственно в регистры MX0 и MY0. Затем, для вычисления суммы первых *N–1* слагаемых, в *N–1* циклах выполняется операция умножения с накоплением с автоматической проверкой условия завершения цикла для вычисления суммы первых *N–1* произведений.

Так реализуется свертка выборки следующего набора данных и коэффициентов. Заключительная команда умножения с накоплением производится с округлением результата с точностью до 24-х старших разрядов регистра MR.

```
.MODULE             fir_sub;
{                   FIR Filter Subroutine
                    Calling Parameters
                             I0 --> Oldest input data value in delay line
                             I4 --> Beginning of filter coefficient table
                             L0 = Filter length (N)
                             L4 = Filter length (N)
                             M1,M5 = 1
                             CNTR = Filter length - 1 (N-1)
                    Return Values
                             MR1 = Sum of products (rounded and saturated)
                             I0 --> Oldest input data value in delay line
                             I4 --> Beginning of filter coefficient table
                    Altered Registers
                             MX0,MY0,MR
                    Computation Time
                             (N - 1) + 6 cycles = N + 5 cycles
                    All coefficients are assumed to be in 1.15 format. }
.ENTRY              fir;
fir:                MR=0, MX0=DM(I0,M1), MY0=PM(I4,M5)
                    CNTR = N-1;
                    DO convolution UNTIL CE;
convolution:          MR=MR+MX0*MY0(SS), MX0=DM(I0,M1), MY0=PM(I4,M5);
                    MR=MR+MX0*MY0(RND);
                    IF MV SAT MR;
                    RTS;
.ENDMOD;
```

Рис. 6. Ассемблерный код для ADSP-21x, реализущий КИХ-фильтр.

Пример: Аналоговый ФНЧ с частотой среза 1 кГц реализуется фильтром Чебышева первого рода шестого порядка (характеризуется наличием пульсаций коэффициента передачи в полосе пропускания и отсутствием пульсаций вне полосы пропускания). На практике этот фильтр может быть представлен тремя фильтрами второго порядка, каждый из которых построен на операционном усилителе и нескольких резисторах и конденсаторах. С помощью современных систем автоматизированного проектирования (САПР) создать фильтр шестого порядка достаточно просто, но чтобы удовлетворить техническим требованиям по неравномерности характеристики 0,5 дБ, требуется точный подбор компонентов (см. [5], гл. 1., стр. 7–9)**.**

Цифровой КИХ-фильтр, построенный по схеме рис. 2 со 129 коэффициентами имеет неравномерность характеристики всего 0,002 дБ в полосе пропускания, линейную фазовую характеристику и намного более крутой спад. На практике такие характеристики невозможно реализовать с использованием одних только аналоговых методов. Кроме того, цифровой фильтр не требует подбора компонентов и не чувствителен к дрейфу частоты, так как она ста-

билизирована на кристалле. Для вычисления выходной выборки фильтр со 129 коэффициентами требует 129 операций умножения с накоплением (MAC), которые должны быть закончены за интервал дискретизации $1/F_д$, чтобы обеспечить работу в реальном масштабе времени. В этом примере частота дискретизации равна 10 кГц, поэтому для обработки достаточно 100 мкс. Семейство ADSP-21xx может выполнять все функции, необходимые для реализации фильтра) за один командный цикл. Поэтому фильтр со 129 коэффициентами требует быстродействия более 129/100 мкс = 1,3 миллиона операций с секунду (MIPS). Существующие ЦСП имеют намного большее быстродействие (для серии 16-разрядных ADSP-218x с фиксированной точкой быстродействие достигает 75MIPS).

3. Базовая архитектура ЦСП семейства ADSP-21xx. Особенности программирования

Лидирующее положение в области разработки и производства ЦСП занимают фирмы Analog Devices, Texas Instruments, Motorola, AT&T Microelectronics и ST Microelectronics. Отличия заключаются в арифметических блоках, производительности, разрядности шин данных, структурой внутренних регистров встроенными модулями интерфейсов и предварительной обработки. Несмотря на многообразие процессоров, все они спроектированы так, чтобы оптимизировать выполнение наиболее часто встречающейся операции ЦОС: умножения с накоплением (суммированием) результатов. Можно выделить пять основных требований, которым должны удовлетворять современные ЦСП: 1) быстрое выполнение арифметических операций; 2) повышенная точность представления операндов; 3) возможность одновременной выборки двух операндов; 4) поддержка циклических буферов; 5) организация циклов с автоматической проверкой условия их завершения. Для быстрой обработки сигналов наиболее подходит гарвардская архитектура построения процессора, в которой памяти программ и данных разделены.

Для примера освоения и приобретения навыков работы с этими устройствами рассмотрим процессоры с фиксированной точкой ADSP 2181, выпускаемые фирмой Analog Devices. Детально с особенностями программирования семейства ADSP 21xx можно ознакомиться в [7…10]. Архитектура ADSP 2181 приведена на рис. 14.

