

Библиотека Инженера



Семенов Б. Ю

Микроконтроллеры MSP430

первое знакомство



Архитектура

Программаторы

Система команд

Средства разработки

Практика применения



Серия «Библиотека инженера»

Б. Ю. Семенов

Микроконтроллеры MSP430:

первое знакомство

**Москва
СОЛОН-ПРЕСС
2006**

Б. Ю. Семенов

Микроконтроллеры MSP430: первое знакомство. — М.: СОЛОН-ПРЕСС, 2006. — 128 с.: ил. — (Серия «Библиотека инженера»)

ISBN 5-98003-265-7

Книга посвящена микроконтроллерам серии MSP430, которые производятся фирмой Texas Instruments. Едва ли удастся найти конкурента этим микроконтроллерам по величине потребляемого тока и производительности, если речь идет о разработке автономных малогабаритных приборов с низковольтным батарейным питанием. Книга в доступной форме поможет разобраться с архитектурой и системой команд MSP430, ориентирует читателя в многообразии «софта» для разработки программного обеспечения, расскажет о способах «прошивки» памяти MSP430, о существующих программаторах. Приведенные схемы, печатные платы и сборочные рисунки позволят собрать несколько несложных программаторов flash-памяти MSP430 самостоятельно.

Книга адресована инженерам, начинающим работать с микроконтроллерами, студентам радиотехнических специальностей вузов, специалистам, занимающимся обслуживанием и ремонтом электронной аппаратуры, радиолюбителям, а также всем тем, кто интересуется перспективной электронной техникой.

Авторский сайт в Интернете находится по адресу <http://www.radioland.mreza.ru>. После выхода этой книги из печати на нем появятся рисунки печатных плат в формате Sprint Layout 3.0 и другие дополнительные материалы по MSP430

КНИГА — ПОЧТОЙ

Книги издательства «СОЛОН-ПРЕСС» можно заказать наложенным платежом (оплата при получении) по фиксированной цене. Заказ оформляется одним из двух способов

1. Послать открытку или письмо по адресу: 123242, Москва, а/я 20.
2. Оформить заказ можно на сайте www.solon-press.ru в разделе «Книга — почтой»

Бесплатно высылается каталог издательства по почте

При оформлении заказа следует правильно и полностью указать адрес, по которому должны быть высланы книги, а также фамилию, имя и отчество получателя. Желательно указать дополнительно свой телефон и адрес электронной почты

Через Интернет вы можете в любое время получить свежий каталог издательства «СОЛОН-ПРЕСС», считав его с адреса www.solon-press.ru/kat.doc

Интернет-магазин размещен на сайте www.solon-press.ru

По вопросам приобретения обращаться ООО «АЛЬЯНС-КНИГА КТК»

Тел. (095) 258-91-94, 258-91-95, www.abook.ru

Сайт издательства «СОЛОН-ПРЕСС» www.solon-press.ru

E-mail: solon-avtor@coba.ru

ISBN 5-98003-265-7

© Семенов Б. Ю., 2006

© Макет и обложка «СОЛОН-ПРЕСС», 2006

Предисловие

Сегодня вести разговор об использовании современных микроконтроллеров очень просто — этому способствует их широкое распространение на отечественном рынке электронных компонентов, обилие книг и тематических публикаций, невероятно большое количество материалов в Интернете, общение с коллегами на электронных форумах. Но, как это ни покажется странным, в то же время разговор о микроконтроллерах невероятно сложен. Причины те же — чрезвычайная распространенность как самих микросхем, так и тематических материалов о них. Парадоксально? Ничуть! Давайте разберемся, почему так происходит сегодня.

Еще недавно эта тема была уделом лишь узкого круга специалистов, на которых с завистью поглядывали многие радиолюбители. Впрочем, и особого разнообразия типов микроконтроллеров тоже не наблюдалось: существовали громоздкие решения на базе микропроцессорных комплектов серий Z80, 580, 1810, 588... Из всего этого набора «выкристаллизовалась» чрезвычайно популярная микроконтроллерная архитектура MCS-51, разработанная фирмой Intel, а также архитектура Z86, появившаяся благодаря работам фирмы Zilog. Сегодня архитектуру MCS-51 можно считать «живой классикой» — она во многом потеряла актуальность, но поскольку в целом является открытой, многие фирмы-производители продолжают ее поддерживать, выпуская модификации микросхем с разными встроенными устройствами. Не теряет она высокого положения и среди профессиональных разработчиков исключительно из-за выработанной привычки работать со знакомым «железом».

Тема практического использования микроконтроллеров 8-разрядной структуры MCS-51 освещена достаточно: имеется ряд публикаций в журнале «Схемотехника» под названием «Микроконтроллер — это просто», вышел трехтомник [1] с одноименным названием, который сегодня можно без труда приобрести в книжных магазинах. На высоком профессиональном уровне выпущен справочник [2] по типовой архитектуре, методам программирования. Поэтому в данной книге об этой серии микроконтроллеров мы говорить не будем. Не будем мы также рассматривать архитектуры, разработанные в развитие MCS-51, а именно 16-разрядную архитектуру MCS-96, в отношении которой имеется издание [3].

Альтернативная архитектура Z86 проектировалась на базе ядра всем известного процессора Z80, который использовался в персональных компьютерах ZX-Spectrum, а позже — в автоматических определителях номера телефонных абонентов. Широкого распространения эта архитектура пока не получила, хотя и заняла достойное место среди множества современных микроконтроллерных архитектур. Следует сказать, что литературы о применении Z86 на отечественном рынке пока немного, хотя издание [4] достаточно неплохо разошлось среди отечественных читателей. Данная тема еще ждет своих авторов, и тому должно способствовать появление линейки микросхем с flash-программируемой памятью, а также пакет бесплатных инструментальных средств программирования и отладки.

Реально воплотить в жизнь слова о том, что «микроконтроллер — это просто», сделать его доступным широкой радиолюбительской аудитории впервые удалось фирме Microchip [5] в серии PIC. Фирма разработала чрезвычайно простое и понятное ядро, выпустила линейку микросхем с разными функциональными возможностями, снабдила разработчиков отличной документацией, бесплатными средствами разработки программ и «прошивки». О популярности PIC свидетельствует тот факт, что на протяжении ряда лет в журнале «Радио», практически в каждом номере, появляются радиолюбительские конструкции на основе этого микроконтроллера. Вышла отличная книга [6], в которой также приведено множество самоделок, которые легко повторить в домашних условиях. Фирма-производитель активно переводит фирменную документацию на русский язык, и ее можно «скачать» с сервера (в домене www.microchip.ru) или получить вместе с фирменным компакт-диском с того же сайта.

Мощным конкурентом архитектуры PIC, стремящимся занять лидирующие позиции, является фирма Atmel [7] с архитектурой типа AVR (AT90, ATMEGA). Все те слова, которые были сказаны в отношении PIC, можно с успехом сказать и в отношении AVR: широкий состав «линейки», наличие бесплатного отладочного программного обеспечения, простота программирования, множество практических конструкций — это все слагаемые успеха AVR у отечественного (и не только) радиолюбителя. Из книг, которые стоит рекомендовать читателю, можно назвать издание [8], содержащее множество практических конструкций и компакт-диск с необходимой документацией. В пользу AVR можно также отнести

и то, что они стремительно дешевеют, когда PIC продолжают оставаться на прежнем ценовом уровне.

Нельзя не сказать и о попытках отечественных производителей насытить рынок микроконтроллерами собственной архитектуры. Несколько лет назад на отечественном рынке появился микроконтроллер КР1878ВЕ с оригинальной архитектурой ТЕСЕЙ, разработанной специалистами ОАО «Ангстрем». Начало было положено неплохое: выпущена недорогая микросхема, предоставлена бесплатная документация, простые средства программирования и отладки. Автор этой книги приложил немало усилий к популяризации этого микроконтроллера, выпустив издания [9] и [10]. К сожалению, ОАО «Ангстрем» ничего больше не сделало для продвижения на рынок своего изделия: не была выпущена линейка, не исправлены ошибки в документации, средства программирования так и остались на начальном уровне, новых версий не вышло. Но, что самое главное, в какой-то момент этот микроконтроллер по цене сравнялся с AVR, продолжая неуклонно дорожать (конечно, сравнительно). Собственно, это обстоятельство стало «последней каплей» и оттолкнуло от него многих, кто готов был освоить архитектуру ТЕСЕЙ.

О чем еще мы не будем говорить в этой книге? Ни слова не будет сказано о 8-разрядных микроконтроллерах архитектур ST62 и ST7 (производитель — STMicroelectronics), о 16-разрядных микроконтроллерах архитектуры C167 (производитель — Infineon), о 16-разрядном семействе M16C (производитель — Mitsubishi Electric). Эти семейства микроконтроллеров хотя и встречаются в прайс-листах отечественных дистрибьюторов электронных компонентов, но применяют их не столь часто. Причин тому несколько: высокая стоимость, отсутствие полного набора бесплатных инструментальных средств, аппаратная избыточность.

Наша книга познакомит читателя с интересным семейством микроконтроллеров **MSP430**, выпускаемых фирмой **Texas Instruments**. Эти микроконтроллеры, конечно, пока менее популярны, чем PIC и AVR, но могут быть использованы там, где очень важно обеспечить минимальное потребление тока, например, в аппаратуре с низковольтным аккумуляторным или батарейным питанием. Читатели узнают о том, какие существуют инструментальные средства программирования и отладки, каковы характеристики семейства MSP430. Будет уделено внимание архитектуре и системе команд. Однако основная цель данной книги — познакомить чи-

тателя со средствами разработки, использующимися при работе с MSP430, так как именно с *их освоения и нужно начинать работу*. Для тех же читателей, кто хочет более подробно познакомиться именно с аппаратными возможностями самого MSP430, *готовится расширенное издание*, в котором эти вопросы будут рассмотрены подробнее, с примерами и практическими конструкциями. Более того, расширенное издание книги предполагается снабдить компакт-диском, на котором читатели найдут фирменную документацию, программное обеспечение, рисунки печатных плат в электронном виде и много другой полезной информации. Компакт-диск поможет продолжить знакомство с микроконтроллерами MSP430.

Б. Ю.Семенов
С.-Петербург
Январь 2006 г.

Глава 1. Что понадобится в первую очередь

В этой главе читатель познакомится с внутренним устройством микроконтроллеров семейства MSP430, с номенклатурой производимых на сегодняшний момент микросхем, с их электрическими параметрами, основными функциональными характеристиками и конструктивным исполнением.

1.1. Что такое — MSP430?

Как это ни покажется странным, но 16-разрядная RISC-архитектура MSP430 была впервые представлена фирмой Texas Instruments в 1996 году, а это значит, что сегодня, по прошествии достаточного количества времени, можно четко сказать, какую нишу заняла данная линейка программируемых компонентов на рынке электронных изделий. Изначально данные микроконтроллеры (МК) предназначались для применения в электронных устройствах с низковольтным батарейным или аккумуляторным питанием, что сделало их незаменимыми в разработке автономных датчиков. Это и неудивительно, ведь потребление микросхем составляет порядка 2 мкА при питании напряжением 2,2 В и тактовой частоте 4 кГц и порядка 250 мкА — при тактовой частоте 1 МГц. В дежурном режиме (sleep mode) микроконтроллер может потреблять 0,7...1,6 мкА. Даже при напряжении питания 3,3 В, а также максимальной тактовой частоте 8 МГц типовая величина тока потребления не превышает 2,8 мА. По оценкам специалистов, применение этих микроконтроллеров позволяет обеспечить автономную работу портативных устройств в течение 10...15 лет без смены гальванических элементов.

Еще одна интересная особенность данного семейства МК — тактирование от генератора с «часовым» кварцевым резонатором 32,768 кГц. В составе микроконтроллеров имеется PLL-устройство, позволяющее умножать тактовую частоту до величины в несколько мегагерц. Дополнительное функциональное удобство, получаемое при таком способе формирования тактирования, — переход из режима пониженного потребления в рабочий режим с

любой тактовой частотой. Если микроконтроллер находится в состоянии ожидания выполнения задачи, энергопотребление снижается, но периферийные устройства продолжают работать, и это способствует быстрому выходу МК на рабочий режим при поступлении в ядро данных для обработки.

Успех того или иного типа МК у потребителя на сегодняшний момент в значительной степени зависит от разнообразия встроенных периферийных устройств, их комбинации в одном кристалле. В каких-то случаях разработчику может понадобиться простой вариант, содержащий один-два таймера и пару портов ввода-вывода, в других случаях — посложнее, с 12-разрядным аналогово-цифровым преобразователем (АЦП), компаратором, стандартным последовательным портом ввода-вывода, таймером-счетчиком, таймером-формирователем широтно-импульсного сигнала (ШИМ), контроллером жидкокристаллического дисплея на 96 или 160 сегментов. Микроконтроллеры MSP430 предоставляют разработчику такую возможность выбора.

Выбор может также осуществляться по объему внутренней памяти. В составе самого «нижнего» МК имеется 1 кбайт постоянной (немодифицируемой) памяти ROM, 128 байт оперативной (модифицируемой) памяти RAM, и стоит такой МК около \$1 (пример — MSP430F1101). Верхнюю границу памяти МК серии MSP430 определить сложно, так как она расширяется выходом новых модификаций. Читателю рекомендуется ориентироваться на величины 60 кбайт памяти ROM и 2 кбайт памяти RAM при стоимости всего кристалла около \$10 (пример — MSP430F1491).

Семейство микроконтроллеров MSP430 в настоящее время можно условно разделить на три крупных группы: кристаллы с маркировкой MSP430x1xx представляют собой базовую группу; кристаллы с маркировкой MSP430x3xx — расширенное семейство со встроенным контроллером ЖКИ, и, наконец, кристаллы MSP430x4xx имеют «на борту» контроллер ЖКИ индикатора с расширенными возможностями. Символом «х» в данном случае обобщены разные модификации микроконтроллеров. Совсем недавно появились сообщения о том, что фирма-производитель выпустила на рынок новую линейку MSP430F20xx с измененным тактовым генератором типа VLO, позволяющую достичь потребление тока около 0,5 мкА в режиме ожидания. Правда, конструктивно эта линейка слишком миниатюрна — корпуса микроконтрол-

леров размером 4x4 мм с 14 выводами. Анонсирована также линейка MSP430FG46xx с расширенной flash-памятью до 120 кБ.

Следует также обратить внимание читателя на то, что при изучении оригинальной технической документации ему могут встретиться упоминания не только о семействе MSP430, которое фирма-производитель называет *квалификационным изделием* (qualified product), которое сопровождается технической документацией и качество которого гарантируется фирмой. Опытные партии микроконтроллеров (prototype product) имеют наименование PMS430, документация на них помечается грифом «предварительно», а сами кристаллы выпускаются партиями не более 10000 шт. Опытные партии серии PMS в некоторых источниках называются *отладочными кристаллами*, так как они имеют многократно программируемую EEPROM-память и используются для отладки программного обеспечения однократно программируемых кристаллов серии MSP430Pxx. Еще один тип кристаллов — экспериментальный (experimental product) — не тестируется фирмой-производителем на соответствие нормативной документации и производится партиями не более 500 шт. Экспериментальные кристаллы маркируются MSX430.

Система обозначений микроконтроллеров MSP430 (production code), применяемая Texas Instruments, показана на рис. 1.1.

Еще одно преимущество микроконтроллеров MSP430 — чрезвычайно простая в освоении система команд. Причем предоставляемые пользователю средства разработки программного обеспечения позволяют работать как в классическом ассемблерном коде, так и на языках высокого уровня (например, на Си).

Но мало написать хорошую программу, необходимо, особенно начинающему, попробовать эту программу в работе с конкретным «железом», отладить как программу, так и электронную схему. Чтобы облегчить выполнение такой задачи, фирма-производитель MSP430 выпускает так называемые Starter Kits (стартовые наборы), в комплект которых входят: печатная плата с установленным микроконтроллером, ЖКИ-индикатором, кнопками, контактными разъемами для подключения других электронных компонентов. Еще одно направление отладочных средств — эмуляторы — также производятся Texas Instruments. Комплект MSP-FET430 (flash emulation tool) стоимостью \$100 позволяет эмулировать микросхемы посредством интерфейса JTAG 1149.1 или последовательного интерфейса. Программное обеспечение для работы со стартовыми

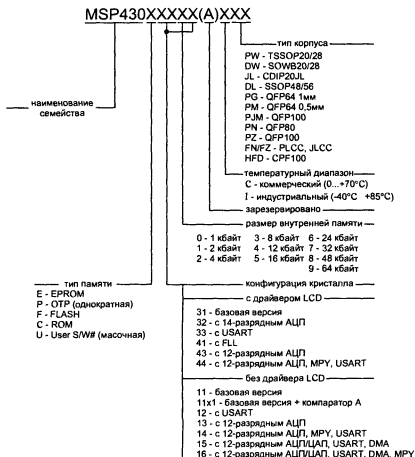


Рис. 1.1. Система обозначений MSP430

наборами и эмуляторами распространяется свободно и его можно скачивать с сайта фирмы-производителя. Существуют также эмуляторы, поставляемые другими фирмами и отличающиеся расширенными функциональными возможностями. Эти продукты стоят денег, порой немалых, поэтому начинающему работать с данным МК профессионально лучше ориентироваться на бесплатное программное обеспечение, не говоря уже о радиолюбителях.

Что еще потребуется для работы с микроконтроллером? Конечно, необходим программатор внутренней памяти МК, который позволит собственно «прошить» созданную программу в память,

«оживить» микроконтроллер. Мы уже упомянули о наличии в составе микросхем интерфейса JTAG и последовательного интерфейса, с помощью которых и может быть запрограммирована внутренняя память. Имеются несложные варианты программаторов, которые читатели смогут изготовить за несколько часов, а также автономное программное обеспечение. Обо всем этом мы расскажем в этой книге, а сейчас приступим к знакомству с архитектурой MSP430.

1.2. Знакомство с архитектурой MSP430

В этом разделе для автора возникает достаточно сложная задача — в каком ключе продолжать начатый рассказ? Ведь книгу могут читать как люди, имеющие достаточно большой опыт в области микроконтроллеров, так и начинающие, кто очень хотел бы освоить приемы работы с ними, но пока для них все МК, что называется, «на одно лицо». Соответственно, стиль подачи материала должен быть разным. Будем считать, что «крутые» профессионалы, скорее всего, обратятся к справочнику или сразу разыщут оригинальную документацию на сайте фирмы-производителя, поэтому ориентируемся на тех людей, кто о микроконтроллерах, по крайней мере, слышал, что-то знает или даже немного работал с ними. Для начинающих же полезными окажутся по ходу текста комментарии и элементарные подробности, на которых автор акцентировал внимание исключительно из-за того, чтобы всем было понятно, о чем идет речь.

Итак, архитектура MSP430. Именно с архитектуры необходимо начинать знакомство с новым типом микроконтроллеров, так как она — уникальна для каждого семейства. Изучая архитектуру, опытный разработчик сразу может оценить возможности данного типа МК, перспективы использования его в своих разработках. Хотя имеются и общие черты, а также можно проследить, откуда «растут ноги» у той или иной архитектуры, что можно считать ее предшественницей. Для MSP430 такой «предтечей» является широко известная архитектура PDP11, реализованная в отечественных вычислительных машинах ДВК, «Электроника-60», «БК-0010», ныне ставших достоянием истории.

На рис. 1.2 показана структура кристалла из «линейки» MSP430. Буква «F» в обозначении, как мы уже знаем, говорит о том, что

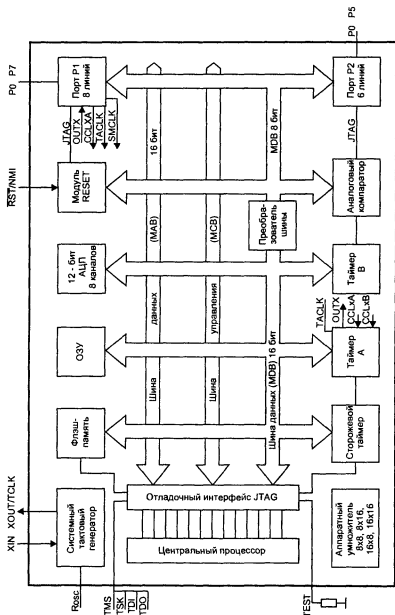


Рис. 1.2. Структура MSP430Fxxx

микроконтроллер имеет встроенную flash-память, которая может быть электрически перепрограммирована многократно. Микроконтроллеры с масочным, однократно программируемым внутренним ПЗУ и ПЗУ с ультрафиолетовым (УФ) стиранием использовать очень неудобно, поэтому мы о них говорить более не будем.

Итак, структура MSP430 использует 16-битную организацию RISC-типа, которая может производить операции не только над байтами, но и над словами (напомним, что машинное слово состоит из двух байт). Центральный процессор идентичен для всех представителей семейства и состоит из трехступенчатого конвейера инструкций, 16-разрядного арифметико-логического (ALU) устройства, 16 внутренних регистров, четыре из которых используются как счетчик команд (PC), указателя стека (SP), регистра состояния (SR) и генератора констант (CG). Счетчик команд предназначен для фиксации текущего адреса выполняемой команды, указатель стека хранит данные о так называемой стековой памяти, о которой мы будем в дальнейшем рассказывать, регистр состояния сохраняет результаты выполнения команд. Генератор констант формирует наиболее часто используемые целые числа (0000h, 0001h, 0002h, 0004h, 0008h, 0FFFFh), дополняя набор из 27 основных команд еще 24-ю, которые эмулируются ядром. Эмулированные команды не замедляют процесс обработки данных, но в ряде случаев позволяют немного сократить программный код, поскольку генератор констант является регистром центрального процессора. Остальные 12 регистров могут быть использованы на усмотрение, однако производители рекомендуют резервировать регистры R4 и R5 для отладочных целей. ALU выполняет простейшие арифметические (сложение, вычитание, сравнение, логические «И», «ИЛИ», «исключающее ИЛИ») операции. В составе микроконтроллеров модификаций MSP430F14x и MSP430F44x имеется также узел, осуществляющий 16-разрядное аппаратное умножение, однако производится эта операция не в ALU, а в периферийном устройстве. Умножитель имеет два 16-разрядных регистра для помещения операндов и три регистра — для размещения результата умножения. Умножать можно как комбинации 16-разрядных чисел, так и комбинации 8-разрядных и 16-разрядных чисел. Поддерживается умножение со знаком (MPYS) и беззнаковое умножение (MPY), знаковое умножение с накоплением (MACS) и беззнаковое умножение с накоплением (MAC).

Чтобы микроконтроллер мог извлекать из памяти данные, производить над ними операции и вновь размещать их в памяти, разработчики всегда закладывают определенный механизм адресации операндов, то есть данных, над которыми производятся в данный момент операции. Предусмотрено семь способов адресации для операндов-источников и четыре — для операндов-приемников. Время выполнения команд занимает от 1 до 6 тактов системного генератора, причем операции типа «регистр-регистр» выполняются всего за 1 такт. Самое большое время занимают двухоперандные команды с абсолютным способом адресации — 6 тактов. Более конкретно о способах адресации мы поговорим позже.

Интересной особенностью микроконтроллеров семейства MSP430 является совмещение в одном адресном пространстве памяти программ, памяти данных и регистров периферийных устройств. Это означает, что одни и те же команды могут быть использованы как для работы с памятью, так и с периферийными устройствами.

Внутренняя память типа flash может быть перепрограммирована даже тогда, когда микроконтроллер установлен непосредственно в устройстве, в котором ему и предназначено работать. Напряжение программирования памяти генерируется самим микроконтроллером, поэтому операция по «перепрошивке» может осуществляться даже в варианте батарейного питания. Память микроконтроллера сегментирована и состоит из нескольких сегментов по 512 байт (основная память), 1–2 сегментов объемом 128 байт (информационная память), а также специального загрузочного ПЗУ, в котором хранится программа загрузчика основной памяти МК с помощью процедуры последовательного доступа. Чтобы «перепрошить» МК в устройстве, необходимо «загрузить» в ОЗУ микроконтроллера программу и выполнять ее именно из этой области памяти — в момент перепрограммирования основная память станет недоступной. В сегментах информационной памяти, расположенной по адресам 01000h...0107Fh и 01080h...010FFh, могут быть сохранены особо важные константы.

Достаточно удобно наличие внутри микроконтроллера так называемого супервизора питания, формирующего сигнал системного сброса при снижении питающего напряжения ниже заданного уровня. К сожалению, супервизор питания предусмотрен только в модификации «4».

Общение с «внешним миром» у микроконтроллеров происходит с помощью портов ввода-вывода. Физически линии портов — это обычные выводы микросхемы, которые могут быть сконфигурированы отдельно на ввод или на вывод сигнала. Может быть также включена альтернативная функция порта, скажем, подключен вход АЦП или компаратора. Разные модификации микроконтроллеров имеют разное количество портов. Модификация MSP430F11xx имеет два порта (P1 и P2), модификация MSP430F12xx — три порта (P1...P3), модификации MSP430F13x, MSP430F14x, MSP430F4xx — целых шесть портов (P1...P6). В наиболее простых модификациях MSP430 порт P2 имеет всего 6 линий. Порты P1 и P2 могут быть использованы для передачи микроконтроллеру сигналов внешних прерываний, причем прерывания, приходящие на разные линии, могут иметь уникальные вектора.

Очень важное периферийное устройство — таймер. Микроконтроллеры серии MSP430 могут иметь в своем составе один или два универсальных таймера (A и B), а также базовый таймер (basic timer), встречающийся только в модификациях MSP430F4xx. Таймер A, имеющийся во всех микроконтроллерах, содержит 16-разрядный счетчик с четырьмя рабочими режимами и тремя регистрами, выполняющими функцию «захват/сравнение». Кроме классических функций счета, захвата и сравнения, этот таймер позволяет формировать широтно-модулированный (ШИМ) сигнал. Регистры захвата/сравнения имеют функцию аппаратной поддержки полнодуплексного последовательного порта (UART) со скоростью обмена информацией от 75 до 115200 бит/с. Таймер A генерирует прерывания по переполнению, по захвату и по условию сравнения, причем достижение условий может быть сконфигурировано по положительному и отрицательному перепадам внешнего сигнала, а также по обоим фронтам.

