Микропроцессор своими руками-5.
По поводу начала проекта встроенного в FPGA микроконтроллера

Иосиф КАРШЕНБОЙМ
iosifk@narod.ru

Почему была написана данная статья и в чем ее цель?

Ответ здесь очень простой. Когда-то автор писал о тех сюжетах, которые были ему более всего понятны и интересны. Но потом оказалось, что они интересны еще и довольно большому числу читателей. По их письмам и выбирались темы для продолжения статей. Поэтому появился цикл статей «Микроконтроллер своими руками». А теперь к читателям еще добавились слушатели. Они попросили меня рассказать «про встроенные в FPGA микроконтроллеры». Собрались вечером на кафедре ЦВТИ Университета телекоммуникации. Не могу сказать, что удалось полно ответить на все заданные в тот вечер вопросы. Но один из них показался мне довольно серьезным, а ответить на этот вопрос мне удалось только частично. Студент Алексей Литвинов спросил: «А как вы начинаете проект»?

И еще один тезис: в 2000 и 2002 годах, когда вышли первые мои статьи о встроенных в FPGA микроконтроллерах, сама тема была новой и малоизученной. Поэтому для читателей были интересны, в первую очередь, статьи, описывающие сам микроконтроллер как таковой. Сейчас, в 2007 году, ситуация значительно изменилась. Описания и исходные коды различных реализаций микроконтроллеров вполне доступны и большей частью достаточно хорошо изучены. Но вот «сопутствующие» проблемы остались, и они совершенно не представлены в публикациях. Поэтому изменилась и направленность статей. Теперь надо рассказать читателям не о каком-то конкретном образце микроконтроллера, более важно показать то, как сделать выбор между теми или иными реализациями микроконтроллеров. А там, где есть проблема выбора, уже десятком строк кода не «отделаешься», приходится долго и обстоятельно описывать все «за» и «против». Вот поэтому и в этой статье, большая часть которой посвящена проблемам выбора и оптимизации, так мало кодов и много аргументов «за» и «против». Конечно, о некоторых из проблем, затронутых в этой статье, уже приходилось бегло упоминать и раньше. Но именно бегло и кратко, поскольку основное внимание было уделено, так сказать, фактической стороне дела — описаниям самих процессоров. А в данной статье все проблемы, которые возникают «ДО ТОГО КАК» объединены вместе и рассмотрены подробно. Дело еще и в том, что проблемы разработки конкретного изделия по уже готовому и разработанному техническому заданию (ТЗ) решаются исполнителями или, скажем так, средним звеном. Таких инженеров достаточно много и, соответственно, много статей на эту тему. А вот первые шаги в проекте обычно выполняют руководящие работники: ведущие специалисты, топ-менеджеры проектов, директора фирм. Таких специалистов в десятки и сотни раз меньше, чем исполнителей, да и делиться своими знаниями и опытом они обычно не любят. Вот именно поэтому статей на тему о том, как начинать проект, очень и очень мало. При этом еще раз хочу подчеркнуть, что тема начала проектирования чрезвычайно важна, поскольку проблема выбора возникает на самых первых этапах проекта. А ведь известно, что чем раньше в проект закрадывается ошибка и чем позднее она исправляется, тем дороже и сложнее оказываются ее исправления и тем больший ущерб делу наносит эта ошибка.

Начало проекта — что это и как делать

Про «микроЭВМ», микроконтроллеры, «однокристалки» написано уже неимоверно много. Так о чем же будет данная статья, и чем она отличается от стандартного курса «по микроЭВМ»?

Есть множество учебников и академических изданий, посвященных этому курсу. То, что там написано, можно образно представить вот в таком стиле: «У нас есть вот это» и «вот как это работает». Это называется Анализ. Далее приводятся многочисленные подробные и не очень подробные описания различных микропроцессоров и микроконтроллеров, но редко когда эти описания связываются с тем, как именно эти самые микроконтроллеры работают в конкретном приложении. Да это и не удивительно. Не удивительно, ибо методики разработки, методики тестирования и вся прочая «кухня» — а это именно то, что называется «ноу-хау» и это «ноу-хау» бережется как зеница ока. «Мы дадим вам полное описание пользователя для этого микроконтроллера, но никогда не объясним, почему мы ввели в набор именно эти команды и почему мы выбрали именно эту архитектуру» — вот лозунг фирм, продающих микроконтроллеры и микроконтроллерные ядра.

Ну а то, что написано здесь, это почти то же самое — «про микроконтроллеры», но только совершенно «с другого конца». «Вот наша задача» — «а вот как ее решить…» А это называется Синтез. И так же как и в теории цепей, известно, что синтез намного труднее анализа, так и здесь: довольно тяжело исчерпывающе все описать и даже осветить все разделы полностью. Многое будет изложено кратко и, к сожалению, часть материала даже придется опустить. Основная трудность синтеза в том, что реализация задачи не всегда предполагает однозначное решение. Решений может быть несколько, а может и не быть ни одного. Такое тоже часто бывает. Поэтому здесь мы только рассмотрим методику постановки задачи и главное, на чем будет сосредоточено внимание читателя — это именно на методике синтеза проекта.

Так как же начать проект?

Начнем с разработки технического задания (ТЗ). Давайте обсудим, что такое разработка, что такое ТЗ и причем тут «поле дураков».

Что хочет заказчик? Что он может рассказать о своих потребностях? Недавно в «Электрониксе» проводилось обсуждение «плохих» заказчиков. Вот, мол, пришел человек, принес бумажку, в которой просит X каналов АЦП, Y каналов ЦАП, все равно какой микроконтроллер и «чтобы было все в шоколаде», как сейчас говорят… И разработчик, поместивший пост в телеконференцию, на-

чинает всячески клеймить этого заказчика за такое непонятное ему задание на разработку. Конечно, это не задание на разработку. И даже не эскиз задания, а скорее филькина грамота, но в этом листке показаны намерения человека стать заказчиком. И прежде чем ругать его, хочется пару слов сказать о задании на разработку, о самой разработке и совсем немного о бизнесе.

