KiT#58(5).qxd 5/3/2006 11:39 AM Page 6

Окончание. Начало в №4`2006

Иосиф КАРШЕНБОЙМ
iosifk.narod.ru

Квадрига Аполлона и микропроцессоры

Конструктивная модель стоимости (COCOMO)

Барри Боехм, «отец» оценки программирования, сформулировал так называемую Конструктивную модель стоимости, или COCOMO [8], для определения стоимости программных проектов. Хотя COCOMO и не совершена, но это, вероятно, наиболее известная модель, позволяющая выполнять прогнозы и количественные оценки.

Боехм определяет три различных режима развития: organic, semidetached и embedded (встроенный), где «встроенный» означает программный проект, сформированный при множестве ограничений в аппаратных средствах, программном обеспечении и иногда даже в инструкциях (в командах процессора). Это довольно хорошее описание типичной встроенной системы, хотя, насколько мы знаем, Боехм не думал о встроенном программном обеспечении.

Вот как по методике COCOMO можно представить требуемое число человеко-месяцев (ММ) для того, чтобы начать поставки работающей «встроенной» системы заказчику:

$$MM = \prod_{i=1.15} F_i \times 2{,}8 \times KSLoC^{1.20},$$

где KSLoC — число строк исходного текста в тысячах, а Fi — 15 различных коэффициентов стоимости.

Коэффициенты стоимости включают: требуемую надежность, сложность изделия, ограничения в режиме реального времени и т. д. Каждому коэффициенту стоимости назначен вес, который изменяется от значения немного меньшего, чем 1,0, до немного большего. Для первого приближения разумно предположить, что коэффициент стоимости изображается от нуля и приблизительно до 1,0 для типичных проектов.

Что же показывает эта формула?

Боехм применил экспоненты с показателем степени, равным 1,20. Но, возможно, для встроенных систем, работающих в реальном масштабе времени, этот показатель будет ближе к 1,5. а для некоторых медленно работающих организаций его значение ближе к 2. Требуемое для проекта число человеко-месяцев растет экспоненциально. Или, в первом приближении можно сказать так — удвойте размер проекта, и проектные задания вырастут больше чем в квадрате. Что же это — гибель больших проектов?

Давайте выберем промежуточное значение показателя степени, равное 1,35, которое находится посередине между оценкой Боехма и оценкой для встроенных систем. На рис. 1 показано, как происходит крах производительности при росте размера программы.

Вот после этого автор статьи может смело ответить всем аппонентам, которые ему задавали вопрос: «А что собственно сложного в тех системах заправки, которые Вы когда-то разрабатывали? Несколько микроконтроллеров, датчики, исполнительные механизмы, ну и что? Да сейчас любой студент за пару часов на лабораторной работе «проворачивает» сотни строк кода.» Ответ виден на графике. Студент работает при уровне производительности 150–200 строк кода в месяц, в то время как для отладки больших систем приходилось работать с производительностью в 30–50 строк кода в месяц. Так что же делать с большими проектами? Отлаживаться годами? Но ведь тогда сам разрабатываемый продукт устареет и будет никому не нужен.

«Разделяй и властвуй»

Этот лозунг известен очень давно. Еще много веков назад древние полководцы предпринимали всевозможные меры к тому, что-