В модификациях MSP430F13x, MSP430F14x, MSP430F43x и MSP430F44x предусмотрен второй таймер (B), содержащий три или семь модулей «захват/сравнение». Разрядность этого таймера можно установить программно из ряда: 8 бит, 10 бит, 12 бит, 16 бит. Третий — базовый — таймер выполняет иную функцию: он генерирует сигналы для работы других периферийных устройств. Зачастую в составе устройств требуется наличие часов реального времени и базовый таймер может формировать тактовый сигнал

для часов. Этот таймер конфигурируется как два 8-разрядных независимых счетчика и как один 16-разрядный.

Обязательным периферийным устройством сегодня является наличие сторожевого таймера (WDT), который может спасти устройство от «зависания» микроконтроллера. Если сторожевой таймер включен, то он начинает генерировать прерывания с вектором сброса, поэтому в работающей программе всегда необходимо периодически «обнулять» счетчик сторожевого таймера. Отметим также, что если использовать WDT по прямому назначению не предполагается, можно применить его как обычный интервальный таймер, сделав «привязку» к внутренним тактовым сигналам ACLK или SMCLK и задав один из восьми возможных коэффициентов деления.

Традиционным периферийным устройством сегодня также считается аналоговый компаратор. Опорное напряжение задается программно, установкой соответствующего коэффициента внутреннего делителя. Когда сигнал на выходе компаратора меняет свое значение, микроконтроллер опознает прерывание с соответствующим вектором.

Асинхронный последовательный аппаратный порт может быть использован в многопроцессорной системе. Наличием такого порта могут «похвастаться» модификации MSP430F12xx, MSP430F13x, MSP430F14x, MSP430F43x и MSP430F44x. Работа по обмену информацией может вестись в двух режимах — асинхронном (UART) и синхронном (SPI). В синхронном режиме поддерживается как работа в master-режиме (ведущий), так и в slave-режиме (ведомый). Новые микроконтроллеры типа MSP430F20x2 и MSP20x3 имеют модуль универсального последовательного интерфейса (USI), конфигурируемый пользователем либо под протокол шины I2C, либо под протокол SPI. В ближайшей перспективе фирмы — разработка модификаций с протоколом шины USB 2.0.

Аналогово-цифровой преобразователь (АЦП) представлен в виде двух модификаций — 12-разрядной и 10-разрядной. 12-разрядный АЦП имеется в составе микроконтроллеров MSP430Fx3x и MSP430Fx4x. Его можно настроить на 8 коммутируемых входов, выходная информация записывается в буфер емкостью 16 слов. Дополнительно АЦП обслуживают встроенный источник опорного напряжения (ИОН), датчик температуры и устройство отключения при снижении напряжения питания. Специально устройство автосканирования позволяет работать АЦП без участия централь-

ного процессора, в этом случае результат автоматически заносится в буфер. Модификация MSP430F1xx2, имеющая «на борту» 10-разрядный АЦП, появилась в номенклатуре микроконтроллеров в 2002 году. Отличие этой модификации от описанной ранее состоит в том, что здесь АЦП имеет прямой доступ к оперативной памяти МК. Встроенный контроллер прямой передачи данных (DTC) высвобождает центральный процессор и позволяет повысить скорость обработки данных до 50 раз по сравнению с другими модификациями микроконтроллеров этой серии.

Последнее устройство, которое мы упомянем в этом разделе, — драйвер управления жидкокристаллическим дисплеем (ЖКИ). Это устройство полностью автономно, то есть имеет собственную память, в которой хранит отображаемую информацию. Устройство само формирует последовательность управляющих сигналов ЖКИ, поэтому дополнительных компонентов не потребуется. Допускается управление 160-сегментным индикатором.

Следует также упомянуть о специализированных микроконтроллерах архитектуры MSP430, которые фирма-производитель разработала для применения в счетчиках электроэнергии. Это линейка MSP430FE43xx, на основе которой можно создавать многотарифные счетчики, позволяющие принести некоторую экономию семейному бюджету.

Итак, в корпусах этой линейки размещены на одном кристалле и микроконтроллер, и система обработки аналоговых сигналов. До настоящего момента многотарифные счетчики приходилось строить на основе нескольких микросхем: в такой набор входили микроконтроллер, flash-память, драйвер жидкокристаллического табло, часы реального времени и источник питания. Теперь ничего этого не нужно — в микроконтроллер MSP430 встроены все необходимые узлы. Кроме того, разработана уникальная система защиты от сбоев. По оценкам специалистов фирмы-производителя, применение этих МК позволяет на 80% эффективнее разрабатывать многотарифные счетчики, по сравнению с классическим вариантом из набора микросхем.

Применение MSP430 позволяет превратить квартирный счетчик в серьезный прибор контроля за потреблением электроэнергии: в его состав легко ввести измерители пиковых значений тока и напряжения, частоты сети, сравнения тока в фазном и нулевом проводе, реактивной мощности, коэффициента мощности нагру-

зок. Точность измерения величин составит около 0,1%, что на порядок выше, чем у обычного механического счетчика.

1.3. Как «прошить» MSP430

Решение задачи, вынесенной в подзаголовок, авторы большинства книг о программируемых компонентах обычно относят в конце, полагая, что сначала необходимо подробно изучить архитектуру, систему команд, номенклатуру микроконтроллеров, средства разработки программного обеспечения. Мы несколько изменим эту классическую последовательность и начнем подробное знакомство именно с задачи «прошивки» внутренней памяти микроконтроллеров.

Чтобы превратить безжизненную микросхему микроконтроллера в работоспособный элемент электронной схемы, выполняющий заданные функции, ее нужно «прошить», или, говоря более строго, разместить в памяти специально разработанную программу, которая будет этим микроконтроллером управлять. Но для осуществления этой конкретной задачи в руках разработчика должно быть несколько необходимых составляющих. Во-первых, исправный микроконтроллер, то есть такой, в котором отсутствует повреждение внутренней структуры, не отломаны выводы и в целом корпус. Во-вторых, персональный компьютер, оснащенный операционной системой WINDOWS. В-третьих, специальная программа поддержки, которая управляет процессом «прошивки». И, в-четвертых, аппаратный программатор, физически связывающий микроконтроллер и компьютер.

Микроконтроллеры, выпускаемые разными фирмами, осуществляют процесс своего программирования тоже по-разному. Иными словами, если читатель работал с МК семейства ATMEGA, выпускаемого фирмой «Atmel», то это не значит, что он сможет напрямую использовать свои наработки для «прошивки» MSP430. Каждое семейство микроконтроллеров имеет свои интерфейсы программирования, свои аппаратные программаторы, свое программное обеспечение поддержки. Но, к счастью, общие подходы к «прошивке» сохраняются для всех типов и семейств, поэтому тем, кто уже на практике сталкивался с микроконтроллерами, будет очень легко читать этот раздел.

Общая схема «прошивки» микроконтроллеров семейства MSP430 показана на рис. 1.3. Обычно программатор подключается к персональному компьютеру (ПК) с помощью порта принтера (LPT) или последовательного порта (COM). Существуют также варианты подключения через USB-интерфейс, но пока они достаточно редки, поэтому мы о них говорить подробно не будем.



Рис. 1.3. Общая схема «прошивки» MSP430

Программатор подключается к микроконтроллеру (МК) с помощью собственного интерфейса программирования, причем в данном случае может быть использовано два варианта таких интерфейсов. Первый называется JTAG-интерфейсом и позволяет, кроме простой «прошивки», осуществлять внутрисхемную отладку программы микроконтроллера. Второй интерфейс имеет условное наименование BSL (bootstrap loader interface). Этот интерфейс использует встроенную на этапе промышленного изготовления МК программу-загрузчик, которая формирует протокол последовательного обмена данными.

Вначале поговорим о JTAG-интерфейсе, основные сведения о котором приведены в фирменной документации SLAA0149. Это интерфейс имеется во всех микроконтроллерах MSP430 и состоит из нескольких сигнальных линий, минимальное количество которых равно четырем (не считая «общего» схемы и линии напряжения питания). Микроконтроллеры, размещенные в 20- или в 28-выводном корпусе, имеют также пятую линию TEST, которая включает JTAG-интерфейс подачей на соответствующий вывод микросхемы логической единицы. Так сделано потому, что данные микроконтроллеры имеют очень мало выводов, что не позволяет постоянно задействовать JTAG. С помощью линии TEST включается так называемая *альтернативная функция* портов ввода-вывода. Полнофункциональный режим порта устанавливается при подаче на вывод TEST нулевого уровня, что должно быть сделано при установке микроконтроллера в устройство, в котором он должен работать.

Функции выводов, названия и типы основных сигналов JTAG-интерфейса приведены в табл. 1.1.

Таблица 1.1

Основные сигналы интерфейса JTAG

Вывод	Тип	Назначение
TMS	входной	Контроль состояния JTAG-интерфейса
TCK	входной	Тактовый вход JTAG-интерфейса
TDI	входной	Вход данных JTAG-интерфейса/Вход TCLK
TDO	выходной	Выход данных JTAG-интерфейса
TEST	входной	Включение JTAG (микросхемы 20 и 28 выводов)

Еще один дополнительный сигнал, который может встретиться в схемах программаторов, — это сигнал TCLK. Данный сигнал представляет собой тактовую последовательность при загрузке данных в память МК и тактирования центрального процессора. Обычно дополнительного вывода для подачи сигнала TCLK не предусматривается и на эту функцию нагружается вывод TDI. Вообще-то отдельный вывод TCLK раньше существовал в виде альтернативной функции вывода XOUT, но оказалось удобнее совместить эту функцию с выводом TDI. В настоящее время вся flash-линейка MSP430 выпускается именно в такой конфигурации, соответствующим образом скорректировано и программное обеспечение.

У микроконтроллеров с малым числом выводов обычно для доступа к JTAG-интерфейсу используются линии порта 1 (port 1). Типовое расположение сигналов для этих МК показано в табл. 1.2.

Таблица 1.2

Альтернативные функции порта 1

Линия порта 1	Линия JTAG
P1.4	TCK
P1.5	TMS
P1.6	TDI/TCLK
P1.7	TDO

Обмен по JTAG-интерфейсу осуществляется с помощью *макросов доступа* (JTAG access macros). Макрос IR_SHIFT осуществляет загрузку 8-битовой JTAG-инструкции в регистр IR процессора. Макрос DR_SHIFT16 загружает 16-битовые данные в регистр DR по линии TDI и в то же самое время выдает по линии TDO данные, помещенные в этот регистр в предыдущем цикле. Таким образом, возможен обмен данными с микроконтроллером как в режиме программирования, так и в режиме внутрисхемной отладки, когда пользователь оперативно получает информацию о текущем состоянии процессора и периферийных устройств.

Макрос задержки DELAY должен быть задействован, когда необходимо сохранить полученные результаты некоторое количество времени, например, если пользователь работает с программным обеспечением, проводит анализ полученных результатов. При выполнении макроса все сигналы, поступающие к микроконтроллеру и получаемые от него, сохраняют свою величину.

Макрос установки SetTCLK сигнала TCLK в состояние логической единицы вступает в действие при подаче высокого уровня на линию TCK и сохранении данных на линиях TMS и TDI. Макрос ClrTCLK устанавливает в логический ноль сигнал TCLK также при сохранении данных на TMS и TDI.

Подробно с диаграммами сигналов при работе макросов можно познакомиться в SLAA0149, а мы перейдем к рассказу о том, каким образом макросы доступа взаимодействуют с «начинкой» микроконтроллера. Сразу же обратим внимание на то, что макрос доступа — это лишь некая «сетка», в которой надо расставить данные соответствующим образом. Именно после расстановки данных, которые потом будут переданы в микроконтроллер, макрос доступа превращается в *коммуникационную инструкцию* (JTAG communication instruction).

Коммуникационные инструкции для работы с flash-памятью условно разделены на пять групп. Первая группа — инструкции контроля шины адресов памяти (controlling the memory address bus). Вторая группа — инструкции контроля шины данных памяти (controlling the memory data bus). Третья группа — инструкции контроля состояния центрального процессора (controlling the CPU). Четвертая группа — инструкции верификации памяти (memory verification). И, наконец, пятая группа — инструкции защиты программного обеспечения (access fuse programming). Пятая группа инструкций позволяет защитить разработанное коммерче-

ское программное обеспечение микроконтроллера от копирования и использования в промышленных масштабах «пиратами». Работает механизм защиты следующим образом. В кристалле МК предусмотрена плавкая перемычка (fuse), которую можно пережечь с помощью соответствующей инструкции, и функция JTAG-интерфейса доступа к памяти после этого отключается навсегда, то есть без возможности восстановления. Обращаем внимание читателя на это обстоятельство: лучше данной возможностью пользоваться только в крайнем случае. Кстати, не всякий программатор может обеспечить «пережигание» перемычки fuse, вследствие чего эта задача часто превращается в самостоятельную, никак не связанную с программированием внутренней памяти.

Существует еще одна возможность осуществить загрузку внутренней памяти микроконтроллера — воспользоваться интерфейсом BSL. Подробно работа с данным интерфейсом описана в SLAA096B и SLAA089A, мы же остановимся на основных моментах.

В чем преимущество BSL-интерфейса перед интерфейсом JTAG? На первый взгляд, преимуществ вообще не удастся обнаружить. BSL-интерфейс не может использоваться для внутрисхемной отладки, с его помощью также не удастся воспользоваться fuse, то есть он обладает очень ограниченным функциональным назначением, позволяя работать только с flash-памятью. Но одно весомое преимущество все-таки имеется: BSL-интерфейс может напрямую работать с последовательным COM-портом компьютера, а это значит, что даже в случае использования бесплатного программного обеспечения удастся нормально работать на платформах WINDOWS «моложе» WIN98, где непосредственный доступ к LPT-порту уже закрыт, в то время как COM-порты остаются доступными «по классике». Но «пережечь» fuse с помощью BSL-интерфейса нельзя — это можно сделать только по JTAG.

Как уже было сказано, во внутренней немодифицируемой памяти микроконтроллера имеется специальный аппаратный загрузчик, который использует для связи со средствами программирования протокол UART. Вызвать загрузчик «к жизни» можно, подав на выводы RST/NMI и TEST (или TCK) специальной управляющей сигнальной последовательности. Вывод TCK используется, если в микроконтроллере отсутствует вывод TEST. В нормальном режиме работы микроконтроллера, когда требуется запустить программное обеспечение пользователя, линия TEST всегда имеет ну-

левой уровень, такой же уровень должен быть и на линии RST/NMI в момент включения питания. После того, как произойдет перепад на линии RST/NMI к высокому уровню, возникнет прерывание с вектором 0FFFFh, откуда начнется выполнение пользовательской программы. Если в момент перепада сигнала на линии RST/NMI сигнал TEST будет иметь высокий уровень, запускается аппаратный загрузчик. Отменим также, что если микроконтроллер не имеет сигнала TEST, что является типичным для МК с обособленным JTAG, используется сигнал TCK, но в этом случае управляющий сигнал должен быть инвертирован. Диаграммы сигналов, использующиеся для запуска программ пользователя

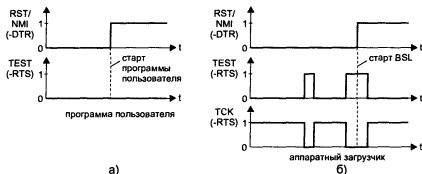


Рис. 1.4. Диаграммы запуска:

а) программа пользователя; б) аппаратный загрузчик

и аппаратного загрузчика, показаны на рис. 1.4.

Как видно из рис. 1.4, б, необходимым условием запуска BSL является наличие двух импульсов на линии TEST при нулевом сигнале на линии RST/NMI. Если количество импульсов меньше двух, BSL запущен не будет. Не сможет запуститься BSL и при активности интерфейса JTAG, а также если на выводе RST активирована альтернативная функция NMI.

После запуска аппаратного загрузчика необходимо задействовать протокол последовательного порта UART. Для этого микроконтроллер должен быть связан собственными линиями последовательного порта (TXBSL, RXBSL, RST/NMI, TEST/TCK) с линиями порта RS-232 компьютера (RX, TX, DTR, RTS) устройством согласования уровней сигналов, о котором мы расскажем чуть позже. Наиболее распространенным случаем является использование ли-

ний порта P1 для обмена данными по BSL: в семействе MSP430F1xx линия P1.1 задействуется на передачу, линия P1.2 — на прием; в семействе MSP430F4xx на передачу включается линия P1.0, на прием — линия P1.1. Но вообще-то перед «прошивкой» конкретного микроконтроллера лучше уточнить назначение линий по оригинальной документации, чтобы не допустить досадных ошибок.

Обмен данными по BSL-интерфейсу происходит с предпочтительной скоростью 9600 бит/с в полудуплексном режиме, пакетами по 8 бит, с контролем четности (even) и одним стоповым битом, по стандартному протоколу SSP. После приема информации микроконтроллером осуществляется обратная передача кода подтверждения прохождения данных (acknowledge). Разработчики не рекомендуют менять скорость обмена данными, так как в противном случае могут возникнуть проблемы с синхронизацией при работе с flash-памятью. Оправданно повысить скорость обмена при программировании кристаллов с большим объемом внутренней памяти, так как это сократит общее время загрузки. Например, для микроконтроллера типа MSP430F149 программирование 60 кбайт памяти на скорости 9600 бит/с займет примерно 80 секунд, на скорости 19200 бит/с — 43 секунды, на скорости 38400 бит/с — 24 секунды.

Читателю следует знать, что аппаратные загрузчики имеют разные версии. Разные загрузчики имеют разные возможности, но базовый функциональный набор у всех BSL-загрузчиков одинаковый. Версия 1.10 «прошивается» в кристаллы типа MSP430F11x, MSP430F13x, MSP430F13x, MSP430F11x1, версия 1.30 — в кристаллы типа MSP430F41x, MSP430F11x, MSP430F11x1, версия 1.40 — в кристаллы типа MSP430F12x, версия 1.50 — в кристаллы типа MSP430F43x и MSP430F44x, версия 1.60 — в кристаллы типа MSP430F12x2, MSP430F43x, MSP430F44x.

Интерфейс BSL имеет защищенные (protected) и незащищенные (unprotected) функции. Набор функций немного меняется от версии к версии, но незначительно. К незащищенным функциям относятся: ввод пароля доступа к защищенным функциям, полное стирание внутренней памяти программ микроконтроллера, передача версии загрузчика (в версиях 1.5 и 1.6), изменение скорости обмена данными по интерфейсу (в версии 1.6.). Защищенные функции включают в себя: чтение сегмента памяти, запись сегмента памяти, стирание сегмента памяти, контрольную проверку чистоты памяти (в версиях 1.5 и 1.6), установку программного

счетчика и старт программы пользователя. Смысл защищенных функций следующий. Чуть ранее мы говорили о том, что в контроллере может быть включена защита (fuse) от несанкционированного доступа к программному обеспечению, которая отключает JTAG-интерфейс, но сохраняет доступ по BSL. Однако в случае включенной защиты доступ к защищенным функциям становится возможным только через пароль, который хранится в адресном пространстве FFE0h-FFFFh. При запуске другого сеанса BSL-доступа придется вновь использовать пароль для доступа к защищенным функциям. В табл. 1.3 приведены наименования функций и их английские транскрипции. При защите от считывания микроконтроллеров, имеющих линию TEST, пропадает возможность перезаписи программ не только через JTAG, но и через BSL, так как в обоих режимах линия TEST используется для входа в режим программирования.

Таблица 1.3

Функции BSL

Функция BSL	Транскрипция
Принять блок данных	RX data block
Принять пароль	RX Password
Стереть сегмент	Erase segment
Полное стирание	Mass erase
Проверка стирания	Erase check
Сменить скорость обмена	Change baud rate
Передать блок данных	TX data block
Передать версию BSL	TX BSL version
Загрузить программный счётчик	Load PC

Данные, поступающие в микроконтроллер по BSL-интерфейсу, обрабатываются микроконтроллером немедленно, то есть поступивший байт данных сразу же размещается в памяти, к моменту начала приема следующего байта операция размещения завершается. Это — несомненно удобно, так как нет необходимости буферизировать данные, накапливать их в промежуточной памяти. Но есть и недостаток: в случае нарушения связи (сбой на линии или нарушение временных требований по передаче пакетов данных) придется повторно запускать BSL-интерфейс и повторять процесс

передачи. Если по команде Load PC в регистр процессора R0 загружается адрес (в пределах всего адресного пространства памяти МК), начинается выполнение программы с этого адреса, а обмен по BSL прерывается.

Теперь поговорим о том, что на техническом жаргоне называется «багами» и «глюками». К сожалению, при разработке протокола обмена по BSL техническими специалистами были допущены некоторые ошибки, которые усложнили работу с этим интерфейсом. Главная ошибка, которая вкралась в версию 1.10, заключается в некорректной работе функции RX data block («принять блок данных»). Поскольку, как уже говорилось, аппаратный загрузчик размещен в памяти, недоступной для обновления ее пользователем, специалистам фирмы пришлось разработать следующую рекомендацию: при работе с BSL сразу же загружать в ОЗУ дополнительную утилиту (patch) силами аппаратного загрузчика, затем «запускать» этот «патч» со стартового адреса 0220h и работать в обычном режиме, устанавливая программный счетчик (R0) на стартовый адрес для функций RX data block и TX data block. Данный «патч» (patch.txt) размещен в архивном файле slaa096b.zip, его можно «скачать» с сайта фирмы или найти на диске в расширенной версии этой книги.

Но не все проблемы с загрузкой можно разрешить при помощи указанного выше «патча». Как пишет специалист по микроконтроллерам Сергей Борщ, имеются кристаллы, у которых BSL-интерфейс не работает вообще. Во-первых, это PMS430F11xx, во-вторых, MSP430F1121 партий выпуска 03AJX4T, 03AK16T, 05ACR4T, 05ACV2T, 05AD2LT, 05AD2XT, 05ADY4T, 05ADY5T. Микроконтроллеры MSP430F1121 всех остальных партий выпуска, а также PMS430F149 и MSP430F149 требуют корректировки аппаратного загрузчика посредством «патча».

Еще один путь решения проблемы с ошибкой загрузчика, предложенный специалистами фирмы, позволяет не только устранить ошибку и успешно «прошить» микроконтроллер, но также расширить функциональные возможности BSL-интерфейса. Этот путь заключается в полной подмене встроенного стандартного загрузчика более свежей версией, размещаемой в ОЗУ, — посредством *загружаемого BSL*. В архивном файле slaa096b.zip имеется версия загружаемого BSL, код находится в файле bl_130v.txt. На сервере фирмы доступны загрузчики bl_150s_14x.txt и bs_150s_14x.txt, размещенные в файле slaa089b.zip. Первая версия представляет собой

полный BSL-загрузчик для семейств МК типа MSP430F13x и MSP430F14x с поддержкой всех функций аппаратного загрузчика версии 1.60. Поскольку данный загрузчик занимает в ОЗУ более 1 кбайт, его удастся использовать только совместно с микроконтроллерами MSP430F1x8 и MSP430F1x9. Вторая версия — так называемый *урезанный BSL* с неполным набором функций, подходящий для всей линейки от MSP430F1x4 до MSP430F1x9. Урезанная версия не поддерживает функцию TX data block, поэтому при ее выполнении придется обращаться к стандартному аппаратному загрузчику. Общим для всех загружаемых версий является отсутствие необходимости передавать пароль — для этих загрузчиков функции нельзя разделить на защищенные и незащищенные.

Мы закончили наше краткое знакомство с принципами «прошивки» микроконтроллеров MSP430. Теперь перейдем собственно к практической части, то есть к освоению программно-аппаратных средств, разработанных для этой цели.

1.4. Прошивка МК с использованием BSL

Принципиальная схема первого программатора MSP430 на основе интерфейса BSL приведена на рис. 1.5. К разъему X1 подключается микроконтроллер, а разъем X2 подключается к СОМ-порту персонального компьютера. Программатор построен на основе микросхем с низким потреблением тока, что позволяет питать его непосредственно от компьютерного порта, причем для питания микросхемы D1 требуется двухполярное напряжение, а для D2 — однополярное положительное. Это положительное напряжение формируют сигналы с вывода 7 и вывода 4 порта, которые «заряжают» конденсаторы C4 и C5 через диоды VD1 и VD2. Сигнал с вывода 3 порта формирует отрицательное напряжение: он «заряжает» через диод VD3 конденсатор C6.

Положительное напряжение питания поступает на стабилизатор D3 с низким падением напряжения, в качестве которого использована микросхема LP2951. Можно также использовать микросхему TPS76030 или LP2980, предварительно скорректировав печатную плату, приведенную ниже. Показанный на схеме стабилизатор имеет резисторный делитель R10, R11, который задает величину выходного напряжения 3 В, которым питается программируемый микроконтроллер. На выходе стабилизатора D3

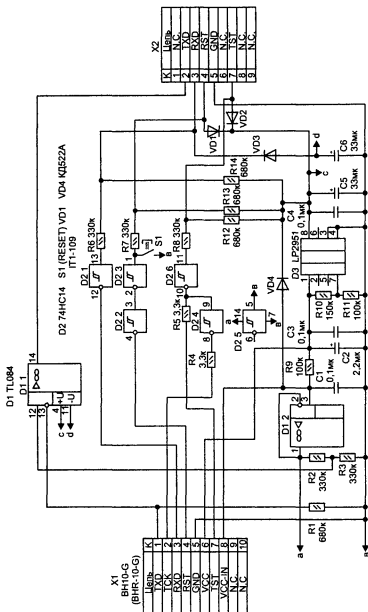


Рис. 1.5. Схема стандартного bootstrap-программатора

также установлен выходной сглаживающий фильтр C2, C3. Напряжение питания МК снимается с вывода 1 стабилизатора и поступает на вывод 6 разъема X1. Интегрирующая RC-цепь R9, C1 «затягивает» появление напряжения на выводе 1 микросхемы D1.2, включенной повторителем. С вывода 1 этой микросхемы питается микросхема D2, а также задается порог срабатывания для элемента D1.1, включенного компаратором, для чего предусмотрен резисторный делитель R2, R3. Внешнее питание можно подать на вывод 8 разъема X1, тогда резистор R9 защитит микросхему D3, а диоды VD1 и VD2 будут закрыты. Питание микросхемы D3 будет осуществляться через диод VD4.

