Средства разработки программного обеспечения для встраиваемых 32-разрядных систем

(часть 1)

Любовь Самойлова (Москва)

В статье рассматриваются современные средства разработки программного обеспечения для встраиваемых систем. Приведён сравнительный анализ компиляторных пакетов для микроконтроллеров ARM.

Популярность процессорного ядра ARM среди 32-разрядных микроконтроллеров (МК) для встраиваемых систем и фактическое превращение архитектуры ARM в промышленный стандарт привели к тому, что аппаратные и программные средства разработки для этих МК предлагают практически все фирмы, выпускающие подобные продукты. Доступность и качество средств разработки и отладки являются важным фактором принятия решения при переходе на новую платформу, поэтому рассмотрим их более подробно.

Для процессоров ARM в настоящий момент существует множество программных средств разработки (компиляторов, отладчиков, интегрированных сред программирования), от свободно распространяемого программного обеспечения (проект GNU) до интегрированных пакетов, предлагаемых крупнейшими мировыми компаниями – производителями средств разработки для встроенных систем, такими как IAR Systems или ARM.

При обсуждении средств разработки ПО для встраиваемых приложений мы будем ориентироваться на задачи, выполняемые в реальном времени без применения операционных систем, что соответствует типичному использованию 8-разрядных микроконтроллеров.

Таблица 1. Размеры и выравнивание основных типов данных языка Си для ядра ARM

Тип	Размер (выравнивание, байт)
char	8-разрядный байт (1)
short	16-разрядное полуслово (2)
int	32-разрядное слово (4)
long	32- разрядное слово (4)
указатель	32- разрядное слово (4)

Особенности программирования на Си для микроконтроллеров ARM

Фактически язык Си является стандартным языком программирования встроенных систем, и компилятор Си является составной частью любой интегрированной среды разработки. Поэтому мы рассмотрим некоторые особенности реализации языка Си для целевой платформы ARM и сравним несколько компиляторных пакетов, входящих в состав популярных средств разработки для встроенных систем, в том числе:

- IAR Embedded Workbench (IAR Systems);
- Real View MDK (ARM-Keil);
- CM-ARM (российская фирма «Фитон»);
- GNU – GCC, входит в состав среды разработки в качестве основного (например, RIDE компании Raisonance, CrossWorks компании Rowley Associates и др.) или дополнительного (MDK, CM-ARM) компиляторного пакета;
- ICCV7 (Image Craft).

Практически все компиляторы для целевой платформы ARM удовлетворяют стандарту ATPCS (ARM Thumb Procedure Call Standard), разработанному компанией – разработчиком ядра. В стандарте описаны требования к представлению основных типов данных и использованию стека и конвенция вызова функций. Размеры и выравнивание целочисленных типов данных языка Си для ARM приведены в таблице 1 в качестве примера.

Ядро ARM использует классическую архитектуру load/store, т.е. в качестве операндов могут использоваться только значения, загруженные в 32-разрядные регистры процессора. Поэтому хотя в системе команд представлены операции чтения из памяти и записи в память со всеми целочисленными типами данных, в том числе со знаковым и беззнаковым расширением, при операциях с 8- и 16-разрядными локальными переменными (**char** или **short**), необходимость преобразования типов может приводить к заметному увеличению размера кода. Например, для функции:

```
short f(short a)
{
if (a > 1)
/* некоторый код */
}
```

будет сгенерирован следующий код:

```
; помещаем знаковый бит
; в старший разряд
MOV     R1,R0,LSL #16
; знаковое расширение
MOV     R1,R1,ASR #16
CMP     R1,#1
```

А для функции

```
int f(int a)
{
if (a > 1)
/* некоторый код */
}
```

соответствующий код будет состоять из одной инструкции

```
CMP R1,#1
```

Поэтому в отличие от 8-разрядных архитектур, рекомендуется по возможности использовать тип **int**

для параметров и локальных переменных, которые размещаются в регистрах.

