Московский государственный технический университет
имени Н.Э.Баумана

БИБЛИОТЕКА ИНЖЕНЕРА-РАЗРАБОТЧИКА
ЭЛЕКТРОННЫХ СИСТЕМ

УЧЕБНОЕ ПОСОБИЕ

Э.В.МЫСЛОВСКИЙ, А.И.ВЛАСОВ, А.С.КУЗНЕЦОВ

ЦИФРОВЫЕ СИГНАЛЬНЫЕ ПРОЦЕССОРЫ

ЧАСТЬ 2:
ЦИФРОВЫЕ СИГНАЛЬНЫЕ ПРОЦЕССОРЫ С
ФИКСИРОВАННОЙ ТОЧКОЙ
СЕМЕЙСТВА ADSP21xx

Кафедра ИУ4
МГТУ
им.Н.Э.Баумана
http://iu4.bmstu.ru

Москва
МГТУ им.Н.Э.Баумана
2003

Московский государственный технический университет имени Н.Э. Баумана

Э.В. Мысловский, А.И. Власов, А.С. Кузнецов

ЦИФРОВЫЕ СИГНАЛЬНЫЕ ПРОЦЕССОРЫ

ЧАСТЬ 2:

ЦИФРОВЫЕ СИГНАЛЬНЫЕ ПРОЦЕССОРЫ С ФИКСИРОВАННОЙ ТОЧКОЙ СЕМЕЙСТВА ADSP21XX

Рекомендовано в качестве учебного пособия по дисциплинам "Микропроцессоры в системах управления" и "Цифровые сигнальные процессоры".

Москва

МГТУ им.Н.Э.Баумана

2003 г.

УДК: 621.396.6
ББК: 32.852
П18

Рецензент: Шитько Ю.М., МГАПИ

Мысловский Э.В., Власов А.И., Кузнецов А.С.

ЦИФРОВЫЕ СИГНАЛЬНЫЕ ПРОЦЕССОРЫ С ФИКСИРОВАННОЙ ТОЧКОЙ СЕМЕЙСТВА ADSP21XX: Учебное пособие. Часть 2. - М.: МГТУ им.Н.Э.Баумана, кафедра ИУ4, 2003. -75 с., ил.

Рассмотрены вопросы архитектуры и методов проектирования программного обеспечения для систем на основе цифровых сигнальных процессоров с плавающей точкой. Основное семейство уделено цифровым сигнальным процессорам фирмы Analog Devices, семейства 21xx. Также внимание уделено отладочным средствам и технической поддержке разработчиков прикладных систем на основе процессоров данного семейства.

Для студентов радиотехнических специальностей, аспирантов и преподавателей. Пособие может быть полезно инженерам системотехникам радиоэлектронной и вычислительной аппаратуры.

Предисловие

Развитие прикладных математических методов, которые позволяют извлекать различного рода информацию из окружающего нас мира сигналов, потребовало создания специального устройства, которое обладало бы гибкой архитектурой и широким спектром команд ориентированных на векторные и циклические операции - таким устройством и стал цифровой сигнальный процессор (DSP). Отличительной особенностью задач цифровой обработки сигналов, как было проиллюстрировано в первой части пособия, является поточный характер обработки больших массивов данных в реальном режиме времени, требующий от технических средств высокой производительности и обеспечения возможности интенсивного обмена с внешними устройствами. Соответствие данным требованиям достигается в настоящее время благодаря специфической архитектуре сигнальных процессоров и проблемно-ориентированной системе команд. Сигнальные процессоры обладают высокой степенью специализации. В них широко используются методы сокращения длительности командного цикла, характерные и для универсальных RISC-процессоров, такие как конвейеризация на уровне отдельных микроинструкций и инструкций, размещение операндов большинства команд в регистрах, использование теневых регистров для сохранения состояния вычислений при переключении контекста. Сравнительно невысокая цена, а также развитые средства разработки программного обеспечения позволяют легко внедрять подобные системы в различные системы управления.

Наиболее распространенной математической операцией, требующейся для задач обработки сигналов, является комбинация сложения и умножения. Суммирующие функции реализуются довольно просто и могут быть выполнены за один такт работы процессора. Функции умножения требуют большего времени выпсушения. Особенно для чисел с плавающей точкой. Такие вычисления для многих процессоров общего назначения могут потребовать несколько сотен тактов.

К цифровым сигнальным процессорам (Digital Signal Processor - DSP) относятся процессоры, для которых характерно использование групп методов обработки сигналов цифровыми способами. Преимущества цифровых методов обработки по сравнению с аналоговыми заключаются: в упрощении работы с памятью, в расширении набора используемых арифметических операций и в повышении допустимой сложности алгоритмов. Основное их преимущество состоит в возможностях значительного увеличения точности вычислений. Главным недостатком является то, что для ряда специальных приложений они оказываются более медленными, чем аналоговые. По типу обрабатываемых данных DSP можно условно разделить на DSP с фиксированной точкой и

DSP с плавающей точкой. Эти два класса существенно различаются по цене. Данное пособие посвящено рассмотрению вопросов применения цифровых сигнальных процессоров с фиксированной точкой.

Материалы пособия являются кратким изложением курсов «Основы цифровой обработки сигналов» и «Цифровые сигнальные процессоры и микроконтроллеры», читаемые авторами в МГТУ им.Н.Э.Баумана студентам кафедры ИУ4 «Проектирование и технология производства ЭА». Пособие разработано в рамках университетской программы фирмы Analog Devices и (www.analog.com) фирмы Autex (www.autex.ru).

Условные обозначения, используемые в учебном пособии:

Условное обозначение примера реализации синтаксических конструкций программного кода, алгоритмов и методик разработки, настройки и тестирования программно-алгоритмического обеспечения (ПАО), средств разработки и отладки.

Задание для самостоятельного выполнения, предназначенное для закрепления теоретического и практического материала и предназначенные для выработки практических навыков.

? Задания для самопроверки, контрольные вопросы.

СОДЕРЖАНИЕ

Введение

Реализация даже несложной обработки сигналов на универсальном процессоре (с расширенным набором команд - CISC) требует значительного быстродействия и далеко не всегда возможна в реальном времени, в то время как даже простые специализированные сигнальные процессоры (ЦСП) нередко справляются в реальном времени с относительно сложной обработкой, а мощные ЦСП способны выполнять качественную спектральную обработку сразу нескольких сигналов. В силу своей специализации ЦСП редко применяются самостоятельно - чаще всего устройство обработки имеет универсальный процессор средней мощности для управления всем устройством, процессами приема/передачи информации, взаимодействия с пользователем, и один или несколько ЦСП - собственно для обработки сигналов различных источников.

По типу обрабатываемых данных сигнальные процессоры можно условно разделить на ЦСП с фиксированной точкой и ЦСП с плавающей точкой. Эти два класса существенно различаются по цене. Использование в сигнальной обработке данных в формате с плавающей точкой обусловлено несколькими причинами. Для многих задач, связанных с выполнением интегральных и дифференциальных преобразований, особую значимость имеет точность вычислений, обеспечить которую позволяет экспоненциальный формат представления данных. Алгоритмы компрессии, декомпрессии, адаптивной фильтрации в цифровой обработке сигналов связаны с определением логарифмических зависимостей и весьма чувствительны к точности представления данных в широком динамическом диапазоне. Работа с данными в формате с плавающей точкой существенно ускоряет и упрощает обработку, повышает надежность программы, поскольку не требует выполнения операций округления и нормализации данных, отслеживания ситуации потери значимости и переполнения. Платой за эти дополнительные комфорт и скорость является высокая сложность функциональных устройств, выполняющих обработку данных в формате с плавающей точкой, необходимость использования более сложных технологий производства микросхем, больший процент отбраковки изделий и как следствие дороговизна микропроцессоров.

Системы на базе процессоров с фиксированной точкой предназначены главным образом для массового потребителя. Низкая стоимость подобных систем позволяет интегрировать их в различную портативную аудиотехнику, которая в своем большинстве уже использует технологию цифровой обработки (MP3 и MiniDisc плееры) и, как правило, содержит в себе сигнальный процессор.

Процессоры для обработки чисел с фиксированной точкой компании Texas Instruments представлены тремя семействами процессоров, базовыми моделями которых являются соответственно TMS320C 10,20,50. Достаточной для реализации алгоритмов адаптивной фильтрации производительностью обладают лишь представители семейств C2xx, C5x и C54x (40, 40 и 66 MIPS соответственно). Значительное внимание при разработке процессоров данных семейств уделялось обеспечению энергосберегающих режимов функционирования. В микропроцессорах поддерживаются активный, периферийный режимы и режим "сна". Потребление тока в активном режиме не превышает 1,5 мА/MIPS при питании 3 В. В периферийном режиме центральный процессор останавливается и работает только периферия, потребление тока составляет 0,25 мА при 3В. В режиме "сна" процессор останавливается до получения сигнала прерывания. Потребление тока в этом режиме составляет 5 мкА.

Семейство с плавающей точкой включает процессоры TMS320C30,40, а семейство TMS320C6x содержит процессоры как с фиксированной, так и с плавающей точкой.

Семейство TMS320C3x отличается высокой производительностью (33 MFLOPS), гибкой системой команд, хорошей аппаратной поддержкой операций с плавающей точкой, мощной системой адресации, поддержкой языка высокого уровня - Си.

Благодаря своей уникальной архитектуре микропроцессоры TMS320C40 совместимы по системе команд с C30, однако, обладают большей производительностью и лучшими коммуникационными возможностями. C40 имеет производительность 60 MFLOPS и максимальную пропускную способность подсистемы ввода-вывода 384 Мбайт/с.

Микропроцессоры компании Analog Devices образуют два основных семейства - ADSP-218x и ADSP-2106x. Семейство ADSP-218x объединяет 16-рязрядные процессоры с общей базовой архитектурой, оптимизированной для высокоскоростных вычислений с фиксированной точкой. Друг от друга представители семейства отличаются, в основном, расположенными на кристалле периферийными устройствами, такими как кэш-память, таймеры, порты и т.д. Семейство ADSP- 218x успешно конкурирует с аналогичной продукцией других компаний благодаря сравнимой производительности при более низкой цене, а также развитой системе технических и программных средств разработки прикладных систем. На сегодняшний день последним представителем этого семейства является процессор ADSP-2189. Он имеет высокую производительность при тактовой

частоте 75 МГц и сниженное до 2,5 В напряжение питания. Такое сочетание позволяет строить на его основе высокопроизводительные системы управления с автономным питанием. Еще одной особенностью является расширенный до 32 Кбайт объем внутрикристальной памяти программ, что позволяет использовать более сложные алгоритмы обработки.

Второе семейство - ADSP-2106x - объединяет 32-разрядные микропроцессоры, ориентированные на алгоритмы с плавающей точкой. Микропроцессоры семейства ADSP-2106x имеют второе название - SHARC, связанное с особенностями их архитектуры, которая объединяет высокоэффективное процессорное ядро, интерфейс с host-процессором, контроллер ПДП, последовательные порты и линки. Процессоры ADSP-21061, 062 работают на частоте 40 МГц и имеют быстродействие до 120 MFLOPS. Такая производительность наряду с оптимизацией системы команд под алгоритмы цифровой фильтрации делает эти процессоры крайне популярными при разработке систем управления реального времени.

Сигнальные процессоры компании Motorola ориентированы главным образом на рынок систем связи. К числу последних достижений фирмы относится 24-разрядный процессор серии DSP56300 - DSP56311 с производительностью 255 MIPS. Помимо DSP-ядра на тактовую частоту 150 МГц, они содержат сопроцессор, выполняющий функции фильтрации и подавления эхо-сигнала с производительностью 105 MIPS, и с ОЗУ емкостью 3 Мбит. Новые схемы, способные поддерживать одновременно несколько голосовых каналов и канатов данных, пригодны и для проводных систем связи и Интернет-телефонии.

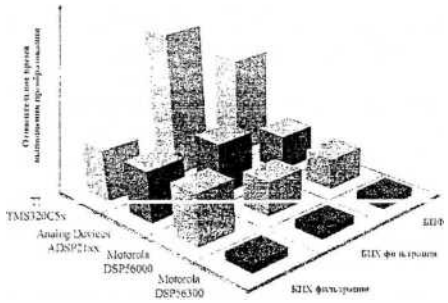

Рис.В.1. Сравнение эффективности архитектур процессоров с равной тактовой частотой

Выбор того или иного процессора - многокритериальная задача, однако, следует отметить предпочтительность процессоров Analog Devices для приложений, требующих выполнения больших объемов математических вычислений (таких как цифровая

фильтрация сигнала, вычисление корреляционных функций и т.п.), поскольку их производительность на подобных задачах выше, чем у процессоров компаний Motorola и Texas Instruments. В то же время для задач, требующих выполнения интенсивного обмена с внешними устройствами (многопроцессорные системы, различного рода контроллеры), предпочтительнее использовать процессоры Texas Instruments, обладающие высокоскоростными интерфейсными подсистемами. Компания Motorola является лидером по объему производства универсальных сигнальных микропроцессоров, большую часть из которых составляют дешевые и достаточно производительные 16- и 24- разрядные микропроцессоры с фиксированной точкой.

На рис.В.1. показано сравнение эффективности архитектур процессоров различных производителей с равной тактовой частотой (производительность процессора DSP56300 компании Motorola выше в два раза за счет наличия двухпроцессорного ядра).

ГЛАВА 1. Основные понятия, определения и термины

1.1 Общие принципы построения цифровых сигнальных процессоров, архитектура цифровых сигнальных процессоров

Цифровым сигнальным процессором называют процессор для выполнения обработки данных и решения задач в реальном масштабе времени с использованием «оцифрованных сигналов». ЦСП относятся к специализированным процессорам и именно поэтому имеют своеобразную архитектуру. Не существует и невозможно создать универсальный процессор способный эффективно решать любые задачи. При создании нового семейства процессоров в первую очередь определяют круг основных задач решаемых им. Поэтому архитектура ЦСП формировалась с учетом специфики алгоритмов ЦОС.

Принято выделять в качестве основных две архитектуры или два подхода к построению процессоров:

- Фон-Неймановский
- Гарвардский

Фон-Неймановский подход предполагает для хранения данных и программы использовать одно устройство памяти. С ЦПУ устройство памяти соединено двумя шинами: шиной адреса и шиной данных.

Гарвардский подход предполагает наличие двух различных устройств памяти для хранения данных и программы. Каждое устройство для общения с ЦПУ имеет свои шину данных и шину адресов.

В системе с гарвардской архитектурой можно одновременно обращаться к различным устройствам памяти, что позволяет ускорить выполнение бинарных операций, операнды которых хранятся в памяти.

Именно поэтому ЦСП преимущественно строят по модифицированной гарвардской архитектуре, при которой операция вида $Z = Z + X*Y$ выполняется за один такт. Однако этим специфика не ограничивается и будет постепенно рассмотрена в следующих параграфах.

Для хранения информации используется память программ и память данных, располагающиеся на кристалле. Устройства на кристалле процессора связаны с памятью шинами. Чаще всего выделяют четыре шины: шину адреса памяти программ (ШАПП), шину данных памяти программ (ШДПП), шину адреса памяти данных (ШАПД), шину

данных памяти данных (ШДПД). Количество шин в различных процессорах может различаться. Наиболее часто увеличивают количество ШДПД. Это дает возможность разным устройствам выполнять операции одновременно. В некоторых процессорах (например, TMS320C5000) производится разделение ШДПД для чтения и ШДПД для записи.

В настоящее время объем внутрикристальной памяти программ и памяти данных ЦСП составляет несколько десятков килобайт, поэтому большинство ЦСП позволяет работать с внешней памятью, связь с которой осуществляется через внешние шину адреса (ВША) и шину данных (ВШД). Таким образом, для внешней памяти, память программ и память данных сосредоточены в одном адресном пространстве, поэтому одновременные обращения могут стать причиной конфликта.

АЛУ (арифметико-логическое устройство) выполняет требуемые арифметические операции.

Умножитель введен для оптимизации выполнения операций умножения с накоплением, и его основная функция выполнять операции вида R = R + X*Y.

Сдвигатель - сдвиги на любое количество разрядов в любом направлении можно производить и в АЛУ, однако для распараллеливания в ряде процессоров фирм Motorolla, Texas Instruments, Analog Devices имеются аппаратно реализованные устройства сдвига.

За управление работой ЦСП и в частности программой отвечает **устройство управления** выполнением программы. В соответствии с командами, читаемыми из памяти программ устройство вырабатывает сигналы управления работой всех устройств процессора. Имеющиеся у устройства регистры содержат информацию о конфигурации процессора, например используемой памяти, о событиях в ходе работы, например о поступившем сигнале прерывания, и инициализируются перед началом работы.

Устройство генерации адреса формирует адреса данных извлекаемых из памяти программ или памяти данных. Данное устройство имеет собственные арифметические модули, что позволяет разгрузить АЛУ при использовании сложных методов адресации, например, таких как циклический буфер или бит-реверсная адресация. Состав внутрикристальной периферии зависит от назначения ЦСП, практически все модели имеют последовательные порты и таймер.

Все ЦСП имеют **регистровые файлы** - набор регистров для хранения операндов и результатов операций. Существует шесть типов операций:

1. регистр, регистр => регистр
2. память, память => регистр
3. память, регистр => регистр
4. память, память => память
5. регистр, регистр => память
6. память, регистр => память

Тип команды регистр, регистр => регистр означает, что операнды находятся в регистрах и результат помещается в регистр. Компиляторы языков высокого уровня наиболее эффективно используют именно этот тип операции. Именно поэтому разработчики стараются увеличить размер регистрового файла, поскольку это позволяет создавать более эффективные компиляторы языков высокого уровня.

Процесс выполнения команды во всех процессорах разбивается на несколько этапов. Конвейерный принцип выполнения команд состоит в том, что различные этапы различных команд выполняются одновременно.

Количество этапов, на которые разбивается выполнение команд в разных процессорах различно:

- В процессорах TI C2x существует три этапа конвейера: выборка, декодирование, выполнение команды.
- В процессорах TI C20x, C5x, C24x количество этапов - четыре: выборка, декодирование, подготовка операнда, выполнение команды.
- В процессорах ADSP 21xx, 2106x - три этапа: выборка, декодирование, выполнение команды.

Время выполнение одного этапа называется командным циклом или тактом. Время командного цикла равно периоду внутреннего генератора синхронизирующих импульсов, и может отличаться от периода внешнего генератора.

Несмотря на то, что каждая команда выполняется за три цикла, результаты выполнения появляются через интервал в один цикл. Период появления результатов в этом случае является временем выполнения команды.

Конвейерное выполнение команд возможно при наличии нескольких функциональных узлов для выполнения одновременно различных этапов с различными данными, относящимися к различным командам.

В частности использование гарвардской архитектуры позволяет одновременно выбирать команду из памяти программ и данные из памяти данных.

На стационарность конвейерного выполнения влияет несколько факторов:

1. **Длина команды.** Для нормальной работы конвейера все команды должны иметь одинаковую длину
2. **Количество условных переходов.** Если выполняется команда условного перехода, то адрес следующей команды становиться известным только после этапа выполнения команды. Поэтому если должна выполняться команда, располагающаяся не следом за командой условного перехода, то возникает, так называемый, конфликт конвейера.

Иногда при выполнении команд конфликты конвейера возникают на аппаратном уровне:

1. **Необходимо на одном этапу осуществить из внешней памяти выборку команд и данных.** Большинство ЦСП имеют одну общую шину для доступа к внешней памяти. Для предотвращения подобных конфликтов необходимо размещать программу и данные по различным блокам память, допускающим совместный доступ.
2. **Возникновение необходимости использовать на этапе выполнения одни и те функциональные устройства.**

При возникновении конфликта на этапе эксплуатации процессор в большинстве случаев автоматически добавляет пустые такты.

1.2 Процессоры с фиксированной и плавающей точкой

Процессоры с фиксированной точкой и процессоры с плавающей точкой отличаются формой представления данных. При этом следует заметить, что все процессоры с плавающей точкой имеют наборы команд как для обработки данных с фиксированной точкой, так и для обработки данных с плавающей точкой. Иными словами, процессоры с плавающей точкой являются расширением процессоров с фиксированной точкой. С другой стороны в процессорах с фиксированной точкой всегда можно организовать обработку данных с плавающей точкой программным путем, однако, соответствующие подпрограммы требуют достаточно много времени для выполнения.

В силу того, что алгоритмы выполнения операций с плавающей точкой более сложны, чем аналогичные алгоритмы с фиксированной точкой, цена процессоров с фиксированной точкой выше.

Поэтому необходим тщательный анализ задачи для правильного выбора семейства ЦСП.

1.3 Основные типы ЦСП

С точки зрения архитектуры, все существующие в настоящее время процессоры можно разделить на следующие основные типы;

- Стандартные ЦСП
- Улучшенные ЦСП
- Процессоры VLIW
- Суперскалярные процессоры
- Гибриды ЦСП/микроконтроллер

1.3.1 Стандартные процессоры ЦОС

Архитектура стандартных ЦСП описана в п. 1.2. Рассмотрим алгоритм КИХ-фильтра, выполняемый на данном процессоре.

Выходной отсчет вычисляется как: $y(n) = \sum_{i=0}^{N-2} h(i) \cdot x(n-i)$

Коэффициенты фильтра хранятся в памяти программ, входные данные хранятся в памяти данных. За один такт происходит считывание из памяти входного отсчета, коэффициента и их перемножение. В процессорах ADSP-21xx эта операция выглядит следующим образом:

MR = MR + MXO * MYO, MXO = DM(I0,M0), MY0 = PM(I4,M4)

Таким образом, выходной отсчет вычисляется за N тактов.

1.3.2Улучшенные ЦСП

Существует три основных пути увеличения производительности ЦСП:

1. Увеличение тактовой частоты процессора
2. Увеличение количества одновременно производимых операций
3. Увеличение количества одновременно выполняемых команд

Процессоры, в которых увеличение производительности достигается увеличением количества одновременно производимых операций, относятся к улучшенным стандартным процессорам. Увеличение количества одновременно производимых операций достигается:

- Увеличением количества дополнительных функциональных модулей (например, сумматоров или умножителей с накоплением)
- Введением специализированных функциональных модулей (например, модулей для построения цифровых фильтров)
- Расширением шин передачи данных для увеличения количества передаваемой одновременно информации
- Использованием памяти с многократным доступом (памяти с возможностью выполнения нескольких обращений за такт)

Следствием подобных мер увеличения производительности является усложнение системы команд и усложнение разработки программного обеспечения. С другой стороны подобная архитектура является не дружественной для компиляторов языков высокого уровня. Как отмечалось выше, компилятор эффективно использует простые команды, характерные для архитектур типа RISC.

Тем не менее, улучшенные стандартные процессоры получили достаточно широкое распространение, среди них можно назвать процессоры DSP56301 (Motorola), TMS320C55x (Texas Instruments), ADSP-2116x (Analog Device).

1.3.3 Процессоры с архитектурой VLIYV

Одним из способов увеличения производительности является увеличение количества команд, выполняемых одновременно. Этот способ реализован в процессорах с архитектурой VLIW (Very Long Instruction Word). Подобные процессоры используют простую систему команд, каждая из которых представляет собой элементарную операцию. Несколько простых команд выполняется одновременно в различных независимых блоках процессора. Команды, которые должны быть выполнены одновременно собираются в пакет и представляют собой, так называемую, суперкоманду. Архитектура предполагает использование регистровых файлов большого размера для хранения операндов и результатов работы всех операционных модулей. Длинные команды (суперкоманды) предполагают так же наличие многоразрядных шин передачи команд и данных.

Недостатком данной архитектуры следует считать большие объемы памяти, требуемые для записи программы с суперкомандами и нерациональное использование этой памяти.

К процессорам, построенным по архитектуре VLIW, следует отнести процессоры фирмы Texas Instruments TMS320C6xxx и процессоры фирмы Motorola MSC8lOx

1.3.4 Суперскалярные процессоры

В процессорах с VLIW архитектурой работу по группировке команд в пакет (суперкоманду) выполняет программист, в суперскалярных процессорах эту работу выполняет сам процессор. Команды группируются в пакет на этапе выполнения, поэтому, с точки зрения программиста все команды являются самостоятельными.

В процессорах с суперскалярной архитектурой вводятся два дополнительных модуля:

- Модуль команд
- Модуль данных

Модуль команд состоит из узла выборки команд, кэша команд и узла отправки на выполнение. Узел выборки команд в течение одного такта выбирает из памяти программ в кэш некоторое количество команд, при этом используются методы предсказания переходов и другие методы, уменьшающие конфликты и задержки конвейера выполнения команд. Если команда уже загружена в кэш, то ее выборка заново не производится, подобный механизм оптимизирует выполнение циклов.

Модуль данных содержит узлы предвыборки данных, кэш данных и систему управления загрузки памяти.

Самым интересным модулем является модуль управления работой конвейера, модуль выбирает из последовательности команд несколько команд, которые могут выполняться параллельно исходя из необходимого чередования данных и загрузки ресурсов вычислительного модуля.

Выполнение команд производится следующим образом:

- Модуль управления работой конвейера уведомляет модуль команд о том, какие команды необходимы для следующей группы
- Модуль данных считывает требуемые данные и посылает их в вычислительный модуль
- Вычислительный модуль выполняет команды и результат помещает в свой регистровый файл
- Необходимые данные переписываются из регистрового файла

вычислительного модуля в память

В конвейере могут возникать следующие конфликтные ситуации

- Предвыборка команд требует дополнительных тактов для заполнения кэша команд
- Предвыборка данных требует дополнительных тактов

Модуль управления конвейером также обрабатывает запросы на прерывания

Поскольку суперскалярные процессоры группируют команды, основываясь на зависимостях данных, один и тот же набор команд может выполняться за разное время на различных этапах выполнения программы. Поэтому время выполнения различных частей программы определить достаточно сложно, что несколько ограничивает применение данного класса процессоров в системах реального времени, хотя и не исключает.

К производимым в настоящее время суперскалярным процессорам следует отнести семейство LSI40xZ фирмы LSI Logic Corporation.

1.3.5 Гибридные процессоры

Во многих приборах и устройствах требуется решать задачи управления и цифровой обработки сигналов, например блоки управления двигателем, сотовые телефоны и т.д. Для удовлетворения данной потребности были созданы гибридные процессоры. Они представляют собой тандем из микроконтроллера и ЦСП.

Гибридный процессор, по сути, представляет собой два процессора: микроконтроллер и ЦСП соединенные интерфейсом обмена, состоящим из общей памяти и устройства управления обменом, которое программируется как ЦСП, так и микроконтроллером. Каждое ядро способно вызывать любое прерывание другого ядра. Таким образом, обеспечивается совместная работа микроконтроллера и ЦСП.

Микроконтроллер строят чаще всего по RISC архитектуре и снабжают, ставшим уже стандартным, набором периферийных устройств:

- Сторожевой таймер
- Программируемый таймер прерываний
- Генератор ШИМ сигналов
- Универсальный асинхронный приемопередатчик UART

Среди гибридных процессоров стоит выделить семейство DSP5665X фирмы Motorola, семейство ADMC3xx фирмы Analog Device, семейство TMS320C24xx фирмы Texas Instruments.

1.4 Обзор семейства ЦСП ADSP-21xx

Семейство ADSP-21xx (фирмы Analog Devices) представляет собой ряд однокристальных микропроцессоров с архитектурой, оптимизированной для цифровой обработки сигналов. Отличия разных процессоров друг от друга заключается в различных дополнениях к базовой архитектуре, кроме того, процессоры ADSP-21msp58/59 имеют на кристалле аналоговый интерфейс для обработки смешанных аналоговых/цифровых сигналов.

Базовая архитектура семейства ADSP-21xx включает в себя следующие функциональные устройства:

- *Вычислительный устройства* - Каждый процессор содержит три независимых вычислительных устройства. АЛУ - для выполнения арифметических операций, умножитель-накопитель, для выполнения операции умножения с накоплением, сдвигатель, для выполнения различных операций сдвига. Все устройства работают с числами с фиксированной точкой разрядностью 16 бит и поддерживают вычисления с повышенной точностью.
- *Генераторы адресов и программный автомат* - Два генератора адреса данных и программный автомат генерируют адреса для доступа к данным на кристалле и во внешней памяти. Программный автомат поддерживает команды условного перехода за один такт и выполнение циклов с нулевыми затратами ресурсов. Два генератора адреса данных позволяют одновременно генерировать адреса для выбора двух операндов.
- *Память* - Процессоры семейства ADSP-21xx построены по модифицированной гарвардской архитектуре. Данные хранятся в памяти данных, команды и данные хранятся в памяти программ. Все процессоры семейства ADSP-21xx содержат на кристалле память, которая занимает часть адресного пространства памяти программ и памяти данных, таким образом возможно подключение устройств вешней памяти. Быстродействие памяти на кристалле позволяет за один такт выбирать из памяти данных операнд, а из памяти программ операнд или команду.
- *Последовательные порты* - Последовательные порты обеспечивают полное сопряжения с аппаратными средствами компадирования. Поддерживаются компадирование с A и u - характеристиками. Порты сопрягаются непосредственно со

многими типами последовательных устройств. Каждый порт может генерировать программированные тактовые синхроимпульсы или принимать внешние тактовые синхроимпульсы.

❖ *Таймер* - Программируемый таймер/счетчик с предварительным делителем частоты разрядностью 8 бит обеспечивает генерацию периодических прерываний.

