ВВЕДЕНИЕ

В настоящее время наблюдается непрерывный рост интереса специалистов-разработчиков к использованию в различных устройствах обработки сигналов цифровых сигнальных процессоров (DSP) производства компании Analog Devices, что обусловлено удобствами применения, доступностью и широкими возможностями самих процессоров.

На сайте Analog Devices (*www.analog.com*) доступно большое количество различных примеров, документации, заметок по применению. Использование этого объема информации существенно облегчает процесс освоения как самого VisualDSP++, так и сигнальных процессоров компании Analog Devices.

Лабораторные работы направлены на изучение основ работы в среде VisualDSP++ (создание проекта, компиляция, отладка, графическое отображение данных), а также приложений цифровой обработки сигналов (проектирование цифровых фильтров на базе DSP и пр.).

Процесс отладки проекта после написания кода программы и успешной компиляции включает три основных этапа:

1) моделирование (Simulation);
2) оценка (Evaluation);
3) эмуляция (Emulation).

При работе в режиме моделирования не требуется наличие сигнального процессора, поскольку VisualDSP++ имитирует его работу с помощью моделирующей программы (симуляторы). Симулятор используется для проверки и отладки программного кода до того, как будет изготовлена плата с процессором.

На втором этапе используется оценочная плата EZ-KIT для того, чтобы определить, какой процессор наилучшим образом подходит для решения конкретной задачи. Плата подключается к компьютеру с помощью кабеля через параллельный, последовательный или USB-порт.

На третьем этапе, когда устройство уже изготовлено, можно выполнить тестирование платы с помощью специального аппаратно-программного модуля — эмулятора. Этот модуль управляет цифровым сигнальным процессором через JTAG-интерфейс и позволяет подробно отследить выполнение программного кода.

В ходе выполнения данных лабораторных работ студенты не просто ознакомятся с архитектурой, системой команд, средствами проектирования и отладочной платой, но и сами проведут полный цикл разработки программы:

· написание программы;
· ее отладка;
· наблюдение за ходом выполнения программы в системе.

В результате этого курса студентам будет предоставлена возможность самим ознакомиться с платой и провести весь цикл разработки, вследствие чего они должны получить ценный практический опыт.

1. ОСНОВЫ РАБОТЫ С VISUAL DSP ++

1.1. ЗАПУСК VISUALDSP ++

Щелкните мышкой по ярлыку «VisualDSP». При первом запуске на экране монитора появится окно Session list. Выберете пункт New session (его также можно вызвать, выбрав меню *Session –> New Session*), в котором необходимо установить соответствующие параметры:

- в поле Debug target выберите пункт «EZ-KIT 218x»;
- в поле Platform выберите «EZ-KIT 218x» (или Platform 0);
- в поле Processor задайте необходимый тип процессора (ADSP-2181).

После этого нажмите Activate. Появится окно *218x serial Communication Settings*, в котором просто нажмите Ok. При появлении окна *Reset* ("Please press the EZ-KIT reset button") нажмите на кнопку RESET на плате, немного подождите и нажмите на Ok.

Далее появится окно среды разработки VisualDSP ++ в режиме отладки программы. Основные окна — окно проекта *Project*, в котором размещаются содержащиеся в проекте файлы и папки, окно дизассемблированного кода *Disassembly*, окно выходной информации *Output Window*, содержащее стандартные текстовые сообщения ввода-вывода, сообщения об ошибках и пр.

Необходимо отметить, что внешний вид среды разработки полностью настраивается под пользователя, позволяя отображать именно ту информацию, которая на данный момент требуется.

1.2. СОЗДАНИЕ НОВОГО ПРОЕКТА VISUALDSP ++

Для создания нового проекта войдите в меню *Project –> New* и в открывшемся диалоговом окне введите название проекта, затем задайте, где он будет храниться на жестком диске (по указанию преподавателя). В появившемся окне *Project Options* установите все необходимые значения, задающие параметры построения системы:

1) в закладке Project:

- в поле Processor установите «ADSP-2181»;
- в поле Type - «DSP executable file» (исполняемый файл);
- в поле Setting for configuration - «Debug».

2) проверьте наличие галочек в соответствующих полях закладок:

-в закладке VIDL (в поле Assembly Source);

-в закладке Assemble (в поле Output Listing File, Generate Debug Information);

Согласно этим установкам исполняемый файл генерируется для процессора ADSP-2181, отладчик может использоваться для отслеживания процесса выполнения программы.

1.3. ОСНОВНЫЕ РАЗДЕЛЫ МЕНЮ СРЕДЫ VISUALDSP ++

Строка главного меню расположена в верхней части рабочего окна. Ниже главного меню находится полоса инструментальных панелей. На каждой панели расположена группа кнопок, соответствующих некоторым командам меню (рис.1.1). Рассмотрим основные разделы главного меню.

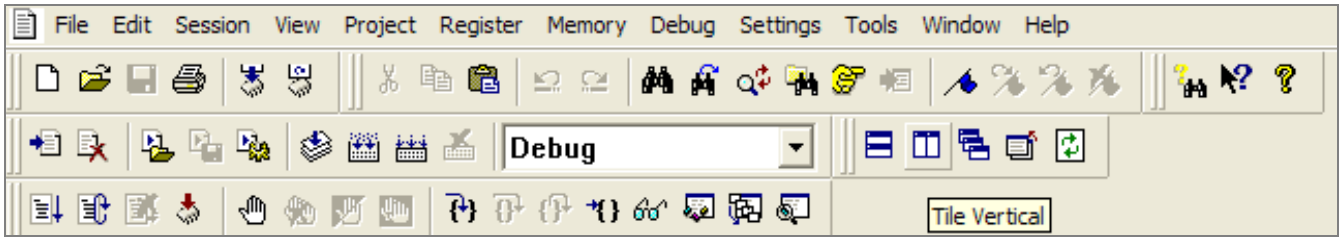


Рисунок 1.1 - Главное меню и панели инструментов VisualDSP ++

Раздел меню ***File*** содержит в себе стандартные операции по сохранению, открытию файлов. Вид рабочего окна можно сохранить (отображаемые при отладке регистры, области памяти и т.п.), выбрав *File –> Workspace –> Save*. Здесь же осуществляется настройка параметров печати и т.д.

В разделе меню ***Edit*** располагаются основные операции редактирования текста программы: вставка *(Past)*, копирование *(Copy)*, поиск слова в тексте *(Find)*, просмотр ошибок *(Previous Error)* и т.п.

Семейство процессоров ADSP или определенный тип процессора можно сменить, выбрав соответствующую вкладку в разделе меню ***Session.***

Раздел меню ***View*** позволяет настроить вид рабочей области для удобной работы. Так, например, можно отображать на экране:

- строку текущего состояния (*Status bar*);
- таблицу редактирования (*Editor tab*);
- окно проекта (*Project Window*);
- окно выходной информации (*Output Window*);
- ряд окон отладки (*Debug Windows*). Это такие окна как: окно дизассемблированного кода (*Disassembly*), окно трассировки (*Trace*), окно выражений (*Expression*), окно стека вызовов (*Call stack*), окно вывода графических результатов (*Plot –> New*) и др.

С помощью команд раздела меню ***Project*** можно создать новый проект (*Project –> New*), добавить файлы и новые папки в готовый проект (*Add to Project*), осуществить компиляцию файла (Ctrl+F7) и компоновку проекта (*Build Project* или клавиша F7), изменить свойства проекта (*Project Options*) и т.д. Перед компиляцией требуется создать линкера *.ldf (см. раздел меню *Tools*). Если компиляция и компоновка прошли успешно, создается объектный файл проекта и образ памяти для загрузки в симулятор.

Register. Это подменю позволяет настроить вид рабочей области. Содержимое регистров в процессе работы можно посмотреть, выбирая необходимые регистры из списка.

- *Computational* – вычислительные регистры (AX, AY, AR, AF, MX, MY, MF, MR, SR, SI, SE, SB);
- *DAGs* – регистры генератора адреса данных (I0-I7, M0-M7, L0-L7);
- *Program Control* – регистры управления программой (PC, CNTR, CYCLES, PM_ADDR, DM_ADDR…);
- *Status* – регистры состояния (ASTAT, SSTAT, MSTAT, AZ, AV, AN, AC, AS, AQ, MV, SS…)
- *Interrupt* – регистры прерываний (IREQ, IMASK, ICNTL, IFC);
- *Control* – регистры управления (TPERIOD, TSCALE, TCOUNT, SPE0…);
- *Flag* – флаги (FI, FO, FL0, FL1, FL2);
- регистр *CE*;
- регистр *PX*;
- *SPOR0, SPORT1* – регистры портов SPOR0 и SPORT1 (TX0, TX1, RX0, RX1, RFSR0, TFSR0…);
- *DMA* – регистры для прямого доступа к памяти (BEAD, BCR, IDMAD…);
- *Custom* – выборочные регистры, т.е. для отображения только тех регистров, просмотр которых необходим при отладке программы;
- *Stack* – стеки (стек счетчика команд (*PC Stack*), стек счетчика (*Counter Stack*), стек циклов (*Loop Stack*), стек состояний (*Status Stack*)).

Memory. Для того чтобы посмотреть распределение исходных массивов данных в памяти, необходимо, выбрав в этом разделе меню пункт *Data*, вызвать окно отображения содержимого области памяти данных. Для отображения памяти программ надо выбрать *Program*.

Раздел меню ***Debug*** предназначено для управления отладкой проекта с помощью следующих команд:

- *Run* – запустить программу на исполнение;
- *Halt* – останов программы (содержимое ОЗУ не теряется);
- *Step into* (клавиша F11)– запустить программу на исполнение следующей строки кода, т.е. пошаговое исполнение программы;
- *Run to cursor* – запустить программу на исполнение до курсора;
- *Reset* – сброс, приведение программы в исходное состояние;
- *Restart* – перезапуск программы.

Раздел меню ***Settings*** требуется для задания параметров настройки для конкретного проекта, например, таких как: точка останова (*Breakpoint*), точка наблюдения (*Watchpoint*), установка прерываний (*Interrupts*) и т.д.

Раздел меню ***Tools*** позволяет осуществлять настройку инструментария для конкретного проекта. Выбрав *Expert Linker –> New* можно создать файл линкера, необходимый для отладки проекта.

Раздел меню ***Window*** позволяет изменить расположение окон редактора кода в рабочей части главного окна. Так, их можно расположить горизонтально (Tile Horisontally), вертикально (Tile Vertically), каскадом (Cascade). Можно также обновить расположение окон (Refresh) и т.д.

Для получения подробной информации о работе в среде VisualDSP ++ необходимо воспользоваться меню ***Help***.

1.4. РАЗРАБОТКА ПРОГРАММНОГО КОДА

Исходный код программы может быть записан на языке ассемблер или создан С компилятором. Ассемблер переводит вашу программу, написанную на языке ассемблер, в объектный код. Создать исходный код можно с помощью любого простого текстового редактора. Не используйте текстовые процессоры, например, Microsoft Word, которые вставляют специальные управляющие коды.

Написание программы заключается в создании исходных секций на языке АССЕМБЛЕР. Каждая секция имеет свое название. В теле программы могут находиться директивы объявления и инициализации переменных, констант, массивов команды и комментарии. Директивы начинаются со знака «**.**». Комментарии отмечаются как: ***/*{comments}*/*** или ***//{comments}***. После каждой команды или директивы должен стоять знак *«;»*.

При создании программ широко используется косвенная адресация, которая использует генераторы адреса данных. При этом используется три регистровых файла: файл регистров модификации (M0-M3 и M4-M7), файл индексных регистров (I0-I3 и I4-I7) и файл регистров длины (L0-L3 и L4-L7). Индексные регистры I0-I3 содержат действительные адреса, используемые для доступа к памяти данных. I4-I7 используются для доступа как к памяти программ, так и данных. Система команд ADSP-218x поддерживает *постмодификацию*. После обращения к памяти с использованием индексного регистра значение этого регистра может быть изменено на величину, хранящуюся в указанном регистре модификации M.

DM(I2,M2) - обращение к памяти данных,

PM(I5, M5) - к памяти программ.

Существует правило, по которому индексные регистры одного генератора адреса данных не могут использоваться совместно с регистрами модификации Mx другого (нельзя: DM(I1,M7)). Регистры Mx являются числами со знаком, по этому модификация может осуществляться как в сторону увеличения, так и уменьшения.

Буфер данных- это множество адресов расположения данных. Параметр CIRC в строке объявления буфера определяет циклический буфер. Если атрибут CIRC не применен, то буфер будет адресоваться как линейный. Если регистр длины Lx = 0, то используется линейный буфер. Если Ln=N, где N - положительное число, то соответствующий регистр In используется для обращения к циклическому буферу. Если In+Mx выходит за пределы буфера, то осуществляется модификация In по модулю Ln.

После перезапуска системы, ADSP-2181 начнет выполнение кода из памяти

программ со стартового адреса 0, по которому расположена инструкция jump start.

При прерывании исполнение программы останавливается и переходит к адресу, указанному в таблице векторов прерываний. Кодовый сегмент, включающий таблицу векторов прерываний, размещается в первых 48 адресах памяти программ.

Неиспользованные адреса в памяти вектора прерываний заполняют инструкцией rti, которая обеспечивает перенос исполнения обратно в основную программу.

Основания счисления, которые могут быть использованы в исходной программе, включают шестнадцатеричную, восьмеричную, двоичную и десятичную. Их обозначают следующим образом:

- для *шестнадцатеричных* чисел, префикс **0x:** ***0x**24FF*
- для *восьмеричных* чисел, префикс **0** (ноль): ***0**777*
- *двоичные* числа обозначаются с префиксом **b#:** **b#***01110100*
- *десятичные* числа используются без префикса: *1024 +1024 -55*

Набор символов. Программное обеспечение средств разработки ADSP-21xx распознает следующие символы:

- буквы верхнего регистра от “A” до “Z”;
- буквы нижнего регистра от “a” до “z”;
- цифры от “0” до “9”;
- ASCII графические символы - т.е. печатные символы отличные от букв и цифр (пунктуации и т.д.)
- ASCII не графические: пробел, табуляция, возврат каретки, перевод строки, перевод страницы.

Чувствительность к регистру. Программное обеспечение средств разработки на языке ассемблер может быть установлено как чувствительным к регистру, с различением между буквами верхнего и нижнего регистров, так и нечувствительным к регистру, принимая буквы верхнего и нижнего регистров за одни и те же знаки (по умолчанию - отключена чувствительность).

Имя переменной может быть длинной до 32 символов и не может начинаться с цифры. Например, *input_array; prog; buffer*…

2. ОПИСАНИЕ АППАРАТНОЙ ЧАСТИ EZ-KIT LITE

EZ-KIT Lite является одной из самых эффективных по стоимости, мощной системой разработки доступной на рынке. В EZ-KIT Lite используется ADSP-2181 – 16-разрядный DSP процессор высокой интеграции с фиксированной точкой и встроенной памятью 32К слов ОЗУ, временем выполнения командного цикла 30нс, портами прямого доступа к памяти и режимами пониженного энергопотребления.

EZ-KIT Lite предоставляет простой путь исследования возможностей процессоров семейства ADSP-21xx и разработки собственных приложений для этих DSP. EZ-KIT Lite является законченной системой разработки, идеальной для начала работы с DSP, которая позволит пройти через все фазы процесса разработки. С EZ-KIT Lite можно:

- оценить DSP фирмы Analog Devices;
- изучить применение DSP;
- разрабатывать DSP-приложения;
- моделировать и проводить отладку приложений;
- создавать опытные образцы.

Спецификации:

Процессор:	ADSP-2181KS-133, работающий с частотой 33МГц (16,667 МГц внешней синхронизации)
Аналоговый интерфейс:	AD1847 стерео кодек
Аналоговые входы:	Одна стереопара с уровнем сигнала 2В эффективного значения переменного тока, связанная с линейными входами Одна стереопара с уровнем сигнала 20мВ эффективного значения переменного тока, связанная с микрофонными входами
Аналоговые выходы:	Одна стереопара с уровнем сигнала 1В эффективного значения переменного тока, связанная с линейными выходами
Мощность:	от 8 до 10В постоянного тока @ 300мА
Окружающая среда:	от 0 до 70 С, от 10% до 90% относительной влажности

Аппаратная часть EZ-KIT Lite состоит из смонтированной печатной платы с размерами 8.9 x 14 см (рисунок 2.1). На печатной плате расположены цифровой сигнальный процессор ADSP-2181, микросхема ППЗУ, кодек AD1847, а также различные схемы поддержки и соединители. Плата представляет собой завершенную систему обработки сигналов, спроектированную для демонстрации возможностей цифрового сигнального процессора ADSP-2181.

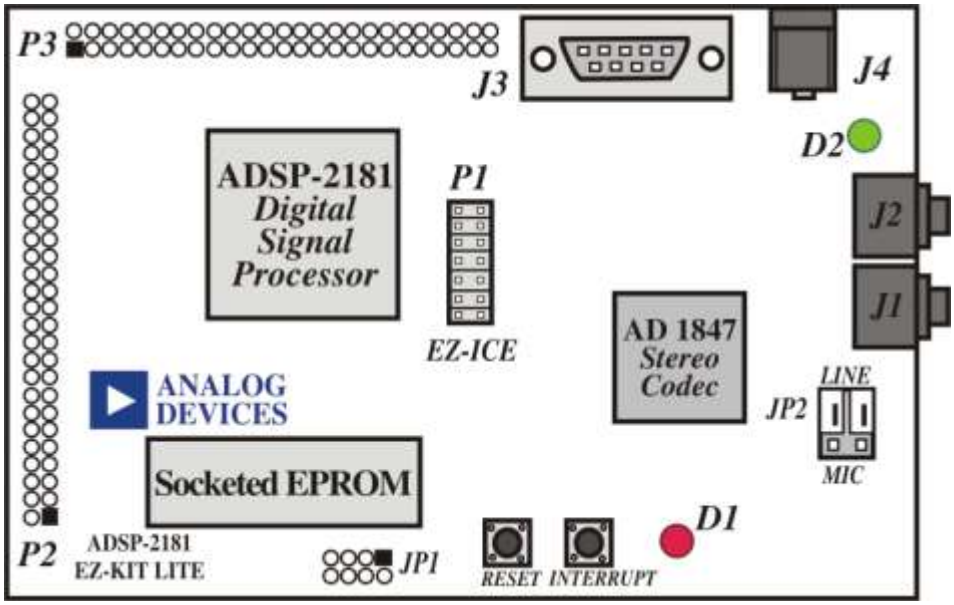


Рисунок 2.1 - Расположение основных элементов на печатной плате EZ-KIT Lite

ADSP 2181 – 16-разрядный микропроцессор, поддерживающий вычисления с фиксированной точкой. Применяется в обработке и сжатии аудио и речевых сигналов, модемах стандартов V.32bis и V.34, многоканальной телекоммуникации, медицинской технике и т.д.

Микросхема *EPROM* (ППЗУ) соединена с процессором через DMA порт.

Кодек *AD1847* подключен к DSP через один из последовательных портов процессора SPORT0. Этот высокоскоростной синхронный последовательный порт передает все данные, управляющие сигналы и информацию о состоянии между DSP и кодеком.

J1 - стерео гнездо, которое используют для ввода линейных аудио сигналов или микрофонных звуковых сигналов.

J2 - стерео гнездо, которое используют для вывода линейных аудио сигналов.

J3 - разъем штепсельного типа. Он используется для связи с управляющим ПК, используя сигналы RS-232 и асинхронный последовательный протокол.

J4 - цилиндрический штекер, который используется для подачи питания на плату.

JP1 - площадка многоконтактного соединителя на восемь выводов. Используется для конфигурирования платы под объем ППЗУ, отличный от 128 Кб микросхемы EPROM.

JP2 - площадка многоконтактного соединителя на шесть выводов. Используется для конфигурирования входного гнезда (J1), переключая режим работы с линейного на микрофонный вход.

P1 - 14-выводный соединитель, используемый для включения внутрисхемного эмулятора ADSP-2181 EZ-ICE.

P2 и *P3* - являются площадками для 50-выводных соединителей. Могут использоваться для доступа к сигналам ADSP-2181 в целях расширения или тестирования.

Кнопка перезапуска (*RESET*). Нажатие этой кнопки заставляет процессор ADSP-2181 и кодек AD1847 перейти в состояние аппаратного сброса и оставаться в нем до того, пока кнопка не будет отпущена.

Кнопка прерывания (*INTERRUPT*). Нажатие этой кнопки заставляет процессор вызвать прерывание IRQE. Это может заставить процессор выполнить обработчику прерывания, если оно разрешено и указан вектор обработчика прерывания IRQE

D1 - красный светодиод, который управляется выводом FL1 процессора ADSP-2181. Программное обеспечение может управлять состоянием этого индикатора записью в соответствующий внутренний регистр.

