Компиляторы Си для PIC18

Использование языков высокого уровня в сложных проектах на базе микроконтроллеров фирмы Microchip до недавнего времени казалось неоправданным. Да, существовали компиляторы Си фирмы HT-SOFT (www.htsoft.com) и семейство компиляторов IAR (www.iar.com), которое включало в себя Си-компилятор для PIC16/17. Также существовал MPLAB C (www.microchip.com), но большим количеством ограничений и даже ошибок. Но из-за малого количества ресурсов и страничного распределения памяти использование этих средств было не очень удобным и оправданым. Кто пользовался HT PICC, наверняка помнит о том, как приходилось вручную распределять переменные по банкам памяти и пользоваться абсолютной адресацией.

Можно упомянуть еще компиляторы для микроконтроллеров Microchip PIC: CCS (http://www.ccsinfo.com/), Bytecraft (http://www.bytecraft.com/compile.html), C2C (http://www.geocities.com/SiliconValley/Network/3656/c2c/c.html), CCXX (http://www.bknd.com/), FED PICMicro® MCU C (http://www.fored.co.uk/ Eindex.htm). Также было сделано несколько портов свободных компиляторов для Microchip — ссылки на них можно найти на http://www.gnupic.org. Но данные компиляторы либо не предлагают демо-версий, либо находятся на начальном уровне разработки, либо имеют ограничения, которые не позволяют применять их в качестве рабочего средства. Далее в статье я попытаюсь объяснить, по каким критериям разделяю компиляторы на «профессиональные» и «экспериментальные». Но все равно кажется странным, что для популярного семейства микроконтроллеров, существовавшего на рынке длительное время, не было создано удобных и функциональных средств разработки. Это можно объяснить тем, что архитектура PIC16 не позволяла переложить на компилятор заботу о распределении ресурсов. Требуемые для языка высокого уровня глубокий стек, указатели с возможностью индексной адресации и т. п. отсутствовали в архитектуре семейства PIC16. Также повлияло разделение памяти программ и данных на банки и сложность манипулирования кодом или данными, занимающими несколько банков. То есть можно назвать архитектуру PIC (12, 16, 17) «неудобной» для реализации Си-компилятора. Все это тормозило разработку хороших компиляторов и использование этих средств разработчиками. Именно это подтолкнуло Microchip к разработке нового ядра PIC18, причем разработчики старались сохранить совместимость (хотя бы на уровне ассемблерных мнемоник) системы команд «младших ПИКов» и PIC18.

Особенности ядра PIC18

Попробуем рассмотреть, какие новшества были внесены в архитектуру PIC18, и как они могут «облегчить жизнь» компилятору языка Си высокого уровня. Я не буду подробно описывать архитектуру PIC18, с которой можно подробно ознакомится в статье PICmicro 18C MCU Family Reference Manual [1] фирменной документации на сайте или компакт-диске Microchip. При первом взгляде на систему команд кажется, что ничего существенного не изменилось, но это совсем не так:

- Следует рассмотреть работу индексных регистров (не отображенную в системе команд). Это INDFn, POSTDECn, POSTINCn, PREINCn, PLUSWn, FSRn (FSRnH + FSRnL), где n=0,1,2. Очевидно, что архитектура предоставляет три указателя, которые могут адресовать всю память данных. При этом поддерживается инкремент/декремент указателя и возможность базово-индексной адресации (при обращении к PLUSWn адрес памяти вычисляется как FSRn+WREG). Это одно из самых значительных архитектурных улучшений, которое позволяет реализовать работу стека данных для хранения параметров и локальных переменных при вызове функции. Таким образом, один из FSR может использоваться как указатель стека, другой как указатель стека при входе в функцию (frame pointer) и использоваться для доступа к локальным переменным, третий использоваться в качестве индекса массива. Следует заметить, что аккумулятор и индексный регистр — это один и тот же WREG, и время работы с локальными переменными будет дольше, чем с глобальными или статическими (непосредственная адресация).
- Следующим по важности улучшением можно назвать доступ к аппаратному стеку переходов. Целесообразность использования аппаратного стека для хранения 3-байтного адреса вызвана необходимостью быстрых вызовов (CALL) и прерываний, при использовании байтной шины данных потребовалось бы 3 лишних машинных цикла. Но при этом возможность манипулировать с указателем аппаратного стека и TOSU, TOSH, TOSL позволяет реализовать переключение контекста и многозадачность. Так что появилась возможность портировать многозадачную RTOS (например, uCOS-II www.micrium.com) на PICmicro! Также можно реализовать LIBC функции setjmp() и longjmp().
- Интересно использование «памяти быстрого доступа» (access bank) для хранения глобальных переменных и передачи параметров в функции.
- Возможность доступа к любой ячейке памяти посредством MOVFF позволяет эффективнее обращаться к большим структурам данных, занимающих несколько банков памяти (если рассматривать данную инструкцию как обращение к памяти, а банк, выбранный в BSR как рабочие регистры, то получим архитектуру, сходную с «настоящими» RISC-процессорами и возможность портировать GNU C компилятор на PIC!).

