Создание приложений на базе процессоров Texas Instruments TMS320F28xx

Владимир ЧЕРНОВ
vladimir.chernov@ebv.com

Как известно, при создании приложения на базе нового процессора наибольшую сложность представляют первые ступени. Цель данной статьи — помочь читателю их преодолеть. В статье приводится описание процессоров семейства TMS320F28x, отладочных средств, примеры программирования. Немного внимания будет уделено ассемблеру, основная часть статьи посвящена программированию с использованием языка С, а также библиотек управления периферией, которые предлагает TI.

Процессоры семейства F28x (C28x) являются представителями платформы TMS320C2000. Данная платформа процессоров изначально создавалась для приложений управления электродвигателями и электроприводами. В процессе развития они приобрели более развитую периферию и мощное 32-разрядное ядро сигнального процессора, обладающее производительностью до 150 MIPS, за счет чего область их применения значительно расширилась. Используются они как в мощных устройствах управления промышленными автоматами, волоконно-оптическими сетями и т. д., так и, учитывая цену от ~$3, в приложениях, где традиционно применялись дешевые микроконтроллеры, что позволило значительно расширить возможности создаваемых приложений. Следует отметить, что данные процессоры имеют архитектуру с фиксированной запятой. В то же время, динамического диапазона 32-разрядной архитектуры с фиксированной запятой достаточно для большинства приложений, где ранее использовались процессоры с плавающей запятой. «Фиксированная запятая» имеет такие несомненные преимущества перед плавающей, как точность, малое время выполнения операций, меньшие стоимость и энергопотребление процессоров. Одной из целевых областей применения F28x являются импульсные источники питания. В частности, специализированный процессор UCD9501 (который также относится к этому семейству) создавался именно для таких приложений. Он имеет встроенный ШИМ контроллер с разрешением 150 пс, что в сочетании с высокоскоростным АЦП, большим набором коммуникационных портов, большим количеством портов ввода/вывода позволяет реализовать на его основе систему управления, к которой для получения законченного устройства необходимо добавить лишь силовую часть. Основные характеристики процессоров приведены в таблице 1.

F28x совместимы по коду с предыдущим семейством F24/240x, что позволяет использовать предыдущие наработки в новых проектах. В то же время, F28x имеет ядро, эффективно работающее под управлением алгоритмов, написанных на языках высокого уровня C/C++, что делает их пригодными не только для написания управляющей части, но и математической, предъявляющей высокие требования к производительности процессора. Архитектура F28x позволяет его использовать как встраиваемый контроллер, так и за счет его высоких вычислительных мощностей (32×32 MAC, 64-разрядные операции) — как центральный вычислительный модуль. Быстрый модуль обработки прерываний, с возможностью автоматического сохранения контекста, позволит обрабатывать большое количество внешних асинхронных событий с минимальным временем задержки. 8-уровневый конвейер позволяет достичь максимальной производительности. В процессорах предусмотрена логика анализа переходов, а также инструкции условного сохранения, оптимизирующие работу конвейера. F28x содержат атомарное АЛУ, выполняющее операции «чтение-модификация-запись» за один машинный цикл, что дает возможность исключить операции промежуточного копирования данных в регистровый файл (архитектура RISC-машины), а также избежать трудно отлавливаемых ошибок потери данных при возникновении прерываний.

Таблица 1. Характеристики процессоров TMS320F28x

Процессор	Тактовая частота, МГц	RAM, Кб	Flash, Кб	Таймеры	CAP/QEP	ШИМ, каналов	АЦП, каналов/время преобразования	Коммуникационные порты					Ввод/вывод	Внешняя память	Корпуса
								CAN	SPI	I²C	UART	MCBSP			
TMS320F2801	60, 100	12	32	3	2/1	8	16/267, 160	1	2	1	1	–	35	–	LQFP 100, BGA 100, Microstar BGA 100
TMS320F28015	60	12	32	3	2/0	8	16/267	–	1	1	1	–	35	–	LQFP 100
TMS320F28016	60	12	32	3	2/0	8	16/267	1	1	1	1	–	35	–	LQFP 100
TMS320F2802	60, 100	12	64	3	2/1	8	16/267, 160	1	2	1	1	–	35	–	LQFP 100, BGA 100, Microstar BGA 100
TMS320F2806	100	20	64	3	4/2	16	16/160	1	4	1	2	–	35	–	LQFP 100, BGA 100, Microstar BGA 100
TMS320F2808	100	36	128	3	4/2	16	16/160	2	4	1	2	–	35	–	LQFP 100, BGA 100, Microstar BGA 100
TMS320F2809	100	36	256	3	4/2	16	16/80	2	4	1	2	–	35	–	LQFP 100
TMS230F2810	150	36	128	3	6/2	16	16/80	1	1	–	2	1	56	–	LQFP 128
TMS230F2811	150	36	256	3	6/2	16	16/80	1	1	–	2	1	56	–	LQFP 128
TMS230R2811	150	40	–	3	6/2	16	16/80	1	1	–	2	1	56	–	LQFP 128
TMS230F2812	150	36	256	3	6/2	16	16/80	1	1	–	2	1	56	512K×16	LQFP 176, BGA 179, Microstar BGA 179
TMS230F2812	150	40	–	3	6/2	16	16/80	1	1	–	2	1	56	512K×16	LQFP 176, BGA 179, Microstar BGA 179
UCD9501	60, 100	12	32	3	4/2	16	16/160	2	4	1	2	–	35	–	LQFP 100

Отладочные средства

Как и любые другие процессоры, F28x начинаются со средств отладки. Именно они позволяют создавать сложные приложения. Texas Instruments предоставляет полный набор всех инструментов для отладки и программирования.

