Нижегородский государственный университет им. Н.И. Лобачевского

Национальный исследовательский университет

Учебно-научный и инновационный комплекс
"Физические основы информационно-телекоммуникационных систем"

Основная образовательная программа
010300 «Фундаментальная информатика и информационные технологии»,
общий профиль, квалификация (степень) бакалавр
Учебно-методический комплекс по дисциплине
«Аппаратные средства вычислительной техники»
Учебно-методический комплекс по дисциплине
«Методы оптимизации и исследование операций»

Основная образовательная программа
090302 «Информационная безопасность телекоммуникационных систем»,
общий профиль, квалификация (степень) специалист
Учебно-методический комплекс по дисциплине
«Аппаратные средства вычислительной техники»

Бугров В.Н., Ивлев Д.Н., Шкелёв Е.И.

ЦИФРОВАЯ ОБРАБОТКА СИГНАЛОВ С ПРИМЕНЕНИЕМ ЦИФРОВЫХ СИГНАЛЬНЫХ ПРОЦЕССОРОВ

Электронное учебно-методическое пособие

Мероприятие 1.2. Совершенствование образовательных технологий, укрепление материально-технической базы учебного процесса

Нижний Новгород
2012

ЦИФРОВАЯ ОБРАБОТКА СИГНАЛОВ С ПРИМЕНЕНИЕМ ЦИФРОВЫХ СИГНАЛЬНЫХ ПРОЦЕССОРОВ. Бугров В.Н., Ивлев Д.Н., Шкелёв Е.И. Электронное учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2012. – 84 с.

Учебно-методическое пособие содержит описание способов цифровой обработки сигналов с использованием цифровых процессоров сигналов. Описаны методика проектирования цифровых фильтров с помощью метода целочисленного нелинейного программирования, цифровой спектральный анализ на основе быстрого преобразования Фурье и способы их реализации на цифровых сигнальных процессорах. Даны описания структуры и характеристик сигнальных процессоров ADSP-21364 и TMS320F28335. Приведена схема лабораторной установки для измерения частотных характеристик реализованных на сигнальном процессоре цифровых фильтров, описана методика измерения частотных характеристик. Даны задания и порядок выполнения лабораторной работы, контрольные вопросы.

Электронное учебно-методическое пособие предназначено для студентов ННГУ, обучающихся по направлению подготовки 010300 «Фундаментальная информатика и информационные технологии», изучающих курсы «Аппаратные средства вычислительной техники» и «Методы оптимизации и исследование операций», и по специальности 090302 «Информационная безопасность телекоммуникационных систем», изучающих курс «Аппаратные средства вычислительной техники».

Цель данной лабораторной работы состоит в изучении способов цифровой фильтрации и спектрального анализа с использованием сигнальных процессоров, а также в освоении способов реализации, средств разработки необходимого программного обеспечения и способов тестирования реальными сигналами. Изучаются: 1) проектирование цифровых фильтров (ЦФ) методом целочисленного нелинейного программирования (ЦНП) и их реализация на цифровых процессорах сигналов с применением целочисленных вычислительных операций и 2) цифровой спектральный анализ на основе быстрого преобразования Фурье (БПФ) с применением вычислительных операций с плавающей точкой.

1. Целочисленное проектирование цифровых фильтров

1.1. Методика проектирования ЦНП-фильтров

Одной из задач, которым посвящена данная лабораторная работа, является задача моделирования и многофункционального синтеза ЦНП-фильтров.

Если рассматривать цифровые фильтры как устройства частотной селекции, то их разработку можно выполнять на основе требований к их частотным характеристикам. Селективные свойства в частотной области определяются коэффициентом передачи

$$H(j\omega) = |H(j\omega)|e^{j\varphi(\omega)} = H(\omega)e^{j\varphi(\omega)},$$

в котором $H(\omega) = |H(j\omega)|$ – амплитудно-частотная (АЧХ), а $\varphi(\omega)$ – фазочастотная (ФЧХ) характеристики. Кроме $H(\omega)$ и $\varphi(\omega)$ важными характеристиками являются фазовая $\tau_{фаз} = \varphi(\omega)/\omega$ и групповая (ГВЗ) $\tau_{гр} = \delta\varphi(\omega)/\delta\omega$ задержки.

Проектируемые цифровые фильтры (как и аналоговые) должны удовлетворять совокупности требований к их характеристикам, включая АЧХ, ФЧХ, групповую и фазовую задержки и эти требования должны удовлетворяться в процессе синтеза ЦФ. Такой синтез принято называть многофункциональным в отличие от многокритериального синтеза, при котором в расчёт берётся лишь одна (например, амплитудно-частотная) характеристика. Многофункциональный синтез ЦФ с учётом требований их практической реализуемости возможен в настоящее время только методами нелинейного математического программирования [1–5].

Математическое программирование – это инвариантная и эффективная методология решения формализованных задач (задач проектирования в частности), максимально ориентированная на существующие вычислитель-

ные системы. Общая идея математического программирования состоит в привязке решения задачи к чётко выраженному инвариантному математическому признаку – экстремуму функции качества синтезируемого объекта $F(\boldsymbol{X})$, зависящей от вектора $\boldsymbol{X}$ искомых параметров синтезируемого устройства. Функцию $F(\boldsymbol{X})$ называют целевой функцией. Для любой проектной задачи такую функцию в общем виде всегда можно сформировать, исходя из требуемого функционирования устройства. В компьютерных пакетах обычно это делает функциональный редактор.

При известной целевой функции, решение задачи синтеза сводится к процедуре минимизации $F(\boldsymbol{X})$, т.е. к отысканию координат глобального экстремума, соответствующего оптимальным параметрам устройства $\boldsymbol{X}^{\mathbf{O}}$. Такая задача обычно решается поисковыми методами [2, 5].

Другим не менее важным требованием к цифровому фильтру является выполнение цифровой фильтрации в реальном времени. Это означает, что все операции алгоритма фильтрации должны выполняться за время, не превышающее период дискретизации входного сигнала. Поэтому особый интерес представляет случай целочисленного квантования пространства параметров фильтра. В этом случае задача математического программирования трансформируется в задачу целочисленного нелинейного программирования (ЦНП). При этом целевая функция $F(\boldsymbol{IX})$ является функцией целочисленных параметров $\boldsymbol{IX}$ и наиболее часто формируется в виде аддитивной свёртки

$$F(\boldsymbol{IX})=\sum_i \beta_i \cdot f_i(\boldsymbol{IX}) \qquad (1.1)$$

частных целевых функций $f_i(\boldsymbol{IX})$, в которые заложены функциональные требования к частотным свойствам фильтра [3, 4]. При этом синтез можно выполнять по одному из трёх критериев:

1) по критерию минимума среднеквадратичного отклонения в ненормированной форме

$$f_i(\boldsymbol{IX}) = \frac{1}{p} \cdot \sum_{n=1}^{p} \left[Y_n(\boldsymbol{IX}) - Y_n^{\mathrm{T}} \right]^2 \quad ; \qquad (1.2)$$

2) по критерию минимума среднеквадратичного отклонения в нормированной форме

$$f_i(\boldsymbol{IX}) = \frac{1}{p} \cdot \sum_{n=1}^{p} \left[\frac{Y_n(\boldsymbol{IX}) - Y_n^{\mathrm{T}}}{Y_n^{\mathrm{T}}} \right]^2 \quad ; \qquad (1.3)$$

а также

3) в виде минимаксного критерия

$$f_i(\boldsymbol{IX}) = \max_n \left\{ | Y_n(\boldsymbol{IX}) - Y_n^{\mathrm{T}} |^2 \right\} . \qquad (1.4)$$

Коэффициенты β_i в (1.2) и (1.3) задают значимость (вес) i-х частных характеристик $f_i(\boldsymbol{IX})$ (i-го частотного окна); $Y_n(\boldsymbol{IX})$ является текущим значением частной характеристики на n-ой дискретной частоте диапазона определения, а Y_n^{T} – её требуемым значением. При решении задачи синтеза целевые функции $f_i(\boldsymbol{IX})$ формируются посредством встроенного а программу синтеза функционального редактора.

Сконструированные таким образом цифровые фильтры (ЦНП-фильтры) обеспечивают максимальное качество фильтрации с учётом заданной совокупности требуемых характеристик при гарантированном целочисленном решении $\boldsymbol{IX}^{\mathrm{O}}$, т.е. при использовании только целочисленных операций для вычисления отклика фильтра на входное воздействие. Это обстоятельство определяет, во-первых, минимизацию времени формирования отклика фильтра, поскольку целочисленные операции выполняются быстрее операций с плавающей точкой, а во-вторых – принципиальную возможность реализации ЦНП-фильтров не только на специализированных цифровых процессорах сигналов (ЦПС) (DSP – *Digital Signal Processor*), но и на простых микроконтроллерах, в которых нет возможности вычислений в формате с плавающей точкой.

1.2. Построение ЦНП-фильтров

Цифровой фильтр является дискретной линейной системой, для которой соотношение между входной $\boldsymbol{x_n}$ и выходной $\boldsymbol{y_n}$ (рис. 1.1)

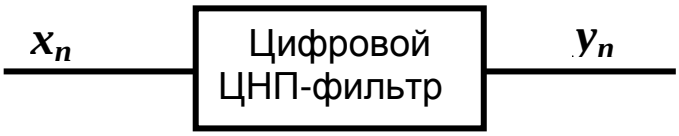


Рис. 1.1. Цифровой фильтр

временными последовательностями определяется разностным уравнением

$$y_n = -\sum_{k=1}^{N} \frac{a_k}{a_0} \cdot y_{n-k} + \sum_{k=0}^{N} \frac{b_k}{a_0} \cdot x_{n-k} , \qquad (1.5).$$

Принципиальной особенностью ЦНП-фильтров, как уже было отмечено выше, является принадлежность его коэффициентов b_k и a_k знакопеременному ряду целых чисел, который может быть как натуральным, так и биномиальным (для нормирующего коэффициента a_0). Интервал изменения ко-

эффициентов определяется разрядностью R используемого микропроцессора. Уравнение (1.5) можно трактовать и как расчетный алгоритм, в котором задержанные величины входной и выходной последовательностей умножаются на постоянные целочисленные коэффициенты b_k и a_k соответственно и результаты умножения суммируются.

По своему построению цифровые фильтры могут быть реализованы как по рекурсивной, так и по нерекурсивной схемам.

1.3. Рекурсивные ЦНП-фильтры

Фильтры, которые описываются полным разностным уравнением (1.5), принято называть **рекурсивными** цифровыми фильтрами, так как в вычислении текущих выходных значений участвуют не только входные данные, но и значения выходной последовательности, вычисленные в предшествующих циклах расчетов.

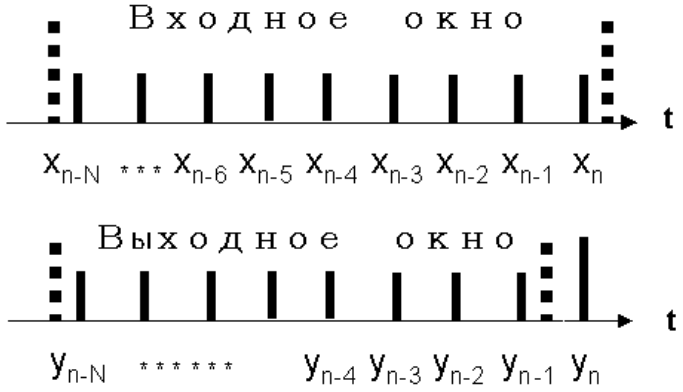


Рис. 1.2. Входное и выходное окно рекурсивного фильтра

Интервал суммирования по k получил название «окна» фильтра. Входное окно фильтра составляет $N+1$ отсчётов (рис. 1.2), выходное – N отсчётов. При этом значение N определяет порядок рекурсивного ЦНП-фильтра. Наличие обратной связи (рекурсии) в (1.5) определяет бесконечный характер импульсной характеристики фильтра. Поэтому рекурсивные фильтры часто называют БИХ-фильтрами или в английской транскрипции IIR-фильтрами (*Infinite Impulse Response*). Применив к (1.5) Z-преобразование получаем частотную характеристику фильтра в виде

$$H(e^{j\omega}) = A\frac{\prod_{i=1}^{N}(1 - z_i e^{-j\omega})}{\prod_{i=1}^{N}(1 - p_i e^{-j\omega})}.$$

Свойства коэффициента передачи $H(e^{j\omega})$ в значительной степени зависят от распределения полюсов и нулей в комплексной *Z*-плоскости. В частности, от распределения нулей и полюсов зависит устойчивость фильтра. Если система устойчива, то все полюса p_i должны лежать внутри единичного круга [3, 4, 6]. Таким образом, условие устойчивости рекурсивного фильтра может быть записано как система неравенств (функциональных ограничений) по всем полюсам коэффициента передачи в *Z*-плоскости:

$$\left|Z_{p_i}\right| < 1. \qquad (1.6)$$

Линейно-разностное уравнение (1.5) соответствует прямой форме аппаратной реализации фильтра. Однако для качественной нормировки всей совокупности требуемых частотных характеристик прямая форма наименее выгодна, т.к. одним нормирующим коэффициентом a_0 этого сделать обычно не удаётся. Для ЦНП-фильтров (как рекурсивного, так и нерекурсивного) наиболее выгодной является последовательная форма построения в виде каскадного включения звеньев второго порядка (рис. 1.3).

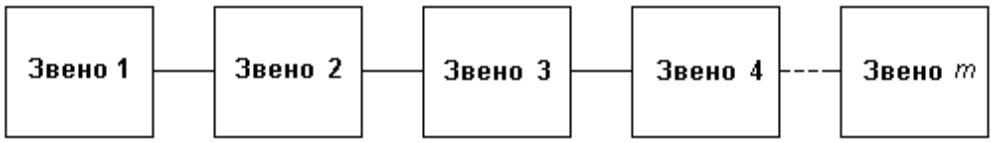


Рис. 1.3. Каскадная (последовательная) форма построения ЦНП-фильтра

Передаточная функция для рекурсивного ЦНП-фильтра, состоящего из каскадного соединения *m*-звеньев второго порядка (*m* = *N*/2), имеет следующий вид:

$$H(z) = \prod_{i=1}^{m} \frac{b_{0i} + b_{1i} z^{-1} + b_{2i} z^{-2}}{a_{0i} + a_{1i} z^{-1} + a_{2i} z^{-2}} \ , \qquad (1.7)$$

где комплексная переменная $z = e^{j\omega}$, $\omega = 2\pi f/F_д$ – цифровая частота, а $F_д$ – частота дискретизации.

Форма (1.7) операторного коэффициента передачи позволяет производить его фрагментацию с отдельной нормировкой совокупности частотных характеристик, соответствующих отдельным фрагментам (звеньям) фильтра. Обусловлено это тем, что нормирующие коэффициенты a_{0i} можно ввести в каждое *i*-ое звено. Из соотношения (1.7) легко получается разностное уравнение для одного звена рекурсивного ЦНП-фильтра:

$$y_n = \left(b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2} - a_1 y_{n-1} - a_2 y_{n-2}\right) / a_0 \qquad (1.8)$$

Как видно из (1.8), при вычислении отклика фильтра должна выполняться операция деления на целочисленный коэффициент a_0, которая может быть реализована операцией сдвига при условии принадлежности каждого i–го коэффициента биномиальному целочисленному ряду:

$$a_{0i} \in \left\{2^q\right\}, \quad q = \overline{0, R-1} \qquad i = \overline{1, m}\,. \tag{1.9}$$

На рис. 1.4 приведена типичная структура звеньев рекурсивного ЦНП-фильтра, соответствующая уравнению (1.9). Как видно, при аппаратной его реализации для вычисления отклика фильтра y_n кроме традиционных операций сложения, умножения и задержки на такт присутствует операция сдвига на $B=log_2 a_0$ бит, с помощью которой, как уже сказано, реализуется целочисленное деление на биномиальный нормирующий коэффициент a_0.

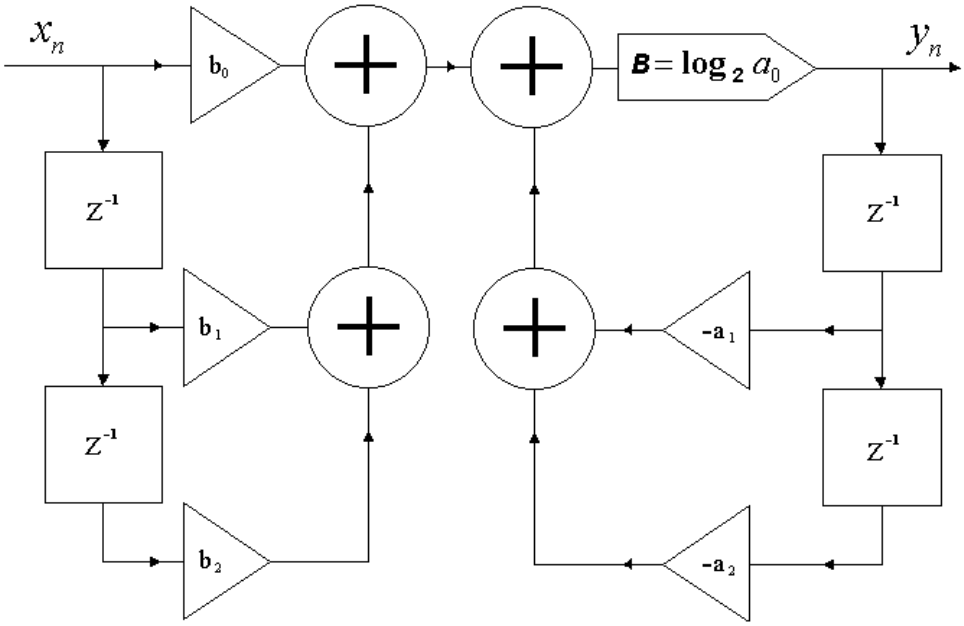


Рис. 1.4. Структура звена рекурсивного ЦНП-фильтра

Задача целочисленного нелинейного программирования при машинном синтезе рекурсивного фильтра записывается так [3]:

$$F^{o}(\boldsymbol{IX}^{\mathbf{0}}) = \min F(\boldsymbol{IX}) \qquad \boldsymbol{IX} \in I^{6m} \tag{1.10}$$

$$\begin{aligned} -2^{R-1} \le a_{di} \le 2^{R-1}-1 \quad & d=\overline{1,2} \;\; i=\overline{1,m} \\ -2^{R-1} \le b_{di} \le 2^{R-1}-1 \quad & d=\overline{0,2} \;\; i=\overline{1,m} \end{aligned}, \tag{1.11}$$

$$a_{0i} \in \{2^q\} \quad q = \overline{0, R-1} \ , \tag{1.12}$$

$$\left|Z_{pi}\right| < 1 \quad i = \overline{1, m} \ , \tag{1.13}$$

$$(|b_{0i}| + |b_{1i}| + |b_{2i}| + |a_{1i}| + |a_{2i}|) < 2^{R-1} \quad i = \overline{1, m} \ , \tag{1.14}$$

где m – число звеньев второго порядка, d – индекс коэффициента передаточной функции звена (1.7), R – разрядность микропроцессора.

Экстремальная задача синтеза (1.10) является общей формулировкой для целочисленного пространства I^{6m} параметров (коэффициентов фильтра), размерностью $6m$. Ограничения (1.11) задают границы изменения этих целочисленных коэффициентов, а соотношение (1.12) определяет принадлежность коэффициентов a_{0i} биномиальному ряду (1.9). С помощью функциональных ограничений (1.13) контролируется в процессе синтеза условие устойчивости (1.6) рекурсивного фильтра по всем полюсам коэффициента передачи, а ограничения (1.14) определяют выполнение требований, связанных с конечной разрядностью производимых процессором целочисленных операций. Многофункциональное задание целевой функции определяется соотношением (1.1), хотя здесь могут быть применены и другие известные способы скаляризации векторных задач [2, 5]. Поисковое итеративное решение экстремальной задачи ЦНП (1.10) в заданном пространстве параметров осуществляет программный алгоритмический комплекс [4, 5], обращаясь к модельному блоку программы для расчёта текущих функциональных характеристик фильтра. Вектор $\boldsymbol{IX}^{\mathbf{O}}$, минимизирующий скалярную целевую функцию $F(\boldsymbol{IX})$ на множестве допустимых целочисленных решений (1.11) и (1.12), является эффективным решением задачи параметрического синтеза рекурсивного ЦНП-фильтра.

1.4. Нерекурсивные ЦНП-фильтры

В нерекурсивных ЦНП-фильтрах текущие значения выходной последовательности y_n вычисляются через прямую линейную свёртку

$$y_n = \sum_{k=0}^{N-1} \frac{b_k}{a_0} x_{n-k} \ . \tag{1.15}$$

значений x_i входных отсчётов с импульсной характеристики b_i, имеющей конечное число N отсчётов. Поэтому такие фильтры называются фильтрами с конечной импульсной характеристикой (КИХ, или FIR – F*inite Impulse Response* фильтрами). Такие фильтры всегда устойчивы. Входное окно КИХ-фильтра (рис. 1.5) составляет N отсчётов, при этом значение N определяет порядок нерекурсивного фильтра.

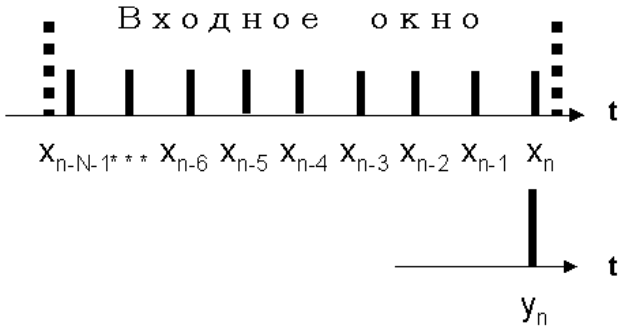


Рис. 1.5. Входное окно нерекурсивного фильтра

Передаточная функция каскадного соединения *m*-звеньев второго порядка нерекурсивного ЦНП-фильтра может быть записана так:

$$H(z)=\prod_{i=1}^{m}\frac{b_{0i}+b_{1i}z^{-1}+b_{2i}z^{-2}}{a_{0i}} \quad . \tag{1.16}$$

Соотношение (1.16) также позволяет отнормировать требуемую совокупность частотных характеристик фильтра и квантовать значения коэффициентов по целочисленному ряду значений. Уравнение для одного звена нерекурсивного ЦНП-фильтра имеет вид:

$$y_n=\left(b_0x_n+b_1x_{n-1}+b_2x_{n-2}\right)/a_0 \ ,$$

где коэффициент a_0 во всех *m* звеньях по-прежнему принадлежит биномиальному целочисленному ряду (1.9). На рис. 1.6 приведена типичная структура звеньев нерекурсивного ЦНП-фильтра.

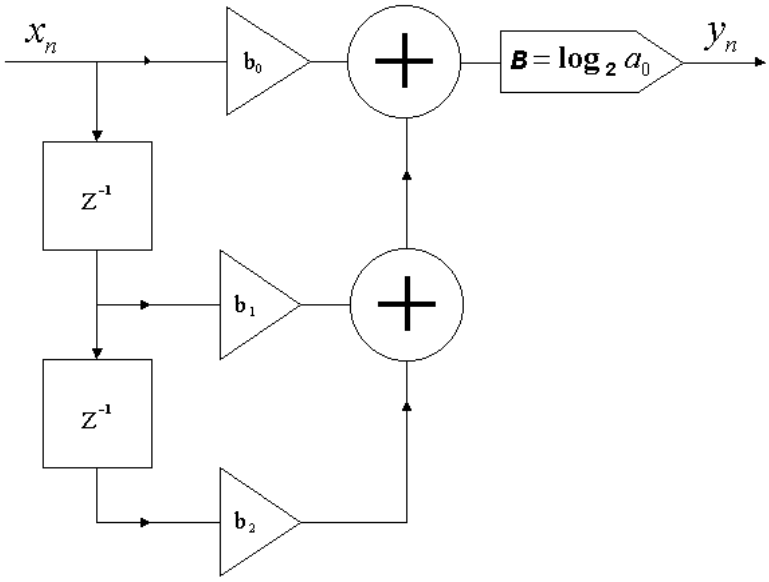


Рис. 1.6. Структура звена нерекурсивного ЦНП-фильтра

Постановка задачи целочисленного нелинейного программирования для машинного синтеза нерекурсивного ЦНП-фильтра выглядит следующим образом:

$$F^{O}(\boldsymbol{IX}^{\mathbf{0}}) = \min F(\boldsymbol{IX}) \qquad (1.17)$$

$$\boldsymbol{IX} \in I^{4m}$$

$$-2^{R-1} \le b_{di} \le 2^{R-1} - 1 \quad d=\overline{0,2} \;\; i=\overline{1,m}\,, \qquad (1.18)$$

$$a_{0i} \in \{2^{q}\} \quad q = \overline{0, R-1}, \qquad (1.19)$$

$$(|b_{0i}| + |b_{1i}| + |b_{2i}|) < 2^{R-1} \;\; i=\overline{1,m}\,, \qquad (1.20)$$

где m – число КИХ-звеньев второго порядка, d – индекс коэффициента передаточной функции звена (1.16), R – разрядность микропроцессора.

Экстремальная задача ЦНП-синтеза (1.17) записана относительно целочисленного пространства коэффициентов фильтра I^{4m}, размерностью $4m$. Ограничения (1.18) задают границы изменения этих целочисленных коэффициентов, а соотношение (1.19) определяет принадлежность коэффициентов a_{0i} биномиальному ряду. Функциональные ограничения (1.20) определяют требования по буферной регистровой памяти микропроцессора. Многофункциональное задание целевой функции, как и в случае с рекурсивным фильтром, определяется соотношением (1.1). Вектор $\boldsymbol{IX}^{\mathbf{O}}$, минимизирующий целевую функцию $F(\boldsymbol{IX})$ на множестве допустимых решений (1.18) и (1.19), является эффективным решением задачи параметрического синтеза нерекурсивного ЦНП-фильтра.

2. Алгоритм быстрого преобразования Фурье

Другая задача данной лабораторной работы – спектральная обработка сигнала с использованием быстрого преобразования Фурье (БПФ), выполняемого на цифровых процессорах сигналов, способных выполнять операции с плавающей точкой.