D2 - зеленый светодиод, который горит, если плата подключена к источнику питания.

Когда к плате подключено питание, цепь сброса держит процессор в течение 30 мс. После этого процессор начинает операцию загрузки. Если зеленый светодиод не горит, проверьте подключение источника питания. Нажмите кнопку сброса, если плата работает неправильно.

3. ПРОГРАММИРОВАНИЕ ПОСЛЕДОВАТЕЛЬНЫХ ПОРТОВ ПРОЦЕССОРА ADSP-218X

3.1. ФУНКЦИОНАЛЬНАЯ СХЕМА ПОСЛЕДОВАТЕЛЬНОГО ПОРТА

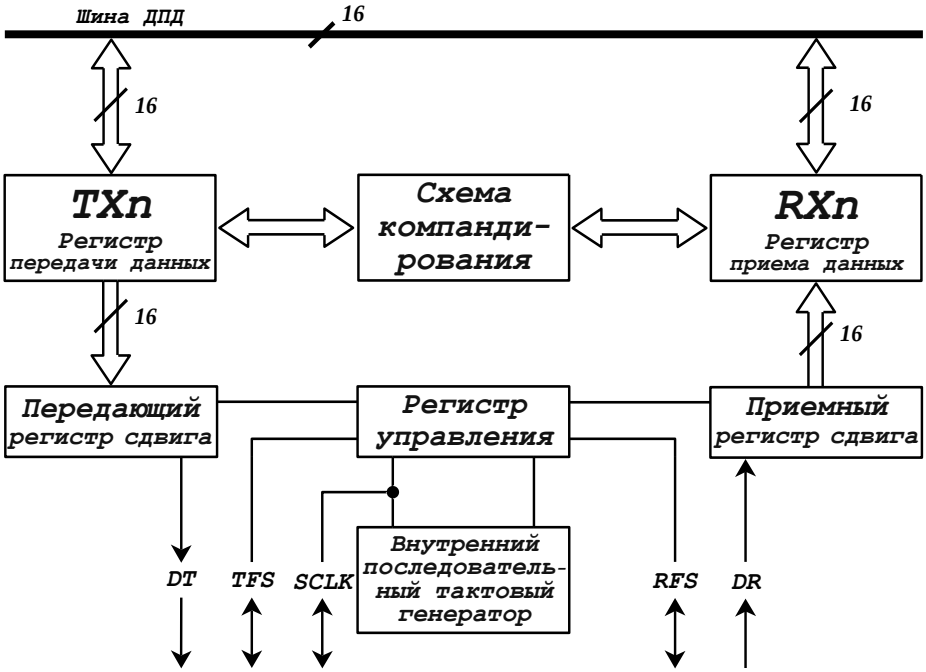


Рисунок 3.1 - Функциональная схема последовательного порта

Аппаратно каждый синхронный последовательный порт (ПП) (SPORT0 и SPORT1) представляет собой интерфейс с 5-ю выводами (рис.3.1):

SCLK - тактовый сигнал;

RFS - сигнал кадровой синхронизации приема;

TFS - сигнал кадровой синхронизации передачи;

DR - прием данных;

DT - передача данных.

Последовательный порт принимает последовательно передаваемые данные на вывод DR и последовательно передает данные через вывод DT. Биты данных передаются и принимаются синхронно с тактовыми синхроимпульсами генератора SCLK, которые могут формироваться внутренне или приниматься извне. Каждый ПП работает со своим собственным последовательным тактовым сигналом. Внешние тактовые синхроимпульсы могут иметь максимальную частоту, равную тактовой частоте процессора (т.е. до 13,824 МГц), внутренние – половине тактовой частоты. Частота генерируемых внутренних тактовых синхроимпульсов определяется содержимым регистра SCLKDIV.

Кадровая синхронизация приема и передачи настраивается и осуществляется независимо друг от друга. Сигналы кадровой синхронизации RFS и TFS используются для указания на начало каждого слова последовательно передаваемых данных или на начало первого слова массива передаваемых данных. Каждая часть порта (принимающая и передающая) генерирует прерывания по завершению передачи слова данных или после передачи целого массива данных.

Автобуферизация обеспечивает механизм последовательного приема и передачи целого блока данных до генерирования прерывания. Эта функция доступна в обоих портах. При разрешенной автобуферизации каждое слово данных последовательно принимается с порта и передается в память данных или считывается из памяти данных и передается в порт за один непроизводительный цикл. Непроизводительный цикл не влияет на следующую команду и задерживает ее выполнение на один такт. Для таких индивидуальных передач слов данных не генерируется никаких прерываний. Используемые при пересылке данных регистры **In** и **Mn** указываются соответствующими группами разрядов в регистре управления автобуферизацией.

Прерывание приема возникает после приема целого буфера данных. Прерывание передачи возникает после того, как последнее слово было загружено в регистр **TXn**, но до начала его передачи.

Порт SPORT0 поддерживает ***многоканальные операции***. В режиме многоканальных операций последовательно передаваемые данные мультиплексируются с разделением по времени. Каждое последующее слово принадлежит следующему каналу, например, блок длиной в 24 слова содержит по одному слову для каждого из 24 каналов. SPORT0 поддерживает 32 и 24 канала и может автоматически выбирать слова для определенных каналов, игнорируя при этом другие. При многоканальном режиме длина слова по-прежнему устанавливается группой разрядов SLEN в регистре управления последовательного порта и может принимать значения от 3 до 16 бит.

3.2. Этапы программирования последовательных портов

При программировании последовательных портов необходимо осуществить их конфигурацию и активизацию.

Конфигурация последовательного порта формируется при помощи установки соответствующих бит в определенных регистрах. Регистры конфигурации SPORT0 занимают ячейки с 0x3FF3 по 0x3FFA; регистры конфигурации SPORT1 занимают ячейки с 0x3FEF по 0x3FF2.

Содержимое регистров конфигурации последовательного порта в соответствии с их адресом следующее:

0x3FFA Разрешение многоканального приема слова через SPORT0 (31-16 биты)

0x3FF9 Разрешение многоканального приема слова через SPORT0 (15-0 биты)

0x3FF8 Разрешение многоканальной передачи слова через SPORT0 (31-16 биты)

0x3FF7 Разрешение многоканальной передачи слова через SPORT0 (15-0 биты)

0x3FF6 Управляющий регистр SPORT0. Существует 2 варианта работы порта: нормальный и многоканальный, отличающиеся значением разрядов в данном регистре. Многоканальные операции запрещаются/разрешаются установкой бита 15 в регистре управления SPORT0.

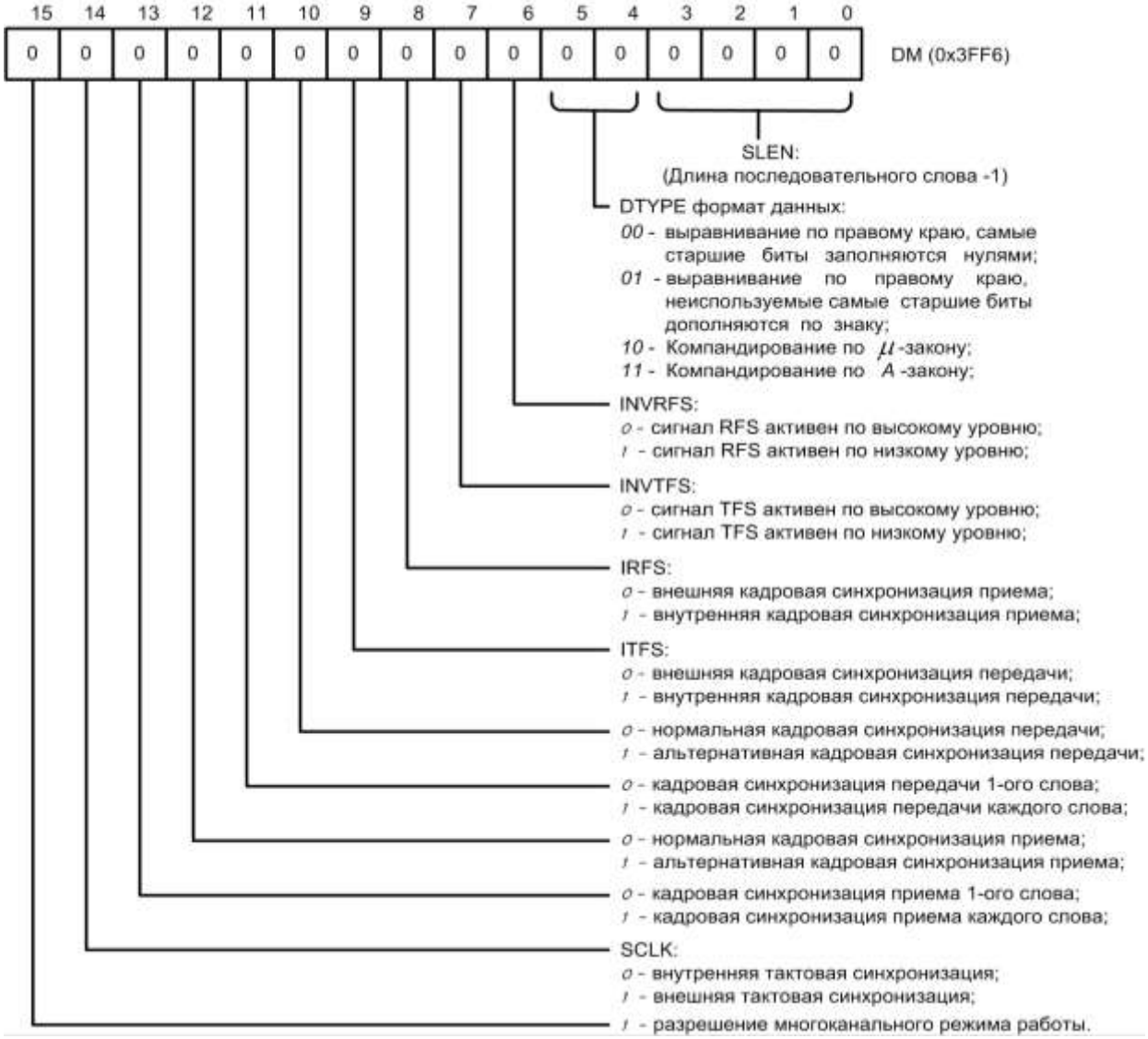
1) Нормальный режим работы. Значения разрядов для этого режима иллюстрирует рисунок 3.2 а.

SLEN – группа разрядов, которые определяют длину последовательно переданного слова:

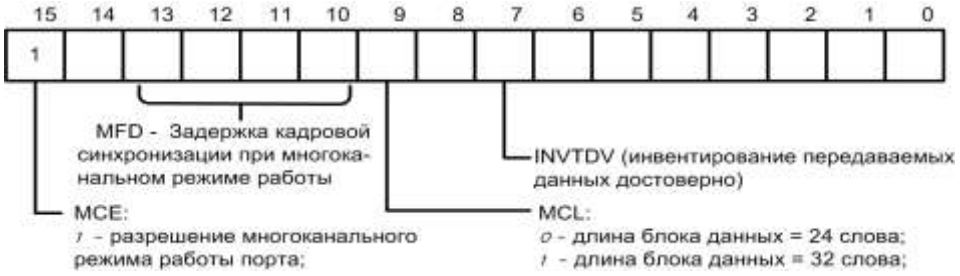
$$Length = SLEN + 1 \quad (1)$$

2) Многоканальный режим работы. Когда 15 бит в регистре управления SPORT0 равен 1, многоканальный режим работы разрешен, а некоторые биты SPORT0 должны быть заданы заново (рисунок 3.2 б). При перезапуске системы, бит 15 сбрасывается, блокируя многоканальные операции и разрешая нормальный режим работы.

Состояние бита длины блока данных (MCL) определяет, используется ли 24 или 32 канала для пересылки блока данных длиной в 24 или 32 слова соответственно. При многоканальном режиме работы длина слова устанавливается группой разрядов SLEN (рисунок 3.2 а). Группа разрядов MFD определяет число циклов тактового генератора между сигналом кадровой синхронизации и первым битом данных.



а)



б)

Рисунок 3.2 - Разряды регистра управления SPORT0:
а) при нормальном режиме работы; б) при многоканальном режиме работы

0x3FF5 16-битовый регистр SCLKDIV, содержащий коэффициент деления тактовых синхроимпульсов SPORT0 (определяет частоту тактовых синхроимпульсов SCLK):

$$Частота\ SCLK = \frac{Частота\ SCLKOUT}{2 \times (SCLKDIV + 1)} \quad (2)$$

(тактовая частота процессора SCLKOUT принимается равной 12,288 МГц).

0x3FF4 Процессор генерирует сигнал кадровой синхронизации приема SPORT0 (RFS) с периодичностью, кратной циклам тактового генератора, на основе значения, содержащегося в данном 16-разрядном регистре делителя кадровой синхронизации приема RFSDIV.

Число циклов тактового генератора между подтверждениями:

$$RFS = RFSDIV + 1. \quad (3)$$

0x3FF3 Регистр управления автобуферизацией SPORT0:

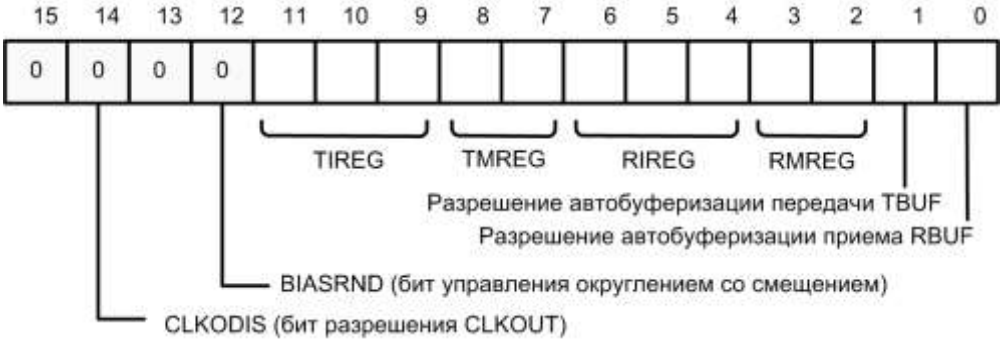


Рисунок 3.3 - Структура регистра управления автобуферизацией SPORT0

Используемые при автобуферизации регистры I и M указываются соответствующими группами бит. TIREG и TMREG – это номера регистров I и M, связанных с буфером передачи. RIREG и RMREG – для приема.

0x3FF2 Регистр управления SPORT 1. Разряды этого регистра аналогичны разрядам регистра управления SPORT0 (рисунок 3.2 а), за исключением 15-ого разряда, который для SPORT 1 означает: вывод флага (только чтение).

0x3FF1 Коэффициент деления тактовых синхроимпульсов SPORT 1 (2)

0x3FF0 Коэффициент деления кадровых синхроимпульсов приема SPORT 1 (3)

0x3FEF Регистр управления автобуферизацией SPORT 1:

Рисунок 3.4 - Структура регистра управления автобуферизацией SPORT1

Инициализацию или изменение значений в регистрах конфигурации последовательного порта можно осуществить записью содержимого регистра по непосредственному адресу. Например:

```
AX0=0x6B27;          DM(0x3FF2) = AX0;
AX0=0;               DM(0x3FF3) = AX0;
```

Активизация последовательных портов производится установкой соответствующих бит в регистре управления системой (рис. 3.5), расположенному по адресу ***0x3FFF***. При перезагрузке системы порты блокируются. Чтобы заблокировать или сделать доступным порт используются 10-12 биты данного регистра.

Рисунок 3.5 - Регистр управления системой

При программировании портов требуется задать соответствующие значения регистров IFC, ICNTL и IMASK. Запись значений в эти регистры – непосредственная.

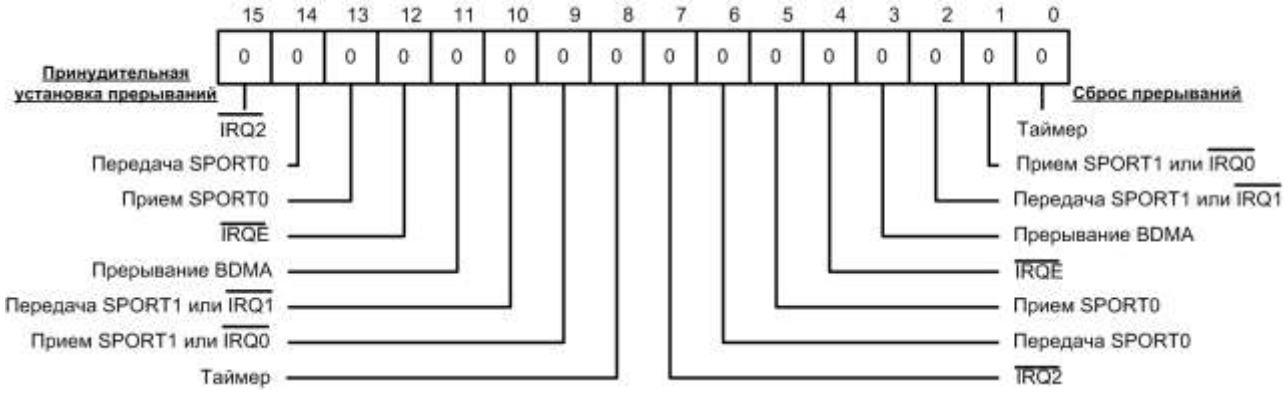


Рисунок 3.6 - Структура регистра IFC

Регистр ***ICNTL*** определяет, каким образом фиксируются внешние прерывания – по фронту или по уровню, а также определяет возможность использования вложенных прерываний.

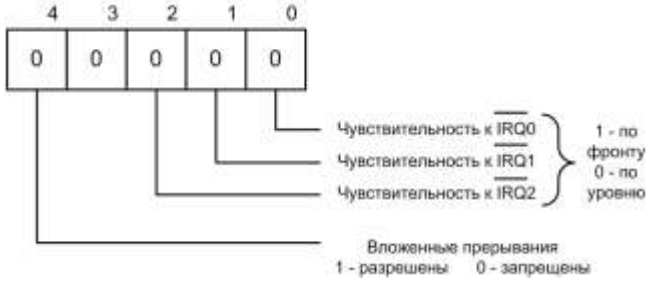


Рисунок 3.7 - Структура регистра ICNTL

Установка в 1 соответствующего бита регистра ***IMASK*** разрешает прерывание, а в 0 – запрещает.