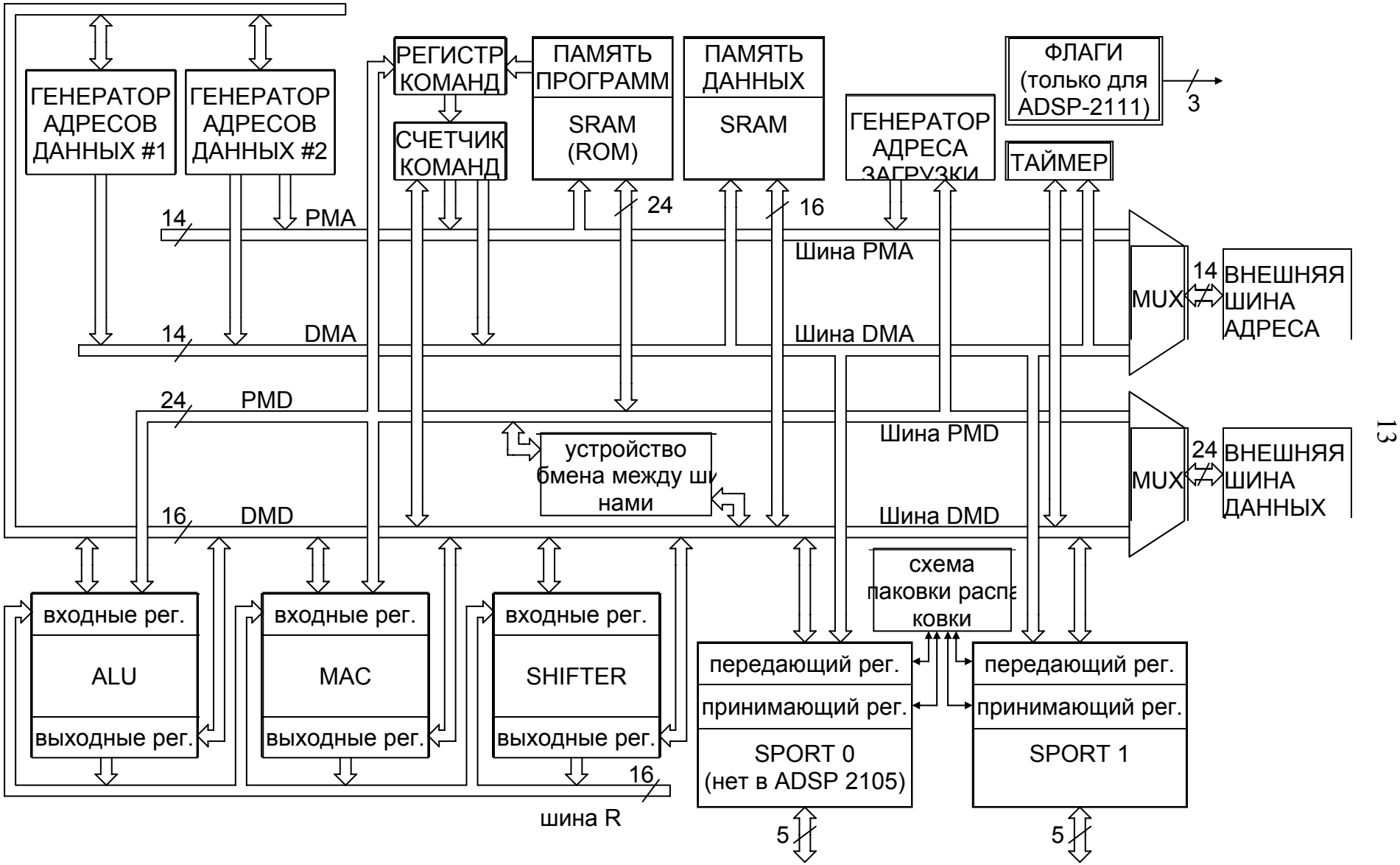

Рис. 14. Базовая архитектура семейства процессоров ADSP 21xx.

Арифметический блок включает арифметическо-логическое устройство (ALU), которое выполняет арифметические и логические операции, умножитель-накопитель (MAC), специально сконфигурированный для выполнения операций умножения и устройство сдвига (SIFTER), позволяющее нормировать числа после выполнения сложения и умножения.

Два генератора адреса данных предназначены для модификации бит адреса при выполнении процессором БПФ.

Программный автомат, состоящий из счетчика, регистра команд и генератора адреса загрузки команд, осуществляет поддержку операций с условными переходами, вызов подпрограмм и возврат в основную программу.

Оперативная память разделена на память данных (16K слов) и память программ (16K слов), однако система команд поддерживает использование памяти программ также и для хранения данных (модифицированная гарвардская архитектура). Разделение основных шин процессора на четыре (шины данных памяти программ, данных памяти данных, адресов памяти и адресов данных) позволяет распараллеливать процессы и одновременно выполнять операции обработки. Шина R служит для пересылки промежуточных результатов между вычислительными блоками.

Для программирования процессора следует представлять его программно-логическую модель (см. рис. 15). Ядро процессоров семейства ADSP-21xx с точки зрения программиста состоит из трех вычислительных устройств, двух генераторов адреса и программного автомата. Обращение к регистрам этих устройств из программы производится по их логическим именам, возле регистров указана их разрядность.

У каждого из регистров вычислителя есть теневой регистр-двойник. Переключение между основными и теневыми регистрами производится программно, командами **ENA SEG_REG (**выбрать вторичный набор регистров, т.е. установить в 1 соответствующий бит в регистре статуса MSTAT) и командой **DIS SEG_REG (**сбросить этот бит).

При разработке программ удобно пользоваться интегрированной средой разработки и отладки (IDDE). Для семейства ADSP соответствующий пакет программ носит название **VisualDSP++**. Он включает в себя все необходимые инструменты

для выполнения проекта, в котором используется как **язык ассемблера**, так и языки высокого уровня (в частности, **язык Си**). Кроме того, в **VisualDSP++** входят текстовый редактор, транслятор, редктор связей, отладчик-симулятор, программа загрузки и другие полезные программы.

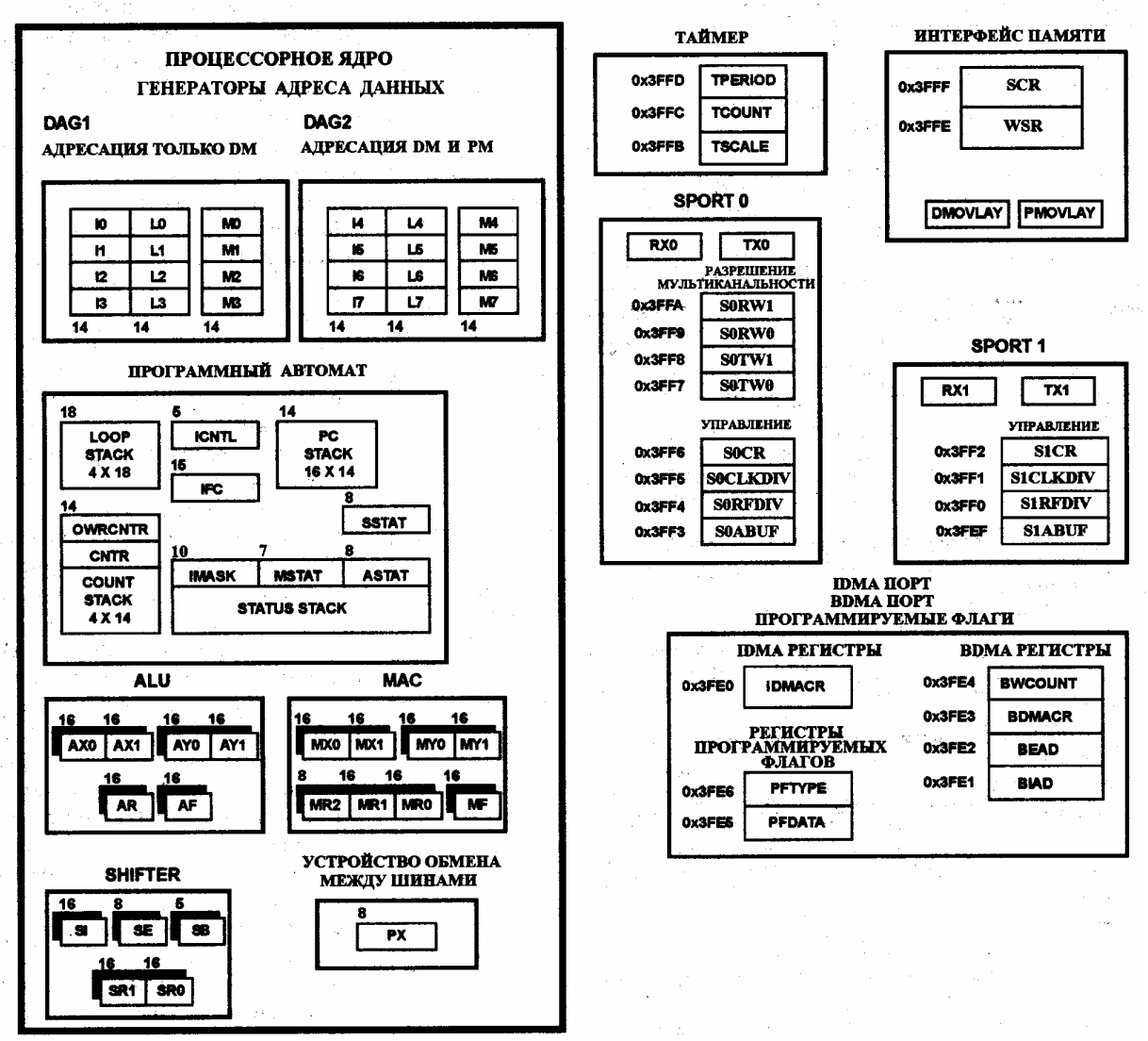

Рис. 15. Доступные для программирования регистры процессоров семейства ASDP218x.

Язык ассемблера состоит из набора команд-инструкций, в которых можно использовать простые операторные выражения, и директив, используемых на этапе трансляции-ассемблирования. Базовый набор команд ADSP-21xx включает традиционные для микропроцессоров операции с блоками вычислителя, команды пересылки данных и инструкции управления. Некоторые примеры типичных инструкций приведены в таблице 1 (полный список команд см. [9], гл. 15):

Таблица 1. Примеры команд ADSP-2181.