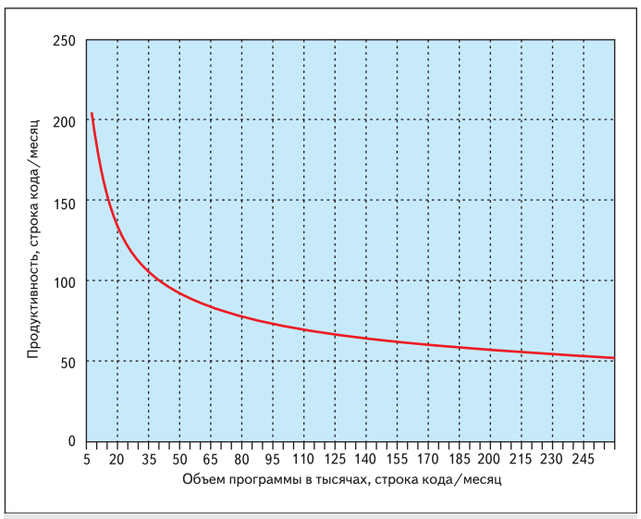

Рис. 1. Крах производительности

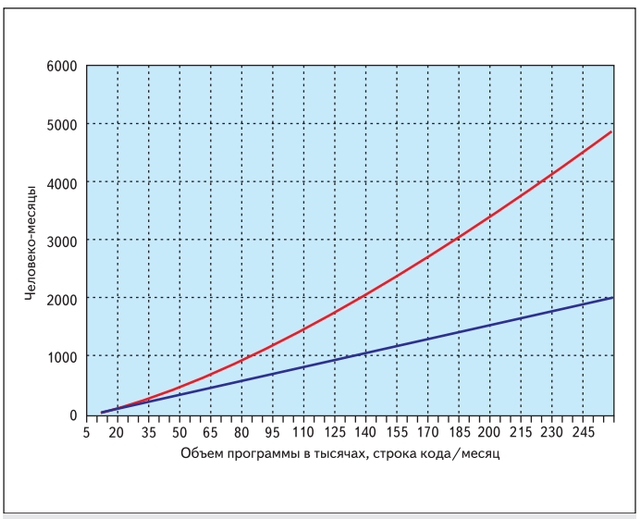

Рис. 2. Как обмануть экспоненциальный рост графика

KiT#58(5).qxd 5/3/2006 11:39 AM Page 7

Таблица 5. Сокращение сроков разработки при различных вариантах разбиения проекта

Число процессоров	Тысяч строк кода	Человеко-месяцев	Сокращение сроков в %	Примечание
1	100	1 403	100	
4	1x50 + 3x20 + 10%	909	65	Добавлено 10% к объему кода для организации связи между контроллерами
5	22	1 030	73	
10	1x20 + 9x10	724	52	

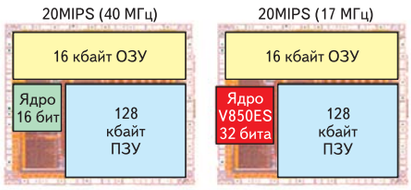

Рис. 3. Соотношение площадей для 16-разрядного и 32-разрядного микропроцессоров

бы разбить противника по частям. Попробуем применить этот же метод к методике разработки сложных проектов.

Что произойдет при разделении проекта на небольшие части? Если судить по графику на рис. 1, мы переместимся по кривой влево, то есть в сторону больших производительностей. Представим, что мы переместились влево до уровня объема программы в 20 KloC. Что же будет с производительностью в этом случае? На рис. 2 показан результат.

Верхняя кривая на рис. 2 представляет собой график, построенный на основании COCOMO. Нижняя кривая представляет собой график, выполненный на предположении того, что мы ведем разработку системы на уровне производительности в 20 KLoC. График, и, следовательно, стоимость будут расти линейно с увеличивающимся объемом программы.

Итак, вот он выход из тупика для больших проектов — необходимо разделить проект! Надо выделить из него маленькие куски, и к каждому такому фрагменту проекта прикрепить крошечную группу разработчиков, работающих в более или менее изолированных условиях, и именно в этом случае графики не будут расти по экспоненте. А если так не сделать, то рано или поздно, программное обеспечение, в конечном счете, раздавит любой новый проект.

Есть несколько способов разбиения проекта на части. Но при разработке встроенного программного обеспечения у нас есть самый эффективный способ, недоступный для IT-программистов. Рассмотрим его более подробно.

Если мы хотим сохранить производительность на высоком уровне, скажем в 20 KLoC, — мы должны разбить программу на части, которые будут выполняться в множестве дискретных объектов, при этом каждый их них не должен иметь объем программы больше, чем 20 KLoC. Инкапсулируем каждую часть программы в ее собственный процессор.

Правильное решение: добавить процессоры просто ради ускорения графика и сокращения затрат на разработку

Здесь позволю себе привести два примера. Первый пример — это то, как смотрят на дело разработчики у нас. Однажды один из коллег задал вопрос в телеконференциях — вот у него есть небольшой производительности микроконтроллер, и он хочет сделать устройство, которое бы управляло — далее следовал почти стандартный набор датчиков и АЦП — и как ему выполнить проект, в одном микроконтроллере или в нескольких? Ответ был почти единогласный — в одном. И никто из участников конференции не задал вопрос о том, насколько трудоемок предполагаемый проект. О чем говорит эта ситуация? Она говорит о том, что большинство проектов, выполняемых участниками телеконференции, небольшие, и они все «помещаются в одной голове». То есть, в соответствии с приведенными выше выкладками, коллеги предложили наиболее эффективное с их точки зрения решение. А почему почти единогласно? Потому что ваш покорный слуга предложил посмотреть статью об интерфейсе LIN [9]. При этом, аналоговая подсистема могла бы быть выделена в отдельный подпроект, и эта часть проекта могла бы быть сделана другими людьми, а это, как было сказано выше, привело бы к общему ускорению разработки.

