БИБЛИОТЕКА ИНЖЕНЕРА-РАЗРАБОТЧИКА ЭЛЕКТРОННЫХ СИСТЕМ

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

Московский государственный технический университет
имени Н.Э.Баумана

УЧЕБНОЕ ПОСОБИЕ

Э.В.МЫСЛОВСКИЙ, А.И.ВЛАСОВ, К.А.МЕНЬШОВ

ЦИФРОВЫЕ СИГНАЛЬНЫЕ ПРОЦЕССОРЫ

ЧАСТЬ 3:
ЦИФРОВЫЕ СИГНАЛЬНЫЕ ПРОЦЕССОРЫ С ПЛАВАЮЩЕЙ ТОЧКОЙ
СЕМЕЙСТВА ADSP2106x

Москва
МГТУ им.Н.Э.Баумана
2003

Московский государственный технический университет имени Н.Э. Баумана

Э.В.Мысловский, А.И.Власов, К.А.Меньшов

ЦИФРОВЫЕ СИГНАЛЬНЫЕ ПРОЦЕССОРЫ

ЧАСТЬ 3:
ЦИФРОВЫЕ СИГНАЛЬНЫЕ ПРОЦЕССОРЫ С ПЛАВАЮЩЕЙ ТОЧКОЙ СЕМЕЙСТВА ADSP2106x

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: доцент Шитько Ю.М., МГАПИ

Мысловский Э.В., Власов А.И., Меньшов К.А.

ЦИФРОВЫЕ СИГНАЛЬНЫЕ ПРОЦЕССОРЫ С ПЛАВАЮЩЕЙ ТОЧКОЙ СЕМЕЙСТВА ADSP2106x. Учебное пособие. Часть 3. - 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 внимание уделено цифровым сигнальным процессорам фирмы Analog Devices, семейства 2106x. 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

Данное учебное пособие является продолжением цикла учебных пособий, освещающих вопросы цифровой обработки сигналов (ЦОС) и применения цифровых сигнальных процессоров и посвящено рассмотрению принципов разработки средств обработки сигналов с плавающей арифметикой. В качестве средств, для иллюстрации методов ЦОС рассматриваются DSP фирмы Analog Devices семейства ADSP2106x.

Корни цифровой обработки сигналов (ЦОС) уходят в шестидесятые и семидесятые года, к появлению цифровых компьютеров. В то время компьютеры были очень дорогими и ЦОС ограничивалась несколькими особо критичными приложениями. Первые разработки были сделаны в четырех ключевых областях: радарах и сонарах (т.к. здесь были затронуты вопросы национальной безопасности), разведке нефтяных месторождений (из-за большой материальной выгоды), исследованиях космоса (из-за уникальности получаемых данных) и медицине (т.к. здесь речь шла о человеческих жизнях).

Революция, произошедшая в мире персональных компьютеров в восьмидесятых и девяностых подтолкнула ЦОС к быстрому развитию. Кроме военных и государственных нужд ЦОС прочно заняла коммерческий рынок в таких областях как: мобильная связь, проигрыватели компакт-дисков, электронная голосовая почта и многих других. В начале восьмидесятых ЦОС изучалась инженерачи-электронщиками на выпускных курсах, десятилетие спустя она стала частью программы на средних курсах. Сегодня ЦОС входит в ряд основных фундаментальных навыков требующихся инженерам во многих областях. Вот лишь некоторые из них: системы управления технологическими и телематическими комплексами, космическая техника (улучшение фотографий полученных из космоса, сжатие данных, обработка сигналов принятых от космических зондов), медицина (диагностическое сканирование, анализ электрокардиограмм, хранение и воспроизведение медицинских изображений), коммерческое применение (сжатие звука и изображений для проведения презентаций, спецэффекты в кино, организация видеоконференций), телефония (сжатие голосовых потоков, уменьшение «эффекта эха», мультиплексирование сигналов, фильтрация), военное применение (радары, сонары, защищенные линии связи), промышленность (геологическая разведка, контроль процессов, неразрушающий контроль изделий), наука (запись и анализ сейсмических сигналов, извлечение полезных данных из сигналов, спектральный анализ, моделирование).

Стимулом для увеличения применения цифровых сигнальных процессоров послужило бурное развитие телекоммуникаций, в частности, мобильной связи и Интернет-технологий (в том числе, IP-телефонии). В мире существует примерно около

миллиарда телефонных номеров. До шестидесятых годов двадцатого столетия аналоговое соединение между двумя номерами осуществлялось с помощью механических переключателей и усилителей. Причем каждое такое соединение требовало пары проводов. Колоссальные затраты на прокладку и обслуживание телефонных коммуникаций заставили крупные телефонные компании искать новые пути. Стремление разработчиков новой аппаратуры рационально использовать существующие физические каналы связи привело к созданию аппаратуры цифрового уплотнения, позволяющей передавать по одному физическому каналу связи нескольких независимых голосовых каналов. Ключевыми элементами этих технологий являются цифровые сигнальные процессоры.

Первые серийные средства компьютерной телефонии появились более 10 лет назад, когда развитие технологии обработки речи позволило преодолеть критический рубеж по критерию функциональность/стоимость и выйти на коммерческие решения. В первую очередь, компьютерная телефония получила стимул к развитию благодаря снижению стоимости цифровых сигнальных процессоров и повышению их производительности, поскольку функциональные возможности устройств обработки речи в основном определяются производительностью ЦСП.

Однако область применения ЦСП в телекоммуникациях простирается далеко за пределы IP-телефонии. Летом 2001 г. корпорация Cisco Systems дополнила свою серию шлюзов AS5000 архитектурой Any Service, Any Port (ASAP), позволяющей использовать эти серверы доступа для пропуска трафика речи, данных, факсимильных сообщений и беспроводных коммуникаций. ASAP предоставляет возможность использовать правила и учетные записи для задания соответствия приложений универсальным портам и наоборот, а также для сбора тарификационных данных. По словам представителей компании, данное усовершенствование стало возможным благодаря развитию технологии цифровых сигнальных процессоров. Они преобразуют аналоговый речевой или видеосигнал в поток мультимедиа-данных и составляют универсальную аппаратную базу для работы с цифровыми сигналами различной природы. Небольшие поставщики услуг смогут использовать ASAP-шлюзы Cisco для предоставления новых сервисов, таких как доступ по карточкам или беспроводная связь, а крупные операторы - для упрощения конфигурации своих сетей. Для предприятий новые устройства упростят задачу выполнения таких функций, как автоматизация разветвленных служб сбыта и централизованное администрирование средств связи (ноутбуков и мобильных телефонов) работников.

Непосредственное влияние оказывают цифровые сигнальные процессоры и на технологии мобильной связи. В 2000 г. ведущие специалисты компаний-гигантов AT&T и Nortel Networks предприняли первую попытку определить понятие сетей мобильной связи 4-го поколения и представили документацию по этому вопросу на конференции Rawcon (Radio and Wireless Conference) в Денвере.

Согласно определению, данному специалистами компаний, сети мобильной связи следующего поколения будет отличать сверхвысокая скорость передачи данных - от 20 Мбит/с и более. На конференции в Нью-Йорке AT&T продемонстрировала работу своей асимметричной сети, названной "сетью беспроводной связи 4-го поколения". Доступ в сеть соединяет в себе технологии EDGE (Enhanced Data Rates for GSM Evolution) - для восходящего трафика, и ODFM (ортогональное мультиплексирование с разделением частот) - для нисходящего трафика. Такая технология позволит с мобильного терминала скачивать потоковое аудио и видео.

В феврале 2003 г. состоялась официальная презентация телефона-игровой консоли Nokia N-Gage. Это событие могло бы показаться не слишком значительным, если бы не широкий резонанс, вызванный появлением телефонов, несомненным для многих достоинством которых является наличие функций игровой консоли. В ответ на это событие под эгидой Texas Instruments был создан консорциум Cellular Media, который займется разработкой открытого интерфейса API (application programming interface) трехмерной графики для мобильных телефонов OpenGL ES. В Cellular Media вошла группа инженеров TI, сформированная как отдельное подразделение в 2002 году и работающая над созданием аппаратных ускорителей для 3D графики для мобильных телефонов, не оснащенных мощным процессором. Дополнительно, эти же ускорители будут применяться для поддержки MMS в телефонах начального и среднего уровня.

Очевидно, что в настоящий момент приоритетной областью применения цифровых сигнальных процессоров являются телекоммуникации, в том числе IP-телефония, мобильные устройства связи и аппаратура доступа в Интернет. Таким образом, именно от уровня телекоммуникационных технологий в ближайшие годы будет зависеть коммерческий потенциал цифровых сигнальных процессоров, которые, в свою очередь, определят спектр возможнойстей различных телекоммуникационных услуг. Поэтому технологии цифровых сигнальных процессоров будут развиваться в первую очередь с учетом их применения в телекоммуникационных средах для передачи больших объемов мультимедийной информации.

Данное учебное пособие посвящено цифровому сигнальному процессору ADSP-21061, являющемуся на сегодняшний день одним из наиболее доступных для применения процессоров семейства 2106x, и позволяющему строить эффективные системы обработки звука, речи, скоростные модемы, цифровые приемники, системы радиолокации, базовые станции систем мобильной связи, системы обработки изображений (видимых и ультразвуковых), высокоэффективные системы измерения и управления и многое другое.

Используемые условные обозначения:

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.

ВВЕДЕНИЕ: ОСНОВНЫЕ ПОНЯТИЯ, ОПРЕДЕЛЕНИЯ И ТЕРМИНЫ

По типу обрабатываемых данных цифровые сигнальные процессоры (ЦСП) 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, - высокая цена микропроцессоров.

В настоящее время стал популярен и другой подход к получению высокой производительности. Большое количество транзисторов на кристалле может быть использовано для создания симметричной мультипроцессорной системы с более простыми процессорами, обрабатывающими целочисленные операнды. Примерами таких, так называемых медийных процессоров, служат Mediaprocessor компании Micro Unity, Trimedia компании Philips, Mpact Media Engine компании Chromatic Research и т.д. Эти процессоры создавались, исходя из потребности обработки в реальном времени видео- и аудиоинформации в мультимедийных ПК, бытовых радиоэлектронных приборах. В связи с более простой схемотехникой по сравнению с универсальными сигнальными процессорами стоимость медийных процессоров достаточно низка (порядка $100), а значение показателя "производительность/стоимость" на два-три порядка больше. Пиковое значение производительности медийных процессоров составляет несколько миллиардов целочисленных операций в секунду.

Отличительной особенностью задач цифровой обработки сигналов 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. В то же время для сигнальных процессоров характерным является наличие аппаратного умножителя, позволяющего выполнять умножение двух чисел за один командный такт.

Существующие две основные архитектуры построения вычислительных систем - Фон Неймана (Von Neumann) и Гарвардская (Harvard). Архитектура Фон Неймана (рис. 1) применяется уже более 40 лет и предусматривает размещение и программ, и данных в одной и той же области памяти. Поэтому только пространство данных или пространство программ может быть доступно в одном цикле обращения к памяти. Данная архитектура обладает рядом положительных черт. Она является более дешевой, требует меньшего количества выводов шины. Архитектура Фон Неймана является более простой в использовании, так как программист может размещать и команды и данные в любом месте свободной памяти.

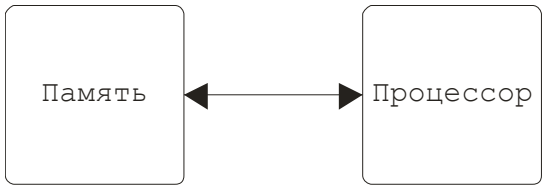


Рис. 1. Структурна схема архитектуры Фон Неймана

Гарвардская архитектура (рис. 2) разделяет пространства памяти данных и программ, предусматривая раздельные шины доступа к каждой из них. Это обеспечивает доступность и данных, и программ в одном цикле выполнения операций процессором, что увеличивает общую скорость обработки. Гарвардская архитектура выделяет одну шину для выборки инструкций (шина адреса), а другую для выборки операндов (шина данных). Но для выполнения ЦСП операций этого недостаточно, так как в основном все они используют по два операнда. Поэтому Гарвардская архитектура применительно к цифровой обработке сигналов использует шину адреса и для доступа к данным. Гарвардская архитектура требует наличия двух шин памяти. Это значительно повышает стоимость производства чипа. Так, например, ЦСП процессор, работающий с 32-битными

словами и в 32-битном адресном пространстве, требует наличия, по крайней мере, 64 выводов для каждой шины памяти, а в сумме получается 128 выводов. Это приводит к увеличению размеров чипа и к трудностям при проектировании схемы.

Рис. 2. Структурная схема Гарвардской архитектуры

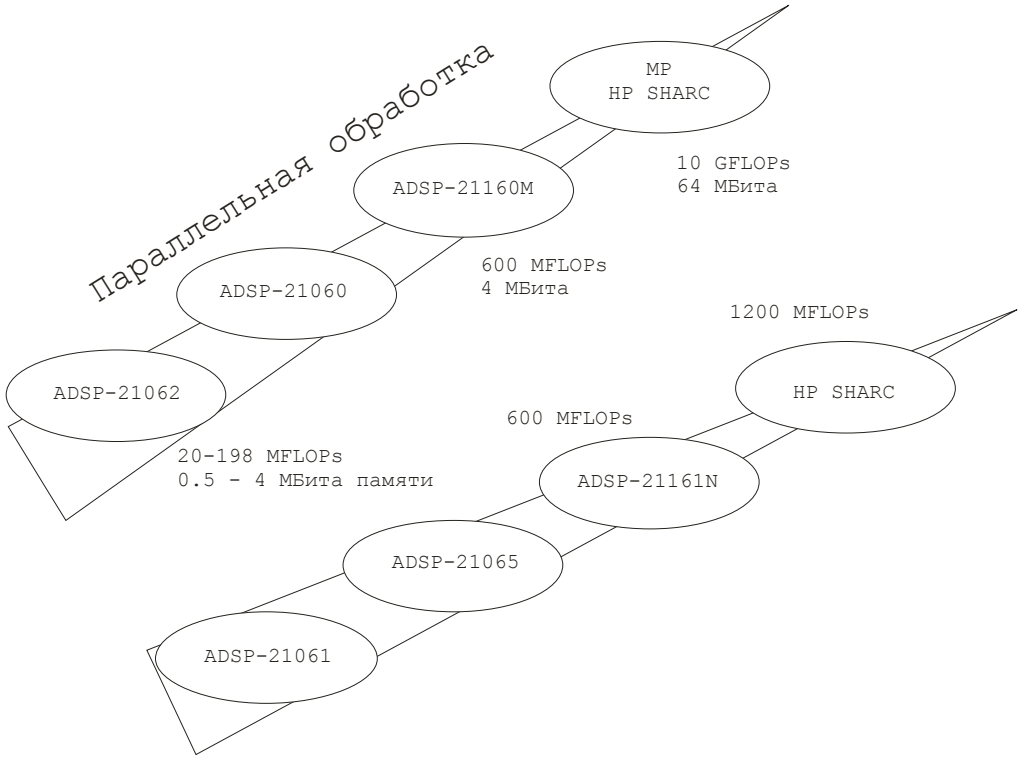


Рис. 3. Эволюция семейств цифровых сигнальных процессоров с плавающей точкой фирмы Analog Devices

В современных процессорах часто применяется модифицированная Гарвардская архитектура, когда для общения с внешней памятью используется один набор шин, в то время как на самом кристалле для увеличения быстродействия они разделены. В таком случае данная архитектура включает в себя кэш-память. Она может быть использована для хранения тех инструкций, которые будут использоваться вновь. При использовании кэш-памяти шина адреса и шина данных остаются свободными, что делает возможным выборку двух операндов. Такое расширение - Гарвардская архитектура плюс кэш - называют расширенной Гарвардской архитектурой или SHARC (Super Harvard ARChitecture). Такой подход минимизирует общую стоимость системы, сохраняя

преимущества Гарвардской архитектуры. Весьма типичной для ЦСП является ситуация, когда медленная внешняя память используется только для начальной загрузки программ и данных во внутреннюю быстродействующую статическую память.