IF EQ AR = AX0+AY0;	при условии нулевого значения в регистре AR, к содержимому AX0 прибавляется содержимое AY0 и результат сохраняется в AR (если условие не выполняется, то выполняется инструкция NOP)
AF = MR1 XOR AY1;	в AF заносится результат логической операции «исключающее ИЛИ», выполняемой с данными в MR1 и AY1
AR = TGLBIT 7 OF AX1	установка в AR значения 7-го бита AX1
MR = MX0*MY0(US);	умножение, причем модификатор в скобках (US) сообщает процессору, что первый операнд – беззнаковый, а второй – знаковый
MR = MR-AR*MY1(SS)	умножение с вычитанием и учетом знаков сомножителей
MR = MR+MX1*MY0(RND)	умножение с суммированием, причем RND означает, что оба операнда – знаковые, а результат следует округлить
IF LT MR = MX0*MX0(UU)	умножение выполняется при условии, что значение в MR меньше нуля, результат сохраняется в MR
SR = ASHIFT SI BY –6(LO)	арифметический (т.е. с учетом знака) сдвиг вправо SI на 6 бит относительно младшей части SR
SR = **= SR OR LSHIFT SI BY 3(HY)**	логическое ИЛИ содержимого SR и результата сдвига SI влево на 3 позиции относительно старшей половины группы разрядов (с добавлением нулей справа)
MX0 = 1234	запись в регистр MX0 числа 1234
IMASK = 0xF	запись в регистр маски прерываний IMASK числа F_{16}
AX0 = DM(I0,M0)	косвенное считывание числа из регистра AX0 в память данных, адрес которой указан в регистре I0, причем по завершению считывания I0 обновляется значением, содержащемся в M0
AX0 = 0 **DM(0x3FFE) = AX0**	т.к. нет инструкции записи константы по непосредственному адресу, то эти команды используются для обнуления регистра DM, который находится по адресу памяти данных 3FFE

PM(I4,M5) = MR1	косвенная запись памяти команд (значением, содержащемся в MR1) по адресу, указываемому I4 и одновременной модификацией I4 значением из M5
JUMP (I4)	переход к команде, содержащейся в ячейке с адресом, указанном в I4 (косвенная адресация)
IF NOT CE JUMP my_label **….** **my_label:**	если счетчик числа повторений ненулевой, то перейти к метке **my_label** (имя метки – определяемый пользователем идентификатор, который используется как адрес перехода)
CNTR = 10	текущее содержимое счетчика цикла CNTRL помещается в стек, после чего в этот регистр помещается значение 10
RTS	возврат из подпрограммы обеспечивает условный возврат к точке программы, откуда произошел вызов подпрограммы
RTI	возврат из подпрограммы обработки прерывания
ENA	разрешить какие-либо иденти-фикаторы или выполнение одного из семи режимов, соответствующих битам регистра MSTAT
ENA TIMER	разрешить обработку прерываний от таймера
DIS	запретить (то же, что и команда ENA)
NOP	нет выполнения никакой операции

Специфической особенностью ЦСП являются многофункциональные инструкции, которые используют преимущества параллелизма семейства и позволяют выполнять комбинацию операций за один цикл. Примеры таких инструкций приведены в табл.2.

Таблица 2. Многофункциональные инструкции ADSP-2181.

умножение с накоплением:
MR = MR+MX0*MY0(SS), MX0 = DM(I0,M1), MY0 = PM(I4,M4)
к содержимому MR прибавляется результат перемножения чисел, находящихся в MX0 и MY0 ((SS) – означает, что оба операнда являются знаковыми)), после чего из памяти данных в MX0 загружается значение из ячейки с адресом, указанным в I0, затем содержимое I0 модифицируется значением из M1, в том же цикле содержимое MY0 заменяется числом хранящемся в памяти программ по адресу, указанному в I4 и значение в I4 заменяется на значение из M4

вычисление с чтением из памяти:
AR = AX0+AY0, AX0 = DM(I0,M3)
сложение в ALU и загрузка операнда из памяти данных
вычисление с записью в память:
DM(I0,M0) = AR, AR = AX0+AY0
в начале цикла идет загрузка старого значения, а затем - вычисление нового
вычисление с пересылкой из регистр-регистр:
AR = AX0+AY0, AX0 = MR2
пересылка регистр-регистр по шине R может быть между любыми регистрами вычислителя, кроме регистров обратной связи AF, MF и регистра SB

В программе на ассемблере вместе с командами содержатся **директивы**, используемые на этапе трансляции программы в машинный код, но сами они не транслируются. В программном коде директивы помечаются точками. Пример:

```
.VAR/DM my_array[128]
```

директива, объявляющая, что в памяти данных формируется массив из 128-ми 16-ти битовых элементов, с началом по адресу **my_array**.

```
.INIT my_array: <filename.dat>
```

эта директива считывает значение из файла данных ***filename.dat*** в массив во время сборки программы (работает только в режиме эмуляции).

```
.include «def2181.h»
```

подключает файл ***def2181.h*** расположенный в той же директории, что и программа, в котором как константы определены имена регистров и их отдельных бит, а также имена адресов векторов прерывания.

4. Описание лабораторного макета

В лабораторных работах используется отладочная плата EZ LAB-2181 и соединенная с ней плата со световыми индикаторами и переключателями (см. рис. 16, 17 и 18). Как показано на рис. 17, вместе ADSP-2181 на лабораторной плате размещены: статическое ПЗУ, микросхема кодека аудиосигналов AD1847, (в этом устройстве объединены антиэлайсинговый ФНЧ, АЦП, ЦАП, и сглаживающий ФНЧ), разъем последовательного порта (для связи с управляющим ПК), разъемы ввода и вывода аналогового сигнала, а также разъемы шин адреса, данных и внешних портов процессора. Питание устройства осуществляется от источника = 9…12 В, ток потребления не более 300 мА.