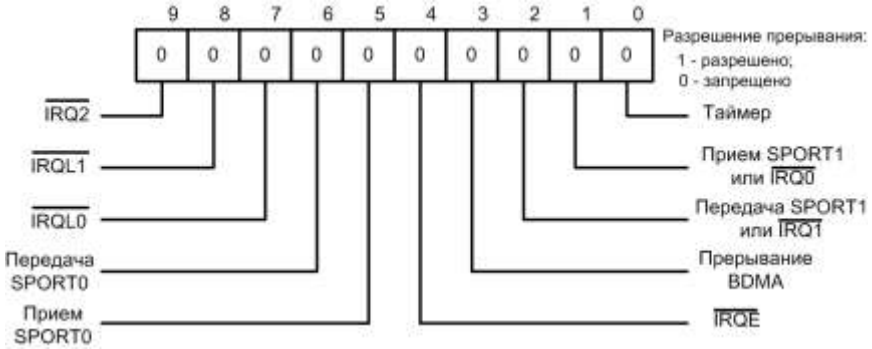


Рисунок 3.8 - Структура регистра IMASK

4. КОДЕК AD 1847

4.1. Функциональная схема AD 1847

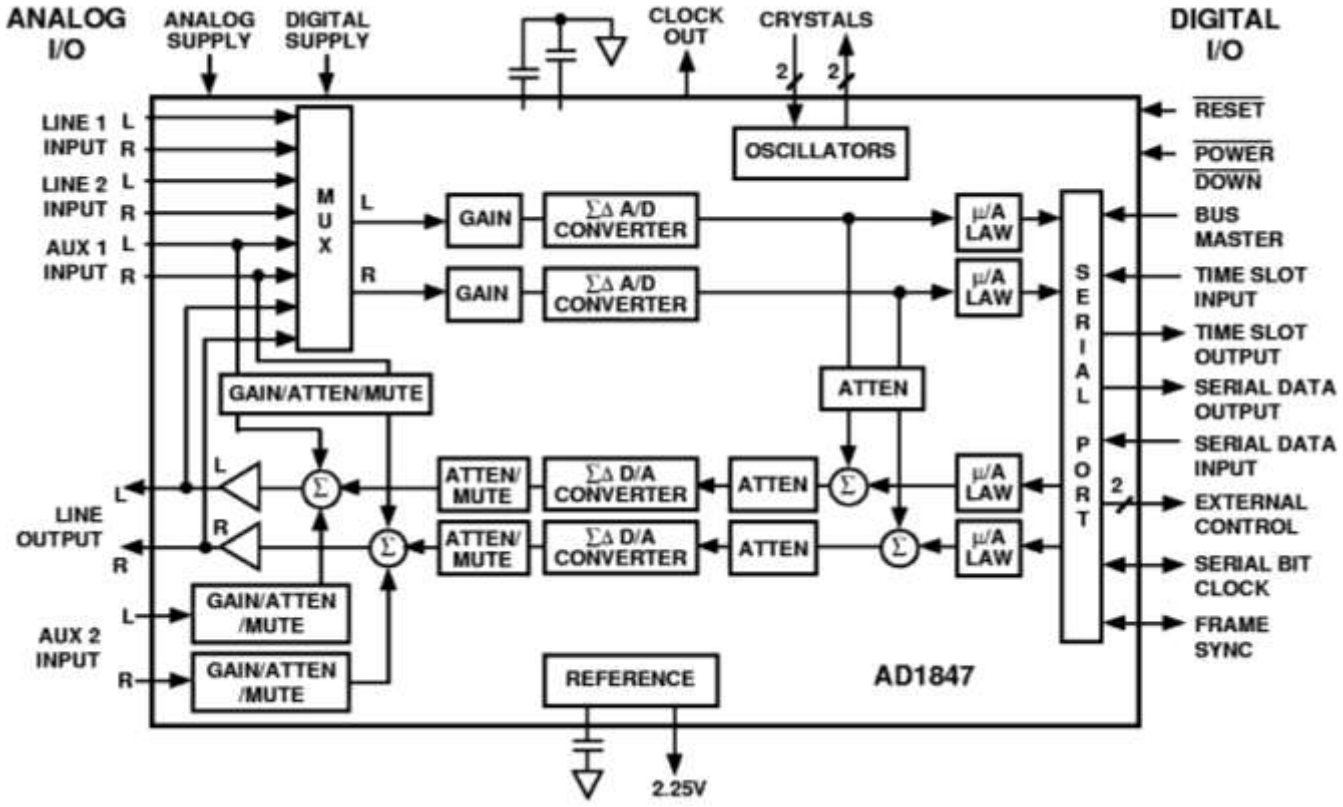


Рисунок 4.1 - Функциональная схема AD 1847

Кодек состоит из аналого-цифрового и цифро-аналогового преобразователей, которые имеют свои выходы и входы. Входами АЦП могут быть 4 стереопары аналоговых сигналов: LINE1, LINE2 и 2 дополнительных входов AUX1 и AUX2. Каждый канал дополнительных входов может быть независимо усилен/ослаблен от +12дБ до -34,5 дБ с шагом 1,5 дБ. Для входов LINE1 и LINE2 эти параметры выбираются из диапазона от 0дБ до +22,5Дб с шагом +1,5 дБ. Все эти четыре входа имеют левый L и правый R каналы. Стерео-выход – LINE OUTPUT. При определении режима работы кодека определяется возможность использования моно или стерео режима.

В каждом из каналов можно задавать свой коэффициент усиления/ ослабления. Последовательный интерфейс совместим с интерфейсом процессоров семейства ADSP-21xx.

AD 1847 работает на двух внутренних кристаллах: XTAL1 и XTAL2. Они обеспечивают широкий диапазон значений тактовых синхроимпульсов. Рекомендуемые частоты кристаллов 16,9344МГц и 24,576МГц. В зависимости от них частота тактовых синхроимпульсов может задаваться в диапазоне 5,5125кГц … 48кГц. Если выбран XTAL1, то выход SCLK генерируется с частотой 12,288МГц.

Последовательный интерфейс совместим с ADSP 21xx. AD 1847 передает импульсы SCLK по последовательной шине DSP.

4.2. Управляющие регистры кодека AD 1847

При установлении связи между DSP и кодеком, необходимо провести инициализацию AD1847. Для полного определения режимов работы кодека, необходимо задать значения управляющих 8-разрядных регистров AD1847. В таблице для каждого адреса регистра (c0, c1, …cf) приведены: названия восьми соответствующих бит, под ними - приблизительные значения в двоичном коде, справа- значения в шестнадцатеричной системе счисления.

Адрес	*Data 7*	*Data 6*	*Data 5*	*Data 4*	*Data 3*	*Data 2*	*Data 1*	*Data 0*	*Значен*
c0	LSS1	LSS0	res	res	LIG3	LIG2	LIG1	LIG0	
	0	***0***	***0***	***0***	***0***	***0***	***1***	***1***	***03***
c1	RSS1	RSS0	res	res	RIG3	RIG2	RIG1	RIG0	
	0	***0***	***0***	***0***	***0***	***0***	***1***	***1***	***03***
c2	LMX1	res	res	LX1G4	LX1G3	LX1G2	LX1G1	LX1G0	
	1	***0***	***0***	***0***	***1***	***0***	***0***	***0***	***88***
c3	RMX1	res	res	RX1G4	RX1G3	RX1G2	RX1G1	RX1G0	
	1	***0***	***0***	***0***	***1***	***0***	***0***	***0***	***88***
c4	LMX2	res	res	LX2G4	LX2G3	LX2G2	LX2G1	LX2G0	
	1	***0***	***0***	***0***	***1***	***0***	***0***	***0***	***88***
c5	RMX2	res	res	RX2G4	RX2G3	RX2G2	RX2G1	RX2G0	
	1	***0***	***0***	***0***	***1***	***0***	***0***	***0***	***88***
c6	LDM	res	LDA5	LDA4	LDA3	LDA2	LDA1	LDA0	
	1	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***80***
c7	RDM	res	RDA5	RDA4	RDA3	RDA2	RDA1	RDA0	
	1	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***80***
c8	res	FMT	C/L	S/M	CFS2	CFS1	CFS0	CSL	
	0	***1***	***0***	***0***	***0***	***1***	***1***	***0***	***46***
c9	res	res	res	res	ACAL	res	res	PEN	
	0	***0***	***0***	***0***	***1***	***0***	***0***	***1***	***09***
ca	XCTL1	XCTL0	CLKTS	res	res	res	res	res	
	0	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***00***
cb	inval	inval	inval	inval	inval	inval	inval	inval	—
cc	FRS	TSSEL	res	res	res	res	res	res	
	1	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***40***
cd	DMA5	DMA4	DMA3	DMA2	DMA1	DMA0	res	DME	
	0	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***00***
ce	inval	inval	inval	inval	inval	inval	inval	inval	—
cf	inval	inval	inval	inval	inval	inval	inval	inval	—

res- резервные биты, равные 0; ***inval*** – запись по этому адресу игнорируется.

c0 (c1): регистр левого (правого) входа AD1847

LSS1:0 (RSS1:0) – выбор левого (правого) входа (LINE1, LINE2..)

LIG3:0 (RIG3:0) – определение 16-уровневого коэффициента усиления (КУ) для выбранного входа (от 0 до 22,5дБ с шагом 1.5дБ).

Значение в таблице (0x03) означает, что выбран левыйL (правыйR) вход LINE1 с КУ = 4,5дБ.

c2 (c3): регистр управления левого (правого) дополнительного входа AUX1

LMX1 (RMX1) – если этот бит =1, то левый (правый) канал входа AUX1 подавляется.

LX1G4:0 (RX1G4:0) – определение 32-уровн. коэффициента усиления/ослабления для левого (правого) дополнительного входа AUX1 (от +12дБ до -34,5дБ с шагом -1.5дБ).

Значение в таблице (0x88) соответствует КУ = 0дБ с подавлением канала входа AUX1.

c4 (c5): регистр управления левого (правого) дополнительного входа AUX2

Тоже самое, но для дополнительного входа AUX2.

c6 (c7): регистр управления левого (правого) ЦАП AD1847

LDM (RDM) – если этот бит =1, то левый (правый) ЦАП подавляется.

LDA5:0 (RDA5:0) – определение 64-уровн. коэффициента ослабления (КО) для левого (правого) ЦАП (от 0дБ до -94,5дБ с шагом -1.5дБ).

Значение в таблице (0x80) соответствует КО = 0дБ с подавлением ЦАП.

c8: регистр формата данных AD1847

CSL - выбор источника тактовых синхроимпульсов. 0- XTAL1, 1- XTAL2.

CFS2:0 – выбор коэффициента деления тактовой частоты, который определяет частоту дискретизации аудиосигналов.

S/M- выбор стерео (=1) или моно (=0) режимов работы. В моно режиме сигналы по обоим каналам идентичны.

C/L – выбор между линейным цифровым представлением аудиосигнала (=0) и нелинейным, компандированным форматом для всех входных и выходных данных (=1).

FMT – бит, определяющий формат всех цифровых аудио входов и выходов (8- , 16 -битные).

Значение в таблице (0x46) означает, что выбран кристалл XTAL1, коэффициент деления=768 (частота дискретизации=32кГц), 16-разрядные данные передаются без компандирования.

c9: регистр конфигурации интерфейса AD1847

PEN – разрешение воспроизведения данных в выбранном формате, если = 1.

ACAL – разрешение (=1)/ запрет (=0) авокалибровки.

Значение в таблице (0x09) соответствует разрешенным режимам воспроизведения и автокалибровки.

ca: регистр управления выводами AD1847

CLKTS – определяет будет ли сигнал CLKOUT сигналом с тремя состояниями.

XCTL1:0 – биты внешнего управления, которые определяют вид ТТЛ логики для выводов XCTL1 и XCTL0 – если=0 - то низкая, если=1 - то высокая.

Значение в таблице (0x00) означает, что CLKOUT – сигнал не с 3 состояниями, а выводы XCTL1 и XCTL0 – с низкой логикой.

cc: регистр разнообразной информации AD1847

FRS – длина кадра. Этот бит определяет количество временных слотов в кадре (FRS=0 – 32 слота, FRS=1 – 16 слотов)

TSSEL – выбор слота для передачи: TSSEL=0 – передаются 3, 4 и 5 временные слоты (1-проводная система), TSSEL=1 – передаются 0, 1 и 2 временные слоты (2-проводная система).

Значение в таблице (0x40) соответствует 1-проводной системе с 16 слотами в кадре.

cd: регистр управления цифрового миксера AD1847

DME – разрешение (когда DME=1) цифрового смешивания выхода АЦП с входом ЦАП.

DMA5:0 – 64-уровневое ослабление цифрового смешивания (до – 94,5 дБ с шагом –1,5дБ).

Значение в таблице (0x00) соответствует запрету микшера.

4.3. Алгоритм взаимодействия ADSP 2181 и AD1847

Изучению алгоритма взаимодействия ADSP 2181 и AD1847 посвящена лабораторная работа №4, в которой звуковые данные поступают на вход платы, а со стерео выхода снимается сигнал, состоящий из двух составляющих (левой и правой). По одному каналу идет исходный сигнал, а по другому- отфильтрованный с помощью одного из двух фильтров. Пример программы, иллюстрирующей данный

алгоритм, приведен ниже.

В начале программы задаются секции описания переменных в памяти данных (data 1), памяти программ (data 2) и таблица векторов прерываний. ***В памяти данных*** объявляется буфер для записи принимаемых значений входного сигнала, полученных от кодека buf1, объявляются флаги и приемный и передающий буферы AD1847. ***В памяти программ*** объявляются буферы с коэффициентами фильтров. Если предполагается обработка какого-либо прерывания, то это обязательно должно быть отражено в ***векторе прерываний***.

Алгоритм взаимодействия между процессором ADSP 2181 и кодеком AD1847 состоит из двух основных этапов: инициализации и непосредственной обработки данных по заданному алгоритму. *ИНИЦИАЛИЗАЦИЯ* в свою очередь включает в себя:

1) Инициализация ADSP 2181, которая включает в себя конфигурацию последовательных портов SPORT0 и SPORT1,установку таймера, системы и памяти.

2) Инициализация AD1847, которая соответствует значению “1” флага stat_flag. Разрешается прерывание по передаче данных из порта SPORT0 ADSP 2181 для пересылки управляющих слов кодеку. Первым отсылается управляющее слово из регистра tx_buf (0xc000), которое сигнализирует кодеку о начале инициализации. При этом в передающий регистр записывается первое значение и вызывается *обработка прерывания по передаче* для инициализации кодека, описанная в самом векторе прерываний. Если stat_flag=1, то осуществляется переход на метку *next_cmd*, в противном случае - rti.

next_cmd: В первый элемент передающего буфера записываем управляющее слово из буфера init_cmds. При этом проверяется условие возврата в начало буфера, т.е. когда все 13 элементов будут переданы. В этом случае флаг состояния stat_flag обнуляется и происходит возврат к метке check_init, где ax0 записывает значение stat_flag и, если он =0, то движемся вниз по программе. Когда же stat_flag =1 (т.е. когда происходит настройка кодека буфером init_cmds), то происходит возврат на метку check_init и ожидается обнуление флага.

Затем выполняется автокалибровка для правого и левого каналов, включение левого и правого ЦАП, очистка требования автокалибровки путем обмена соответствующими управляющими словами.

Процедура инициализации завершается передачей управляющего слова очистки на кодек, очисткой очереди прерываний (ifc) и определением разрешенных прерываний с помощью регистра imask.

После процедуры инициализации происходит ожидание прерывания по приему последовательным портом процессора SPORT0 данных для обработки, поступающих через кодек (метка *input_samples*). В этой части осуществляется непосредственная обработка данных по заданному алгоритму.

Подпрограмма обслуживания прерывания по таймеру (*switch_fir*) осуществляет выбор типа фильтра, который задается флагом fir_flag. Когда fir_flag = 0 - осуществляется ВЧ-фильтрация с использованием буфера коэффициентов coef2, в противном случае - НЧ-фильтрация с использованием буфера коэффициентов coef. Переключение фильтров осуществляется с частотой, задаваемой в соответствующих регистрах таймера.

ТЕКСТ ПРОГРАММЫ ДЛЯ ЛАБ. РАБ №4

/**** ОПИСАНИЕ ПЕРЕМЕННЫХ В ПАМЯТИ ДАННЫХ ******/**

```
.section/dm data1;
.var/circ  buf1[57];                          //Буфер, для записи сигналов,примнимаемых от кодека
.var/circ  rx_buf[3];                         //приемный буфер для  AD1847
.var/circ  tx_buf[3] = 0xc000, 0x0000, 0x0000;  //передающий буфер для AD1847
.var/circ  init_cmds[13]=                     //регистры управления AD1847:
        0xc003,                               //регистр управления левого входа
        0xc103,                               //регистр управления правого входа
        0xc288,                               //регистр управления левого внешнего входа AUX1
        0xc388,                               //регистр управления правого внешнего входа AUX1
        0xc488,                               //регистр управления левого внешнего входа AUX2
        0xc588,                               //регистр управления правого внешнего входа AUX2
        0xc680,                               //регистр управления левого ЦАП
        0xc780,                               //регистр управления правого ЦАП
        0xc856,                               //регистр формата данных AD1847
        0xc909,                               //регистр конфигурации интерфейса
        0xca00,                               //регистр управления выводом AD1847
        0xcc40,                               //регистр различной информации AD1847
        0xcd00;
.var stat_flag;                               //флаг состояния (используется для разграничения режимов
                                              // начальной инициализации и непосредственно выполнения
                                              // программы)
.var fir_flag;                                //флаг, используемый при работе таймера (или IRQE)
```

/**** ОПИСАНИЕ ПЕРЕМЕННЫХ В ПАМЯТИ ПРОГРАММ ******/**

```
.section/pm data2;
.var/circ coef[]="fir3.dat";                  //НЧ-фильтр
.var/circ coef2[]="fir2.dat";                 //ВЧ-фильтр
```

/**** ОПИСАНИЕ ПРЕРЫВАНИЙ ******/**

```
        .section/pm IVreset;                  //Запуск программы после Reset
 jump start; rti; rti; rti;
.section/pm IVirq2;                           //IRQ2
                rti; rti; rti; rti;
.section/pm IVirql1;                          //IRQL1(по уровню)
                rti; rti; rti; rti;
.section/pm IVirql0;                          //IRQL0(по уровню)
                rti; rti; rti; rti;
.section/pm IVsport0xmit;                     //Передача SPORT0
                ar=dm(stat_flag);
                ar=pass ar;
                if eq rti;
                jump next_cmd;
.section/pm IVsport0recv;                     //Прием SPORT0
 jump input_samples; rti; rti; rti;
.section/pm IVirqe;                           //IRQE  (по нажатию кнопки INTERRUPT)
        jump switch_fir; rti; rti; rti;       //если прерывание IRQE не нужно, то в этой строке пушут:
                                              // rti; rti; rti; rti;
.section/pm IVbdma;                           //Прерывания BDMA
                rti; rti; rti; rti;
.section/pm IVirq1;                           //Передача SPORT1 / IRQ1
                rti; rti; rti; rti;
.section/pm IVirq0;                           //Прием SPORT1 / IRQ0
                rti; rti; rti; rti;
.section/pm IVtimer;                          //Таймер (если не  требуется прерывание по таймеру)
                rti; rti; rti; rti;
//.section/pm IVtimer;                        //Таймер (если требуется прерывание по таймеру)
//        jump switch_fir; rti; rti; rti;
.section/pm IVpwrdwn;                         //Понижение потребляемой мощности
                rti; rti; rti; rti;
```

/**** ПРОГРАММА ******/**

```
.section/pm program;
start:
        i0 = rx_buf;                    //Принимающий буфер AD1847
        l0 = length(rx_buf);
        i1 = tx_buf;                    //Передающий буфер AD1847
        l1 = length(tx_buf);
        i3 = init_cmds;                 //Управляющие слова для AD1847
        l3 = length(init_cmds);
        m1 = 1;
i2 = buf1; l2 = length(buf1); m2=1;     //Буфер принимаемых сигналов
i7 = coef; l7 = length(coef); m7=1;     //Буфер коэффициентов НЧ-фильтра
```

/*---------- ИНИЦИАЛИЗАЦИЯ ADSP 2181 ----------*/

// КОНФИГУРАЦИЯ ПОСЛЕДОВАТЕЛЬНОГО ПОРТА SPORT0:

```
        ax0=****;
```

*/*разрешить автобуферизацию приема и передачи; указать регистры для косвенной адресации для tx_buf и rx_buf, используя один регистр модификации M1 для tx_buf и rx_buf, т.е. TMREG=RMREG=01, а значения индексных регистров I определить исходя из текста программы*/*

```
dm(0x3ff3)=ax0;                 //Регистр автобуферизации SPORT0
        ax0=0;
dm(0x3ff4)=ax0;                 //Генерирование сигнала кадровой синхронизации RFS
                                //(RFSDIV не используется)
dm(0x3ff5)=ax0;                 //Определение частоты тактовой синхронизации SCLK
                                //(SCLKDIV не используется)
        ax0=****;
```

/ разрешить многоканальный режим работы; внутренняя тактовая синхронизация; задержка кадровой синхронизации на 1, т.е. MDF=0001; длина блока данных=32слова; внешняя кадровая синхронизация приема; сигналы RFS и TFS активны по высокому уровню; выравнивание данных по правому краю; длина последовательного слова = 16 */*

```
dm(0x3ff6)=ax0;                 // Управляющий регистр SPORT0
        ax0=0x0007;             // 0000 0000 0000 0111
dm(0x3ff7)=ax0;                 //Разрешение многоканальной передачи   биты 15-0
dm(0x3ff8)=ax0;                 //слова через SPORT0                   биты 31-16
dm(0x3ff9)=ax0;                 //Разрешение многоканального приема    биты 15-0
dm(0x3ffa)=ax0;                 //слова через SPORT0                   биты 31-16
```

// КОНФИГУРАЦИЯ ПОСЛЕДОВАТЕЛЬНОГО ПОРТА SPORT1:

```
        ax0=0;
dm(0x3fef)=ax0;                 //Регистр автобуферизации SPORT1
dm(0x3ff0)=ax0;                 //Генерирование сигнала кадровой синхронизации RFS
dm(0x3ff1)=ax0;                 //Определение частоты тактовой синхронизации SCLK
dm(0x3ff2)=ax0;                 //Управляющий регистр SPORT1
```

// УСТАНОВКА ТАМЕРА:

```
        ax0=0;                  //Таймер не будет использоваться
dm(0x3ffd)=ax0;                 //TPERIOD
dm(0x3ffc)=ax0;                 //TCOUNT
dm(0x3ffb)=ax0;                 //TSCALE
```

