УДК 621.398
К 84
УДК 621.398.725/727(07)

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ
(ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ)

П.Г. КРУГ

ПРОЦЕССОРЫ ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ

Учебное пособие
по курсу «Микропроцессоры»
для студентов, обучающихся по направлению
«Информатика и вычислительная техника»

МОСКВА ИЗДАТЕЛЬСТВО МЭИ 2001

УДК 621.398
К 84
УДК 621.398.725/727(07)

Утверждено учебным управлением МЭИ в качестве учебного пособия
Рецензенты:
проф., д-р. техн. наук Петров О.М., проф., д-р техн. наук Фролов А.Б.
Подготовлено на кафедре Информационно-измерительной техники

Круг П.Г.
Процессоры цифровой обработки сигналов: Учебное пособие.
М.: Издательство МЭИ. 2001 – 128 с.
ISBN 5-7046-0778-0

Рассматривается описание базовой архитектуры процессоров цифровой обработки сигналов (ПЦОС) и обзор процессоров серий TMS320 компании Texas Instruments Inc.
Пособие включает практический курс, базирующийся на основе ПЦОС TMS320C26, TMS320F206 и TMS320C6211. Отражает опыт подготовки российских и иностранных студентов в области цифровой обработки сигналов в Московском энергетическом институте (техническом университете).
Для студентов, обучающихся по направлению «Информатика и вычислительная техника».

ISBN 5-7046-0778-0

СОДЕРЖАНИЕ

СПИСОК АББРЕВИАТУР

АЛУ – арифметическо-логическое устройство
АЦП – аналого-цифровой преобразователь
ЗУ – запоминающее устройство
ИВП – интерфейс внешней памяти
ИСУ (CAN) – интерфейс сети управления (Control Area Network)
КИХ – конечная импульсная характеристика
МКБПП - многоканальный буферизованный последовательный порт
ОЗУ (RAM) – оперативное запоминающее устройство
ПДП – прямой доступ к памяти
ПЗУ (ROM) – постоянное запоминающее устройство
ПЦОС (DSP) – процессор цифровой обработки сигналов (Digital Signal Processor)
УУ – устройство управления
ХПИ – хост «порт-интерфейс»
ЦАП – цифро-аналоговый преобразователь
ЦП (ЦПУ, CPU) – центральный процессор (центральное процессорное устройство, Cental Processor Unit)
ЧПУ – числовое программное управление
ШИМ – широтно-импульсный модулятор
ШР – шина расширения

CLKR/CLKX – Receive/Transmit Clock (цифровые тактовые сигналы приема/передачи)
DARAM – оперативное запоминающее устройство с двойным доступом
DR/DX – Data Receive/Transmit (цифровые сигналы приема/передачи данных)
DSK – (отладочное средство) Digital Starter Kit (цифровой модуль для начальных работ)
EPROM – постоянное запоминающее устройство с ультрафиолетовым стиранием
FIFO – (память) First Input – First Output («первый вошел – первый вышел»)
FSR/FSX – Receive/Transmit Frame Synchronization (цифровые сигналы кадровой синхронизации приема/передачи)
ICR – Interrupt Control Register (регистр контроля прерываний)
IFR – Interrupt Flag Register (регистр флагов прерываний)
IMR – Interrupt Mask Register (регистр масок прерываний)
MMAC – миллионов операций умножения с накоплением
MIPS – Million Instruction per Second (миллионов инструкций в секунду)
MFLOPS – Million Floating-Point Operation per Second (миллионов операций с плавающей точкой в секунду)
RINT/XINT – Receive/Transmit Interrupt (цифровые сигналы маскируемого прерывания приема/передачи)
RSR/XSR - синхронный регистр приема/передачи последовательного порта
SBSRAM - синхронная пакетная статическая оперативная память
SDTR – Synchronous Data Transmission Register (синхронный регистр приема и передачи)
SDRAM - статическая оперативная память с динамическим рандомизированным доступом
SRAM - статическая оперативная память
SSP – Synchronous Serial Port (синхронизированный последовательный порт)
SSPCR – Synchronous Serial Port Control register (регистр управления послед. портом)
TASM – (язык программирования) турбо-ассемблер
VLIW – (технология) Very Long Instruction Word (очень длинное командное слово)

1. ВВЕДЕНИЕ

1.1. Процессоры цифровой обработки сигналов и реальный масштаб времени

Процессоры цифровой обработки сигналов находят широкое применение в самых различных областях, так, как они способны обеспечивать работу в реальном масштабе времени как существующих, так и принципиально новых устройств.

Процессор цифровой обработки сигналов (сигнальный процессор, Digital Signal Processor) – это микропроцессор, особенностью работы которого является поточный характер обработки больших объемов данных в реальном масштабе времени и, как правило, с интенсивным обменом данных с внешними устройствами. ПЦОС реализуется на основе так называемой *базовой архитектуры (DSP Basic Architecture)*.

Реальный масштаб времени (реальное время работы, Real Time Scale) – это такой режим работы устройства, при котором регистрация и арифметическая обработка (а при необходимости и анализ, визуализация, сохранение, систематизация, синтез и передача по каналам связи) данных производится без потерь информации, поступающей от ее источника [3].

Первый ПЦОС - TMS320C10 был выпущен компанией Texas Instruments Inc. в 1982 году и благодаря целому ряду удачных технических решений сразу получил широкое распространение [4]. Первоначально он был задуман как контроллер, способный эффективно управлять работой модема, а также для оборонных нужд.

Одним из основоположников ПЦОС стал ученый из США Джек Килби. За достижения в технологии полупроводников, совместно с российским ученым Ж. Алферовым, Д. Килби получил нобелевскую премию в области физики в 2000 году.

1.2. Цифровая обработка сигналов

Цифровая обработка сигнала (Digital Signal Processing) — это арифметическая обработка в реальном масштабе времени последовательности значений амплитуды сигнала, определяемых через равные временные промежутки [2].

Аналоговая обработка сигнала, традиционно применяемая в большинстве устройств, является во многих случаях более практичным и дешевым способом достижения требуемого результата. Однако тогда, когда требуется высокая точность обработки данных, реализация устройства в весьма компактном и миниатюрном виде, достижение высокой стабильности характеристик устройства в различных

температурных условиях функционирования, цифровая обработка оказывается единственно приемлемым решением [2,7].

1.3. Типовые задачи, решаемые ПЦОС

На основе ПЦОС создаются устройства в которых требуется реальный масштаб времени выполнения практически любых арифметических задач. В то же время, можно выделить ряд типовых, наиболее распространенных задач, решаемых с применением ПЦОС:

- фильтрация сигналов;
- свертка двух сигналов (смешение сигналов);
- вычисление значений авто и кросс-корреляционной функции двух сигналов;
- усиление, нормализация или преобразование сигналов;
- прямое и обратное Фурье-преобразование;
- и др.

1.4. Основные направления применения ПЦОС

ПЦОС широко применяются в следующих областях (Табл. 1.1) [13]:

- телекоммуникации;
- приборостроение (цифровые функционально- и проблемно-ориентированные приборы);
- автоматизация в промышленности;
- управление техническими системами;
- автомобилестроение;
- медицина;
- обработка голоса и речи;
- обработка изображений и графические станции;
- бытовые приборы;
- оборонные отрасли;
- и др.

Таблица 1.1. Основные направления применения ПЦОС

Системы общего назначения	**Графика и обработка образов**	**Приборостроение**
Цифровая фильтрация Свертка Корреляционный анализ Преобразование Гильберта Быстрое преобразование Фурье Адаптивная фильтрация Взвешивание сигналов Синтез сигналов	3-D вращение изображений Техническое зрение роботов Передача и сжатие изображений Распознавание образов «Улучшение» изображений Гомоморфическая обработка Графические рабочие станции Анимация Цифровые карты	Спектральные анализаторы Функциональные генераторы Анализаторы переходных процессов Цифровые фильтры Синтезаторы сигналов
Обработка голоса и речи	**Управление**	**Оборона**
Голосовая почта Речевые вокодеры Распознавание речи Верификация говорящего «Улучшение» речи Синтез речи Системы «Текст-речь»	Управление дисководом Следящие системы Управление роботом Управление лазерным принтером Управление электроприводом	Специальная связь Обработка радаров Звуковая обработка Обработка изображения Навигация Наведение ракеты на цель
Телекоммуникации		**Автомобилестроение**
Подавление помех ADPCM кодирование Цифровые PBX Сетевые усилители Мультиплексирование каналов Модемы DTMF кодирование/ декодирование	Шифрование данных Факсы Сотовые телефоны Наушники Цифровая речь Коммутация пакетов X.25 Видеоконференции Системы связи	Управление двигателем Вибрационный анализ Управление торможением Установление местонахождения Навигация Голосовые команды Автопилоты
Бытовые приборы	**Промышленность**	**Медицина**
Цифровые магнитофоны Цифровое телевидение Музыкальные синтезаторы Игрушки и «настольные» игры Справочные автоматы (информаторы)	Роботы Конвейеры Станки с ЧПУ Ограничение доступа Промышленные мониторы	Слуховые аппараты Многофункциональный мониторинг Ультразвуковое оборудование Диагностические приборы Протезирование

1.5. Терминологический аспект

1.5.1. «Процессор» или «контроллер»?

Как правильно назвать микросхему, которую вы впервые увидели в документации, на схеме или на плате устройства: *«процессор»* или *«контроллер»*?

В первую очередь, следует обратить внимание на основную функцию, которую решеает данная микросхема. Если основная задача устройства связана с обработкой данных (Data Processing) – то это процессор, если же с управлением (Control) – то это контроллер.

Другими словами, одна и та же микросхема может служить и процессором, и контроллером в зависимости от того, в каком устройстве и для каких целей она используется.

1.5.2. «Микропроцессор» или «нанопроцессор»?

В слове *«микропроцессор»* присутствует однозначная ссылка на то, что схема изготовлена с использованием микротехнологии.

В настоящее время достижения в области полупроводников позволяют изготавливать еще более миниатюрные схемы с использованием нанометровой технологии. Однако среди специалистов термин *«нанопроцессор»* практически не используется.

В то же время в отечественной и зарубежной технической литературе и в материалах компаний-производителей все чаще используется термин *«процессор»*.

2. СОВРЕМЕННЫЕ ПЦОС

2.1. Базовая архитектура ПЦОС

Отличительной чертой задач цифровой обработки сигналов является поточный характер обработки больших объемов данных в реальном масштабе времени, требующий от технических средств высокой производительности и возможности интенсивного обмена с внешними устройствами. Это достигается в настоящее время благодаря специфической архитектуре ПЦОС, называемой базовой архитектурой ПЦОС.

Базовая архитектура ПЦОС – это совокупность характерных особенностей процессора, направленная на повышение его производительности и отличающая ПЦОС от микросхем других типов.

Она обусловлена:

применением модифицированной гарвардской архитектуры;

широкому использованием конвейерного режима работы;

наличием специализированного устройства умножения;

наличием специальных команд для цифровой обработки сигналов;

реализацией короткого командного цикла.

2.1.1. Гарвардская архитектура

Гарвардская архитектура применяется для повышения производительности (быстродействия) и гибкости работы ПЦОС.

Гарвардская архитектура в классическом варианте подразумевает размещение программы и данных в раздельных ЗУ и их передачу по раздельным шинам. Это позволяет полностью совмещать во времени выборку и исполнение команд.

Модифицированная гарвардская архитектура допускает обмен содержимым между памятью программ и памятью данных, что расширяет возможности устройства.

2.1.2. Конвейер

Конвейер также применяется в целях повышения производительности ПЦОС.

Так, в четырехкаскадном ***конвейере*** (например, в ПЦОС TMS320C2xx) ПЦОС может обрабатывать одновременно четыре команды, причем все команды находятся на разных стадиях выполнения.

В четырехкаскадном конвейере предварительную выборку команды, дешифрирование, выборку операнда и исполнение команд можно осуществлять независимым образом. Пока производится предварительная выборка команды N, предыдущая команда N─1 дешифрируется, команда N─2 выбирает операнд, а N─3 исполняется.

Следует отметить, что конвейер реализован автоматически таким образом, что специалист, программирующий ПЦОС никак не замечает его наличие.

2.1.3. Специализированное устройство умножения

Для ПЦОС характерным является наличие аппаратного умножителя, позволяющего выполнять умножение двух чисел за один командный такт. В универсальных же процессорах умножение обычно реализуется за несколько тактов, как последовательность операций сдвига и сложения.

2.1.4. Специальные команды для цифровой обработки сигналов

Другой особенностью ПЦОС является включение в систему специальных команд как, например, умножение с накоплением (MAC): C = A • B + C, с указанным в команде числом выполнений в цикле и с правилом изменения индексов используемых элементов массивов A и B; инверсия битов адреса, разнообразные битовые операции. В ПЦОС реализуется аппаратная поддержка программных циклов и кольцевых буферов, когда один или несколько операндов извлекаются из памяти в цикле исполнения команды.

2.1.5. Короткий командный цикл

В ПЦОС широко используются методы сокращения длительности командного цикла, характерные, в том числе, для универсальных RISC-процессоров.

2.2. ПЦОС с фиксированной и плавающей точкой (арифметикой)

ПЦОС различных компаний-производителей образуют два класса, существенно различающихся по цене: более дешевые ПЦОС обработки данных в формате с фиксированной точкой и более дорогие ПЦОС, аппаратно поддерживающие операции над данными в формате с плавающей точкой.

Использование данных в формате с плавающей точкой обусловлено несколькими причинами. Для многих задач, связанных с выполнением интегральных и дифференциальных преобразований, особую значимость имеет точность вычислений, обеспечить которую позволяет экспоненциальный формат представления данных. Алгоритмы компрессии, декомпрессии, адаптивной фильтрации в цифровой обработке сигналов связаны с определением логарифмических зависимостей и весьма чувствительны к точности представления данных в широком динамическом диапазоне.

Работа с данными в формате с плавающей точкой существенно упрощает и ускоряет обработку, повышает надежность программы, поскольку не требует выполнения операций округления и нормализации данных, отслеживания ситуаций потери значимости и переполнения.

Платой за дополнительные «комфорт и скорость» является высокая сложность функциональных устройств, выполняющих обработку данных в формате с плавающей точкой, необходимость использования более сложных технологий производства микросхем, больший процент отбраковки изделий и, как следствие, дороговизна ПЦОС.

2.3. Обзор ПЦОС серий TMS320 компании Texas Instruments

Анализ сложившегося рынка показывает, что доминирующие позиции в ближайшем будущем будут занимать крупные компании-производители ПЦОС, такие как Texas Instruments Inc., Analog Devices, Motorola и ряд других, которые способны создавать не только ПЦОС низкой себестоимости, но и ежегодно инвестировать новые разработки и ноу-хау, создавать принципиально новые модели и платформы.

В числе наиболее распространенных ПЦОС можно назвать изделия следующих компаний — Motorola (56002, 96002), Intel (i960), Texas Instruments Inc. (TMS320) и Analog Devices (21xx, 210xx) [5]. Выбор процессоров той или иной компании для реализации конкретного проекта — многокритериальная задача, и сформулировать более или менее четкую методику выбора практически невозможно. В связи с этим, рассмотрим изделия одной из них, компании Texas Instruments Inc., доля которой на рынке ПЦОС превышает 50% [1].

Компания Texas Instruments Inc. на рубеже столетий оказалась в заметно обновленном виде [6]. Руководство компании приняло стратегическое решение сконцентрировать силы на упрочении лидирующего положения на рынке ПЦОС, а также других изделий, в первую очередь аналоговых, необходимых для системной интеграции процессоров в прикладные системы.

В период с 1998 по 2000 г. компанией Texas Instruments Inc. были проданы подразделения по производству компьютеров-ноутбуков, схем памяти, оборонной электроники и были приобретены известные фирмы, занимающиеся разработкой прикладного программного обеспечения для ПЦОС (GO DSP, TARTAN, AMATI, Spectron Microsystems). В результате в 1998 году доля Texas Instruments Inc. на рынке ПЦОС вплотную приблизилась к 50% (по результатам 1997 года - 45%). Кроме того, компания Texas Instruments Inc. вышла на первое место в мире по продажам аналоговых и аналого-цифровых схем [1]. Этому способствовало также состоявшееся в 2001 году слияние с компанией Burr-Brown.

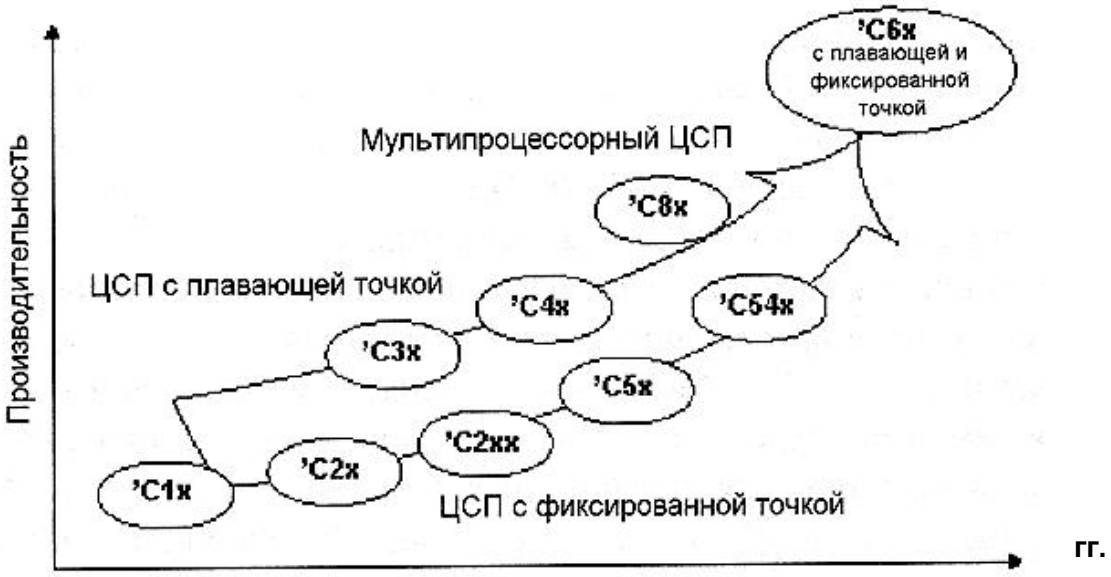


Рис 2.1. Серии ПЦОС TMS320 компании Texas Instruments Inc.

Следует отметить, что в 1998 году компания выделила на исследования и разработки новых ПЦОС 1,2 млрд долларов - величину, близкую к годовому обороту некоторых конкурентов в области производства ПЦОС.

В области технических решений в компании Texas Instruments Inc. также произошел ряд существенных изменений. В 1999 году начато массовое производство кремния по запатентованной технологии TimeLine с разрешением 0,18 мкм.

ПЦОС с фиксированной точкой компании Texas Instruments Inc. представлены сериями ПЦОС (рис. 2.1) [6]: TMS320C1x, TMS320C2x, TMS320C2xx, TMS320C5x и TMS320C62x. Класс ПЦОС с плавающей точкой включает ПЦОС TMS320C3x, TMS320C4x и TMS320C67x. ПЦОС TMS320C8x также поддерживает операции с плавающей точкой и представляет собой мультипроцессорную систему, выполненную в одном кристалле.

Три серии – TMS320C2000, TMS320C5000 и TMS320C6000, по мнению экспертов компании, в ближайшем будущем должны покрыть весь диапазон возможных применений ПЦОС, предоставляя потребителю выбор ПЦОС по критерию "производительность / стоимость / потребляемая мощность".

Серия ПЦОС TMS320C2000 предназначена для решения задач встроенных применений и управления [6]; процессоры отличаются развитой периферией и невысокой стоимостью.

Данную серию представляют универсальные ПЦОС подсерии TMS320C20x и подсерии TMS320C24x для цифрового управления электродвигателями.

Серия ПЦОС TMS320C5000 ориентирована на рынок малопотребляемых портативных устройств и мобильной связи. ПЦОС подсерии TMS320C54xx оптимизированы по быстродействию (до 200 MIPS) и минимальному энергопотреблению (до 32 mA/MIPS) [14]. При этом массовое использование технологии 0,18 мкм позволило снизить стоимость отдельных ПЦОС данной подсерии до 5 $ при производительности 100 MIPS.

Серия ПЦОС TMS320C6000 характеризуется максимальной производительностью для применений, требующих предельных скоростей вычислений как с фиксированной, так и с плавающей точкой. Обе подсерии, TMS320C62x ─ ПЦОС с фиксированной точкой и быстродействием 1600 MIPS и TMS320C67x ─ ПЦОС с плавающей точкой и производительностью от 1 GFLOPS, программно совместимы. Типовые области применений ПЦОС серии TMS320C6000 [14] ─ многоканальные модемы, базовые станции, устройства обработки изображений и др.

ПЦОС всех трех серий могут комплектоваться современными средствами разработки и отладки программ, объединенных единым пользовательским интерфейсом на базе программных средств Code Explorer и Code Composer Studio [8,9].

3. ПЦОС ПОПУЛЯРНОЙ СЕРИИ TMS320C2X

3.1. Структура ПЦОС TMS320C2x

ПЦОС серии TMS320C2x имеют архитектуру аналогичную ПЦОС первого поколения TMS320C1x, но обладают повышенной производительностью и более широкими функциональными возможностями [6,10-12]. Все ПЦОС серии TMS320C2x могут использовать по 64К слов памяти программ и данных, имеют по шестнадцать 16-разрядных портов ввода/вывода и последовательный порт.

ПЦОС серии TMS320C2x (рис. 3.1) имеют возможность использования внешнего контроллера ПДП. Умножитель микропроцессоров помимо операций умножения позволяет выполнять за один такт возведение в квадрат.

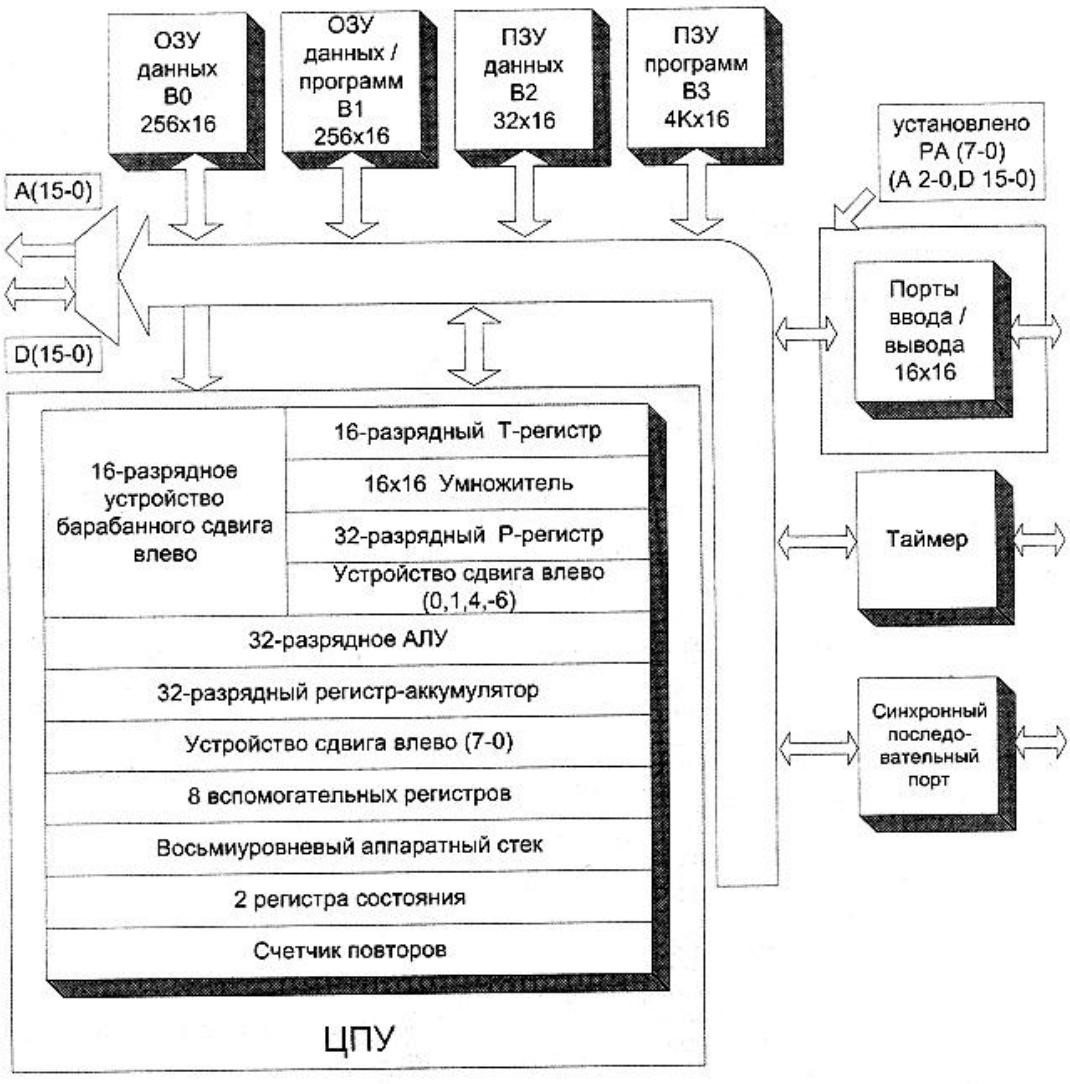


Рис. 3.1. Структура ПЦОС TMS320C2x

В процессоры серии включена аппаратная поддержка кратного выполнения команды, реализован режим двоичной инверсно-косвенной адресации, предназначенный для эффективной реализации быстрого преобразования Фурье.

Основные технические характеристики ПЦОС серии TMS320C2x приведены в табл. 3.1.

Таблица 3.1. Основные характеристики ПЦОС серии TMS320C2x

ПЦОС	Техно-логия	Время цикла, нс	Внутренняя память			Поддержка внешней памяти		Ввод/ вывод		
			ОЗУ	ПЗУ	ППЗУ	Данных	Программ	Посл.	Пар.	ПДП
TMS320C20	NMOS	200	544	----	----	64К	64К	1	16x16	+
TMS320C25	CMOS	100	544	4К	----	64К	64К	1	16x16	+
TMS320C25-50	CMOS	80	544	4К	----	64К	64К	1	16x16	+
TMS320E25	CMOS	100	544	----	4К	64К	64К	1	16x16	+
TMS320C26	CMOS	100	1568	256	----	64К	64К	1	16x16	+

Основные отличия архитектуры ПЦОС серии TMS320C2x от ПЦОС первого поколения TMS320C1x [2,4,6,10-12]:

- выполнение умножения и сохранения результатов в ПЦОС TMS320C2x осуществляется за один командный цикл;
- наличие команд, поддерживающих вычисления с плавающей точкой;
- микросхема ПЦОС включает внутреннее маскируемое ПЗУ программ (ROM) размером 4Кслов для TMS320C25 или ПЗУ с ультрафиолетовым стиранием (EPROM) 4Кслов для TMS320E25;
- выполнение программ осуществляется из памяти программ RAM, расположенной на кристалле;
- объем памяти программ (RAM) — 544 слова, из которых 256 могут быть использованы как память данных;
- расширяемая внешняя память может иметь объем 128К слов (64К слов — память программ, 64К — память данных);
- ПЦОС TMS320C2x содержит внешний интерфейс для организации многопроцессорных связей и средства синхронизации для доступа к разделяемой памяти;
- реализована возможность перемещения содержимого памяти данных и памяти программ блоками;

- реализована возможность организации циклов ожидания при доступе к медленной внешней памяти или медленным периферийным устройствам;
- ПЦОС TMS320C2x содержит на кристалле таймер и последовательный порт;
- микросхема ПЦОС включает пять (TMS320C20) или восемь (TMS320C25) вспомогательных регистров и специальное арифметическое устройство для них;
- микросхема ПЦОС включает аппаратный стек размером четыре слова для TMS320C20 или размером восемь слов для TMS320C25 и возможность программного расширения стека в памяти данных;
- наличие команд обработки битовых данных;
- наличие трех маскируемых пользователем прерываний;
- наличие режима прямого доступа к памяти (только для TMS320C25).

3.2. Графическая среда отладчика ПЦОС TMS320C26

Отладчик, входящий в комплект модуля DSK Starter Kit и работающий в операционой системе MSDOS (рис. 3.2).

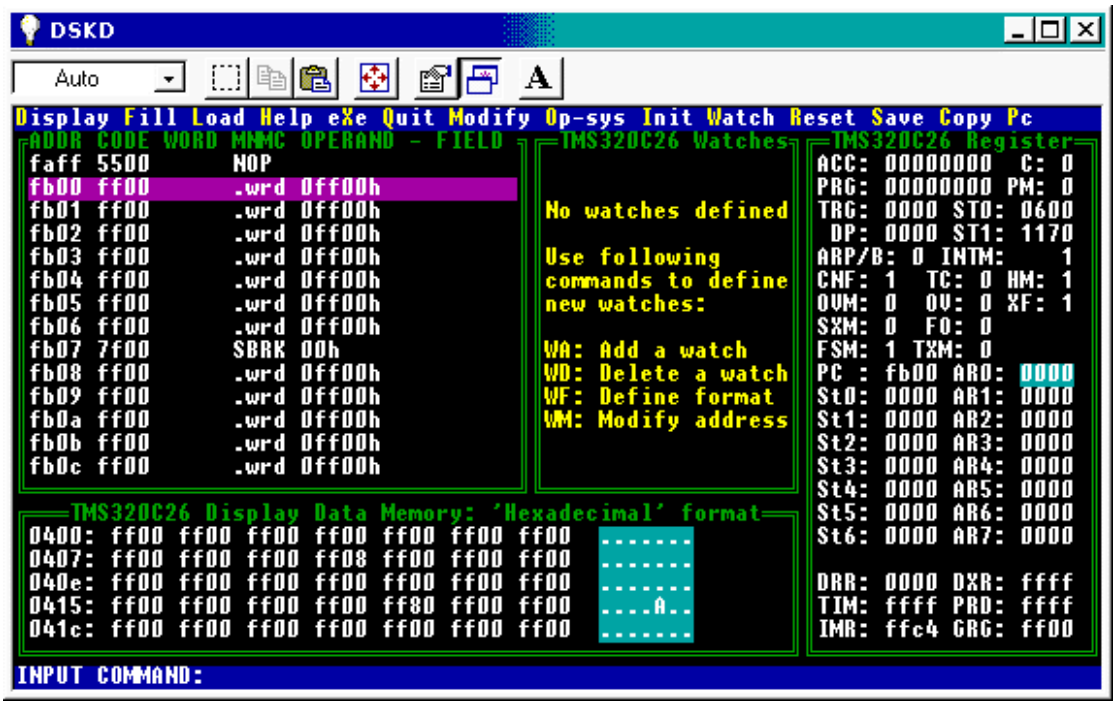


Рис. 3.2. Типовое экранное изображение при работе с отладчиком

Окно программы разделено на несколько функциональных областей: меню; командная строка; область, отображающая состояние основных

регистров и флагов процессора *(TMS320C26 Register)*; область наблюдения *(TMS320C26 Watch)*; области, отображающие распределение в памяти команд ассемблера *(TMS320C26 Reverse Assembler)* и содержимое ячеек памяти в шестнадцатеричной форме *(TMS320C26 Display Data Memory)*.

Названия и назначение пунктов главного меню программы, а также применяемые для их вызова «горячие клавиши» приведены в Табл. 3.2.

Таблица 3.2. Назначение позиций основного меню отладчика

Пункт меню	Горячая клавиша	Назначение
Display	**D**	Просмотр содержимого программной памяти (Program), памяти данных (Data), содержимого статусных регистров и состояния флагов (Status), списка проставленных точек останова (Breakpoints) и др.
Fill	**F**	Заполнение программной памяти (Program) и памяти данных (Data)
Load	**L**	Загрузка созданных на этапе ассемблирования DSK- или COFF-файлов (DSK или COFF), заполнение программной области (Program-hex) или области данных (Format) содержимым файла с расширением *dta*
Help	**H**	Помощь по программе
EXec	**X**	Выполнение загруженной программы в разных режимах: полностью (Run), до заданного адреса (Addr), пошаговым способом (Singlestep/space), заданного количества шагов (Num_steps), загрузка подпрограмм с заданной точкой входа (Call) и др.
Quit	**Q**	Выход из программы
Modify	**M**	Изменение содержимого регистров (Register), программной памяти (Program) и памяти данных (Data)
Op-sys	**O**	Сеанс MS-DOS
Init	**I**	Инициализация процессора
Watch	**W**	Изменение списка (Add, Delete), модификация адресов точек наблюдения (Modify) и изменение формата представления результатов наблюдения (Format)
Reset	**R**	Сброс модуля
Save	**S**	Позволяет сохранить содержимое программной памяти (Program-hex) и памяти данных (Format).
Copy	**C**	Добавление содержимого программной памяти к содержимому памяти данных (Program → data), перемещение участка памяти данных (Data → data) и т.п.
Pc	**P**	Изменение содержимого регистров процессора

Для выполнения действий, соответствующих выбранным пунктам меню, от пользователя часто требуется введение одного или нескольких параметров, как, например, адреса ячейки памяти или ее нового содержимого. Ввод производится в режиме диалога в соответствующем окне. По окончании ввода данных необходимо нажать *Enter*. Пути выхода из диалогового окна: нажать *Esc***;** нажать *Enter***;** нажать *Q* и *Enter***.**

4. УНИВЕРСАЛЬНЫЕ ПЦОС СЕРИИ TMS320C2000

Серия TMS320C2000 (TMS320C2xx) включает недорогие ПЦОС, применяемые в телекоммуникации, приборостроении, промышленности, управлении, в оборонных и других отраслях.

Благодаря наилучшему соотношению производительность-стоимость среди процессоров серий TMS320 ($0,12 за MIPS) ПЦОС TMS320C2xx получили широкое распространение.

ПЦОС TMS320C2xx характеризуют следующие свойства [2,13,14]:

- совместимость по программному коду с сериями 'C1x, 'C2x, т.е. все программы, написанные для процессоров более ранних серий - TMS320C1x и TMS320C2x, могут исполняться на ПЦОС серии TMS320C2xx без изменений. Это позволяет свободно модифицировать давно разработанные и выпускающиеся устройства на основе ПЦОС;
- расширенная система команд для ускорения алгоритмов и оптимизации операций языков высокого уровня;
- высокая производительность (до 40 MIPS);
- низкое потребление энергии благодаря наличию энергосберегающего режима.

Модифицированная гарвардская архитектура, предусматривающая раздельные шины команд и данных, позволяет одновременно выбирать инструкции и операнды. Возможность обмена между памятью программ и данных увеличивает гибкость ПЦОС. Так, коэффициенты, расположенные в памяти программ, могут быть переданы в память данных, что приводит к экономии памяти, выделяемой для коэффициентов.

ПЦОС TMS320C2xx (TMS320F206) имеет перепрограммируемую энергонезависимую flash-память.

Наличие четырехкаскадного конвейера позволяет ПЦОС TMS320C2xx выполнять большинство команд за один такт. Процессор содержит средства управления прерываниями, повторного выполнения операций и вызова подпрограмм и функций. Типовая структура ПЦОС TMS320C2xx приведена на рис 4.1 [2]. Все ПЦОС серии имеют одинаковое процессорное ядро и отличаются различными конфигурациями памяти и внутрикристальной периферией. Все устройства, кроме TMS320C209, имеют по одному синхронному и одному асинхронному последовательному порту.

Синхронный порт предназначен для обмена с другим процессором, кодеком и внешними периферийными устройствами. Порт имеет два буфера памяти емкостью по четыре слова с дисциплиной доступа FIFO и механизмом генерации прерываний. Максимальная скорость обмена через синхронный порт равна половине тактовой частоты процессора (для 40 МГц — скорость обмена 20 Мбит/с).