Для многих, даже можно смело сказать — для большинства тех, с кем приходилось общаться раньше, понятие «разработка» сводилось к микроконтроллеру, операционной системе, слоям в печатной плате, припою, микросхемам и программным инструментам. Но на самом деле сейчас это не совсем так. Как и для большинства инженеров «старой закалки», так и для автора этих строк было время, когда «работала» вот такая цепочка синонимов: электроника-разработка-схемы-микроконтроллеры-FPGA и т. д. И надо сказать, что для определенной части знакомых и бывших сослуживцев эти термины и до сих пор являются синонимами их деятельности. Но ведь жизнь не стоит на месте и задает все новые и новые уроки. Поэтому для автора этих строк теперь «заработала» вот такая цепочка синонимов: электроника-люди-бизнес-деньги и т. д. Под словом «деньги» здесь понимается не то, что «налом», а то, что служит инструментом для производства. А куда же делись «схемы-микроконтроллеры-FPGA»? Эти термины, конечно, остались, но из цели они превратились в средство. Нет, автор не превратился в бизнесмена, торгующего проектами вразвес или оптом. Просто теперь поменялись приоритеты. И далее автор попытается подробней объяснить этот тезис. Безусловно, и при «старом режиме» наши «отцы-командиры» объясняли нам: главное, это чтобы все чертежи и схемы были в архиве, и чтобы они были выполнены на кальках с соответствующими подписями. А для справки могу сказать, что комплект конструкторской документации (КД) только на одну стойку управления, в который входили все документы, начиная от технического задания, технических условий, схем и кончая инструкцией по упаковке изделия, так вот такой комплект КД представлял собой стопку документов высотой почти в человеческий рост. Но главное отличие от тех дней в том, что в то время можно было «согласовать», «откорректировать» и иногда перенести сроки, а теперь, в ряде случаев, это уже невозможно. Да и предприятия тогда не закрывались из-за невыполненных в срок проектов.

И еще одно замечание о том, как «тогда» и как «сейчас». Вот, мол, «тогда» любой технарь мог стать директором завода, а «сейчас» — только менеджеры да экономисты. Да, известно много примеров когда в директора заводов «выходили» из слесарей, не имея экономического образования. Но при этом не надо забывать, что заводами тогда еще руководили министерства, Госплан и, зачастую, ЦК. И денег добавляли «по личному распоряжению», и задания могли установить «сверх плана». Именно там экономисты планировали что, куда и сколько. Как они это делали, и насколько это было эффективно — сейчас об этом нет смысла рассуждать. Только можно сказать одно — ракеты в космос были запущены и атомные подводные лодки отправлялись в походы. А сегодня, когда предприятия уже никто не опекает «сверху», не имея экономического образования и опыта — просто нельзя браться за дело.

Итак, сейчас главная линия — это «люди-бизнес-деньги. Про деньги здесь рассуждать не будем, остановимся только на первых двух пунктах: люди и бизнес. С чего начинается любой бизнес? С того, что люди договариваются о совместной деятельности. Так вот, очень важно понять то, что любой договор — это документ о том, что люди вступают во взаимоотношения. В данном случае мы будем говорить о трудовых взаимоотношениях. Ну, а когда люди вступают в трудовые взаимоотношения, то они должны договориться о том, что надо сделать, в какие сроки и за какое вознаграждение. И, соответственно оформить конкретный договор об этой деятельности. Правила составления договоров мы здесь тоже обсуждать подробно не будем, это прерогатива юристов. Главное, это то, чтобы в договоре было четко обозначено:

1. То, что надо сделать.
2. То, как изделие будет испытываться на работоспособность.
3. Критерии, по которым изделие (партии изделий) будут считаться соответствующими п. 1.
4. Что делать в том случае, если все изделие или его часть не соответствует пп. 1–3.
5. Порядок оплаты за выполненную работу.
6. Ответственность сторон.

Возможно, что кроме этих пунктов, в договоре могут быть и другие. Но уж эти-то 6 пунктов должны быть обязательно. И все дело начинается с первых трех. Обычно они и называются техническим заданием. И именно с них и начинается разработка. Подчеркиваем, разработка — это прежде всего грамотно составленное ТЗ, которое должно быть выполнено в срок и за ту сумму денег, которая обозначена в договоре. По крайней мере, автора этих строк жизнь учила именно так. А микроконтроллер, операционная система и все остальное — это только средство для выполнения ТЗ и договора. Равно как и привлеченные для выполнения работ деньги. Наверное, должно быть всем известно, что если то устройство, что вы сделали, не выполняет хотя бы один из пунктов ТЗ, то работа считается невыполненной. Поэтому при составлении ТЗ необходимо учесть все факторы, которые могут встретиться при разработке. И тут должно быть такое правило — если сомневаешься, что сможешь выполнить этот пункт ТЗ, то лучше его не вносить. Такой пункт можно будет потом ввести в ТЗ как приложение, как дополнение или как новую версию ТЗ.

И что удивительно: разработке ТЗ до сих пор не учат в институтах. Изменились многие курсы, но такой науке, науке формулирования задания на разработку, похоже, не учат нигде. Дело же это оказалось нужным и востребованным. А востребована эта наука оказалась еще и потому, что старые «отцы-командиры» с их методами работы, большей частью, оказались не у дел. Провал 1990-х привел к тому, что связь поколений разработчиков местами прервалась. В появившихся же новых мелких фирмах руководителям пришлось самим добывать новый опыт. Хотя слово «новый» здесь применено только в том смысле, что этот опыт новый именно для них.

Итак, уважаемые читатели, несколько фраз для передышки, и за дело!

Вот что автор написал в одной из статей:

«Известный герой детских сказок любил сначала закапывать деньги, а потом долго не мог вспомнить, что он делал накануне вечером и куда делись монеты. Чтобы у нас с вами такого не произошло, попробуем сначала «нарисовать известное поле». На старом языке этот этап назывался разработкой технического задания».

Два способа разработки ТЗ

Как же собственно ведется разработка ТЗ? Как и любое другое изделие — либо расчетным методом, либо на основе экспертных оценок, либо комбинированным способом.