// УСТАНОВКА СИСТЕМЫ И ПАМЯТИ:

```
        ax0=0;
dm(0x3ffe)=ax0;                 //Регистр DM_wait
        ax0=0x1000;             //Доступен SPORT0
dm(0x3fff)=ax0;                 //Регистр управления системой, с помощью которого
                                //осуществляется АКТИВИЗАЦИЯ последовательных портов

        ifc=0x00ff;             //Очистка очереди прерываний - 0000 0000 1111 1111
nop;
        icntl=0;                //Вложенные прерывания запрещены
        mstat=B#1000000;
```

```
                    /*------------------------ ИНИЦИАЛИЗАЦИЯ AD 1847 ------------------------*/
        ax0=1;
dm(stat_flag)=ax0;                          //Очистка флага
imask=**;           /*разрешается только  прерывания на передачу SPORT0  и IRQE  ИЛИ  только
                    прерывания на передачу SPORT0  и Таймера*/
        ax0=dm(i1,m1);                      //Запись управляющего слова из буфера tx_buf
                                            //(первого элемента из буфера tx_buf)
tx0=ax0;                                    //Запись в регистр передачи данных TX0, при этом
                                            //вызывается прерывание по передаче
                                            //Если флаг=0, то-вылет; нет, то-переход на next_cmd
check_init:
ax0 = dm(stat_flag);                        //Ожидание до целой выборки
af=pass ax0;                                //Буфер переслать к кодеку вместе с next_cmd
if ne jump check_init;

        ay0=2;
check_acih:                                 //Автокалибровка для правого канала
ax0=dm(rx_buf);                             //Ожидание инициализации пока кодек выйдет
ar=ax0 and ay0;                             //из режима автокалибровки
if eq jump check_acih;

check_acil:                                 //Автокалибровка для левого канала
ax0=dm(rx_buf);                             //Ожидание инициализации пока кодек выйдет
ar=ax0  and ay0;                            //из режима автокалибровки
if ne jump check_acil;
idle;
        ay0 = 0xbf3f;
ax0=dm(init_cmds + 6);                      //Включение левого ЦАП
ar  = ax0 and ay0;
dm(tx_buf) = ar;
idle;
ax0=dm(init_cmds + 7);                      //Включение правого ЦАП
ar  = ax0 and ay0;
dm(tx_buf) = ar;
idle;
        ax0=0xc901;                         //Очистка требования автокалибровки
dm(tx_buf)=ax0;
idle;
        ax1=0x8000;                         //Управляющее слово очистки
dm(tx_buf)=ax1;

ifc=0x00ff;                                 //Очистка очереди прерываний
nop;
/*Когда требуется прерывание от таймера, то должно быть написано так (вышеприведенная установка таймера должна
быть закомментирована):

//УСТАНОВКА ТАМЕРА:
        ax0=0xffff;
dm(0x3ffd)=ax0;                             //TPERIOD
        ax0=0xffff;
dm(0x3ffc)=ax0;                             //TCOUNT
        ax0=0xff;
dm(0x3ffb)=ax0;                             //TSCALE
        ax0=0;
dm(fir_flag)=ax1;
ena timer;
imask=**;           //разрешаются прерывания по приему SPORT0 и от таймера (записать в
                    // шестнадцатеричной и двоичной системах )
*/
imask=**;           //разрешается прерывание по приему SPORT0 и от IRQE (записать в
                    // шестнадцатеричной и двоичной системах )

talkthru: idle;                             /* ОЖИДАНИЕ ПРЕРЫВАНИЯ */
jump talkthru;
```

```
/*---------- ПОДПРОГРАММЫ ОБСЛУЖИВАНИЯ ПРЕРЫВАНИЙ ----------*/

input_samples:          //ПРЕРЫВАНИЕ ПО ПРИЕМУ SPORT0 - ОБРАБОТКА ДАННЫХ
                        //СТЕРЕО РЕЖИМ С ФИЛЬТРАЦИЕЙ
ena sec_reg;            //Использование теневого банка регистров
dis m_mode;
mx1 = dm(rx_buf+1);     //Получение данных от кодека
ax1=mx1;
dm(i2,m2)=mx1;          //Запись принятых данных в принимающий буфер buf1

                        /*                                      */
                        /*                                      */
                        /*   ПРОЦЕДУРА ОБРАБОТКИ ДАННЫХ         */
                        /*   (ФИЛЬТРАЦИЯ)                       */
                        /*                                      */
                        /*                                      */

        dm(tx_buf + 1) = mr1;       //Отфильтрованный сигнал (справа)
        dm(tx_buf + 2) = ax1;       //Исходный сигнал (слева)
        rti;

next_cmd:               //ПРЕРЫВАНИЕ ПО ПЕРЕДАЧЕ ДЛЯ ИНИЦИАЛИЗАЦИИ КОДЕКА
ena sec_reg;
ax0=dm(i3, m1);         //Получение следующего управляющего слова и  размещение в передающий слот
dm(tx_buf)=ax0;
ax0=i3;
ay0 =init_cmds;
ar=ax0 - ay0;
if gt rti;              //Выход в chec_init
ax0=0x8000;
dm(tx_buf)=ax0;         //Запрет модификации режима кодека
ax0=0x0;
dm(stat_flag)=ax0;      //Переустановить флаг состояния
rti;

switch_fir:             //ПРЕРЫВАНИЕ ОТ ТАЙМЕРА (или IRQE )
ar=dm(fir_flag);
ar=pass ar;
if eq jump chan;
i7=coef;                //НЧ-фильтр
ax1=0;
dm(fir_flag)=ax1;       //флаг fir_flag=0
rti;
chan:
i7=coef2;               //ВЧ-фильтр
ax1=1;
dm(fir_flag)=ax1;       //флаг fir_flag=1
rti;

trs;
```

ЛАБОРАТОРНАЯ РАБОТА №1.

ГЕНЕРИРОВАНИЕ СИГНАЛОВ В СРЕДЕ VISUALDSP++

Цель работы: знакомство с правилами и методикой программирования на ЦПОС, приобретение начальных навыков работы в среде VisualDSP++: получение навыков записи, компиляции и отладки программы на языке ассемблер, отслеживание содержимого регистров в процессе выполнения программы, графический просмотр результатов.

Описание лабораторной установки

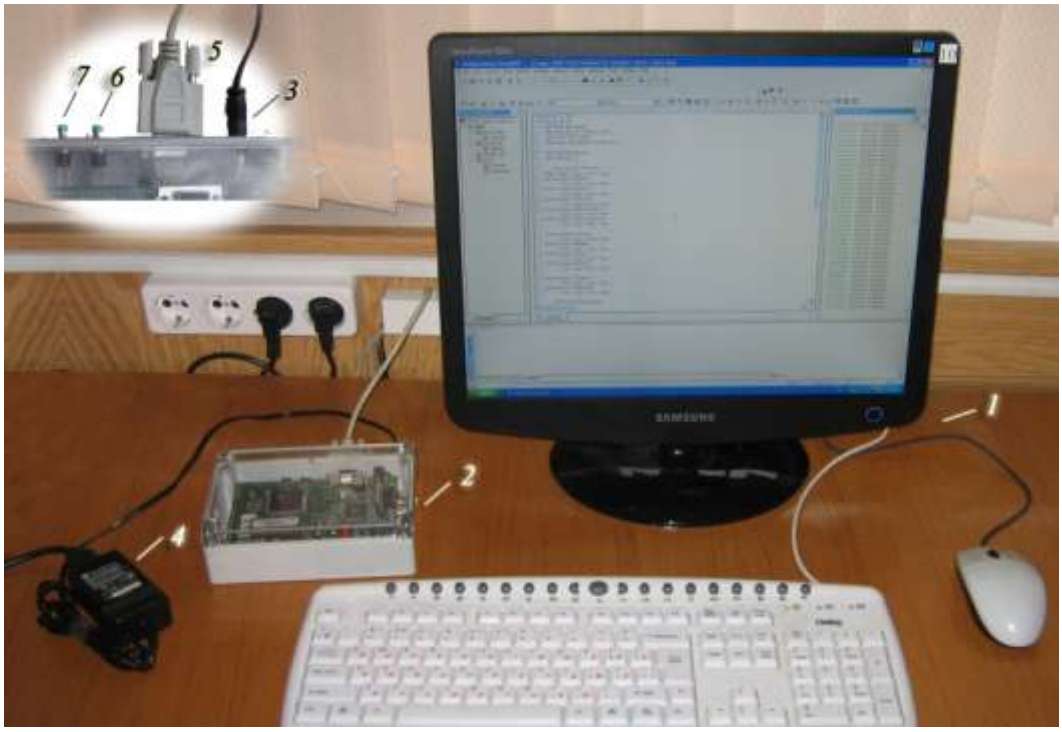


Рисунок 5.1 - Лабораторная установка

Лабораторная установка для первых трех работ состоит из (рис 5.1):

- персональный компьютер *(1)*, с установленной средой Visual DSP++;
- непосредственно плату IZ-KIT Lite *(2)*, на цилиндрический штекер *J4* *(3)* которой подается питание через специальный адаптер *(4)*, соединенную с ПК *(1)* через разъем *(5)* штепсельного типа *J3*;

Для работы так же используются кнопки Reset *(7)* и Interrupt *(6)*.

Домашнее задание

Перед выполнением лабораторной работы необходимо выполнить домашнее задание.

ТЕОРИЯ:

1. Внимательно ознакомиться с содержанием, порядком выполнения работы и требованиями, предъявляемыми к отчету.

2. Тщательно изучить основы работы со средой VisualDSP++ и особенности разработки программного кода, которые изложены выше.

3. Ознакомиться с расположением основных элементов на плате EZ-KIT Lite, их предназначением.

4. Разобраться в примере программы, приведенном ниже. Обратить внимание на структуру программы, способы задания переменных и констант для различных областей памяти, описание вектора прерываний, организацию цикла, способы адресации и т.д.

ПРАКТИКА:

5. В соответствии с вариантом задания написать текст программы на языке ассемблер. Для этого необходимо знать, что:

1) уже существует готовый файл значений *data.dat*, состоящий из N=64 значений (записанных в шестнадцатеричном коде) функции Sin(x), задающий (таблично) данную функцию на периоде 2π;

2) ЦАП выдает колебания в диапазоне частот от 20Гц до 20кГц. Один период сигнала (Sin (x) или Cos(x)), заданного буфером из 64 значений генерируется с частотой Fo=4кГц;

3) для того, чтобы сгенерировать колебание заданной частоты F_3 необходимо взять из файла data.dat каждое j-ое значение, где $j = F_з / F_o$, и записать эти значения в отдельный циклический буфер. При этом уменьшается частота дискретизации, но возрастает частота сигнала;

4) функцию y= Cos(x) можно получить сдвигом всех значений функции y=Sin(x) влево, т.е. величины из файла data.dat необходимо брать, начиная не с первого значения, а со значения функции y=Sin (x), соответствующего углу 90˚, т.е. с шестнадцатого значения.

Структура программы, написанной на Ассемблере

```
                // Объявление констант:
#define  N 128                          /*задание константы N, которая равна 128*/
#define  buf_size 64                    /*задание константы buf_size, которая равна 64*/

                // Описание  переменных, буферов  в ПД:
.SECTION/DM    data1;                   /*Секция описания переменных, буферов в памяти данных DM */
.VAR/CIRC  Buf_A[buf_size]="data.dat";  /*Задание циклического буфера размером buf_size в памяти
                                          данных DM  и его заполнение значениями из файла data.dat */

                // Описание  переменных, буферов  в ПП:
.SECTION/PM    data2;                   /*Секция описания переменных, буферов в памяти программ PM */
.VAR Buf_S[N];                          /*Задание буфера Buf_S на N элементов в памяти программ PM*/
.VAR Buf_C[N];                          /*Задание буфера Buf_C на N  элементов в памяти программ PM*/

                //  Задание вектора прерываний:
.SECTION/PM        IVreset;             /* Запуск программы после reset */
    JUMP start; nop; nop; nop;
.SECTION/PM        IVirq2;              /* IRQ2 */
    rti; nop; nop; nop;
.SECTION/PM        IVirql1;             /* IRQL1(по уровню) */
    rti; nop; nop; nop;
.SECTION/PM        IVirql0;             /* IRQL0(по уровню) */
    rti; nop; nop; nop;
.SECTION/PM        IVsport0xmit;        /* Передача SPORT0 */
    rti; nop; nop; nop;
```

```
.SECTION/PM      IVsport0recv;     /* Прием SPORT0 */
    rti; nop; nop; nop;
.SECTION/PM      IVirqe;           /* IRQE  (по фронту) */
    rti; nop; nop; nop;
.SECTION/PM      IVbdma;           /* Прерывания прямого побайтового доступа к памяти BDMA */
    rti; nop; nop; nop;
.SECTION/PM      IVirq1;           /* Передача SPORT1 / IRQ1 */
    rti; nop; nop; nop;
.SECTION/PM      IVirq0;           /* Прием SPORT1 / IRQ0 */
    rti; nop; nop; nop;
.SECTION/PM      IVtimer;          /* Таймер */
    rti; nop; nop; nop;
.SECTION/PM      IVpwrdwn;         /* Понижение потребляемой мощности */
    rti; nop; nop; nop;

            // Программа:
.SECTION/PM    program;
start:                            /* Метка начала программы */
    I0=Buf_A;                     /* I0 содержит адрес первого элемента буфера Buf_A */
    M0=4;                         /* M0 указывает на сколько будет изменяться указатель I0. Т.к.  M0=4,
                                     то из буфера Buf_A будет выбираться каждый 4-тый  элемент */
    L0=buf_size;                  /* Задание длины циклического буфера. Если Ln=0, то буфер не
                                       является циклическим */
    I4=Buf_S;                     /* I4 содержит адрес первого элемента буфера Buf_S */
    M4=1;                         /*I4 будет изменяться на 1, т.е. обращение будет к каждому элементу*/
    L4=0;                         /*  Buf_S - не циклический. */
    I5=Buf_C;     M5=1;     L5=0; /* I5 содержит адрес первого элемента буфера Buf_C */
    I1=Buf_A;                     /* I1 содержит адрес первого элемента буфера Buf_A */
    L1= buf_size;
    M1=16;                        /*смещение буфера значений на 16 точек влево (на 90 градусов)*/
    AR=DM(I1,M1);
    M1=1;                         /*установка регистра модификации для возможности обращения к
                                     каждому элементу*/
    CNTR=N;                       /* Установка значения счетчика */
label1:                           /*Метка цикла*/
    AX0=DM(I0,M0);                /* В AX0 загружается значение из буфера, находящееся по адресу,
                                     указанному в I0*/
    PM(I4,M4)=AX0;                /* Содержание регистра AX0 записывается по адресу, указанному в
                                     регистре I4 (в первый элемент буфера Buf_B). Затем указатель I4
                                     модифицируется на величину, содержащуюся  в M4 . Т.е. если I4
                                     измениться на 1, то при следующих  выполнениях цикла он будет
                                     указывать на 2-ый элемент буфера Buf_A, на 3-ий эл-т  и т.д.*/
    MX0=DM(I1,M1);
    PM(I5,M5)=MX0;

    if not CE Jump label1;        /* Если счетчик не пуст, т.е. неравен 0, то возвращение на метку lable1.
                                     При каждом выполнении цикла, значение счетчика уменьшается на 1*/
    nop;                          /* Конец программы */
```

Порядок выполнения работы

1. Создать папку *ADSP_lab1* для будущего проекта в директории, указанной преподавателем.

2. Запустить VisualDSP++ с помощью ярлыка на рабочем столе.

3. Просмотреть предназначение всех кнопок, расположенных на панели инструментов среды VisualDSP++. Знание этих кнопок поможет быстрее и удобнее работать с данной средой в процессе отладки программы.

4. Создать новый проект в папке *ADSP_lab1* под названием: *Lab1*. *dpj*.

5. Создать файл *Lab1*. *dsp* в той же папке и добавить его в проект (в папку *Source files*). В нем написать заранее подготовленный текст программы, соответствующий

варианту задания.

6. Добавить *в окно проекта* Project папку DATA, а также файл *data.dat*, находящийся в указанной преподавателем директории. Этот файл поместить с помощью простого перетаскивания в окне Project в папку DATA.

7. Сгенерировать файл линкера с расширением **.ldf* , необходимый для отладки проекта. Для этого в разделе меню *Tools* выбрать строку *Expert Linker,* а в ней команду *New*. В открывающихся окнах нажать OK. В рабочей области откроется созданный файл. Закройте его и проверьте, что он добавлен в папку Linker Files в окне Project.

8. Выполнить компиляцию файла (*Build File* или *Ctrl+F7*). Если Ассемблер выявил ошибки, исправить их и повторить компиляцию. После успешного выполнения компиляции файла, осуществить компоновку проекта (*Build Project* или *F7*).

9. Запустить программу на исполнение в пошаговом режиме *F11* или, используя команду *Run to cursor*. Для быстрого выполнения всей программы можно воспользоваться командой *Run (F5)*, предварительно поставив точку останова *breakpoint* на последней строке программы. Для этого поставить курсор на последней строке программы, нажать правую кнопку мыши и выбрать команду *Insert breakpoint*. Слева от команды должна появиться красная точка. Желтая стрелка указывает на команду, которая будет исполняться на следующем шаге. Во время выполнения программы:

1) проследить изменение содержимого всех регистров, используемых в программе;

2) просмотреть изменение регистров генератора адреса данных I_i (для тех, которые изменяются в процессе выполнения программы);

3) просмотреть изменение содержимого регистра счетчика CNTR во время выполнения цикла;

4) просмотреть распределение памяти программ *PM* и памяти данных *DM* и удостовериться в правильности выбора значений функций из начального файла *data.dat* (записать несколько значений из буферов, в которые записываются сгенерированные значения заданных функций Sin и Cos из файла data.dat);

10. Вывести графики полученных функций для просмотра в окне *Plot* среды VisualDSP++. Для этого в разделе меню *View* выбрать вкладку *Debug Windows –> Plot –> New*. При этом появится окно *Plot Configuration*. Чтобы вывести графики необходимо:

- в поле *Data Setting* выбрать тип памяти *Memory* (PM или DM), в которой записан сформированный буфер, содержащий значения заданной функции;
- рядом с полем *Address* нажать на кнопку *Browse* и выбрать название того буфера, содержимое которого требуется отобразить графически;
- в поле *Count* задать количество значений, которое требуется вывести на экран (2 периода функции);
- после этого нажать на кнопку *Add*;
- для того чтобы отобразить еще один график, необходимо проделать все вышеперечисленные пункты еще один раз, выбрав название другого буфера, содержимое которого требуется отобразить графически и в конце обязательно нажать *Add*;

11. Зафиксировать результаты, необходимые для отчета (см. ниже).

Содержание отчета

Отчет о проделанной лабораторной работе оформляется каждым студентом индивидуально. На титульном листе необходимо указать № бригады, свою фамилию и т.д. Отчет должен включать в себя:

1) цель работы;
2) рисунок, иллюстрирующий расположение элементов на плате EZ-KIT Lite;
3) результаты выполнения работы:

- текст написанной рабочей программы для заданного варианта;
- значения регистров и распределение памяти данных (программ), которые требуется записать в п.9 порядка выполнения работы;
- графики функций, выведенные в окне Plot среды VisualDSP ++ (п.10);

4) выводы о проделанной работе и полученных результатах.

Таблица 1 - Варианты заданий

Номер бригады	Частота колебания F_3, кГц		Номер бригады	Частота колебания F_3, кГц	
	Sin (x)	*Cos (x)*		*Sin (x)*	*Cos (x)*
1	***4***	***8***	***6***	***12***	***4***
2	***8***	***12***	***7***	***8***	***4***
3	***12***	***8***	***8***	***4***	***12***
4	***16***	***4***	***9***	***20***	***8***
5	***20***	***4***	***10***	***16***	***8***

Вопросы к защите

1. Определение DSP (ЦСП) и их роль в современных технологиях.
2. Основные области применения процессоров ADSP-218x.
3. Особенности разработки программного кода на языке ассемблер.
4. В каких целях можно использовать плату EZ-KIT Lite.
5. Основные элементы, расположенные на плате EZ-KIT Lite. Каковы их функции.
6. Для чего предназначена среда VisualDSP++.
7. Сколько этапов включает в себя процесс отладки проекта в среде VisualDSP++.
8. Каковы особенности работы в режиме моделирования в среде VisualDSP++.
9. Как создать новый проект в среде VisualDSP++.
10. Для чего предназначен раздел меню Project.
11. Какие системы счисления используются в ADSP-218x.
12. Какие действия можно выполнить, используя разделы меню Register, Memory.
13. Как можно настроить вид рабочей области, отображаемые в ней окна.
14. Как добавить файл в созданный проект.
15. Каким образом можно создать файл линкера.
16. Основные разделы главного меню среды VisualDSP++ и их предназначение.
17. Способы адресации, используемые в ADSP-218x.
18. Вектор прерываний ADSP-218x.