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 выполнения. Особенно для чисел с плавающей точкой. Такие вычисления для многих процессоров общего назначения могут потребовать несколько сотен тактов.

Несомненно, в будущем, функции ЦОС будут все чаще применяться в обычных микропроцессорах и микроконтроллерах. Сильнейшими движущими факторами для таких изменений являются мультимедийные приложения и Интернет. Эти приложения развиваются так быстро, что через десять - двенадцать лет, вполне возможно, именно цифровые сигнальные процессоры и будут называться «обычными» микропроцессорами.

1. АРХИТЕКТУРА ЦИФРОВЫХ СИГНАЛЬНЫХ ПРОЦЕССОРОВ С ПЛАВАЮЩЕЙ ТОЧКОЙ

Рассматриваемый в данном разделе процессор Analog Devices ADSP-21061, построенный на основе модифицированной Гарвардской архитектуры, позволяет выполнять вычисления с высокой точностью (32/40 бит в формате с плавающей точкой и 32 бита с формате с фиксированной точкой) и обрабатывает почти все команды за один командный цикл.

В таблице 1 приведены технические характеристики некоторых процессоров семейства SHARC™.

Таблица 1. Технические характеристики процессоров SHARC™

Название	Производительность (MFLOPs)	Цикл (не)	ОЗУ, Кбит	Послед. поргы	Питание, В	Темп. диапа
ADSP-210651	180	16,6	544	2	3,3	К, РК
ADSP-21061	**100, 120**	**30, 25**	**1024**	**2**	**5**	**К, И**
ADSP-21061L	120, 133	25,22.7	1024	2	3,3	К, РК
ADSP-21062	100, 120	30,25	2048	2	5	к, и
ADSP-21062L	100, 120	30, 25	2048	2	3,3	к, РК
ADSP-21060	100, 120	30, 25	4096	2	5	К, и, в
ADSP-21060L	100, 120	30,25	4096	2	3,3	к, и, в
AD14060	480	25	16384	8	5	к, и, в

Условные обозначения: К – коммерческий (0 — +70)

РК - расширенный (-40 — +70)

И - индустриальный (-40 — +85)

В - военный (-55 — +125)

Процессор Analog Devices ADSP□21061, построенный на основе модифицированной Гарвардской архитектуры, позволяет выполнять вычисления с высокой точностью (32/40 бит в формате с плавающей точкой и 32 бита с формате с фиксированной точкой) и обрабатывает почти все команды за один командный цикл.

СРЕДСТВА РАЗРАБОТКИ, ТЕСТИРОВАНИЯ И ОТЛАДКИ ПРИЛОЖЕНИЙ ДЛЯ ЦИФРОВЫХ СИГНАЛЬНЫХ ПРОЦЕССОРОВ (ЦСП)

В справочной литературе, как правило, сначала описываются особенности архитектуры того или иного семейства сигнальных процессоров, в учебном же пособии мы посчитали целесообразным, сначала кратко описать принципы работы с отладочными средствами и лишь только после этого перейти к рассмотрению архитектуры конкретного семейства цифровых сигнальных процессоров и особенностях разработки программного обеспечения.

К средствам разработки и отладки приложений для ЦСП относятся различного вида симуляторы - средства, эмулирующие работу ЦСП, и аппаратно-программные модули - представляющие собой комплект соединенных между собой на плате устройств, в совокупности образующих полноценную систему цифровой обработки сигналов, программирование и управление которым, как правило, осуществляется с ПЭВМ через последовательный (COM) порт.

Симулятор ЦСП.

Симулятор представляет собой программу, моделирующую работу реального аппаратного обеспечения (цифрового сигнального процессора). При моделировании аппаратного обеспечения симулятор использует файл описания конфигурации для настройки области памяти и моделирования портов ввода/вывода. Прогон программы на симуляторе позволяет отладить её и оценить производительность перед запуском на реальном оборудовании.

После отладки программы на симуляторе можно использовать тестовую схему эмулятора для проверки работы на реальном процессоре.

В качестве симулятора для ADSP21061 может использоваться программа SHARC EZ-KIT Lite Simulator, внешний вид главного окна которой показан на рис. 4.

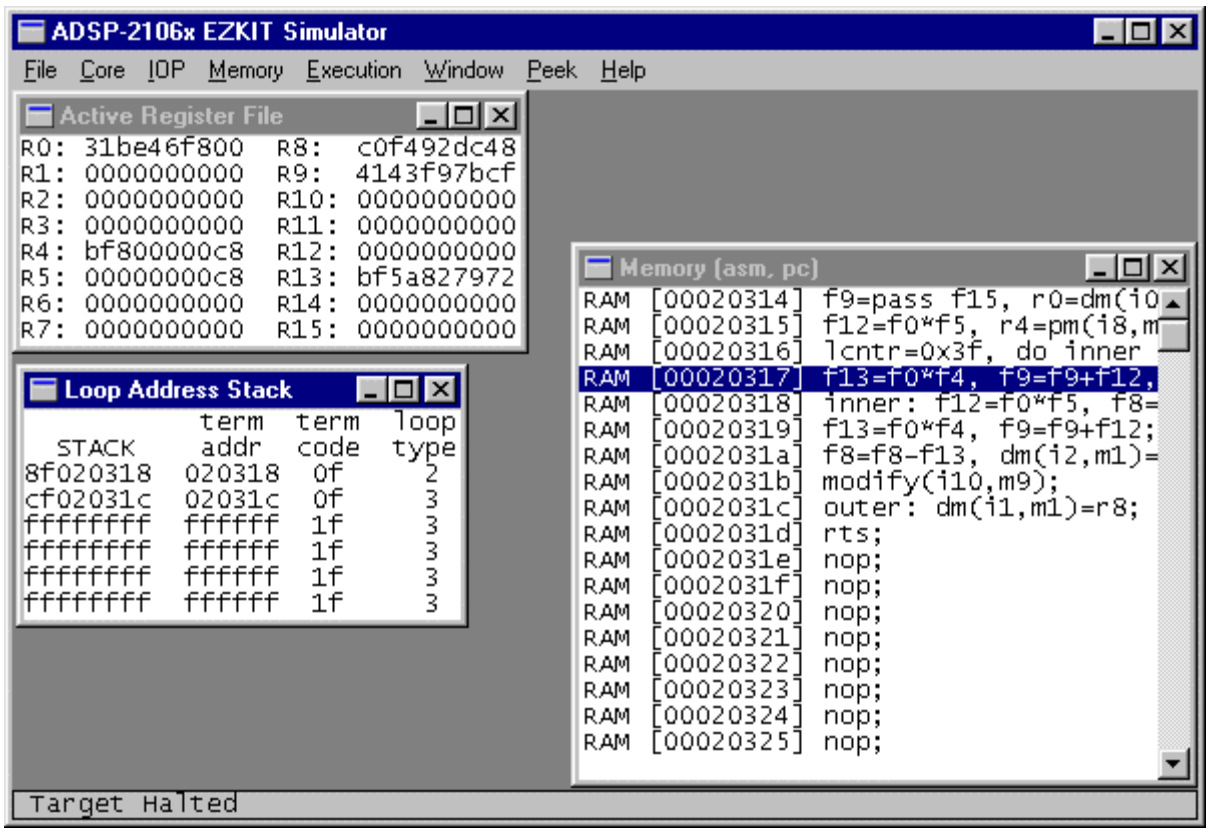


Рис. 4. Главное окно программы SHARC EZ_KIT Lite Simulator.

Для запуска симулятора из Windows следует открыть группу **SHARC EZ-KIT Lite** и выбрать пункт **EZ-KIT Lite Simulator**, При старте симулятор загружает стандартный файл описания конфигурации (21k.ach), или файл описания, путь к которому задан в конфигурационном файле симулятора wsim060.ini. Для загрузки .exe файла сначала необходимо очистить память симулятора командой меню **Execution|Simulator Reset**, а затем воспользоваться командой меню **File|Load File** для загрузки нового файла.

Симулятор позволяет открывать окна для контроля за различными элементами «виртуальной системы». Эти элементы поделены на категории в строке меню: **Core** (Ядро), **IOP** (Ввод/вывод) и **Memory** (Память). В окнах для каждого элемента выводятся значения тех или иных параметров, текущее состояние элементов, передаваемые или хранимые данные и т.д. в зависимости от конкретного объекта. Например, открыв окно обзора памяти командой **Memory|Memory**, можно увидеть окно, отображающее содержимое памяти «виртуальной системы».

Задание: Получение параметров конфигурации эмулятора.

Определить является ли конфигурация эмулятора стандартной (стандартная конфигурация эмулятора представлена в табл. 2)

Создание файла описания конфигураций

Каждая система на основе ADSP-210xx имеет уникальную аппаратную конфигурацию и может использовать различные количества доступной памяти. Файл описания конфигурации создается для описания конкретной конфигурации вашей системы, в том числе распределения памяти и описания отображаемых на память портов ввода/вывода. Линкер использует этот файл для распределения кода и данных в имеющемся объеме памяти; симулятор и эмулятор используют этот файл для точного моделирования вашей системы; а загрузчик - для определения адресов размещения кода и для инициализации данных.

Файл описания конфигурации имеет расширение **.ach**.

В файле описания архитектуры имена присваиваются

а) самой системе,

б) сегментам памяти

в) отображаемым на память портам ввода/вывода.

Имена сегментов затем используются в программе для сопоставления кода (и данных) тому или иному сегменту. Эти сопоставления затем обрабатываются ассемблером и линкером для определения точного положения сегментов в памяти.

Все имена должны быть уникальными. Имя - это строка из букв, цифр и знаков нижнего подчеркивания (_). Имя должно начинаться с буквы или нижнего подчеркивания. Любое имя, кроме имени сегмента может иметь длину до 32 символов. Имя сегмента не может быть длиннее восьми символов. Сами имена являются регистрово-независимыми, т.е. различия между строчными и прописными буквами не делается, однако такое различие может существовать для некоторых аргументов имен.

Некоторые имена зарезервированы в качестве ключевых слов, их список приведен ниже. Ключевые слова нельзя использовать как имена.

.BANK	DM	NORMAL	PM
.COMPILER	DM0	PARITYCK	PMO
.ENDSYS	DM1	PCRTS	PM1
.PROCESSOR	DM2	PGMODE	PORT
.REGISTER	DM3	PGSIZE	RAM
.SEGMENT	DMAPARITYCK	PGWAIT	RESERVED
.SYSTEM	DRAM		ROM
ADSP21020	EITHER		RTRTS

BEGIN	EMOVERLAY	SCRATCH
BOTH	END	SRAM
CHEAP	EXTERNAL	WIDTH
CINIT	INTERNAL	WTMODE
CIRC		WTSTATES
CSTACK		

Также в качестве имен нельзя использовать ключевые слова ассемблера, приведенные в следующих главах. Если в качестве имени используется ключевое слово, линкер выдает ошибку на этапе компиляции.

Файл описания архитектуры можно создать с помощью любого текстового редактора, не оставляющего в тексте служебных символов.

Ниже приведен пример файла описания архитектуры для процессора ADSP-21061 SHARC. В данном примере весь код программы и все данные хранятся во внутренней памяти.

Таблица 2. Пример файла описания конфигурации

```
.SYSTEM SHARC_EZKIT_Lite;

.PROCESSOR = ADSP21061;



!   ------------------------------------------------------------

!   Внутренняя память (Block 0)

!   ------------------------------------------------------------

.SEGMENT/RAM/BEGIN=0x00020000 /END=0x00020084 /PM/WIDTH=48        seg_rth;

.SEGMENT/RAM/BEGIN=0x00020085 /END=0x00020094 /PM/WIDTH=48        seg_init;

.SEGMENT/RAM/BEGIN=0x00020095 /END=0x000202ff /PM/WIDTH=48        seg_knlc;

.SEGMENT/RAM/BEGIN=0x00020300 /END=0x00021fff /PM/WIDTH=48        seg_pmco;

.SEGMENT/RAM/BEGIN=0x00023000 /END=0x00023fff /PM/WIDTH=32        seg_pmda;



!   ------------------------------------------------------------

!   Внутренняя память (Block 1)

!   ------------------------------------------------------------
```

```
.SEGMENT/RAM/BEGIN=0x00024000 /END=0x00025fff /DM/WIDTH=32        seg_dmda;

.SEGMENT/RAM/BEGIN=0x00026000 /END=0x00026fff /DM/WIDTH=32 /cheap seg_heap;

.SEGMENT/RAM/BEGIN=0x00027000 /END=0x00027e7f /DM/WIDTH=32        seg_stak;

.SEGMENT/RAM/BEGIN=0x00027e80 /END=0x00027fff /DM/WIDTH=32        seg_knld;


.ENDSYS;
```

? Определите конфигурацию системы, заданную в файле из таблицы 2.

На сколько сегментов разделена память программ?

На сколько сегментов разделена память данных?

Каков объем каждого сегмента в словах?

Каков объем каждого сегмента в битах?

Разработка исходных кодов программ

Исходные тексты программ на языке ассемблера могут содержать один или несколько сегментов кода или данных. Каждый файл с исходным текстом компилируется отдельно (при помощи ассемблера для семейства ADSP-21000) и затем полученные файлы с объектным кодом соединяются вместе (при помощи линкера для семейства ADSP-21000) образуя файл с выполняемым кодом.

На рис. 5 показаны входные и выходные файлы ассемблера.