Расчетный способ проектирования первый раз «проходят» школьники, теперь уже, наверное, классе во втором. Бассейн, труба «А», труба «Б». Разделили, сложили, умножили, посмотрели в «Ответ». Ну, а если никто не знает, что там и как должно работать изделие? Если это первая посадка на Луну? Тогда эксперт говорит: «На основании моего опыта надо делать так-то и так-то». Сказал же Королев: «Луна твердая», и аппарат благополучно совершил посадку на поверхность Луны, а вовсе не утонул в толстом слое «лунной пыли», о котором писал Бредбери. Поскольку разработка техники, и в частности электронной техники, гораздо сложнее бассейна и двух труб, то обычно разработка ТЗ слабо поддается абсолютно точному прогнозированию и точному подсчету. Гораздо чаще используется метод экспертных оценок. Но, как вы понимаете, метод экспертных оценок довольно субъективен, поскольку он отражает взгляды и методы работы конкретного эксперта. Для больших проектов, чтобы получить более достоверную информацию, к делу привлекаются несколько экспертов, и производится оценка рекомендаций, выданных разными специалистами. Но в жизни часто получается так, что ТЗ формирует только один эксперт, и в его роли выступает только один человек — руководитель проекта. И тогда весь проект сильно зависит от него. И далее, как говорится, возможны варианты…

Теперь пара слов об этих вариантах или о том, как надо и как не надо составлять ТЗ. Здесь есть два способа. Один я называю «от гайки М3», второй «от алгоритма».

Первый способ составления ТЗ — «от гайки М3»

«У меня есть гайка М3, этой гайкой я могу прикрутить железную ручку к модулю. А модуль — это такой кусок пластмассы, на который «влезут» 10–20 микросхем. А эти 10–20 микросхем смогут … и еще при этом смогут … и… А дальше видно будет, что получится».

Второй способ — «от алгоритма»

«У меня есть алгоритм, описанный в ТЗ. Чтобы реализовать этот алгоритм, мне необходимо иметь возможность обработать XX данных за квант времени, для этого мне надо иметь YY памяти, ZZ разрядность шин и SS вычислительную мощность. Такую блок-схему вычислений я могу так-то и так-то разделить между soft-вычислителем и hard-вычислителем. Для этого мне нужны такие-то процессоры и такие-то ПЛИСы. И для их размещения будет достаточно такой-то PCB с таким-то количеством слоев… (Дальше пропускаем, потому что все должно закончиться словами… Надеюсь, вы догадываетесь какими: «…гайка М3»).

В чем же разница? Все равно: и там, и тут есть «гайка М3». А в том, что в первом варианте гайка точно будет привинчена «насмерть». Но будут ли выполняться все алгоритмы, о которых говорится в ТЗ? Как показывает практика, в первом варианте всегда что-то забывается и не учитывается. И обнаруживается это только потом, когда прибор уже сделан, и все переделки и исправления достаточно дороги или вообще невозможны.

Итак, вам надо составить первую проработку ТЗ, а вы еще никогда такого не делали. Что надо сделать в таком случае? Чтобы себя обезопасить от всяких мыслимых «проколов» (а обо всех немыслимых проколах вы узнаете, к сожалению, только потом…) попробуйте «нарисовать поле дураков». Возьмите лист бумаги и напишите в левый столбик — «функции, которые должны выполняться». Как когда-то меня учили программисты: «перестань нам рассказывать, какие у тебя там микросхемы, ты скажи, что они делают и в каком порядке». А в правой колонке — какие средства нужно задействовать, чтобы эти функции выполнялись. Итак, первично — это алгоритм, описанный в ТЗ. Вот в таком варианте все алгоритмы должны выполняться, потому что они полностью должны быть поддержаны аппаратными средствами. А что касается «гайки М3», то, возможно, в результате разработки конструктив может быть выбран другой, и, соответственно, гайка может иметь другие размеры, например дюймовые.

И еще надо добавить пару слов об обработке исключений.

Как-то так получилось, что автору приходилось встречать среди разработчиков ТЗ большей частью оптимистов. Видно, само их положение в служебной иерархии наложило на их характер свой отпечаток, поскольку это все были начальники, большие и не очень, но все же начальники. А все начальники хотят верить, что их указание обязательно и беспрекословно должно быть выполнено. Ну, просто выполнено и все… И ничего другого не может быть. Если кто не читал «Записки Инженера», где немного об этом написано, то вот их любимый тезис «про монету». «Подбрасываем монету в воздух, она падает и встает точно на ребро!!!» И никак иначе! Что значит: «…а если зависнет в воздухе…»? Такого быть НЕ МОЖЕТ! Только на ребро и все!!! На самом деле, как вы, читатель, и подозреваете, бывают еще и другие случаи. Вот именно их-то и надо отразить в ТЗ. Что должно происходить с разрабатываемым устройством, если «не происходит событие № 1, не отвечает абонент № 2 и т. д.». Необходимо предусмотреть не только возможности для отладки, передачи сообщений, но и для разборки в «кляузных ситуациях». Вот например, заказчик клянется, что никакое резервирование по питанию не нужно и проблем с этим не будет… Но, если ваша аппаратура не отличает сбой по питанию от собственных сбоев, то будьте уверены, что заказчик именно вашу аппаратуру объявит некачественной, но никогда по своей доброй воле не будет проводить работы по улучшению электропитания. И если ваша аппаратура не фиксирует и не документирует такие сбои или не позволяет на время пуско-наладочных работ подключить дополнительное контрольное оборудование, то доказать такому заказчику вашу правоту будет невозможно. И так далее, и тому подобное. Примеров таких может быть сколько угодно. Зачем приводятся такие пространные рассуждения? Просто дело в том, что для отработки исключений будет необходимо добавить ресурсов, которые по объему могут быть сопоставимы с ресурсами, требуемыми для основной задачи. Да и трудоемкость, необходимая для обработки исключений, может составлять значительную часть от всей трудоемкости проекта.