ЛАБОРАТОРНАЯ РАБОТА №2.

ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ. ИЗУЧЕНИЕ АРИФМЕТИЧЕСКИХ УСТРОЙСТВ ПРОЦЕССОРА ADSP-218X

***Цель работы*:** Приобретение навыков использования вычислительных возможностей сигнального процессора для реализации базовых алгоритмов цифровой обработки сигналов; изучение работы АЛУ, умножителя, устройства сдвига и т.д., а также различных режимов их работы.

Домашнее задание

Подготовка домашнего задания является ***обязательным*** условием для успешного выполнения лабораторной работы. Поэтому, предварительно необходимо:

ТЕОРИЯ:

1. Внимательно ознакомиться с целью, содержанием, порядком выполнения лабораторной работы, требованиями, предъявляемыми к отчету.
2. Изучить различные блоки, входящие в состав функциональной схемы процессоров семейства ADSP-218x, такие как: арифметические устройства (АЛУ, умножитель, сдвигатель), генератор адреса данных, программный автомат; возможные режимы работы этих устройств и функции, выполняемые ими (сложение, сложение с повышенной точностью, сдвиг, умножение и т.д.).

ПРАКТИКА:

3. Рассчитать величины, которые должны получаться в итоге выполнения программы, т.е. результаты операций сложения, вычитания, умножения и сдвига.
4. В соответствии с вариантом задания написать текст программы на языке ассемблер. При этом *необходимо учитывать*:

а. Программу писать с использованием секций (а не модулей). Названия секциям давать следующие:

.SECTION/DM data1; – Секция описания переменных, буферов в памяти данных DM;
.SECTION/PM data2; – Секция описания переменных, буферов в памяти программ PM;
.SECTION/PM program; – Секция, содержащая в себе программный код, соответствующий варианту задания.

Все эти секции необходимо описать в тексте программы. Также необходимо описать весь вектор прерываний (см. лабораторную работу №1).

б. Операцию деления нужно заменять на умножение обратно-пропорциональным числом;

в. Для выполнения умножения целых чисел перед началом этой операции в тексте программы необходимо написать ENA M_MODE, а для запрета умножения целых чисел – DIS M_MODE.

г. Для использования в вычислениях дробных чисел, их предварительно необходимо перевести в дополнительный код;

Порядок выполнения работы

1. Создать папку *ADSP_lab2* для будущего проекта в директории, указанной преподавателем.

2. Запустить VisualDSP++ с помощью ярлыка на рабочем столе.

3. Создать новый проект в папке *ADSP_lab2* под названием: *Lab2*. *dpj*.

4. Создать новый файл с расширением *Lab2*.dsp в той же папке и добавить его в проект (в папку *Source files*). В нем написать заранее подготовленный текст программы, соответствующий варианту задания.

5. Если задание предусматривает создание файлов данных (*.dat), то их создать в папке *ADSP_lab2* и затем добавить в проект (для этого создать в окне *Project* папку *DATA*).

6. Сгенерировать файл линкера с расширением **.ldf* , необходимый для отладки проекта. Проверить наличие сгенерированного файла в папке Linker Files в окне Project.

7. Затем выполнить компиляцию файла (*Build File* или *Ctrl+F7*). Если Ассемблер выявил ошибки, исправить их и повторить компиляцию. После успешного выполнения компиляции файла, осуществить компоновку проекта (*Build* или *F7*).

8. Запустить программу на исполнение в пошаговом режиме (*F11*). В случае выполнения однотипных операций использовать возможность выполнения до курсора (*Run to Cursor*). В ходе выполнения программы:

- зафиксировать результаты, необходимые для отчета (см. ниже);
- удостовериться в правильности получаемых результатов;
- проследить изменение содержимого всех регистров, используемых в программе;
- просмотреть распределение и содержание областей памяти программ *PM* и памяти данных *DM,* в которые записываются полученные результаты (например, содержимое буферов, значения переменных);

Содержание отчета

Отчет о проделанной лабораторной работе оформляется каждым студентом индивидуально. На титульном листе необходимо указать № бригады, свою фамилию. Отчет должен включать в себя:

1) Цель работы и задание, соответствующее варианту;

2) Домашние расчеты ожидаемых в ходе выполнения программы результатов;

3) Текст отлаженной рабочей программы на языке Ассемблер с комментариями;

4) Результаты выполнения работы, подтверждающие правильность написанной программы. Т.е. фиксируемые результаты должны демонстрировать, что в результате проводимых операций над заданными числами и буферами, получается верный результат. При этом для наглядности, можно фиксировать промежуточные значения регистров и памяти. В связи с тем, что для каждого варианта - свое задание, то и снимаемые результаты будут индивидуальны. Требуется:

– если не указано другого, то результаты записывать в шестнадцатеричном виде. Результаты в памяти программ в десятичном формате не просматривать!

– записать распределение памяти, отображающее содержимое буферов или значения переменных, в которые записываются полученные результаты. При этом

указать адреса и значения ячеек памяти.

– при выполнении операций сдвига для наглядности результаты (содержащиеся в переменных, буферах или регистрах) записывать в двоичном и шестнадцатеричном коде. При этом отмечать, каким регистрам (SR0 или SR1) соответствуют данные значения.

– если требуется записать какое-то значение в память по определенному адресу, то в отчете отметить адрес соответствующей ячейки памяти и ее содержимое.

5) выводы о проделанной работе и полученных результатах.

Варианты заданий

Вариант 1.

1) Определить две переменные в памяти данных (*value_A* и *value_B*).

Определить в памяти программ буфер *REZ* размером *N*=7, N задать как константу, а в памяти данных – циклический буфер *Buf* на 16 элементов.

2) Сложить два числа: 17908 и 0x637C, результат суммы записать в первую переменную; вычислить разность и результат записать во вторую переменную.

3) Вычислить произведение с повышенной точностью двух чисел *Inpt1* = $0x13579BDF_{16}$ (определено как массив, состоящий из 2-х слов в памяти данных) и *Inpt2* = $0x2468ACE0_{16}$ (определено как массив, состоящий из 2-х слов в памяти программ). Результат сохранить в 3ей - 5ой ячейке объявленного массива *Rez*.

4) Заполнить буфер *Buf* значениями 60, 56, 52, … 0, организовав цикл.

Вариант 2.

1) Определить в памяти данных циклический буфер *Bufer* размером *N*=16 и буфер *MLT* на 4 элемента.

Определить 3 переменные (2 в памяти данных – *Vr1* и *Vr2*, и 1 в памяти программ *Sum*), каждую размером 2 слова для хранения значений с повышенной точностью.

2) Произвести сложение с повышенной точностью двух переменных, а результат записать в третью (конкретные значения этих переменных 0x147AD036 и 0x258BE147 ввести в соответствующие ячейки памяти в процессе отладки).

3) Произвести последовательно логический сдвиг числа 0xB769 вправо (hi на 1, 2, 3… 16 разрядов, организовав цикл. Результатом этой операции заполнить циклический буфер *Bufer* (брать значения из SR1).

4) Умножить 179 на 0x70C2 и результат (MR1 и MR0) сохранить в первых двух ячейках буфера *MLT*. Осуществить это же умножение, но с округлением. Полученный результат записать в 3 и 4 элементы буфера *MLT*.

Вариант 3.

1) Определить циклический буфер *Spool* в памяти данных размером 16.

2) Найти результат операции NOT от результата выполнения логического сложения над двумя числами (0x2345 и 456) и записать его в переменную *Log*, определенную в памяти программ.

3) Полученный 16-ти разрядный результат последовательно логически сдвинуть на 1,2…8 разрядов влево с использованием опорного сигнала (LO).

4) Значения регистров SR1 и SR0 записать в буфер *Spool*. Обнулить каждую вторую ячейку буфера *Spool*.

Вариант 4.

1) Определить 3 переменные: 2 в памяти данных (*Prm1* и *Prm2*) и 1 в памяти программ (*Prm3*).

2) Умножить 0.0625 и 0.125 и результат (из MR1) записать в переменную *Prm3*. (Просмотр результатов при этом осуществлять в режиме *Fractional*, который можно выбрать, нажав правую кнопку мыши в окне отображения *Registers*.)

3) Сложить два числа: 2012 и 0x5A1, результат суммы записать в переменную *Prm1*; вычислить разность и результат записать в *Prm2*.

4) Произвести арифметический сдвиг результата, полученного от суммирования вправо на 2 разряда (hi). (В отчет записать значение регистра *SR*)

5) Записать число 0x2654 в регистр *I7* и уменьшить его на 4, 8, 12, 16. Результаты записать в ячейки памяти с адресами 0x1000…0x1003 с использованием косвенной адресации.

Вариант 5.

1) Определить 3 переменных: 1 в памяти данных (*Coef_A*) и 2 в памяти программ (*Coef_B* и *PROD*), каждую размером 2 слова для хранения значений с повышенной точностью.

Создать файл *number.dat*, в котором записать 4 числа: 0x1234, 0x2345, 0x3456, 0x4567 и добавить этот файл в проект. Создать буфер *NMB* в памяти данных и инициализировать его значениями из этого файла.

2) Произвести умножение с повышенной точностью переменных *Coef_A* и *Coef_B*, а результат (из регистров MR1 и MR0) записать в переменную *PROD*. При этом конкретные значения переменных *Coef_A* и *Coef_B* (0x147AD036 и 0x258BE147) ввести в соответствующие ячейки памяти в процессе отладки.

3) Произвести сложение четырех чисел буфера *NMB*, а результат сохранить в переменной *SUM*, которую определить в памяти данных.

4) Произвести арифметический сдвиг значения *SUM* влево на 4 разряда относительно регистра SR0. Результат сдвига (значение SR) сохранить в массиве *SDV*, определенном в памяти данных.

Вариант 6.

1) Определить буфер *Buf_Y* в памяти программ и 2 буфера в памяти данных (*Buf_A* и *Buf_B*) размером 4 элемента.

2) Заполнить буфер *Buf_A* значениями: 0x0001, 0x00A0, 0x0003, 0x0004, а буфер *Buf_B* значениями: 0x0006, 0x0030, 0x0011, 0x0005.

3) Вычислить значения переменной y по формуле: $y = \frac{4a + 3b}{0.2}$, где переменные a и b являются элементами буферов из памяти данных. Записать полученные результаты y в буфер *Buf_Y*.

4) Определить в памяти данных переменную *SUMMA* и записать в нее результат сложения всех четырех элементов буфера результатов y.

5) Произвести логический сдвиг значения *SUMMA* вправо на 3 разряда. Результат сдвига (SR0 и SR1) записать в буфер *SDVIG[2]*, который определить в памяти программ.

Вариант 7.

1) Определить циклический буфер *BFR* в памяти программ размером 18.
   Определить переменную в памяти программ *Rez*.
2) Произвести вычитание с повышенной точностью двух чисел *mn* = 0x167AA036 и *sbt* = 0x456BA147.
3) 24 старших разряда результата записать в переменную *Rez* в памяти программ.
4) Обнулить элементы буфера *BFR* через один, начиная с первого. А каждый второй элемент буфера *BFR* заполнить значениями 72, 64, 56, ….

Вариант 8.

1) Определить два циклических буфера в памяти данных (*Cbuf1* и *Cbuf2*), размером 32.
2) Заполнить буфер *Cbuf1* значениями 2,4,6,…
3) Заполнить буфер *Cbuf2* значениями из буфера *Cbuf1*, возведенными в квадрат.
4) Вычислить разность между шестнадцатыми элементами буферов *Cbuf1* и *Cbuf2*.
5) Произвести арифметический сдвиг полученного числа вправо на 5 разрядов относительно SR1, после чего вычислить его модуль (взяв значение из регистра SR1).

Вариант 9.

1) Определить в памяти данных буфер *GEN* размером 256 элементов, а в памяти программ - буфер *ITOG*, состоящий из 2-х элементов.
2) Заполнить буфер *GEN* значениями от 0 до 255, используя генератор адреса данных и организовав цикл.
3) Произвести логическое сложение и логическое умножение чисел 0x5AF0 и 0xA50F. Результаты записать в ячейки буфера *ITOG* (в отчете результаты этих операций представить в двоичном коде).
4) Записать число 0x1f2e3d в ячейку памяти программ по адресу 0x3f00.

Вариант 10.

1) Определить один буфер в памяти программ (*bufC*) и 2 буфера в памяти данных (*bufA* и *bufB*) размером *N*=12 элементов, где *N* определить как константу.
2) Заполнить буфер *bufA* значениями 1, 3, 5, 7, … буфер *bufB* заполнить удвоенными значениями буфера *bufA*.
3) Сложить между собой элементы буферов *bufA* и *bufB*, результатами заполнить буфер *bufC*.
4) Определить в памяти данных массив *Mac* на 4 элемента. Умножить 2 числа: 0x15A2 и 1537, а результат (MR1 и MR0) записать в первые 2 элемента массива

Mac. Умножить эти же числа с округлением результата, полученный результат записать в 3 и 4 элементы массива *Mac*.

Вопросы к защите

К защите лабораторной работы студент обязан владеть необходимым теоретическим материалом по цифровым сигнальным процессорам ADSP-218x. Так же студент должен грамотно объяснять полученные результаты, уметь правильно связать их с теорией и проследить соответствие этих результатов принципам функционирования тех или иных блоков функциональной схемы цифровых сигнальных процессоров семейства ADSP-218x. Владение средой VisualDSP также является одним из обязательных требований, предъявляемых к студенту на защите лабораторной работы.

ЛАБОРАТОРНАЯ РАБОТА №3.

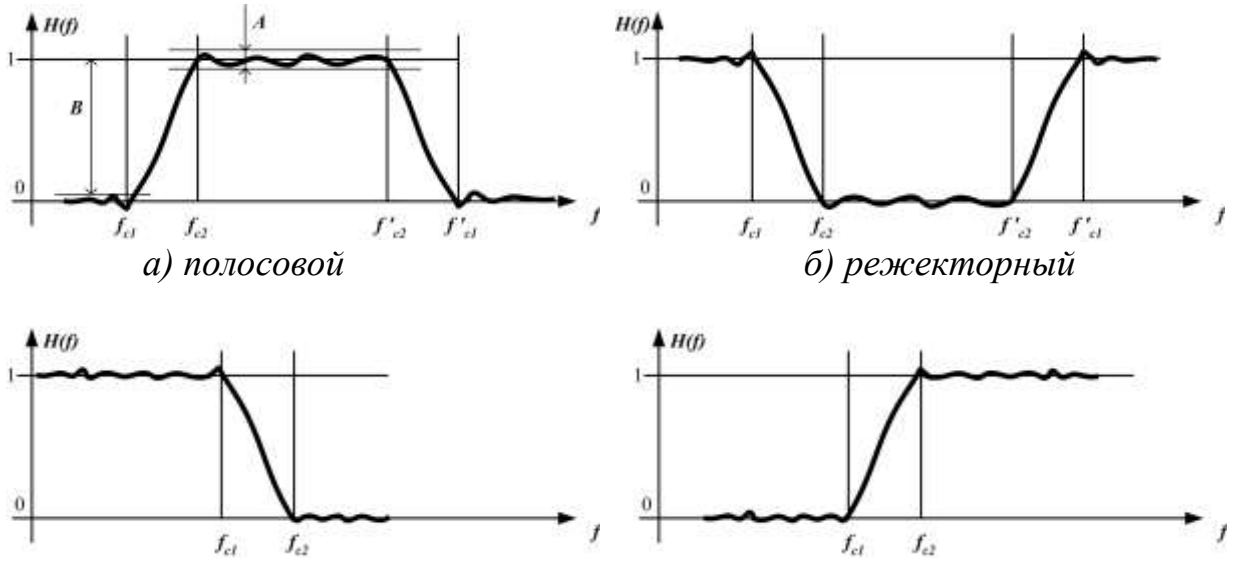
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ КИХ-ФИЛЬТРА

Цель работы: реализация на процессоре ADSP-218x фильтра с конечной импульсной характеристикой (КИХ-фильтра).

Теоретическая часть

Фильтр частотной селекции - это устройство, пропускающее или подавляющее частоты определённого диапазона в составе спектра входного сигнала.

Существует четыре основных типа фильтров частотной селекции - полосовой(П), режекторный(Р), низкочастотный(НЧ) и высокочастотный(ВЧ):



а) полосовой *б) режекторный*

в) низкочастотный *г)* высокочастотный

Рисунок 5.2 - Виды фильтров частотной селекции

НЧ-фильтр пропускает все частоты ниже заданной частоты среза f_{c1} и подавляет частоты, превышающие другую заданную частоту f_{c2} — частоту среза зоны непрозрачности фильтра (рис. 5.2.в).

ВЧ-фильтр пропускает все частоты выше заданной частоты среза полосы пропускания f_{c2} и подавляет частоты ниже частоты среза зоны непрозрачности f_{c1} (рис. 5.2.г).

Полосовой фильтр пропускает только определённую полосу частот и подавляет оставшуюся часть сигнала (рис. 5.2.а). Он характеризуется двумя частотами пропускания и двумя частотами подавления и имеет соответственно две зоны подавления и одну пропускания.

Режекторный фильтр подавляет частоты из определённого диапазона, пропуская на выход все остальные гармоники спектра сигнала (рис. 5.2.б).

Всегда существует погрешность, которая выражается в том, что коэффициент передачи фильтра в зоне непрозрачности не равен нулю, а в полосе пропускания —

не равен единице. Поэтому перед тем, как начать проектирование фильтра, следует задать допустимые уровни погрешности воспроизведения желаемой АЧХ. Итак, при разработке цифрового фильтра частотной селекции исходными данными являются:

— частота дискретизации (sampling frequency) — $f_Д$;

— частоты среза полосы пропускания и частоты среза зоны непрозрачности;

— допустимый уровень неравномерности АЧХ в полосе пропускания - $\varepsilon_{1доп}$;

— допустимый уровень боковых лепестков АЧХ в зоне непрозрачности - $\varepsilon_{2доп}$.

При этом порядок КИХ-фильтра N можно оценить по следующему выражению:

$$N = \alpha\beta L\ \varepsilon_{1доп}, \varepsilon_{2доп}$$

где α - показатель прямоугольности АЧХ фильтра, β - показатель узкополостности АЧХ фильтра, L — логарифмический показатель частотной избирательности.

Чем больше порядок N, тем продолжительнее реакция фильтра. Высокий порядок фильтра требует больших вычислительных затрат и необходимых ресурсов емкости оперативной памяти. Однако чем больше порядок N, тем ближе реальные частотные характеристики к идеальным.

Вход $x(n)$ и выход $y(n)$ КИХ-фильтра связаны друг с другом выражением:

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

где h_k, $k=0,1,..,N-1$ -коэффициенты фильтра; x и y- отсчеты входного и выходного сигналов.

Домашнее задание

Подготовка домашнего задания является ***обязательным*** условием для успешного выполнения лабораторной работы. Предварительно необходимо:

ТЕОРИЯ:

1. Внимательно ознакомиться с целью, содержанием, порядком выполнения лабораторной работы и требованиями, предъявляемыми к отчету.

2. Ознакомиться с теоретическими сведениями, касающимися фильтров с конечной импульсной характеристикой, уделив особое внимание алгоритму формирования выходного сигнала.

ПРАКТИКА:

3. Рассчитать коэффициенты КИХ-фильтра с параметрами, соответствующими номеру бригады. Для этого предлагается воспользоваться средством “Filter Design & Analysis Tool”, встроенным в среду MATLAB. Алгоритм расчета следующий:

1) Запустить среду MATLAB.

2) Вызвать блок проектирования фильтров, набрав в командном окне MATLAB команду ***FDATool***.

3) В поле *Filter Type* выбрать тип проектируемого фильтра (НЧ, ВЧ, П, Р);

4) В поле *Design Method* выбрать следующие параметры: FIR- КИХ-фильтр и equiripple (метод чебышевской аппроксимации).

5) В поле *Filter Order* выбрать проектирование с минимальным порядком –

Minimum order.

6) Значение параметра *Density Factor*, характеризующего плотность распределения точек аппроксимации по оси частот, установить равным 16.

7) В поле *Frequency Specifications* выбрать в качестве единицы измерения частоты герцы (*Hz*), частоту дискретизации *Fs* и соответствующие определенному типу фильтра частоты среза f_{c1}, f_{c2}, f_{c1}' и f_{c2}'.