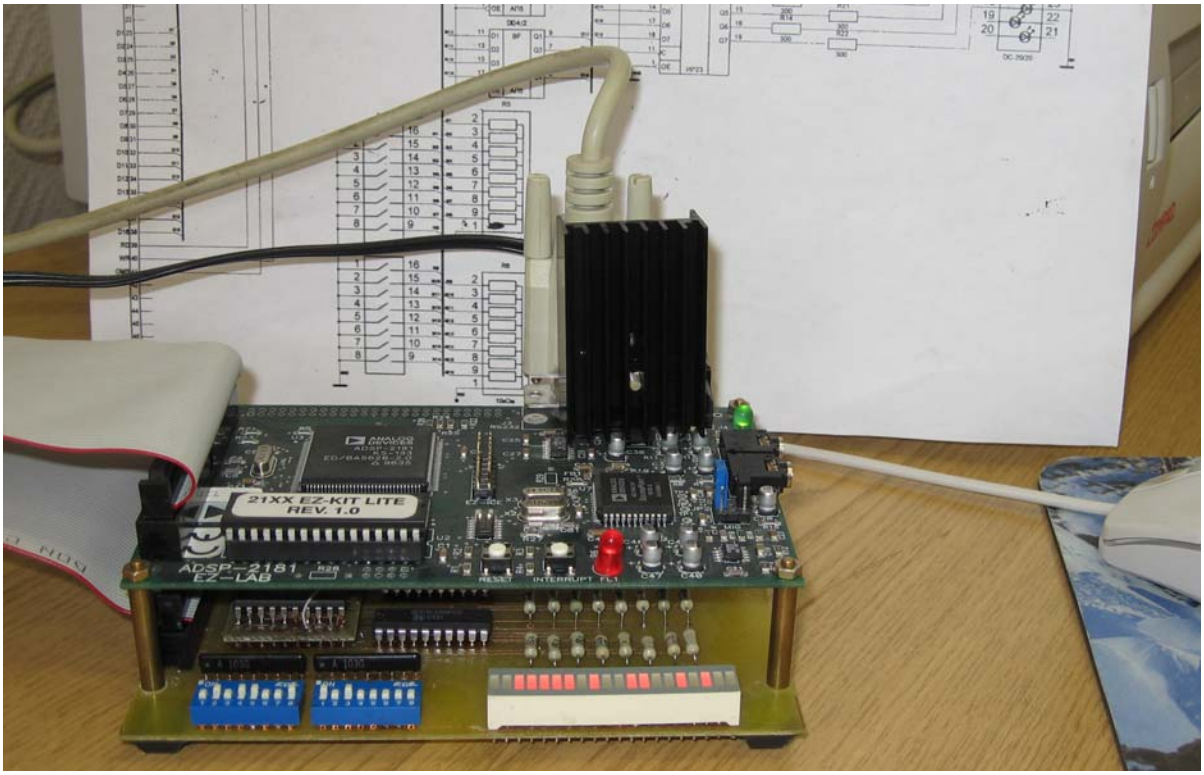

Рис. 16. Лабораторный макет устройства для работы с ЦСП.

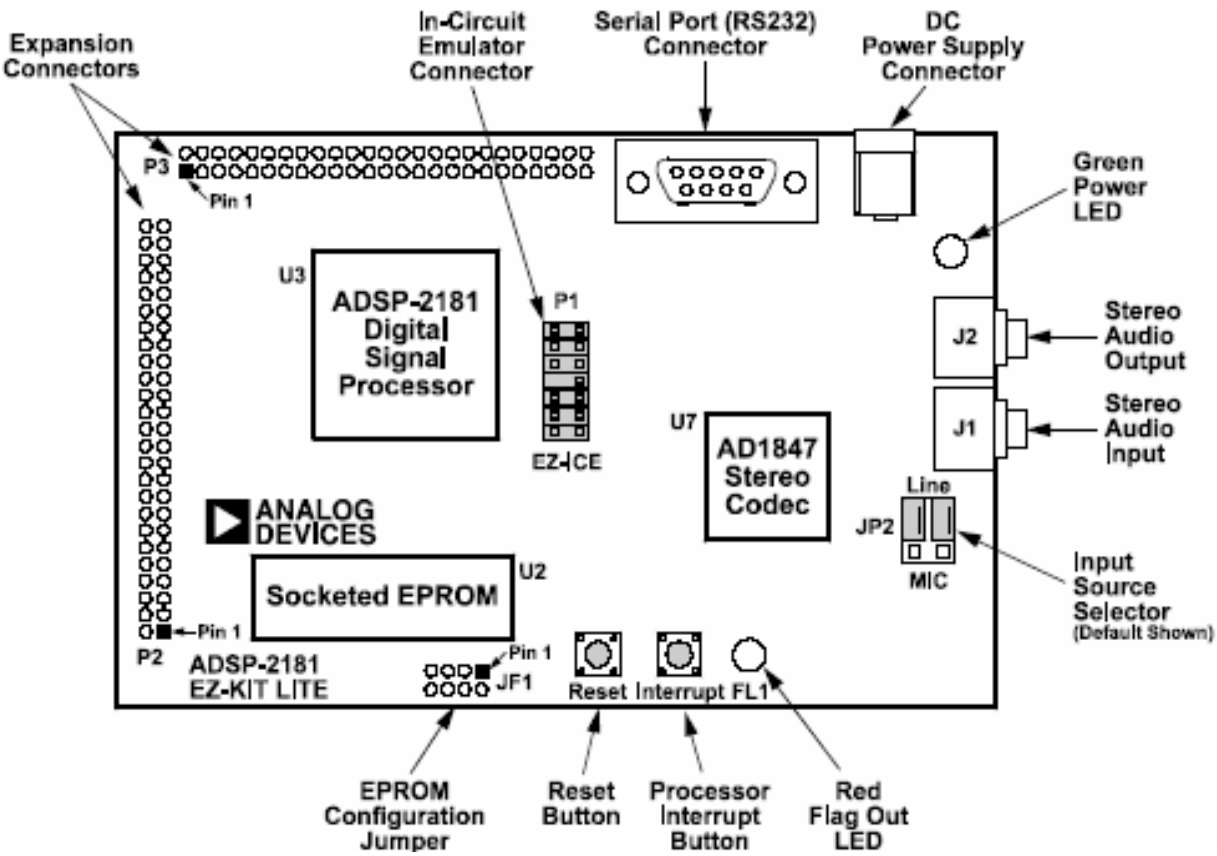

Рис. 17. Расположение элементов на учебной плате с ADSP 2181.