Быстрое преобразование Фурье является разновидностью дискретного преобразования Фурье (ДПФ), оптимизированного с точки зрения числа и скорости выполнения вычислительных операций. Наиболее распространены алгоритмы БПФ по основанию 2. Дискретное преобразование Фурье определяется выражением

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j(2\pi/N)nk}, \quad k = 0,1,\ldots,N-1, \qquad (2.1)$$

и применяется в отношении конечной последовательности $x(n)$ $(0 \le n \le N-1)$. Выражение (2.1) можно представить в более удобной форме:

$$X(k) = \sum_{n=0}^{N-1} x(n) W^{nk} , \qquad (2.2)$$

где $W = e^{-j2\pi/N}$. Коэффициенты W^{nk} в (2.2) представляют периодическую в зависимости от n и k последовательность с периодом N, т.е.

$$W^{(n+mN)(k+lN)} = W^{nk}; \; m, l = 0, \pm1, \pm2, \ldots \qquad (2.3)$$

Из (2.1) следует, что дискретное преобразование Фурье комплексной последовательности $x(n)$ требует $(N-1)^2$ комплексных операций умножения и $N(N-1)$ комплексных операций сложения. При достаточно больших N для вычисления ДПФ необходимо чрезмерно большое количество вычислительных операций. В быстром преобразовании Фурье объём вычислений значительно меньше. Уменьшение количества выполняемых операций достигается разбиением исходной N-точечной последовательности на две более короткие последовательности, ДПФ которых комбинируются таким образом, чтобы получилось ДПФ исходной N-точечной последовательности. Такое разбиение удобно делать тогда, когда N является чётным числом. Каждая из полученных после разбиения $N/2$-точечных последовательностей требует около $(N/2)^2$, а результирующее N-точечное ДПФ около $(N/2)^2 \cdot 2 = N^2/2$ операций умножения. Следовательно, число операций умножения сокращается почти вдвое. Выигрыш в два раза является приближённым, поскольку не учитывается, каким образом из ДПФ меньшего размера образуется искомое N-точечное ДПФ.

Процедуру разбиения можно продолжить и применить её сначала в отношении $N/2$-точечных, затем в отношении $N/4$-точечных последовательностей и т.д. вплоть до получения последовательностей с минимальным (равным 2) числом точек.

Применим описанный выше способ разбиения N-точечной последовательности $x(n)$, полагая N равным степени 2. Введём $N/2$-точечные последовательности

$$\begin{aligned} x_1(n) &= x(2n), \; n = 0, 1, \ldots, \frac{N}{2} - 1, \\ x_2(n) &= x(2n+1), \; n = 0, 1, \ldots, \frac{N}{2} - 1, \end{aligned} \qquad (2.4)$$

состоящие из чётных $x_1(n)$ и нечётных $x_2(n)$ членов $x(n)$. Тогда N-точечное ДПФ последовательности $x(n)$ можно представить в виде суммы

$$X(k) = \sum_{\substack{n=0 \\ n\,\text{чётные}}}^{N-1} x(n) W_N^{nk} + \sum_{\substack{n=0 \\ n\,\text{чётные}}}^{N-1} x(n) W_N^{nk} = \qquad (2.5)$$

$$= \sum_{n=0}^{N/2-1} x_1(2n) W_N^{2nk} + \sum_{n=0}^{N/2-1} x_2(2n+1) W_N^{(2n+1)k}. \qquad (2.6)$$

С учётом того, что

$$W_N^2 = [e^{j(2\pi/N)}]^2 = e^{j[2\pi/(N/2)]} = W_{N/2}, \qquad (2.7)$$

выражение (3.6) можно представить в виде

$$X(k) = \sum_{n=0}^{N/2-1} x_1(n) W_{N/2}^{2nk} + W_N^k \sum_{n=0}^{N/2-1} x_2(n) W_{N/2}^{2nk}, \qquad (2.8)$$

$$X(k) = X_1(k) + W_N^k X_2(k), \qquad (2.9)$$

где $X_1(k)$ и $X_2(k)$ равны $N/2$-точечным ДПФ последовательностей $x_1(n)$ и $x_2(n)$. Из формулы (2.9) следует, что N-точечное ДПФ $X(k)$ может быть разложено на два $N/2$-точечных ДПФ, результаты которых объединяются согласно (2.9).

Поскольку $X(k)$ определено при $0 \le k \le N-1$, то $X_1(k)$ и $X_2(k)$ определены при $0 \le k \le N/2-1$. Поэтому необходимо доопределить формулу (2.9) для $k \ge N/2$:

$$X(k) = \begin{cases} X_1(k) + W_N^k X_2(k), \ 0 \le k \le \dfrac{N}{2} - 1, \\ X_1(k - \dfrac{N}{2}) + W_N^k X_2(k - \dfrac{N}{2}), \ \dfrac{N}{2} \le k \le N-1. \end{cases} \qquad (2.10)$$

Рассмотренная схема вычислений может быть использована для расчета $N/2$-точечных ДПФ в соответствии с формулами (2.9) и (2.10). Каждая из последовательностей $x_1(n)$ и $x_2(n)$ разбивается на две последовательности, состоящие из чётных и нечётных членов.

Описанный выше алгоритм был назван алгоритмом **с прореживанием по времени**, поскольку на каждом этапе входная последовательность разделяется на две обрабатываемые последовательности меньшей длины, т.е. входная последовательность прореживается на каждом этапе.

Базовая операция алгоритма вычисления БПФ с прореживанием по времени состоит в том, что два входных числа A и B объединяются для получения двух выходных чисел X и Y следующим образом:

$$\begin{aligned} X &= A + W_N^k B, \\ Y &= A - W_N^k B. \end{aligned} \qquad (2.11)$$

В литературе для обозначения этой операции используется термин «бабочка».

Есть другая распространённая форма алгоритма БПФ – алгоритм БПФ **с прореживанием по частоте**. Однако в данной работе реализация этого алгоритма на цифровом процессоре сигналов не предусмотрена.

3. Учебная программа анализа и синтеза цифровых фильтров

Используемое для анализа и синтеза цифровых фильтров программное обеспечение позволяет разрабатывать рекурсивные и нерекурсивные цифровые фильтры различного порядка в широкой области допустимых изменений целочисленных параметров (коэффициентов) фильтра, проводить подробный анализ полученного оптимального решения в частотной области, выводить на печать графики частотных характеристик синтезированного фильтра, а также формировать файл протокола решения текущей задачи синтеза. Блок-схема компьютерной программы приведена на рис. 3.1.

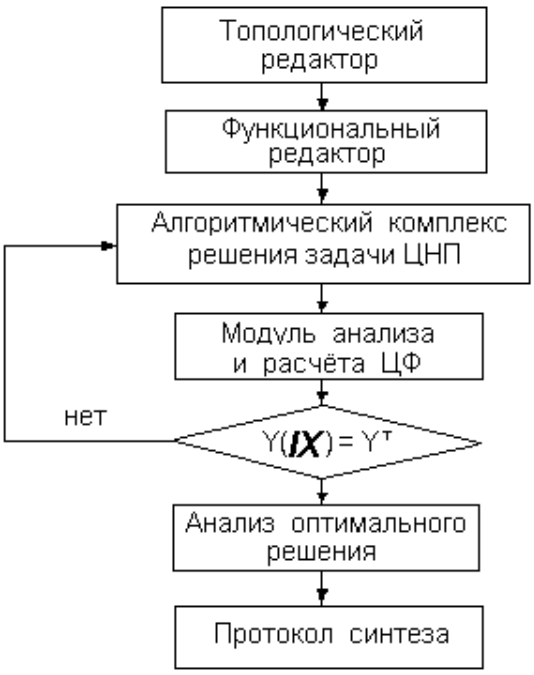


Рис. 3.1. Блок-схема учебной программы синтеза

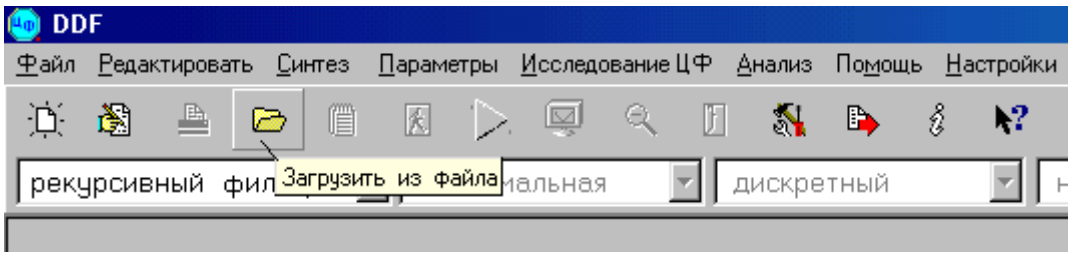
Рассмотрим взаимодействие основных модулей программы в контексте выполнения основных этапов решения конкретной задачи синтеза цифрового фильтра.

На первом этапе формируется типовой топологический файл задания на синтез name.top, имя которого определяет пользователь. Для выполнения конкретного задания к лабораторной работе в программе сформированы типовые файлы заданий на синтез ЦНП-фильтров:

– нерекурсивного второго порядка (файл FIR_2p.top),

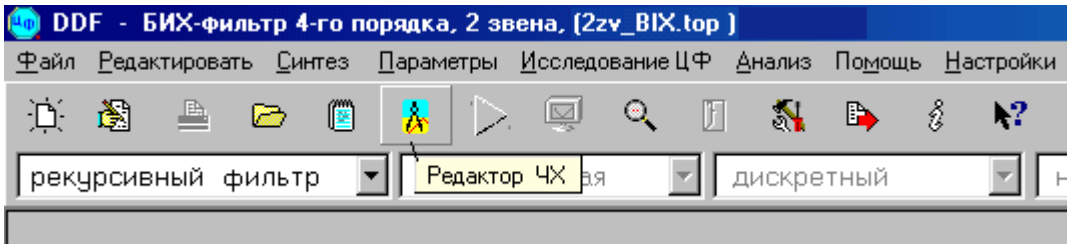
– нерекурсивного четвёртого порядка (файл FIR_4p.top),
– рекурсивного фильтра второго порядка (файл IIR_2p.top) и
– рекурсивного фильтра четвёртого порядка (файл IIR_4p.top).

В перечисленных файлах содержится описание структуры синтезируемого фильтра, определяются границы изменения его варьируемых коэффициентов и их начальное значения, указывается порядок и тип синтезируемого фильтра. Формирование и редактирование файла задания на синтез осуществляет топологический редактор программы, вызов которого осуществляется соответствующей «горячей» кнопкой основного меню программы. После редактирования файл задания необходимо загрузить в программу, после чего начинается её выполнение.



Загруженные исходные данные можно просмотреть на панели варьируемых параметров.

На втором этапе необходимо ввести требуемые характеристики синтезируемого фильтра и сформировать целевую функцию в форме (1.1). Для этого служит графический редактор функциональных характеристик (функциональный редактор), который необходимо вызвать из основного меню программы, нажав кнопку «Редактор ЧХ»

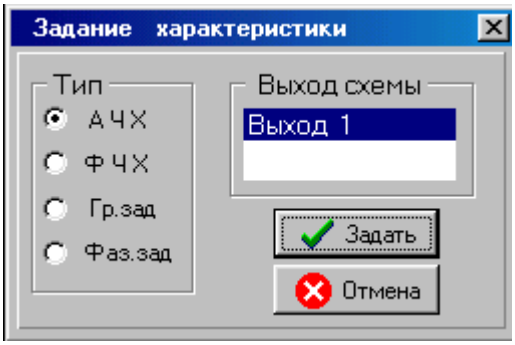


На переднюю панель функционального редактора выведены:
– графическое поле ввода графика требуемой характеристики фильтра;
– элементы управления, задающий тип характеристики и номер частотного окна;
– поля ввода веса характеристики и допустимой её неравномерности;
– элемент управления для задания типа сходимости в каждой точке оцифровки;
– элементы графического редактирования текущего окна редактора.

Для ввода требуемых характеристик фильтра и при работе в функциональном редакторе рекомендуется следующая последовательность действий:

1. Задать число используемых функциональных окон (вводимых частотных характеристик синтезируемого фильтра), используя указанный на рисунке инструмент.

2. Задать тип частотной характеристики окна – АЧХ, ФЧХ, групповая или фазовая задержка.

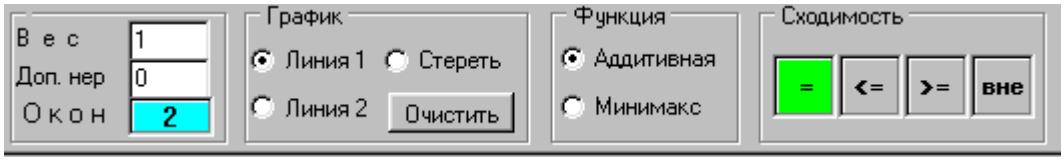


3. Задать границы и шкалу частот функционального окна согласно выбранной размерности (Гц, кГц или МГц). Задать вид частотной шкалы – линейная (ЛИН) или логарифмическая (ЛОГ).

4. Задать границы функционального окна по вертикали (по оси Y), а также размерность и тип шкалы (например, для АЧХ, линейная ЛИН или в децибелах ДБ).

5. В режиме «Линия 1» задать кусочно-линейную аппроксимацию требуемой частотной характеристики фильтра путём построения графика, используя левую клавишу манипулятора «мышь». График требуемой характеристики должен занимать весь установленный для оси частот диапазон и не должен иметь разрывов. Для ввода сложных графиков рекомендуется использовать шаблоны характеристик функционального редактора.

6. Задать типы частных критериев (типы сходимости текущей характеристики к требуемой) для каждой подлежащей оцифровке точки частотной характеристики. Типы критериев задаются цветом с помощью кнопочной панели СХОДИМОСТЬ.



7. Сформировать целевую функцию окна в аддитивной (1.2) или минимаксной (1.4) форме (задаётся с помощью кнопочной панели ФУНКЦИЯ).

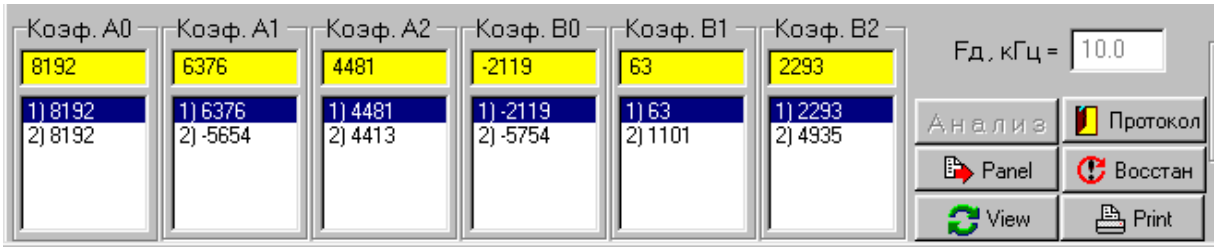
8. Указать веса текущего функционального окна в поле «Вес».

9. Запустить процесс оцифровки введённой частотной характеристики путём нажатия кнопки «Оцифровка». Оцифрованную частотную характеристику рекомендуется сохранить на диске, используя панель ФАЙЛ.

Приведённую выше последовательности действий необходимо выполнить для всех частотных характеристик, соответствующих числу заказанных в п.1 окон. После этого нажатием кнопки «Выход» завершается работа в функциональном редакторе.

На третьем этапе программный алгоритмический комплекс осуществляет поисковое итеративное решение экстремальной задачи ЦНП-синтеза (1.10) или (1.17) в заданном пространстве целочисленных варьируемых коэффициентов фильтра [3-5], обращаясь к модельному блоку программы для расчёта текущих функциональных характеристик фильтра по заданной его модели. Старт синтеза осуществляется нажатием соответствующей «горячей» кнопкой основного меню программы.

На четвёртом этапе (после нажатия кнопки «Анализ» основного меню) осуществляется исследование найденного эффективного решения задачи ЦНП-синтеза в модуле анализа пакета с построением графиков всех характеристик цифрового фильтра, их распечаткой и формированием стандартного протокола решения задачи синтеза.



В приложении 1 приведён пример протокола синтеза полосового БИХ-фильтра четвёртого порядка с указанием его основных функциональных показателей и оптимальных параметров – целочисленных коэффициентов полосового фильтра. В протоколе также приводятся эти коэффициенты в формате, необходимом для программирования микропроцессорного контроллера или сигнального процессора.

4. Описание цифровых платформ на базе сигнальных процессоров TMS320F28335 и ADSP-21364

4.1. Краткое описание устройств цифровой обработки сигналов

В качестве обработчиков сигналов в данной лабораторной работе используются цифровой сигнальный контроллер (ЦСК, DSC – Digital Signal Controller) TMS320F28335 из семейства C2000 фирмы Texas Instruments и цифровой сигнальный процессор (ЦСП, DSP – Digital Signal Processor) ADSP-21364 из семейства ЦСП ADSP-2136x фирмы Analog Devices. Оба процессора в своём составе помимо вычислительных устройств имеют необходимые устройства ввода/вывода, постоянную, перепрограммируемую и оперативную память.

Следует отметить, что ЦСК TMS320F28335, по сути, обладает функциями и возможностями сигнального процессора, т.к. позволяет выполнять высокоскоростную цифровую обработку сигналов. Термин «контроллер» по отношению к TMS320F28335 применяется потому, что он предназначен для применения в различных системах управления (управление двигателями, сложными энергетическими установками, системами автомобиля). Сочетание большого количества внешних интерфейсов и достаточно мощного вычислительного ядра позволяет обойтись без применения дополнительного сигнального процессора во многих задачах, где требуется обрабатывать большое количество данных в режиме реального времени с использованием различных математических операций. Сама фирма-производитель во многих инструкциях по применению TMS320F28335 часто называет его цифровым сигнальным процессором.

Целевым назначением ЦСП ADSP-21364 является обработка аудиосигналов в режиме реального времени.

Микроконтроллеры TMS320F28335 имеют встроенный аналого-цифровой преобразователь. Использование же ЦСП ADSP-21364 предполагает подключение внешнего устройства для оцифровки аналогового сигнала. Таким устройством в данной лабораторной работе является аудио кодек AD1835A фирмы Analog Devices. Кодек AD1835A включён в состав целевой микросистемы (в состав отладочной платы, содержащей ЦСП ADSP-21364) и связь с ним осуществляется через выделенный для работы с ним последовательный интерфейс.

4.2. Описание отладочных плат процессоров

На рис. 4.1 показан вид отладочной платы, содержащей сигнальный контролер TMS320F28335.

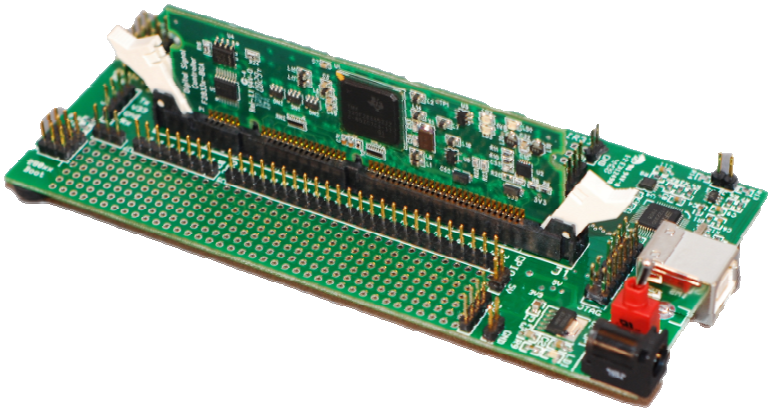


Рис. 4.1. Отладочная плата сигнального контроллера TMS320F28335

Помимо контролера TMS320F28335 плата содержит все необходимые для обеспечения работы контроллера элементы (стабилизатор напряжения питания, генератор тактовых импульсов, фильтры цепей питания и т.д.), USB-эмулятор интерфейса JTAG, разъём для подключения внешнего эмулятора JTAG, мост USB-UART, разъём для подключения внешнего источника питания, разъёмы для подключения к выводам микроконтроллера, разъём для подключения USB-кабеля.

На рис. 4.2 приведено изображение отладочной платы с сигнальным процессором ADSP-21364.

Рис. 4.2. Отладочная плата сигнального процессора ADSP-21364

Помимо процессора ADSP-21364 и элементов, обеспечивающих его функционирование, плата включает:

- флэш-память объёмом 1 Мбайт;
- флэш-память SPI объёмом 512 кбит;
- статическое ОЗУ (SRAM) объёмом 512 кбайт;
- микросхему стерео-кодека AD1835, включающую 2 АЦП и 4 ЦАП;
- разъёмы для ввода/вывода звуковых стерео-сигналов;
- USB-эмулятор интерфейса JTAG;
- разъём для подключения внешнего эмулятора JTAG;
- разъёмы для подключения к выводам микроконтроллера;
- 8 светодиодов общего назначения;
- 4 программируемых пользователем кнопки;
- разъём для подключения внешнего источника питания;
- разъём для подключения USB-кабеля.

4.3. Цифровой сигнальный контроллер TMS320F28335

Сигнальный контроллер TMS320F28335 имеет гарвардскую архитектуру и ядро (CPU – *Central Processor Unit*) типа C28x с двумя шинами для чтения программы и данных (рис. 4.3) и с отдельной шиной для записи данных. Вычислительные операции наряду с CPU выполняет интегрированный в кристалл контроллера сопроцессор с плавающей точкой FPU (*Floating Point Unit*).

Внутренняя память процессора имеет 16-разрядную организацию и состоит

- из постоянной памяти начальной загрузки Boot ROM (*Read only Memory*) объёмом 4К x 16,
- из памяти 1К x 16, загружаемой при включении (OTP – *On-time Programmable* ROM),
- из Flash-памяти 256К x 16
- и из разделённой на блоки *M*0 (1К x 16), *M*1 (1К x 16) и *L*0 – *L*7 (4К x 16) статической оперативной памяти (RAM – *Random Access Memory*).

Содержимое Boot ROM определяется производителем и не меняется в процессе работы сигнального контроллера. В Boot ROM находятся начальный загрузчик, используемый при программировании Flash-памяти, а также стандартные таблицы (такие как таблицы sin/cos), используемые при выполнении математических операций.

Во Flash-памяти размещается предназначенный для работы процессора программный код. При программировании OTP используются те же регистры, что и при программировании Flash-памяти.

Для адресации внутренней памяти используются

- 22-разрядная шина адреса памяти программ PAB (*Program Address Bus*),
- 32-разрядная шине адреса чтения данных DRAB (*Data-Read Address Bus*)
- и 32-разрядная шина адреса записи данных PWAB (*Program-Write Address Bus*).

Данные перемещаются также по 32-разрядным шинам:

- по шине чтения памяти данных DRDB (*Data-Read Data Bus*),
- по шине чтения памяти программ PRDB (*Program-Read Data Bus*)
- и по шине записи в память данных и в память программ DWDB (*Data-/Program Write Bus*).

Наличие такого количества шин позволяет одновременно за один цикл обращения к памяти выбирать инструкцию, читать и записывать данные.

Внешняя память и внешние устройства ввода/вывода подключаются к процессору через трёхшинную внешнюю магистраль, имеющую шину адреса (External address bus), шину данных (External data bus) и шину управления, по которой передаются сигналы управления памятью (Memory control) прерываниями (Interrupts) и инициализацией (Initialization). Внешние шины данных и адреса формируются путём мультиплексирования внутренних шин процессора.

Ядро C28x (рис. 4.3) содержит арифметическо-логическое устройство ALU с возможностью выполнения сдвиговых операций по входу (Input shift) и выходу (Output shift), умножитель (Multiplier) с возможностью выполнения операций сдвига к выходному результату (Product shift) и арифметический блок ARAU (*Address Register Arithmetic Unit*), предназначенный для вычисления адресов операндов, а также для формирования адресов ветвлений (переходов и вызовов).

ALU служит для выполнения арифметических и логических операций. В составе ALU имеется 32-разрядный аккумулятор ACC, составленный из двух 16-разрядных регистров AH и AL. Операциям ALU могут предшествовать операции сдвига. Операции сдвига могут быть применены также к результату работы ALU.

Блок ARAU работает с восемью 32-разрядными вспомогательными регистрами (AR – *Auxiliary Registers*) XAR0-XAR7, которые служат для поддержки режимов косвенной регистровой адресации с пред- или постинкрементацией/декрементацией, а также могут выполнять роль регистров общего назначения. Среди команд процессора, использующих косвенную адресацию через регистры XAR0-XAR7 есть такие, которые не требуют их явного указания. При использовании таких команд тот или иной из регистров XAR*n* выбирается с помощью специального регистра-указателя ARP (*Auxiliary Reg-*

ister Pointer), в который предварительно посредством команды загрузки заносится номер *n* нужного регистра XAR*n*.