Конвенция вызовов

Распределение регистров в соответствии со стандартом ATPCS приведено в таблице 2. Как следует из таблицы, одновременно функция может содержать не более 14 локальных регистровых переменных, включая параметры (R0 – R11, R12 и R14). Остальные локальные переменные размещаются в памяти (стеке). Доступ к размещённым в памяти данным является более затратной операцией, и по возможности его следует избегать.

Принимая во внимание особенности конвенции вызовов и распределения регистров, можно сформулировать следующие простые правила оптимизации размера кода:

- если количество скалярных параметров функции не превышает 4, они будут передаваться через регистры;
- если необходимо передать в функцию более четырёх параметров, следует запаковать их в структуру и передавать указатель на неё;
- не следует использовать более десяти локальных переменных одновременно (при условии, что у функции четыре параметра).
- следует избегать функций с переменным числом параметров.

Влияние выравнивания на размещение структур в памяти

Из-за выравнивания занимаемый структурой размер зависит от порядка, в котором члены структуры разных типов входят в состав структуры, например:

```
struct {char c; short s; char d;
int i;}
```

будет занимать в памяти 12 байт:

```
|c|_|s|s|d|_|_|_|i|i|i|i|
|1|2|3|4|5|6|7|8|9|A|B|C|
```

Та же самая структура, объявленная как

```
struct {char c; char d; short s;
int i;}
```

будет занимать 8 байт, поскольку при таком размещении не требуется дополнительных байтов для выравнивания внутри структуры.

Битовые поля

Если для работы с регистрами специального назначения МК используются структуры с битовыми полями, что позволяет сделать код для операций с отдельными разрядами периферийных устройств хорошо читаемым и легко поддерживаемым, следует обратить внимание на реализацию работы с битовыми полями в конкретном компиляторе.

Так, компилятор GCC имеет такую особенность: если битовое поле не выходит за пределы одного байта, для операций с этим битовым полем всегда используются «оптимальные» байтовые инструкции чтения и записи, что может приводить к Data Abort при обращении к регистрам периферии размером в 4-байтное слово.

По стандарту языка Си битовые поля имеют тип **int**. Такие компиляторы, как IAR, RVCT, CM-ARM, поддерживают расширение языка Си, позволяющее описывать битовые поля любым базовым целочисленным типом. При этом выравнивание ячейки памяти, в котором поле находится, и обращение к битовому полю определяется типом. Если битовое поле описано как **short**, то и для обращения к нему компилятор всегда будет использовать соответствующие инструкции чтения и записи полуслова; если оно описано как **int** – компилятор будет использовать только инструкции чтения и записи слова. Более подробно этот механизм описан в документации на компиляторы.

Представление памяти микроконтроллера

Существенным моментом, связанным даже не с компилятором, а с компоновщиком и средой разработки, является представление карты памяти микроконтроллера (ROM/Flash, RAM, отображаемые в память регистры периферийных устройств, векторы прерываний) программными средствами и механизм управления размещением кода и данных в физической памяти.

Как правило, распределение памяти для МК одного производителя сохраняется от одного семейства МК к другому: размеры физической памяти изменяются, но стартовые адреса для памяти одного и того же типа остаются неизменными. Микроконтроллеры разных фирм с архитектурой ARM, даже с одинаковыми ядрами ARM, значительно отличаются друг от друга. Поэтому на практике могут оказаться существенными следующие особенности ПО: поддерживается ли МК конкретного производителя средой разработки, насколько наглядно и просто можно модифицировать распределение памяти, насколько гибко компилятор и компоновщик могут управлять размещением сегментов с кодом и данными.