А вот второй пример: один мой знакомый переводил статью, написанную инженером-китайцем. Статья о том, как микроконтроллер управляет стиральной машиной. И вот среди сетований на плохой английский автора, он вдруг сказал мне следующее: «И никак я не пойму, зачем же они в стиральной машине устроили локальную сеть? Какой в этом смысл? Разве не проще бы было применить один микроконтроллер, тем более, что их теперь так много и можно выбрать такой, который удовлетворял всем заданным параметрам?» А смысл, как мы теперь видим, в том, что разработчики контроллера для стиральной машины применили, скорее всего, интеллектуальные датчики — датчики с выходом на сеть. И вместо того, чтобы заниматься обработкой аналоговой информации от термометра, центральный контроллер просто посылает запрос и получает ответ, соответствующий значению температуры. Возможно, что проект велся разными группами разработчиков. Возможно, эти люди работали на разных предприятиях. Существенно здесь то, что при таком построении проект было сделать проще, а значит, и быстрее.

При разбиении проекта на несколько подпроектов упрощается ведение проекта, внесение изменений. Проект становится более понятным при сопровождении.

И как говорит в таком случае Джек Ганссл: «Транзисторы дешевы. Разработчики дороги. Разбейте систему на маленькие части, установите локальные процессоры в каждой части и затем используйте маленькую группу программистов, чтобы разработать каждую подсистему. Сократите взаимодействия и связь между компонентами и между инженерами».

Сравним несколько вариантов выполнения проекта. За основу возьмем вариант, выполненный на одном процессоре с объемом программы в 100 тыс. строк кода. Вычисление трудозатрат на разработку по COCOMO дает 1403 человеко-месяца.

Данные, приведенные в таблице 5, показывают, что при выделении даже половины проекта в три микроконтроллера общая трудоемкость сокращается на 35%. Если провести разбиение проекта так, как указано в четвертом варианте, то проект можно сделать вдвое быстрее! Однако здесь надо добавить, что такое сокращение сроков разработки возможно только для проектов, которые можно разделить на части.

Затраты на программирование — это еще не все

Затраты на изделие будут состоять не только из затрат на разработку. К ним добавятся еще и затраты на производство. А это печатные платы и те самые добавленные процессоры. Но кроме затрат, связанных непосредственно с программированием и производством, есть еще дополнительные расходы. Они называются разовыми затратами на разработку (Non-recurring engineering — NRE). Разовые затраты — это затраты на подготовку рабочего места, оснащение приборами и программами. И здесь важно понимать, что чем быстрее производится разработка, тем больше изделий может быть продано, и соответственно, расходы NRE, отнесенные на одно изделие, будут ниже.

Да, добавление процессоров на печатную плату повышает стоимость изделия. Но транзисторы особенно дешевы в ASIC. Полный 32-разрядный центральный процессор может иметь всего 20–30 тыс. вентилей. На рис. 3 довольно схематично показано соотношение площадей кристалла для 16-разрядного и 32-разрядного процессоров (данные фирмы NEC). Как видно, площади процессоров в основном определяются площадями ПЗУ

KiT#58(5).qxd 5/3/2006 11:39 AM Page 8

и ОЗУ. Площади ядер процессоров имеют практически одинаковые размеры и, кроме того, при той же производительности ядро 32-разрядного процессора может работать на более низкой тактовой частоте. И именно поэтому такие производители микропроцессоров, как NEC, отказываются от 16-разрядных микроконтроллеров в пользу 32-разрядных. Таким образом, добавление процессоров в ASIC позволяет значительно повысить производительность — особенно в том случае, когда процессоры используют общую память программ. Более подробно этот случай будет рассмотрен в приложении к сетевым процессорам.