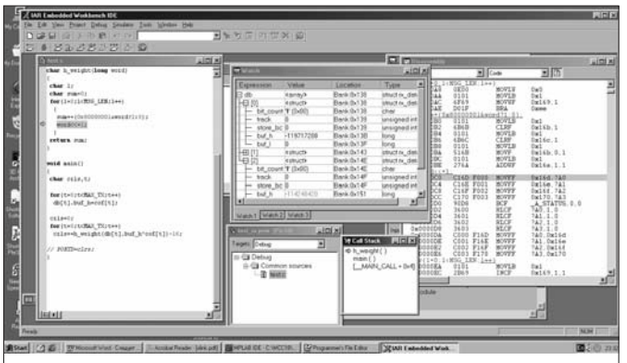

Рис. 1. *Среда IAR*

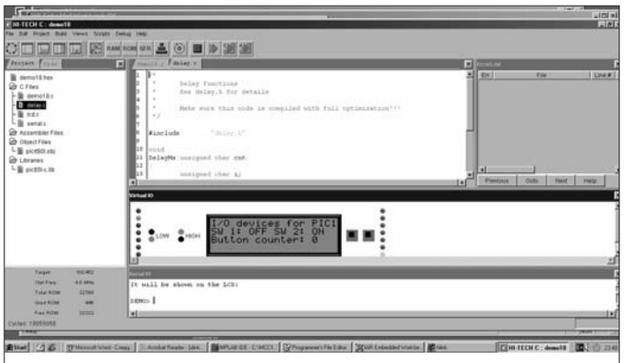

Рис. 2. *HTIDE*

- Необходимо упомянуть новую возможность табличного доступа к программной памяти с помощью команд TBLRD, TBLWR, что позволяет хранить и работать с большими массивами данных в программной памяти.
- Умножение 8-разрядных чисел за один такт.

Несмотря на кажущуюся схожесть архитектуры и набора инструкций семейств PIC16 и PIC18, новое семейство имеет архитектуру гораздо более приспособленную для реализации компиляторов Си и языков высокого уровня. Подтверждением этому является быстрый выход Си-компиляторов для PIC18. При этом IAR System предлагает не только Си, но и упрощенный вариант C++.

Интегрированные среды для работы с компиляторами Си для PIC18

Для удобной работы с компилятором Си он должен обладать интегрированной средой с менеджером проектов, симулятором и/или интерфейсом эмулятора, текстовым редактором и т. д. Про среды разработки от ведущих производителей компиляторов Си можно сказать следующее:

- IAR предлагает одинаковую среду (рис. 1) для разных процессоров (что должно облегчить перенос исходного кода на другую платформу и не требовать дополнительного времени на освоение).
- HTSOFT на момент работы над статьей предложила бета-версию среды HTIDE, которая пока еще не обладает устойчивостью, но зато имеет интересную возможность моделирования периферийных устройств (рис. 2).
- В среде от CCS отсутствует симулятор, то есть среда представляет собой менеджер проектов и редактор, ориентированный на работу с CCS C. В составе редактора есть средство, позволяющее «сгенерить» часть кода для инициализации микроконтроллера PIC Wizard (рис. 3).
- MPLAB от Microchip (16-bit версии 5.XX) — наиболее популярная среда разработки, позволяющая подключать любой из перечисленных компиляторов и имеющая наибольшую поддержку аппаратных эмуляторов (например, ICE2000), включая недорогие дебаггеры-от-