Например, для компоновщика GNU (ld) конфигурация и размещение сег-

Таблица 2. Использование регистров в соответствии со стандартом ATPCS

Регистр	Использование
R15	PC, счётчик команд
R14	LR, содержит адрес возврата при передаче управления инструкцией BL
R13	SP, указатель стека программ
R12	
R11	
R10	
R9	
R8	
R7	Для размещения локальных переменных; вызываемая функция сохраняет используемые регистры в стеке
R6	
R5	
R4	
R3	Для передачи параметров, возврата значения и в качестве временных (scratch) регистров; вызываемая функция не сохраняет значения этих регистров в стеке
R2	
R1	
R0	

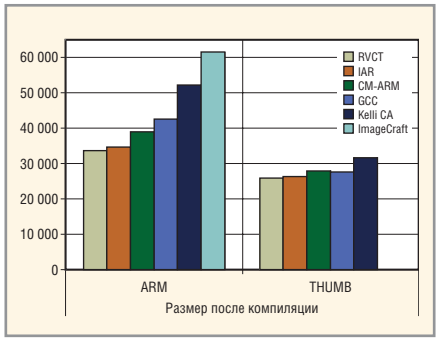

Рис. 1. Суммарный размер кода, полученный после компиляции тестовых Си-файлов без библиотек

Слева результаты компиляции в систему команд ARM, справа – в систему команд Thumb. Для Image Craft в демо- и стандартной версии генерация Thumb-кода не поддерживается, поэтому результат не показан.

ментов (секций) в памяти задаётся с помощью довольно сложного скриптового языка и не обладает достаточной гибкостью.

Маленькие «хитрости»

Следует иметь в виду, что в системе команд ARM/Thumb есть инструкции знакового и беззнакового умножения, но отсутствуют инструкции деления (оно реализуется программно). Поэтому рекомендуется вместо операций деления использовать операции сдвига, где это возможно.

При разработке систем реального времени следует принимать во внимание, что инструкции ARM являются «атомарными», т.е. исполнение инструкции не может быть прервано обработчиком прерывания. Если в одной инструкции осуществляется запись в большое число регистров, использование инструкций чтения и записи группы регистров может существенно увеличивать время реакции на возникновение прерывания. Эти инструкции обычно генерируются компилятором в прологе и эпилоге функций с целью сохранения и восстановления регистров, используемых для размещения локальных переменных функции (за исключением регистров R0 – R3 и R12, которые являются scratch-регистрами). Во избежание таких задержек следует сокращать количество локальных переменных в функции (тогда почти все они могут быть размещены в scratch-регистрах). Кроме того, некоторые компиляторы имеют специальную опцию, позволяющую ограничить количество сохраняемых/восстанавливаемых регистров.

Какой компилятор выбрать

Прежде чем перейти к сравнению компиляторов, дадим краткую характеристику функциональности самих программных сред разработки, в состав которых они интегрированы.

IAR EWARM 4.41A компании IAR Systems. Продуманная оболочка с хорошими возможностями разработки и отладки, имеет единый интерфейс для различных семейств микропроцессоров. Среда работает только с собственным компилятором IAR, подробно и качественно документирована, содержит много примеров.

Keil µVision3 компании Keil. Входит в состав *Keil-ARM Real View MDK*. Простая в освоении оболочка, хорошо известная разработчикам систем на основе целевой архитектуры MCS-51. В состав пакета входит свежая версия компилятора от компании ARM RVCT 3.0 (Real View Compilation Tools), но среда также предусматривает интеграцию с собственным компилятором от Keil, с компилятором GNU для ARM (GCC) и с более ранней версией компиляторного пакета от ARM, ADS (ARM Development Suite). Имеются некоторые трудности при установке на один компьютер нескольких пакетов µVision для разных семейств микроконтроллеров.

CM-ARM. Среда разработки компании «Фитон». Работает с собственным компилятором CMC-ARM и с компилятором GCC; предусмотрена возможность интеграции с компилятором IAR. Простая в освоении, хорошо оснащённая оболочка, имеющая русский интерфейс. Визуализированная настройка распределения памяти: можно задать неограниченное число областей памяти ROM/RAM/Flash и управлять размещением сегментов. Кроме того, CM-ARM может использоваться самостоятельно в качестве отладчика.