В FPGA этот принцип тоже наблюдается довольно отчетливо. Если несколько лет назад контроллер в FPGA проигрывал по характеристикам стандартным микроконтроллерам или микроконтроллерам в ASIC, то теперь ситуация становится несколько иной. Микросхемы FPGA значительно подешевели и могут конкурировать со стандартными микроконтроллерами как по цене, так и по производительности. И это особенно заметно по обилию процессорных ядер, предлагаемых фирмами-изготовителями FPGA и по развитым средствам поддержки проектирования для этих ядер. Особенно хочется отметить, что шины, применяемые этими производителями, ориентированы на работу с несколькими активными абонентами, что значительно упрощает интеграцию.

Конечно, микроконтроллеры в FPGA никогда не смогут догнать лучших представителей ASIC. Но, тем не менее, они имеют свою нишу на рынке, и эта ниша постепенно расширяется. При проектировании в FPGA применение микропроцессоров дает особенно большой выигрыш по производительности. Замена «тяжелого» статического автомата на IP-ядро микроконтроллера позволяет перейти от отладки алгоритма в «железе» к отладке в «софте». Или, другими словами, от отладки в ModelSim к симулятору программ микроконтроллера. В чем разница? Симулятор ModelSim предназначен для отладки аппаратных средств. Если микропроцессор выполняет каждую команду за несколько тактов, то ModelSim все это и просимулирует. И уже дело пользователя разбираться, что и в каком такте находится. По сравнению с ModelSim программный симулятор микроконтроллера позволяет менять переменные «на лету», то есть непосредственно во время сессии симуляции, обходить какие либо ветви программы, повторно выполнять части программы. И, конечно, для программного симулятора намного проще выполнять различные трассировки и остановки по контрольным точкам.

А теперь надо сказать о добавлении микроконтроллеров в FPGA. Автор уже имеет опыт выполнения подобных проектов. Часть одного такого проекта описана автором ранее [10]. Когда выполнялось задание по разработке встроенного в FPGA микроконтроллера, алгоритм удалось разделить на две части. Одна часть алгоритма, та, которая управляла микросхемами ЦАП, АЦП и флагами, была реализована в 17-разрядном микроконтроллере. Он был ориентирован на анализ готовности, пересылку данных, небольшую DSP-обработку и общение с хостом по последовательному каналу связи. Вторая часть алгоритма — та, что управляла полусотней реле, также была реализована на микроконтроллере. Но для этого микроконтроллера основными задачами были проверка входных битовых флагов и текущего состояния алгоритма, а также управление битовыми переменными, которые проецировались на реле. Поэтому второй микроконтроллер был битовым. Он выполнял команды битовой обработки и пересылки результатов в память.

Такое разделение алгоритмов позволило упростить оба микроконтроллера, получить более компактный код в памяти команд для каждого из них и ускорить сроки разработки и отладки.

Контрольный вопрос

Заканчивая тему разделения проекта, позволю себе задать читателям контрольный вопрос. Вот что написал коллега в телеконференции: «Помогите мне сделать гальваническое разделение сигнала от микроконтроллера. Я уже попробовал, соединил источник сигнала с микроконтроллером напрямую, сделал программу и получил нужный мне результат. И вот теперь я хочу иметь гальваническое разделение». Как и следовало ожидать, он получил массу ответов об аналоговой гальванической развязке, о ее нестабильности, высокой стоимости и т. д. А теперь представим, какой ответ надо было дать «в стиле Джека Ганссла». Итак, уважаемый читатель, каков будет ваш вариант ответа?

И такой ответ дал коллеге автор этой статьи. Ответ был следующий: «Если Вы уже отладили программу и получили требуемый результат, то надо взять еще один процессор, самый дешевый из той же серии, загрузить в него отлаженную программу, а развязку между процессорами сделать по цифровому интерфейсу». При таком подходе к делу процесс разработки значительно упрощается и элементная база унифицируется. Нет необходимости изучать работу других компонентов для новых узлов схемы.

Риск, шампанское и цена ошибки

Всем известна фраза о риске и о шампанском. Она трактует риск как условие победы. И мы к этому привыкли. Но, как говорил один мой сослуживец, «все это так, но только трошечки не так». Когда автор много-много лет назад только начинал заниматься водным слаломом, ему пришлось пройти обучение на туристском семинаре при клубе туристов. И вот что оказалось. Как объяснял наш наставник, только «чайник» сидит в лодке без спасательного жилета. Сидит верхом на своем рюкзаке. «Супера» — совсем другое дело. Каска, спасжилет, герметичные мешки для вещей, страховочная веревка. Ну и зачем «суперам» все эти предосторожности, например на спокойном озере, где нет порогов? Ответ прост — так меньше риска! Рисковать можно в бурных порогах, но и при этом все условия для спасения должны быть выполнены. И, как говорили нам наставники, все эти нехитрые, с первого взгляда меры, потребовали своей цены. А цена — от травм и переломов и до жизней. Хочешь по-своему — заплати свою цену за риск. Правда, однажды, когда автор попытался дать совет «крутому чайнику», то в ответ услышал такую фразу: «Ты парень, сколько лодок уже сломал? Ни одной? Да я уже три лодки сломал, а ты мне советовать лезешь!»