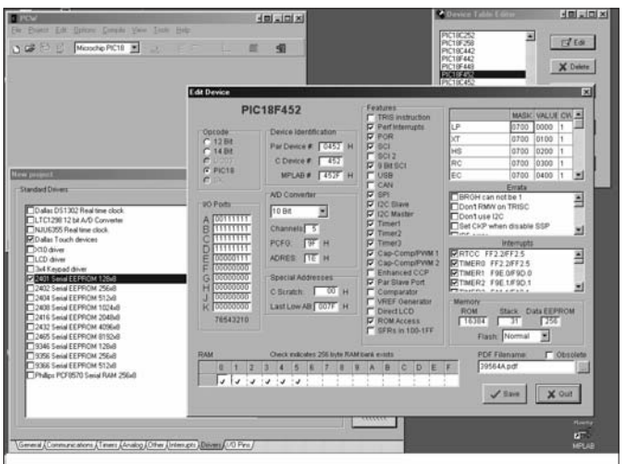

Рис. 3. *PCW и утилиты конфигурации (CCS)*

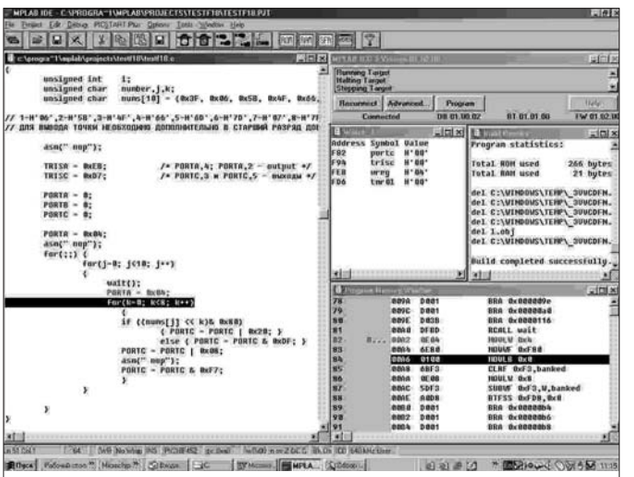

Рис. 4. *Среда MPLAB IDE 5.70 + ICD2 дебаггер*

Таблица 1. *Сравнение интегрированных сред разработки и отладки, работающих с компиляторами Си*

Интегрированная среда	MPLAB - IDE	HI-TIDE	IAR IDE	CCS C	Phyton Project-MC
«Свой» компилятор Си	MPLAB - C18	HI-TEC PICC18	IAR PIC18	CCS C	Нет
Программный симулятор	Есть	Есть	Есть	Нет	Есть
Поддержка аппаратных отладочных средств	ICE2000, ICD2, PROMATE II, PICSTART+	Нет	ICE2000	Нет	Эмулятор PICE-MC
«Мастер» (wizard) для генерации подпрограммы инициализации контроллера	Нет	Нет	Нет	Есть	Нет
Ассемблер и линкер (возможность включения модулей: ассемблерных или объектных)	Есть	Есть	Есть	Нет	Есть
Удобство работы, функциональность	Хорошее	Нет данных	Отличное	Среднее	Очень хорошее

ладчики-программаторы ICD (DV164001) и ICD2 (DV164007). Также для разработки проектов с использованием HT-PICC и CCS C среда MPLAB потребуется как симулятор или эмулятор (при наличии аппаратной поддержки) для отладки программ (рис. 4).

Следует заметить, что среда MPLAB не является очень удобной для отладки Си-приложений, так как в ней отсутствует возможность просмотра сложных структур данных, которые удобно создавать, используя возможности языка. Отсутствует окно просмотра переменных watch (есть возможность побайтного просмотра только глобальных символов); отсутствует возможность отладки Си «без входа в функцию», «до выхода» (step out, step over); нет возможности просмотра стека. То есть использовать среду MPLAB для отладки Си-проектов довольно неудобно. За длительное время существования этой среды производителем не были проведены улучшения отладки Си-программ. Огромный плюс — привычная среда (для бывших пользователей ассемблера) и удобство работы с отладочными средствами Microchip (эмуляторы, дебаггеры, программаторы).