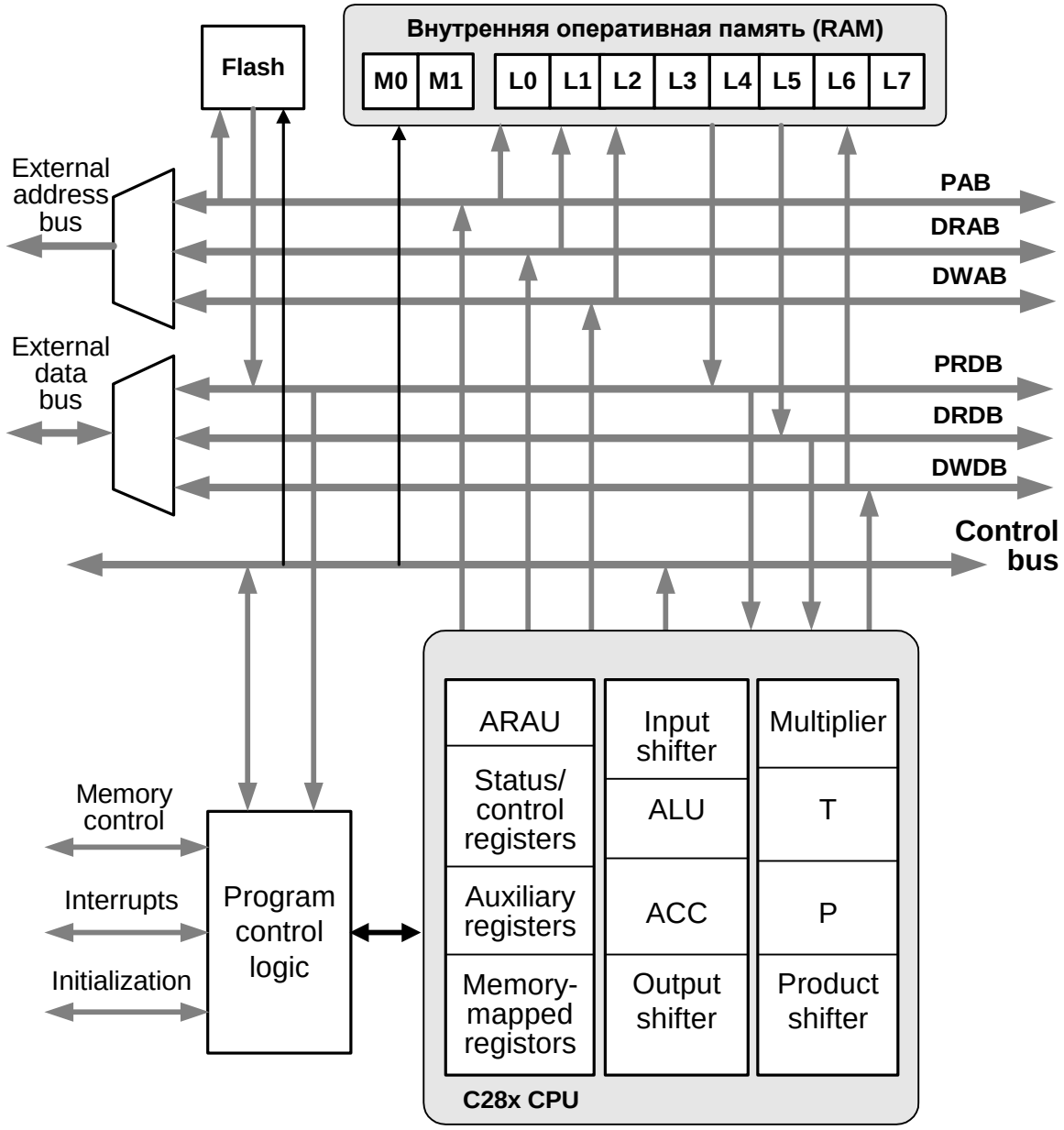


Рис. 4.3. Структура шин и состав ядра сигнального контроллера TMS320F28335

Есть особенность в использовании регистра XAR7. Заключаются она в его применении для косвенно-регистровой адресации данных, переходов и вызовов процедур в пределах всего 4M x 16 адресного пространства памяти программы. При косвенно-регистровых вызовах через регистр XAR7 его содержимое заносится в программный счётчик PC (*Program Counter*), а адрес возврата сохраняется в стеке. Для работы со стеком ARAU использует указатель стека SP (*Stack Pointer*).

Имеется разновидность вызовов, при которых адрес возврата сохраняется не в стеке, а в специальном регистре RPC (*Return PC Pointer*). При этом содержимое самого регистра RPC предварительно (до поступления команды вызова) сохраняется в стеке. При возврате программный счётчик восстанавливается по значению регистра RPC, а содержимое регистра RPC восстанавливается по значению, взятому из стека. При косвенных вызовах с использованием регистра RPC адрес вызова может содержаться в любом из вспомогательных регистров XAR*n*.

В командах процессора используются как 32-разрядные XAR0-XAR7, так 16-разрядые вспомогательные регистры AR0-AR7. Последние образованы младшими 16-ю разрядами регистров XAR0-XAR7. Регистры AR0-AR7 используются преимущественно в качестве регистров общего назначения, в частности при выполнении 16-битовых операций сравнения и как счётчики циклических операций.

В режиме прямой адресации, когда исполнительный адрес содержится в команде, применяется постраничная адресация памяти данных. В этом случае память данных разбивается на 64-словные страницы. На начало текущей страницы всегда показывает 16-разрядный **указатель страницы DP** (*Data Page*), и исполнительный адрес получается путём добавления к содержимому регистра DP 6-битового смещения, содержащегося в команде. Поэтому всегда перед прямым обращением по нужному адресу требуется предварительная установка соответствующего значения указателя страницы DP.

С помощью ARAU осуществляется также обращение к отображённым на память регистрам (Memory-mapped registers) процессора, в частности адреса портов устройств ввода/вывода (УВВ) как расположенных на кристалле, так и внешних по отношению к микросхеме процессора УВВ.

Умножитель выполняет 16- или 32-битовые операции умножения и умножения-накопления/вычитания с фиксированной точкой с 32- или 64-битовым результатом. При выполнении умножения 16 x 16 один из операндов берётся из одного из регистров общего назначения, из памяти данных или непосредственно из команды, а другой – из предварительно загруженного регистра множимого – регистра временного хранения XT (*Temporary Register*). Результат сохраняется в 32-разрядном регистре результата P (*Product Register*) и в аккумуляторе ACC. Регистр P может быть представлен в виде двух 16-разрядных регистров PH:PL. При умножении 32 x 32 один сомножитель поступает из предварительно загруженного регистра XT или из памяти программ, а другой – из памяти данных или из регистра процессора. 64-разрядный результат поочерёдно по 32 разряда извлекается из регистра результата P. Оба операнда в операциях умножения могут быть знаковыми и беззнаковыми, а также один из сомножителей может иметь знак, а другой –

быть числом без знака. Работа умножителя организована так, что к результату его работы могут быть легко применены операции сдвига.

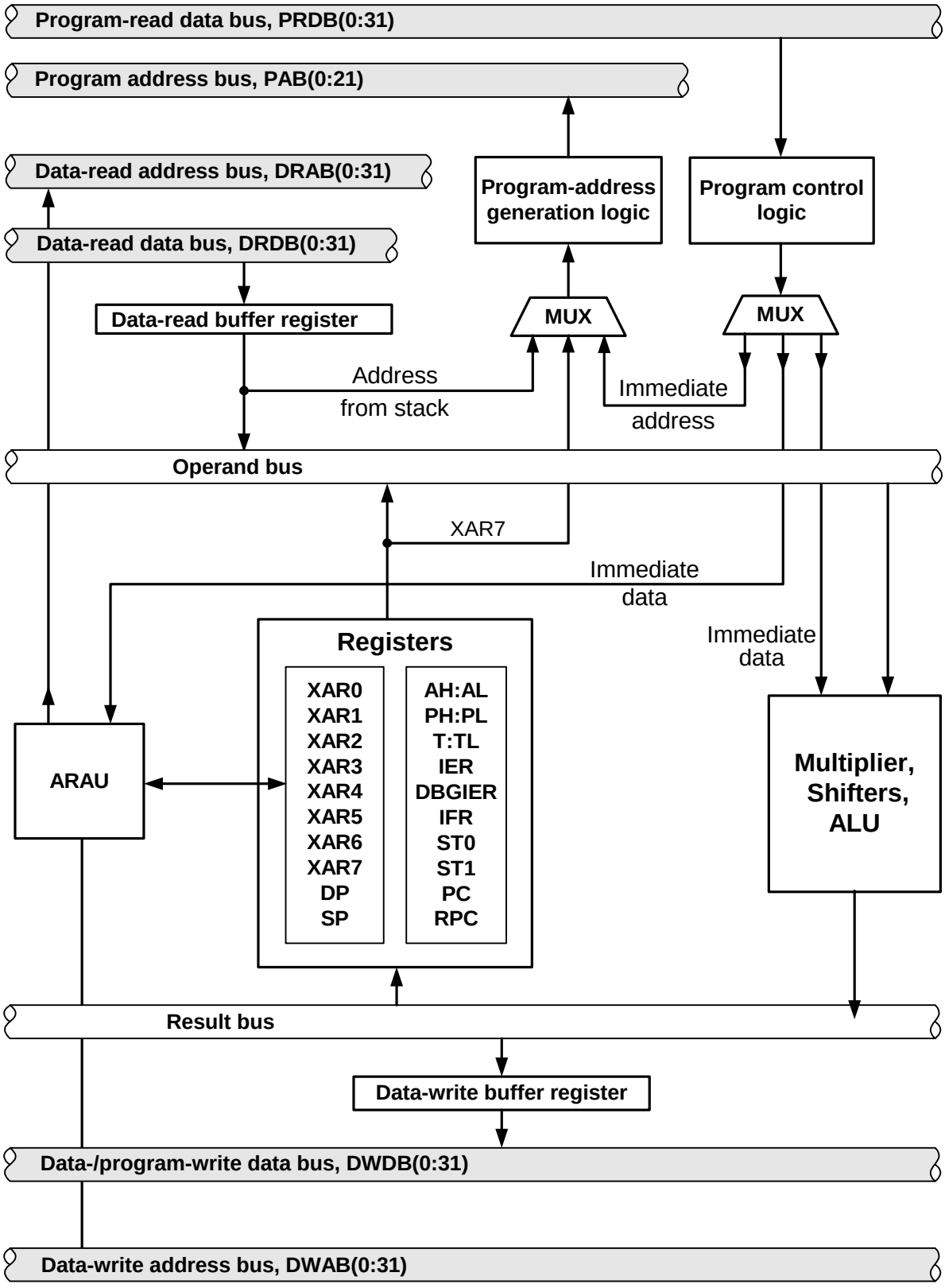


Рис. 4.4. Связь операционных блоков ядра с внутренними шинами процессора

Более подробно связь операционных блоков ядра процессора и его регистров с памятью показана на рис. 4.4. Важно то, что вычислительные компоненты ядра объединены двумя дополнительными шинами – шиной операндов (Operand bus) и шиной результатов (Result bus). Вычислительные устройства (ALU, умножитель и сдвигатель) работают:

- с непосредственными данными (Immediate data), взятыми из инструкций. Данные берутся из устройства PCL (Program control logic), которое управляет выполнением инструкций, выбранных из памяти программ под управлением генератора адресов программы PAGL (Program-address generation logic). Непосредственные данные в блок PCL поступают через мультиплексор MUX;
- с данными, взятыми из памяти программ (PM) или из памяти данных (DM). Такие данные передаются через буферный регистр Data-read buffer register при выполнении операций с данными из памяти.

Результат работы вычислительных устройств передаётся на шину Result bus и далее в один из регистров общего назначения (в блок Registers) или через буферный регистр Data-write read buffer register на шину DWDB записи данных в память.

Генератор адресов программы PAGL формирует адреса инструкций – адреса переходов, вызовов и возвратов из процедур, – используя значения регистров указателей XAR0-XAR7, содержимое стека и непосредственные адреса переходов/вызовов (Immediate address), прописанные в командах и поступающие из блока управления выполнением инструкций PCL.

Кроме упомянутых выше регистров, ядро C28x имеет два регистра состояния – *Status Registers* ST0 и ST1 и регистры управления прерываниями. К последним относятся:

- регистр IFR (*Interrupt flag register*) для приёма поступающих в устройство управления прерываниями PIE (*Peripheral Interrupt Expansion Block*) запросов,
- регистр IER (*Interrupt enable register*) для разрешения или запрещения восприятия отдельно выделенных запросов прерывания,
- регистр DBGIER (*Debug interrupt enable register*) для разрешения или запрещения обработки отдельно выделенных запросов прерывания в режиме отладки.

Упрощенная структура всего сигнального контроллера TMS320F28335 представлена на рис. 4.5. Все компоненты системы объединены двумя внутренними шинами, к которым относятся:

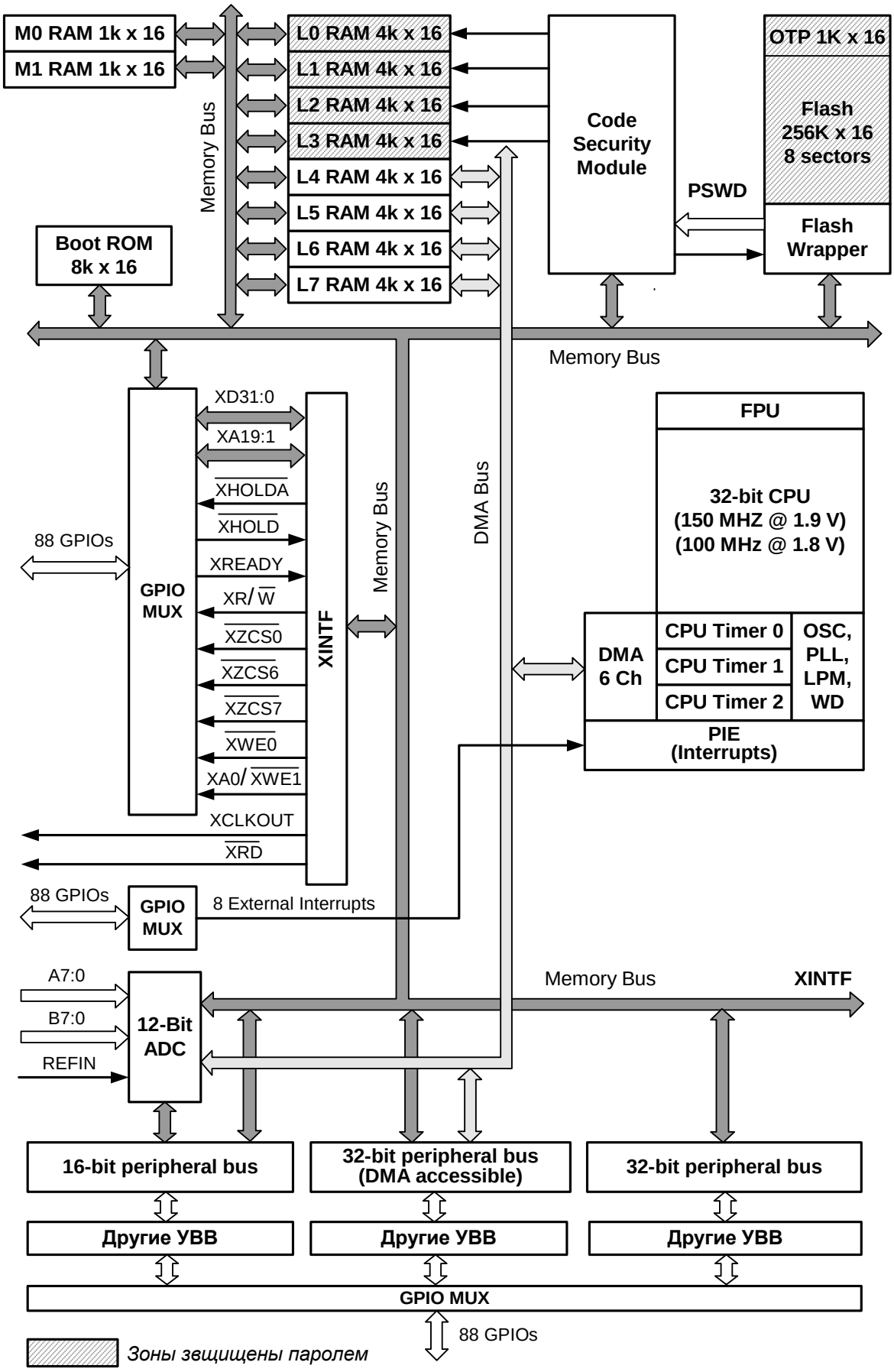


Рис. 4.5. Упрощенная структура сигнального контроллера TMS320F28335

1) шина Memory Bus, обеспечивающая связь ядра с внутренней памятью, а также связь с внешней памятью через внешний интерфейс XINTF (*External Interface*) и с внешними устройствами ввода/вывода через интерфейс с внешней 16-разрядной (16-bit peripheral bus) и внешней 32-разрядной (32-bit peripheral bus) шинами. Структура шины Memory Bus соответствует представленной на рис. 4.3 и рис. 4.4 гарвардской архитектуре процессора;
2) шина DMA Bus, которой управляет 6-канальный (6 ch) контроллер прямого доступа к памяти DMAC (*Direct Memory Access Controller*). Контроллер DMA обеспечивает ввод данных с прямым доступом к четырём внутренним блокам памяти *L*4-*L*7. В режиме прямого доступа вводятся данные от 12-разрядного аналого-цифрового преобразователя (12-bit ADC), а также осуществляется обмен данными с устройствами ввода/вывода, подключаемыми через выделенную для режима DMA внешнюю 32-разрядную шину (32-bit peripheral bus – DMA accessible).

Внешний интерфейс XINTF имеет 20 адресных линий XA[19:1], 32 линии данных XD[31:0] и три линии $\overline{\text{XZCS0}}$ (*External Interface zone* 0 *chip select*), $\overline{\text{XZCS6}}$ (*External Interface zone* 6 *chip select*) и $\overline{\text{XZCS7}}$ (*External Interface zone* 7 *chip select*) для селекции трёх выделенных во внешнем адресном пространстве зон (зоны 0 – 4K x 16, 6 – 1M x 16 и 7 – 1M x 16). Для каждой из зон устанавливаются свои временные характеристики сигналов управления и своё время ожидания готовности. Это относится, в частности, к времени ожидания, которое зависит от длительности сигнала XREADY, вырабатываемого предназначенным для этого генератором.

Длительность сигнала XREADY кратна периоду импульсов внешней синхронизации XCLKOUT и определяется содержимым соответствующего регистра процессора. Это позволяет работать с разными типами запоминающих устройств и с различными устройствами ввода/вывода.

В качестве управляющих сигналов при работе с внешней памятью используются сигналы (стробы) XR/$\overline{\text{W}}$ (*External Interface read, not write strobe*), $\overline{\text{XRD}}$ (*External Interface Read Enable*), $\overline{\text{XWE0}}$ (*External Interface Write Enable* 0) и $\overline{\text{XWE1}}$ (*External Interface Write Enable* 1). При 16-битовом обмене данными с внешней памятью используются все 20 линий шины адреса. При этом младший разряд адреса передаётся по линии XA0/$\overline{\text{XWE1}}$, которая предварительно конфигурируется под передачу по ней младшего разряда XA0. Для записи 16-разрядных слов обычно используется строб $\overline{\text{XWE0}}$, а для чтения – строб $\overline{\text{XRD}}$.

Обмен 32-разрядными словами происходит с использованием 19 старших разрядов внешней шины адреса. Чтение памяти происходит под управлением строба $\overline{\text{XRD}}$. Для записи в память применяются два строба $\overline{\text{XWE0}}$ и

$\overline{\text{XWE1}}$. Последний передаётся по соответствующим образом настроенной линии XA0/$\overline{\text{XWE1}}$. Необходимость в использовании второго строба записи $\overline{\text{XWE1}}$ обусловлена тем, что память имеет 16-разрядную организацию, а внешняя шина данных является 32-разрядной. Поэтому применение двух стробов $\overline{\text{XWE0}}$ и $\overline{\text{XWE1}}$ позволяет одновременно обращаться к двум 16-разрядным микросхемам памяти и таким образом за один цикл записывать в память 32-разрядное слово.

Строб XR/$\overline{\text{W}}$ предоставляет дополнительную возможность управления процессом чтения/записи.

При обмене данными с УВВ в режиме прямого доступа к памяти используются сигналы сигнал запроса захвата внешней шины $\overline{\text{XHOLD}}$ и сигнал подтверждения прямого доступа $\overline{\text{XHOLDA}}$ (*Hold Acknowledge*) устройству, реализующему передачу данных с прямым доступом к памяти. Таким устройством может быть внешний контроллер прямого доступа. При поступлении во внешний интерфейс XINTF активного уровня сигнала подтверждения захвата $\overline{\text{XHOLDA}}$, связанные с линиями XA[19:0], XD[31:0], $\overline{\text{XZCS0}}$, $\overline{\text{XZCS6}}$, $\overline{\text{XZCS7}}$, XR/$\overline{\text{W}}$, $\overline{\text{XWE0}}$, XA0/$\overline{\text{XWE1}}$ и $\overline{\text{XRD}}$ контакты микросхемы процессора переводятся в высокоимпедансное состояние. Тем самым предоставляется возможность управления внешней магистралью устройству, запросившему прямой доступ в внешней памяти.

Для подключения внешней памяти и внешних устройств ввода/вывода к микросхеме процессора используются 88 GPIO (*General Purpose Input/Output*) контактов общего назначения, способ использования которых определяется путём соответствующих программно реализуемого управления работой мультиплексора GPIO MUX и программной установки способа приёма/передачи информации через GPIO контакты.

В данной лабораторной работе из всех интегрированных в микросхему процессора устройств ввода/вывода используется только аналого-цифровой преобразователь (АЦП). В отличие от других УВВ входные сигналы на АЦП подаются не через мультиплексор GPIO MUX, а через отдельно выделенные для него контакты микросхемы A7:0 и B7:0. АЦП имеет два независимых устройства выборки-хранения, которые принимают сигналы от двух 8-канальных аналоговых мультиплексоров. Именно этим объясняется наличие двух групп контактов A7:0 и B7:0 для входных сигналов АЦП. Необходимое для аналого-цифрового преобразования опорное напряжение подаётся также через выделенный контакт REFIN.

Через контакты общего назначения принимаются запросы прерываний (8 External Interrupts) от внешних устройств. Поступившие запросы направляются в блок управления обработкой прерываний PIE.

Внешний доступ к OTP, к Flash-памяти и к четырём блокам *L*0-*L*3 оперативной памяти защищён паролем PSWD, который проверяется устройством Code Security Module, работающим совместно с устройством защиты Flash-памяти Flash Wrapper. Пароль проверяется всякий раз при попытке изменить или прочитать содержимое защищённой области памяти.

В микросхему процессора интегрированы сторожевой таймер WD (*Watch Dog*), три таймера CPU Timer 0-2, генератор внутренней синхронизации OSC (*Oscillator*) с системой фазовой автоподстройки частоты PLL (*Phase Locked Loop*), а также устройство управления энергосбережением LPM (*Low Power Module*) .

4.4. Цифровой сигнальный процессор ADSP-21364

Изучаемый в лабораторной работе микропроцессор (МП) ADSP-21364 относится к семейству ADSP-2136x цифровых процессоров сигналов (ЦПС), производимых фирмой Analog Devices. Процессоры имеют супергарвардскую архитектуру (*Super Harvard Architecture*) и являются разновидностью МП, работающих под управлением одного потока команд с множественным потоком данных (ОКМД; SIMD – *Single-instruction*, *Multiple-data*). Процессор построен как система на кристалле, в которой помимо ядра имеются быстродействующая память и многофункциональная подсистема ввода/вывода с возможностью конфигурации для работы с различными периферийными устройствами.

Ядро процессоров семейства ADSP-2136x

В состав ядра процессора (рис. 4.6) входят 5-уровневое конвейерное устройство управления последовательностью выборки команд УУПВК (5-stage Sequencer), кэш команд (Instruction Cache), два генератора адресов данных (DAG1/2), таймер (Timer) и два процессорных элемента (ПЭ) PEx и PEy с 32-/40-разрядной архитектурой. Обмен данными ядра с внутренней памятью осуществляется по шине Core Bus, имеющей две 64-разрядные шины данных PMD (*Program Memory Data*) и DMD (*Data Memory Data*). Для работы с внешней памятью и устройствами ввода/вывода служат шина ядра (Core BUS) и шина ввода/вывода IOB (*Input/Output BUS*), образованные соответственно 32-рзрядными шинами данных CMD (*Core Memory Data*) и IOD (*Input/Output Data*) совместно с соответствующими шинами адреса CMA (*Core Memory Address*) и IOA (*Input/Output Address*) (на рис. 4.6 шины адреса не показаны). Шины CMA и CMD образуют внешнюю (системную) магистраль (PERIPHERIAL BUS), которая связана с шинами ядра через мультиплексоры шин адреса и данных (Cross Bar), являющиеся составной частью внешнего порта.

Внутренняя память

Внутренняя память процессора (Internal Memory) делится на блоки. Память процессоров ADSP-2136x состоит из четырёх блоков, которые могут быть сконфигурированы под различные комбинации адресного пространства для хранения программного кода и данных. Каждый блок внутренней памяти имеет по четыре столбца 16-разрядных слов, что позволяет ядру работать с 16-, 32-, 48- и 64-разрядными данными. Два блока памяти могут быть сконфигурированы как оперативное (RAM) или как постоянное (ROM) запоминающие устройства. При этом для программного кода и для данных могут использоваться различные области адресного пространства в соответствии с задаваемой картой памяти. Всё это обеспечивает гибкую систему распределения внутренней памяти процессора между программным кодом и данными.

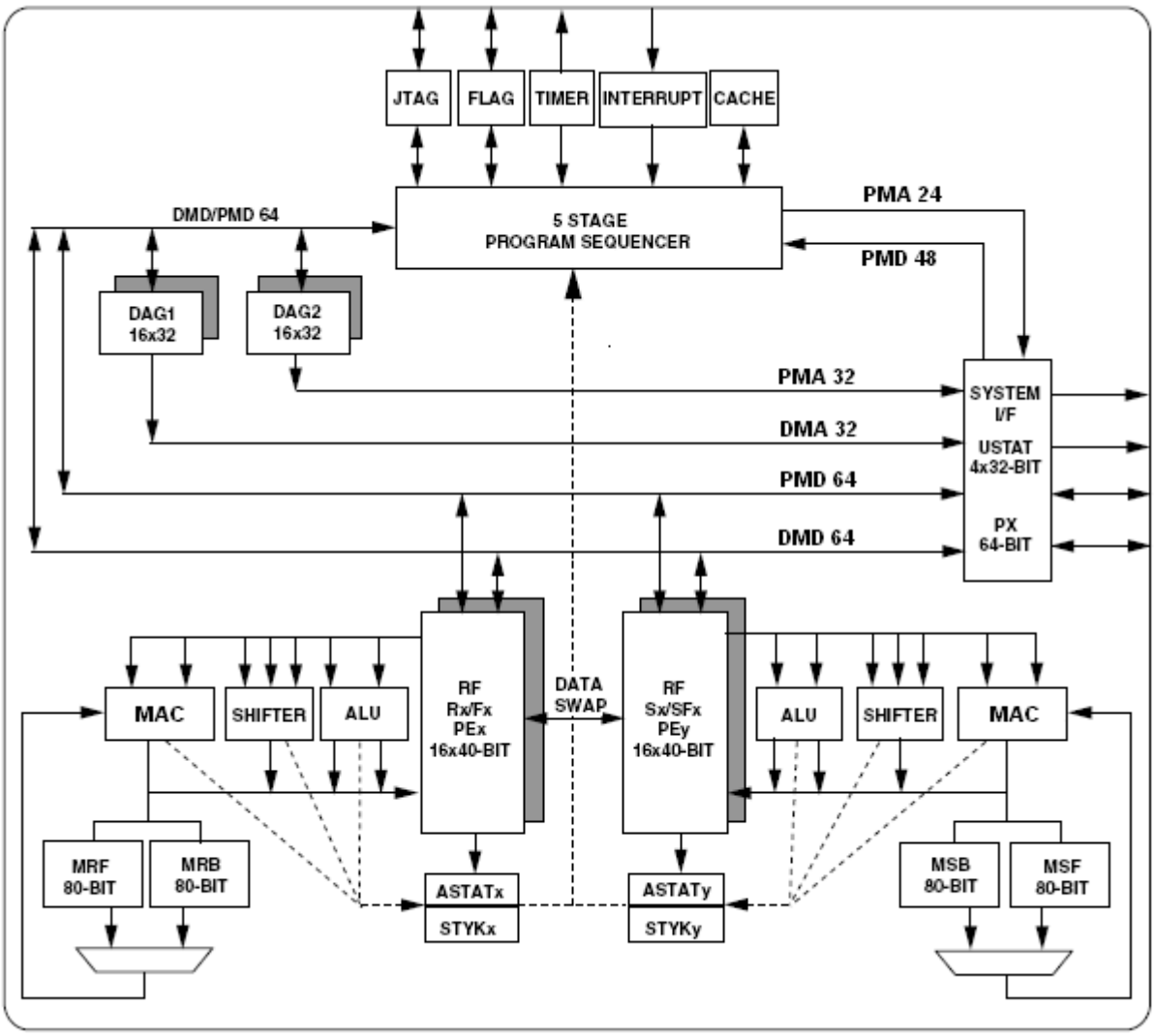


Рис. 4.6. Структура ядра процессоров семейства ADSP-2136x

Структурное построение ADSP-21364

Структурное построение процессоров семейства ADSP 2136x (рис. 4.6, 4.7) основано на использовании двух независимых шин для обращения к памяти программ PM (*Program Memory*) и к памяти данных DM (*Data Memory*), предназначенных соответственно для выборки команд и передачи двойного (адресуемого двумя генераторами DAG1/2) набора данных. Образованы они двумя шинами адреса

– PMA (*Program Memory Address*) и
– DMA (*Data Memory Address*),

и двумя шинами данных

– PMD (*Program Memory Data*),
– DMD (*Data Memory Data*).