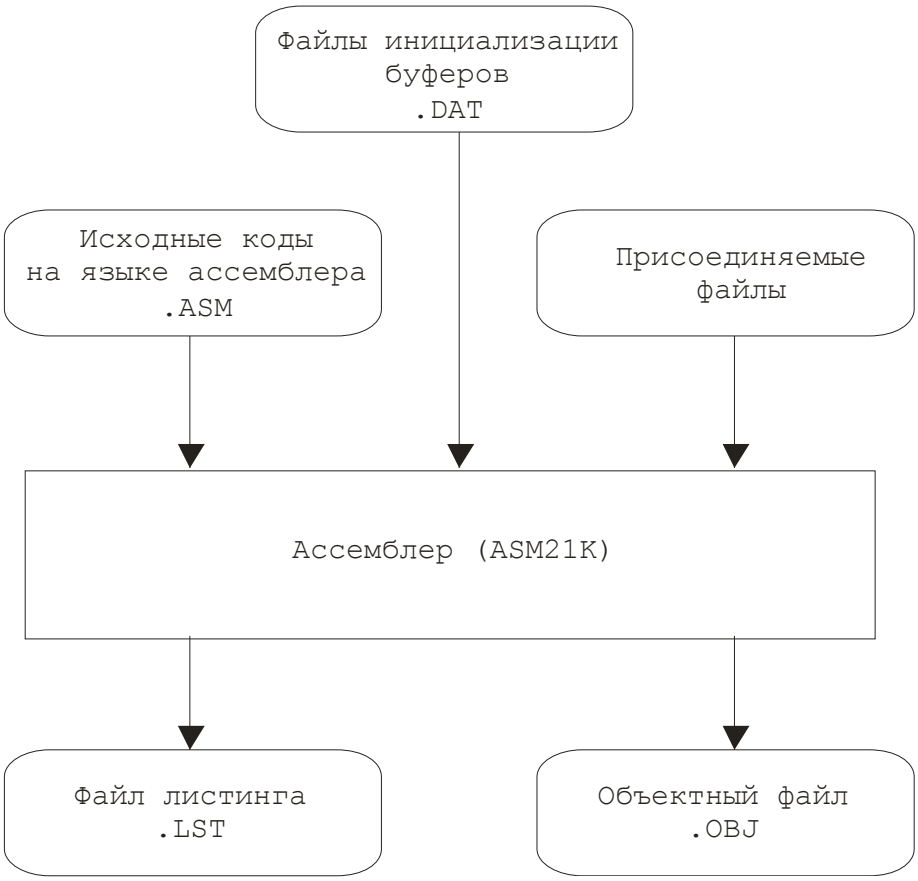


Рис. 5. Входные и выходные файлы ассемблера

В качестве входного файла ассемблера используется файл с исходным текстом программы. Он может содержать директивы предписывающие ассемблеру считать файл с расширением .DAT и/или подключать дополнительные файлы при помощи директивы #include. Ассемблер создает два вида выходных файлов: двоичный файл объектного кода (*.OBJ) и (при необходимости) текстовый **файл листинга** (*.LST). Объектный файл, содержащий код программы и определения меток, затем обрабатывается линкером. Файл листинга используется как справочный для лучшего понимания и документирования программ.

В исходных кодах программ и файлах данных можно использовать следующие форматы данных:

- десятичный с плавающей точкой
- шестнадцатеричный
- восьмеричный
- двоичный

Десятичный с плавающей точкой: Формат задается десятичным числом со знаком и десятичной точкой. Запись числа может содержать строчную или прописную букву E за которой указывается значение показателя степени (в десятичном виде со

знаком).

Примеры: 1.0, -1,701413e-18

Десятичный целочисленный: Формат задается десятичным числом со знаком без десятичной точки. Первая цифра не может быть нулем, в противном случае число будет трактоваться как восьмеричное (см. ниже).

Примеры: 5, -684

Когда непосредственные данные представленные в десятичном целочисленном формате записываются в 40-битовый регистр или в память, соответствующее 32-битное значение помещается в наиболее значимые биты. Младшие восемь бит заполняются нулями.

Шестнадцатеричный: Шестнадцатеричный формат задается шестнадцатеричными числами с префиксом 0x или H#. Могут использоваться и строчные и заглавные буквы. Примеры: 0x12345678, H#F00

Когда непосредственные данные, представленные в шестнадцатеричном формате, записываются в 40-битовый регистр или в память, соответствующее 32-битное значение помещается в наиболее значимые биты. Младшие восемь бит заполняются нулями. Значения, представленные более чем восемью шестнадцатеричными цифрами, обрезаются до восьми наиболее значимых. Исключение составляет случай когда шестнадцатеричное тесло состоящее из десяти цифр записывается в 40-битовый" регистр. Этот формат может использоваться для передачи расширенного 40-битиого значения с плавающей точкой стандарта IEEE.

Восьмеричный: Восьмеричный формат задается восьмеричными числами с предшествующим нулем.

Примеры: 0777, 01234567

Двоичный: Двоичный формат задается двоичными цифрами (0 и 1) с префиксом B#. Префикс может быть строчным или прописным.

Примеры: B#01110100, b#1010

Структура программы

Каждый оператор внутри файла может быть инструкцией или директивой. Инструкция компилируется в объектный код. Все инструкции ассемблера ADSP-210x0 заканчиваются точкой с запятой (;). Директива — это команда ассемблеру, влияющая на

процесс компиляции. Директивы начинаются со знака решетки (#) или с точки (.). Директивы, начинающиеся с точки, заканчиваются точкой с запятой (;).

Директивы препроцессора	#include “const.h” #define PI 3.14159
Сегмент данных в памяти данных	.PRECISION = 40; .ROUND ZERO; .SEGMENT/DM datal; .VAR fxd[10] = 1,2,3,4,5,6,7,8,9,0xA; .VAR flt[5] = PI,PI/2,1.0, 2.0,2.0/3.0; .VAR rad • • .ENDSEG;
Сегмент портов (проецируемый в программную память)	.SEGMENT/PM portseg; .PORT clk_in; .PORT tick; • • .ENDSEG;
Сегмент кода	.SEGMENT/PM codesegl; инструкция инструкция • • .ENDSEG;
Сегмент кода (условное ассемблирование)	.SEGMENT/PM codeseg2; инструкция инструкция • • • #ifdef somevar инструкция инструкция #endif инструкция • • • .ENDSEG;

Рис. 6. Структура программы на языке ассемблер для ADSP-21061

Обычно файл с исходным текстом содержит несколько или все элементы

программы, показанные на рис. 6. Программа состоит из различных сегментов и директив. Сегментация программы должна быть организована достаточно логично для лучшего понимания текста и для удовлетворения некоторым аппаратным ограничениям. Поскольку программная память и память данных используются для хранения данных и кода, для этих областей памяти применяется разбиение на сегменты. Память данных обычно содержит сегменты данных и отображаемые на область памяти порты. Программная память может содержать сегменты данных, отображаемые порты, и сегменты кода программы. В каждой области памяти может использоваться дополнительное сегментирование применяемое для логического объединения элементов или для удовлетворения аппаратных ограничений.

Файл с текстом программы обычно начинается с одной или нескольких директив для настройки среды ассемблера. Типичные директивы состоят из стандартных директив препроцессора языка С или директив ассемблера.

Обычно директивы препроцессора языка С используются для подключения внешних файлов или для определения макроконструкций (макросов).

Пример:

```
#include "const.h"
#define PI 3.14159
```

Необходимо различать макрос и имя (символ). Макрос может определять большое выражение или задавать простое значение константы. Макрос, определяемый директивой #define, используется препроцессором. Препроцессор заменяет каждое название макроса, встретившееся в тексте, его значением. В примере на рис. 6 символы PI в программе будут заменены символами 3.14159 при подготовке файла к ассемблированию.

Также в программах обычно встречаются директивы ассемблера, задающие точность и способ округления, использующиеся во время работы программы.

Пример:

```
.PRECISION = 40;
.ROUND_ZERO;
```

Директивы, показанные в примере на рис. 6:

1. предписывает ассемблеру подключить код из внешнего файла
2. определяет макрос PI со значением 3.14159
3. устанавливает режим хранения данных с плавающей точкой с 40-битной точностью
4. устанавливает режим округления «к нулю» для данных с плавающей точкой, которые не могут быть представлены в 40-битном формате.

За предварительными директивами следует один или несколько сегментов. Каждый сегмент начинается директивой .SEGMENT и заканчивается директивой .ENDSEG. В примере на рис. 6 показаны четыре сегмента. Первый сегмент содержит данные, хранящиеся в памяти данных. В этом сегменте объявляются и инициализируются переменные и буферные массивы. Следующий сегмент содержит отображаемые на память порты ввода/вывода. Каждому порту присваивается уникальное имя, через которое к нему обращаются при чтении или записи. Последние два сегмента содержат код программы — обычно это инструкции, но также могут встречаться и директивы (например, директивы условного ассемблирования).

Каждая строка текста в исходном файле не должна быть длиннее 200 символов. Программные метки помещаются в начале строки и оканчиваются двоеточием:

```
startup: AY0=2; {начало программы}
```

В таблице 3 приведен список зарезервированных ключевых слов ассемблера. Ключевые слова нельзя использовать в качестве меток, имен портов или сегментов. Кроме того, нельзя использовать зарезервированные слова файла описания архитектуры, приведенные в предыдущей главе.

Таблица 3. Зарезервированные ключевые слова ассемблера

.BANK	COS	F4	L10	MR0F	PSA3S	SSFR
.ENDSYS	CURLCNTR	F5	L11	MR1B	PSA4E	SSI
.PROCESSOR	DAB1	F6	L12	MR1F	PSA4S	SSIR
.SEGMENT	DAB2	F7	L13	MR2B	PUSH	ST
.SYSTEM	DADDR	F8	L14	MR2F	PX	STACK
ABS	DAI1	F9	L15	MRB	PX1	STEP
AC	DAI2	FADDR	L2	MRF	PX2	STKY
ACT	DB	FDEP	L3	MS	R0	STS
ADSP21020	DEC	FEXT	L4	MV	R1	SUF
ADSP_21020	DM	FIX	L5	NE	R10	SUFR
AND	DM0	FLAG0_IN	L6	NEWPAGE	R11	SUI
ASHIFT	DM1	FLAG1_IN	L7	NOFO	R12	SUIR
ASTAT	DM2	FLAG2_IN	L8	NOFZ	R13	SV
AV	DM3	FLAG3_IN	L9	NOP	R14	SZ
B0	DMA1E	FLOAT	LA	NORMAL	R15	TCOUNT
B1	DMA1S	FMERG	LADDR	NOT	R2	TF
B10	DMA2E	FOREVER	LCE	NU	R3	TGL
B11	DMA2S	FR	LCNTR	OR	R4	TPERIOD
B12	DMADR	FTA	LE	P20	R5	TRUE
B13	DMAPARITYCK	FTB	LEFTO	P24	R6	TST
B14	DMBANK1	FTC	LEFTZ	P32	R7	UF
B15	DMBANK2	GE	LOAD	P40	R8	UI
B2	DMBANK3	GLOBAL	LOG2	PACK	R9	UNPACK
B3	DMDATA	GT	LOGB	PARITYCK	RAM	UNTIL
B4	DMWAIT	HEAP	LOOP	PASS	READ	UR
B5	DO	I0	LSHIFT	PC	RECIPS	USF
B6	DOVL	I1	LT	PCSTK	RND	USFR
B7	DRAM	I10	M0	PCSTKP	ROM	USI
B8	ECE	I11	M1	PGFAULT	ROT	USIR
B9	EITHER	I12	M10	PGMODE	ROUND_MINUS	USTAT1
BCLR	ELSE	I13	M11	PGSIZE	ROUND_NEAREST	USTAT2
BEGIN	EMOVERLAY	I14	M12	PGWAIT	ROUND_PLUS	UUF
BIT	EMUCLK	I15	M13	PM	ROUND_ZERO	UUFR
BITREV	END	I2	M14	PM0	RS	UUI
BOTH	ENDSEG	I3	M15	PM1	RSQRTS	UUIR
BSET	EQ	I4	M2	PMADR	RTI	VAR
BTGL	EX	I5	M3	PMBANK1	RTS	WITH
BTST	EXP	I6	M4	PMDAE	SAT	WTMODE
BY	EXP2	I7	M5	PMDAS	SCALB	WTSTATES
CA	EXTERN	I8	M6	PMDATA	SE	XOR
CACHE	EXTERNAL	I9	M7	PMWAIT	SEGMENT	
CALL	F0	IDLE	M8	POP	SET	
CDFAULT	F1	IF	M9	PORT	SF	
CH	F10	IMASK	MANT	POVL0	SHF	
CI	F11	IMASKP	MAX	POVL1	SHP	
CL	F12	INC	MIN	PRECISION	SI	
CLIP	F13	INTERNAL	MOD	PSA1E	SIN	
CLR	F14	IRPTL	MODE1	PSA1S	SQR	
CODE	F15	JUMP	MODE2	PSA2E	SR	
COMP	F2	L0	MODIFY	PSA2S	SRAM	
COPYSIGN	F3	L1	MR0B	PSA3E	SSF	

? Откройте в любом текстовом редакторе тексты программ-примеров из папки EZ-KIT\21K\EXAMPLES\061. (Место установки программного обеспечения для работы с отладочным модулем можно узнать у преподавателя. Процедура установки ПО описана в следующих разделах этого пособия.)

Для каждой программы ответьте на следующие вопросы:

1. Используются ли в программе директивы препроцессора языка C?
2. Используются ли в программе директивы ассемблера?
3. Сколько сегментов организованно в программе?
4. Какие из них расположены в памяти данных, а какие — в памяти программ?
5. Используются ли в программе метки?

Для каждой программы ответьте на следующие вопросы:

1. Используются ли в программе директивы препроцессора языка С?
2. Используются ли в программе директивы ассемблера?
3. Сколько сегментов организованно в программе?
4. Какие из них расположены в памяти данных, а какие — в памяти программ?
5. Используются ли в программе метки?

Компиляция

Для компиляции программ, написанных на языке ассемблера, необходимо:

1. Запустить компилятор (asm21k.exe), передав ему в качестве входного параметра имя файла с исходным кодом программы. Кроме того, рекомендуется использование параметров -adsp21060 и -I.
2. После окончания работы компилятора запустить линкер (Id21k.exe), передав ему в качестве входного параметра имя файла с объектным кодом программы, созданного при помощи компилятора. Кроме того, рекомендуется использование параметра-m.

(Более подробно о работе с компилятором и линкером рассказано во второй части данного пособия.)

Задание: Провести компиляцию программ-примеров из папки EZ-KIT\21K\EXAMPLES\061. (Место установки программного обеспечения для работы с отладочным модулем можно узнать у преподавателя. Процедура установки ПО описана в следующих разделах этого пособия.)

Отладка

Для начала отладки необходимо загрузить выполняемый файл (.exe). Файл описания конфигурации загружается при старте симулятора. Кроме того, в конфигурационном файле симулятора можно задать имя запускаемого файла, и он также будет загружаться каждый раз при старте симулятора. Для очистки памяти симулятора от загруженных в нее файлов используется команда **Execution|SimuIator Reset.** После её применения в память могут быть загружены новые программы. Команда Execution|Chip Reset сбрасывает только состояния регистров. Отлаживаемая программа остается в памяти и может быть запущена сначала.

Задание: Провести отладку программ, откомпилированных в предыдущем задании.