Данные средства можно разделить на программные и аппаратные. К первым относится Code Composer Studio, или сокращенно CCS. CCS — интегрированная среда разработки, включающая удобный редактор, компиляторы C/C++, ассемблер, линкер, отладчик с множеством средств визуализации, встраиваемую операционную систему реального времени. При помощи CCS можно подключать файлы ввода/вывода, что очень удобно при отладке приложения. Другими словами, он позволяет писать и отлаживать программы. Бесплатную 90-дневную версию CCS можно загрузить с WEB-сайта Texas Instruments — **w ww .ti.c om**.

К аппаратным средствам относятся в первую очередь JTAG-эмуляторы. Данные устройства позволяют загружать код программы и данные в процессор, программировать Flash-память, вести отладку приложения непосредственно в разрабатываемой системе. Эмуляторы подключаются к специально выделенным выводам процессора, через стандартный 14-контактный разъем с одной стороны и к компьютеру (порт принтера, USB, PCI, Ethernet) — с другой. Далее, используя CCS, можно осуществлять доступ практически ко всем ресурсам процессора, проводить как пошаговую отладку, так и отладку в реальном времени, без остановки процессора. Используя возможности RTDX, подключают исследуемую программно-аппаратную систему к приложениям на PC, например LabVIEW или MatLab. К аппаратным средствам также относятся Starter Kit — стартовые наборы разработчика. Данные наборы позволяют разработчику в короткий срок изучить процессор на практике, не создавая при этом собственной аппаратной платформы. В набор включается плата с процессором (TMS320F2808 или TMS320F(R)2812), CCS, источник питания и набор кабелей. На плате также интегрирован JTAG-эмулятор. Таким образом, разработчик получает готовую аппаратную платформу и полный набор средств для разработки и отладки приложений на ней. Из отличий двух наборов F2808, F2812 следует отметить, что хотя F2808 и имеет более скромный набор памяти, плата на нем содержит более развитый набор периферии: I^2C Flash, трансивер UART, трансивер CAN.

Для разработки собственной программно-аппаратной системы разработчику необходим набор средств разработки: Code Composer Studio + JTAG-эмулятор + собственная аппаратная платформа. Для учебных целей достаточно Starter Kit. С помощью него можно писать и отлаживать программы на реальной аппаратной платформе, при этом также возможно подключать данную платформу к различным внешним устройствам, в том числе устройствам измерения, осциллографам и т. п.

Программирование Flash-памяти процессора через JTAG-эмулятор осуществляется как внешней утилитой, так и встраиваемым в CCS (plug-in) приложением, которое можно загрузить с WEB-сайта **w ww .ti.c om**.

Для загрузки программы в память процессора можно воспользоваться встроенным загрузчиком (находится в масочном ПЗУ процессора). Загрузка осуществляется через любые коммуникационные порты. Далее программу можно переписать во Flash-память средствами процессора. Такой алгоритм позволит исключить необходимость использования дорогого JTAG-эмулятора. Однако для отладки приложений на встроенной системе в режиме реального времени эмулятор необходим.

Архитектура TMS320F28x

Сигнальные процессоры TI имеют достаточно мощную и сложную архитектуру. Рассмотрим ее лишь поверхностно, в степени, достаточной для начала освоения процессоров. В деталях будут описаны моменты, необходимые для экспериментов, приведенных в данной статье. Архитектуру F28x (рис. 1) разделим на вычислительную часть — центральный процессор, память и периферию. В данной части статьи рассмотрим центральный процессор и память.

Архитектура центрального процессора

Центральный процессор (ЦП) отвечает за выполнение инструкций программы и управление их потоком. ЦП выполняет арифметические и логические операции, сдвиг и умножение. Функциональная схема центрального процессора приведена на рис. 2.

ЦП включает в себя следующие основные блоки:

- Блок управления потоком команд. Организует последовательность выполнения команд программы.
- Блок генерации адреса команд.