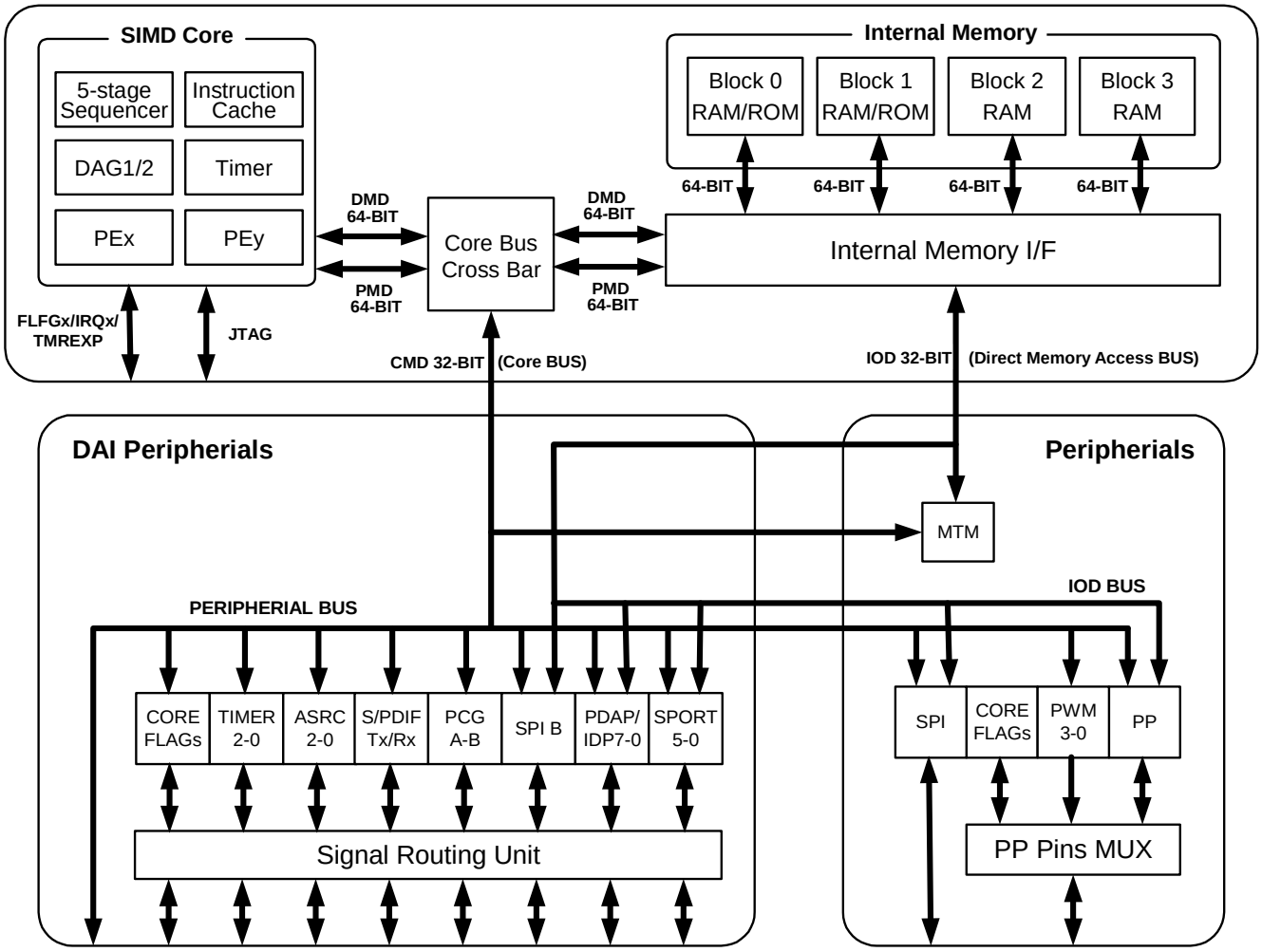


Рис. 4.7. Структурная схема ЦПС ADSP-21364

Шина адреса памяти программы PMA и шина адреса памяти данных DMA используются Для передачи адресов команд и данных. Шина данных PMD и шина данных DMD используются для передачи данных или команд,

хранящихся в памяти любого типа. Доступ к внутренней памяти осуществляется через набор коммутаторов, входящих в состав интерфейса с внутренней памятью Internal Memory I/F (*Internal Memory Interface/Fetch*), являющегося составной частью системного интерфейса System I/F (рис. 4.7). Пересылка данных между 64-разрядными шинами PMD и DMD или между файлами RF 40-разрядных регистров и 64-разрядной шиной DMD происходит через 64-разрядный регистр PX, состоящий из двух 32-разрядных регистров PX1 и PX2, которые, как и регистр PX, программно доступны. Программно доступными являются также четыре 32-разрядных регистра USTATx (*User-Defined Status Reg*isters), значения отдельных разрядов в котором задаются пользователем или копируются из соответствующего регистра состояния с целью использования их как флагов общего назначения. Кроме того, регистры USTATx можно использовать для временного хранения 32-разрядных слов.

В памяти данных хранятся данные, в памяти программы – и команды, и данные. Именно такое распределение данных позволяет осуществлять двойное обращение к памяти данных. Однако при этом необходимо учитывать то, что к памяти программ одновременно могут обращаться два устройства – устройства управления последовательностью выборки команд УУПВК и генератор адреса данных DAG2. Это приводит к возникновению конфликтных ситуаций на шине PM, при которых двойное обращение к памяти данных становится невозможным. Для уменьшения вероятности таких событий память программы кэшируется. Инструкции, при выборке которых возникла конфликтная ситуация, запоминаются, в результате чего следующий выбор такой инструкции произойдёт не из PM, а из кэш-памяти. Поэтому двойное обращение к памяти данных реализуется только тогда, когда соответствующая команда находится в кэше, что случается в большей части обращений к памяти данных. (Подробнее см. раздел «Кэш команд»).

При обращении к памяти программ PM УУПВК использует 24 разряда шины PMA и 48 разрядов шины данных PMD, что соответствует числу разрядов в командах процессора. При работе с памятью данных DM используются полная 32-битовая разрядность адресных шин DMA и PMA и 64-битовая разрядность шин PMD и DMD.

Внутренняя память является двухпортовой. Один из портов используется ядром, а другой – процессором ввода/вывода IOP (*Input/Output Processor*), управляющим обменом данными с устройствами ввода/вывода (УВВ).

Для обмена данными с УВВ служит входящий в состав IOP контроллер прямого доступа к памяти (КПДП), имеющий 25 каналов для работы с устройствами ввода/вывода без посредничества ядра. Для передачи данных КПДП использует отдельные шины ввода/вывода – шину данных IOD (*Input/Output Data*) и шину адреса IOA (*Input/Output Address*). Поэтому в одном цикле к каждому блоку памяти могут независимо обращаться ядро процес-

сора и контроллер прямого доступа к памяти, а также устройства ввода/вывода, для которых обмен данными в режиме прямого доступа не предусмотрен. Это позволяет выполнить за один цикл две передачи данных из ядра и одну из устройства ввода/вывода. Для копирования содержимого внутренней памяти используется стандартный режим прямого доступа к памяти (режим MTM – *memory-to-memory*).

В состав IOP входят стандартные для семейства ADSP 2136x периферийные устройства (Peripherials):

– 8- или 16-разрядный параллельный порт PP (*Parallel Port*) и
– четыре широтно-импульсных модулятора PWM0-3 (*Pulse-Width Modulator* 0-3),
– синхронный последовательный интерфейс SPI (*Serial Peripheral Interface*).

Одним из наиболее важных применений параллельного порта PP является использование его не только в качестве интерфейса с устройствами ввода/вывода, но и как интерфейса с внешним статическим запоминающим устройством (SRAM – *Static Random Access Memory*) и с FLASH памятью. В таком случае адрес и данные передаются в мультиплексном режиме через 16 контактов (AD15–0) микросхемы ЦПС под управлением синхронизирующего сигнала ALE, который служит для фиксации передаваемого адреса. Через PP могут передаваться 8-разрядные слова в 24-битовое адресное пространство, или 16-разрядные слова с 16-битовым адресом.

Порт SPI служит для последовательной передачи данных в синхронном режиме. Данные передаются и принимаются множеством устройств с использованием общего тактирования, обеспечиваемого ведущим. Передача управления новому ведущему определяется посредством разрешающего сигнала со стороны текущего ведущего.

Процессор ввода/вывода IOP в семействе ADSP-2136x имеет специализированный цифровой аудио интерфейс DAI (*Digital Audio Interface*), через который осуществляется связь с различными периферийными устройствами профильного назначения. Для подключения к периферийным устройствам используются 20 контактов (контакты DAI_P20-1) микросхемы ЦПС. Данные от устройства или к устройству передаются по шине ядра или шине ввода/вывода. Необходимых для этого соединения устанавливаются путём программирования коммутаторов, входящих в составе сигнального маршрутизатора SRU (*Signal Routing Unit*). DAI имеет

– шесть полнодуплексных последовательных портов SPORT0-5 (*Serial Port* 0-5),
– два выделенных совместимых с SPI порта SPI B – один с непосредственным выходом на выделенные для него контакты микросхемы

ЦПС, другой – с выходом на контакты микросхемы через маршрутизатор DAI,
- два генератора точного времени PCG A-B (*Precision Clock Generators* A-B),
- порты входных данных с конфигурацией «восемь портов данных» IDP0-7 (*Input Data Port* 0-7) или «параллельный порт сбора данных» PDAP (*Parallel Data Acquisition Port*),
- прёмопередатчик (S/PDIF Tx/Rx),
- три 8-канальных преобразователя с синхронно/асинхронной частотой дискретизации ASRC (*synchronous/Asynchronous Sample Rate Converter*),
- три дополнительных таймера (Timer0-2).

Преобразователи ASRC приспособлены для работы с соответствующими аудио кодеками такими, как используемый в исследуемой в данной лабораторной работе микросистеме аудио кодек AD1835A фирмы Analog Devices.

Через оба периферийных блока IOP передаётся слово состояния ядра (CORE FLAGs)

Для портов ввода/вывода нет выделенного адресного пространства и их адреса отображены на память. На память отображены также адреса некоторых регистров процессора. Часть из этих регистров имеет мнемонические (используемые в командах Ассемблера) обозначения. Другая часть требует явного указания адресов, находящихся в выделенной для них области памяти.

Для пересылки данных из УВВ в память или в УВВ из памяти IOP использует систему коммутаторов и буферы для временного хранения передаваемых данных. С их помощью реализуется выборка данных и интерфейс с внутренней памятью – Internal Memory I/F (*Interface/Fetch*).

Используя шины DM и PM, ядро процессора и устройство ввода/вывода могут обращаться к внешней или внутренней памяти в одном и том же цикле. Имеется система арбитража шин для обработки конфликтующих обращений. Приоритеты арбитража фиксированы и установлены в следующем порядке (по убыванию): шина DM, шина PM, устройство ввода-вывода. Кроме того, время обращения устройства ввода/вывода не может выходить за временные пределы, устанавливаемые для каждого пакетного обращения, поэтому предоставление шины ядру процессора никогда не задерживаются более чем на 4 цикла.

Вычислительные устройства процессорных элементов

Каждый процессорный элемент (рис. 4.6) имеет регистровый файл данных с регистрами Rx/Fx (ПЭ PEx) и Sx/SFx (ПЭ PEy) и три независимых

вычислительных устройства (ВУ): арифметико-логическое устройство (ALU), умножитель-аккумулятор (MAC), устройство сдвига (Shifter). Операнды и промежуточные результаты работы вычислительных устройств хранятся и запоминаются в регистровых файлах RF. Для быстрого переключения контекста, например, при обработке прерываний, регистровый файл имеет два набора регистров (первичный и вторичный), каждый из которых включает 16 40-разрядных регистров.

Необходимо сделать общее замечание, касающееся особенности команд, управляющих вычислительными операциями процессорных элементов. Все эти команды в общем случае являются условными, но наличие условия выполнения команды не является обязательным и его в команду можно не включать.

Регистровые файлы

Регистровые файлы ADSP-2136x не имеют регистров, ориентированных на работу с каким-либо вычислительным устройством. Поэтому регистры Rx/Fx и Sx/SFx могут рассматриваться как регистры общего назначения. Регистровые файлы Shifter обеспечивают интерфейс между внутренними шинами данных процессора и вычислительными устройствами. Они также служит для хранения операндов и локальных результатов. Регистровые файлы состоят из 16 первичных и 16 дополнительных (вторичных) регистров. На рис. 4.6 вторичные регистры показаны как тени от первичных. Наличие вторичных регистров облегчает быстрое контекстное переключение, т.к. даёт возможность вызывать процедуры обработки прерываний, не прибегая к сохранению регистров процессора. Это значительно сокращает время обработки запросов прерываний особенно при запросах с одним уровнем вложения.

Все регистры – 40-разрядные. 32-разрядные данные из вычислительного устройства всегда выравниваются влево: при считывании регистра 8 младших разрядов игнорируются, при записи – заполняются нулями. Обмен данными между памятью программы и памятью данных с регистровым файлом происходит по 64-разрядным шинам данных памяти программы PMD и памяти данных DMD. Одно обращение по шине PMD и по шине DMD может происходить за один цикл.

Регистровый файл является многопортовым. Для обмена данными между исполнительными единицами (ALU, MAC и Shifter) используются 10 портов. Ещё два порта служат для обеспечения связи между шинами PMD и DMD. Обмен данными всегда происходит 40-разрядными словами. При передачах по шинам PMD и DMD данные из регистрового файла размещаются в 40 старших разрядах, а младшие разряды заполняются нулями.

Имя одного и того же регистра из регистрового файла может начинаться с префиксов R или F (процессорный элемент PEx) и с префиксов S или SF

(процессорный элемент PEy). Префиксы R и S в имени регистра ставятся, если производятся вычислениях с фиксированной точкой, а префиксы F и SF – для вычислений с плавающей точкой.

Для облегчения быстрого контекстного переключения регистровые файлы содержат дополнительные (вторичные) регистры. Подключение первичных или вторичных регистров происходит независимо в каждой половине регистровых файлов (рис. 4.8) – в младшей (регистры R0-R7, S0-S7) и в старшей (регистры R8-R15, S8-S15). Активный набор определяется двумя битами в регистре режима MODE1 – битами SRRFH (*Secondary Registers For Register File High Enable*) и SRRFL (*Secondary Registers For Register File Low Enable*).

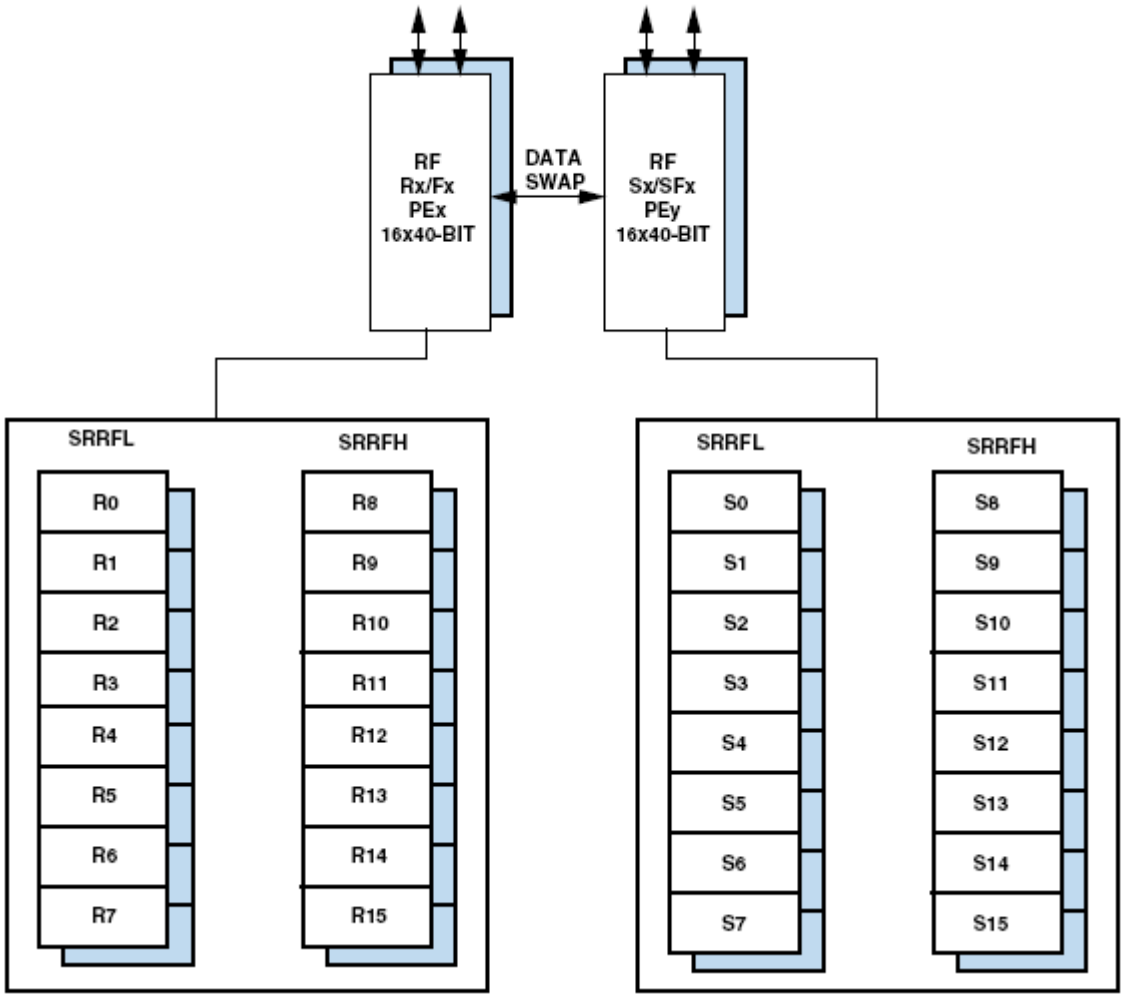


Рис. 4.8. Регистровые блоки вычислительных элементов

Предусмотрен также обмен данными (DATA SWAP) между регистрами регистровых файлов процессорных элементов.

Процессорные элементы PEx и PEy

Процессорные элементы PEx и PEy имеют следующие функциональные особенности:

- PEx и PEy поддерживают 32-битовые операции с фиксированной и плавающей точкой и 40-битовые операции с повышенной точностью над числами в формате IEEE.
- ALU выполняет логические и арифметические операции над данными в формате с фиксированной и плавающей точкой.
- MAC выполняет операции умножения над данными с фиксированной и плавающей точкой и операции умножения с накоплением/вычитанием над данными с фиксированной точкой.
- Сдвигатель Shifter производит поразрядные логические и арифметические сдвиги, сдвиги битовых полей 32-разрядных операндов, а также выполняет операции нормализации, денормализации и вычисления показателя порядка чисел, представленных в формате с плавающей точкой.
- ALU и MAC поддерживают одновременное выполнение операций с фиксированной и плавающей точкой. При выполнении операций умножения с фиксированной точкой разрядность результата может находиться в пределах от 32 до 80 разрядов.

Вычислительные устройства (ALU, MAC, Shifter) приспособлены для совместной параллельной работы. В режиме параллельных вычислений работают также сами процессорные элементы PEx и PEy. Выходные значения одних вычислительных устройств в каждом ПЭ могут передаваться другим ВУ через соответствующие шины многопортовых регистровых файлов RF и использоваться ими при выполнении следующей инструкции. Операция чтения/записи в один регистр выполняется в течении времени исполнения одной инструкции.

Все вычислительные устройства помимо результата предоставляют информацию о состоянии (флаги операций), которые хранятся в 32-разрядных регистрах состояния ASTATx и ASTATy (ASTAT – *Arithmetic Status*) каждого процессорного элемента, а также в дополнительных регистрах STKYx и STKYy (*Sticky Status*). С помощью флагов контролируются

- появление нулевого результата в ALU (AZ – ALU *Zero/Floating-Point Underflow*),
- переполнение ALU (AV – ALU *Overflow*)
- возникновение отрицательное результата в ALU (AN – ALU *Negative*),
- выход за пределы 32-разрядной сетки в операции ALU (AV – ALU *Fixed-Point Carry*),

– недействительность операций с плавающей точкой в ALU (AI – ALU *Floating-Point Invalid Operation*),
– переполнение MAC (MV – *Multiplier Overflow*),

а также другие возникающие при работе процессора события. Флаги ALU используются в процессе управления ходом программы, например, при организации ветвлений и циклов. Старшие 31–24 биты регистров ASTATx/y (биты CACC – *compare accumulation results of last eight compare operations*) служат для запоминания флагов результата при выполнении ALU операций сравнения. Каждый новый флаг операции сравнения заносится в старший (MSB) 31-й разряд, а предшествующий сдвигается в сторону младшего 24-го разряда. Таким образом, можно запомнить флаги восьми операций сравнения и использовать их значения в ходе выполнения программы.

Работу микросистемы на базе ЦПС сопровождают различного рода события. События могут иметь как внутреннее (по отношению к ядру и процессорным элементам), так и внешнее (вызванное работой УВВ) происхождение. События внутри ядра и внутри ПЭ, называют исключительными ситуациями, или просто «исключениями». Исключительные ситуации всегда сопровождаются запросами прерывания (прерывания выполняемой программы) и, как правило, следующим за запросом вызовом процедуры обработки прерывания ISR (*Interrupt Service Routine*). Выполнение ISR зависит от установленной системы приоритетов, а также от установок в системе обслуживания прерываний, в частности от того, разрешена или не разрешена обработка запроса регистром маски путём установки или сброса в нём соответствующего бита.

Запросы прерываний, вызываемые устройствами ввода/вывода, также обслуживаются по установленной системе приоритетов и при условии, что восприятие запросов не замаскировано программными установками.

Регистры STKY служат для запоминания и индикации флагов операций. Особенность работы с регистрами STKY состоит в том, что при смене флага в ASTAT занесённое ранее в регистр STKY значение сохраняется до тех пор, пока его не очистит исполняемая программа, после чего процессор сможет вновь заносить значение нужного флага в регистр STKY.

Арифметическо-логическое устройство

ALU имеет два входа (вход X и вход Y) для двух операндов и выполняет:

– сложение, вычитание и вычисление среднего для чисел с фиксированной точкой;
– операции вычисления двоичного логарифма, определения диапазона значений и операции с мантиссой для чисел с плавающей точкой;

– арифметические операции с повышенной точностью (сложение с переносом, вычитание с заёмом) для чисел с фиксированной точкой;
– логические операции И (AND), ИЛИ (OR), исключающее ИЛИ (XOR), отрицание (NOT);
– вычисление абсолютной величины (функция ABS), минимального (функция MIN) и максимального (функция MAX) значений, а также операции ограничения (функция CLIP), сравнения (функция COMPARE) и пересылки (функция PASS);
– преобразование форматов;
– взятие обратной величины (функция RECIPS) и вычисление квадрата (функция RSQRTS) для чисел с плавающей точкой.

Операции с фиксированной точкой являются 32-разрядными. Значения операндов берутся из 32-х старших разрядов 40-разрядных регистров регистровых файлов RF.

Команды ALU

Операции ALU производятся только с теми данными, которые находятся в регистровом файле. Имя одного и того же регистра из регистрового файла может начинаться с префиксов R или F (процессорный элемент PEx) и с префиксов S или SF (процессорный элемент PEy). Префиксы R и S в имени регистра ставятся, если производятся вычислениях с фиксированной точкой, а префиксы F и SF – для вычислений с плавающей точкой. Например, следующие команды используют одни и те же регистры:

```
F0 = F1*F2;        /* умножение с плавающей точкой */
R0 = R1*R2;        /* умножение с фиксированной точкой */
```

Префиксы F, R, S и SF не влияют на передачу 32-разрядных (или 40-разрядных) данных. Они определяют только, как ALU, MAC и устройство сдвига обрабатывают данные. В Ассемблере не учитывается регистр клавиатуры. Поэтому в именах регистров применимы как прописные, так и строчные буквы.

Выполняемые ALU операции перечислены в таблицах 4.1 и 4.2. В таблице 4.1 представлены операции с фиксированной, а в таблице 4.2 операции с плавающей точкой. Для обозначения команд в Ассемблере процессоров ADSP 2106x используется алгебраическая нотация, приближающая его к языку высокого уровня.

В представленных ниже таблицах используются следующие обозначения и символы:

– Rn, Rx и Ry обозначают регистры, интерпретируемые как регистры с фиксированной точкой;

– Fn, Fx и Fy обозначают регистры, интерпретируемые как регистры с плавающей точкой;
– регистры Rx/Fx и Ry/Fy являются регистрами-источниками операндов, поступающих соответственно на X и Y входы ALU;
– регистры Rn и Fn представляют регистры-приёмники, в которых сохраняются результаты операций;
– заключённые в круглые скобки обозначения регистров – (Rn), (Rx), (Ry), (Fn), (Fx) и (Fy) – показывают их содержимое.