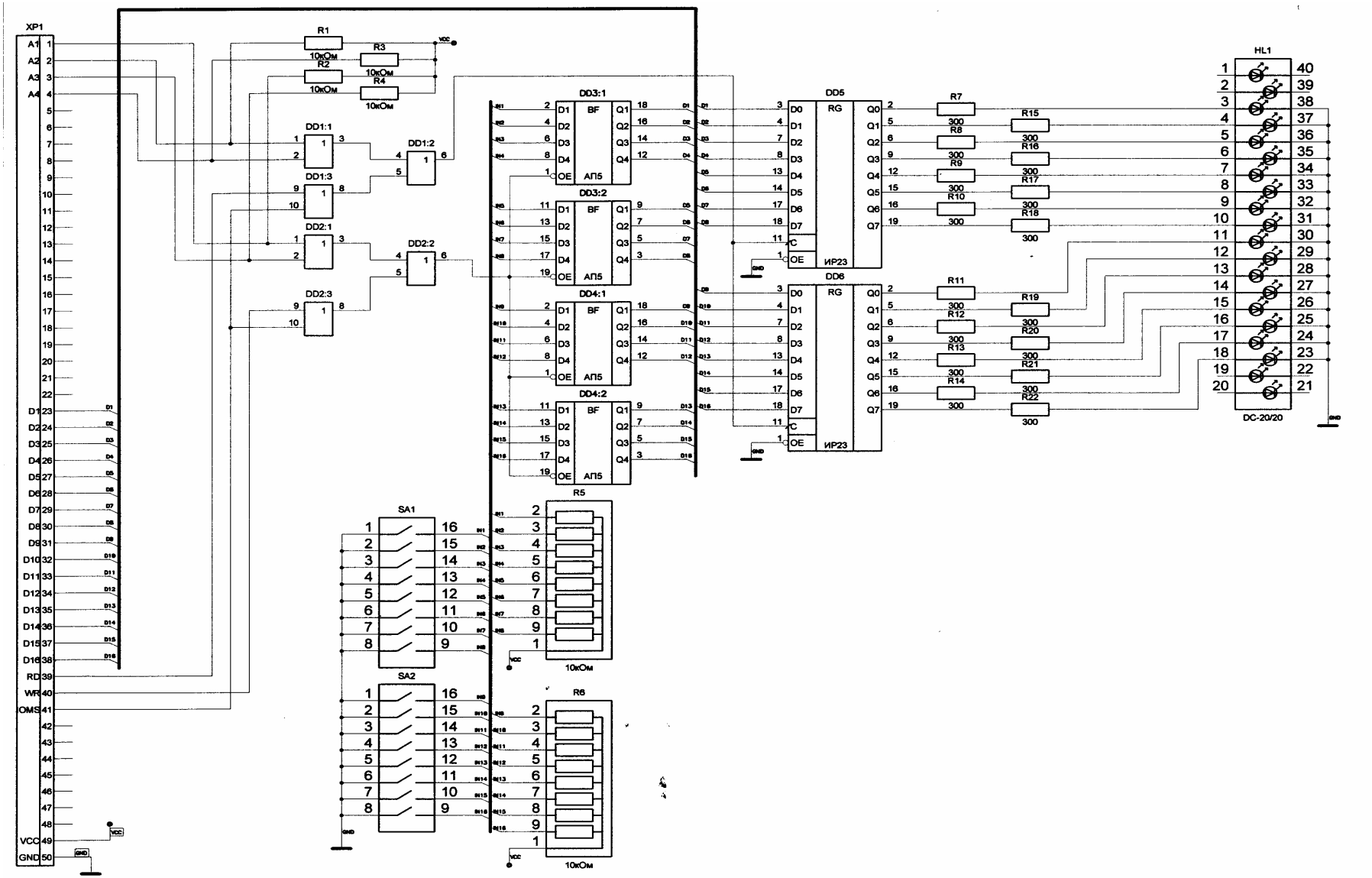

Рис. 18. Принципиальная электрическая схема платы со светодиодами и переключателями

Как следует из рис. 18, переключатели и светодиоды подключены к выходной шине данных процессора при помощи простейшего дешифратора адреса на основе «логического И» (микросхемы ЛЛ1), буферных приемопередатчиков шины (микросхемы АП5) и буферных регистров-защелок (микросхемы ИР23).

5. Лабораторная работа ЦСП-1. Разработка простейшего устройства на основе ADSP 2181

Цель работы: ознакомится с основными элементами интегрированной среды проектирования VisualDSP++. На примере программы, реализующей считывание и запись цифровых данных, изучить основные стадии разработки проекта. Разработать устройство, реализующее считывание положений флажков-переключателей и различные алгоритмы вывода информации.

Задание 1. Внимательно изучите возможности отладочной платы и платы ввода-вывода. Уясните, как осуществляется ввод информации от переключателей и вывод сигналов управления светодиодами. Определите адрес для считывания положений переключателей и адрес для записи световых индикаторов, которые затем должны использоваться в программе.

Задание 2. Изучите особенности работы с интегрированной средой проектирования и отладки систем VisualDSP++. Следуя приведенным ниже инструкциям, выполните по шагам все действия необходимые для разработки проекта. Создайте папку, в которой будет храниться ваш проект ***«CODE»***. Поместите туда, файл с исходным шаблоном проекта ***«first00.asm»***, а также файл с управляющей информацией для редактора связей ***«ADSP-2181.ldf»,*** содержащий инструкции, необходимые для компиляции и ориентированные на данный тип ЦСП.

2.1. Разработка проекта с примером программы ввода-вывода

2.1.1. Запустите на выполнение программу VisialDSP++. В процессе загрузки потребуется проверка, что к ПК подключена плата EZ-LAB. Для инициализации по запросу следует нажать кнопку сброса (**reset**) на плате. После этого на экране появится типичное окно, интерфейса программы, подобное тому, что изображено на рис. 19. По рис. 19 и приведенной ниже таблице 3, пользуясь справочной системой (клавиша {F1}), внимательно изучите входящие в интерфейс пользователя элементы интерфейса и их назначение.

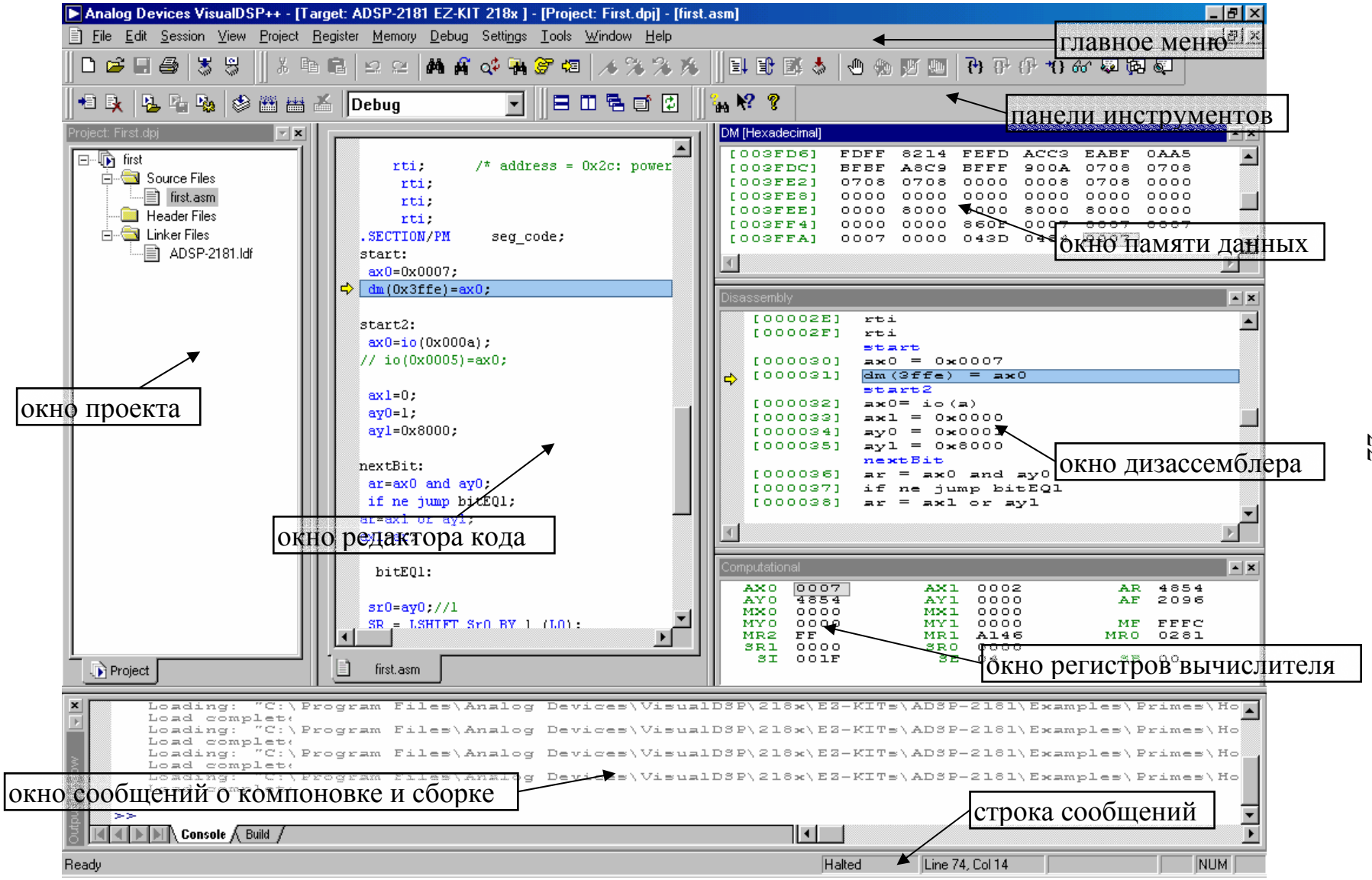

Рис. 19. Пример пользовательского интерфейса программы VisualDSP++.

Таблица 3. Основные инструменты VisualDSP++.