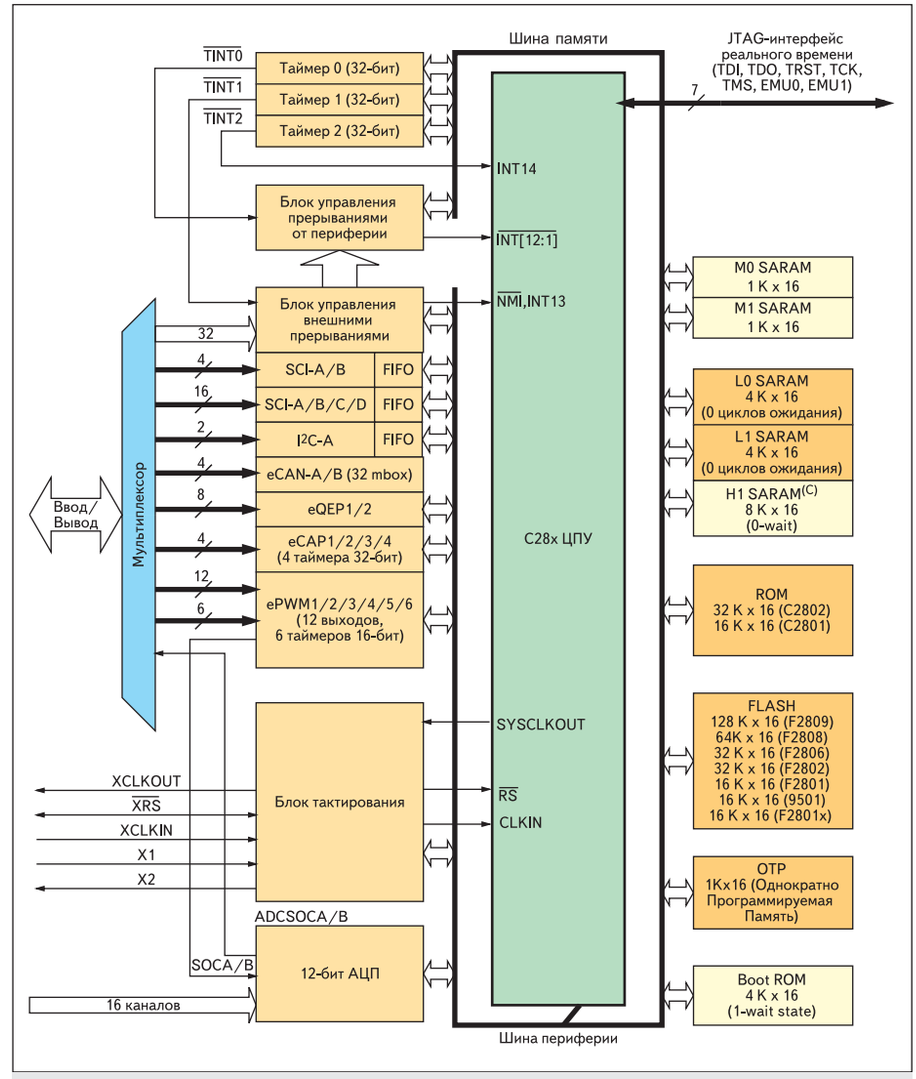

Рис. 1. Архитектура процессоров TMS320F28x

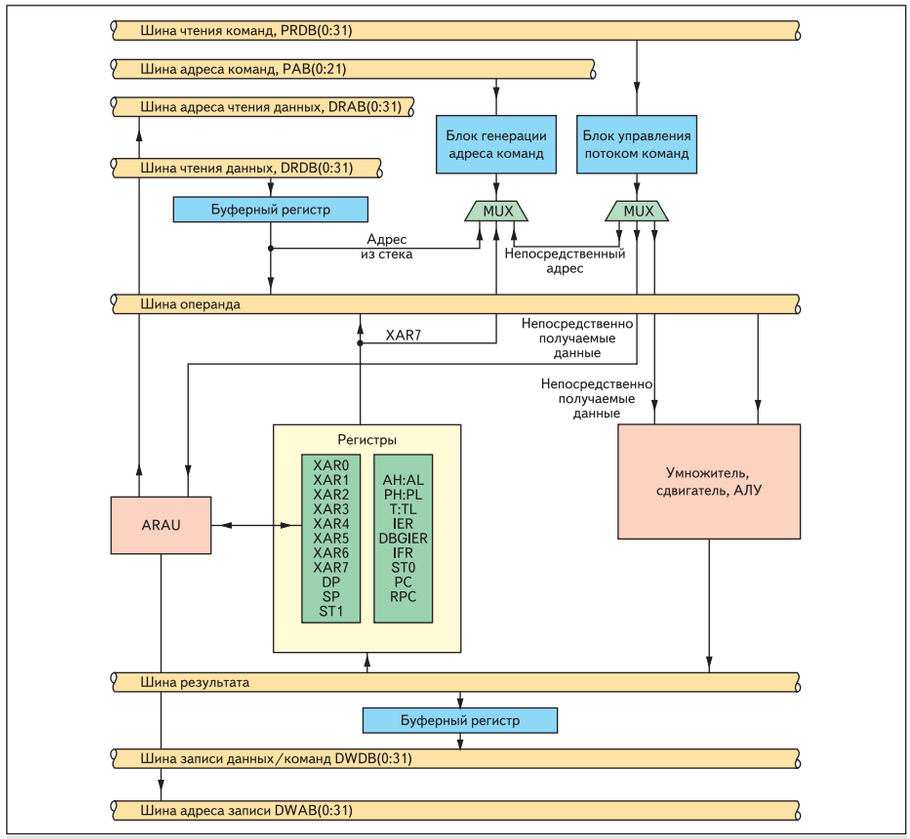

Рис. 2. Функциональная схема центрального процессора

Формирует адрес на шине команд.

- Арифметический модуль адресных регистров (ARAU).
  Модуль генерирует значения адресов для обмена с памятью данных. Полученные значения помещаются на шины адреса чтения и записи данных. Модуль также отвечает за увеличение/уменьшение значения в регистре — указателе стека (SP), а также вспомогательных регистров (XAR0, XAR1, XAR2, XAR3, XAR4, XAR5, XAR6, XAR7).
- Атомарное арифметико-логическое устройство (ALU).
  ALU выполняет 2-операндные арифметические и логические операции. Операнды в АЛУ поступают из регистров, памяти или из блока управления потоком инструкций. Результат может быть сохранен в регистре или памяти данных.
- Умножитель с фиксированной запятой.
  Умножитель выполняет перемножение двух операндов вида 32×32 с 64-битным результатом. Помимо самого умножителя для выполнения операции умножения задействованы также регистр множимого (XT), регистр произведения (P), аккумулятор (ACC). Результат может быть получен либо в регистре XT, либо в аккумуляторе. Умножитель взаимодействует непосредственно с ALU.
- Регистры центрального процессора.
  Используются для промежуточного хранения данных при вычислениях.