Аппаратно-программный отладочный модуль SHARC EZ-KIT Lite

Модуль SHARC EZ-KIT Lite является одним из наиболее удобных и простых инструментальных средств, позволяющих разрабатывать собственные приложения для систем цифровой обработки сигналов. Данный модуль представляет собой комплект соединенных между собой на плате устройств, в совокупности образующих полноценную систему цифровой обработки сигналов. Ядром модуля является сигнальный процессор ADSP-21061 с тактовой частотой 40 МГц. В состав основных периферийных устройств входят: 16-битный сигма-дельта стерео кодек, устройство синхронно-асинхронного приема/передачи (UART), реализующее интерфейс с персональным компьютером посредством порта RS-232 и внешняя память загрузки программы (EPROM). Структурная схема модуля показана на рис. 7.

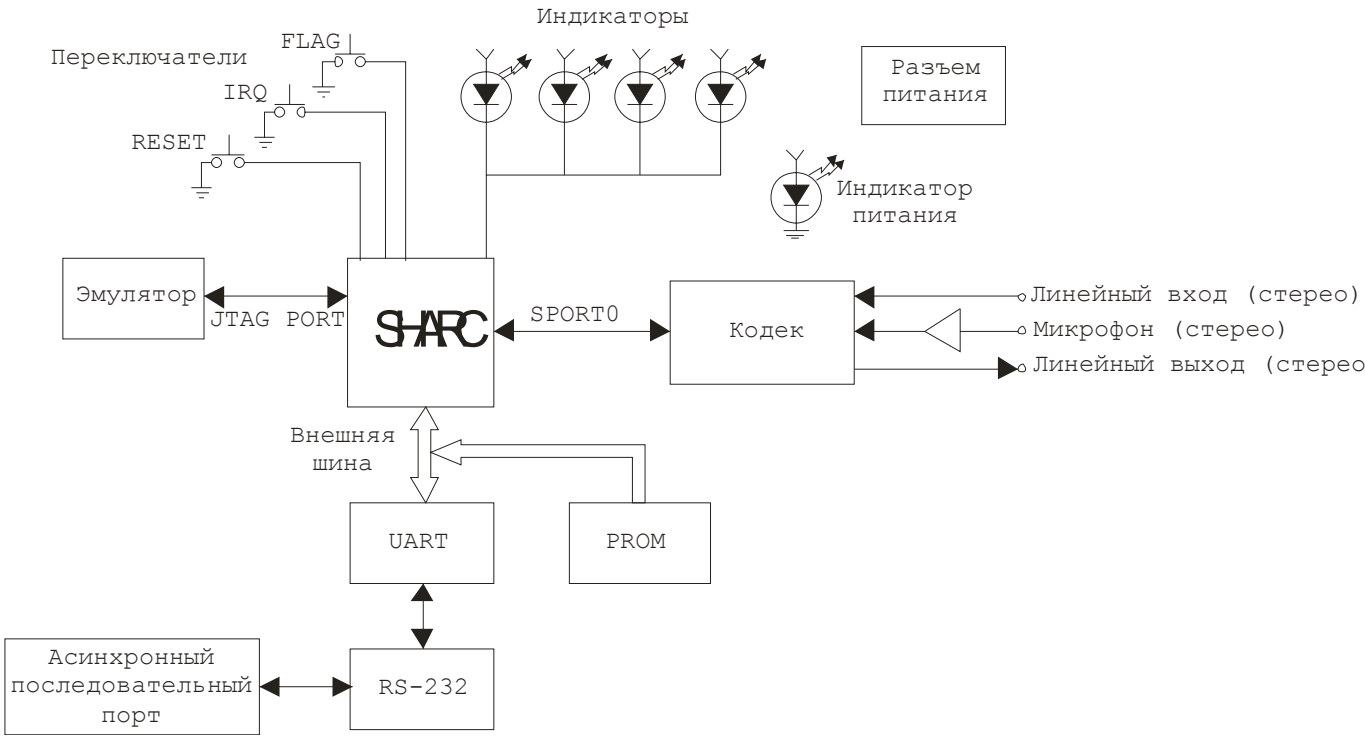


Рис. 7. Структурная схема модуля SHARC EZ-KIT Lite

Плата цифровой обработки может функционировать как автономно, так и с подключением к персональному компьютеру при помощи интерфейса RS-232 через COM-порт. Программные средства, входящие в комплект вместе с платой, позволяют эффективно и быстро вести разработку и отладку программ, их загрузку во внутреннюю память процессора и исполнение. Также поддерживается возможность работы с внутрисхемным симулятором EZ-ICE, соединяемым с модулем SHARC EZ-KIT Lite через порт тестовой отладки и контроля JTAG.

Рис. 8. Внешний вид отладочного модуля

В комплект отладочного модуля входят:

- плата модуля (рис. 8);

- блок питания 12 В, 0.6 А;
- кабель дня подключения к последовательному порту компьютера.

Ниже приведен перечень основных устройств, входящих в состав модуля.

- *Устройство синхронно-асинхронного приема/передачи (UART),* Микросхема PC16550D с дополнительным драйвером линии реализует интерфейс RS-232 для связи с персональным компьютером (ПК). Функционально PC16550D аналогична UART, применяемым для организации интерфейса в ПК (например, i8250). Она содержит набор регистров для приема и передачи битовых потоков данных и позволяет регулировать скорость передачи.
- *Стерео кодек AD1847.* Кодек, входящий в состав EZ-KIT Lite исполняет роль цифро-аналогового и аналого-цифрового преобразователей и обеспечивает ввод и вывод стерео сигналов. Прямое соединение кодека с процессором обеспечивается посредством синхронного последовательного порта. Входные фильтры могут быть при необходимости программно отключены.
- Share
- RS232
- JTAG
- *Память загрузки (EPROM).* Внешняя память загрузки позволяет хранить программу процессора и загружать ее в процессор при включении питания или сбросе. С помощью перемычек на плате имеется возможность изменения конфигурации памяти.
- *Элементы управления и индикации.* В схему платы включены светодиоды, отображающие состояние некоторых флагов в регистрах процессора и регистрирующие поступление некоторых прерываний. Имеется возможность “вручную” изменить состояние одного из этих флагов, либо выполнить прерывание, а также аппаратный сброс. Это можно сделать с помощью установленных на плате кнопочных переключателей.
- Разъем питания

Кроме того, на плате имеется 7 свободных разъемов, предоставляющих возможность расширения функций модуля и подключения к нему дополнительных устройств.

?

1. Перечислите основные модули, входящие в SHARC EZ-KIT Lite.
2. Какое функциональное назначение имеет JTAG интерфейс.
3. Приведите основные характеристики ADSP21061.

Подготовка SHARC EZ-KIT Lite к работе

Установка программного обеспечения

Отладочный модуль SHARC EZ-KIT Lite способен работать в автономном режиме, но в большинстве случаев разработка, отладка и компиляция программ, их загрузка в память модуля, просмотр и изменение содержимого памяти и прочие действия производятся с персонального компьютера. Поэтому для полноценной работы необходимо подключить модуль к ПК и установить на компьютере требуемое программное обеспечение.

Для работы с отладочным модулем SHARC EZ-KIT Lite компьютер должен отвечать следующим минимальным требованиям

- 386 процессор;
- VGA-видеоадаптер и монитор;
- 2 Мб ОЗУ;
- 20 Мб свободного места на жестком диске;
- наличие свободного COM-порта (RS-232);
- Windows 98;

Чтобы установить программное обеспечение для работы с отладочным модулем необходимо закрыть все активные приложения Windows, запустить файл install.exe из корневого каталога установочного компакт-диска модуля SHARC EZ-KIT Lite и следовать инструкциям мастера установки. По завершению установки компьютер требуется перезагрузить.

Задание: Установите ПО для работы с отладочным модулем SHARC EZ-KIT Lite на компьютер, указанный преподавателем.

Включение отладочного модуля

На рис. 9 показаны основные разъемы и органы управления, использующиеся при работе с отладочным модулем SHARC EZ-KIT Lite.

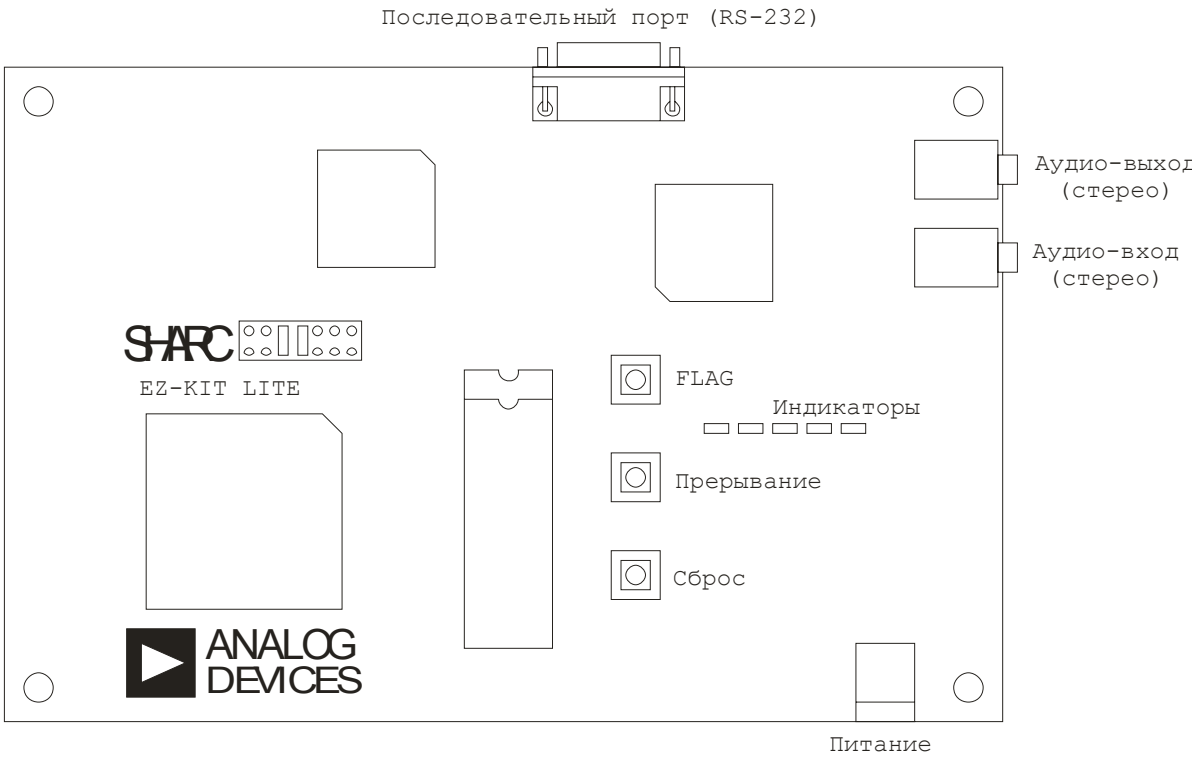


Рис. 9. Внешние разъемы и органы управления отладочного модуля

Порядок подключения:

1. Выключите компьютер, на который установлено программное обеспечение для работы с модулем.

2. Подключите кабель RS-232 из комплекта отладочного модуля к соответствующему разъему на модуле и к последовательному порту компьютера. **Внимание! На компьютере следует использовать последовательный порт, номер которого был выбран при установке программного обеспечения.**

3. Подключите к аудио-входу модуля источник звукового сигнала (микрофон, линейный выход звуковой платы компьютера и т.п.).

4. Подключите к аудио-выходу модуля приемник звукового сигнала (наушники, линейный вход звуковой платы компьютера и т.п.).

5. Включите компьютер, на который установлено программное обеспечение для работы с модулем, и дождитесь окончания загрузки.

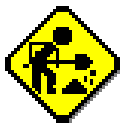
6. Подключите питание отладочного модуля.

Нажмите кнопочный переключатель Reset, расположенный на поверхности модуля. Модуль исправен, если:

7. расположенный на плате индикатор **Power** горит ровным светом;

8. мигают индикаторы **Flag2** и **Flag3** (индикатор **FIag3** мигает с большей частотой);

9. на аудио-выход модуля поступает звуковой сигнал.



Задание: Подключите отладочный модуль и проверьте его функционирование.

Программы управления отладочным модулем EZ-KIT LITE HOST

Программа SHARC EZ-KIT Lite Host - это основная программа для связи и управления отладочным модулем SHARC EZ-KIT Lite. С её помощью возможно:

1. запускать демонстрационные программы
2. загружать, считывать или просматривать содержимое памяти программ и данных
3. загружать и запускать собственные программы Цифровой Обработки Сигналов

Для запуска программы щелкните значок **EZ-KIT Lite Host** из группы **SHARC EZ-KIT Lite**. На рис. 10 показано основное окно программы.