ImageCraft. Среда работает с собственным компилятором и, в отличие от остальных, не содержит встроенного симулятора-отладчика. Кроме того, демоверсия компилятора плохо документирована и не поддерживает генерацию кода в режиме Thumb.

Результаты тестов

Для тестирования был взят набор файлов, ранее использованный компанией Raisonance для сравнения качества кода, генерируемого компилятором GNU и компиляторами IAR, Keil и ARM. Исходные тексты файлов были загружены с интернет-страницы компании и адаптированы для исполнения на микроконтроллере LPC2106 фирмы Philips. Основной целью было проанализировать, насколько конкурентоспособны появившиеся на рынке недорогие решения (включая текущую версию компилятора GCC) по сравнению с известными пакетами компаний IAR и Keil-ARM.

Набор для тестирования включал:

- криптографические алгоритмы, которые по своей природе отличаются избыточностью и повторяемостью вычислений, что позволяет наглядно оценить качество генерируемого компиляторами кода;
- решение задачи «Ханойской башни»;
- два «стандартных» теста, DHRYSTONE и WHETSTONE, многие годы используемых для тестирования компиляторов и процессоров. В

Таблица 3. Набор использованных тестов

des.c	Криптографический алгоритм
dhry.c	Известный тест DHRYSTONE, включающий все основные конструкции языка
lucifer.c	Криптографический алгоритм
mars.c	Криптографический алгоритм
playfair.c	Криптографический алгоритм
rijndael.c	Криптографический алгоритм
sha.c	Криптографический алгоритм, применяемый в смарт-картах
towers.c	Короткое решение задачи «Ханойская башня»
whets.c	Известный тест WHETSTONE , включающий операции с плавающей точкой

тесте DHRYSTONE используются все основные конструкции языка, а в WHETSTONE тестируются операции с плавающей точкой.

Полностью тесты перечислены в таблице 3.

При компиляции использовались ключи, обеспечивающие максимальную оптимизацию кода по размеру. На рисунке 1 показан суммарный размер кода, полученный в результате компиляции только самих исходных файлов. Суммарный размер исполняемых файлов, вместе с библиотечными функциями и startup-модулем, показан на рисунке 2.

Анализ показывает, что для большинства компиляторных пакетов размер исполняемых файлов существенно зависит от того, какие функции ввода-вывода используются. Поэтому на рисунке 3 представлен суммарный размер исполняемых файлов для минимизированного варианта функции printf (не поддерживающей некоторые спецификаторы формата, например, с плавающей точкой), если он содержится в поставляемой с компилятором библиотеке. Для компилятора Keil суммарный размер исполняемых файлов не приводятся: в состав демоверсии пакета не входят библиотеки операций с плавающей точкой для типа double, что делает невозможным корректное сравнение результатов.

Насколько оптимален по размеру код, генерируемый компиляторами, видно на рис. 1. Безусловным лидером является компилятор RVCT, что неудивительно, поскольку разработчиками являются создатели архитектуры ARM, много лет совершенствующие свой компиляторный пакет. С другой стороны, заметно, что компилятор Keil хуже, чем можно было ожидать: размер кода на 55% больше для режима ARM и на 25% – для режима Thumb.

Новые компиляторы повели себя по-разному. Компилятор CM-ARM показал размер кода всего на 8% хуже, чем RVCT в режиме Thumb, и на 16%, чем RVCT в режиме ARM, а компилятор компании Image Craft сильно проиграл другим продуктам (82% в режиме ARM; демоверсия не поддерживает генерацию кода в режиме Thumb). Код, сгенерированный компилятором GCC для режима ARM, всего на 26% больше по размеру, чем код, выданный RVCT, и на 7% больше для режима Thumb.

Что касается размера исполняемых файлов, собранных вместе с библиотечными функциями, то порядок несколько меняется (см. рис. 2). Лидером по-прежнему остаётся RVCT , но GCC существенно проигрывает всем остальным. Причина этого известна: распространяемая в составе дистрибутива GNU библиотека newlib является «универсальной» и слабо адаптирована к разработке встроенных систем; в частности, она не имеет редуцированных функций ввода-вывода.