- Учитывая пожелания разработчиков, Microchip выпустил 32-bit MPLAB 6.XX. Эта новая интегрированная среда предназначена полностью заменить MPLAB 5.XX. В новой среде переделан графический интерфейс, добавлены новые возможности, в редакторе появилось цветовое выделение операторов и директив компилятора. Кроме того, эта среда может работать с аппаратными средствами отладки, подключенными к USB-интерфейсу. Среда имеет возможность подключать компиляторы Си, но на момент выхода статьи поддерживалось подключение только MPLAB C18. Возможность подключения компиляторов других фирм появится в ближайшем будущем.
- Невозможно не упомянуть очень распространенную у нас альтернативную MPLAB среду отечественной фирмы Фитон — (www.phyton.ru),.в которой предусмотрено подключение Си компиляторов (например, HT-PICC).

Из рассматриваемых в статье сред разработки невозможно выявить однозначного лидера (см. таблицу 1). С одной стороны, отличное впечатление производит среда от IAR. Замечательно проработана периферия при работе с симулятором — отладчиком C-SPY, отображение SFR реализовано очень удобно. Главным недостатком является отсутствие поддержки аппаратных средств (например, ICD2). Поэтому для отладки приложений можно порекомендовать использование C-SPY в качестве симулятора для отладки логики работы программы, а MPLAB совместно с ICD2 (или ICE2000) — для проверки работы микроконтроллера в составе готового изделия.

Очень неплохо реализован просмотр переменных и сложных структур данных в среде от «Фитон», к ней же подключается внутрисхемный эмулятор собственного производства, что упрощает отладку. Недостаток — отсутствие демо-версий (за среду без Си-компилятора необходимо платить) и невозможность использования аппаратных средств отладки производства Microchip (например, недорогого ICD2). Также есть надежда, что в новой среде MPLAB 6.XX Microchip исправит недостатки, присущие предыдущей версии, и добавит подключение компиляторов других фирм.

Из вышесказанного следует, что выбор среды в первую очередь зависит от имеющихся в распоряжении разработчика аппаратных средств отладки (или их отсутствия).

Знакомство с компиляторами

Перед установкой можно ознакомится с так называемыми баг-листами — списком обнаруженных и устраненных ошибок компилятора. Как правило, все производители сопровождают новую версию компилятора таким списком. По нему видно состояние программы и перспективы ее развития.

MICROCHIP MPLAB C18: ошибок много и в компиляторе и в препроцессоре, и исправление многих достаточно критических ошибок занимает долгое время (если проследить историю развития). Использовать этот компилятор в большом коммерческом проекте кажется рискованным. Еще год назад вряд ли можно было бы рекомендовать использовать этот компилятор. Но выход за короткий промежуток времени нескольких новых интересных версий позволяет надеяться, что компилятор будет «доведен до ума».

HI-TECH PICC-18 COMPILER: очень хорошо зарекомендовал себя компилятор производства HI-TECH для PIC16, поэтому можно предположить, что для PIC18 будет не хуже. Список ошибок (bugfixes.txt) в директории установки показывает, что ошибки устраняются, а отклонения от стандарта и ограничения не являются критическими. Есть версия, работающая в Linux.

CCS C: список ошибок найти не удалось, но какое-то представление об ограничениях может дать список сравнения компиляторов, приведенных на сайте CCS: http://www.ccsinfo.com/ compare.html. Следует отметить, что в CCS C отсутствует отдельный ассемблер, линковщик. И единственным, очевидным способом использования ассемблера в CCS C проекте мне кажется использование директив #asm / #endasm.

IAR C/EC++ Compiler for Microchip PIC18: данный компилятор не только C, но и урезанный вариант C++ (Embedded C++). По моему мнению, предлагаемый вариант (диалект) C++ похож на Borland 3.1. Многие разработчики использовали Borland 3.1 для разработки DOS-приложений и оценили удобства, которые можно получить при «написании Си-программ на C++». В баг-листе сообщается, что обнаруженные и неисправленные ошибки отсутствуют.

Следующим важным элементом среды разработки Си является наличие библиотек. Желательно иметь (как минимум) LIBC и MATH. По отсутствию или только частичной реализации библиотек в остальных компиляторах я не рассматриваю эти компиляторы как «профессиональные». Во-первых, исходники библиотек существуют в свободном виде и для нормального компилятора не должно вызывать проблемы собрать библиотеку, во-вторых, если проект компилятора имеет серьезное развитие и поддержку, то написать свою версию библиотеки является следующим шагом, после того как компилятор заработал.

