Микропроцессор своими руками

Иосиф Каршенбойм

lk@lmail.loniis.spb.su

Введение

В этой статье приведен пример разработки микроконтроллера в FPGA. Статья может быть интересна как студентам и начинающим разработчикам, так и профессионалам, которые имеют опыт разработки устройств, и в том числе и в FPGA, но по тем или иным причинам не имели возможности разрабатывать и использовать «встроенные в FPGA» микропроцессоры. Дальше приведено два вступления — для студентов и для профессионалов. Хочется чтобы для каждой из этих групп читателей данная статья представляла определенный интерес. Но поскольку мотивация для прочтения этой статьи для двух групп разная, то далее следуют два вступления.

Все файлы, приведенные в данной статье, будут доступны в Интернете на сайте автора www.iosfk.narod.ru.

Введение для студентов

К написанию этой статьи меня подтолкнуло следующее: моя дочь, которая учится в довольно известном питерском вузе, сейчас «проходит» курс микропроцессоров. Но «проходит» — это именно то слово, которое здесь наиболее уместно. Ибо заучивание слов лектора и скромные «лабы» — так и мы учили когда-то курс ЭВМ. Но чтобы было понятно, о чем идет речь, придется сделать небольшой экскурс в недавнюю, для меня, историю.

Мой опыт работы с ЭВМ начинается с кошмарных воспоминаний о машине, по-моему это был «Проминь», которую установили на нашей кафедре в ЛЭТИ, где я, тогда еще студент четвертого курса, работал на полставки лаборантом. Итак, год, наверное, 1975-й. И курсовики и дипломы с применением расчетов на ЭВМ еще только входили в моду. И вот курсовик «Расчет трансформатора в баке с маслом» — название именно потому и запомнилось, что это была моя первая попытка автоматизировать инженерный труд при помощи ЭВМ, как сказали бы теперь. Мне, как сотруднику кафедры, разрешили провести часть расчетов на ЭВМ. Был написан алгоритм, составлена программа и вот уже получено время для работы на машине. Дальше — беда. Оказывается что в машине, где команды задавались специальными кодирующими планками, которые нужно было втыкать в специальное наборное поле, не хватает команд умножения для реализации моего алгоритма. Даже после того, как я сбегал на соседнюю кафедру и одолжил там планки для умножения от такого же «Проминя», выяснилось, что не хватает двух команд. Конечно, курсовик был рассчитан на логарифмической линейке. Вернемся в сегодняшний день. Многие из сегодняшних студентов просто не представляют, насколько стремительно меняется техника. Но в темпе изменения техники должны меняться и приоритеты для инженерного труда, и методики обучения. Так, как учились вчера, уже нельзя учиться сегодня. Сегодня молодой инженер, не умеющий пользоваться ORCAD'ом, PICAD'ом, симуляторами электрических цепей типа Electronics Workbench, не способный описать схему на VHDL и написать сотню строчек кода на C++, достоин только сожаления. Если вчера было модно программировать видеоигры, базы данных и бухгалтерские программы, то сегодня рынок уже поменялся. Сегодня открылся новый огромный рынок — программирование FPGA. Если 2–3 года тому назад программирование FPGA понималось как разработка периферии к стандартным микропроцессорам, то теперь ситуация коренным образом изменилась. Теперь наступило время «систем на кристалле». Более подробно об этом будет написано во введении для профессионалов. Программирование FPGA в скором времени плавно перейдет в разработку заказных микросхем — «систем на кристалле». И первые российские успехи в этом направлении уже есть (см. Л1).

Итак, заканчивая это вступление, нужно сказать следующее. Если вас не учат этому в вашем ВУЗе — требуйте, требуйте также настойчиво, как требуют тушенку в рекламе. Вы имеете на это право. Если вы не научитесь этому сейчас, то дальше догонять других будет очень тяжело. Но, если научиться современным технологиям в вузе нет возможности, то вспомните, что сказал Скруджу МакДаку его папа: «Скрудж, работай головой, а не руками». Большая часть из программ, выпускаемых крупными и известными фирмами — производителями ПО, имеют студенческую версию, т. е. они *совершенно бесплатны*. Они доступны на сайтах самих фирм и их дилеров. Что касается Altera, то фирма рассылает CD, на котором есть и студенческая версия ПО, статьи и описания на микросхемы. Для того, чтобы данная статья послужила в полную меру, возьмите книгу Л2. К книге также приложен CD с необходимым ПО и примерами по обучению ПО MAXPLUS.