8) В поле *Magnitude Specifications* выбрать в качестве единицы измерения амплитуды сигнала децибелы (dB), а затем задать неравномерность АЧХ в полосе пропускания (A_{pass}) и уровень подавления сигнала в зоне непрозрачности (A_{stop}) в соответствии с вариантом задания.

9) Произвести расчет коэффициентов фильтра, нажав кнопку *Design Filter*.

10) Выбрать в меню *File* пункт *Export*. В появившемся окне в поле *Export to* выбрать *Text-file*, а в поле *Export as – coefficients* и нажать на кнопку *Apply*. В открывшемся окне выбрать папку, в которой будет сохранен файл с коэффициентами, который назвать как: *firN.dat*, где ***N***-номер бригады.

11) Полученный текстовый файл *firN.dat* содержит коэффициенты фильтра в десятичном формате, разделенные символом переноса строки (например, 0.001865□ 0.003442□ 0.004552□ ….).

Для дальнейшего использования в среде Visual DSP++ необходимо подчеркнуть дробный формат этих чисел, для чего в конце каждого из них добавить символ **r** (например, 0.001865r□ 0.003442r□ 0.004552r□ ….).

4. Входной сигнал представляет собой сумму двух гармонических колебаний с частотами 100Гц и 1кГц. 300 отсчетов входного сигнала хранятся в файле *signal.dat*, который уже создан и его надо будет добавить в проект во время выполнения лабораторной работы.

5. Написать текст программы, осуществляющей цифровую фильтрацию на сигнальном процессоре ADSP-218x, не забыв запретить режим умножения целых чисел.

Порядок выполнения работы

1. Создать папку *ADSP_lab3* для будущего проекта в директории, указанной преподавателем.

2. Запустить VisualDSP++.

3. Создать новый проект в папке *ADSP_lab3* под названием: *Lab3*. *dpj*.

4. Создать новый файл *Lab3.dsp* в той же папке и добавить его в проект (в папку *Source files*). В нем написать заранее подготовленный текст программы, соответствующий варианту задания.

5. Сгенерировать файл линкера с расширением **.ldf* , необходимый для отладки проекта. Проверить наличие сгенерированного файла в папке Linker Files в окне Project.

6. Создать в проекте папку *DATA* и добавить в нее файл *firN.dat*, содержащий рассчитанные дома коэффициенты фильтра, а также файл отсчетов входного сигнала *signal.dat*, находящийся в директории, указанной преподавателем.

7. Выполнить компиляцию файла. Если Ассемблер выявил ошибки, исправить их и повторить компиляцию. После успешного выполнения компиляции файла,

осуществить компоновку проекта.

8. Запустить программу на исполнение в пошаговом режиме (*F11*). В случае выполнения однотипных операций можно воспользоваться командой *Run to Cursor*. В ходе выполнения программы:

- зафиксировать результаты, необходимые для отчета (см. ниже);
- проследить изменение содержимого регистров, используемых в программе;
- просмотреть распределение и содержание областей памяти программ *PM* и памяти данных *DM*, в которые записываются полученные результаты (содержимое буферов, значения переменных);
- построить в одном окне Plot графики двух функций: входного сигнала, загружаемого из файла *signal.dat* и полученного выходного сигнала КИХ-фильтра *y(n)*. Количество выводимых на экран значений *Count* = 300. Зафиксировать графики для отчета.
- во втором окне Plot просмотреть и зарисовать характеристику фильтра, задаваемую рассчитанными в домашнем задании коэффициентами h_k.

Содержание отчета

Отчет о проделанной лабораторной работе оформляется *каждым студентом индивидуально*. На титульном листе необходимо указать № бригады и свою фамилию. Отчет должен включать в себя:

1) цель работы;
2) амплитудно-частотная характеристика заданного типа фильтра;
3) параметры синтезируемого фильтра (из таблицы заданий);
4) текст отлаженной рабочей программы на языке Ассемблер с комментариями;
5) графики полученных функций из окна Plot (входной и выходной сигналы, характеристика фильтра).
6) выводы о проделанной работе и полученных результатах.

Таблица 2 - Варианты заданий к лабораторной работе №3

№ бригады	Тип фильтра	Уровень пропускания ***A***, дБ	Уровень подавления ***B***, дБ	Частота дискретизации, $F_Д$, кГц	Частоты среза			
					f_{c1}, Гц	f_{c2}, Гц	f'_{c2}, Гц	f'_{c1}, Гц
1	НЧ	0,80	80	12	150	750	–	–
2	П	0,60	60	12	200	800	1500	2100
3	ВЧ	0,75	65	12	250	950	–	–
4	Р	0,50	50	12	300	900	1700	2300
5	НЧ	0,60	60	12	100	650	–	–
6	ВЧ	0,70	55	12	200	800	–	–
7	П	0,80	70	12	150	900	1550	2300
8	НЧ	0,50	50	12	250	900	–	–
9	Р	1,0	60	12	200	800	1400	2000
10	ВЧ	0,40	40	12	300	900	–	–

ЛАБОРАТОРНАЯ РАБОТА №4.

ПРОГРАММИРОВАНИЕ ПОСЛЕДОВАТЕЛЬНЫХ ПОРТОВ И РЕЖИМА РАБОТЫ ТАЙМЕРА

Цель работы: изучение принципов и основных режимов функционирования последовательных портов процессора ADSP-2181, алгоритма его взаимодействия с кодеком AD1847, исследование mono- и stereo-режимов AD 1847, фильтрации аудиосигналов и вывод результатов на осциллограф.

В ходе лабораторной работы предполагается исследование прерываний, поступающих от таймера, по приему и передаче данных последовательным портом SPORT0 кодеку, прерывания IRQE, вызываемое нажатием кнопки на плате IZ-KIT Lite.

Описание лабораторной установки

Рисунок 5.3 - Лабораторная установка

Для исследования возможностей платы IZ-KIT Lite, исследуемых в данной лабораторной работе, лабораторная установка включает в себя (рис. 5.3):

- персональный компьютер *(1)*, с установленной средой Visual DSP++;
- непосредственно плату IZ-KIT Lite *(2)*, на цилиндрический штекер *J4* *(3)* которой подается питание через специальный адаптер *(4)*, соединенную с ПК через разъем штепсельного типа *J3* *(5)*;
- осциллограф для снятия выходных сигналов *(6)* (полученных в процессе выполнения программы);
- аудио-провод *(7)* (удлинитель), соединяющий звуковой выход компьютера со стерео гнездом *J1* платы (используют для ввода аудио сигналов);

- стерео-разъем *(8)* для снятия аудио сигналов со стерео гнезда *J2* платы (выход), состоящий из трех элементов *(9, 10, 11)*. К среднему элементу разъема *(11)* присоединяется земля щупа, а с двух других *(9 и 10)* - снимаются сигналы по левому и правому выходным каналам;

- щуп *(12)* для соединения стерео-разъема с осциллографом и выбора канала, по которому просматривается сигнал (левого или правого).

Домашнее задание

Подготовка домашнего задания является обязательным условием для успешного выполнения лабораторной работы. Предварительно необходимо:

ТЕОРИЯ:

1. Внимательно ознакомиться с целью, содержанием, порядком выполнения работы и требованиями, предъявляемыми к отчету.
2. Тщательно изучить теоретический материал по программированию последовательных портов, принципам работы таймера-счетчика и его регистров
3. Ознакомиться с описанием функциональной схемы стерео кодека AD1847, назначением его управляющих регистров и механизмом взаимодействия с последовательными портами процессора.
4. Внимательно рассмотреть и проанализировать алгоритм программы (***см. пункт 4.3***), демонстрирующей установление связи между процессором и кодеком на плате EZ_KIT. Найти место, где должна происходить обработка входных данных.

ПРАКТИКА:

1. Определить значения следующих регистров, используемых при конфигурации SPORT0 (эти значения рассчитываются дома в соответствии с теорией, изложенной выше и представляются в двоичной и шестнадцатеричной системах счисления):

- *Регистр автобуферизации SPORT0* (разрешить автобуферизацию приема и передачи; указать регистры для косвенной адресации для tx_buf и rx_buf, используя один регистр модификации M**1** для tx_buf и rx_buf, т.е. TMREG=RMREG=01, а значения индексных регистров I определить исходя из текста программы);

- *Управляющий регистр SPORT0* (разрешить многоканальный режим работы; внутренняя тактовая синхронизация; задержка кадровой синхронизации на 1, т.е. MDF=0001; длина блока данных=32слова; внешняя кадровая синхронизация приема; сигналы RFS и TFS активны по высокому уровню; выравнивание данных по правому краю; длина последовательного слова = 16).

- *Регистр imask* - четыре значения (при инициализации AD1847 и перед ожиданием прерывания– для прерывания от таймера ***И*** от IRQE).

2. Написать текст процедуры фильтрации под соответствующей меткой в программе *input_samples* на бумаге.
3. Разработать 2 варианта программы:

ПРОГРАММА 1: ***для прерывания IRQE (при нажатии кнопки INTERRUPT).***

ПРОГРАММА 2: ***для прерывания от таймера***

НЕОБХОДИМЫЕ ИЗМЕНЕНИЯ ТЕКСТА ПРОГРАММЫ ВВОДЯТСЯ НЕПОСРЕДСТВЕННО ВО ВРЕМЯ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ (Дома только составляется предварительный план, саму программу (шаблон

основного текста программы *Lab4.dsp*) и коэффициенты ВЧ- и НЧ-фильтров дает преподаватель во время выполнения лабораторной работы).

Порядок выполнения работы

Перед началом выполнения лабораторной работы необходимо подойти к преподавателю, показать и объяснить ему предполагаемые значения следующих регистров.

1. Открыть папку *ADSP_lab4.*
2. Запустить VisualDSP++ с помощью ярлыка на рабочем столе.
3. Создать новый проект в папке *ADSP_lab4* под названием: *Lab4. dpj*.
4. Создать в проекте папку *DATA* и добавить в нее файлы *fir2.dat* и *fir3.dat*, содержащие коэффициенты ВЧ- и НЧ-фильтра.
5. Добавить в проект файл *Lab4.dsp* (в папку *Source files*), содержащий основной текст программы из директории, указанной преподавателем.
6. В файл *Lab4.dsp* внести заранее подготовленные дополнения, в соответствие с домашним заданием **для ПРОГРАММЫ 1 (прерывание IRQE).**
7. Сгенерировать файл линкера с расширением **.ldf* , необходимый для отладки проекта. Проверить наличие сгенерированного файла в папке Linker Files в окне Project.
8. Выполнить компиляцию файла. Если Ассемблер выявил ошибки, исправить их и повторить компиляцию. После успешного выполнения компиляции файла, осуществить компоновку проекта.
9. Запустить аудио-файл на выполнение из папки MUSIC (находится в директории, указанной преподавателем*)*. Аудио-файлы представляют собой совокупность музыки и речи (песни).
10. Запустить программу на исполнение (*F5*).
11. В ходе выполнения программы с помощью щупа просмотреть сигналы, идущие по обоим каналам (элементы стерео-разъема 9 и 10).
12. Зафиксировать результаты, необходимые для отчета (см. ниже);
13. ПОКАЗАТЬ РЕЗУЛЬТАТЫ ПРЕПОДАВАТЕЛЮ
14. Изменить программу для обработки прерываний от таймера (**ПРОГРАММА 2**), а то, что относиться к прерыванию IRQE - *закомментировать*.
15. Выполнить пункты 7-12 для второго варианта программы.
16. ПОКАЗАТЬ РЕЗУЛЬТАТЫ ПРЕПОДАВАТЕЛЮ

Содержание отчета

Отчет должен включать в себя:

1) Цель работы;
2) Значения регистров, которые надо определить по домашнему заданию.
3) Импульсные характеристики фильтров (в окне Plot просмотреть и зарисовать характеристику фильтров, задаваемых коэффициентами из файла *fir2.dat* и *fir3.dat*) ;
4) Текст варианта отлаженной рабочей программы на языке Ассемблер;
5) Сигналы, снятые с осциллографа и их краткое описание.
6) Выводы о проделанной работе и полученных результатах.

ПРИЛОЖЕНИЕ А. ЗНАЧЕНИЯ СИГНАЛОВ, ХРАНЯЩИХСЯ В ГОТОВЫХ ФАЙЛАХ *.dat, ИСПОЛЬЗУЕМЫХ ПРИ ВЫПОЛНЕНИИ ЛАБОРАТОРНЫХ РАБОТ

Таблица 3 – Значения функции Sin(x), заданные в файле *data.dat* (лабораторная работа №1). *Начало*

N отчета	Значение сигнала	N отчета	Значение сигнала	N отчета	Значение сигнала	N отчета	Значение сигнала
1	0x0000	17	0x7fff	33	0x0000	49	0x8001
2	0x0c8b	18	0x7f61	34	0xf375	50	0x809f
3	0x18f8	19	0x7d89	35	0xe708	51	0x8277
4	0x2527	20	0x7a7c	36	0xdad9	52	0x8584
5	0x30fb	21	0x7640	37	0xcf05	53	0x89c0
6	0x3c56	22	0x70e1	38	0xc3aa	54	0x8f1f
7	0x471c	23	0x6a6c	39	0xb8e4	55	0x9594
8	0x5133	24	0x62f1	40	0xaecd	56	0x9d0f
9	0x5a81	25	0x5a81	41	0xa57f	57	0xa57f
10	0x62f1	26	0x5133	42	0x9d0f	58	0xaecd
11	0x6a6c	27	0x471c	43	0x9594	59	0xb8e4
12	0x70e1	28	0x3c56	44	0x8f1f	60	0xc3aa
13	0x7640	29	0x30fb	45	0x89c0	61	0xcf05
14	0x7a7c	30	0x2527	46	0x8584	62	0xdad9
15	0x7d89	31	0x18f8	47	0x8277	63	0xe708
16	0x7f61	32	0x0c8b	48	0x809f	64	0xf375

Таблица 4 – Значения входного сигнала, заданные в файле *signal.dat* (лабораторная работа №3)

N	Значение	N	Значение	N	Значение	N	Значение	N	Значение	N	Значение	N	Значение	N	Значение
1	0.9999999999	31	-0.5000000000	61	0.0000000000	91	-0.5000000000	121	0.9999999999	151	-0.5000000000	181	0.0000000000	211	-0.5000000000
2	0.9323274693	32	-0.4591806800	62	-0.0663020655	92	-0.4068447238	122	0.9323274693	152	-0.4591806800	182	-0.0663020655	212	-0.4068447238
3	0.7472609477	33	-0.3022642316	63	-0.2472609477	93	-0.1977357684	123	0.7472609477	153	-0.3022642316	183	-0.2472609477	213	-0.1977357684
4	0.4938441703	34	-0.0782172325	64	-0.4938441703	94	0.0782172325	124	0.4938441703	154	-0.0782172325	184	-0.4938441703	214	0.0782172325
5	0.2390738004	35	0.1460441546	65	-0.7390738004	95	0.3539558454	125	0.2390738004	155	0.1460441546	185	-0.7390738004	215	0.3539558454
6	0.0499502113	36	0.3036031793	66	-0.9159756150	96	0.5624222244	126	0.0499502113	156	0.3036031793	186	-0.9159756150	216	0.5624222244
7	-0.0244717419	37	0.3454915028	67	-0.9755282581	97	0.6545084972	127	-0.0244717419	157	0.3454915028	187	-0.9755282581	217	0.6545084972
8	0.0337775114	38	0.2538287271	68	-0.8998029151	98	0.6121966767	128	0.0337775114	158	0.2538287271	188	-0.8998029151	218	0.6121966767
9	0.2067727288	39	0.0466316785	69	-0.7067727288	99	0.4533683215	129	0.2067727288	159	0.0466316785	189	-0.7067727288	219	0.4533683215
10	0.4455032621	40	-0.2269952499	70	-0.4455032621	100	0.2269952499	130	0.4455032621	160	-0.2269952499	190	-0.4455032621	220	0.2269952499
11	0.6830127019	41	-0.5000000000	71	-0.1830127019	101	-0.0000000000	131	0.6830127019	161	-0.5000000000	191	-0.1830127019	221	0.0000000000
12	0.8523479859	42	-0.7053322194	72	0.0136774179	102	-0.1606931844	132	0.8523479859	162	-0.7053322194	192	0.0136774179	222	-0.1606931844
13	0.9045084972	43	-0.7938926261	73	0.0954915028	103	-0.2061073739	133	0.9045084972	163	-0.7938926261	193	0.0954915028	223	-0.2061073739
14	0.8215856826	44	-0.7476728974	74	0.0444397212	104	-0.1183525064	134	0.8215856826	164	-0.7476728974	194	0.0444397212	224	-0.1183525064
15	0.6215724127	45	-0.5845653032	75	-0.1215724127	105	0.0845653032	135	0.6215724127	165	-0.5845653032	195	-0.1215724127	225	0.0845653032
16	0.3535533906	46	-0.3535533906	76	-0.3535533906	106	0.3535533906	136	0.3535533906	166	-0.3535533906	196	-0.3535533906	226	0.3535533906
17	0.0845653032	47	-0.1215724127	77	-0.5845653032	107	0.6215724127	137	0.0845653032	167	-0.1215724127	197	-0.5845653032	227	0.6215724127
18	-0.1183525064	48	0.0444397212	78	-0.7476728974	108	0.8215856826	138	-0.1183525064	168	0.0444397212	198	-0.7476728974	228	0.8215856826
19	-0.2061073739	49	0.0954915028	79	-0.7938926261	109	0.9045084972	139	-0.2061073739	169	0.0954915028	199	-0.7938926261	229	0.9045084972
20	-0.1606931844	50	0.0136774179	80	-0.7053322194	110	0.8523479859	140	-0.1606931844	170	0.0136774179	200	-0.7053322194	230	0.8523479859
21	0.0000000000	51	-0.1830127019	81	-0.5000000000	111	0.6830127019	141	0.0000000000	171	-0.1830127019	201	-0.5000000000	231	0.6830127019
22	0.2269952499	52	-0.4455032621	82	-0.2269952499	112	0.4455032621	142	0.2269952499	172	-0.4455032621	202	-0.2269952499	232	0.4455032621
23	0.4533683215	53	-0.7067727288	83	0.0466316785	113	0.2067727288	143	0.4533683215	173	-0.7067727288	203	0.0466316785	233	0.2067727288
24	0.6121966767	54	-0.8998029151	84	0.2538287271	114	0.0337775114	144	0.6121966767	174	-0.8998029151	204	0.2538287271	234	0.0337775114
25	0.6545084972	55	-0.9755282581	85	0.3454915028	115	-0.0244717419	145	0.6545084972	175	-0.9755282581	205	0.3454915028	235	-0.0244717419
26	0.5624222244	56	-0.9159756150	86	0.3036031793	116	0.0499502113	146	0.5624222244	176	-0.9159756150	206	0.3036031793	236	0.0499502113
27	0.3539558454	57	-0.7390738004	87	0.1460441546	117	0.2390738004	147	0.3539558454	177	-0.7390738004	207	0.1460441546	237	0.2390738004
28	0.0782172325	58	-0.4938441703	88	-0.0782172325	118	0.4938441703	148	0.0782172325	178	-0.4938441703	208	-0.0782172325	238	0.4938441703
29	-0.1977357684	59	-0.2472609477	89	-0.3022642316	119	0.7472609477	149	-0.1977357684	179	-0.2472609477	209	-0.3022642316	239	0.7472609477
30	-0.4068447238	60	-0.0663020655	90	-0.4591806800	120	0.9323274693	150	-0.4068447238	180	-0.0663020655	210	-0.4591806800	240	0.9323274693

Таблица 4 – *Продолжение*

N	Значение	N	Значение	N	Значение	N	Значение	N	Значение	N	Значение	N	Значение	N	Значение
241	0.9999999999	249	0.2067727288	257	0.0845653032	265	0.6545084972	273	-0.3022642316	281	-0.5000000000	289	0.0954915028	297	-0.7390738004
242	0.9323274693	250	0.4455032621	258	-0.1183525064	266	0.5624222244	274	-0.0782172325	282	-0.7053322194	290	0.0136774179	298	-0.4938441703
243	0.7472609477	251	0.6830127019	259	-0.2061073739	267	0.3539558454	275	0.1460441546	283	-0.7938926261	291	-0.1830127019	299	-0.2472609477
244	0.4938441703	252	0.8523479859	260	-0.1606931844	268	0.0782172325	276	0.3036031793	284	-0.7476728974	292	-0.4455032621	300	-0.0663020655
245	0.2390738004	253	0.9045084972	261	-0.0000000000	269	-0.1977357684	277	0.3454915028	285	-0.5845653032	293	-0.7067727288		
246	0.0499502113	254	0.8215856826	262	0.2269952499	270	-0.4068447238	278	0.2538287271	286	-0.3535533906	294	-0.8998029151		
247	-0.0244717419	255	0.6215724127	263	0.4533683215	271	-0.5000000000	279	0.0466316785	287	-0.1215724127	295	-0.9755282581		
248	0.0337775114	256	0.3535533906	264	0.6121966767	272	-0.4591806800	280	-0.2269952499	288	0.0444397212	296	-0.9159756150		