И снова о ТЗ. А бывает еще и вот так

Ах да! Это только кажется, уважаемый читатель, что уже перечислены все варианты создания ТЗ. Все же в реальной жизни есть еще один вариант, тот, про который не очень-то хочется вспоминать, но вот приходится. Это тот самый случай, который один из моих бывших сослуживцев выражал такой фразой: «Жарьте, ребята, масло подвезут!!!» Техническое задание, выполненное по такой методике, думаю, тоже многим известно. Известно, и чем заканчивается разработка по такому заданию.

А где посмотреть то, как правильно оформляются эти документы?

Поскольку этот момент довольно важен, то он выделен в отдельный пункт. Вот ссылка на русскоязычный ресурс: **http://authorit.ru/?c=8&b=3**.

На кого похож слон?

Есть такая старая басня, в которой говорится о том, что несколько слепых людей повстречали на дороге слона, а потом обменялись мнениями о том, как выглядит этот самый слон. Один настаивал на том, что слон похож на колонну, другой говорил, что на метлу, а третий — что на… Ну и так далее. Так вот, когда исполнитель первый раз встречается с новым заказчиком, то их разговор об объекте разработок точно напоминает разговор слепых о слоне. Дело в том, что каждый видит объект разработок только со своей точки зрения. Заказчик не всегда понимает, что и как должен делать электронный блок управления, а исполнитель не знает условий работы того физического объекта, которым надо управлять. И пока заказчик и разработчик не выработают общую точку зрения на вопрос и не найдут общий язык, до тех пор разработка будет обречена на провал.

Так как же научиться вот этому: «Мы с тобой одной крови, ты и я»? А надо работать с заказчиком! Надо попытаться узнать у заказчика или совместно с ним составить алгоритм работы объекта и сопоставить этому алгоритму алгоритм работы разрабатываемого электронного устройства. В качестве примера хочется привести зарисовку того, как это делалось при разработке систем управления (СУ) в самом начале 1980-х. Каждая СУ представляла собой специализированный многомашинный электронный комплекс, который оперировал с тысячами дискретных и с сотнями аналоговых параметров в реальном времени. Так вот специально для целей отладки ТЗ была построена модель СУ и модель объекта управления. Обе они работали на разных машинах. Машины были соединены друг с другом так, что изменения параметров одной модели вызывали реакцию в другой. Теперь представьте себе три команды специалистов: управленцы, технологи и электронщики. Столы завалены огромными листами бумаги, на которых изображены блок-схемы технологических процессов. На одной машине командуют управленцы: «Подаем команду номер такую-то». На другой машине — технологи: «А у нас по техпроцессу так не положено…». Ну и тут же замечание по алгоритму работы СУ — СУ пропустила команду, не выдала сигнала об ошибке на монитор и т. д. И пока наши программисты не научились разбираться в последовательности действий при различных технологических процессах, а управленцы не разобрались с ТЗ — дело не пошло. Ну а технологи? Они продолжали менять алгоритмы работы объекта до последнего дня. То одну ветвь процесса

поправят, то другую, то где-то поставят новый насос, то новый клапан… Потом правильность работы проверялась на реальной СУ и электронном имитаторе объекта. Такая отладка проводилась уже не на двух машинах, и хотя СУ представляла собой минимальный вариант самой простой системы, но все же это были уже сотни стоек и они занимали целый этаж в большом здании. И на этом этапе работ задание на разработку реальной СУ дополнялось и уточнялось. И только после таких многократных «подгонок по фигуре» алгоритмы были зафиксированы в ТЗ, и процессы отладки были перенесены на реальный объект. И если кто-то думает, что далее «как по маслу», так нет. Далее процесс доработки проходил еще около двух лет, до тех пор, пока не были учтены все… Нет! Надо написать так: пока не были учтены выявленные на тот момент недостатки.

А далее хочется привести только маленький пример того, что было, когда не учли один пункт ТЗ. Скорее, не то чтобы не учли, а не проработали с заказчиком. Причем рассказывается здесь только о том, что автор видел сам. Когда сделали первые СУ и установили их на объекте, то выяснилась такая «маленькая деталь». Системы были распределены по объекту и охватывали объект управления в радиусе до 12 км. Когда начали испытывать первую систему на объекте заказчика и дело дошло до приемо-сдаточных испытаний, то неожиданно выяснилось, что для того, чтобы система заработала, надо сделать «ВКЛ» Да-да, забыли о простейшей вещи — о том, кто будет включать питание. Причем это «ВКЛ» надо было сделать в сотне аппаратных. В суете и спешке об этом не подумали, не до того было. Да и заказчик об этом не сказал. А не сказал потому, что не было у заказчика опыта работы с такой техникой, которая делает все в автоматизированном режиме. Вот и пришлось срочно создавать сначала группу, а потом и целый сектор, который разработал специальную стойку, которая дистанционно делала «ВКЛ». Потом эти стойки, по штуке на аппаратную, надо было так же бодро смонтировать в эти самые сотни аппаратных, связать кабелями управления. Кабели надо было протащить по эстакадам или уложить в короба трасс по тем самым 12 км в диаметре. А кабельные вводы в сооружения — это герметезированные сальники, и их пришлось вскрывать. Но самое главное — пришлось в ТЗ вводить специальную главу о «ВКЛ», срочно делать комплект конструкторской документации (КД), проводить его через ПЗ и утверждать у заказчика. Конструкторская документация — это КД на стойку, кабели, кроссовые ящики, кроссовые шкафы, системы. Вот что значит: «немного не учли в ТЗ»…

Разделение проекта на части

Теперь вы уже имеете представление о том, что такое ТЗ, и какие алгоритмы работы вам нужно реализовать.

Теперь давайте посмотрим, есть ли у нас возможность для разбиения алгоритма на отдельные, слабо связанные между собой ветви. Еще раз подчеркиваем, что разбивать на части нужно алгоритм, и в зависимости от того, как удастся это выполнить, будут приняты дальнейшие решения относительно аппаратной реализации.

О том, как это делается и на основании каких критериев, подробно написано в статье [1]. Если удается разбить проект на несколько частей, то для каждой части выделяется свое ТЗ, где и указывается, что и как должна делать данная часть проекта и как она взаимодействует с остальными частями. При этом подразумевается, что выполнять частичное ТЗ будут соисполнители.