На рис. 1.6 приведена печатная плата программатора, а собрать его можно по рис. 1.7. Элементы использованы стандартные: резисторы С2-33Н, конденсаторы К50-68 и К10-17. Разъем X1 — десятиштырьковый двухрядный, X2 — типовой серии DB или DRB.

В табл. 1.4 показаны варианты подключения разъема X1 программатора к микроконтроллерам MSP430 разных модификаций.

Таблица 1.4

Подключение программатора к МК

Вывод X1	Наименование сигнала	MSP430F11x1	MSP430F14x, MSP430F13x	MSP430F4xx
1	TXD	P1.1	P1.1	P1.0
2	TCK	—	TCK	TCK
3	RXD	P2.2	P2.2	P1.1
4	RST	RST/NMI	RST/NMI	RST/NMI
5	GND	GND	GND	GND
6	VCC	VCC	VCC	VCC
7	TST	Test	—	—
8	VCC_IN	VCC	VCC	VCC
9	Не задействован	—	—	—
10	Не задействован	—	—	—

А теперь поговорим о «софте». Чтобы «оживить» аппаратный программатор, необходимо запустить программный продукт поддержки. Достаточно большое количество фирм распространяет «софт» для выполнения данной функции, правда, не бесплатно.

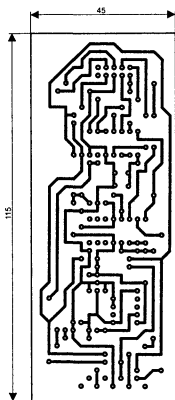


Рис. 1.6. Печатная плата

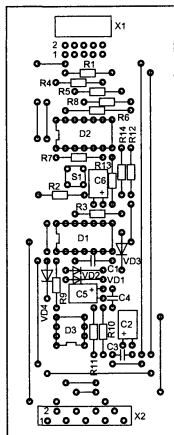


Рис. 1.7. Сборочный рисунок

Мы воспользуемся доступными и свободно распространяемыми программами. Все они обладают разной степенью удобства для пользователя, разными возможностями, но вполне пригодны для нормальной работы с MSP430.

Наиболее просто получить программу `bsldemo.exe`, которая входит в комплект оригинального файла `slaa096b.zip`. Программа представляет собой консольное DOS-приложение, которое можно «запустить» в режиме сеанса MS-DOS либо в командной строке, если в операционной системе данный сеанс отсутствует. Несмотря на то, что `bsldemo` позиционируется фирмой-производителем как

демонстрационная программа работы с BSL-интерфейсом, она имеет все необходимые функциональные возможности. Конечно, работать с командной строкой сегодня многие считают уже достаточно архаичным занятием, это все же лучше, чем полное отсутствие «софта».

Первоначальное знакомство с программой можно начинать с вызова подсказки. Это можно сделать, набрав в командной строке следующую конструкцию:

```
bsldemo.exe -h
```

Программа выведет на экран полную структуру командной строки, которая будет выглядеть следующим образом:

```
bsldemo [-h] [-c<port>] [-p<file>] [-w] [-l] [-m<num>]
[-a<file>] [-b<file>] [-f<num>] [-m<num>] [+ecpvrw] <file>
```

В данной структуре предусмотрено две разновидности дополнительных параметров. Первая, имеющая перед собой значок «-», называется опциями (options), вторая, со значком «+», — модификаторами программного потока (program-flow modifiers). Квадратные скобки указывают на то, что параметры могут отсутствовать, их набор определяется пользователем в каждом конкретном случае.

Программа имеет десять опций. Опция `-h` кратко назовет пользователю наименования параметров, их назначение и правила работы с ними в том случае, если под рукой нет описания программы.

Опция `-c<port>` определяет номер COM-порта, который задействуется для программирования. Например, конструкция `-cCOM1` заставит программу работать с портом COM1. По умолчанию задействован порт COM2, то есть, если данная опция не будет указана, программа обратится к COM2.

Опция `-p<file>` загружает в микроконтроллер пароль доступа к защищенным функциям, который должен быть размещен в текстовом файле. К примеру, конструкция `-pInt_vect.txt` заставит программу загрузить пароль из текстового файла `Int_vect.txt`.

Опция `-w` работает следующим образом: по окончании программирования внутренней памяти программа переходит в режим ожидания нажатия клавиши `<Enter>`, после чего завершает свою работу. Разработчики рекомендуют задействовать данную опцию при питании программируемого микроконтроллера от COM-порта.

Опция `-l` предназначена для комплексного выполнения функций контроля стирания памяти, программирования и верифика-

ции памяти за один сеанс «связи». К сожалению, данную опцию нельзя использовать для работы с микроконтроллерами, имеющими ошибку аппаратного загрузчика, так как сначала приходится выполнять загрузку «патча» или подгрузить в оперативную память файл альтернативного загрузчика.

Опция `-a` выполняет загрузку «патча», содержащегося в текстовом файле. К примеру, опция `-aPatch.txt` загрузит «патч» из файла `Patch.txt`.

Опция `-b` аналогична опции `-a`, с той лишь разницей, что подгружает в оперативную память альтернативный загрузчик (загружаемый BSL).

Опция `-f` определяет максимальное число байтов данных, передаваемых в составе одного пакета.

Опция `-m` задает число циклов стирания flash-памяти. Разработчики рекомендуют для полной очистки памяти задавать не менее 20 циклов, то есть включать конструкцию данной опции `-m20`.

А теперь — о модификаторах программного потока. С помощью данных модификаторов осуществляется работа с памятью микроконтроллера. Модификаторы можно включать один за другим, тогда программа выполнит команды последовательно. Если читатель вернется на несколько страниц назад, к табл. 1.3, то узнает, что модификаторы в явном и неявном виде реализуют набор функций BSL.

Модификатор `+e` выполняет функцию полного стирания памяти (mass erase). Модификатор `+c` предназначен для проверки стирания памяти (erase check). Модификатор `+r` дает программе команду на «прошивку» в МК файла `<file>`, помещаемого в конце командной строки, а модификатор `+v` осуществляет верификацию «прошитого» файла с образцовым, указанным в `<file>`. Оставшиеся модификаторы предназначены для реализации функции сброса микроконтроллера (модификатор `+r`) и ожидания нажатия клавиши `<Enter>` перед завершением работы программы (модификатор `+w`).

Чтобы читателю было понятно действие тех или иных параметров, приведем несколько наиболее типичных примеров записи командной строки:

- 1) `bsldemo.exe test.txt`
- 2) `bsldemo.exe -l -w -cCOM1 test.txt`
- 3) `bsldemo.exe +vrw -pInt_vect.txt test.txt`
- 4) `bsldemo.exe -bBl_150s_14.txt test.txt`

В первом случае, когда опущены все опции и модификаторы, программа выполнит очистку памяти, проверку чистоты памяти, программирование данных из файла test.txt и верификацию памяти. Второй случай аналогичен первому, за исключением того, что программа будет работать с портом COM1, выполнит все действия за один сеанс связи, затем войдет в режим ожидания нажатия клавиши <Enter> для завершения работы. В третьем случае данные из файла Int_vect.txt используются в качестве пароля доступа к защищенным функциям, затем производится верификация памяти, сброс микроконтроллера и ожидание нажатия клавиши <Enter>. И, наконец, четвертый случай: в оперативную память микроконтроллера загружается альтернативный загрузчик, затем происходит полное стирание памяти, «прошивка» памяти данными из файла test.txt и сброс микроконтроллера. Задавать модификатор верификации здесь нет необходимости — обычно альтернативные загрузчики выполняют верификацию памяти в процессе «прошивки» автоматически.

Существуют и более удобные в пользовании программы для работы с BSL, которые используют возможности операционной системы Windows. Одна из таких программ имеет достаточно длинное название «Загрузчик файлов через Bootstrap Loader for MSP430» (файл bootloader.exe) и разработана В. Э. Можаровым. Программа не требует инсталляции, сразу после запуска появляется окно, показанное на рис. 1.8.



Рис. 1.8. Программа загрузки через BSL, разработанная В. Можаровым

Набор функций, предоставляемых этой программой, весьма ограничен — имеются только самые необходимые: очистка памяти МК, запись в память МК, получение версии аппаратного BSL, выбор номера COM-порта, инвертирование сигнала TEST. Программируемый файл может быть открыт в программе с расширением .hex (intel standart) либо с расширением .txt (MSP430-txt standart). Сохранена совместимость по линиям физического интер-

фейса с описанным выше «железом». Простота этой программы должна заинтересовать в первую очередь начинающих работу с микроконтроллерами.

Второй вариант бесплатной утилиты для программирования MSP430 по BSL-интерфейсу разработан совместно двумя авторами: Рустемом Калимуллиним и Александром Кратько. Их утилита носит название MspFet, ее можно свободно «скачать» с авторского сайта [<http://mspfet.hellos.biz> или <http://kurt.on.ufanet.ru>]. На сайте постоянно происходит обновление версий программатора, помещается и другая полезная информация для пользователей MSP430.

MspFet — мощная программа, использующая все возможности, заложенные разработчиками MSP430 в части их «прошивки». Авторы программы предлагают два варианта дистрибутива. В первом случае требуется просто распаковать все файлы в одну папку из zip-архива, занимающего всего 86 кбайт. Во втором случае exe-файл с инсталлятором займет на диске 152 кбайт, но установка программы сведется к двум щелчкам «мыши», а также будут созданы ярлыки на рабочем столе и в главном меню.

На рис. 1.9 показано основное окно утилиты MspFet. Центральную часть окна занимает развернутый файл (в виде шестнадцатеричных HEX-кодов), который предполагается «прошить» в память МК или произвести с ним другие операции (сверить с уже «зашитым», считать из памяти информацию в файл). Над окном файла располагаются меню «файл», «команды», «утилиты». Наиболее часто используемые команды и утилиты продублированы кнопками справа от окна файла. Под файловым окном расположена строка выбора типа программируемого микроконтроллера (при выборе конкретного типа МК пользователь информируется о его объемах памяти) и строка выполненных операций. Программа «работает» с расширениями файлов .bin, .dat, .tsk, .txt, .hex, .bin.

При нажатии кнопки «настройка» появляется окно конфигурации, показанное на рис. 1.10. Это окно позволяет настроить программу на работу с разными типами физических программаторов (адаптеров), задать набор функций в режиме автопрограммирования. Понятно, что при работе с загрузчиком типа BSL необходимо с помощью радиокнопки подсветить опцию BSL, выставить номер COM-порта. Две другие опции (RAW и FET) используются в случае, если предполагается работать с интерфейсом JTAG, и об этом мы подробно поговорим чуть позже.

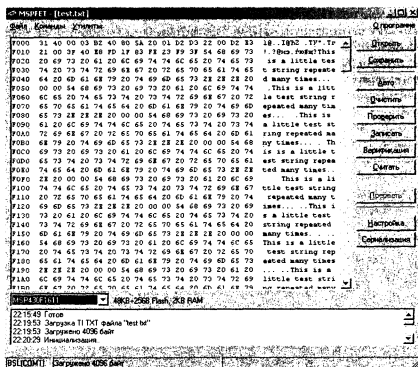


Рис. 1.9. Основное окно утилиты MspFet

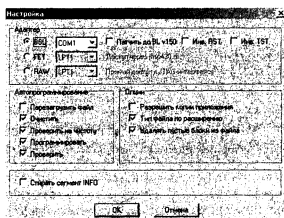


Рис. 1.10. Окно «настройка» утилиты MspFet

Выбрать набор опций для режима автопрограммирования поможет соответствующая панель данного окна, в которой пользователю нужно лишь расставить «галочки», и программа последовательно выполнит заданные операции.

Еще одно окно (рис. 1.11) авторы программы назвали словом «сериализация». Использовать возможности, заложенные здесь, имеет смысл только в промышленных масштабах, когда выпускается серия одинаковых изделий. В память микроконтроллера может быть занесен уникальный серийный номер, а также считан из памяти. Серийный номер может быть типа «байт» (byte), «слово» (word), «двойное слово» (dword) или «строка символов» (char). Тип серийного номера задается в панели «формат серийного номера», здесь же указывается начальный адрес размещения его в памяти, а также начальное значение. Текущий серийный номер указывается ниже, изменение его на единицу (инкрементирование и декрементирование) осуществляется кнопками «inc(+)» и «dec(-)». Файл конфигурации можно сохранить или загрузить, для чего предусмотрены диалоги «открыть» и «сохранить».

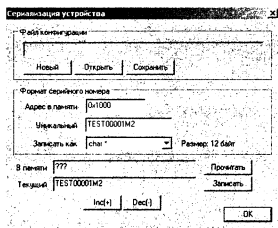


Рис. 1.11. Окно «сериализация» утилиты MspFet

Приятно отметить, что авторы программы позаботились о русскоязычном пользователе, создав возможность выбора языка (русский и английский). По умолчанию устанавливается английский язык, но его можно сменить на русский в меню «утилиты/язык интерфейса».

А теперь мы поговорим о других программаторах, использующих интерфейс BSL. Даже беглый поиск этих устройств в Интернете показывает, что достаточно много фирм предлагают приобрести разработанные ими аппаратные и программные средства для работы с BSL. Аппаратные части, как правило, изготовлены на высоком техническом уровне, размещены в удобных корпусах, поставляются в комплекте со всевозможными соединительными кабелями, переходными панельками, источником питания и т. д. Стоимость таких комплектов составляет сотню-другую у. е., что заставит глубоко задуматься прежде всего отечественного радиолюбителя: стоит ли тратить эти деньги? С большой долей уверенности можно сказать: для работы с MSP430 в промышленных условиях потратиться стоит, поскольку фирменные изделия окажутся надежнее самоделок. Ну а радиолюбителям, как говорится, «сам Бог велел» поработать руками и смастерить программатор самостоятельно. Кстати, маркетинговая политика некоторых зарубежных фирм очень поможет в этом деле.

Существует и успешно работает на рынке часть фирм, которые не считают необходимым скрывать электрические схемы, программное обеспечение и прочую документацию на свои программаторы, а просто публикуют их в Интернете, но, тем не менее, предлагают приобрести у них готовый, промышленно выполненный, аппаратный программатор. Цены на такую продукцию весьма доступные, но если не хочется потратить даже эти деньги, можно спокойно заняться изготовлением самостоятельно. Примером такой фирмы может служить американская компания SoftBaugh [<http://www.softbaugh.com>], разработавшая Bootloader for MSP430 flash (BLMSPF). Печатная плата аппаратной части программатора, которую фирма предлагает приобрести за \$95, показана на рис. 1.12.

Схема аппаратной части программатора показана на рис. 1.13. Обращаем внимание читателя на то, что данная схема несколько отличается от той, которую предлагает SoftBaugh. В частности, исключены некоторые элементы, которые не влияют на работоспособность, подобраны аналоги компонентов схемы.

Питание +5 В от стандартного источника, обеспечивающего ток не менее 200 мА, подается на разъем X1. Далее установлены два интегральных стабилизатора D1 и D2, преобразующие это напряжение в 3,3 В, причем стабилизатор D1 питает программируемый микроконтроллер, а стабилизатор D2 — функциональные

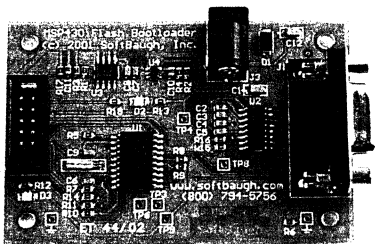


Рис. 1.12. Фирменная печатная плата программатора BLMPF

узлы схемы программатора. Микросхема D3, в качестве которой использован микроконтроллер MSP430F1121A, формирует протокол связи BSL (программа «прошивки» для этого МК находится в составе файла BLMPF_v134.zip, ее можно «скачать» с сайта фирмы). Микросхема D4 — преобразователь уровней интерфейса RS-232.

Изготовить такой программатор несложно, если в наличии будет уже «прошитая» микросхема D3. А если нет? Здесь на память приходит старинная загадка: «Что первично — курица или яйцо?» Другими словами, чтобы «прошивать» микроконтроллеры, нужен программатор, но чтобы изготовить данный программатор — нужно «прошить» микроконтроллер... К чему «городить огород», если можно использовать предыдущий программатор? Разработчики BLMPF уверяют, что их продукт более надежен, менее подвержен сбоям. Поэтому вначале требуется изготовить стандартный BSL-программатор и уже с его помощью «прошить» МК для BLMPF.

Требования к элементам, входящим в схему программатора, — стандартные. Резисторы — типа C2-33, конденсаторы — K10-17 и K50-68. Если позволят возможности, можно использовать SMD-компоненты. Микросхему D1 можно заменить на TPS7201, микросхему D2 — на TPS77001, включив их по типовым схемам.

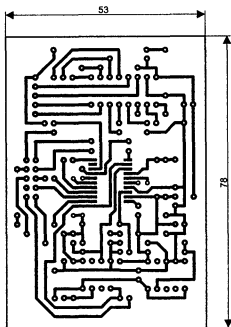


Рис. 1.14. Печатная плата

Собрать программатор можно на печатной плате (рис. 1.14) по сборочному рисунку (рис. 1.15). Микроконтроллер устанавливается со стороны печатных проводников (рис. 1.15, б).

Настало время рассказать о программном обеспечении. Фирма SoftBaugh предлагает несколько программ поддержки, обладающих разными функциональными возможностями. На рис. 1.16 показано главное (и единственное) функциональное окно загрузочной программы «MSP430 Flash Bootloader» (файл BootWrite.exe). Основную часть окна занимает информационная панель (event log), сообщающая пользователю в виде текста о процессе программирования, о возможных ошибках, которые могут возникнуть в процессе «прошивки». Дополнительно текстовая информация фиксируется на диске в файле EventLog.txt.

Справа от информационной панели расположены кнопки управления программой. В выпадающем списке в правом верхнем

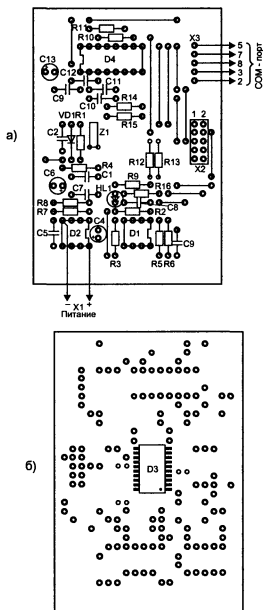


Рис. 1.15. Сборочный рисунок: а) сторона установки компонентов; б) сторона печатных проводников

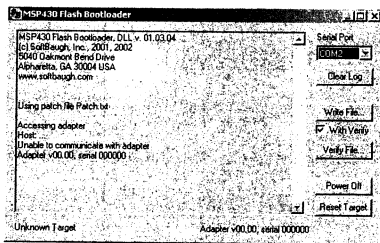


Рис. 1.16. Главное окно программы BootWrite.exe

углу (Serial port Dropdown) можно выбрать номер COM-порта компьютера для работы с программатором. Ниже помещается кнопка очистки информационной панели (clear log). При запуске программатора, а также при смене номера COM-порта автоматически загружается «заплата» к встроенному в микроконтроллер BSL-загрузчику из файла patch.txt.

Еще ниже — кнопка «прошивки» памяти микроконтроллера (write file). При ее нажатии открывается диалог выбора файла «прошивки», а после выбора файла (в формате .txt или .hex) автоматически запускается процесс стирания памяти, затем проходит запись данных во flash-память. Следует отметить, что верификация данных после записи может быть отключена, и чтобы все же верифицировать данные, необходимо установить флажок в поле «проверка» (with verify). Отдельная верификация выполняется при нажатии кнопки «проверка файла» (verify file), когда открывается диалог выбора файла и собственно проходит отдельная проверка.

Кнопка отключения питания (reset target) предназначена для снятия напряжения питания с программируемого микроконтроллера. Эта кнопка введена разработчиками для того, чтобы не повредить микроконтроллер в моменты коммутации разъема программатора. Кнопка сброса (reset target) выполняет функцию сброса (RST) микроконтроллера.

Под информационной панелью имеются два текстовых информационных поля. Левое поле (target version display) позволяет узнать, к какому семейству относится программируемый микроконтроллер (например, в случае MSP430F140 будет выведена надпись F149), а также версию встроенного BSL-загрузчика. Правое поле (adapter version display) предназначено для вывода номера фирменной версии аппаратного программатора и его серийного номера. Облегченная версия «MSP430 Flash Bootloader» называется «Progress Writer» (файл ProgressWriter.exe). В ней информационная панель заменена индикатором типа progress bar, в остальном же по функциональным возможностям аналогична BootWrite.exe.

Фирма предлагает вниманию опытных пользователей также расширенную версию этой программы (файл BootFlash.exe), главное окно которой показано на рис. 1.17. Расширенная версия отличается тем, что пользоваться ей сможет только тот пользователь, кто знаком с последовательностями команд BSL-интерфейса. Как

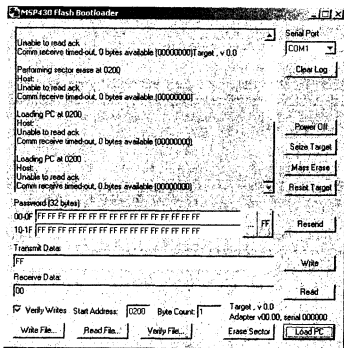


Рис. 1.17. Главное окно программы BootFlash.exe

видно из рисунка, введены панели и кнопки для работы с паролем. В правой и нижней части окна размещены собственно командные кнопки. О назначении кнопок можно догадаться по надписям на них — они совпадают с наименованиями команд.

Чуть раньше мы уже встречались с возможностью сериализации «прошивки» микроконтроллеров. Специально для этих целей SoftBaugh выпустила программу «MSP430 PRGS Serializing Shell» (файл SerialProgShell.exe), главное окно которой показано на рис. 1.18. К сожалению, программа защищена ключом, и чтобы его получить, необходимо отправить на адрес разработчика уникальный номер, который копируется из окошка при запуске неавторизованной программы.

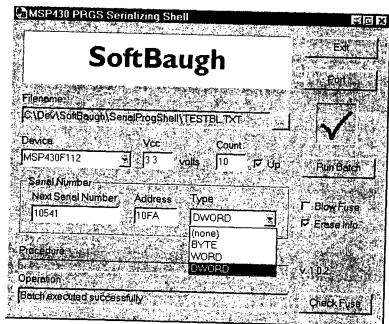


Рис. 1.18. Главное окно программы SerialProgShell.exe

Вероятнее всего, читатель, достаточно много времени проводящий в сети Интернет, найдет и другие программаторы, использующие BSL-интерфейс. Разобраться в их работе после прочтения данного раздела, надеемся, окажется несложной задачей.

1.5. Прошивка МК с использованием JTAG

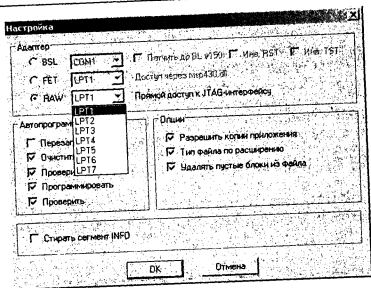
Интерфейс JTAG, как уже было сказано выше, обладает намного более широкими функциональными возможностями, чем интерфейс BSL: его можно использовать не только для «прошивки», но и для отладки в реальном времени работающей схемы. Об отладочных функциональных возможностях JTAG мы поговорим позже, а сейчас познакомимся с этим интерфейсом в режиме «прошивки».

Если читатель помнит, в разделе 1.3 упоминалась утилита MspFet, которая имеет возможность работы со JTAG. В данном случае подключение аппаратного адаптера осуществляется к LPT-порту компьютера, а доступ к нему может быть осуществлен как напрямую (функция RAW на рис. 1.19, а), так и при помощи драйвера msp430.dll (функция FET на рис. 1.19, б). Эти возможности позволяют работать с MspFet как в операционной среде Windows 98 (в режиме RAW), так и в последующих ее версиях, включая Windows XP (в режиме FET).

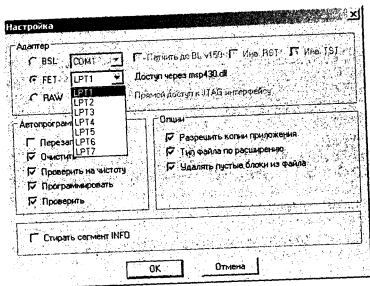
Что такое драйвер msp430.dll? Это специализированная библиотека, которая реализует основные протоколы доступа по JTAG-интерфейсу. Данной библиотекой оснащаются многие среды разработки программного обеспечения MSP430, имеющие возможность отладки в «реал-тайме». Чтобы функция FET могла работать корректно, необходимо скопировать файлы msp430.dll и hil.dll в папку с exe-файлом программы MspFet. В режиме RAW программа не требует дополнительных драйверов, реализуя протоколы доступа по JTAG собственными средствами, но этот режим будет работать в операционных средах не новее Windows 98.