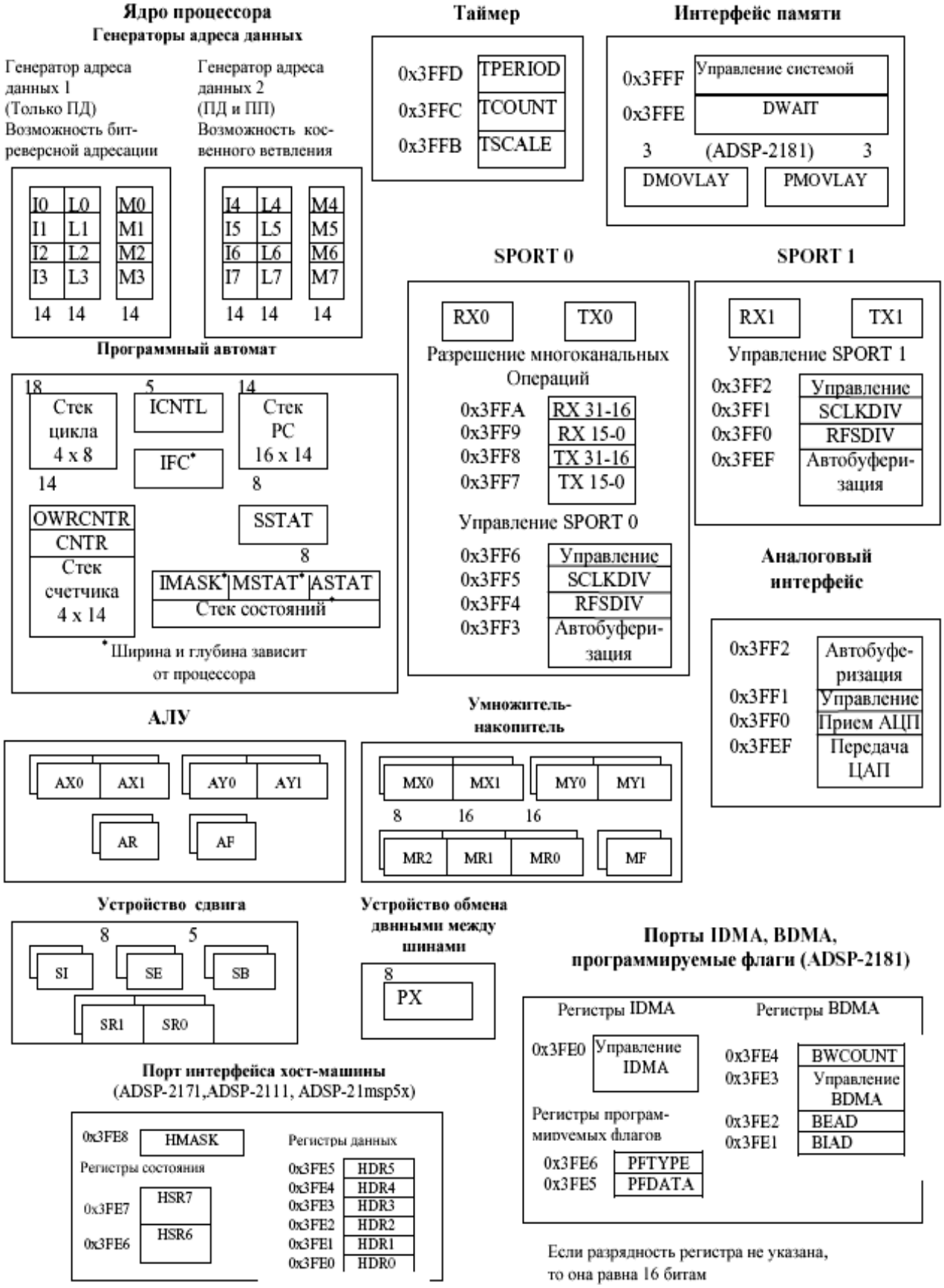
Таблица 5 – Значения коэффициентов ВЧ-фильтра, заданные в файле *fir2.dat*, (лабораторная работа №4).

N	Значение	N	Значение	N	Значение	N	Значение	N	Значение	N	Значение	N	Значение	N	Значение
1	0.002754	8	0.007026	15	-0.007146	22	-0.041997	29	0.937873	36	-0.041997	43	-0.007146	50	0.007026
2	0.008289	9	0.006575	16	-0.011348	23	-0.046815	30	-0.061659	37	-0.036857	44	-0.003453	51	0.007030
3	0.003020	10	0.005667	17	-0.015991	24	-0.051183	31	-0.060275	38	-0.031548	45	-0.000312	52	0.006830
4	0.006043	11	0.004238	18	-0.020980	25	-0.054951	32	-0.058017	39	-0.026212	46	0.002255	53	0.006006
5	0.006006	12	0.002255	19	-0.026212	26	-0.058017	33	-0.054951	40	-0.020980	47	0.004238	54	0.006043
6	0.006830	13	-0.000312	20	-0.031548	27	-0.060275	34	-0.051183	41	-0.015991	48	0.005667	55	0.003020
7	0.007030	14	-0.003453	21	-0.036857	28	-0.061659	35	-0.046815	42	-0.011348	49	0.006575	56	0.008289
														57	0.002754

Таблица 6 – Значения коэффициентов НЧ-фильтра, заданные в файле *fir3.dat*, (лабораторная работа №4).

N	Значение	N	Значение	N	Значение	N	Значение	N	Значение	N	Значение	N	Значение	N	Значение
1	0.001213	8	0.005776	15	0.016990	22	0.030037	29	0.035969	36	0.030037	43	0.016990	50	0.005776
2	0.001103	9	0.007020	16	0.018933	23	0.031520	30	0.035838	37	0.028398	44	0.015096	51	0.004671
3	0.001578	10	0.008399	17	0.020896	24	0.032826	31	0.035450	38	0.026636	45	0.013270	52	0.003703
4	0.002164	11	0.009907	18	0.022855	25	0.033929	32	0.034808	39	0.024778	46	0.011535	53	0.002869
5	0.002869	12	0.011535	19	0.024778	26	0.034808	33	0.033929	40	0.022855	47	0.009907	54	0.002164
6	0.003703	13	0.013270	20	0.026636	27	0.035450	34	0.032826	41	0.020896	48	0.008399	55	0.001578
7	0.004671	14	0.015096	21	0.028398	28	0.035838	35	0.031520	42	0.018933	49	0.007020	56	0.001103
														57	0.001213

Приложение Б. Регистры процессора ADSP-2181



ПРИЛОЖЕНИЕ В. ТИПЫ КОМАНД И УСЛОВНЫЕ ОБОЗНАЧЕНИЯ

Команды процессоров семейства ADSP 2100 сгруппированы в следующие категории:

– вычислительные команды АЛУ, умножителя и устройства сдвига;
– команды пересылки данных;
– команды управления последовательностью выполнения программы;
– многофункциональные команды;
– остальное.

В ходе описания команд используются следующие условные обозначения:

Квадратные скобки **[ ]**	Все, что дается в квадратных скобках, является дополнительной частью оператора команды
Параллельные линии **| |**	В них заключается список операндов. Должен быть выбран один из перечисленных в списке операндов. Если вертикальные линии находятся внутри квадратных скобок, то операнд является дополнительным для данной команды.
ЗАГЛАВНЫЕ БУКВЫ	Заглавными буквами обозначаются названия команд (например, ADD), названия регистров или операндов.
Операнды	Операнды некоторых команд показаны прописными буквами. Эти операнды могут принимать различные значения в кодах ассемблера.
<*exp*>	Обозначает порядок (величину сдвига) в команде непосредственного сдвига; должен выражаться 8-разрядной знаковой целочисленной константой.
<*data*>	Обозначает непосредственное значение данных. Это также может быть символ (метка адреса или имя переменной/буфера), отмеченный оператором "%" или "^".
<*addr*>	Обозначает непосредственное значение адреса, который кодируется в команде. В качестве <addr> может использоваться либо непосредственное значение (константа) либо метка программы.
<*reg*>	Относится к любому доступному регистру
<*dreg*>	Используется по отношению к любому регистру данных

Непосредственные значения <exp>, <data>, <addr> могут быть выражены константами в десятичном, шестнадцатеричном, восьмеричном или двоичном формате.

1. ГРУППА КОМАНД АЛУ

1.1. Сложение/Сложение с переносом

Проверка необязательного условия и, если оно истинно, выполнение сложения заданного типа. Если условие не выполняется, то нет операций и происходит переход к следующей команде. При отсутствии условия в команде операция сложения выполняется без условия. Операнды хранятся в регистрах данных, которые определены в команде.

[IF условие]	AR AF	= хор	+ уор + C + уор + C +постоянная +постоянная+C	;

хор: AX0, AX1, AR, MR0, MR1, MR2, SR1, SR0
уор: AY0, AY1, AF
Разрешенные условия:
EQ – равно нулю
NE – не равно нулю
GT – больше нуля
GE – больше или равно нулю
LT – меньше нуля
LE – меньше или равно нулю
AV – переполнение в АЛУ
AC – перенос в АЛУ
NOT AV – нет переполнения в АЛУ
NOT AC – нет переноса в АЛУ
NEG – операнд X последней команды ABS был отрицательным
POS – операнд X последней команды ABS был положительным
MV – переполнение в умножителе-накопителе
NOT MV – нет переполнения в умножителе-накопителе
NOT CE – счетчик не пуст

Пример: IF EQ AR=AX0 + AY0 + C;
AR=AR+512;

1.2. Вычитание 1

При операции вычитания второй операнд вычитается из первого операнда, к разности может добавляться бит переноса АЛУ (AC) минус 1, и полученный результат записывается в регистр назначения.

[IF условие]				
	AR AF	= хор	- уор -уор + C - 1 + C - 1 -постоянная -постоянная+C-1	;

хор, уор, разрешенные условия – см. пункт 1.1.

Пример: IF GE AR = AX0 –AY0;

1.3. Вычитание 2

[IF условие]					
	AR AF	=	уор -	хор хор + C - 1	;
			- хор + C - 1 -хор+постоянная -хор+постоянная+C-1		

Пример: IF GT AR=AY0 – AX0 + C - 1;

1.4. И / ИЛИ / Исключающее ИЛИ

Выполнение заданной логической операции (логическое И, ИЛИ, исключающее ИЛИ).

[IF условие]					
	AR AF	= хор	AND OR XOR	уор постоянная	;

Пример: AR=AX0 XOR AY0;

```
IF MV AR=MR0 AND 8192;
```

1.5. Проверка бита, установка бита, сброс бита, переключение бита

Выполнение операций с битом.

[IF условие]	AR AF	=	TSTBIT n OF xop SETBIT n OF xop CLRBIT n OF xop TGLBIT n OF xop	;

n – номер бита регистра ***xop*** (0=самый младший бит).
TSTBIT – операцией логического И выбранного бита с 1;
SETBIT – устанавливает выбранный бит в 1;
CLRBIT – сбрасывает выбранный бит в 0;
TGLBIT – изменяет состояние выбранного бита на противоположное.

Пример: `AF=TSTBIT 5 OF AR;`

Выполнение этих команд влияет на состояние битов AZ и AN в регистре состояний арифметических устройств ASTAT. Например, для проверки бита и задания соответствующего условного ветвления может быть использована следующая команда:

```
AF=TSTBIT 5 OF AR;
IF NE JUMP set;          /*переход к "set", если бит 5 регистра AR установлен*/
```

1.6. Команда PASS / СБРОС

Команда PASS 0 является одним из способов очистки содержимого регистра AR. По команде PASS выполняется передача значения в регистр AR или AF из регистров xop, yop или загрузка констант.

[IF условие]	AR AF	= PASS	xop yop постоянная	;

Пример: `IF GE AR= PASS AY0;`
`AR = PASS 8191;`

1.7. Отрицание

Отрицание исходного операнда и запись результата в регистр назначения.

[IF условие]	AR AF	= –	xop yop	;

Пример: `IF LT AR= - AY0;`

1.8. Команда NOT

Выполнение поразрядного инвертирования исходного операнда и запись результата в регистр назначения.

[IF условие]	AR AF	= NOT	xop yop	;

Пример: `IF NE AF = NOT AX0;`

1.9. Нахождение абсолютного значения

Нахождение абсолютного значения исходного операнда и запись результата в регистр назначения.

[IF условие] | AR / AF | = ABS xop ;

Пример: IF NEG AF = ABS AX0;

1.10. Инкремент

Инкрементирование исходного операнда и запись результата в регистр назначения.

[IF условие] | AR / AF | = yop + 1;

Пример: IF GT AF = AF+1;

1.11. Декремент

Декрементирование исходного операнда и запись результата в регистр назначения

[IF условие] | AR / AF | = yop - 1;

Пример: IF EQ AR = AY1 - 1;

1.12. Деление

DIVS yop, xop;
DIVQ xop;

xop: AX0, AX1, AR, MR0, MR1, MR2, SR1, SR0
yop: AY1, AF

Операция деления с одинарной точностью, когда 32-разрядный числитель делится на 16-разрядный знаменатель и в результате получается 16-разрядное частное, выполняется за 16 циклов.

Деление может быть либо знаковым, либо беззнаковым, но числитель и знаменатель обязательно должны быть в одном и том же формате: знаковом или беззнаковом. При операции деления старшая половина бит числителя помещается в любой из разрешенных регистров yop, младшая половина бит числителя - в регистр AY0, а знаменатель - в любой разрешенный регистр xop. Затем выполняется операция деления с примитивам DIVS и DIVQ. По окончанию операции деления частное будет находиться в регистре AY0.

Чтобы выполнить деление знаковых чисел, следует сначала вычислить знак частного, для чего один раз выполняется команда DIVS. Затем команда DIVQ выполняется столько раз, сколько бит остается в частном (т.е. для знакового деления с одинарной точностью DIVS выполняется один раз, а DIVQ - 15 раз).

Чтобы разделить беззнаковое число на беззнаковое, следует сначала поместить старшую половину бит числителя в регистр AF, а затем установить бит AQ равным нулю, непосредственно сбросив этот бит в регистре состояний арифметических устройств ASTAT, что служит указанием на положительный знак частного. Затем команда DIVQ выполняется столько раз, сколько бит имеется в частном (т.е. для беззнакового деления с одинарной точностью DIVQ выполняется 16 раз).

2. ГРУППА КОМАНД УМНОЖИТЕЛЯ–НАКОПИТЕЛЯ

2.1. Умножение

Если условие истинно, перемножение операндов и запись результата в регистр назначения. Если условие не выполняется, то нет операций. При отсутствии условия в команде операция умножения выполняется без условия. Операнды хранятся в регистрах данных, которые определены в команде. Когда MF является операндом назначения, в этом регистре хранятся только биты 31-16 произведения.

```
[IF условие] | MR | = xop * | yop | | (SS) | ;
             | MF |        | xop |   | (SU) |
                                     | (US) |
                                     | (UU) |
                                     | (RND)|
```

xop: MX0, MX1, MR2, MR1, MR0, AR, SR1, SR0
yop: MY0, MY1, MF
Разрешенные условия: EQ, NE, GT, GE, LT, LE, AV, NOT AV, AC, NOT AC, NEG, POS, MV, NOT MV, NOT CE.

Прежде всего, определяют, является ли соответствующий операнд знаковым (S) или беззнаковым (U). Сначала определяется xop, затем - yop. Формат данных не задается по умолчанию; один из форматов должен быть определен. Если за операндами следует RND (округление), умножитель-накопитель перемножает два операнда, округляет результат до 24 самых старших бит (или округляет биты 31-16 до 16 бит, если в результате умножения не происходит переполнения) и записывает результат в регистр назначения. Операнды xop и yop рассматриваются как числа в формате с дополнительным кодом.

Пример: IF EQ MR = MX0*MF (UU);
MF=SR0*SR0 (SS);

2.2. Умножение с накоплением

Перемножение операндов, добавление произведения к значению, содержащемуся в регистре MR, и запись результата в регистр назначения.

```
[IF условие] | MR | = MR + xop * | yop | | (SS) | ;
             | MF |              | xop |   | (SU) |
                                           | (US) |
                                           | (UU) |
                                           | (RND)|
```

Пример: IF GE MR = MR + MX0 * MY1 (SS);
MR=MR+MX0*MX0 (SS);

2.3. Умножение с вычитанием

Перемножение операндов, вычитание полученного произведения из значения, содержащегося в регистре MR, и запись результата в регистр назначения.

```
[IF условие] | MR | = MR - xop * | yop | | (SS) | ;
             | MF |              | xop |   | (SU) |
                                           | (US) |
                                           | (UU) |
                                           | (RND)|
```

Пример: IF LT MR = MR – MX1 * MY0 (SS);

MR=MR – MX0*MX0 (SS);

2.4.Сброс (Clear)

Запись в заданный регистр значения 0. Все 40 бит регистра MR или все 16 бит регистра MF становятся равными нулю.

[IF условие] | MR / MF | = 0 ;

Пример: IF GT MR = 0;

2.5. Пересылка данных регистра MR

Выполняется пересылка данных, содержащихся в регистре MR

[IF условие] | MR / MF | =MR [(RND)] ;

Содержимое регистра MR может округляться по границе 15-ого и 16-ого битов произведения (т.е. остаются 24 старших разряда), для чего в команде следует поставить RND. Если в качестве регистра назначения определен регистр MF, то в этот регистр записываются биты 31-16 полученного произведения.

Пример: IF EQ MF= MR (RND);

2.6. Насыщение регистра MR по условию

IF MV SAT MR;

Проверка бита переполнения умножителя-накопителя (MV) в регистре состояния арифметических устройств, и в случае, когда этот бит установлен, насыщение 32 младших бит 40-разрядного регистра MR; если бит MV не установлен, то нет операций.

3. ГРУППА КОМАНД УСТРОЙСТВА СДВИГА

3.1. Арифметический сдвиг

Осуществляется арифметический сдвиг битов операнда на число позиций и по направлению согласно коду сдвига в регистре SE. Положительные коды сдвига обозначают сдвиг влево (вверх), отрицательные коды -сдвиг вправо (вниз).

[IF условие] SR = [SR OR] ASHIFT xop | (HI) / (LO) |;

xop: SR, SR1, SR0, AR, MR2, MR1, MR0

Разрешенные условия: EQ, NE, GT, GE, LT, LE, AV, NOT AV, AC, NOT AC, NEG, POS, MV, NOT MV, NOT CE.

Сдвиг может производиться относительно старшей (SR1) половины выходной группы разрядов (HI) или относительно младшей (SR0) половины выходной группы разрядов (LO). Над результатом, полученным после операции сдвига, путем выбора опции SR OR может производиться операция логического ИЛИ с текущим содержимым регистра SR.

По команде ASHIFT с положительным кодом биты операнда сдвигаются вправо. Группа из 32 разрядов дополняется по знаку влево (самый старший бит входного значения дублируется влево), а правые биты группы заполняются нулями.

Для сдвига числа с двойной точностью для обеих половинок такого числа используется тот же код сдвига. В первом цикле, старшие биты числа сдвигаются по команде ASHIFT относительно

старших бит выходной группы разрядов (опция HI); в следующем цикле по команде LSHIFT сдвигаются младшие биты числа и при этом используются опции LO и OR.

Пример: IF LT SR= SR OR ASHIFT SI (LO);

3.2. Логический сдвиг

Осуществляется логический сдвиг битов операнда на число позиций и по направлению согласно коду сдвига, который определяется регистром SE.

[IF условие] SR = [SR OR] LSHIFT хор | (HI) (LO) |;

Сдвиг может производиться относительно старшей половины выходной группы разрядов (HI) или относительно младшей половины выходной группы разрядов (LO).

По команде LSHIFT с положительным кодом биты операнда сдвигаются влево на число позиций, определяемое значением в регистре SE. 32-разрядная выходная группа справа заполняется нулями.

По команде LSHIFT с отрицательным кодом биты операнда сдвигаются вправо на число позиций, определяемое значением в регистре SE. 32-разрядная выходная группа слева заполняется нулями.

Пример: IF GE SR= LSHIFT SI (HI);

3.3. Нормализация

Осуществляется арифметический сдвиг битов входного операнда таким образом, чтобы в нем остался только один знаковый бит.