Замечание о том, как документировать проект

Есть довольно много литературных источников с рекомендациями о том, как надо выполнять проект. Кроме этого, есть такие документы, как ISO, где изложены современные принципы выполнения проекта. Эти темы достойны отдельных публикаций, поэтому далее их развивать мы не будем. Здесь же хочется остановиться на тех приемах работы, которые применял в своей работе автор. По стандарту ISO9001 каждому работнику положено иметь рабочую тетрадь, в которой отражается весь ход работ. Также очень удобно иметь где-то на сервере файл, в который будут вноситься замечания, выявленные по ходу выполнения работ. Своевременная фиксация замечаний говорит о том, что они будут учтены при внесении исправлений в проект.

Одновременно с этим должны быть начаты работы по формированию сопроводительных документов — описаний, протоколов согласований и прочего. Любое изменение в проекте должно одновременно вноситься во все документы. Ввели дополнительный регистр в микроконтроллере — производим правку сразу всех документов: описание микроконтроллера, описание ассемблера, листинг и т. д.

Если же проект ведется группой разработчиков, то необходимо применить дополнительные меры по синхронизации файлов, сделанными разными разработчиками.

Определите систему контроля версий проекта. Выработайте соглашение о названии версий проекта. Определите порядок внесения изменений, такой, чтобы было ясно, чем новая версия отличается от предыдущей.

ТЗ есть. Алгоритм есть. Но все ли мы о нем знаем?

Поскольку в данной статье описываются аспекты, связанные с началом проектирования, то необходимо включить сюда еще один пункт. Возможно, что кому-то покажется, что он несколько «выпадает» из общего повествования о soft-процессорах, но, тем не менее, он здесь необходим. До сих пор описывалось то, как получить задание и как выяснить у клиента подробности, необходимые для успешного завершения разработки. Теперь считается, что мы уже абсолютно все знаем и можем начать реализацию проекта. Однако прежде чем начинать проект, давайте посмотрим, можем ли мы как-то упростить проектирование и, следовательно, сократить сроки. Сейчас мы рассмотрим следующий пример. Работая в ЛОНИИСе автор получил задание сделать в FPGA многоканальный HDLC-контроллер. А если говорить точнее — 30-канальный. Дело было в середине 1990-х, из программных инструментов у автора в распоряжении был только MAX-Плюс какой-то ранней версии. Ни до МоделСима, ни до VHDL тогда у автора дело не дошло. Так вот, сам автомат HDLC-контроллера обрабатывал один канал примерно за 10 тактов. Далее можно было бы размножить такой автомат 30 раз. Но тогда бы проект «не влез» в существовавшие тогда кристаллы. Поэтому был применен другой способ. Автомат «микропроцессорным способом» менял контекст под каждый канал HDLC. То есть все данные из своих регистров он сбрасывал в сегмент памяти, соответствующий текущему каналу, а затем из сегмента памяти, соответствующего следующему каналу, производил загрузку регистров. При таком подходе нужно было иметь только одно быстродействующее ядро и 30 сегментов памяти. А симуляция обработки одного кадра имела 10×30 — всего 300 тактов на байт, передаваемый по 30 каналам. Для симуляции обработки приема хотя бы 20 байт необходимо было сделать симуляцию уже в 6000 тактов, потом же вручную разобраться с тем, что, как и где отрабатывается в модели. Конечно, сейчас при применении МоделСима, Verilog'a или VHDL дело значительно упростилось. Но, тогда, как вы помните, у автора таких инструментов не было. А уж симулировать такой HDLC-контроллер на «четверке» и вовсе было безумным занятием. Но все же выход был найден. Для того чтобы значительно упростить дело, была составлена программная модель контроллера на «обычном» Си. В этой модели и были обкатаны различные последовательности данных. Было проверено определение и формирование кадров, битстаффинг и все остальные «премудрости» интерфейса. Далее программные переменные, использованные в модели, для проекта в FPGA были заменены триггерами, счетчиками и мультиплексорами. Проект был сделан довольно быстро, без больших затрат времени на симуляцию.

Сегодня подобная технология называется SystemC, SystemVerilog и т. д. Примером может быть документация, находящаяся на сайтах [2–6].

Без сомнения, рано или поздно мы к такой технологии придем. Но для ее изучения

необходимы время и силы. А «обычный» Си доступен любому разработчику уже сегодня.

И, отвлекаясь от проблем, связанных только с FPGA, можно сделать следующие выводы. Сегодня в распоряжении разработчиков имеется достаточный набор программных инструментов типа State CAD, различных Визардов и других аналогичных инструментов, позволяющих упростить разработку проектов. Сегодня во многих случаях, прежде чем начинать проект на целевой платформе, можно сделать программно-аппаратный макет разрабатываемого устройства. Отладить программу на PC при помощи визуальных приложений, а уже потом перенести отлаженные Си-коды на целевую платформу. Такой способ разработки также поможет определиться с требованиями, предъявляемыми к аппаратному обеспечению, ресурсам и быстродействию.

Дальше надо перейти к выбору аппаратной платформы...

Давайте введем и определим термин «инженерозависимость». Им мы будем квалифицировать применяемые в проекте компоненты. Например, такие компоненты, как простой трансивер физического уровня или, например, операционный усилитель, слабо «зависят» или вовсе не «зависят» от участвующих в разработке инженеров. За исключением некоторых дополнительных функций и потребляемой мощности такие микросхемы жестко регламентированы требованиями и промышленными стандартами. И выбор таких компонентов определяется скорее экономическими причинами — стоимостью, условиями поставки и т. д. А микроконтроллеры можно квалифицировать как сильно «инженерозависимый» компонент. Наличие опыта работы с определенным типом микроконтроллера, библиотек из проверенных программных фрагментов кода, а также отладочных программных и аппаратных средств привязывают разработчиков к тем или иным видам микроконтроллеров. Переход же на другой тип микроконтроллера всегда связан с определенными трудностями.