❖ *Порт интерфейса хост-машины* - Порт интерфейса хост-машины обеспечивает непосредственное соединение (без буферных схем) с хост-процессором. Порт интерфейса с хост-машиной имеет 16 выводов для данных и 11 управляющих выводов. Такие процессоры, как например Motorolla 68000, Intel 8051 с помощью интерфейса хост-машины могут быть легко сопряжены с процессорами семейства ADSP-21xx. Порты прямого доступа к памяти - Имеющийся в процессоре ADSP-21S1 порт прямого доступа к памяти (IDMA) и порт прямого побайтного доступа к памяти (BDMA). Внутренний порт доступа к данным имеет 16 разрядную мультиплексированную шину адреса и данных и поддерживает работу с 24 разрядной памятью данных. Запись в этот порт выполняется асинхронно и не влияет на быстродействие программы. Порт прямого доступа с байтной организацией позволяет обеспечить начальную загрузку памяти данных и памяти программ.

❖ *Аналоговый интерфейс* - Некоторые процессоры имеют на кристалле средства поддержки обработки аналоговых сигналов. Эти средства включают: аналого-цифровой преобразователь (АЦП), цифро-аналоговый преобразователь (ЦАП), аналоговые и цифровые фильтры и средства параллельного сопряжения с базовой архитектурой процессора. Преобразователи используют сигма-дельта модуляцию для получения выборки с избыточной дискретизацией.

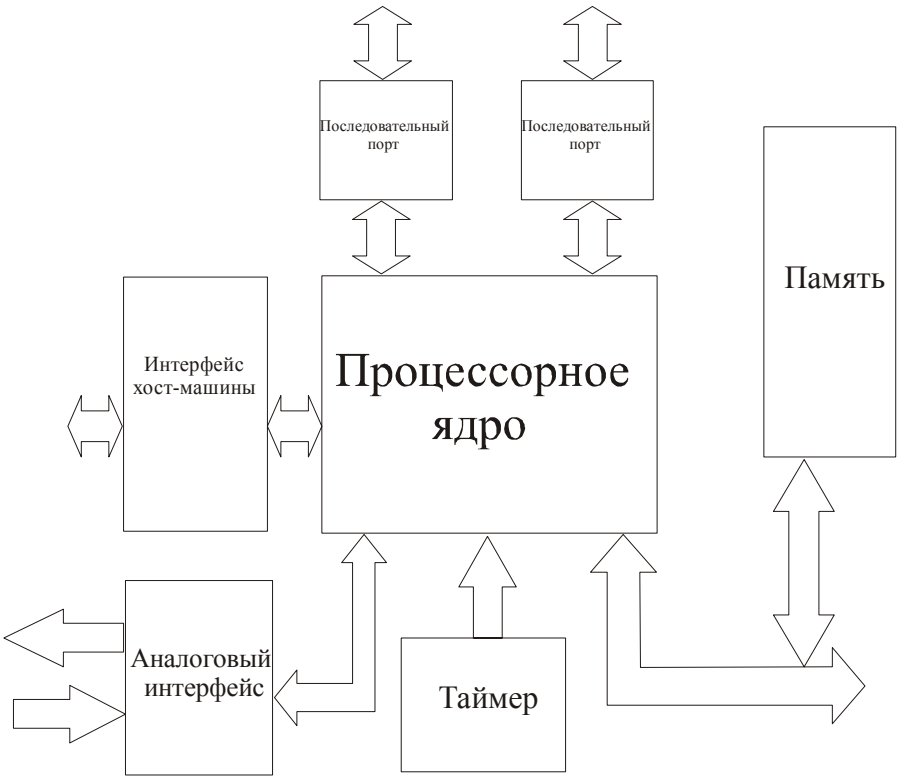

Рис. 1. Базовая архитектура семейства ADSP-21xx.

Архитектура процессоров семейства ADSP-21xx имеет высокую степень параллелизма, отвечающую требованиям цифровой обработки. За один такт любой процессор семейства может:

- Сгенерировать новый программный адрес.
- Выбрать следующую команду.
- Выполнить одну или две операции пересылки данных
- Обновить один или два указателя адреса данных.
- Выполнить вычисление.

В течение одного и того же цикла процессоры, имеющие соответствующие функциональные устройства, могут:

- Принимать и/или передавать данные через последовательный(ые) порт(ы).
- Принимать и/или передавать данные через порт интерфейса хост-машины.
- Принимать и/или передавать данные через аналоговый интерфейс.
- Принимать и/или передавать данные через DMA порты.

Согласно «Руководству пользователя процессорами семейства ADSP-2100» имеются модели не рекомендуемые для применения в новых разработках см. таблицу 1.1, и модели рекомендуемые для применения в новых разработках см. таблицу 1.2.

Таблица 1.1 Модели, не рекомендованные для применения в новых разработках

Функциональные устройства	**2101**	**2103**	**2105**	**2115**	**2111**	**2171**	**2173**	**21msp58**
Арифметико-логическое устройство	+	+	+	+	+	+	+	+
Умножитель-накопитель	+	+	+	+	+	+	+	+
Устройство сдвига	+	+	+	+	+	+	+	+
2 генератора адреса данных	+	+	+	+	+	+	+	+
Программный автомат	+	+	+	+	+	+	+	+
Размер ОЗУ памяти данных(слов)	1K	1K	512	512	1K	2K	2K	2K
Размер ОЗУ памяти программ(слов)	2K	2K	1K	1K	2K	2K	2K	2K
Таймер	+	+	+	+	+	+	+	+
Последовательный порт 0	+	+	-	+	+	+	+	+
Последовательный порт 1	+	+	+	+	+	+	+	+
Порт интерфейса хост-машины	-	-	-	-	+	+	+	+
Порт прямого доступа к памяти	-	-	-	-	-	-	-	-
Аналоговый интерфейс	-	-	-	-	-	-	-	+
Напряжение питания	5В	3.3В	5В	5В	5В	5В	3.3В	5В
Быстродействие (миллионов команд в секунду)	20	10	20	20	20	33	20	26

Таблица 1.2 Модели, рекомендованные для применения в новых разработках

Функциональные устройства	**2104**	**2104L**	**2181**	**2183**	**2185**	**2185L**	**2186**	**2186L**	**2187**	**2187L**
Арифметико-логическое устройство	+	+	+	+	+	+	+	+	+	+
Умножитель-накопитель	+	+	+	+	+	+	+	+	+	+
Устройство сдвига	+	+	+	+	+	+	+	+	+	+
2 генератора адреса данных	+	+	+	+	+	+	+	+	+	+
Программный автомат	+	+	+	+	+	+	+	+	+	+
Размер ОЗУ памяти данных(слов)	256	256	16K	16K	16K	16K	8K	8K	32K	32K
Размер ОЗУ памяти программ(слов)	512	512	16K	16K	16K	16K	8K	8K	32K	32K
Таймер	+	+	+	+	+	+	+	+	+	+
Последовательный порт 0	+	+	-	+	+	+	+	+	+	+
Последовательный порт 1	+	+	+	+	+	+	+	+	+	+
Порт интерфейса хост-машины	-	-	-	-	-	-	-	-	-	-
Порт прямого доступа к памяти	-	-	+	+	+	+	+	+	+	+
Аналоговый интерфейс	-	-	-	-	-	-	-	-	-	-
Напряжение питания	5В	3.3В	5В	3.3В	5В	3.3В	5В	3.3В	5В	3.3В
Быстродействие (миллионов команд в секунду)	20	12.5	40	52	40	52	40	52	40	52

1.5 Типовые области применения цифровых сигнальных процессоров с фиксированной точкой

Основные области применения процессоров с фиксированной точкой:

- Эхо-компенсаторы
- Музыкальные инструменты
- Генераторы функций
- Синтезаторы
- Цифровые фильтры
- Спектроанализаторы
- Ультрозвуковая диагностика
- Обработка речевых сигналов и двумерных изображений •> Локальные сети
- Системы управления
- Навигационное оборудование
- Гидро- и радиолокационные системы

Вопросы для самопроверки:

1. В чем отличия между гарвардской и фон-неймановской архитектурой?
2. В чем особенности архитектуры процессоров цифровой обработки сигналов?
3. Назовите виды процессоров цифровой обработки сигналов?
4. Приведите сравнительные характеристики ЦСП с плавающей и фиксированной точкой.
5. Дайте определение основным функциональным устройствам, входящим в состав ЦСП.
6. Дайте характеристику областям применения ЦСП в зависимости от их основных характеристик.

Задание:

Используя интернет ресурсы http://dsp.iu4.bmstu.ru, http://www.scanti.ru, http:// www.autex.ru, http://www.argussoft.ru, http://www.motorola.com, http://www.analog.com. http://www.ti.com и др.) проведите поисковые исследования по одной из следующих тем:

- Современные тенденции развития архитектурных решений ЦСП.
- Области применения ЦСП.

- Обобщение результатов сравнительных тестов производительности ЦСП различных фирм,

Составить краткий обзор интернет - ресурсов фирм производителей электронных систем на базе ЦСП в зоне *.ru, при составлении обзора отработать перечень ключевых слов для формирования поисковых запросов.

ГЛАВА 2. Проектирование

2.1 Этапы жизненного цикла разработки программы

Системы на базе DSP - это аппаратно-программные комплексы, жизненный цикл разработки включает следующие этапы:

- Анализ и постановка задачи. На этом этапе определяются требования к программе, формулируются требования к входным и выходным данным, их структуре, форматам и методам ввода/вывода в проектируемой системе. Эти артефакты являются предварительными и могут уточняться в ходе дальнейшей разработки.
- Выбор цифрового сигнального процессора. На основе артефактов, выработанных на предыдущем шаге разработки, производится выбор семейства цифровых сигнальных процессорах.
- Определение структуры данных. Исходя из требований программы, производится разработка структуры данных.
- Выбор языка программирования. Исходя из артефактов, созданных на предыдущих шагах, производится выбор языка программирования. На этот выбор может влиять множество факторов: сложность задачи, квалификация разработчиков, сроки исполнения и т.д.
- Разработка алгоритма и структуры программы. На этом этапе производится разработка архитектуры программы, создается спецификация на программные модули и их интерфейсы. Определяются используемые стандартные компоненты, рекомендуется максимально использовать стандартные компоненты. Рекомендуется использовать методы структурного программирования. Имеется два подхода к разработке программы: снизу вверх и сверху вниз. Для получения более подробной информации рекомендуется обратиться к соответствующей литературе. В настоящее время ряд фирм выпустили средства разработки, поддерживающие разработку объектно-ориентированных систем. При разработке объектно-ориентированных систем рекомендуется воспользоваться унифицированным рациональным процессом разработки программ. Для получения более подробной информации рекомендуется обратиться к соответствующей литературе.
- Написание программы. На этом этапе необходимо распределить обязанности между участниками разработки. Интерфейсы модулей, выработанные на предыдущем шаге, рекомендуется зафиксировать. Желательно установить требования к оформлению исходного кода и комментариям.
- Получение исполняемой программы. Исполняемый код программы создается с помощью средств, входящих в пакет разработки.

❖ Тестирование и отладка программы. Отладка программы представляет собой процесс поиска ошибок в программе. Существует множество методов тестирования программы. Наиболее широко используемый метод тестирования - тестирование по вариантам использования. На вход подаются все возможные (разрешенные и запрещенные) входные сигналы и проверяется корректность выходных данных. Помимо тестирования программы необходимо выполнить ее профилирование, то есть измерение времени выполнения программы и каждое ее части.

❖ Получение загрузочной программы. Загрузочная программа создается с помощью средств, входящих в пакет разработки. После прошивки программы в память процессора рекомендуется повторить процесс тестирования.

2.2 Лингвистическое обеспечение

Для программирования систем, построенных на основе цифровых сигнальных процессоров, в основном применяются язык ассемблера и язык высокого уровня С.

Языки ассемблера являются машинно-ориентированными языками и, следовательно, для каждого типа процессоров существует свой язык. Почти каждая команда ассемблера эквивалентна машинной команде. Используемые символические обозначения обычно отражают и содержательный смысл. В ассемблере допускается оформление повторяющейся последовательности операндов в виде одной макрокоманды. Версии языка, допускающие создание подобных конструкций, носят название макроассемблера.

При программировании на языке ассемблера программисту доступны все ресурсы конкретного процессора. Эго позволяет программисту разрабатывать эффективные программы. Проблемы, связанные с периферийными устройствами лучше решать средствами языка ассемблера. Однако программирование на ассемблере предполагает знание архитектуры процессора.

Современные версии ассемблера имеют набор инструкций, облегчающих построение структурированных программ.

Следует отметить, что знание языка ассемблера остается необходимым условием получения «хороших» программ.

Практически для всех процессоров ЦПОС язык С является допустимым языком программирования. Достоинствами языка С являются:

❖ Переносимость. Разработку программы можно выполнить на платформе

имеющей более широкие средства для разработки и отладки программы, например на компьютерах, совместимых с PC/AT, после чего перенести ее на выбранную платформу.

❖ При использовании языка С программист, вообще говоря, не обязан знать архитектуру процессора.

❖ Язык С позволяет делать ассемблерные вставки, для оптимизации мест, критичных к скорости выполнения.

❖ Для программирования процессоров построенных по архитектуре VLIW требуется хорошее знание архитектуры процессора для написания пакетов инструкций, выполняющихся параллельно. Кодогенератор языка С генерирует оптимизированный код.

❖ Разработка программы на языке высокого уровня значительно сокращает время разработки программы.

Выбор тех или иных специализированных средств разработки зависит от типа выбранной элементной базы.

Так, например, если используется ДСП фирмы Analog Devices, то и в качестве инструментального средства, как правило, выбирают VisualDSP.

Основными инструментальными пакетами для ЦСП фирмы Texas Instruments являются:

- Texas Instrument C/ASM. TI C-компилятор, который позволяет компилировать программы с ассемблерными вставками. Эта программа полностью совместима с ANSI-стандартом и предназначена для использования с одним сигнальным процессором. Основная проблема с TI-компилятором состоит в том, что он не имеет удачно завершенного интерфейса между PC-машиной и ЦСП-системой. Например, он не имеет реализации stdio, и генерирует только объектный код, а не самозагружаемую последовательность.
- Tartan C/C++. Tartan Inc.- фирма, которая специализируется на ADA-компиляторах. Фирма работает с TI, обеспечивая С и С++ утилиты для ЦСП. Как и в случае с TI, компиляторы этой фирмы имеют некоторые ограничения в плане совместимости с host-платформами.
- 3D Parallel C. Это инструментальная среда фирмы 3L Ltd., которая производит системы для программирования с параллельной обработкой данных. Фирма приобрела TI-компилятор и добавила много новых возможностей. Система включает в себя ядро, которое выполняется на каждом сигнальном процессоре и обеспечивает полную планировку задач и управление параллельными процессами. Ядро использует

прямой доступ к памяти через COM-порты для коммуникаций между процессорами. Пакет обеспечивает полный инструментарий stdio для всех ЦСП системы.

Среди операционных систем, ориентированных на решение задач ЦОС, можно упомянуть Helios. Helios - это параллельная операционная система, которая выполняется на сигнальных процессорах. Она обеспечивает полное POSIX окружение для инструментальных и запускаемых пользователем приложений. Имеется встроенный C-компилятор и ассемблер, которые написаны специально для Helios. Главное отличие этого компилятора состоит в том, что он использует байтовую неявную адресацию с циклом чтение-модификация-запись. Это дает возможность оптимального сохранения байта данных в памяти, однако может незначительно снизить быстродействие систем, а также сузить диапазон адресации DSP. Другие ОС, такие как Virtuoso и SPOX, которые добавляют свои собственные возможности к TI-компилятору, позволяют реализовать широкий спектр вычислений и обработки данных в реальном времени. Использование библиотек функций обработки ускоряет и упрощает разработку приложений, обеспечивая легкость развития и поддержки программного обеспечения. Возможность для пользователей оптимизировать свои приложения весьма затруднительна без знаний архитектуры системы и системы команд сигнального процессора. Использование готовых библиотек функций является решением проблемы, т.к. эти библиотеки написаны экспертом и могут иметь возможности, как для расширения, так и сужения диапазона применения за счет малого количества используемых в них функций.

Операционная система реального времени SPOX представляет собой многозадачную операционную систему. Для взаимодействия с виртуальными аппаратными измерительными средствами и host-ЭВМ используется набор драйверов, который может изменяться и дополняться разработчиком. Принцип построения ОС - модульный: ядро - отладчик - линкер - библиотеки, с тремя уровнями привилегий:

1) обработка аппаратных прерываний,
2) обработка программных прерываний,
3) управляющие задачи с уровнем приоритета от 0 до 7.

SPOX является масштабируемой системой, при необходимости, изменив в файле конфигурации распределение задач по узлам разрабатываемой системы, можно легко перейти от виртуального однопроцессорного режима к многопроцессорному. Одно из основных преимуществ данной ОС - простота миграции между платформами. Например, для обеспечения обработки сигналов в реальном времени в системах на основе Win32 существует специальная версия SPOX - IA-SPOX, представляющая собой набор

виртуальных драйверов VxD кольца 0 ОС Win32. Прикладное программное обеспечение может быть создано с помощью любых средств разработки, а его взаимодействие с ядром будет осуществляться посредством специального модуля WinSpox (IA-SPOX).

Операционная система Virtuoso может поддерживать как однопроцессорные, так и многопроцессорные системы (в зависимости от версии). Она поддерживает два уровня обработки прерываний и имеет два ядра: наноядро и микроядро, которое обслуживает процессы в режиме многозадачности. Модульность ОС обеспечивается за счет возможности внедрения библиотек: ввода/вывода, графики, отладки, математических функций и т.п. Данная ОС имеет практически такие же свойства, что и SPOX, но поддерживает большее число популярных сигнальных процессоров и даже транспьютеры.

Отладка прикладных программ позволяет избежать ошибок и сбоев в работе системы. Tartan и Helios имеют собственные отладчики, предлагая интерфейс командной строки и отладчик высокого уровня. Tl-отладчик написан специально для DSP TI и использует JTAG-порт для доступа ко всем внутренним регистрам и периферийным схемам DSP. Он позволяет вести отладку на уровне исходного кода приложения и проследить, как программа взаимодействует с аппаратными ресурсами. Существуют версии отладчика для DOS и Windows, поддерживающие системы только с одним процессором на системной шине, однако UNIX, OS/2 и Windows NT версии позволяют отлаживать мультипроцессорные системы в многооконном интерфейсе отладчика. Отладчик дает возможность устанавливать точки прерывания не только по командам процессоров, но и по некоторым событиям в процессе вычислений. 3L предлагает Debugger Support Kit, который очень полезен, поскольку позволяет отлаживать исходный текст одного процессора без видимости других процессоров или ядра системы. Это является очень важной особенностью данного отладчика. Наиболее распространенным и простым интерактивным отладчиком является Code Composer фирмы Go DSP Inc., который позволяет также анализировать процесс обработки данных в системе и отображать результаты в виде соответствующих диаграмм. Давать какие-либо рекомендации по выбору программного обеспечения для DSP-систсм довольно трудно, здесь основным критерием являются особенности решаемой задачи. Начальным комплектом средств для разработки собственных DSP-приложений может быть следующий набор: ассемблер и отладчик, а также библиотеки математических функций и обработки сигналов. Данные средства включаются практически в любой набор инструментальных средств для ЦСП-систем начального уровня.

В настоящее время, все большее распространение для разработки программ ЦОС получает язык С++. Он является более строго типизированным, чем язык С, и позволяет разрабатывать более сложные, эффективные и надежные программы в более короткие сроки, чем язык С.

Вопросы для самоконтроля:

1. Перечислите этапы жизненного цикла разработки программы.
2. Что включает базовый набор инструментальных средств для решения задач ЦОС.
3. Дайте характеристику операционным системам, ориентированным на решение задач ЦОС.

ГЛАВА 3. Средства разработки программного обеспечения для семейства цифровых сигнальных процессоров ADSP-21xx

Знакомиться с принципами разработки систем на базе ЦСП начнем с анализа простейших операций. Операции сложения и вычитания совсем просто выполняются микропроцессорами общего назначения за один или незначительное число командных циклов. Цифровое сложение подобно десятичному сложению. Приведенный на рис.3.1, пример показывает сложение «1» и «2». Десятичный результат равен «3».

Умножение и деление являются более сложными операциями. Операция цифрового умножения может состоять из ряда операций «сдвига» и «сложения». В нашем примере показано умножение «2» на «4». При 4-разрядном двоичном представлении десятичного числа 2 второй бит дважды сдвигается влево, чтобы умножить 2 на 4 (т.к. $4 = 2^2$).

Микропроцессоры общего назначения оказываются весьма медленными при выполнении операций умножения и деления. Для выполнения одной операции умножения им обычно требуется осуществить целый ряд операций сдвига, сложения и вычитания.

Операция сдвига влево первый раз обращает число «0010» в число «0100», а второй раз - в число «1000». Блок будет содержать «1000».

Цифровая обработка сигналов по своей природе требует большого количества вычислений вида: A = B*C + D.

Этим объясняется необходимость в специальном устройстве, которое могло бы выполнять умножение и сложение за один командный цикл. Введение подобных устройств в процессоры, предназначенные для решения задач общего характера, не связанных с обработкой сигналов, нецелесообразно. Вместе с тем, для ЦПОС наличие такого специального устройства является обязательным требованием.

В системе команд большинства процессоров имеется специальная команда, которая обеспечивает умножение, суммирование и сохранение результата в одном цикле. Эта команда обычно называется «MAC» (сокращение от Multiply - умножать, ADD-складывать, Accumulate - накапливать). При работе с цифровыми процессорами вы очень часто будете встречаться с этой командой или ее разновидностями.

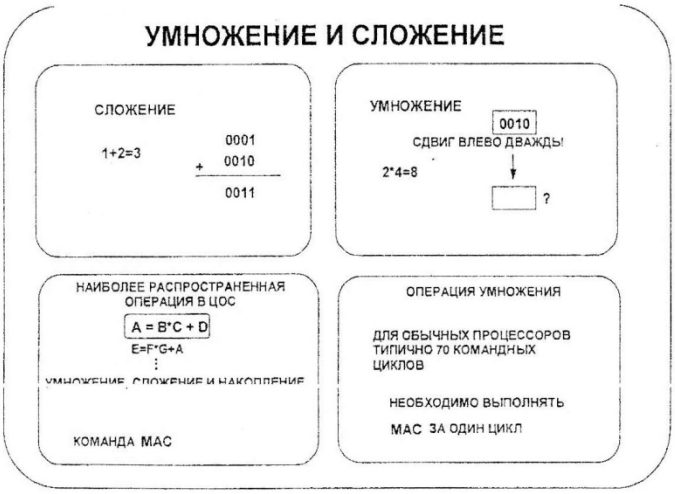

Рис.3.1. Основные операции над данными.

Рис.3.2. Обобщенная схема разработки ЦОС

В качестве инструментальных средств для разработки программного обеспечения для систем ЦОС используются: ассемблеры, языки высокого уровня, симуляторы и эмуляторы.

Ассемблеры выполняют трансляцию (т.е. перевод) текста программы, написанной в символической форме на языке ассемблера, в систему команд в машинном коде. Машинный код представляет собой совокупность двоичных цифр. Рассмотрим следующие две строки

‘ADD A, B’

‘1110001001010010001001’

Ассемблеры принимают команды в виде текста и преобразуют их в машинный язык. Это освобождает нас от необходимости запоминать двоичные коды команд процессора.

Языки высокого уровня подобны языкам ассемблеров, но значительно более дружественны. Ассемблеры имеют самые основные команды, такие как умножение, сложение и сравнение. Языки высокого уровня имеют и команды высокого уровня, такие как печать и неоднократное повторение. Поэтому писать программы легче на языках высокого уровня.

Хотя на языке высокого уровня программу написать легче, с помощью языка ассемблера можно создавать программы, которые выполняются быстрее. По этим соображениям в процессорах используются оба языка. Иногда критический ко времени участок программы необходимо писать на языке ассемблера. Полная программа может иметь участки в кодах ассемблера и участки на языке высокого уровня. Программирование на обоих языках легко комбинировать в линейно выполняемой программе. Языки ассемблеров и языки высокого уровня позволяют программировать процессоры для выполнения разнообразных функций.

Программные симуляторы имитируют работу ЦСП (пример симулятора, разработанного на Java можно посмотреть на сайте http://dsp.iu4.bmstu.ru - раздел «Виртуальная лаборатория»). Симулятор процессора представляет собой программную модель ЦСП, который моделирует почти все функциональные возможности и напряжения на выводах процессора. Симуляторы используются для отладки программы и анализа результатов, прежде чем программа будет размещена в памяти процессора. Симуляторы весьма полезны для предварительной оценки работы конкретного процессора.

Эмулятор действует почти так же, как ЦСП, но позволяет контролировать и наблюдать результаты выполнения команд. Современные эмуляторы не заменяют ЦСП, но контролируют состояние функциональных узлов и блоков ЦСП путем последовательного сканирования. Используя эмулятор, можно увидеть все внутренние обмены в ЦСП на каждом шаге. Разработчики могут в реальном времени пошагово выполнять команды, проверяя уровни напряжения на соответствие конкретной операции и контролируя каждый результат, во времени. Эмуляторы являются неоценимым сродством в среде проектирования систем ЦОС.

Поскольку возможность реализации проверяется посредством имитационного моделирования, следует начать с составления программы. Прежде всего, разрабатывается

программное обеспечение модели, как правило, на языке, далее с использованием симуляторов, эмуляторов и т.п. проводится привязка модели к особенностям аппаратной реализации, далее проводится оптимизация исходного кода на языке ассемблера. На первом этапе определяются сложность и модули программы. Каждый модуль отлаживается отдельно. После отладки всех модулей компилируется и отлаживается полная программа. Если она работает согласно требованиям, то переходят к следующему этану проектирования, в противном случае цикл проектирования повторяется до тех пор, пока не будет выполнено техническое задание на данном этапе.

3.1. Интегрированная среда разработки Visual DSP

Интегрированная среда разработки Visual DSP была разработана компанией Analog Device в целях повышения удобства работы с отладочными устройствами. Пакет Visual DSP включает в себя две составляющие: интегрированную среду разработки (IDE) и отладчик.

3.1.1 Интегрированная среда разработки

Интегрированная среда разработки является MDI-приложением и включает себя средства настройки проекта, полнофункциональный текстовый редактор, средства настройки и вызова ассемблера, линкера и отладчика. Окно среды разработка показано на рис.3.3.

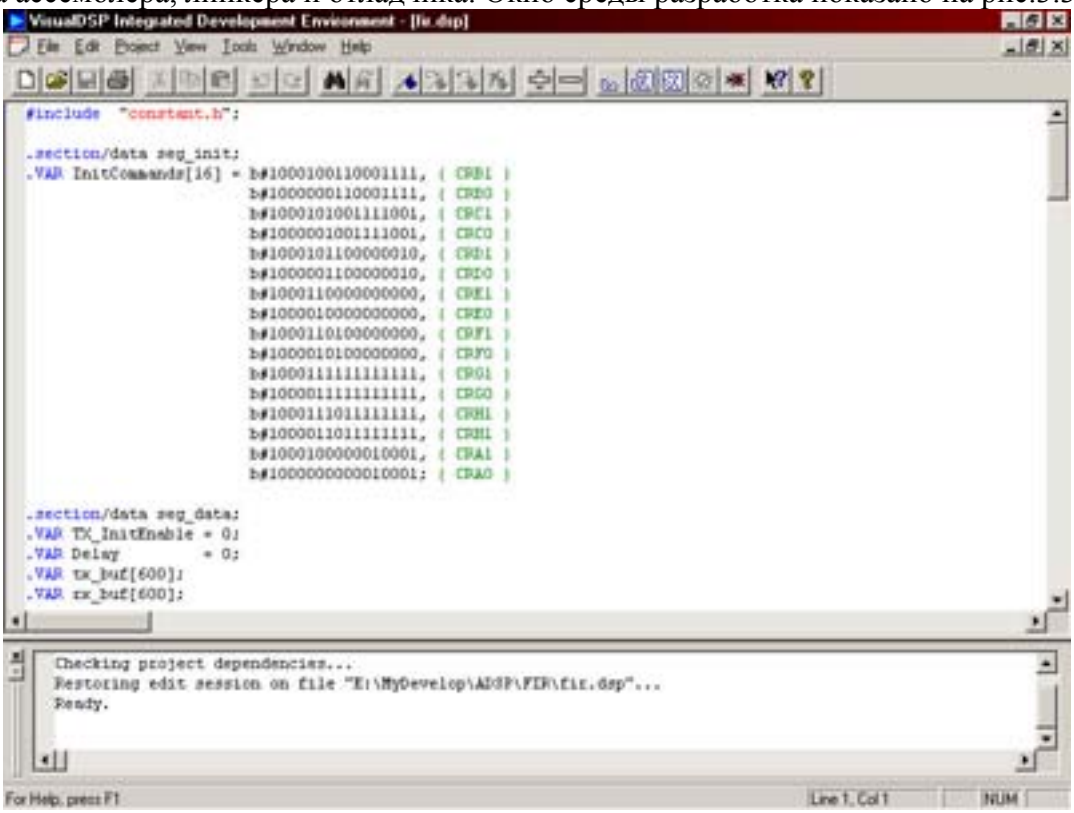

Рис.3.3.Вид интегрированной среды разработки

3.1.2 Создание нового проекта

Для создания нового проекта необходимо выбрать пункт New в меню File (см. Рис. 3.4).