Введение для профессионалов

Время, скажем так, стандартных микросхем заканчивается. Начинается новое время — время «систем на кристалле». Разработка самих систем, частей таких систем — библиотечных мегафункций или ядер, становится не только заманчивой перспективой, но и реальностью. Причем прибыльной реальностью. Так, по данным из Интернета, фирма, продающая лицензию на свое ядро процессора для телекоммуникаций, получает за каждую продажу лицензии 150 тыс. долл. единовременно, и по 7 долл. с каждой выпущенной по их лицензии микросхемы за первый год выпуска и по 2 долл. с каждой микросхемы за последующие годы. Представьте, что это процессор для сотового телефона или видеокарты, и посчитайте.

За последние полгода технологический рывок изготовителей микросхем привел к тому, что появились микросхемы, способные работать с ядром процессора на частоте до 300 МГц. Ресурс, который можно занять для проекта, так-же вырос. Появились аппаратные блоки, например, процессор ARM-7 в микросхемах серии «Экскалибур» или умножители в серии «Стратикс». А это значит, что микропроцессор, реализованный в FPGA, способен составить конкуренцию DSP средней производительности. А при выполнении блоков, выполняющих умножение и деление по заданному пользователем алгоритму, микропроцессор, реализованный в FPGA и ориентированный на конкретную задачу пользователя, будет иметь производительность не ниже, а, может быть, и выше, чем самые быстрые DSP.

Рост популярности того или иного направления можно оценить по количеству сайтов в Интернете, на которых отражаются интересы разработчиков.

Фирмы-производители микросхем предлагают различные ядра, оптимизированные под их продукцию, для встраивания в проекты пользователя, например **http://www.altera.com/ipmegastore/index.html**.

Фирма Triscend предлагает 40-MHz 8051-ядро на кристалле, с FPGA, доступной пользователю. Аналогичный продукт есть и у фирмы Atmel — FPSLIC.

Более подробное описание по предлагаемым продуктам приведено в Л3, 4.

Фирма Altera предлагает микроконтроллеры на ядре an ARM9T (**http://www.arm.com/**) или на ядре MIPS32 4K (**http://www.mips.com/**).

Часть проекта фирмы Altera ™ по встроенным решениям, процессор Excalibur, а также его вариант реализации — программное ядро встроенного процессора Nios ™.

Nios — это встраиваемый процессор общего назначения, с перестраиваемой конфигурацией, который легко вписывается в устройство Altera APEX ™, оставляя большинство логики, доступной для размещения там периферийных устройств и пользовательских функций. Встраиваемое ядро процессора Nios — конвейерный RISC-процессор, в котором команды выполняются за один цикл частоты синхронизации. Более подробно об этом процессоре можно прочитать в Л5 — 13.

Кроме широко известных сайтов фирм-производителей, появились и поддерживаются и наши, российские сайты, такие, как **www.asicdesign.ru**, белорусский **http://nit.gsu.unibel.by/IEESD-2000**.

Проекты и их части перепродаются как интеллектуальная собственность, то есть они становятся товаром. Сайт по перепродаже проектов — **http://www.hellobrain.com/**. Но вместе с тем существуют и развиваются некоммерческие центры, такие, как сайты предлагающие открытые проекты, — и **www.opencores.org**. Есть также и сайты, поддерживающие разработку именно процессоров в FPGA **http://www.fpgacpu.org/**.

Рост интереса к разработке именно процессоров отражением в росте публикаций открытых проектов на сайте **www.opencores.org**. Сейчас там предлагается 35 проектов микропроцессоров от крошечного tiny8 до 32-битного Yellow Star.

Что же касается российских разработчиков и разработчиков стран СНГ, то здесь несколько иная ситуация. Фирмы-производители микросхем имеют программу по поддержке партнеров-разработчиков ядер (мегафункций). Это и реклама, и сервис, и многое другое. Но для российских разработчиков попасть в партнеры пока еще практически невозможно, и это резко ограничивает их возможности.