Приведённые на рис. 3 размеры исполняемых файлов, собранных с редуцированными библиотеками ввода-вывода (если они существуют в составе соответствующего дистрибутива), снова изменяют «тройку лидеров». На первом месте оказывается IAR, а второе и третье места занимают RVCT и CM-ARM.

Безусловно, по компактности генерируемого кода и универсальности нет равных компиляторным пакетам компаний IAR и ARM, но и сами пакеты, и программные средства разработки, в состав которых они входят, стоят достаточно дорого.

У пакета CM-ARM, кроме сравнимого по качеству компилятора и хорошего отладчика, есть ещё одно несомненное достоинство – наличие документации и поддержки на русском языке.

Компилятор компании Image Craft генерирует не слишком компактный код и имеет существенные ограничения. Так, генерация кода в режиме Thumb доступна – по утверждению продавца – только в дорогой Advanced-версии пакета; отсутствует собственный отладчик и возможность запускать компилятор из командной строки (по крайней мере, в демоверсии). По-видимому, этот компилятор находится ещё на начальной стадии разработки и не может составить конкуренции остальным пакетам.

И, наконец, если вы предпочитаете сэкономить на компиляторе и прочих программных средствах разработки, – используйте проект GNU: в Сети можно найти достаточно компактные библиотеки для встроенных систем на основе микроконтроллеров ARM различных производителей. Однако использование свободно распространяемых средств – это всегда лотерея. Они, как правило, слабо документированы и имеют «подводные камни», некоторые из которых были упомянуты выше.

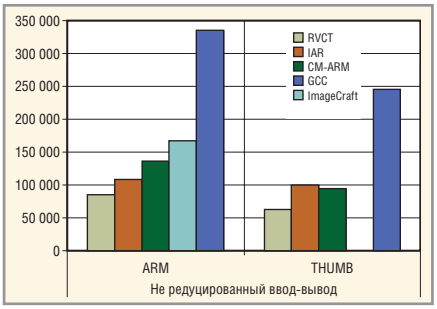

Рис. 2. Суммарный размер кода исполняемых файлов вместе с библиотеками
Слева результаты компиляции в систему команд ARM; справа – в систему команд Thumb (для Image Craft результат не показан)

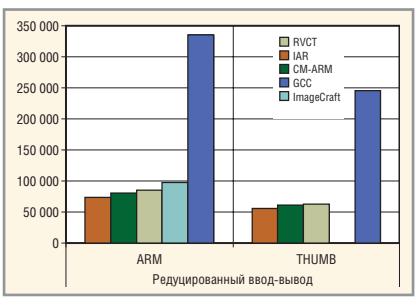

Рис. 3. Суммарный размер кода исполняемых файлов вместе с библиотеками при сокращённом варианте функции printf для пакетов IAR, CM-ARM и Image Craft
Слева – результаты компиляции в систему команд ARM; справа – в систему команд Thumb (для Image Craft результат не показан)

Для всех коммерческих компиляторных пакетов, которые мы рассмотрели, существуют демоверсии. Дистрибутив пакета GCC имеется как в исходных кодах, так и в бинарном виде. Практически все демоверсии снабжены многочисленными примерами для микроконтроллеров ARM различных производителей, что позволяет самостоятельно оценить удобство работы с конкретным пакетом и выбрать тот, который подходит именно вам. Для принятия окончательного решения следует провести анализ средств отладки, которым будет посвящена вторая часть статьи.

Продолжение следует

Литература

1. Procedure Call Standard for the ARM Architecture, www.arm.com/miscPDFs/8031.pdf.
2. IAR Systems, www.iar.se.
3. RealView MDK, www.keil.com/arm.
4. www.phyton.ru.
5. CrossWorks for ARM, www.rowley.co.uk/ arm.
6. Image Craft, imagecraft.com/.
7. Описание тестов: www.raisonance.com/products/STR7/benchmark.php.