Рис. 3.5 Дерево проекта

Появиться окно со списком действий, необходимо выбрать New Project. После этого на экране появится окно с корнем дерева проекта (см. Рис.3.5). Сначала оно пустое, для успешного создания проекта в него необходимо добавить минимум два файла: файл архитектуры (.LDF) и файл с исходным текстом (.DSP в случае написания программы на языке ассемблера) Перемещение между файлами проекта осуществляется выбором необходимого файла из дерева проекта. Добавление - удаление файлов проекта осуществляется с помощью кнопок панели инструментов или с помощью меню Project.

3.1.3 Компиляция проекта

Для компиляции проекта следует в зависимости от необходимости использовать следующие кнопки панели инструментов:

При компиляции одного файла .

При компиляции всех файлов .

В нижнем окне отображается процесс компиляции, и все найденные ошибки,

3.1.4 Запуск отладчика

Запуск отладчика осуществляется нажатием на кнопку панели инструментов. Перед первым запуском отладчик определяет отладочное средство: симулятор или EZ-KIT см. Главу 4). Если в качестве отладочного средства используется EZ-KIT то программа пытается с ним связаться через последовательный порт RS-232. При установлении связи открывается окно программы (см. Рис. 3.6).

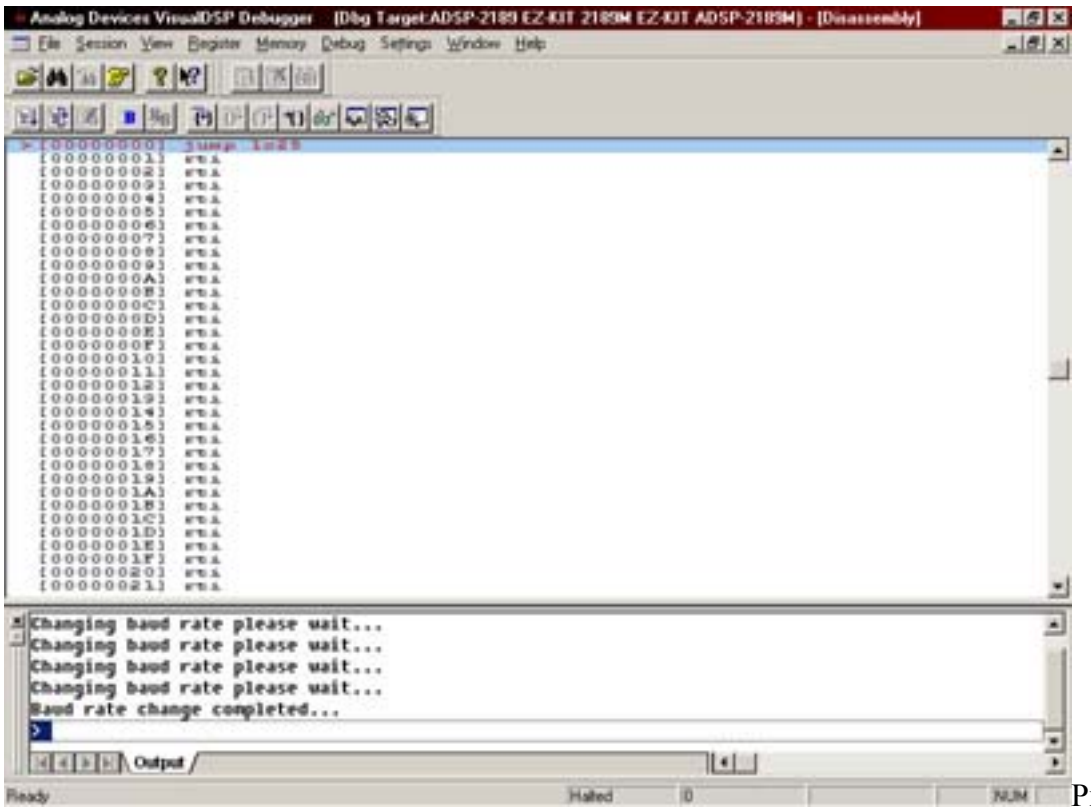

Рис. 3.6 Окно отладчика

Для запуска программы в непрерывном режиме нажмите клавишу F5 или выберите меню Debug и пункт Run (см. Рис. 3.7).

Рис.3.7 Окно отладчика

Для выполнения программы до курсора надо выбрать пункт

Run To Cursor или нажмите Ctrl+F10 или нажмите на панели инструментов. Для трассировки программы используйте пункт Step Over (пошаговый режим без захода в подпрограммы, кнопка панели инструментов) или Step Into (пошаговый режим с заходом в подпрограммы, кнопка панели инструментов). Отладчик позволяет

использовать точки останова и просматривать ресурсы процессора. Окно ресурса вызывается на экран через меню Register (см. Рис 3.8). Пункт Computational отображает содержимое регистров (основного банка), пункт Alternate Computational отображает содержимое альтернативного банка регистров.

Рис.3.8 Меню Register

Пункт DAGS отображает состояние генераторов адресов. Пункт Program Control отображает окно с состоянием внутренних регистров процессора. Пункт Interrupt предназначен для вызова окна, отображающего регистры контроллера прерываний. Выбор пункта Flags приводит к отображению состояния флагов процессора. Пункты SPORTO и SPORT1 отображают состояния последовательных портов SPORTO и SPORT1 соответственно. Меню Stacks предназначено для отображения стеков процессора.

Задание:

- Установите на своей рабочей станции Visual DSP для ЦСП 218x.
- Выполните проверку на работоспособность пакета Visual DSP, изучив тестовые программы, приведенные в каталоге sample.

3.2 Основные понятия языка ассемблера

Как было отмечено, любые действия ЦСП по обработке данных регламентируются набором инструкций, представленных на языке Ассемблер для данного типа ЦСП. Часть программы на языке ассемблера называют модулем; модули, которые являются входными данными ассемблера, называются исходными модулями. Каждый

модуль должен содержаться в отдельном файле. Ассемблированные модули линкер связывает в программу.

Ассемблер включает три исполняемых компонента:

- Препроцессор языка С
- Препроцессор языка ассемблера
- Ядро ассемблера

3.3 Препроцессор

Препроцессор языка С обрабатывает директивы С, например, #define, #include. Препроцессор языка ассемблера обрабатывает директивы ASM-218x, например, VAR, CONST.

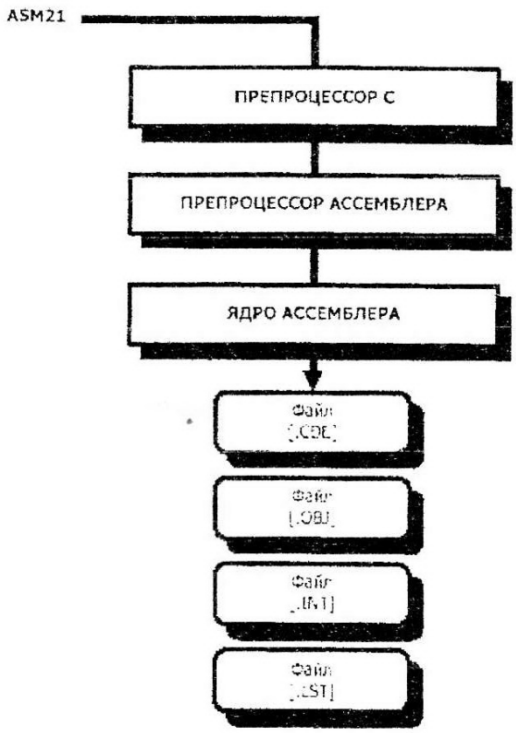

Рис.3.9. Процесс выполнения ассемблера

3.4 Запуск ассемблера

Запуск ассемблера выполняется командой *ASM21 filename [.ext] [swich...]*

Где filename - входной файл, содержащий модуль исходного кода. Дополнительные ключи управляют работой ассемблера, они могут быть набраны как в верхнем, так и в нижнем регистре.

Таблица 3.1 Ключи ассемблера

Ключ	Воздействие
-c	Устанавливает чувствительность к регистру символов
-dид=[символ]	Определяет идентификатор для С препроцессора, аналогичен *#define*
-i [глубина]	Раскрывает содержимое включенных файлов в листинге
-l	Создает файл листинг
-m[глубина]	Раскрывает макросы в листинге
-o имя_файла	Переименование выходного файла
-s	Отменяет семантическую проверку сложных инструкций
-215n	Специальная ассемблерная инструкция для процессоров ADSP-21msp5x
-2171	Специальная ассемблерная инструкция для процессоров ADSP-217x
-2181	Специальная ассемблерная инструкция для процессоров ADSP-218x

3.5 Правила языка ассемблера

3.5.1 Символы и ключевые слова

В языке ассемблера ADSP-21xx допускается использовать следующие символы:

- Заглавные буквы латинского алфавита - A-Z
- Строчные буквы латинского алфавита - a-z
- Цифры - 0-9
- ASCII графические символы, символы пунктуации
- Управляющие ASCII символы: пробел, табуляция и т.д.

3.5.2 Идентификаторы

Идентификаторы нужны для обозначения различных объектов программы - переменных, подпрограмм, меток и т.п. В языке ассемблера идентификатор - это последовательность из латинских букв, цифр и следующих знаков: “_”, “-”. Различаются идентификаторы длиной до 32 символов. Кроме того, идентификаторы являются зависимыми от регистра, то есть идентификатор WORD и идентификатор word - различны.

Запрещено начинать идентификаторы с цифры.

Зарезервированы следующие идентификаторы:

ABS	DO	JUMP	MR2	RX0
AC	EMODE	L0	MSTAT	RX1
AF	ENA	L1	MV	SAT
ALT_REG	ENDMACRO	L2	MX0	SB
AND	ENDMOD	L3	MX1	SEG
AR	ENTRY	L4	MY0	SEGMENT
AR_SAT	EQ	L5	MY1	SECTION
ASHIFT	EXP	L6	NAME	SET
ASTAT	EXPADJ	L7	NE	SETDATA
AUX	EXTERN	LE	NEG	SHIFT
AV	EXTERNAL	LEFTMARGIN	NEWPAGE	SI
AV_LATCH	FILE	LOCAL	NOP	SR
AX0	FOREVER	LOOP	NORM	SR0
AX1	FLAG_IN	LSHIFT	NOT	SR1
AY0	FLAG_OUT	LT	OR	SS
AY1	GE	M0	PAGE	SSTAT
BIT_REV	GLOBAL	M1	PAGELENGTH	STATIC
BM	GT	M2	PAGEWIDTH	STS
BY	I0	M3	PASS	SU
C	I1	M4	PC	TEST
CACHE	I2	M5	PM	TIMER
CALL	I3	M6	POP	TYPE
CE	I5	M7	PORT	TOGGLE
CIRC	I6	MACRO	POS	TOPSTACK
CLR	I7	MF	PREVIOUS	TRAP
CLEAR	ICTRL	M_MODE	RAM	TRUE
CNTR	IDLE	GO_MODE	REGBANK	TX0
CONST	IF	MODIFY	RESET	TX1
DIS	IFC	MODULE	RND	UNTIL
DIVS	IMASK	MR	ROM	US
DIVQ	INCLUDE	MR0	RTI	UU
DM	INIT	MR1	RTS	VAR
				XOR

3.5.3 Форматы представления данных

В языке ассемблера ADSP-218x имеются средства для записи целых чисел в двоичной, десятичной, восьмеричной и шестнадцатеричной системах счисления. Десятичные числа записываются без спецификаторов, шестнадцатеричные - с префиксным спецификатором 0x (нуль и x) или H#, восьмеричные - с префиксным спецификатором 0(нуль), двоичные - с префиксным спецификатором B#.

Пример:

Определение различных чисел языка ассемблера ADSP-218x

Десятичные

- 12
- +67
- -89

Восьмеричные

- 066
- +077
- -054

Внимание - не используйте префикса 0 для десятичных чисел, поскольку, например 044<>44

Шестнадцатеричные

- OxOFl
- 0xA2
- H#1B

Двоичные

- B#011
- B#1001001

3.5.4 Арифметические и логические операторы языка ассемблера

Язык ассемблера поддерживает арифметические и логические выражения. Существуют две разновидности выражений:

- Арифметические или логические операции с двумя или более целыми константами (25 + 14, (15 — 4)*4, OXff & H#12)
- Операция плюс или минус числовая константа (flag + 1, data - 6)

Операторы языка ассемблера схожи с операторами языка С.

- +,- - сложение, разность
- *,/ - умножение, деление
- % - деление по модулю
- «,»- двоичный сдвиг влево и вправо
- & - логическое И (AND)
- | - логическое ИЛИ (OR)
- ^ - логическое ИСКЛЮЧАЮЩЕЕ-ИЛИ (XOR)
- LENGTH(symbol) - размер symbol
- ADDRESS (symbol) — начальный 16-разрядный адрес symbol
- symbol - адрес symbol
- PAGE(symbol) - значение старших 8 бит адреса symbol

Адрес буфера и операторы определения длины

Язык ассемблера поддерживает два специальных оператора. Оператор определения адреса ^ и оператор определения длины %.

^bufer - Возвращает адрес буфера

%bufer - Возвращает размер буфера

Пример:

Пример работы с буффером .VAR buf[3] – 1,2,3;

I0 = ^buf; / в 10 заносится адрес buf */*

*L0 = %buf /*в L0 заносится размер буфера, то есть - 3 */*

Сложением и вычитанием констант могут быть образованны простые выражения

^bufer + constant

^bufer - constant

%bufer + constant

%bufer - constant

Пример: Пример адресной арифметики

^buf + 3

%buf - 1

Смысл подобных выражений следующий: в случае оператора адреса - мы модифицируем этот адрес, а в случае оператора определения длины - изменяем длину. Не рекомендуется использовать увеличение длины, гак как это может привести к корреляции данных, что, в свою очередь, приводит к трудноопределимым ошибкам.

3.5.6 Комментарии

Вы можете вставлять комментарии в любом месте исходного текста, заключив их в скобки {}, за исключением строк директив С препроцессора. Применение вложенных комментариев не разрешено. Чтобы использовать комментарии в строках содержащих директивы С препроцессора (начинающиеся с символа #) используйте правила С

#директива /* комментарий */

3.5.7 Директивы препроцессора С

Ассемблер ASM-21xx включает С препроцессор, который позволяет использовать директивы, указанные в таблице

Таблица 3.2 Директивы препроцессора

Директива	Значение
#include	Вставить текст из другого исходного файла
#defme	Макроопределение. Позволяет определять различные макросы, например *#define add AR = AXO + AY0*
#undef	Отмена макроса, например, после директивы *#undef add* макрос add перестаёт существовать
#if	Директива условного ассемблирования, условие задается некоторой константой
#ifdef	Директива условного ассемблирования, условие задается наличием выбранного макроопределения
#ifndef	Директива условного ассемблирования, условие задается отсутствием выбранного макроопределения
#else	Включение текста по альтернативной ветке дирректив #if , #ifdef, #ifndef
#endif	Завершения включения условного текста

3.6 Директивы языка ассемблера

Директивы ассемблера управляют процессом ассемблирования. Они обрабатываются препроцессором, но в отличие от инструкций не генерируют при ассемблировании код. Директива ассемблера начинается с точки и заканчивается точкой с запятой. Некоторые директивы имеют параметры и аргументы. Параметры следуют сразу за директивой и разделяются косой чертой; аргументы следуют после параметров.

.DIRECTIVE/параметр/параметр ...аргумент; {комментарий}

3.6.1 Программные модули (.MODULE)

Директива .MODULE обозначает начало программного модуля и определяет название модуля. Исходный файл может содержать только один модуль.

Директива имеет форму:

.MODULE/параметр/параметр... имя модуля;

В качестве параметров могут выступать: RAM или ROM тип памяти, ABS=адрес абсолютный стартовый адрес (не используйте с STATIC), SEG=seg_name размещение в указанном сегменте, BOOT=0-7 размещение копии на странице начальной загрузки, STATIC статичное размещение модуля в памяти

Параметры BOOT и STATIC используются только в системах с памятью начальной загрузки (т.е. все процессоры семейства, за исключением ADSP-2100). Второй способ размещения модулей на страницах начальной загрузки реализуется при использовании ключа редактора связей -i.

Если тип памяти не определен, то по умолчанию принимается тип RAM (ОЗУ). Параметр ABS размещает код модулей программ в определенных адресах памяти программ, что делает их неперемещаемыми. Это означает, что редактор связей резервирует память для модулей по указанным адресам. Модули, которые не имеют параметра ABS, перемещаемы. Параметр SEG размещает модуль в указанный сегмент памяти, который объявлен в файле системной конфигурации. Если вы определяете оба параметра ABS и SEG и указываете абсолютные адреса, которых нет в данном сегменте, вы увидите сообщение об ошибке при запуске редактора связей.

Параметр BOOT используется для размещения копии модуля на странице памяти начальной загрузки с указанным номером. Модуль будет сохранен в памяти начальной загрузки памяти до тех пор, пока он не будет загружен и выполнен. Можно разместить копии модулей на нескольких загрузочных страницах, позволяя получить доступ к его программам или данным, например:

Пример: Декларация модуля

.MODULE/BOOT-0/BOOT=1/BOOT=2

Другой способ реализации этого, использование параметра STATIC который сохраняет модуль в памяти программ, когда загружаются страницы начальной загрузки.

Параметр BOOT применяется также ко всем переменным .VAR и объявлениям буферов данных внутри модуля - помните, что память начальной загрузки и память программ, обычно в основном, содержат как программу, так и данные.

Директива .ENDMODE указывает на завершение программного модуля. Программа ассемблера останавливается, когда достигает директивы .ENDMODE. Примеры объявления модулей:

Пример: Объявление перемещаемого модуля filter-routine размещенного в сегменте памяти с именем fir, который определен в выходном .LDF файле системного конфигуратора.

.MODULE/SEG=fir filter-routine;

Пример: Объявляение модуля main-prog, который должен быть размещен в памяти RAM по адресу 40 (шестнадцатеричный)

.MODULE/RAM/ABS=0x0040 main-prog;.

Система может иметь до 8 загружаемых страниц. Когда выбираются атрибуты загружаемых модулей параметрами RAM, ROM, SEG и ABC, они применяются к памяти, где размещен код, во время выполнения, а не к памяти начальной загрузки. Таким образом, при конфигурировании распределения памяти во время исполнения, вы должны оперировать терминами памяти программ и памяти данных. Редактор связей определяет расположение программ и данных в памяти, в соответствии с вашими объявлениями сегментов для системного конфигуратора и вашим объявлением модуля на ассемблере. Редактор связей конструирует также страницы памяти начальной загрузки, но вы не можете напрямую указать расположение модуля в загрузочной памяти.

Процессор не может получить и выполнить инструкцию из памяти начальной загрузки; содержимое страницы памяти начальной загрузки должно быть вначале загружено во внутреннюю память программ, и только затем код выполняется. Если вы хотите, чтобы модуль (или переменная/буфер) существовали во внутренней ли внешней памяти процессора, в течение выполнения некоторой страницы начальной загрузки, необходимо ассоциировать ее определителем BOOT с этой страницей. Это заставит редактор связей оставить пространство для объекта во время выполнения кода страницы.

Выходной файл листинга редактора связей .шар показывает расположение вашей программы в области загрузочной памяти, также как соответствующее отражение в памяти программ во время выполнения. Например, следующая директива заново объявляет модуль main_prog от объявленного ранее. Модуль будет сохранен на странице 0:

Пример: Сохранение модуля main_prog на странице 0

```
.MODULE/RAM/ABS-0x0040/BOOT=0 main_prog;
```

параметры RAM и ABS этой директивы применяются к внутренней памяти программ процессора (считая, что MMA=0).

Далее приведен пример, который сохраняет копии перемещаемого модуля на нескольких загрузочных страницах:

Пример: Сохранение копии перемещаемого модуля shifter на нескольких загрузочных страницах

```
.MODULE/RAM/BOOT=0/BOOT=2/BOOT=3 shifter;
```

Если разрабатывается программный модуль, который будет использоваться на нескольких страницах начальной загрузки, например, содержащий подпрограммы, вы захотите, чтобы код оставался на месте, при загрузке различных страниц. Для того чтобы достигнуть этого, при объявлении имени модуля нужно указать параметр STATIC Параметр ABS не может использоваться совместно с параметром STATIC. Определитель STATIC предотвратит перезапись модуля, как в том случае, когда при сбросе загружается страница 0 (если MMAP=0), так и в том, когда программно вызываются страницами 1-7. Редактор связей гарантирует это при размещении вашей программы в памяти. Если модуль не объявлен как STATIC, он может быть частично или полностью перезаписан содержимым какой-либо загрузочной страницы. Это применимо к модулю как в случае с внутренней, так и с внешней памятью.

Когда редактор связей распределяет память для хранения вашей программы, он разбивает ее на 9 независимых частей: незагружаемая память программ/данных и загружаемые страницы 0-7. Незагружаемая память определена как начальная структура памяти программ и памяти данных перед загрузкой какой-либо страницы или исполнения кода. Девять частей структуры содержатся независимо одна от другой, если в объявлении программного модуля (или буфера данных) не используется параметр STATIC. При

отсутствии параметра STATIC редактор связей принимает, что каждая из девяти частей стартует с чистого состояния памяти программ и памяти данных, и что каждая загружаемая страница имеет доступ ко всему пространству памяти.

Следуйте следующему правилу: если у вас есть программный модуль или буфер данных, который не должен быть перезаписан при загрузке новой страницы, вы должны использовать при его объявлении параметр STATIC. В противном случае, редактор связей принимает, что для загрузки новой страницы доступна вся память, и допустимо переписывать любые существующие программы/данные.

Пример: Декларация модулей routinel и bootfilter

.MODULE/SEG=ext_pm/RAM routinel;

.MODULE/RAWBOOT=0 bootfilter;

.VAR/SEG=ext_pm/PM/RAM coeffs[16];

Пока подпрограмма *routinel* не объявлена как STATIC, редактор связей игнорирует ее расположение в памяти при определении места для буфера *coeffs* в памяти программ. Поэтому редактор связей может зарезервировать место для *coeffs* в области памяти, которая частично перекрывает *routinel*. При загрузке страницы 0,буфер *coeffs* загружают во внутреннюю память программ ADSP-218x,откуда она копируется во внешнюю память программ PM. Тем не менее, если при объявлении *routinel* указать параметр STATIC редактор связей зарезервирует ее адреса в пространстве памяти программ и расположит coeffs где-нибудь в другом месте внешней памяти программ.

Пример: Декларация статического модуля routinel
.MODULE/SEG=ext_pm/RAM/STATIC routinel;

3.6.2 Переменные данные и буфера (.VAR)

Директива .VAR объявляет буферы данных. Буфер данных - это множество адресов расположения данных. Переменная объявляется как буфер с единственным адресом. Вы должны объявить все переменные и буфера до использования их в программе. Если буфер инициализирован директивой .INIT объявление и инициализация должны выполняться в одном и том же модуле. Директива .VAR имеет форму:

.VAR /параметр/параметр ...имя буфера [длина],... ;

Объявление по умолчанию подразумевает однословную переменную, перемещаемую в памяти RAM. Одна директива .VAR может объявить несколько буферов, разделенных запятыми на одной строке (до 200 символов).

При многочисленном объявлении переменных и буферов в одной строке, редактор связей размещает их в смежных областях памяти. Если при таком объявлении используется параметр CIRC, то создается единственный кольцевой буфер, а отдельные буфера будут линейными. С директивой могут использоваться следующие параметры: PM или DM, размеры в программной памяти или данных RAM или ROM, тип памяти ABS=адрес абсолютный адрес (не использовать со STATIC), SEG-сегмент размещение буфера в сегменте, объявленным системным конфигуратором, CIRC кольцевой буфер, STATIC предотвращает перезапись буфера во время загрузки. Параметр STATIC используют только для систем с памятью начальной загрузки, т.е. для всего семейства процессоров, за исключением ADSP-2100.

Буфера могут быть размещены как в памяти программ, PM, так и в памяти данных, DM по умолчанию. Тип памяти по умолчанию устанавливается в RAM для памяти DM и PM. Параметр ABS размещает буфер с указанного стартового адреса, делая его неперемещаемым. Параметр SEG размещает буфер в указанном сегменте памяти, который был объявлен в файле системного конфигуратора.

Параметр CIRC определяет кольцевой буфер. Буфер будет адресоваться в линейном виде, если не применен атрибут CIRC. Параметр STATIC предотвращает перезапись буфера, когда загружается страница начальной загрузки. Если вы хотите использовать буфер в программах с нескольких загрузочных страниц, он должен оставаться неизменным во время загрузки. Это выполняется приданием буферу атрибута STATIC. Статические буфера управляются редактором связей точно так же, как статические модули - смотрите раздел «Статические модули» для уточнения деталей.

Для объявления переменной применяется директива .VAR без указания длины буфера:

Пример: Объявление переменной seed с адресом OxOOOA .
.VAR/DM/RAM/ABS=0x000A seed;

Этот оператор объявляет однословную переменную с именем seed в памяти данных RAM по адресу 10 (десятичный). Следующий пример показывает объявление буфера:

Пример: Объявление линейного буфера coefficients в сегменте pmdata памяти программы

.VAR/PM/RAM/SEG=pmdata coefficients[10];

Здесь линейный буфер объявлен в памяти программ RAM, который перемещаем в пределах сегмента с именем pmdata. Название буфера coefficients и он состоит из 10 записей в памяти программ. Длина буфера должна быть помещена внутри скобок. Ниже приведен пример объявления перемещаемого кольцевого буфера, длина которого определяется величиной константы laps

Пример: Объявление кольцевого буфера data_buffer в памяти данных .CONST taps = 15;

.VAR/DM/CIRC data_buffer[taps];

Кольцевой буфер может быть размещен в памяти с некоторыми ограничениями, связанными с характеристиками процессоров ADSP-21xx по аппаратной реализации адресации кольцевого буфера. Вообще, кольцевой буфер должен стартовать с базового адреса, который кратен 2n, где n - количество бит, требуемых для представления длины буфера в двоичном виде. Редактор связей будет придерживаться этих требований для размещения кольцевых буферов. Вы должны иметь это в виду, если явно выбрали базовый адрес буфера параметром ABS. Приведенные ниже примеры помогут понять, где можно размещать циклические буферы в памяти. Следующий оператор объявляет кольцевой буфер из пяти позиций:

Пример: Объявление кольцевого буфера aa

.VAR/CJRC aa[5];

Так как для представления длины *aa* необходимо три разряда, редактор связей присвоит буферу базовый адрес кратный 8.Три младших значимых разряда (МЗР) этого адреса нули. Если на одной строке объявлены несколько буферов и используется параметр CIRC, создается один кольцевой буфер, а отдельные буфера будут только простыми линейными буферами. Например, следующее объявление создает один 15-словный кольцевой буфер.

Пример: Объявление одного кольцевого буфера

.VAR/CIRC aa[5], bb[5], cc[5];

Базовый адрес кольцевого буфера равен адресу *aa*; этот символ будет использоваться для доступа к буферу в программе. Адрес *bb* это *aa* +5, а адрес cc это aa+10. Три пятисловных буфера могут быть индивидуально доступны как линейные. Так как величина 15 требует четырех разрядов для двоичного представления, кольцевой буфер aa будет размещен по адресу, который кратен 16 (четыре младших значащих разряда равны нулю).

Следующий пример показывает использование трех директив для объявления трех различных циклических буферов:

Пример: Объявление трех кольцевых буферов

.VAR/CIRC aa[5];

.VAR/CIRC bb [5];

.VAR/CIRC cc[5]:

Поскольку они объявлены отдельно, буфера не будут объединены. В следующем примере создаются структуры для поисковых таблиц синуса/косинуса:

Пример: Объявление одного кольцевого буфера для поисковых таблиц синуса и косинуса

.VAR/CIRC sin[256], cos[768],

Здесь определен один кольцевой буфер, который имеет длину 1024.Чтобы получить доступ к буферу из программы, вы должны инициализировать индексные регистры DAG и регистры длины буфера следующими инструкциями:

Пример: Загрузка указателей на поисковые таблицы в регистры

I0 = ^cos; {^ это оператор получения адреса}

L0=1024;

I1= ^sin;

L1=1024;

Эти инструкции загружают регистры I0 и I1 базовыми адресами cos и sin. Соответствующие регистры L загружаются значением длины кольцевого буфера для разрешения кольцевой адресации. Кольцевой буфер реализуется только при ненулевом значении регистра L.

3.6.3 Инициализация переменных и буферов (.INIT)

Директива .INIT используется для инициализации переменных и буферов в ПЗУ. Редактор связей помещает данные инициализации в файл образа памяти, который загружается разделителем программ для записи в ППЗУ. Разделитель переводит ПЗУ части этого файла в формат, совместимый с промышленным стандартом программатора ППЗУ. Инициализирующие значения могут быть перечислены в директиве или указаны во внешнем файле; директива .INIT принимает одну из следующих форм:

.INIT имя_буфера: константа, константа,...;.