В SIMD режиме все представленные в таблицах инструкции используют дополнительные (вторичные регистры).

Таблица 4.1

Операции с фиксированной точкой	**Описание операций**
Rn = Rx + Ry	Сложение (Rx) и (Ry)
Rn = Rx – Ry	Вычитание (Ry) из (Rx)
Rn = Rx + Ry + CI	Сложение (Rx) и (Ry) с переносом
Rn = Rx – Ry + CI – 1	Вычитание (Ry) из (Rx) с заёмом
Rn = (Rx + Ry)/2	Вычисление среднего значения регистров Rx и Ry
COMP(Rx, Ry)	Сравнение (Rx) и (Ry). Инструкция предназначена для манипуляций с флагами CACC, значения которых сохраняются (накапливаются) в старших 31-24 битах регистров ASTATx/y. Флаги сравнения CACC устанавливаются или сбрасываются в зависимости от того, является ли X-операнд большим или меньшим Y-операнда. Каждый новый флаг операции сравнения заносится в старший (MSB) 31-й разряд ASTAT регистра, а предшествующий сдвигается в сторону младшего (LSB) 24-го разряда. Таким образом, сохраняются восемь флагов последовательно выполненных операций сравнения. Результаты операций отбрасываются. Приложение может использовать сохранённые флаги для реализации трёхмерных clip-операций, в частности при реализации трёхмерной графики
COMPU(Rx, Ry)	Беззнаковое сравнение (Rx) и (Ry)
Rn = Rx + CI	Добавление к (Rx) флага переноса CI
Rn = Rx + CI – 1	Заём из регистра Rx
Rn = Rx + 1	Увеличение на единицу (Rx)
Rn = Rx – 1	Уменьшение на единицу (Rx)
Rn = –Rx	Инверсия знака (Rx)
Rn = ABS Rx	Абсолютное значение (Rx)
Rn = PASS Rx	Пересылка (Rx) в регистр Rn

Rn = Rx OR Ry	Побитовое «ИЛИ» (Rx) и (Ry)
Rn = Rx XOR Ry	Побитовое «исключающее ИЛИ» (Rx) и (Ry)
Rn = NOT Rx	Побитовое отрицание (Rx)
Rn = MIN(Rx, Ry)	Минимальное из значений (Rx) и (Ry)
Rn = MAX(Rx, Ry)	Максимальное из значений (Rx) и (Ry)
Rn = CLIP Rx by Ry	Ограничение (Rx) верхним пределом, заданным в регистре Ry

Двоичный код чисел в операциях с плавающей точкой имеет два поля: поле мантиссы и поле показателя степени 2. В обоих полях числа представлены в дополнительном коде, старшие разряды которого показывают знак мантиссы и знак показателя.

Таблица 4.2

Операции с плавающей точкой	**Описание операций**
Fn = Fx + Fy	Сложение (Fx) и (Fy)
Fn = Fx – Fy	Вычитание (Fy) из (Fx)
Fn = ABS (Fx + Fy)	Абсолютное значение суммы (Fx) и (Fy)
Fn = ABS (Fx – Fy)	Абсолютное значение разности (Fx) и (Fy)
Fn = (Fx + Fy)/2	Среднее значение регистров Fx и Fy
COMP(Rx, Ry)	Сравнение (Fx) и (Fy)
Fn = –Fx	Инверсия знака (Fx)
Fn = ABS Fx	Абсолютное значение (Fx)
Fn = PASS Fx	Пересылка (Fx) в регистр Fn
Fn = RND Fx	Округление (Fx) до значения, представленного 32-разрядным кодом
Fn = SCALB Fx by Ry	Установить показатель числа с плавающей точкой путём добавления к нему (Ry)
Rn = MANT Fx	Выделить мантиссу из (Fx)
Rn = LOGB Fx	Определить показатель экспоненциального множителя числа в формате с плавающей точкой, хранящегося в регистре Fx (вычислить двоичный логарифм экспоненциального множителя). Операция обратная SCALB
Rn = FIX Fx	Преобразовать число с плавающей точкой из регистра Fx в 32-разрядное число с фиксированной точкой. Необходимое при этом правило округления определяется битом TRUNC в регистре режима MODE1. При TRUNC = 1 величина мантиссы округляется до максимального, а при TRUNC = 0 – до ближайшего целого значения.
Rn = FIX by Ry	Выполнить преобразование FIX над (Fx) после добавления к показателю степени целочисленной величины из регистра Ry

	Пример: Rn = FIX Fx by 31; /* *fixed-point* 1.31 *format* */
Rn = TRUNC Fx	Выполнить преобразование с округлением числа с плавающей точкой из регистра Fx в число с фиксированной точкой. Округление производится путём отбрасывания младших разрядов
Rn = TRUNC Fx by Ry	Выполнить преобразование TRUNC над (Fx) после предварительной добавки к показателю экспоненты целого числа из Ry
Fn = FLOAT Rx	Преобразовать 32-разрядное число с фиксированной точкой из регистра Rx в число с плавающей точкой. Правило округления определяется битом TRUNC в регистре режима MODE1
Fn = FLOAT Rx by Ry	Выполнить преобразование FLOAT над (Rx) с учётом масштаба, определяемого содержимым регистра Ry. При преобразовании целое число из Ry добавляется к показателю экспоненты. Пример: Fn = FLOAT Rx by 31; /* *floating-point* [-1.0 *to* 1.0] */
Fn = RECIPS Fx	Вычислить величину обратную (Fx)
Fn = RSQRTS Fx	Вычислить величину 1/(Fx)½ обратную корню квадратному от (Fx). Результат берётся из таблицы, находящейся во внутренней ROM и индексированной 4-разрядным кодом, определяемым в процессе выполнения операции
Fn = Fx COPYSIGN Fy	Присвоить знак числа из Fy числу в Fx без изменения показателя и мантиссы. Результат помещается в регистр Fn
Fn = MIN(Fx, Fy)	Вычислить минимальное из значений регистров Fx и Fy
Fn = MAX(Fx, Fy)	Вычислить максимальное из значений регистров Fx и Fy
Fn = CLIP Fx by Fy	Ограничить значение операнда Fx верхним пределом, установленным в регистре Fy

Умножитель-аккумулятор

Умножитель-аккумулятор (MAC) выполняет операции умножения и умножения/накопления с фиксированной и плавающей точкой. Имеет два входа (вход X и вход Y) для двух операндов. Результат операции MAC сохраняется либо в одном из регистров регистрового файла, либо в 80-разрядном регистре результата MR. При операциях с плавающей точкой MAC выполняет действия над 32- или 40-разрядными числами, получая 32- или 40-разрядный результат. Операции умножения с фиксированной точкой производятся над 32-разрядными числами, а результат перемножения сохраняется в 80-разрядном регистре результата. Работой MAC управляют с помощью инструкций, реализующих

- умножение с плавающей точкой,
- умножение с фиксированной точкой,
- умножение/накопление с фиксированной точкой (опционально с округлением),

- умножение/вычитание с фиксированной точкой (опционально с округлением),
- округление результата перемножения
- ограничение результата перемножения

Результат операций, производимых умножителями-аккумуляторами, может пересылаться в один из двух 80-разрядных регистров результатов MR – в основной MRF/MSF (*Foreground Register*) или в дополнительный MRB/MSB (*Background Register*) регистры. Аббревиатура MR используется для обозначения пары регистров MRF/MSF и MRB/MSF. Активируется либо один, либо другой регистры MR, что даёт дополнительную возможность быстрого контекстного переключения. Однако в отличие от других регистров, имеющих дополнительный набор регистры, MRF/MSF и MRB/MSB могут использоваться совместно в одно и то же время. Регистры имеют одинаковый формат – каждый делится на регистры MR2, MR1 и MR0 (рис. 4.9), содержимое которых может отдельно считываться и модифицироваться. Поэтому наряду с использованием каждого из регистров MR как основного и дополнительного их можно использовать как два параллельных накопителя. Эта особенность облегчает работу с комплексными числами.

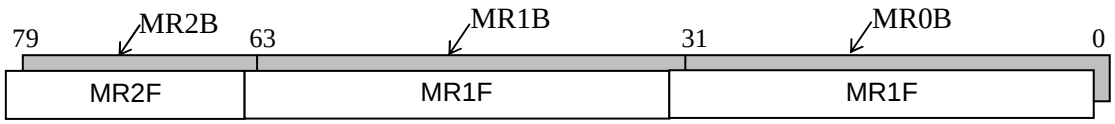


Рис. 4.9.

Инструкции, используемые умножителем-аккумулятором для выполнения операций с фиксированной точкой представлены в таблице 4.3

Таблица 4.3

Операции MAC	**Описание операций**
Rn = Rx * Ry	Умножение (Rx) и (Ry) с сохранением результата в регистре Rn регистрового файла
MRF = Rx * Ry	Умножение (Rx) и (Ry) с сохранением результата в 80-разрядном регистре результата MRF
MRB = Rx * Ry	Умножение (Rx) и (Ry) с сохранением результата в 80-разрядном регистре результата MRB
Rn = MRF + Rx * Ry	Умножение (Rx) и (Ry) с накоплением в регистре MRF и с сохранением результата в регистре Rn регистрового файла
Rn = MRB + Rx * Ry	Умножение (Rx) и (Ry) с накоплением в регистре MRB и с сохранением результата в регистре Rn регистрового файла
MRF = MRF + Rx * Ry	Умножение (Rx) и (Ry) с накоплением в регистре MRF и с сохранением результата в регистре MRF
MRB = MRB + Rx *	Умножение (Rx) и (Ry) с накоплением в регистре MRB и с со-

Ry	хранением результата в регистре MRB
Rn = MRF – Rx * Ry	Умножение (Rx) и (Ry) с вычитанием из регистра MRF и с сохранением результата в регистре Rn регистрового файла
Rn = MRB – Rx * Ry	Умножение (Rx) и (Ry) с вычитанием из регистра MRB и с сохранением результата в регистре Rn регистрового файла
MRF = MRF – Rx * Ry	Умножение (Rx) и (Ry) с вычитанием из регистра MRF и с сохранением результата в регистре MRF
MRB = MRB – Rx * Ry	Умножение (Rx) и (Ry) с вычитанием из регистра MRB и с сохранением результата в регистре MRB
Rn = SAT MRF	Установка максимального целочисленного значения (насыщение) регистра MRF с сохранением результата в регистре Rn регистрового файла
Rn = SAT MRB	Установка максимального целочисленного значения (насыщение) регистра MRB с сохранением результата в регистре Rn регистрового файла
MRF = SAT MRF	Установка максимального значения (насыщение) регистра MRF в формате с фиксированной точкой
MRB = SAT MRB	Установка максимального значения (насыщение) регистра MRB в формате с фиксированной точкой
Rn = RND MRF	Округление содержимого регистра MRF в формате с фиксированной точкой до 32-разрядного целочисленного значения с сохранением результата в регистре Rn регистрового файла
Rn = RND MRB	Округление содержимого регистра MRB в формате с фиксированной точкой до 32-разрядного целочисленного значения с сохранением результата в регистре Rn регистрового файла
MRF = RND MRF	Округление содержимого регистра MRF до 32-разрядного целочисленного значения с сохранением результата в регистре MRF
MRB = RND MRB	Округление содержимого регистра MRB до 32-разрядного целочисленного значения с сохранением результата в регистре MRB
MRF = 0	Обнуление регистра MRF
MRB = 0	Обнуление регистра MRB
MR*x*F = Rn	Загрузка регистра MR*x*F (*x* = 0,1,2) содержимым регистра Rn регистрового файла
MR*x*B = Rn	Загрузка регистра MR*x*B (*x* = 0,1,2) содержимым регистра Rn регистрового файла
Rn = MR*x*F	Сохранение регистра MR*x*F (*x* = 0,1,2) в регистре Rn регистрового файла
Rn = MR*x*B	Сохранение регистра MR*x*B (*x* = 0,1,2) в регистре Rn регистрового файла

В инструкциях с операциями умножения, а также в командах насыщения и округления используются модификаторы, обозначающие тип операндов. Операции умножения требуют указания типа операндов по X и Y вхо-

дам умножителя, для чего используются заключённые в круглые сочетания символов (SSF), (SSI), (SSFR), (SUF), (SUI), (SUFR), (USF), (USI), (USFR), (UUF), (UUI) или (UUFR). Включённые в модификаторы символы обозначают:

операнд со знаком (*Signed input*) – символ S;
операнд без знака (*Unsigned input*) – символ U;
целочисленный операнд (*Integer input*) – символ I;
операнд – дробное число (*Fractional input*) – символ F;
FR – операнд – дробное число, результат с округлением (*Fractional input, Rounded output*) – символы FR.

Примером использования модификаторов в командах умножения являются инструкции

```
MRF = Rx * Ry (SSF);    /* signed x signed/fractional */
MRF = Rx * Ry (SUF);    /* signed x unsigned/fractional */
MRF = Rx * Ry (USF);    /* unsigned x signed/fractional */
MRF = Rx * Ry (UUI);    /* unsigned x unsigned/integer */
```

В качестве модификаторов однооперандных команд используются:

(SF), (SI), (UF) и (UI) – в инструкциях SAT;
(SF) и (UF) – в инструкциях RND.

MAC выполняет только одну операцию с плавающей точкой – операцию умножения

Fn = Fx * Fy.

Сдвигатель

Сдвигатель выполняет битовые операции с 32-разрядными числами в формате с фиксированной точкой. К таким операциям относятся:

- арифметические, логические и циклические сдвиги содержимого регистров и образующих его битовых полей;
- манипуляции с разрядами битовых полей, включая проверку, сброс, установку и смену значения;
- манипуляции с битовыми полями, связанные с извлечение, перемещением в битовом пространстве и с сохранением результата манипуляций;
- включая определение показателя степени, нормализация и денормализация чисел с фиксированной точкой;

– преобразование 32-разрядных чисел в формате с плавающей точкой в 16-разрядные числа с плавающей точкой и выполнение обратного преобразования.

Операции с плавающей точкой связаны лишь с упаковкой 32-разрядных чисел в 16-разрядный формат и с обратной операцией распаковки 16-разрядных чисел с плавающей точкой в 32-разрядный формат.

В таблице 4.4 представлен перечень выполняемых сдвигателем операций. Как и в операциях ALU и MAC регистры Rx/Fx и Ry являются операндами-источниками, а регистры Rn/Fn – операндами-приёмниками. Преобразуемый операнд берётся из регистров Rx/Fx и подаётся на X вход. В регистрах Ry задаются параметры сдвиговых операций. Параметры операций являются целыми числами. Поэтому для их представления не требуются регистры Fy с плавающей точкой. Кроме того, параметры сдвиговых операций могут задаваться в инструкциях в виде непосредственных данных. К ним относятся:

data8 – 8-разрядные данные, определяющие направление и величину сдвига;

bit6 – 6-разрядное число, определяющее место расположения (начало) битового поля;

len6 – 6-разрядное число, определяющее размер битового поля.

Таблица 4.4

Операции сдвигателя	Описание операций
Rn = LSHIFT Rx by Ry | <data8>	Логический сдвиг (Rx) с размещением результата в регистре Rn. Величина и направление сдвига определяются содержимым регистра Ry или 8-разрядным полем data8 в инструкции
Rn = Rn OR LSHIFT Rx by Ry | <data8>	Логический сдвиг (Rx) и логическое сложение результата сдвиговой операции с исходным содержимым регистра Rn. Величина и направление сдвига определяются содержимым регистра Ry или 8-разрядным полем data8 в инструкции. Результат сохраняется в регистре Rn
Rn = ASHIFT Rx by Ry | <data8>	Арифметический сдвиг (Rx) с размещением результата в регистре Rn. Величина и направление сдвига определяются содержимым регистра Ry или значение 8-разрядного поля data8 в инструкции
Rn = Rn OR ASHIFT Rx by Ry | <data8>	Арифметический сдвиг (Rx) и логическое сложение результата сдвиговой операции с исходным содержимым регистра Rn. Величина и направление сдвига определяются содержимым регистра Ry или значением 8-разрядного поля data8 в команде. Результат сохраняется в регистре Rn

Rn = ROT Rx by Ry | <data8>	Циклический сдвиг (Rx). Результат разместить в регистре Rn. Величина и направление сдвига определяются содержимым регистра Ry или значением 8-разрядного поля data8 в команде
Rn = BCLR Rx by Ry | <data8>	Очистить в регистре Rx бит, определяемый значением регистра Ry или непосредственным значением data8. Результат разместить в регистре Rn
Rn = BSET Rx by Ry | <data8>	Установить в регистре Rx бит, определяемый значением регистра Ry или непосредственным значением data8. Результат разместить в регистре Rn
Rn = BTGL Rx by Ry | <data8>	Инвертировать в регистре Rx бит, определяемый значением регистра Ry или непосредственным значением data8. Результат разместить в регистре Rn
BTST Rx by Ry | <data8>	Проверить в регистре Rx бит, определяемый значением регистра Ry или непосредственным значением data8
Rn = FDEP Rx by Ry | <bit6>:<len6> Rn = FDEP Rx by Ry | <bit6>:<len6>(SE)	Переместить битовое поле (с расширением знака при наличии модификатора SE – sign-extended) из регистра Rx, содержащего 32-разрядное число с фиксированной точкой, в регистр Rn. Перемещаемое из регистра Rx битовое поле начинается с младшего разряда числа с фиксированной точкой, имеет длину len6 и переносится в регистр Rn со смещением, определяемым значением bit6. Значения 6-разрядных двоичных чисел len6 и bit6 берутся из соответствующих полей регистра Ry или как непосредственные значения из инструкций
Rn = Rn OR FDEP Rx by Ry | <bit6>:<len6> Rn = Rn OR FDEP Rx by Ry | <bit6>:<len6>(SE)	Переместить битовое поле (с расширением знака при наличии модификатора SE – sign-extended) из регистра Rx, содержащего 32-разрядное число с фиксированной точкой, в регистр Rn, и выполнить операцию логического сложения с исходным значением регистра Rn. Перемещаемое из регистра Rx битовое поле начинается с младшего разряда числа с фиксированной точкой, имеет длину len6 и переносится в регистр Rn со смещением, определяемым значением bit6. Значения 6-разрядных двоичных чисел len6 и bit6 берутся из соответствующих полей регистра Ry или как непосредственные значения из инструкций
Rn = FEXT Rx by Ry | <bit6>:<len6> Rn = FEXT Rx by Ry | <bit6>:<len6> (SE)	Переместить из регистра Rx в регистр Rn битовое поле (с расширением знака при наличии модификатора SE). Длина и расположение перемещаемого поля из регистра Rx определяется значениями len6 и bit6, взятыми из регистра Ry

	или как непосредственные значения из инструкции. В регистре Rn перемещаемое битовое поле располагается, начиная с младшего разряда 32-разрядного числа в формате с фиксированной точкой
Rn = EXP Rx	Определить показатель степени числа в регистре Rx в формате с фиксированной точкой. Операция сопровождается нормализацией исходного числа с фиксированной точкой
Rn = LEFTZ Rx	Определить количество лидирующих нулевых бит в числе с фиксированной точкой из регистра Rx. Результат занести в поле bit6 регистра Rn
Rn = LEFTO Rx	Определить количество лидирующих бит со значением «1» в числе с фиксированной точкой из регистра Rx. Результат сохранить в поле bit6 регистра Rn
Rn = FPACK Fx	Преобразовать 32-разрядное число с плавающей точкой из регистра Fx в 16-разрядное число с плавающей точкой с 11-разрядной мантиссой, 4-разрядным показателем (два бита – это биты знака мантиссы и знака показателя). Результат заносится в регистр Rn
Fn = FUNPACK Rx	Преобразовать 16-разрядное число с плавающей точкой из регистра Rx в 32-разрядное число с плавающей точкой. Результат сохранить в регистре Fn

Генераторы адресов данных

Генераторы адресов данных (*Data Address Generators* – далее DAGs) генерируют адреса для перемещения данных в памяти, а также извлечения их из памяти. DAGs поддерживают несколько режимов адресации:

- выдача адреса с постмодификацией – обеспечивает адрес передачи данных для памяти программ PM и памяти данных DM и автоматическое увеличение значения адреса для выполнения следующей передачи;
- выдача предмодифицированного адреса – обеспечивает адрес передачи данных с его предварительным изменением;
- модификация (изменение) адреса – образование нового адреса, который сохраняется в индексном регистре для обеспечения следующей передачи;
- формирование бит-реверсного адреса – при передаче данных формируется адрес с обратным порядком разрядов (реверсирование текущего адреса не производится);
- одновременная загрузка первичных и вторичных регистров (*Broadcast Data Loads*) регистровых файлов процессорных элементов с целью поддержки SIMD режима работы.

Каждый из генераторов адресов данных имеет (рис. 4.10):

1. *Регистры индекса* (I0-I7 *в генераторе* DAG1 *и* I8-I15 *в генераторе* DAG2). Каждый регистр индекса хранит адрес и действует как указатель.
2. *Регистры модификации* (M0-M7 *в* DAG1 *и* M8-M15 *в* DAG2). Регистры модификации обеспечивают значение, на которое изменяется (пред- или постмодифицируется) регистр индекса, используемый как указатель адреса.
3. *Регистры длины и базового адреса цикличесих буферов* (L0-L7 *и* B0-B7 *в* DAG1 *и* L8-L15 *и* B8-B15 *в* DAG2). Регистры длины и базового адреса определяют диапазон адресов и начальный адрес циклических буферов.

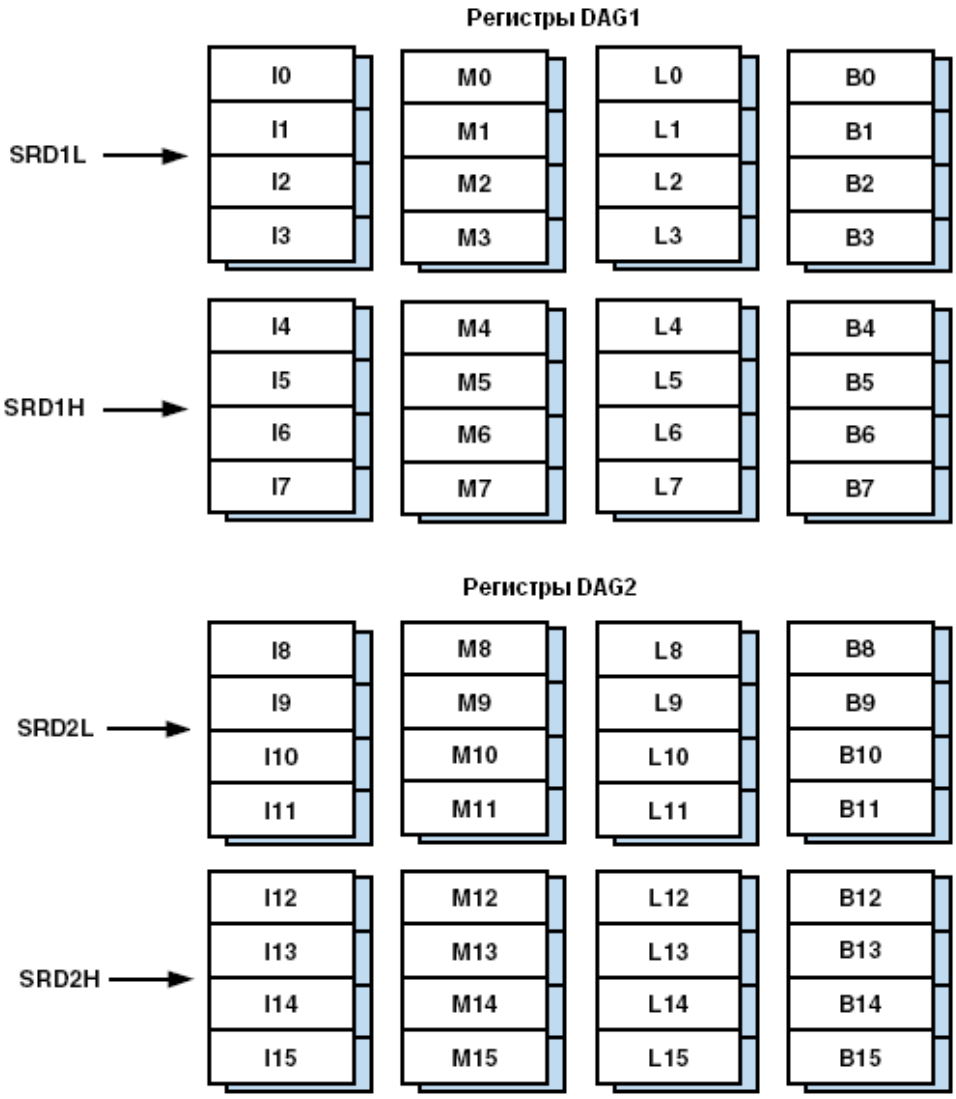


Рис. 4.10. Регистровые модели генераторов адресов данных

Регистры I, M, L и B после сброса процессора содержат случайные значения. Перед использованием любого из регистров I программа должна инициализировать соответствующий ему регистр L необходимым значением – нулём для линейной адресации или значением длины буфера при работе с циклическим буфером.

Каждый генератор подобно регистровым файлам RF имеет вторичный набор регистров для реализации быстрого контекстного переключения. Активный наборов регистров DAGs определяется значением соответствующих разрядов в регистре режима MODE1: SRD1H – *Secondary Registers For* DAG1 *High Enable*; SRD1L – *Secondary Registers For* DAG1 *Low Enable*; SRD2H – *Secondary Registers For* DAG2 *High Enable*; SRD2L – *Secondary Registers For* DAG2 *Low Enable*. Регистры генераторов адресов данных переключаются независимо от регистров регистровых файлов.

В ассемблере ADSP-2136x операции с пред- и постмодификацией исполнительного адреса отличаются положением индекса и модификатора в команде. Если регистр I стоит в команде перед модификатором, то это соответствует постмодификации. Если модификатор стоит впереди, то это указывает на предмодификацию. Например, команда

```
R6 = PM(I15,M12);
```
Косвенная адресация с постмодификацией

служит для загрузки регистра R6 из ячейки памяти программы PM с адресом, который хранится в I15. В регистр I15 возвращается значение, равное I15 + M12:

При исполнении команды

```
R6 = PM(M12,I15);
```
Косвенная адресация с предмодификацией

сначала образуется адрес I15 + M12, а потом этот адрес передаётся в PM.