По этому критерию у MPLAB C18 отсутствует большая часть библиотеки вычислений с плавающей точкой, в HT PICC18 реализована большая часть стандартных библиотек. Список стандартных функций реализованых в CCS C уступает HT-PICC, но превосходит MPLAB C. В IAR C/EC++ реализована не только libc и математическая библиотека, но и предлагается частичная реализация билиотеки C++. Это конечно не libstdc++ (вообще использование этой библиотеки в простых встраиваемых системах неоправдано), но некоторые полезные свойства в dlib имеются.

Так как с использованием микроконтроллеров Microchip разрабатываются «встраиваемые системы», то желательно иметь библиотеки обслуживания периферийных устройств: RS232, CAN, I2C и подобные. В состав CCS C и MPLAB C18 входят библиотеки поддержки периферийных устройств — кристаллов PIC18. Это дает возможность предположить, что если такая библиотека будет работоспособна, то это поможет сохранить время при разработке приложения, использующего стандартный протокол.

Таким образом, если вы ориентированы на обработку чисел с плавающей точкой, наиболее заслуживающим внимания является продукция IAR, как компилятор, так и среда разработки/отладки. В случае разработки изделий на базе сложных периферийных протоколов (например, CAN) рекомендуем обратить внимание на MPLAB C18 и CCS. Для пользователей программного обеспечения Microchip и имеющих наработки на ассемблере MPASM, возможно наиболее привлекательным будет использование MPLAB C18. Компилятор от HI-TECH лишен как ярких достоинств, так и недостатков, но следует заметить, что хорошо зарекомендовал себя компилятор от HI-TECH для «младших» PIC-ов, и разработчикам, желающим перенести свои проекты, разработанные для PIC16 с использованием компилятора от HI-TECH, имеет смысл использовать HI-TECH PICC-18.

Для работы с ICD2 понадобится MPLAB IDE. Более доступным вариантом является использование MPLAB IDE (среда) совместно с MPLAB C18 PICC-18 или CCS C (компиляторы).

Для программистов, работающих на Си или поддерживающих несколько семейств микроконтроллеров, следует учитывать, что переносимость проектов HT-PICC, CCS C и MPLAB C18 на другую платформу может вызвать трудности из-за использования нестандартных типов данных, спецификаторов и ключевых слов.

Так как в статье делается обзор компиляторов, то свойства остальных средств разработки (ассемблера, линкера, остальных утилит) не рассматриваются.

Сравнения и тесты

Проведем несколько тестов с компиляторами MPLAB C18, IAR PIC18, HT PIC18 и CCS C, оценим их работу и возможности, и проверим, насколько верны наши предположения по использованию архитектуры PIC18. Данная статья не претендует на объективное сравнение характеристик различных компиляторов и остальных средств разработки, входящих в состав соответствующих программных пакетов. Предлагаемые для теста примеры являются либо фрагментами реальных проектов, либо специально предназначены для тестирования определенных свойств компиляторов. Полученные результаты по скорости выполнения и размеру исполняемого модуля могут быть улучшены за счет изменения опций оптимизации и переписывания исходного кода, но общая картина должна сохраниться.

Проведем несколько практических экспериментов, так как рассмотрение фирменной документации и таблиц сравнения компиляторов (которые у разных

производителей показывают совершенно разный результат) вряд ли является достаточным основанием для выбора средства разработки.

Для того чтобы подключить в среду MPLAB IDE компилятор (подробные инструкции см. в файле http://www.microchip.com/download/tools/picmicro/code/third/51234a.pdf на сайте или CD-ROM)), следует в Project->Install Language Tool указать пути к исполняемым файлам линкера, компилятора, ассемблера (при этом для средств HI-TECH следует указывать во всех случаях picc18.exe). А для сборки проекта, состоящего из нескольких файлов в Project->Edit Project->Node Properties… Language Tool, следует указывать линковщик, а потом с помощью кнопки Add Node подключать исходные файлы на Си или ассемблере с соответствующим средством и опциями.