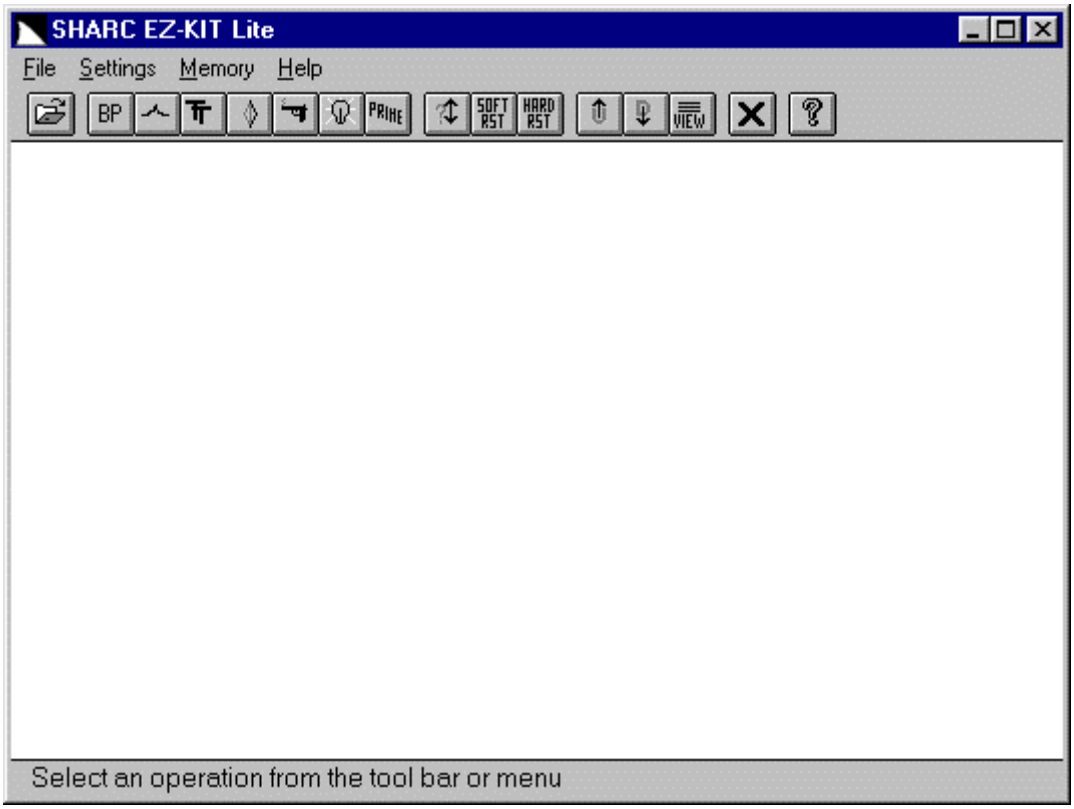
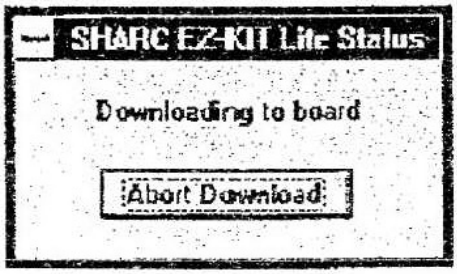
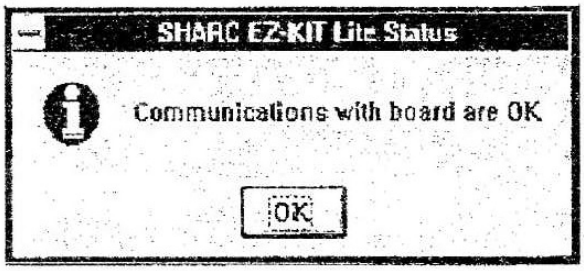
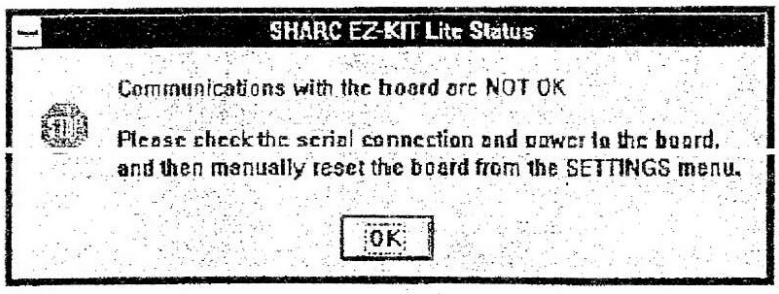
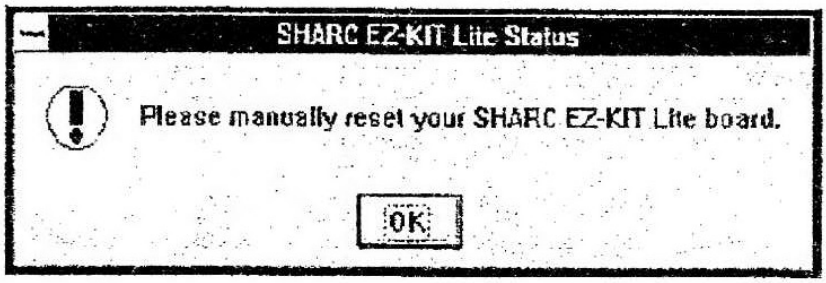


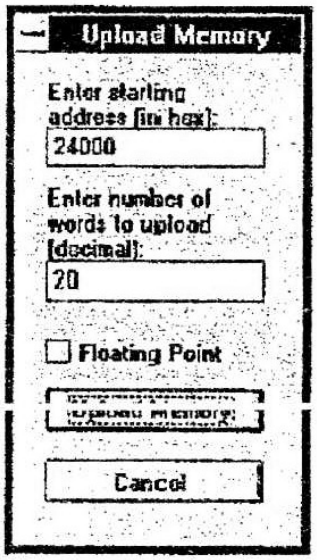
Рис. 10. Основное окно программы управления

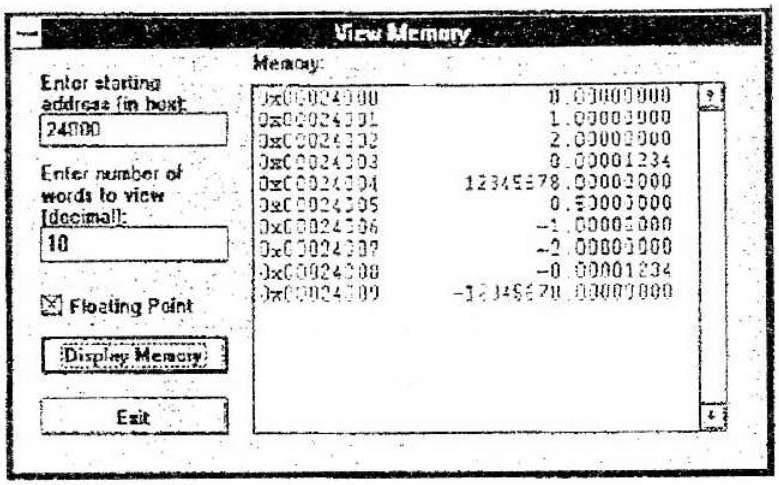
Функциональное назначение пунктов меню

File	
Open	При выборе команды Open... (или нажатии комбинации клавиш Ciri-O, или щелчке на кнопке панели инструментов) появится диалоговое окно, в котором возможно выбрать программу для загрузки в EPROM отладочного модуля.
	После того как вы выбрали программу для загрузки в EPROM, вы увидите окно, показанное ниже. Программа начнет загружаться в память модуля EZ-KIT Lite. Загрузку можно прервать, щелкнув кнопку Abort Download.  После загрузки программы в модуль, она запускается автоматически.
Demo	Загрузка демонстрационных программ.
Settings	
Select Com Port	Эта команда позволяет задать последовательный порт (COM-порт) к которому подключен отладочный модуль SHARC EZ-KIT Lite. При наведении указателя мыши на эту команду появляется выпадающее меню, содержащее список портов: COM1, COM2, COM3, или COM4. Порт, по которому осуществляется связь в данный момент, отмечен галочкой. После выбора нового порта программа попытается установить связь с отладочным модулем. Номер порта для связи записывается в файл ezsharc.ini в директории программы.
Select Port Baud Rate	Эта команда позволяет задать скорость связи по последовательному порту (в бодах). При наведении указателя мыши на эту строку появляется выпадающее меню с пятью вариантами скорости соединения: 9600, 19200, 38400, 57600 и 115200. Текущая скорость отмечена галочкой. При выборе новой скорости программа проводит проверку связи с отладочным модулем. Скорость связи записывается в файл ezsharc.ini в директории программы.

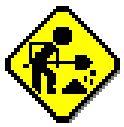
Test Communications	Эта команда позволяет проверить связь программы управления с отладочным модулем SIIARC EZ-KIT Lite. Если проверка связи прошла успешно, вы увидите следующее окно:  Если связаться с отладочным модулем не удалось, программа выведет на экран следующее предупреждение:  В этом случае следует: • Проверить подключение кабеля к последовательному порту компьютера и порту отладочного модуля. • Проверить подается ли питание на модуль. • Проверить установленный номер порта и скорость связи (см. выше). После проверки этих параметров выберите команду Manually Reset The Board из меню Settings. Если связь не восстановилась, попробуйте перезапустить программу управления.
Soft Reset	При выборе этой команды выполнение программы, загруженной в память отладочного модуля, прекращается. Программа управления перехватывает контроль над отладочным модулем, но не нарушает содержимого памяти.
Hard Reset	Эта команда сбрасывает процессор отладочного модуля и перезагружает его из EPROM. Отладочный модуль выполнит процедуру самодиагностики при включении и запустит стандартную демонстрационную программу (Peter Gunn Theme).
Manual Reset	Эту команду следует использовать при нарушениях связи или в тех случаях, когда в работе отладочного модуля появляются ошибки. При

	выборе этой команды вы увидите следующее окно:  Нажмите кнопку Reset на отладочном модуле SHARC EZ-KIT, а затем щелкните кнопку OK. Программа управления попытается восстановить связь с модулем и сообщит о результатах.

Memory	При помощи команд этого меню можно просматривать, изменять и сохранять на диске содержимое памяти отладочного модуля. Команды этого меню можно использовать в любое время, даже при выполняющейся на модуле программе.
Upload Memory from EZ-KIT Lite	Эта команда позволяет считывать содержимое внутренней памяти SHARC и сохранять его в файле. При выборе этой команды вы увидите следующее диалоговое окно:  Введите корректный адрес памяти SHARC (в шестнадцатеричном формате) в первом поле ввода и количество слов для чтения (в десятичном формате) во втором поле ввода. Если вы хотите, чтобы считанные данные интерпретировались как значения с плавающей точкой, отметьте флажок Floating Point. После этого щелкните на кнопке Upload Memory. Вы увидите диалоговое окно записи и сможете указать в нем местоположение файла, в который будут

	записаны полученные данные. После указания пути и имени файла данные будут считаны из памяти модуля и записаны в файл в ASCII формате.
Download Memory to EZ-KIT Lite	Эта команда позволяет записывать значения из файла на компьютере в память отладочного модуля. При выборе этой команды вы увидите диалоговое окно, похожее на предыдущее. Введите корректный адрес памяти SHARC (в шестнадцатеричном формате) в первом поле ввода и количество слов, которое вы хотите записать, (в десятичном формате) во втором поле ввода. После этого щелкните на кнопке Download Memory. Вы увидите диалоговое окно чтения и сможете указать в нем местоположение файла, из которого будут записаны хранящиеся в нем данные.
View Memory Range	Эта команда позволяет просматривать содержимое памяти отладочного модуля. При выборе этой команды вы увидите следующее диалоговое окно:  Введите корректный адрес памяти SHARC (в шестнадцатеричном формате) в первом поле ввода и количество слов для отображения (в десятичном формате) во втором поле. Если вы хотите чтобы считанные данные интерпретировались как значения с плавающей точкой, отметьте флажок Floating Point. После щелчка на кнопке Display Memory данные считываются из памяти отладочного модуля SHARC и отображаются в области Memory.

Далее следуют задания, целью которых является отработка действий с данным ПО.



Задание 1: Тестирование модуля и ПО на работоспособность.

Проверьте подключение и исправность отладочного модуля. Загрузите в память модуля какую-либо программу (например, из набора демонстрационных программ меню File\Demo) и проверьте ее на работоспособность.

Задание 2: Работа с подсистемой памяти модуля.
Выведите на экран содержимое адресов памяти модуля, в которых расположена загруженная программа. Модифицируйте содержимое памяти модуля. (Примечание. После выполнения второго шага данного задания программа в памяти модуля будет повреждена. Для восстановления работоспособности проведите инициализацию модуля при помощи команды меню Settings\Manual Reset и повторно загрузите программу в память.)

ADSP-2106X - ОБЗОР АРХИТЕКТУРЫ И СИСТЕМЫ КОМАНД

Вычислительные устройства ADSP-2106x специально оптимизированы для выполнения алгоритмов Цифровой Обработки Сигналов (ЦОС). ADSP-2106X содержит три вычислительных устройства: Арифметико-Логическое Устройство (АЛУ), устройство умножения (умножитель) и устройство сдвига. Процессор может выполнять операции над числами с фиксированной и плавающей точкой. Каждое вычислительное устройство выполняет одну инструкцию за такт.

АЛУ выполняет стандартный набор арифметических и логических операций над числами с фиксированной и плавающей точкой. Устройство умножения производит умножение чисел с фиксированной и плавающей точкой, а также умножение со сложением и умножение с вычитанием над числами с фиксированной точкой. Устройство сдвига выполняет арифметические и логические сдвиги, побитную обработку, выборку и вставку битовых полей на 32-битных операндах а также операцию извлечения показателя степени.

Вычислительные устройства работают параллельно (рис. 11). Выходные данные любого вычислительного устройства на следующем цикле могут использоваться как входные данные для любого устройства. Вычислительные устройства принимают входные данные и помещают результаты в 10-портовый блок регистров, состоящий из шестнадцати первичных и шестнадцати альтернативных регистров. Доступ к блоку регистров также может производиться по шинам программной памяти и памяти хранения данных процессора ADSP- 2106x для передачи данных между вычислительными устройствами и внешней памятью или другими частями процессора.

В исходном коде программы на ассемблере отдельные регистры регистрового блока при операциях над числами с плавающей точкой обозначаются префиксом “F”. При операциях над числами с фиксированной точкой используется префикс “R”.

Например, следующие инструкции используют одни и те же регистры:
F0=F1 * F2; *умножение с плавающей точкой*

R0=R1 * R2; *умножение с фиксированной точкой*

Префиксы F и R не влияют на 32- или 40-битную передачу данных; они лишь определяют, каким образом АЛУ, умножитель, или устройство сдвига обрабатывают данные. Префиксы F или R могут набираться большими или маленькими буквами; ассемблер не учитывает регистр.

Префиксы F и R не влияют на 32- или 40-битную передачу данных; они лишь определяют, каким образом АЛУ, умножитель, или устройство сдвига обрабатывают данные. Префиксы F или R могут набираться большими или маленькими буквами; ассемблер не учитывает регистр.

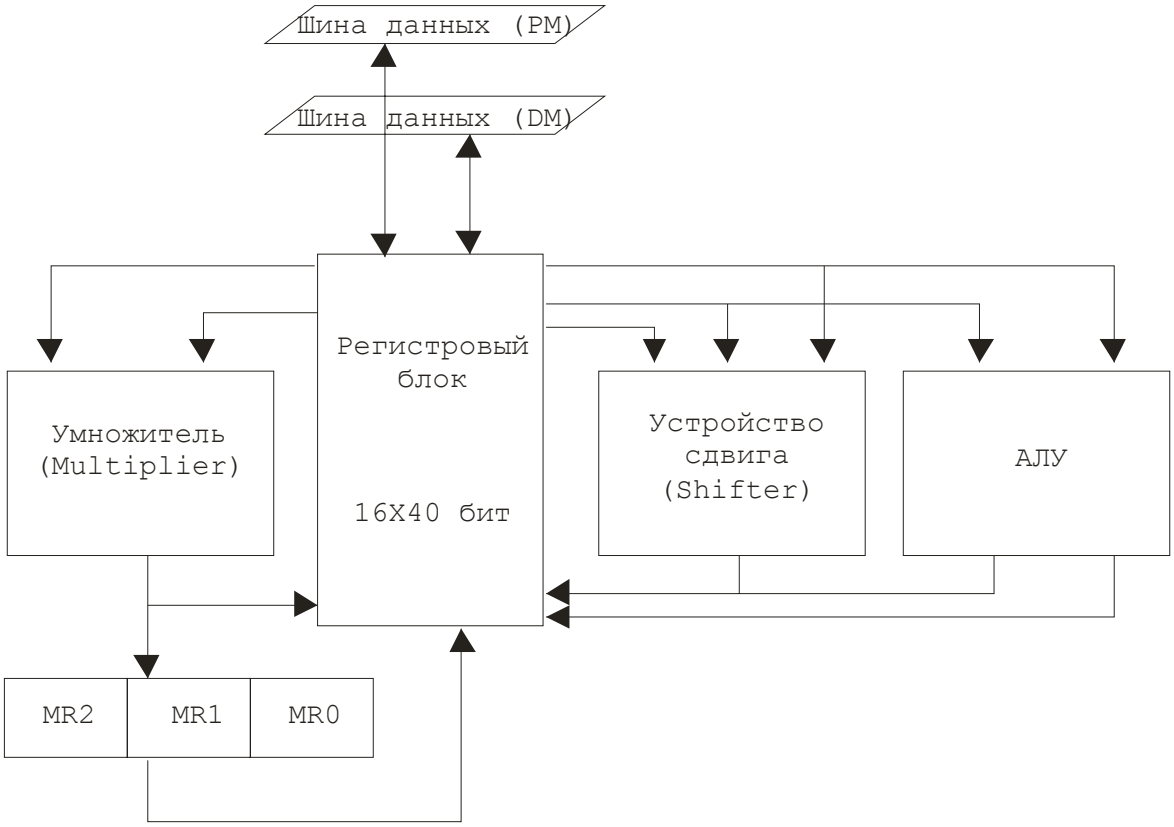


Рис. 11. Вычислительные устройства ADSP-2106x

АЛУ

АЛУ выполняет арифметические операции над числами с фиксированной и плавающей точкой и логические операции над числами с фиксированной точкой. Инструкции АЛУ по обработке чисел с фиксированной точкой работают с 32-битными операндами и возвращают 32-битный результат с фиксированной точкой. Инструкции по

обработке чисел с плавающей точкой работают с 32- или 40-битными операндами и возвращают 32- или 40-битный результат с плавающей точкой.