.INIT имя_буфера: ^.....буфер или %другой_буфер, ...;

.INIT имя_буфера: <файла >;

Операторы ^ и % использоваться для инициализации буфера или переменной базовым адресом или длиной, или даже другими буферами. Любые комбинации констант, указателей адресов буфера и величин длины буфера могут быть заданы через запятую. Примеры:

Пример: Примеры инициализаций

/ Этот оператор инициализирует переменную seed шестнадцатеричной константой*/*

.INITseed: 0x3FFF;

/ Этот оператор инициализирует буфер seed_values списком констант */*
.INIT seed_values: 1,2,3,5,7;

/ Здесь переменная buffer_ptr инициализируется указателем стартового адреса буфера input_buf Вы можете инициализировать только часть данных буфера, задавая смещение */*

INIT buffer_ptr: ^input_buf,

/ Теперь инициализирующие величины будут размещены, начиная с адреса; buffer_name + offset */*

.INIT buffer_name[offset] ;

/ Следующий оператор, например, инициализирует восьмой, девятый и десятый элементы буфера coeffs величинами 2,3 и 4 */*

.INIT coeffs[7] : 2,3,4;

Третья форма директивы .INIT указывает имя файла, который содержит инициализирующие величины. Ассемблер устанавливает указатель на этот файл, и данные присоединяются при запуске редактора связей. Следующий пример заставляет редактор связей инициализировать буфер cos содержимым файла *cosines.dat*:

Пример: Инициализация буфера cos

.INIT cos: <cosines.dat>;

Если файл с данными находится в текущей директории операционной системы, только необходимо указать в скобках только имя файла. Если файл находится в другом каталоге, вы должны указать путь к этому каталогу и имя файла. Например, если файл inits.dat для буфера с именем samples размещен в директории DOS C:\2101\filter3\,тогда директива .INIT должна быть применена следующим образом:

Пример: Инициализация буфера samples

.INIT samples : < c:\2101\filter3\inits.dat>;

Это позволит редактору связей найти файл. Данный способ широко используется для загрузки буферов данными, выработанными другими программами, такими как нахождение коэффициентов фильтра. После того, как редактор связей считает и присоединит содержимое файла, изменение данных потребует лишь выполнить повторную компоновку программы. Переменные данных и буферов могут еще быть инициализированы с помощью семиразрядного ASCII кода. Следующий пример инициализирует один четырехпозиционный буфер данных inputs ASCII кодами A,B,C,D. ASCII коды размещаются в семи МЗР памяти данных (16 разрядов) или памяти программ (8,16 или 24 разрядов).

Пример: Инициализация буфера inputs

.INIT inputs : _ABCD_;

Специальный синтаксис директивы .INIT .INIT24 позволяет сохранять 24-разрядные данные в памяти программ. Это позволяет получить доступ к младшим 8 разрядам каждого 24- разрядного слова памяти программ при инициализации буферов данных или переменных в исходной программе.

Пример: Примеры адресной арифметики при инициализации

/ Эта запись вычисляет 16-разрядный адрес */*

.INIT var: ^label + 10

/ То следующая запись вычисляет 24-разрядный адрес */*

.INIT24 var: ^label + 10

Созданный редактором связей .EXE образ памяти содержит программные коды и инициализирующие данные. Генерация этого файла не гарантирует, тем не менее, что программа и данные будут загружены в память; файл только перечисляет, что должно быть представлено в памяти, чтобы программа запустилась корректно. Вы должны предоставить данные, которыми память будет по-настоящему загружена. После того как редактор связей создаст файл образа памяти для вашей программы, существует два способа завершить инициализацию: 1) запрограммировать микросхему ППЗУ для буферов, расположенных в ПЗУ. 2) записать в вашу программу код, копирующий инициализирующие данные в буферы, расположенные в ОЗУ. Вы можете инициализировать буферы, расположенные в ПЗУ используя разделитель программ для записи в ППЗУ одним из следующих способов: создание файлов для программирования приборов ППЗУ для внешней памяти программ или данных, или создание файлов для программирования приборов ППЗУ для памяти начальной загрузки; переменные и буферы во внутренней памяти программ будут инициализированы во время загрузки (для процессоров ADSP-21xx с внутренней и загрузочной памятью). Переменные и буферы, расположенные в памяти программ или данных ОЗУ должны быть инициализированы вашей программой. Исключением является внутренняя память программ ОЗУ ADSP-2101.ADSP- 2105,ADSP-21msp50.3TO пространство ОЗУ может быть инициализировано при загрузке (как описано выше). Три типа памяти должны быть инициализированы в исходном коде:

- внешняя память программ ОЗУ
- внешняя память данных ОЗУ
- внутренняя память данных ОЗУ

Чтобы инициализировать буферы в этих областях памяти вы должны иметь данные, сохраненные в ПЗУ, и включить в вашу программу код, копирующий данные в соответствующую область памяти. Пример такой подпрограммы показан ниже:

Пример: Копирование буфера sin_input в буфер sin table

```
.VAR/PM/ROM sin_init[64];

.VAR/DM/RAM sin_table[64];

.INIT sin_init: <sin.dat>;

{копировать инициализированный буфер, sin_init,из PM ROM в DM RAM}

M0=1;

M4=1;

IO=^sin_table;

I4=^sin_init;

CNTR=%sin_table;

DO sin_copy UNTIL CE;

AXO=PM(I4,M4);

sin_copy: DM(IO,MO)=AXO;
```

3.6.4 Определение макроса (.MACRO)

Макрос создается с помощью ассемблерной директивы .MACRO Макрос используется для повторения часто используемых последовательностей инструкций в вашем исходном коде. Передачей аргументов макросу реализуется подобие подпрограммы, которая может быть использована в различных программах.

Макро вложения ограничены только размером свободной оперативной памяти. Вложенные макросы должны быть объявлены следующим образом: внутренний макрос

первый,..., внешний макрос последний. Все константы, используемые в макросах, должны быть объявлены перед объявлением макросов.

Макрос определяется двумя директивами:

.MACRO имя макроса (, аргумент,...);

...

.ENDMACRO;

Каждый оператор внутри макроса может быть инструкцией, директивой или макро включением. Директива .ENDMACRO отмечает конец макроопределения. Макрос вызывается по своему имени. Чтобы выполнить макрос с именем quickloop используйте следующую команду в вашем исходном коде:

quickloop; вызов макроса

Макровызов не должно содержать дополнительных операторов (т.е. инструкций, директив препроцессора или других макровключений) на той же строке исходного кода. Аргументы макроса принимают форму:

%n n = O, 1, 2, ... 9

Следующий пример определяет макрос с тремя аргументами:

Пример: Пример определения макроса с тремя аргументами

.MACRO memory_transf(%0, %1, %2);

В коде макроса, аргументы маркируются служебными символами %1,%2,%3,и т.д. При вызове макроса служебные символы замещаются величинами аргументов, переданных в макрос. Должно быть передано правильное число аргументов. Передаваемые аргументы могут быть одними из:

Аргумент	Исключения
константы или выражения	Нет
символы	все, кроме MACRO, ENDMACRO, CONST, INCLUDE
^символ	"^%n"не разрешено
%буфер	"%%n"не разрешено

Операторы ^ и % не могут быть использованы с аргументами, замещающими служебные символы в макроопределении. Тем не менее, аргументы, переданные в макрос, могут использовать эти операторы. Например:

Пример: Вызов макроса

reed_data (^input),

Пример: Макрос копирующий содержимое одного буфера в другой

{MACRO объявление}

```
.MACRO memory_transf (%0, %1, %2, %3, %4) { допускает 5 аргументов }

        .LOCAL transf;

        14=%0; {устанавливает 14 как адрес источника}

        15=%1; {устанавливает 15 как адрес приемника}

        M4=1; {устанавливает указатель на инкремент 1}

        CNTR=%2; {устанавливает длину буфера}

        DO transf UNTIL CE; { перенос данных }

        SI=%3(I4, M4) {переносит из типа %3 памяти}

        transf:

        %4(I5, M4)=SI; {переносит в тип %4 памяти}

        .ENDMACRO

        {MACRO вызов}

        Memory_transf (^coeff_table, ^buffer, buff_length, PM,DM):
```

3.6.5 Локальная метка в макросах (.LOCAL)

Директивой .LOCAL задают программные метки, используемые в макросе. Директива .LOCAL указывает ассемблеру создавать уникальную версию метки при каждом включении макроса. Это предотвращает ошибку дублирования меток в случае, когда макрос вызывается несколько раз в одном программном модуле. Директива .LOCAL имеет формат:

.LOCAL метка_макроса,…;

Ассемблер создает уникальные версии метки макроса добавляя к ней номер; это может посмотреть в программе моделирования или в файле листинга .LST разрешено раскрытие макросов.

3.6.6 Размещение программ и данных в сегментах памяти

Для инициализации данных, сохраненных в загрузочной памяти, загрузчик должен выполнить операцию копирования из внутренней памяти программ, после окончания загрузки.

Директива .SECTION подобна параметру /SEG директив .MODULE и .VAR и имеет следующий формат:

.SECTION/сегмент имя_сегмента_pm

Директива .SECTION/PM имя_сегмента_pm указывает редактору связей на необходимость разместить все программы и данные модуля в сегменте имя_сегмента_pm памяти программ. Директива .SECTION/DATA имя_сегмента_dm указывает редактору связей на необходимость разместить все структуры данных модуля в сегменте имя_сегмента_dm, памяти данных. Сегменты имя_сегмента_pm и имя_сегмента_dm должны быть предварительно определены в файле описания архитектуры .LDF системного конфигуратора. Обычно, чтобы расположить все программы и данные исходного модуля в определенном системным конфигуратором сегменте памяти, вы должны повторить параметр /SEG в директиве .MODULE и всех директивах .VAR внутри модуля. Директива .SECTION используются для исключения многократного повторения параметров /SEG.

Ниже приводится пример, в котором модуль располагают в памяти данных в сегмент с именем Audio_Samples

Пример: Пример описания секции с именем Audio_Samples

.SECTION/DM Audio_Samples;

.MODULE/RAM Sample_Input;

.VAR/DM/RAM/CIRC sample_buffer[I5]

.VAR/DM/RAM other_buffer[5];

.VAR/DM/RAM another_buffer[5];

.VAR/DM/RAM variablel;

программа SAMPLE_INPUT...

.ENDMOD;

Программа для подпрограммы SAMPLE_INPUT будет размещения в памяти программ.

3.6.7 Глобальные структуры данных

Директива .GLOBAL позволяет переменным, буферам и портам быть доступными извне модуля, в котором они объявлены. Для доступа к одной из этих структур из других модулей вы должны объявить ее с директивой .GLOBAL Директива .GLOBAL имеет формат:

.GLOBAL внутренний символ,…;

Пример: Объявление глобального буфера coeffs

.VAR/PM/RAM coeffs[10];

.GLOBAL coeffs; {делает буфер видимым снаружи модуля}

С тех пор как символ сделан глобальным, другие модули могут обращаться к нему, идентифицируя символ как внешний.

3.6.8 Глобальные программные метки

Директива .ENTRY позволяет обращаться к программным меткам в других модулях. Это позволяет использовать метку для вызова подпрограммы или межмодульных переходов.

Директива .ENTRY имеет формат:

.ENTRY программная_метка,...;

Пример: Объявление глобальной программой метки

.ENTRY fir_start; {делает метку видимой снаружи модуля}

С тех пор,как метка объявлена директивой .ENTRY другие модули могут обращаться к ней, идентифицируя метку как внешнюю.

3.6.9 Внешние символы

Директива .EXTERNAL позволяет программному модулю обращаться к глобальным структурам данных (переменным, буферам и портам)и программным меткам, объявленным в других модулях.

Символ должен быть определен до этого с помощью директив .GLOBAL или .ENTRY в тех модулях, где он впервые объявлен. Другие модули должны использовать директиву .EXTERN'AL для открытия доступа к внешним символам. Директива имеет формат:

.EXTERNAL внешний символ,...;

Пример: Объявление внешнего символа

.EXTERNAL flr_start; {метка в другом модуле}

3.7 Редактор связей

Редактор связей (компоновщик) ADSP-21xx генерирует исполняемую программу путем связывания отдельно ассемблированных модулей. Основным выходом редактора связей является файл отображения памяти с расширением .DXE. Этот файл загружают в программу моделирования для отладки. После того, как программа полностью отлажена, применяют программу разбиения памяти для программатора ППЗУ. Как было описано в предыдущей главе, ассемблер обрабатывает каждого модуль исходного кода и создает объектный файл .OBJ ,файл кода .CDE и файл инициализации .INT . Объектный файл содержит информацию по размещению в памяти и символьную информацию, в то время как файл кода содержит коды операций ADSP-21xx с помеченными неразрешенными символами. Файл инициализации содержит информацию, относящуюся к переменным данным и буферам. Инициализация данных должна обеспечиваться файлом данных, указанным с помощью директивы ассемблера .INIT. Редактор связей читает данные из этого файла и объединяет их в файл .DXE. Изменения в инициализируемых данных потребует повторного вызова редактора связей. Редактор связей сканирует каждый ассемблированный модуль и разрешает обращения между модулями к глобальным и внешним символам. Он назначает (раздает) адреса перемещаемых программ и фрагментов данных. Редактор связей читает также файл описания архитектуры .I.DF для создания

карты системной памяти и размещения в ней программы и данных. Идентификация архитектурного файла для редактора связей происходит включением ключа -a в строке вызова программы. Редактор связей может создавать три различных файла. Файл отображения памяти (memory image file).DXE создается всегда - это исполняемая программа, которая содержит разные коды операций и данные, которые должны быть расположены в памяти.

Вспомогательный файл распределения (memory listing file).MAP суммирует информацию, относящуюся к созданной программе. Вспомогательный файл таблицы символов (symbol table file).SYM перечисляет все символы, встреченные редактором связей и их абсолютные адреса. Этот файл также используется программами моделирования (симуляторами) ADSP-21xx и эмуляторами

Инициализирующие файлы данных .DAT не обозначены запуске редактора связей явно поскольку они перечислены директивой .INIT в файле исходного кода. Файлы данных объединяют с помощью редактора связей. Если в файлах с данными сделаны изменения, необходимо просто запустить редактор связей заново. (**Примечание:** с тех пор как директива ассемблера .VAR используется для объявления, как однословных переменных, так и многословных буферов данных, термин «буфер данных «включает в себя как переменные, так и буфера.).

3.7.1 Запуск редактора связей

Редактор связей вызывают, перечисляя файлы, которые должны быть скомпонованы:

LD21 файл1..] [-ключ...]

Пример: Запуск редактора связей для компоновки файлов mine.s subl.s subl2.s и файла описания архитектуры archfile.ldf

LD21 mine subl subl2 -a archfile

Каждый входной файл должен содержать только один модуль. Если файлы отсутствуют в текущем каталоге вашей операционной системы, с каждым именем файла должен быть путь к нему. Имена файлов должны идентифицировать ассемблерные выходные файлы без расширений (т.е. .CDE .OBJ .INT).Выходные файлы редактора связей получают по умолчанию имя 210X. Вы можете переименовать эти файлы,

используя ключ -e. Редактор связей может быть также вызван с ключом -i который называет отдельный файл, содержащий список файлов для компоновки:

LD2i -i файл все [-ключ ...]

В этом случае редактор связей читает дополнительный файл *файл_все*, который является простым текстовым файлом, содержащим в каждой строке одно имя файла.

Другие ключи управляют различными операциями редактора связей. Они могут быть введены как в верхнем, так и в нижнем регистре. Несколько ключей должны быть отделены друг от друга хотя бы одним пробелом.

Редактор связей размещает память для модулей в соответствии с порядком, в котором они были перечислены. Для модулей, которые содержат объявления циклических буферов данных, изменения в порядке входных файлов могут определить, сможет ли программа быть успешно скомпонована в доступном пространстве памяти. Поэтому: модули, содержащие циклические буферы различных размеров должны быть перечислены для редактора связей в порядке понижения размеров буферов. Это позволит редактору связей размещать вначале буфера большего размера, которые имеют большие ограничения на допустимые базовые адреса. Если вы забыли синтаксис для вызова редактора связей, наберите в командной строке: LD21 -help

Эта команда покажет синтаксис вызова программы и покажет список доступных ключей. Если вы ассемблировали исходную программу, сгенерированную С компилятором, редактор связей должен быть вызван с ключом -c.

Ключи редактора связей перечислены в таблице 3.3.

Таблица 3.3 Ключи редактора связей

Таблица 3.3 Ключи редактора связей

Ключ	Воздействие
-a имя_файла	указание файла описания архитектуры *имя_файла* .LDF
- c	создание стека для компилированных С программ (DM)
-dir каталоги	указание директорий для поиска файлов библиотек -dryrun быстрый запуск для теста на ошибки (не создает . EXE файл)
- e имя_файла	присваивает выходным файлам имя *имя_файла*
-g	создает файл таблицы символов .SYM
-i файл_все	указание командного списочного файла -lib компоновка с С библиотекой рабочих программ (используется только с «-c»)
- p	размещает копию библиотечной подпрограммы на загрузочных страницах -pmstack перемещение С стека в память программ (РМ) (используется только с «-c»)
-rom	используется ПЗУ версия С библиотеки программ (используется только с «-c»)
-s размер	создает динамическую память («кучу »)С (используется только с «-c»)
-user имя_файла	поиск библиотечного файла, созданного утилитой построителя библиотеки LIB21 -x создает файл распределения памяти .MAP

3.7.2. Как работает редактор связей

Данный раздел дает представление о том, как работает редактор связей, когда он обрабатывает программные модули. При лучшем понимании принципов работы редактора связей, вы сможете структурировать свои программы для более эффективного использования памяти.

Работа редактора связей состоит в комбинировании ассемблированных модулей и инициализации данных в исполняемой программе, называемой файлом отображения памяти .DXE .Две основные задачи: размещение памяти и разрешение символов.

3.7.3 Распределение памяти

Редактор связей читает каждый модуль кода и объявления переменных данных/буферов для определения типа памяти, в которой они должны быть расположены - ОЗУ или ПЗУ, память программ или память данных, имя сегмента и т.д.

Редактор связей читает также содержимое файла-описания архитектуры .ACH, чтобы определить, какие пространства памяти доступны и какие у них характеристики. На основе этой информации происходит размещение каждого модуля, буфера и переменной в соответствующем типе памяти. Если объект имеет абсолютный адрес, указанный параметром ABS, его называют неперемещаемым. Если абсолютный адрес не присвоен, объект считается перемещаемым. Редактор связей присваивает адрес каждому перемещаемому объекту. Если вы захотите, чтобы в памяти процессора (внешней или внутренней) во время выполнения отдельной загрузочной страницы существовала незагружаемая подпрограмма или буфер данных, вы должны использовать параметр BOOT чтобы связать его с этой страницей. При совместном использовании с параметром STATIC редактор связей зарезервирует пространство для объекта во время выполнения страницы.

Выходной файл карты распределения памяти программы (.MAP) показывает ,как расположение вашей программы в памяти начальной загрузки, так и соответствующее отображение кода в памяти во время выполнения (после выполнения начальной загрузки).

Этот файл помогает понять перенос из памяти начальной загрузки в рабочую память, Вы не можете указать область, где модуль будет размещен в памяти начальной загрузки – редактор связей самостоятельно реализует эту функцию, составляя эффективно упакованные загрузочные страницы.

3.7.4 Разрешение символов

Чтобы разрешить программные символы, редактор связей должен поставить в соответствие каждому символу определенный адрес в пространстве памяти. Имена программных меток, переменных/буферов являются символами, которые определены в исходном коде. Ассемблер просто пропускает их редактору связей, задача которого определить адрес каждого из символов, после размещения в памяти всех модулей. Обращение к символу может происходить только внутри модуля, где он определен, пока он не определен директивами ENTRY или GLOBAL. Эти директивы ассемблера расширяют диапазон обращения к символу. Другие модули должны объявить этот символ директивой EXTERNAL перед обращением к нему. Для каждой ссылки EXTERNAL редактор связей ищет в других модулях определения ENTRY или GLOBAL. При нахождении нескольких совпадений выводится сообщение об ошибке. Если поиск не выявил определений символа, редактор связей включает поиск библиотечного файла в соответствии с последовательностью, описанной в разделе «Последовательность поиска библиотеки». Этот поиск включает просмотр переменной среды окружения ADIL и аргументов ключей -user -dir

Если подключаемый символ не найден с помощью поиска в библиотеке, редактор связей выводит сообщение, об ошибке.

После того как распределение памяти завершено, и все внешние ссылки разрешены, редактор связей присваивает каждому символу значение адреса. Редактор связей вырабатывает файл таблицы символов .SYM если задан ключ -g ,который содержит список всех встреченных программных символов и их адресов. Этот файл показывает, какие символы могут быть доступны каждому модулю.

3.7.5 Построение единой библиотеки для быстрого доступа

Программное обеспечение разработчика ADSP-21XX включает построитель библиотеки - утилиту LIB21, которая позволяет вам записать несколько библиотечных подпрограмм и упаковать их в один файл для быстрого доступа. После создания библиотечного файла необходимо вызвать редактор связей с ключом -user *имя_библиотеки*, который позволяет находить, выделять и компоновать необходимые подпрограммы из файла *имя_библиотеки*.

Использование утилиты LIB21 и ключа -user подразумевает то же самое, что и применение ключа -dir или переменной среды окружения ADIL

Преимущество утилиты LIB21 состоит в том, что редактор связей работает быстрее. Утилита LIB21 вызывается одним из двух способов:

LIB21 имя_библиотеки файл1 ...] [-V версия]

или

LIB21 имя_библиотеки -i списочный_файл [-V версия]

Выходной файл *имя_библиотеки.A* объединяет в себе отдельно ассемблированные модули, перечисленные в командной строке (файл1 , файл 2 и т.д.).Эти модули должны быть указаны без расширений. Ключ -i имеет такое же действие, как и при вызове редактора связей. Файл *списочный_файл* содержит список с одним именем файла на строке. Ключ –V позволяет вставить номер версии для подпрограмм в библиотечном модуле; аргумент *версия* является символьной строкой. Строка версии не влияет на исполнение программы.

Ниже приведен пример создания библиотечного файла быстрого доступа:

Пример: Запуск редактора связей для компоновки библиотеки быстрого доступа filter.a из файлов taps.s coeffs.s start_input.s

LIB21 filter taps coeffs start input -v V1.0

Утилита LIB21 создаст файл FILTER.A объединяющий три входных модуля. Символьная строка версии «V1.0 вставляется в файл. Редактор связей можно вызывать строкой:

Пример: Запуск редактора связей для компоновки программы, с использованием библиотеки быстрого доступа filter.a, из файлов main.s sum.s graph.s

LD21 main sum graph -user filter

Что вызовет компоновку модулей main sum и graph. Полагается, что из файла FILTER.A будет использована одна или более библиотечных подпрограмм, и редактор связей выделит требуемые функции и включит их в процесс компоновки.

3.8 Описание файла архитектуры

Описание файла архитектуры начинается с директивы ARCHITECTURE (имя_архитектуры). Затем следует описание подключаемых модулей ($OBJECTS = модуль1, модуль2 ...). После этого описывается распределение памяти, например:

Пример: Пример описания распределения памяти в фате описания архитектуры.

```
/*
Секция      Память    Начальный адрес   Конечный адрес   Ширина, бит
seg_inttab  PM        0x00000           0x0002F          24
seg_code    PM        0x00030           0x018FF          24
seg_data2   PM        0x02000           0x02600          24
seg_data1   DM        0x00000           0x02fff          16
seg_code    DM        0x03000           0x037ff          16
seg_code    DM        0x03800           0x03C00          16
*/
        MEMORY
        {
        seg_inttab { TYPE(PM RAM) START(0x00000) END(0x0002F) WIDTH(24) }
        seg_code  { TYPE(PM RAM) START(0x00030) END(0x018FF) WIDTH(24) }
        seg_data2   { TYPE(PM RAM) START(0x02000) END(0x02600) WIDTH(24) }

        seg_data1  { TYPE(DM RAM) START(0x00000) END(0x02fff) WIDTH(16) }
        seg_heap   { TYPE(DM RAM) START(0x03000) END(0x037ff) WIDTH(16) }
        seg_stack  { TYPE(DM RAM) START(0x03800) END(0x03C00) WIDTH(16) }
        }
```

Директива SECTIONS объявляет используемые сегменты. При создании программ многопроцессорных систем, создается один .LDF файл. Указание на то, какому процессору принадлежит сегмент, осуществляется директивой PROCESSOR.

Задание: В системе Visual DSP создайте проект, введите последовательно вышеприведенные примеры, откомпилируйте проект и запустите его в отладчике.

Пример: Пример файла описания архитектуры

```
ARCHITECTURE(ADSP-2189)
#ifdef DEBUG
SEARCH_DIR( $ADI_DSP\218x\lib;.\Debug )
#else
SEARCH_DIR( $ADI_DSP\218x\lib;.\Release )
#endif
$OBJECTS = 218x_hdr.doj, 218x_exit.doj, $COMMAND_LINE_OBJECTS , libio.dlb , libc.dlb ;
// 2189 содержит 32K (24-bit) слов в ОЗУ памяти программ и 48K (16-bit)слов в ОЗУ памяти
// данных
MEMORY
{
  seg_inttab { TYPE(PM RAM) START(0x00000) END(0x0002F) WIDTH(24) }
  seg_code  { TYPE(PM RAM) START(0x00030) END(0x018FF) WIDTH(24) }
        seg_data2  { TYPE(PM RAM) START(0x02000) END(0x02600) WIDTH(24) }
        seg_data1  { TYPE(DM RAM) START(0x00000) END(0x02fff) WIDTH(16) }
        seg_heap  { TYPE(DM RAM) START(0x03000) END(0x037ff) WIDTH(16) }
  seg_stack  { TYPE(DM RAM) START(0x03800) END(0x03C00) WIDTH(16) }
}
PROCESSOR p0
{
  LINK_AGAINST( $COMMAND_LINE_LINK_AGAINST)
  OUTPUT( $COMMAND_LINE_OUTPUT_FILE )
  SECTIONS
  {
    sec_inttab
    {
      INPUT_SECTIONS( $OBJECTS(interrupts) )
    } >seg_inttab
    sec_code
    {
      INPUT_SECTIONS( $OBJECTS(program) )
    } >seg_code
    sec_data1
    {
      INPUT_SECTIONS( $OBJECTS(data1) )
    } >seg_data1
    sec_data2
    {
      INPUT_SECTIONS( $OBJECTS(data2) )
    } >seg_data2
    sec_stack
    {
      ldf_stack_limit = .;
      ldf_stack_base = . + MEMORY_SIZEOF(seg_stack) - 1;
    } >seg_stack
sec_heap
    {
      .heap = .;
          .heap_size = MEMORY_SIZEOF(seg_heap);
.heap_end = . + MEMORY_SIZEOF(seg_heap) - 1;
    } >seg_heap
  }
}
```

1. Для чего необходим ассемблер?

2. Какие функции выполняет редактор связей?

3.Объяснить состав и назначение секций файла конфигурации?

4. Пояснить, что выполняет и зачем нужна каждая секция в примере?

ГЛАВА 4. Эмулятор EZ-KIT ADSP-2189M

Эмулятор EZ-KIT представляет собой аппаратно-программный комплекс, предназначенный для отладки программного обеспечения. Структурная схема EZ-KIT приведена на рис.4.1, (питание 7.5В (+/- 5%) при 4А, температурный диапазон 0-70 град. С).

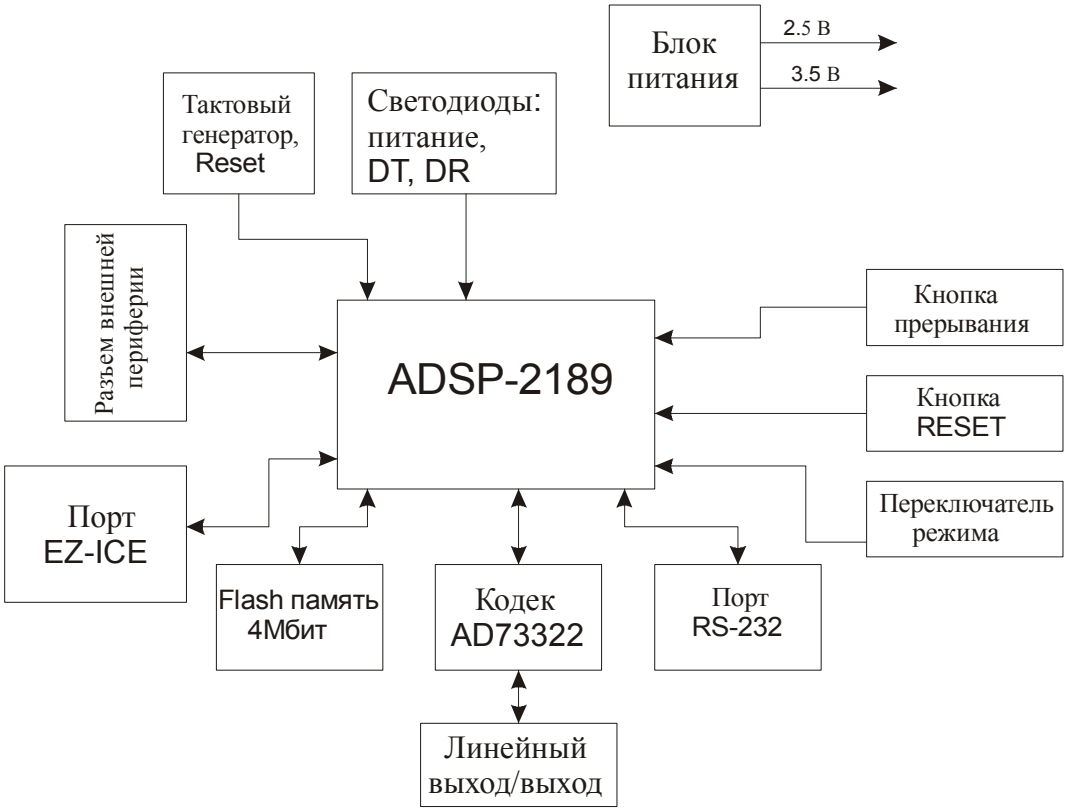

Рис.4.1 Структурная схема платы EZ-KIT

Общий вид платы EZ-KIT приведен на рис. 4.2.