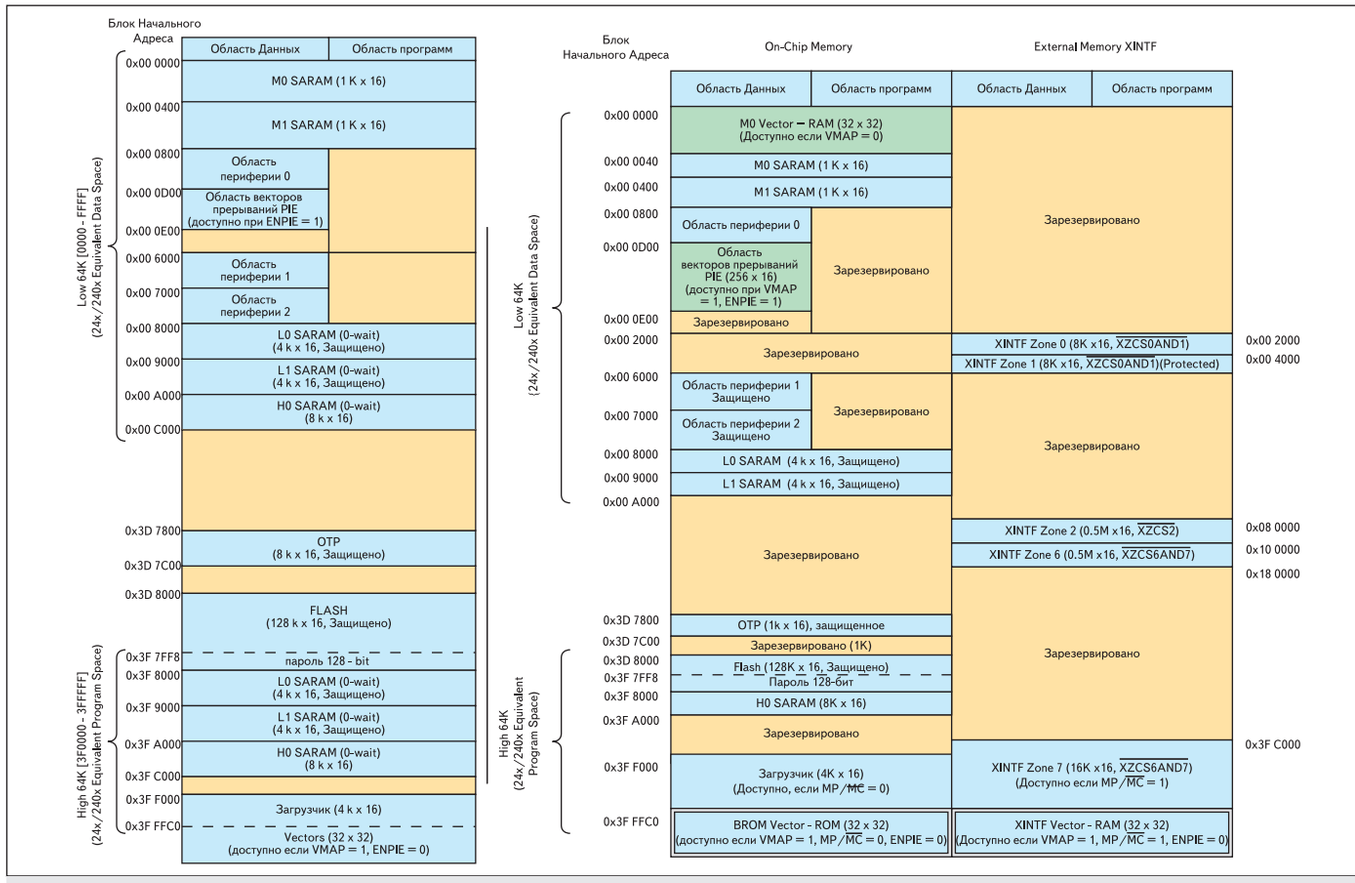

Рис. 3. Карта памяти процессоров TMS320F2808, TMS320F2812

Память

Область памяти C28x разделяется на две подобласти — программ и данных. Процессор обладает несколькими типами памяти, которые могут быть использованы как память программ, так и данных (назначаются программистом), они включают однопортовое ОЗУ (SARAM); расширенное ОЗУ; Flash-память; однократно программируемое ПЗУ; загрузочное ПЗУ, которое содержит загрузчик и стандартные таблицы чисел, используемые в математических алгоритмах.

Центральный процессор C28x может обращаться и к памяти на кристалле, и к внешней памяти. C28x использует 32 бит для адресации данных и 22 бит для команд, что теоретически позволяет использовать 4 Г слов (x16) данных и 4 М слов команд. Блоки памяти доступны для использования, как для данных, так и для команд. Карта памяти процессоров TMS320F2808, F2812 приведена на рис. 3.

Семейство процессоров F28x содержит модуль защиты памяти, позволяющий защитить паролем блоки L0, L1 SARAM, Flash, OTP. Каждый пароль состоит из 128 бит и хранится во Flash-памяти. Вся защищаемая область разделена на 16 блоков, каждый из которых защищен своим паролем. Надежность такого пароля можно оценить следующим образом: исходя из длины 128 бит, существует $2^{128} = 3{,}4\times10^{38}$ паролей. При попытке использовать один пароль каждые 8 тактов при частоте 100 МГц, нам понадобится 8×10^{23} лет, чтобы перебрать все возможные комбинации.

Совместимость с предыдущими семействами

В C28x реализован режим совместимости с процессорами C27x и C2xLP, что позволяет проще реализовать переход с предыдущих семейств. Рабочий режим процессора определяется состоянием бит OBJMODE и AMODE регистра статуса (ST1). Бит OBJMODE позволяет установить совместимость с кодом, скомпилированным для C28x (OBJMODE = 1) или C27x (OBJMODE = 0). AMODE позволяет установить совместимость с C28x/C27x-адресацией (AMODE = 0) или C2xLP (AMODE = 1).