Аналогичным образом строятся команды обращения к памяти данных с заменой аббревиатуры PM на аббревиатуру DM. Но при построении команд необходимо учитывать специализацию генераторов DAG1 и DAG2: генератор DAG1 работает с памятью данных, а генератор DAG2 – с памятью программ. Это показано и на рис. 4.6, из которого следует, что DAG1 связан только с шиной DMA, а DAG2 – только с шиной PMA.

Любой индексный регистр I может модифицироваться содержимым любого регистра-модификатора M в одном и том же генераторе (в DAG1 или в DAG2).

Допустимы непосредственные модификаторы. Значение непосредственного модификатора зависит от типа команды.

Структура команд и многофункциональные команды

Все команды МП ADSP-2136x являются 48-разрядными и допускают условное и безусловное исполнение. Кроме того, одна команда может управ-

лять одновременно несколькими исполнительными устройствами, одним из которых является вычислительное устройство, а другими – устройства, реализующие сохранение регистров в памяти и/или загрузку данных из памяти в регистры.

Вычислительные устройства выполняют операции только с данными, находящимися в регистровом файле. Команды, предназначенные для управления вычислительными устройствами, являются однофункциональными. К однофункциональным командам относятся также команды для каждого из генераторов адресов данных и команды управления выполнением программы (команды переходов, вызовов и возвратов).

Из относительно простых вычислительных команд и команд загрузки и сохранения регистров в памяти выстраиваются более сложные многофункциональные инструкции, с помощью которых организуется параллельное выполнение вычислительных операций и операций обращения к памяти программ и к памяти данных. Двоичные коды таких команд включают код вычислительной операции и кодовые поля для загрузки или сохранения регистров в памяти. Такая структура команды показывается и в её мнемоническом обозначении:

$$\textit{вычисление}, \left|\begin{array}{l}\text{DM(Ia,Mb) = dreg1}\\ \text{dreg1 = DM(Ia,Mb)}\end{array}\right|, \left|\begin{array}{l}\text{PM(Ic,Md) = dreg2}\\ \text{dreg2 = PM(Ic,Md)}\end{array}\right|, . \qquad (4.1)$$

В содержащихся в (4.1) мнемониках и ниже используются обозначения, представленные в таблице 4.5.

Таблица 4.5

Система обозначений в командах МП ADSP-2136x

Обозначение	Значение
ПРОПИСНЫЕ буквы	Ключевые слова ассемблера (такой способ написания используется только в приведённом здесь списке команд; ассемблер не чувствителен к регистру)
|….|	Между вертикальными разделителями помещается список операндов. Выбирается один из перечисленных в списке операндов
;	Точка с запятой – окончание команды
,	Запятая – разделяет параллельные операции в команде
курсив	Необязательная часть команды
вычисление	Операции ALU, MAC, устройства сдвига или многофункциональная операция ВУ
dreg	Регистры данных: R0-R15 или F0-F15
Ia	I0-I7 – индексные регистры DAG1

Mb	M0-M7 – регистры модификации DAG1
Ic	I8-I15 – индексные регистры DAG2
Md	M8-M15 – регистры модификации DAG2
условие	Условие выполнения команды
ureg	Универсальный регистр. К универсальным регистрам относятся все именованные регистры (регистры данных, системные регистры) процессора, но не регистры, отображённые на память
<data>	Данные – непосредственное значение

Команды составлены из разделённых запятыми мнемонических выражений, которые в общем случае обозначают вычислительные инструкции (операции ALU, MAC и сдвигателя) и инструкции пересылки данных. Через индексные регистры Ia косвенно адресуется память данных, а через регистры Ic – память программы. Значения индексных регистров обновляются добавлением соответствующих модификаторов – значений регистров Mb или Md. Аббревиатуры Dreg1,2 обозначают регистры регистровых файлов RF.

Вычислительные команды (команды ALU, MAC и сдвигателя) и команды обмена данными в общем случае являются условными. Однако наличие условия выполнения команды не является обязательным и его в команду можно не включать. В листинге 4.1 перечислены команды (простые и многофункциональные), выполнение которых может быть условным.

Доступ к памяти данных и памяти программы из регистрового файла реализуется с помощью косвенной регистровой адресации по указанным в команде регистрам I с применением пред- или постмодификации, если значение указанный в команде регистр M не равно нулю.

Вычислительные операции, из поля «*вычисление*» также могут быть многофункциональными. Многофункциональные вычисления подразделяются на три типа:

- двойное сложение/вычитание в ALU,
- параллельные операции MAC и ALU,
- параллельное умножение и сложение/вычитание.

В параллельных операциях MAC и ALU выполняются умножение или умножение/накопление и одна из следующих операций ALU: сложение, вычитание, усреднение, преобразование из формата с фиксированной точкой к формату с плавающей точкой и наоборот.

Допустимы параллельные операции умножения и двойного сложения/вычитания. При этом выполняется умножение или умножение/накопление и вычисляется сумма и разность входных операндов ALU. Результаты операций могут возвращаться в любой регистр регистрового

файла. Имеются ограничения на выбор входных регистров-операндов, что необходимо учитывать при формировании многофункциональных команд.

Листинг 4.1

```
IF  условие  вычисление;
IF  условие  вычисление,  |DM(Ia,Mb)| = ureg ;
                          | PM(Ic,Md)|
IF  условие  вычисление,  |DM(Mb,Ia)| = ureg ;
                          |PM(Md,Ic) |
IF  условие  вычисление,  ureg = |DM(Ia,Mb)|
                                 | PM(Ic,Md)| ;
IF  условие  вычисление,  ureg = |DM(Mb,Ia)|
                                 | PM(Md,Ic)| ;
IF  условие  вычисление,  |DM(Ia,<data>)| = dreg ;
                          |PM(Ic,<data>)|
IF  условие  вычисление,  |DM(<data>,Ia) |= dreg ;
                          |PM(<data>,Ic) |
IF  условие  вычисление,  dreg = |DM(Ia,<data>)|
                                 | PM(Ic,<data>)| ;
IF  условие  вычисление,  dreg = |DM(<data>,Ia) |
                                 | PM(<data>,Ic) | ;
IF  условие  вычисление,  ureg1 = ureg2 ;
IF  условие  непосредст-  |DM(Ia,Mb)| = dreg ;
             венный       | PM(Ic,Md)|
             сдвиг
IF  условие  непосредст-  dreg = |DM(Ia,Mb) |
             венный               |PM(Ic,Md) | ;
             сдвиг
IF  условие  вычисление,  MODIFY |(Ia,Mb)|
                                 |(Ic,Md)| ;
```

Программный автомат

Программный автомат служит для выборки из РМ инструкций (команд), управляющих работой процессора, и выполняет следующие функции:

- определяет адреса выборки команды,
- проверяет условия и определяет адреса ветвлений (переходов, вызовов, возвратов),
- организует выполнение циклически повторяющихся операций.

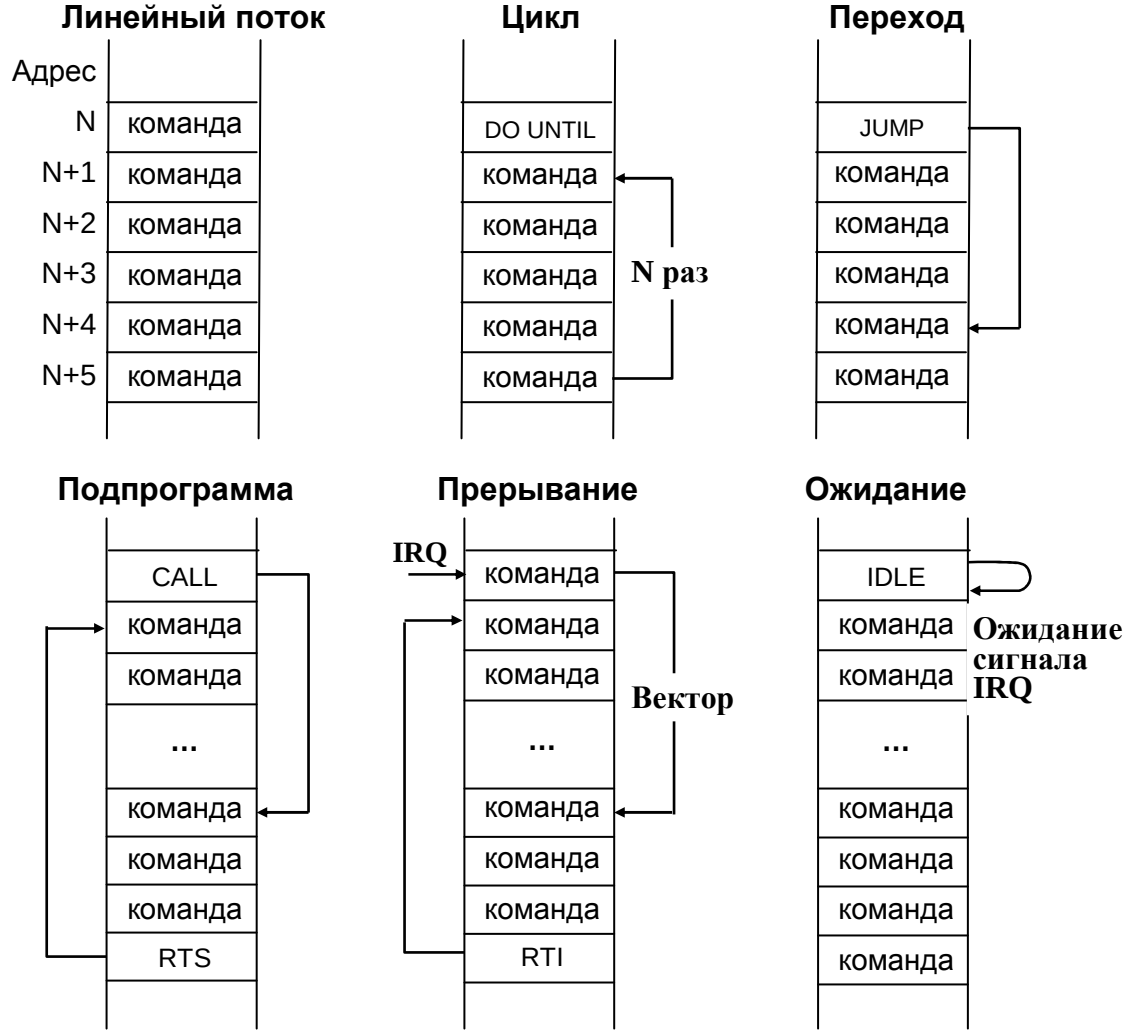


Рис 4.11.

Большей частью процессор выполняет программу линейно, последовательно извлекая команды из памяти и исполняя их (рис. 4.11). Изменения в линейном потоке команд обусловливаются *переходами* (JUMP), *вызовами процедур* (CALL) *и циклами* (DO UNTIL – делай, пока выполнено условие). Вызов процедуры предполагает возврат, который выполняется по команде RTS – *Return from Subroutine*. Наряду с программными существуют аппарат-

ные (асинхронные по отношению к исполняемой программе) вызовы, обусловленные происходящими в системе событиями, сопровождающимися запросами прерываний (IRQ – *Interrupt Requests*). Возврат из процедур обработки запросов прерываний осуществляется с помощью отдельной команды RTI (*Return from Interrupt*).

Процессор может прекратить выполнение программы, переходя в режим о*жидания* сигналов от устройств ввода/вывода – запросов прерываний IRQ. Для перехода в такой режим используется команда IDLE, после которой прекращается выборка инструкций из памяти и останавливаются операции по обработке данных. Работа процессора возобновляется после поступления запроса прерывания. Процессор обрабатывает поступивший запрос, после чего продолжает стандартное выполнение программы.

В своей работе программный автомат использует (см. рис. 4.12):

- устройство управления последовательностью выборки команд (Program Sequencer) с имеющимися в его составе программным счётчиком PC, регистр декодированного адреса DADDR (*Decode Address Register*) и регистр адреса выборки FADDR (*Fetch Address Register*)
- аппаратный стек программного счётчика (PC Stack), в котором сохраняется 24-разрядный адрес возврата из процедур и адрес перехода на первую команду при выполнении циклов. Вершина стека соответствует регистру PCSTK (*Program Counter Stack Register*). На адрес вершины стека (от 0 при свободном стеке до 31 при полностью занятом) показывает регистр PCSTKP (*Program Counter Stack Pointer Register*);
- устройство управления выполнением циклических операций (циклов) (Loop Sequencer);
- аппаратный стек циклов (Loop Stack) с шестью уровнями вложения, обеспечивающий поддержку вложенных циклов. В его составе имеются стек адреса цикла и стек счётчика цикла. Стек адреса цикла состоит из шести 32-разрядных ячеек. Каждая 32-разрядная ячейка состоит из 24-разрядного адреса завершения цикла, 5-разрядного кода завершения цикла и 2-разрядного кода типа цикла. Адрес завершения цикла, код завершения цикла и код типа цикла помещаются в стек после выполнения команды DO UNTIL. Содержимое извлекается из стека на последней итерации или при выполнении команды перехода, стимулирующей преждевременное завершение цикла. Вершина стека адреса цикла соответствует регистру LADDR. Число уровней вложения фиксируется в регистре LCNTR;
- логику условий, в которую передаются флаги арифметических операций из регистров состояния ASTATx и ASTATy, а также сигналы готовности Input, поступающие от внешних устройств через контакты микросхемы процессора;

– контроллер прерываний Iterrupt Control с регистром Latch для фиксации запросов прерываний от УВВ и с регистром маски Mask для запрета восприятия отдельно выделенных запросов;
– аппаратный стек для сохранения содержимого регистров ASTATx и ASTATy при вызовах процедур обработки прерываний и для их восстановления при возвратах из прерываний.

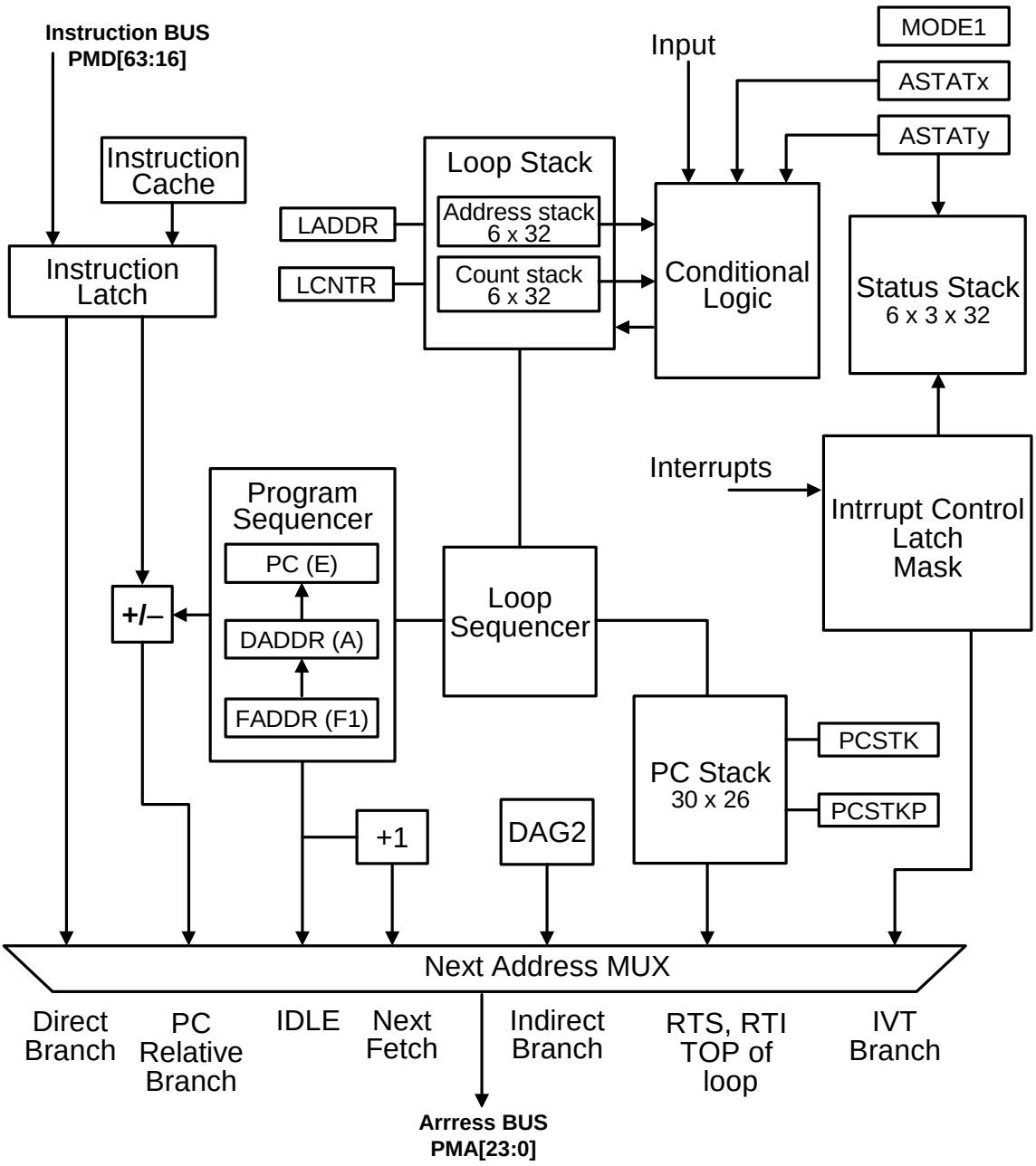


Рис. 4.12. Структурная схема программного автомата

Адрес выбираемой из памяти инструкции определяется по одному из источников, связанных с соответствующими входами мультиплексора сле-

дующего адреса (Next Address MUX). Адрес выбираемой из PM команды определяется:

- по значению программного счётчика PC;
- по адресу Next Fetch, полученному посредством линейного наращивания адреса опережающей выборки;
- по адресу Direct Branch, указанному в команде (в команде прямого перехода или вызова);
- по адресу Indirect Branch косвенного перехода или вызова, указанному в индексном регистре генератора адресов данных DAG2;
- по адресу перехода относительно PC (PC Relative Branch);
- по взятому из стека PC адресу возврата из подпрограммы по команде RTS или процедуры обработки прерывания по команде RTI;
- по адресу текущей команды, когда происходит откат (восстановление) прежнего адреса и возникает необходимость перезагрузки конвейера;
- по адресу TOP of loop первой команды исполняемого цикла DO UNTIL, хранящемуся стеке цикла;
- по вектору прерывания (IVT Branch), выдаваемому контроллером прерываний Interrupt Control и находящемуся в таблице прерываний IVT (Interrupt Vector Table).

Операции по определению исполнительного адреса PM распределены на 5-уровневый конвейер. Последовательность конвейерных операций выглядит следующим образом:

- уровень 1 (Fetch1) – на этом уровне в память программ направляется адрес, сформированный одним из источников, подключённым к мультиплексору следующего адреса. Адрес выборки берётся из регистра FADDR;
- уровень 2 (Fetch2) – на этом уровне выбранная команда фиксируется в регистре команд;
- уровень 3 (Decode) – принятая инструкция декодируется и выявляются условия её выполнения. Декодированный адрес сохраняется в регистре DADDR;
- уровень 4 (Address) – определяется адрес и условия выполнения перехода/вызова. Если условие выполнено, то адрес перехода/вызова передаётся на уровень Fetch1;
- уровень 5 (Execute) – на этой стадии проходящая по конвейеру инструкция выполняется. Адрес помещается в программный счётчик, содержимое которого увеличивается на 1. Результат операций сохраняется в регистре или из регистра переписывается в память. При обработке запроса прерывания на уровень Fetch1 передаётся адрес соответствующего элемента таблицы прерываний IVT

При определении адреса следующей команды проверяются корректность вычислений и текущее состояние процессора. Программный автомат управляет процессом формирования адреса выборки и его декодирования, используя регистры FADDR (*Fetch Address Resistor*) и DADDR (*Decode Address Resistor*) и контролирует содержимое программного счётчика PC (*Program Counter*).

Переходы, вызовы и возвраты из процедур могут быть условными и безусловными. В частности, команда условного перехода формируется по правилу

IF <*условие*> JUMP <*исп. адрес*>.

В поле <*условие*> показываются флаги ALU или MAC такие как

AZ – флаг нулевого результата ALU (ALU *zero*) или потери значимости при выполнении операций с плавающей точкой,

AV – флаг переполнения (ALU *overflow*),

AN – флаг нулевого результата (ALU *result negative*),

AC – флаг переноса при выполнении операций с фиксированной точкой (ALU *fixed-point carry*),

MN – флаг отрицательного результата умножителя (*Multiplier result negative*),

MV – флаг переполнения умножителя

и другие признаки. Например, переход по нулевому результату операции ALU происходит по команде

IF AZ JUMP <*исп. адрес*>.

При определении исполнительного адреса перехода/вызова используются все режимы адресации, включая непосредственную и косвенно-регистровую адресацию с использованием индексных регистров и регистров-модификаторов генератора адресов данных DAG2.

Программный автомат поддерживает исполнение команд перехода

IF <*условие*> JUMP <*исп. адрес*>, <*операция*>;

IF <*условие*> JUMP <*исп. адрес*>, ELSE < *операция* >;

IF <*условие*> JUMP <*исп. адрес*>, ELSE < *операция и обмен данными с* DM>.

Условный переход по исполнительному адресу <*исп. адрес*> происходит, если условие перехода выполнено. Исполнительный адрес <*исп. адрес*> может быть указан в команде (прямой переход), либо в одном из индексных регистров генератора адреса данных (косвенно-регистровый переход). Аналогичным образом строятся команды вызовов.

Команда перехода может выполняться параллельно с работой вычислительных устройств и с пересылкой данных по шине DMD.

Есть две разновидности циклов: циклы организованные по счётчику (по заданному числу итераций) и циклы с завершением по условию.

Команды циклов DO UNTIL – это условные (с использованием счётчика цикла, в который перед началом загружается число итераций) или безусловные циклы. Все команды DO UNTIL содержат указатель на последнюю команду тела цикла в виде 12-разрядного *смещения Reladdr*12 *относительно* PC. При выполнении команды DO UNTIL процессор либо бесконечно (*Forever*) повторяет циклическую последовательность команд, либо до обнуления счётчика цикла. Условием завершения условного цикла является появление признака CE (*Counter Expired*) обнуления счётчика цикла Для выхода из любого цикла можно воспользоваться командами условных переходов.

При формировании команд циклов используется синтаксис

DO <*Reladdr*12> UNTIL [CE, *Forever*];

*Reladdr*12 – 12-разрядное смещение адреса последней команды в теле цикла относительно счётчика команд. Квадратными скобками обозначена возможность выбора перечисленных в них условий. Например, в цикле

```
        DO label UNTIL CE;
        команда 1;
        команда 2;
              ⋮
label:  команда N;
```

*Reladdr*12 соответствует адресу последней команды (команды N), обозначенной меткой label.

Когда процессор выполняет команду DO UNTIL, программный автомат помещает адрес последней команды и условие завершения цикла в стек окончания цикла. Адрес вершины цикла, являющийся адресом команды, следующей за командой DO UNTIL, помещается в стек начала цикла. При организации вложенных условных циклов DO UNTIL CE используется стек счётчика цикла. Если цикл организован как цикл с заданным числом итераций (цикл DO UNTIL CE), то всеми стеками управляет программный автомат. Однако такое управление отсутствует, если цикл завершается досрочно по команде условного перехода или вызова. Поэтому при выходе из цикла до его завершения необходимо отслеживать содержимое стеков программного автомата и корректировать это содержимое с помощью команд непосредственного доступа к стекам.

В связи с конвейерной обработкой программный автомат проверяет условие завершения цикла и декрементирует счётчик итераций всякий раз перед концом тела цикла. В зависимости от результата проверки состояния

счётчика итераций следующая выбранная команда находится либо вне тела цикла, либо по адресу возврата к его вершине. Если условие завершения цикла ложно, то программный автомат повторяет цикл, выбирая команду по адресу первой команды, сохранённому в стеке начала цикла. Если условие завершения истинно, то выбирается команда, следующая за последней командой цикла и извлекаются данные из стеков цикла.

Изменение содержимого стеков может осуществляться посредством явных или неявных операций. Неявными операциями со стеком являются: обращение к стеку при выполнении процессором команд перехода (CALL/RETURN и DO UNTIL) и ответ на прерывание. Явными операциями со стеком являются команды проталкивания в стек и выталкивания из стека – PUSH и POP.

При организации циклов по заданному числу итераций команда загрузки счётчика цикла и команда DO UNTIL могут выполняться как одна многофункциональная команда.

Преждевременный выход из цикла с использованием команд условных переходов предполагает явное обращение к стеку цикла и стеку счётчика с целью коррекции их содержимого.

Кэш команд

У процессоров ADSP-2136x нет физического разделения на память программ и память данных. Внутренняя память процессора делится на блоки, в которых могут быть размещены и программа, и данные. Внешняя память разбита на банки. Место нахождения программы и данных определяется тем, как память сконфигурирована. Поэтому при обращении к памяти могут возникать конфликтные ситуации, если при передаче данных по шине DM происходит обращение к тому блоку памяти, из которого процессор выбирает команду. Конфликтная ситуация при обращении к блоку возникает потому, что в один и тот же момент времени доступ к блоку можно получить только по одной шине.

Обычно программный автомат выбирает команду из памяти в каждом цикле. Иногда возникают ситуации, в которых одновременно (в одном цикле) инициируются обращения к данным и программе по шине PM. Возникает конфликт при обращении к шине. Это может произойти, если команда выбирается из памяти программ в то время, когда выполняется команда пересылки данных по шине данных PM.

Для устранения подобных конфликтных ситуаций процессор кэширует команды. Когда возникают проблемы с выборкой какой-то команды, процессор записывает эту команду в кэш для того, чтобы в будущем исключить возникновение конфликта. Программный автомат проверяет кэш команд при каждом обращении по шине данных PM. Если необходимая команда находится в кэше, то для её выборки не требуется обращения к памяти программ

и команда из кэша выбирается параллельно с обращением к данным памяти программы и без задержки. Если необходимой команды в кэше нет, то команда выбирается из памяти в цикле, следующем за обращением к данным памяти программы. Таким образом добавляется один непроизводительный цикл. Команда загружается в кэш, и если кэш разблокирован и не зафиксирован, то она будет доступна при повторных возникновениях подобного конфликта.