Поэтому и в данном варианте при определении пути решения задачи необходимо принять следующее решение. Сначала необходимо рассмотреть применение того микроконтроллера, который разработчик знает и «любит». И только потом рассмотрим все остальные варианты. При этом также необходимо учитывать то, что на изучение нового типа микроконтроллера разработчику понадобится дополнительное время.

При выборе аппаратной платформы нам необходимо произвести выбор между «стандартными» микросхемами микроконтроллеров и FPGA. Назовем для определенности «стандартные» микропроцессоры, выполненные в виде микросхемы, термином «ASIC-микропроцессор».

Конечно, в FPGA может быть размещен аналог «стандартного» ASIC-микроконтроллера, но об этом мы поговорим чуть позже. Итак, еще раз — ASIC-микропроцессор или FPGA. А возможно «только ASIC-микропроцессор» или «связка — FPGA и ASIC-микропроцессор». в любом случае алгоритм разбивается так, чтобы под его решение целиком или частично подходил готовый «стандартный» ASIC- или FPGA-микроконтроллер.

Достоинства очевидны. в аппаратном плане — ASIC-микроконтроллер в большинстве случаев можно выбрать более быстродействующий и с меньшим потреблением, чем FPGA-микроконтроллер. ASIC-микроконтроллер может иметь больше ресурсов памяти, чем FPGA-микроконтроллер. Для стандартных микроконтроллеров есть много наработок в виде готовых библиотек, различных операционных систем, подпрограмм, описаний примеров применения и т. д. То же самое можно сказать и об инструментальных средствах разработки. Трансляторы, симуляторы, отладчики — все поставляется. И еще один параметр необходимо упомянуть — инженерный опыт.

Теперь недостатки. Рассмотрим «стандартный» ASIC-микроконтроллер. Обычно по сравнению со встроенными в FPGA микроконтроллерами стандартные ASIC-микроконтроллеры не всегда удовлетворяют разработчика по набору периферии, а зачастую и шина ASIC-микроконтроллера «во внешний мир» слишком медленная. Поэтому, если алгоритм задачи разбивается так, что можно подобрать для них «стандартный» микроконтроллер, но при этом выясняется, что для данного микроконтроллера связь с периферией будет далеко не оптимальной, то такой микроконтроллер выгоднее разместить в FPGA как встроенный. «Встроенные в FPGA» микропроцессоры и микроконтроллеры на их основе имеют главное преимущество перед обычными ASIC-микроконтроллерами средней производительности: они абсолютно синхронны со всем остальным проектом, расположенным в этой же микросхеме. Если устройство, которое вы проектируете, работает в реальном времени и с большими потоками данных, которые вы должны извлекать из периферии и отдавать в периферию, то задача синхронизации становится достаточно серьезной. Все быстрые «мелкие» «стандартные» ASIC-микроконтроллеры работают асинхронно (относительно периферии в FPGA) и не имеют аппаратного входа «Готовность», поэтому они могут синхронизироваться с периферией только программно, а для программной привязки их к синхронному проекту в FPGA нужно, во-первых, несколько команд процессора, что займет несколько тактов синхрочастоты как со стороны процессора, так и со стороны FPGA, и, во-вторых, это также требует ресурса микросхемы FPGA и, в-третьих, занимает довольно много места на плате. Быстрые «крупные» ASIC-процессоры имеют возможность аппаратной синхронизации по входу «Готовность», но дороги и занимают еще больше места на плате. Да и применение «крупного» ASIC-процессора для небольших задач нецелесообразно. А это значит, что при том же быстродействии ядра процессора в случае применения FPGA-микроконтроллера получится выигрыш по производительности в 2–3 раза.

Возможно, что при таком подходе — все решение в целом, а именно встроенный в FPGA микроконтроллер + периферия — будет давать преимущества благодаря своей компактности. Может быть, преимущества появятся из-за оптимизации связи с самодельной периферией, а, следовательно, из-за повышения производительности всего устройства.

В ряде случаев для подобных проектов будет целесообразно использовать FPGA со встроенными хард-процессорами. Или применить для реализации такого микроконтроллера софт-ядро, но с командами, соответствующими «стандартному» микроконтроллеру.

Если алгоритм явно не разбивается так, чтобы можно было подобрать для них «стандартный» микроконтроллер, то в этом случае нам и придется делать в FPGA управляющий автомат или микроконтроллер с «самодельной системой команд».

Что же мы выбрали? При выборе аппаратной платформы мы решили, что будем применять FPGA. Дошли до узла управления в FPGA. И вот теперь более конкретно надо решить следующий вопрос: А как построить узел управления? Простой статический автомат или микроконтроллер.

Взвесим ЗА и ПРОТИВ

В статье [7] были приведены все аспекты применения статического автомата в FPGA. Позволим себе коротко повторить основной тезис.

Чтобы сравнить микроконтроллер с конечным автоматом, необходимо сравнить трудоемкость следующих работ:

- Чтобы в новом проекте реализовать заданную последовательность действий можно либо каждый раз заново создавать конечный автомат, либо взять уже готовый микроконтроллер, адаптировать его к заданным условиям и, написав небольшую программу для выполнения последовательности команд, запустить. Причем написание программы для микроконтроллера намного проще написания и отладки конечного автомата на языках HDL.
- Чтобы исправить ошибку в конечном автомате, необходимо переработать весь проект, в котором описан автомат, а в варианте микроконтроллера можно только переписать программу. В случае применения автомата фиттер перекладывает проект каждый раз по-новому. А при применении микроконтроллера меняется только память программ, поэтому ядро процессора мо-

жет быть принудительно привязано к определенным ресурсам в кристалле, и таким образом могут быть исключены ошибки, зависящие от перекладки цепей внутри микросхемы.