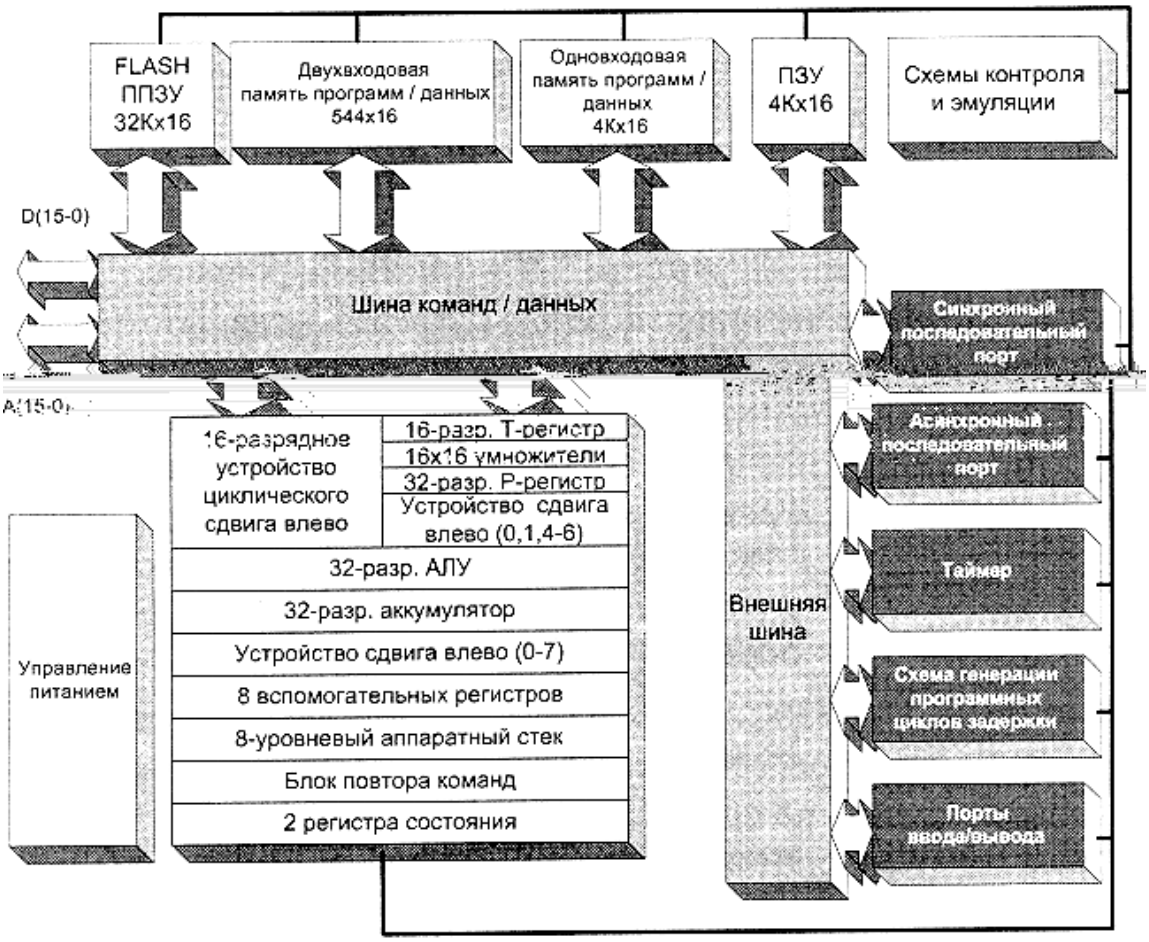


Рис. 4.1. Структура ПЦОС TMS320C2xx

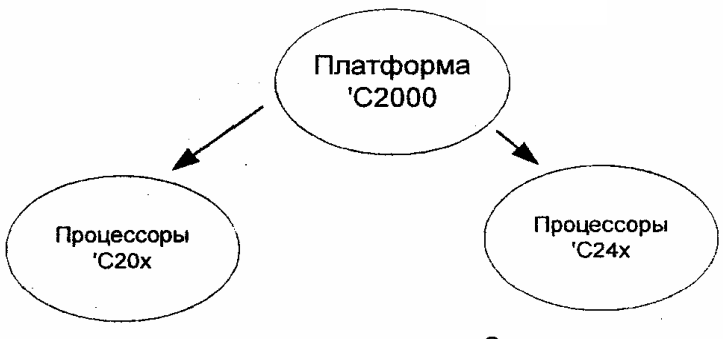


Рис. 4.2. Подсерии ПЦОС TMS320C2xx

Асинхронный последовательный порт предназначен для обмена данными с другими устройствами. При обмене используется 8-битовое представление данных с 1 стартовым и 1 или 2 стоповыми битами.

Скорость обмена может достигать 250 000 10-битовых знаков в секунду.

Серия TMS320C2xx состоит из двух подсерий ПЦОС (рис. 4.2) [6]: TMS320C20x – оптимизированых по цене ПЦОС широкого применения и TMS320C24x - ПЦОС для систем управления (например, управления электроприводом).

4.1. ПЦОС TMS320C20x

TMS320C20x – ПЦОС общего назначения с производительностью до 40 MIPS. Имеют полностью статическую структуру для уменьшения энергопотребления, встроенную Flash-память и широкий набор периферийных устройств. ПЦОС TMS320C20x специально оптимизированы по стоимостным параметрам, чтобы обеспечить их применение в недорогих массовых устройствах, но при этом остаются мощными ПЦОС (рис 4.3) [6].

4.1.1. Ядро (центральный процессор) ПЦОС TMS320C20x

Процессоры серии базируются на одном и том же ядре (центральном процессоре). Это 16-разрядное ядро ПЦОС общего назначения включает 32-разрядное АЛУ, 32-разрядный аккумулятор, однотактовый умножитель 16x16, набор сдвигателей (shifters) и регистры состояния и управления. Ядро имеет развитые средства адресации - 8 дополнительных 16-разрядных адресных регистров, для формирования адреса памяти.

Ядро TMS320C20x построено по модифицированной гарвардской архитектуре с разделением шин и памяти для команд и данных. Такая архитектура позволяет выполнять параллельную выборку команд и операндов. В совмещении с четырехкаскадным конвейером это позволяет процессору производить параллельную обработку нескольких команд.

4.1.2. Память

ПЦОС TMS320C20x имеют на кристалле как обычное ОЗУ, так и ОЗУ с двойным доступом (DARAM). Использование ОЗУ с двойным доступом обеспечивает параллельную выборку двух операндов или выборку и сохранение операндов. Это увеличивает скорость выполнения вычислений за счет большего параллелизма и устранения задержек доступа к ОЗУ. Память с двойным доступом разбита на 2 блока.

4.1.3. Flash-память

Процессор TMS320C20x (TMS320F206) имеет встроенную перепрограммируемую энергонезависимую flash-память. Это позволяет, во-первых, обойтись без внешнего ПЗУ команд, а во-вторых, существенно повысить гибкость и возможности устройства за счет перепрограммирования процессора прямо на плате. Все операции по перепрограммированию flash-памяти могут быть выполнены программно

без каких-либо дополнительных устройств. Следует отметить также, что flash-память работает на частоте процессора без задержек, что обеспечивает возможность не использовать ОЗУ для хранения критичных участков программы, а отвести его целиком для хранения данных.

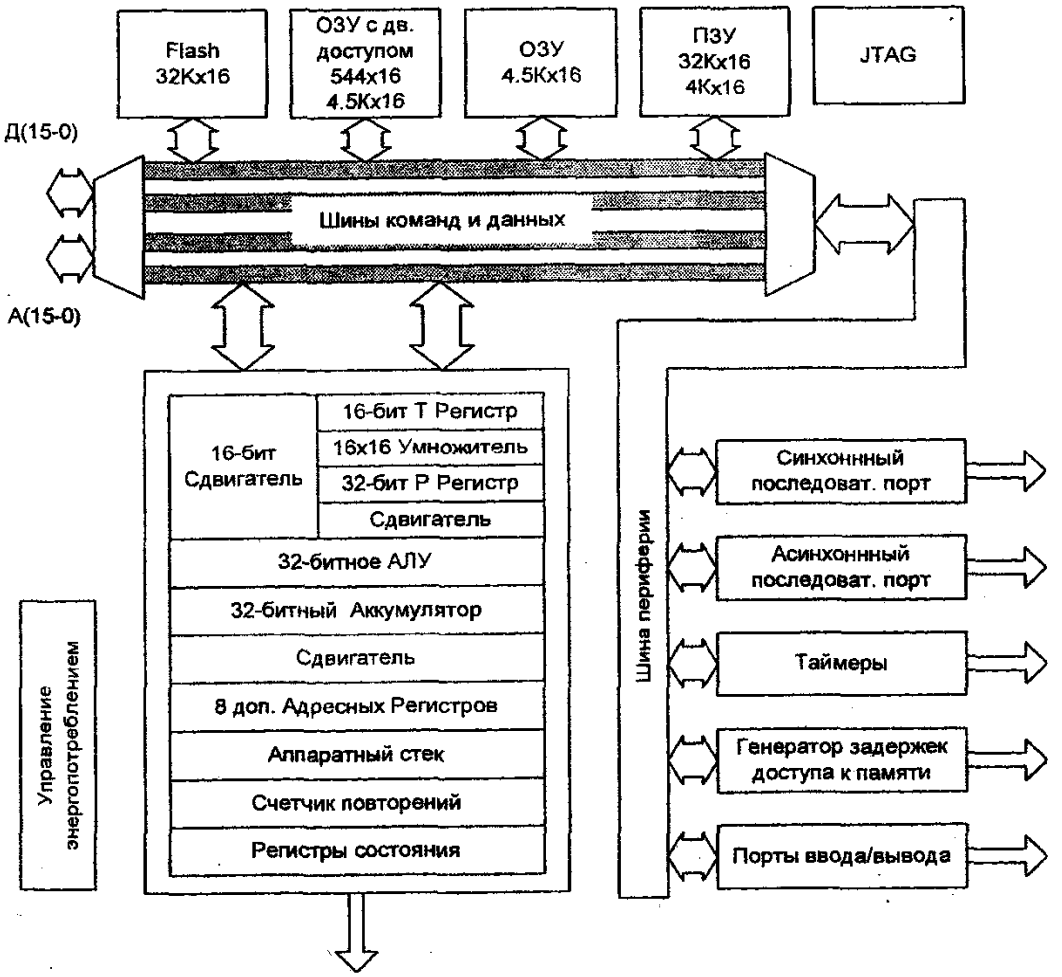


Рис. 4.3. Структура ПЦОС TMS320C20x

4.1.4. Начальный загрузчик

Некоторые модели серии TMS320C20x имеют аппаратный начальный загрузчик, который позволяет загружать программу из внешнего 8-разрядного ПЗУ в ОЗУ команд процессора.

4.1.5. Тактовый генератор

Внутренний тактовый генератор формирует тактовые сигналы процессора. Может работать как с внешним источником тактовых сигналов, так и с кварцевым резонатором, подключаемым непосредственно к встроенному в ПЦОС генератору. Имеет коэффициенты умножения

частоты 0.5 (поделить входную частоту на 2), 1, 2, 4. Наличие умножителя частоты позволяет работать с низкой тактовой частотой и использовать более дешевый тактовый генератор или резонатор.

4.1.6. Программируемый генератор задержек

Устройство служит для увеличения длительности цикла обмена по шине до 7 машинных тактов при обмене с медленными внешними устройствами или с медленной внешней памятью. Он работает без использования какого-либо внешнего аппаратного обеспечения. Количество тактов задержки задается для набора областей памяти через регистры конфигурации.

4.1.7. Расширенный синхронный последовательный порт

Устройство представляет собой синхронный программируемый последовательный порт с максимальной тактовой частотой, равной половине тактовой частоты процессора, что дает скорость передачи до 20 МБит/с. Он может использоваться для подключения ЦАП, АЦП, кодеков и другой периферии с последовательным интерфейсом, а также для соединения процессоров в многопроцессорных устройствах. Со стороны процессора как в приемном, так и в передающем канале порта имеется FIFO глубиной четыре. Порт имеет гибкую программируемую структуру генерации тактовых импульсов и синхронизации, а также программируемые режимы работы. Возможна работа как с 8-, так и с 16-разрядными данными, подключение устройств по протоколу SPI и многоканальная работа для прямого подключения к большинству речевых и телефонных кодеков и плат интерфейса.

4.1.8. Асинхронный последовательный порт

Устройство работает в дуплексном режиме с двойной буферизаций и поддерживает скорость передачи до 2,5 МБит/с. Порт работает с 8-разрядными данными. Может использоваться для соединения по RS-232. Асинхронный последовательный порт имеет средства определения входной скорости обмена данными. 16-разрядный программируемый делитель тактовой частоты позволяет программировать стандартные скорости обмена без необходимости специального подбора частоты тактового генератора.

4.2. ПЦОС TMS320C24x

ПЦОС TMS320C24x - высокоинтегрированные ПЦОС, оптимизированные для цифровых систем управления оборудованием, электродвигателями и т.п. Все процессоры данного семейства базируются на едином ядре платформы TMS320C2xx и имеют широкий набор

периферийных устройств, разработанных для задач построения управляющих систем. Этот набор включает АЦП, ШИМ, последовательные порты, таймеры, порты ввода-вывода. Процессоры серии TMS320C24x выпускаются как со встроенным ПЗУ, так и со встроенной Flash-памятью, что позволяет строить на их базе устройства с возможностью изменения прошитой программы прямо на плате. Это необходимо при построении гибких перенастраиваемых систем с множеством конфигураций для различных типов задач.

Применение ПЦОС TMS320C24x позволяет отказаться от традиционной связки ПЦОС и микроконтроллера и разместить все в одной микросхеме. При этом, ПЦОС TMS320C24x имеет гораздо лучшие параметры, чем большинство популярных контроллеров. Используя ПЦОС TMS320C24x разработчики получают высокоинтегрированное устройство, специально оптимизированное для цифровых систем управления электродвигателями с возможностями прямого съема данных и прямого управления устройством через встроенную периферию.

ПЦОС TMS320C24x предназначены для решения широкого круга задач, таких как управление мощными электромоторами в промышленности, встроенные системы управления во всевозможную технику - вентиляторы, кондиционеры, системы электронных измерений. ПЦОС TMS320C24x могут применяться и в автомобилестроении в системах управления расходом топлива, тормозных антиблокировочных системах - там, где особенно важна именно высокая скорость принятия решения, а следовательно, высокая производительность, характерная для ПЦОС. Ядро, структура шин и памяти ПЦОС TMS320C24x аналогичны устройствам ПЦОС TMS320C20x, которые рассмотрены в предыдущей главе. Далее мы рассмотрим специфичные для ПЦОС TMS320C24x периферийные устройства.

4.2.1. АЦП

ПЦОС TMS320C24x содержит два 10-разрядных АЦП со встроенным устройством выборки-хранения (УВХ). Минимальное время преобразования АЦП равно 0,85 мкс. Процессор имеет 16 аналоговых входов, которые подаются на АЦП через два 8-входовых мультиплексора. Каждый из двух АЦП работает независимо и может запускаться как от внешнего вывода запуска, так и программно.

4.2.2. Последовательные порты

ПЦОС TMS320C24x имеет 2 вида последовательных портов - асинхронный SCI и синхронный SPI.

Асинхронный последовательный порт предназначен для обмена с периферией или другим процессором по стандартному асинхронному протоколу NRZ и имеет:

программируемую скорость передачи данных;
программируемую разрядность данных (от 1 до 8 бит);
программируемое количество стоповых бит (1 или 2);
логику детектирования и индикации ошибок;
мультипроцессорный режим работы со стартом по детектированию свободной линии;
периферийный режим работы со стартом по адресному биту;
полу- или полно дуплексный режим работы;
двойную буферизацию;
отдельные прерывания для приемника и передатчика.

Синхронный последовательный порт - скоростной порт для обмена данными длинной от 1 до 8 бит. Имеет программируемую фазу и полярность тактового сигнала. Может работать в режиме ведущий/ведомый, что позволяет обеспечивать связь нескольких устройств.

4.2.3. Оптимизированный блок управления

Данный модуль предназначен для построения законченных быстродействующих систем управления электроприводом. Он обеспечивает все функциональные возможности по организации ШИМ и ввода-вывода для управления всеми типами электроприводов.

В состав блока управления входят три таймера и до 9 компараторов, что в сочетании с гибкой логикой генерации импульсов дает возможность организации до 12 ШИМ выходов. При этом поддерживается широкий спектр ШИМ-режимов.

Кроме того, в блок управления входят до 4 входов, два из которых имеют схемы декодирования сигналов от оптических датчиков.

4.2.4. Интерфейс сети управления

ПЦОС TMS320C24x имеют в своем составе модуль ИСУ. Эта сеть использует последовательный мультимастерный протокол обмена, который обеспечивает эффективную поддержку распределенной сети управления в реальном времени. При этом обеспечивается высокий уровень защиты данных и высокая скорость передачи данных - до 1 МБит/с (при частоте системы 13 МГц) или 769 КБит/с (при частоте системы 10 МГц). Шина ИСУ - идеальное решение для приложений, работающих в зашумленной среде с высоким уровнем помех, таких как автомобили или другие промышленные приложения, требующие надежную последовательную связь по общей проводке.

По мультимастерной шине передаются приоритетные сообщения длиной до 8 байт с использованием протокола арбитража шины и механизма обнаружения ошибок, что обеспечивает высокую достоверность данных.

Модуль ИСУ обеспечивает обмен в соответствии со спецификацией CAN2.0B с некоторыми дополнительными возможностями. Использование аппаратного модуля существенно снижает нагрузку на процессор при обмене данными. При обмене по шине используется понятие объект сообщения. Модуль ИСУ может быть конфигурирован как несколько объектов сообщения (до 16), при этом у каждого объекта имеется собственная конфигурация, сегмент управления, буфер данных, идентификатор и схема фильтрации. Процессор управляет модулем ИСУ и обменивается сообщениями через коммуникационную память.

Фактически модуль ИСУ представляет собой отдельное устройство со своим внутренним процессором протокола, ПЗУ команд и ОЗУ данных.

4.2.5. Сторожевой таймер и модуль прерываний

Сторожевой таймер служит для контроля работы аппаратного и программного обеспечения и генерирует сигнал сброса процессора, если в течение определенного периода времени к нему не было обращений с записью корректного ключа.

Модуль прерываний реального времени служит для периодической генерации прерываний через установленный интервал с программируемой частотой от 1 до 4096 прерываний в секунду.

4.2.6. Технические параметры

ПЦОС подсерии TMS320C24x предназначены для реализации систем управления и имеют производительность до 30 MIPS и напряжение питания 3,3 В. ПЦОС выпускаются как со встроенным ПЗУ, так и со встроенной Flash-памятью, при этом аналогичные ПЦОС с ПЗУ и с Flash-памятью совместимы вывод-в-вывод, что позволяет производить отладку на Flash-ПЦОС, а выпускать продукцию с применением более дешевых процессоров с ПЗУ, без каких-либо изменений в устройстве. Для обмена с другими устройствами в систему включен новый последовательный интерфейс SCI и 40 контактов ввода/вывода.

ПЦОС TMS320C24x содержат до 16 каналов ШИМ с расширенными функциями и до 4 таймеров. Модуль АЦП имеет время преобразования 500 ns и до 16 входных каналов.

Отметим также, что ПЦОС TMS320C24x работают в двух температурных расширенных диапазонах: от –40 С до +85 С и от –40 С до +125 С.

Данные возможности позволяют строить на базе ПЦОС TMS320C24x одночиповые высокопроизводительные и надежные системы управления [6].

4.3. Средство отладки ПЦОС TMS320C2xx - DSK Code Explorer

TMS320C2xx DSK Code Explorer представляет собой визуальную среду разработки программ процессоров цифровой обработки сигналов серии TMS320C2xx компании Texas Instruments Inc.

Программа предназначена для работы под управлением операционной системы Ms Windows и имеет простой и удобный графический интерфейс, основанный на системе меню. *Code Explorer* позволяет осуществлять исполнение и загрузку в память ПЦОС предварительно ассемблированных с помощью соответствующей утилиты программ с последующей визуализацией результатов во временной и частотной областях, а также просмотр состояния и изменение содержимого регистров процессора, областей программной памяти и памяти данных.

4.3.1. Знакомство с графической средой отладчика

Основное окно программы представлено на рис. 4.4.

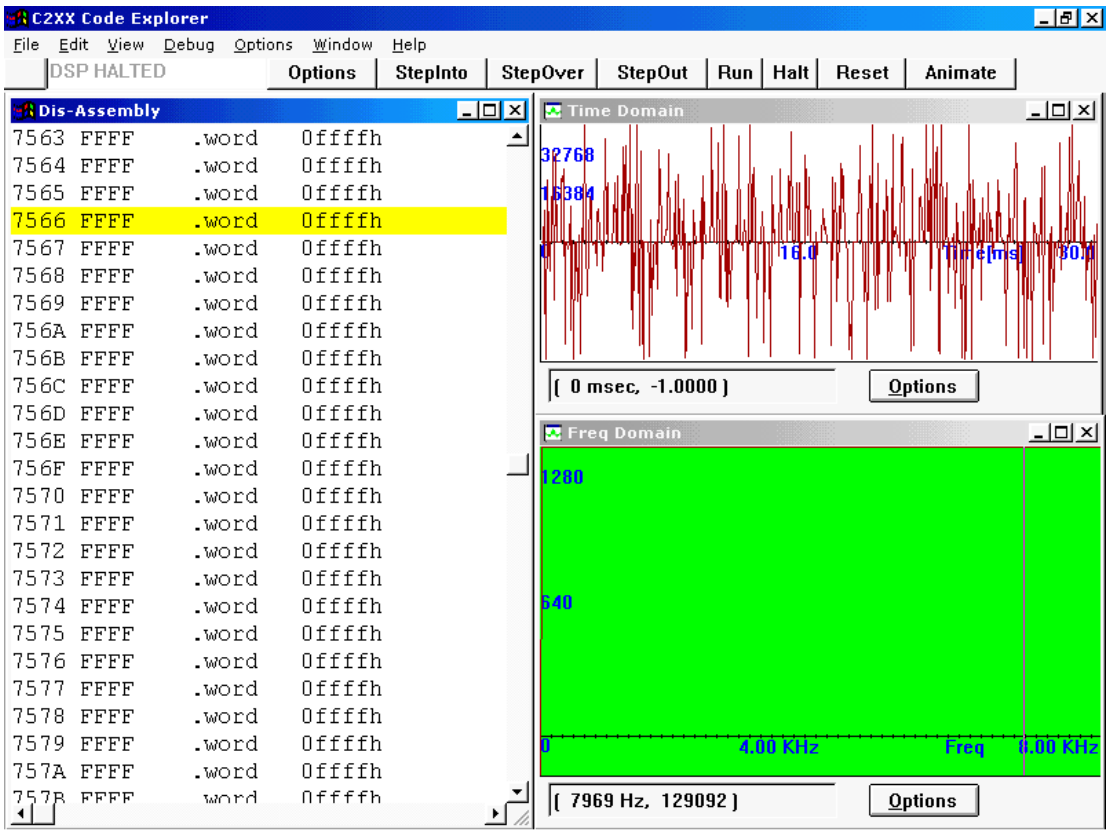


Рис. 4.4. Основное окно Code Explorer

В верхней части окна расположено меню, реализующее основные функции программы. Его наиболее часто используемые команды выведены на расположенную непосредственно под строкой меню панель инструментов в виде кнопок (*Options, StepInto, Run* и т.д.). Наряду с кнопками на панели инструментов находится статусное окно, отображающее текущее состояние ПЦОС (*DSP Halted, DSP Run и Animate*).

4.3.2. Основное меню Code Explorer

Главное меню программы изображено на рис. 4.5.

Команды этого меню позволяют реализовать базовые операции программы, связанные с загрузкой пользовательской программы в память процессора и сохранением результатов ее работы.

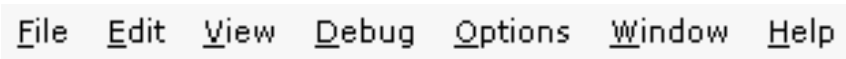


Рис. 4.5. Главное меню Code Explorer

4.3.3. Пункт меню File

Команды этого меню (рис. 4.6.) позволяют реализовать базовые операции программы, связанные с загрузкой пользовательской программы в память процессора и сохранением результатов ее работы.

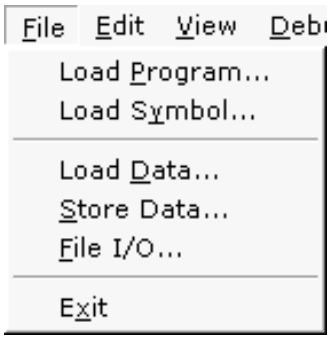


Рис. 4.6. Подпункты меню File

Использование команды *Load Program…* позволяет загрузить в программную память ПЦОС объектный код программы, генерируемый утилитой DSK ассемблера и располагающийся в файле с расширением *dsk*. После выбора этого пункта меню необходимо указать требуемый *dsk*-файл. Как только начинается процесс загрузки, Code Explorer пытается найти в директории-источнике *dsk*-файла файл с расширением *lst* для генерации

информации об имеющихся в пользовательской программе символах (например, метках).

В случае, если источник символьной информации не найден или не соответствует содержанию загруженного *dsk*-файла, будет выведено предупреждающее сообщение. Для того чтобы гарантировать возможность использования символьной информации при отладке программы, необходимо выполнять ассемблирование ее текста с ключом *-l*, что вызовет автоматическую генерацию необходимого *lst*-файла.

Перед тем, как загрузить программу, необходимо удостовериться, что она не конфликтует с ядром программы *Code Composer*, занимающим области *0h ÷160h* программной памяти и *60h ÷ 7Fh* памяти данных (блок *B2*).

При необходимости отладчик позволяет загрузить в память ПЦОС только информацию о символах, содержащихся в пользовательской программе. Для этого используется команда *Load Symbol*. Ее применение может быть полезным в такой ситуации, когда отладчик не в состоянии загрузить объектный код программы (например, этот код расположен в ROM). Исполнение этой команды вызывает очистку существующей таблицы и при этом, однако, не приводит к модификации памяти и установке точки входа программы.

Для загрузки памяти ПЦОС непосредственными данными, начиная с указанного адреса, используется команда *Load Data…*. Команда *Store Data…* позволяет сохранить в файл содержимое памяти процессора с некоторого адреса. Подразумевается, что загружаемые (сохраняемые) данные хранятся в текстовом файле формата *GO DSP*. Понятие формат в данном случае подразумевает обязательное присутствие в составе файла заголовка и блока сохраняемых данных.

Заголовок должен иметь следующий формат:

Идентификационный_номер
Формат_данных
Стартовый_адрес
Страница_памяти
Длина

где: идентификационный_номер – фиксированной число, равное *1651;*
формат_данных – целое число в диапазоне *1 … 4*, где 1 соответствует шестнадцатеричному, 2 – целочисленному формату, 3 говорит о том, что данные в файле представлены в виде длинных целых, а 4 – в виде чисел с плавающей точкой;
стартовый_адрес – начальный адрес целевого блока памяти данных;
длина - определяет количество элементов данных в нем;
страница_памяти – номер страницы памяти данных, в состав которой входит блок.

Вариант такого файла см. в *Примере 4.1.*

Code Explorer позволяет осуществить как потоковый ввод дынных в ПЦОС из файла, так и их потоковый вывод из памяти процессора в файл на диске непосредственно на этапе выполнения программы.
Это дает удобную возможность симуляции сложного для написания кода с использованием совокупности заранее подготовленных значений. К примеру, необходимо создать программу, которая работает с таблицей значений синуса.

Пример 4.1. Содержимое текстового файла в формате GO DSP

```
1651 1 800 1 10
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
```

Вместо того, чтобы разрабатывать непростую подпрограмму вычисления значений этой тригонометрической функции, можно поместить рассчитанную с помощью привычных средств таблицу в текстовый файл и подгружать его содержимое на необходимом этапе выполнения программы в заданную область памяти ПЦОС динамически. Следует, однако, отметить, что данная возможность не предназначена для реализации связи в реальном режиме времени.

Файловый ввод/вывод в программе базируется на использовании концепции *пробных точек*. Они позволяют пользователю ввести или вывести совокупность значений, извлечь содержимое области памяти, начиная с заданной *пробной точки*. Пробную точку можно установить в любой строке программы. Когда одна них достигается при исполнении кода, связанный с пробной точкой объект (им может быть файл, график или окно памяти) обновляется и выполнение программы продолжается дальше. Т.е. располагая их в определенных местах программы, пользователь получает возможность использовать функциональность файлового ввода/вывода.

Каждый файл может быть связан или с вводом информации в ПЦОС, или с выводом из него. При этом в каждой пробной точке поток данных может быть либо считан, либо записан в соответствующий связанный файл.

При выборе подпункта *File I/O…* появляется диалоговое окно, изображенное на рис. 4.7.

Прежде чем заполнять поля диалога информацией о конкретном файле, необходимо создать в тексте программы пробную точку, с которой он будет связан, и оставить ее неопределенной *(unconnected)*. В принципе сам диалог предоставляет возможность создания пробной точки, однако гораздо удобнее и нагляднее установить ее в тексте программы заранее.

Примечание. *Для установки пробной точки позиционируйте курсор на нужной строке программы, щелкните правой кнопкой мыши и выберите в появившемся контекстном меню Toggle Probe Point.*

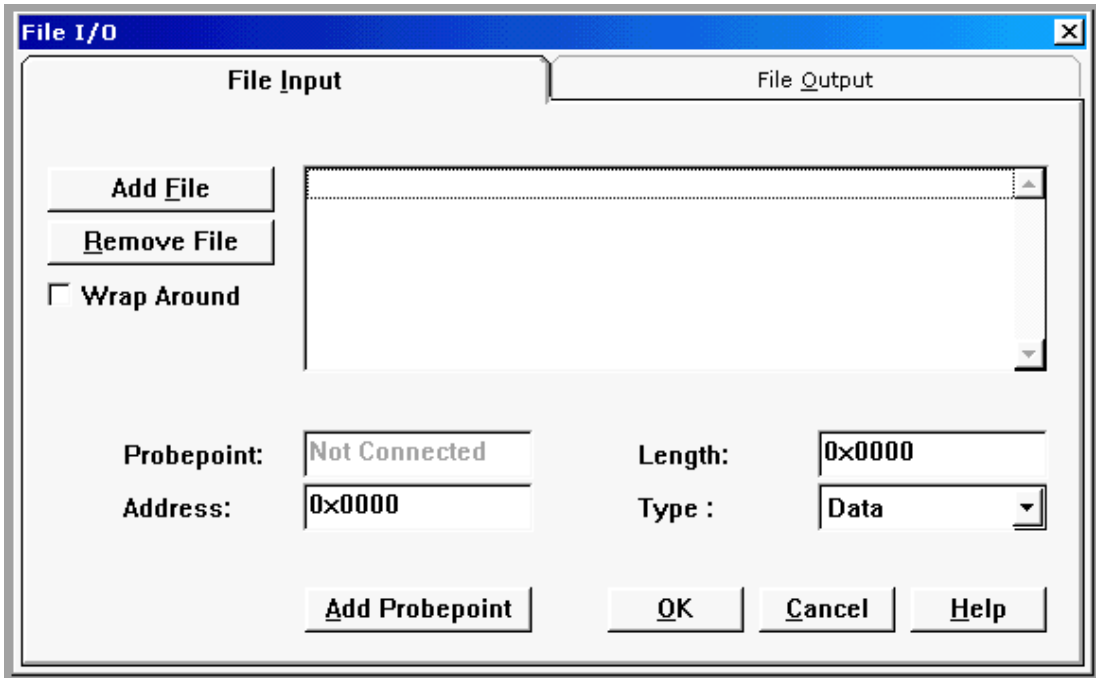


Рис. 4.7. Диалог File I/O

Диалог, изображенный на рис. 4.7., имеет две вкладки, первая из которых определяет параметры связанных файлов для ввода, а вторая – для вывода.

Для добавления нового связанного файла необходимо:

- воспользоваться кнопкой *Add File* (после выбора соответствующий файл помещается в расположенном справа окне списка, которое, в общем случае, содержит несколько вхождений);

Примечание: *Выбираемый файл может быть COFF-объектом или файлом формата GO DSP.*

- установить соответствие выбранного объекта и существующей пробной точки (кнопка *Add Probepoint*);

- определить тип целевой памяти процессора – программная память или память данных (раскрывающийся список *Type:*);
- установить стартовый адрес области памяти, из которой (в которую) необходимо считать (записать) данные в (из) файла (поле *Address:*) и количество передаваемых элементов данных (поле для ввода *Length:*).

После подтверждения и проверки на допустимость установленные соответствия могут использоваться в программе.

4.3.4. Пункт меню Edit

Команды этого меню (рис. 4.8.) позволяют редактировать содержимое памяти и регистров ПЦОС.

Для заполнения области памяти определите тип последней, ее начальный адрес и длину в полях *Page*, *Address* и *Length* соответственно. Также необходимо определить шаблон для заполнения в поле для ввода *Fill Pattern*.

Рис. 4.8. Меню Edit

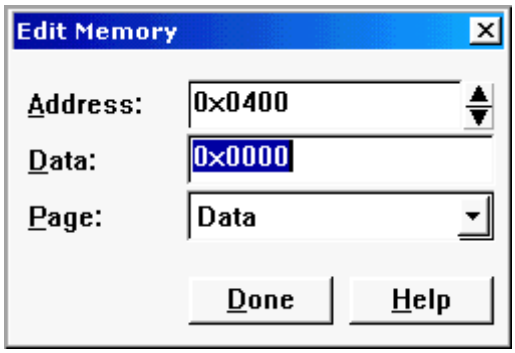


Рис. 4.9. Диалоговое окно изменения содержимого ячейки памяти

Команда *Edit Memory...* позволяет редактировать содержимое указанной ячейки памяти. При выборе соответствующего пункта меню появляется диалоговое окно, изображенное на рис. 4.9.

Задайте тип модифицируемой памяти в раскрывающемся списке *Page* и определите адрес ячейки и ее новое содержание в полях *Address* и *Date* соответственно.

Команда *Copy Memory*... позволяет копировать содержимое блоков памяти. При выборе соответствующего пункта меню появляется диалоговое окно, изображенное на рис. 4.10.

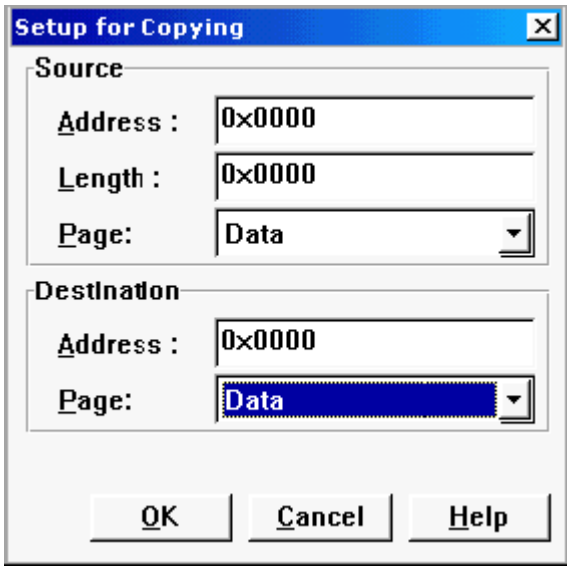


Рис. 4.10. Диалог копирования области памяти

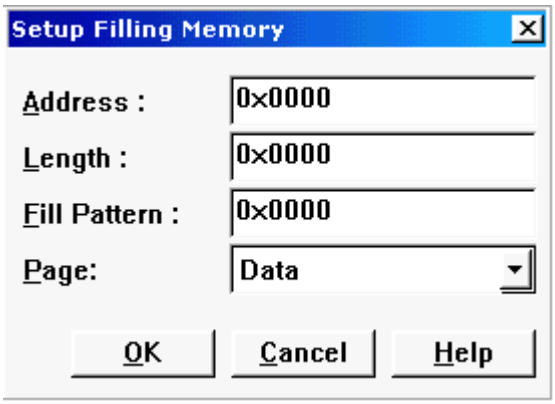


Рис. 4.11. Диалог заполнения области памяти

Для копирования информации задайте тип копируемого блока памяти, его начальный адрес и длину в группе полей ввода *Source* (поля *Page*, *Length* и *Address* соответственно) и определите тип блока памяти назначения и его начальный адрес в полях *Address* и *Page* группы *Destination*.

Команда *Fill Memory*... позволяет осуществить заполнение заданной области памяти в соответствии с заданным шаблоном. При выборе пункта меню появляется диалоговое окно, изображенное на рис. 4.11.

Команда *Edit Registers*... позволяет редактировать содержимое регистров процессора. При выборе соответствующего пункта меню появляется диалоговое окно, изображенное на рис. 4.12.

Для модификации содержимого аккумулятора или регистров ПЦОС задайте желаемый объект и его новое содержимое в раскрывающемся списке *Register* и поле для ввода *Value* соответственно.

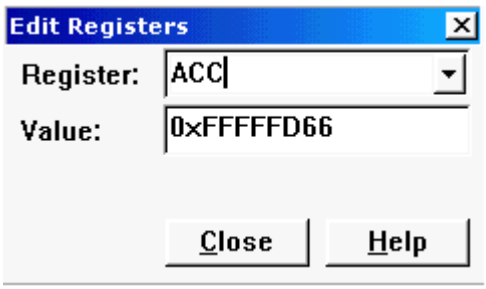


Рис. 4.12. Диалог редактирования регистров

4.3.5. Пункт меню View

Команды этого пункта меню (рис. 4.13) позволяют реализовать просмотр содержимого памяти и регистров процессора, а также визуализировать результаты работы программы.

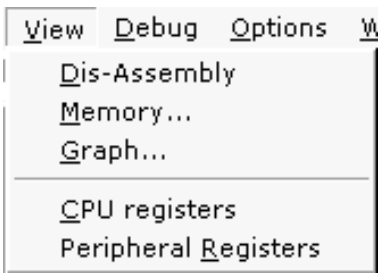


Рис. 4.13. Пункт меню View

Команда *Dis-Assembly* выводит на экран окно *Disassembly* (рис. 4.14) содержащее дизассемблированные команды наряду с символической и полупервичной информацией, необходимой для эффективной отладки.

Это окно возникает в результате дизассемблирования объектного кода программы начиная со строки, адрес которой определяется текущим

значением программного счетчика персонального компьютера. На любом этапе выполнения программы такая строка выделяется желтым цветом.

Для изменения стартового адреса окна необходимо нажать находящуюся в панели инструментов кнопку *Options* и в появившемся диалоговом окне определить новое значение или корректный программный символ (например, метку).