Для чего нужно было это «лирическое» отступление? Просто на этом примере хочется еще раз подчеркнуть: риск риску — рознь. Есть риск оправданный, а есть риск, на который идут либо по глупости, либо от бесшабашности.

Разработка нового устройства — это всегда риск. Даже если у вас есть опыт разработки аналогичных изделий, даже если уже готов эскизный вариант. Все равно вы рискуете, так как проект может не получиться. Любая разработка — это риск. Ошибки в ТЗ, в схеме, в конструкции, задержки с комплектацией и т. д. И все это вы, наверное, знаете. Но это неизбежный риск. А вот теперь давайте посмотрим, как уменьшить другие факторы риска. Далее хочется привести часть письма в телеконференцию, где автор пытается объяснить свою точку зрения на риск разработчика. В теме, которая обсуждалась в тот раз, шла речь об экономии вентилей в FPGA и о нестандартных триггерах Шмитдта. Автор же предложил использовать цифровую обработку сигнала, которая заведомо снимала все проблемы по обработке сигнала, но, возможно, потребовала бы больше вентилей в FPGA.

«Задача разработчика — уменьшить риски. Если Вы заложили избыточность, но сделали проект в срок — Вы можете лишиться только части прибыли из-за более дорогих комплектующих. Но если Вы где-то решили «сильно» экономить и заложить решения, «которые не всегда», то, возможно, весь проект не заработает. Но только вы об этом узнаете в тот самый злополучный последний день. И вот представьте, что проект НЕ работает. Фирма — банкрот! Как вам такая альтернатива? Поэтому всегда можно и нужно для первого изделия заложить избыточность. Проверить, что работает все остальное, а я уверен, что проблема с оптроном не самая сложная. И уже потом, переходя к серии, можно и нужно оптимизировать проект. Но, повторяю, проект при этом УЖЕ будет работать и у Вас будет финансирование для продолжения работ. Вот об этом я стараюсь

KiT#58(5).qxd 5/3/2006 11:39 AM Page 9

написать. Надо уменьшать риск! Надо применять проверенные решения, которые помогут избежать каких-либо проблем. Экономить вентили — это очень неправильно. А что касается моего опыта, то, к моему сожалению, я уже прошел через описанную здесь ситуацию, когда руководитель проекта мне заявлял: «Здесь у нас инофирма и ваш советский опыт мне не нужен...». Вот поэтому и предостерегаю... Постарайтесь меня понять и не обижайтесь».

Но риск, связанный с разработкой, это только часть того, что приходится преодолевать при современном производстве. А что такое современное производство? «Евроремонт» в офисах, модная секретарша в приемной, новенькие компьютеры? Нет, нет и нет, господа читатели. Современное производство — это новые взгляды на стиль работы. И именно это главное. Вот только позволю себе привести пару фраз, показывающую разные взгляды на жизнь «у нас и у них». Несколько лет назад автору этой статьи довелось быть в служебной командировке на заводе, где проводился монтаж плат. Завод этот расположен в 50 километрах от Эдинбурга. Одним словом, шотландская провинция. Завод — это, собственно, легкая постройка в промышленной зоне на окраине поселка. Несколько автоматов по установке компонентов, несколько печей, человек 20 работников — вот и все. Когда мы с напарником спросили Джо — инженера, который с нами работал, — можем ли мы задержаться вечером, чтобы поработать сверхурочно, то он просто вытащил из кармана связку ключей и отдал их автору этой статьи. Что касается Джо, то он когда-то работал в СССР, поэтому он понимал русский юмор, и я уже об этом знал. И тогда я задал Джо такой вопрос: «Джо, это что ключ от всего завода? Да у меня дома для почтового ящика ключ более «крутой». А где же вышки с прожекторами и автоматчики с собаками?» Джо понял мою шутку. А вот его ответ заставил меня задуматься о различии во взглядах на жизнь. Ответ был такой: «А у нас тут нечего красть, да и некому!». Конечно, это провинция, а не мегаполис, но все же! И на шутку вроде совсем не похоже…