C28x режим: в данном режиме программист может пользоваться всеми преимуществами ядра C28x — способами адресации и набором команд. Для работы в режиме C28x после сброса процессора его нужно переключить в этот режим. Для этого необходимо установить бит OBJMODE в «1», используя инструкции C28OBJ или SETC OBJMODE.

Режим совместимости по исходному коду с C2xLP: позволяет запускать перекомпилированные программы, написанные для C2xLP.

Режим совместимости с C27x: 100%-ная совместимость с C27x по объектному коду. После сброса процессор переходит в данный режим.

Программные средства проектирования — Code Composer Studio

В этой части статьи остановимся более подробно на создании программной части проекта, попытаемся по шагам разобрать, что для этого нужно сделать.

Основным средством для программирования и отладки, как уже отмечалось выше, является Code Composer Studio.

Чтобы стандартизовать разработку приложений для своих DSP, компания TI создала стандарт COFF (Common Object File Format). Данный стандарт обладает набором свойств, делающих его мощным средством создания программных приложений. Он очень эффективен при создании приложения командой программистов.

Как и в большинстве известных средств программирования в CCS, программа может состоять из множества файлов. Каждый файл программного кода, согласно COFF, называется модулем. Каждый модуль, включая объявление необходимых для его функционирования ресурсов, может быть написан независимо от других таких же модулей. Для создания модулей можно воспользоваться редактором — как CCS, так и любым другим, способным создавать ASCII-файлы. Стандартные расширения для файлов ассемблера .asm, C — .c.

После компиляции множество модулей соединяется в одну программу посредством линкера. Он оптимально разделяет ресурсы процессора между модулями. Линкер использует .cmd-файлы, в которых описано, где должна быть размещена каждая секция каждого модуля. Результатом работы является .out-файл, бинарный код, исполняемый процессором. Также может быть сгенерирован .map-файл, отчет о результатах компоновки исполняемого кода.

Code Composer Studio использует принцип «проекта». Иначе говоря, вы создаете проект для каждой программы, исполняемой DSP. В проекте хранится вся информация, необходимая для создания исполняемой программы, например, здесь перечисляются файлы исходного кода, файлы заголовков, командные файлы, свойства компиляции. На рис. 4 показано окно управления проектом. Информация «проекта» хранится в .pjt-файле, который создается и изменяется CCS автоматически.

Управлять проектом можно как через меню Project (главное меню), так и через выпадающее меню, которое можно вызвать, щелкнув правой кнопкой мыши на нужном проекте в окне управления проектом (рис. 4). Например, добавить файл можно, выбрав пункт Add Files из любого описанного меню. Добавить файл можно, и перетащив его из «проводника» Windows.

Параметры сборки проекта (Build Options)

Набор параметров проекта указывает средствам генерации исполняемого кода (компилятору, ассемблеру, линкеру) необходимые системные требования. При создании любого проекта CCS автоматически генерирует два набора параметров, называемых конфигурациями (*Configurations*): *Debug* и *Release*. Последнюю можно понимать как оптимизированную.

Для задания параметров в CCS был создан графический интерфейс, на рис. 4 показан пример окна Build Options. Окно содержит 4 закладки, наиболее важные из них две: *Compiler* и *Linker*. Под закладкой Compiler содержится 8 страниц параметров, задающих уровни оптимизации, конкретный процессор, под который компилируется программа, и т. д. Под закладкой Linker также содержится большое количество опций, но наиболее важные из них 3: -o <filename> — указывает

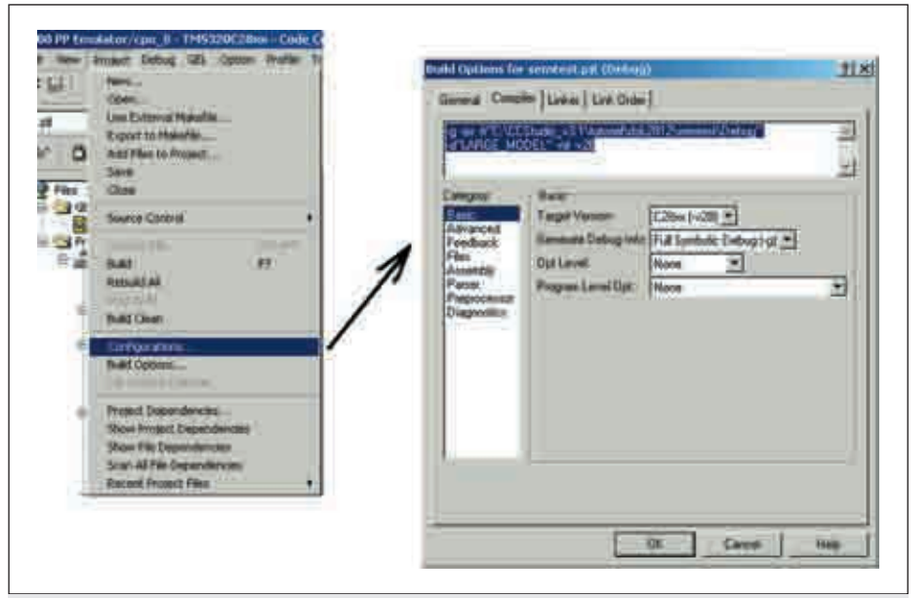

Рис. 4. Окно Build Options

имя исполняемого (загружаемого в процессор) файла; -m <filename> — указывает линкеру создать .map-файл, файл отчета о работе линкера; -c — указывает линкеру на необходимость автоинициализации глобальных и статических переменных.

Все опции могут быть заданы как выставлением необходимых галочек, так и командной строкой. Оба способа абсолютно идентичны, при проставлении галочек командная строка генерируется автоматически.