На рис. 1.20 показана схема несложного аппаратного адаптера, который может быть использован в качестве JTAG-интерфейса персонального компьютера с лишь одной оговоркой: он не поддерживает установку бита защиты (fuse). Питание программируемой микросхемы осуществляется непосредственно от LPT-порта. В качестве микросхемы D2 может быть использована МСР601 или другой rail-to-rail операционный усилитель с однополярным питанием 2,7...6,0 В. Транзистор VT1 заменим на BC846, BC847, BC850. Печатная плата приведена на рис. 1.21, а «сборка» — на рис. 1.22.

На рис. 1.23 показана схема JTAG-адаптера, предлагаемого фирмой Olimex (<http://www.olimex.com>). Он мало чем отличается



а)



б)

Рис. 1.19. Включение функций доступа по JTAG в утилите MspFet: а) функция RAW; б) функция FET

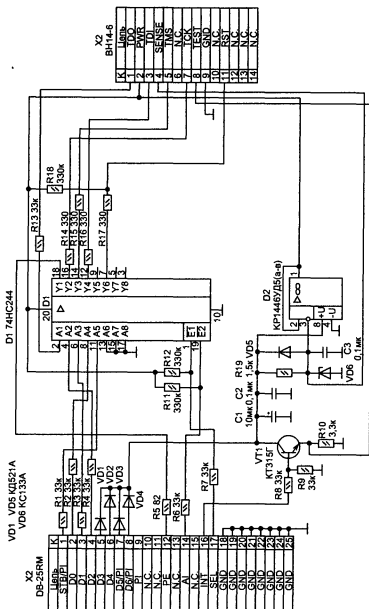


Рис. 1.20. Схема JTAG-программатора для MspFet

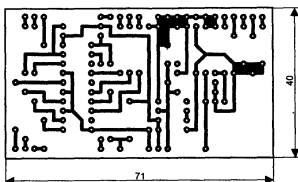


Рис. 1.21. Печатная плата

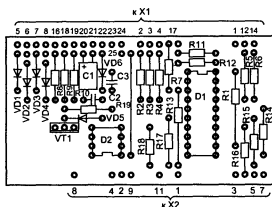


Рис. 1.22. Сборочный рисунок

от приведенного выше адаптера, разве что появилась микросхема D1 стабилизатора.

Более сложный программатор-отладчик приведен на рис. 1.24. Он в значительной степени отличается от вариантов, показанных на рис. 1.20 и 1.23, хотя, при внимательном рассмотрении, и здесь можно обнаружить знакомые узлы. Как и в предыдущих случаях, этот адаптер подключается к LPT-порту персонального компьютера. Его поддерживают такие программные продукты, как Iar Embedded Workbench, Iar Kikstart Software и GCC. Адаптер не требует внешнего питания, но если такая необходимость возникнет, можно получить питание от внешнего блока, подав его на разъем X1.

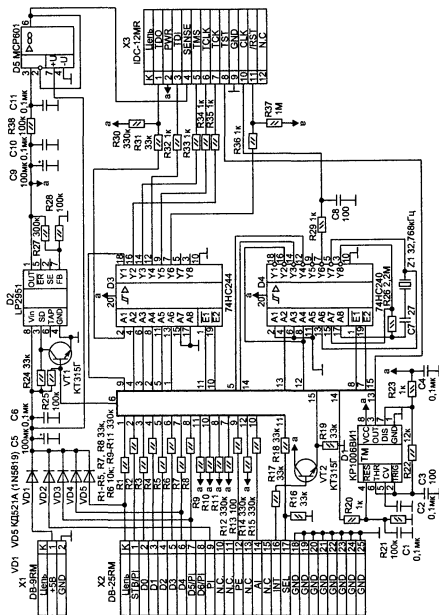


Рис. 1.24. Схема ITAG-адаптера с расширенными возможностями

Если читателю не хочется возиться с изготовлением данного адаптера, он может заказать готовую плату в Интернете по адресу <http://www.e-kit.ru>.

Более удачный с точки зрения спектра используемых операционных систем программатор — это программатор с доступом по последовательному COM-порту. Схема такого устройства, называемого `pySerJTAG`, приведена на рис. 1.25.

Данный адаптер предназначен для работы в среде GCC (<http://mspgcc.sourceforge.net>). Он выполняет функции загрузки оперативной памяти микроконтроллера, периферийных устройств и flash-ПЗУ; стирания сегментов памяти или полной памяти; выполнения команды по соответствующему адресу; остановки центрального процессора в любой момент; сброса центрального процессора; копирования программы пользователя, сохраненной в памяти адаптера, в отлаживаемый микроконтроллер (данная функция реализуется аппаратно, нажатием кнопки «Start clone MSP»).

Программное обеспечение для «заливки» в микроконтроллер адаптера находится в файле `serjtag.zip` и при загрузке размещается в верхних 8 кБ flash-памяти. Авторы программного обеспечения предупреждают пользователя, что данный продукт носит характер beta-версии, то есть тестировался не полностью, а значит, возможны некоторые ошибки.

На схеме рис. 1.25 разъем X1 предназначен для подключения внешнего питания, разъем X2 — для связи с компьютером, а X3 образует JTAG-интерфейс. Единичный индикатор HL1 (АСТ) сигнализирует об активном обмене между компьютером и программатором, а также между программатором и микроконтроллером. Индикатор HL2 (FC) предупреждает о прохождении клонирования программой программатором. Особое внимание следует уделить джамперам j4 и j5, которые настраивают скорость обмена данными по COM-порту.

* * *

Наш обзор средств программирования микроконтроллеров MSP430 был бы неполным, если бы мы не упомянули коммерческие продукты, предназначенные для этой цели.

Программатор MSP-ADT430 относится к достаточно старым средствам работы с микроконтроллерами MSP430. Сегодня уже нельзя приобрести аппаратный адаптер к этому программатору (по крайней мере, не бывший в употреблении). Также с сайта фирмы удалена программа поддержки. Но мы все же упомянем об

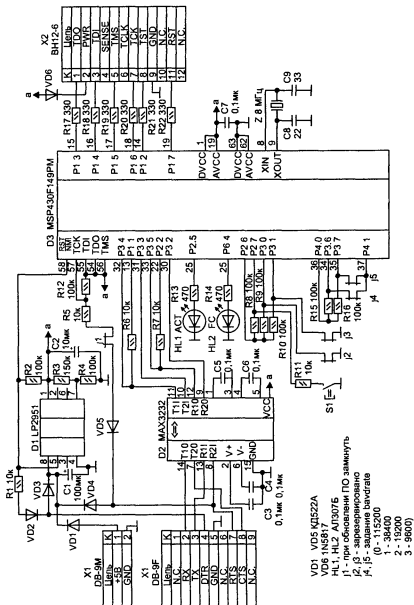


Рис. 1 25 Схема JTAG-адаптера с доступом по COM-порту

На рис. 1.27 приведен внешний вид адаптера, а на рис. 1.28 — главного окна программы поддержки MSP430-Programmer. Файл «прошивки» открывается в панели «source» и может иметь расширение .txt, .a43, .hex. В поле «target» можно выбрать тип микроконтроллера, который предполагается «прошивать». Ниже располагаются три вкладки, которые отражают основные режимы работы: «program», «erase/verify», «read». Еще ниже — опции настройки программатора.

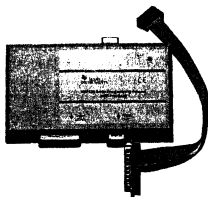


Рис. 1.27. Внешний вид адаптера MSP-PRGS430

Более новый программатор последовательного типа, поставляемый Texas Instruments, называется MSP-GANG430. Его внешний

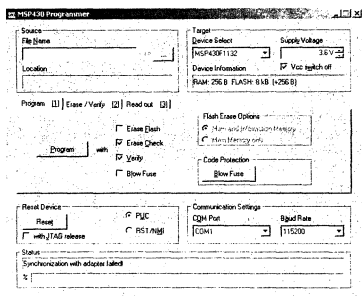


Рис. 1.28. Главное окно программы MSP430-Programmer

вид показан на рис. 1.29, а стоимость практически такая же, как и MSP-PRGS430, но данный программатор позволяет одновременно работать с 8 кристаллами, что выгодно отличает его от других типов адаптеров. Само собой разумеется, что при необходимости работа может осуществляться и с одним кристаллом. Инструкция по эксплуатации MSP-GANG430 приведена в slau101a.pdf, дистрибутив программы поддержки — в файле GANG430_R125.exe.

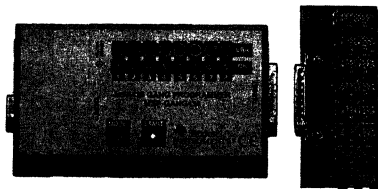


Рис. 1.29. Внешний вид адаптера MSP-GANG430

На рис. 1.30 показано основное окно программы поддержки MSP430-Flash Gang Programmer. Оно мало чем отличается от предыдущей программы по набору функций, за исключением панели «result», которая отражает возможности мультипроцессорного доступа.

В последнее время широкое распространение получил интерфейс связи с ПК типа USB, а вот интерфейсы типа LPT или COM начинают отходить на второй план. Фирмы-производители отладочных средств для MSP430 отреагировали на данную конъюнктуру разработкой USB-адаптеров. Пока эти адаптеры существуют в виде сугубо коммерческих продуктов, следовательно, найти что-то для самостоятельного изготовления пока невозможно.

Фирма CrossStudio (<http://www.crosstudio.co.uk>) предлагает приобрести адаптер JTAG типа MSP430 USB CrossConnect (рис. 1.31), который «работает» с оригинальной средой разработки программного обеспечения CrossWorks for MSP430. По заявлению разработчиков CrossWorks, они не планируют вводить совместимость адаптера с другими средами разработки, например, с IAR. Скорость «прошивки» для этого адаптера — 28 кб/с, обеспечена совме-

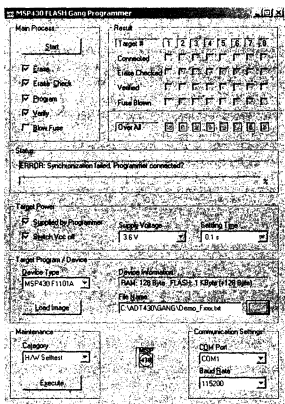


Рис. 1.30. Главное окно программы MSP430-Flash Gang Programmer

стимость практически со всеми современными версиями «окошек» (Windows XP, Windows 2000, Windows ME, Windows 98SE). Стоимость набора составляет \$150. Единственный недостаток, который имеется в этом адаптере, — невозможность работы с битом fuse.

Еще один коммерческий USB-программатор предлагает уже знакомая читателю фирма SoftBaugh (рис. 1.32). Внешне он представляет собой небольшую коробочку с разъемами, которая с одной стороны подключается к USB-порту ПК, а с другой — к JTAG-интерфейсу отлаживаемого микроконтроллера.

Стоимость полной версии программатора составляет \$199 при очевидных преимуществах, среди которых главные — это работа с fuse и совместимость с IAR. Однако чтобы IAR смог корректно ра-

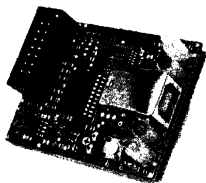


Рис. 1.31. Внешний вид адаптера MSP430 USB CrossConnect



Рис. 1.32. Внешний вид USB-адаптера от SoftBaugh

ботать с данным адаптером, необходимо в папку IARxx/430/bin распаковать драйвера MSP430.dll, SBMSP430.dll и Vendor_USBP.dll, которые находятся в файле USBP_Beta3_040428.zip.

На рис. 1.33 приведен внешний вид программы поддержки этого адаптера, называемой USBP Programmer. Хорошо видно, что программное обеспечение обладает всеми необходимыми функциями, включая сериализацию.

Читатель наверняка уже обратил внимание на то, что работы с fuse микроконтроллеров является как бы отдельной задачей: в некоторых программаторах эта функция имеется, но может также отсутствовать. Повторимся еще раз — работа с fuse имеет смысл на уровне промышленного производства, а радиолюбителям нет смысла защищать свой программный код. И все же, если такая необходимость возникнет, можно воспользоваться специальными адаптерами, выполняющими эту и только эту функцию.

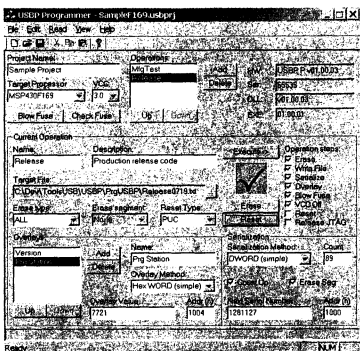


Рис. 1.33. Главное окно программы USBP Programmer

Первое устройство работы с fuse показано на рис. 1.34. Его можно заказать у фирмы e-KIT (<http://www.e-kit.ru>). Работает устройство следующим образом. С помощью стандартного кабеля выполняется подключение к JTAG-интерфейсу защищаемого микроконтроллера, затем подается питание. При этом должен загореться зеленый светодиод на верхней крышке устройства, сигнализируя об успешном пережигании fuse. Если зеленый светодиод мигает, значит, микроконтроллер ранее уже был защищен. Имеется также красный светодиод, который относится к аварийной диагностике: его однократное мигание свидетельствует о том, что не удалось определить тип микроконтроллера, двукратное мигание — о невозможности прожига fuse, и трехкратное — об ошибке связи по JTAG-интерфейсу. Питание устройства осуществляется от источника с напряжением 9...12 В, его стоимость составляет порядка \$25.

Второй вариант адаптера, позволяющего программировать fuse, предложен создателями MSPFET. Как видно из рис. 1.35, устрой-



Рис. 1.34. Внешний вид устройства работы с fuse от e-KIT

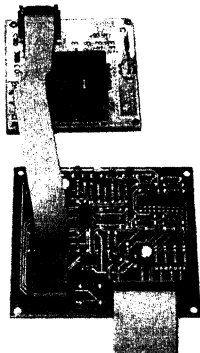


Рис. 1.35. Внешний вид устройства работы с fuse от создателей MSPFET

во представляет собой небольшую печатную плату, подключаемую к обычному JTAG-адаптеру, при этом сохраняются все возможности JTAG-интерфейса. Стоимость адаптера составляет \$55.

Эксклюзивную линейку коммерческих продуктов для программирования MSP430 выпускает фирма Elprotronic Inc, располагающаяся в г. Онтарио (Канада). Сайт фирмы можно посетить по адресу [<http://www.elprotronic.com>]. В основном, конечно, продукция фирмы интересна с точки зрения альтернативного программного обеспечения фирменных программаторов, в меньшей степени — с точки зрения аппаратных средств (приобрести их затруднительно). Тем не менее мы все равно расскажем о продукции Elprotronic Inc. Итак, фирма производит и поставляет всем желающим в течение максимум 2—3 недель следующую продукцию:

- программное обеспечение FET-Pro430 для поддержки программаторов, выпускаемых Texas Instruments, как для варианта USB, так и для варианта параллельного порта; программа имеет только JTAG-интерфейс; стоимость полной версии (поддержка серпализации) составляет \$119, облег-

- ченная версия без поддержки серпализации — \$69, студенческая (учебная) версия — \$59 (вообще-то учебные версии в приличном обществе принято поставлять бесплатно);
- программно-аппаратный комплект FlashPro430 с собственным аппаратным программатором USB, поддержкой интерфейсов JTAG и BSL; этот комплект может быть использован как JTAG-отладчик внутрисхемного типа; стоимость комплекта колеблется от \$293 (полная версия) до \$206 (студенческая версия);
 - программно-аппаратный комплект GangPro430, включающий в себя аппаратный USB-программатор с интерфейсом JTAG; этот комплект также можно использовать для внутрисхемной отладки по JTAG; этот программатор может осуществлять «прошивку» одновременно шести кристаллов, что в значительной степени поможет при промышленном серийном производстве; несколько программаторов GangPro430 можно подключать к разным USB-портам одного компьютера и нарастить число одновременно «прошиваемых» кристаллов; стоимость комплекта составляет \$311;
 - программно-аппаратный комплект FlashPro430-PP, включающий в себя аппаратный программатор с поддержкой параллельного порта; этот комплект поддерживает интерфейсы JTAG и BSL; в комплект поставки входят собственно программатор, диск с программным обеспечением и соединительный кабель; стоимость комплекта составляет от \$168 (стандартная версия) до \$101 (ограниченная версия), студенческой версии почему-то нет;
 - отладочная плата MSP430F169 с установленным микроконтроллером, различными интерфейсами; эта плата поможет начать работу с MSP430; ее стоимость в полной комплектации составит \$202, а в студенческой — \$181.

На рис. 1.36 показано главное окно программы FET-Pro430. Эта программа использует для «общения» с компьютером стандартный драйвер `msp430.dll`, а значит, может напрямую работать с фирменными стартовыми наборами MSP430-FET (параллельный интерфейс) и MSP-FET430UIF (USB-интерфейс). Также программатор может работать с набором MSP430-JTAG от Olimex или с программаторами от SoftBaugh. В принципе, также должен работать с самодельными JTAG-программаторами, описанными в этой книге.

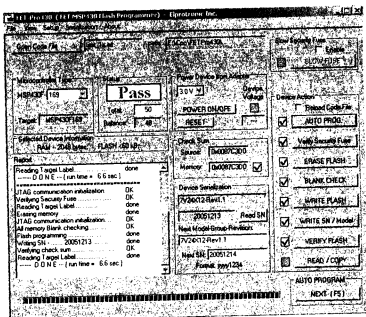


Рис. 1.36. Главное окно программы FET-Pro430

Программа поддерживает все микроконтроллеры серии MSP430, поддерживает установку fuse-бита (если программатор имеет аппаратную возможность устанавливать fuse-бит), поддерживает расширения загрузочных файлов типа .txt, .s19, .hex, полная версия программы может работать в режиме сериализации, причем код может быть считан со специального bag-считывателя. Подробная информация об использовании программатора может быть получена прямо на сайте (имеется описание в pdf-формате). Там же по ссылке [<http://www.elprotronic.com/download.html>] можно «скачать» 30-дневную пробную trial-версию программы.

На рис. 1.37 показан внешний вид комплекта FlashPro430. Надо сказать, вид достаточно скромный. Однако внешнее впечатление обманчиво: к одному компьютеру можно подключить до 8 таких программаторов, 60 кб памяти микроконтроллера здесь будут программироваться не более 2,3 секунды. Осуществляется настройка скорости обмена по JTAG-интерфейсу: ее можно установить 4 Мб/с, 1 Мб/с или 400 кб/с. Скорость «общения» по BSL составляет 9,6 кб/с и 350 кб/с. Естественно, имеется возможность

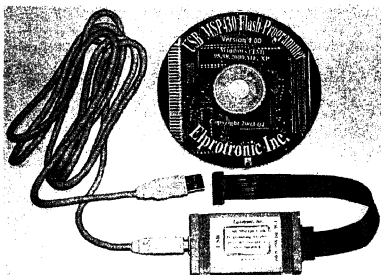


Рис. 1.37. Комплект FlashPro430

сериализации, которую поддерживает программная часть комплекта, показанная на рис. 1.38.

Программное обеспечение поддерживает стандартный драйвер `mbsp430.dll` и может работать со средами программирования от IAR, CrossWorks и других фирм. Пробную версию можно «скачать» с сайта фирмы.

Еще один комплект GangPro430 показан на рис. 1.39. Он во многом напоминает предыдущий комплект, но имеются и отличия, главное из которых — возможность работы с шестью программаторами от одного USB-порта и поддержка 8 портов. Таким образом, «прошивка» 48 кристаллов займет не более 5 секунд.

На рис. 1.40 приведено основное окно программы поддержки GangPro430. Здесь имеются индивидуальные настройки для каждого из шести каналов, режим автопрограммирования и сериализации.

И, наконец, немного о комплексном подходе к разработке конструкций на основе MSP430. Отечественная фирма Фитон (<http://www.phyton.ru>) предлагает полный набор средств работы с данными микроконтроллерами, среди которых — JTAG-отладчик PICD, поддерживающий полный набор отладочных функций и связь с персональным компьютером по протоколам USB1.1 и USB2.0. Обеспечена поддержка всех типов микроконтроллеров се-

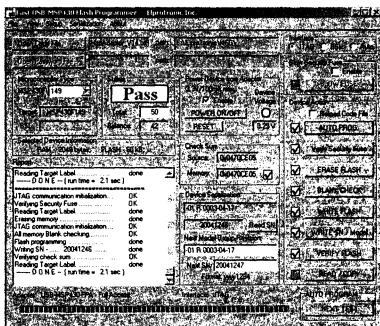


Рис. 1.38. Программа поддержки FlashPro430



Рис. 1.39. Комплект GangPro430

мейства MSP430, мониторинг состояния микросхемы в процессе выполнения программы пользователя в реальном времени, стыковка со стандартными средствами программирования.

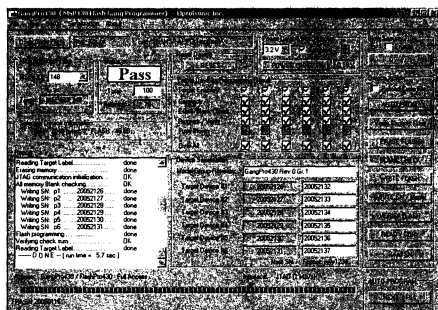


Рис. 1.40. Программа поддержки GangPro430

Глава 2. Среда — не только день недели

Самое время познакомить читателя с основными инструментами, использующимися при разработке программного обеспечения для микроконтроллеров — со средами программирования, отладчиками, компиляторами, менеджерами проектов. Читать эту главу лучше всего рядом с включенным персональным компьютером и с желанием тут же пополнить копилку своих знаний на практике.

2.1. Начать можно с IAR

Вне всякого сомнения, вопрос — с чего начать — один из самых главных для тех, кто готовится осваивать искусство программирования. Обычно предполагается, что «для начинающих» все должно быть предельно просто, доступно, недорого (в идеале — бесплатно). Кроме этого, у начинающего должны легко вырабатываться основные навыки при минимуме затраченных усилий, иначе (особенно если работа с микроконтроллерами не входит в сферу профессиональных интересов) можно просто бросить «это скучное занятие»...

Всем тем, кто хочет эффективно продвигаться по пути освоения MSP430, шведская фирма IAR Systems (<http://www.iar.se>, <http://www.iar.com>) предлагает воспользоваться средой программирования MSP430 на языке высокого уровня «Си/Си++» и ассемблере под названием IAR Embedded Workbench for MSP430. Базовые версии среды включают в себя менеджер проектов (integrated development environment), редактор текстовых файлов, «Си/Си++»-компилятор, ассемблерное приложение, библиотеку математических функций и операций, программный симулятор-отладчик, отладчик устройства в реальном времени (поддержка внутрисхемной эмуляции). Совместимость обеспечивается для операционных систем типа Windows 98/ME/NT4/2000/XP, то есть для всех ОС, широко распространенных на сегодняшний момент. Особое внимание читателя следует обратить на то, что эта среда разработки обеспечивает совместимость на аппаратном уровне с JTAG-интерфейсом отлаживаемого микроконтроллера, а значит, позволяет (с помощью модуля C-SPY, входящего в состав IAR

EWB) увидеть работу разработанной программы тут же, на реальном «железе». Иными словами, пользователю становится доступной упомянутая функция внутрисхемной эмуляции.

Фирма предлагает три варианта установки EWB. Первый вариант — лицензионный, наиболее полный по функциональным возможностям и позволяющий работать без всяких ограничений. Приобрести лицензионный дистрибутив (full version) можно, обратившись как в фирму-разработчик, так и к ее дилерам (так рекомендует поступать IAR Systems). Правда, на Интернет-конференциях, посвященных применению микроконтроллеров, можно встретить другую информацию: лицензионный дистрибутив можно запросто «скачать» с ftp-сервера фирмы (<ftp://ftp.iar.se>) в виде файла EW430-XXX.exe, где «XXX» — порядковый номер версии. При установке программа, естественно, «попросит» лицензионный номер, о приобретении которого нужно побеспокоиться заранее, обратившись к фирме-производителю. Есть также различные неофициальные варианты, связанные с поиском «лекарства от жадности» в Интернете, но мы об этих вариантах говорить не будем по вполне понятным причинам...

Конечно, лицензионный вариант наиболее предпочтителен для профессиональной работы с MSP430, но, к сожалению, стоимость данной версии EWB, по некоторым сведениям составляющая около \$3500, заставляет задуматься даже «промышленников», не говоря уже о радиолюбителях. Серьезная работа с MSP430 все же заставит «раскошелиться» тех, кто намерен соблюдать авторские права IAR Systems, но для начинающих освоение MSP430 платить деньги не нужно, причем на вполне законных основаниях! Дело в том, что фирма Texas Instruments выпускает серию так называемых «стартовых наборов» (kickstart, «резкий старт»), позволяющих начать работу с микроконтроллером «в железе». В комплект типового стартового набора входят: печатная плата с установленной на ней панелью под микроконтроллер и разъемом JTAG-интерфейса (подключается к LPT-порту ПК), собственно микросхемы микроконтроллера, соединительные кабели и компакт-диск. Условное наименование этих наборов — MSP-FET430, их стоимость колеблется от \$50 до \$200 в зависимости от типа поставляемого МК.

Читатели вправе сами оценить, стоит ли им «потратиться» на «стартовый набор» или же изготовить макетную плату своими руками. Но вот программное обеспечение (среда программирования), поставляемое в комплекте с «резким стартом», пригодится в

любом случае. Главное его преимущество в том, что за данный дистрибутив не нужно платить, его можно просто «скачать» с сайта Texas Instruments. Разработчиком ПО для «стартового набора» выступила уже знакомая нам IAR Systems, взяв за основу EWB и несколько «урезав» его мощности. В результате получилась IAR Embedded Workbench Kickstart for MSP430, внешне практически ничем не отличающаяся от полной (лицензионной) версии, но с ограничением четырьмя кб размера компилируемого файла из Си-приложения, с отсутствующими в библиотеках функциями чисел с «плавающей точкой», тригонометрическими и многими другими функциями. Разработчики не стали ограничивать размер компилируемого файла из ассемблерного приложения, также не скрыли от пользователя библиотеки простейших арифметических операций с «плавающими» числами (вычитание, сложение, умножение, деление).