Для MPLAB C18 требуется включать скрипт линкера (*.lkr) в список узлов (Add Node) проекта. Также следует помнить, что в составе MPLAB C18 используется другой линковщик (не тот, который входит в MPLAB 5.xx) — поэтому следует указать его.

Для подключения к MPLAB IDE проекта, собираемого IAR C/C++, следует помнить, что IAR-линковщик (XLINK) является универсальным и находится в директории common\bin, в то время как компилятор, ассемблер и симулятор зависят от типа процессора и для PIC18 находятся в pic18\bin.

CCS C во время установки прописывает данные в инициализационный файл MPLAB, и проблем с созданием CCS C проектов не возникает.

Такие «танцы с бубном» вокруг MPLAB не являются свойством Си, но увы, затрудняют переход на Си разработчиков, привыкших обходиться ассемблером. Эти недостатки MPLAB, связанные с универсальностью и простотой этой среды (также следует помнить, что MPLAB поставляется бесплатно), еще раз подчеркивают удобство среды IAR или PICE/PDS-PIC от «Фитон». Специализированные среды (IAR, «Фитон») способны работать с одним компилятором, но отладочная информация обрабатывается в них лучше, и процесс отладки становится более удобным и быстрым.

Для продолжения экспериментов пришлось переправить исходный код примера, чтобы он был «более» ANSI-шным, но при этом описанная конструкция языка сохранилась.

Во всех приведенных ниже примерах требуется либо сконфигурировать функцию printf, либо закомментировать ее. Вместо printf можно провести запись в какую-либо volatile переменную, например PORTD. Это нужно для того, чтобы «умные» компиляторы (HT-PICC и IAR) не выбросили из кода вычисления, результат которых не используется. Для сборки приведенных примеров в CCS C их следует дополнить (последние две строчки конфигурируют printf через последовательный порт)

```
#device PIC18F452
#use delay (clock=2000000)
#use rs232 (baud=9600)
```

Сравнения результатов будем проводить по отсчетам средств компиляции, а время исполнения измерять в симуляторе IAR C-SPY или Microchip MPLAB IDE. Единица времени во всех измерениях одна — машинный цикл, называемый «тактом» (что не совсем корректно для PIC, т. к. один машинный цикл выполняется за 4 такта генератора).

Пример (фрагмент программы помещен в main), на котором «спотыкается» HI-TECH PIC18 выглядит следующим образом:

```
#define MSG_LEN 23
#define MAX_TX 3

struct rx_data
{
 char bit_count;
 unsigned track : 1;
 unsigned store_bc : 7;
 long buf_h;
 long buf_l;
};

long cof[]={0xf8dd4258,
0xf9a42bb0,
0xf930b51c,
0xfa2561cc};

struct rx_data db[MAX_TX];

char h_weight(long word)
{
 char l;
 char sum=0;
 for(l=0;l<MSG_LEN;l++)
 {
 sum+=(0x8000000l&word?1:0);
 word<<=1;
 }
 return sum;
}

void main()
{
 char crls,t;

 for(t=0;t<MAX_TX;t++)
 db[t].buf_h=cof[t];

 crls=0;
 for(t=0;t<MAX_TX;t++)
 crls+=h_weight(db[t].buf_h^cof[t])-16;

//для того, чтобы компилятор не «выбросил» код из-за того, что данные потом нигде не используются
PORTD=crls;
//printf(«%x»,clrs);
}
```

	IAR C	MPLAB C18	CCS C
Размер функции h_weight, программных слов	132	194	–
Программа целиком, программных слов	2433	1320	–
Время исполнения, такты	2720	4017	4225

Программа создает в памяти массив, а затем обрабатывает его содержимое. При этом MPLAB C18 и IAR C/EC++ компилируют код. Сравним полученный исполнимый код при включенной оптимизации: код функции h_weight, скомпилированный IAR занимает 132 слова, а в MPLAB 194. Но в то же время полный код программы в IAR C 2433 байт (видимо, большой размер связан с подключением библиотечных функций) а время исполнения 2720 тактов. А в MPLAB C18 1320 байт и 4017 тактов. Время исполнения CCS 4225 тактов. Таким образом, сложные адресации в Си вы-

ражениях IAR C обрабатывает лучше. На рис. 1 представлена среда IAR: во время отладки примера и можно видеть, как удобно реализован просмотр структур данных и весь процесс отладки.