Командный файл линкера (Linker Command File)

Если взглянуть на код С-программы, можно увидеть, что она состоит из следующих частей: непосредственно исполняемого кода и различного типа данных (глобальные, локальные переменные). В терминологии Texas Instruments эти части называются секциями (sections). Все секции имеют свои названия (табл. 2). Разделение кода программ и данных позволяет достичь гибкости при распределении ресурсов памяти, а также отвечает Гарвардской архитектуре процессоров. Разработчик может разместить, например, часть кода в масочной памяти, часть кода во Flash, данные в ОЗУ, внешнем и внутреннем. На примере 1 приведена простейшая С-программа.

```
int a = 10;
int b = 8;

void main(void)
{
        long y;
        y= a +b;
}
```

Пример 1. С-программа

Рассмотрим расположение отдельных ее элементов. Каждый элемент будет автоматически размещен компилятором в соответствующей ему области памяти:

- *int a; int b;* — объявление переменных, выделение памяти в секции .ebss;
- *значения 10, 8* — будут размещены в секции .cinit;
- *long y;* — объявление локальной переменной, будет выделена память в секции .stack;
- *y = a + b;* — исполняемый код программы, разместится в секции .text.

Таким образом, программа будет разделена на следующие части, каждая из которых будет расположена в своей секции памяти:

- глобальные переменные;
- начальные значения глобальных переменных;
- локальные переменные;
- исполняемый код программы.

Необходимость разделения программы на секции обусловлена еще и тем, что процессоры TI содержат различные типы памяти: энергонезависимая (Flash), однократно программируемое ПЗУ, масочное ПЗУ (программируется TI при изготовлении микросхемы); энергозависимая (ОЗУ), которая также различается (младшие 64 К имеют большую гибкость в адресации, стек располагается только в них; существуют процессоры, часть оперативной памяти которых является двухпортовой). Программист может вручную расположить отдельные части кода в памяти, исходя из соображений производительности или принципов функционирования алгоритмов.

Таблица 2. Некоторые основные секции

Название	Описание	Область памяти
Инициализирующиеся секции		
.text	Код	Программ
.cinit	Глобальные инициализированные и статические переменные	Программ
.econst	Константы	Данных
.pinit	Таблица глобальных конструкторов (C++)	Программ
Неинициализирующиеся секции		
.ebss	Глобальные и статические переменные	Данных
.stack	Стек	Младшие 64 К данных
.esysmem	Память для функции malloc()	Данных

Линкер «собирает» секции из всех файлов и распределяет между ними память. Как уже отмечалось, объявление и расположение секций в памяти производится командами .cmd-файла. В последних версиях CCS проект может содержать несколько таких файлов. Расположение и длина секций определяются посредством команд MEMORY и SECTION.

Описание карты памяти

Область MEMORY описывает конфигурацию памяти создаваемой системы. Команда MEMORY имеет следующий синтаксис:

```
MEMORY
{
    <Имя>: origin = 0x<????>, length = 0x<????>
}
```

Например, мы имеем 64 к Flash-памяти, начинающейся с адреса 3E8000h, ее объявление будет следующим:

```
MEMORY
{
    FLASH: origin = 0x3E8000, length = 0x10000
}
```

Пример 2. Объявление области памяти, названной FLASH, длиной 10 000 h, начинающейся с адреса 3E8000h

Если нам необходимо добавить еще области памяти, используем ту же команду (пример 3).

Как уже говорилось, сигнальные процессоры TI имеют Гарвардскую архитектуру, т. е. в них разделены память программ и память данных. В .cmd-файле необходимо указывать, к какому типу памяти относится данная область. Для этого предусмотрена команда PAGE 0, определяющая память программ, и PAGE 1, определяющая память данных (пример 3).

```
MEMORY
{
    PAGE 0: /*Память программ*/
      FLASH: origin = 0x3E8000, length = 0x10000

    PAGE 1: /*Память данных*/
      M0SARAM: origin = 0x0000, length = 0x0400
      M1SARAM: origin = 0x0400, length = 0x0400
}
```

Пример 3. Множественное объявление областей памяти

Объявление секций

Для объявления соответствия секций областям памяти используется команда SECTIONS (пример 4).

```
MEMORY
{
    PAGE 0: /*Память программ*/
      FLASH: origin = 0x3E8000, length = 0x10000

    PAGE 1: /*Память данных*/
      M0SARAM: origin = 0x0000, length = 0x0400
      M1SARAM: origin = 0x0400, length = 0x0400
}

SECTIONS
{
    .text :>    FLASH      PAGE 0
    .ebss :>    M0SARAM    PAGE 1
    .cinit :>   FLASH      PAGE 0
    .stack :>   M1SARAM    PAGE 1
}
```

Пример 4. Листинг .cmd-файла простейшей программы

Проект 1

В данной главе создадим первый простейший проект, который будет служить основой для более сложных последующих примеров.

Описание проекта

Используемый процессор: TMS320F2808, вся оперативная память должна быть распределена между секциями.

Расположение секций:

- .text — H0SARAM
- .cinit — H0SARAM
- .ebss — M0SARAM
- .stack — M1SARAM

В экспериментальных целях вся наша программа располагается в оперативной памяти, использованию Flash-памяти будет уделена отдельная часть статьи.

Примечание: *данный проект может быть реализован на любом процессоре F28x, разница будет лишь в значениях, указанных в командном файле.*

Создание проекта

1. Для проведения экспериментов с проектом вам понадобится «целевой» процессор. В качестве такового может служить Starter Kit, ваша собственная плата + JTAG-эмулятор или же симулятор (симулятор пригоден только для первых проектов). Подключите оборудование. Инструкция по подключению прилагается либо к Starter Kit в случае его использования, либо к JTAG-эмулятору, если вы используете собственную плату. При использовании

Starter Kit для отладки следующий шаг — переход к п. 3.