	**операции с файлами:** создать, открыть, сохранить, распечатать, загрузить программу, перезагрузить новую
	**редактирование кода программ:** вырезать, копировать, вставить, отменить, снова восстановить, поиск, повторить поиск, заменить, искать текст, перейти, показать исходный файл, установить закладку, перейти к следующей, прейти к предыдущей, снять все закладки
	**операции с проектом:** добавить файл, удалить из проекта, открыть окно опций, собрать из выбранных файлов, собрать проект с учетом обновленных файлов, собрать заново, остановить сборку, режим работы
	**отладка программы:** старт или продолжение, запуск сначала, остановка, сброс процессора, назначение точки остановки, удаление всех точек остановки, разрешить или отменить данную точку, в пошаговом режиме разрешить вход в подпрограмму, обойти подпрограмму, выйти из подпрограммы, открыть окна просмотра выражений, Locals, Stack, Disassembly

2.1.2. Создайте новый проект. Для этого выберите пункт **«New»** функции **«Project»** в соответствующем разделе главного меню. В окне **Project/Project options…** назначьте тип процессора (ADSP-2181). Дайте проекту имя и сохраните проект, вызвав соответствующее меню функции **«Project»**.

2.1.3. Теперь следует включить в проект файл кодом программы на языке ассемблер или C++. Для этого следует в окне проекта выделить строку ***«*Source Files»** и вызвать контекстное меню, нажав правую кнопку мыши. Затем следует выбрать опцию ***«*Add to project»/«File(s)…»***. Появится стандартное диалоговое окно открытия файла, позволяющее указать путь к файлу и его имя. Добавьте в проект файл шаблона программы ***«first00.asm»***. Для этого найдите файл в файловой системе и дважды щелкните левой кнопкой мыши на соответствующей пиктограмме этого окна или

просто используйте команду ***«Открыть»***. Аналогично, используя включите в проект файл с управляющей информацией для редактора связей ***«ADSP-2181.ldf»***. Открыв файловое окно Windows, убедитесь, что необходимые для работы над проектом файлы созданы. Главный файл проекта может быть только один. Он имеет расширение ***.drj*** и содержит служебную информацию для построения программы: перечень исходных файлов проекта и начальные сведения об условиях построения проекта.

2.1.4. Откройте окно ***«*Computational»**, выбрав соответствующий пункт меню ***«*Register»**. Поскольку проект уже настроен для работы с типом процессора ADSP-2181, то регистры его вычислителя, следить за которыми полезно в процессе отладки программы, отобразятся в этом окне автоматически. Вы также можете сами изменять значения регистров, щелкая левой кнопкой мыши на соответствующем регистре. Пользуясь необходимыми панелями инструментов, вызываемыми из меню, а также открывающимися списками функций, откройте и расположите на экране рабочие окна проекта **«Disassembly»** и **«Dump memory»** так, как это показано на рис. 19. В этих окнах в пошаговом режиме выполнения программы можно наблюдать, как изменяется состояние регистров и памяти в ходе работы.

2.1.5. Отредактируйте файл шаблона, вписав в него строки с операторами чтения и записи данных так, как это показано ниже (строки для ввода выделены жирным шрифтом):

```
#include    «def2181.h»
.SECTION/DM data1;
.VAR temp;
.SECTION/PM interrupts;/*ниже зарезервированы места для*/
                       /*подпрограмм обработки прерываний */
jump start;  /* address = 0x00: reset interrupt vector */
   rti;
   rti;
   rti;
rti;          /* address = 0x04: IRQ2 interrupt vector */
   rti;
   rti;
   rti;
rti;         /* address = 0x08: IRQL1 interrupt vector */
   rti;
   rti;
   rti;
rti;         /* address = 0x0c: IRQL0 interrupt vector */
```

```
   rti;
   rti;
   rti;
rti;          /* address = 0x10: SPORT0 tx interrupt vector */
   rti;
   rti;
   rti;
rti;          /* address = 0x14: SPORT1 rx interrupt vector */
   rti;
   rti;
   rti;
rti;          /* address = 0x18: IRQE interrupt vector */
   rti;
   rti;
   rti;
rti;          /* address = 0x1c: BDMA interrupt vector */
   rti;
   rti;
   rti;
rti;     /* address = 0x20: SPORT1 tx or */
              /* IRQ1 interrupt vector */
   rti;
   rti;
   rti;
rti;          /* address = 0x24: SPORT1 rx or */
              /* IRQ0 interrupt vector    */
   rti;
   rti;
   rti;
nop;          /* address = 0x28: timer interrupt vector */
   rti;
   rti;
   rti;
rti;          /* address = 0x2c: power down interrupt vector*/
   rti;
   rti;
   rti;
.SECTION/PM seg_code; /*окончание сегмента обработки */
                      /*прерываний*/
start:
   ax0=0x0007;        /* инициализация памяти */
   dm(0x3ffe)=ax0;
                      /* здесь д.б. перестановка битов */
start2:
   ax1=io(0xXXXX);    /* здесь д.б. адрес на ввод данных */
   io(0xZZZZ)=ax1;    /* здесь д.б. адрес на вывод данных */
jump start2;
```

2.1.6. Следующий шаг заключается в создании машинного кода из исходного текста программы. Для выполнения этой процедуры следует выбрать опцию **«Build project»** в меню **«Projects»** или

нажать клавишу {F7}. После выполнения программы ассемблирования в окне **«Output Window»** будет выведена информация о результатах. Если выдано сообщение об ошибках, то следует найти их, исправить, редактируя код программы и затем повторить сборку проекта. В случае успеха в этом окне будет выдано сообщение *«Build completed successfully»*. Обратите внимание, что после компиляции в рабочей папке проекта автоматически создается папка **«Debug»**, которая содержит файлы объектным кодом (**.doj**) и исполняемым кодом (**.dxe**) При этом в режиме работы с отладочной платой исполняемый код автоматически загружается в процессор и тот запускается на выполнение программы.

Проверьте, что отлаженная и загруженная в процессор на отладочной плате программа работает без логических ошибок. Для этого наблюдайте реакцию светодиодов на изменение положения переключателей. Обратите внимание, в каком располагаются биты входных и выходных данных, соответствующие положениям переключателей и светодиодов.

2.1.7. Для наблюдения за ходом работы программы при ее отладке и поиске логических ошибок, запустите программу в режиме симулятора, выбрав в меню **«Debug»** пункт «**Step into»** или нажав клавишу {F11}. О переходе в этот режим свидетельствует выделение строки кода в окне редактирования программы (см. рис. 19). Желтая стрелка слева указывает на инструкцию, которая будет выполняться на следующем шаге в соответствии с содержимым счетчика команд PC. Выбрав опцию **«Go to cursor»** из меню **«Debug»** можно сразу перейти к нужному месту программы.

Процесс отладки заключается в отслеживании и управлении ходом выполнения программы с помощью окна редактирования и специальных окон, показывающих состояния рабочих регистров и содержимого памяти.

Запустите пошаговый режим работы и, нажимая несколько раз клавишу {F11} наблюдайте, как изменяются по шагам данные в регистре **AX0** и в памяти данных.

При отладке программы полезно остановить ее после выполнения какого-либо оператора, чтобы проанализировать содержимое регистров и памяти. Для задания точек остановки в ассемблерном коде программы, нужно выбрать пункт меню **«Insert Breakpoint»** или {F9}. Красный круг, появившийся в полях слева от кода,

отмечает заданное место остановки. После нажатия клавиши {F5} или выбора пункта **«Run»** в меню **«Debug»**, программа начнет выполняться и остановится перед выполнением отмеченной вами команды. Возобновить выполнение программы можно, выбрав соответствующий пункт меню, или нажав {F11}.