Третий вариант, который также можно бесплатно «скачать» с сайта IAR Systems, имеет ограничение по времени работы, равное 30 дням. Это — так называемая «демоверсия» полного лицензионного ПО, называемая IAR Embedded Workbench Evaluation for MSP430. Установка и использование evaluation-версии имеет некоторые особенности. Во-первых, необходимо «скачать» с сайта фирмы последнюю версию дистрибутива (на момент написания этой книги «действующей» была версия 3.30a). Запомните: дистрибутив в имени файла всегда маркируется как EW430-EV-XXX.exe. Затем на сайте фирмы нужно заполнить регистрационную форму, подробно ответив на все ее вопросы во всех полях. Отвечать можно и «невпопад», главное — чтобы e-mail был правильный, тогда через некоторое время от фирмы придет письмо, в котором, кроме благодарности за внимание к ее продукту, содержатся лицензионный номер (license number) и лицензионный ключ (license key). Номер и ключ нужно будет скопировать в соответствующие поля при инсталляции EWB.

Evaluation-версия EWB предназначена для тех пользователей, которых уже не устраивает EWB Kickstart, но приобретать лицензионный продукт они опасаются. Именно для них предоставляется месяц бесплатной работы в практически полной по функциональным возможностям среде программирования, со снятым ограничением на размер генерируемого кода, нормальными библиотеками функций. В демонстрационной версии, правда, ограничены возможности симулятора работы микроконтроллера,

но опять же не стоит забывать, что это — плата за «бесплатность». Следует также сказать, что некоторые умельцы научились «продлевать жизнь» evaluation-версии и снимать ограничения генерации кода, о чем они с удовольствием делятся на интернет-форумах. Мы пройдем мимо этой «аптеки», но расскажем, как совершенно законным способом повысить функциональные возможности EWB Kickstart, воспользовавшись evaluation-версией. Итак, необходимо установить одновременно обе версии, а затем из папок evaluation-версии в соответствующие папки kickstart-версии скопировать файлы с расширениями `.r43` и `.xcl`. После этого evaluation-версию можно смело удалять. При последующем запуске программы, в опциях проекта (раздел `XLINK`), необходимо указать конфигурационный файл `*.xcl`.

Перейдем к описанию основных окон, с которыми придется активно работать. При этом пока мы будем считать, что у нас уже есть исходные тексты программы для микроконтроллера. Итак, после установки пакета в главном меню будет создан раздел `IAR systems`, войдя в который следует запустить программу, наведя указатель на ярлык с названием `IAR Embedded Workbench`. После запуска на экране появится основное окно, показанное на рис. 2.1.

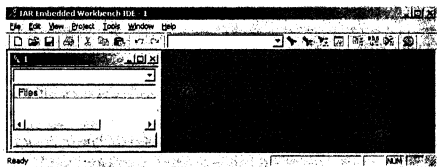


Рис. 2.1. Основное окно программы
IAR Embedded Workbench for MSP430

В левой части окна, обозначенной *workspace*, будет всегда отображаться структура того набора файлов, с которым пользователь в данный момент работает. Мы будем называть эту совокупность файлов *проектом*. Пока основное окно пусто, и чтобы начать создание программного обеспечения, нам нужно создать новый проект. Сделать это можно так: в меню *project* необходимо выбрать

пункт *create new project*, затем в открывшемся окне (в поле *project templates*) подсветить шаблон *empty project* и нажать кнопку *OK*. Программа откроет стандартное диалоговое окно, в котором необходимо выбрать на диске место, где предполагается проект разместить (или создать новую папку), а также набрать имя проекта, например, *project_01*. Теперь левая панель в основном окне пустовать не будет: в первой строчке появится название проекта и обозначение структуры проекта — по умолчанию устанавливается структура *debug* (отладка), но также может быть установлена и структура *release* (финальная). Пока мы не будем говорить о том, чем эти структуры отличаются, а просто запомним этот факт.

Звездочка справа от слова *debug* означает, что в проекте имеются изменения, которые не были сохранены на диске. В данном случае необходимо сохранить файлы только что созданного проекта. Для сохранения проекта в меню *file* выбираем пункт *save workspace* и сохраняем проект в файле с расширением *.eww*, например, *my_projects.eww*. Теперь звездочка исчезнет, а в выбранной для проекта папке появится несколько служебных файлов, в том числе и файл *project_01.ewp*, который содержит информацию о первом созданном проекте. Далее мы можем создавать через *create new project* новые проекты: *project_02.ewp*, *project_03.ewp* и т. д., сохраняя информацию в одном файле *my_projects.eww* через *save workspace*. Вы наверняка уже догадались, что файл *my_projects.eww* хранит структуру окна *workspace*. Обратите также внимание на то, что если открыто несколько проектов, в нижней части окна *workspace* появятся вкладки, выбирая которые можно просмотреть структуру каждого проекта.

Мы создали структуру проектов, теперь нужно узнать, как можно добавить к проекту файлы. С помощью функции поиска файлов необходимо найти в компьютере несколько файлов с расширением *.c* и скопировать их в папку с созданным проектом. К примеру, это будут файлы *tutor.c* и *utilities.c*. Теперь переходим в окне *workspace* к вкладке *project_01* и в меню *project* выбираем пункт *add files*. Через диалог ищем указанные выше файлы и добавляем их к указанному проекту. Те же файлы можно добавить и к остальным проектам. Результат добавления файлов показан на рис. 2.2. Файл можно удалить из проекта, щелкнув по его названию в окне проектов и выбрав пункт *remove*.

Следующий шаг — установка опций проекта. Установка может быть выполнена для всех файлов, имеющихся в данный момент в

окне *workspace*, а также для наименований проектов. Подсвечиваем в указанном окне *project_01* и выбираем в меню *project* пункт *options*. Появляется окно, показанное на рис. 2.3. Здесь достаточно много опций, и сразу разобраться во всем затруднительно. Поэтому воспользуемся стандартными установками.

Для шаблона *general options* выбираем тип микроконтроллера MSP430F149 (вкладка *target*), для поля *output file* опцию *executable* (вкладка *output*), для поля *library* в выпадающем списке — *CLIB* (вкладка *library configuration*). Остальные опции без изменений.

Опции C/C++ компилятора устанавливаются в том же окне (рис. 2.3) для шаблона C/C++ *compiler*. Здесь на вкладке *code* в поле *optimizations* из выпадающего списка выбирается *size/none (best debug support)*, на вкладке *output* устанавливается галочка *generate debug info*, на вкладке *list* — галочки для *output list file* и *assembler mnemonics*. Здесь можно также индивидуально настроить окно сообщений, которые будут выводиться при компиляции проекта, но эти настройки могут выполнить только опытные пользователи, поэтому мы не останавливаемся и переходим к рассказу о компиляции проектов.

Подсвечиваем файл *tutor.c* в окне *workspace*, затем в меню *project* выбираем пункт *compile*. В нижней части основного окна программы появилось еще одно окно *messages* с некоторым текстом. Мы видим, что компиляция файла не прошла, так как имеется ошибка (*fatal error [re005]*) — не найден включенный файл *tutor.h*. Что делать? С помощью поиска файлов находим данный файл, а также, заодно с ним, и файл *utilities.h*. Переносим эти файлы в папку проектов и заново запускаем компиля-

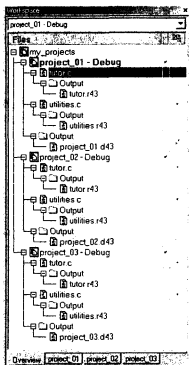


Рис. 2.2. Окно *workspace*

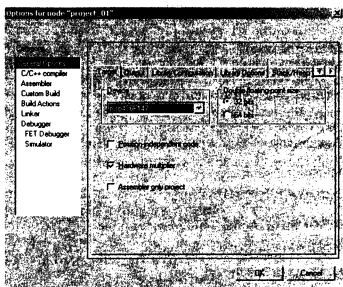


Рис. 2.3. Окно options

цию. Вид сообщений в окне будет таким, как показан на рис. 2.4. Он говорит о том, что компиляция прошла успешно, нет ошибок и предупреждений. Напомним, что предупреждение (warning) отличается от ошибки (error) тем, что не останавливает процесс компиляции, а лишь сообщает о возможном неустойчивом функционировании создаваемого программного обеспечения в некоторых ситуациях.

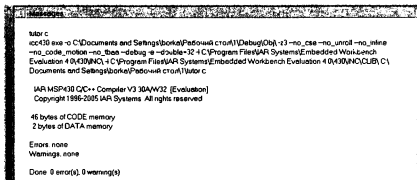


Рис. 2.4. Окно messages

Подробные результаты компиляции находятся в папке *debug* (создана автоматически), вложенной в папку проектов. В данной папке имеются каталоги *exe* (для загружаемых файлов), *list* (для файлов листинга) и *obj* (для объектных файлов). Объектные файлы имеют расширение *.r43* и используются как входные для программы IAR XLINK. Загружаемые файлы имеют расширение *.d43* и используются отладчиком IAR C-SPY. Каталог загружаемых файлов будет пуст, пока все объектные файлы не будут связаны в один загружаемый.

Не обязательно обращаться непосредственно к папкам, чтобы просмотреть их содержимое — достаточно продолжить работу с окном *workspace* и раскрыть структуру, называющуюся *output*. Эта структура создается для каждого входного файла, а содержимое файлов может быть отображено в левом окне.

Теперь настроим отображение листинга. Во-первых, в окне *workspace* двойным щелчком раскрываем файл листинга *utilities.lst*. Во-вторых, в меню *tools* выбираем пункт *options* и в открывшемся окне на вкладке *editor* ставим галочку в поле *scan for changed files*, нажимаем *OK*. В-третьих, выбираем отображение файла *utilities.c*, в меню *project* выбираем пункт *options*, в открывшемся окне подсвечиваем шаблон *C/C++ compiler*, ставим галочку в поле *override inherited settings* и на вкладке *code* в зоне *optimizations* из выпадающего списка выбираем *high (maximum optimization)*, нажимаем *OK* и вновь компилируем файл. Если мы теперь перейдем к файлу листинга, то увидим, что его содержимое автоматически изменилось при повторной компиляции, а также изменился размер выходного файла из-за изменения условий оптимизации при компиляции (размер можно посмотреть в конце файла листинга). Чтобы отключить оптимизацию, нужно вернуться к пункту *options*, на вкладке *code* в зоне *optimizations* из выпадающего списка выбрать *none (best debug support)*, снять галочку с *override inherited settings*, еще раз откомпилировать файл *utilities.c*.

Последнее, что необходимо выполнить при создании загрузочного файла, — это «собрать» его из объектных файлов с помощью IAR XLINK. Его настройкой мы сейчас и займемся. Подсвечиваем в окне *workspace* наш *project_1*, в меню *project* выбираем пункт *options* и в открывшемся окне подсвечиваем шаблон *linker*. В открывшихся опциях линкера не меняем ничего и нажимаем *OK*. Теперь включаем процесс сборки загрузочного файла, выбирая в меню *project* пункт *make*. В результате мы получим загрузочный

файл `project_1.d43`, который позже может быть загружен в отладчик IAR C-SPY Debugger, так как содержит всю необходимую отладочную информацию. Если же нужно дополнительно сгенерировать файл, который пригоден для непосредственной «загрузки» в память микроконтроллера, мы должны в шаблоне *linker* на вкладке *output* установить галочку в поле *allow C-SPY specific extra output format*, а также проверить наличие галочки в поле *generate extra output format* на вкладке *extra output*. Выбор формата генерируемого файла, который требуется для программатора, осуществляется из выпадающего списка *output format*. В результате этого действия будет дополнительно генерироваться файл `project_1.a43`.

2.2. IAR поможет в отладке

Первый шаг в науке создания программного обеспечения нами успешно пройден: мы научились транслировать исходные тексты и «собирать» загрузочные файлы. Причем эти операции помогают нам избегать так называемых синтаксических ошибок, то есть ошибок, связанных с форматом операторов и команд текстовой формы записи программы. К примеру, если мы ошибемся с буквой, не поставим точку, ошибемся со структурой подключаемых модулей, на этом этапе ошибки легко «отлавливаются» через листинг и быстро устраняются. Гораздо больших затрат времени и сил требует именно функциональная отладка, когда «вроде бы все работает», но «что-то не так». Понять, что именно «не так», поможет специальная программа-отладчик, которая также входит в комплект IAR EWB for MSP430 и называется C-SPY debugger. Именно в этой программе пользователь сможет наглядно увидеть, как происходит взаимодействие ядра микроконтроллера с периферией, как изменяется содержимое его регистров и ячеек памяти, как выполняются команды.

В предыдущем разделе нам удалось создать файл `project_1.d43`, который будет входным для C-SPY. Но прежде чем начать работу с отладчиком, рекомендуется его настроить. Для этого в окне *workspace* подсвечиваем *project_1* и в меню *project* выбираем пункт *options*. В открывшемся окне производим выбор шаблона *debugger*, затем в поле *driver* из выпадающего списка выбираем *simulator* (что соответствует виртуальной отладке программного обеспечения), ставим галочку в поле *run to main*, нажимаем *OK*. Наш отладчик настроен.

Теперь загружаем в отладчик наш проект. В меню workspace выбираем пункт debug и приступаем к изучению окна отладки, показанного на рис. 2.5. Сравните это окно с тем, что мы видели в основном режиме и убедитесь, что оно мало поменялось, разве что справа появилось окно *disassembly*, в котором представлен дизассемблированный (в мнемониках ассемблера) код программы.

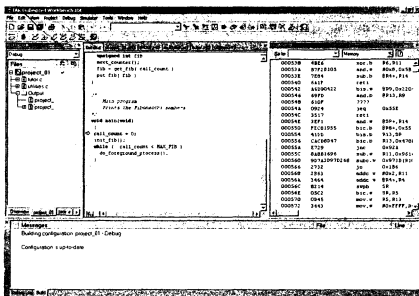


Рис. 2.5. Окно работы отладчика C-SPY

Чтобы просмотреть исходный текст программы, необходим двойной щелчок мыши на названии файла в окне *workspace*. При этом в центральном окне будет отображаться текст. Обратите внимание, что в файле `tutor.c` стрелка с левой рамки указывает на оператор `call_count`, подсвеченный зеленым цветом. В меню *debug* выбираем пункт *step over*, и стрелка переместится к оператору `init_fib`. Если мы еще раз выберем пункт *step over*, стрелка переместится к оператору `while`, а если выберем пункт *step into*, то попадем в подпрограмму (или функцию), которая размещается в файле `utilities.c`. Если есть необходимость выполнять программу по-операторно, нужно пользоваться пунктом *next statement*.

При отладке очень важно знать, какие величины имеют в данный момент те или иные переменные. Войдя в меню *view* и вы-

брав пункт *auto*, мы выведем на экран окно *auto* (рис. 2.6), в котором будут отображаться наименования переменных и их величины. Для этой цели также можно использовать окно *watch*, которое также можно вызвать, войдя в меню *view*. Необходимую переменную в это окно можно просто «перетащить» из окна исходного текста, выделив ее курсором мыши. Для удаления переменной из окна *watch* нужно просто выделить ее и нажать *delete*.

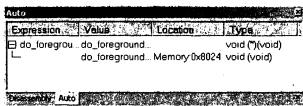


Рис. 2.6. Окно переменных *auto*

Обычно при отладке используется пошаговый метод выполнения программы. Но если пользователь уверен, что до определенного шага в программе нет ошибок, а исходные данные для дальнейших операций могут быть получены только в результате работы программы, он может задать точки останова и осуществить мониторинг данных в этих точках. Чтобы установить *breakpoint* (точку останова), необходимо перевести курсор на нужную строку в окне текста программы и в меню *edit* выбрать пункт *toggle breakpoint*. После этого в нужной строке появится слева красный крестик, а сама строка будет подсвечена красным. Повторный выбор этого пункта уничтожает точку останова. А запустить программу до точки останова поможет пункт *go* в меню *debug*.

Но не будем также забывать, что программа, написанная на языке высокого уровня (C/C++), на самом деле транслируется командами ассемблера. Текст в ассемблерных командах можно видеть в окне *disassembly*, которое расположено справа от окна исходного текста. Таким образом, у пользователя есть возможность осуществить пошаговое выполнение программы на уровне ассемблерных инструкций. Работа с окном *disassembly* осуществляется следующим способом: в меню *debug* выбираем пункт *reset* и, если окно *disassembly* не открыто, открываем его выбором опции *disassembly* в меню *view*. Дальнейшая работа с этим окном аналогична работе с окном исходного текста программы.

Отладчик позволяет просматривать и изменять содержимое регистров микроконтроллера, осуществляя полную эмуляцию их работы. Для этого в меню *view* нужно выбрать пункт *register*, и на экране появится окно, показанное на рис. 2.7. В верхней части этого окна имеется выпадающий список, с помощью которого мы выбираем группы тех регистров, которые нам необходимы в работе.

Можно также непосредственно работать с памятью микроконтроллера, просматривая ее содержимое и модифицируя данные. В меню *view* выбираем пункт *memory* — на экране мы увидим окно, показанное на рис. 2.8. В верхней части окна есть выпадающий список, с помощью которого выбираем вид памяти: *flash*, *sfr*, *ram*, *info*, *memory*. Окно разделено на две части. В левой располагаются коды, в правой — ASCII-символы, соответствующие этим кодам. Выделение курсором определенного дампа данных в той или иной части окна автоматически приводит к подсвечиванию аналогичного дампа в противоположной части окна. Изменить содержимое окна можно так: выделить нужный символ и заменить его.

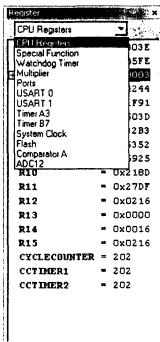


Рис. 2.7. Окно регистров *register*

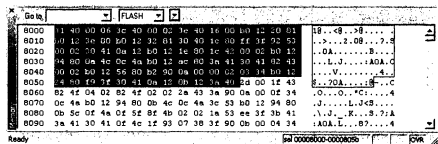


Рис. 2.8. Окно памяти *memory*

Окно операций ввода-вывода можно вызвать выбором пункта *terminal i/o* в меню *view*. А выйти из отладчика можно, выбрав пункт *stop debugging* в меню *debug*.

2.3. Альтернатива первая: IDE 430

Практически по всем параметрам эта среда может соперничать с вышеназванным продуктом: она гораздо проще в использовании, имеет в своем составе менеджер проектов, ассемблер, Си-компилятор и встроенный программатор. Правда, знатоки утверждают, что объем генерируемого загрузочного файла у этой программы несколько больше, чем у IAR, но это — если и беда, то очень незначительная. Более значительным недостатком для начинающего пользователя может считаться отсутствие отладчика и необходимость покупки лицензии, которая стоит \$99 за полнофункциональную версию программы и \$20 за получение обновлений в течение года. Процедура регистрации следующая: после установки программы в меню *options* выбирается пункт *register software* и в открывшейся форме в поле *your personal ID* фиксируется персональный номер продукта. Затем этот номер отправляется по указанному в форме адресу (естественно, нужно произвести и оплату).

В принципе, можно работать и с демонстрационной (незарегистрированной) версией IDE 430, но нужно помнить, что ограничения в данном случае будут такими: программа не сможет работать с текстами, в которых содержится более 10 вызовов функций и более 50 переменных. Также придется отлаживать программу непосредственно «в железе» — IDE 430 сможет «отловить» только синтаксические ошибки. Сайт производителя находится по адресу <http://www.ide430.com>, демонстрационная версия программы может «скачиваться» оттуда.

После установки программы и ее запуска из главного меню появится окно, изображенное на рис. 2.9. Теперь создаем в корневом каталоге папку *project* (замечено, что если в названии любой вложенной папки будут присутствовать русские буквы, компилятор будет выдавать сообщение об ошибке). Из папки, в которой находится IDE 430, переносим в созданную папку файлы *example2.c*, *sfr320.h*, *math.h*. После этого мы можем создать проект: в меню *project* выбираем пункт *new project*, даем ему имя (например, *project*) и сохраняем его в созданной директории — этим мы создадим

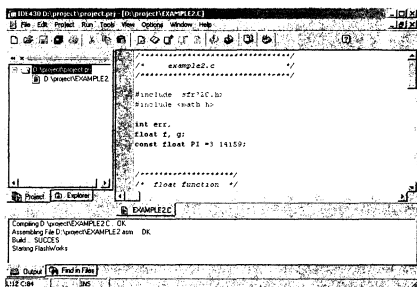


Рис. 2.9. Главное окно программы IDE 430

файл с расширением .prj. В левой панели окна появится менеджер проекта с необходимой папкой, к которой можно добавлять файлы (с расширениями .c или .asm) по щелчку правой кнопки мыши (выбор пункта *add files*).

Создание загрузочного файла производится в два этапа. Первая операция — ассемблирование/компиляция (она выполняется при выборе в меню *project* пункта *compile/assemble*). После выполнения этой операции будет создан файл с расширением .asm, в который войдут все модули программы. При успешном прохождении операции в log-панели, расположенной внизу окна, будут выведены сообщения «Compiling...OK» и «Assembling File...OK». Вторая операция — сборка (осуществляется выбором пункта *build* в меню *project*). После сборки будет создан загрузочный файл с расширением .txt, а в log-панели мы увидим сообщение «Build ... SUCCESS».

Созданный файл можно «зашить» по JTAG-интерфейсу (поддержка LPT) с помощью встроенной программы FlashWorks, внешний вид которой показан на рис. 2.10, а вызвать эту программу можно выбором пункта *flashworks* в меню *tools*. Назначение опций программы дополнительных комментариев не требует, особенно если вы внимательно читали предыдущую главу.

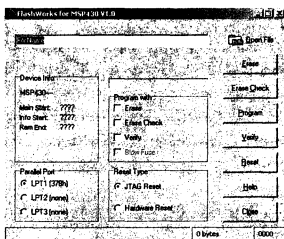


Рис. 2.10. Окно приложения FlashWorks к IDE 430

2.4. Альтернатива вторая: ICC 430

Разрабатывать программное обеспечение для MSP430 можно с помощью среды программирования ICC 430 от фирмы ImageCraft (адрес в Интернете <http://www.imagecraft.com/software>). Как и описанные выше, эта среда разработки имеет менеджер проектов, C-компилятор, ассемблер, линкер (сборщик), application builder (о нем чуть позже) и программатор. А если воспользоваться входящим в комплект поставки отладчиком NoICE Debugger 430, разработанным Джоном Хартманом (адрес в Интернете <http://www.noicedebugger.com>), то получим полноценную замену IAR. Но спешим предупредить читателя, что здесь также имеется отрицательная сторона: фирма бесплатно распространяет триал-версию (trial version) пакета программ с ограничением функционирования интервалом 45 суток и ограничением на объем генерируемого кода величиной 10 кб. Впрочем, если появится желание продлить период работы триал-версии, по адресу license@imagecraft.com можно связаться с разработчиками и продлить этот период.

Отладчик NoICE Debugger 430 имеет несколько иные условия использования — это evaluation-версия с ограничением периода работы до 30 суток. После окончания этого периода авторы рекомендуют позаботиться о приобретении лицензионной полной версии.

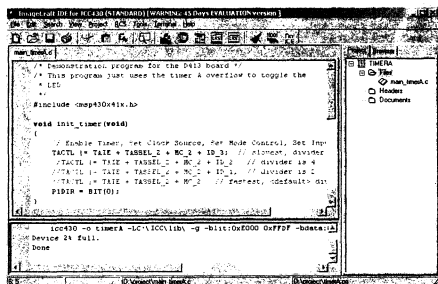


Рис. 2.11. Главное окно программы ICC 430

На рис. 2.11 показано главное окно программы ICC 430. Слева расположена панель исходного текста, справа — менеджер проекта. Работа с проектом осуществляется через меню *project*, в котором имеются пункты *new* (создать новый проект), *open* (открыть проект), *make* (компилировать проект). Загрузочный файл генерируется в формате *.hex*.

В комплекте ICC 430 имеются несколько дополнительных приложений, которые помогут разработчику в написании программного обеспечения. Во-первых, это ICC 430 application builder (рис. 2.12), который формирует конфигурацию для конкретного микроконтроллера. Вызов приложения осуществляется из меню *tools*.

Во-вторых, это программатор MSP 430 flash programmer (рис. 2.13). Программатор совместим со стандартным TI FET (LPT-порт).

И, наконец, отладчик NoICE Debugger 430, главное окно которого показано на рис. 2.14. В отладчик можно загрузить файл типа *.hex*, тогда будет осуществлено дизассемблирование загрузочного файла, то есть представление его в ассемблерных мнемониках. Здесь можно также просматривать текст в исходном виде, задавать точки останова, пошагово выполнять инструкции, наблюдать со-

стояние регистров. В нижней части окна имеются вкладки *data*, *output*, *view*, *watch*, *memory*, которые скомпонованы очень удачно.