Преимущества микропроцессора, «встроенного в FPGA»

«Встроенные в FPGA» микропроцессоры и микроконтроллеры на их основе имеют главное преимущество перед обычными 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» микроконтроллеры работают асинхронно (относительно периферии в Altera), и не имеют аппаратного входа «Готовность», поэтому они могут синхронизироваться с периферией только программно, а для программной привязки их к синхронному проекту в Altera нужно, во-первых, несколько команд процессора, что займет несколько тактов синхрочастоты, во-вторых, это также требует ресурса микросхемы FPGA и, в-третьих, занимает довольно много места на плате. Быстрые «крупные» процессоры имеют возможность аппаратной синхронизации по входу «Готовность», но дороги и занимают еще больше места на плате. Да и применение «крупного» процессора для небольших задач нецелесообразно. А это значит, что при том же быстродействии ядра процессора получится выигрыш по производительности в 2–3 раза.

Следующее преимущество — специализированные команды пользователя. Это значит, что проектируя микроконтроллер, пользователь может произвести предварительное программирование и определить в потоке команд, выполняемых процессором, группы наиболее часто повторяющихся команд. Если теперь группу таких команд объединить в одну специализированную команду, то быстродействие процессора для данного класса задач увеличится, а программировать его станет легче. Специализированные команды пользователя (см. например, описание команд процессора NIOS), могут быть однотактные или мгоготактные. Они могут выполняться в ALU микропроцессора или в дополнительном вычислительном блоке, подключаемом к ALU, например FFT, FIR и т. д.

Еще одно преимущество — микроконтроллер становится «невидимым». То есть микроконтроллер конечно есть, просто увидеть его уже нельзя. Это не шкаф, не каркас, не набор плат и даже не корпус микросхемы. Это теперь просто файл, который входит в другой файл. Но что удивительно, свои функции он выполняет не хуже, а часто лучше, чем его «старший брат».

И последнее, что необходимо отметить — микроконтроллер получает ту периферию и в таком количестве, как нужно пользователю. Периферия же может быть самой экзотической: от простого UART'а и до контроллеров Ethernet MAC 10/100 или сопроцессоров DSP.

Среди библиотечных элементов, описывающих периферию для микропроцессора, доступны следующие:

- универсальный Асинхронный Приемопередатчик (UART),
- таймер,
- параллельный ввод — вывод (PIO),
- интерфейс SRAM,
- SDRAM-контроллер,
- интерфейс FLASH памяти,
- последовательный периферийный интерфейс (SPI),
- контроллер I2C,
- модулятор ширины импульса (PWM),
- IDE-контроллер диска,
- контроллер Локальной сети 10/100 Ethernet (MAC),
- контроллер USB.

Конечно, этот список далеко не полный, но он дает представление о том, какой уровень разработок библиотечных элементов достигнут.

Подключив требуемые библиотечные элементы, можно сформировать необходимый для конкретного применения микроконтроллер.

У нас же есть конечный автомат, зачем нам что-то еще?

Часто можно услышать такие рассуждения: «Для обработки чего-то сложного 16-битного или 32-битного, конечно, применим процессор. Но вот для чего-то мелкого зачем нам эти программы, ассемблеры и т. д. У нас же есть конечный автомат, ну и еще пригоршня триггеров. Обойдемся и этим».