Инструкции АЛУ включают в себя:

- Сложение, вычитание, вычисление среднего для операндов с плавающей точкой.
- Сложение, вычитание, вычисление среднего для операндов с фиксированной точкой.
- Обработку чисел с плавающей точкой: двоичный логарифм, масштабирование и др.
- Сложение с переносом, вычитание с заёмом, увеличение, уменьшение для операндов с фиксированной точкой.
- Логические И, ИЛИ, Исключающее ИЛИ, НЕ.
- Функции: абсолютное значение, минимум, максимум, усечение, сравнение.
- Преобразование формата.
- Обратная дробь и обратный квадратный корень.

Таблица 4. Инструкция АЛУ

Инструкция	Флаги ASTAT								Флаги STKY			
операции с фиксированной точкой	AZ	AV	AN	AC	AS	AI	AF	CACC	AUS	AVS	AOS	AIS
c Rn=Rx+Ry	*	*	*	*	0	0	0	–	–	–	**	–
c Rn=Rx–Ry	*	*	*	*	0	0	0	–	–	–	**	–
c Rn=Rx+Ry+CI	*	*	*	*	0	0	0	–	–	–	**	–
c Rn=Rx–Ry+CI–1	*	*	*	*	0	0	0	–	–	–	**	–
Rn=(Rx+Ry)/2	*	0	*	*	0	0	0	–	–	–	–	–
COMP(Rx,Ry)	*	0	*	0	0	0	0	*	–	–	–	–
Rn=Rx+CI	*	*	*	*	0	0	0	–	–	–	**	–
Rn=Rx+CI–1	*	*	*	*	0	0	0	–	–	–	**	–
Rn=Rx+1	*	*	*	*	0	0	0	–	–	–	**	–
Rn=Rx–1	*	*	*	*	0	0	0	–	–	–	**	–
c Rn=–Rx	*	*	*	*	0	0	0	–	–	–	**	–
c Rn=ABS Rx	*	*	0	0	*	0	0	–	–	–	**	–
Rn=PASS Rx	*	0	*	0	0	0	0	–	–	–	–	–
c Rn=Rx AND Ry	*	0	*	0	0	0	0	–	–	–	–	–
c Rn=Rx OR Ry	*	0	*	0	0	0	0	–	–	–	–	–
c Rn=Rx XOR Ry	*	0	*	0	0	0	0	–	–	–	–	–
c Rn=NOT Rx	*	0	*	0	0	0	0	–	–	–	–	–
Rn=MIN(Rx,Ry)	*	0	*	0	0	0	0	–	–	–	–	–
Rn=MAX(Rx,Ry)	*	0	*	0	0	0	0	–	–	–	–	–
Rn=CLIP Rx BY Ry	*	0	*	0	0	0	0	–	–	–	–	–
операции с плавающей точкой												
Fn=Fx+Fy	*	*	*	0	0	*	1	–	**	**	–	**
Fn=Fx–Fy	*	*	*	0	0	*	1	–	**	**	–	**
Fn=ABS(Fx+Fy)	*	*	0	0	0	*	1	–	**	**	–	**
Fn=ABS(Fx–Fy)	*	*	0	0	0	*	1	–	**	**	–	**
Fn=(Fx+Fy)/2	*	0	*	0	0	*	1	–	**	–	–	**
COMP(Fx,Fy)	*	0	*	0	0	*	1	*	–	–	–	**
Fn=–Fx	*	*	*	0	0	*	1	–	–	**	–	**
Fn=ABS Fx	*	*	0	0	*	*	1	–	–	**	–	**
Fn=PASS Fx	*	0	*	0	0	*	1	–	–	–	–	**
Fn=RND Fx	*	*	*	0	0	*	1	–	–	**	–	**
Fn=SCALB Fx BY Ry	*	*	*	0	0	*	1	–	**	**	–	**
Fn=MANT Fx	*	*	0	0	*	*	1	–	–	**	–	**
Fn=LOGB Fx	*	*	*	0	0	*	1	–	–	**	–	**
Fn=FIX Fx BY Ry	*	*	*	0	0	*	1	–	**	**	–	**
Fn=FIX Fx	*	*	*	0	0	*	1	–	**	**	–	**
Fn=FLOAT Rx BY Ry	*	*	*	0	0	0	1	–	**	**	–	–
Fn=FLOAT Rx	*	0	*	0	0	0	1	–	–	–	–	–
Fn=RECIPS Fx	*	*	*	0	0	*	1	–	**	**	–	**
Fn=RSQRTS Fx	*	*	*	0	0	*	1	–	–	**	–	**
Fn=Fx COPYSIGN Fy	*	0	*	0	0	*	1	–	–	–	–	**
Fn=MIN(Fx,Fy)	*	0	*	0	0	*	1	–	–	–	–	**
Fn=MAX(Fx,Fy)	*	0	*	0	0	*	1	–	–	–	–	**
Fn=CLIP Fx BY Fy	*	0	*	0	0	*	1	–	–	–	–	**

Rn, Rx, Ry – любой регистр, использующийся для операций с фиксированной точкой;

Fn, Fx, Fy – любой регистр, использующийся для операций с плавающей точкой;

C – инструкция совместима с ADSP-21xx.

* — флаг устанавливается или сбрасывается в зависимости от результатов операции;

** — в зависимости от результатов операции флаг может быть установлен (но не сброшен);

- — состояние флага не зависит от данной операции.

Примеры инструкций АЛУ:

```
F8=F8-F13
R4=R3+R2+CI
R3=LOGB(F9)
R3=R3-R4+CI-1
F5=(F5+F6)/2

R5=R2 AND R3
R3=R3 XOR R3
R4=ABS R2
F7=MIN(F4,F5)
F2=RSQRTS F8
```

Умножитель

Умножитель выполняет умножение чисел с фиксированной и плавающей точкой и умножение с накоплением чисел с фиксированной точкой. Умножение с накоплением чисел с фиксированной точкой может быть выполнено как с накопленным сложением так и с накопленным вычитанием. Умножение с накоплением чисел с плавающей точкой может быть выполнено с помощью параллельного использования АЛУ и умножителя, с использованием мультифункционалъных инструкций.

Инструкции умножителя по обработке чисел с плавающей точкой работают с 32- или 40-битными операндами и возвращают 32- или 40-битный результат с плавающей точкой. Инструкции по обработке чисел с фиксированной точкой работают с 32-битными операндами и возвращают 80-битный результат с фиксированной точкой. Входные данные обрабатываются как дробные или целые, беззнаковые или записанные в дополнительном коде.

Инструкции умножителя включают в себя:

- Умножение с плавающей точкой
- Умножение с фиксированной точкой
- Умножение со сложением чисел с фиксированной точкой, с возможным округлением
- Умножение с вычитанием чисел с фиксированной точкой, с возможным

округлением

- Регистр округления
- Регистр насыщения
- Регистр очистки

Таблица 5. Инструкция умножителя

Инструкция	Флаги ASTAT				Флаги STKY			
	MU	MN	MV	MI	MUS	MOS	MVS	MIS
MRxB Rn = MRxF / MRxB	0	0	0	0	–	–	–	–
операции с плавающей точкой								
Fn = Fx * Fy	*	*	*	*	**	–	**	**
операции с фиксированной точкой								
Rn / MRF / MRB = Rx*Ry (S/U, S/U, F/I/FR)	*	*	*	0	–	**	–	–
Rn = MRF + Rx*Ry (S/U, S/U, F/I/FR) Rn = MRB MRF = MRF MRB = MRB	*	*	*	0	–	**	–	–
Rn = MRF – Rx*Ry (S/U, S/U, F/I/FR) Rn = MRB MRF = MRF MRB = MRB	*	*	*	0	–	**	–	–
Rn = SAT MRF (SI/UI/SF/UF) Rn = SAT MRB MRF = SAT MRF MRB = SAT MRB	*	*	*	0	–	**	–	–
Rn = RND MRF (SF/UF) Rn = RND MRB MRF = RND MRF MRB = RND MRB	*	*	*	0	–	**	–	–
MRF / MRB = 0	0	0	0	0	–	–	–	–
MRxF = 0	0	0	0	0	–	–	–	–

Примечание: В таблице не указаны операции умножения с накоплением для чисел с плавающей точкой.

* — флаг устанавливается или сбрасывается в зависимости от результатов операции;

** — в зависимости от результатов операции флаг может быть установлен (но не сброшен);

- — состояние флага не зависит от данной операции.

Rn, Rx, Ry = R15-RO — регистр, использующийся для операций с фиксированной точкой;

Fn, Fx, Fy = F15-F0 — регистр, использующийся для операций с плавающей точкой;

MRxF = MR2F, MR1F, MR0F — первичные аккумуляторы результата умножителя.

MRxB = MR2B, MR1 B, MR0B — вторичные аккумуляторы результата умножителя.

Примеры инструкций умножителя:

F13=F0*F4
R3=R2*R5 (SSI)
R4=MRF+R3*R2 (USFR)
R12=MRB-R3*R11 (UUF)
MRF=0
MR=R3
R2=MR

Устройство сдвига

Устройство сдвига работает с 32-битными операндами с фиксированной точкой. Инструкции устройства сдвига включают в себя:

- Сдвиг и циклический сдвиг
- Битовые операции, такие как установка, сброс, переключение, и проверка значений битов
- Операции по выборке и помещению битовых полей
- Поддержку для совместимых с семейством ADSP-2100 операций преобразования фиксированная/плавающая точка (выборка показателя, количество предшествующих нулей или единиц)

Таблица 6. Инструкции устройства сдвига

	Инструкция	Флаги SZ	SV	SS
c	Rn=LSHIFT Rx BY Ry	*	*	0
c	Rn=LSHIFT Rx BY <data8>	*	*	0
c	Rn=Rn OR LSHIFT Rx BY Ry	*	*	0
c	Rn=Rn OR LSHIFT Rx BY <data8>	*	*	0
c	Rn=ASHIFT Rx BY Ry	*	*	0
c	Rn=ASHIFT Rx BY <data8>	*	*	0
c	Rn=Rn OR ASHIFT Rx BY Ry	*	*	0
c	Rn=Rn OR ASHIFT Rx BY <data8>	*	*	0
	Rn=ROT Rx BY Ry	*	0	0
	Rn=ROT Rx BY <data8>	*	0	0
	Rn=BCLR Rx BY Ry	*	*	0
	Rn=BCLR Rx BY <data8>	*	*	0
	Rn=BSET Rx BY Ry	*	*	0
	Rn=BSET Rx BY <data8>	*	*	0
	Rn=BTGL Rx BY Ry	*	*	0
	Rn=BTGL Rx BY <data8>	*	*	0
	BTST Rx BY Ry	*	*	0
	BTST Rx BY <data8>	*	*	0
	Rn=FDEP Rx BY Ry	*	*	0
	Rn=FDEP Rx BY <bit6>;<len6>	*	*	0
	Rn=Rn OR FDEP Rx BY Ry	*	*	0
	Rn=Rn OR FDEP Rx BY <bit6>;<len6>	*	*	0
	Rn=FDEP Rx BY Ry (SE)	*	*	0
	Rn=FDEP Rx BY <bit6>;<len6> (SE)	*	*	0
	Rn=Rn OR FDEP Rx BY Ry (SE)	*	*	0
	Rn=Rn OR FDEP Rx BY <bit6>;<len6> (SE)	*	*	0
	Rn=FEXT Rx BY Ry	*	*	0
	Rn=FEXT Rx BY <bit6>;<len6>	*	*	0
	Rn=FEXT Rx BY Ry (SE)	*	*	0
	Rn=FEXT Rx BY <bit6>;<len6> (SE)	*	*	0
c	Rn=EXP Rx (EX)	*	0	*
c	Rn=EXP Rx	*	0	*
	Rn=LEFTZ Rx	*	*	0
	Rn=LEFTO Rx	*	*	0
	Rn=FPACK Fx	0	*	0
	Fn=FUNPACK Rx	0	0	0

* — состояние флага зависит от данных;

Rn, Rx, Ry — любой регистр; в зависимости от инструкции могут использоваться битовые поля;

Fn, Fx — любой регистр рассматривается как слово с плавающей точкой;

c — инструкция совместима с ADSP-2100.

Примеры инструкций устройства сдвига:

```
R4=LSHIFT R3 BY R1
R4=ASHIFT R3 BY 4
R8=BSET R1 BY R2
R4=EXP R8
R1=FPACK F2
F2=FUNPACK R1
```

Регистровый блок

Регистровый блок предоставляет интерфейс между внутренними шинами данных процессора и вычислительными устройствами, а также место для хранения операндов и промежуточных результатов. Блок регистров состоит из 16 первичных и 16 альтернативных (вторичных) регистров. Все регистры 40-битные. 32-битные данные от вычислительных устройств выравниваются по левому краю; при чтении из регистра восемь младших битов игнорируются, а при записи в регистр - заполняются нулями.

Доступ к блоку регистров из программной памяти и памяти данных проводится через шину программной памяти (PM Data bus) и шину памяти данных (DM Data bus), соответственно. За один такт может совершаться одна выборка данных через шину PM и/или одна выборка через шину DM. Шина DM 40-разрядная и обмен данными между ней и регистрами ведется напрямую, Обмен данными между регистрами и 48-разрядной шиной PM ведется только для старших 40 битов. При передаче значения из регистра в шину PM, в младшие восемь битов записываются нули.