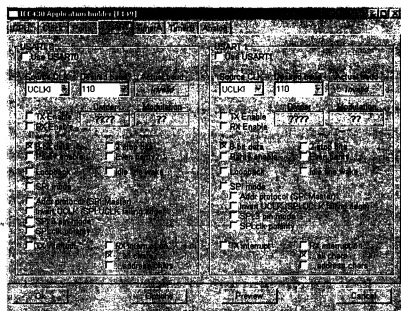


Рис. 2.12. Окно ICC 430 application builder

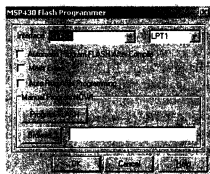


Рис. 2.13. Окно MSP 430 flash programmer

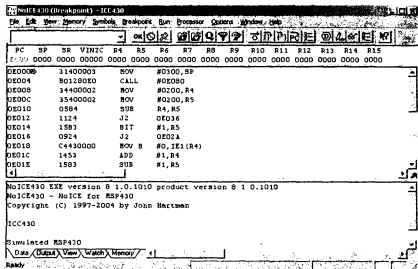


Рис. 2.14. Окно отладчика NoICE Debugger 430

2.5. Альтернатива третья: SBSIM 430

Для тех, кто хочет осваивать программирование MSP430 на языке высокого уровня, эта среда разработки мало чем сможет помочь — она имеет в своем составе только ассемблер, линкер, симулятор (отладчик) и менеджер проекта. Соответственно, работать придется с ассемблерными мнемониками — как при написании программы, так и при отладке ее. Мало того, среда поддерживает собственный программатор (от SoftBaugh) типа BLMSPF или PRGS430 и не имеет бесплатной версии. Демонстрационная версия ограничивается 90 сутками работы. Стоимость лицензионной версии составляет \$100 и \$10 за поддержку обновлений. Страницу производителя в Интернете можно найти по адресу <http://www.softbaugh.com>.

На рис. 2.15 показано главное окно программы SBSIM 430. Менеджер проекта и таблица состояния регистров находятся слева, на разных вкладках. Справа — панель исходного текста программы. Внизу — панель системных сообщений, состояния памяти и событий. Стандартный способ управления проектом осуществляется через меню *project*. Загрузочный файл формируется в двух вариантах — с расширением .hex и .txt.

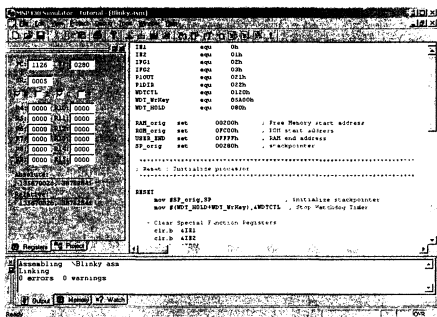


Рис. 2.15. Главное окно SBSIM 430

В целом работать с SBSIM 430 достаточно просто.

2.6. Альтернатива четвертая: CrossStudio for MSP 430

Альтернатива очень серьезная — эта среда разработки в своем составе имеет большой набор функций, способных полноценно заместить IAR: в ее составе есть менеджер проектов, ассемблер, C-компилятор, линкер, симулятор «ядра», поддержка всех известных программаторов и физических отладочных средств, богатые библиотеки математических функций и отличная справочная система. К сожалению, CrossStudio — это тоже коммерческий проект. Бесплатная evaluation-версия, которую можно «скачать» с сайта производителя <http://www.crossstudio.co.uk>, будет функционировать в течение 30 дней, но и эту версию нужно активировать, чтобы она смогла компилировать файлы. Делается это следующим образом: в меню *help* выбирается пункт *about crossStudio*

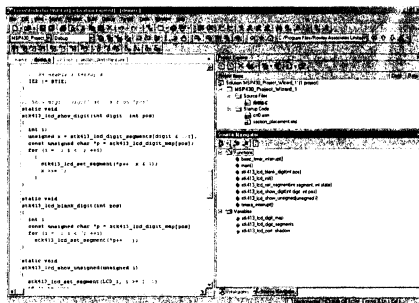


Рис. 2.16. Главное окно CrossStudio for MSP 430

и на вкладке *product activation* в открывшемся окне записывается *registration key*, который отправляется фирме-производителю. Фирма присылает *activation key*, по которому можно работать в течение пробного периода. Стоимость лицензионной версии составляет \$495.

В силу того, что принципы работы с этой программой очень похожи на те, которые были описаны ранее, мы не будем останавливаться на всех ее функциях, которых достаточно много. Внешний вид главного окна программы показан на рис. 2.16.

2.7. Альтернатива пятая: MSP GCC

Рекомендовать этот продукт начинающим освоение MSP430 очень бы не хотелось — во-первых, этот пакет программ не имеет стандартного менеджера проектов, а после установки «забывает» диск множеством исполняемых файлов, с которыми нужно работать «консольно». Во-вторых, отсутствует описание, которое позволяет достаточно просто «включиться» в работу. В-третьих,

структура исходного файла (то есть синтаксис) не совмещается с другими названными средами разработки. Однако «продвинутые» разработчики, которые работают с MSP430 долго и всерьез, утверждают, что данный продукт имеет наиболее высокие функциональные возможности при минимальных финансовых затратах: в пакет включена масса библиотек стандартных функций, библиотека для работы с плавающей арифметикой 32-разрядных чисел, библиотека огромного количества математических функций. Кроме этого, в пакете GCC есть ассемблер, дизассемблер, линкер, отладчик, выход на внутрисхемную отладку по JTAG, BSL-загрузчик.

Если у читателя будет желание, он может попробовать загрузить этот пакет программ с сайта <http://msp gcc.sourceforge.net> или <http://www.gnu.org>.

2.8. Альтернатива шестая: OCEAN

Этот продукт разработан фирмой Goepel Electronic (<http://www.goepel.com>), которая достаточно давно работает с интерфейсом JTAG. Показанный на рис. 2.17 OCEAN — это, собственно, симулятор, который моделирует работу процессорного ядра. Но к данному средству отладки прилагается также аппаратный адаптер, подключаемый к LPT-порту компьютера (рис. 2.18). При обращении программы к периферийным устройствам микроконтроллера MSP430 осуществляется эмуляция через JTAG, а отладочная среда визуально (даже в виде мнемосхем) отражает состояние микроконтроллера. Хотя в этой среде можно разрабатывать программное обеспечение, делать это возможно только в ассемблере — язык высокого уровня не поддерживается.

К недостаткам этого продукта можно отнести его высокую стоимость (порядка \$1300) при низких функциональных возможностях и устаревшем интерфейсе: OCEAN работает с операционными системами не новее Windows 98 (не поддерживается доступ к LPT-порту через драйвер). Поэтому данную среду разработки едва возможно рекомендовать современному разработчику.

2.9. Альтернатива седьмая: Project-430

И в заключение этой главы поговорим еще об одном пакете средств разработки программного обеспечения. Разговор наш будет

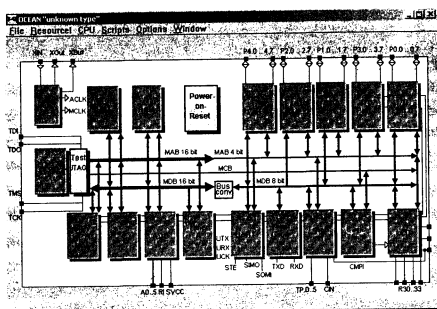


Рис. 2.17. Главное окно OCEAN

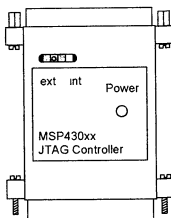


Рис. 2.18. Аппаратное сопряжение OCEAN с ПК

происходить под флагом «поддержите отечественного разработчика», так как Project-430 разработала и поставляет отечественная фир-

ма «Фитон» совместно с отечественной же фирмой «МикроКосм». Естественно, все программные продукты, входящие в этот пакет, являются коммерческими, но на сервере <ftp://ftp.phyton.ru/pub/ru/project430/updated/project430.exe> можно получить демонстрационную версию в виде однофайлового самораспаковывающегося архива. Сайт фирмы-поставщика находится здесь: <http://www.phyton.ru>.

Что такое Project-430? Это уже знакомый нам программно-аппаратный комплекс, состоящий из JTAG-отладчика, программно-го симулятора ядра, компилятора ассемблера, C-компилятора, текстового редактора, менеджера проектов и программатора. Вся структура пакета выполнена на современном уровне: синтаксические конструкции текста расцвечиваются разными цветами, переход из текстового режима в режим компиляции и отладки происходит автоматически, то есть не требует запускать дополнительные программы и открывать в них промежуточные файлы. Также предусмотрена совместимость с фирменными отладочными средствами типа MSP-FET 430. Но самое главное: предусмотрен русский интерфейс и справка пользователя на русском же языке. Это, конечно, большое подспорье для тех, кто с английским не знаком вовсе или знает его недостаточно хорошо.

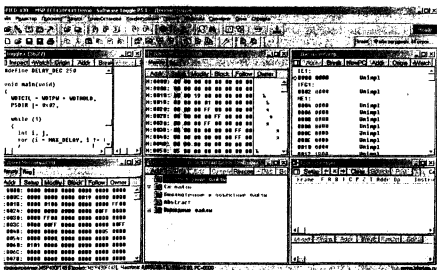


Рис. 2.19. Главное окно PDS-430

На рис. 2.19 показано главное окно отладчика-симулятора PDS-430, в составе которого имеются кросс-ассемблер MCA-430 и кросс-компилятор языка СИ типа MCC-430. Внешний вид отечественной среды разработки во многом напоминает продукт IAR. Компиляция и сборка программы здесь сопровождается подсвечиванием ошибочного синтаксиса и сообщениями в специальном окне, поэтому можно сразу обратиться к конкретной строке программы, в которой допущена ошибка (или ошибки).

Стоимость пакета и условия покупки лучше узнать непосредственно у производителя.

2.10. Что такое RTOS?

В этом небольшом подразделе мы поговорим о таком недавно появившемся инструменте разработчика, как многозадачная операционная система реального времени (RTOS). Читатель, который хотя бы немного знаком с принципом построения современных компьютерных многозадачных операционных систем, наверняка задумается: зачем микроконтроллеру операционная система, да еще и позволяющая обрабатывать несколько задач? Ведь обычно такие операционные системы требуют значительных аппаратных ресурсов — достаточно хотя бы вспомнить классические «окошки»...

Итак — зачем RTOS для микроконтроллера? Если разработчик программного обеспечения отлично ориентируется в ассемблере, он, при необходимости, без труда просчитает, какое время займет у него выполнение той или иной последовательности команд. Но такой подход хорош лишь для относительно несложных программ. Как же поступать, если программа сложная и разветвленная? И уж тем более сложно «считать», если эта программа написана на языке высокого уровня, в котором операторы эмулируются последовательностью ассемблерных команд, причем эмуляция выполняется самыми разнообразными методами. Здесь уж точно разработчик запутается...

Если возникает необходимость решения нескольких задач в условиях реального времени, да еще и на языке высокого уровня, выручит RTOS. Но не нужно думать, что RTOS, подобно классической операционной системе, загружается в микроконтроллер и где-то в его недрах ждет поступления задач. Это далеко не так: RTOS представляет собой набор файлов, которые на этапе на-

стройки среды программирования устанавливаются в соответствующую директорию и подключаются к программе через директиву *include*. После включения RTOS пользователю уже не нужно заботиться о выстраивании приоритетов прерываний, об организации задач. В откомпилированном файле уже будут содержаться необходимые задачи с соответствующими приоритетами, причем микроконтроллер будет следить, чтобы их приоритетность не нарушалась. По сути RTOS — это подключаемая библиотека с набором соответствующих функций, подобная, скажем, библиотеке с функцией извлечения квадратного корня или вычисления интеграла (что нельзя выполнить простейшим набором команд).

Начинающим работу с микроконтроллерами едва ли стоит углубляться во все особенности RTOS и усиленно изучать принципы работы с ней. В свое время, если такая необходимость появится, читатель сможет разыскать всю необходимую информацию в Интернете. И все же мы немного задержимся в этой области программирования.

За редчайшим исключением, все RTOS на сегодняшний момент представляют собой коммерческие продукты, то есть их придется приобретать за деньги, и, порой, немалые. Наиболее известен продукт RTOS от фирмы Pumpkin, называемый Salvo [<http://www.pumpkininc.com>]. Дистрибутив поставляется в виде самораспаковывающегося *exe*-архива, содержащего собственно *include*-файлы, техническую документацию, рекомендации по применению и другую полезную информацию. Salvo выпускается в пяти разных комплектациях. Также следует помнить, что для разных процессоров RTOS также должна быть разной, подобно тому, как классические «окошки» прекрасно работают на PC-совместимых машинах, но совершенно непригодны, скажем, для ZX-Spectrum. Предполагая использовать RTOS, всегда полезно знать, для какого микроконтроллера она написана.

Salvo Lite — это свободно распространяемая (оценочная) версия RTOS, которую можно «скачать» с сайта производителя бесплатно. Версия Lite может использоваться для создания работоспособного программного обеспечения с поддержкой не более 3 задач и не более 5 событий. Кроме этого, оценочная версия закрыта по исходному коду.

Salvo Tiny — версия начального уровня с ограниченным набором свойств, но с поддержкой неограниченного количества задач.

Ориентировочная стоимость этой версии составляет приблизительно \$200.

Salvo SE — версия среднего уровня. Набор свойств у SE версии больше, чем у версии Tiny, однако количество поддерживаемых задач составляет 7, а число событий — 12. Стоимость версии — \$450.

Salvo LE — так называемый «дистрибутив высокого уровня». Набор свойств еще более широкий, чем у SE, плюс — неограниченное количество задач и событий любого типа. Стоимость — \$750.

Salvo PRO — наиболее полная версия RTOS со всеми свойствами. Эта версия поставляется с исходными кодами и стоит \$1250.

Существует альтернативная RTOS для многих типов микроконтроллеров, в том числе и для MSP430, разработанная Краси-миром Костадиновым [<http://jacos.narod.ru>]. Называется эта система jacOS, и ее можно свободно скачать с упомянутого сайта (файл `libr_v1060_430.zip`).

Теперь поговорим об общих принципах организации RTOS. Основной элемент, с которым работает многозадачная система, — это *задача*. Следовательно, программист должен уметь организовывать задачи и запускать их на выполнение. Что такое задача? По своей сути задача — то же самое, что и любая другая функция, только имеющая процедуру вспомогательной инициализации, переключатели контекста и определенный приоритет. Запущенная задача может «крутиться» в цикле до бесконечности, пока ее не остановит другая задача и не возьмет управление на себя. Старая задача будет отложена.

В составе RTOS имеется так называемый планировщик заданий, который принимает решение о наиболее готовой к выполнению задаче и запускает ее на выполнение. После этого задача берет функции управления на себя и после выполнения должна вернуть эти функции планировщику, иначе другие задачи не получат шанса быть запущенными. Также планировщик отслеживает приоритет задач и по приоритету решает, что нужно запускать на выполнение. Если же несколько задач имеют один и тот же приоритет, планировщик будет «общаться» с ними последовательно, руководствуясь готовностью, — другими словами, он будет запускать их «по кругу».

Поясним сказанное на примере. Допустим, наш микроконтроллер должен считать импульсы с порта и выводить их на цифровой

индикатор. Соответственно, создается две задачи: чтение из порта и вывод на индикатор. Мы полагаем, что обе задачи имеют одинаковый приоритет. Здесь уже RTOS будет решать, что сделать сначала — читать или выводить. Решение будет таким: RTOS попеременно будет запускать задачу чтения и задачу вывода на индикатор.

Очень мощное средство RTOS, выручающее разработчика, — это возможность обмена задачами информацией. Для пояснения немного видоизменим задачу. Теперь нам необходимо, скажем, несколько раз прочесть информацию из порта, а затем единожды вывести ее на индикатор. Как здесь поступить? Вот для таких случаев RTOS имеет так называемые *двоичные семафоры* (они принимают значение 0 или 1), которые сигнализируют о необходимости переключения задач. Отдаленно семафоры напоминают флаги регистра статуса микроконтроллера, но организуются они программно. Кроме того, бывают также семафоры-счетчики и другие виды семафоров.

Значительное преимущество RTOS перед классическими способами программирования микроконтроллеров состоит в том, что любая задача запускается на выполнение при возникновении событий. Нет необходимости что-то «крутить» в длинных циклах с опросами готовности — налицо экономия ресурсов.

Еще одно интересное свойство RTOS — возможность задержать готовую задачу. Допустим, мы считываем с порта микроконтроллера импульсы, чтобы определить их частоту и вывести на индикатор. Если приоритет задачи вывода на индикатор окажется равным или даже более высоким, чем приоритет задачи подсчета импульсов, в считанные данные обязательно вкрадется временная ошибка. Мы обязаны задержать задачу вывода на индикатор, пока не истечет определенный промежуток времени подсчета импульсов. Задержка выполняется при помощи системного таймера RTOS.

Вне всякого сомнения, начинающему работать с MSP430 едва ли стоит сразу стартовать с глубокого изучения RTOS, но запомнить эту возможность нужно, чтобы когда-нибудь, когда «рука окрепнет», вернуться к ней.

Глава 3. Как оживить микроконтроллер

Итак, среда разработки программного обеспечения выбрана, установлена и освоена хотя бы на начальном этапе. Вы уже умеете создавать проекты из готовых файлов, компилировать их, линковать, загружать программу в микроконтроллер. Ну теперь-то, наконец, можно считать, что обучение закончено? Оказывается, нет: пока мы не сможем написать ни строчки текста программы самостоятельно, а если и напишем — транслятор сообщит о куче ошибок. А это значит, что нам нужно познакомиться с основными правилами управления микроконтроллером.

3.1. Мой адрес — не дом и не улица

Вернемся на некоторое время к первой главе и вспомним такое словосочетание, как «архитектура микроконтроллера». В ближайшее время мы будем постоянно обращаться к типовым элементам архитектуры MSP430 и учиться управлять ими.

Один из ключевых элементов архитектуры, который есть практически во всех разновидностях микроконтроллеров, — это *адресное пространство*. Вообще с этим понятием мы сталкиваемся ежедневно в повседневной жизни. Достаточно вспомнить хотя бы концертный зал, в пространстве которого каждый зритель адресуется к месту согласно купленному билету. Еще один пример — это поезд дальнего следования, в котором также занимают места строго определенным образом. Точно так же устроен и микроконтроллер: его адресное пространство сформировано таким образом, чтобы пользователь однозначно знал о назначении каждой ячейки.

В зрительном зале мы находим свое кресло, пользуясь трехмерной системой адресации: «партер-амфитеатр-балкон-ложа» (1), «ряд» (2), «место» (3). В микропроцессорной технике также встречается многомерность с адресацией памяти, но MSP430 — не тот случай. Этот микроконтроллер построен по так называемой фон-Неймановской схеме с единым адресным пространством, а значит, адресация конкретной ячейки осуществляется всего лишь в одномерной системе координат. Другими словами, каждая ячейка адресного пространства MSP430 имеет единственный уникальный идентификационный номер. Его принято называть *адресом*.

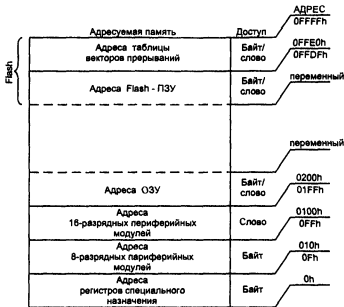


Рис. 3.1. Адресное пространство MSP430

Адресное пространство микроконтроллера MSP430 схематически показано на рис. 3.1. Оно состоит из 64 кб информационных ячеек, или, в пересчете в количественные показатели, — 65535. Каждая ячейка состоит из 8 бит. Хорошо видно, что в адресном пространстве можно получить доступ практически ко всем ресурсам микроконтроллера: здесь находятся:

а) адреса регистров специального назначения (SFR), которые используются при конфигурации отдельных периферийных устройств; регистры занимают адресное пространство с 0h до 0Fh и организованы *побайтно*; конкретные функции, закрепленные за SFR, можно узнать из непосредственного описания микроконтроллера;

б) адреса 8-разрядных периферийных модулей, занимающие адресное пространство от 010h до 0FFh и организованные *побайтно*;

в) адреса 16-разрядных периферийных модулей, занимающие адресное пространство от 0100h до 01FFh и организованные *пословно*;

г) адреса ОЗУ, начинающиеся от адреса 0200h и организованные *пословно/побайтно*; адрес другой границы ОЗУ не фиксирован

и может отличаться в зависимости от наименования микроконтроллера;

д) адреса flash-ПЗУ, оканчивающиеся адресом 0FFFFh и организованные *пословно/побайтно*; начальный адрес flash-ПЗУ также не фиксирован — он разный для каждого микроконтроллера;

е) в адресном пространстве flash-ПЗУ адреса с 0FFE0h по 0FFFFh занимает так называемая таблица векторов прерываний, при этом вектор с наивысшим приоритетом закреплен за самым последним словом адресного пространства (0FFFEh).

Сразу же запомните, что поскольку MSP430 является 16-разрядным устройством, он может работать как с байтами, так и со словами, имеющими *младший байт* и *старший байт*. Соответственно, команды МК могут обращаться к адресному пространству и побайтно, и пословно. Здесь есть одно важное правило: младший байт всегда располагается по четному адресу, а старший байт в следующем за ним нечетном адресе. К примеру, если младший байт слова размещен по адресу xxx8h, то старший байт слова нужно всегда искать по адресу xxx9h.

3.2. Их всего шестнадцать, и все они в ЦПУ

Кого всего шестнадцать? Да регистров центрального процессора! Просьба не путать их с регистрами специального назначения, расположенными в адресном пространстве памяти. Это — именно *регистры ЦПУ*, отдельные 16-разрядные ячейки, подключенные с помощью шины данных памяти (MDB) и шины адресов памяти (MAB) к адресному пространству памяти, а с другой стороны — к арифметико-логическому устройству (АЛУ). Регистры ЦПУ имеют обозначения R0...R15, причем регистры R0...R3 не могут быть использованы произвольно, поэтому они называются *специальными регистрами ЦПУ*. Регистры R4...R15, напротив, заданы как *регистры ЦПУ общего назначения* и предоставлены в полное распоряжение пользователя.

Поговорим подробнее об этих регистрах.

Регистр R0 носит название *программный счетчик* (PC/R0). Собственно, этот 16-разрядный регистр хранит адрес следующей команды, которая подлежит выполнению микроконтроллером. Именно по содержимому программного счетчика ЦПУ «узнает»,

что ему выполнять далее. Во время выполнения очередной команды в счетчик заносится адрес следующей команды и процесс ее отработки не прерывается.

Регистр R1 именуется *указатель стека* (SP/R1). Стекковая память напоминает корзину с бельем: чем больше в корзину положили белья, тем выше стопка. В основном стекковую память используют для хранения адресов возврата из подпрограмм и прерываний — при завершении работы подпрограммы (или обработчика прерывания) сохраненный в стеке адрес возврата переносится в программный счетчик.

Регистр R2 — третий важный специальный регистр, называемый *регистром статуса* (SR/R2). Содержимое этого регистра, во-первых, управляет режимами функционирования ЦПУ, а во-вторых, по его содержимому можно узнавать результат выполнения команды и принимать решения о дальнейшей работе программы. Некоторые биты регистра R2 зарезервированы на будущее (это биты 15...9), остальные задействованы.

- Бит «0» (C) — бит переноса, который устанавливается, если при выполнении операции с байтами или словами образуется перенос (то есть появление единицы в следующей позиции разрядной сетки).
- Бит «1» (Z) — бит нуля. Этот бит устанавливается, когда в результате операции с байтами или словами результат равен нулевому.
- Бит «2» (N) — бит отрицательного результата. Устанавливается, если при выполнении операции с байтами или словами образуется отрицательный результат.
- Бит «3» (GIE) — бит разрешения прерываний. При установке бита разрешаются маскируемые прерывания (источник — периферийные устройства), при сбросе бита все маскируемые прерывания запрещаются.
- Бит «4» (CPUOFF) — бит выключения ЦПУ. Когда этот бит установлен, ЦПУ отключено. Этот бит в основном используется для организации режимов малого энергопотребления.
- Бит «5» (OSCOFF) — бит выключения осциллятора (тактового генератора). Когда этот бит установлен, работа генератора от LFXT1 запрещается. Этот бит также используется для режима малого энергопотребления.
- Бит «6» (SCG0) — бит выключения системного тактового генератора «0».

- Бит «7» (SCG1) — бит выключения системного тактового генератора «1».
- Бит «8» (V) — бит переполнения. Этот бит устанавливается, если в результате математических операций над числами образуется переполнение, например, при сложении двух положительных чисел получается отрицательный результат.

У регистра R2 есть альтернативная функция — так называемый *генератор констант* (R2/CG1). Точно такую же функцию выполняет регистр R3 — это тоже *генератор констант* (R3/CG2), но, в отличие от R2, у регистра R3 альтернативных функций не существует. Как же удастся задействовать альтернативную функцию? Дело в том, что константы генерируются путем изменения способа адресации регистра (адресуются в неявном виде). А что дает нам наличие генератора констант? Возможность поддержки ассемблером MSP430 дополнительных 24 эмулированных команд к 27 физическим. Напомним, что физические команды декодируются ЦПУ по уникальному коду операции (КОП), а эмулированные команды не имеют собственного КОП и при трансляции заменяются эквивалентными физическими командами. Однако эмулированные команды удобно использовать при разработке программного обеспечения на ассемблере.

Остальные регистры (R4...R15) не несут конкретных функций и могут быть использованы в качестве ячеек для хранения любых данных, значений адресов или индексов. Кроме этого, они доступны как в случае работы со словами, так и с байтами.

3.3. И снова об адресации

Здесь мы поговорим о том, каким образом работать с данными в адресном пространстве, как их можно найти, извлечь, выполнить некоторые операции и снова передать в память. По-другому, мы будем говорить о режимах адресации операндов источника и получателя.

Разработчики MSP430 предусмотрели семь режимов адресации операнда-источника и четыре — операнда-получателя. Для операнда-источника предусмотрены: регистровый режим, индексный режим, символьный режим, безусловный (абсолютный) режим, косвенный регистровый режим, косвенный автоинкремент, прямой

режим. Для операнда-получателя: регистровый режим, индексный режим, символьный режим, безусловный (абсолютный) режим.

Регистровый режим — самый простой способ адресации. Если мы хотим переслать данные из регистра R9 в регистр R10 (как в регистр R9 эти данные попали — вопрос другой), то выполняем команду ассемблера:

```
MOV R9, R10
```

Эта команда не меняет содержимое R9, но изменит содержимое R10. Кроме того, значение программного счетчика PC будет увеличено на 2.