Кроме наглядного просмотра дизассемблированного текста программы, окно Diasassembly позволяет установить в этом тексте точку останова двойным щелчком левой кнопкой мыши на нужной строке программы. Такая строка впоследствии будет выделена пурпурным цветом.

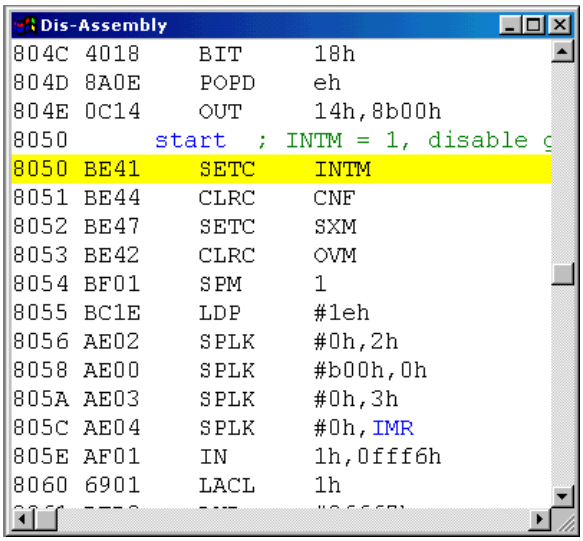


Рис. 4.14. Окно Disassembly

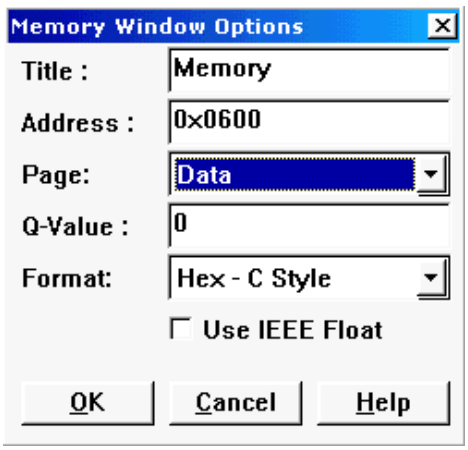


Рис. 4.15. Окно настройки режима просмотра памяти

Команда *Memory...* позволяет осуществить просмотр и редактирование памяти ПЦОС. Ее выполнение приводит к появлению на экране диалога, изображенного на рис. 4.15.

В нем необходимо уточнить тип памяти для просмотра (Program, Data или I/O), стартовый адрес для вывода информации и формат вывода данных (шестнадцатеричный, шестнадцатеричный в стиле C (с префиксом 0x), знаковое целое, беззнаковое целое, с плавающей точкой и т.д.) в полях *Page*, *Address* и *Format* соответственно. Кроме того, при использовании целочисленного представления содержимого ячеек памяти, в поле *Q-Value* может задаваться добротность. Для идентификации каждого из открытых окон используется поле *Title*.

После выполнения необходимых настроек на экране появляется окно с содержимым памяти выбранного типа (см. рис. 4.16).

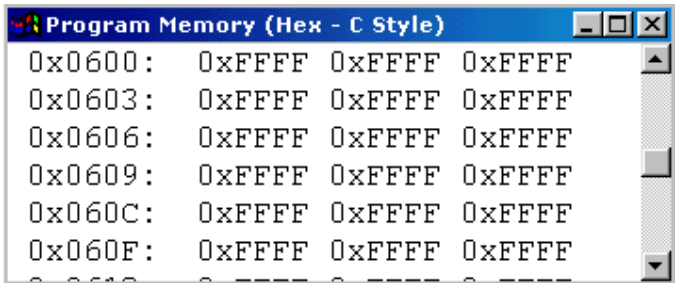


Рис. 4.16. Окно просмотра содержимого программной памяти процессора

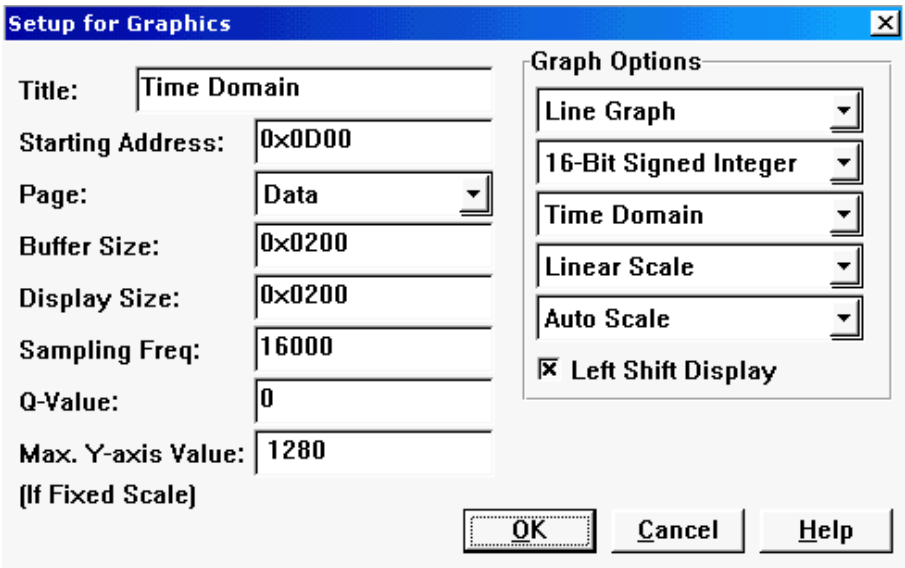


Рис. 4.17. Диалоговое окно настройки параметров отображения графики

При этом содержимое ячейки памяти по любому адресу может быть изменено двойным щелчком левой кнопки мыши по соответствующей строке окна. Это вызывает появление уже рассмотренного выше диалогового окна “Редактирование памяти”.

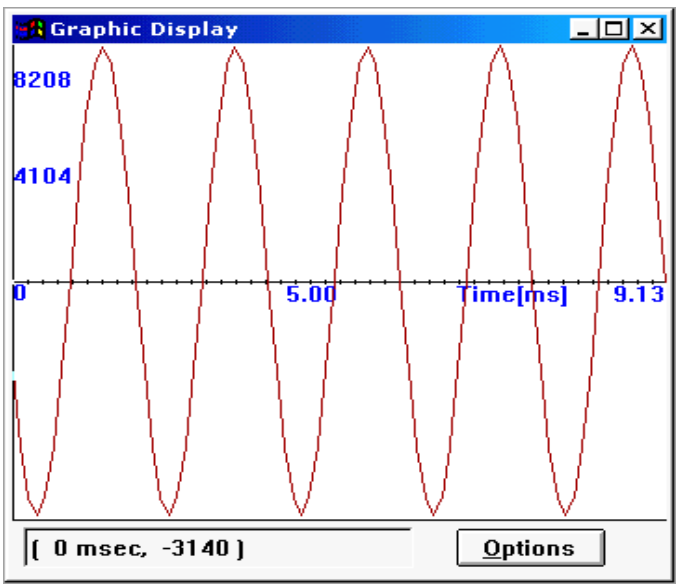
Для визуализации результатов работы программы или исходных данных используется команда *Graph*…. На рис. 4.17 приведено диалоговое окно настройки параметров графического отображения данных, появляющееся при выполнении соответствующей команды.

В левой части диалогового окна сверху-вниз задаются следующие параметры:

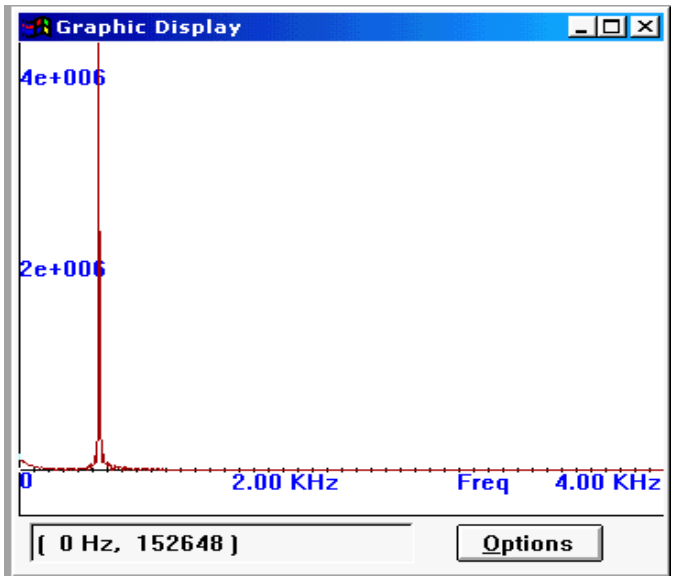
- *Title* – уникальный заголовок каждого графического окна (для удобства идентификации каждого из них)
- *Starting Address* – стартовый адрес блока памяти, служащего источником визуализируемых данных;
- *Page* – тип адресуемой памяти;
- *Buffer Size* – размер буфера, используемого для хранения графики перед выводом на экран;
- *Display Size* – определяет ту часть буфера, которую необходимо вывести на экран;
- *Sampling Freq* – частота дискретизации, используемая при аналого-цифровом преобразовании;
- *Q-Value* – добротность (ненулевое значение добротности вводится, чтобы показать, что целочисленные данные должны восприниматься как данные с фиксированной точкой);
- *Max. Y-axis Value* – используется для ограничения выводимой информации по оси Y, если выбран режим “фиксированная шкала”;
- В правой части сверху-вниз расположены следующие опции графической настройки:
- *Line Graph\Bar Graph* – если выбран первый вариант, то при построении графика для соединения данных используются линии, в противном случае из каждой отображаемой точки проводится перпендикуляр на ось времени (частоты);
- *Integer\Float* – при подготовке к выводу графики визуализируемые данные могут быть интерпретированы как целочисленные со знаком, целочисленные без знака, числа с фиксированной или плавающей точкой;
- *Time Domain\Frequency Domain* - если выбран первый вариант, строится график во временной области, при этом данные в буфере не подвергаются предварительной математической обработке; в противном случае данные в буфере подвернаются прямому преобразованию Фурье с количеством выборок, определяемым размером буфера, и выводятся на визуализацию в частотной области;

- *Linear Scale \Log Scale* – линейный или логарифмический режим построения;
- *Auto Scale \Fixed Scale* – автоматическое масштабирование пределов по оси Y или “фиксированный” режим;
- *Left Shift Display* – если эта опция не выбрана, то целевое содержимое памяти ПЦОС просто копируется в видеобуфер, в противном случае перенос происходит со сдвигом влево.

После обработки всех сделанных настроек программа выдает на экран окно, содержащее графическую информацию (рис. 4.18 а и б).



а



б

Рис. 4.18. Визуализация данных
а – во временной области
б - в частотной области

Как можно заметить из рисунков, каждое из окон имеет кнопку *Options*, с помощью которой после очередной перерисовки может быть снова вызвано диалоговое окно настройки графики, рассмотренное выше, и любые его параметры могут быть изменены.

С помощью подпункта *CPU-Registers* можно осуществить просмотр и изменение содержимого регистров общего назначения. Соответствующее окно приведено на рис. 4.19. После двойного щелчка левой кнопкой мыши по выбранному регистру открывается рассмотренное ранее окно редактирования содержимого регистра.

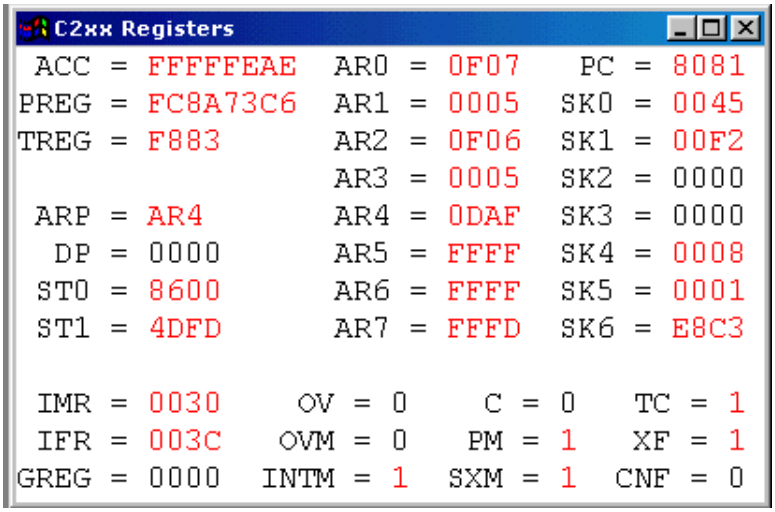


Рис. 4.19. Окно просмотра состояния регистров общего назначения

Для просмотра и изменения содержимого периферийных регистров процессора используется команда *Peripherial Registers* (см. рис. 4.20).

Используемый здесь принцип редактирования аналогичен таковому для регистров общего назначения.

Рис. 4.20. Окно просмотра содержимого периферийных регистров ПЦОС

4.3.6. Пункт меню Debug

Команды этого пункта меню (рис. 4.21) позволяют эффективно отлаживать пользовательскую программу. При выполнении команды *Breakpoints*... на экране появляется диалоговое окно (рис. 4.22), которое позволяет редактировать список точек останова при отладке программы. Все имеющиеся на данный момент точки останова перечислены в окне списка *Breakpoints* и могут быть в любой момент удалены или модифицированы с помощью расположенных справа кнопок редактирования.

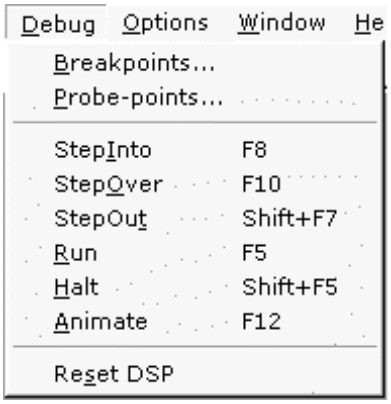


Рис. 4.21. Пункт меню Debug

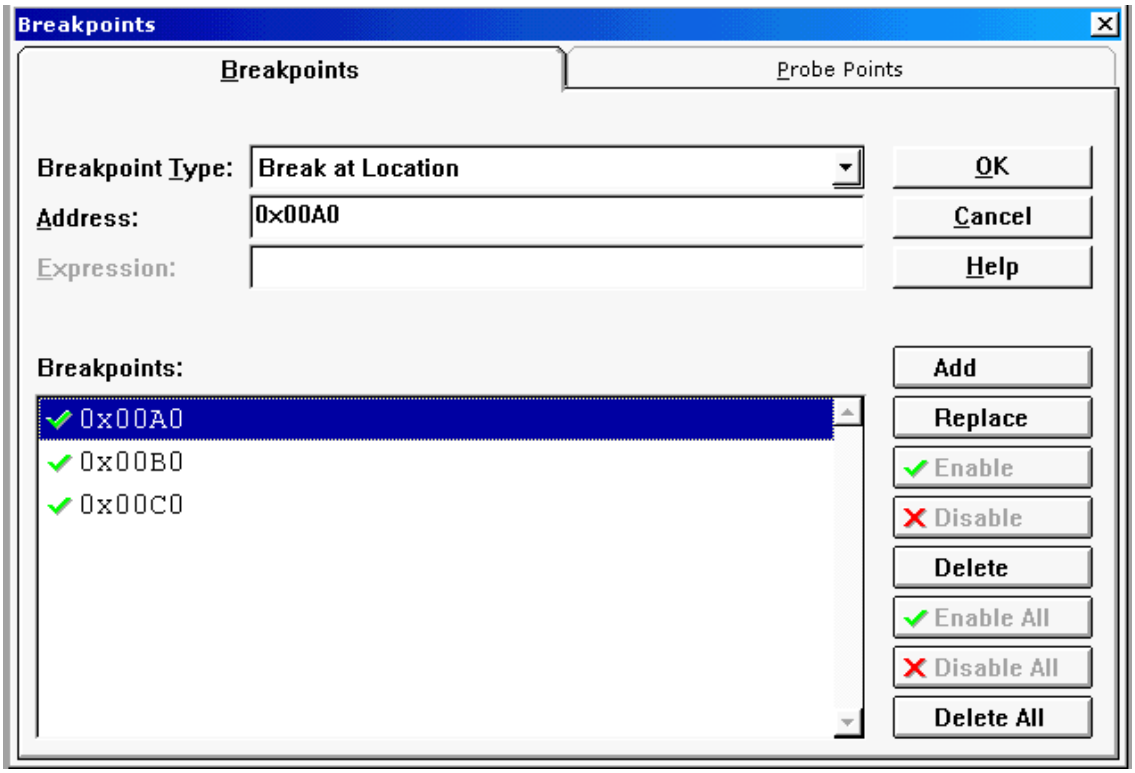


Рис. 4.22. Диалоговое окно управления точками останова программы

Для добавления новой точки останова вводится ее тип в поле *Breakpoint Type*.

Вы можете выбрать останов по достижении заданного адреса (Break at Location), для этого в поле *Address* введите адрес точки останова, либо останов по достижении заданного символа (*Break at Symbol*), пометив в поле *Expression* существующий программный символ (например, метку).

Концепция пробных точек, а также методы их создания и использования в программе были рассмотрены в п. 4.3.3 (назначение подпункта меню *File I/O…* меню *File*). Диалоговое окно *Probe Points…* (рис. 4.23) предоставляет альтернативную возможность создания пробных точек.

В нижней части диалогового окна располагается список всех, имеющихся на данный момент в программе, пробных точек. Удаление и модификация каждой из точек этого списка осуществляется с помощью кнопок редактирования, расположенных справа от окна списка.

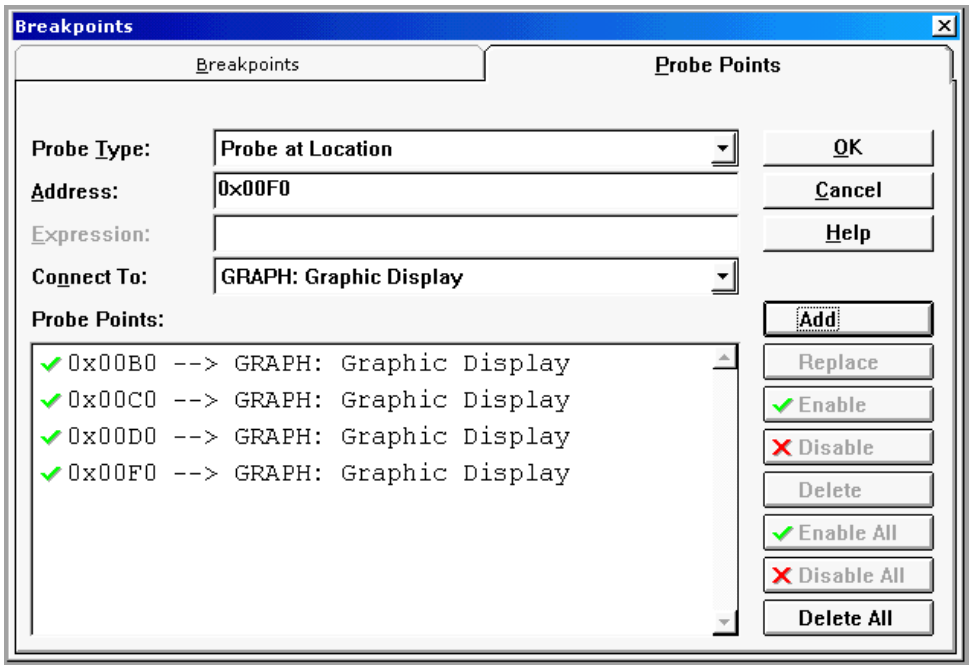


Рис. 4.23. Диалоговое окно управления пробными точками программы

Для добавления новой точки определите ее тип с помощью раскрывающегося списка *Probe Type* — пробная точка может определяться по адресу программной памяти (*Probe at Location*) или по программному символу (*Probe at Symbol*). После этого информация об адресе или символе должна быть внесена в поля *Address* или *Expression* соответственно.

Каждая пробная точка в программе должна быть связана с графическим окном или файлом для ввода/вывода. Для установления связи

между точкой и объектом выберите последний из списка имеющихся объектов *Connect To* и нажмите кнопку *Add*. Новая пробная точка будет добавлена в список существующих с указанием связанного с ней объекта. Если выбран не графический или файловый объект, то статус пробной точки будет считается неопределенным (*No Connection*).

Далее в меню *Debug* следуют семь команд, которые активно используются на этапе отладки программы:

Run – запуск программы на исполнение;

Halt – останов программы;

Animate – режим прогона программы по точкам останова. Эта команда вызывает выполнение команды *Run*, а затем ждет пока не будет достигнута одна из точек останова. Тогда происходит обновление всех имеющихся в данный момент окон (с выдачей информации о содержимом регистров и памяти, а также графической информации). После того, как все связанные с этим обновлением операции будут выполнены, программа возобновляет свою работу вплоть до достижения следующей точки останова и т.д. Причем пауза между окончанием всех связанных с обновлением окон операций и продолжением выполнения программы может варьироваться пользователем с помощью пункта меню *Options* → *Animate Speed;*

Reset ПЦОС - прерывает выполнение программы, выполняет процедуру инициализации всех регистров процессора и вызывает перезагрузку ядра DSK;

StepInto – вызывает пошаговое выполнение инструкций кода программы;

StepOver – выполняется до вызова подпрограммы, если нет необходимости осуществлять пошаговое выполнение последней. При этом код функции будет выполнен за один шаг, а пошаговое выполнение программы восстановится после достижения следующей за инструкцией вызова подпрограммы строки кода;

StepOut –в режиме пошагового выполнения программы внутри подпрограммы вызывает ее завершение в режиме реального времени. При этом выполнение кода прекращается по инструкции возврата из соответствующей подпрограммы.

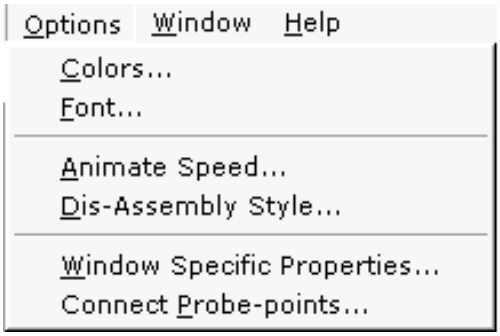
Примечание. *Назначение кнопок StepInto, StepOut, StepOver, Run, Halt, Animate и Reset панели инструментов аналогичнр назначению одноименных пунктов меню Debug.*

4.3.7. Пункт меню Options

Команды этого меню (рис. 4.24) позволяют настроить интерфейс программы, а также параметры отображения графической информации и выполнения программы в режиме *Animate***.** Пункт меню *Animate Speed…*,

как уже упоминалось, позволяет изменять длительность паузы при выполнении команды *Animate* между окончанием всех связанных с обновлением окон операций и продолжением выполнения программы.

Диалоговое окно настройки длительности пауз при выполнения программы приведено на рис. 4.25.



4.24. Меню Options

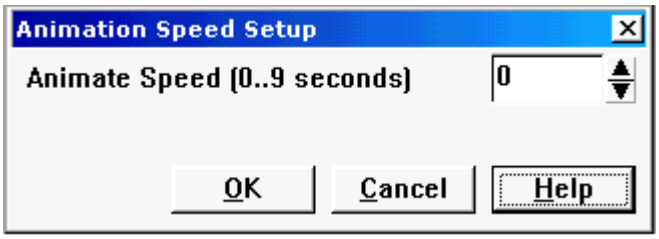


Рис. 4.25. Диалоговое окно настройки длительности пауз при выполнения программы

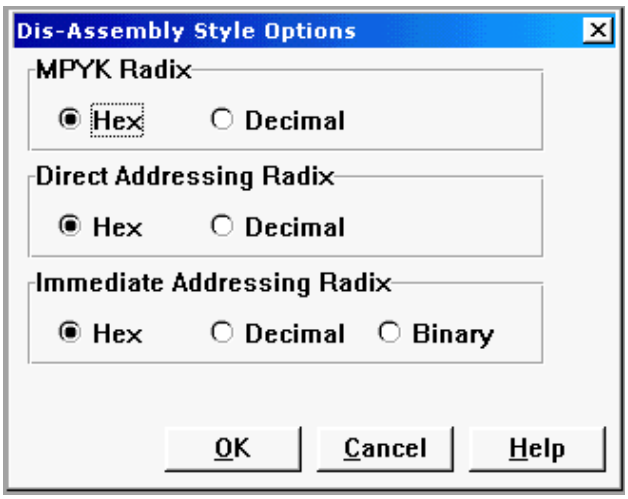


Рис. 4.26. Диалог настройки стиля окна дизассемблирования

Команда *Dis-Assembly Style…* позволяет менять стиль окна дизассемблирования (рис. 4.26).

С помощью данного диалогового окна пользователь может поменять систему счисления представленных в окне дизассемблирования данных, имеющих отношение к инструкции MPYK, прямой и обратной адресации.

Команды *Window Specific Properties…* и *Connect Probe-Point…* выводят на экран диалоговые окна настройки параметров отображения графики и управления пробными точками программы, которые были рассмотрены выше.

4.4. Обзор средства ассемблирования программ TASM

4.4.1. Вызов ассемблера из командной строки

TASM представляет собой табличный CROSS-ассемблер, предназначенный для работы в среде MS-DOS. Термин “табличный” подразумевает отсутствие фиксированной встроенной таблицы инструкций, определяющей однозначную трансляцию мнемонического представления пользовательской программы в шестнадцатеричные коды процессора.

Для корректной работы ассемблера требуется указание места расположения и имени требуемой трансляционной таблицы. Таким образом, TASM может использоваться для ассемблирования программ, написанных для достаточно широкого круга процессоров серий TMS320 – необходимо лишь каждый раз правильно указать соответствующую таблицу перевода. Это придает TASM дополнительную гибкость и универсальность.

Вызов TASM осуществляется командной строкой

tasm [-option_flag] src_file [obj_file [list_file [exp_file [sym_file]]]]

Указанные параметры командной строки предназначены:

Option_flag – определяет допустимые режимы ассемблирования и представляет собой комбинацию одной или нескольких перечисленных в табл. 4.1 опций. Обязательный параметр *src_file* определяет имя файла для ассемблирования.

Необязательные опции *obj_file* (имя объектного файла), *list_file* (имя листинг- файла), *exp_file* (имя файла с рыскрытой структурой макросов), и *sym_file* (имя файла с символьной информацией) при наличии в группе *option_flag* опций *-k*, *-l*, *-e* и *-s* определяют пользовательские имена целевых файлов.

4.4.2. Формат исходных данных

Для работы TASM в командной строке необходимо определить имя файла, содержащего данные для ассемблирования и имеющего расширение *asm*.

При этом каждая строка соответствующего файла должна удовлетворять такому шаблону:

[Метка] **[Операция]** **[Операнд]** **[Комментарии]**

Если первый символ строки ассемблируемой программы начинается с буквы, то все, что за ней следует вплоть до символа пробел или ‘:’ воспринимается TASM как метка. Ассемблер ассоциирует с каждой меткой соответствующей ей значение программного счетчика. Длина метки не может превышать 32 символа.

Поле *Операция* определяет действие, которое должно быть выполнено целевым процессором, и представляет собой мнемоническую форму записи одной из его инструкций (например, ADD). Оно может начинаться с любого по счету символа строки кроме первого и не чувствительно к используемому буквенному регистру.

Поле *Операнд* определяет используемые в инструкции данные и может содержать метки (например, LABEL1), числовые консанты (например, 100), операторы (+, -, *, >>, <<, <, >, >=, <=, =, !=, /, % и т.д.), символ текущего значения прораммного счетчика $, заключенные в апострофы символы ASCII, текстовые строки (они должны быть заключены в кавычки) и скобки.

Если в строке программы в качестве поля *Операнд* встретилось выражение, то ассемблер вычисляет его и использует полученное значение при формировании объектного кода программы.

Примечание. *При формировании текста пользовательской программы необходимо помнить, что понятие приоритета операций для TASM не определено и выражение 1 + 3•2 воспринимается им как (1+3)•2. Для того чтобы избежать ошибок, используйте в выражениях скобки для явного задания приоритета операций. Так, в упомянутой ситуации необходимо использовать выражение 1 +(2•3)*

Используемые в выражениях численные данные могут иметь различный формат, определяемый специально введенными префиксом или суффиксом. При этом суффикс H говорит о том, что используемые данные имеют шестнадцатеричный формат, O – восьмеричный, b – двоичный, d – десятичный (например, 12O, 3H). Вместо суффиксов могут быть использованы префиксы $, @ и % (данные в десятичном представлении в префиксе не нуждаются).

Таблица 4.1. Опции командной строки TASM

Опция	Описание	Пример использования
-table	Определяет используемую при ассемблировании трансляционную таблицу	При ассемблировании для TMS320F2xx и использовании таблицы TASM203.TAB **tasm -203 source.asm**
-ttable	Альтернативная форма предыдущей опции	При ассемблировании для TMS320F206 и использовании таблицы TASMF206.TAB **tasm -tf206 source.asm**
-a1 **-a2** **-a4** **-a8**	Диагностика ошибок при косвенной адресации; проверка на наличие неиспользуемых в аргументах данных; проверка на неуникальность меток; проверка на присутствия бинарных операторов в начале выражений.	**tasm -203 -a1 source.asm**
-c	Запись объектного файла в виде непрерывного блока	**tasm -203 -c source.asm**
-d	Определяет макрос или просто его метку	**tasm -203 –dExit source.asm** Такое определение метки Exit в командной строке используется при условном ассемблировании
-e	В нормальном режиме ассемблирования код макросов, определенных с помощью директивы #define, не раскрывается в генерируемом листинг-файле; данная опция блокирует это ограничение.	**tasm -203 -a1 -e source.asm**
-ffillbyte	Инициализирует входное пространство памяти процессора в соответствии с заданным шаблоном *fillbyte*	**tasm -203 -f00 source.asm**
-i	Если эта опция задана, то ассемблер не различает строчные и прописные буквы при трансляции меток.	**tasm -203 -i source.asm**
-k	Объектный код генерируется в формате DSK	**tasm -203 -k source.asm**
-l[al]	В листинг файл выводится информация о всех метках, встретившихся при ассемблировании; суффикс ***a*** указывает на то, что должна быть выведена информация обо всех метках программы, ***l*** означает вывод подробной информации по меткам	**tasm -203 -la source.asm**
-q	Не генерировать листинг-файл	**tasm -203 -q -k source.asm**
-s	Генерировать файл с символьной информацией (файл с расширением ****.sym***) по окончании ассемблирования	**tasm -203 -s -k source.asm**
-y	Генерировать информацию о времени ассемблирования	**tasm -203 –y -k source.asm**

Комментарием в тексте программы TASM считает любую последовательность символов, начинающуюся с символа ‘;’.

Пример 4.2. дает представление о возможных вариантах структуры строк пользовательской программы.

Пример 4.2. Фрагмент текста программы

```
START   .PS     8000h       ; Начало программы
        .ENTRY
        B       $ + 10O
        LAR     AR0, #((COEFFS >> 2) & FF)
```

Таблица 4.2. Базовые директивы TASM

Формат директивы	Описание	Пример
[метка] .BLOCK *выраж*	Вынуждает программный счетчик пропустить *выраж* число байт без инициализации содержимого соответствующих ячеек памяти	**Byte1:** **.block 1**
[*меткаl*] .BLOCK *выраж1 [,выраж …]*	Осуществляет инициализацию *N* ячеек памяти (начиная с ячейки, на которую указывает программный счетчик) значениями *выраж1, выраж2, …, выражN*	**Byte2:** **.byte 10, ‘C’**
#DEFINE *макро_метка* [(*спис_арг*)] [*макроопределение*]	Одна из самых функциональных директив ассемблера. Позволяет организовать параметризованную (со списком аргументов *спис_арг*) подстановку вхождений метки макро_метка в программе. Каждое вхождение TASM заменяет строкой *макроопределение*	**#DEFINE VAR1_LO (VAR1 & 255)**
#DEFCONT *выраж*	Позволяет организовать многострочное определение макросов, инициированное директивой #DEFINE	**#DEFINE ADD(xx,yy) clc** **#DEFCONT ** **lda xx** **#DEFCONT ** **adc yy** **#DEFCONT ** **sta xx**
.DS *выраж*	Определяет адрес *выраж*, начиная с которого будет осуществляться инициализация ячеек памяти данных пользовательскими значениями (например, с помощью директивы .WORD)	**.DS 0100b**
#IFDEF *метка* **блок инструкций 1** **[#ELSE блок инструкций 2]** **#ENDIF**	Одна из форм реализации условного ассемблирования. В данном случае, если значение ***метка*** определено в программе, ассемблируется первый блок инструкций, нет – второй.	**#IFDEF PROC1** **zac** **#ENDIF**

Окончание таблицы 4.2. Базовые директивы TASM

Формат директивы	Описание	Пример
#IFNDEF *метка* блок инструкций 1 [#ELSE блок инструкций 2] #ENDIF	В отличии от предыдущего случая первый блок ассемблируется, если *метка* не определена	**#IFNDEF PROC2 zac #ENDIF**
#IF *выраж* блок инструкций 1 [#ELSE блок нструкций 2] #ENDIF	Первый блок ассемблируется, если значение *выраж* отлично от нуля	**#IF ($ >= 1000h) zac #ENDIF**
.END	Присутствует в каждой программе и говорит о необходимости прекращения генерации объектного кода при ассемблировании. Строки пользовательского файла, следующие за этой директивой, ассемблером игнорируются	**.End**
.ENTRY	Обязательно присутствует в каждой программе, Используется для задания ячейки памяти в программном пространстве ПЦОС, с которой начнется выполнение пользовательской программы	**.ENTRY**
метка* .EQU *выраж (метка* .SET *выраж)	Используйте эту директиву для присвоения метке *метка* значения выражения *выраж*	**FALSE .equ 0 MASK .equ 0FFh**
#INCLUDE (.include) имя_файла	Считывает и ассемблирует содержимое указанного файла. Имя последнего должно быть заключено в кавычки	**#INCLUDE “macros.h”**
.LIST (.NOLIST)	Разрешает (запрещает) вывод информации в листинг-файл, начиная со строки программы, в которой она расположена.	**; инфорация об инструкциях заносится в листинг-файл .nolist ; здесь - уже нет**
[*метка*] .TEXT "*строка*"	Размещает содержимое указанной строки последовательно, посимвольно и в кодах ASCII в памяти процессора, начиная с ячейки памяти, на которую указывает отладчик	**message .text “string”**
.TITLE "*строка*"	Используется при печати как заголовок каждой новой распечатываемой страницы и одновременно служит названием программы	**.title “Counter”**
[метка] .WORD выраж1[, выраж2…]	Осуществляет инициализацию следующих определенных РС 2*N байт памяти значениями ***выраж1 ... выраж2***	**data_table: .word 12 .word 2*$ .word (2*$)+1 .word (2*$)&F00**

4.4.3. Директивы ассемблера

Определенные для TASM базовые директивы и их описание с примерами использования представлены в Табл. 4.2.

4.5. Последовательный порт ПЦОС TMS320C2xx

4.5.1. Обзор последовательного порта

В число периферийный устройств ПЦОС TMS320F206 входит синхронный последовательный порт (рис. 4.26). Он позволяет осуществить непосредственную связь с последовательными устройствами, такими как кодеки и последовательные АЦП.

Рассматриваемый порт имеет два встроенных четырехуровневых FIFO-буфера с возможностью генерации прерываний и передачи данных в широком диапазоне скоростей в непрерывном или дискретном режимах. В режиме внутренней синхронизации максимальная скорость передачи данных определяется половиной тактовой частоты процессора и составляет 10 кбод.

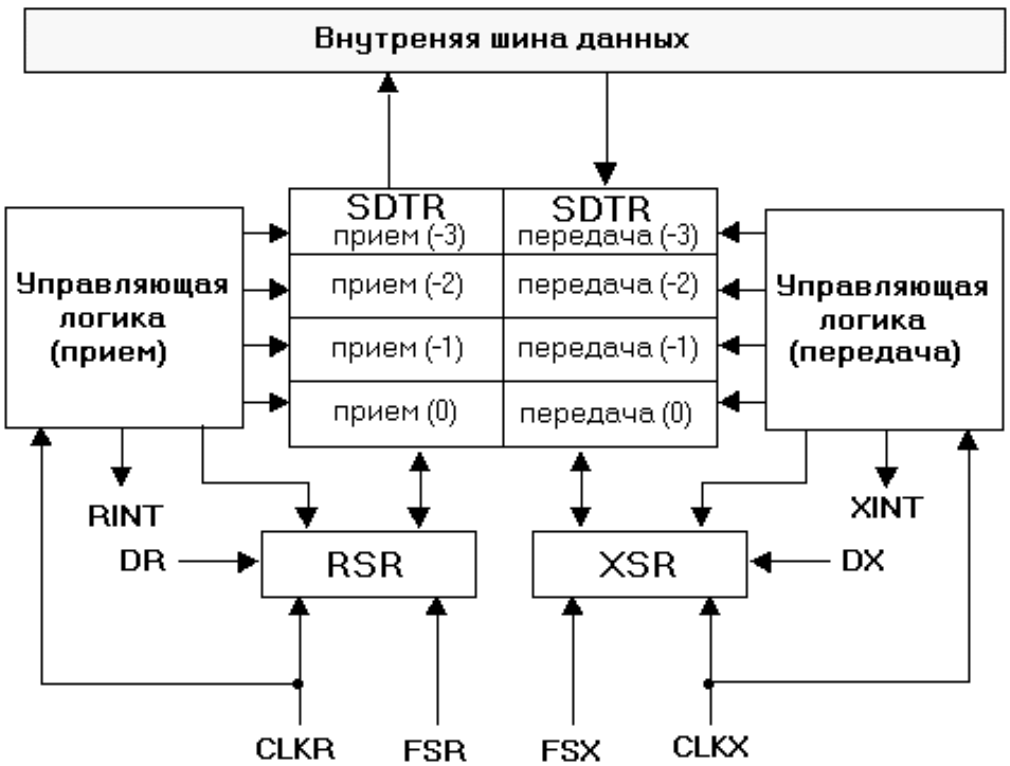


Рис. 4.26. Структурная схема последовательного порта

4.5.2. Сигналы последовательного порта

Приведенные на рис. 4.26 сигналы имеют следующее назначение:

- *CLKR (CLKX)* – тактовый сигнал приема (передачи). Используется для синхронизации процесса приема (передачи) информации в последовательный порт и, в зависимости от настройки, может генерироваться самим процессором или происходить от внешнего источника;
- *FSR (FSX)* – сигнал кадровой синхронизации приема (передачи) используется для синхронизации приема (передачи) “кадров” информации и может иметь как внутреннее, так и внешнее происхождение (зависит от установок);
- *DR (DX)* – принимаемые (передаваемые) последовательные данные. При этом вывод приема (передачи) последовательных данных *DR (DX)* ПЦОС для обеспечения корректного обмена информации подключается к соответствующему выводу последовательной передачи *DX* (последовательного приема *DR*) другого устройства.
- *RINT (XINT)* – сигнал маскируемого прерывания приема (передачи) генерируется буфером приема (передачи) последовательного порта при условии.