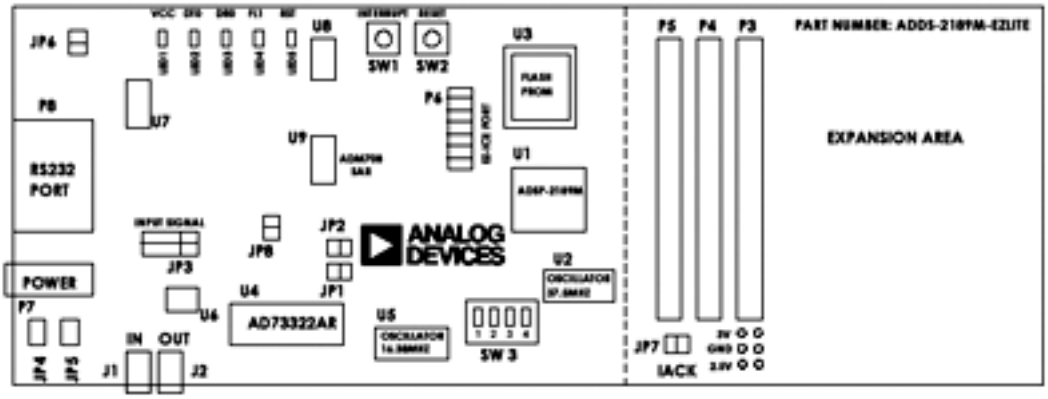

Рис.4.2 Общий вид платы EZ-KIT

На плате установлены разъем питания, разъем DB9 для связи с персональным компьютером по RS-232, а так же линейные аналоговые вход и выход. Конфигурирование платы осуществляется при помощи перемычек (jumper).

JP1 - При установке соединяет вход и выход кодека AD73322AR (по умолчанию выключен)

JP2 - Если установлен - кодак отключен (по умолчанию выключен, кодек включен)

JP3 - Включает/выключает входное и выходное усиление

JP4&JP5 - При установке изменяют коэффициент усиления линейного входа с 47 до 400

JP6 - Установка запрещает работу последовательного порта.

JP8 - Переключает рабочее напряжение I/O с 2.5В до 3В

На плате имеются так же две кнопки - SW1 кнопка генерирует внешнее прерывание ADSP- 2189, SW3 - генерирует сигнал Reset.

Для включения эмулятора необходимо подключить его к последовательному порту персонального компьютера (COM), включить блок питания и нажать кнопку Reset.

1. Подключите эмулятор к персональному компьютеру.
2. В системе Visual DSP создайте проект, наберите любой из вышеописанных примеров, откомпилируйте проект и запустите его в отладчике. В пошаговом режиме посмотрите, как изменяется состояние процессора.

ГЛАВА 5. Основные сведения об архитектуре семейства 21xx

Семейство ADSP-21xx представляет собой ряд однокристальных микрокомпьютеров, которые объединяет общая архитектура, оптимизированная для цифровой обработки сигналов и других операций в области высокоскоростной обработки данных. Сигнальный процессор отличается от обычного микропроцессора следующими характерными принципами:

- использование гарвардской архитектуры
- сокращение длительности командного цикла
- применение конвейеризации
- применение аппаратного умножителя
- включение в систему команд специальных команд ЦОС

Гарвардская архитектура подразумевает хранение программы и данных в различных ЗУ. Соответственно на кристалле есть две адресные шины две шины данных.

Конвейерный режим используется для сокращения длительности командного цикла. Обычно применяется двух- или трехкаскадный конвейер, что позволяет на разных стадиях одновременно выполнять две или три инструкции.

Аппаратный умножитель применяется для сокращения времени выполнения одной из основных операций ЦОС - операции умножения.

Специализированные команды ЦОС. Система команд сигнальных процессоров оптимизирована для выполнения базовых задач ЦОС. Обычно добавляются операции: умножение с накоплением, битовые операции, инверсия бит адреса, кольцевые буфера и многое другое.

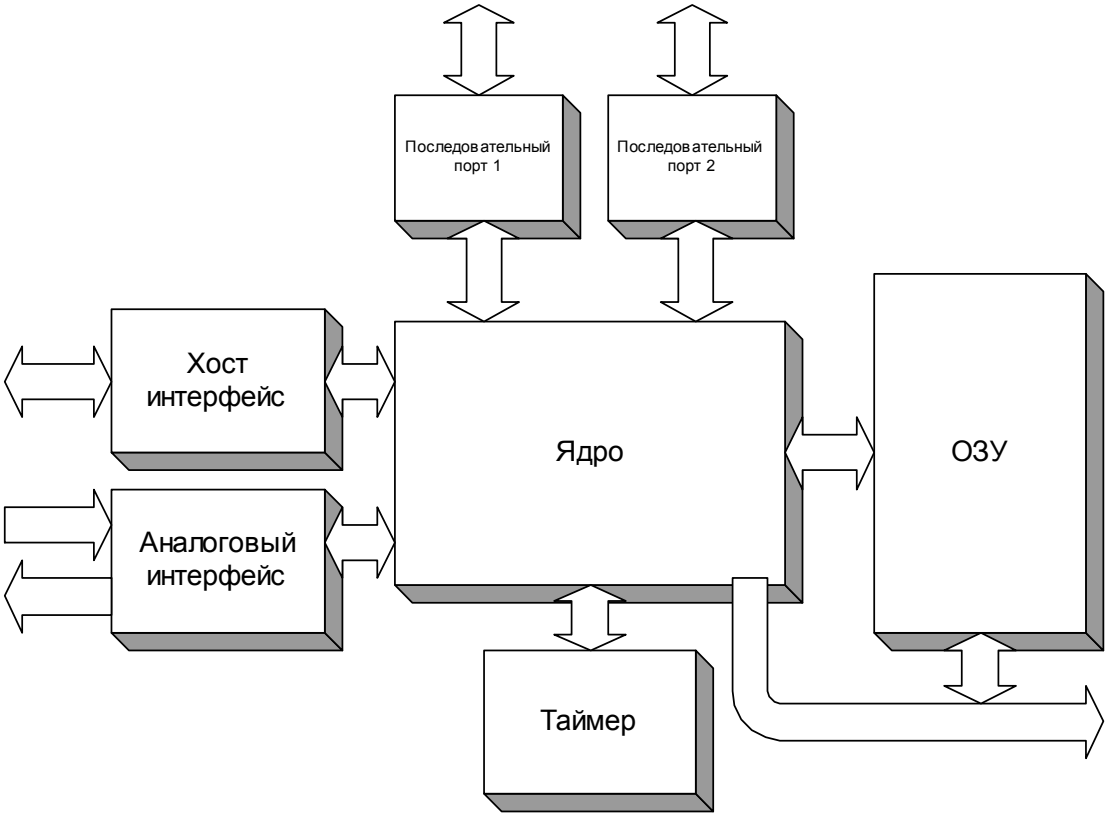

Рис. 5.1 Структурная схема ADSP-21xx

Процессоры семейства 218x имеют в своем составе следующие функциональные устройства:

- арифметико-логическое устройство ALU
- множитель накопитель MAC
- устройство барабанного сдвига SHIFTER
- генераторы адресов данных DAG и генератор адресов инструкций PS
- последовательные порты SPORTs
- таймер
- порт интерфейса с хост-процессором

5.1 Вычислительные устройства

К вычислительным устройствам относятся: арифметико-логическое устройство ALU, множитель накопитель MAC, устройство барабанного сдвига SHIFTER.

АЛУ выполняет стандартный набор арифметических и логических операций, а так же примитивы деления. MAC выполняет за один такт операцию умножения и сложения/вычитания, операции MAC имеют вид - MR=MX*MY+/-MR. Данная операция получила название умножение с накоплением и широко используется при создании

различных фильтров. Вычислительные устройства организованны параллельно, что позволяет любому устройству использовать результаты, полученные на предыдущем цикле. Для этой цели служит шина внутренних результатов R.

5.1.1 Арифметико-логическое устройство

АЛУ во всех процессорах семейства 21xx является 16 разрядным с фиксированной точкой. Знаковые числа представляются в дополнительном коде. Остальные представляются в виде беззнакового числа или строки бит.

Операциями со строками бит являются логические операции NOT, AND, OR, XOR. При выполнении этих операций АЛУ не заботиться о знаке числа и положении десятичной точки.

Числа без знака могут принимать только положительные значения и поэтому имеют вдвое больший диапазон, чем числа со знаком.

Знаковые числа представляются в процессоре в формате 1.15, то есть левый бит числа обозначает его знак, остальные 15 бит содержат значение.

Операции АЛУ изменяют следующие флаги в регистре состояния процессора:

AN - флаг отрицательного числа, устанавливается в 1, если в результате операции бит знака имеет значение 1

AV- флаг переполнения, устанавливается в 1 при непредсказуемом смене знака, например если в случае сложения двух положительных чисел - результат отрицателен.

AC - флаг переноса, устанавливается в 1, если длина результата операции больше длины регистра

AZ - флаг нуля, устанавливается в 1 результат операции нулевой.

AO - флаг состояния частного]

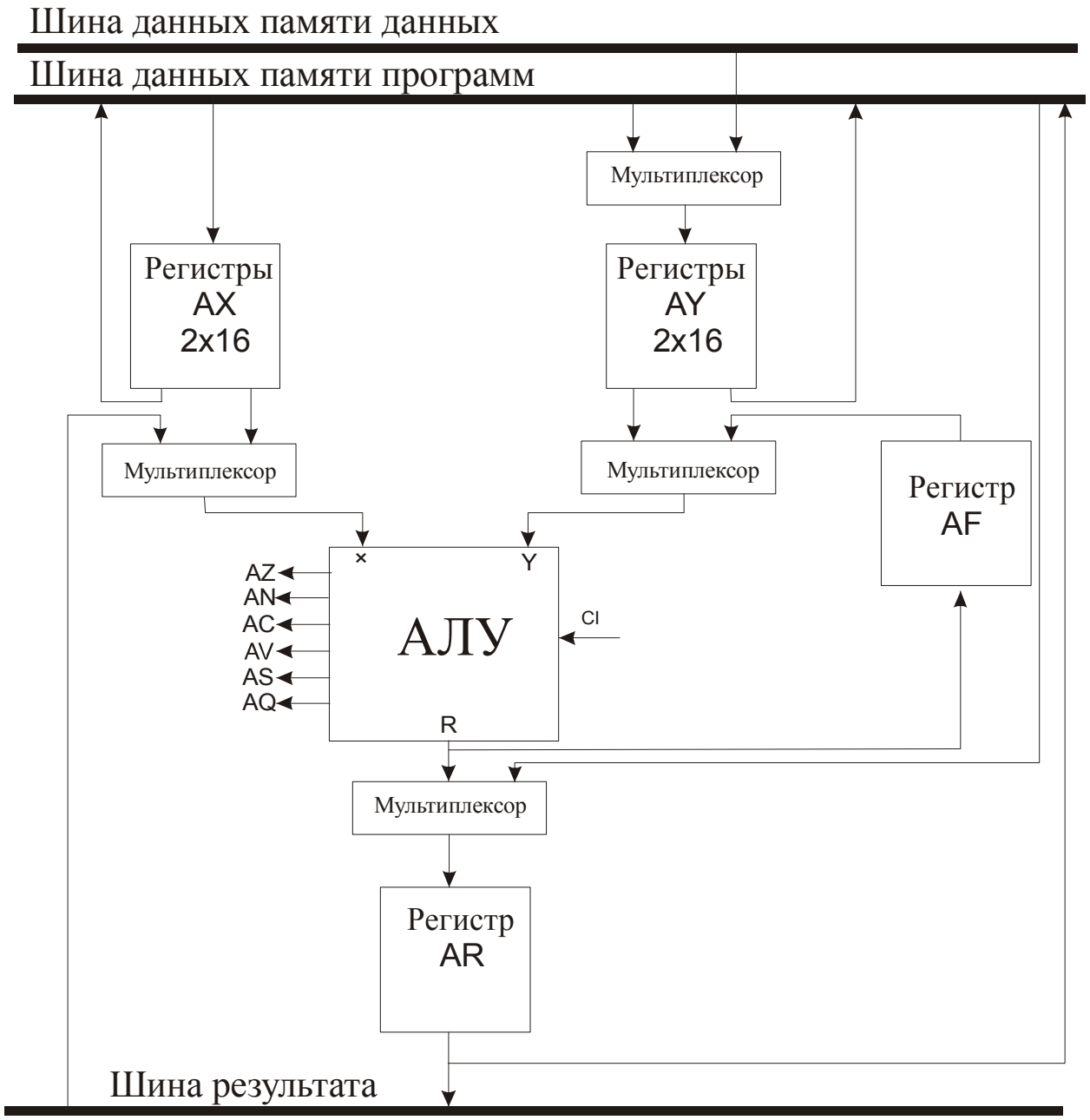

Рис.5.2 Блок-схема АЛУ

АЛУ имеет два блока 16 разрядных регистров операндов AX(AX0,AX1) и AY (AY0,AY1) в зависимости от режима установленного битом 0 в регистре состояния процессора (MSTAT) одна пара регистров является основной, например AXO и AY0, другая - теневой. Это очень удобно для создания программ обработки прерывания выполняющих арифметические операции, так как нет необходимости сохранять регистры операндов в стеке и по - окончанию обработки их оттуда извлекать, достаточно просто переключить блок. Регистром результата является регистр AR. Данные в порт Y могут загружаться по линии обратной связи через регистр AF.

Таблица 5.1 Источники ввода данных в регистры ввода/вывода АЛУ

Источники ввода данных в порт X	Источники ввода данных в порт Y	Направление вывода данных через порт R
AXO, AX1	AY0, AY1	AR
AR	AR	AF
MRO, MR1, MR2		
SRO, SR1		

Стандартные функции АЛУ:

R=X+Y	Сложить операнды X и Y
R=X+Y+CI	Сложить операнды X и Y и бит переноса
R=X-Y	Вычесть из Y из X
R=X-Y+CI-1	Вычесть из Y из X с заемом
R=Y-X	Вычесть из X из Y
R=Y-X+CI-1	Вычесть из X из Y с заемом
R=-X	Перевести X в дополнительный код
R=-Y	Перевести Y в дополнительный код
R=Y+1	Инкремент Y
R=Y-1	Декремент Y
R=PASS X	Пропустить X
R=PASS Y	Пропустить Y
R=0 (PASS 0)	Обнуление результата
R=ABS X	Модуль X
R=X AND Y	Операция логического И X&Y
R=X OR Y	Операция логического ИЛИ $X \vee Y$
R=X XOR Y	Операция ИСКЛЮЧАЕЩЕГО ИЛИ
R=NOT X	Инверсия X
R=NOT Y	Инверсия Y

Вышеперечисленные функции следует использовать с учетом таблицы 5.1. Например, рассмотрим операцию вычитания R=X-Y, согласно таблице на месте X могут стоять следующие регистры: AXO, AX1, AR, MRO, MR1, MR2, SRO, SR1. На месте Y - AYO, AY1, AR, на месте R - AR, AF. Таким образом, АЛУ может произвести следующую операцию AF=MR0-AY0.

С регистра состояния процессора MSTAT установкой бита 3 устанавливается режим насыщения, то есть в регистр AR загружается максимальное положительное или максимальное отрицательное число при возникновении переполнения. Зависит это от состояния флагов.

Таблица 5.2 Режим насыщения

Переполнение (AV)	Перенос (AC)	Содержимое AR
0	0	В соответствии с АЛУ
0	1	В соответствии с АЛУ
1	0	0111111111111111
1	1	1000000000000000

Режим защелки устанавливается включением бита 2 регистра состояния процессора MSTAT. Суть режима в следующем - после возникновения переполнения бит AV остается установленным до тех пор, пока не будет сброшен прямой записью нуля.

Пример: Несколько примеров использования АЛУ.

/ Сложение регистра AXO с AY0 */*

AR =AX0 + AY0;

/ Вычитание из регистра MR регистра AY0 */*

AR = MR + AY0;

/ Инверсия SR0 */*

AR = NOT SRO;

1. Разработайте программы сложения и вычитания двух чисел.
2. Разработайте программу создающую переполнение в регистре AR, проверьте содержимое регистра AR в зависимости от установки бита 3 в регистре MSTAT и флагов AV и AC.

5.1.2 Умножитель - накопитель

В умножителе есть два входных порта X и Y разрядностью 16 бит и выходной порт результата P разрядностью 32 бита. Результат разрядностью 32 бита передается в сумматор разрядностью 40 бит, который прибавляет/вычитает новый результат из содержимого регистра умножителя MR. Регистр имеет размер 20 бит и аппаратно состоит из трех регистров MRO, MR1, MR2 размером 16, 16 и 8 бит соответственно.

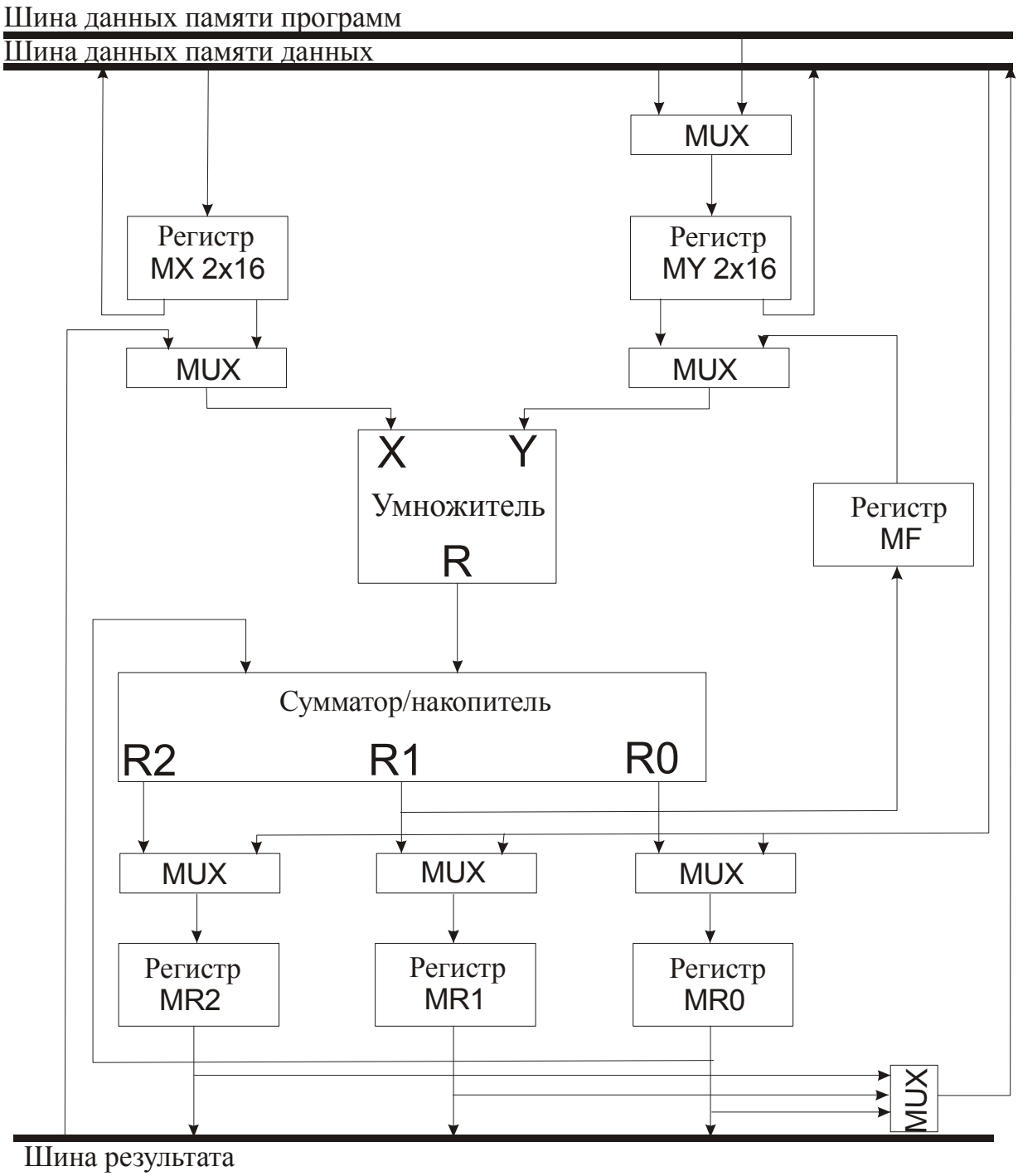

Рис. 5.3 Структурная схема умножителя-накопителя

Порт ввода X может принимать данные как из регистров MX, так и из любого регистра подключенного к шине R. Порт ввода Y может принимать данные либо из регистра MY, либо из регистра MF.

Выходное значение может поступать либо в MR, либо в MF. MF является регистром обратной связи и содержит биты 16-31 регистра MR. Стоит заметить, что запись и считывание одного регистра в умножителе-накопителе производиться за один цикл. В умножителе имеется двойной ряд регистров MR, MF, MX, MY. Выбор ряда

осуществляется битом 0 регистра состояния процессора MSTAT, если значение бита 0 - выбран основной ряд, 1 - альтернативный.

Таблица 5.3 Источники ввода данных в регистры ввода/вывода умножителя

Источники ввода данных в порт X	Источники ввода данных в порт Y	Направление вывода данных через порт R
MXO, MXI	MYO, MY1	MR
AR	MF	MF
MRO, MR1, MR2		
SRO, SR1		

Стандартные функции умножителя:

X*Y Умножение операндов X и Y

MR+X*Y Умножение операндов X и Y и сложение результата с регистром MR

MR-X*Y Умножение операндов X и Y и вычитание полученного результата из регистра MR

O Обнуление результата

Процессоры семейства ADSP-2Ixx имеют два режима умножения - умножение дробных чисел в формате 1.15 и умножение целых чисел в формате 16.0.

В режиме умножения дробных чисел 32-х разрядный результат регулируется по формату, то есть добавляются биты знака и перед добавлением к регистру MR производиться сдвиг влево на один бит. В режиме умножения целых чисел никаких сдвигов, перед добавлением к регистру MR, не производится.

Режим умножения задается битом 4 регистра состояния процессора MSTAT. Если указанный бит имеет значение 1 - выбран режим умножения целых чисел, иначе - режим умножения дробных чисел.

В умножителе, так же как и в АЛУ существует режим насыщения. По переполнению в арифметическом регистре состояния устанавливается флаг MV.

Операция насыщения зависит от состояния бита MV и старшего бита регистра MR2. В таблице 5.4 представлена операция насыщения.

Таблица 5.4 Режим насыщения умножителя

Переполнение (MV)	Старший бит MR	Содержимое MR
0	0 или 1	без изменений
1 1	0 1	00000000 0111111111111111 1111111111111111 11111111 1000000000000000 0000000000000000

В сумматоре умножителя-накопителя имеется возможность округления 40 результата на границе между 16 и 15 битами. Округление может быть задано как часть кода команды. Округленное выходное значение направляется либо в регистр MR, либо в регистр MF.

В сумматоре используется округление без смещения. Условный метод округления со смещением заключается в добавлении 1 к 15 биту в цепочке сумматора. Округление по данному методу приводит всегда к положительному смещению, так как срединное значение (MR=0x8000) всегда округляется в сторону увеличения. Это смещение устраняется принудительной установкой бита 16 при обнаружении такого половинного значения. В результате при срединном значении нечетные числа округляются до четных в сторону увеличения, а четные до нечетных в сторону уменьшения. При большом размере выборки суммарная ошибка стремиться к нулю.

В процессорах семейства ADSP-218x имеется так же режим округления со смещением. Устанавливается он битом BIASRND (бит 12) в регистре управления автобуферизацией порта SPORTO. Такой режим используется при реализации алгоритмов, в которых используется округление со смещением, например в алгоритмах сжатия речи в системах GSM.

В качестве примера использования умножителя-накопителя рассмотрим КИХ-фильтр.

Пример: Пример использования умножителя накопителя

```
/*
КИХ - фильтр
I0 – буфер, хранящий прошлые отсчеты
L0 – размер буфера I0, равный порядку фильтра
I4 – буфер, хранящий коэффициенты фильтра
L4 –количество коэффициентов
 */
fir :
MR=0 , MX0 =DM(I0 ,M1), MY0 =PM(I4 ,M5);
DO sop UNTIL CE ;
sop :
MR=MR+MX0*MY0(SS), MX0=DM(I0,M1), MY0=PM(I4,M5);
MR=MR+MX0 *MY0 (RND);
IF MV SAT MR;
RTS ;
```

1. Разработайте и отладьте программу, вычисляющую скалярное произведение любых двух векторов

2. Исследуйте режимы округления, устанавливая различные флаги операции умножения с накоплением.

5.1.3 Устройство циклического сдвига

Устройство сдвига обеспечивает полный набор функций сдвига для 16 разрядных входных значений, в результате на выходе получается 32 разрядное значение. К операциям устройства циклического сдвига относятся операции логического и арифметического сдвига, операция нормализации, операции нахождения порядка числа и блока чисел.

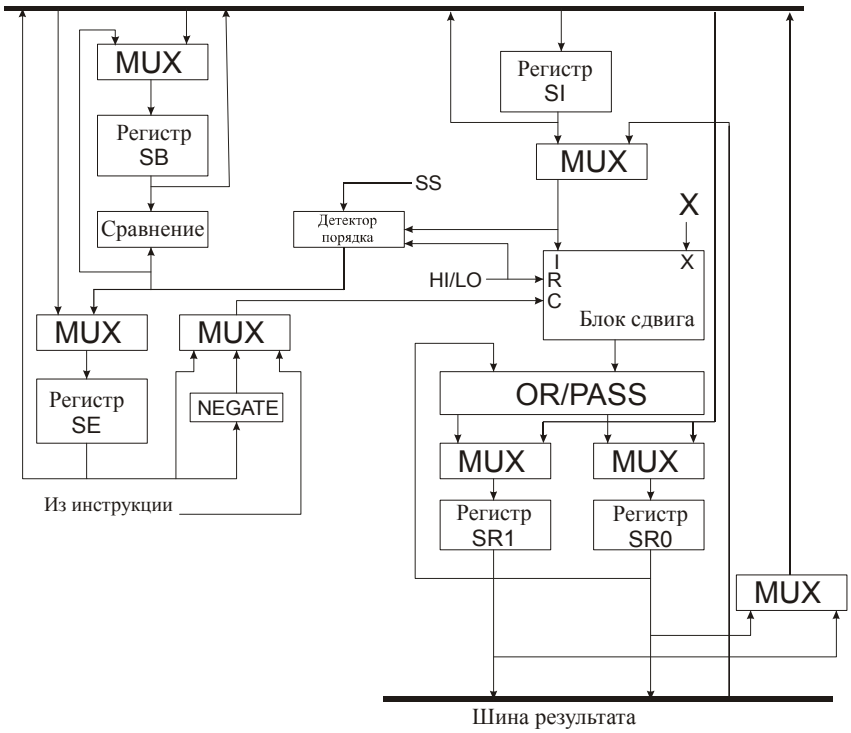

Рис. 5.4 Структурная схема устройства циклического сдвига

В устройстве сдвига можно выделить следующие элементы: массив устройства сдвига, логическое устройство ИЛИ/ПЕРЕДАЧА, определитель порядка и устройство сравнения порядка.

Массив устройства сдвига представляет собой матрицу 16 на 32 бита. В него вводится 16 разрядное значение, которое может быть помешено в любое место 32 разрядной области результата, от крайней правой до крайней левой за один цикл. Таким образом, возможно 49 расположений в 32-разрядной области результата. Расположение в массиве 16 входных бит определяется сигналом (C) и опорным сигналом HI/LO. Массив устройства сдвига с обслуживающими его логическими устройства окружен набором регистров. Регистр ввода SI предназначен для ввода данных в массив устройства сдвига и определитель порядка. Считывание из регистра SI и запись в него осуществляется через шину DMD. Ввод данных в массив устройства сдвига и определитель порядка так же осуществляется из регистров AR, SR или MR по R-шине. Результат имеет разрядность 32 бита и помещается в пару регистров SRI, SR0. Загрузка в регистры SRI, SR0 производиться шины DMD, а результат помещается на шину DMD или на шину R. Регистр SR имеет обратную связь с устройством ИЛИ/ПЕРЕДАЧА для обеспечения сдвига двойной точности.

Регистр SE («порядок устройства сдвига») имеет размер 8 бит и содержит значение порядка для проведения операций нормализации-денормализации. Загружается в SE 8 младших битов с шины DMD.

Регистр SB («блок устройства сдвига») используется для операций с блочно плавающей точкой, во время которых в нем содержится значение блочного порядка, то есть значение, на которое надо произвести сдвиг самого большого числа в блоке. 5-разрядный регистр SB загружается с младших 5 бит шины DMD.

Когда производится считывание регистров SE и SB, старшие биты заполняются значением бита знака до размера 16 бит. Считывание осуществляется через шину DMD. Запись и считывание регистров SE, SI и SB могут производиться за один цикл. Считывание из регистров производиться в начале цикла, запись - в конце.

Как и в ALU и MAC в устройстве циклического сдвига имеется два набора регистров SE, SI, SB, SR - основной и дополнительный. Выбор набора осуществляется установкой бита 0 в регистре состояния MSTAT.