[IF условие] SR = [SR OR] NORM хор | (HI) (LO) |;

Число позиций сдвига берется из регистра SE. В регистр SE, используя команду нахождения порядка, может загружаться соответствующий код сдвига, который позволяет удалить излишние знаковые биты; загруженный код сдвига будет иметь отрицательный знак и равняться числу знаковых бит минус единица.

Пример: SR= NORM SI (HI);

3.4. Нахождение порядка

Выполняется операция нахождения порядка.

[IF условие] SE = EXP хор | (HI) (LO) (HIX) |;

Операция EXP находит порядок входного операнда, что необходимо для последующей операции нормализации (NORM). После выполнения команды EXP полученный код сдвига записывается в регистр SE и используется в последующих операциях как порядок входного значения. Полученный код сдвига зависит от режима, использованного при нахождении порядка (HI, LO или HIX).

Пример: IF GT SE = EXP MRl (HI);

3.5. Нахождение блочного порядка

Операция нахождения блочного порядка выполняется над рядом чисел и в результате

находится порядок наибольшего по модулю числа.

[IF условие] SB = EXPADJ xop ;

В начале блока чисел регистр SB должен быть установлен в исходное состояние и содержать минимальное возможное значение -16. Каждый раз после выполнения команды EXPADJ найденный в ней порядок операнда сравнивается с текущим содержимым регистра SB. Если новое значение порядка превышает текущее значение в регистре SB, содержимое этого регистра обновляется. Таким образом, в конце в регистре SB окажется максимальное найденное значение порядка.

Пример: IF GT SB= EXPADJ SI;

3.6. Непосредственный арифметический сдвиг

Арифметический сдвиг битов операнда на число позиций и по направлению, которые определяются постоянной в поле порядка (<exp>). Положительные постоянные вызывают сдвиг влево (вверх), отрицательные - вправо (вниз).

SR = [SR OR] ASHIFT xop BY <exp> | (HI) (LO) |;

<*exp*> - любая постоянная от -128 до 127.

По команде ASHIFT с положительным кодом биты операнда сдвигаются влево; по той же команде с отрицательным кодом биты операнда сдвигаются вправо. 32-разрядная выходная группа дополняется влево по знаку (самый старший бит входного значения дублируется влево), а вправо - нулями. Биты, сдвигаемые влево от старшего бита (SR_{31}) в 32-разрядном поле назначения, опускаются. Биты, сдвигаемые вправо от самого младшего бита (SR_0) в 32-разрядном поле назначения, также опускаются.

Пример: SR= SR OR ASHIFT SR0 BY 3 (LO);

3.7.Непосредственный логический сдвиг

Логический сдвиг битов операнда на число позиций и по направлению, которые определяются постоянной в поле порядка (<exp>).

SR = [SR OR] LSHIFT xop BY <exp> | (HI) (LO) |;

По команде LSHIFT с положительным кодом биты операнда сдвигаются влево. 32-разрядное выходное поле слева и справа заполняется нулями. По команде LSHIFT с отрицательным кодом биты операнда сдвигаются вправо. 32-разрядное выходное поле слева и справа заполняется нулями.

Пример: SR= LSHIFT SR1 BY -6 (HI);

4. КОМАНДЫ ПЕРЕСЫЛКИ ДАННЫХ

4.1. Пересылка данных между регистрами

Пересылка содержимого исходного регистра по указанному назначению. Содержимое исходного регистра всегда выравнивается в регистре назначения вправо после пересылки.

reg = reg;

Разрешенные регистры **reg*:*** AX0, AX1, AY0, AY1, AR, MX0, MX1, MY0, MY1, MR2, MR1, MR0, SI, SE, SR1, SR0, I0-I7, M0-M7, L0-L7, SB, PX, ASTAT, MSTAT, SSTAT (только для

чтения), IMASK, ICNTL, CNTR, OWRCNTR (только для записи), RX0, RX1, TX0, TX1, IFC (только для записи).

Пример: I7= AR;

4.2. Непосредственная загрузка регистра

Перемещение определенного значения данных по назначенному адресу.

reg = <data>;
dreg = <data>;

data: <constant>, '%'<symbol>, '^' <symbol>,
где % – это оператор"длины", ^ – "указатель на".

Пример: I0 = ^ data_buffer;
L0 = %data_buffer;

4.3. Считывание из памяти данных (*прямая адресация*)

Содержимое ячейки памяти по указанному адресу <addr> перемещается в регистр назначения.

reg = DM (<addr>);

Пример: SI = DM(ad_port0);

4.4. Считывание из памяти данных (*косвенная адресация*)

Содержимое ячейки памяти данных, адрес которой находится суммированием регистров Ix и Mx (при условии, что Lx принимает нулевое значение) перемещается в регистр назначения dreg.

dreg = DM(I0 , M0) ;
I1 M1
I2 M2
I3 M3
I4 M4
I5 M5
I6 M6
I7 M7

Используется косвенный режим адресации с пост-модификацией адреса. При косвенной адресации линейного буфера регистр L, соответствующий используемому регистру I, должен быть установлен равным нулю.

Пример: AY0 = DM(I3,M1);

4.5. Считывание из памяти программ (*косвенная адресация*)

Содержимое ячейки памяти программ, адрес которой находится суммированием регистров Ix и Mx (при условии, что Lx принимает нулевое значение) перемещается в регистр назначения dreg.

dreg = PM(I4 , M4)
I5 M5
I6 M6
I7 M7

В регистр назначения загружаются 16 самых старших бит шины данных памяти программ ($ШДПП_{23-8}$), таким образом, что бит $ШДПП_8$ становится битом 0 регистра назначения. Если разрядность регистра назначения меньше 16 бит, при загрузке опускаются самые старшие биты. Биты $ДПП_{7-0}$ всегда загружаются в регистр PX и могут игнорироваться или считываться в последующем цикле.

Пример: MX1 = PM(I6,M5);

4.6. Запись в память данных (*прямая адресация*)

Перемещение содержимого исходного регистра в определенную в командном слове ячейку памяти данных.

DM (<addr>) = reg;

Пример: DM(cntr_port0)=AR;

4.7. Запись в память данных (*косвенная адресация*)

Содержимое исходного регистра пересылается в заданную в командном слове ячейку памяти данных.

DM(I0 | I1 | I2 | I3 | I4 | I5 | I6 | I7 , M0 | M1 | M2 | M3 | M4 | M5 | M6 | M7) = dreg | <data> ;

Пример: DM(I2 ,M0) = MR1;

4.8. Запись в память программ (косвенная адресация)

Содержимое исходного регистра пересылается в заданную в командном слове ячейку памяти программ.

PM(I4 | I5 | I6 | I7 , M4 | M5 | M6 | M7) = dreg;

Пример: PM(I6,M5) = AR;

4.9. Считывание/запись в область ввода/вывода

При выполнении этих команд производятся пересылки данных между регистрами данных процессора и областью памяти ввода/вывода.

IO(<адрес>) = dreg; /* Запись в область ввода/вывода */
dreg = IO(<адрес>); /* Считывание из области ввода/вывода */

Пример: IO= AX0;
MY1 = IO(2047);

5. КОМАНДЫ УПРАВЛЕНИЯ ПОСЛЕДОВАТЕЛЬНОСТЬЮ ВЫПОЛНЕНИЯ ПРОГРАММЫ

5.1. Переход (JUMP)

По команде JUMP выполнение программы продолжается по указанному в команде адресу.

```
[IF условие]  JUMP | (I4)   |;
                   | (I5)   |
                   | (I6)   |
                   | (I7)   |
                   | <addr> |
```

Могут использоваться прямой или косвенный режимы адресации. При прямой адресации (используется непосредственное значение адреса или метка) адрес программы содержится непосредственно в командном слове. При переходе с косвенной адресацией адрес берется из выбранного регистра I, содержимое которого в данном случае не пост-модифицируется.

Пример: IF NOT CE JUMP top loop; /*CNTR декрементируется*/

5.2. Вызов (CALL)

Команда CALL предназначена для вызова подпрограмм.

```
[IF условие]  CALL | (I4)   |;
                   | (I5)   |
                   | (I6)   |
                   | (I7)   |
                   | <addr> |
```

Для команды CALL также может использоваться как прямая, так и косвенная адресация.

Пример: IF AV CALL scale down;

5.3. Переход или вызов в соответствии с состоянием вывода "Flag In"

Проверка состояния вывода FI процессора и, если на этом выводе единица, выполнение заданного перехода или вызова. Если FI равно нулю, никаких операций не выполняется. При отсутствии условия проверки состояния флага выполняется обыкновенная команда перехода или вызова.

```
IF | FLAG_IN     | | JUMP | | <addr> |;
   | NOT FLAG_IN | | CALL |
```

По команде JUMP выполнение программы продолжается по указанному в команде адресу. Команда CALL предназначена для вызова подпрограмм.

Пример: IF FLAG_IN JUMP service_proc;

5.4. Управление состоянием вывода "Flag Out"

```
[IF условие] | SET    |  | FLAG_OUT | [,...];
             | RESET  |  | FL0      |
             | TOGGLE |  | FL1      |
                       |  | FL2      |
```

Если условие истинно, то на заданном выводе флага устанавливается единица *SET*, ноль *RESET* или работают как тумблеры *TOGGLE*. Если условие не истинно, то выполнение программы продолжается со следующей команды. При отсутствии условия данная операция осуществляется

без условия. Модификация состояний нескольких флагов может осуществляться за счет включения в одну команду нескольких операторов, отделяемых друг от друга запятыми.

Пример: IF MV SET FLAG OUT, RESET FL1;

5.5.Возвращение из подпрограммы (RTS)

По команде RTS происходит возвращение в главную программу из подпрограммы.

[IF условие] RTS;

Пример: IF LE RTS;

5.6. Возвращение из подпрограммы обслуживания прерывания (RTI)

По команде RTI происходит возвращение в главную программу из подпрограммы обслуживания прерывания.

[IF условие] RTI;

Пример: IF MV RTI;

5.7. Команда организации цикла DO UNTIL

Команда DO UNTIL организует циклы с нулевыми издержками. Цикл программы начинается с команды, непосредственно следующей за командой DO, заканчивается по указанному в команде адресу и повторяется до тех пор, пока не удовлетворяется заданное условие окончания цикла (если такое условие задается) или повторяется бесконечно (если такое условие не указано).

DO <addr> [UNTIL условие окончания];

Пример: DO loop label UNTIL CE; /*CNTR декрементируется с каждым прохождением через цикл*/

5.8. Ожидание (IDLE)

По команде IDLE процессор ожидает прерывания в течение неопределенного периода времени находясь в состоянии пониженной мощности. По приходу прерывания оно обслуживается, и программа продолжает выполняться, начиная с команды, следующей за командой IDLE.

IDLE;
IDLE(n); ***/*Команда IDLE с замедлением частоты работы процессора*/***

Команда IDLE(n) замедляет внутреннюю тактовую частоту процессора для еще большего уменьшения потребления энергии. Уменьшенная тактовая частота является программируемой долей от нормальной тактовой частоты и задается по выбору делителем n, который указывается в тексте команды: n = 16, 32, 64 или 128.

6. ПРОЧИЕ КОМАНДЫ

6.1.Управление стеками

По командам управления стеками содержимое назначенных стеков выталкивается или в эти стеки, наоборот, помещаются значения. Вся команда целиком выполняется в течение одного цикла независимо от числа стеков, над которыми производятся вышеназванные действия.

```
[ | PUSH | STS | ]   [,POP CNTR]  [,POP PC]  [,POP LOOP];
  | POP  |
```

По команде PUSH STS выполняется приращение указателя стека состояний на единицу так, чтобы он указывал на новую доступную ячейку в этом стеке, а затем в стек состояний процессора помещается значение регистра состояний арифметических устройств (ASTAT), регистра состояния режима (MSTAT) и регистра маскирования прерываний (IMASK). Эта операция автоматически выполняется каждый раз при входе в подпрограмму обслуживания прерывания.

По любой команде POP из назначенного в ней стека выталкивается его верхнее значение, а указатель этого стека декрементируется.

Пример: POP CNTR, POP PC, POP LOOP;

6.2.Управление режимом

Разрешение (ENA) или блокирование (DIS) назначенного режима процессора. Соответствующий бит состояния режима устанавливается в регистре состояния режима MSTAT при разрешении режима и сбрасывается при его блокировании.

```
| ENA |     | BIT_REV   | [, ...];
| DIS |     | AV_LATCH  |
            | AR_SAT    |
            | SEC_REG   |
            | G_MODE    |
            | M_MODE    |
            | TIMER     |
```

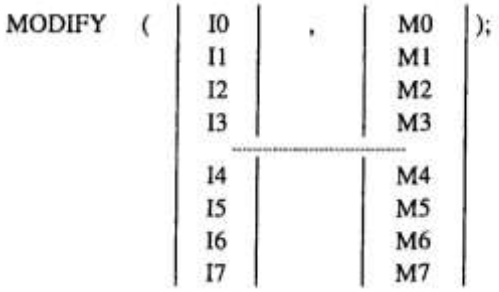
Биты регистра MSTAT:

BIT_REV	Режим бит-реверсной адресации для генератора адреса данных №1
AV_LATCH	Режим фиксации состояния переполнения АЛУ
AR_SAT	Режим насыщения регистра AR АЛУ
SEC_REG	Банк данных теневого регистра
G_MODE	Режим GO разрешен
M_MODE	Режим помещения результата в умножителе-накопителе
TIMER	Таймер активизирован

Пример: DIS AR_SAT, ENA M_MODE;

6.3. Модификация адреса

Модификация содержимого индексного регистра Ix осуществляется как при индексировании адреса памяти, но при этом не происходит никаких пересылок данных.



При адресации линейных (т.е. нециклических) буферов регистр L, соответствующий используемому регистру I, должен быть установлен равным нулю.

Пример: MODIFY (I1, M1);

6.4. Команда NOP (нет операций)

В течение одного цикла не производится никаких операций. Выполнение программы продолжается со следующей за командой NOP команды.

NOP;

6.5. Разрешение/блокирование прерываний

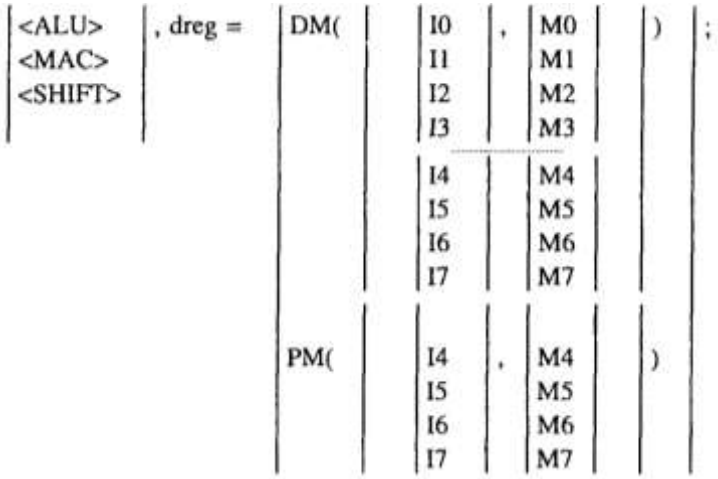
После перезапуска процессора все прерывания по умолчанию разрешены. При выполнении команды DIS INTS все прерывания маскируются, а содержимое регистра IMASK при этом не меняется. По команде ENA INTS все немаскированные прерывания обслуживаются заново.

ENA INTS;
DIS INTS;

7. МНОГОФУНКЦИОНАЛЬНЫЕ КОМАНДЫ

7.1. Вычисление с одновременным считыванием из памяти

Выполнение обозначенной арифметической операции и пересылка данных.



Разрешенные dreg: AX0, AX1, AY0, AY1, AR, MX0, MX1, MY0, MY1, MR0, MR1, MR2, SI, SE, SR0, SR1

При операции считывания данные пересылаются в регистр назначения. Когда арифметическая операция выполняется в комбинации со считыванием из памяти, используется

режим косвенной адресации с пост-модификацией адреса. Считывание из регистров (и памяти) осуществляется в начале цикла процессора, а запись - в конце цикла.

Вычислительная операция должна быть безусловной. Разрешены все операции АЛУ (ALU), умножителя-накопителя (MAC) и устройства сдвига (SHIFT), за исключением непосредственного сдвига и команд АЛУ DIVS и DIVQ.

Пример: `AR = AX0 + AY0, AX0 = DM(I0, M0);`

7.2. Вычисление с одновременной пересылкой данных между регистрами

Выполнение заданной арифметической операции и пересылка данных.

```
| <ALU>  |
| <MAC>  |  , dreg = dreg;
| <SHIFT>|
```

Разрешенные dreg: AX0, AX1, AY0, AY1, AR, MX0, MX1, MY0, MY1, MR0, MR1, MR2, SI, SE, SR0, SR1

Благодаря тому, что процессор сначала считывает, а затем только записывает данные, один и тот же регистр может использоваться как исходный в одном операторе и как регистр назначения в другом операторе. В таких случаях из регистра берется одно значение в начале цикла, а конце цикла в него записывается новое значение.

Пример: `AR = AX0 + AY0, AX0 = MR1;`

7.3. Вычисление с одновременной записью в память

Выполнение обозначенной арифметической операции и пересылка данных.

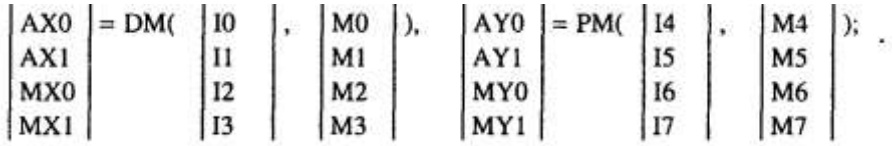
```
| DM( | I0 | , | M0 | ) | = dreg, | <ALU>  | ;
|     | I1 |   | M1 |   |         | <MAC>  |
|     | I2 |   | M2 |   |         | <SHIFT>|
|     | I3 |   | M3 |   |         |        |
|     | I4 |   | M4 |   |
|     | I5 |   | M5 |   |
|     | I6 |   | M6 |   |
|     | I7 |   | M7 |   |
| PM( | I4 | , | M4 | ) |
|     | I5 |   | M5 |
|     | I6 |   | M6 |
|     | I7 |   | M7 |
```

При операции записи содержимое исходного регистра пересылается по указанному адресу в памяти. Когда арифметическая операция выполняется в комбинации с записью в память, используется режим косвенной адресации с пост-модификацией адреса.

Пример: `DM(I0, M0) =AX0, AR = AX0 + AY0;`

7.4. Одновременное считывание из памяти данных и памяти программ

Выполнение нескольких операций считывания из памяти, одно - из памяти данных, другое - из памяти программ.

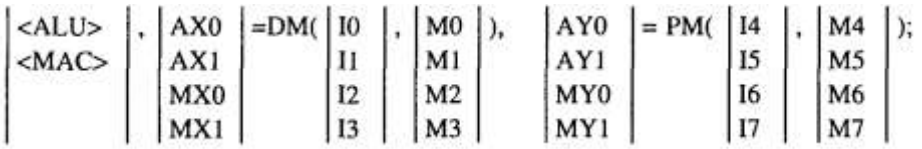


При каждой операции считывания содержимое ячейки памяти пересылается в регистр назначения. Регистрами назначения являются регистры X АЛУ и умножителя-накопителя для выборки из памяти данных и регистры Y - для выборки из памяти программ. Для данной двойной выборки из памяти используется режим косвенной адресации с пост-модификацией адреса.

Пример: AX0 = DM(I1, M2), AY1 = PM(I5, M5);

7.5. Выполнение операций АЛУ/умножителя-накопителя с одновременным считыванием из памяти данных и памяти программ

Данная команда объединяет операцию АЛУ или умножителя-накопителя с двойным считыванием из памяти данных и памяти программ.



При операциях считывания содержимое указанной в команде ячейки памяти пересылается в регистр назначения. Регистрами назначения для считывания из памяти данных являются регистры X АЛУ и умножителя-накопителя, регистрами назначения для считывания из памяти программ - регистры Y. Используется косвенная адресация с пост-модификацией адреса. Результаты вычисления должны записываться в регистр R вычислительного устройства: результаты АЛУ - в регистр AR, результаты умножителя-накопителя - в регистр MR.

Пример: MR = MR + MX0*MY0(UU), MX0 = DM(I0,M0), MY0 = PM(I4,M4);

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Руководство пользователя по сигнальным микропроцессорам семейства ADSP-2100/ Под ред. А.Д. Викторова. СПб гос. электротехнический университет. – СПб, 1997. – 520с.:ил.

2. Цифровые процессоры обработки сигналов семейства ADSP-218x: Учеб. Пособие /А.А.Зайцев, Т.В.Евдокимова; Рязан. гос. радиотехн. акад. Рязань, 2005. 44с.