4.5.3. Регистры последовательного порта

Единственным программно-доступным регистром буферов передачи и приема последовательного порта является отображаемый в пространство ввода/вывода по адресу FFF0h *синхронный регистр приема и передачи SDTR*. С помощью этого регистра осуществляется запись (инструкция *OUT*) и чтение (инструкция *IN*) информации в последовательный порт.

Регистры *RSR* и *XSR* осуществляют контроль передачи информации между FIFO-буферами и выводами порта и не доступны из программы пользователя.

RSR – синхронный регистр приема последовательного порта. Каждое слово данных, принимаемое по линии сигнала *DR*, вдвигается “старшим битом вперед” в этот регистр по одному биту в течение одного периода синхронизирующего сигнала *CLKR*. После того, как слово данных окажется полностью записанным в рассматриваемый регистр, его содержимое в соответствии с рис. 4.26 переписывается в нижнюю ячейку (прием (0)) принимающего FIFO-буфера.

XSR – синхронный передающий регистр последовательного порта. Каждое слово данных, в соответствии с рис. 4.26 передается из ячейки нижнего уровня передающего FIFO-буфера (передача (0)) в этот регистр. После завершения процесса переноса данных в регистр, *XSR* выдвигает данные “старшим битом вперед” по выводу последовательной передачи *TX*.

4.5.4. Прерывания

Синхронный последовательный порт поддерживает два аппаратных прерывания, осведомляющих процессор о необходимости обслуживания приемо-передающих FIFO-буферов:

- прерывание передачи *(XINT)* вызывает переход по адресу 000Ah программной памяти, если выполнено условие срабатывания триггера прерывания передачи. Это условие задается битами *FT0* и *FT1* управляющего регистра *SSPCR* последовательного порта (подробно рассмотрен в п. 4.5.5). Уровень приоритета данного прерывания равен 8;
- прерывание приема *(RINT)* вызывает переход по адресу 0008h в программной памяти, если выполнено условие срабатывания триггера прерывания приема. Это условие задается битами *FR0* и *FR1* управляющего регистра *SSPCR* последовательного порта. Уровень приоритета данного прерывания – 7.

Для маскирования прерываний *XINT* и *RINT* используются биты №4 и 3 *регистра масок прерываний IMR* соответственно. Определение статуса обработки этих прерываний может быть осуществлено с помощью установки/сброса битов №4 и №3 *регистра флагов прерываний IFR*. При этом бит №4 отвечает за прерывание передачи, а №3 – за прерывание приема. Подробнее об обработке прерываний смотрите п. 4.6.

Примечание. *Для предотвращения повторных прерываний приема и передачи от SSP, необходимо сбросить соответствующие биты регистра масок прерываний IFR в соответствующей подпрограмме обработки прерываний перед возвратом в основную программу.*

4.5.5. Регистр SSPCR

Регистр управления последовательным портом *SSPCR* представляет собой отображаемый в пространство ввода/вывода по адресу FFF1h регистр, выполняющий следующие функции:

установки режимов работы последовательного порта;

индикации состояния процесса обмена данными;

установки условий срабатывания триггеров приема и передачи;

индикации возникающих в ходе обмена ошибок;

сброса порта.

Подробнее структура этого управляющего регистра представлена в табл. 4.3.

Таблица 4.3. Назначение битов управляющего регистра SSPCR

№	Назва-ние	Вид доступа	После сброса	Назначение
15 14	FREE, SOFT	Чтение/ Запись	0	Представляют собой специальные эмулирующие биты, которые определяют состояние процесса передачи данных последовательного порта при достижении точки останова в отладчике верхнего уровня: если FREE = 1, то состояние бита SOFT не имеет значения и процесс передачи последовательных данных продолжается; если FREE = 0, то при SOFT = 0 процесс обмена прекращается немедленно. При SOFT = 1 прекращение обмена инициируется лишь по окончании передачи последнего слова данных
13	TCOMP	Чтение	0	Бит сбрасывается, если все слова из передающего FIFO-буфера переданы; если передающий буфер не пуст, то данный бит установлен
12	RFNE	Чтение	0	Бит сбрасывается, если приемный FIFO-буфер пуст; если содержит данные, то данный бит установлен
11 10	FT1, FT0	Чтение/ Запись	0	Определяет условие срабатывания триггера прерывания передачи в зависимости от содержимого передающего FIFO-буфера. Если условие выполняется, то происходит генерация прерывания XINT и данные могут быть переданы в соответствующий FIFO-буфер с использованием инструкции OUT. Если оба бита сброшены, то прерывание генерируется при полном передающем буфере. При сброшенном (установленном) FR1 и установленном (сброшенном) FT0 генерация происходит при заполнении передающего буфера на 75% (50%). Если оба управляющих бита установлены, то прерывание генерируется при 25%-ном заполнении передающего буфера
9 8	FR1, FR0	Чтение/ Запись	0	Определяет условие срабатывания триггера прерывания приема в зависимости от содержимого принимающего FIFO-буфера. Если это условие выполняется, то происходит генерация прерывания RINT и данные могут быть считаны из соответствующего FIFO-буфера с использованием инструкции IN. Если оба управляющих бита сброшены, то прерывание генерируется при непустом приемном буфере. При сброшенном (установленном) FR1 и установленном (сброшенном) FR0 генерация происходит при заполнении приемного буфера по меньшей мере на 50% (75%). Если оба управляющих бита установлены, то прерывание генерируется при полном буфере
7	OVF	Чтение	0	Устанавливается всякий раз, когда принимающий FIFO-буфер полон и в регистр RSR по линии последовательного приема пришло очередное слово данных. Содержимое буфера не переписывается. При первой операции чтения из порта этот бит сбрасывается
6	IN0	Чтение	0	Позволяет использовать вывод CLKR как вход. IN0 отражает текущий уровень на этом выводе

Окончание таблицы 4.3. Назначение битов управляющего регистра SSPCR

№	Назва-ние	Вид доступа	После сброса	Назначение
5	**XRST**	**Чтение/ Запись**	**0**	Сброс этого бита вызывает сброс передающей логики последовательного интерфейса. Для ее вывода из состояния сброса необходимо установить бит XRST
4	**RRST**	**Чтение/ Запись**	**0**	Сброс этого бита вызывает сброс принимающей логики последовательного интерфейса. Для ее вывода из состояния сброса необходимо установить бит RRST
3	**TXM**	**Чтение/ Запись**	**0**	Определяет источник импульсов кадровой синхронизации. Если этот бит сброшен, то выбирается внешний источник и FSX конфигурируется как вход для приема соответствующих импульсов. В противном случае кадровую синхронизацию передачи осуществляет микропроцессор, вывод FSX конфигурируется как выход, который посылает импульсы кадровой синхронизации в начале каждой передачи. Импульсы синхронизируются по отношению к сигналу CLKX. При работе последовательного порта в режиме цифровой “закольцовки” данный бит следует установить
2	**MCM**	**Чтение/ Запись**	**0**	Определяет источник импульсов синхронизации передачи отдельных бит слов данных для последовательного порта. Если этот бит сброшен, то выбран внешний источник синхронизации и вывод CLKX конфигурируется как вход для приема соответствующего сигнала. В противном случае источником синхронизирующего сигнала является сам ПЦОС, и вывод CLKX конфигурируется как выход. Частота соответствующего сигнала определяется при этом как половина частоты сигнала CLKOUT1
1	**FSM**	**Чтение/ Запись**	**0**	Задает режим передачи последовательных данных. При установленном бите выбран дискретный режим передачи и импульс кадровой синхронизации необходим для последовательной передачи/приема каждого слова данных. В противном случае инициируется непрерывный режим передачи, при котором единственный импульс кадровой синхронизации необходим для передачи последовательного потока слов данных.
0	**DLB**	**Чтение/ Запись**	**0**	Включает/отключает режим цифровой “закольцовки”. При сброшенном бите упомянутый режим работы последовательного порта деактивирован и выводы DR, FSR и CLKR соединены с соответствующими выводами другого устройства. В противном случае порт работает в режиме цифровой “закольцовки”, при котором выводы DR и FSR внутренне соединены с выводами DX и FSX соответственно

4.5.6. Порядок работы с последовательным портом

Запись в последовательный порт осуществляется по следующей схеме:

1) В начале процесса последовательной передачи необходимо произвести инициализацию последовательного порта записью соответствующего управляющего слова в регистр *SSPCR*.
2) Процессор может записать до четырех слов данных в передающий FIFO-буфер через регистр *SDTR*.
3) Если регистр *XSR* пуст (завершена последовательная передача предыдущего слова данных), то первое из записанных в передающий FIFO-буфер слов данных копируется в *XSR*.
4) Регистр *XSR* побитно выдвигает данные (начиная со старшего разряда) через вывод *DX*.
5) После завершения процесса последовательной передачи регистр *XSR* дает сигнал передающему буферу и процесс продолжается по одному из двух путей:

- если FIFO-буфер пуст, процесс повторяется с шага 2;
- если выполнены условия, определяемые битами *FT0* и *FT1* регистра *SSPCR*, генерируется прерывание *XINT* с целью запроса дальнейших данных для последовательной передачи и процесс останавливается.

Чтение из последовательного порта осуществляется по следующей схеме:

1) В начале процесса последовательного приема необходимо осуществить инициализацию последовательного порта записью соответствующего управляющего слова в регистр *SSPCR*;
2) Последовательные данные по выводу *DR* побитно вдвигаются (начиная со старшего разряда) в регистр *RSR*;
3) По завершении сдвигового процесса *RSR* копирует данные в принимающий FIFO-буфер;
4) Далее процесс продолжается по одному из двух путей:

- если приемный FIFO-буфер не полон, процесс повторяется с шага 1;
- в противном случае, если выполнены условия, определяемые битами *FR0* и *FR1* регистра *SSPCR*, буфер генерирует прерывание *RINT*, запрашивая обработку;

5) При получении запроса на обработку процессор может считать данные из приемного FIFO-буфера через регистр *SDTR*.

4.6. Прерывания ПЦОС TMS320C2xx

4.6.1. Обзор системы прерываний

Прерываниями называются имеющие аппаратное или программное происхождение сигналы, которые заставляют процессор прервать последовательное выполнение инструкций пользовательской программы и

перейти к выполнению специальной подпрограммы, называемой *подпрограммой обработки прерывания (Interrupt Service Routine)*.

Как вытекает из определения, прерывания разделяются на *аппаратные* и *программные*. Изучаемый ПЦОС поддерживает прерывания обоих типов.

Говоря об аппаратных прерываниях, их следует подразделить на *внешние* и *внутренние*. Внешние прерывания вызываются сигналами, подаваемыми на внешние выводы микропроцессора, а внутренние – сигналами от внутренней периферии (асинхронного порта, таймера и т.п.).

По возможности блокирования из пользовательской программы прерывания подразделяются на *маскируемые* (могут быть запрещены программно) и *немаскируемые* (не могут быть запрещены программно).

ПЦОС TMS320C2xx осуществляет обработку прерываний в три последовательных фазы:

прием сигнала запроса прерывания;

подтверждение запроса прерывания;

выполнение подпрограммы обработки прерывания.

4.6.2. Описание прерываний TMS320C2xx

Адрес подпрограммы обработки прерывания называется *вектором прерывания*. Векторы прерываний изучаемого процессора храняться в программной памяти, начиная с нулевого адреса, и образуют *таблицу прерываний* (табл. 4.4).

Таблица 4.4. Прерывания ПЦОС TMS320C2xx

Адрес вектора прерывания	Название	Приоритет	Описание
0000h	**$\overline{\text{RS}}$**	**1**	Аппаратный сброс (немаскируемое прерывание)
0002h	**$\overline{\text{HOLD}}$ / $\overline{\text{INT1}}$**	**4**	Маскируемое прерывание №1
0004h	**$\overline{\text{INT2}}$ / $\overline{\text{INT3}}$**	**5**	Маскируемые прерывания №2 и №3
0006h	**TINT**	**6**	Маскируемое прерывание таймера
0008h	**RINT**	**7**	Маскируемое прерывание приема синхронного последовательного порта
000Ah	**XINT**	**8**	Маскируемое прерывание передачи синхронного последовательного порта
000Ch	**TXRXINT**	**9**	Маскируемое прерывание приема/передачи асинхронного последовательного порта
000Eh	**-**	**10**	Зарезервировано

Окончание таблицы 4.4. Прерывания ПЦОС TMS320C2xx

Адрес вектора прерывания	Название	Приоритет	Описание
0010h	**INT8**	**-**	Программное прерывание
0012h	**INT9**	**-**	Программное прерывание
0014h	**INT10**	**-**	Программное прерывание
0016h	**INT11**	**-**	Программное прерывание
0018h	**INT12**	**-**	Программное прерывание
001Ah	**INT13**	**-**	Программное прерывание
001Ch	**INT14**	**-**	Программное прерывание
001Eh	**INT15**	**-**	Программное прерывание
0020h	**INT16**	**-**	Программное прерывание
0022h	**TRAP**	**-**	Вектор инструкции TRAP
0024h	**$\overline{\text{NMI}}$**	**3**	Немаскируемое прерывание
0026h	**-**	**2**	Зарезервировано
0028h	**INT20**	**-**	Программное прерывание
002Ah	**INT21**	**-**	Программное прерывание
002Ch	**INT22**	**-**	Программное прерывание
002Eh	**INT23**	**-**	Программное прерывание
0030h	**INT24**	**-**	Программное прерывание
0032h	**INT25**	**-**	Программное прерывание
0034h	**INT26**	**-**	Программное прерывание
0036h	**INT27**	**-**	Программное прерывание
0038h	**INT28**	**-**	Программное прерывание
003Ah	**INT29**	**-**	Программное прерывание
003Ch	**INT30**	**-**	Программное прерывание
003Eh	**INT31**	**-**	Программное прерывание

4.6.3. Регистры и биты контроля обработки прерываний

Обработка прерываний в процессоре TMS320F206 осуществляется на основании содержимого следующих регистров:

флагов прерываний (IFR) — отображаемый в пространство памяти данных по адресу 0006h 16-битный регистр;

контроля прерываний (ICR) — отображаемый в пространство ввода/вывода по адресу FFECh 16-битный регистр;

масок прерываний (IMR) — отображаемый в пространство памяти данных по адресу 0004h 16-битный регистр.

Кроме того, при обработке прерываний большое значение имеет девятый бит статусного регистра *ST0*, именуемый *битом режима прерываний INTM*.

Установка бита INTM приводит к запрещению всех демаскированных прерываний, а его сброс – к разрешению последних. Данный бит автоматически устанавливается в “1” после того, как процессор переходит к выполнению процедуры обработки маскируемого прерывания, по завершении которой для обеспечения возможности обработки других маскируемых прерываний необходимо программно сбросить этот бит. Для программной установки бита *INTM* можно воспользоваться инструкцией *SETC INTM*, а для его сброса – *CLRC INTM*. Следует подчеркнуть, что управляющий бит *INTM* не оказывает влияние на обработку немаскируемых прерываний – они обрабатываются независимо от его состояния.

Регистр *IFR* (табл. 4.5) содержит флаги всех маскируемых прерываний. Если маскируемое прерывание достигает процессора, соответствующий флаг устанавливается в “1”. Это означает, что обработка данного прерывания отложена. Для того чтобы отменить запрос на маскируемое прерывание, необходимо сбросить соответствующий бит *IFR* записью в него единичного значения.

Таблице 4.5. Битовая структура регистра IFR

№	Назва-ние	Вид доступа	После сброса	Назначение
6 ÷15	**-**	**Чтение**	**0**	Установлен – прерывание TXRXINT отложено; сброшен – прерывание TXRXINT не приходило
5	**TXRX INT**	**Чтение/ Запись**	**0**	Установлен – прерывание XINT отложено; сброшен – прерывание XINT не приходило
4	**XINT**	**Чтение/ Запись**	**0**	Установлен – прерывание XINT отложено; сброшен – прерывание XINT не приходило
3	**RINT**	**Чтение/ Запись**	**0**	Установлен – прерывание RINT отложено; сброшен – прерывание RINT не приходило
2	**TINT**	**Чтение/ Запись**	**0**	Установлен – прерывание TINT отложено; сброшен – прерывание TINT не приходило
1	**INT2/ INT3**	**Чтение/ Запись**	**0**	Установлен – прерывания INT2/INT3 отложены; сброшен – прерывания INT2/INT3 не приходили
0	**HOLD/ INT1**	**Чтение/ Запись**	**0**	Установлен – прерывание INT1 отложено; сброшен – прерывание INT1 не приходило

Регистр *IMR* используется для маскирования внешних и внутренних аппаратных прерываний. Для маскирования прерывания необходимо установить соответствующий бит регистра *IMR*, а для демаскирования – сбросить. Маскированное прерывание не обрабатывается процессором.

Действие данного регистра не распространяется на немаскируемые прерывания ($\overline{\mathbf{NMI}}$ и $\overline{\mathbf{RS}}$).

Структура регистра масок прерываний приведена в табл. 4.6.

Как очевидно из структуры рассмотренных выше регистров прерывания *INT2* и *INT3* разделяют общие биты регистров *IFR* и *IMR*. Для определения текущего статуса обработки (“отложенности”) и маскирования каждого из этих прерываний по отдельности используется *регистр контроля прерываний ICR*.

В этом регистре также определяется чувствительность ПЦОС к сигналу прерывания *INT1*.

Таблице 4.6. Битовая структура регистра IMR

№	Назва-ние	Вид доступа	После сброса	Назначение
6 ÷15	**-**	**Чтение**	**0**	Установлен – прерывание TXRXINT не маскировано; сброшен – прерывание TXRXINT маскировано
5	**TXRXINT**	**Чтение/ Запись**	**0**	Установлен – прерывание XINT не маскировано; сброшен – прерывание XINT маскировано
4	**XINT**	**Чтение/ Запись**	**0**	Установлен – прерывание XINT не маскировано; сброшен – прерывание XINT маскировано
3	**RINT**	**Чтение/ Запись**	**0**	Установлен – прерывание RINT не маскировано; сброшен – прерывание RINT маскировано
2	**TINT**	**Чтение/ Запись**	**0**	Установлен – прерывание TINT не маскировано; сброшен – прерывание TINT маскировано
1	**INT2/ INT3**	**Чтение/ Запись**	**0**	Установлен – прерывания INT2/INT3 не маскированы; сброшен – прерывания INT2/INT3 маскированы
0	**HOLD/ INT1**	**Чтение/ Запись**	**0**	Установлен – прерывание INT1 не маскировано; сброшен – прерывание INT1 маскировано

4.6.4. Обработка маскируемых прерываний

Обработка маскируемых прерываний в ПЦОС TMS320F206 происходит в соответствии с изложенным ниже алгоритмом:

- поступает запрос прерывания на ПЦОС;
- устанавливается соответствующий бит регистра *IFR*;

- если бит *INTM* статусного регистра *ST0* установлен, осуществляется переход на шаг 10, в противном случае осуществляется переход на шаг 4;
- если соответствующее прерывание маскировано (сброшен бит регистра *IMR*) — на шаг 10, в противном случае осуществляется переход на шаг 5;
- ПЦОС подтверждает запрос прерывания;
- устанавливается бит *INTM*;
- программный счетчик сохраняется в стеке;
- осуществляется переход к подпрограмме обработки данного прерывания;
- инструкция возврата из подпрограммы обработки прерывания форсирует восстановление программного счетчика из стека;
- программа продолжает выполняться в обычном режиме.

5. МАЛОПОТРЕБЛЯЕМЫЕ ПЦОС СЕРИИ TMS320C5000

Рассмотрим ПЦОС серии TMS320C5000 на примере процессоров подсерии TMS320C54x, которые отличает комбинирование модифицированной гарвардской архитектуры с тремя внутренними шинами данных и одной шиной команд [2, 14]. Такая внутренняя организация ПЦОС обеспечивает высокую степень параллельности выполнения команд. Эту серию характеризует высокоспециализированная система инструкций, наличие на кристалле дополнительных периферийных устройств и увеличенный объем внутренней памяти. Все это обеспечивает известную гибкость и производительность систем, проектируемых на основе данных ПЦОС.

Три шины данных используются для чтения операндов и записи результата операции одновременно с выборкой инструкции в одном процессорном цикле. Общий объем адресуемой процессором памяти, составляющий 192 16-разрядных слов, разбит на 3 специализированных сегмента: команд, данных и ввода/вывода — каждый из которых может иметь размер до 64 Кслов. Внутри ПЦОС может быть расположена ROM объемом до 48 16-разрядных и до 10 Кслов двухвходовой RAM. В ПЦОС предусмотрена опция защиты данных во внутренней памяти от сканирования. При установке режима защиты ни одна из команд не сможет получить доступ к содержимому внутрикристальной памяти.

Для ускорения выполнения типовых операций сигнальной обработки помимо «стандартных» для ПЦОС блоков барабанного сдвига и адресной арифметики ПЦОС содержит ряд дополнительных функциональных модулей, повышающих его гибкость и производительность. Блок умножения с накоплением (*MAC*) выполняет над 17 битовыми операндами операции вида $S = S + a \cdot b$ за один процессорный такт. Подобные операции характерны для алгоритмов фильтрации, свертки, вычисления корреляционной функции.

Для быстрого вычисления значений функции $y = exp(x)$ микропроцессор содержит блок перекодировки значения аккумулятора *(EXP Encoder)*, интерпретируемое как значение аргумента функции. Данный блок вычисляет соответствующее значение за один такт.

Эффективную реализацию оператора Витерби обеспечивает блок *(CMPS Operation),* выполняющий за один цикл операцию сравнения-выбора с накоплением *(Add/Compare Selection).*

АЛУ ПЦОС способно выполнять арифметические или булевы операции над комплексными числами (используя два регистра-аккумулятора — *ACCA* и *ACCB*) или может функционировать как два 16-разрядных АЛУ, выполняющих одновременно две 16-разрядных операции. АЛУ и МАС могут выполнять операции в цикле одновременно.

Устройство барабанного сдвига осуществляет сдвиг данных на 0—31 разряд влево или 0—16 разрядов вправо за один такт, а также совместно с блоком вычисления экспоненциальной функции обеспечивает нормализацию содержимого аккумулятора за один такт. Дополнительные возможности сдвига позволяют процессору масштабировать данные, выделять разряды числа, предотвращать возникновение переполнения и потери значимости.

Все ПЦОС подсерии TMS320C54x имеют одинаковую структуру, однако отличаются друг от друга расположенной на кристалле периферией, соединенной с ЦПУ. В состав периферийных устройств входят (рис. 5.1) [2]:

программно-управляемый генератор тактов ожидания;
программный переключатель банков памяти;
параллельные порты ввода/вывода;
аппаратный таймер и генератор тактовых импульсов.

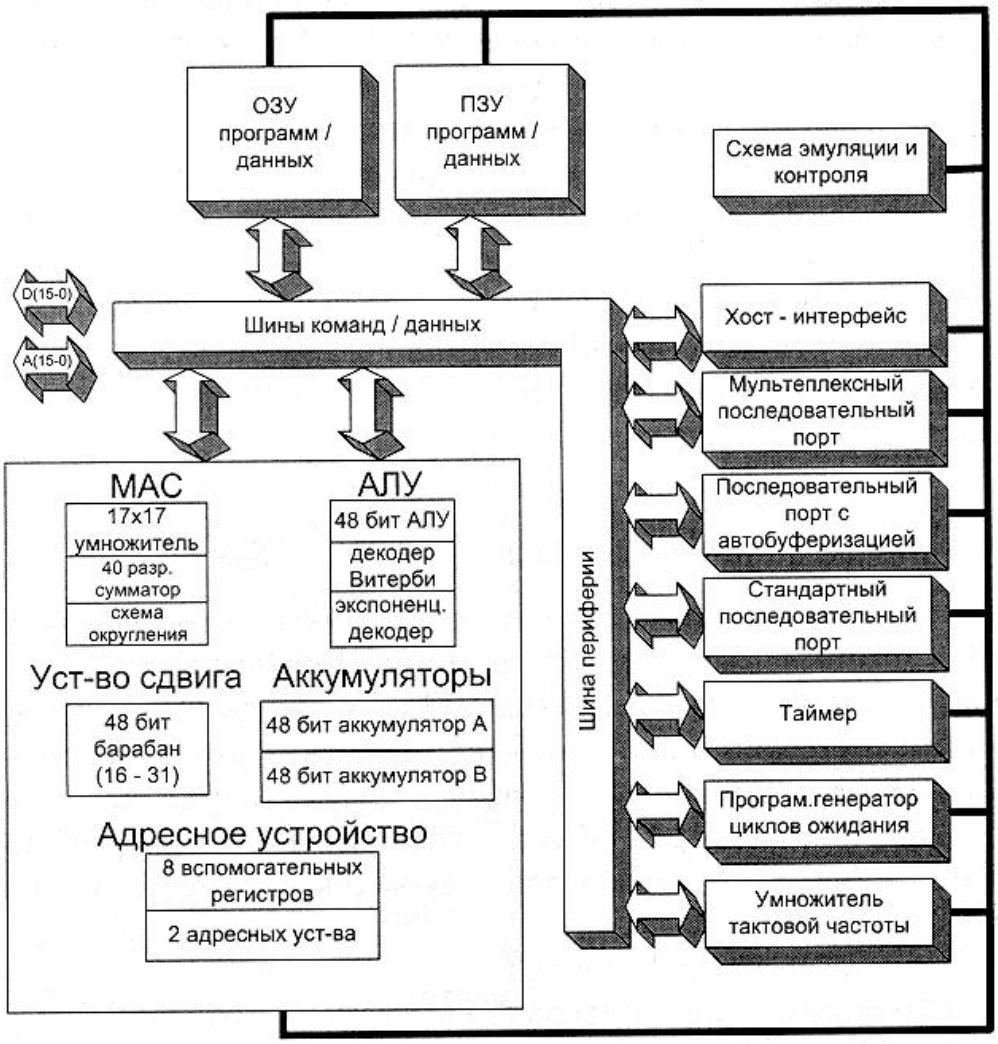


Рис. 5.1. Структура ПЦОС TMS320C54x

Генератор тактов ожидания позволяет увеличить число тактов внешней шины для работы с медленной внешней памятью и внешними устройствами.

Переключатель банков памяти позволяет автоматически добавлять один такт при пересечении границ банка памяти внутри программного адресного пространства или при переходе от пространства адресов команд к пространству адресов данных. Этот дополнительный такт позволяет устройству памяти освободить шину раньше, чем другое устройство получит доступ к ней, избегая тем самым конфликтной ситуации при обращении к памяти.

Микропроцессоры семейства имеют 64 К портов ввода/вывода. Эти порты предназначены для связи с внешними устройствами с использованием минимума дополнительных внешних декодирующих схем. Интерфейс главного порта (HPI) — 8-разрядный параллельный порт, предназначенный для связи ПЦОС и хост-процессора системы. Обмен данными между хост-процессором и ПЦОС осуществляется через внутрикристальную HPI-память объемом 2К 16-разрядных слов, которая может также использоваться как память команд или данных общего назначения. Скорость обмена по HPI составляет до 160 Мбайт/с.

ПЦОС данной подсерии содержат высокоскоростные дуплексные последовательные порты, позволяющие связываться с другими процессорами, кодеками и т.д. В ПЦОС реализованы следующие разновидности последовательных портов:

универсальный;

мультиплексный с временным уплотнением;

с автобуферизацией.

Универсальный последовательный порт использует два отображаемых в память регистра: передачи данных и приема данных. Передача и прием данных сопровождается генерацией маскируемого прерывания, которое может быть обработало программно. Буферизированный последовательный порт позволяет осуществлять непосредственный обмен между устройством и памятью, не используя при этом ресурсов процессора. Максимальная скорость обмена по последовательному порту может составлять до 40 Мбайт/с.

Как и в сериях TMS320C5x, TMS320C2xx, в ПЦОС TMS320C54x реализована эффективная трехуровневая система управления энергопотреблением. Низкое, управляемое энергопотребление, высокая производительность (до 66 MIPS) и широкие функциональные возможности при невысокой цене предопределили востребованность ПЦОС TMS320C54x в следующих областях: сотовые и радиотелефоны, персональные системы радиовызова, «персональные цифровые ассистенты» (PDA), техника беспроводной передачи данных (радиосети) и т.д.

6. ВЫСОКОПРОИЗВОДИТЕЛЬНЫЕ ПЦОС СЕРИИ TMS320C6000

Серию TMS320C6000 компании Texas Instruments Inc. от рассмотренных ранее отличает большая производительность и стоимость (рис. 6.1).

Высокая производительность достигается за счет внедрения параллельной архитектуры VelociTI, реализованной на основе технологии VLIW (“Very Long Instruction Word” или «очень длинного командного слова»), а также за счет применения ряда других аппаратных решений и средств разработки.

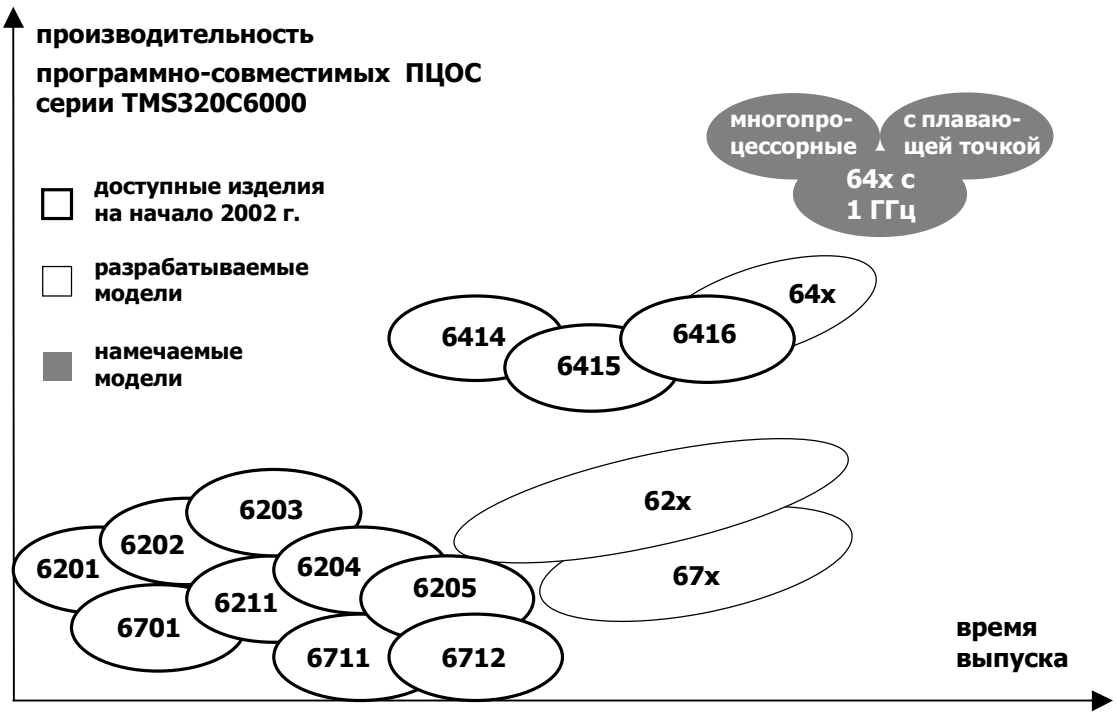


Рис. 6.1. Высокопроизводительные ПЦОС серии TMS320C6000

По оценкам специалистов, применение данной архитектуры в будущем позволит, при сохранении совместимости по командам, достичь рубежей 8000 MIPS для ПЦОС с фиксированной точкой и 3 GFLOPS для ПЦОС с плавающей.

Ожидается, также существенное удешевление ПЦОС данной серии как за счет общего снижения стоимости устройств при совершенствовании технологии, так и за счет выпуска новых моделей ПЦОС.

Изготовляются и широко применяются следующие три разновидности ПЦОС серии TMS320C6000 (табл. 6.1):

- подсерия ПЦОС TMS320C62x – устройства с фиксированной точкой и производительностью от 1200 до 2400 MIPS;
- подсерия ПЦОС TMS320C64x – устройства с фиксированной точкой и производительностью от 3200 до 4800 MIPS. Данные ПЦОС являются наиболее скоростными (табл. 6.2.) и предназначены как для широкого применения (TMS320C6414), так и для использования в мультимедийных (TMS320C6415) и телекоммуникационных (TMS320C6416) приложениях [14];
- подсерия ПЦОС TMS320C67x – устройства с плавающей точкой и производительностью от 600 до 1350 MFLOPS.

Таблица 6.1. Производительность подсерий ПЦОС TMS320C6000

Подсерии TMS320C6000	Производительность			
	Тактовая частота, МГц	MIPS/ MFLOPS	MMAC (16-разрядные слова)	MMAC (8-разрядные слова)
TMS320C62x	150-300	1200-2400 MIPS	300-600	300-600
TMS320C64x	400-600	3200-4800 MIPS	1600-2400	3200-4800
TMS320C67x	100-225	600-1350 MFLOPS	200-550	200-550

Таблица 6.2. Оценка продолжительности выполнения популярных алгоритмов

Подсерии TMS320C6000	Быстрое преобразование Фурье (FFT) комплексный спектр, длительность N = 1024, Radix 4		Фильтрация сигналов (Digital Filtering) фильтр с КИХ, число выходных точек M = 100 (64 – для TMS320C67x)	
	тактов процессора	мкс	тактов процессора	мкс
TMS320C62x	13228	66,0	6410	23,0
TMS320C64x	6002	12,0	1019	2,0
TMS320C67x	18055	108,3	2216	13,3

При проектировании ПЦОС серии TMS320C6000 особое внимание изготовителя уделялось снижению времени, которое понадобится пользователю для разработки и выпуска конечных систем.

Сокращению этих сроков способствует свойство совместимости устройства с фиксированной точкой с соответствующим устройством с плавающей.

ПЦОС TMS320C67x имеют совместимость по командам и по выводам микросхем с соответствующими ПЦОС TMS320C62x, что позволяет разработчику быстро выполнять прототипы, используя плавающую точку, и легко переходить к ПЦОС с фиксированной точкой для снижения стоимости изделия при производстве. То есть вначале разработчик может взять за основу ПЦОС с плавающей точкой, отработать все элементы устройства, определить оптимальные алгоритмы обработки данных. При этом большие запасы по производительности и по точности вычислений позволяют заниматься именно алгоритмами, а не экономией ресурсов. После, когда все параметры определены, наступает этап оптимизации системы с учетом наработанных решений и перевод ее на более дешевый ПЦОС с фиксированной точкой [2].

Данный подход, предопределил переход от аппаратно-ориентированной среды разработки к программным моделям, что делает процесс разработки более быстрым, дешевым и простым.

Изготовитель также производит широкий ассортимент аналоговых и аналого-цифровых устройств, ориентированных на применение совместно с ПЦОС серии TMS320C6000.

6.1. Архитектура VelociTI

Все ПЦОС серии TMS320C6000 основаны на одном и том же 32-разрядном ядре центрального процессора с высоко параллельной и детерминированной архитектурой VelociTI (рис. 6.2).

Архитектура ядра ПЦОС серии TMS320C6000 включает 8 модулей - два умножителя и шесть АЛУ. Все модули максимально независимы, что дает компилятору и оптимизатору множество комбинаций их использования. На каждом такте ПЦОС выбирается восемь 32-битных RISC-подобных инструкций. Предусмотренная в архитектуре VelociTI упаковка команд позволяет исполнять эти восемь инструкций параллельно, последовательно или параллельно/последовательно. Эта оптимизированная схема существенно снижает размер кода, количество выборок команд и потребление питания. При добавлении функции плавающей запятой к шести из восьми функциональных модулей из ПЦОС с фиксированной точкой 'C62x получается ПЦОС с плавающей точкой – 'C67x. При этом система команд 'C62x - расширение системы команд 'C67x и весь код написанный для 'C62x будет выполняться на 'C67x без модификаций самого кода.