Индексный режим — несколько более сложный для понимания. Здесь мы оперируем уже не непосредственно с данными, а с адресами. Синтаксис команды ассемблера, соответствующий этому режиму, будет такой:

```
MOV X(R9), Y(R10)
```

где X и Y — целые числа.

Так вот, в регистре R9 теперь содержится некоторый адрес, который в сумме с числом X дает адрес операнда-источника в адресном пространстве микроконтроллера, то есть адрес того, что нужно перемещать. В регистре R10 также содержится некий адрес, который в сумме с числом Y дает адрес, куда нужно поместить операнд.

Символьный режим — немного напоминает индексный режим в том смысле, что оперировать приходится не с самими данными, а с их адресами. Синтаксис команды ассемблера:

```
MOV EDE, TONI
```

Какие действия выполнит эта команда? Она скопирует операнд, находящийся по адресу (EDE-PC) в ячейку с адресом (TONI-PC), где PC — содержимое знакомого нам программного счетчика.

Абсолютный режим — можно считать разновидностью символьного режима с той лишь разницей, что здесь не учитывается содержимое PC, а пересылка операнда будет осуществлена с адреса EDE на адрес TONI. Синтаксис команды ассемблера в этом режиме:

```
MOV &EDE, &TONI
```

Этот режим адресации рекомендуется использовать для аппаратной периферии, адреса регистров которой в адресном пространстве фиксированы, чтобы программное обеспечение, разра-

ботанное для одного типа микроконтроллера, могло быть использовано и для другого с минимальной доработкой.

Косвенный регистровый режим — близок к индексному режиму, однако может работать как с байтами, так и со словами. Он тоже осуществляет пересылку содержимого с исходного адреса (R9) по адресу назначения (R10), но действителен только для операнда источника. Синтаксис команды ассемблера:

```
MOV.B @R9, 0(R10)
```

Здесь символ «В» свидетельствует о проведении операции над байтом. После выполнения операции содержимое регистров не изменяется.

Косвенный автоинкрементный режим — аналогичен косвенному регистровому режиму с той лишь разницей, что содержимое регистра-источника инкрементируется (увеличивается) после выполнения операции на 1 для байтов и на 2 для слов. Синтаксис команды ассемблера:

```
MOV.B @R9+, 0(R10)
```

Данный режим удобен для обработки массивов однотипных данных (таблиц), так как указание последующего адреса происходит без дополнительных ресурсных затрат.

Прямой режим — осуществляет непосредственную пересылку константы, находящейся в теле команды, по адресу назначения, указанному также в команде. Синтаксис команды ассемблера:

```
MOV #36h, TONI
```

Здесь #36h — константа, которая пересылается по адресу метки TONI.

3.4. Стоит только командовать

Как мы уже говорили, основа работы любого микроконтроллера — это выполнение последовательности простейших команд. Чтобы научиться «командовать», нужно узнать, какие именно команды понимает микроконтроллер.

Итак, как мы уже ранее сказали, полный набор команд ассемблера MSP430 содержит 27 основных команд, поддерживаемых микроконтроллером аппаратно, и дополнительные 24 команды, которые микроконтроллером физически не поддерживаются, но

эмулируются компилятором. Как это понимать? Очень просто: при трансляции эмулированная команда будет автоматически заменена на эквивалентную из набора основных команд. В принципе, при разработке программного обеспечения можно обходиться только основными командами, записывая соответствующую текстовую конструкцию, но это не слишком разумный подход. Эмулированные команды облегчают написание типовых операций, а значит, разработчику не придется постоянно обращаться к справочнику.

Мнемоники команд MSP430 приведены в табл. 3.1. Следует сразу же запомнить, что все команды можно разделить на три вида: команды с двумя операндами, команды с одним операндом и безоперандные команды (команды переходов). Отличие их состоит в том, что в двухоперандных командах участвуют регистр-источник (обозначается *src*) и регистр-получатель (обозначается *dst*), в однооперандных командах — только регистр-источник или только регистр-приемник, а команды переходов операндов вообще не имеют — они выполняют операцию перехода программы к заранее заданной метке по некоторым условиям, о которых мы поговорим чуть позже. Эмулированные команды могут также не содержать операндов, однако не нужно относить такие команды к безоперандным, поскольку они могут эмулироваться одно- и двухоперандными основными командами. В табл. 3.1 для таких команд указывается количество операндов в мнемонике и через знак «/» — количество реальных операндов. Можно заметить, что эмулированные команды позволяют сократить количество операндов, если не физически, то хотя бы в тексте программы. Эмулированные команды не увеличивают размер кода и не снижают производительности микроконтроллера.

Таблица 3.1

Набор команд MSP430

Мнемоника	Выполняемая операция	Вид	Количество операндов
ADC	Сложение флага C с получателем	эмул	1/2
ADD	Сложение источника с получателем	осн	2
ADDC	Сложение источника и флага C с получателем	осн	2
AND	Операция «И» источника и получателя	осн	2
BIC	Очистка битов в получателе	осн	2

Мнемоника	Выполняемая операция	Вид	Количество операндов
BIS	Установка битов в получателе	осн	2
BIT	Проверка битов в получателе	осн	2
BR, BRANCH	Переход по назначению	эмул	1/2
CALL	Вызов получателя	осн	1
CLR	Очистка получателя	эмул	1/2
CLRC	Очистка флага C	эмул	-/2
CLRN	Очистка флага N	эмул	-/2
CLRZ	Очистка флага Z	эмул	-/2
CMP	Сравнение источника и получателя	осн	2
DADC	Десятичное сложение флага C с получателем	эмул	1/2
DADD	Десятичное сложение источника и флага C с получателем	осн	2
DEC	Декремент получателя	эмул	1/2
DECD	Двойной декремент получателя	эмул	1/2
DINT	Запрещение прерываний	эмул	-/2
EINT	Разрешение прерываний	эмул	-/2
INC	Инкремент получателя	эмул	1/2
INCD	Двойной инкремент получателя	эмул	1/2
INV	Инвертирование получателя	эмул	1/2
JC/JNC	Переход, если C уст/если наивысший или такой же	осн	0
JEQ/JZ	Переход если равно/переход, если Z установлен	осн	0
JGE	Переход, если больше или равно	осн	0
JL	Переход, если меньше	осн	0
JMP	Переход простой	осн	0
JN	Переход, если флаг N установлен	осн	0
JNC/JLO	Переход, если C не установлен/переход, если низший	осн	0
JNE/JNZ	Переход, если не равно/переход, если Z не установлен	осн	0
MOV	Пересылка источника в получатель	осн	2
NOP	Нет операции	эмул	-/2
POP	Вызов элемента из стека в получатель	эмул	1/2

Мнемоника	Выполняемая операция	Вид	Количество операндов
PUSH	Помещение источника в стек	осн	1
RET	Возврат из подпрограммы	эмул	-/2
RETI	Возврат из прерывания	осн	1
RLA	Арифметическая ротация влево	эмул	1/2
RLC	Ротация влево через флаг C	эмул	1/2
RRA	Арифметическая ротация вправо	осн	1
RRC	Ротация вправо через флаг C	осн	1
SBC	Вычитание инвертированного C из получателя	эмул	1/2
SETC	Установка флага C	эмул	-/2
SETN	Установка флага N	эмул	-/2
SETZ	Установка флага Z	эмул	-/2
SUB	Вычитание источника из получателя	осн	2
SUBC	Вычитание источника и инв. флага C из получателя	осн	2
SWPB	Обмен байтов в слове	осн	1
SXT	Распространение знака	осн	1
TST	Проверка получателя	эмул	1/2
XOR	Исключающее ИЛИ источника и получателя	осн	2

Мнемоники, связанные с операндами, могут иметь префикс «В» или «W», также префикс может вообще отсутствовать, например, MOV.B, MOV.W, MOV. Эти префиксы означают разрядность операндов. Префикс «В» означает, что операнды имеют размерность байта (*byte*), а префикс «W» или его отсутствие — что операнды имеют разрядность слова (*word*). Выполнение команд может, как правило, сопровождаться изменением состояния флагов в регистре статуса, и это обстоятельство нужно учитывать.

* * *

Команда ADC осуществляет операцию сложения флага переноса C с операндом получателя (*dst*), при этом предыдущее содержимое получателя утрачивается, поскольку туда записывается результат. Команда влияет на флаги регистра статуса: флаг N устанавливается, если результат отрицательный, и сбрасывается, если результат положительный; флаг Z устанавливается, если результат нулевой, во всех других случаях сбрасывается; флаг C устанавли-

вается, если результат был инкрементирован от значения 0FFFFh к 00000h, во всех других случаях сбрасывается; флаг V устанавливается при арифметическом переполнении; содержимое битов OSCOFF, CPUOFF и GIE не изменяется.

Синтаксис команды:

```
ADC dst  
ADC.W dst  
ADC.B dst
```

Данная команда является эмулированной, то есть ее эквивалент из набора основных команд будет ADDC #0,dst или ADDC.B #0,dst.

* * *

Команда ADD осуществляет операцию сложения содержимого источника с содержимым получателя. При этом значение операнда источника не изменяется, а предыдущее содержимое операнда получателя теряется, поскольку туда записывается результат. Команда влияет на флаги регистра статуса: флаг N устанавливается, если результат отрицательный, и сбрасывается, если результат положительный; флаг Z устанавливается, если результат нулевой, и сбрасывается во всех других случаях; флаг C устанавливается, если в результате выполняется перенос, и сбрасывается, если перенос отсутствует; флаг V устанавливается в результате арифметического переполнения, во всех остальных случаях не изменяется; содержимое битов OSCOFF, CPUOFF и GIE не изменяется.

Синтаксис команды:

```
ADD src,dst  
ADD.W src,dst  
ADD.B src,dst
```

Команда является аппаратной для MSP430.

* * *

Команда ADDC осуществляет операцию сложения операнда источника с флагом C и операндом приемника. При этом значение операнда источника не изменяется, а предыдущее содержимое операнда теряется, поскольку туда записывается результат. Команда влияет на флаги регистра статуса: флаг N устанавливается, если результат отрицательный, и сбрасывается, если результат положительный; флаг Z устанавливается, если результат нулевой, и сбрасывается во всех других случаях; флаг C устанавливается, если произошел перенос из старшего бита результата, и сбрасыва-

ется, если переноса не было; флаг V устанавливается, если произошло арифметическое переполнение, и сбрасывается, если арифметического переполнения не было; содержимое битов OSCOFF, CPUOFF и GIE не изменяется.

Синтаксис команды:

```
ADDC src,dst
ADDC.W src,dst
ADDC.B src,dst
```

Команда является аппаратной для MSP430.

Команда **AND** осуществляет операцию логического «И» источника и получателя. При этом над операндом источника и операндом получателя выполняется функция логического умножения, значение операнда источника не меняется, результат записывается на место операнда получателя. Команда влияет на флаги регистра статуса: флаг N устанавливается, если в результате выполнения команды устанавливается старший бит результата, и сбрасывается, если старший бит результата не устанавливается; флаг Z устанавливается, если результат нулевой, и сбрасывается во всех других случаях; флаг C устанавливается, если результат выполнения команды не нулевой, во всех других случаях флаг сбрасывается; флаг V сбрасывается во всех случаях; содержимое битов OSCOFF, CPUOFF и GIE не изменяется.

Синтаксис команды:

```
AND src,dst
AND.W src,dst
AND.B src,dst
```

Команда является аппаратной для MSP430.

Команда **BIC** осуществляет очистку битов получателя. При этом инвертированный операнд источника и операнд получателя перемножаются по правилу логического умножения, а результат записывается на место операнда получателя, значение операнда источника не изменяется. Команда не влияет на флаги регистра статуса, также не меняется содержимое битов OSCOFF, CPUOFF и GIE.

Синтаксис команды:

```
BIC src,dst
BIC.W src,dst
```


BIC.B src,dst

Команда является аппаратной для MSP430.

* * *

Команда **BIS** выполняет операцию логического «ИЛИ» между операндом источника и операндом получателя. При этом результат пересылается в операнд получателя, а операнд источника не изменяется. Команда не влияет на флаги регистра статуса, также не меняется содержимое битов OSCOFF, CPUOFF и GIE.

Синтаксис команды:

BIS src,dst

BIS.W src,dst

BIS.B src,dst

Команда является аппаратной для MSP430.

* * *

Команда **BIT** осуществляет операцию проверки битов получателя. При этом над операндом источника и операндом получателя выполняется операция логического «И», но это не отражается на их содержимом, а результат можно узнать только в регистре статуса. Флаг N устанавливается, если установлен старший бит результата, в противном случае сбрасывается. Флаг Z устанавливается, если результат нулевой, иначе сбрасывается. Флаг C устанавливается, если результат не нулевой, иначе сбрасывается. Флаг V сбрасывается всегда. Не меняется содержимое битов OSCOFF, CPUOFF и GIE.

Синтаксис команды:

BIT src,dst

BIT.W src,dst

BIT.B src,dst

Команда является аппаратной для MSP430.

* * *

Команда **BR** осуществляет безусловный переход в любое место адресного пространства памяти микроконтроллера с использованием любого из названных способов адресации. При этом биты в регистре статуса не изменяются.

Синтаксис команды:

BR dst

Команда является эмулированной командой MSP430. Ее аппаратный эквивалент может быть записан как:

MOV dst, PC

Примечание — другая запись этой команды имеет ключевую мнемонику **BRANCH**. Соответственно действие команды **BRANCH** такое же, как и команды **BR**, разница состоит только в записи.

* * *

Команда **CALL** осуществляет вызов подпрограммы, причем вызов может производиться по любому адресу в пределах адресного пространства памяти микроконтроллера и любыми названными способами адресации. Адрес возврата из подпрограммы автоматически сохраняется в стеке. Команда не влияет на биты регистра статуса.

Синтаксис команды:

CALL dst

Команда является аппаратной для MSP430.

* * *

Команда **CLR** выполняет операцию очистки операнда получателя. При этом биты в регистре статуса не изменяются.

Синтаксис команды:

CLR dst

CLR.W dst

CLR.B dst

Команда является эмулированной командой MSP430 и может быть записана в аппаратном эквиваленте так:

MOV #0, dst

MOV.W #0, dst

MOV.B #0, dst

В результате выполнения команды в операнд получателя заносится нулевое значение.

* * *

Команда **CLRC** очищает бит **C** в регистре статуса. Значения остальных флагов в регистре статуса сохраняются. Не меняется содержимое битов **OSCOFF**, **CPUOFF** и **GIE**.

Синтаксис команды:

CLRC

Команда является эмулированной командой MSP430 и может быть записана в аппаратном эквиваленте так:

BIC #1, SR

В результате выполнения команды флаг C приобретает нулевое значение.

* * *

Команда **CLRN** очищает бит N в регистре статуса. Значения остальных флагов в регистре статуса сохраняются. Не меняется содержимое битов OSCOFF, CPUOFF и GIE.

Синтаксис команды:

CLRN

Команда является эмулированной командой MSP430 и может быть записана в аппаратном эквиваленте так:

BIC #4, SR

В результате выполнения команды флаг N приобретает нулевое значение.

* * *

Команда **CLRZ** очищает бит Z в регистре статуса. Значения остальных флагов в регистре статуса сохраняются. Не меняется содержимое битов OSCOFF, CPUOFF и GIE.

Синтаксис команды:

CLRZ

Команда является эмулированной командой MSP430 и может быть записана в аппаратном эквиваленте так:

BIC #2, SR

В результате выполнения команды флаг Z приобретает нулевое значение.

* * *

Команда **CMP** осуществляет операцию операнда источника и операнда получателя. При этом операнд источника вычитается из операнда получателя, результат вычитания отражается только на состоянии регистра статуса. Значения операндов после выполнения команды не меняются. В регистре статуса флаг N устанавливается, если результат отрицательный, и сбрасывается — если положительный; флаг Z устанавливается, если результат равен нулевому, во всех остальных случаях сбрасывается; флаг C устанавливается, если произошел перенос из старшего бита результата, и сбрасывается, если переноса не было; флаг V устанавливается, если произошло арифметическое переполнение результата,

во всех остальных случаях сбрасывается. Не меняется содержимое битов OSCOFF, CPUOFF и GIE.

Синтаксис команды:

```
CMP src,dst
CMP.W src,dst
CMP.B src,dst
```

Команда является аппаратной для MSP430.

* * *

Команда DADC осуществляет десятичное сложение флага переноса C с операндом получателя. При этом флаг N устанавливается, если значение старшего разряда результата равно 1; флаг Z устанавливается, если результат нулевой; флаг C устанавливается, если операнд получателя был инкрементирован от 270Fh (9999) к 0000h (0000) в случае операции со словом или инкрементируется от 63h (99) к 00h (00) в случае операции с байтом; состояние флага V не может быть четко зафиксировано. Не меняется содержимое битов OSCOFF, CPUOFF и GIE.

Синтаксис команды:

```
DADC dst
DADC.W dst
DADC.B dst
```

Команда является эмулированной командой MSP430 и может быть записана в аппаратном эквиваленте так:

```
DADD #0,dst
DADD.B #0,dst
```

Следует заметить, что к операнду суммируется флаг переноса, который был зафиксирован в регистре статуса *до* выполнения команды, а не *после*.

* * *

Команда DADD выполняет операцию десятичного сложения операнда источника, операнда приемника и флага переноса. При этом операнды источника и получателя обрабатываются как четыре BCD (двоично-десятичных) числа с положительными знаками. Флаг переноса и значение операнда источника суммируются с операндом получателя, результат фиксируется в операнде получателя, значение операнда источника не изменяется. Команда влияет на флаги в регистре статуса: флаг N устанавливается, если старший разряд результата равен 1; флаг Z устанавливается, если резу-

лътат нулевой; флаг C устанавливается, если результат превышает 9999 (при работе с операндами-словами) или 99 (при работе с операндами-байтами); значение флага V не фиксируется. Не меняется содержимое битов OSCOFF, CPUOFF и GIE.

Синтаксис команды:

```
DADD src,dst  
DADD.W src,dst  
DADD.B src,dst
```

Команда является аппаратной для MSP430.

* * *

Команда **DEC** выполняет операцию декремента операнда получателя. При выполнении команды операнд получателя уменьшается на 1 (декрементируется), а предыдущее значение операнда теряется. Команда влияет на флаги регистра статуса: флаг N устанавливается, если результат выполнения команды отрицательный, и сбрасывается — если результат положительный; флаг Z устанавливается, если операнд до выполнения команды содержал единицу (нулевой результат выполнения команды), и сбрасывается в любом другом случае; флаг C сбрасывается, если операнд содержал ноль (отрицательный результат выполнения команды), и устанавливается во всех других случаях; флаг V устанавливается, если исходное значение операнда получателя было 8000h (при операциях со словами) или 80h (при операциях с байтами), во всех остальных случаях сбрасывается. Не меняется содержимое битов OSCOFF, CPUOFF и GIE.

Синтаксис команды:

```
DEC dst  
DEC.W dst  
DEC.B dst
```

Команда является эмулированной для MSP430 и может быть записана в аппаратном эквиваленте так:

```
SUB #1,dst  
SUB.B #1,dst
```

Эта команда часто используется для пересылок однотипных данных внутри памяти микроконтроллера. Необходимо следить, чтобы области операндов-источников и операндов-приемников не накладывались при пересылках.

* * *

Команда **DECD** выполняет операцию двойного декремента операнда получателя. При выполнении команды операнд получателя уменьшается на 2 (дважды декрементируется), а предыдущее значение операнда теряется. Команда влияет на флаги регистра статуса: флаг **N** устанавливается, если результат выполнения команды отрицательный, и сбрасывается — если результат положительный; флаг **Z** устанавливается, если операнд до выполнения команды содержал двойку (нулевой результат выполнения команды), и сбрасывается в любом другом случае; флаг **C** сбрасывается, если операнд содержал нуль (отрицательный результат выполнения команды), и устанавливается во всех других случаях; флаг **V** устанавливается, если исходное значение операнда получателя было 8001h, а также 8000h (при операциях со словами) или 81h, а также 80h (при операциях с байтами), во всех остальных случаях сбрасывается. Не меняется содержимое битов **OSCOFF**, **CPUOFF** и **GIE**.

Синтаксис команды:

```
DECD dst
DECD.W dst
DECD.B dst
```

Команда является эмулированной для **MSP430** и может быть записана в аппаратном эквиваленте так:

```
SUB #2, dst
SUB.B #2, dst
```

Понятно, что эта команда также может быть эмулирована двойным выполнением команды **DEC**, правда, тогда вырастет объем исходного кода и увеличится время выполнения операции.

* * *

Команда **DINT** запрещает все прерывания. При этом флаги в регистре статуса не изменяются, бит **GIE** сбрасывается. Не меняется содержимое битов **OSCOFF** и **CPUOFF**.

Синтаксис команды:

```
DINT
```

Команда является эмулированной для **MSP430** и может быть записана в аппаратном эквиваленте так:

```
BIC #8, SR
```

Физически команда выполняется так: константа 08h инвертируется и логически перемножается с содержимым регистра статуса **SR**, результат помещается в регистр статуса.

Производители рекомендуют при использовании команды DINT записывать в тексте программы после нее команду NOP (нет операции), чтобы гарантированно защитить последующие инструкции от прерываний.

* * *

Команда EINT разрешает выполнение прерываний. При этом флаги в регистре статуса не изменяются, бит GIE устанавливается. Не меняется содержимое битов OSCOFF и CPUOFF.

Синтаксис команды:

EINT

Команда является эмулированной командой MSP430 и может быть записана в аппаратном эквиваленте так:

BIS #8,SR

Физически команда выполняется так: константа 08h и содержимое регистра статуса SR складываются логически, результат помещается в регистр статуса. Следует также запомнить, что команда, следующая в программе за командой EINT, будет выполнена в любом случае, даже если поступивший во время запрета запрос на обработку прерывания ожидает разрешения прерываний. Иными словами, желательно выполнить после разрешения прерываний команду NOP (нет операции).

* * *

Команда INC выполняет операцию инкремента операнда получателя. При выполнении команды операнд получателя увеличивается на единицу (инкрементируется), а предыдущее значение операнда теряется. Команда влияет на флаги регистра статуса: флаг N устанавливается, если результат выполнения команды отрицательный, и сбрасывается — если результат положительный; флаг Z устанавливается, если операнд до выполнения команды содержал FFFFh (операнд-слово), а также FFh (операнд-байт), и сбрасывается в любом другом случае; флаг C устанавливается, если операнд до выполнения команды содержал FFFFh (операция с операндом-словом), а также FFh (операция с операндом-байтом), и сбрасывается в любом другом случае; флаг V устанавливается, если операнд до выполнения команды содержал 7FFFh (операция с операндом-словом), а также 7Fh (операция с операндом-байтом), и сбрасывается в любом другом случае. Не меняется содержимое битов OSCOFF, CPUOFF и GIE.

Синтаксис команды:

```
INC dst
INC.W dst
INC.B dst
```

Команда является эмулированной для MSP430 и может быть записана в аппаратном эквиваленте так:

```
ADD #1, dst
```

Нетрудно заметить, что команда INC противоположна команде DEC.

* * *

Команда **INCD** выполняет операцию двойного инкремента получателя, то есть увеличение содержимого операнда получателя на два. При выполнении команды исходное содержимое получателя теряется. Команда влияет на флаги регистра статуса: флаг N устанавливается, если результат выполнения команды отрицательный, и сбрасывается — если результат положительный; флаг Z устанавливается, если операнд до выполнения команды содержал **FFFEh** (операнд-слово), а также **FEh** (операнд-байт), и сбрасывается в любом другом случае; флаг C устанавливается, если операнд до выполнения команды содержал **FFFEh** или **FFFFh** (операция с операндом-словом), а также **FEh** или **FFh** (операция с операндом-байтом), и сбрасывается в любом другом случае; флаг V устанавливается, если операнд до выполнения команды содержал **7FFEh** или **7FFFh** (операция с операндом-словом) а также **7Fh** или **7FFh** (операция с операндом-байтом) и сбрасывается в любом другом случае. Не меняется содержимое битов **OSCOFF**, **CPUOFF** и **GIE**.

Синтаксис команды:

```
INCD dst
INCD.W dst
INCD.B dst
```

Команда является эмулированной для MSP430 и может быть записана в аппаратном эквиваленте так:

```
ADD #2, dst
ADD.B #2, dst
```

Команда **INCD** противоположна команде **DECD**.

* * *

Команда **INV** осуществляет инвертирование операнда получателя, при этом предыдущее содержимое операнда получателя теряется. Команда влияет на флаги регистра статуса: флаг **N** устанавливается, если результат выполнения команды отрицательный, и сбрасывается в остальных случаях; флаг **Z** устанавливается, если операнд содержал значение **FFFFh** (операнд-слово) или **FFh** (операнд-байт), и сбрасывается в остальных случаях; флаг **C** устанавливается, если результат нулевой; флаг **V** устанавливается, если исходное содержимое операнда приемника было отрицательное, в остальных случаях флаг сбрасывается. Не меняется содержимое битов **OSCOFF**, **CPUOFF** и **GIE**.

Синтаксис команды:

```
INV dst
INV.W dst
INV.B dst
```

Команда является эмулированной для **MSP430** и может быть записана в аппаратном эквиваленте так:

```
XOR #0FFFFh, dst
XOR.B #0FFh, dst
```

Команда **INV** устанавливает единицы в тех битах операнда, где были нули, и сбрасывает нули там, где были единицы.