Следующим примером проверим работу вычислений с плавающей точкой. Используются тригонометрические и логарифмические функции из стандартных библиотек. Так как тестовые программы используют стандартные функции, параметры компиляции которых были оптимизированы при сборке библиотек, то можно не уделять внимания опциям компилятора. Опираясь на опыт, влияние параметров оптимизации на сгенеренный код не очевидно из описания, и следует поставить несколько экспериментов, прежде чем будет найдено «волшебное слово», заставляющее компилятор сгенерить лучший код.

```
#include <math.h>
#include <stdio.h>

main()
{
 float x,y;
 int i;

 for(i=0,x=0.1,y=0;i<10;i++,x+=0.03)
 {
 y+=sin(x)*cos(x)/log(1+x*3.3);
 }

 printf(«%f\n»,y);
}
```

	IAR C	HT-PICC 18		CCS C
Программа целиком, программных слов	5938	4978	4608	–
Время исполнения, такты	70762	328620	313690	405500

MPLAB C18 не компилирует пример из-за отсутствия в библиотеке математических функций. Код, полученный IAR C, выполняется за 70762 такта, ответ совпадает с вычисленным на ПК, код программы занял 5938 байт. CCS C компилирует пример, но время исполнения составляет 405500 тактов (посмотрите на таблицу сравнения компиляторов на сайте CCS :-)). В HT-PICC плавающая точка может представляться либо 32-битными, либо 24-битными числами. В первом случае код 4978 байт и время выполнения 328620 тактов. Во втором 4608 и 313690 соответственно. Таким образом, регулируя размер представления чисел с плавающей точкой в HT-PICC можно обеспечить выигрыш по ресурсам. На вычислениях с плавающей точкой выигрыш IAR C значителен.

Проверим работу библиотечных функций вывода (для работы sprintf требуется сконфигурировать библиотеку печати).

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

char a[20];

main()
{
 int i;
 char b[20];

 for(;;)
 {
 i=rand()&0xfff;
 sprintf(b,»%x»,i);
 strcpy(a,b);
 printf(«%s\n»,a);
 }

}
```

	IAR C		HT-PICC 18
Время исполнения, такты	6818	8423	3704

Исполнение в IAR C занимает 6818 тактов при размещении переменных статически, и 8423 такта при использовании стека. При использовании HT-PICC время исполнения 3704 такта. То есть целочисленные и строковые операции в HT-PICC реализованы эффективнее.

Для IAR EC++ напишем небольшой пример с использованием C и C++ и сравним объемы. Для того чтобы включить возможности EC++ необходимо в Project->Options… General указать Stack и DLIB, тогда в опциях компилятора появится возможность Enable EC++ syntax.

Также на этом примере можно проследить влияние параметров оптимизации на выходной код. Из рассматриваемых компиляторов большое количество параметров оптимизации предлагают HT-PICC и IAR. Если в HT-PICC можно поставить максимальное значение оптимизации, следует учитывать, что не всегда увеличение приводит к улучшению кода. В IAR C можно выбирать тип оптимизации: скорость или размер и тип размещения данных (статически или в стеке).

Программа тестирует использование частых вызовов функций и работу с указателями:

```
#include <stdio.h>

#define GEN_POLY 0x81
#define POLY_MASK 0x400
#define STN 4

int next(unsigned int * state)
{
 int ret;
 ret=*state & 0x1;
 *state&=~POLY_MASK;
 if (ret) *state^=(GEN_POLY);
 *state>>=1;
 return ret;
}


main()
{
 unsigned int st[STN];
 int i,j,t;

 for(i=0;i<STN;i++)
 {
 st[i]=POLY_MASK-1;
 }

 for(i=0;i<20;i++)
 {


 for(j=0;j<STN;j++)
 for(t=0;t<=j;t++)
 next(st+j);
 }

 for(i=0;i<STN;i++)
 {
 printf(«%x\n»,st[i]);
 }

}
```

При размещении переменных «статически» получаются следуюшие результаты (см. табл. на стр. 22).

При использовании стека (компилятор IAR) время исполнения увеличивается приблизительно на 10000 тактов (инструкция LFSR разрешена) и в случае оптимизации по скорости составляет 45750 тактов. Изменение размера кода при оптимизации по размеру составляет около 10%.

Таким образом, при многочисленных вызовах функции и работе с указателями лучшие показатели у CCS C.

Перепишем то же самое на C++