Задание 3. Модифицируйте, разработанную выше программу, так, чтобы порядок включения переключателей соответствовал порядку зажигания светодиодов. Для перестановки битов входных данных воспользуйтесь следующим фрагментом кода, поместив его в указанном месте исходной программы:

```
start2:
    ax0=io(0x000a);
    ax1=0;
    ay0=1;
    ay1=0x8000;
nextBit:
    ar=ax0 and ay0;
    if ne jump bitEQ1;
    ar=ax1 or ay1;
    ax1=ar;
bitEQ1:
    sr0=ay0;//1
    SR = LSHIFT Sr0 BY 1 (LO);
    ay0=sr0;
    sr0=ay1;//0x8000
    SR = LSHIFT Sr0 BY -1 (LO);
    ay1=sr0;
    ar=pass ay1;
    if ne jump nextBit;
endRotate:
```

Задание 4. Разработайте программу, осуществляющую вывод на светодиоды инкрементный счет. Проверьте работу программы в пошаговом режиме.

Задание 5. Разработайте программу, осуществляющую вывод индикации на светодиоды в режиме «бегущие огни», т.е. последовательно включающую и выключающую соседние светодиоды. Примерный код соответствующего фрагмента:

```
start2:
   ay0=1;
     io(0x0005)=ay0;     //вывести на светодиоды единицу
   nextBit:
     CNTR=0x776;         //счетчик тактов задержки
   DO delay UNTIL CE;    //задержка
```

```
    idle;
delay: nop;
    sr0=ay0;            //сдвиг единицы влево
    SR = LSHIFT Sr0 BY 1 (LO);
    ay0=sr0;
    io(0x0005)=sr0;     //выводим новое значение на светодиоды
    ax0=0x8000;
    none=ay0-ax0;
   if ne jump nextBit;
    ay0=1;
    io(0x0005)=ay0;
  jump nextBit;
```

Проверьте работу программы в пошаговом режиме и в режиме непрерывного выполнения.

Задание 6. Взяв за основу программу из задания 3, разработайте программу перемножения двух чисел, задаваемых побайтно с помощью переключателей. Результат перемножения выведите на светодиоды. Пример фрагмента кода программы перемножения:

```
    ay1=0x00ff;         //выделить младший байт
    ar=ax1 and ay1;
    mx0=ar; //сохранить младший байт в регистре умножителя mx0
    ay1=0xff00;    //выделить старший байт
    ar=ax1 and ay1;
    sr0=ar;             //передать слово в регистр сдвига
    SR = LSHIFT Sr0 BY -8 (LO); //сдвиг в младший байт
    my0=sr0; //полученный старший байт занести в умножитель
    mr=mx0 * my0(su);   //умножение старш. и младш. байтов
    io(0x0005)=mr0;     //вывести результат умножения
 jump start2;
```

Проверьте работу программы в пошаговом режиме и в режиме непрерывного выполнения. По индикации на светодиодах убедитесь, что счет выполняется без ошибок.

6. Лабораторная работа ЦСП-2. Пример разработки цифрового КИХ-фильтра на основе сигнального процессора ADSP-2181.

Цель работы: На примере программы, реализующей цифровой КИХ-фильтр, изучить возможности сигнального процессора ADSP-2181 по управлению вводом-выводом аналоговых сигналов и их фильтрации, а также принципы работы цифровых фильтров. Исследовать цифровой КИХ-фильтр, реализованный с помощью макетной платы EZ LAB-2181.

Задание 1. Проанализируйте код программы, реализующей цифровой КИХ-фильтр. Демонстрационная программа, использующаяся в данной работе, состоит из нескольких модулей. Базовый модуль управляет вводом-выводом аналоговых сигналов. Аналоговый сигнал с микрофонного входа оцифровывается с помощью аудио стерео-кодека AD1847 (16 бит, частота дискретизации от 5.5 до 48 кГц) и передается в последовательный порт сигнального процессора. ЦСП считывает сигнал с последовательного порта; далее оцифрованный сигнал либо выводится без изменений через этот же порт обратно на кодек, либо сначала обрабатывается с помощью программного модуля, реализующего КИХ-фильтр (256-звенный), а затем передается в кодек, который преобразует цифровой сигнал обратно в аналоговый. В программе реализован также генератор шума, который позволяет продемонстрировать работу цифрового фильтра без подачи внешнего сигнала на макетную плату.

1.1. Создайте папку, в которой будет храниться ваш проект ***«FIR»***. Поместите туда файл с исходным кодом демонстрационной программы ***«firdemo.asm»***, файл с коэффициентами фильтра ***«fir.dat»***, а также файл с управляющей информацией для редактора связей ***«ADSP-2181.ldf»***. Запустите на выполнение VisualDSP++, создайте новый проект, а затем включите в него файлы с кодом программы (в раздел «**Source Files**»), коэффициентами фильтра («**Data Files**») и управляющей информацией («**Linker Files**», подробнее см. п. 2.1.1 – 2.1.3 лабораторной работы ЦСП-1).

1.2. Внимательно изучите код программы цифрового фильтра (***«firdemo.asm»***). Особенное внимание обратите на части программы, отвечающие за инициализацию цифрового сигнального процессора ADSP-2181 и его портов ввода-вывода (состояние процессора определяется значением регистров **Sys_Ctrl_Reg**, **mstat**, **Sport0_Ctrl_Reg**, **Sport0_Autobuf_Ctrl**,) за инициализацию аудиокодека, а также организацию прерываний (регистры **icntl**, **imask**) и тактирования схемы (регистры **Sport0_Rfsdiv**, **Sport0_Sclkdiv**). Изучите комментарии в программе к значениям регистров.

1.3. Изучите часть кода, относящуюся к реализации цифрового фильтра, найдите переменные, отвечающие за работу цифрового фильтра. Отключите использование фильтра. Определите по тексту программы, как задается массив коэффициентов для реализа-

ции того или иного фильтра и как в данной программе можно изменять характеристики фильтров.

1.4. Создайте машинный код из исходного текста программы. Загрузите исполняемый код в процессор (см. п. 2.1.6 лабораторной работы ЦСП-1).

Задание 2. Выполните тестирование программы цифрового фильтра, реализованного на ADSP-2181.

2.1. Для проверки работоспособности программы удобно использовать графические возможности визуализации содержимого любой области памяти данных или программ, которые предоставляет разработчику в VisualDSP++ (см. рис. 20).