2. Если вы впервые запускаете Code Composer Studio, вам необходимо выполнить некоторые предустановки с помощью программы cc_setup.exe (задать тип эмулятора и процессора), иначе переходите к п. 3.
   2.1. Запустите программу cc_setup.exe. В списке *Available Factory Boards* выберите искомые процессор и эмулятор. Процессор может быть виртуальным, для этого нужно выбрать конфигурацию со значением *simulator* в поле *Platform*.
   2.2. Нажмите кнопку Add внизу окна. Заданная конфигурация появится в области *System Configuration*.
   2.3. В некоторых случаях может понадобиться редактирование конфигурации, например, в случае использования эмулятора, подключаемого к порту принтера, адрес которого отличается от 378h. Для этого необходимо щелкнуть правой кнопкой мыши на выбранной конфигурации, в поле *System Configuration*, в появившемся меню выбрать пункт *Properties*. Далее в окне *Connection Properties* выбираем закладку *Connection Properties*. В поле *Value* задаем необходимые параметры.
   2.4. Нажать кнопку *Save&Quit*. Если вы не собираетесь останавливаться на достигнутом, в появившемся окне (*Start Code Composer Studio On Exit*) необходимо выбрать *Yes*.
3. Запускаем Code Composer Studio — cc_app.exe (если еще не запущен после п. 2). CCS имеет возможность программно отключать/подключать эмулятор (начиная с версии 3), что позволяет в случае необходимости менять или просто на время отключать отлаживаемую систему. По умолчанию, после запуска система отключена. Для ее подключения необходимо выбрать пункт *Debug* в главном меню, в появившемся подменю выбрать *Connect*. Состояние подключено/отключено отображается в левом нижнем углу окна CCS.
4. Создайте на рабочем диске папку, где будут находиться ваши проекты, предположим, это будет *C:\F28\projects*. В меню *Project* необходимо выбрать пункт *New*. В появившемся окне, в поле Project Name, вводим название проекта — *lab1*. В поле Location необходимо ввести *C:\F28\projects\lab1*. После нажатия кнопки *Finish* будет создан .pjt-файл, в котором будет храниться вся информация о проекте.
5. Выбираем пункт меню: *File → New → Source File*. Далее сохраняем созданный файл с именем *lab1.c* в папку проекта (C:\F28\projects\lab1): *File → Save*.
6. Добавляем созданный файл в проект: Project → Add files to Project. Необходимо выбрать файл lab1.c.
7. Повторяем пп. 4, 5, файл — lab1.cmd.
8. Ввод программы.
   В поле проекта (находится в левой части окна CCS, если поле отсутствует, необходимо поставить галочку, выбрав пункт меню View→Project) открываем *Project→lab1.pjt→ Source→lab1.c*. Должен открыться файл lab1.c, в котором будет содержаться исполняемый код нашей программы.
   Вводим программу из примера 1 и сохраняем ее.
9. Ввод командного файла.
   Вызываем файл lab1.cmd.
   Создаем командный файл согласно приведенному выше описанию проекта.
   В конец описания SECTONS необходимо добавить следующую строку:
   .reset: > H0SARAM PAGE 0, TYPE = DSECT
   Данная строка указывает на поле reset, используемое библиотекой *rts2800_ml.lib* (п. 9). Выражение TYPE = DSECT указывает линкеру не заполнять секцию. Секция .reset — часть библиотеки rts2800_ml.lib.
   Сохраняем результат.
   Примечание: *создать командный файл предлагается читателю самостоятельно, в случае затруднений в конце статьи (пример 5, 6) приведены листинги этого файла для процессоров TMS320F2812 и TMS320F2808.*
10. Подключение к проекту библиотеки *rts2800_ml.lib*.
    Данная библиотека необходима для проведения начальных установок в C/C++ программах и выполняет следующие действия:
    а) установку конфигурационных регистров;
    б) установку стека и вторичного системного стека;
    в) инициализацию глобальных переменных (из области .cinit);
    г) вызов конструкторов глобальных объектов (.pinit);
    д) вызов функции main;
    е) вызов функции, вызывающей деструкторы глобальных объектов, после завершения main.
    Библиотека добавляется в проект аналогично п. 5 и находится в папке с Code Composer Studio (c:\CCStudio_v3.1\c2000\cgtools\lib\rts2800_lib_ml.lib), там же находится листинг этой библиотеки.

Опции сборки проекта (Project Build Options)

Для данного проекта достаточно стандартных предустановок, необходимо изменить лишь размер выделяемого стека и указать линкеру на необходимость создавать .map-файл.

11. Необходимо выбрать пункт меню Project → Build Options.… Выбрать закладку *Linker*. Далее, в поле *Stack Size(-stack)* вводим значение 200.
12. Проверяем правильность данных в полях *Output Filename (-o)* и *Map Filename(-m)*.
13. В поле *Autoinit Model* выбираем *Run-Time Autoinitialization (-c)*. Данный пункт указывает компилятору на необходимость автоматической инициализации переменных из .cinit.
14. Сохраняем проект со всеми настройками: *Project → Save*

Компиляция, сборка и загрузка проекта

15. В целях обеспечения более удобной работы при отладке приложения, его многократной компиляции и загрузки, рекомендуется настроить среду следующим образом:
    а) чтобы подключение CCS к эмулятору происходило автоматически каждый раз при запуске первого. Для чего необходимо выйти в меню *Options → Customize…*, закладка *Debug Properties*, поставить галочку напротив *Connect to the target…*
    б) загрузка откомпилированного кода также должна происходить автоматически — в случае отсутствия ошибок. Действия: в том же самом меню (Options → Customize…) закладка *Program/Project Load*, галочка напротив *Load Program After Build*.
16. Сборка проекта.
    Меню *Project → Build* или кнопка Build на панели инструментов.
    Если все было введено без ошибок, откомпилированная программа загрузится в процессор (или симулятор). Счетчик команд автоматически будет указывать на _c_int00, что можно увидеть в окне *Disassembly Window*.