Главное — то, как построено производство, насколько оно стабильно. И здесь могу порекомендовать вам, уважаемый читатель, изучить интервью с генеральным директором ОАО «Морион» Яковом Вороховским [11]. Особенно внимательно отнеситесь к тому, насколько сложно, даже в условиях стабильно работающего производства, показать партнерам и заказчикам, что предприятие принимает все меры по снижению всех составляющих риска. И только когда заказчики поймут, что их поставщик не делает рискованных авантюр, и все то, что они должны получить по контракту, будет отгружено вовремя, вот только тогда можно будет называть производство современным.

Так как же можно уменьшить риск при разработке проекта? Для этого есть много способов. Часть из них покажется настолько тривиальными, что читатель, может быть, спросит: «а разве другие так не делают?»

Меньшие системы содержат меньше ошибок

Итак, разделили проект на части. Каждая часть стала более простой, чем весь проект в целом, следовательно, такая часть содержит значительно меньше ошибок. А чем меньше абсолютное число ошибок, тем меньше и вероятность появления (норма) ошибок в части проекта. По оценкам, приводимым специалистами по проверке программ, нормы ошибки для больших функций были в 2–6 раз выше, чем для меньших. Разделение необходимо для того, чтобы сократить график разработки и отправить клиентам изделие более высокого качества.

Национальный институт стандартов и технологии — NIST (National Institute of Standards and Technology) нашел, что слабое тестирование приводит к потере приблизительно $22 млрд в год из за программных отказов [12]. Испытание программ является тяжелым трудом, поскольку программы растут, и число ветвей выполнения программы взрывообразно увеличивается. Роберт Гласс оценивает, что при каждом 25%-ном увеличении программы происходит удвоение ее сложности из-за увеличения количества сложных вызовов, созданных функциональными запросами, выборами путей решения и т. п. [13].

Стандартные методы, проверяющие программное обеспечение, в ряде случаев просто не справляются с поиском ошибок. Большинство исследователей находит, что обычная отладка и оценки QA (оценки качества) находят только половину ошибок программы.

Разделение системы на меньшие куски, распределенные по множественным процессорам, где в каждом есть единственная сильно связная функция, уменьшает число путей управления и, несомненно, делает систему более тестируемой.

IT-отрасли очень быстро развиваются, и их участникам необходимо успевать за рынком. Не успел с поставкой продукции — и на рынок может выйти конкурент с аналогичной продукцией. Примером могут быть «войны» стандартов, связанных с записью информации на оптические диски. Если фирма не успевает выйти на рынок, то он может быть «завоеван» другой компанией, которая «продвигает» на рынок свой формат записи. Но даже при такой гонке нельзя поставлять на рынок плохо работающее изделие. Изделие, цена которого не будет превышать нескольких сотен долларов, может вызвать отказ технологического объекта ценой во многие тысячи и даже миллионы долларов. А это значит, что даже один судебный иск, выигранный против вашей фирмы, может фактически привести к ее банкротству. Сделайте разделение проекта, чтобы ускорить график разработки, отправьте изделие более высокого качества, и так, чтобы оно было должным образом проверено. Уменьшите риск тяжбы.

Добавление процессоров увеличивает системную производительность, и не удивительно, что при этом одновременно происходит сокращение времени на разработку. Но здесь необходимо сделать еще одно замечание. Всем известна поговорка: «запас карман не тянет». Как правильно выбрать микроконтроллер или FPGA? Как надо начинать проект? Какой запас необходимо сделать на «непредвиденные расходы»? Для начала разработки проекта, или его части, надо создать Техническое задание (ТЗ). Как надо составлять ТЗ? Автор может предложить методику, изложенную в «Записках Инженера» о ТЗ и гайке М3 [6]. Затем можно выполнить «оценочный проект» и его просимулировать и откомпилировать. Компилятор сообщит вам требования к аппаратному обеспечению. А после этого необходимо выбрать аппаратные средства так, чтобы на этапе отладки первых образцов был свободный аппаратный ресурс, который можно было бы задействовать для вспомогательных программ и для тестирования. После окончания отладки в большинстве случаев можно выбрать аппаратные средства с меньшим ресурсом, необходимым для работы серийного изделия. Невыполнение этого несложного правила приводит к тому, что система, загруженная на 90% от возможности процессора, удваивает время на разработку (по сравнению с тем случаем, когда процессор загружен на 70% или меньше) [14]. При 95%-ной загрузке процессора ожидайте увеличения втрое проектного графика. Когда надо добавить только горстку байтов для того, чтобы закончить проект, даже наиболее тривиальная новая подпрограмма может потребовать недели, поскольку разработчики вынуждены будут переделывать большие фрагменты кода, чтобы освободить немного памяти команд. Аналогичная ситуация происходит и в том случае, когда не хватает времени на обработку данных из-за низкой тактовой частоты центрального процессора.