Режимом работы кэш-памяти можно управлять. За это отвечает соответствующий регистр процессора. В частности, использование кэша можно запретить или разрешить.

Процессор выполняет команды в конвейерном режиме. Если при выполнении команды, находящейся по адресу *n*, делается обращение к данным в PM и возникает конфликт при обращении к тому же блоку памяти, то этот конфликт связан с выборкой команды по адресу *n*+3, поскольку при последовательном выполнении программы идёт опережающая выборка и конвейерным выполнением команд. Как следствие, в кэше сохраняется команда с адресом *n*+3, а не та команда, для выполнения которой потребовалось обращение к данным памяти программы. При последовательном выполнении программы это не сказывается на эффективности работы кэш-памяти. Однако при ветвлениях и переходах вероятность кэш-промахов увеличивается.

Управление SISD и SIMD режимами

Ядро МП ADSP-2136x содержит два процессорных элемента (PEx и PEy), способных работать в режиме с одним потоком инструкций и двумя потоками данных. Управление работой процессорных элементов осуществляется путём установки или сброса соответствующих бит в регистре режима MODE1. Бит PEYEN (бит 21) в регистре MODE1 запрещает или разрешает работу процессорного элемента PEy. Если бит 21 равен 0, МП ADSP-2136x работает в SISD режиме с использованием только одного вычислительного элемента PEx. После установки бита 21 в единицу становится возможным использование обоих PEx и PEy элементов.

JTAG интерфейс

Для тестирования системы используется стандарт IEEE P1149 JTAG (*Joint Test Automation Group*) – последовательный интерфейс «Объединенной рабочей группы по автоматизации тестирования». Этот стандарт определяет метод поочередного сканирования состояний входа-выхода каждого компонента системы. Последовательный порт JTAG используется также внутрисхемным эмулятором для доступа к встроенной системе поддержки тестирования. Эмуляторы используют порт JTAG для текущего контроля и управления процессором, установленным на целевой печатной плате, при отладке

системы и ее программного обеспечения. Делается это с использованием программного симулятора в сочетании с устройством отладки, подключаемым к оному из последовательных или параллельных портов инструментальной ЭВМ.

Сканирование состояний входов/выходов (сканирование границ) позволяет разработчику системы проверять схему соединений на печатной плате с привлечением минимального количества специального контрольного оборудования. Сканирование возможно благодаря наличию возможности управления и отслеживания каждого входа и выхода на каждом кристалле с помощью набора последовательно просматриваемых защелок. Каждый вход и выход подсоединяется к своей защелке, а защелки соединяются в длинный регистр сдвига, так что данные могут считываться или записываться в них через последовательный тестовый порт.

Сканирование границ предусматривает разнообразные функции, которые можно выполнять для каждого входного и выходного сигнала. Каждый вход имеет защелку, которая контролирует значение входного сигнала, а также через неё могут вводиться данные в кристалл. Аналогично, каждый выход имеет защелку, которая контролирует значение выходного сигнала, и может также выводить данные вместо выходного значения. Для двунаправленных выводов возможна комбинация функций ввода и вывода. Взаимодействие с JTAG портом осуществляется с помощью сигналов:

- TCK (входной) – сигнал тактовой синхронизации операций тестовой логики. Используется для синхронизации данных в защелках просмотра и управляющей последовательности тестового конечного автомата;
- TMS (входной) – выбор тестового режима. Первичный управляющий сигнал. Синхронный по отношению к TCK. Последовательность значений на входе TMS задает текущее состояние порта тестирования;
- TDI/TCLK (входной/выходной) – вход тестовых данных или вход синхронизации. Отсюда последовательные входные данные, подаются в защелки просмотра;
- TDO/TDI (выходной/входной) – выход тестовых данных. На этот контакт поступают последовательные выходные данные, извлекаемые из защелок просмотра. Может быть запрограммирован для приема входных данных.

JTAG интерфейс используется также для загрузки во flash-память микроконтроллера программ.

Программируемый таймер

Программируемый таймер (Timer) в составе ядра служит:

- для отсчётов интервалов времени с заданным периодом повторения и

– для генерации периодически повторяющихся запросов прерывания с высоким или низким приоритетом. Запросы прерывания возникают всякий раз, когда таймер отработал очередной интервал времени. Одновременно с запросом активизируется сигнал Timer Expired на выделенном для него контакте микросхемы ЦПС.

Контрольные вопросы по изучению ЦПС семейства ADSP-2136x

1. В чём состоит особенность гарвардской архитектуры микропроцессоров семейства ADSP-2136x.
2. Какие устройства входят в состав микросхемы процессора.
3. Назовите вычислительные устройства процессорных элементов и перечислите выполняемые ими операции.
4. Перечислите основные операции ALU.
5. Перечислите основные операции умножителя-аккумулятора.
6. Перечислите основные операции сдвигателя.
7. Первичные и вторичные регистры регистровых файлов и генераторов адресов данных. В чём заключается особенность применения вторичных регистров.
8. Регистровая модель генераторов адресов данных.
9. Режимы адресации данных
10. Каким образом на языке Ассемблера обозначаются операции с фиксированной и плавающей точкой.
11. Программный автомат – предназначение, состав и принцип функционирования.
12. Перечислите команды, в выполнении которых участвует программный автомат.
13. Чем отличается последовательность выполнения команд переходов от последовательности выполнения команд вызовов.
14. Каким образом программный автомат определяет адрес выбираемой из памяти программы инструкции.
15. Каким образом осуществляет процессор обработку запросов прерывания от устройств ввода/вывода.
16. Однофункциональные и многофункциональные команды и их отображение в языке Ассемблера.
17. Какими средствами процессор поддерживает выполнение многофункциональных команд.
18. Каким образом организуется работа процессора в SISD и SIMD режимах.

5. Средства программирования цифровых платформ

Реализация цифровой обработки сигнала сводится к программированию сигнального процессора (контроллера), т.е. к занесению в его ПЗУ соответствующей алгоритму обработки программы, которая будет принимать поступающие от АЦП оцифрованные отсчёты сигнала $x(t)$. Фильтровая обработка соответствует рекурсивному (1.5) или нерекурсивному (1.15) алгоритмам. При спектральной обработке используется описанное в разделе 2 быстрое преобразование Фурье.

5.1. Программирование цифрового сигнального контроллера TMS320F28335

Программирование сигнального контроллера ведётся в среде *Code Composer Studio v 3.3* на языке Си.

В состав среды программирования входят редактор, компилятор, компоновщик и инструментальные средства отладки. Компилятор представляет собой узкоспециализированный компилятор языка Си. Загрузка программы в память контроллера осуществляется с помощью USB-программатора XDS-100, находящегося на отладочной плате.

Программирование в среде *Code Composer Studio v 3.3* начинается с загрузки или создания файла проекта, к имени которого добавляется расширение *.pjt. Обычно для всех входящих в проект файлов создаётся отдельный каталог (папка), так что место расположения файл-проекта находится по пути <папка проекта>\<имя проекта>.pjt.

В проекте необходимо создать или открыть модуль main.c, который содержит программу цифровой обработки сигнала.

Модуль main.c также содержит программные компоненты, необходимые для инициализации системы обработки. Если в микросхему процессора интегрирован аналого-цифровой преобразователь, то задаётся частота дискретизации аналогового сигнала, способ тактирования АЦП, устанавливаются диапазон и способ подачи аналогового сигнала на его входы. Те же действия должны выполняться также при использовании внешнего аналого-цифрового преобразователя. Характер этих действий зависит от способа подключения АЦП к микропроцессору и от особенностей его функционирования.

Обычно частота дискретизации определяется таймером, который периодически с заданной частотой генерирует сигналы запроса прерывания. В процедуру обработки запросов прерывания от таймера включаются команды считывания оцифрованных данных и запуска очередного цикла преобразования. Такой способ управления процессом аналого-цифрового преобразования использован при программировании сигнального контроллера

TMS320F28335. В этом случае программа инициализации системы обработки должна установить соответствующий режим работы таймера.

Сигнальный контроллер TMS320F28335 не имеет встроенного цифро-аналогового преобразователя. Поэтому ЦАП является внешним по отношению к МП устройством, и процесс инициализации должен включать настройку (конфигурацию) выходных контактов микросхемы контроллера, через которые устанавливается его связь с ЦАП.

После выполнения программирования функции обработки сигнала необходимо собрать проект и, если отсутствуют ошибки компилятора и компоновщика, загрузить полученный исполняемый код в память процессора с помощью встроенных загрузчика и отладчика.

5.2. Программирование цифрового сигнального процессора ADSP-21364

Программирование сигнального процессора выполняется в среде разработки *VisualDSP++ 5.0* на языке Си аналогично программированию контроллера TMS320F28335 в среде *Code Composer Studio*. Отличием является расширение файла проекта – в среде *VisualDSP++ 5.0* он имеет расширение .dpj.

6. Описание лабораторной установки

Для выполнения данной лабораторной работы используются: 2^x-канальный цифровой генератор сигналов произвольной формы Tektronix AFG3022B, 2^x-канальный цифровой осциллограф Tektronix TDS1001B, отладочная плата с сигнальным контроллером TMS320F28335 (либо с сигнальным процессором ADSP-21364), а также три программных модуля:

- программа параметрического синтеза цифровых ЦНП-фильтров;
- среда *Code Composer Studio* для программирования цифрового сигнального контроллера TMS320F28335 или среда Visual DSP++ для программирования цифрового сигнального процессора ADSP-21364;
- цифровой измеритель частотных характеристик фильтра.

Перед началом работы приборы должны быть включены, а программные модули загружены в оперативную память компьютера. Схема лабораторной установки представлена на рис. 6.1.

Программирование микропроцессоров осуществляется с помощью соответствующих встроенных программаторов через интерфейс JTAG.

Измерение частотных характеристик (ЧХ) фильтра осуществляется на реальном сигнале с помощью автоматизированной измерительной системы, включающей в себя 2^x-канальные цифровые генератор сигналов произвольной формы и осциллограф, а также разработанную в среде виртуальных при-

боров LabVIEW программу для генерации и обработки тестового сигнала, вычисления ЧХ и управления генератором и осциллографом.

Программа на персональном компьютере генерирует многочастотный сигнал, включающий в себя множество дискретных частот (взаимно ортогональных гармонических сигналов) из заданного пользователем интервала измерения частоты. Сгенерированный программой сигнал в цифровом виде загружается в память генератора сигналов произвольной формы. По команде запуска, поступающей с ПК, генератор воспроизводит записанный в память многочастотный сигнал в аналоговой форме, синхронно выводя его на два выхода (выход 1, выход 2). Сигнал с 1-го выхода генератора подаётся на 1-й вход цифрового осциллографа, а со 2-го выхода – на вход АЦП отладочной платы. Прошедший через реализованный на цифровом процессоре (ЦП) цифровой фильтр сигнал переводится в аналоговую форму с помощью ЦАП и поступает на 2-й вход осциллографа. Осциллограф оцифровывает сигналы одновременно на обоих входах и передаёт их в программу на ПК, которая вычисляет дискретные Фурье-преобразования этих тестовых сигналов.

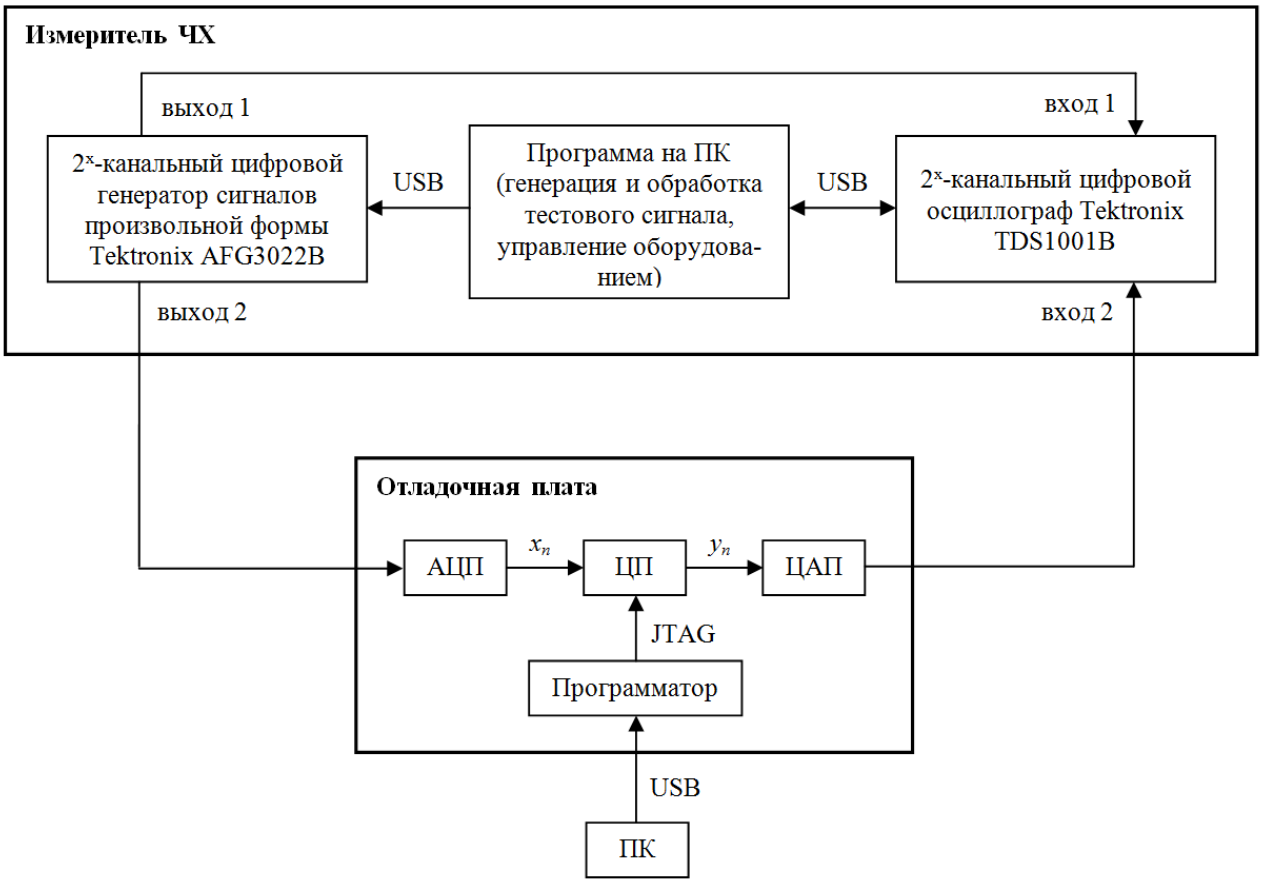


Рис. 6.1. Схема лабораторной установки

Комплексный коэффициент передачи цифрового устройства обработки от входа АЦП до выхода ЦАП вычисляется путём деления комплексного

спектра сигнала на входе 2 осциллографа на комплексный спектр сигнала на входе 1 (сигнал на входе 1 осциллографа дублирует сигнал на входе АЦП):

$$\dot{K}(k) = \frac{\dot{X}_{выход\ ЦАП}(k)}{\dot{X}_{вход\ АЦП}(k)} = \frac{\dot{X}_{вход\,2}(k)}{\dot{X}_{вход\,1}(k)}.$$

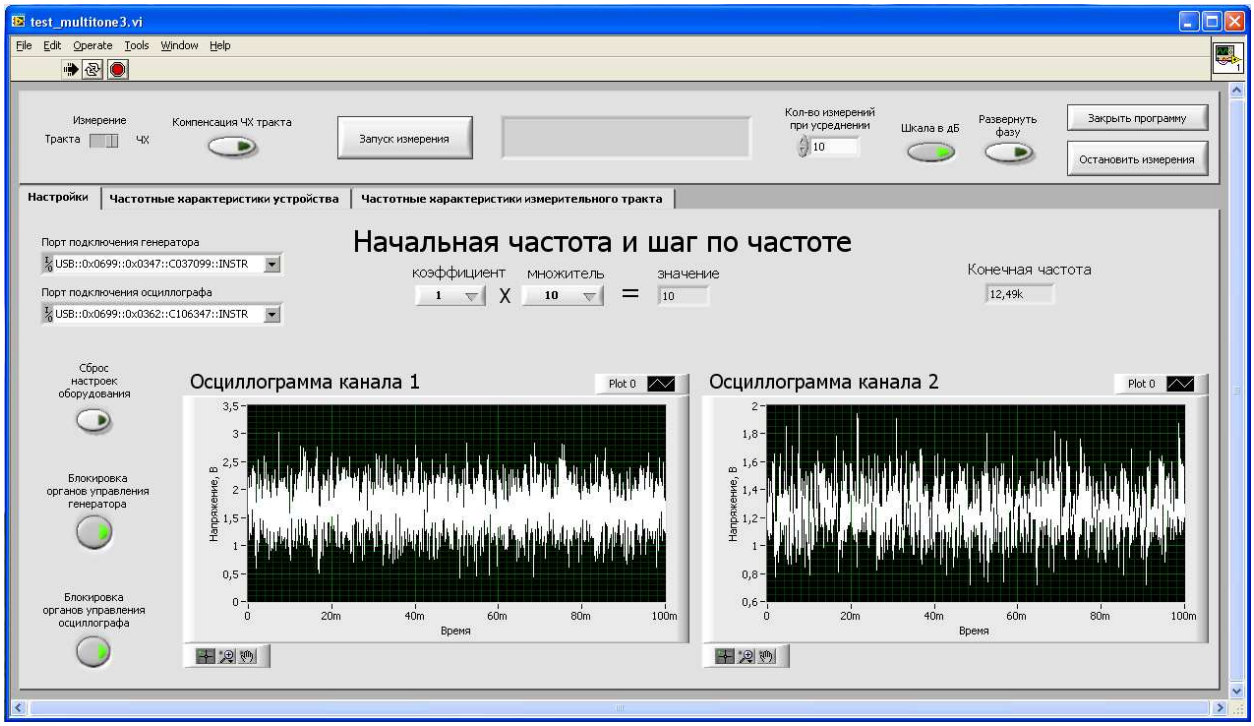
Генератор сигналов произвольной формы «проигрывает» тестовый сигнал непрерывно, т.е. за одним периодом тестового сигнала сразу без паузы следует второй и т.д. Благодаря этому установление колебаний в исследуемой системе (затухание переходных процессов) происходит до момента запуска осциллографа. На один период тестового сигнала приходится целое число периодов каждой гармонической составляющей, что обеспечивает непрерывность во времени каждого гармонического сигнала и позволяет избежать процедуры синхронизации момента запуска осциллографа с моментом начала нового периода тестового сигнала.

По окончании оцифровки осциллографом одного периода тестового сигнала и вычисления по приведённой выше формуле комплексного коэффициента передачи вычисляются и выводятся на графики амплитудно-частотная (АЧХ) и фазо-частотная (ФЧХ) характеристики тестируемого устройства. Осциллограммы сигналов на входе и выходе тестируемого устройства также отображаются на графиках. Любой график может быть скопирован в буфер обмена в графическом виде или в виде последовательностей числовых значений выборок средствами LabVIEW.

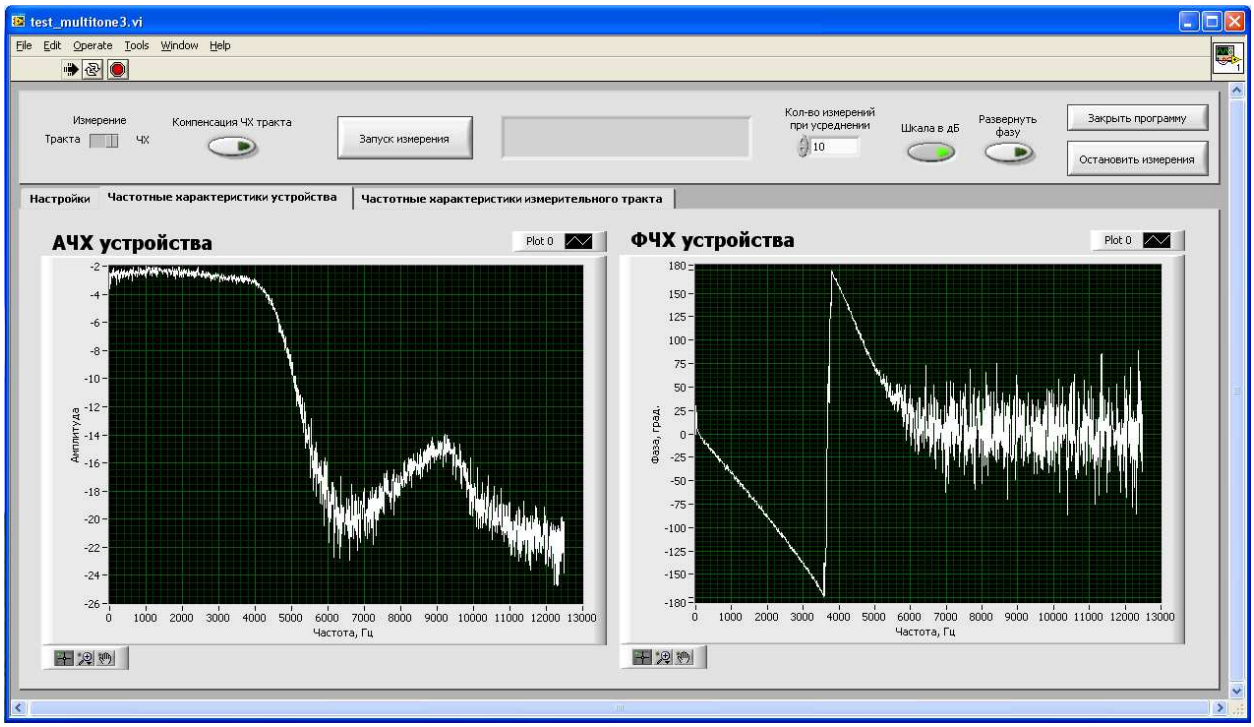
Вид интерфейса программного модуля измерителя частотных характеристик показан на рис. 6.2 а), б).

При использовании программы необходимо выполнить следующую последовательность действий:

- включить осциллограф и генератор сигналов произвольной формы и дождаться перехода их в состояние готовности;
- открыть программу;
- на вкладке «Настройки» в поле «Порт подключения генератора» нажать на стрелку в правом краю поля и выбрать команду «Refresh»;
- ещё раз нажать на стрелку в правом краю поля «Порт подключения генератора» и выбрать строку, содержащую серийный номер генератора (данный номер указан на задней панели генератора);
- аналогичным образом выбрать строку с серийным номером осциллографа в поле «Порт подключения осциллографа»;
- запустить программу, нажав на кнопку с изображением стрелки, которая находится сверху под строкой меню;
- на вкладке «Настройки» с помощью полей «коэффициент» и «множитель» выбрать начальную частоту в измеряемых частотных характеристиках (при этом шаг изменения частоты всегда совпадает с начальной частотой – это особенность используемого способа измерения);



а)



б)

Рис. 6.2. Интерфейс программы

- выбрать необходимые значения параметров отображения АЧХ и ФЧХ с помощью кнопок «Шкала в дБ» (для вертикальной шкалы графика АЧХ) и «Развернуть фазу» (для вертикальной шкалы графика ФЧХ);
- установить необходимое количество измерений при усреднении для уменьшения влияния шума в поле «Кол-во измерений при усреднении» (если выбрано значение 1, то усреднения не происходит);
- нажать на кнопку «Запуск измерения» и дождаться окончания процесса измерения (количество выполненных на текущий момент измерений отображается в поле сообщений справа от кнопки «Запуск измерения»);
- после окончания измерения вид частотных характеристик будет представлен на графиках вкладки «Частотные характеристики устройства»;
- при необходимости изменения настроек программы (кроме изменения портов подключения генератора и осциллографа) данные изменения можно выполнить, не останавливая программу (однако, если программа в данный момент выполняет измерение, то процесс измерения вначале необходимо остановить кнопкой «Остановить измерения»), после чего для запуска измерений снова нажать кнопку «Запуск измерения».

7. Задание и порядок выполнения работы

7.1. Синтез и оценка селективных свойств рекурсивных и нерекурсивных цифровых фильтров различного порядка

Данное задание выполняется с помощью программы параметрического синтеза цифровых ЦНП-фильтров. Для выполнения задания необходимо синтезировать ЦНП-фильтры различной частотной селекции, а затем оценить σ - среднеквадратичную ошибку (СКО)

$$\sigma = \sqrt{\frac{1}{p} \cdot \sum_{n=1}^{p} \left[Y_n(IX) - Y_n^{T} \right]^2}$$

выполнения заданных требований (требуемой АЧХ фильтра). Значение СКО выводится на панель синтеза компьютерной программы. Частота дискретизации Fд=10 кГц.

Синтез полосового фильтра

1. Осуществить синтез КИХ-фильтра второго порядка. Порядок выполнения задания в пакете синтеза следующий:
   - загрузить файл **FIR_2p.top** исходных данных к синтезу;
   - в функциональном редакторе (ФР) ввести требуемую АЧХ фильтра (рис. 7.1); последовательность работы в ФР приведена выше;
   - синтезировать фильтр, зафиксировать СКО выполнения заданных требований;

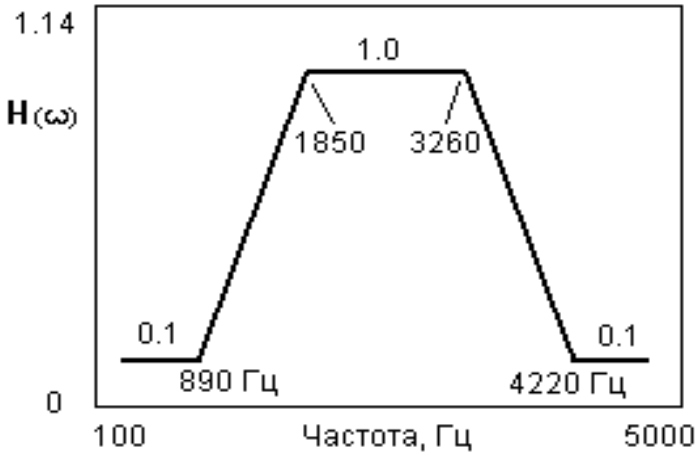


Рис. 7.1. Требуемая АЧХ полосового фильтра

2. В той же последовательности синтезировать полосовой КИХ-фильтр четвёртого порядка (файл данных **FIR_4p.top**).

3 Синтезировать полосовой БИХ-фильтр второго порядка (файл исходных данных **IIR_2p.top**) по той же АЧХ (рис. 7.1).