Отладка

17. Запускаем процессор на выполнение функций инициализации библиотеки rts2800_ml.lib, до начала функции main.
    Debug → Go Main.
18. Просмотр содержимого памяти.
    View → Memory.
    - В поле Address необходимо набрать *&b*. Символ «&» указывает на то, что берется адрес переменной b, а не ее значение.
    - Для удобства просмотра в поле *Format* выберите *Hex — TI Style*.
    - Значение в поле Page указывает на тип памяти (данных или программ), в данном случае необходимо выбрать значение *Data*.
    - Нажмите OK.

    После проделанных указанных выше действий появится окно, отображающие содержимое памяти. В нем можно изменять значения отдельных ячеек, выбрав их двойным щелчком мыши.
19. Окно просмотра (Watch Window).
    View → Watch Window.
    - В появившемся окне нажмите на закладку *Watch Locals*. Под данной закладкой отображаются все локальные переменные текущей функции.

- Для добавления данных в окно перейдите к закладке Watch1: сюда можно вводить названия любых необходимых переменных, создавая удобные наборы для просмотра. Переменную можно добавить также сочетанием клавиш Ctrl+w, предварительно выделив ее в тексте программы.

20. Выполнение программы.

Теперь можно переходить к основным действиям отладки — выполнению программы. Здесь CCS представляет стандартный набор команд:

- *Debug → Run (F5)* — пуск программы до точки останова;
- *Debug → Run Free (Alt+F5)* — пуск программы, минуя точки останова;
- *Debug → Step Over (F10)* — пошаговое исполнение программы, все функции выполняются за один шаг;
- *Debug → Step Into (F11)* — пошаговое исполнение, в случае вызова функции отладчик пошагово выполняет вызываемую функцию;
- *Debug → Step Out (Shift+F11)* — отладчик выполняет текущую функцию до ее завершения, т. е. передает управление функции, вызвавшей данную функцию.

Существует также возможность пошагового исполнения программ по дизассемблированному коду. Команды выполнения аналогичны вышеописанным, находятся они в меню Debug→Assembly/Source Stepping→…

Большинство из описанных здесь команд имеют кнопки на панели инструментов главного окна CCS.

Таким образом, нажимая клавишу F11, можно проследить правильность выполнения программы. Состояние переменных будет изменяться, что видно из окон просмотра памяти и Watch Window.

Итак, был сделан первый шаг. Выполнив действия, приведенные выше, вам удалось (автор на это надеется) запустить на TMS320F28x простейшую программу, а также провести ее отладку — заставить сдвинуться мощнейший гоночный автомобиль с места. В продолжение статьи мы заставим его двигаться с максимальной скоростью и грациозно обходить всевозможные препятствия. В следующей статье будет уделено внимание программированию на ассемблере, а также более детальному рассмотрению С-программ. Цель продолжения статьи — библиотеки описаний периферии, позволяющие не затрачивать больших усилий на программировании последней, а сконцентрироваться на основных алгоритмах, управляющих функционированием системы. Отдельная часть статьи будет посвящена DSP/BIOS — встраиваемой операционной системе реального времени.

В заключение, приведем полные .cmd-файлы, используемые в проекте 1 для процессоров TMS320F2812 и TMS320F2808.

```
MEMORY
{
      PAGE 0:          /* Память программ */
        H0SARAM:    org = 0x3F8000,      len = 0x2000

      PAGE 1:          /* Память данных*/
        M0SARAM:    org = 0x000000,      len = 0x0400
        M1SARAM:    org = 0x000400,      len = 0x0400
        L0SARAM:    org = 0x008000,      len = 0x1000
        L1SARAM:    org = 0x009000,      len = 0x1000
}

SECTIONS
{
      .text:    > H0SARAM     PAGE 0
      .ebss:    > M1SARAM     PAGE 1
      .cinit:   > H0SARAM     PAGE 0
      .stack:   > M0SARAM     PAGE 1
      .reset:   > H0SARAM     PAGE 0, TYPE = DSECT /* not using
.reset section */
}
```

Пример 5. Листинг .cmd- файла для процессора TMS320F2812, используемый в Проекте 1

```
MEMORY
{
      PAGE 0:          /* Память программ*/
        H0SARAM:    org = 0x00A000,      en = 0x2000

      PAGE 1:          /* Память данных*/
        M0SARAM:    org = 0x000000,      len = 0x0400
        M1SARAM:    org = 0x000400,      len = 0x0400
        L0SARAM:    org = 0x008000,      len = 0x1000
        L1SARAM:    org = 0x009000,      len = 0x1000
}

SECTIONS
{
      .text:    > H0SARAM     PAGE 0
      .ebss:    > M1SARAM     PAGE 1
      .cinit:   > H0SARAM     PAGE 0
      .stack:   > M0SARAM     PAGE 1
      .reset:   > H0SARAM     PAGE 0, TYPE = DSECT /* not using
.reset section */
}
```

Пример 6. Листинг .cmd-файла для процессора TMS320F2812, используемый в Проекте 1