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 мик-

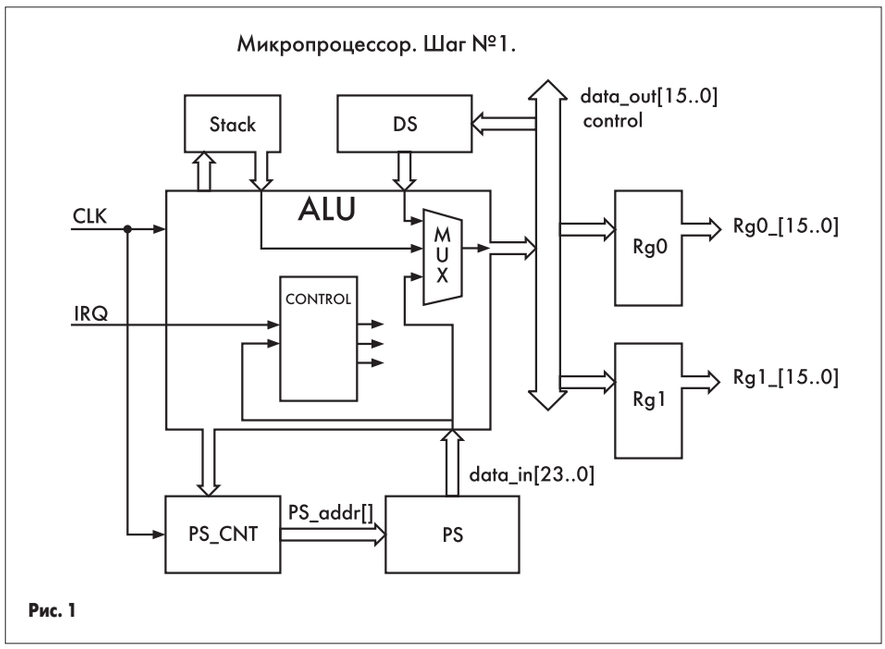

Рис. 1

роконтроллер, адаптировать его к заданным условиям, и, написав небольшую программу, запустить. Причем написание программы для микроконтроллера намного проще написания конечного автомата на языках AHDL, VHDL и так далее.

- Чтобы изменить алгоритм работы конечного автомата, необходимо его полностью переписывать, что требует много времени и сил, в микроконтроллере достаточно изменить микропрограмму.
- Чтобы исправить ошибку в конечном автомате, необходимо переработать весь проект, в котором описан автомат, а в варианте микроконтроллера можно только переписать программу.
- 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.
- И последнее, но очень существенное добавление. Конечный автомат при увеличении количества состояний становится все более и более медленнодействующим, 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 это быстродействие не зависит от конкретного применения, от изменений или доработок программы при отладке.

Если интерес к этой теме еще не пропал, то не стоит дожидаться, пока статический автомат в ваших устройствах разрастется в жуткого «монстра», при доработках и отладках, и от него придется отказаться. Тогда все остальное в этой статье тоже должно быть интересно и для вас.

Здесь конец лирике и начало проекта

Материал, приведенный в данной статье, написан на основе реальной разработки, описанной в Л14. Упрощенная модель микропроцессора, описанная в данной статье, служит только примером для разработки или изучения. Но тем не менее, она может быть легко доработана для практического использования. Методика разработки позволяет оценить трудоемкость, определить ресурс, необходимый для реализации микроконтроллера в FPGA.

Весь процесс разработки будет состоять из следующих этапов:

1. Разработка задания на проектирование.
2. Разработка блок-схемы микропроцессора.
3. Разработка полей кодов операций.
4. Разработка кодов команд.
5. Описание на AHDL блоков, входящих в микропроцессор.
6. Описание микропроцессора на AHDL.
7. Написание микропрограммы.
8. Симуляция микропроцессора с микропрограммой.
9. Выводы.

Что мы хотим получить?

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

Итак, мы хотим спроектировать:

1. RISC-процессор, потому что не хочется делать разборку многословных команд.
   Пусть все наши команды выполняются за один такт синхрочастоты.
2. Применим для процессора Гарвардскую структуру. Будем считать, что загрузка памяти команд нам не нужна и все команды будут храниться в памяти команд, что и происходит при инициализации микросхемы. Назовем память команд — Program Space (PS). Ограничим область адресов PS 16 битами. Далее возможна страничная адресация памяти, но для большинства случаев и этого уже будет достаточно. Для конкретной реализации микропроцессора введем параметр, описывающий разрядность шины адресов PS.
3. Расположим память данных в отдельной области памяти. Назовем ее Data Space (DS). Ограничим область адресов DS также 16-ю битами. Для конкретной реализации микропроцессора введем параметр, описывающий разрядность шины адресов DS.
4. Представим область ввода-вывода в виде набора регистров, дадим им номера 0..15.
5. Для реализации режима реального времени потребуется вход запроса прерывания — IRQ.
6. Чтобы отрабатывать прерывания и вызовы подпрограмм, нам будет необходим стек.
7. Необходимо задать разрядность шины данных. Выберем для определенности разрядность, равную 16 битам.
8. Будем считать, что системный сигнал СБРОС, действующий внутри кристалла, будет применяться и для нашего микропроцессора.