4 В той же последовательности синтезировать полосовой БИХ-фильтр четвёртого порядка (файл исходных данных **IIR_4p.top**). После чего в модуле анализа программы подробно исследовать полученное оптимальное решение. Задавая различные значения коэффициентов фильтра, оценить изменение его АЧХ, получить неустойчивое решение задачи. Для оптимальных коэффициентов сохранить график АЧХ синтезированного БИХ-фильтра четвёртого порядка для отчёта как на всём главном интервале изменения цифровой частоты $f=f_N$ (f_N=0.5Fд – частота Найквиста), так и на интервале частот вплоть до f=Fд, f=2Fд и f=4Fд. Сформировать протокол решения данной задачи. Сохранить для отчёта найденные значения оптимальных коэффициентов полосового рекурсивного фильтра четвёртого порядка.

Сравнивая СКО реализации заданной АЧХ, оценить селективные возможности рекурсивных и нерекурсивных полосовых фильтров различного порядка.

Синтез гауссова фильтра

Нормированная резонансная характеристика для гауссовой кривой определяется следующим образом:

$$y\ (\xi) = e^{-\frac{\xi^2}{\alpha}},$$

где $\xi = f - f_0$ - абсолютная расстройка от резонансной частоты, а параметр α определяет нормированную полосу пропускания гауссовой кривой:

$$\alpha = \frac{\Pi^2}{4\ \ln\sqrt{2}},$$

здесь *П* – абсолютная полоса пропускания по уровню 0,7. Выполнение задания в пакете синтеза осуществляется в следующей последовательности.

1. Осуществить синтез гауссова КИХ-фильтра второго порядка (файл исходных данных **FIR_2p.top**). Зафиксировать СКО. Для ввода требуемой гауссовой АЧХ (рис. 7.2) использовать панель шаблонов характеристик функционального редактора либо файл характеристики **Gauss.x**.

2. В той же последовательности синтезировать гауссов КИХ-фильтра четвёртого порядка (файл данных **FIR_4p.top**).

3. Синтезировать гауссов БИХ-фильтра второго порядка (файл исходных данных **IIR_2p.top**) по той же АЧХ (рис. 7.2).

4. В той же последовательности синтезировать гауссов БИХ-фильтра четвёртого порядка (файл исходных данных **IIR_4p.top**). Сохранить для отчёта график АЧХ и найденные значения оптимальных коэффициентов рекурсивного гауссового фильтра четвёртого порядка.

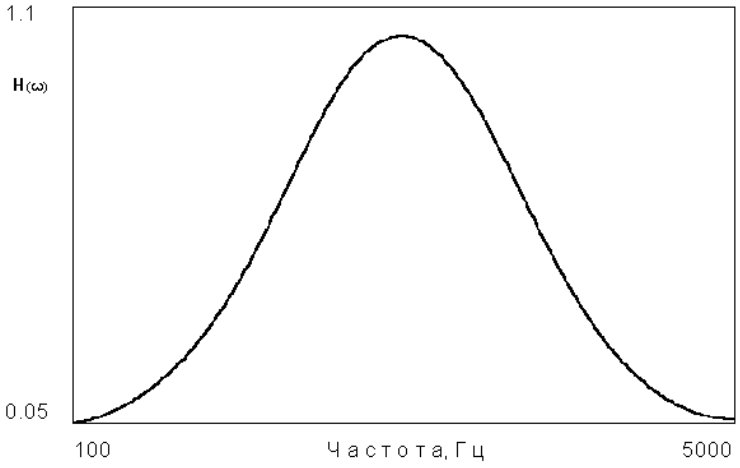


Рис. 7.2. Требуемая АЧХ гауссова фильтра

Сравнивая СКО реализации заданной АЧХ, оценить селективные возможности рекурсивных и нерекурсивных гауссовых ЦНП-фильтров различного порядка.

7.2. Многофункциональный синтез рекурсивного фильтра нижних частот с линейной фазой

В данном задании необходимо осуществить синтез рекурсивного ЦНП-фильтра сначала по одной характеристике (АЧХ), а затем по двум его частотным характеристикам (АЧХ и ФЧХ) для частоты дискретизации Fд=10 кГц. Порядок выполнения задания следующий.

1. Загрузить файл **IIR_4p.top** исходных данных к синтезу ФНЧ.
2. В функциональном редакторе заказать два окна синтеза и ввести требуемую АЧХ фильтра (рис. 7.3) в первое окно, а требуемую ФЧХ (рис. 7.4) - во второе. Как видно, данная характеристика определяет требо-

вание линейности ФЧХ в полосе пропускания цифрового фильтра нижних частот.

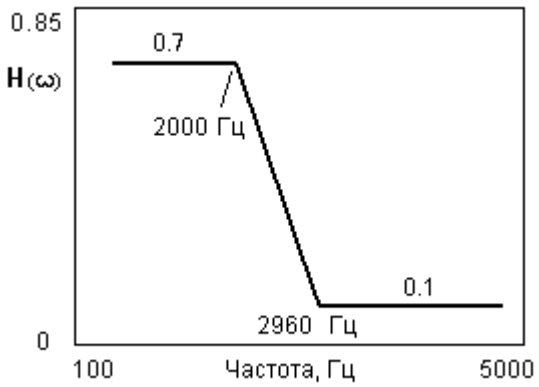


Рис. 7.3. Требуемая АЧХ

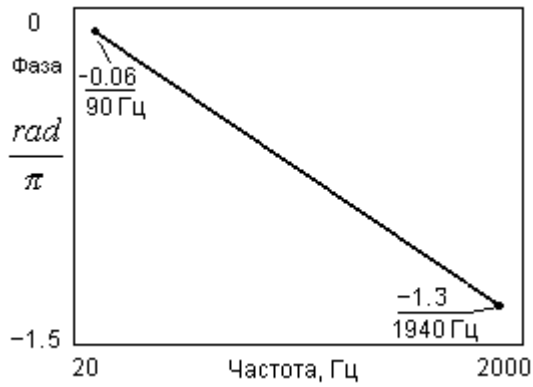


Рис. 7.4. Требуемая ФЧХ

3. Задать вес β_1=0,5 для окна АЧХ и вес β_2=0 для окна ФЧХ и синтезировать фильтр. В модуле анализа исследовать полученное оптимальное решение. Оценить фазовые искажения в полосе пропускания (от 10 до 2000 Гц) фильтра по синтезу. Сохранить график АЧХ и ФЧХ синтезированного фильтра для отчёта. Сформировать протокол решения задачи.
4. В модуле *IIR_Filtr.c* проекта, открытого в среде программирования, написать тело функции расчёта отклика рекурсивного ЦНП-фильтра согласно приведённой в разделе 1 его модели (см. образец в приложении 2).
5. Из протокола синтеза ввести в программу реализации фильтра на сигнальном процессоре (контроллере) оптимальные коэффициенты синтезированного фильтра. Осуществить трансляцию и загрузку программы в сигнальный процессор (контроллер). Инициализировав работу цифрового устройства, с помощью измерителя частотных характеристик произвести измерение АЧХ и ФЧХ синтезированного фильтра в интервале частот от **f**min=10 Гц до **f**max=12490 Гц. Отмасштабировать ось частот ФЧХ фильтра таким образом, чтобы на графике отображался только диапазон частот в полосе пропускания (**f**min=40 Гц, **f**max=2000 Гц). Сохранить графики измерений для отчёта.
6. Перезагрузив исходные данные, задать вес β_1=0,5 для окна АЧХ и вес β_2=1 для окна ФЧХ и синтезировать фильтр по двум требуемым характеристикам. Найденные значения коэффициентов сохранить для отчёта. Для новых значений оптимальных коэффициентов произвести измерение АЧХ и ФЧХ фильтра, выполнив п.п. 4 и 5 настоящего задания. Оценить амплитудные и фазовые искажения в полосе пропускания фильтра. Сохранить графики измерения АЧХ и ФЧХ синтезированного фильтра для задачи многофункционального синтеза.

7.3. Алгоритм быстрого преобразования Фурье

Задание предполагает реализацию алгоритма БПФ на сигнальном процессоре (контроллере). Порядок выполнения задания приведён ниже.

1. В модуле *FFT.c* проекта, открытого в среде программирования, написать тело функции, реализующей алгоритм БПФ (см. образец в приложении 2).
2. Скомпилировать проект, загрузить исполняемый код в память процессора и запустить программу.
3. Отключить подачу сигнала на выход 2 генератора сигналов произвольной формы кнопкой «On» на панели генератора (погасшая кнопка означает, что сигнал отключен), настроить 2-й канал генератора на генерацию непрерывного синусоидального сигнала с размахом напряжения (удвоенной амплитудой) 3 Вольта, с постоянной составляющей, равной 1,65 В, и частотой 3 кГц.
4. Настроить осциллограф следующим образом: на вход 1 подать синхросигнал с выхода отладочной платы, на вход 2 – сигнал с ЦАП; установить синхронизацию осциллографа по заднему фронту (спаду) импульса синхросигнала в канале 1, масштаб по горизонтали установить равным 10 мс/деление.
5. Включить подачу сигнала на выход 2 генератора сигналов произвольной формы кнопкой «On» на панели генератора; на осциллографе подобрать подходящий масштаб по вертикали в каждом канале и сдвинуть по горизонтали осциллограммы таким образом, чтобы спад импульса синхронизации в канале 1 находился у левой границы экрана.
6. Сохранить в файл на USB носитель снимки экрана осциллографа для нескольких значений частоты синусоидального сигнала (в диапазоне от 1 до 12 кГц).
7. Перестроить генератор на генерацию периодической последовательности прямоугольных импульсов (значение размаха напряжения и постоянной составляющей оставить такими же, как для синусоидального сигнала) и сохранить снимки экрана осциллографа на USB носителе при нескольких значениях частоты следования импульсов.

8. Список литературы

8.1. Основная литература

1. Мину М. Математическое программирование. Теория и алгоритмы. М., Наука, 1990, 488 с.
2. Воинов Б.С., Бугров В.Н., Воинов Б.Б. Информационные технологии и системы: поиск оптимальных, оригинальных и рациональных решений. М., Наука, 2007, 730 с.
3. Бугров В.Н. Проектирование цифровых фильтров методами целочисленного нелинейного программирования. // Вестник ННГУ, 2009, № 6. с. 61 – 70.
4. Бугров В.Н., Лупов С.Ю., Земнюков Н.Е., Корокозов М.Н. Дискретный синтез цифровых рекурсивных фильтров. // Вестник ННГУ, 2009, № 2. с. 76 – 82.
5. Богатырев Ю.К., Бугров В.Н., Воронков Ю.В. Компьютерный анализ и синтез радиотехнических устройств. Учебное пособие. // Н.Новгород, изд. НГТУ, 1996, 96 с.
6. Кривошеев В.И. Цифровая обработка сигналов. Учебное пособие. //Н.Новгород: Изд.ННГУ, 2006 – 207 с.
7. Антонью А. Цифровые фильтры: анализ и проектирование. М., Радио и Связь , 1983.
8. Электронная версия перечня технических характеристик сигнального контроллера TMS320F28335 / sprs439m.pdf (http://www.ti.com/litv/pdf/sprs439m).
9. Электронная версия перечня технических характеристик сигнального процессора ADSP-21364 ADSP-21362_21363_21364_21365_21366.pdf (http://www.analog.com/static/imported-files/data_sheets/ADSP-21362_21363_21364_21365_21366.pdf).
10. Шкелев Е.И. Электронные цифровые системы и микропроцессоры. Учебное пособие. //Н.Новгород: Изд. ННГУ, 2004 – 152 с.
11. Знакомство с сигнальными контроллерами TMS320C28x (электронная версия) / spraam0a.pdf (http://www.ti.com/litv/pdf/spraam0a)
12. Знакомство с процессорами SHARC (электронная версия) / GettingStartedwithSharcProcessors.pdf (http://www.analog.com/static/imported-files/tech_docs/GettingStartedwithSharcProcessors.pdf)
13. Code Composer Studio Development Tools v3.3. Getting Started Guide / spru509h.pdf (http://www.ti.com/litv/pdf/spru509h)
14. Code Composer User’s Guide / spru296.pdf (http://www.ti.com/litv/pdf/spru296)

15. VisualDSP++ 5.0 Getting Started Guide / 10910966550_gs_guide.pdf (http://www.analog.com/en/processors-dsp/content/vdsp_getting_started_guide/fca.html)

8.2. Дополнительная литература

16. Баскаков С.И. Радиотехнические цепи и сигналы. – М.: Высшая школа, 2005 – 270 с..
17. Бугров В.Н., Воронков Ю.В. Формализация задачи структурно-параметрического синтеза радиоэлектронных систем. // Радиотехника, 2001, № 9, с. 57 - 62.

9. Содержание отчета по лабораторной работе

1. Модель и структура звена рекурсивного и нерекурсивного цифрового ЦНП-фильтра.
2. Постановка задачи синтеза рекурсивного ЦНП-фильтра.
3. Постановка задачи синтеза нерекурсивного ЦНП-фильтра.
4. Описание блок-схемы и компьютерной программы синтеза фильтра.
5. Краткое описание структуры и возможностей сигнального процессора (контроллера).
6. Результаты синтеза и оценка селективных возможностей рекурсивных и нерекурсивных ЦНП-фильтров различного порядка. Частотные характеристики синтезированных цифровых фильтров.
7. Результаты многофункционального синтеза рекурсивного фильтра нижних частот. Измерения АЧХ и ФЧХ синтезированного фильтра. Оценка амплитудных (СКО) и фазовых искажений в полосе пропускания фильтра.
8. Результаты измерения спектров синусоидальных и прямоугольных сигналов.
9. Листинги основных программных модулей *IIR_Filtr.c* и *FFT.c*.
10. Интерпретация результатов, общие выводы по работе .

В приложении 4 приводится вешний вид титульного листа отчёта по данной лабораторной работе.

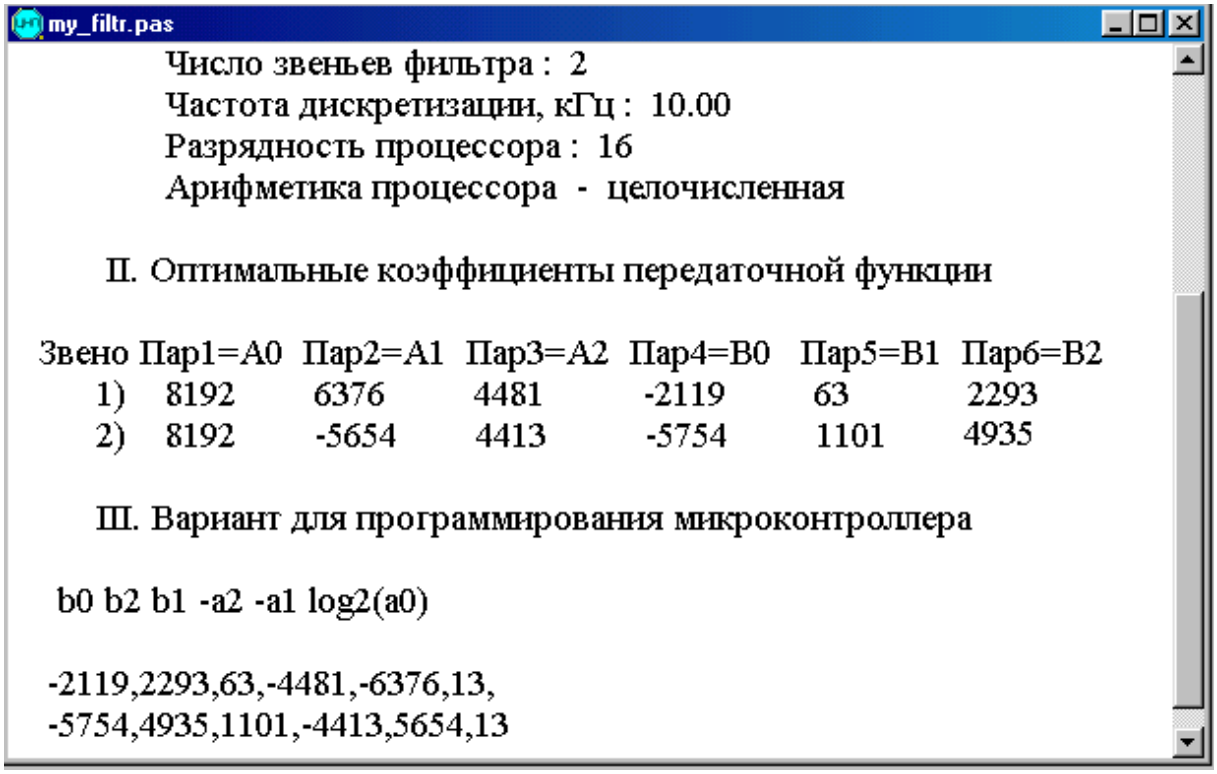
10. Контрольные вопросы

1. Назначение и определение цифрового фильтра.
2. Какие характеристики имеет цифровой фильтр в частотной области ?
3. Современные требования к устройствам цифровой фильтрации.
4. В чём особенности математического программирования как методологии проектирования радиоэлектронных устройств ?

5. Какие достоинства имеют цифровые фильтры, спроектированные методологией целочисленного нелинейного программирования ?
6. Модели рекурсивных ЦНП-фильтров. Структура их построения и условие устойчивости.
7. Модели нерекурсивных ЦНП-фильтров. Структура их построения.
8. Постановка задачи ЦНП-синтеза фильтра.
9. В чём состоит методология поискового решения экстремальной задачи синтеза цифровых фильтров ?
10. Какие основные требования предъявляются к алгоритмам поисковой минимизации ?
11. Общая структура программы синтеза ЦНП-фильтра.
12. Сигнальный процессор и сигнальный контроллер. Их структуры и характеристики.
13. Методика измерения характеристик цифрового фильтра на реальном сигнале.
14. Особенности алгоритма БПФ.

Приложение 1

Протокол синтеза полосового рекурсивного ЦНП-фильтра четвёртого порядка



Приложение 2

Образцы программного кода для цифровых сигнальных процессоров

1. Рекурсивный фильтр

```
#define IIR16_NBIQ   2       // число звеньев

// коэффициенты звеньев в порядке B0i, B2i, B1i, A2i, A1i, log_2(A0i):
const short IIR16_COEFF[6*IIR16_NBIQ+1]={
    1618,-4234,-2798,-4632,2007,13,
     -1069,-1677,-1940,-1736,6517,13
                                        };

int DBuffer[2*IIR16_NBIQ+1];      // линии задержки
int EBuffer[2*IIR16_NBIQ+1];

int IIR_Filtr(int Data) {  // функция, реализующая рекурсивный фильтр
  int32 temp;
  short *COEFF=(short*)IIR16_COEFF;      // указатель на коэффициенты
  short *D = (short*)DBuffer;         // указатель на линию задержки
  short *E = (short*)EBuffer;
  short Xc, pvalue;
  int i;

  pvalue = (short)Data;
  for(i=0;i<IIR16_NBIQ;i++)
  {
     Xc = pvalue;
     temp = (long)(*COEFF++)*Xc + (long)(*COEFF++)*(*D++) +
(long)(*COEFF++)*(*D--) + (long)(*COEFF++)*(*E++) +
(long)(*COEFF++)*(*E--);
     *D++ = *D;
     *E++ = *E;
     pvalue = (short)(temp>>*COEFF++);
     *D++ = Xc;
     *E++ = pvalue;
  }
   return (int) pvalue;
}
```

2. Алгоритм БПФ

```
#define FFT_WINDOW      512 // ширина окна БПФ

float XX[FFT_WINDOW+1];
float YY[FFT_WINDOW+1];
int i=1;
float max;
```

```
void rfft(float X[],int N) //функция, реализующая БПФ
{
int I,I0,I1,I2,I3,I4,I5,I6,I7,I8, IS,ID;
int J,K,M,N2,N4,N8;
float A,A3,CC1,SS1,CC3,SS3,E,R1,XT;
float T1,T2,T3,T4,T5,T6;

M=(int)(log(N)/log(2.0));               /* N=2^M */

/* ----Digit reverse counter----------------------------------- */
J = 1;
for(I=1;I<N;I++)
        {
        if (I<J)
                {
                XT    = X[J];
                X[J]  = X[I];
                X[I]  = XT;
                }
        K = N/2;
        while(K<J)
                {
                J -= K;
                K /= 2;
                }
        J += K;
        }

/* ----Length two butterflies---------------------------------- */
IS = 1;
ID = 4;
do
        {
        for(I0 = IS;I0<=N;I0+=ID)
                {
                I1    = I0 + 1;
                R1    = X[I0];
                X[I0] = R1 + X[I1];
                X[I1] = R1 - X[I1];
                }
        IS = 2 * ID - 1;
        ID = 4 * ID;
        }while(IS<N);

/* ----L shaped butterflies------------------------------------ */
N2 = 2;
for(K=2;K<=M;K++)
        {
        N2    = N2 * 2;
        N4    = N2/4;
        N8    = N2/8;
        E     = (float) 6.2831853071719586f/N2;
        IS    = 0;
        ID    = N2 * 2;
        do
                {
```

```
                for(I=IS;I<N;I+=ID)
                        {
                        I1 = I + 1;
                        I2 = I1 + N4;
                        I3 = I2 + N4;
                        I4 = I3 + N4;
                        T1 = X[I4] +X[I3];
                        X[I4] = X[I4] - X[I3];
                        X[I3] = X[I1] - T1;
                        X[I1] = X[I1] + T1;
                        if(N4!=1)
                                {
                                I1 += N8;
                                I2 += N8;
                                I3 += N8;
                                I4 += N8;
                                T1 = (X[I3] +
X[I4])*.707106781186547524f;
                                T2 = (X[I3] -
X[I4])*.707106781186547524f;
                                X[I4] = X[I2] - T1;
                                X[I3] = -X[I2] - T1;
                                X[I2] = X[I1] - T2;
                                X[I1] = X[I1] + T2;
                                }
                        }
                        IS = 2 * ID - N2;
                        ID = 4 * ID;
                }while(IS<N);
        A = E;
        for(J= 2;J<=N8;J++)
                {
                A3 = 3.0 * A;
                CC1   = cos(A);
                SS1   = sin(A);  /*typo A3--really A?*/
                CC3   = cos(A3); /*typo 3--really A3?*/
                SS3   = sin(A3);
                A = (float)J * E;
                IS = 0;
                ID = 2 * N2;
                do
                        {
                        for(I=IS;I<N;I+=ID)
                                {
                                I1 = I + J;
                                I2 = I1 + N4;
                                I3 = I2 + N4;
                                I4 = I3 + N4;
                                I5 = I + N4 - J + 2;
                                I6 = I5 + N4;
                                I7 = I6 + N4;
                                I8 = I7 + N4;
                                T1 = X[I3] * CC1 + X[I7] * SS1;
                                T2 = X[I7] * CC1 - X[I3] * SS1;
                                T3 = X[I4] * CC3 + X[I8] * SS3;
                                T4 = X[I8] * CC3 - X[I4] * SS3;
                                T5 = T1 + T3;
                                T6 = T2 + T4;
```

```
                                T3 = T1 - T3;
                                T4 = T2 - T4;
                                T2 = X[I6] + T6;
                                X[I3] = T6 - X[I6];
                                X[I8] = T2;
                                T2    = X[I2] - T3;
                                X[I7] = -X[I2] - T3;
                                X[I4] = T2;
                                T1    = X[I1] + T5;
                                X[I6] = X[I1] - T5;
                                X[I1] = T1;
                                T1    = X[I5] + T4;
                                X[I5] = X[I5] - T4;
                                X[I2] = T1;
                                }
                        IS = 2 * ID - N2;
                        ID = 4 * ID;
                        }while(IS<N);
                }
        }
return;
}
```

Приложение 3

Замечания при работе в среде разработки Code Composer Studio v. 3.3

Типичный порядок работы:

1. Запустить Code Composer Studio v3.3
2. Открыть проект (Project->Open…) по адресу <папка проекта>\<имя проекта>.pjt.
3. В дереве проекта в папке Source открыть файл main.c (IIR_Filtr.c, FFT.c).
4. {изменение кода}
5. Скомпилировать проект (Project->Build).
6. Запустить проект.

Запуск проекта:

1. Подать питание на плату (тумблер на плате), включить генератор, осциллограф.
2. Наладить соединение между компьютером и процессором (Debug->Connect).
3. Загрузить код программы в память процессора (File->Load Program) по адресу <папка проекта>\Debug\<имя проекта>.out.
4. Начать исполнение программы (Debug->Run)

Для перезапуска проекта необходимо переподключить процессор (Debug->Disconnect, выключить и включить тумблер питания на отладочной плате), далее повторить пункты 2-4.

Отладка проекта:

После изменения текста программы необходимо:

1. Сохранить изменения (Project->Save)
2. Скомпилировать проект (Project->Build)
3. Переподключить процессор (Debug->Disconnect, тумблер питания, Debug->Connect)
4. Загрузить код программы в память процессора (File->Load Program)

При необходимости открыть новый проект следует закрыть текущий (Project->Close) и закрыть окна, относящиеся к этому проекту.

Примечания:

Не рекомендуется изменять части кода, открытые для изменения командами EALLOW – EDIS, а также переносить защищённые участки кода между файлами, даже правильный перенос без изменений может привести к нежелательным последствиям!

При возникновении проблем с нехваткой памяти можно переназначить области памяти, блоки не должны перекрываться, либо выходить за пределы адресного пространства. Распределение памяти производится в модуле 28335_RAM_lnk_M.cmd. Возможные проблемы могут возникнуть при недостатке памяти для программы (в моих проектах секция RAML123 на листе памяти PAGE 0) либо памяти стека (секция RAMLSTACK на листе памяти PAGE 0).

При возникновении проблем с инициализации целочисленной либо вещественной арифметики стоит проверить наличие подключенных библиотек в меню настройки компиляции проекта (Project->Build Options) и правильности заданного пути к используемой библиотеке (для вещественной арифметики - rts2800_fpu32.lib, для целочисленной - rts2800_ml.lib).

Приложение 4

Нижегородский государственный университет
им. Н.И.Лобачевского

Радиофизический факультет

Кафедра радиотехники

ЦИФРОВАЯ ОБРАБОТКА СИГНАЛОВ НА СИГНАЛЬНОМ ПРОЦЕССОРЕ

Отчет по лабораторной работе

Исполнители:
студенты IV курса гр.446
Петров А.А.
Андреева И.Н.

Работу принял:
доцент Бугров В.Н.

Н.Новгород, 2012

Оглавление