Рассмотрим подробнее архитектуру ядра ПЦОС серии TMS320C6000. На рис. 6.2 изображена упрощенная схема именно ядра, без периферии и внешних шин, иллюстрирующая архитектуру VelociTI [2].

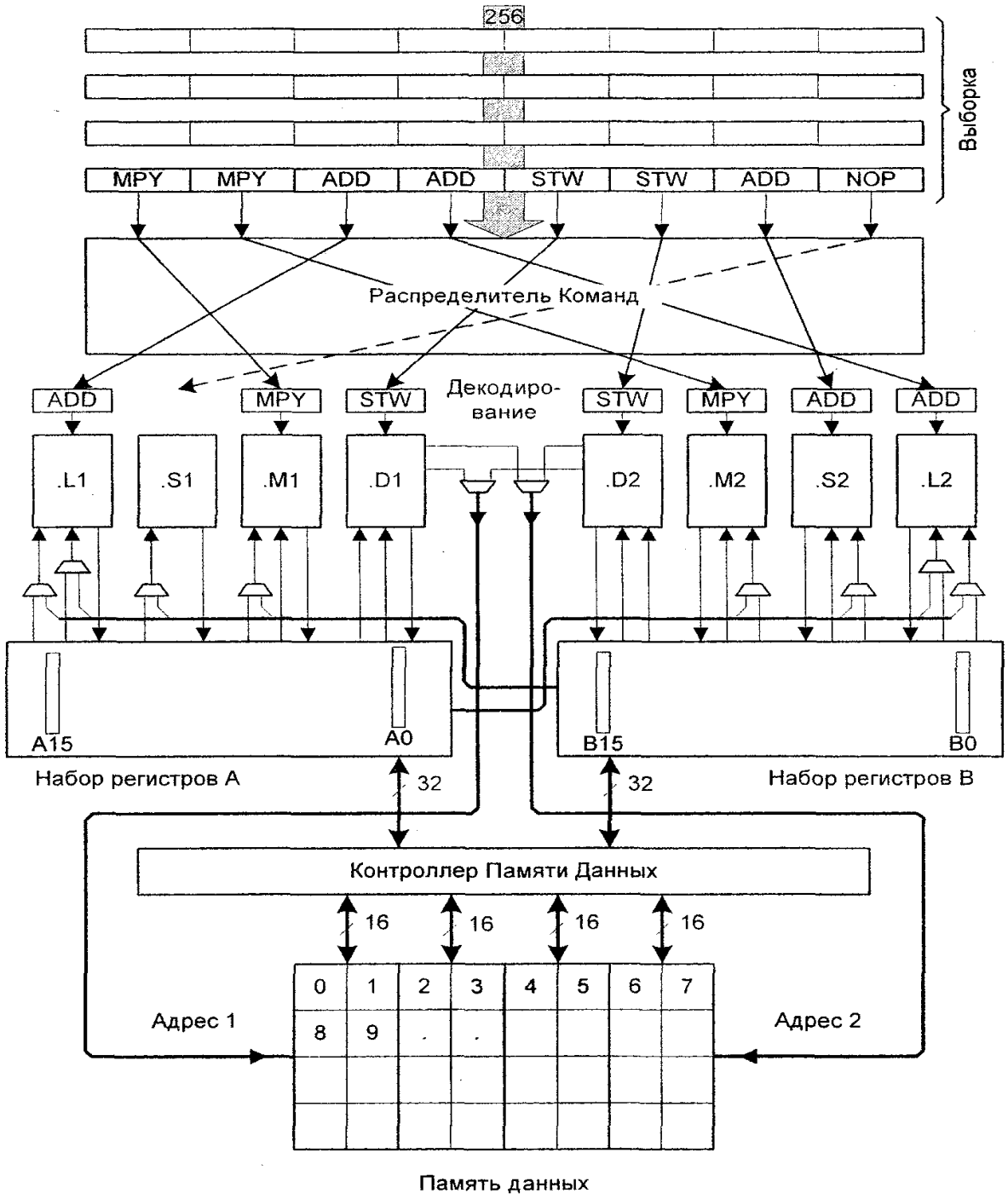


Рис. 6.2. Архитектура VelociTI

Как видно из рисунка, ПЦОС использует очень длинные инструкции (256 бит) для выдачи до 8 команд по 32 бита для каждого из восьми функциональных модулей в каждом такте. Выбираются инструкции всегда по 256 бит, однако длина исполняемого пакета может быть разной, как показано на рисунке. Переменная длина выполняемой команды позволяет существенно сэкономить память – это отличительная черта TMS320C6000 от остальных ПЦОС с очень длинным командным словом.

Ядро ПЦОС TMS320C6000 имеет два набора функциональных модулей. Каждый набор включает в четыре модуля и регистровый файл. Каждый файл состоит из 16 32-разрядных регистров, таким образом, всего в ядре 32 32-разрядных регистра.

Два набора функциональных модулей, связанных с двумя наборами регистров, создают разделение ядра на стороны A и B. Четыре модуля с каждой стороны ПЦОС имеют произвольный доступ к регистровому файлу данной стороны. Кроме того, каждая сторона имеет шину, соединенную с регистровым файлом другой стороны. При доступе к регистрам своей стороны возможен доступ к регистрам всех модулей одновременно в одном такте.

Другой особенностью архитектуры ПЦОС TMS320C6000 является использование стратегии сохранения/загрузки, при которой все команды работают с регистрами. При этом два адресных модуля D1 и D2 выделяются только под передачу данных между регистровым файлом и памятью. Шины адреса, управляемые D-модулями, позволяют использовать адрес, сгенерированный в одном регистровом файле, для операций с данными в другом регистровом файле.

Ядро ПЦОС TMS320C6000 поддерживает широкий набор режимов косвенной адресации, включая линейный или кольцевой с 5- или 15-битным смещением.

Все команды могут быть условными, и большинство команд могут использовать любой из 32 регистров. Некоторые регистры могут быть выделены для поддержки специфических режимов адресации или для хранения условий для условных команд. Два M-модуля выделены под умножители 16x16. Два S- и два L-модуля выполняют арифметические, логические операции и операции перехода, и при этом результаты их выполнения доступны в каждом такте (возможна задержка до 5 тактов конвейера, но большинство команд выполняются за 1 такт).

Процесс обработки команды в ядре начинается после выборки 256-битовой инструкции из внутренней памяти команд, которая также может быть сконфигурирована как кэш-память команд. Далее каждая из 32-битных команд распределяется на свой модуль для исполнения. При этом у команд, выполняемых на разных модулях, проверяется младший бит. Он устанавливается в 1 для всех команд, которые должны выполняться одновременно. Команды, которые собраны для одновременного

выполнения (до 8 команд), образуют пакет выполнения. Ноль ставится в младшем бите команды, которая нарушает последовательность выполнения и откладывает команду на следующий пакет выполнения. Всего в выборке может быть до 8 пакетов выполнения. Очередной пакет размещается для выполнения в модулях в каждом такте. До окончания выполнения пакета следующий пакет выборки из памяти не выбирается. Эта «стратегия» позволяет существенно экономить память команд и менять режим работы программы от одновременного параллельного выполнения 8 команд на 8 модулях до практически последовательного выполнения команд, в зависимости от требований алгоритма.

Обратим внимание на организацию памяти данных. Как уже упоминалось, данные из функциональных модулей помещаются в регистры, а затем по адресам, генерируемым D-модулями, идет обмен с памятью данных. При этом каждый из регистровых файлов соединен 32 разрядными шинами с диспетчером памяти. Диспетчер организует одновременную выборку из памяти по четырем шинам до 64 разрядов по двум подаваемым адресам. При этом память дробится на множество мелких банков, что практически исключает конфликты доступа к памяти. Такое решение обеспечивает доступ без задержек при параллельных потоках обращения и при возможности адресовать отдельно каждый байт памяти.

Фактически вся память данных ПЦОС организована не как двухпортовая, а как многопортовая, и количество одновременно выбираемых данных может меняться [2].

6.2. Структура и состав ПЦОС серии TMS320C6000

На рис. 6.3 показана внутренняя структура ПЦОС TMS320C6000 [6].

Как видно из рисунка, сам ПЦОС можно условно разделить на несколько частей. Во-первых, это собственно ядро процессора, структура которого рассматривалась ранее. Во-вторых, области памяти данных и памяти команд; в третьих – размещенная на кристалле периферия. Все эти части связаны между собой двумя контроллерами – памяти команд или кэш-памяти и памяти данных. Эти блоки связывают ядро ПЦОС и банки памяти (с их специфической конфигурацией и доступом) с традиционными шинами, к которым подключаются периферийные модули и внешние устройства.

Рассмотрим подробнее периферийные устройства ПЦОС TMS320C6000 [6].

6.2.1. Контроллер ПДП

Устройство предназначено для передачи данных из памяти в память без участия центрального процессора. Контроллер ПДП имеет четыре

основных программируемых и пять дополнительных каналов. Кроме того, контроллер ПДП используется при начальной загрузке программы в память ПЦОС при старте (bootloader).

6.2.2. Хост «Порт-интерфейс» (ХПИ)

ХПИ используется как для обмена данными с управляющим контроллером, так и для асинхронного обмена. ХПИ – это 16-разрядный параллельный порт, который обеспечивает прямой доступ к памяти ПЦОС. При этом ПЦОС является управляющим устройством для данного интерфейса, что существенно упрощает процедуру доступа. ПЦОС может обмениваться информацией, как через внутреннюю, так и через внешнюю память. Кроме того, ПЦОС может иметь прямой доступ к большинству устройств размещенной на кристалле периферии.

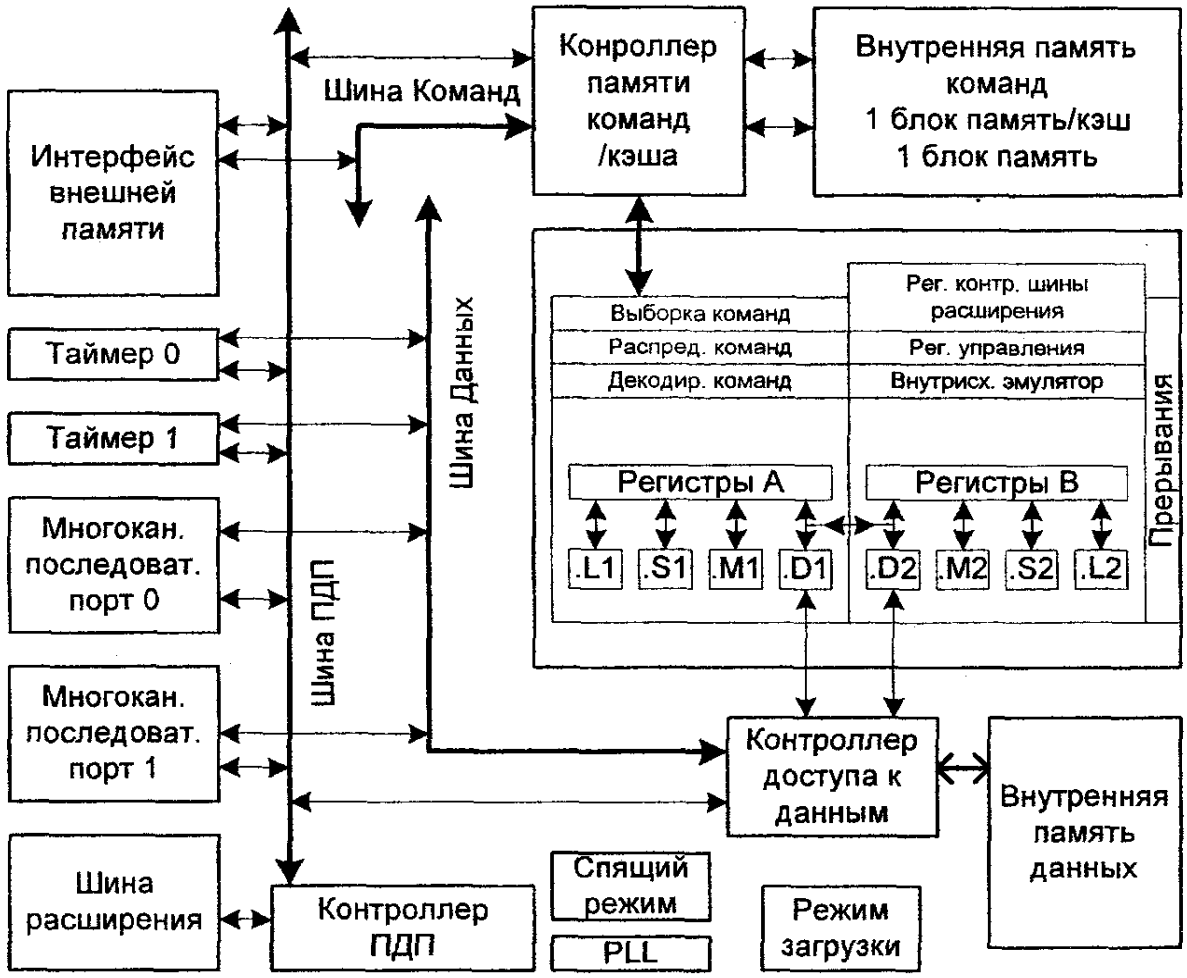


Рис. 6.3. Структура ПЦОС TMS320C6000

6.2.3. Шина расширения (ШР)

ШР является расширением как ХПИ, так и ИВП (см. ниже). С использованием ШР можно реализовать 32-разрядный ХПИ, который будет работать аналогично штатному 16-разрядному. ШР также может реализовать синхронный протокол обмена между хост ПЦОС и ЦПУ, что дает возможность прямого подключения к большому набору стандартных шин хост ПЦОС. Также к шине расширения могут быть подключено синхронное FIFO и асинхронные периферийные устройства.

6.2.4. Интерфейс внешней памяти (ИВП)

ИВП это специальный блок, предназначенный для обмена данными с внешней памятью и быстродействующими внешними устройствами. ИВП может принимать запросы на обмен с внешней памятью от трех: контроллеров памяти данных, программной памяти-КЭШ и ПДП. Поскольку сам ПЦОС – очень скоростное устройство, то ИВП не только выводит наружу классическую шину, но и имеет специальные сигналы для непосредственного подключения быстродействующего синхронного внешнего ОЗУ как динамического (SDRAM), так и статического (SBSRAM). Кроме того, к ИВП можно подключить и обычное статическое ОЗУ, ПЗУ, FIFO и другие устройства.

6.2.5. Начальный загрузчик

ПЦОС TMS320C62x и TMS320C67x могут иметь множество режимов начальной загрузки, которые определяют, что именно будет делать ПЦОС после сброса при подготовке к инициализации. Они могут включать загрузку программы с внешнего ПЗУ через ИВП или загрузку программы через ХПИ/ШР из внешнего устройства.

6.2.6. Многоканальный буферизованный последовательный порт (МКБПП)

Это последовательный скоростной порт, базирующийся на стандартном последовательном порте, как и в ПЦОС других серий. Он имеет возможность читать и записывать данные в память без участия центрального процессора через контроллер ПДП. Кроме того, у него существуют многоканальные расширения, совместимые со стандартами El, Tl, SCSA и MVIP.

Отметим следующие функциональные возможности последовательного порта:

полнодуплексная работа;

двойная буферизация данных (позволяет поддерживать непрерывность потока);

независимые тактовые частоты и схемы синхронизации для приема и передачи данных;

прямое подключение микросхем аналоговых интерфейсов, микросхем ЦАП и АЦП с последовательным интерфейсом.

МКБПП, по сравнению со стандартным последовательным портом, имеет дополнительные возможности:

прямое подключение к шинам;

многоканальный обмен при количестве каналов до 128;

переменный размер данных 8, 12, 16, 20, 24 и 32 бита;

встроенное u-Law и A-Law компандирование;

возможность передачи первым старшего или младшего разряда данных;

программируемая полярность сигналов синхронизации и тактовых сигналов данных;

гибкое программирование внутренних тактовых импульсов и синхронизации.

6.2.7. Таймер

ПЦОС серии TMS320C6000 имеют два 32-разрядных таймера, которые могут быть использованы для:

задания временных событий;

реализации счетчиков;

генерации импульсов

прерывания ПЦОС;

посылки синхроимпульсов в контроллер ПДП.

6.2.8. Селектор прерываний

Периферия ПЦОС TMS320C6000 может иметь до 32-х источников прерываний. ЦПУ имеет возможность обрабатывать 12 прерываний. Селектор прерываний дает возможность выбора тех 12 прерываний, которые будут использоваться, и также дает возможность смены полярности внешних входов прерываний.

6.2.9. «Спящие» режимы

Логика снижения потребляемой мощности позволяет снимать тактовые сигналы с элементов ПЦОС для снижения энергопотребления. Несмотря на свое предназначение для базовых станций, ПЦОС TMS320C6000 также имеют режимы снижения энергопотребления. КМОП схемы в основном потребляют энергию в момент переключения, и чем выше частота работы, тем больше это потребление. При включении «спящих» режимов у ПЦОС снимается тактовая частота сначала с ядра ПЦОС, затем с периферии, размещенной на кристалле, и последний «третий» режим снимает тактовую частоту практически со всего кристалла, в том числе и с блока умножения частоты.

ПЦОС имеет встроенный умножитель частоты с возможностью умножения внешней тактовой частоты на 2 и на 4, что делает возможным работу с низкой входной частотой и упрощает проектирование.

6.2.10. Габаритные размеры

Все ПЦОС серии TMS320C6000 выпускаются в 352-выводных или 452-выводных корпусах BGA со стороной квадрата 35, 27 и 18 мм.

6.3. Средства разработки для ПЦОС серии TMS320C6000

Для разработчиков устройств на базе ПЦОС серии TMS320C6000 предлагается широкий набор мощных средств разработки и отладки. Новая архитектура ПЦОС данного семейства предполагает и новый подход к процессу разработки. ПЦОС серии TMS320C6000 позволяют уменьшить время и стоимость создания проекта за счет переноса большей части работы на программное обеспечение средств разработки. Разработчику остается написать алгоритм на языке высокого уровня, а его реализация и оптимизация с использованием всех преимуществ архитектуры ПЦОС серии TMS320C6000 перекладывается на компилятор. Это снимает одну из основных трудностей при работе на ПЦОС с длинным командным словом – распараллеливание алгоритма. Такой подход имеет ряд преимуществ – во-первых, существенно сокращается срок разработки и качество получаемого продукта за счет сосредоточения именно на реализуемой задаче, а не на средствах ее реализации. Во-вторых, повышается качество и оптимальность кода за счет того, что автоматический оптимизатор всегда помнит все особенности архитектуры ПЦОС и использует их по максимуму. Надо отметить, что время разработки сокращается и за счет существенного уменьшения времени отладки из-за отсутствия ошибок в коде низкого уровня, которые часто возникают по вине разработчика (что-то забыл или не учел) [6].

Развитые средства оценки времени выполнения алгоритмов позволяют принципиально оценить производительность алгоритма еще на начальных этапах его разработки, что снимает проблему возможной нехватки ресурсов после макетирования устройства.

Процесс реализации алгоритма на ПЦОС TMS320C6000 протекает в несколько стадий. Вначале разработчик пишет алгоритм на языке Си или на ассемблере, и компилятор переводит его программу в машинный код с использованием всех возможностей ПЦОС, таких как конвейерная обработка и интеллектуальное нахождение параллелизма в исходной программе для задействования возможностей параллельной обработки команд в ПЦОС. После наступает этап оценки производительности кода программными средствами, что позволяет оценить достигнутые результаты и провести оптимизацию кода без обращения к аппаратному

обеспечению. И только следующим шагом идет проверка на макете устройства или отладочном модуле.

Программные средства, предназначенные для разработки программ для ПЦОС TMS320C6000 [6]:

С-компилятор, ассемблер и компоновщик;
отладчик;
программный симулятор;
среда Code Composer Studio.

6.3.1. Высокоуровневый С-компилятор, ассемблер и компоновщик

Данные программные продукты представляют собой набор средств для компиляции кода. Специально ориентированы на реализацию оптимальных программ, созданных по алгоритмам цифровой обработки сигналов. Имеет широкий набор встроенных средств оптимизации, как общего плана, так и специализированных для ПЦОС серии TMS320C6000. Является ANSI совместимым компилятором. В состав данного продукта входит ассемблерный оптимизатор - средство для перевода последовательного ассемблерного кода в параллельную, специфичную для ПЦОС серии TMS320C6000 форму.

6.3.2. Программный симулятор

Продукт позволяет производить проверку выполнения кода на персональном компьютере без ПЦОС. Имеет функциональные возможности и интерфейс, аналогичный отладчику. В июне 1999 года появилась новая версия программного симулятора - Code Composer Simulator.

6.3.3. Code Composer Studio

Многомодульный программный продукт представляет собой мощную интегрированную отладочную среду как для ПЦОС TMS320C6000, так и ПЦОС других серий. Имеет развитый оконный интерфейс, встроенные средства редактирования, возможность дизассемблирования и вызова внешнего компилятора, имеет расширенные средства визуализации данных.

По оценкам изготовителей, среда Code Composer Studio станет стандартом и останется, чуть ли не единственным продуктом для программирования ПЦОС компании Texas Instruments Inc. по крайней мере, до 2020 года.

6.4. ПЦОС подсерии TMS320C62x

Самым первым изделием серии TMS320C6000 был выпущен ПЦОС TMS320C6201, который оперирует с данными в формате с фиксированной

точкой со скоростью 1600 MIPS на тактовой частоте 200 МГц. Как и другие устройства серии, ПЦОС TMS320C6201 может интегрироваться с использованием широкого набора инструментальных средств, и применяться для экономичного решения многих задач, требующих высокопроизводительной обработки сигналов. В числе типовых областей применения ПЦОС подсерии TMS320C62x компании Texas Instruments Inc. можно упомянуть:

- сети TCP/IP с неограниченной полосой пропускания;
- универсальная беспроводная связь;
- удаленная медицинская диагностика;
- радикально новые возможности телефонии;
- персональные средства информационного обеспечения и защиты.

В то же время для увеличения производительности возможно использование ПЦОС подсерии TMS320C62x и в уже существующих прикладных системах. Таких как:

базовые станции мобильной связи;

модемные пулы и серверы удаленного доступа;

xDSL модемы следующего поколения и кабельные модемы;

многоканальные телефонные платформы, включая центральные офисные коммутаторы, PBX и системы речевой передачи сообщений;

мультимедийные системы.

ПЦОС TMS320C6201 — помимо ядра (ЦП) содержит:

1 Мбит внутрикристалльной памяти (512 Кбит для программы, 512 Кбит для данных);

32-битовый интерфейс внешней памяти, поддерживающий стандарты памяти SDRAM (статическая память с динамическим рандомизированным доступом), SBSRAM (синхронная пакетная статическая память), SRAM (статическая память);

два последовательных расширенных буферизированных порта (EBSP);

16-битовый порт host-ПЦОС;

два канала доступа к памяти данных с возможностью начальной загрузки;

генератор интервалов времени.

ПЦОС подсерии TMS320C62x построены в соответствии архитектурой VelociTI и технологии VLIW, которые позволяют обеспечить лучшую эффективность функционирования за счет ослабления ограничений на порядок и способ выполнения инструкций.

ПЦОС TMS320C6201 состоит из трех основных частей: центрального процессора (или «ядра»), периферийных устройства и памяти [2].

Ядром TMS320C6201 является VelociTI-процессор с восемью функциональными модулями, включая 2 умножителя и 6 АЛУ. Модули взаимодействуют через два регистровых файла, содержащих по 16 32-

разрядных регистров. Таким образом, ЦП может выполнять до 8 команд за один такт.

Программный параллелизм выявляется на этапе компиляции, анализ зависимости по данным аппаратными средствами на стадии выполнения не производится. Код программы выполняется на независимых функциональных устройствах в той же последовательности, в которой программируется.

В ПЦОС используется упаковка инструкций, сокращающая размеры кода и время выборки команд. 256-разрядная шина памяти программ позволяет выбирать за один такт восемь 32-разрядных команд. Все инструкции содержат условия их выполнения, что позволяет сократить расходы производительности ПЦОС на выполнение переходов и увеличить степень параллелизма обработки.

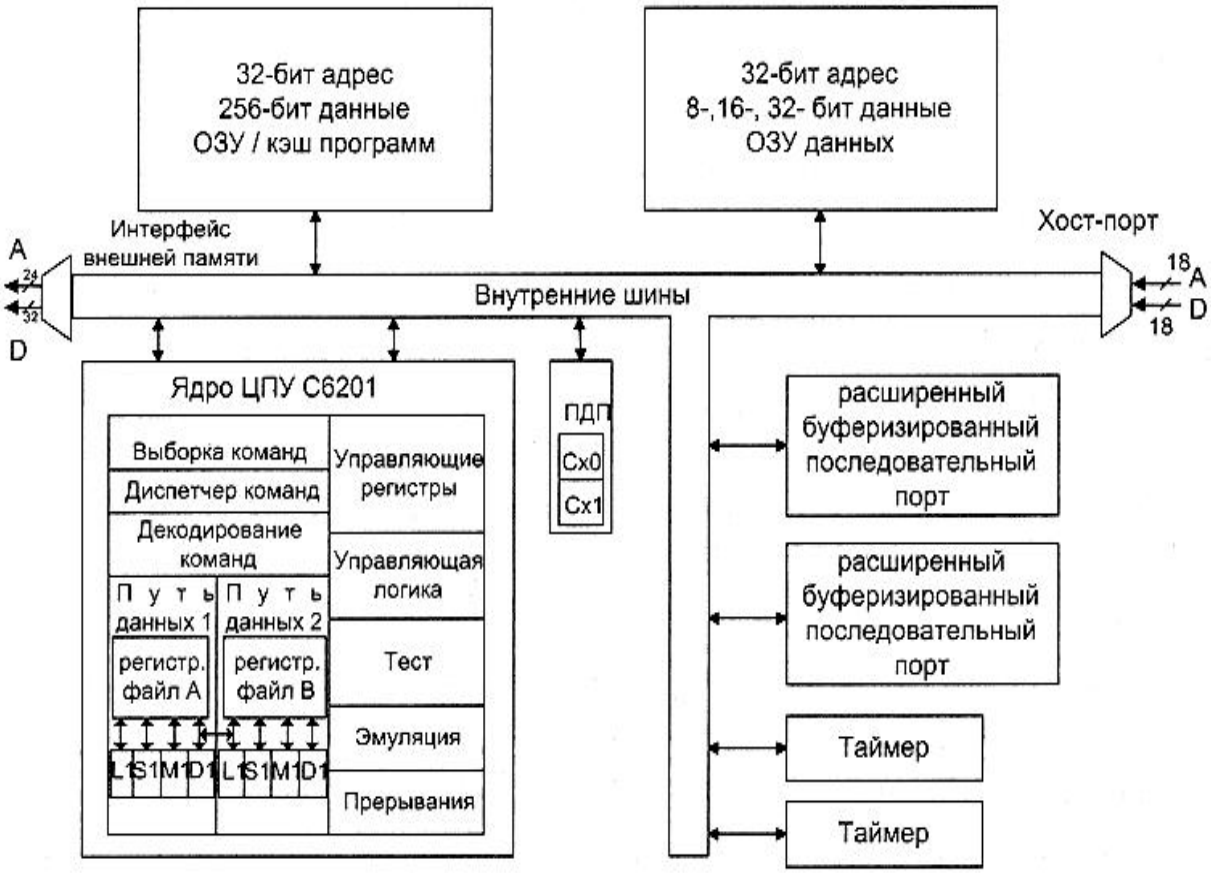


Рис. 6.4. Структура ПЦОС TMS320C6201

ПЦОС может оперировать с 8/16/32-разрядными данными. Для приложений, требующих высокой точности вычислений, предусмотрена возможность вычислений с 40-разрядными операндами. Для результатов всех основных арифметических операций выполняется округление и

нормализация. В ПЦОС реализованы операции над битовыми полями, такие как «выделить» (extract), «установить» (set), «очистить» (clear), «подсчет битов» (bit counting).

ЦП имеет два тракта обработки данных, каждый из которых содержит функциональные модули (.L, .S, .M, .D) и регистровый файл (16 32-разрядных регистров). Функциональные модули выполняют сдвиг, умножение, логические и адресные операции. Все операции выполняются над регистрами. Два набора устройств адресации данных (.D1 и .D2) отвечают исключительно за все пересылки данных между регистровым файлом и памятью. Управляющий регистровый файл определяет различные аспекты функционирования ПЦОС.

Процесс обработки VLIW начинается с выборки из памяти команд 256-битового пакета. Команды связываются для совместного выполнения в пакет (до 8 команд) по значению младшего бита команды (LSB).

Устройство выборки-декодирования-диспетчеризации команд может направлять к функциональным модулям до восьми 32-разрядных команд за один такт по каждому из путей обработки (A и B).

В ПЦОС подсерии TMS320C62x реализованы прямой и циклический (для регистров A4-A7 и B4-B7) способы адресации, которые определяются регистром режима адресации (AMR).

ПЦОС TMS320C6201 имеет 14 прерываний, соответствующих сигналу Reset, немаскируемому прерыванию (NMT) и прерываний с номерами 4-15.

ПЦОС подсерии TMS320C62x содержат внутрикристальную память, которая может использоваться как память программ или программный кэш. Интерфейс внешней памяти ПЦОС (EMIF) объединяет в единое адресное пространство внутреннюю и внешнюю памяти.

Внутрикристальная память разделена на память данных и программ.

ПЦОС TMS320C6201 имеет два 32-разрядных порта к памяти данных и один 256-разрядный порт к памяти программ для выборки инструкций. TMS320C6201 имеет по 64 Кбайт памяти данных и программ на кристалле. В ПЦОС используется расслоение памяти данных (четыре 16-разрядных банка) для повышения скорости выборки за счет одновременного обращения к различным банкам памяти.

Дополнительно ПЦОС подсерии TMS320C62x могут содержать на кристалле интерфейс внешней памяти (EMIF), контроллер ПДП (DMA), интерфейс хост-порта (HPI), средства энергосбережения, расширенные буферизированные последовательные порты, 32-разрядные таймеры.

7. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

7.1. Программирование ПЦОС TMS320C26

Целью данных лабораторных работ является ознакомление с архитектурой, системой комманд и методами программирования ПЦОС TMS320C26 [4].

В лабораторных работах используется функциональный модуль DSK, расширяющий функциональные возможности ПЦОС TMS320C26, содержащий АЦП и ЦАП, и подсоединенный к персональному компьютеру через последовательный порт RS-232.

В ходе выполнения работ при программировании используется транслятор ассемблера и отладчик.

Лабораторная работа № 26-1

Задание

1) Осуществить инициализацию области памяти ПЦОС десятичными значениями 0 – 100. В качестве начального адреса памяти принять **0400h**.
2) Вычислить и занести в аккумулятор сумму всех членов последовательности.

Методические указания. Пример программы

Точкой входа в программу следует выбрать **0fb00h**.

Для определения начального адреса программной памяти и инициализации этим значением программного счетчика необходимо поместить в начало программы следующие строки:

```
.ps          0fb00h
.entry
```

В силу особенностей архитектуры сигнального процессора TMS320C26 для выполения задания необходимо правильно произвести конфигурацию 4 блока оперативной памяти (B0, B1, B2 и B3). При заданном начальном адресе размещения таблицы в памяти данных (0400h) необходимо настроить блок B0 как программную память, а все остальные блоки – как память данных. Для этого после указанных выше директив **.ps** и **.entry** необходимо поместить следующую стоку:

```
conf          1
```

Для ознакомления с картами программной памяти и памяти данных ПЦОС TMS320C26 обратитесь к Таблице П2 и Таблице П3 Приложений.

Далее приводится простейшая программа, реализующая запись двух целых чисел в память ПЦОС (по адресам 0400h 0401h) и их сложение с помещением результата в аккумулятор.

Пример 7.1. Сложение двух шестнадцатиразрядных целых чисел

```
.ps     0fb00h      ;   установить текущий адрес программной памяти
.entry              ;   загрузить прогр. счетчик текущим адресом памяти
DA1     0400h       ;   определяем и инициализируем две константы
DA2     0401h       ;   адресами исходных слагаемых

lark    AR1, 100    ;   загрузить 100 во вспомогательный регистр AR0
sar  AR1,DA1        ;   записать содержимое AR1 по адресу DA1
lark    AR1,200     ;   реализовать загрузку 200 в ячейку памя-
sar  AR1,DA2        ;   ти с адресом DA2

lrlk AR2,DA1        ;   загрузить в AR2 адрес первого слагаемого
larp    AR2         ;   сделать AR2 текущим
lac  *+,0           ;   загрузить в аккумулятор первое слагаемое с нулевым
                    ;   сдвигом и инкрементировать адрес, хранящийся в AR2
add *,0             ;   добавить к аккумулятору 2-ое слагаемое без сдвига
.end                ;   закончить ассемблирование
```

Указания к выполнению

Выполнение лабораторной работы осуществляется в три этапа:

программи рование,

ассемблирование,

работа с программой-отладчиком.

Программирование

На этом этапе осуществляется составление программы на ассемблере TMS320C26 в соответствии с предложенным заданием.

Для последующего тестирования и отладки текст программы необходимо набрать в редакторе текстов **ASCII** (например в **Notepad** для **Windows**) и сохранить в файле с именем **BrN_1.asm**, где N – номер вашей бригады.

Ассемблирование

Для ассемблирования полученной программы необходимо:

1. Переместить файл с ее текстом в директорию **C:\C26,** в котором произведена инсталляция функционального модуля DSK.
2. При работе в ОС Windows осуществить вызов командного процессора MS-DOS, выбрав **Start → Programms → MS-DOS-session** (альтернативным способом вызова является выбор **Start → Run** и ввод в

появившемся диалоговом окне **command** с последующим подтверждением ввода).

3. С помощью командной строки сделать директорию **C:\C26** текущей.
4. Осуществить вызов ассемблера DSK из командной строки по шаблону:

dska [*filename*] [-*option*]

где *filename* – имя файла, содержащего текст вашей программы; – *option* – опции, задающие способ ассемблирования программы (*filename* может не содержать расширение файла!).

Примечание. *В данном случае необходимо ввести*

dska BrN_1

В случае успешного ассемблирования на экран будут выведены строки, как показано на рис. 7.1 и ассемблер сформирует исполняемый файл с именем **BrN_1.dsk**. Этот файл содержит набор инструкций в машинных кодах, понятных микропроцессорам серии TMS320C2x.

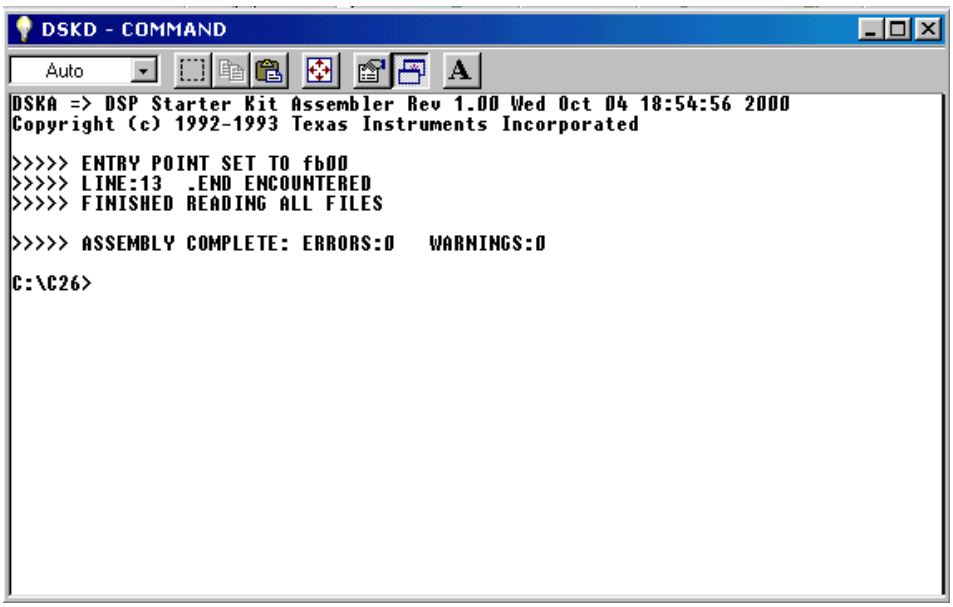


Рис. 7.1. Результат ассемблирования программы, не содержащей ошибок

В противном случае ассемблер сгенирирует сообщение об ошибке (рис. 7.2) и файл с расширением **.dsk** не создастся. Наряду с информацией о количестве ошибок ассемблер указывает строки, в которых, по его мнению, была допущена ошибка. Это существенно упрощает поиск и исправление ошибок на этапе ассемблирования.

5. Провести ассемблирование программы с параметром ***–l*** и с просмотром полученных результатов в соответсвующем файле.