```
#include <stdio.h>

const unsigned int GEN_POLY=0x81;
const unsigned int POLY_MASK=0x400;
const unsigned int STN=4;

class poly {

unsigned int state;

public:

 poly() {state=POLY_MASK-1;};

 int next()
 {
 int ret;
 ret=state & 0x1;
 state&=~POLY_MASK;
 if (ret) state^=(GEN_POLY);
 state>>=1;
 return ret;
 };

 int get_state() {return state;};
};

//extern «C»
int main(void)
{
 poly st[STN];
 int i;

 for(i=0;i<20;i++)
 {
 for(int j=0;j<STN;j++)
            for(int t=0;t<=j;t++)


 st[j].next();
 }

 for(i=0;i<STN;i++)
 {
 printf(«%i : %x\n»,i,st[i].get_state());
 }

}
```

В данном тесте можно сравнить только IAR C против IAR EC++. Так как EC++ работает только с включенным стеком, то параметры будут хуже (как можно видеть из приведенных выше примеров способ хранения локальных переменных оказывает решающее воздействие на эффективность кода). После компиляции с выключенной оптимизацией получается 57200 тактов. С оптимизацией по скорости 46943 тактов (размер 8885 байт). С оптимимизацией по размеру 8637 байт (55506 тактов). То есть для перехода от C (с использованием метода хранения переменных на стеке) к EC++ требуются не очень высокие затраты, и в некоторых случаях можно пожертвовать эффективностью программы в пользу удобочитаемости. Также можно предположить, что C++ компилятор будет в дальнейшем улучшен.

	IAR C			HT-PICC 18	MPLAB C18	CCS C
	Без оптимизации	По коду	По размеру			
Программа целиком, программных слов				4144	1080	223
Время исполнения, такты	42000	32000	37300	15549	17508	14652

Итоги

По результатам комплексной оценки компиляторов из рассмотренных в статье средств наибольшее внимание следует уделить средствам IAR и HI-TECH C18. MPLAB C и CCS C можно применять в случае простых конструкций, без использования плавающей точки и функций стандартных библиотек. По эффективности кода управляющих конструкций (скорости и объему) IAR C/EC++ уступает остальным рассмотренным компиляторам (но значительно превосходит их на вычислениях с плавающей точкой и оптимизации сложных выражений). Возможно, что на программах большего размера, чем рассмотренные примеры, разница в скорости исполнения между ними уменьшится. Также следует учесть, что библиотеки в IAR занимают гораздо больше места, и сравнение приводилось по размеру всего исполнимого модуля, а не отдельной функции.

В статье остался нерассмотренным вопрос использования «периферийных» библиотек, входящих в состав MPLAB C18 и CCS C. Среди таких библиотечных функций особенно будет интересна работа с CAN (PIC18F458/448/258/248 имеют встроенный CAN-контроллер). Также интересна разработка метода оптимального кодирования для того или иного компилятора, но этот вопрос требует длительного исследования и накопления опыта, поэтому мы расскажем об этом в следующих выпусках. С другой стороны, компиляторы MPLAB C18 и CCS C быстрыми темпами совершенствуются (благо архитектура PIC18 этому способствует), и можно предположить, что через короткий промежуток времени ситуация кардинально изменится.

Но для полноценной разработки нельзя ориентироваться на выбор только компилятора. Должен также присутствовать полноценный программный симулятор и/или эмулятор. Поэтому окончательно сделать выбор помогут именно имеющиеся в распоряжении разработчика внутрисхемные средства отладки.

Дальнейшее изучение возможности использования Си на платформе PIC18 нужно проводить самостоятельно. Почти все рассматриваемые в статье программные продукты имеют демо-версии (с ограничением на время эксплуатации), и наилучшая рекомендация — попробовать собрать простой проект на языке Си (или смешанный Си плюс ассемблер) и понять, обеспечивает ли компилятор выигрыш по времени разработки и удобству дальнейшего сопровождения изделия. Время, потраченное на освоение новых средств разработки (в общем-то, незначительное), окупится скоростью и удобством разработки новых проектов.