Если в качестве источника и приемника данных указан один и тот же регистр из регистрового блока, то в первой половине цикла происходит чтение, а во второй - запись. Таким образом, старое значение используется в качестве операнда до того как оно будет перезаписано новым результатом. Если в одном цикле происходит несколько записей результата в один и тот же регистр из разных источников, то проводится только запись с наивысшим приоритетом. Приоритет определяется источником, из которого записываются данные; приоритеты от высшего к низшему распределены следующим образом:

1. Память данных или регистр
2. Программная память
3. АЛУ
4. Умножитель
5. Устройство сдвига

Примеры передачи информации между регистрами и памятью:

```
F5=PM(I9,M8)
DM(I2,M1)=F9
```

Альтернативные (вторичные) регистры

Для облегчения быстрого переключения в соответствии с различными ситуациями, регистровый блок имеет набор альтернативных регистров. Каждая половина регистрового блока — нижняя (регистры с R0 по R7) и верхняя (с R8 по R15) — могут независимо друг от друга переключаться на набор альтернативных регистров. Активный блок регистров (основной или альтернативный) выбирается установкой двух битов в регистре состояния MODE1. Обмен данными между наборами регистров может осуществляться размещением данных в одной половине регистрового блока и активацией альтернативного набора регистров в другой половине.

ПОРЯДОК ВЫПОЛНЕНИЯ ПРОГРАММЫ

В ADSP-2106x программа чаще всего выполняется линейно; т.е. процессор последовательно выполняет инструкции программы. Возможны и другие способы организации выполнения программ:

- Циклы: одна последовательность команд выполняется несколько раз.
- Подпрограммы: процессор временно прерывает последовательное выполнение программы, чтобы выполнить набор инструкций из другой части программы.
- Переходы: выполнение инструкций переключается на другую часть программы.
- Прерывания: особый вид подпрограмм, в котором выполнение подпрограммы начинается по какому-либо событию, произошедшему во время работы программы.
- Останов: специальная инструкция, прерывающая операции процессора и останавливающая его в текущем состоянии. При возникновении прерывания процессор обрабатывает его и продолжает нормальное выполнение программы.

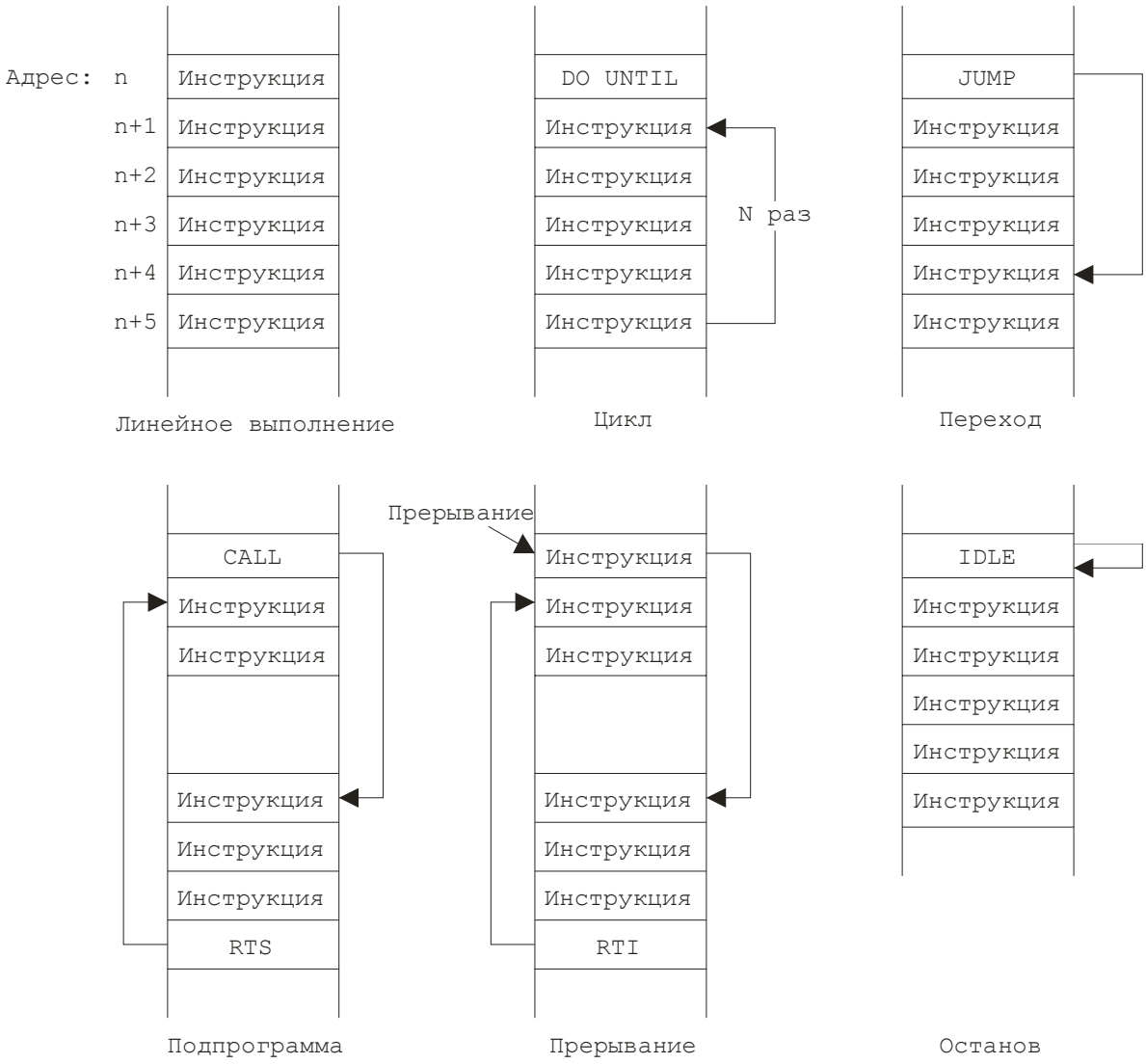


Рис. 12. Различные порядки выполнения программы

Примеры передачи управления и организации циклов в программе:
LCNTR=100, DO fmax UNTIL LCE;

DO end UNTIL FLAG1_IN;
CALL init (DB);

JUMP start;

ТАКТЫ ВЫПОЛНЕНИЯ ИНСТРУКЦИИ

ADSP-2106x выполняет инструкции затри такта:

- В такте выборки ADSP-2106x считывает инструкцию из кэш-памяти для инструкций, или из программной памяти.
- В течение такта декодирования, инструкция расшифровывается.
- В такте выполнения, ADSP-2106x выполняет инструкцию - производится операция, заданная инструкцией.

Эти циклы перекрываются (конвейеризируются) как показано на рис. 13. При последовательном выполнении программы во время выборки инструкции декодируется инструкция, считанная в предыдущем такте, и выполняется инструкция, считанная два такта назад. Таким образом, процессор выполняет одну инструкцию за такт.

Время (такты)	Выборка	Декодирование	Исполнение
1	0x08		
2	0x09	0x08	
3	0x0A	0x09	0x08
4	0x0B	0x0A	0x09
5	0x0C	0x0B	0x0A

Рис. 13. Последовательность обработки инструкций при конвейеризации

ВЫБОРКА КОМАНД

Последовательное выполнение

Адрес следующей инструкции определяется выполняемой в данный момент инструкцией и текущим состоянием процессора. В обычной ситуации (если не возникает специальных условий), ADSP-2106x выполняет инструкции из программной памяти последовательно, просто увеличивая адрес выборки.

Ветвления

Ветвление возникает в том случае, если следующий адрес выборки не является последовательно расположенным за предыдущим адресом. ADSP-2106x поддерживает такие виды ветвлений как переходы, вызовы подпрограмм и возврат. При выборке команд, единственная разница между переходом и вызовом подпрограммы состоит в том, что во время вызова подпрограммы в стек записывается адрес возврата, который считывается при возврате из подпрограммы. Переход передает управление в другую часть программы без возможности возврата.

Циклы

В ADSP-2106x циклы организуются при помощи инструкции DO UNTIL. Инструкция DO UNTIL заставляет процессор повторять определенный участок программы до тех пор, пока не выполнится условие выхода из цикла.

2. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

МЕТОДИКА РАЗРАБОТКИ ПРОГРАММ НА ЯЗЫКЕ АССЕМБЛЕР

Средства разработки

При разработке ПО используются следующие средства разработки:

- Редактор
- Ассемблер (компилятор)
- Линкер
- Загрузчик
- Отладчик

Этапы разработки

- Процесс разработки начинается с определения целевой платформы. Для этого создается специальный файл описания архитектуры, в котором задается конфигурация оборудования. Информация из этого файла используется линкером, симулятором и эмулятором ADSP. На рис. 14 показан процесс разработки программы для микропроцессоров семейства ADSP-21xxx.
- Создание программы начинается с написания файлов исходного кода на языке ассемблера. Каждый файл впоследствии компилируется отдельно. Затем несколько откомпилированных файлов соединяются в выполняемую программу при помощи линкера.

При объединении файлов линкер использует информацию из файла описания конфигурации для определения расположения сегментов кода и данных в памяти. Неперемещаемые сегменты кода и данных размещаются по заданным адресам памяти, в том случае если эти области памяти имеют верные атрибуты.

Линкер создает единый файл с выполняемой программой, которая затем может быть загружена в симулятор или эмулятор для тестирования.

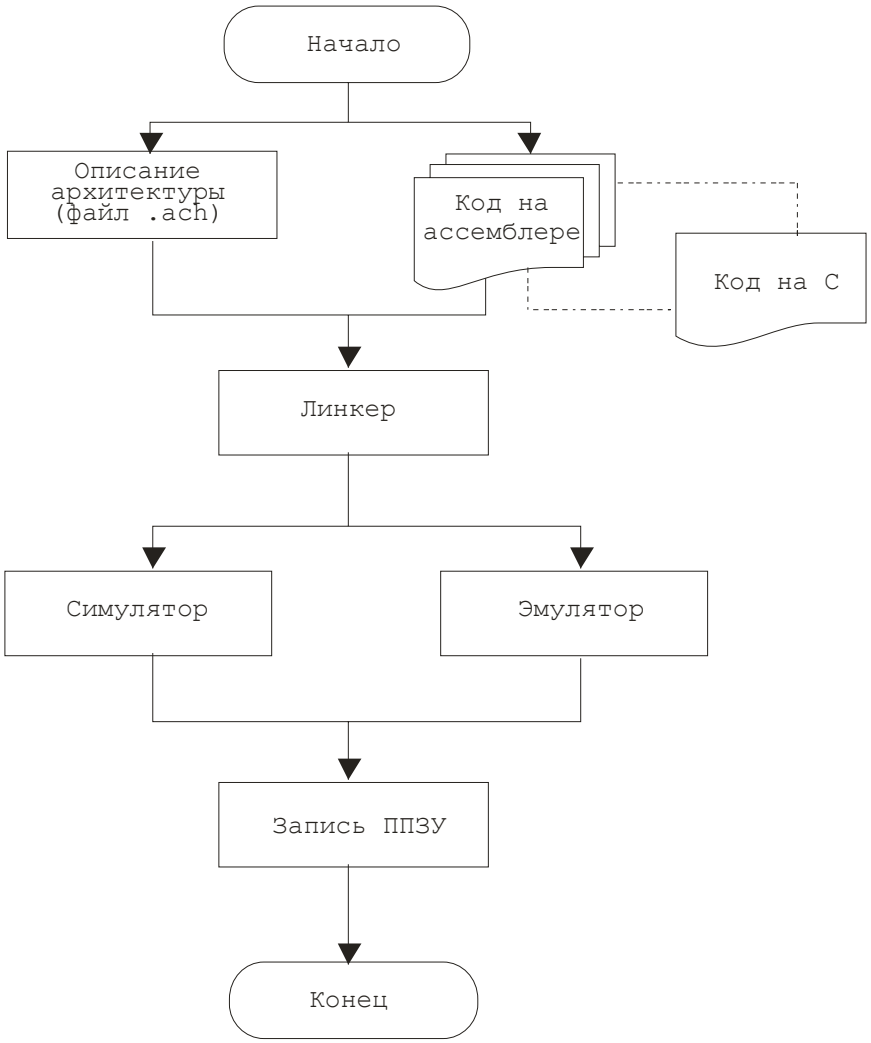


Рис. 14. Разработка систем на основе ADSP-21xxx

Подготовка исходного кода

Исходный код программы может создаваться при помощи любого простого текстового редактора, не оставляющего в файле служебной информации (например программы «Блокнот» из комплекта поставки Microsoft Windows). Кроме того, существуют профессиональные средства разработки приложений ЦОС, позволяющие создавать программы с использованием языка С и включающие в себя развитые средства отладки и тестирования программ. К числу таких средств принадлежит среда разработки Visual DSP.

Ассемблирование кода

Ассемблер для семейства ADSP-21000 переводит текст программы на языке ассемблера в объектный код. В случае если файлов с текстами несколько, они компилируются по отдельности и затем соединяются в один выполняемый модуль при помощи линкера.

Ассемблер состоит из двух компонент:

1. Препроцессор (ASM21K.EXE)
2. Ассемблер (A21000.EXE)

Обычно при запуске ассемблера запускается препроцессор и, затем, собственно ассемблер. При помощи ключей, вводимых в командной строке (см. табл. 7) можно запустить тот или иной компонент отдельно.

Для запуска ассемблера используется команда:

ASM21K [-ключ ...]*filename*[.ext]

При помощи необязательных ключей можно варьировать условия компиляции. Ключи могут вводиться как в верхнем, так и в нижнем регистрах. Если используются несколько ключей, они должны отделяться друг от друга минимум одним пробелом. *Filename.ext* - это имя входного файла. Файл может иметь любое расширение; если расширение не указывается, ассемблер считает что применяется стандартное расширение **.ASM**. Для вызова подсказки используется ключ -h:

ASM21K-h

При этом на экран будет выведен синтаксис командной строки ассемблера и список возможных ключей. Любая из программ входящих в набор для разработки приложений отображает такой список в том случае, если она запускается с ключом -h (help) (или если в синтаксисе была допущена ошибка).

Таблица 7. Возможные ключи ассемблера и их аргументы

Ключ	Действие
-ADSP21060	Разрешить использование инструкций 21060
-pp	Запуск только препроцессора
-sp	Запуск только ассемблера (без препроцессора)
-didentifier[=literal]	Задание идентификатора для препроцессора
-l	Создание файла листинга (.LST)
-o name	Имя для выходного файла (.OBJ)
-v	Вывод дополнительной информации
-h	Помощь

Ограничения ассемблера

Ассемблер для семейства ADSP-21000 имеет ряд ограничений:

- Подключаемые файлы должны иметь расширение .h

- При использовании обратной косой черты в качестве символа продолжения строки (\) после неё на той же строке не должно находиться ни одного символа; сразу за косой чертой должен следовать символ возврата каретки. В противном случае (например, при пробелах или комментариях после обратной косой черты) ассемблер выдает ошибку.
- При использовании инструкции Rn=FDEP Rx BY <bit6>:<1епб> в позиции <bit6> могут использоваться только положительные числа. Отрицательные числа в этой позиции вызывают ошибку синтаксиса, не обрабатываемую ассемблером.

Линкер

Линкер семейства ADSP-21xxx объединяет откомпилированные по отдельности файлы в единую запускаемую программу. Основной файл, создаваемый линкером это **файл образа памяти** программы с расширением .EXE. Этот файл впоследствии загружается в симулятор или эмулятор для отладки. После отладки применяется программа прошивки PROM для загрузки программы в целевое изделие.