Примечание. *Вызов ассемблера с этим параметром приводит к генерации наряду с исполняемым файлом текстового с листингом ассемблера. Этот файл имеет расширение .lst и содержит информацию о результатах ассемблирования каждой строчки программы как на рис. 7.2.*

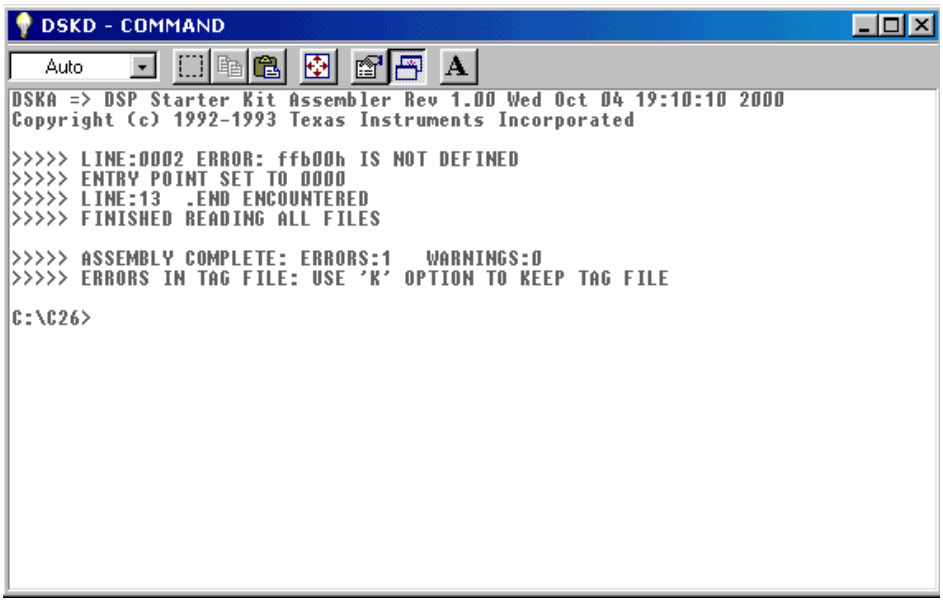


Рис. 7.2. Результат ассемблирования ошибочного файла

Работа с отладчиком

Запуск отладчика

Отладка полученной после ассемблирования программы производится с помощью входящего в комплект DSK отладчика.

Для его запуска необходимо:

включить модуль DSK соответствующим тумблером “**DSK Power**”;

с помощью командной строки сделать директорию **C:\C26** текущей;

осуществить вызов программы-отладчика путем ввода команды:

dskd c2

или

debugger

Примечание. *После выполнения данного пункта задания возможно появление сообщения об ошибке. В этом случае необходимо выйти из программы путем ввода Q или Esc, а затем осуществить повторный вызов отладчика в соответствии с рис.7.3.*

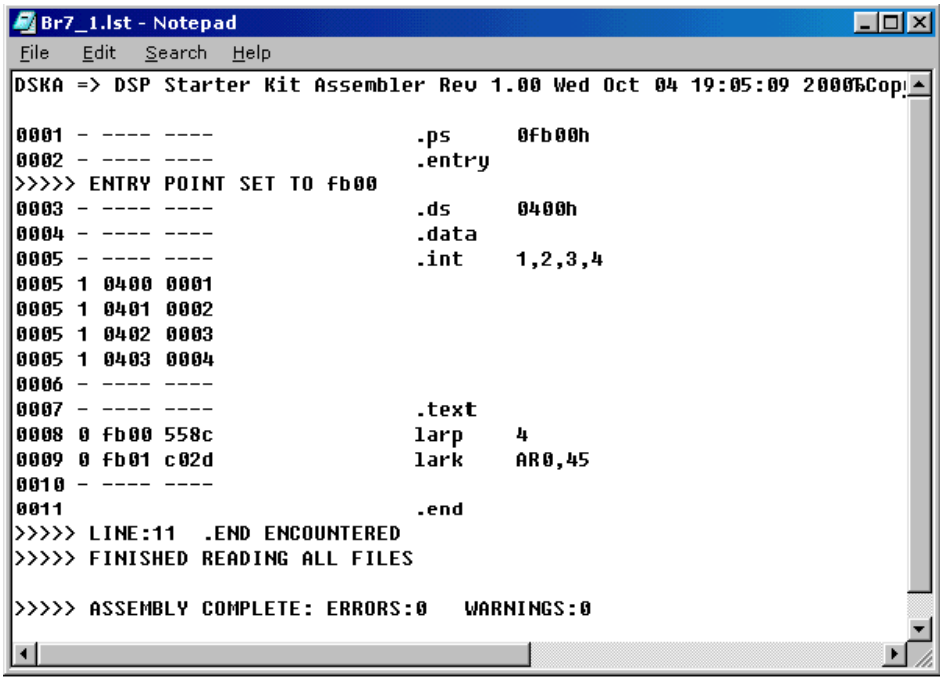


Рис. 7.3. Содержание листинг-файла

Ознакомление с графической оболочкой отладчика

В случае успешного запуска программы выполните:

1. Ознакомьтесь с графической оболочкой отладчика с помощью п. 4.3;
5. Осуществите загрузку файла **BrN_1.dsk,** полученного после ассемблирования, в память ПЦОС;
6. Загрузите программу без точек останова;
7. Проанализируйте содержимое регистров процессора;
8. Воспользовавшись возможностью просмотра содержимого ячеек памяти по конкретному адресу, убедитесь в том, что запись десятичных чисел произведена в указанную область памяти;
9. Осуществите сброс модуля;
10. Повторно запустите отладчик;
11. Выполните п.2.;
12. Осуществите пошаговое выполнение программы, наблюдая за постепенным изменением содержимого регистров;
10. Выйдите из программы.

Лабораторная работа № 26-2

Задание

Разработать на ассемблере TMS320C26 программу, осуществляющую вычисление минимума заданной функции на заданном интервале (Табл. 7.1).

Таблица 7.1. Вариант задания в соответствии с номером бригады

Вариант	Функция	Отрезок	Шаг аргумента
1	x^2-12x+37	[0,12]	1
2	x^2-14x+51	[1,13]	1
3	x^2-16x+67	[2,14]	1
4	x^2-18x+82	[3,15]	1

Методические указания

Пример программы

1. Память ПЦОС конфигурировать директивой **conf 1;**
2. Таблицу значений аргументов с заданным шагом поместить в область памяти начиная с адреса 0400h;
3. Таблицу соответствующих значений функции разместить в ОЗУ начиная с адреса 0424h;
4. Вычисленное значение минимума разместить по адресу 0430h.

Ниже приводится текст программы, осуществляющей перемножение содержимого предварительно инициализируемых ячеек памяти данных с адресами 0400h и 0401h. Младшее слово результата, имеющее разрядность 16 бит, сохраняется в ячейке памяти с адресом 0402h, старшее – 0403h.

Пример 7.2. Перемножение двух шестнадцатиразрядных целых чисел

```
.ps     0fb00h      ;   установить текущий адрес программной памяти
.entry              ;   загрузить программный счетчик текущим адресом прогр. памяти
.ds     0400h       ;   установить текущий адрес памяти данных
.text               ;   ассемблировать в программную память
conf    1           ;   конфигурировать RAM микропроцессора
.data               ;   ассемблировать в память данных
.int    8,9         ;   инициализировать целочисленными множителями
                    ;   ячейки памяти с адресами 0400h и 0401h
.text               ;   ассемблировать в программную память
lrlk    AR1,0400h   ;   загрузить в AR1 адрес первого множителя
larp    1           ;   сделать текущим AR1
lt      *+          ;   загрузить регистр T содержимым текущего регистра
mpy     *+          ;   вычислить и поместить в P произведение
                    ;   T*(содержимое текущего регистра)
pac                 ;   сохранить содержимое регистра P в аккумулятор
sacl    *+          ;   записать младшее слово аккумулятора в память по адресу 0402h
sach    *+          ;   записать старшее слово аккумулятора в память по адресу 0403h
.end                ;   закончить ассемблирование
```

Указания к выполнению лабораторной работы

1. Составить в соответствии с заданием программу на языке ассемблера ПЦОС TMS320C26.
2. Сохранить программу в файле **BrN_2.asm**, где N – номер вашей бригады.
3. Переместить этот файл в директорию **C:\C26.**
4. Осуществить ассемблирование программы.
5. Включите лабораторный стенд.
6. Вызвать отладчик.
7. С помощью меню ***Fill*** → ***Data*** инициализировать облась памяти данных, в которой будет производиться работа, нулевыми значениями.
8. Загрузить исполняемый файл **BrN_2.dsk** в память с помощью меню.
9. Осуществить выполнение программы.
10. Убедится в правильности полученных результатов.
11. С помощью меню ***Watch*** → ***Add*** дополнить список точек наблюдения адресами ячеек памяти, в которые будут записаны значения функции, а также адрес ячейки памяти, в которую будет осуществляться вывод результата.
12. Осуществить пошаговое выполнение программы, прослеживая за изменением содержимого соответствующих ячеек памяти в окне наблюдения (***TMS320C26 Watches***).
13. С помощью меню *eXec* → *Addr o*существить выполнение программы до момента запуска процедуры поиска минимума.
14. Выйти из программы.

7.2. Программирование ПЦОС TMS320F206

Целью выполнения лабораторных работ данного раздела является ознакомление с архитектурой, системой команд и методами программирования ПЦОС TMS320F206, изготовленного с использованием статической CMOS-технологии [8].

В лабораторных работах раздела используется функциональный модуль DSK, расширяющий функциональные возможности ПЦОС TMS320F206, содержащий АЦП и ЦАП, и подсоединенный к персональному компьютеру через последовательный порт RS-232.

В ходе выполнения работ используется TASM-ассемблер и средство отладки Code Explorer для снижения трудоемкости и временных затрат при создании прикладных программ.

Лабораторная работа 206-1

Задание

1. Осуществить инициализацию области памяти процессора десятичными значениями в диапазоне (50 - N) ÷ (100 – N), где N – номер бригады. При этом, в качестве начального адреса памяти принять 0200h.
2. Рассчитать и расположить в памяти процессора, начиная с адреса 0264h зачения квадратов содержимого инициализированных ячеек памяти.
3. При использовании средства отладки Code Explorer, осуществить визуализацию сформированной в памяти процессора таблицы квадратов во временной области.

Методические указания

Ниже приводятся некоторые пояснения относительно того, как выбрать и настроить пространства памяти изучаемого процессора. Для улучшения понимания данного вопроса рекомендуется ознакомится с п. 4.

Память ПЦОС серии TMS320F2000 организована в четыре индивидуально адрессуемых пространства: программной памяти, локальной и глобальной памяти данных, а также памяти ввода/вывода. При этом общий объем адрессуемой памяти составляет 244K 16-битных слов.

По количеству обращений, которое допускает оперативная память изучаемого процессора в течение одного машинного цикла, ее можно подразделить на память одиночного (SARAM) и двойного доступа (DARAM).

Объем адресуемой SARAM процессора TMS320F206 составляет 4K 16-битных слов, причем она может одновременно адрессоваться как программная память и как память данных. Непосредственно после сброса ПЦОС и выполнения стартовых процедур инициализации пространство ячеек памяти с адресами 8000h ÷ 8FFFh адресуется как программная память.

Объем адресуемой DARAM изучаемого ПЦОС составляет 544 16-битных слов и разбит на два блока B0 и B1. При этом ячейки блока B1 всегда адресуются как память данных, а за конфигурацию блока B1 отвечает бит CNF статусного регистра ST1. Если бит установлен, то реконфигурируемый блок отображается на адресное пространство 0FE00h ÷ 0FEFFh (0FF00h ÷ 0FFFFh) программной памяти процессора. В противном случае - на адресное пространство 0200h ÷ 02FFh (0100h ÷ 01FFh) памяти данных.

После краткого обзора особенностей организации памяти TMS320F206 следует определить основные этапы ее программного конфигурирования.

Пользовательская программа должна содержать определение точки входа программы, т.е. адреса ячейки программной памяти, с которой

процессор должен начать выполнение кода. Обычно, это определение осуществляется в два этапа:

используя директиву **.PS**, загружают счетчик команд требуемым адресом точки входа;

с помощью директивы **.ENTRY** определяют статус текущего содержимого PC как адрес точки входа программы.

После определения точки входа необходимо правильно конфигурировать память процессора. В данной лабораторной работе для этого необходимо определить статус блока памяти B0 как блока памяти данных, сбросив управляющий бит CNF статусного регистра ST1, так как лишь при таком условии целевая область памяти 0200h ÷ 02FFh будет являться блоком памяти данных.

Далее приводится пример несложной программы, демонстрирующий общие приемы программирования ПЦОС. Перед его анализом необходимо изучить список инструкций изучаемого ПЦОС (п. П.2), а также ознакомиться с описанием средства ассемблирования TASM (п. 4.4).

Пример программы

Программа, приведенная в качестве примера, осуществляет перемножение двух шестнадцатеричных чисел с помещением младшего слова результата в ячейку памяти данных.

Пример 7.3. Перемножение двух шестнадцатиразрядных целых чисел

```
.title "Example"          ; Название программы
; Определяем номер страницы памяти, в пределах которой будут размещаться операнды
#define        M_PAGE      4
; Определяем адрес целевой ячейки памяти
#define  D_ADDR        202h
; Заносим операнды в память данных
; Определяем адрес ячейки памяти данных начиная с которой будет существлено
; размещение операндов последовательные байты операндов
.ds      0200h
; Записываем первый  операнд 01h по адресу 0200h
.word    02h
; Записываем второй операнд 05h по адресу 0201h
.word    05h
; Конфигурирование и начальные установки
```

```
; Инициализируем счетчик комманд

        START: .ps      8000h

; и определяем ячейку памяти с адресом 8000h как точку входа в программу

                .entry

; Конфигурируем блок B0 DARAM для отображения в память даных путем
; сброса флага CNF регистра ST1.

                clrc    CNF
; Смысловая часть.
; Допустим, используем прямую адресацию. Следовательно необходимо загрузить номер целевой
; страницы памяти № 4 в DP. Начальный адрес соответствующий данной странице - 0200h.


                ldp     #M_PAGE

; Загружаем регистр T первым операндом. Указание нулевого смещения на
; странице памяти №4 вызовет помещение в T-регистр содержимого ячейки
; памяти с начальным адресом этой страницы, равным 0200h.


                lt      0

; Перемножаем содержимое регистра T с содержимым ячейки памяти заданной
; единичным смещением на четвертой странице памяти, т.е. имеющей адрес 201h.
; Результат операции помещается в регистр P.

                mpya    1

; Перед помещением произведения в аккумулятор необходимо обнулить биты PM статусного
; регистра ST1. Это позволит предотвратить управляемый сдвиг результата предыдущей
; операции, который осуществляется соответствующим регистром на выходе регистра P.

                spm     0

; Помещаем содержимое P-регистра в аккумулятор без сдвига.

                pac

; Помещаем адрес целевой ячейки памяти во временный регистр AR1.

                lar     AR1, #D_ADDR

; и делаем этот регистр текущим.

                mar     *, AR1

; Сохраняем содержимое аккумулятора в целевой ячейке, используя косвенную адресацию.

                sacl    *

; Заканчиваем ассемблирование

                .end
```

Указания к выполнению

Выполнение лабораторной работы осуществляется в три этапа:
программирование,
ассемблирование,
работа с отладчиком.

Программирование

На этом этапе составляется программа с использованием набора команд (инструкций) ПЦОС TMS320F206 и в соответствии с заданием на лабораторную работу.

Для последующего тестирования и отладки текст программы необходимо набрать в редакторе текстов **ASCII** (например в **Notepad** для **Windows**) и сохранить в файле с именем **BrN_3.asm** (где N – номер бригады).

Ассемблирование

Для ассемблирования полученной программы необходимо:

1. При работе в ОС Windows осуществить вызов командного процессора MS-DOS выбрав **Start → Programms → MS-DOS session** (альтернативным способом вызова является выбор **Start → Run** и ввод в появившемся диалоговом окне **command** с последующим подтверждением ввода).
2. С помощью командной строки сделать директорию ***C:\F206*** текущей.
3. Создать каталог с названием **BrN**, где N – номер бригады и переписать в него текстовый файл с программой.
4. Осуществить вызов TASM из командной строки по шаблону, описанному в п. 4.4.

Примечание. *В данном случае необходимо ввести*

tasm *-203 -k -l BrN_1.asm*

В случае успешного ассемблирования на экран будут выведены строки, как показано на рис. 7.4, и ассемблер сформирует исполняемый файл с именем **BrN_3.dsk,** содержащий объектные коды для ПЦОС, а также листинг файл с расширением **BrN_3.dsk.** В противном случае ассемблер сгенирирует сообщение об ошибке (рис. 7.5).

При этом наряду с информацией о количестве ошибок ассемблер указывает номера строк, в которых, была допущена ошибка.

Это существенно упрощает поиск и исправление неточностей на этапе ассемблирования. Особенно полезно в данном случае пользоваться листинг-файлом, который отличается от сходного asm-файла присутствием в начале каждой строки ее порядкового номера и соответствующего значения программного счетчика. При этом, располагая сведениями о

номере ошибочной строки, можно достаточно просто локализовать ошибку.

Следует также отметить, что в конце каждого .lst-файла в табличной форме выводится информация об использованных в пользовательской программе символах (метка, именах вспомогательных регистров и т.п.).

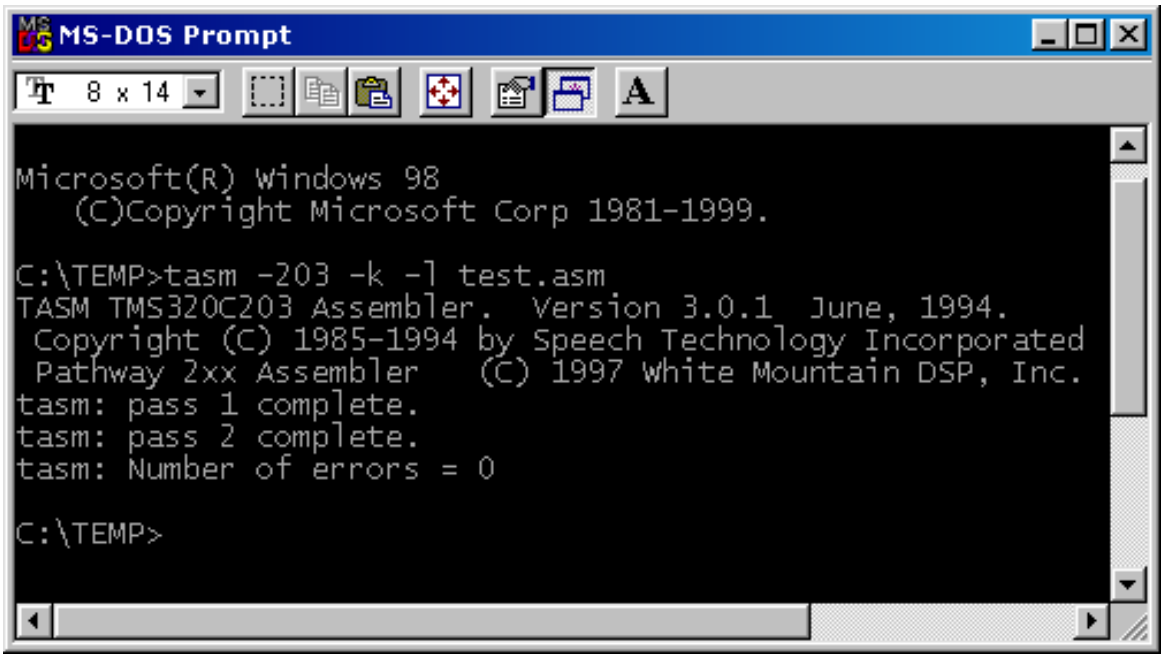


Рис. 7.4. Результат ассемблирования программы, не содержащей ошибок

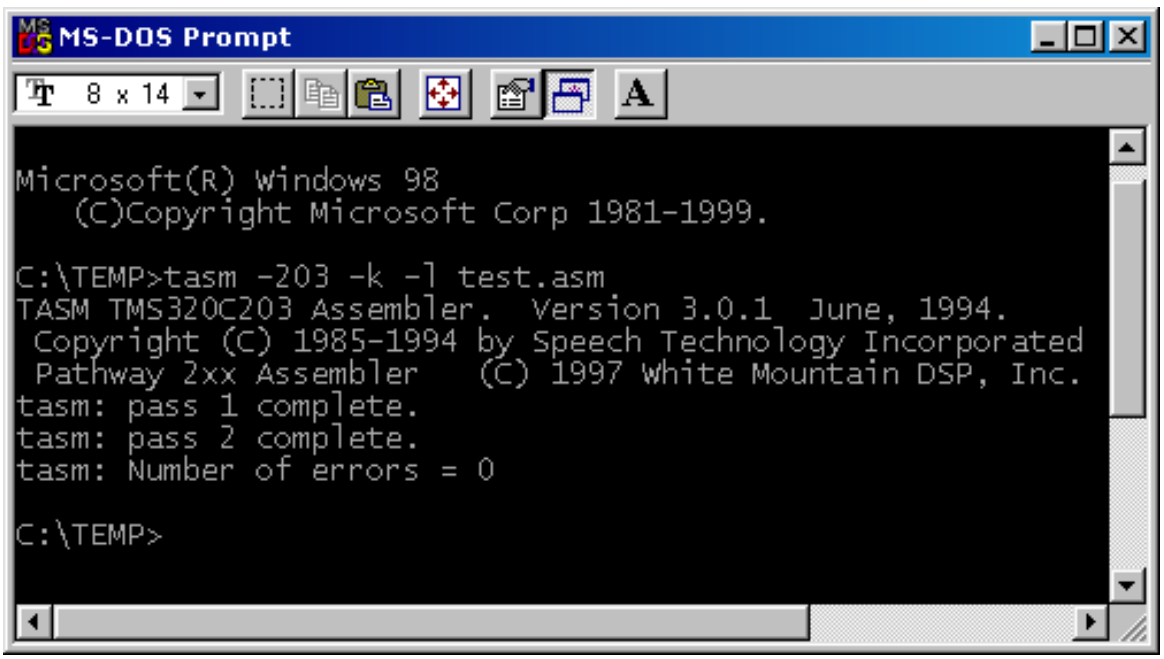


Рис. 7.5. Результат ассемблирования ошибочного файла с программой

Работа с отладчиком

Перед началом отладки, полученного в результате ассемблирования объектного кода пользовательской программы, необходимо ознакомиться с содержанием раздела *I Приложений* с целью получения информации о работе в среде отладки **Code Explorer**.

После чего рекомендуется выполнить следующую последовательность действий.

1. Включите модуль DSK.
2. Установите переключатель **Mode** в положение **Debug.** Запустите средство отладки Code Explorer двойным щелчком левой кнопки мыши по соответствующему ярлыку на “Рабочем столе” Windows.На запрос об имени порта, к которому подключен модуль DSK, введите **COM1.**
3. Осуществите загрузку объектного кода из файла **BrN_3.dsk** в программную память процессора. При этом в главном окне приложения должно появиться дочернее окно Dis-Assembly.
4. Установите точку останова на строке, следующей за строкой последней инструкции загруженной программы.
5. Осуществите “непрерывный прогон” кода нажатием кнопки **Run** в панели инструментов.
6. Воспользуйтесь средствами Code Explorer для просмотра модифицированных областей памяти данных, чтобы убедиться в правильности работы рограммы.
7. Просмотрите содержимое области программной памяти, содержащей загруженный код.
8. Воспользовавшись графическими средствами Code Explorer осуществите визуализацию таблицы исходных данных и результатов в разных окнах.
9. Закройте все окна.
10. Выполните перезагрузку объектного кода программы.
11. Воспользуйтесь возможностями пошаговой отладки, предоставляемыми Code Explorer, выполните программу в пошаговом режиме, наблюдая за изменением содержимого регистров процессора с помощью окна **CPU registers**.
12. Очистите инициализированные пользовательской программой области программной памяти и памяти данных, воспользовавшись командой **Fill Memory…** с шаблоном заполнения, соответствующим коду пустой операции 8B00h.
13. Выйдите из программы Code Explorer.
14. Выключите питане модуля DSK.

Лабораторная работа № 206-2

Задание

1. Осуществить 200 измерений мгновенных значений синусоидального напряжения с частотой 1кГц. При этом использовать первый аналоговый вход модуля DSK, синхронный последовательный порт ПЦОС TMS320F206 и память ПЦОС, начиная с адреса 0200h.
2. С помощью отладочного средства **Code Explorer** визуализировать сформированную в памяти ПЦОС последовательности выборок.

Методические указания

Для выполнения лабораторной работы необходимо иметь представление об организации системы прерываний ПЦОС TMS320F206, а также изучить работу его синхронного последовательного порта (пп. 4.5, 4.6).

Используемый в данной лабораторной работе принцип связи микропроцессора с источником аналогового сигнала (генератором синусоидальных колебаний) достаточно прост – вход приема последовательных данных **DX** микропроцессора подключен к выходу последовательных данных микросхемы кодека TLC320AC02C, входящей в состав модуля DSK. Эта микросхема выполняет аналого-цифровое преобразование поступающей с аналогового входа DSK информации.

При написании прораммы, управляющей процессом передачи данных через синхронный последовательный порт процессора очень важно корректно настроить порт. Исходя из изложенного выше принципа взаимодействия ПЦОС TMS320F206 и микросхемы TLC320AC02C следует, что его необходимо настроить на работу в режиме внешней синхронизации (импульсы тактовой и кадровой синхронизации должны иметь внешний по отношению к микропроцессору источник). В данном случае целесообразно использовать режим непрерывной передачи последовательных данных.

Для обеспечения обработки принимаемых SSP данных в соответствии с поставленным заданием следует воспользоваться генерируемым портом прерыванием RINT (для определенности нужно настроить порт так, чтобы упомянутое прерывание генерировалось при непустом приемном буфере). Для этого необходимо составить подпрограмму обработки соответствующего прерывания и поместить ее вектор в таблицу векторов прерываний микропроцессора.

Основная таблица векторов прерываний ПЦОС TMS320F206 расположена в программной памяти, начиная с адреса 00000h. Следует

подчеркнуть, что программный доступ к соответствующим ячейкам памяти запрещен и пользователь не может изменить их содержимое. Но в памяти микропроцессора предусмотрена альтернативная таблица векторов прерываний, берущая начало с адреса 08000h и имеющая ту же последовательную структуру, что и основная. Каждый вектор прерывания состоит из двух 16-битных слов и представляет собой код инструкции безусловного перехода к процедуре обработки соответствующего прерывания. Зачем же необходима такая двойственность? Просто основная таблица в качестве вектора обработки каждого из прерываний содержит инструкцию безусловного перехода на вектор соответствующего прерывания в альтернативной таблице. Таким образом, именно альтернативной таблицей в конечном счете пользователь определяет требуемые ему процедуры обработки прерываний.

Говоря об обработке прерываний процессором в режиме высокоуровневой отладки (в режиме использования отладчика высокого уровня Code Explorer), следует сделать некоторые важные замечания:

обмен данными между программой-отладчиком Code Explorer и ПЦОС осуществляется через ***асинхронный последовательный порт***, что делает ***нежелательной*** работу с этим портом в режиме отладки пользовательской программы;

для корректного функционирования отладчика необходимо при работе с прерываниями оставлять прерывание асинхронного порта ***демаскированным.***

Ниже приводится пример несложной программы, демонстрирующий общие принципы настройки и обмена данными через синхронный последовательный порт. Сначала необходимо проанализировать список инструкций изучаемого ПЦОС (Приложение П.5, табл. П.7) и принципы работы со средством ассемблирования TASM (п. 4.4).

Пример программы

Пример 7.4. Чтение синхронного последовательного порта

```
                .title          "SSP Demo"

; В процессе ассемблирования подключаем заголовочный файл, содержащий объявления часто
; инспользуемых при прорграммировании TMS320F206 меток:
; EN_RINT,  EN_TXRXINT, IMR, IOSR, SSPCR, INTM, CNF, WSGR и др.
; Убедитесь в правильности данного пути на Вашей ЭВМ!

                .include "c:\\F206\\include\\pathway.inc"

; Установка для управляющего регистра SSP при сбросе

 SSPCFG1        .set            04000h

; Установка для управляющего регистра SSP для вывода из состояния сброса:
; -       непрерывный режим передачи,
```

```
; -        прерывание RINT генерируется при непустом приемном FIFO-буфере,
; -        внешняя тактовая и кадровая синхронизация,
; -        режим  цифровой “закольцовки” деактивирован

SSPCFG2         .set            04030h

; Определение стартового адреса для ассемблирования в память данных

                .ds             200h

; Резервируем ячейку памяти для хранения приходящих с кодека данных.
; В начале программы эта ячейка используется в качестве буфера

aicData         .word    0

; Ассемблируем в программную память альтернативную таблицу прерываний

                .ps 8000h

                b               0000h           ; 00 – сброс программы-монитора
                b               0000h           ; 02 – сброс программы-монитора
                b               0000h           ; 04 – сброс программы-монитора
                b               0000h           ; 06 – сброс программы-монитора
                b               recvIsr         ; 08 – вектор прерывания RINT
                b               0000h           ; 0a – сброс программы-монитора
                b               0000h           ; 0c – сброс программы-монитора
                .space          2*16            ; 0e – зарезервировано
                b               0000h           ; 10 – сброс программы-монитора
                b               0000h           ; 12 – сброс программы-монитора
                b               0000h           ; 14 – сброс программы-монитора
                b               0000h           ; 16 – сброс программы-монитора
                b               0000h           ; 18 – сброс программы-монитора
                b               0000h           ; 1a – сброс программы-монитора
                b               0000h           ; 1c – сброс программы-монитора
                b               0000h           ; 1e – сброс программы-монитора
                b               0000h           ; 20 – сброс программы-монитора
                b               0000h           ; 22 – сброс программы-монитора
                b               0000h           ; 24 – сброс программы-монитора
                .space          2*16            ; 26 – зарезервировано
                b               0000h           ; 28 – сброс программы-монитора
                b               0000h           ; 2a – сброс программы-монитора
                b               0000h           ; 2c – сброс программы-монитора
                b               0000h           ; 2e – сброс программы-монитора
                b               0000h           ; 30 – сброс программы-монитора
                b               0000h           ; 32 – сброс программы-монитора
                b               0000h           ; 34 – сброс программы-монитора
                b               0000h           ; 36 – сброс программы-монитора
                b               0000h           ; 38 – сброс программы-монитора
                b               0000h           ; 3a – сброс программы-монитора
                b               0000h           ; 3c – сброс программы-монитора
                b               0000h           ; 3e – сброс программы-монитора

; Начало новой секции для ассемблирования в программную память

                .ps             8050h

; Определяем точку входа в программу

                .entry
```

```
; Запрещаем все демаскированные прерывания

START:      setc        INTM

; Конфигурируем блок B0 DARAM для отображения в память даных путем сброса флага CNF
; регистра ST1

            clrc        CNF

; Загружаем в указатель страницы памяти 4

            ldp         #04h

; Инициализируем генератор состояний ожидания

            splk        #0000h, aicData
            out         aicData, WSGR

; Настройка и сброс SSP

            splk        #SSPCFG1, aicData
            out         aicData, SSPCR

; Вывод  SSP из состояния сброса и окончательные установки

            splk        #SSPCFG2, aicData
            out         aicData, SSPCR

; Настройка микросхемы кодека TLC320AC02C (содержит АЦП и ЦАП).
; Сброс микросхемы TLC320AC02C через регистр статуса асинхронного порта

            in          aicData, IOSR
            lacl        aicData

; Сброс бита IO3 регистра IOSR.

            and         #0fff7h
            sacl        aicData
            out         aicData, IOSR

; Вывод микросхемы TLC320AC02C из состояния сброса через регистр статуса асинхронного
; порта.

            in          aicData, IOSR
            lacl        aicData

; Установка бита IO3 регистра IOSR

            or          #08h
            sacl        aicData
            out         aicData, IOSR

            ldp         #0h

;  Демаскируем необходимые прерывания

            splk        #EN_RINT | EN_TXRXINT, IMR

; Цикл ожидания прихода демаскированных прерываний
```

```
LOOP:       idle
            b       LOOP

; Процедура обработки прерывания RINT

recvIsr:    ldp     #04h

; Загружаем данные из порта в ячейку памяти aicData

            in      aicData,  SDTR

            ldp     #0h

; Снова разрешаем все демаскированные прерывания

            clrc    INTM

; Объявляем статус прерывания RINT как обработанное

            splk    #CLR_RINT, IFR

; Возвращаемся в вызывающую программу

            ret

; Конец ассемблирования

            .end
```

Указания к выполнению

Выполнение лабораторной работы осуществляется в три этапа:
программирование,
ассемблирование,
работа с отладчиком.

Программирование

На этом этапе осуществляется составление программы с использованием набора команд (инструкций) ПЦОС TMS320F206 и в соответствии с заданием на лабораторную работу.

Для последующего тестирования и отладки текст программы необходимо набрать в редакторе текстов **ASCII** (например в **Notepad** для **Windows**) и сохранить в файле с именем **BrN_4.asm** (где N – номер бригады).

Ассемблирование

Для ассемблирования программы необходимо:

1. При работе в ОС Windows осуществить вызов командного процессора MS-DOS выбрав **Start** → **Programms** → **MS-DOS session** (альтернативным способом вызова является выбор **Start** → **Run** и ввод

в появившемся диалоговом окне **command** с последующим подтверждением ввода).
2. С помощью командной строки сделать каталог ***C:\F206*** текущим.
3. Создать каталог с названием **BrN**, где N – номер бригады и переписать в него текстовый файл с программой.
4. Осуществить вызов TASM из командной строки в соответствии с шаблоном, описанном в п. 4.4.

Примечание. *В данном случае необходимо ввести*

tasm *-203 -k -l BrN_2.asm*

Работа с отладчиком

Перед началом отладки полученного в результате ассемблирования объектного кода пользовательской программы необходимо ознакомиться с содержанием п. 4.3 – с отладочным средством **Code Explorer**.

Далее рекомендуется следующая последовательность действий.

1. Включите модуль DSK.
2. Подключите к первому аналоговому входу DSK выход генератора синусоидального напряжения частотой 1кГц.
3. Установите переключатель **Mode** в положение **Debug.**
4. Запустите средство отладки Code Explorer двойным щелчком левой кнопки мыши по соответствующему ярлыку на “Рабочем столе” Windows.
5. На запрос об имени порта, к которому подключен модуль DSK, введите **COM1.**
6. Осуществите загрузку объектного кода из файла **BrN_4.dsk** в программную память процессора. При этом в главном окне приложения должно появиться дочернее окно Dis-Assembly.
7. Установите точку останова на строке, следующей за строкой последней инструкции загруженной программы.
8. Осуществите “непрерывный прогон” кода нажатием кнопки **Run** в панели инструментов.
9. Воспользуйтесь средствами Code Explorer для просмотра модифицированных областей памяти данных, чтобы убедиться в правильности работы программы.
10. Просмотрите содержимое области программной памяти, содержащей загруженный код.
11. Воспользовавшись графическими средствами Code Explorer, осуществите визуализацию считанной последовательности данных.
12. Закройте все окна.
13. Выполните перезагрузку объектного кода программы.

14. Поставьте точку останова в процедуре обработки прерывания и, запустив программу нажатием кнопки **Run**, убедитесь, что SSP действительно генерирует требуемое прерывание.
15. Повторите шаг 14 несколько раз наблюдая за последовательным изменением содержимого ячеек в целевой области памяти.
16. Выйдите из программы Code Explorer.
17. Выключите питане модуля DSK.

7.3. Программирование ПЦОС TMS320C6211

Целью выполнения лабораторных работ данного раздела является ознакомление с архитектурой и методами программирования ПЦОС TMS320C6211. Изучается функциональный модуль DSK, расширяющий функциональные возможности ПЦОС TMS320C6211, содержащий АЦП и ЦАП, и подсоединенный к персональному компьютеру через параллельный порт Centronix.

В ходе выполнения работ при программировании используется среда разработки программ для ПЦОС Code Composer Studio, предназначенная для снижения трудоемкости и временных затрат при создании прикладных программ. Code Composer Studio позволяет разрабатывать программы для ПЦОС на языке Си, автоматически транслировать ее в ассемблер, а затем переводить в код, непосредственно загружаемый в ПЦОС.

Лабораторная работа № 62-1

Задание

1. Создать новый проект.
2. Добавить файлы в проект.
3. Изменить текст программы.
4. Произвести компиляцию и запустить программу.

Методические указания

Создание нового проекта

В начале при помощи “Code Composer Studio” следует создать (добавить) файл с исходным кодом к новому проекту, а также подключить к все необходимые библиотеки.

1. На первом шаге следует создать стандартными средствами Microsoft Windows новую папку с именем **BrN_5** (где N – номер бригады) в каталоге:

C:\ti\myprojects

2. Далее необходимо скопировать все файлы из каталога C:\ti\c6000\tutorial\hello1 в папку, созданную в пункте 1.
3. Из меню Windows Пуск (кнопка в нижнем левом углу) выбрать Программы – “Code Composer Studio `C6000 DSK Tools – Code Composer Studio” (или запустите программу с рабочего стола).

Примечание. *Если при запуске программы у вас появятся сообщение об ошибке, обратитесь к лаборанту!*

4. Выберите из меню “Project” позицию “New”. Далее, из открывшегося окна “Save New Project As” перейдите в рабочую папку (созданную в п. 1). Назовите проект: «BrN_5», где N соответствует номеру вашей бригады. Этот файл BrN_5.mak сохранит все настройки проекта и файлы, в него входящие.