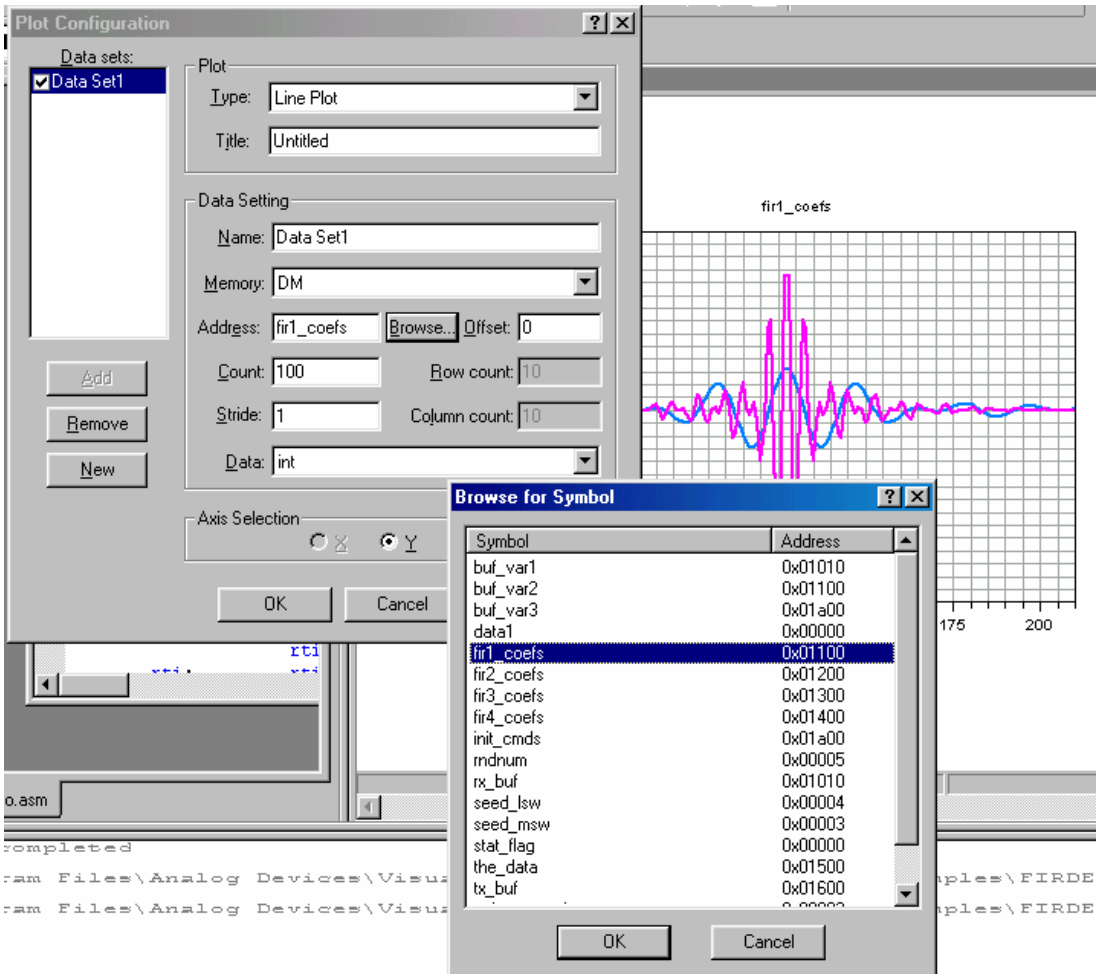

Рис. 20. Настройки средств визуализации VisualDSP++ и графическое изображение значений массивов коэффициентов фильтров.

Для этого используя команды **«Debug Windows»/«View»/«Plots»/«New…»**, откроте диалоговое окно конфигурирования графика **«Plot Configuration»**. В меню **«Type»** следует выбрать

тип графика (линейный, зависимость X-Y, спектрограмма и т.д.) и задать его имя в строке **«Title»**. Затем следует определить параметры вывода данных: тип области данных (**«Memory»**, **PM** – программ, **DM** – данных и т.д.), начальный адрес-метку (**«Address»**), сдвиг (**«Offset»**), число выводимых точек (**«Count»**), тип данных (**«Type»**) и др. Для поиска начального адреса того или иного массива данных, определенных в программе, удобно воспользоваться функцией поиска **«Browse»**. После задания всех параметров следует нажать **«Add»,** и в левой части окна должно появиться название графика, который будет выведен в рабочем окне проекта после ввода **«Ok»**. При необходимости можно задать вывод нескольких массивов данных в одном окне (кнопка **«New»**), произвольно задавать оси для функциональных зависимостей (**«Axis Selection»**); с помощью меню **«Settings»** – определить цвета линий и фона, размер полей, сетку, надписи на осях, масштаб, **min/max** значения и т.д.

Выведите на график (панель **«Plots»**) импульсные характеристики фильтров, реализованных в данной программе (массивы по 256 точек). Для этого используйте связь между коэффициентами фильтра и его импульсной характеристикой (см. раздел 2 и [5, 9]).

2.2. Подайте на макетную плату (на разъем **«IN»**) синусоидальный сигнал с частотой 20 Гц и амплитудой 0.5 – 1 В с внешнего генератора. Запустите на выполнение код, загруженный в процессор (**«Debug»/«Run…»** или {F5}). Поскольку цифровой фильтр отключен, входной сигнал должен появиться на выходе макетной платы (разъем **«OUT»**). Подключите к аналоговому выходу платы EZ LAB осциллограф и сравните входной и выходной сигналы. Изменяя частоту входного сигнала, определите рабочий диапазон частот лабораторного макета.

2.3. Остановите выполнение программы (**«Debug»/«Halt…»** или {Shift+F5}). После остановки VisualDSP++ считывает данные из памяти процессора. Выведите на график выходные данные кодека (массив **«the_data»,** число точек массива должно соответствовать числу звеньев фильтра).

2.4. Выполните п.п. 2.2–2.3, задавая значения частоты входного сигнала от генератора 200 Гц и 2 кГц. Настройте окно **«Plots»** для визуализации выходных данных. На экране осциллографа и в соответствующем окне VIsualDSP++ наблюдайте выходные данные. Определите, как изменяется амплитуда выходного сигнала.

2.5. Включите фильтрацию сигнала и выполните п.п. 2.2–2.4. Изменяя частоту входного сигнала, убедитесь в работоспособности реализованного на ADSP-2181 цифрового фильтра. Оцените границы полосы пропускания.

Задание 3. Выполните исследование цифрового фильтра.

Предложите варианты схем исследования АЧХ и ФЧХ цифровых фильтров. Соберите одну из схем по указанию преподавателя и выполните исследования характеристик цифрового фильтра, реализованного на EZ-LAB ADSP-2181. Определите тип фильтра и его полосу пропускания.

Библиографический список

1. Баскаков С.И. Радиотехнические цепи и сигналы: Учебник, М.: Высш. школа, 1983. – 536 с.
2. Айфичер Э.С., Джервис Б.У. Цифровая обработка сигналов: практический подход. 2-е изд. Пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 992 с.
3. Солонина А.И., Улахнович Д.А., Яковлев Л.А. Алгоритмы и процессы цифровой обработки сигналов / СПб.: БХВ-Петербург, 2001. – 464 с.
4. А. Оппенгейм, Р. Шафер Цифровая обработка сигналов Пер. с англ. С.А. Кулешова под ред. А.С. Ненашева / М.: Техносфера, 2006. – 856 с.
5. www.eks.fel.mirea.ru / Учебник по DSP: *1DSP.pdf…10DSP.pdf.*
6. Сато. Ю. Обработка сигналов. Первое знакомство. / Пер. с яп. под ред. Ёсифуми Амэмия. – М.: Издательский дом «Додека-XXI», 2002. – 176 с.
7. Марков С. Цифровые сигнальные процессоры. Книга 1. М: фирма МИКРОАРТ, 1996. – 144 с.
8. Основы цифровой обработки сигналов: Курс лекций / Авторы: А.И. Солонина, Д.А. Улахнович, С.М. Арбузов, Е.Б. Соловьева, И.И. Гук. – СПб.: БХВ–Петербург, 2003.– 608 с.
9. Руководство пользователя по сигнальным микропроцессорам семейства ADSP-2100. / Пер. с англ. О.В. Луневой. Под. Ред. А.Д. Викторова. / Санкт-Петербургский государственный электротехнический университет. – СПб., 1997. – 527 с.
10. www.dian.ru / О. Вальпа. Цифровые сигнальные процессоры (цикл статей в журнале «Компоненты и технологии» 2004- 2006 г.г.).