Как упоминалось в главе посвященной ассемблеру, после компиляции для каждого файла с исходным текстом создаются файлы объектного кода (*.OBJ). Линкер использует данные из файла описания архитектуры (*.ACH) и одного или нескольких объектных файлов (*.OBJ), объединяя их в файл образа памяти (*.EXE). Кроме того, линкер может искать и присоединять к программе код из файлов-библиотек, которые создаются при помощи программы-библиотекаря. На рис. 15 показаны входные и выходные файлы линкера.

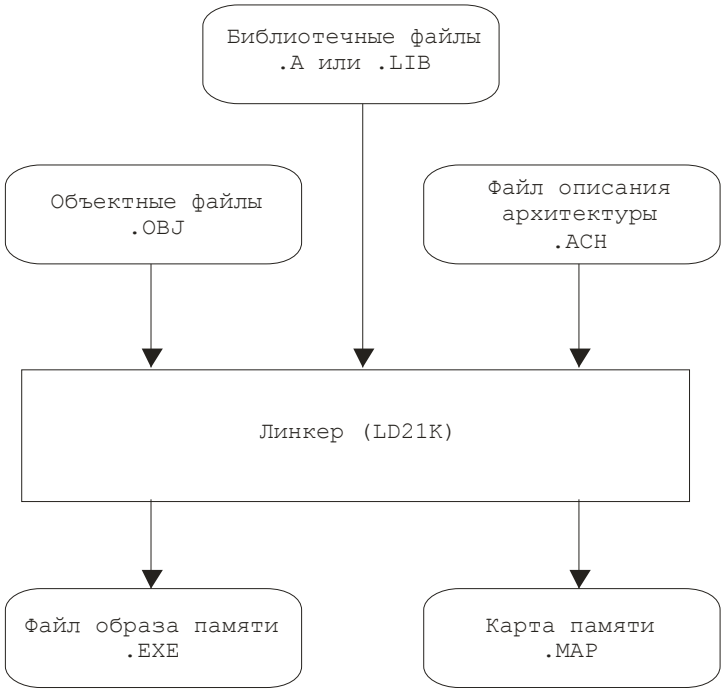


Рис. 15. Входные и выходные файлы линкера

Запуск линкера

Обычно линкер запускается следующей командой:

ld21k[-ключ ...] файл1 [файл2 ...]

Если файлы находятся не в текущей директории, то вместе с каждым именем файла следует указать путь к нему. Имена файлов должны ссылаться на файлы созданные ассемблером. Если не указано их расширение, подразумевается расширение .OBJ. В том случае если имя выходного файла не указано явным образом, ему присваивается имя первого из входных файлов и расширение .EXE.

Если вы забыли синтаксис командной строки, можно применить ключ -h:

LD21K-h

При этом на экран будет выведен синтаксис командной строки линкера и список возможных ключей. Любая из программ входящих в набор для разработки приложений отображает такой список в том случае, если она запускается с ключом -h (help) (или если в синтаксисе была допущена ошибка).

Таблица 8. Возможные ключи линкера их аргументы

Ключ	Действие
-a archfile	Задается файл описания архитектуры archfile.ach
-o executable	Выходному файлу присваивается имя executable
-i file_all	Список объединяемых файлов берется из файла file_all
-m	Создается файл .MAP
-s	Из исполняемого файла исключается таблица символов
-x	Исключается информация о локальных символах
-v	Вывод дополнительной информации
-h	Помощь

Ограничения линкера

Линкер для семейства ADSP-21xxx имеет следующее ограничение:

- Если библиотечный файл содержит имя сегмента, которое ассемблером было обрезано до восьми символов, линкер может выдавать ошибку несовпадения имени сегмента. Не применяйте названия сегментов длиннее восьми символов, поскольку, например имя сегмента

«too_long_segment_name» будет сокращено до «too_long».

ОТЛАДКА И ТЕСТИРОВАНИЕ

После компиляции всех файлов с исходными текстами программы и объединения их в исполняемый файл, созданное приложение можно протестировать при помощи программы-симулятора, позволяющей осуществлять пошаговое выполнение команд, контроль состояния регистров процессора, наблюдение за состоянием стека и памяти и т.п.

Для запуска симулятора из Windows следует открыть группу **SHARC EZ-KIT Lite** и выбрать пункт **EZ-KIT Lite Simulator**. При старте симулятор загружает стандартный файл описания конфигурации (21k.ach), или файл описания, путь к которому задан в конфигурационном файле симулятора wsim060.ini. Для загрузки исполняемого файла сначала необходимо очистить память симулятора командой меню **Execution|Simulator Reset**, а затем воспользоваться командой меню **File|Load File** для загрузки нового файла.

3. ЛАБОРАТОРНЫЕ РАБОТЫ ПО ИССЛЕДОВАНИЮ АЛГОРИТМОВ ЦОС НА SHARC EZ-KIT LITE

ЛР №1: Умножение матрицы MxN на вектор Nx1

Умножение матрицы на вектор выполняется при помощи двухуровневого (вложенного) цикла. Во внутреннем цикле — *row* — выполняется перемножение двух элементов, прибавление полученного значения к результату предыдущих перемножений и выборка двух следующих значений для перемножения (все операции выполняются за один такт!). Этот цикл выполняется N раз (по числу столбцов).

Внешний цикл — *column* — сохраняет результат перемножения в буфере mat_c, и очищает аккумулятор R8 для следующего выполнения внутреннего цикла. Этот цикл выполняется M раз (по числу строк).

Программа, реализующая данный алгоритм приведена в таблице 9.

Таблица 9. Умножение матрицы MxN на вектор Nxl

```
/* константы размерности */
#define        M 4
#define        N 4

#ifndef example
.GLOBAL mxnxnx1;
.EXTERN mat_a, mat_b, mat_c;
#endif

#ifdef example
.SEGMENT/DM dm_data;
.VAR  mat_a[M*N]="mat_a.dat";
.VAR  mat_c[M];
.ENDSEG;

.SEGMENT/PM pm_data;
.VAR mat_b[N]="mat_bb.dat";
.ENDSEG;

.SEGMENT/PM rst_svc;
        dmwait=0x21;
        pmwait=0x21;
        jump setup;
.ENDSEG;

.SEGMENT/PM pm_code;
setup:  m1=1;
        m9=1;
        b0=mat_a; l0=@mat_a;
        b1=mat_c; l1=0;
        b8=mat_b; l8=@mat_b;
        call mxnxnx1;
        idle;
.ENDSEG;
#endif

        /* здесь начинается перемножение */
```

```
.SEGMENT/PM pm_code;
        /* очистка f8 */
mxnxnx1:    r8=r8 xor r8, f0=dm(i0,m1), f4=pm(i8,m9);          f12=f0*f4,
f0=dm(i0,m1), f4=pm(i8,m9);
            lcntr=M, do column until lce;
            lcntr=N, do row until lce;
row:    f12=f0*f4, f8=f8+f12, f0=dm(i0,m1), f4=pm(i8,m9);
column:     r8=r8 xor r8, dm(i1,m1)=f8;
            rts;
.ENDSEG;
```

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

ЛР №2: Трансверсальный КИХ-фильтр

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.

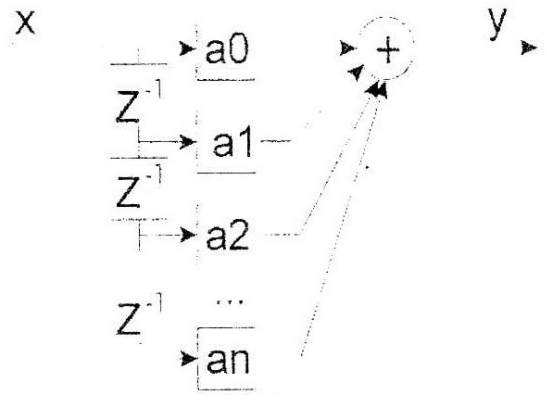


Рис. 16. Структура КИХ-фильтра

В таблице 10 приведена программа трансверсального КИХ-фильтра

Таблица 10. Программа КИХ-фильтра

```
.GLOBAL fir;
.EXTERN coefs,dline;
.SEGMENT/PM seg_pmco;
fir:  r12=r12 xor r12, dm(i0,m0)=f0;
      r8=r8 xor r8, f0=dm(i0,m0), f4=pm(i8,m8);
   lcntr=r1, do macs until lce;
macs: f12=f0*f4, f8=f8+f12, f0=dm(i0,m0), f4=pm(i8,m8);
   rts (db);
   f12=f0*f4, f8=f8+f12;
   f0=f8+f12;
.ENDSEG;
```

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.

ЛР №3: БИХ - фильтр

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. 17.

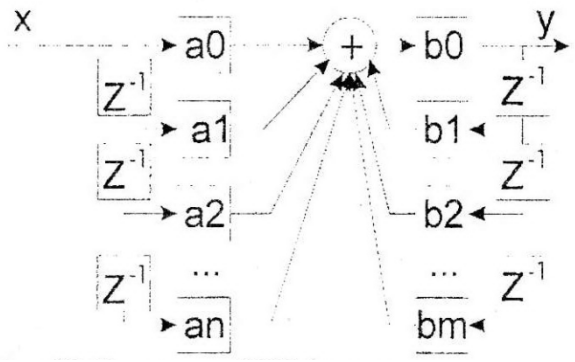


Рис. 17. Структура БИХ-фильтра

В таблице 11 приведена программа рекурсивного БИХ-фильтра.

Таблица 11. Функция БИХ-фильтра

```
.GLOBAL cascaded_biquad;

.SEGMENT/PM seg_pmco;
cascaded_biquad:
   b1=b0;
   f12=f12-f12, f2=dm(i0,m1), f4=pm(i8,m8);
   lcntr=r0, do quads until lce;
   f12=f2*f4, f8=f8+f12, f3=dm(i0,m1), f4=pm(i8,m8);
   f12=f3*f4, f8=f8+f12, dm(i1,m1)=f3, f4=pm(i8,m8);
   f12=f2*f4, f8=f8+f12, f2=dm(i0,m1), f4=pm(i8,m8);
quads:
   f12=f3*f4, f8=f8+f12, dm(i1,m1)=f8, f4=pm(i8,m8);

    rts (db), f8=f8+f12;
    nop;
    nop;
.ENDSEG;
```

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

ЛР №4: Дискретное преобразование Фурье

Дискретное преобразование Фурье (DFT) широко используется в программах цифровой обработки сигналов. В таблице 12 приведена программа преобразования Фурье.

Таблица 12. Программа дискретного преобразования Фурье

```
#define N 64    /* Число входных отсчетов */

.SEGMENT/DM    seg_dmda;
.VAR input[N]= "test64.dat";
.VAR real[N];
.VAR imag[N];
.ENDSEG;

.SEGMENT/PM    seg_pmda;
.VAR sine[N]= "sin64.dat";
.ENDSEG;

.SEGMENT/PM    seg_rth;
               nop; nop; nop; nop;
               NOP;
               USTAT2= 0x108421;
               DM(WAIT)=USTAT2;
               JUMP start;

.ENDSEG;

.SEGMENT/PM    seg_pmco;
start:      M1=1;
               M9=1;
               B0=input;
               L0=@input;
               I1=imag;
               L1=0;
               CALL dft (DB);
                I2=real;
                L2=0;
end:        IDLE;

/* Подпрограмма вычисления DFT */

dft:        B8=sine;
               L8=@sine;
               B9=sine;
               I9=sine+N/4;
               L9=@sine;
               I10=0;
               L10=0;
               F15=0;
               LCNTR=N, DO outer UNTIL LCE;
                F8=PASS F15, M8=I10;
                F9=PASS F15, F0=DM(I0,M1), F5=PM(I9,M8);
                F12=F0*F5, F4=PM(I8,M8);
                LCNTR=N-1, DO inner UNTIL LCE;
                 F13=F0*F4, F9=F9+F12, F0=DM(I0,M1), F5=PM(I9,M8);
inner:          F12=F0*F5, F8=F8-F13, F4=PM(I8,M8);
                F13=F0*F4, F9=F9+F12;
                F8=F8-F13, DM(I2,M1)=F9;
                MODIFY(I10,M9);
outer:         DM(I1,M1)=F8;
                RTS;
.ENDSEG;
```

Измерить время выполнения программы, приведенной в примере, и количество MIPS, которые занимает эта программа, оптимизировать исходный код по критерию минимального значения MIPS.

ПРИЛОЖЕНИЕ

ПОРЯДОК ПРОВЕДЕНИЯ ПРАКТИЧЕСКИХ РАБОТ

1. Получить у преподавателя номера выполняемых работ и номер варианта для работы.
2. Подключить EZ-KIT к COM-порту компьютера
3. Подключить разъем питания
4. Нажать клавишу Reset
5. Провести контроль функционирования
6. Разработать блок-схему реализуемого алгоритма согласно варианту задания
7. Разработать, ввести и отладить программу
8. Оценить результаты работы программы
9. Отключить EZ-KIT от компьютера
10. Отключить питание EZ-KIT
11. Оформить отчет

СОДЕРЖАНИЕ ОТЧЕТА

Отчет студента должен содержать:

1. Краткий теоретический конспект с ответами на контрольные вопросы
2. Функциональную схему процессора ADSP-21061
3. Блок-схемы и алгоритмы разработанных программ
4. Листинги разработанных программ.
5. Методику и результаты оценки достоверности полученных результатов.

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 Instalments, 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

1. ГОСТ 2.701-84 Схемы. Виды и типы. Общие требования к выполнению
2. ГОСТ 2.114-95 Технические условия
3. ГОСТ 19.002-80 Схемы алгоритмов и программ. Правила выполнения
4. ГОСТ 19.003-80 Схемы алгоритмов и программ. Обозначения условные графические
5. ГОСТ 19.004-80 Термины и определения
6. ГОСТ 19.101-77 Виды программ и программных документов
7. ГОСТ 19.102-77 Стадии разработки
8. ГОСТ 19.103-77 Обозначения программ и программных документов
9. ГОСТ 19.104-78 Основные надписи
10. ГОСТ 19.105-78 Общие требования к программным документам
11. ГОСТ 34.201 -89 Информационная технология. Виды, комплектность и обозначения документов при создании автоматизированных систем
12. ГОСТ 34.603-92 Информационная технология. Виды испытаний автоматизированных систем
13. ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом
14. ГОСТ 19.201-78 Техническое задание, требования к содержанию и оформлению
15. ГОСТ 19.202-78 Спецификация. Требования к содержанию и оформлению
16. ГОСТ 19.301-79 Программа и методика испытаний. Требования к содержанию и оформлению
17. ГОСТ 19.401-78 Текст программы. Требования к содержанию и оформлению
18. ГОСТ 19.402-78 Описание программы
19. ГОСТ 19.403-79 Ведомость держателей подлинников
20. ГОСТ 19.503-79 Руководство системного программиста. Требования к содержанию и оформлению
21. ГОСТ 19.504-79 Руководство программиста. Требования к содержанию и оформлению
22. ГОСТ 19.506-79 Описание языка. Требования к содержанию и оформлению

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

СОДЕРЖАНИЕ