- Конечный автомат должен иметь ограниченное количество состояний, так как это требует дополнительных логических ячеек, в то время как микроконтроллер по количеству состояний ограничен только объемом памяти программ, а это на несколько порядков больше.
- Конечный автомат при увеличении количества состояний становится все более и более медленно действующим, так как рост числа дополнительных логических ячеек приводит к увеличению времени прохождения сигнала. Каждое изменение автомата может привести к необходимости повторной верификации проекта.
- Команды, выполняемые микропроцессором, определены по времени выполнения и не зависят от программы, выполняемой на данном процессоре. Поэтому микропроцессор обычно выполняется с требуемым быстродействием, и оно не зависит от конкретного применения, изменений или доработок программы при отладке.
- Отладка автомата путем добавления состояний или изменения алгоритма работы должна производиться только путем полной или частичной перекомпиляции проекта с последующей перезагрузкой FPGA. В случае применения микроконтроллера целесообразно иметь режим загрузки памяти программ контроллера. При этом нет необходимости производить компиляцию проекта. Достаточно только выполнить действия по ассемблированию программы и ее загрузке.
- И последнее, но очень существенное примечание. Если к ядру процессора добавить узел внутрисхемной отладки, то пользователь может получить уровень отладки программы, принятый при отладке ASIC-процессоров. Это возможность прочитать-записать данные в регистр, сделать останов, изменить ход выполнения программы.

Автомат — просто разрабатывать, сложно программировать и отлаживать

Автомат невозможно применить в последующих разработках без значительных переделок. Микроконтроллер, соответственно, наоборот.

Граница здесь может быть определена довольно условно. Обычный тезис «любителей автоматов» звучит примерно так: «Автомат — это очень просто, и лучше всегда использовать его». Просто — да! Разрабатывать при помощи State CAD'a быстрее — да! Но вот только всегда ли он будет лучше?

В качестве примера рассмотрим применение статического автомата с точки зрения управления битовыми переменными. Зададим только один вопрос: «Сколько же ресурсов потребуется такому автомату при управлении хотя бы не сотней, а парой десятков дискретных параметров, если все эти параметры управляются по разным алгоритмам?» Ответ будет очевиден — реализовать такой автомат либо невозможно, либо нецелесообразно, так как в конце концов мы получим неотлаживаемого «монстра».

То же самое будет и в любом другом случае при необходимости реализовать автомат для управления меньшим числом входных и выходных переменных, но по более сложному алгоритму. Но, можно предположить, что при числе состояний автомата, превышающих 50, отладка такого автомата превращается в кошмар.

И вот тогда целесообразней перейти к применению микроконтроллера.

Долго колебались, но все же выбрали микроконтроллер (наверное, мы фанатики микроконтроллеров?)

Встроенный в FPGA микропроцессор, преимущества и недостатки разных реализаций. Готовый или самодельный?

Реализации процессоров, встроенных в FPGA, теперь многократно описаны и легко доступны как на сайтах производителей микросхем, так и на сайтах различных изданий или открытых проектов. Микропроцессор, имеющий «стандартный» набор команд, ориентирован на широкий круг задач пользователя. Примерами таких процессоров могут быть Nios [8], picoBlase [9], MCS-51 или AVR. Ядра этих микропроцессоров могут быть встроены в FPGA. Далее мы рассмотрим преимущества и недостатки различных реализаций узла управления. Какую из этих реализаций выбрать? Для этого прежде чем рассматривать различные реализации, надо дать себе ответ на главный вопрос: а что именно будет самым существенным критерием при выборе стратегии оптимизации проекта? Затраченное на проектирование время? Количество вложенных денег? Простота понимания сути проекта другими работниками фирмы, которым, возможно, придется продолжать или сопровождать проект? Ответ: оптимизация по производительности узла управления и затраченным ресурсам. Причем еще раз подчеркнем, что эти критерии взаимно противоречивы. И выбор зависит от каждого конкретного случая, так что единых рекомендаций здесь нет.

Преимущества «стандартного» микропроцессора, «встроенного в FPGA»

Описывая преимущества встроенных в FPGA «стандартных» микроконтроллеров, не будем повторять то, что частично описано выше. Хочется привести фрагмент статьи [8], написанной автором в 2002 г. Как мы видим за 5 прошедших лет эта тенденция сохранилась и развилась. И если в 2002 г ядра процессоров еще были экзотикой, то сейчас они стали одними из основных IP-core, которые предлагают как изготовители кристаллов, так и сторонние разработчики. За эти годы были потрачены огромные усилия на то, чтобы значительно упростить процесс проектирования, компиляции и отладки встроенных процессорных ядер. В программные инструменты ведущих производителей кристаллов встроены Визарды, позволяющие формировать ресурсы процессора, добавлять необходимую в данном проекте периферию. Так же в состав этих программных инструментов теперь входят и компиляторы, позволяющие обрабатывать коды программ этих процессоров. На многие из этих IP-core портированы операционные системы. Все это — несомненные преимущества «стандартных» микроконтроллеров.

Но ведь у каждой медали есть две стороны. Нельзя говорить только о достоинствах и не указать на недостатки. А они, к сожалению, есть.

«Ядра — чистый изумруд» и «Ядерные колхозы»

В конце 1990-х годов эпоха расцвета специализированных микросхем завершилась. Сегодня вряд ли можно себе представить начинку сотового телефона, состоящую из отдельных микросхем процессора, сигнального процессора, отдельного контроллера дисплея и т. д. Развитие технологии по значительному увеличению ресурсов микросхем приводит к кардинальным изменениям в проектировании устройств. Сегодня «горячая тема» — разработка программной модели ядер устройств. Описания ядер на языках группы VHDL позволяет компоновать проекты из готовых и отлаженных частей. Далее все части собираются в один проект для создания «системы на кристалле».

Фирмы-производители микросхем предлагают различные ядра, оптимизированные под их продукцию, для встраивания в проекты пользователя. Части проекта, которые могут быть повторно использованы, становятся товаром. Появился рынок по перепродаже проектов и их частей — **http://www.hellobrain.com/**. Но вместе с тем существуют и развиваются некоммерческие центры, такие как **http://www.fpgacpu.org/**.