Сдвиг входного значения зависит от значений управляющего кода C и опорного сигнала HI/LO. Управляющий код имеет размер 8 бит и указывает направление и число позиций сдвига. Если C положителен - сдвиг осуществляется влево, отрицателен - вправо. Управляющий код может загружаться либо из регистра SE либо напрямую из команды.

Опорный сигнал HI/LO определяет опорную точку сдвига. Если C в состояние HI, то все сдвиги происходят относительно старших бит (регистра SR1), в состоянии LO - относительно младших бит (регистра SR0). При операциях сдвига биты справа от входного значения (при сдвиге влево) заполняются нулями, а слева (при сдвиге вправо) могут заполнятся либо значением из флага переноса AC, либо значением старшего бита входного значения, либо нулем.

Логика OR/PASS позволяет сдвинутым секциям числа повышенной точности быть правильно скомбинированными, Когда выбран PASS результат операции загружается в SR без изменения, когда выбран OR результат операции подвергается операции логического или с предыдущим значением SR.

- Устройство сдвига выполняет следующие операции:
- Арифметический сдвиг (ASHIFT)
- Логический сдвиг (LSHIFT)
- Нормализация (NORM)
- Определение экспоненты (EXP)
- Блочное изменение экспоненты (EXPADJ)
- Определение экспоненты блока

Эта функция определяет степень самого большого по масштабу числа в блоке. Она выполняется инструкцией EXPADJ. Последовательность шагов - следующая:

Загрузить -16 в SB

Регистр SB используется для записи степени всего блока. Возможные значения в нем после завершения серии операций от -15 до 0. Логика сравнения степени обновляет его содержимое, если новое значение больше текущего. Таким образом, загрузка -16 в этот регистр служит для его инициализации.

```
Обработка первого элемента массива
Array(1)=11110101 10110001
Exp=-3
-3>-16, SB=-3

Обработка второго элемента массива
Array(2)=00000001 01110110
Exp=-6
-6<-3, SB=-3 Значение остается с предыдущего такта
Обработка остальных элементов массива
```

Таким образом, операция EXPADJ просмотровая операция, в том случае если степень следующего элемента больше чем предыдущего в SB помещается новое значение степени, таким образом, после просмотра всех элементов в SB оказывается значение максимальной степени.

Немедленный сдвиг просто сдвигает операнд влево/вправо на нужное число разрядов.

Пример: Несколько примеров использования устройства циклического сдвига

```
/* Сдвиг влево на пять разрядов относительно SR1 */
SI=0xB6A3;
SR=LSHIFT SI BY -5 (HI); /* Сдвиг относительно SR1 */

/*Вход SI: 10110110 10100011
Значение сдвига: -5
                    SR1                 SR0
Выход SR: 00000101 10110101 | 00011000 00000000   */

/* Сдвиг вправо на пять разрядов относительно SR0 */
SI=0xB6A3;
SR=LSHIFT SI BY 5 (LO); /* Сдвиг относительно SR0 */

/* Вход SI: 10110110 10100011
Значение сдвига: +5
                    SR1                 SR0
Выход SR: 00000000 00010110 | 11010100 01100000 */

/* Арифметический сдвиг влево на пять разрядов относительно SR1 */
SI=0xB6A3;
SR=LSHIFT SI BY -5 (HI); /* Сдвиг относительно SR1 */
/* Вход SI: 10110110 10100011
Значение сдвига: -5
                    SR1                 SR0
Выход SR: 11111101 10110101 | 00011000 00000000 */
```

1. Реализуйте с помощью сдвига линию задержки.

5.1.4 Операция денормализации

Денормализацией называется преобразования числа с плавающей точкой в число с фиксированной точкой. Осуществляется данное преобразование за счет последовательности сдвигов. Для начала регистр SE должен содержать значение степени. Это значение может быть специально считано или быть результатом предыдущей операции. Далее производиться сдвиг.

Пример: Пример операции денормализации

```
/*Операция денормализации
Пусть SE=-3*/

SR=ASHIFT SI (HI) ;

/* старшая половина числа в SI=10110110 10100011 */
/* результат в SR=11110110 11010100 01100000 00000000*/

SR=SR OR LSHIFT SI (LO);

/* младшая половина числа в SI=01110110 01011101 */
/* результат в SR=11110110 11010100 01101110 11001011*/
```

5.1.5 Нормализация

Числа с избыточными знаковыми битами нуждаются в нормализации. Данная операция может рассматриваться как операция преобразования числа с фиксированной точкой в число с плавающей точкой.

Нормализация процесс двухстадийиый. Первый шаг определяет порядок числа, второй производит собственно сдвиг.

Пример: Пример операции нормализации

```
/* Операция нормализации */

AR=11110110 11010100

/* определяем порядок числа */
SE=EXP AR (HI)
/* SE=-3 */
SR=NORM AR (HI)
/* SR=10110110 10100000 00000000 00000000 */
```

5.2 Генераторы адресов данных и счетчик инструкций

Два выделенных генератора адресов данных DAG и многофункциональный счетчик инструкций обеспечивают эффективное использование вычислительных

устройств. Генераторы адресов данных обеспечивают генерацию адресов памяти данных, когда данные пересылаются из выходных или во входные регистры. С каждым указателем может быть связана длина для реализации кольцевых буферов. DAG 1 может генерировать адреса для памяти данных, DAG 2 - как для памяти данных, так и для памяти программ. В выходном адресе DAG 1 может меняться порядок следования битов перед выдачей на шину адреса. Это облегчает адресацию в алгоритмах БПФ.

5.2.1 Счетчик инструкций

Счетчик инструкций формирует адреса инструкций для памяти программ. Он управляет регистром инструкций, который содержит исполняемую в данный момент команду. Регистр инструкции буферизует исполнение программы. Чтобы минимизировать циклы ожидания, счетчик команд выполняет условные переходы, вызовы и возвраты из подпрограмм за один цикл. Он имеет внутренний счетчик вложенности циклов и стек циклов, что позволяет исполнять вложенные циклы без потерь времени.

Во время исполнения инструкции процессором генератор адресов инструкций загружает из памяти следующую инструкцию. Следующий адрес может выбираться из следующих источников:

- Счетчик инструкций PC
- Стек счетчика инструкций
- Регистр инструкций
- Контроллер прерываний

Из счетчика инструкций команда выбирается, если выполнение программы последовательное. Значение из стека счетчика инструкций выбирается в случае возврата из подпрограммы, возврата из обработчика прерывания, перехода к следующей итерации цикла. На самом деле адрес следующей инструкции всегда содержится в счетчике инструкций. Просто при переходе к подпрограмме, входе в цикл, в стек засылается адрес возврата, и при выполнении команды возврата происходит выталкивание из стека значения счетчика инструкций.

Значение из регистра инструкций загружается в счетчик инструкций при выполнении команды безусловного перехода.

Контроллер прерываний загружает в PC значение адреса обработчика прерываний в случае распознавания корректного прерывания.

Со счетчиком инструкций связан стек. В стек заталкиваются значения PC при выполнении подпрограмм, обработчиков прерываний, циклов (DO UNTIL). Стек рассчитан на 16 14-и битных элемента (так как PC имеет размер 14 бит). Поскольку для стека PC не существует команд PUSH PC, POP PC введены следующие инструкции:

srg-TOPSTACK; / Выталкивается верхний элемент стека */*

NOP;/ с задержкой на 1 цикл, так что NOP нужно ставить обязательно */*

TOPSTACK=srg; / Засылаем в стек значение srg */ srg - может быть любым регистром..*

Однако не рекомендуется вручную модифицировать счетчик инструкций, так как это может привести к трудно находимым ошибкам.

Для более полного понимания рассмотрим пример:

Пример: Вызов подпрограммы

```
.section/pm seg_rth;
/* Определяем вектора прерывания */
/* и точку начала программы */
/* инструкцией JUMP start */

 JUMP start; NOP; NOP; NOP;
RTI; NOP; NOP; NOP;
RTI; NOP; NOP; NOP;
 RTI; NOP; NOP; NOP;
SPORT0_TX_INT:     RTI; NOP; NOP; NOP;
SPORT0_RX_INT:     RTI; NOP; NOP; NOP;
 RTI; NOP; NOP, NOP;
 RTI; NOP; NOP; NOP;
 RTI; NOP; NOP; NOP;
 RTI; NOP; NOP; NOP;
NOP; RTI; RTI; RTI;
RTI; NOP; NOP; NOP;
.sectipn/pm seg_code;
Scalar:
        .....
        RTS; /* После выполнение команды в PC значение из стека */

Start:
        .....
        CALL scalar; /* Перед переходом по адресу scalar в стек засылается адрес */
             ..... /* инструкции следующей за CALL scalar */
```

Инструкции DO UNTIL обеспечивают выполнение циклов без тактов ожидания, используя цикловый компаратор и стек PC. Каждый процессорный цикл компаратор сравнивает следующий адрес, получаемый генератором адресов инструкций с адресом последней инструкции цикла (адрес последней инструкции устанавливается в инструкции

DO UNTIL, адрес первой инструкции загружается в стек PC при первом входе в цикл). Если эти адреса совпадают, то происходит извлечение из стека в счетчик инструкций адреса первой инструкции цикла. Условия завершения цикла указаны в таблице 5.5.

Таблица 5.5. Условия завершения цикла

Синтаксис	Условие	Верно тогда, когда:
EQ	Равен нулю	*AZ=1*
NE	не равен нулю	*AZ=0*
LT	Меньше нуля	*AN .XOR. AV=1*
GE	Больше, либо равен нулю	*AN .XOR. AV=0*
LE	Меньше, либо равен нулю	*(AN .XOR. AV) .OR. AZ=1*
GT	Больше нуля	*(AN .XOR. AV) .OR. AZ=0*
AC	Перенос АЛУ	*AC=1*
NOT AC	нет переноса	*AC=0*
AV	Переполнение АЛУ	*AV=1*
NOT AV	нет переполнения	*AV=0*
MV	Переполнение умножителя	*MV=1*
NOT MV	нет переполнения умножителя	*MV=0*
NEG	Аргумент отрицателен	*AS=1*
POS	Аргумент положителен	*AS=0*
CE	Число повторений истекло*	
FOREVER	«вечный цикл»	

* Число повторений устанавливается в счетчик циклов CNTR

Пример: Условный цикл

CNTR=6;

DO bops UNTIL CE;

*Loops: MR=MR+MX0*MY0(SS), MX0=DM(I0,M0), MY0=PM(I6,M6);*

При работе со счетчиком циклов его значение загружается в стек, который может содержать до 4-х значений, что позволяет организовывать до 4-х вложенных циклов. При выполнении инструкций JUMP и CALL загрузка адреса происходит непосредственно в PC. Например, инструкция JUMP start загружает в PC адрес метки start. Логика условных переходов реализуется командой IF условие THEN действие, где условие одно из условий таблицы 5.5.

Пример: Использование команды условного перехода

AR = AXO, AF = AXO AND A Y0;

IF NE AXO = - AXO;

5.2.2 Генераторы адресов данных

В процессорах семейства ADSP-21xx имеется два генератора адресов данных DAG1 и DAG2. Оба генератора обеспечивают косвенную адресацию, и производят автоматическую модификацию адресов. Отличие генератора DAG1 от DAG2 состоит в том, что DAG1 может осуществлять выборку только из памяти данных (DM), а генератор DAG2 как из памяти данных (DM) так и из памяти программ (PM) при этом функционирование двух генераторов осуществляется параллельно. Каждый генератор имеет в своем составе три блока регистров: блок регистров адреса или индексных регистров (I-блок) 4x14, блок регистров длины (L-блок) 4x14, и блок регистров модификации (M-блок) 4x14. Распределение регистров между блоками приведено в таблице 5.6.

Таблица 5.6 Регистры генераторов адресов

	DAG1	DAG2
Блок I-регистров	I0-I3	I4-I7
Блок L-регистров	L0-L3	L4-L7
Блок M-регистров	MO-M3	M4-M7

Генераторы адресов данных имеют два режима адресации: линейную и кольцевую. Индексные регистры содержат реальные адреса памяти, по которым осуществляется доступ к памяти. Регистры длины содержат информацию о длине буфера в режиме кольцевой адресации, в режиме линейной адресации значение в регистре длины должно быть нулевым. Именно так осуществляется выбор метода адресации. В регистры модификации записывается число-модификатор, которое добавляется к значению адреса, хранящегося в индексном регистре. Генераторы адреса данных используют схему пост-модификации, то есть после каждой косвенной адресации к значению регистра I добавляется значение из регистра M. Для индексных регистров и регистров длины существует взаимооднозначное соответствие, это означает, что при установке значения адреса в индексный регистр значение длины необходимо загружать в регистр длины с тем же номером. Однако для регистров модификации такого соответствия нет. Для модификации можно использовать любой регистр модификации из тех, которые принадлежат данному генератору. Например, при работе с массивом адрес которого хранится в регистре 10 можно использовать регистры модификации MO, Ml, M2, M3 но использовать, например, регистр M5 нельзя. Индексные регистры и регистры длины - беззнаковые, старшие два разряда шины DMD заполняются нулями. Регистр модификации

- знаковый, поэтому старшие два разряда шины DMD заполняются знаковым расширением.

Пример: Инициализация и использование генератора адресов

```
Start:
        I0=%Buf1;      /* Загружаем DAG 1*/
        M1= 1;
        L0=0;
        I5=%Buf2;      /* Загружаем DAG 2 */
        M7=1;
        L5=0;
        MX0=DM(I0,M1), MX0=PM(I5,M7);
        CNTR=10;
        DO loops UNTIL CE;
                Loops: MR=MR+MX0*MY0, MX0=DM(I0,M1), MX0=PM(I5,M7);
```

Прием, показанный в примере (MR=MR+MX0*MY0, MX0=DM(I0,M1), MX0=PM(I5,M7);) достаточно широко используется на практике, особенно при реализации фильтров, поскольку используются различные устройства (MAC, DAG 1, DAG 2), которые работают параллельно, три команды выполняются за один цикл!

При использовании кольцевой адресации для вычисления следующего адреса используется следующая формула

Следующий адрес=(I+M-B) mod(L) + B

Где I-индексный регистр, M-модификатор, L-регистр длины, B-базовый адрес. Эти числа должны удовлетворять ограничению |M|<L.

В генераторе адреса DAG 1 имеется режим изменения порядка бит в адресе на обратный (реверс бит). Этот режим полезен для реализации алгоритмов FFT. Точка отсчета середины адреса находится между 6 и 7 битами адреса.

Таблица 5.7 Операция инверсии

	Линии шины DMA (N=14)
В нормальном порядке	13 12 11 10 09 08 07 06 05 04 03 02 01 00
В инвертированном порядке	00 01 02 03 04 05 06 07 08 09 10 11 12 13

Адресация с изменением порядка бит в адресе на обратный - режим, включаемый установкой бита 1 в регистре MSTAT. При установке этого бита в регистре I значение хранится в нормальном порядке, реверс производится лишь при выдаче адреса на шину DMA.

Возможно вывести с изменением порядка бит на обратный, так же и адрес шириной меньше 14 бит. Для этого нужно установить значение в регистре M такое, что при реверсе бит, появлялся нужный адрес.

1. Разработайте программу, вычисляющую БПФ по 128 точкам. Работоспособность проверьте, сравнивая с аналогичной операцией пакета MATLAB.

5.3 Контроллер прерываний

При получении сигнала прерывания контроллер прерываний передает управление команде, расположенной по соответствующему адресу. Вектора прерываний хранятся в памяти программы через четыре ячейки, что позволяет кодировать короткие обработчики прерываний без использования команд перехода, непосредственно в этих ячейках. Возврат в исполняемую программу осуществляется командой RTI. Прерывания могут устанавливаться программно, через регистр IFC. Адреса векторов прерываний для процессора ADSP-2189 приведены в таблице 4.8.

Таблица 5.8 Схема приоритетов прерываний

Адрес	Источник прерывания
0x0000(высокий приоритет)	Запуск программы после RESET
0x002C	Понижение потребляемой мощности
0x0004	IRQ2
0x0008	IRQL1 (по уровню)
0x000C	IRQL0 (по уровню)
0x0010	Передача SPORT0
0x0014	Прием SPORT0
0x0018	IRQE (по уровню)
0x001C	Прерывание прямого побайтового доступа к памяти
0x0020	Передача SPORT1/IRQ1
0x0024	Прием SPORT1/IRQ0
0x0028 (низкий приоритет)	Таймер

Когда поступает запрос на прерывание, он откладывается до конца выполнения текущей инструкции. Затем контроллер сравнивает запрос с регистром маски IMASK. Если прерывание в этот момент не замаскировано, в стек PC помещается PC (который содержит адрес следующей инструкции главной программы). В стек статуса помещаются ASTAT, MSTAT и IMASK в том порядке, в котором перечислены. После помещения 1MASK в стек статуса он автоматически загружается значением, которое определяет, возможны ли вложенные прерывания. После этого процессор выполняет пустой цикл

(NOP), в то время как загружается инструкция, расположенная по адресу соответствующего вектора прерывания. После окончания обработки прерывания инструкцией RTI выталкиваются из стека PC значение PC и из стека статуса ASTAT, MSTAT и 1MASK и исполнение программы продолжается со следующей инструкции.

ICNTL

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	INT_ENA	0	IRQ2	IRQ1	IRQ0

IRQ0 - Реакция IRQ0 (1 - по спаду, 0 - по уровню)

IRQ1 - Реакция IRQ1 (1 - по спаду, 0 - по уровню)

IRQ2 - Реакция 1RQ2 (1 - по спаду, 0 - по уровню)

INT_ENA - Разрешение вложенных прерываний

Регистр маски IMASK (0 - прерывание разрешено, 1 - прерывание запрещено)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	IRQ2	WHIP	RHIP	TSPORT0	RSPORT0	TSPORT1	RSPORT1	TIMER

Здесь и далее:

TIMER - Прерывание от таймера

RSPORT1 - Прерывание приема SPORT1

TSPORT1 - Прерывание передачи SPORT1

RSPORTO - Прерывание приема SPORTO

TSPORTO - Прерывание передачи SPORTO

RHIP - Чтение хост-машины

RHIP - Запись хост-машины

IRQ2 - Прерывание IRQ2

IFC

Очистка прерываний

5	4	3	2	1	0
IRQ2	TSPORT0	RSPORT0	TSPORT1	RSPORT1	TIMER

Установка прерываний

11	10	9	8	7	6
IRQ2	TSPORT0	RSPORT0	TSPORT1	RSPORT1	TIMER

Пример: Установка обработчика прерываний

```
.section/pm seg_rth;
        JUMP start; NOP; NOP; NOP;
        RTI; NOP; NOP; NOP;
        RTI; NOP; NOP; NOP;
        RTI; NOP; NOP; NOP;
        RTI; NOP; NOP; NOP;
        RTI; NOP; NOP; NOP;
        JUMP irqe; NOP; NOP; NOP;  /* Устанавливаем прерывание по внешнему выводу */
        RTI; NOP; NOP; NOP;
        RTI; NOP; NOP; NOP;
        RTI; NOP; NOP; NOP;
        NOP; RTI; RTI; RTI;
        RTI; NOP; NOP; NOP;

        /* Подпрограмма обработки прерываний */
.section/pm seg_code;
SPORT0_rx_int_handler:
        ENA SEC_REG;
        AX0 = RX0;
        TX0 = AX0;
        RTI;
```

5.4 Режим пониженного энергопотребления

В режим пониженного энергопотребления процессор переходит инструкцией IDLE. В режим нормального энергопотребления процессор возвращается по прерыванию. Вызов инструкции может быть без параметров, или в виде:

IDLE n; / где n=16,32,64,128 */*

Данная инструкция заставляет процессор ожидать прерывание в еще более экономичном режиме, деля частоту на n.

5.5 Шины

Внутренние устройства связываются пятью шинами. Шина адреса памяти данных DMA и шина адреса памяти программ PMA используются для указания адресов относящихся к памяти данных и памяти программ. Шины данных памяти данных DMD и шина данных памяти программ PMD используются для данных соответствующего адресного пространства. Шина внутренних результатов R прямо связывает различные внутренние устройства. Ширина шины PMA - 14 бит, что обеспечивает доступ к 16Кбайтам инструкций и данных. Шина PMD имеют ширину 24 бита, что обеспечивает загрузку 3 байтных команд за один цикл.

Ширина шины DMA - 14 бит обеспечивает доступ к 16 Кбайтам данных. Ширина DMD имеет ширину 16 бит.

Устройство обмена между шинами PMD-DMD позволяет пересылать данные с одной шины на другую и содержит логику для преодоления разницы ширины в 8 бит между двумя шинами.

5.6 Последовательные порты

Почти все процессоры семейства 21xx имеют 2 двунаправленных последовательных порта SPORTs с двойной буферизацией. Эти порты используют синхронную передачу данных и используют сигналы кадровой синхронизации, чтобы контролировать потоки данных. Аппаратно каждый последовательный порт представляет собой интерфейс с пятью выводами:

SCLK - Сигнал синхронизации

RFS - Сигнал приема кадра

TFS - Сигнал передачи кадра

DR - Вход приема

DT-Выход передачи

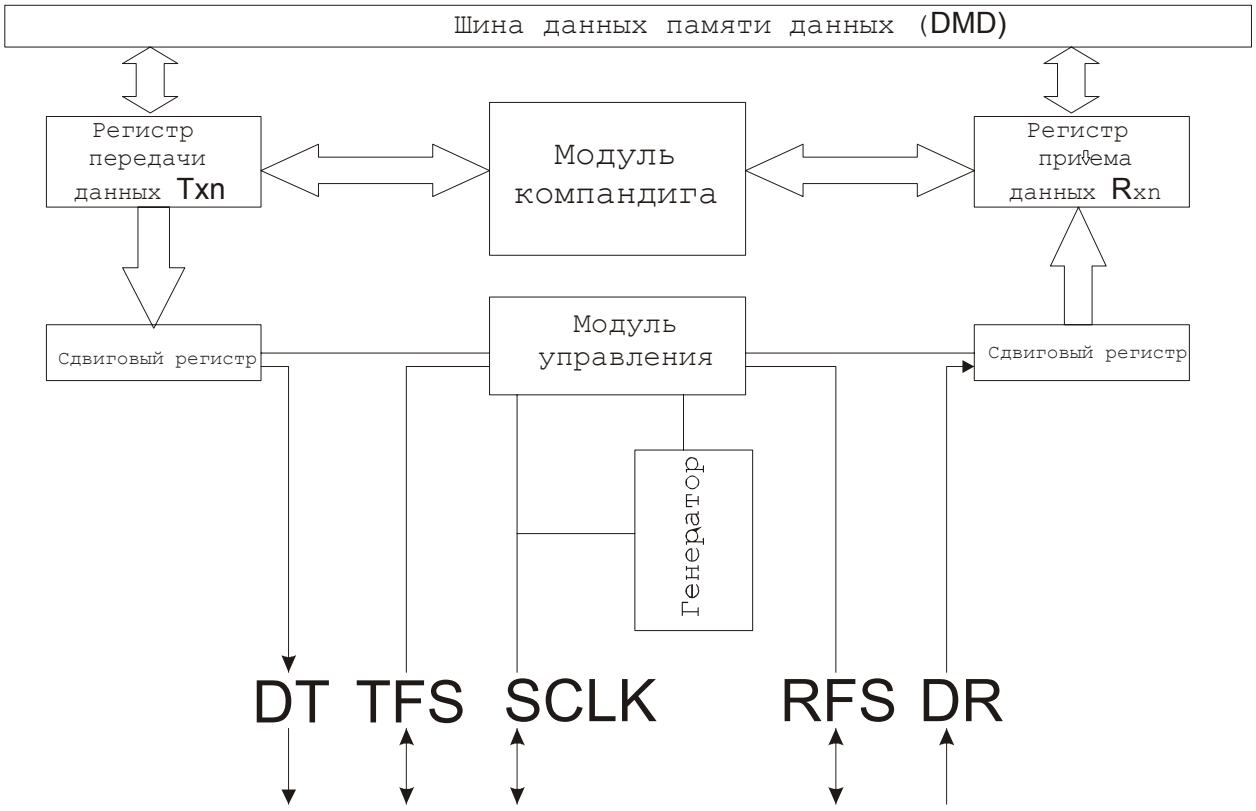

Рис. 5.5 Структурная схема последовательного порта

Порт принимает последовательно передаваемые данные на выводе DR и последовательно передает данные на вывод DT. За счет двойной буферизации, то есть и приемная и передающая часть имеют отдельные буферные регистры и регистры сдвига,

прием и передача осуществляется одновременно. По окончании передачи слова данных (кадра) на выводе TFS генерируется сигнал кадровой синхронизации, указывающий на окончание передачи текущего слова данных и на начало передачи следующего слова данных. Последовательный порт может использовать внешнюю синхронизацию или генерировать свои синхроимпульсы в широком диапазоне частот от 0 Гц.

Последовательный порт SPORT 1 имеет альтернативную конфигурацию, в которой выводы DR и DT конфигурируются как входы прерываний IRQ0 и IRQ1.

Каждый последовательный порт имеет прерывания приема и передачи. Приоритеты прерываний следующие:

Высший

Передача SPORTO

Прием SPORTO

Передача SPORT1

Прием SPORT1

Низший

5.6.1 Работа последовательного порта

После записи слова данных в регистр TX последовательного порта этот порт готов для передачи; побитовая передача инициализируется сигналом TFS. После начала передачи каждое слово данных, записанное в регистр TX, передается на регистр сдвига. Каждый бит слова данных в регистре сдвига сдвигается по переднему фронту сигнала синхронизации.

После окончания передачи первого бита слова данных, из регистра сдвига, последовательный порт генерирует прерывание передачи. Несмотря на то, что передача еще продолжается, становится возможным запись нового слова данных в регистр TX. (Вышесказанное не справедливо для режима автобуферизации, о котором будет подробно рассказано ниже). В принимающей части последовательного порта биты накапливаются во внутреннем регистре в порядке поступления. По окончании приема всего слова данных оно переписывается в регистр RX, и последовательный порт генерирует прерывание.

5.6.2 Регистры конфигурирования последовательного порта

Конфигурирование последовательного порта осуществляется записью определенных значений в регистры конфигурации. SPORT0 имеет следующие регистры конфигурации:

Регистры разрешения многоканального приема SPORT0

0x3FFA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

0x3FF9

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

X=1 – Канал разрешен, X=0 – Канал запрещен

Регистры разрешения многоканальной передачи SPORT0

0x3FF8

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

0x3FF7

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

X=1 – Канал разрешен, X=0 – Канал запрещен

Регистр управления SPORT0

0x3FF6

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MCE	ISCLK	RFSR	RFSW	TFSR	TFSW	ITFS	IRFS	INVTFS	INVRFS	DTYPE		SLEN			

SLEN - Длина слова в битах (1. .16)

DTYPE - Формат данных

INVRFS - Инвертирование кадровой синхронизации приема

INVTFS - Инвертирование кадровой синхронизации передачи

IRFS - Разрешение приема внутренней кадровой синхронизации

ITFS - Разрешение передачи внутренней кадровой синхронизации

TFSW - Передача ширины кадровой синхронизации

TFSR - Прием ширины кадровой синхронизации

ISCLK - Генерация внутренних тактов для SPORT

MCE - Разрешение многоканального режима

Делитель синхроимпульсов (SCLKDIV) SPORT0

0x3FF5

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Делитель частоты кадровой синхронизации приема (RSFDIV) SPORT0

0x3FF4

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Регистр управления автобуферизацией SPORT0

0x3FF3

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	TIREG			TMREG		RJREG			RMREG		TBUF	RBUF

RBUF=1 Разрешена автобуферизация приема, =0 запрещена автобуферизация приема

TBUF=1 Разрешена автобуферизация передачи, =0 запрещена автобуферизация передачи

Регистр управления SPORT1

0x3FF2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FLAG_OUT	ISCLK	RFSR	RFSW	TFSR	TFSW	ITFS	IRFS	INVTFS	INVRFS	DTYPE		SLEN			

SLEN - Длина слова в битах (1.. 16)

DTYPE - Формат данных

INVRFS - Инвертирование кадровой синхронизации приема

INVTFS Инвертирование кадровой синхронизации передачи

IRFS - Разрешение приема внутренней кадровой синхронизации

ITFS - Разрешение передачи внутренней кадровой синхронизации

TFSW - Передача ширины кадровой синхронизации

TFSR - Прием ширины кадровой синхронизации

ISCLK - Генерация внутренних тактов для SPORT

FLAG_OUT - Значение вывода FLAG_OUT (только для чтения)

Делитель синхроимпульсов (SCLKDIV) SPORT1

0x3FF1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Делитель частоты кадровой синхронизации приема (RSFDIV) SPORT1

0x3FF0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Регистр управления автобуферизацией SPORT1

0x3FEF

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	TIREG			TMREG		RIREG			RMREG		TBUF	RBUF

RBUF=1 Разрешена автобуферизация приема, =0 запрещена автобуферизация приема

TBUF=1 Разрешена автобуферизация передачи, =0 запрещена автобуферизация передачи

Широко используется два способа конфигурирования последовательного порта. При первом способе используется непосредственная запись значений в регистры конфигурации:

Пример: Установка режима автобуферизации

```
AX0 = OxFFE1;        {Содержимое регистра AX0 записывается }

DM(0x3FF3) = AXO; {в регистр управления автобуферизацией SPORT0}
```

Второй способ основан на непрерывности адресов конфигурационных регистров. Инициализация проводиться за один цикл, но данный способ требует выполнения подготовительных действий:

Пример: Установка режима автобуферизации

```
I2=0x3FEF;

M2=1;

 L2= 12;

 DM(I2, M2) = 0x6B25;

 DM(I2, M2) = 0;
```

Таким образом, если при конфигурировании последовательного порта необходимо установить значения всех регистров предпочтительным является второй способ. Если при конфигурировании необходимо установить значение части регистров, то тут необходимо подсчитать, какое количество циклов займет решение первым способом и решение вторым способом и выбрать наиболее короткий путь.