Добавление файлов в проект

1. Выберите из верхнего меню “Project – Add Files to Project”. В появившемся окне выберите файл hello.c и нажмите на кнопку «Открыть».
2. Выберите из верхнего меню “Project – Add Files to Project”. В появившемся окне выберите файл vectors.asm и нажмите на кнопку «Открыть». Этот файл содержит команды ассемблера, необходимые, чтобы заставить пакеты выборки обслуживания прерывания **Сброса** (interrupt service fetch packets) выполнять переход к точке входа программы: c_int00. Для более сложных программ вы можете задать дополнительно вектора прерываний в файле vectors.asm или использовать установки DSP/BIOS для автоматического задания всех векторов прерывания.
3. Выберите из верхнего меню “Project – Add Files to Project”. В появившемся окне измените тип файлов на “Linker Command File (*.cmd)”, выберите файл hello.cmd и нажмите на кнопку «Открыть». Этот файл размещает секции в памяти.
4. Выберите из верхнего меню “Project – Add Files to Project”. В появившемся окне измените тип файлов на “Library Files (*.lib)”. Перейдите в директорию c:\c6000\cgtools\lib, выберите файл rts6201.lib и нажмите на кнопку «Открыть». Эта библиотека осуществляет поддержку связи в реальном времени между DSP и компьютером.

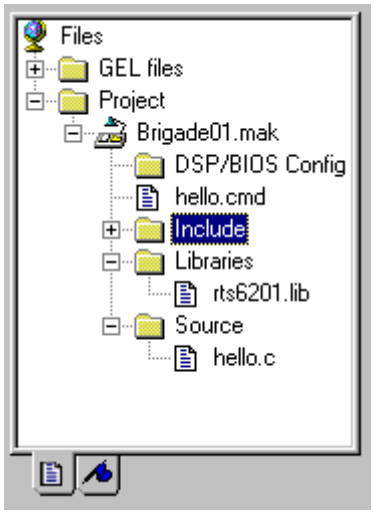


Рис. 7.6. Проект View Window

Увидеть все файлы, присоединенные к проекту, можно, используя значок «+» возле файла имени проекта, библиотек (“Libraries”), и текста программ (“Source”) в окне просмотра проекта (“Project View Window”) (рис. 7.6).

Изменение текста программы

Для просмотра программы вызовите файл ***hello.c*** из окна просмотра проекта. В отдельном окне появится текст программы на языке Си, выводящая строку «Hello World !» на экран.

Текст программы можно изменить, например, добавив строку «Brigade N is ready with a Job !», где N соответствует номеру вашей бригады.

Компиляция и запуск программы

Для компиляции и запуска программы выполните следующее.

1. Выберите из верхнего меню “Project – Rebuild All”. Среда Code Composer Studio (заново) произведет компиляцию и дизассемблирует файлы в проекте. Сообщения о происходящих процессах будут выводиться в окно внизу экрана.
2. Выберите из верхнего меню “File – Load Program”. Выберите в появившемся окне только что созданный файл “BrN_5.out” и нажмите “Открыть”. Среда Code Composer Studio загрузит откомпилированную программу в ПЦОС и откроет окно Ассемблера.
3. Выберите из верхнего меню “Debug – Run”. Программа должна запуститься и выдать в окне “Stdout” сообщения «Hello World !» и «Brigade N is ready with a Job !», где N соответствует номеру вашей бригады (рис. 7.7).
4. Нажмите Shift + F5 для остановки программы.

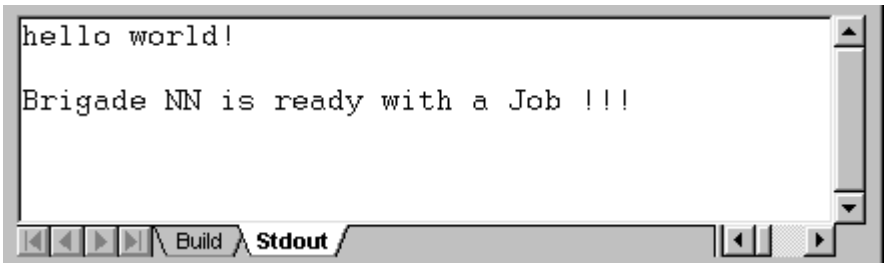


Рис. 7.7. Окно “Stdout” после успешного выполнения задания

Лабораторная работа № 62-2

Задание

1. Создать новый проект. Подготовьте новый файл для разработки проекта на языке Си. Подключите к проекту необходимые исходные файлы разработанной программы и файлы стандартных библиотек.
2. Создать массив псевдослучайных чисел со значениями не более N (номер N выбирается из таблицы вариантов заданий).
3. Вывести на экран значения элементов массива.
4. Вычислить с помощью программы и вывести на экран максимальное, минимальное или среднее значение массива.
5. Изменить значения элементов массива согласно таблице вариантов заданий.

Методические указания

В соответствии с номером вашей бригады определите размерность массива (матрицы), параметры, которые необходимо вычислить, а также диагональ, значения которой потребуется изменить (п. 5 Задания).

Таблица 7.2. Варианты задания

№ бригады	Разрядность матрицы	Параметр, который необходимо вычислить	Диагональ, значения которой необходимо изменить
1.	5x5	мин. значение	Главная диагональ

2.	6x6	макс. значение	Второстепенная диагональ
3.	7x7	среднее значение (мин+макс)/2	Главная диагональ
4.	8x8	мин. значение	Второстепенная диагональ
5.	9x9	макс. значение	Главная диагональ

Создание проекта

1. Для того чтобы рабочие файлы не «терялись» на диске, создайте в каталоге C:\ti\myprojects при помощи стандартных средств Microsoft Windows свой каталог, назвав его, например, “brN_6”, где N - номер вашей бригады.
2. При помощи команды “File – New – Source File” создайте новый файл на языке Си и запишите его в только что созданную папку.
3. При помощи команды “Project – New” создайте новый проект и запишите его в созданную Вами папку.
4. При помощи команды “Project – Add Files to Project” добавьте в проект библиотеку rts6201.lib из папки c:\c6000\cgtools\lib, не забыв в появившемся окне изменить тип файлов на “Library Files” (*.lib).
5. При помощи команды “Project – Add Files to Project” добавьте в проект файл с разработанной вами программой на языке Си.

Некоторые сведения о языке Си

Заголовок программы

Стандартная программа на языке Си начинается с заголовков, в которых указывается имя библиотеки, в которой описаны функции, применяемые в программе.

Пример 7.5. Заголовок программы на языке Си

```
# include <stdlib.h>
```

Выполнение программы начинается с выполнения функции **main {}**.

Объявление переменных

Типы данных, применяемых при программировании на языке Си представлены в табл. 7.3.

Объявление переменной может быть произведено в начале функции, в которой она используется, или непосредственно при использовании.

Таблица 7.3. Типы данных

Тип данных	Обозначение	Диапазон	Размер (бит)
Целые числа	Int	-32768 … 32767	16
Числа с плавающей запятой	float	3.4e-38 … 3.4e38	32
Знак (символ)	char	-128 … 127	8

Пример 7.6. Простейшая программа на языке Си

```
main
{
   int example = 5;
}
```

Генерация псевдослучайных значений

Пример 7.7. Присваение переменной x случайного значения из диапазона 0 .. 10

```
x = rand() % 10;
```

Оператор FOR

Оператор FOR выполняет оператор или блок операторов пока условное выражение не становится равно конечному выражению. Инициализирующее выражение выполняется один раз до проверки равенства.

Синтаксис оператора:

for (инициализация; условное выражение; конечное выражение)
оператор или блок операторов

Пример 7.8. Использование оператора For

```
for (i = 1; i <= 100; i++)
   {
   };
```

Компиляция и запуск программы

1. Выберите из верхнего меню: “Project – Rebuild All”. Среда Code Composer Studio (заново) откомпилирует и дизассемблирует файлы в проекте. Сообщения о происходящих процессах будут выводиться в окно внизу экрана.
2. Выберите из верхнего меню: “File – Load Program”; в появившемся окне – файл “xxxxxxx.out” (где xxxxxxx.c – имя вашего файла с исходным текстом программы) и нажмите “Открыть”. Среда Code Composer Studio загрузит откомпилированную программу в ПЦОС и откроет окно Ассемблера.
3. Выберите из верхнего меню: “Debug – Run”.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

1. Какой микропроцессор можно считать ПЦОС ?
2. Что такое «реальный масштаб времени» ?
3. Сформулируйте определение цифровой обработки сигналов.
4. Укажите основные достоинства и недостатки цифрового способа обработки аналоговых сигналов.
5. Перечислите типовые задачи, решаемые ПЦОС.
6. Каковы основные направления применения ПЦОС и систем на их основе ?
7. Поясните терминологическую разницу между словами «процессор» и «конроллер», а также словами «микропроцессор» и «нанопроцессор».
8. Какие функциональные, архитектурные и структурные особенности характерны для ПЦОС ?
9. Какую основную задачу решают все компоненты *базовой архитектуры ПЦОС* ?
10. Дайте краткую характеристику основных компонент *базовой архитектуры ПЦОС*.
11. В чем заключается отличие модифицированной гарвардской архитектуры от традиционной?
12. Назовите основные компании-производители ПЦОС.
13. Приведите классификацию ПЦОС компании Texas Instruments Inc.
14. В каких областях целесообразне применять ПЦОС серий TMS320C20x, TMS320C24x, TMS320C5000 и TMS320C6000 ?

15. Назовите состав и назначение внутренних шин и основных функциональных блоков ПЦОС (TMS320C2x, TMS320C2xx, TMS320C24x, TMS320C54x, TMS320C62x) компании Texas Instruments Inc.
16. Что представляет собой Flash-память и в каких ПЦОС она применяется?
17. Назовите преимущества и недостатки интеграции АЦП и ЦАП в корпус ПЦОС.
18. В чем заключаются особенности использования последовательного порта и прерываний в ПЦОС серии TMS320C2000 ?
19. Назовите специальные команды ассемблера для цифровой обработки сигналов, применяемые в ПЦОС TMS320C2000.
20. В чем заключается отличие архитектуры VelociTI от традиционной VLIW-архитектуры ?
21. Перечислите достоинства и недостатки ПЦОС с фиксированной и плавающей арифметикой.
22. Какие существуют программные средства разработки программ для ПЦОС ?
23. На каких языках можно программировать в среде Code Composer Studio ?

ЛИТЕРАТУРА

1. Букашкин С.А., Лисицын Г.Ф., Миронов В.Г. Применение цифровых процессоров обработки сигналов – магистральный путь развития современных средств телекоммуникаций // Сборник докладов 3-й межд. конф. «Цифровая обработка сигналов и ее применение». Т. 1. Москва. 2000. с. 3-4.
2. Корнеев В.В., Киселев А.В. Современные микропроцессоры. М.: Нолидж, 1998. 240 с.
3. Круг П.Г. Лабораторный практикум на основе сигнальных процессоров нового поколения. Доклады международной конференции «Информационные средства и технологии». Т. 2. М.: Издательство «Станкин». 2000. с. 242 – 245.
4. Ланнэ А.А. Цифровой процессор обработки сигналов TMS320C10 и его применения. Л.: Изд-во «ВАС», 1990.
5. Марков С. Цифровые сигнальные процессоры. М.: «Микроарт», 1996.
6. Процессоры цифровой обработки сигналов компании Texas Instruments Inc. М.: «СКАН», 1999.
7. Рабинер Л., Гоулд Б. Теория и применение цифровой обработки сигналов: Пер. с англ. под ред. Ю.Н. Александрова М.: Мир, 1978. 848 с.
8. Code Composer Studio. User’s Guide. Texas Instruments Inc. 2000.
9. Code Composer Studio. Tutorial. Texas Instruments Inc. 2000.

10. Krug P. The Digital Signal Processors. Teaching Edition. Publishing House of MPEI. Moscow. 1999. 48 p.
11. TMS320C2x DSP Starter Kit. User's Guide. Microprocessor Development Systems. Texas Instruments Inc. 1993. 108 p.
12. TMS320C2x User's Guide. Microprocessor Development Systems. Texas Instruments Inc. 1991. 682 p.
13. TMS320C2xx User's Guide. Texas Instruments Inc. January 1997.
14. http://www.ti.com

ПРИЛОЖЕНИЯ

П.1. Структура памяти ПЦОС TMS320C26

Таблица П.1. Карта программной памяти ПЦОС TMS320C26

Область	Начальный адрес	Конечный адрес	Характеристика области
1	**0000h**	**00FFh**	Содержит векторы прерываний и загрузочное ПЗУ
2	**0100h**	**0FFFh**	Зарезервирована
3	**1000h**	**F9FFh**	Внешняя память
4	**FA00h**	**FBFFh**	Блок B0
5	**FC00h**	**FDFFh**	Внешняя память
6	**FE00h**	**FFFFh**	Внешняя память

Таблица П.2. Карта памяти данных ПЦОС TMS320C26

Область	Начальный адрес	Конечный адрес	Страница памяти	Характеристика области
1	**0000h**	**0005h**	**0**	Содержит регистры, лежащиее в памяти
2	**0006h**	**005Fh**	**0**	Зарезервирована
3	**0060h**	**007Fh**	**0**	Блок B2
4	**0080h**	**01FFh**	**1-3**	Зарезервирована
5	**0200h**	**03FFh**	**4-7**	Не существует
6	**0400h**	**05FFh**	**8-11**	Блок B1
7	**0600h**	**07FFh**	**12-15**	Блок B3
8	**0800h**	**FFFFh**	**16-511**	Внешняя память

П.2. Команды (инструкции) ассемблера ПЦОС TMS320C26

Таблица П.3. Система команд ПЦОС TMS320C26

Кома-нда	Описание	Операнды
1	**2**	**3**
ABS	Получить абсолютное значение аккумулятора	*None*
ADD	Прибавить к содержимому аккумулятора со сдвигом	*dma[,shift];{ind}[,shift[,next ARP]]; #k; #lk[,shift]*
ADDC	Прибавить к содержимому аккумулятору с переносом	*dma;{ind}[,next ARP]*
ADDH	Прибавить к верхнему слову аккумулятора	*dma;{ind}[,next ARP]*
ADDK	Прибавить к содержимому аккумулятора константу	*#k*
ADDS	Прибавить к содержимому аккумулятора без знака	*dma;{ind}[,next ARP]*
ADDT	Прибавить к содержимому аккумулятора со сдвигом, определенным в T-регистре	*dma;{ind}[,next ARP]*
ADLK	Прибавить к содержимому аккумулятора длинное целое со сдвигом	*#lk[,shift]*
ADRK	Прибавить к содержимому временного регистра константу	*#k*

AND	Логическое умножение с содержимым аккумулятора	*Dma;{ind}[,next ARP] ; #lk[,shift]*
ANDK	Логическое умножение содержимого аккумулятора с константой со сдвигом	*#lk[,shift]*
APAC	Прибавить содержимое регистра P к содержимому аккумулятора	*None*
B	Безусловный переход	*pma[,{ind}[,next ARP]]*
BACC	Переход по адресу, находящемуся в аккумуляторе	*None*
BANZ	Переход при ненулевом временном регистре	*pma[,{ind}[,next ARP]]*
BBNZ	Переход при Bit ≠ 0	*pma[,{ind}[,next ARP]]*
BBZ	Переход при Bit = 0	*pma[,{ind}[,next ARP]]*
BC	Переход при наличии переноса	*Pma*
BCND	Условный переход	*pma,cond$_1$[,cond$_2$[,...]]*
BGEZ	Переход при значении содержимого аккумулятора ≥ 0	*pma[,{ind}[,next ARP]]*
BGZ	Переход при значении содержимого аккумулятора > 0	*pma[,{ind}[,next ARP]]*
BIOZ	Переход при нулевом статусе ввода/вывода	*Pma*
BIT	Тест-бит	*dma,bit code; {ind},bit code[,next ARP]*
BITT	Тест-бит, определенный в T-регистре	*dma; {ind}[,next ARP]*
BLEZ	Переход при значении содержимого аккумулятора ≤ 0	*pma[,{ind}[,next ARP]]*
BLKD	Копирование блока из памяти данных в память данных	*Dma1,dma2; dma1,{ind}[,next ARP]*

Продолжение таблицы П.3. Система команд ПЦОС TMS320C26

1	2	3
BLKP	Копирование блока из памяти программ в память данных	*Pma,dma; pma,{ind}[,next ARP]*
BLZ	Переход при значении содержимого аккумулятора < 0	*pma; pma,{ind}[,next ARP*
BNC	Переход при отсутствии переноса	*pma[,{ind}[,next ARP]]*
BNV	Переход при отсутствии переполнения	*pma[,{ind}[,next ARP]]*
BNZ	Переход при значении содержимого аккумулятора ≠ 0	*pma[,{ind}[,next ARP]]*
BV	Переход при переполнении	*pma[,{ind}[,next ARP]]*
BZ	Переход при значении содержимого аккумулятора = 0	*Pma*
CALL	Безусловный вызов подпрограммы	*pma[,{ind}[,next ARP]]*
CC	Условный вызов подпрограммы	*Pma,cond$_1$[,cond$_2$][,...]*
CLRC	Очистить управляющий бит	*control bit*
CMPL	Вычисление дополнения аккумулятора	*None*
CMPR	Сравнение временного регистра со вспомогательным регистром AR0	*CM*
CNFD	Конфигурировать блок памяти как память данных	*None*
CNFP	Конфигурировать блок памяти как программную память	*None*
DINT	Запретить прерывания	*None*
DMOV	Перемещение данных в память данных	*dma; {ind}[,next ARP]*

EINT	Разрешить прерывания	*None*
IDLE	Ожидание до появления прерывания	*None*
IN	Считать данные из порта	*dma,PA; {ind},PA[,next ARP]*
INTR	Программное прерывание	*K*
LAC	Загрузить в аккумулятор со сдвигом	*dma[,shift]; {ind}[,shift[,next ARP]]*
LACC	Загрузить в аккумулятор со сдвигом	*dma[,shift$_1$]; {ind}[,shift$_1$[,next ARP]]; #lk[,shift$_2$]*
LACK	Загрузить в аккумулятор константой	*8-bit constant*
LACL	Загрузить в младшее слово аккумулятора и очистить старшее	*dma; {ind}[,next ARP]*
LACT	Загрузить в аккумулятор со сдвигом, определенным в Т-регистре	d*ma; {ind}[,next ARP]*
LALK	Загрузить в аккумулятор длинное целое со сдвигом	*#lk[,shift]*
LAR	Загрузить во вспомогательный регистр	*AR,dma; AR,{ind}[,next ARP]; AR,#k; AR,#lk*
LARK	Загрузить во вспомогательный регистр константой	*AR, 8-bit constant*
LARP	Загрузить в указатель на вспомогательный регистр	*3-bit constant*
LDP	Загрузить в указатель на страницу памяти	*dma; {ind}[, next ARP]; #k*
LDPK	Загрузить в указатель на страницу памяти (константа)	*9-bit constant*
LPH	Загрузить в верхний Р-регистр	*dma; {ind}[, next ARP]*
LRLK	Загрузить во вспомогательный регистр (константа)	*AR, lk*
LST	Загрузить в статусный регистр	*dma; {ind}[, next ARP]*

Продолжение таблицы П.3. Система команд ПЦОС TMS320C26

1	2	3
LST	Загрузить в статусный регистр n	*#n,dma; dma; #n,{ind}[, next ARP]*
LST1	Загрузить в ST1	*dma; {ind}[,next ARP]*
LT	Загрузить в Т-регистр	*dma; {ind}[,next ARP]*
LTA	Загрузить в Т-регистр и поместить в аккумулятор продукт предыдущей операции	*dma; {ind}[,next ARP]*
LTD	Загрузить в Т-регистр, поместить в аккумулятор продукт предыдущей операции и переместить данные	*dma; {ind}[,next ARP]*
LTP	Загрузить в Т-регистр и сохранить содержимое Р-регистра в аккумуляторе	*dma; {ind}[next ARP]*
LTS	Загрузить в Т-регистр и вычесть продукт предыдущей операции	*dma; {ind}[,next ARP]*
MAC	Выполнить умножение и сохранить результат операции в аккумуляторе	*pma,dma; pma, {ind}[,next ARP]*
MACD	Выполнить умножение и сохранить результат операции в аккумуляторе с перемещением данных	*dma,pma; pma, {ind}[,next ARP]*
MAR	Изменить содержимое вспомогательного регистра	*dma; {ind}[,next ARP]*
MPY	Выполнить умножение	*dma; {ind}[,next ARP]; #k; #lk*
MPYA	Выполнить умножение и сохранить результат предыдущей операции в аккумуляторе	*dma; {ind}[,next ARP]*

MPYK	Выполнить умножение с константой	*13-bit constant*
MPYS	Выполнить умножение и вычесть продукт предыдущей операции	*dma; {ind}[,next ARP]*
MPYU	Выполнить умножение без знака	*dma; {ind}[,next ARP]*
NEG	Инвертировать аккумулятор	*None*
NMI	Немаскируемое прерывание	*None*
NOP	Пустая операция	*None*
NORM	Нормализовать содержимое аккумулятора	*None, {nd}*
OR	Логическое «ИЛИ» с содержимым аккумулятора	*Dma, {ind}[,next ARP]; #lk[,shift]*
ORK	Логическое «ИЛИ» константы с содержимым аккумулятора	*#k[,shift]*
OUT	Вывести данные в порт	*dma,PA; {ind},PA[,next ARP]*
PAC	Загрузить в аккумулятор содержимое P-регистра	*None*
POP	Загрузить в нижнее слово аккумулятора значение из вершины стека	*None*
POPD	Загрузить в память данных значение из вершины стека	*dma; {ind}[,next ARP]*
PSHD	Поместить содержимое памяти данных в стек	*dma; {ind}[,next ARP]*
PUSH	Поместить содержимое нижнего слова аккумулятора в стек	*None*
RC	Сбросить бит переноса	*None*
RET	Безусловное возвращение из подпрограммы с задержкой (выборочно)	*[D]*
RETC	Условное возвращение из подпрограммы	*$cond_1$[,$cnod_2$][,…]*

Продолжение таблицы П.3. Система команд ПЦОС TMS320C26

1	2	3
ROL	Циклический сдвиг содержимого аккумулятора влево	*None*
ROR	Циклический сдвиг содержимого аккумулятора вправо	*None*
ROVM	Сбросить режим переполнения	*None*
RPT	Повторить следующую команду	*dma; {ind}[,next ARP]; #k; #lk*
RPTK	Повторить команду количество раз, указанное константой	*#k*
RSXM	Сбросить режим знакового расширения	*None*
RTC	Сбросить тестовый флаг / флаг управления	*None*
RXF	Сбросить внешний флаг	*None*
SACH	Сохранить старшее слово аккумулятора со сдвигом	*dma[,shift]; {idn}[,shift[,next ARP]]*
SACL	Сохранить младшее слово аккумулятора со сдвигом	*dma[,shift]; {idn}[,shift[,next ARP]]*
SAR	Сохранить содержимое вспомогательного регистра	*AR,dma; AR,{ind}[,next ARP]*
SBLK	Вычесть из содержимого аккумулятора длинное целое со сдвигом	*#lk[,shift]*
SBRK	Вычесть из содержимого вспомогательного регистра константу	*#k*
SC	Установить бит переноса	*None*
SETC	Установить бит управления	*control bit*

SFL	Сдвинуть содержимое аккумулятора влево	*None*
SFR	Сдвинуть содержимое аккумулятора вправо	*None*
SOVM	Установить режим переполнения	*None*
SPAC	Вычесть содержимое P-регистра из аккумулятора	*None*
SPH	Сохранить верхнее слово P-регистра	*dma; {ind}[,next ARP]*
SPL	Сохранить нижнее слово P-регистра	*dma; {ind}[,next ARP]*
SPLK	Параллельное сохранение константы	*#lk,dma*
SPM	Установить режим сдвига на выходе P-регистра	*2-bit constant*
SQRA	Вычислить квадрат и сохранить результат предыдущей операции в аккумуляторе	*dma; {ind}[,next ARP]*
SQRS	Вычислить квадрат и вычесть результат предыдущей операции из содержимого аккумулятора	*dma; {ind}[,next ARP]*
SST	Сохранить статусный регистр	*dma; {ind}[,next ARP]*
SST	Сохранить статусный регистр n	*#n,dma; #n,{ind}[,next ARP]*
SST1	Сохранить статусный регистр ST1	*dma; {ind}[,next ARP]*
SSXM	Установить режим знакового расширения	*None*
STC	Установить тестовый флаг / флаг контроля	*None*
SUB	Вычесть из содержимого аккумулятора со сдвигом	*dma[,shift]; {ind}[,shift[,next ARP]]; #k; #lk[,shift]*
SUBB	Вычесть из содержимого аккумулятора с заемом	*dma; {ind}[,next ARP]*
SUBC	Условное вычитание	*dma; {ind}[,next ARP]*
SUBH	Вычитание из содержимого верхнего слова аккумулятора	*dma; {ind}[,next ARP]*

Окончание таблицы П.3. Система команд ПЦОС TMS320C26

1	2	3
SUBK	Вычесть константу из содержимого аккумулятора	*#k*
SUBS	Вычитание из содержимого верхнего слова аккумулятора в режиме знакового расширения	*dma; {ind}[,next ARP]*
SUBT	Вычитание из содержимого аккумулятора со сдвигом, определенным в T-регистре	*dma; {ind}[,next ARP]*
SXF	Установить внешний флаг	*None*
TBLR	Чтение таблицы	*dma; {ind}[,next ARP]*
TBLW	Запись таблицы	*dma; {ind}[,next ARP]*
TRAP	Программное прерывание	*None*
XOR	Выполнить исключительное «ИЛИ» с содержимым аккумулятора	*dma; {ind}[,next ARP]; #lk[,shift]*
XORK	Выполнить исключительное «ИЛИ» константы с содержимым аккумулятора со сдвигом	*#lk[,shift]*
ZAC	Обнулить аккумулятор	*None*
ZALH	Обнулить нижнее слово аккумулятора и загрузить старшее слово аккумулятора	*dma; {ind}[,next ARP]*
ZALR	Обнулить нижнее слово аккумулятора и загрузить старшее слово аккумулятора с округлением	*dma; {ind}[,next ARP]*
ZALS	Обнулить нижнее слово аккумулятора и загрузить старшее	*dma; {ind}[,next ARP]*

	слово аккумулятора в режиме знакового расширения	

П.3. Расположение выводов ПЦОС TMS320C26

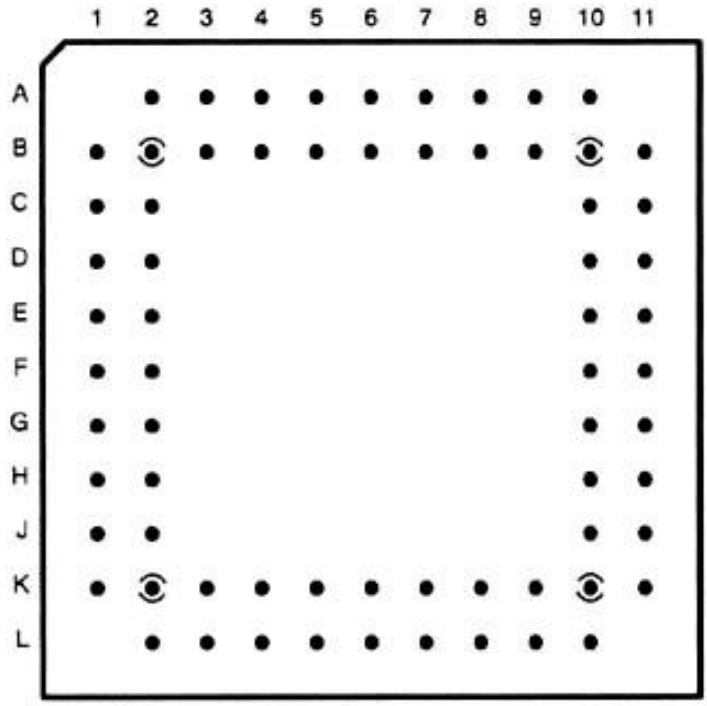


Рис. П.1. Вид сверху ПЦОС TMS320C26 в керамическом исполнении

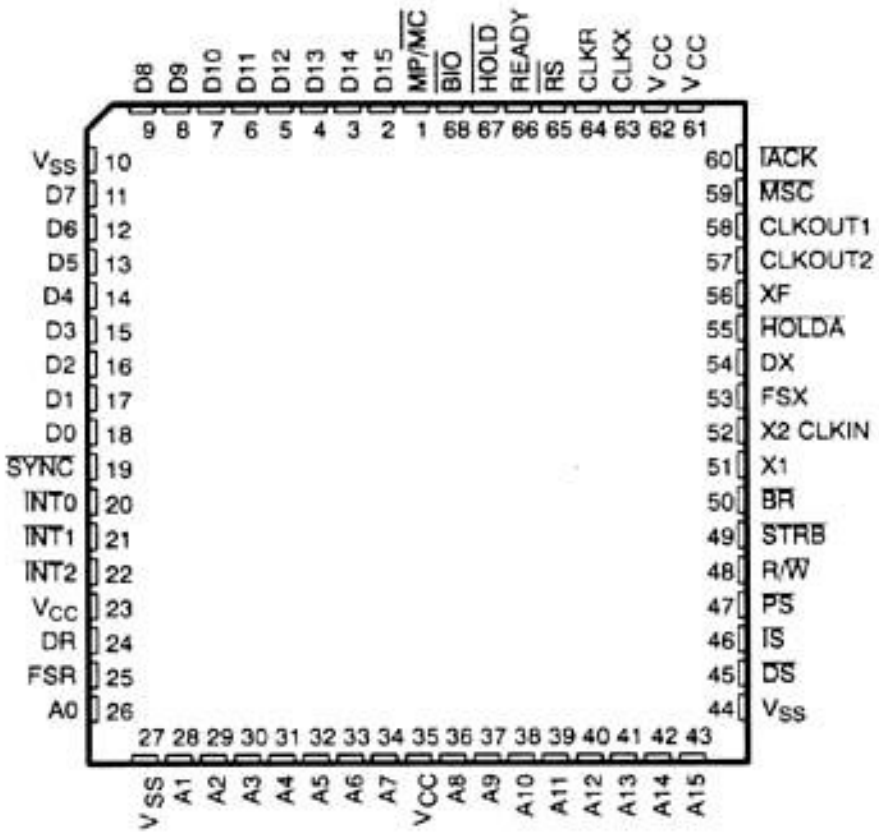


Рис. П.2. Вид сверху ПЦОС TMS320C26 в пластиковом исполнении

П.4. Структура памяти ПЦОС TMS320F206

Таблица П.4. Карта программной памяти ПЦОС TMS320F206

Область	Начальный адрес	Конечный адрес	Характеристика области
1	**0000h**	**003Fh**	Содержит векторы прерываний, 64 × 16 слов
2	**0040h**	**3FFFh**	Блок №1 внутрипроцессорной Flash памяти, 16К× 16 слов, $MP/\overline{MS} = 0$
			Внешняя память, $MP/\overline{MS} = 1$
3	**4000h**	**7FFFh**	Блок №2 внутрипроцессорной Flash памяти, 16К× 16 слов, $MP/\overline{MS} = 0$
			Внешняя память, $MP/\overline{MS} = 1$
4	**8000h**	**8FFFh**	Блок внутренней SARAM памяти, 4К × 16 слов, PON = 1

			Внешняя память, PON = 0
5	**9000h**	**FDFFh**	Внешняя память
6	**FE00h**	**FEFFh**	Блок B0 внутрипроцессорной DARAM, 256 × 16 слов, CNF = 1, отображается на область 0FF00h ÷ 0FFFFh
			Внешняя память, CNF = 0
7	**FF00h**	**FFFFh**	Блок B0 внутрипроцессорной DARAM, 256 × 16 слов, CNF = 1, отображается на область 0FE00h ÷ 0FEFFh
			Внешняя память, CNF = 0

Таблица П.5. Карта памяти данных ПЦОС TMS320F206

Область	**Начальный адрес**	**Конечный адрес**	**Характеристика области**
1	**0000h**	**005Fh**	Содержит расположенные отображаемые в память регистры и зарезервированную память, 96 × 16 слов
2	**0060h**	**007Fh**	Блок B2 внутрипроцессорной DARAM, 32 × 16 слов
3	**0080h**	**00FFh**	Зарезервирована
4	**0100h**	**01FFh**	Блок B0 внутрипроцессорной DARAM, 256 × 16 слов, CNF = 0, отображается на область 0200h ÷ 02FFh
			Зарезервирована, CNF = 1
5	**0200h**	**02FFh**	Блок B0 внутрипроцессорной DARAM, 256 × 16 слов, CNF = 0, отображается на область 0100h ÷ 01FFh
			Зарезервирована, CNF = 1
6	**0300h**	**03FFh**	Блок B1 внутрипроцессорной DARAM, 256 × 16 слов, отображается на область 0400h ÷ 04FFh
7	**0400h**	**04FFh**	Блок B1 внутрипроцессорной DARAM, 256 × 16 слов,

			отображается на область 0300h ÷ 03FFh
8	**0500h**	**07FFh**	Зарезервирована, 768 × 16 слов
9	**0800h**	**17FFh**	Внутрипроцессорная SARAM, 4K × 16 слов, DON = 1
			Внешняя память, 4K × 16 слов, DON = 0
10	**1800h**	**FFFFh**	Внешняя память, 14K× 16 слов

Таблица П.6. Карта памяти пространства ввода/вывода ПЦОС TMS320F206

Область	**Начальный адрес**	**Конечный адрес**	**Характеристика области**
1	**0000h**	**FEFFh**	Внешнее пространство ввода/вывода, 63,7 × 16 слов
2	**FF00h**	**FF0Fh**	Зарезервировано для тестовых целей, 15 × 16 слов
3	**FF10h**	**FFFFh**	Внутренние периферийные регистры процессора, 240 × 16 слов

П.5. Команды (инструкции) ассемблера ПЦОС TMS320F206

Таблица П.7. Система команд ПЦОС TMS320F206

Кома-нда	**Описание**	**Операнды**
1	**2**	**3**
ABS	Абсолютное значение аккумулятора	*None*
ADD	Прибавить к аккумулятору со сдвигом	*dma[,shift]; {ind}[,shift[,next ARP]]; #k; #lk[,shift]*
ADDC	Прибавить к аккумулятору с переносом	*dma;{ind}[,next ARP]*
ADDH	Прибавить к верхнему слову аккумулятора	*dma;{ind}[,next ARP]*
ADDK	Прибавить к аккумулятору восьмибитную константу	*#k*
ADDS	Прибавить к аккумулятору без знакового расширения	*dma;{ind}[,next ARP]*
ADDT	Прибавить к аккумулятору со сдвигом, определенным в Т-регистре	*dma;{ind}[,next ARP]*

ADLK	Прибавить к аккумулятору шестнадцатибитную константу со сдвигом	*#lk[,shift]*
ADRK	Прибавить к содержимому временного регистра восьмибитную константу	*#k*
AND	Логическое “И” с содержимым аккумулятором	*dma;{ind}[,next ARP] ; #lk[,shift]*
ANDK	Логическое “И” содержимого аккумулятора и восьмибитной константой со сдвигом	*#lk[,shift]*
APAC	Прибавить содержимое регистра P к аккумулятору	*None*
B	Безусловный переход	*pma[,{ind}[,next ARP]]*
BACC	Переход по адресу, содержащемуся в аккумуляторе	*None*
BANZ	Переход при ненулевом содержимом временного регистра	*pma[,{ind}[,next ARP]]*
BBNZ	Переход при ненулевом тестовом бите	*pma[,{ind}[,next ARP]]*
BBZ	Переход при нулевом тестовом бите	*pma[,{ind}[,next ARP]]*
BC	Переход при наличии переноса	*pma*
BCND	Условный переход	*pma,cond$_1$[,cond$_2$[,...]]*
BGEZ	Переход при аккумуляторе ≥ 0	*pma*
BGZ	Переход при аккумуляторе > 0	*pma*
BIOZ	Переход при нулевом статусе ввода/вывода	*pma*
BIT	Копировать определенный бит памяти содержимого ячейки памяти данных в тестовый бит статусного регистра ST1	*dma, bit code; {ind},bit code[,next ARP]*
BITT	Копирование определенного бита ячейки памяти данных в тестовый бит статусного регистра ST1. 4 младших бита регистра T определяют какой бит ячейки памяти данных должен быть скопирован	*dma; {ind}[,next ARP]*
BLDD	Перемещение блока из памяти данных в память данных	*#lk, dma; #lk, {ind}[, next ARP]; dma, #lk; {ind}, #lk, [, next ARP]*

Продолжение таблицы П.7. Система команд ПЦОС TMS320F206

1	2	3
BLEZ	Переход при аккумуляторе ≤ 0	*pma, pma[,{ind}[,next ARP]]*
BLKD	Копирование блока из памяти данных в память данных	*dma1,dma2; dma1,{ind}[,next ARP]*
BLKP	Копирование блока из программной памяти в память данных	*pma,dma; pma,{ind}[,next ARP]*
BLKP	Перемещение блока из программной памяти в память данных	*#pma, dma; #pma, {ind}[, next ARP]*
BLZ	Переход при аккумуляторе < 0	*pma; pma,{ind}[,next ARP*
BNC	Переход при отсутствии переноса	*pma[,{ind}[,next ARP]]*
BNV	Переход при отсутствии переполнения	*pma[,{ind}[,next ARP]]*
BNZ	Переход при аккумуляторе ≠ 0	*pma[,{ind}[,next ARP]]*
BV	Переход при переполнении	*pma[,{ind}[,next ARP]]*
BZ	Переход при аккумуляторе = 0	*pma*
CALA	Косвенный вызов подпрограммы, адрес которой определен в аккумуляторе	*None*