Больше — значит больше

Сегодня на рынке уже появились 32-разрядные микроконтроллеры по цене $1. IP-ядра 32-разрядных процессоров стали таким же «ширпотребом», каким раньше было ядро MCS51.

Сегодняшние специализированные интегральные схемы, например такие, как сетевые микропроцессоры, уже содержат несколько ядер для обработки пакетов. DSP-процессоры (например, BlackFin) становятся «двухголовыми». Процессоры типа Sharc изначально проектировались для работы в мультипроцессорных кластерах. Так что теперь программисты могут разносить код в несколько процессоров. И это, как было описано выше, позволяет резко ускорить время разработки. Меньшее время поставки изделия означает

KiT#58(5).qxd 5/3/2006 11:39 AM Page 10

более низкие технические затраты. Когда NRE должным образом амортизируется при продаже изделий в промышленных количествах, меньшие технические затраты приводят к более низкой стоимости товаров. Клиент получает новый товар быстрее, и этот товар получается более дешевым.

Или, возвращаясь к старым методам работы, вы можете продолжить разрабатывать огромные монолитные программы, выполняющиеся на одном процессоре, теряя возможность выиграть и победить и превращая ситуацию в ту, где каждый проигрывает.

И еще один неожиданный вывод, полученный из этой статьи

В процессе написания этой статьи, автор пришел к следующему неожиданному выводу. Что мы обсуждали здесь? Только производительность труда разработчиков-программистов. Однако давайте посмотрим на дело шире. В промышленности работают как малые и средние, так и большие предприятия. Но вот только большие предприятия не заинтересованы в маленьких проектах. Для больших предприятий «мелкотемье» губительно. Оно требует полного комплекта документации, проработки всех экономических параметров. А у больших предприятий их управленческие службы обычно не имеют соответствующих мощностей. Мало того, они подчас не умеют реализовывать большие проекты по частям. То есть, если в процессе работы над большим проектом какая-то его часть может быть выделена как IP, то она должна быть оформлена как отдельный продукт и немедленно выставлена для продажи. Но для больших предприятий такой стиль работы представляется слишком хлопотным. И, поскольку у больших предприятий накладные расходы, отнесенные на единицу стоимости продукции — гораздо больше, чем у малых и средних, то в соответствии с приведенными здесь выкладками они имеют низкую производительность и поэтому обречены иметь низкую эффективность. Кроме того, в больших коллективах происходит нивелирование лучших разработчиков до уровня средних. Вот почему отечественные гиганты не смогли выиграть гонку с малыми и средними предприятиями. Где же выход? Выход в том, что в промышленности необходимо иметь разные предприятия. И малые, и средние, и большие. Именно малые предприятия, имеющие наибольшую эффективность при малых и средних проектах, выигрывают чаще всего еще и потому, что они наиболее эффективно используют своих лучших программистов. И если наша промышленность не сможет эффективно генерировать малые предприятия и поддерживать их, то тогда и большие предприятия будут обречены на вечное отставание от мирового уровня. А вот результат работы малых и средних предприятий должен быть интегрирован в продукцию больших. Именно так и поступают многие малые фаблесс-компании. Они производят разработки, доводят продукцию до внедрения и продают права на ее производство большим фирмам. А уже большие фирмы получают свою прибыль за счет увеличения объема продаж. Так что можно сказать так: если будет рост малого и среднего бизнеса, значит и будет рост всей отрасли в целом. Пример? Да за примером далеко ходить не надо. Фирмы Sun и Actel, как и многие другие фаблесс-компании, уверенно работают, хотя и не имеют своего кремниевого производства. Компания ARM — тоже довольно известный пример. Но это предприятия средние. А мелких предприятий — и не перечесть!