Далее о «ядерных колхозах». Как только появились ядра процессоров как товарные продукты, так тут же в игру вступили фирмы, разработчики программного обеспечения. Это в первую очередь операционные системы реального времени, компиляторы языков высокого уровня, отладчики, симуляторы и т. д. Не остались в стороне и фирмы, разработчики вспомогательного оборудования. Здесь можно упомянуть о производителях как «стартовых наборов», так и змерительного и испытательного оборудования. Таким образом, весь комплекс средств для разработки, отладки и производства систем со встроенными микроконтроллерами становится товарным продуктом, что позволяет значительно ускорить темпы разработки и сократить трудоемкость разрабатываемого проекта. (*Цитата из статьи «Микроконтроллер для встроенного применения — NIOS. Конфигурация шины и периферии». «Компоненты и технологии». 2002. № 2–5*)

Недостатки «стандартного» микропроцессора, «встроенного в FPGA»

К ним можно отнести то, что в FPGA сложно организовать развитую систему команд или наоборот, есть определенные трудности, когда необходимо сократить число применяемых команд. Увеличение числа команд микропроцессора, равно как и увеличение способов доступа к данным различной разрядности, приводит к значительным затратам ресурса в FPGA. Например, одно из таких «узких мест» — это обработка битовых банных. Блоки памяти, находящиеся в FPGA, с которыми оперирует микропроцессор, обычно организованы как байтные или словные. Таким образом, выделение бита из байта или слова требует достаточно сложной обработки, а прямое управление битами при такой организации памяти невозможно. Особенно это относится к тем применениям, где нет необходимости реализовывать мощный микропроцессор. Более того, обычно ядро встроенного стандартного микропроцессора уже содержит всю периферию, которую «положено» иметь данному микропроцессору. Однако если часть этой периферии не нужна в данном проекте, она будет занимать ресурс кристалла, но не будет использоваться. В более развитых проектах пользователю предлагаются программные средства для настройки периферии, так что пользователь может выбрать только ему нужную периферию, и только она будет добавлена к проекту. То же самое относится и к различным узлам процессора. Например, в MCS-51 «положено» иметь битовое поле данных. И под это битовое поле данных закладываются команды и определенная часть ресурса. А пользователю, возможно, именно эта опция процессора в его конкретном проекте будет не нужна. Конечно, разработчик может выделить и убрать неиспользуемые периферийные блоки. Но это представляет собой дополнительную задачу, так как неизбежно встает вопрос верификации доработанного микропроцессора. А в том случае, когда мегафункция микропроцессора зашифрована или поставляется как файл списка связей, доработка микропроцессора под требования проекта может оказаться невозможной. И, кроме того, даже если пользователь уберет из проекта часть ресурсов микропроцессора, то при использовании стандартных для данного типа микропроцессоров ассемблеров и компиляторов у пользователя нет гарантии, что компилятор не будет пользоваться теми командами, которые будут обращаться к той части микроконтроллера, которая была удалена из проекта.

Какой микроконтроллер мы все-таки хотим?

Здесь надо описать то, как делаются нестандартные микропроцессоры и как под них подстраивается система команд и компиляторы. Итак, нестандартные микроконтроллеры. Почему мы их применяем?

Далее мы уже будем рассматривать только FPGA-микроконтроллеры и, как вы понимаете, только софт-процессоры. Поскольку хард-процессоры в FPGA можно тоже рассматривать как «стандартные». Как уже было сказано выше, нестандартный микроконтроллер — это такой микроконтроллер, у которого набор выполняемых им команд отличается от стандартных ASIC-микроконтроллеров. Для чего они нужны? Ответов может быть по крайней мере два. Первый ответ: хотим сделать учебный проект, а что дальше делать с таким контроллером — мы пока не знаем. Ответ второй: надо сделать прибор так, чтобы при очень скромных ресурсах кристалла получить максимум производительности. Так вот, наиболее частая причина для разработки нестандартного микроконтроллера в FPGA и состоит именно в том, что производительности микроконтроллера со стандартным набором команд не хватает или микроконтроллер со стандартным набором команд занимает слишком много ресурсов.

Теперь давайте сузим область рассмотрения микроконтроллеров

Не бывает микроконтроллера «на все случаи жизни», не бывает проектов без противоречий и ограничений. Что же за задачи выполняет микроконтроллер?

Как объяснить методику разработки микроконтроллеров? Для начала давайте уточним тезис о том, что не бывает контроллеров «на все случаи жизни». Либо контроллер «общего назначения», либо узкоспециализированный. Контроллер «общего назначения» — это, скорее всего, как раз и будет тот самый контроллер, который предлагает фирма-изготовитель. А здесь мы рассматриваем специализированные контроллеры. Давайте рассмотрим несколько задач. Первая задача — обработка датчиков в реальном времени, вторая — обработка данных, поступающих потоком.

Продолжение следует.

Литература

1. Квадрига Аполлона и микропроцессоры // Компоненты и технологии. 2006. № 4, 5.
2. **http://en.wikipedia.org/wiki/SystemC**
3. **http://www.mentor.com/products/c-based_design/index.cfm**
4. **http://www.celoxica.com/products/dk/default.asp**
5. **http://www.impulsec.com/**
6. **www.altera.com/c2h**
7. Микропроцессор своими руками. Часть 1 // Компоненты и технологии. 2002. № 6, 7.
8. Микроконтроллер для встроенного применения — NIOS. Конфигурация шины и периферии // Компоненты и технологии. 2002. № 2–5.
9. **http://www.xilinx.com/ipcenter/processor_central/picoblaze/picoblaze_user_resources.htm**
10. Микропроцессор своими руками-2. Битовый процессор // Компоненты и технологии. 2003. № 6, 7.
11. Tomaszewski E. Explicitly Parallel RISC (EPRISC), **http://www.opencores.org/articles.cgi/view/4**
12. **http://www.jwdt.com/~paysan/4stack.html**
13. Chapman K, Creating Embedded Microcontrollers (Programmable State Machines). Part 1, 2, 3, 03/28/2002, **www.xilinx.com**
14. An Overview of the ADSP-219x Pipeline. Engineer To Engineer Note. EE-123. **www.analog.com**
15. U17135EJ1V1UM00.pdf; V850E2 32-bit Microprocessor Core Architecture. **http://www.eu.necel.com**
16. ADuC7024_25_PrD.pdf, **www.analog.com**
17. **http://www.trash.net/~luethi/study/silverbird/silverbird.html**