5.6.3 Регистры последовательного порта

Прием и передача данных последовательным портом осуществляется путем записи/чтения регистров TX/RX соответственно. Эти регистры не отображены на карте памяти, но распознаются мнемоникой ассемблера, регистрами передачи являются регистры TXO, TX1 для портов SPORT0 и SPORT1 соответственно.

Пример: Прием и передача отсчетов в последовательный порт SPORT0

```
/* Команда передает число 0x2000 для передачи в SPORT0 */

AX0 = 0x2000;

TX0 = AX;

/* Команда считывает из SPORT0 полученное значение */

AX0 = RX0
```

5.6.4 Активизация последовательного порта

Последовательные порты активизируются битами в регистре управления системой. Этот регистр находится по адресу 0x3FFF

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			E/D SPORT0	E/D SPORT 1	MODE SPORT1	BFORCE	BPAGE			BWAIT			PWAIT		

PWAIT- Количество тактов ожидания программной памяти B

WAIT - Количество тактов ожидания загрузочной памяти

SPACE - Количество тактов ожидания программной памяти

BFORCE - Количество тактов ожидания программной памяти

MODE SPORT1 - Режим SPORT1

E/D SPORT1 - Активизация SPORT1

E/D SPORT0 - Активизация SPORT0

Активизация последовательных портов осуществляется установкой 1 в соответствующих битах регистра управления системой. Бит 10 определяет конфигурацию SPORT1. Если бит 10 установлен в 1 то SPORT 1 работает как обычный последовательный порт, если бит 10 установлен в 0 то выводы SPORT1 имеют следующее назначение:

Название вывода	Альтернативное название	Альтернативная функция
RFS1	IRQ0	Внешнее прерывание 0
TFS1	IRQ1	Внешнее прерывание 1
DR1	FI	Вход флага
DT1	FO	Выход флага
SCLK1	SCLK1	Синхронизация

5.6.5 Синхронизация последовательных портов

Каждый последовательный порт использует свои синхросигналы. Синхроимпульсы могут быть внутренними, то есть генерироваться процессором, или внешними.

Бит ISCLK — бит 14 регистра управления SPORT0 или SPORT1 определяет источник тактовых синхроимпульсов для каждого последовательного порта. Когда этот бит равен 1, процессор сам генерирует синхросигналы, когда бит равен 0 процессор ожидает синхросигнала от внешнего источника. После перезапуска бит ISCLK сбрасывается, таким образом, для обоих портов устанавливается режим работы с внешними синхросигналами. Со следующего цикла после установки бита ISCLK начинается генерация внутренних синхроимпульсов, независимо от того, разблокирован или заблокирован соответствующий последовательный порт.

Внешние тактовые синхроимпульсы могут иметь частоты равные тактовой частоте процессора, то есть до 13.824МГц. Частота внутренних тактовых

синхроимпульсов функционально связана с тактовой частотой процессора и определяются содержимым регистра SCLKDIV (0x3FF5 для SPORT0 и 0x3FFl для SPORT 1).

SCLKDIV	Частота SCLK
20479	300Гц
5119	1200Гц
639	9600Гц
95	64кГц
3	1,536МГц
2	2,048МГц
0	6,144МГц

Если значение SCLKDIV меняется, когда последовательный порт разблокирован, то частота SCLK меняется по переднему фронту следующего импульса.

5.6.6 Длина слова

Каждый последовательный порт может обрабатывать слова длиной от 3 до 16 бит. Количество бит в слове устанавливается в регистре управления каждого последовательного порта установкой группы бит SLEN. Длина слова равна SLEN+1. Например, для работы со словами длиной в 4 бита, SLEN устанавливается равным B#0011. Слова данных, имеющие длину меньше 16 бит, выравниваются по правому разряду в регистрах данных последовательного порта. **Запрещено устанавливать SLEN в 0 и 1.**

5.6.7 Кадровая синхронизация

При возникновении необходимости синхронизировать передачу каждого слова (кадра) имеется режим кадровой синхронизации. Переход в этот режим осуществляется установкой битов 13 - для приема и 11 - для передачи регистра управления последовательным портом. При включении и после сброса эти биты обнуляются. При нулевом значении этих битов сигнал кадровой синхронизации генерируется только для первого переданного и принятого слова. При установленных в 1 битах сигнал генерируется при передаче и приеме каждого слова.

Источники сигналов кадровой синхронизации могут быть как внутренними, так и внешними, и различаться для приема и передачи. Например, для передачи может быть установлен внешний источник, а на прием внутренняя кадровая синхронизация. Источник устанавливается битами IRFS и ITFS в регистре управления последовательным портом. При установке бита в 1 используется внутренний источник сигналов кадровой синхронизации, при установке бита в 0 используется внешний источник. Число циклов тактового генератора между подтверждением приема определяется как RFSDIV+1, где RFSDIV значение, записанное в регистр делителя кадровой синхронизации приема RFSDIV (0x3FF4 для SPORT0 и 0x3FF0 для SPORT1).

В нормальном кадровом режиме сигналы кадровой синхронизации проверяются на заднем фронте SCLK. Если сигнал кадровой синхронизации зарегистрирован, принимаемые данные фиксируются по заднему фронту следующего импульса SCLK, а передаваемые данные передаются на следующем переднем фронте сигнала SCLK.

В альтернативном режиме, сигналы кадровой синхронизации должны приходить в том же цикле, что и передается или принимается бит данных. Принимаемые данные фиксируются на заднем фронте SCLK, а передаваемые данные передаются на переднем фронте сигнала SCLK, но сигнал кадровой синхронизации проверяется только для первого бита.

Кадровые режимы устанавливаются независимо для приема и передачи. Если бит RFSW в регистре управления последовательным портом сброшен в 0, используется получение данных с кадровой синхронизацией в нормальном режиме, если установлен в 1 - используется альтернативный режим. Для выбора режима кадровой синхронизации при передаче данных служит бит TFSW. Его назначение аналогично RFSW.

5.6.8 Пример конфигурации

Пример: Пример конфигурации последовательного порта

{ Подпрограмма остановки последовательного порта}

```
CodecStop:
    imask = 0x0001;  {Запрещаем все прерывания кроме таймера }
    ax0 = 0;
    dm (SPORT0_Autobuf) = ax0;  { Обнуляем регистр автобуферизации}
    dm (SPORT0_Control_Reg) = ax0;  { Обнуляем регистр управления}

    ax0 = dm (System_Control_Reg);
    ay0 = b#1110111111111111;
    ar = ax0 and ay0;
    dm (System_Control_Reg) = ar;   {Запрещаем работу последовательного
порта}
    rts;

CodecInit:
        call CodecStop;
        ifc = b#00000011111110;
        nop;
        { 1. Устанавливаем внешнюю синхронизацию }
        ax0 = dm(SPORT0_Control_Reg);
        ay0 = b#1011111111111111;
        ar = ax0 and ay0;
        dm(SPORT0_Control_Reg) = ar;
        { 2. Устанавливаем длину слова равной 16 бит }
        ax0 = dm(SPORT0_Control_Reg);
        ay0 = b#0000000000001111;
        ar = ax0 or ay0;
        dm(SPORT0_Control_Reg) = ar;
        { 3. Сигналы кадровой синхронизации по передачи и приему требуются
для каждого слова}
        ax0 = dm(SPORT0_Control_Reg);
        ay0 = b#0010100000000000;
        ar = ax0 or ay0;
        dm(SPORT0_Control_Reg) = ar;
        { 4. Для приема устанавливаем внутреннюю кадровую синхронизацию }


        ax0 = dm(SPORT0_Control_Reg);
        ay0 = b#1111111011111111;
        ar = ax0 and ay0;
        dm(SPORT0_Control_Reg) = ar;
        { 5. Для передачи устанавливаем внешнюю кадровую синхронизацию }
        ax0 = dm(SPORT0_Control_Reg);
        ay0 = b#0000001000000000;
        ar = ax0 or ay0;
        dm(SPORT0_Control_Reg) = ar;
        { 6. Передавить ширину кадровой синхронизации не нужно}
        ax0 = dm(SPORT0_Control_Reg);
        ay0 = b#1110101111111111;
        ar = ax0 and ay0;
        dm(SPORT0_Control_Reg) = ar;
        { 7. Синхронизация по переднему фронту}
        ax0 = dm(SPORT0_Control_Reg);
        ay0 = b#1111111100111111;
        ar = ax0 and ay0;
        dm(SPORT0_Control_Reg) = ar;
        {8. Отключаем автобуферизацию }
        ax0 = dm(SPORT0_Autobuf);
        ay0 = b#0000111111111100;
        ar = ax0 or ay0;
        dm(SPORT0_Autobuf) = ar;
        ax0 = b#0001000000000000;
        ay0 = dm (System_Control_Reg);
        ar = ax0 or ay0;
       dm (System_Control_Reg) = ar;
        icntl = 2;
        Imask = b#0001110001;
Rts;
```

Прием, использованный для модификации регистров, является в общем случае более выгодным, чем непосредственная запись в регистр необходимого значения, поскольку считывая имеющееся значение и модифицируя логической операцией один или

несколько бит, мы не затрагиваем остальных установок, что во многих случаях позволяет избежать трудно обнаружимых ошибок.

5.6.9 Автобуферизация

При работе без автобуферизации при приеме или начале передачи слова генерируется прерывание. Механизм автобуферизации позволяет передавать и принимать данные блоками. Автобуферизация использует способность генераторов адреса данных осуществлять адресацию циклических буферов. При разрешенной автобуферизации каждое слово данных последовательно передается в или из памяти данных за один непроизводительный цикл. Таким образом, при выполнении приема или передачи слова прерывание не генерируется, просто программа задерживается на один цикл (именно так образуется непроизводительный цикл). Выбор используемого генератора осуществляется в регистре управления автобуферизацией (0x3FF3 для SPORT0 и 0x3FEF для SPORT1). Биты TIREG и TMREG определяют номер I - регистра и M - регистра соответственно, для передачи. Биты RIREG и RMREG определяют номер I - регистра и M - регистра соответственно, для приема. Биты TBUF и RBUF отвечают за запрещение автобуферизации.

1. Разработайте программу осуществляющую получение данных из порта SPORT0 и пересылку данных в SP0RT1.

5.6.10 Компандинг (упаковка и распаковка данных)

Компандинг - это процесс логарифмического кодирования и декодирования данных, обеспечивающий минимизацию количества пересылаемых байт, иначе говоря, сжатие. Процессоры семейства ADSP-21xx аппаратно поддерживают сжатие как по A-закону, так и по u-закону. Тип сжатия, как и формат несжатых данных, устанавливается битами DTYPE в регистре управления последовательным портом.

5.7 Таймер

Программируемый интервальный таймер обеспечивает периодическую генерацию прерываний. 8-битный масштаб позволяет ему декрементировать содержимое 16-битного регистра-счетчика в диапазоне от каждого цикла до каждого 256 цикла. Прерывание генерируется, когда регистр-счетчик обнуляется. Регистр-счетчик автоматически загружается из 16-битного регистра интервала и отсчет времени немедленно возобновляется.

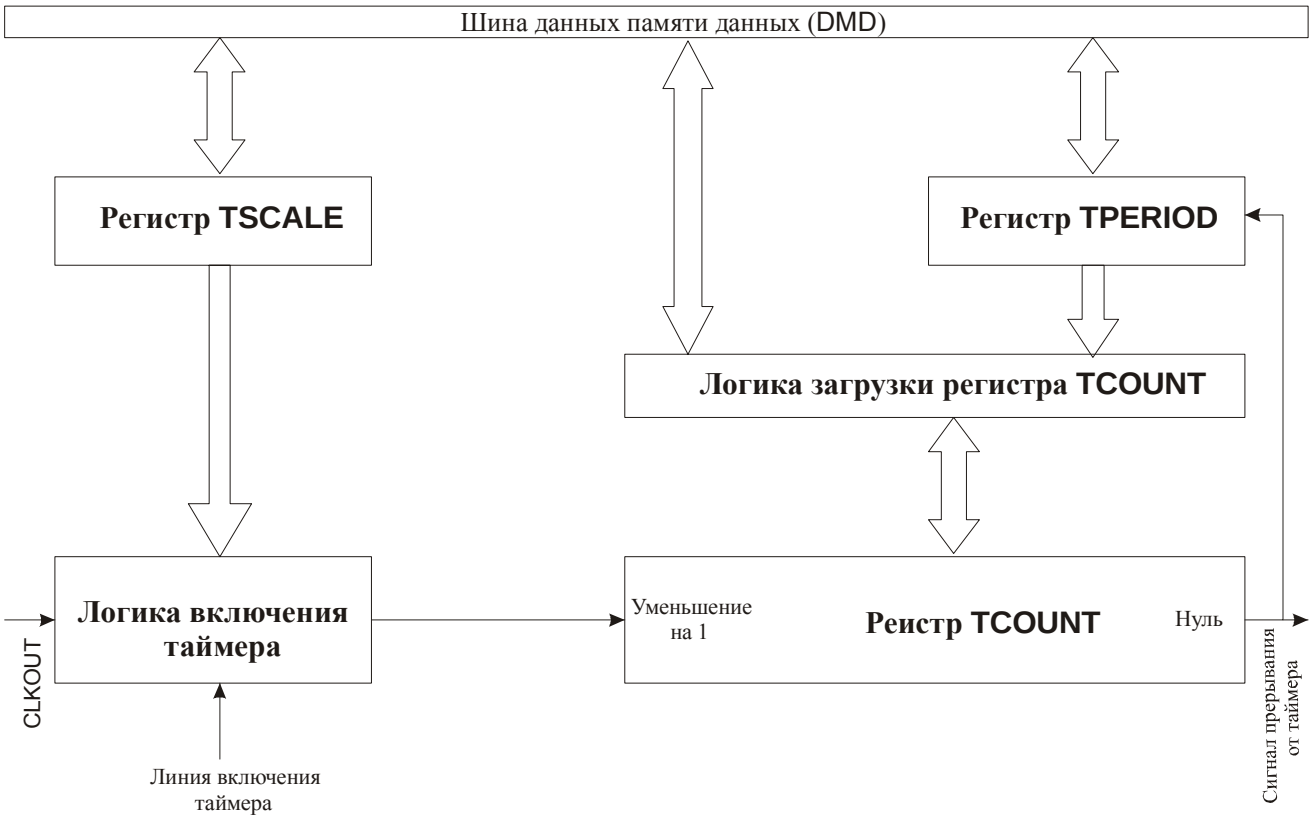

Рис. 5.7 Структурная схема таймера

Таймер включает в себя два 16-битных регистра - TCOUNT и TPERIOD, и один 8-битный - TSCALE. Расширенные инструкции управления позволяют разрешать и запрещать таймер установкой и очисткой бита 5 в регистре MSTAT.

Регистр-счетчик TCOUNT
0x3FFC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

При разрешенном таймере, значение, записанное в этом регистре, декрементируется каждый процессорный цикл. При достижении нуля вырабатывается сигнал прерывания.

Регистр периода TPERIOD
0x3FFD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

После обработки прерывания в регистр TCOUNT загружается значение, находящееся в регистре TPERIOD.

TSCALE содержит коэффициент масштабирования, на единицу меньший, чем количество процессорных циклов между декрементами счетчика. Таким образом, прерывание происходит каждые (TPERIOD+l)*(TSCALE+l) циклов. Однако первое прерывание генерируется через (TCOUNT+l)*(TSCALE+l) циклов.

Пример: Конфигурирование таймера

Config timer:

AR=120,

DM(TCOUNT) =AR;

AR=120:

DM(TPERIOD)=AR;

AR=0;

DM(TSCALE) =AR;

RTS;

Разработайте программу генератора синусоидальных колебаний.

ГЛАВА 6. Задания по лабораторным работам на EZ-KIT - 2189M

Лабораторная работа №1 «Умножение векторов (свертка)»

Цель работы:

Изучение приемов работы с АЛУ

Теоретическое введение:

Свертка достаточно часто встречающаяся операция в обработке цифровых сигналов. Рассмотрим прямую форму перемножения векторов (свертки)

$$\begin{pmatrix} z_1 \\ \dots \\ z_n \end{pmatrix} = \begin{pmatrix} x_1 \\ \dots \\ x_n \end{pmatrix} \times \begin{pmatrix} y_1 & \dots & y_n \end{pmatrix}, \text{ где } z_i = \sum_{j=1}^{n} x_i \cdot y_j$$

Программа, реализующая данный алгоритм приведена в примере.

Пример: Умножение векторов

```
/* Определение аргументов и буфера, хранящего результат  */
.section/data seg_data;
        .VAR vect1[2] = 1,2;
        .VAR vect2[2] = 2,1;
        .VAR result[2];
/* Подпрограмма умножения векторов */
.section/pm seg_code;
vect_mul:
        CNTR = 2;
        DO loop1 UNTIL CE;
                MR = 0;
                MX0 = DM(I0, M0);
                CNTR = 2;
                I1 = vect2;
                DO loop2 UNTIL CE;
                        MY0 = DM(I1, M0);
                loop2:  MR = MR + MX0 * MY0(SS);
        loop1:  DM(I2, M0) = MR0;
        RTS;
/* Секция инициализации и вызова подпрограммы vect_mul*/
.section/pm seg_code;
start:
        I0 = vect1;
        I2 = result;
        M0 = 1;
        CALL vect_mul;
```

Однако данный способ не слишком рационален, поскольку требует n умножений и n сложений, более рациональным является следующий алгоритм вычисления:

$$z_i = x_i \cdot \sum_{j=1}^{n} y_j$$

подпрограмма, реализующая данный алгоритм приведена в следующем примере.

Пример: Оптимизированное умножение векторов

```
/* Определение аргументов и буфера, хранящего результат  */
.section/data seg_data;
        .VAR vect1[2] = 1,2;
        .VAR vect2[2] = 2,1;
        .VAR result[2];
/* Подпрограмма умножения векторов */
            mul_vect:
                CNTR = 2;
                DO mul UNTIL CE;
                    AR = 0;
                    CNTR = L1;
                    DO add UNTIL CE;
                        add:AY0 = DM(I1, M0),      AR = AR + AY0;
                    MY0 = DM(I0, M0);
                    MR = AR * MY0(US);
                mul:DM(I2,M0) = MR0;
            RTS;
/* Секция инициализации и вызова подпрограммы vect_mul*/
        start:  I0 = vect1;
                I1 = vect2;
                L1 = 2;
                I2 = result;
                M0 = 1;
                CALL mul_vect;
```

Задание:

1. Написать и отладить программу сложения векторов.
2. Написать и отладить программу сложения матриц (элементы в линейном буфере хранить либо по столбцам, либо по строкам).
3. Написать и отладить программу умножения матриц.
4. Измерить время выполнения программы, приведенной в примере, и количество MIPS, которые занимает эта программа.
5. Оформить отчет

Лабораторная работа №2 «Трансверсальный КИХ- фильтр»

Цель работы:

Изучение приемов работы с умножителем-накопителем

Теоретическое введение:

Трансверсальный называется фильтр, осуществляющий взвешенное суммирование ряда отсчетов входного сигнала и не использующий выходные отсчеты:

$$y_i=a_0x_i+a_1x_{i-1}+...+a_mx_{i-m}$$

Таким образом, трансверсальный фильтр – это фильтр с конечной импульсной характеристикой $\{h_k\}=\{a_0,a_i...a_k\ldots a_m\}$

Его системная функция $H(Z) = a_0 + a_1Z^{-1}+...a_mZ^{-m}= (a_0Z^m + a_kZ^{m-k}+a_mZ^0)/Z^m$

является конечной дробно-рациональной, с m-кратным полюсом при **Z=0** и имеющей **m** нулей, определяемых коэффициентами $\{a_к\}$.

Структура фильтра может быть изображена в виде, который поясняет название - от англ. **transverse** - поперечный.

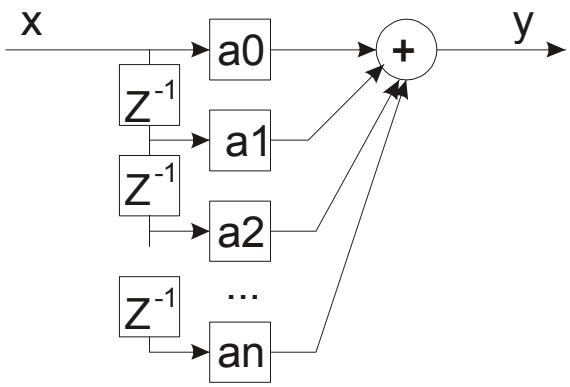

Рис. 6.1 Структура КИХ-фильтра

Основными элементами фильтра служат устройства задержки отсчетных значений на один интерват дискретизации (Z^{-1}), а также масштабные звенья, выполняющие в цифровой форме операции умножения на соответствующие коэффициенты.

С выходов масштабных звеньев сигнаты поступают на сумматор, на выходе которого образуется отсчет выходного сигната.

Частотная характеристика трансверсального фильтра получается подстановкой **Z=exp(jωt)** в **H(z)**:

$K(j\omega)=a_0+a_1\exp(-j\omega\tau)+ a_2\exp(-j\omega\tau)...\exp(jM\omega\tau)$

Подбором коэффициентов вид ЧХ можно видоизменять в широких пределах.

Пример **$y_i=1/3(x_i+x_{i-1}+x_{i-2})$** усреднение трех отсчетов (Ц.Ф. второго порядка).

Если **ωτ<120** (на один период входного колебания приходится не менее трех) отсчетов, фильтр ведет себя как ФНЧ. Наличие более высокочастотных сигналов (**ωτ>120**) приводит к появлению паразитных низкочастотных составляющих, отсутствующих во входном колебании (эффект наложения). Поэтому цифровому фильтру должен предшествовать аналоговый режектор.

В примере приведена программа трансверсального КИХ – фильтра

Пример: Программа фильтра КИХ - фильтра

```
/*
I0 – буфер, хранящий прошлые отсчеты
L0 – размер буфера I0, равный порядку фильтра
I4 – буфер, хранящий коэффициенты фильтра
L4 –количество коэффициентов
 */
fir:
 MR = 0, MX0 = DM(I1,M1), MY0 = PM(I4,m4);    /* Чтение входного значения и
коэффициента*/
 DM(I0,M3) = MX0;                              /* Устанавливаем входное значение
на линию задержки */

  CNTR=Samps_per_Bin;
  DO mult_acc UNTIL CE;
.repeat (Taps-1);
        MR=MR+MX0*MY0(SS), MX0=DM(I0,M3), MY0=PM(I4,M4);
.end_repeat;
MR=MR+MX0*MY0(SS), MX0=DM(I0,M1), MY0=PM(I4,M4);
SR=ASHIFT MR2 (HI), MX0=DM(I1,M1);
SR=SR OR LSHIFT MR1 (LO), DM(I0,M3) = MX0;
mult_acc:
MR = 0, DM(I2,M1)=SR0;                    /* Записываем выходное значение */
rts (db);
MX0 = DM(I0,M1);                  /* Обновляем указатель линии задержки */
MX0 = DM(I1,M3);
```

Задание:

1. Разработайте программу полосового КИХ-фильтра с полосой пропускания

от 20Гц до 10кГц.

2. Разработайте программу режекторного КИХ-фильтра с полосой задержки от 5кГц до 15кГц.

3. Измерить время выполнения программы, приведенной в примере, и количество MIPS, которые занимает эта программа.

4. Оформить отчет.

Лабораторная работа №3 «Реализация БИХ - фильтра на языке С»

Цель работы:

Освоение средств языка С для разработки ПО для ЦСП

Теоретическое введение:

БИХ фильтром или рекурсивным фильтром называется фильтр, осуществляющий взвешенное суммирование ряда отсчетов входного сигнала и взвешенное суммирование выходных отсчетов:

$$y_i=a_0x_i+a_1x_{i-1}+...+a_nx_{i-n}+ b_0y_i+b_1y_{i-1}+...+b_my_{i-m}$$

Таким образом, рекурсивный фильтр - это фильтр с бесконечной импульсной характеристикой $\{h_k\}$.

Структура фильтра изображена на рис. 6.2.

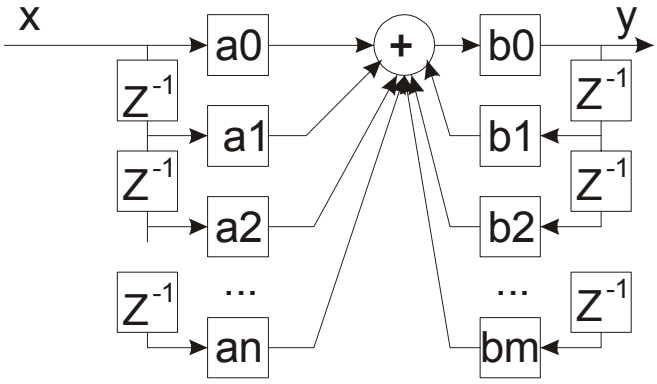

Рис. 6.2. Структура БИХ-фильтра

Основными элементами фильтра служат устройства задержки отсчетных значений на один интервал дискретизации (Z^{-1}), а также масштабные звенья, выполняющие в цифровой форме операции умножения на соответствующие коэффициенты.

С выходов масштабных звеньев сигналы поступают на сумматор, на выходе которого образуется отсчет выходного сигнала.

В примере приведена программа рекурсивного БИХ - фильтра

Пример: Функция БИХ – фильтра

```
/*********************************************************************
Описание: БИХ фильтр
        Вход:
                pInDelay – линия задержки входных сигналов
                pOutDelay - линия задержки выходных сигналов
                pAFilter, pBFilter - коэффициенты
                sample – входной отсчет
*********************************************************************/
#define ORDERA 10
#define ORDERB 10
int iir(int *pInDelay, int *pOutDelay, int *pAFilter, int *pBFilter)
{
        int i;
        int res;
        for(i = 0; i < ORDERB; i++)
                res += (pInDelay[i] * pBFilter[i]) << 1;

        for(i = 0; i < ORDERA; i++)
                res += (pOutDelay[i] * pAFilter[i]) << 1;

        return res;
}
```

Поскольку использование языков высокого уровня позволяет ускорить разработку программ, а так же обеспечивает многоплатформенность (на уровне исходного кода), автор рекомендует, как можно более активно использовать в разработках языки высокого уровня.

Задание:

1. Написать и отладить на языке С программу полосового БИХ - фильтра с полосой пропускания от 20Гц до 10кГц.
2. Написать и отладить на языке С программу режекторного БИХ - фильтра с полосой задержки от 5 кГц до 15кГц.
3. Измерить время выполнения программы, приведенной в примере, и количество MIPS, которые занимает эта программа
4. Оформить отчет

Лабораторная работа №4 «Быстрое преобразование Фурье»

Цель работы:

Практическое изучение основ спектрального анализа

Теоретическое введение:

Операция скремблирования (инвертирования бит) широко используется в программах вычисления быстрого преобразования Фурье (БПФ).

Математическая постановка задачи.

Выражение для преобразования Фурье имеет следующий вид:

$$X(\omega) = \int_{-\infty}^{\infty} x(t)\, e^{-i\omega t} dt \qquad (1)$$

Дискретное преобразование Фурье (ДПФ) ориентировано на дискретные или цифровые вычисления при дискретизации по времени и частоте. Оно ограничено вычислениями в конечном множестве дискретных точек, представляющих данные, и образуется квантованием временных t и w, отсюда:

$$X(k\omega_0) = \sum_{n=0}^{N-1} x(n) \cdot e^{-2\pi i k n / N} = \sum_{n=0}^{N-1} x(n) \cdot W^{-nk} \qquad (3)$$

$$W = e^{2\pi i / N} = \left[\cos\left(\frac{2\pi}{N}\right) - i\sin\left(\frac{2\pi}{N}\right)\right] \qquad (4)$$

Прямая оценка ДПФ требует большой вычислительной мощности, для комплексных величин:

$$X(k) = XR(k) + iXI(k) = \sum_{n=0}^{N-1} [xr(n) + ixi(n)]\left(\cos\frac{2\pi kn}{N} - i\sin\frac{2\pi kn}{N}\right)$$

Отсюда:

$$XR(k) = \sum_{n=0}^{N-1} xr(n) \cdot \cos\frac{-2\pi nk}{N} + xi(n) \cdot \sin\frac{-2\pi nk}{N}$$

$$XI(k) = \sum_{n=0}^{N-1} xi(n) \cdot \cos\frac{-2\pi nk}{N} + xr(n) \cdot \sin\frac{-2\pi nk}{N}$$

Отсюда следует, что прямая оценка ДПФ требует $4N^2$ умножений и примерно столько же сложений. В этой оценке игнорируются вычисления, включающие синус и косинус, которые вычисляются особо, причем вычисление может быть выполнено заранее. Данную постановку задачи можно изложить в матричном виде, более близком к реализации в режиме распараллеливания вычислений:

Постановка задачи в матричной форме

Дана матрица w размерности NxN вида:

$$\|W_{nk}\| = \exp(-27\pi ikn/N). \qquad (8)$$

Очевидно, она симметричная в силу свойств экспоненты.