CALL	Безусловный вызов подпрограммы	*pma[,{ind}[,next ARP]]*
CC	Условный вызов подпрограммы	*pma,cond$_1$[,cond$_2$][,...]*
CLRC	Сбросить управляющий бит	*control bit*
CMPL	Вычисление дополнения аккумулятора	*None*
CMPR	Сравнение вспомогательного регистра с AR0. CM – двухбитная константа, определяющая тип операции сравнения (00_2 - проверка условия AR(ARP) = AR0, 01_2 - AR(ARP) < AR0, 10_2 - AR(ARP) > AR0, 11_2 - AR(ARP) ≠ AR0). Результат проверки записывается тестовый бит статусного регистра ST1. Если бит установлен – условие выполняется, если нет – нет.	*CM*
CNFD	Конфигурировать блок памяти как память данных	*None*
CNFP	Конфигурировать блок памяти как программную память	*None*
DINT	Запретить прерывания	*None*
DMOV	Перемещение данных в память данных. Содежимое адресуемой ячейки памяти данных копируется в следующий по порядку больший адрес	*dma; {ind}[,next ARP]*
EINT	Разрешить прерывания	*None*
IDLE	Ожидание до появления прерывания	*None*
IN	Считать из заданного порта PA шестнадцитибитное значение в адресуемую ячейку памяти	*dma,PA; {ind},PA[,next ARP]*
INTR	Программное прерывание. Передает управление программой по адресу программной памяти, определенному константой K (принимает значения от 0 до 32)	*K*
LAC	Зарузить аккумулятор со сдвигом	*dma[,shift]; {ind}[,shift[,next ARP]]*
LACC	Зарузить аккумулятор со сдвигом	*dma[,shift$_1$]; {ind}[,shift$_1$[,next ARP]]; #lk[,shift$_2$]*

Продолжение таблицы П.7. Система команд ПЦОС TMS320F206

1	2	3
LACK	Зарузить аккумулятор восьмибитной константой	*8-bit constant*
LACL	Загрузить младшее слово аккумулятора и очистить старшее	*dma; {ind}[,next ARP]*
LACT	Загрузить аккумулятор со сдвигом, определенным в T-регистре. Требуемый сдвиг аккумулятора определяется четырьмя младшими регистра T	d*ma; {ind}[,next ARP]*
LALK	Загрузить аккумулятор шестнадцатибитной константой со сдвигом	*#lk[,shift]*
LAR	Загрузить вспомогательный регистр	*AR,dma; AR,{ind}[,next ARP]; AR,#k; AR,#lk*
LARK	Загрузить вспомогательный регистр восьмибитной константой	*AR, 8-bit constant*
LARP	Загрузить указатель на вспомогательный регистр	*3-bit constant*
LDP	Загрузить указатель на страницу памяти	*dma; {ind}[, next ARP]; #k*
LDPK	Загрузить указатель на страницу памяти константой	*9-bit constant*
LPH	Загрузить старшее слово P-регистра	*dma; {ind}[, next ARP]*

LRLK	Загрузить вспомогательный регистр шестнадцатибитной константой	*AR, #lk*
LST	Загрузить статусный регистр. Загружает содержимое адресуемой ячейки памяти в статусный регистр ST0	*dma; {ind}[, next ARP]*
LST	Загрузить статусный регистр n. Загружает содержимое адрессуемой памяти данных в статусный регистр STn (1 или 2)	*#n,dma; dma; #n,{ind}[, next ARP]*
LST1	Загрузить статусный регистр ST1	*dma; {ind}[,next ARP]*
LT	Загрузить Т-регистр	*dma; {ind}[,next ARP]*
LTA	Загрузить Т-регистр и поместить в аккумулятор продукт прошлой операции	*dma; {ind}[,next ARP]*
LTD	Загрузить Т-регистр и поместить в аккумулятор продукт прошлой операции и переместить данные	*dma; {ind}[,next ARP]*
LTP	Загрузить Т-регистр, сохранить содержимое Р-регистра в аккумуляторе	*dma; {ind}[next ARP]*
LTS	Загрузить Т-регистр и вычесть продукт предыдущей операции	*dma; {ind}[,next ARP]*
MAC	Перемножить содержимое ячейки памяти данных с содержимым ячейки программной памяти и поместить результат предыдущей операции (сдвинутый в соответствии с содержимым статусных битов PM) в аккумулятор	*pma,dma; pma, {ind}[,next ARP]*
MACD	Перемножить содержимое ячейки памяти данных с содержимым ячейки программной памяти и поместить результат предыдущей операции (сдвинутый в соответствии с содержимым статусных битов PM) в аккумулятор. Если адресуемая ячейка памяти данных находится в блоках B0, B1 или B2, содержимое этой ячейки копируется в следующую по порядку ячейку с большим адресом	*dma,pma; pma, {ind}[,next ARP]*
MAR	Изменить содержимое вспомогательного регистра	*dma; {ind}[,next ARP]*
MPY	Перемножить содержимое Т-регистра с содержимым адресуемой ячейки памяти с помещением результата операции в Р-регистр	*dma; {ind}[,next ARP]; #k; #lk*

Продолжение таблицы П.7. Система команд ПЦОС TMS320F206

1	2	3
MPYA	Перемножить содержимое Т-регистра с содержимым адресуемой ячейки памяти, восьмибитной или шестнадцатибитной константой с помещением результата операции в Р-регистр и сохранением результата предыдущей операции (сдвинутого в соответствии с содержимым статусных битов PM) в аккумуляторе	*dma; {ind}[,next ARP]*
MPYK	Перемножить содержимое Т-регистра с знаковой тринадцатибитной константой с сохранением результата операции в Р-регистре	*13-bit constant*
MPYS	Вычесть содержимое Т-регистра из содержимого адресуемой ячейки памяти данных, расположив результат операции в Р-регистре, с последующим вычитанием результата предыдущей операции (сдвинутый в соответствии с содержимым статусных битов PM) из содержимого аккумулятора	*dma; {ind}[,next ARP]*
MPYU	Перемножить беззнаковое содержимое регистра Т с беззнаковым содержимым адресуемой ячейки памяти данных, расположив результат операции в аккумуляторе	*dma; {ind}[,next ARP]*

NEG	Инвертировать аккумулятор	*None*
NMI	Немаскируемое прерывание	*None*
NOP	Пустая операция	*None*
NORM	Нормализовать содержимое аккумулятора	*None, {nd}*
OR	Логическое “ИЛИ” с содержимым аккумулятора	*dma, {ind}[,next ARP]; #lk[,shift]*
ORK	Логическое “ИЛИ” шестнадцатибитной константы с содержимым аккумулятора	*#k[,shift]*
OUT	Вывести шестнадцатибитной значение из ячейки памяти данных в заданный порт ввода/вывода PA	*dma,PA; {ind},PA[,next ARP]*
PAC	Загрузить аккумулятор содержимым P-регистра	*None*
POP	Загрузить нижнее слово аккумулятора верхней ячейкой стека	*None*
POPD	Загрузить ячейку памяти данных содержимым верхней ячейки стека	*dma; {ind}[,next ARP]*
PSHD	Поместить содержимое ячейки памяти данных в стек	*dma; {ind}[,next ARP]*
PUSH	Поместить содержимое нижнего слова аккумулятора в стек	*None*
RC	Сбросить бит переноса	*None*
RET	Безусловное возвращение из подпрограммы	*None*
RETC	Условное возвращение из подпрограммы	*$cond_1$[,$cnod_2$][,…]*
ROL	Циклический сдвиг содержимого аккумулятора влево	*None*
ROR	Циклический сдвиг содержимого аккумулятора вправо	*None*
ROVM	Сбросить режим переполнения	*None*
RPT	Повторить следующую команду	*dma; {ind}[,next ARP]; #k; #lk*
RPTK	Повторить команду колличество раз, указанное шестнадцатибитной константой	*#k*
RSXM	Сбросить режим знакового расширения	*None*

Продолжение таблицы П.7. Система команд ПЦОС TMS320F206

1	2	3
RTC	Сбросить тестовый бит	*None*
RXF	Сбросить внешний флаг	*None*
SACH	Сохранить старшее слово аккумулятора со сдвигом	*dma[,shift]; {ind}[,shift[,next ARP]]*
SACL	Сохранить младшее слово аккумулятора со сдвигом	*dma[,shift]; {ind}[,shift[,next ARP]]*
SAR	Сохранить содержимое вспомогательного регистра	*AR,dma; AR,{ind}[,next ARP]*
SBLK	Вычесть из содержимого аккумулятора шестнадцатиричную константу со сдвигом	*#lk[,shift]*
SBRK	Вычесть из содержимого вспомогательного регистра восьмибитную константу	*#k*
SC	Установить бит переноса	*None*
SETC	Установить указанный бит управления C, CNF, HM, INTM, OVM, SXM, TC или XF	*control bit*

SFL	Сдвинуть содержимое аккумулятора влево	*None*
SFR	Сдвинуть содержимое аккумулятора вправо	*None*
SOVM	Установить режим переполнения	*None*
SPAC	Вычесть содержимое Р-регистра из содержимого аккумулятора	*None*
SPH	Сохранить верхнее слово Р-регистра со сдвигом, определенным в статусных битах РМ	*dma; {ind}[,next ARP]*
SPL	Сохранить нижнее слово Р-регистра со сдвигом, определенным в статусных битах РМ	*dma; {ind}[,next ARP]*
SPLK	Выполнить параллельное сохранение константы. Параллельный логический блок (ПЛБ) поддерживает эту битовую манипуляцию независимо от АЛУ, так что аккумулятор остается незадействованным	*#lk,dma*
SPM	Установить режим сдвига на выходе Р-регистра. Двухбитная константа копируется в управляющее сдвигом содержимого Р-регистра РМ-поле статусного регистра ST1 и может принимать следующий значения: 00_2 - отсутствие сдвига, 01_2 - результат умножения сдвигается влево на один разряд с заполнением крайней правой позиции нулем, 10_2 - результат умножения сдвигается влево на четыре бита с заполнением крайних правых позиций нулями, 11_2 - результат умножения сдвигается на шесть бит вправо направо режиме знакового расширенияж с потерей соответствующего числа младших разрядов	*2-bit constant*
SQRA	Добавить содержимое регистра Р (со сдвигом, определенным в статусными битами РМ) к содержимому аккумулятора. После этого содержимое адресуемой ячейки памяти загружается в Т-регистр, вычисляется, возводится в степень 2 и сохраняется в регистре Р	*dma; {ind}[,next ARP]*

Окончание таблицы П.7. Система команд ПЦОС TMS320F206

1	2	3
SQRS	Вычесть содержимое регистра Р (со сдвигом, определенным в статусными битами РМ) из содержимого аккумулятора. После этого содержимое адресуемой ячейки памяти загружается в Т-регистр, вычисляется, возводится в степень 2 и сохраняется в регистре Р	*dma; {ind}[,next ARP]*
SST	Сохранить статустный регистр ST0	*dma; {ind}[,next ARP]*
SST	Сохранить статустный регистр n	*#n,dma; #n,{ind}[,next ARP]*
SST1	Сохранить статустный регистр ST1	*dma; {ind}[,next ARP]*
SSXM	Установить режим знакового расширения	*None*
STC	Установить тестовый флаг	*None*
SUB	Вычесть из содержимого аккумулятора со сдвигом	*dma[,shift]; {ind}[,shift[,next ARP]]; #k; #lk[,shift]*
SUBB	Вычесть из содержимого аккумулятора с заемом	*dma; {ind}[,next ARP]*
SUBC	Условное вычитание	*dma; {ind}[,next ARP]*
SUBH	Вычетание из содержимого верхнего слова аккумулятора	*{ind}[,next ARP]*
SUBK	Вычесть восьмибитную константу из содержимого аккумулятора	*#k*

SUBS	Вычетание из содержимого верхнего аккумулятора без знакового расширения	*dma; {ind}[,next ARP]*
SUBT	Вычетание из содержимого аккумулятора со сдвигом, определенным в Т-регистре	*dma; {ind}[,next ARP]*
SXF	Установить внешний флаг	*None*
TBLR	Чтение таблицы. Перенос слова из программной памяти в память данных, адрес которой указан в младшем слове аккумулятора	*dma; {ind}[,next ARP]*
TBLW	Чтение таблицы. Перенос слова из памяти данных в программную память, адрес которой указан в младшем слове аккумулятора	*dma; {ind}[,next ARP]*
TRAP	Программное прерывание	*None*
XOR	Выполнить “Исключительное ИЛИ” с содержимым аккумулятора	*dma; {ind}[,next ARP]; #lk[,shift]*
XORK	Выполнить “Исключительное ИЛИ” шестнадцатибитной константы с содержимым аккумулятора со сдвигом	*#lk[,shift]*
ZAC	Обнулить аккумулятор	*None*
ZALH	Обнулить младшее слово аккумулятора и загрузить старшее	*dma; {ind}[,next ARP]*
ZALR	Обнулить младшее слово аккумулятора и загрузить старшее с округлением	*dma; {ind}[,next ARP]*
ZALS	Обнулить аккумулятор и загрузить нижнее слово без знакового расширения	*dma; {ind}[,next ARP]*

П.6. Расположение выводов корпуса ПЦОС TMS320F206

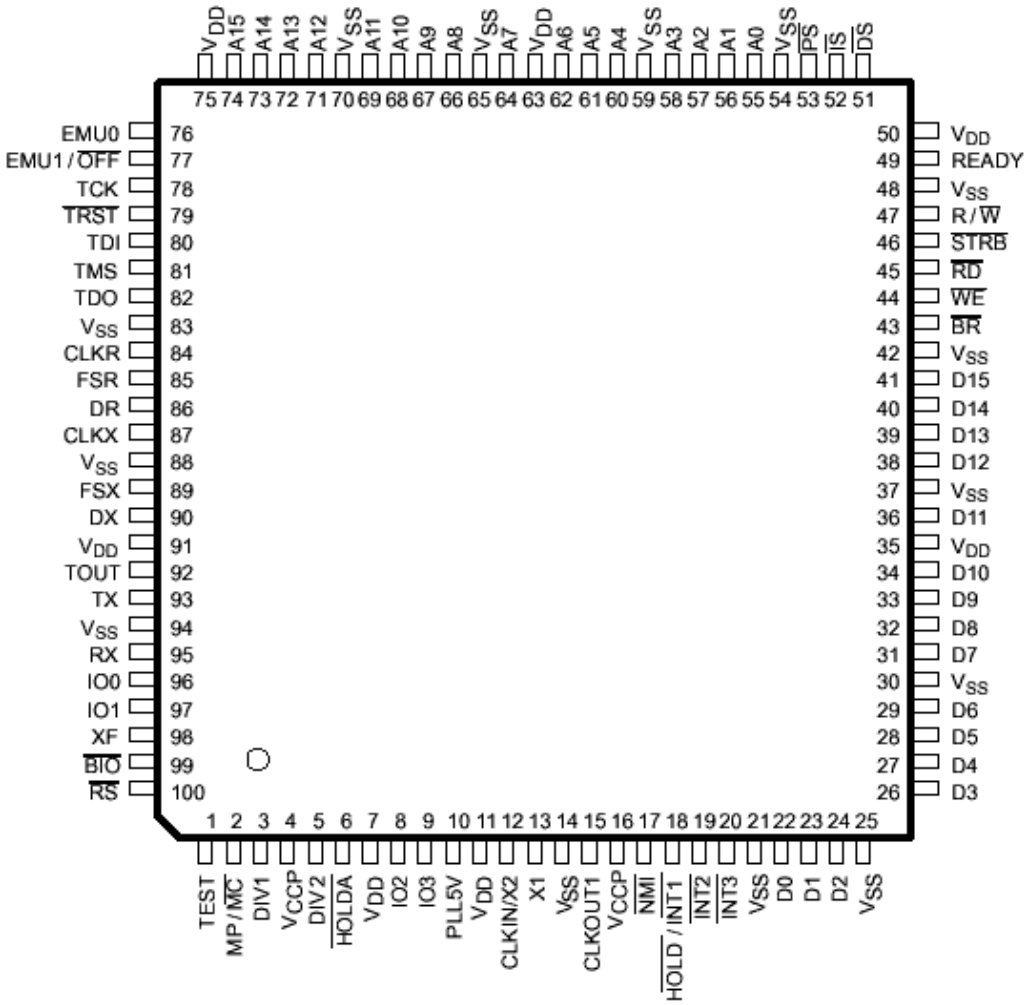


Рис. П.3. Расположение выводов ПЦОС TMS320F206 (вид сверху)

П.7. Назначение выводов ПЦОС TMS320F206

Таблица П.8. Назначение выводов ПЦОС TMS320F206

Сигнал	NN	I/O/Z/ PWR/ GND	Описание
1	**2**	**3**	**4**
Шины данных и адреса			
A15 MSB **A13** **A12** **A11** **A10** **A9** **A14** **A8** **A7** **A6** **A5** **A4** **A3** **A2** **A1** **A0 (LSB)**	**74** **73** **72** **71** **70** **69** **68** **67** **66** **65** **64** **63** **62** **61** **60** **59**	**O/Z**	Параллельная адресная шина (**A15** (MSB) ÷ **A0** (LSB)). Используется для адресации внешней программной памяти (память данных) или устройств ввода/вывода. Находится в высокоимпедансном состоянии при активном низком уровне $\overline{\mathbf{OFF}}$
D15 MSB **D14** **D13** **D12** **D11** **D10** **D9** **D8** **D7** **D6** **D5** **D4** **D3** **D2** **D1** **D0 LSB**	**41** **40** **39** **38** **37** **36** **35** **34** **33** **32** **31** **30** **29** **28** **27** **26**	**O/Z**	Параллельная шина данных (**D15** (MSB) ÷ **D0** (LSB)). Используется для передачи данных между TMS320F206 и внешней программной памятью (памятью данных) или устройствами ввода/вывода. Находится в высокоимепедансном состоянии, когда не осуществляется вывод ($\frac{\mathbf{R}}{\overline{\mathbf{W}}}$) или при наличии активного низкого уровня на $\overline{\mathbf{RS}}$ или $\overline{\mathbf{HOLD}}$. Переходят в высокоимпедансное состояние при активном низком уровне $\overline{\mathbf{OFF}}$
Сигналы управления			
$\overline{\mathbf{PS}}$ $\overline{\mathbf{IS}}$ $\overline{\mathbf{DS}}$	**53** **52** **51**	**O/Z**	Сигналы выбора адрессуемого пространства (память данных, программная память, устройство ввода/вывода). Имеет высокий уровень, пока для сообщения со внешним адресным пространством принят низкий уровень. Находится в высокоимпедансном состоянии при активном низком уровне $\overline{\mathbf{OFF}}$

Продолжение таблицы П.8. Назначение выводов ПЦОС TMS320F206

1	2	3	4
READY	**49**	**I**	Вход готовности к приему данных. Подача активного высокого уровня на этот вход говорит о том, что внешнее устройство готово к передаче данных. Если устройство не готово (**READY** = 0), TMS320F206 ждет один цикл и проверяет **READY** снова. Если **READY** не используется, на него должен подаваться активный высокий уровень
$\frac{\mathbf{R}}{\overline{\mathbf{W}}}$	**47**	**O/Z**	Сигнал чтения/записи. Показывает направление передачи данных при взаимодействии с внешним устройством. Обычно в режиме чтения этот вход находится в состоянии “1”. При выполнении записи имеет низкий уровень сигнала. Находится в высокоимпедансном состоянии при активном низком уровне $\overline{\mathbf{OFF}}$
$\overline{\mathbf{RD}}$	**45**	**O/Z**	Сигнал выбора операции чтения, в активном состоянии инициирует внешний цикл чтения. Активный уровень на этом выводе появляется при выполнении любых операций чтения внешней программной памяти, памяти данных или устройства ввода/вывода. Переходит в высокоимпедансное состояние при активном низком уровне $\overline{\mathbf{OFF}}$. Данный вывод может быть запрограммирован на выполнение функций инвертированного сигнала $\frac{\mathbf{R}}{\overline{\mathbf{W}}}$ вместо $\overline{\mathbf{RD}}$. FRDN-бит регистра PMST контролирует этот выбор
$\overline{\mathbf{WE}}$	**44**	**O/Z**	Сигнал разрешения записи. Его срез говорит что внешнее устройство управляет внешней шиной данных (**D15 ÷ D0**). Это устройство может извлечь данные по фронту сигнала $\overline{\mathbf{WE}}$
$\overline{\mathbf{STRB}}$	**46**	**O/Z**	“Строб”-сигнал. Находится в состоянии “1”. Переходит в “0” для индикации внешнего цикла шины. Находится в высокоимпедансном состоянии при активном низком уровне $\overline{\mathbf{OFF}}$
Мультипроцессорные сигналы			
$\overline{\mathbf{BR}}$	**43**	**O/Z**	Сигнал запроса шины. Имеет активный высокий уровень, когда инициирован доступ к глобальной памяти данных. Переходит в высокоимпедансное состояние при активном низком уровне $\overline{\mathbf{OFF}}$
$\overline{\mathbf{HOLDA}}$	**6**	**O/Z**	Сигнал подтверждения захвата. Низкий уровень говорит о том, что TMS320A206 перешел в режим захвата, линии контроля данных и памяти, адресные линии контроля находятся в высокоимпедансном состоянии и доступны внешнему устройству для обеспечения доступа к локальной памяти DSP. Переходит в высокоимпедансное состояние при активном низком уровне $\overline{\mathbf{OFF}}$
XF	**98**	**O/Z**	Программируемый пользователем внешний флаговый выход используется для подачи сигнала другим процессорам при работе в микропроцессорной системе или как универсальный вывод. Переходит в высокоимпедансное состояние при активном низком уровне $\overline{\mathbf{OFF}}$
$\overline{\mathbf{BIO}}$	**99**	**I**	Вход управления переходом опрашивается с помощью команды **BIOZ**. Если на $\overline{\mathbf{BIO}}$ низкий уровень сигнала, TMS320F206 выполняет переход

Продолжение таблицы П.8. Назначение выводов ПЦОС TMS320F206

1	2	3	4
IO0 **IO1** **IO2** **IO3**	**96** **97** **8** **9**	**I/O/Z**	Входные/выходные выводы, программно управляемые с помощью контрольного регистра последовательного порта ASPCR. После сброса процессора эти выводы интерпретируются как входы. Выводы **IO0 – IO3** могут быть использованы как универсальные выводы и при контроле взаимодействия для UART. Они переходят в высокоимпедансное состояние при активном низком уровне $\overline{\textbf{OFF}}$. Вывод **IO0** также используется как выход синхронизации кадров когда синхронный последовательный порт (SSP) работает в многоканальном режиме
Операции инициализации, сброса и прерывания			
TEST	**1**	**I**	Зарезервированный входной вывод. В нормальном режиме работы на этот вход подается $\mathbf{V_{ss}}$
$\overline{\textbf{NMI}}$	**17**	**I**	Немаскируемое прерывание, внешнее, которое не может быть маскировано битом режима прерывания (INTM) или регистра масок прерываний IMR. Если на этом выходе активный низкий уровень, то микропроцессор переходит по соответствующему вектору прерывания. Если этот вывод не используется он должен быть соединен с высоким потенциалом
$\overline{\textbf{HOLD}}$ $\overline{\textbf{INT1}}$	**18**	**I**	Сигналы $\overline{\textbf{HOLD}}$ и $\overline{\textbf{INT1}}$ разделяют общий вывод. Оба воспринимаются как сигналы прерываний. Если бит MODE в регистре управления прерываниями ICR равен 0, логика захвата может быть определена программным путем в комбинации с инструкцией IDLE. После сброса бит MODE регистра ICR сброшен и вывод функционирует в режиме захвата
$\overline{\textbf{INT2}}$ $\overline{\textbf{INT3}}$	**19** **20**	**I**	Внешние прерывания пользователя. Приоритет этих входов и их маски определяются регистром масок прерываний ICR и битом режима прерывания INTM. Входы $\overline{\textbf{INT2}}$ и $\overline{\textbf{INT3}}$ могут быть опрошены и сброшены с помощью регистра флагов прерываний (IFR)
MP $\overline{\textbf{MC}}$	**2**	**I**	Вход выбора режима работы TMS320F206 (микропроцесор/ микрокомпьютер). Если имеет низкий уровень (режим микрокомпьютера), происходит отображение внутрипроцессорной Flash-памяти на пространство программной памяти. В режиме микропроцессора устройство использует внепроцессорную память. Сигнал на этом входе используется только при сбросе процессора, запоминаясь в младшем бите регистра PMST
$\overline{\textbf{RS}}$	**100**	**I**	Вход сброса. Вынуждает TMS320F206 завершить исполнение программы и сбрасывает програмный счетчик в 0. Если $\overline{\textbf{RS}}$ имеет низкий уровень, исполнение начинается с нулевого адреса програмной памяти. $\overline{\textbf{RS}}$ воздействует на регистры и биты состояния

Выводы питания			
V_{CCP}	4 16	PWR	Должен быть соединен непосредственно с V_{DD}

Продолжение таблицы П.8. Назначение выводов ПЦОС TMS320F206

1	2	3	4
V_{DD}	7,11 35 50 63 75 91	PWR	Питание
V_{SS}	14 21 25 30 37 42 48 54 59 65 70 83 88 94	GND	Общий
Сигналы синхронизации			
TOUT	92	O/Z	Выход таймера. Сигнал появляется на этом выводе, если внутренний таймер процессора досчитывает до нуля. Длительность импульса совпадает с длительностью одного цикла сигнала **CLKOUT1**. Вывод переходит в высокоимпедансное состояние при активном низком уровне $\overline{OFF}$
CLK OUT1	15	O/Z	Главный выход синхронизации процессора. Период этого сигнала совпадает с машинным циклом ПЦОС. Внутренний машинный цикл процессора ограничен фронтом сигнала **CLKOUT1**. Вывод переходит в высокоимпедансное состояние при активном низком уровне $\overline{OFF}$
CLKIN X2 X1	12 13	I O	Вход синхронизации ПЦОС. Может одновременно функционировать как внешний вход генератора **CLKIN** и его внутренний вход в совокупности с выводом **X1** в качестве внутреннего выхода генератора ПЦОС
DIV1 DIV2	3 5	I	**DIV1** и **DIV2** установки для режима синхронизации. Уровни на них не должны меняться, пока на входе $\overline{RS}$ активный низкий уровень
PLL5V	10	I	Поскольку процессор TMS320F206 допускает питание лишь напряжением в 5V, на этот вход должен подаваться высокий уровень
Сигналы управления передачей данных через последовательный порт			
CLKR	83	I/O/Z	Вход синхросигнала приема. Предназначен для подачи внешнего

			синхросигнала при передаче данных от DR-входа (входа приема) в сдвиговый регистр приема последовательного порта RSR. Должен присутствовать в течение приема данных через последовательный порт. Если последовательный порт не используется и установлен бит IN0 регистра SSPCR, вывод **CLKR** можно использовать в качестве входа процессора. Если SSP используется в многоканальном режиме, то этот вывод также используется как выход кадровой синхронизации.

Продолжение таблицы П.8. Назначение выводов ПЦОС TMS320F206

1	2	3	4
CLKX	**87**	**I/O/Z**	Вывод синхронизации передачи. Предназначен для подачи внешнего синхросигнала при синхронизации передачи данных от сдвигового регистра передачи последовательного порта XSR на выход передачи данных DX. Если сброшен бит MCM синхронного регистра контроля последовательного порта SSPCR, то этот вывод выпролняет функции входа. Если этот бит установлен, то на выводе **CLKX** образуется сигнал с частотой, равной половине частоты сигнала на выходе **CLKOUT1**, т.е. он функционирует как выход. Если последовательный порт не используется, то вывод переходит в высокоимпедансное состояние при активном низком уровне $\overline{\textbf{OFF}}$. После выполнения сброса процессора работает в режиме ввода информации.
DR	**86**	**I**	Вход приема последовательных данных. Передает данные в сдвиговый регистр приема последовательного порта RSR.
DX	**90**	**O/Z**	Выход передачи последовательных данных. Осуществляет передачу последовательных данных от сдвигового регистра передачи последовательного порта XSR Вывод переходит в высокоимпедансное состояние при активном низком уровне $\overline{\textbf{OFF}}$.
FSR	**85**	**I/O/Z**	Вход импульсов кадровой синхронизации для входа приема последовательных данных. Срез импульса FSR инициирует процесс приема последовательных данных и начинает синхронизацию работы регистра RSR. Вывод переходит в высокоимпедансное состояние при активном низком уровне $\overline{\textbf{OFF}}$. Если регистр SSR используется в многоканальном режиме, FSR функционирует в качестве выхода кадровой синхронизации.
FSX	**89**	**I/O/Z**	Вход/выход импульсов кадровой синхронизации передачи последовательных данных. Срез импульса FSX инициирует процесс передачи последовательных данных и начинает синхронизацию работы регистра XSR. При последующем сбросе вывод FSX начинает работать на вход. Этот процесс может управляться программно – чтобы инициировать работу FSX на ввод, необходимо установить бит TMX статусного регистра. Вывод переходит в высокоимпедансное состояние при активном низком уровне $\overline{\textbf{OFF}}$
TX	**93**	**O/Z**	Выход асинхронной передачи данных. Вывод переходит в высокоимпедансное состояние при активном низком уровне $\overline{\textbf{OFF}}$
RX	**95**	**I**	Вход асинхронного приема данных.
Тестовые сигналы			

$\overline{\textbf{TRST}}$	**79**	**I**	Вход сигнала тестового сброса интерфейса IEEE № 1149.1 (JTAG). Если на нем высокий уровень, осуществляется системный контроль операций, выполняемых устройством. В противном случае тестовый сигнал игнорируется и процессор выполняет операции в обычном режиме. Если тестовый сигнал $\overline{\textbf{TRST}}$ не используется, необходимо применять внешнй регистр

Окончание таблицы П.8. Назначение выводов ПЦОС TMS320F206

1	2	3	4
TCK	**78**	**I**	Тестовый вход синхронизации интерфейса JTAG. Обычно на этот вход подается произвольный сигнал синхронизации с используемым на 50% периодом. По фронту этого сигнала изменения входных сигналов **TMS** и **TDI** порта тестового доступа (TAP) принимаются контроллером порта тестового доступа, регистром комманд или выбранным тестовым регистром. Изменение выходного сигнала порта тестового доступа TAP (**TDO**) происходит по срезу сигнала **TSK**
TMS	**81**	**I**	Выбор режима тестирования интерфейса JTAG. Этот сигнал записывается в контроллер TAP по срезу сигнала **TCK**
TDI	**80**	**I**	Вход данных для тестирования интерфейса JTAG. По фронту сигнала **TCK** запоминается в выбранном регистре (команд или данных)
TDO	**82**	**O/Z**	Выход данных для тестирования интерфейса JTAG. По срезу сигнала **TCK** на этом выходе появляется содержимое выбранного регистра (команд или данных). Этот выход находится в высокоимпедансном состоянии, исключая моменты сканирования данных
EMU0	**76**	**I/O/Z**	Вывод эмулятора 0. Когда на входе $\overline{\textbf{TRST}}$ низкий уровень - выполняются условия . В противном случае этот сигнал используется как прерывание от (к) эмулирующей системы и при сканировании интерфейса JTAG выполняет функции ввода и вывода.
$\frac{\textbf{EMU1}}{\overline{\textbf{OFF}}}$	**77**	**I/O/Z**	Вывод эмулятора 1. Этот вывод запрещает любой вывод информации. Если на входе $\overline{\textbf{TRST}}$ высокий уровень, вывод используется как прерывание от (к) эмулирующей системы и при сканировании интерфейса JTAG выполняет функции ввода и вывода. Активный низкий уровень на этом выводе вызывает переход всех выводов процессора в высокоимпедансное состояние. Следует отметить, что выход $\overline{\textbf{OFF}}$ используется исключительно для тестовых и эмуляционных целей, потому что для его реализации должны выполняться: $\overline{\textbf{TRST}}$ = 0, **EMU1** = 1 и $\frac{\textbf{EMU1}}{\overline{\textbf{OFF}}}$ = 0.

П.6. Расположение выводов корпуса ПЦОС TMS320C6211

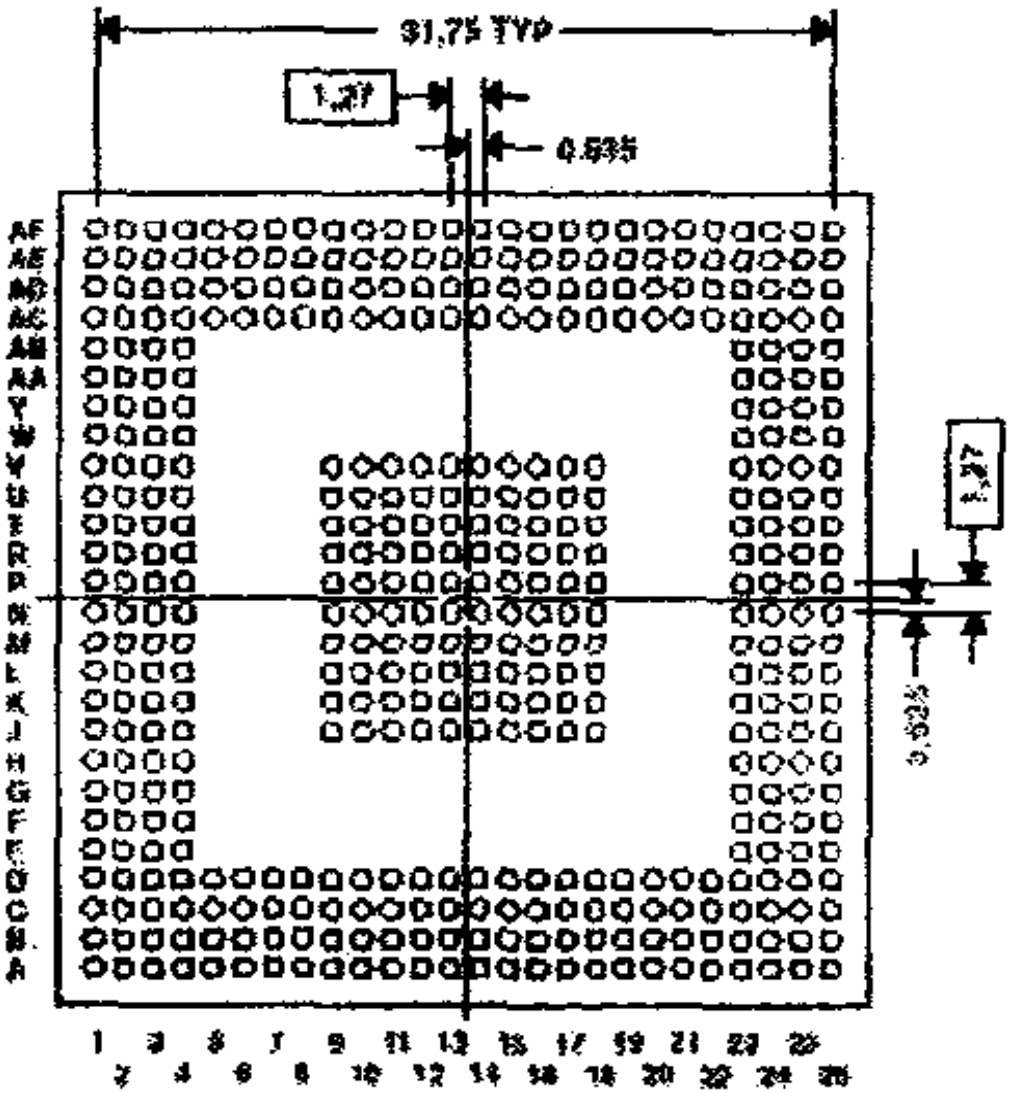


Рис. П.4. Расположение выводов ПЦОС TMS320C6211 (вид сверху)

P.G. Krug
The Digital Signal Processors. Teaching Edition. Moscow: Publishing House of MPEI. 2001 – 128 pp.

The description of the Digital Signal Processors (DSP) basic architecture and the review of TMS320 Generation of Texas Instruments Inc. are considered.
The book contains the practical course, which based on DSK TMS320C26, DSK TMS320F206 and DSK TMS320C6211 equipment. The book reflects the experience of teaching Russian and foreign students in DSPs at the Moscow Power Engineering Institute (Technical University).
For students dealing with Computer Science.

Учебное издание

Круг Петр Германович

ПРОЦЕССОРЫ ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ

Учебное пособие
по курсу “Микропроцессоры” для студентов, обучающихся
по направлению “Информатика и вычислительная техника”

Редактор издательства Н.А. Черныш
ЛР № 020528 от 05.06.97 г.

Темплан издания МЭИ 2001
Подписано к печати __.__.__. Формат 60x84/16

Печ. л. 8,0. Тираж 100. Изд. № ____.

Издательство МЭИ, 111250, Москва, Красноказарменная, д. 14
Типография ЦНИИ «Электроника»,
117415, Москва, просп. Вернадского, д. 39