* * *

Команда **JC** осуществляет операцию перехода при условии установки переноса. Близкая к ней команда **JHS** осуществляет переход, называемый в технической документации «по условию наивысший (higher) или равный». Выбор той или иной команды определяется в каждом конкретном случае. Команда **JHS** позволяет сравнивать два беззнаковых числа в диапазоне от 0 до 65536 по условию «больше или равно». Действие этих команд следующее: проверяется флаг переноса **C** регистра статуса и по результатам проверки принимается решение. Если флаг установлен, 10-разрядная величина смещения (с учетом знака) из команды прибавляется к программному счетчику и происходит переход. Если же флаг переноса **C** сброшен, команда не выполняет переход. Команды не влияют на состояние флагов в регистре статуса.

Синтаксис команды:

```
JC label
JHS label
```

Здесь **label** — это метка, к которой происходит переход по условию.

Команды являются аппаратными для MSP430.

* * *

Команда **JEQ** и эквивалентная ей **JZ** осуществляют переходы по условиям «если равно» и «если ноль». В данном случае проверяется флаг **Z** регистра статуса. Если флаг установлен, 10-разрядная величина смещения (с учетом знака) из команды прибавляется к программному счетчику и происходит переход. Если же флаг **Z** сброшен, команда не выполняет переход. Команда **JEQ** позволяет сравнить два числа и осуществить переход, если эти числа равны. Команды не влияют на состояние флагов в регистре статуса.

Синтаксис команды:

JEQ label

JZ label

Команды являются аппаратными для MSP430.

* * *

Команда **JGE** осуществляет переход по условию «если больше или равно». В данном случае проверяются флаги **N** и **V** регистра статуса. Если оба флага установлены или сброшены, 10-разрядная величина смещения (с учетом знака) из команды прибавляется к программному счетчику и происходит переход. Если установлен или сброшен только один флаг, переход не выполняется. Данная команда позволяет осуществлять сравнение двух чисел со знаком. Команда не влияет на состояние флагов в регистре статуса.

Синтаксис команды:

JGE label

Команда является аппаратной для MSP430.

* * *

Команда **JL** осуществляет переход по условию «если меньше». В данном случае проверяются флаги **N** и **V** регистра статуса. Если один из флагов установлен, 10-разрядная величина смещения (с учетом знака) из команды прибавляется к программному счетчику и происходит переход. Если установлены или сброшены оба флага, переход не выполняется. Данная команда позволяет осуществлять сравнение двух чисел со знаком. Команда не влияет на состояние флагов в регистре статуса.

Синтаксис команды:

JL label

Команда является аппаратной для MSP430.

* * *

Команда **JMP** осуществляет безусловный переход к указанной метке. При выполнении команды 10-разрядная величина смещения (с учетом знака) из команды прибавляется к программному счетчику и происходит переход. Нетрудно заметить, что данная команда заменяет эмулированную инструкцию **BRANCH**. Команда не влияет на состояние флагов в регистре статуса.

Синтаксис команды:

JMP label

Команда является аппаратной для MSP430.

* * *

Команда **JN** осуществляет переход по условию «если не равно». При выполнении команды проверяется флаг **N** в регистре статуса. Если флаг установлен, 10-разрядная величина смещения (с учетом знака) из команды прибавляется к программному счетчику и происходит переход. Если флаг **N** сброшен, переход не выполняется. Команда не влияет на состояние флагов в регистре статуса.

Синтаксис команды:

JN label

Команда является аппаратной для MSP430.

* * *

Команда **JNC** осуществляет переход по условию «если перенос не установлен». При выполнении команды проверяется флаг **C** в регистре статуса. Если флаг **C** сброшен, 10-разрядная величина смещения (с учетом знака) из команды прибавляется к программному счетчику и происходит переход. Если флаг **C** установлен, переход не выполняется. Эту команду можно использовать для сравнения беззнаковых чисел в диапазоне от 0 до 65536. Команда не влияет на состояние флагов в регистре статуса.

Синтаксис команды:

JNC label

Команда является аппаратной для MSP430.

* * *

Команда **JNE** и близкая к ней команда **JNZ** осуществляют переход по условиям «если не равно» и «если не ноль». При выпол-

нении команды проверяется флаг Z в регистре статуса. Если флаг Z сброшен, 10-разрядная величина смещения (с учетом знака) из команды прибавляется к программному счетчику и происходит переход. Если флаг Z установлен, переход не выполняется. Команда JNE может осуществлять сравнение двух чисел. Команды не влияют на состояние флагов в регистре статуса.

Синтаксис команды:

```
JNE label
JNZ label
```

Команды являются аппаратными для MSP430.

* * *

Команда MOV осуществляет операцию пересылки содержимого операнда источника в операнд получателя. При выполнении команды операнд источника не изменяется, а предыдущее содержимое операнда получателя теряется. Команда не влияет на флаги регистра статуса, содержимое битов OSCOFF, CPUOFF и GIE не изменяется.

Синтаксис команды:

```
MOV src,dst
MOV.W src,dst
MOV.B src,dst
```

Команда является аппаратной для MSP430.

* * *

Команда NOP не выполняет никакой операции и используется для организации временных задержек (если это необходимо) или для отладки программного обеспечения. Команда не влияет на флаги регистра статуса, содержимое битов OSCOFF, CPUOFF и GIE не изменяется.

Синтаксис команды:

```
NOP
```

Команда является эмулированной для MSP430 и может быть записана в аппаратном эквиваленте так:

```
MOV #0,R3
```

Также команда может быть эмулирована и другими способами с разными временными задержками. Об этих способах можно узнать в оригинальной технической документации slau049d.pdf. Фирма-производитель призывает пользоваться эмуляцией с осто-

рожностью, так как в некоторых случаях можно нарушить функционирование некоторых периферийных устройств.

* * *

Команда **POP** выполняет операцию пересылки слова или байта в операнд получателя. Действие команды следующее: содержимое стека, на которое указывает **TOS** (указатель стека), перемещается в ячейку операнда получателя, а затем указатель стека увеличивается на два. Команда не влияет на флаги регистра статуса, содержимое битов **OSCOFF**, **CPUOFF** и **GIE** не изменяется.

Синтаксис команды:

```
POP dst
POP.W dst
POP.B dst
```

Команда является эмулированной для **MSP430** и может быть записана в аппаратном эквиваленте так:

```
MOV @SP+,dst
MOV.W @SP+,dst
MOV.B @SP+,dst
```

Необходимо запомнить, что указатель системного стека **SP** всегда инкрементируется на два, даже если осуществляется извлечение байта (инструкция с префиксом **.B**).

* * *

Команда **PUSH** помещает операнд источника в стек. При выполнении команды указатель стека **SP** декрементируется на два, затем содержимое операнда источника заносится в ячейку стековой памяти, адрес которой указан в **TOS**. Команда не влияет на флаги регистра статуса, содержимое битов **OSCOFF**, **CPUOFF** и **GIE** не изменяется.

Синтаксис команды:

```
PUSH src
PUSH.W src
PUSH.B src
```

Как и в предыдущем случае, необходимо запомнить, что указатель системного стека **SP** всегда декрементируется на два, даже если осуществляется пересылка байта (инструкция с префиксом **.B**).

Команда является аппаратной для **MSP430**.

* * *

Команда **RET** осуществляет операцию возврата из подпрограммы. Выполняется она так: адрес возврата из подпрограммы, помещенный в стек инструкцией **CALL**, пересылается в программный счетчик, затем программа продолжает выполняться с адреса кода, следующего за адресом команды вызова подпрограммы. Команда не влияет на флаги регистра статуса, содержимое битов **OSCOFF**, **CPUOFF** и **GIE** не изменяется.

Синтаксис команды:

RET

Команда является эмулированной для **MSP430** и может быть записана в аппаратном эквиваленте так:

MOV @SP+, PC

Не следует путать эту команду с командой возврата из прерывания, описание которой приводится ниже.

* * *

Команда **RETI** осуществляет операцию возврата из прерывания. При выполнении команды происходит следующее: регистр статуса восстанавливает все флаги в исходном состоянии (которое было до обработки прерывания) из стека; указатель стека **SP** инкрементируется на два; счетчик команд также восстанавливает исходное значение (до обработки прерывания). Содержимое битов **OSCOFF**, **CPUOFF** и **GIE** из системного стека не восстанавливается.

Синтаксис команды:

RETI

Команда является аппаратной для **MSP430**.

* * *

Команда **RLI** осуществляет операцию арифметической ротации «вправо». При этом содержимое операнда получателя сдвигается по разрядной сетке влево на одну позицию, старший бит «вдвигается» во флаг **C** регистра статуса, а во младший бит операнда заносится ноль. Если значение операнда более **4000h** (операнд-слово), а также более **40h** (операнд-байт) или меньше **C000h** (операнд-слово), а также **C0h** (операнд-байт), происходит переполнение (устанавливается флаг **V**). Команда влияет на другие флаги регистра статуса, кроме уже упомянутых флагов **C** и **V**: флаг **N** устанавливается, если результат выполнения команды отрицате-

льный; флаг Z устанавливается при нулевом результате. Содержимое битов OSCOFF, CPUOFF и GIE не изменяется.

Синтаксис команды:

```
RLA dst  
RLA.W dst  
RLA.B dst
```

Команда является эмулированной командой MSP430 и может быть записана в аппаратном эквиваленте так:

```
ADD dst, dst  
ADD.B dst, dst
```

Действие команды по-другому называется «умножение со знаком 2».

* * *

Команда RLC выполняет операцию арифметической ротации «влево через перенос». При выполнении команды содержимое операнда получателя сдвигается побитно влево на один бит, при этом содержимое флага переноса C заносится в младший бит операнда, а старший бит операнда сдвигается в ячейку флага переноса C. Если значение операнда более 4000h (операнд-слово), а также более 40h (операнд-байт) или меньше C000h (операнд-слово), а также C0h (операнд-байт), происходит переполнение (устанавливается флаг V). Команда влияет на другие флаги регистра статуса, кроме флагов C и V: флаг N устанавливается, если результат выполнения команды отрицательный; флаг Z устанавливается при нулевом результате. Содержимое битов OSCOFF, CPUOFF и GIE не изменяется.

Синтаксис команды:

```
RLC dst  
RLC.W dst  
RLC.B dst
```

Команда является эмулированной для MSP430 и может быть записана в аппаратном эквиваленте так:

```
ADDC dst, dst
```

Применение этой команды имеет незначительные оговорки, с которыми можно познакомиться в оригинальной технической документации.

* * *

Команда **RRA** выполняет операцию арифметической ротации «вправо». Действие команды следующее: содержимое операнда получателя сдвигается вправо (побитно) на один бит, старший бит сохраняется и вдвигается в соседний бит, младший бит сдвигается в ячейку флага переноса **C**. Команда влияет на флаги регистра статуса: флаг **N** устанавливается, если результат операции отрицательный; флаг **Z** устанавливается, если результат нулевой, флаг **V** сбрасывается. Содержимое битов **OSCOFF**, **CPUOFF** и **GIE** не изменяется.

Синтаксис команды:

```
RRA dst
RRA.W dst
RRA.B dst
```

Команда является основной для **MSP430**.

* * *

Команда **RRC** осуществляет ротацию «влево через перенос». При выполнении команды содержимое операнда получателя сдвигается вправо на один бит (побитно), флаг переноса **C** занимает место старшего бита операнда, а младший бит операнда вдвигается в ячейку флага переноса **C**. Команда влияет на флаги регистра статуса: флаг **N** устанавливается, если результат операции отрицательный; флаг **Z** устанавливается, если результат нулевой, флаг **V** устанавливается, если исходное содержимое операнда положительно и флаг переноса перед выполнением команды установлен. Содержимое битов **OSCOFF**, **CPUOFF** и **GIE** не изменяется.

Синтаксис команды:

```
RRC dst
RRC.W dst
RRC.B dst
```

Команда является основной для **MSP430**.

* * *

Команда **SBC** выполняет операцию вычитания инвертированного переноса или операнда получателя. При выполнении команды флаг переноса **C** суммируется со значением операнда источника, уменьшенного на единицу, предыдущее содержимое операнда источника теряется. Команда влияет на флаги регистра статуса: флаг **N** устанавливается, если результат операции отрицательный; флаг **C** устанавливается, если имеется перенос из старшего разряда результата или если есть заем; флаг **Z** устанавливается, если ре-

зультат нулевой, флаг V устанавливается, если произошло арифметическое переполнение. Содержимое битов OSCOFF, CPUOFF и GIE не изменяется.

Синтаксис команды:

```
SBC dst  
SBC.W dst  
SBC.B dst
```

Команда является эмулированной для MSP430 и может быть записана в аппаратном эквиваленте так:

```
SUBC #0, dst  
SUBC.B #0, dst
```

В данном случае инверсия флага переноса реализует процедуру заема.

* * *

Команда SETC реализует операцию установки флага переноса C. Содержание остальных флагов не изменяется. Содержимое битов OSCOFF, CPUOFF и GIE также не изменяется.

Синтаксис команды:

```
SETC
```

Команда является эмулированной для MSP430 и может быть записана в аппаратном эквиваленте так:

```
BIS #1, SR
```

Эта команда дополняет команду CLRC.

* * *

Команда SETN реализует операцию установки флага N. Содержание остальных флагов не изменяется. Содержимое битов OSCOFF, CPUOFF и GIE также не изменяется.

Синтаксис команды:

```
SETN
```

Команда является эмулированной командой MSP430 и может быть записана в аппаратном эквиваленте так:

```
BIS #2, SR
```

Эта команда дополняет команду CLRN.

* * *

Команда SUB выполняет операцию вычитания операнда источника из операнда получателя. При выполнении команды операнд источника не изменяется, а предыдущее значение операнда получа-

теля теряется. Команда влияет на флаги регистра статуса: флаг N устанавливается при отрицательном результате выполнения команды; флаг Z устанавливается при нулевом результате; флаг C устанавливается при переносе из старшего бита результата либо если нет заема; флаг V устанавливается при арифметическом переполнении. Содержимое битов OSCOFF, CPUOFF и GIE не изменяется.

Синтаксис команды:

```
SUB src,dst
SUB.W src,dst
SUB.B src,dst
```

Команда является основной для MSP430.

Команда SUBC (или ее мнемонический эквивалент SBB) выполняет операцию вычитания операнда источника и инвертированного флага переноса C из операнда получателя. При выполнении команды значение операнда источника не меняется, а предыдущее содержимое операнда получателя теряется. Команда влияет на флаги регистра статуса: флаг N устанавливается при отрицательном результате выполнения команды; флаг Z устанавливается при нулевом результате; флаг C устанавливается при переносе из старшего бита результата либо если нет заема; флаг V устанавливается при арифметическом переполнении. Содержимое битов OSCOFF, CPUOFF и GIE не изменяется.

Синтаксис команды:

```
SUBC src,dst
SUBC.W src,dst
SUBC.B src,dst
```

Для мнемонического эквивалента:

```
SBB src,dst
SBB.W src,dst
SBB.B src,dst
```

Команда является основной для MSP430.

Команда SWPB выполняет операцию обмена байтами в слове операнда получателя. При выполнении команды биты 15...8 слова занимают место битов 7...0, а бывшие на этом месте биты 7...0 занимают освободившиеся биты 15...8. Команда не влияет на флаги

регистра статуса, содержимое битов OSCOFF, CPUOFF и GIE также не изменяется.

Синтаксис команды:

SWPB dst

Команда является основной для MSP430.

* * *

Команда SXT выполняет операцию, называемую «распространение знака». Что это такое? Как известно, знак двоичного числа хранится в старшем разряде (например, в 7-м, если это 0 байт). При выполнении команды седьмой бит слова копируется в ячейки 8...15 — это и есть «распространение знака». Команда влияет на флаги регистра статуса: флаг N устанавливается при отрицательном результате выполнения команды; флаг Z устанавливается при нулевом результате; флаг C устанавливается при ненулевом результате; флаг V сбрасывается всегда. Содержимое битов OSCOFF, CPUOFF и GIE не изменяется.

Синтаксис команды:

SXT dst

Команда является основной для MSP430.

* * *

Команда TST осуществляет операцию «проверка получателя». Здесь содержимое операнда получателя сравнивается с нулем и согласно этому устанавливаются флаги в регистре статуса: флаг N устанавливается при отрицательном результате выполнения команды; флаг Z устанавливается при нулевом результате; флаг C устанавливается всегда; флаг V сбрасывается всегда. Содержимое битов OSCOFF, CPUOFF и GIE не изменяется.

Синтаксис команды:

TST dst

TST.W dst

TST.B dst

Команда является эмулированной командой MSP430 и может быть записана в аппаратном эквиваленте так:

CMP #0, dst

CMP.B #0, dst

При выполнении этой команды содержимое операнда получателя не изменяется.

* * *

Команда **XOR** выполняет операцию «исключающее ИЛИ» между содержимым операнда источника и операнда получателя. Результат выполнения команды помещается в ячейку операнда получателя, его предыдущее значение теряется, а содержимое операнда источника не изменяется. Команда влияет на флаги регистра статуса: флаг **N** устанавливается при отрицательном результате выполнения команды (установлен старший разряд); флаг **Z** устанавливается при нулевом результате; флаг **C** устанавливается при ненулевом результате; флаг **V** устанавливается, если оба операнда отрицательные.

Синтаксис команды:

```
XOR src,dst
XOR.W src,dst
XOR.B src,dst
```

Команда является основной для MSP430.

Итак, вы познакомились с полным набором ассемблерных команд MSP430. Этот перечень позволит тем, кто уже немного работал с другими микроконтроллерами, оценить описываемую здесь архитектуру, прикинуть открывающиеся перспективы. Ну а тот, кто знакомится с микроконтроллерами, что называется, «с чистого листа», теперь знает о принципах управления микроконтроллером и, в принципе, может начинать практически работать с ним.

3.5. А что дальше?

Собственно, на этом можно было бы поставить точку и сказать, что первое знакомство читателя с микроконтроллерами MSP430 состоялось: в общих чертах перестала быть загадкой архитектура, известен набор аппаратных средств программирования и отладки, описаны программаторы и среды разработки программного обеспечения, как на ассемблере, так и на языке высокого уровня. Что дальше? А дальше, если читатель решил для себя, что ему стоит продолжить знакомство с MSP430, возможно несколько вариантов. Первый вариант — потратить время на поиск в Интернете описанных прикладных программ, зайти на сайт фирмы-производителя, получить фирменную документацию, заняться ее изучением «в подлиннике». Второй вариант — приобрести книги [11],

[12], [13], [14], в которых фирменная документация уже переведена на русский язык, подключиться к обсуждению на конференции [<http://сахара.ru/echo/msp.html>], где «тусуются» те, кто непосредственно занят разработкой устройств на MSP430. Но есть и третий вариант — дождаться выхода из печати второго издания книги «Микроконтроллер MSP430: первое знакомство». Это издание сейчас готовится к выходу.

Какие дополнительные сведения читатель сможет найти во втором издании? Будет рассказано о тактировании, об аппаратном умножении чисел, о таймерах, о встроенных периферийных устройствах, о режимах пониженного энергопотребления. Кроме этого, в соответствующей главе будет приведена типовая структура программы и рассказано о последовательности команд, о включаемых файлах. Отдельную главу предполагается посвятить несложным практическим конструкциям, которые пригодятся и радиолюбителям, и профессионалам. Дополнительно предполагается включить в состав книги компакт-диск, на котором терпеливые читатели найдут свободно распространяемые версии прикладных пакетов, рисунки печатных плат в электронном виде, «прошивки» и необходимые для углубленного знакомства файлы документации по применению MSP430. Пока это издание не вышло из печати, автор приглашает читателей принять участие в его подготовке: советами, предложениями, критическими замечаниями. Также пригодятся и практические материалы по этому микроконтроллеру: описания несложных практических конструкций, рекомендации по применению, рассказы «из опыта». Самые интересные материалы обязательно будут опубликованы в книге с указанием авторства. Материалы можно направлять по адресу электронной почты borka@mail.ru с указанием в теме сообщения «для книги по MSP430».

Литература

- [1] Фрунзе А. В. Микроконтроллер — это просто. В 3-х тт.
- [2] Бродин В. Б., Шагурин И. И. Микроконтроллеры: архитектура, программирование, интерфейс. — М.: ЭКОМ, 1999.
- [3] Козаченко В. Ф. Руководство по применению 16-разрядных микроконтроллеров INTEL MCS-196/296 во встроенных системах управления. — М.: ЭКОМ, 1997.
- [4] Гладштейн М. А. Микроконтроллеры семейства Z86 фирмы Zilog. Руководство программиста. — М.: Додэка, 1999.
- [5] Сайт фирмы «Microchip» <http://www.microchip.com/>.
- [6] Заец Н. Электронные самоделки. Для быта, отдыха и здоровья. — М.: СОЛОН-Пресс, 2004.
- [7] Сайт фирмы «Atmel» <http://www.atmel.com/>.
- [8] Голубцов М. С. Микроконтроллеры AVR: от простого к сложному. — М.: СОЛОН-Пресс, 2003.
- [9] Семенов Б. Ю. Современный тюнер своими руками: УКВ Сте-рео + микроконтроллер. — М.: Солон-Р, 2001. (2-е изд. 2004)
- [10] Семенов Б. Ю. Шина I²C в радиотехнических конструкциях. — М.: Солон-Р, 2002. (2-е изд., дополненное, 2004).
- [11] Семейство микроконтроллеров MSP430x1xx. Руководство пользователя. — М.: ЗАО «Компэл», 2005.
- [12] Семейство микроконтроллеров MSP430x2xx. Руководство пользователя. — М.: ЗАО «Компэл», 2005.
- [13] Семейство микроконтроллеров MSP430x4xx. Руководство пользователя. — М.: ЗАО «Компэл», 2005.
- [14] Семейство микроконтроллеров MSP430. Рекомендации по применению. — М.: ЗАО «Компэл», 2005.
- [15] Диск CD-ROM «MSP430: описание, руководство пользователя, рекомендации по применению». Версия 2.1. — М.: ЗАО «Компэл», 2005.
- [16] Диск CD-ROM «MSP430 ultra-low-power microcontrollers». v1.3. — М.: ЗАО «Сканти-Рус», 2003.
- [17] Диск CD-ROM «MSP430 ultra-low-power microcontrollers». v.slac001G. — М.: ЗАО «Компэл», 2003.
- [18] Диск CD-ROM «MSP430 ultra-low-power microcontrollers». v.slac001H. — М.: ЗАО «Компэл», 2004.
- [19] Журнал «Электронные компоненты» № 5, 2002.
- [20] Chris Nagy «Embedded system design using the TI MSP430 Series». TI Inc, Dallas, 2003.

Содержание

Предисловие	3
Глава 1. Что понадобится в первую очередь	7
1.1. Что такое — MSP430?	7
1.2. Знакомство с архитектурой MSP430	11
1.3. Как «прошить» MSP430	18
1.4. Прошивка МК с использованием BSL	27
1.5. Прошивка МК с использованием JTAG	45
Глава 2. Среда — не только день недели	65
2.1. Начать можно с IAR	65
2.2. IAR поможет в отладке	73
2.3. Альтернатива первая: IDE 430	77
2.4. Альтернатива вторая: ICC 430	79
2.5. Альтернатива третья: SBSIM 430	82
2.6. Альтернатива четвертая: CrossStudio for MSP 430	83
2.7. Альтернатива пятая: MSP GCC	84
2.8. Альтернатива шестая: OCEAN	85
2.9. Альтернатива седьмая: Project-430	85
2.10. Что такое RTOS?	88
Глава 3. Как оживить микроконтроллер	92
3.1. Мой адрес — не дом и не улица	92
3.2. Их всего шестнадцать, и все они в ЦПУ	94
3.3. И снова об адресации	96
3.4. Стоит только скомандовать	98
3.5. А что дальше?	123
Литература	125

Магазины Москвы и городов Московской области

Торговый Дом «Библио-Глобус»

Адрес ул. Мясницкая, д 6
Тел 928-35-67

Магазин

«Московский Дом Книги»

Адрес Новый Арбат, д 8
Тел 203-82-42, 291-78-32

Магазин

«Дом технической книги»

Адрес Ленинский пр-т, д 40
Тел 137-60-38, 137-60-39

Магазин «Молодая Гвардия»

Адрес ул Б Полянка, д 28.
Тел 238-26-86, 238-50-01.

Магазин «Дом книги на Соколе»

Адрес Ленинградский пр-т, д 78, к 1
Тел. 152-82-82, 152-45-11

Магазин

«Дом книги на Войковской»

Адрес Ленинградское шоссе, д 13,
стр 1
Тел 150-99-92, 150-69-17

Торговый дом книги «Москва»

Адрес ул Тверская, д 8, стр 1.
Тел 797-87-16, 229-73-55

Магазин «Дом книги на Новой»

Адрес Шоссе Энтузиастов, д. 24/43
Тел 361-68-34, 362-25-16

Магазин

«Дом книги в Медведково»

Адрес Заревый проезд, д 12
Тел. 478-48-97.

Магазин

«Дом книги в Сокольниках»

Адрес ул Рукаовская, д 27
Тел 264-81-21

Магазин «Мир школьника»

Адрес ул Шоссейная, 1
Тел 179-57-17

ПБОЮЛ Тимохина И.Ю.

Тел 533-44-44, 537-60-59

Магазин «Дом книги»

Адрес г Подольск, пр-т Ленина, д 158
Тел (276) 300-76

ООО «Элара»

Адрес г Сергиев-Посад
Тел (254) 212-52, 455-05

Сеть магазинов «Букберри»

Адрес. Калужское ш , ТРК «Мега»
Тел 789-65-02

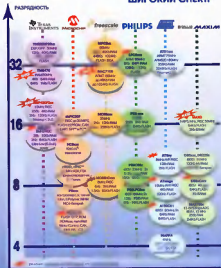
ТЦ «Глобал-Сити»

Адрес ул. Кировоградская, д 14
Тел.. 959-42-39

ТК «Галерея Аэропорт»

Адрес Ленинградский пр-т, д 62а
Тел 771-72-61

МИКРОКОНТРОЛЛЕРЫ ШИРОКИЙ СПЕКТР



www.1024m.computer.ru

Компэл
www.compel.ru