Дискретным преобразованием Фурье комплексного вектора a будет комплексный вектор b вида: b = wa или

$$b_j = \sum_{k=0}^{N-1} a_k \exp((2\pi i/N)*kj), \quad 0 \le j \le N-1. \qquad (9)$$

$$Rl(b_j) = \sum_{k=0}^{N-1} \left[Rl(a_k)\cos(2\pi kj/N) - Im(a_k)\sin(2\pi kj/N) \right] \qquad (10^a)$$

$$Im(b_j) = \sum_{k=0}^{N-1} \left[Im(a_k)\cos(2\pi kj/N) + Rl(a_k)\sin(2\pi kj/N) \right] \qquad (10^b)$$

Иными словами, операция вычисления ДПФ вектора сводится к нахождению произведения матрицы на вектор в комплексной области. Значит, в наиболее простом случае это вычисление может производиться однослойной адаптивной нейросети, содержащей 2n нейронов с входами каждый.

Нейронная модель

Предпосылками эффективного приложения параллельных алгоритмов для вычисления являются [16]:

- полная адекватность алгоритма ДПФ нейробазису, (в данном случае алгоритм содержит только операции сложения и умножения);
- выполнение на нейронных сетях операций умножения и сложения за одинаковое время (это один из тезисов построения нейронных алгоритмов), что позволяет обойтись без декомпозиции алгоритма типа БПФ и позволяет вычислять ДПФ любой

размерности;

- ДПФ является одной из базовых операций во многих задачах обработки сигналов и может использоваться в силу своей адекватности нейронному базису для построения универсального нейрокомпьютера для обработки сигналов (сжатия и декодирования информации и др.).

Классическим вариантом использования параллельных нейросетевых алгоритмов для вычисления спектра является использование нейросети с нейронами типа «Адалин» (adaptive linear neuron - адаптивная модель нейрона) для реализации дискретного преобразования Фурье. Адалин представляет собой единичный многовходовой нейрон без нелинейного преобразования (функции активации) с контуром настройки весовых коэффициентов, изображенный на рис. 1.

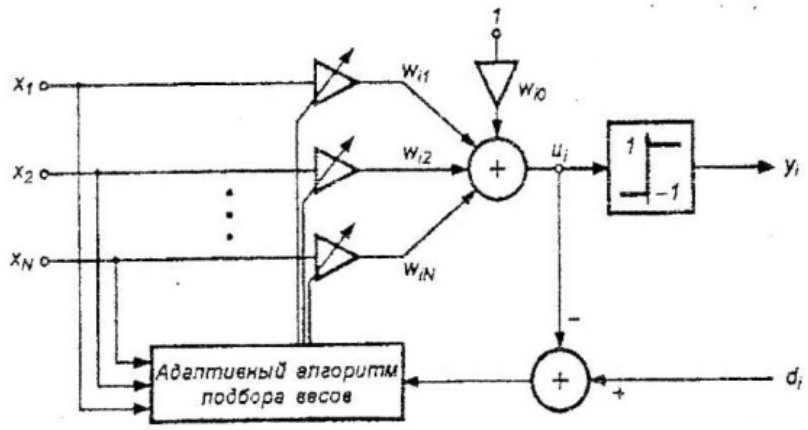

Рисунок 1. Структурная схема нейрона типа «Адалин»

Адаптивный подбор весовых коэффициентов осуществляется в процессе минимизации квадратичной ошибки, определяемой по формуле (11).

$$E(w) = \frac{1}{2}e_i^2 = \frac{1}{2}[d_i - (\sum_{j=0}^{N} w_{ij}x_j)]^2 \qquad (11)$$

Алгоритмы обучения нейросети типа «Адалин»

В связи с выполнением условия непрерывности целевой функции (11) стало возможным применение алгоритма градиентного обучения.

Возьмем в качестве критерия оптимизации минимум среднеквадратической ошибки.

функционал оптимизации будет иметь вид: $F = \{e\}^2(U)$.

Далее, записав градиентную обучающую процедуру для обучения весовых коэффициентов Адалина:

$$w_{ij}(t+1) = w_{ij}(t) + \mu(\nabla F) \tag{12}$$

$$(\nabla F)_{ij} = \frac{\partial e^2}{\partial w_{ij}} = 2e\frac{\partial(x_j w_{ij} - y^*)}{\partial w_{ij}} = 2ex_j \tag{13}$$

Из (12) и (13) следует, что:

$w_{ij}(t+1) = w_{ij}(t) + \eta e_i x_j$ - дискретный способ или

$\frac{dw_{ij}}{dt} = \mu e_i x_j$ - аналоговый способ (путем решения разностных уравнений)

где $e_i = (d_i - \sum_{j=0}^{N} w_{ij} x_j)$

Недостатком приведенной структуры с точки зрения практической реализации является отсутствие фиксированной разомкнутой системы, производящей вычисление дискретного преобразования Фурье любого заданного вектора за фиксированное число тактов. Для каждого вектора происходит свой процесс настройки. Время обучения определяется начальными значениями весов и динамикой процесса минимизации среднеквадратической ошибки. Число итераций в общем случае можно лишь оценить с различной степенью достоверности.

Полученные в ходе моделирования оценки спектров отдельных типовых сигналов позволяют сформировать набор векторов вход-выход, который в дальнейшем может быть использован, как для обучения (в данном случае реализации для ДПФ нейроалгоритма), так и для последующего контроля правильности (адекватности) созданной уточненной программной модели.

Полученные в ходе моделирования оценки спектров отдельных типовых сигналов позволяют сформировать набор векторов вход-выход, который в дальнейшем может быть использован, как для обучения (в данном случае реализации для ДПФ нейроалгоритма), так и для последующего контроля правильности (адекватности) созданной уточненной программной модели. Проведение начального моделирования на базе типовых пакетов позволяет лучше понять алгоритм функционирования, его суть и качественно оценить зависимость между входными и выходными параметрами. Переход к построению

программной модели позволяет полностью отработать алгоритмический базис (все вычислительные процедуры создаются внутри модели) и оценить числовые погрешности, что достигается переводом всех данные в модели в определённую разрядную сетку. Если ошибка оказывается несущественной, т.е. сигнал не будет пересекать границу принятия решения, разрядности выбраны правильно, в противном случае их следует пересмотреть. Программная модель, разработана на языке C++, включает три компонента:

fft.h - комплект функций, реализующих ДПФ.

module.h - комплект функций визуализации.

nfft.cpp - основной модуль модели.

input.sig - входной вектор

output.sig - выходной вектор

Листинг программной модели представлен в таблице 1.

Листинг компонента подпрограммы спектрального оценивания на С [16].

```
////////////////////////////////////////////////////////////
// Компонент обеспечивающий вычисление ДПФ функции v1.12 //
// Включает в себя:                                       //
// функцию FFT - функция вычисления модуля энергетической//
// характеристики при заданной частоте                    //
// Функция BPF - функция быстрого преобразования Фурье,   //
// возвращает модуль энергетической характеристики        //
// при заданной частоте.                                  //
////////////////////////////////////////////////////////////
#include <math.h>
#include <iostream.h>
#include <stdio.h>
#include <fourier.h>
////////////////////////////////////////////////////////////
// функция FFT - функция вычисления модуля энергетической//
// характеристики при заданной частоте                    //
// Входные параметры: int N - размер входного вектора,    //
// double Y[] - входной вектор,                           //
// double W - заданная циклическая частота,               //
// double dt - промежуток между отсчетами,                //
////////////////////////////////////////////////////////////
double FFT(int N,double Y[],double W, double dt){
double Re = 0;
double Im = 0;
for (int i = 0; i < N; i++){
      Re = Re + Y[i]*cos(W*i*dt); //Re Вычисление вещественной части
```

```
      Im = Im + Y[i]*sin(W*i*dt); //Im Вычисление мнимой части
}
return sqrt( Re*Re + Im*Im )*dt; // возвращение модуля энергетической
                                 //характеристики при заданной частоте
}
////////////////////////////////////////////////////////////
// Функция BPF - функция быстрого преобразования Фурье,//
// возвращает модуль энергетической характеристики     //
// при заданной частоте.                               //
// Входные параметры: int N - размер входного вектора, //
// double Y[] - входной вектор,                        //
// double W - заданная циклическая частота,            //
// double dt - промежуток между отсчетами,             //
////////////////////////////////////////////////////////////
double BPF(int N, double Y[], double W, double dt){
double Arg = 0;   //
double H_Re = 0;  //
double H_Im = 0;  //
double Re_X = 0;  //
double Re_Z = 0;  //

  double Im_X = 0;  //
  double Im_Z = 0;  // }Объявление вспомогательных переменных
  double Re_Sz = 0; //
  double Im_Sz = 0; //
  double Re_S = 0;  //
  double Im_S = 0;  //
  double lSl = 0;   //
  // представляем входной вектор Y через два вспомогательных X и Z
  for (int i = 0; i < N/2 ; i++){
        Arg =2 * W * i / N;
        H_Re = cos(Arg); // гармоника вещественной части
        H_Im = sin(Arg); // гармоника мнимой части
        Re_X = Re_X + Y[2*i] * H_Re;    // Re part of X
        Im_X = Im_X + Y[2*i] * H_Im;       // Im part of X
        Re_Z = Re_Z + Y[2*i+1] * H_Re;   // Re part of Z
        Im_Z = Im_Z + Y[2*i+1] * H_Im;     // Im part of Z
  }
  Re_Sz = Re_Z*cos(W / N) - Im_Z*sin(W / N); // Re part of Sz
  Im_Sz = Im_Z*cos(W / N) - Re_Z*sin(W / N); // Im part of Sz
  Re_S = Re_X + Re_Sz;  // Re part of S
  Im_S = Im_X + Im_Sz;  // Im part of S
  lSl = sqrt ( Re_S*Re_S + Im_S*Im_S ); // module S
  return lSl;
  }
  const int SIZE = 16;
  const double ABS = .001;
  class Nero_BPF{
```

```
const double ABS = .001;
class Nero_BPF{
private:
      double W[SIZE][SIZE];  // matrix of weightes
      double *y;             // output signal
      double *x;             // input signal
public:
      Nero_BPF(){
      }
      void Teach(double *);   // procedure teach net
      void Engine();              // function summ weigthes
      void Read_Matrix();
      void Write_Matrix();
};

void Nero_BPF :: Engine(){
for (int i = 0; i < SIZE; i++){
      for (int j = 0; j < SIZE; j++){
            y[i] = y[i] + x[j]*W[i][j];
      }
}}
void Nero_BPF :: Teach(double *d){
this->Engine();
```

```
randomize();
for (int i = 0; i < SIZE; i++){
      for (int j = 0; j < SIZE; j++){
            W[i][j] = 10*random(1);
      }
}
for (i = 0; i < SIZE; i++){
      while (fabs(y[i] - d[i]) < ABS){
            this->Engine();
            for ( int j = 0; j < SIZE; j++){
                        W[i][j] = W[i][j] + x[j]*(d[i] - y[i]);
            }
      }
}}
void Nero_BPF :: Read_Matrix(){
FILE *fp;
char Url[] = "123";
if ( (fp = fopen(Url,"r")) == NULL){
      printf("cann't open %s",Url);
      exit;
}
char Matrix[] = "";
fscanf(fp,"%s",Matrix);           //read from file MATRIX
char tmp[] = "";
int LENGTH = 0;
LENGTH = strlen(Matrix);
for (int k = 0; k < LENGTH; k++){

    if (Matrix[k] != " "){
          tmp = strcat(tmp,Matrix[k]);
    }
    if (tmp != ""){
          W[i][j]
    }}
fclose(fp);}
```

Подпрограмма скремблирования приведена в примере.

Пример: Подпрограмма быстрого преобразования Фурье

```
/* Константы для БПФ */
#define   N                   1024
#define   Ndiv2               (N/2)
#define   log2N       10
#define Mod_Value     64
#define Refft_Bitrev    0x0001
#define Inputreal_Bitrev        0x0009

.section/data data1;
.VAR    twid_imag [Ndiv2] = "twid_sin.dat";
.VAR    groups = 1;
.VAR    node_space = Ndiv2;

.section/data seg_buf1;
.VAR    Inputreal [N+2] = "inreal.dat";

.section/data seg_buf2;
.VAR    Refft[N+2];

.section/pm data2;
.VAR/init24   twid_real [Ndiv2] = "twid_cos.dat";
.VAR   Inputimag [N+2] = "inimag.dat";

.section/pm IVreset:
      JUMP start; NOP; NOP;


/* Код программы */
.section/pm program;
start:
        dmpg2 = page(twid_real);
        M0 = 0;
        L0 = length(twid_imag);
        AX1 = twid_imag;
        REG(b0) = AX1;
        M1 = 1;
        L1 = 0,
        M4 = 0;
        L4 = length(twid_real);
        AX1 = twid_real;
        REG(b4) = AX1;
        M5 = 1;
        L5 = 0;
```

```
M6 = -1;
L6 = 0;
L2 = 0;
L3 = 0;
L7 = 0;

CNTR = log2N -1;

DO stage_loop UNTIL CE;          /* Вычисление БПФ */
        I0 = twid_imag;
        I1 = Inputreal;
        I2 = Inputreal;
        I4 = twid_real;
        I5 = Inputimag;
        I6 = Inputimag;
        SI = DM(groups);
        CNTR = SI;
        SR = LSHIFT SI BY 1(LO);
        DM(groups) = SR0;
        SI = DM(node_space);
        M2 =SI;
        M7 =SI;
        MODIFY(I1,M2);
        MODIFY(I5,M7);

        DO group_loop UNTIL CE;
                        MY0 = PM(I4,M5), MX0 = DM(I1,M0);
                MR = MX0*MY0(SS), MX1 = PM(I5,M4);
                MY1 = DM(I0,M1);
                CNTR = SI;                              /* CNTR = длина "бабочки" */

                        DO bfly_loop UNTIL CE;
                                MR = MR-MX1*MY1(RND), AY0 = DM(I2,M0);
                                AR = MR1+AY0, AX1 = PM(I5,M5);
                                DM(I2,M1) = AR, AR = AY0-MR1;
                                MR = MX0*MY1(SS), DM(I1,M1) = AR;
                                MR = MR+MX1*MY0(RND), AY1 = PM(I6,M4), MX0 =
DM(I1,M0);
                                AR = MR1+AY1, MX1 = PM(I5,M6);
                                PM(I6,M5) = AR, AR = AY1-MR1;
                        bfly_loop:
                                MR = MX0*MY0(SS), PM(I5,M5) = AR;
                                MY0 = PM(I5,M7), MX0 = DM(I1,M2);
                group_loop:
                        MY0=PM(I6,M7), MX0=DM(I2,M2);
                        SR=ASHIFT SI BY -1 (LO);
        stage_loop:
                DM(node_space)=SR0;

        I0 = twid_imag;
        I1 = Inputreal;
```

```
I2 = Inputreal;
M2 = 2;
I3 = Refft_Bitrev;
M3 = Mod_Value;
I4 = twid_real;
I5 = Inputimag;
I6 = Inputimag;
M6 = 2;
MODIFY(I1,M1);
MODIFY(I5,M5);
MY0 = PM(I4,M5), MX0 = DM(I1,M2);
MR = MX0*MY0(SS), MX1 = PM(I5,M6);
MY1 = DM(I0,M1);
CNTR = Ndiv2;
DO last_loop UNTIL CE;
        MR = MR-MX1*MY1(RND), AY0 = DM(I2,M2);
        AR = MR1+AY0, AY1 = PM(I6,M4);
        ENA BIT_REV;
        DM(I3,M3) = AR, AR = AY0-MR1;      /* Читаем реальные части */
        MR = MX0*MY1(SS), DM(I3,M3) = AR;
        DIS BIT_REV;
        MR = MR+MX1*MY0(RND), MY0 = PM(I4,M5), MX0 = DM(I1,M2);
        AR = MR1+AY1, MX1 = PM(I5,M6);
        PM(I6,M5) = AR, AR = AY1-MR1;
        MY1 = DM(I0,M1);
last_loop:
        MR = MX0*MY0(SS), PM(I6,M5) = AR;

I3 = Inputreal_Bitrev;
M3 = Mod_Value;


      I5 = Inputimag;
      ENA BIT_REV;
      CNTR = N;
      DO bit_rev_imag UNTIL CE;
            AX0 = PM(I5,M5);          /* Реальный части */
      bit_rev_imag:
            DM(I3,M3) = AX0;
            DIS BIT_REV;

looping:    JUMP looping;
```

Задание:

1. Измерить время выполнения программы, приведенной в примере, и количество MIPS, которые занимает эта программа.
2. Написать и отладить на языке С программу БПФ.
3. Оформить отчет

Приложение

1. Порядок проведения лабораторных работ

1. Получить у преподавателя номера выполняемых работ и номер.
2. Подключить EZ-KIT к COM-порту компьютера
3. Подключить разъем питания
4. Нажать клавишу Reset
5. По листингу приведенному в каждой лабораторной работе разработать блок-схему реализуемого алгоритма
6. Ввести и отладить программу, приведенную в листинге
7. Оценить результаты работы программы (для работ №№1-3 оценка осуществляется путем просмотра соответствующих регистров, для работ №№4-5 оценка производится по векторам, которые необходимо получить у преподавателя)
8. Отключить EZ-KIT от компьютера
9. Отключить питание EZ-K.IT
10. Оформить отчет

2. Содержание отчета

Отчет студента должен соответствовать ГОСТ и должен содержать:

1. Краткий теоретический конспект с ответами на контрольные вопросы по разделам
2. Листинги всех программ разработанных или запущенных студентом
3. Методика оценки корректности работы программы: входные - выходные вектора, оценка достоверности полученных результатов.

3. Варианты заданий для лабораторных работ №№2-3

№	Порядок фильтра	Коэффициенты фильтра	Вид входного сигнала
1	8	{0.83 0.51 0.44 0.28 0.32 0.33 0.45 0.78}	Треугольный F = 20кГц
2	10	{0.65 0.51 0.66 0.24 0.32 0.45 0.46 0.78 0.99 0.43}	Треугольный F = 40кГц
3	9	{0.66 0.24 0.32 0.45 0.46 0.78 0.99 0.45 0.78}	Прямоугольный F = 20кГц
4	11	{0.83 0.51 0.44 0.24 0.32 0.45 0.46 0.78 0.28 0.32 0.33}	Прямоугольный F = 40кГц
5	8	{0.44 0.24 0.32 0.45 0.46 0.78 0.45 0.78}	Треугольный F = 20кГц
6	10	{0.46 0.78 0.28 0.32 0.28 0.32 0.33 0.45 0.78 0.24}	Треугольный F = 40кГц
7	9	{0.78 0.28 0.32 0.28 0.32 0.33 0.33 0.45 0.78}	Прямоугольный F = 20кГц
8	11	{0.46 0.78 0.28 0.32 0.32 0.45 0.46 0.78 0.28 0.32 0.33}	Прямоугольный F = 40кГц
9	8	{0.44 0.24 0.32 0.45 0.46 0.78 0.45 0.78}	Треугольный F = 20кГц
10	10	{0.65 0.51 0.66 0.24 0.32 0.45 0.46 0.78 0.99 0.43}	Треугольный F = 40кГц
11	9	{0.66 0.24 0.32 0.45 0.46 0.78 0.99 0.45 0.78}	Прямоугольный F = 20кГц
12	11	{0.28 0.32 0.32 0.45 0.46 0.45 0.46 0.78 0.28 0.32 0.33}	Прямоугольный F = 40кГц
13	8	{0.45 0.46 0.45 0.46 0.78 0.28 0.32 0.33}	Треугольный F = 20кГц
14	10	{0.65 0.51 0.66 0.24 0.32 0.45 0.46 0.78 0.99 0.43}	Треугольный F = 40кГц
15	9	{0.66 0.24 0.32 0.45 0.46 0.78 0.99 0.45 0.78}	Прямоугольный F = 20кГц
16	11	{0.83 0.51 0.65 0.51 0.66 0.24 0.32 0.78 0.28 0.32 0.33}	Прямоугольный F = 40кГц
17	8	{0.32 0.32 0.45 0.46 0.78 0.28 0.32 0.33}	Треугольный F = 20кГц
18	10	{0.65 0.51 0.51 0.66 0.24 0.32 0.45 0.78 0.99 0.43}	Треугольный F = 40кГц
19	9	{0.32 0.45 0.46 0.78 0.99 0.78 0.99 0.45 0.78}	Прямоугольный F = 20кГц
20	11	{0.51 0.66 0.24 0.32 0.78 0.45 0.46 0.78 0.28 0.32 0.33}	Прямоугольный F = 40кГц

Московский ордена Ленина, ордена Октябрьской Революции и ордена Трудового Красного Знамени.
ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Н. Э. БАУМАНА
Факультет: Информатики и систем управления
Кафедра: Проектирование и производство ЭА (ИУ4)

ОТЧЕТ

По лабораторной работе №3
"Реализация БИХ фильтра"

По курсу: Цифровые сигнальные процессоры

Студент: _____
(фамилия, инициалы) (индекс группы)

Преподаватель: _____
(фамилия, инициалы)

отметка о защите	дата защиты

Москва
2003

Содержание

Задание

Наименование разработки: Реализация БИХ фильтра.

Цель работы: Исследование принципов построения БИХ фильтра на моделях, реализованных на языке С (с программно заданным сигналом) и с использованием EZ-KIT - 2189M (исходный сигнал получается с микрофона).

Решаемые задачи: Разработать БИХ фильтр

Форма отчетности: По результатам разработки предоставляется: отчет (файл otchet.doc), исходный текст программы (файлы iir.h, iir.c, main.c, исходный ассемблерный файл ППО для EZ-KIT - 2189M).

1. Математическое обеспечение

БИХ фильтром или рекурсивным фильтром называется фильтр, осуществляющий взвешенное суммирование ряда отсчетов входного сигнала и взвешенное суммирование выходных отсчетов:

$$y_i=a_0x_i+a_1x_{i-1}+...+a_nx_{i-n}+ b_0y_i+b_1y_{i-1}+...+b_my_{i-m}$$

Таким образом, рекурсивный фильтр - это фильтр с бесконечной импульсной характеристикой $\{h_k\}$.

Структура фильтра изображена на рис. 1.

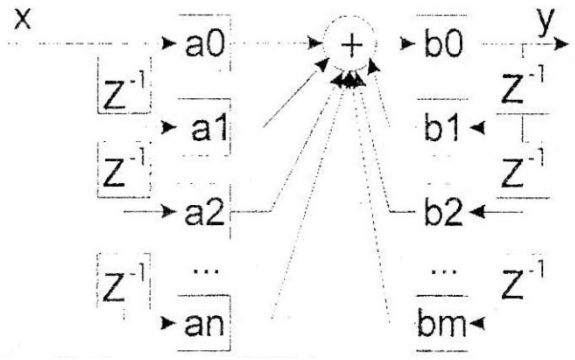

Рис. 1. Структура БИХ-фильтра

Основными элементами фильтра служат устройства задержки отсчетных значений на один интервал дискретизации **(Z^{-1})**, а также масштабные звенья, выполняющие в цифровой форме операции умножения на соответствующие коэффициенты.

С выходов масштабных звеньев сигналы поступают на сумматор, на выходе которого образуется отсчет выходного сигнала.

2. Лингвистическое обеспечение

В качестве лингвистического обеспечения для разработки модели использован язык С, для разработки ППО под EZ-KIT - 2189M соответствующая версия ассемблера.

3. Программное обеспечение

3.1 Использование библиотечных функций

Для реализации данной программы были использованы следующие стандартные библиотеки:

stdio.h - модуль содержащий функции стандартного ввода-вывода

3.2 Алгоритм программы согласно ЕСПД.

3.3 Листинг программы для ADSP-21xx.

Таблица 1. Исходный листинг программы.

```
/*****************************************************************
Файл: iir.h
Описание: БИХ фильтр
        Вход:
                pInDelay – линия задержки входных сигналов
                pOutDelay - линия задержки выходных сигналов
                pAFilter, pBFilter - коэффициенты
                sample – входной отсчет
*****************************************************************/
#ifndef IIR_H_INCLUDED
#define IIR_H_INCLUDED

extern int iir(int *pInDelay, int *pOutDelay, int *pAFilter, int *pBFilter)

#endif // IIR_H_INCLUDED
/*****************************************************************
Файл: iir.c
Описание: БИХ фильтр
        Вход:
                pInDelay – линия задержки входных сигналов
                pOutDelay - линия задержки выходных сигналов
                pAFilter, pBFilter - коэффициенты
                sample – входной отсчет
*****************************************************************/
#define ORDERA 10
#define ORDERB 10
int iir(int *pInDelay, int *pOutDelay, int *pAFilter, int *pBFilter)
{
        int i;
        int res;
        for(i = 0; i < ORDERB; i++)
                res += (pInDelay[i] * pBFilter[i]) << 1;
        for(i = 0; i < ORDERA; i++)
                res += (pOutDelay[i] * pAFilter[i]) << 1;
        return res;
}
```

4. Оценка результатов выполнения программы

Описание входных - выходных векторов, полученных значений, их качественная оценка.

5. Методическое обеспечение.

Программа предназначена для использования на отладочном комплекте EZ-KIT-2189M в среде Visual DSP.

Состав комплекта:

- Документация по разработке (отчет) - файл otchet.doc.
- Исходный текст программы - файлы parser.cpp, parser.h, stack.cpp, stack.h, makefile.

Список использованных источников

1. Б.Страуструп, Язык программирования C++, 3-е изд./Пер. с англ. - СПб.; М.: «Невский Диалект» - «Издательство БИНОМ», 1999 г. - 991 с., ил.

2. Intel Architecture Software Developer's Manual, Floating-Point Unit (fpu.pdf)

3. Mike Eddy, Coprocessor (09LMARFC05.pdf)

Список использованных источников

1. С. Марков Цифровые сигнальные процессоры.-М.: МИКРОАРТ, 1996.-144 с.
2. Бортовая и промышленная электроника- АО Каскод, 1997. - 88 с.
3. Designer's reference manual- Analog Devices, Inc, 1996.
4. DSP/MSP products reference manual- Analog Devices, Inc, 1995.
5. High speed design techniques-Analog Devices, Inc, 1996.
6. ADSP-21000 Family. Application Handbook- Analog Devices, Inc, 1994.
7. ADSP-2100 Family. Application Handbook- Analog Devices, Inc, 1994.
8. ADSP-2106x Share User's Manual- Analog Devices, Inc, 1996.
9. ADSP21020 User's manual- Analog Devices, Inc, 1995.
10. Designer’s CD reference manual- Analog Devices, Inc, CD-ROM, 1996.
11. TMS320C8x (MVP) online reference- Texas Instruments, Inc, CD-ROM,1995.
12. TMS320C6x Digital Signal Processors- Texas Instruments, Inc, CD-ROM,1997.
13. Digital Signal Pricessing. A Multimedia reference Guide- Texas Instruments, Inc, CD-ROM. 1994.
14. В.А. Шахнов, А.И.Власов, А.С. Кузнецов, Ю.А.Поляков Нейрокомпьютеры: архитектура и реализация. - М.: Машиностроение. 2000. - 24 с. (Библиотечка журнала информационные технологии №9).
15. Б.Страуструп, Язык программирования С++, 3-е изд./Пер. с англ. - СПб.; М.: «Невский Диалект» - «Издательство БИНОМ», 1999 г. - 991 с., ил.
16. А.А.Адов Спектральное оценивание диагностических сигналов с использованием нейросетевых парадигм// Сборник научных трудов. Том 2. Молодежная научно-техническая конференция «Наукоемкие технологии и интеллектуальные системы - 2003». - Москва, МГТУ им.Н.Э.Баумана, 16-17 апреля 2003 г. С.62-69.
17. Стешенко В.Б. Современные алгоритмы ЦОС: пути реализации и перспективы применения.
18. Е. Угрюмов. Цифровая схемотехника.

Список стандартов

ГОСТ 2.701 -84 Схемы. Виды и типы. Общие требования к выполнению

ГОСТ 2.114-95 Технические условия

ГОСТ 15.002-80 Схемы алгоритмов и программ. Правила выполнения

ГОСТ 19.003-80 Схемы алгоритмов и программ. Обозначения условные графические

ГОСТ 19.004-80 Термины и определения

ГОСТ 19.101-77 Виды программ и программных документов

ГОСТ 19.102-77 Стадии разработки

ГОСТ 19.103-77 Обозначения программ и программных документов

ГОСТ 19.104-78 Основные надписи

ГОСТ 19.105-78 Общие требования к программным документам

ГОСТ 34.201-89 Информационная технология. Виды, комплектность и обозначения документов при создании автоматизированных систем

ГОСТ 34.603-92 Информационная технология. Виды испытаний автоматизированных систем

ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом

ГОСТ 19.201-78 Техническое задание, требования к содержанию и оформлению

ГОСТ 19.202-78 Спецификация. Требования к содержанию и оформлению

ГОСТ 19.301-79 Программа и методика испытаний. Требования к содержанию и оформлению

ГОСТ 19.401-78 Текст программы. Требования к содержанию и оформлению

ГОСТ 19.402-78 Описание программы

ГОСТ 19.403-79 Ведомость держателей подлинников

ГОСТ 19.503-79 Руководство системного программиста. Требования к содержанию и оформлению

ГОСТ 19.504-79 Руководство программиста. Требования к содержанию и оформлению

ГОСТ 19.506-79 Описание языка. Требования к содержанию и оформлению

ГОСТ 19.508-79 Руководство по техническому обслуживанию. Требования к содержанию оформлению.