Поэтому жаль, что у нас в стране нет инкубаторов, выращивающих мелкие фирмы так, как это принято на Западе, нет и технопарков в промышленных масштабах, так чтобы средние и мелкие фирмы ощущали помощь и поддержку при развитии, росте и продвижении своих товаров и услуг.

В заключение

Автор надеется, что эта статья вызовет читательский интерес. Но некоторые моменты в статье являются спорными. Да, разбиение вычислителя на несколько процессоров — это основной процесс, который сейчас имеет место повсеместно. Начиная от многоядерных процессоров AMD и Intel, сетевых процессоров, в которых для обеспечения производительности используются процессорные кластеры — и до встроенных в FPGA процессоров, где к основному софт-процессорному ядру можно подключить один или несколько сопроцессоров. И хорошо, если речь идет о процессорах в FPGA, где можно использовать сильно связанные структуры. А если использовать слабосвязанные структуры, то при разбиении системы на несколько процессоров необходимо добавить программное обеспечение для передачи информации между процессорами. Отладка сетевого программного обеспечения тоже требует определенных усилий и опыта. Передача данных по сети требует времени на пересылку и обработку сообщений. При настройке сети потребуются дополнительные отладочные средства.

Но в целом можно сказать, что разбиение системы на отдельные подсистемы действительно помогает сократить усилия по разработке и отладке. Мало того, в отдельных подсистемах снижаются требования к быстродействию процессоров, к их объемам памяти и другим ресурсам. И опыт, полученный автором при отладке многопроцессорных систем управления это действительно подтверждает. Особый интерес данная методика представляет при проектировании узлов в FPGA. Разделение алгоритма на несколько встроенных микроконтроллеров, причем микроконтроллеров разной архитектуры и с разной системой команд, позволило оптимизировать разработку и значительно сократило сроки.

Снижение цены на микроконтроллеры привело к еще одному «облегчению жизни» для разработчиков. На рынке появились «умные» датчики (со встроенными микроконтроллерами). Для работы с такими датчиками уже не надо выполнять операции, характерные для аналоговых измерений. Процессор отправляет вызов и получает цифровой код, соответствующий давлению или температуре. При таком подходе разработчику системы уже не надо знать, как производить калибровку датчика, или как долго надо фильтровать код, чтобы получить истинное значение параметра. Появляются программные инструменты, позволяющие автоматизировать обмен сообщениями по интерфейсу LIN. Для того чтобы спроектировать распределенное устройство обработки данных разработчику теперь необходимо только при помощи соответствующего программного инструмента сформировать базу данных и определить кадры сообщений. Программный инструмент генерирует С-коды, которые встраиваются в контроллеры.

Хочется верить, что темы, затронутые в этой статье, будут полезны как отечественным разработчикам, так и их менеджерам.

Литература

1. http://www.iosifk.narod.ru/engineer_storys.htm
2. Ganssle J. Subtract software costs by adding CPUs. Embedded.com.
3. Demarco, Tom and Timothy Lister. Peopleware: Productive Projects and Teams, Dorset House Publishing Company, New York, 1999.
4. Jones, Capers. Applied Software Measurement: Assuring Productivity and Quality, McGraw Hill, New York, 1991.
5. Brooks, Frederick. The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition, Addison-Wesley Professional, New York, 1995.
6. http://iosifk.narod.ru/nat_m3.pdf
7. Jones, Capers. Private study conducted for IBM, 1977.
8. Barry Boehm. Software Engineering Economics, Prentice Hall, Upper Saddle River, NJ, 1981.
9. Каршенбойм И. Микроконтроллеры NEC для автомобильной электроники-2 // Компоненты и технологии. 2006. № 1.
10. Каршенбойм И. Микропроцессор своими руками-3 // Компоненты и технологии. 2006. № 3–4.
11. Татьяна Юрасова. Поставщик дворов их глобальных величеств. ttp://www.expert.ru/rus_business/2006/03/interveiw_vorohovskiy/
12. RTI: The Economic Impact of Inadequate Software Testing, Planning Report 02-3, May 2002. http://www.nist.gov/director/prog-ofc/report02-3.pdf
13. Glass R. Facts and Fallacies of Software Engineering, Addison-Wesley, New York, 2002.
14. Davis A. 201 Principles of Software Development, McGraw-Hill, New York, 1995.