Построение ядра микропроцессора

Для упрощения положим, что:

А) аккумулятор/регистр общего назначения будет один,

Б) область DS будет состоять из одного регистра,

В) область ввода-вывода тоже будет состоять из одного регистра,

Г) применим самую простую структуру выборки данных из памяти, то есть асинхронную выборку и без конвейера команд,

Д) будем считать, что вход запроса прерывания выставляется синхронно с тактовой частотой и длительность входа запроса — 1 такт синхрочастоты.

Данные упрощения сделаны только для сокращения текстов описаний, так как в рамках журнальной статьи приводить обширные описания довольно затруднительно. Тем не менее, счетчик адресов DS полностью аналогичен счетчику адресов PS. Увеличение количества регистров связано только с увеличением описания в области дешифратора управляющих сигналов, которые вырабатываются при операциях записи в эти регистры и с мультиплексором выходных шин от регистров на внутреннюю шину микропроцессора. Вход запросов прерывания обычно обслуживается контроллером прерывания, который фиксирует запрос по фронту или по уровню и выставляет вектор прерывания. Контроллер прерывания может быть рассмотрен отдельно и не является необходимым звеном для данного проекта. Таким образом, приведенные сокращения не должны оказывать существенного влияния на излагаемый материал.

На основании задания и сделанных упрощений на микропроцессор получим архитектуру, показанную на рис. 1.

Микропроцессор состоит из набора следующих блоков:

Узел контроля для выработки управляющих воздействий на все блоки микропроцессора — ALU,

Счетчика адресов памяти программ — PS_CNT,

Блока памяти программ — PS,

Блока памяти данных — DS,

Стека — Stack,

Регистров — Rg0 и Rg1.

На входы микропроцессора подадим сигнал синхрочастоты — CLK и сигнал запроса прерывания — IRQ.

Системный сигнал СБРОС подается на микропроцессор, так же как и на все остальные блоки системы (здесь он не показан).

Выходами микропроцессора будут сигналы с выходов регистров Rg0 и Rg1.

Счетчик команд, при загрузке микросхемы или при инициализации системы по сигналу СБРОС, устанавливается в состояние 0 и далее производит счет адресов памяти программ. Адреса поступают на вход блока PS. Данные, хранящиеся в памяти программ, выбираются в соответствии с поступившим адресом. Далее данные с выхода PS поступают в ALU. Для ALU — это будут коды команд микропроцессора. Блоки DS, Stack, Rg0 и Rg1 обмениваются сигналами с блоком ALU по одной внутренней шине данных.

Сигнал со входа IRQ поступает на ALU и дешифрируется в ALU как код команды, причем сигнал IRQ имеет приоритет в выполнении над кодом команды, поступившим в ALU от PS.

Выберем команды, отвечающие нашим задачам

Опишем группы команд, которые должен исполнять микропроцессор.

1. Служебные команды:

NOP	Нет операции

2. Группа команд загрузки:

LDI Reg, < C >	Загрузка в регистр-приемник константы – (данных из памяти команд по текущему адресу)

3. Группа команд пересылки:

MOV Reg, Reg	Запись содержимого одного регистра в другой
MOV Reg, [Mem]	Запись содержимого регистра в память
MOV [Mem], Reg	Запись из памяти в регистр

4. Команды ветвления:

JMP Addr	Переход по абсолютному адресу
CALL Addr	Вызов подпрограммы (с записью адреса возврата в стек)
RET	Возврат из подпрограммы (по содержимому стека)

Для изучения принципов работы микропроцессора, такого набора команд будет достаточно. На практике набор команд можно дополнять и расширять до необходимого для выполнения конкретного набора задач пользователя. Арифметические операции убраны из списка команд, так как выполнение математических операций в FPGA достаточно описано, и в рамках данной статьи они не рассматриваются.

Определим поля команд

Чтобы определить поля команд необходимо определить наибольшее требуемое поле для выполнения заданных команд. Очевидно, что для данного набора команд самое большое поле требуется для команды MOV Reg, [Mem] и MOV [Mem], Reg. Поле [Mem] — имеет разрядность 16 бит, поле Reg — 4 бита (для 16 регистров). Поэтому выберем разрядность памяти PS — 24 бита. Тогда команда непосредственной записи из памяти в регистр будет выглядеть так:

23....Коп – 4 бита...20	19.....Reg – 4 бита ...16	15 ...Mem – 16 бит.. 0

Команда MOV Reg, Reg пересылки из регистра в регистр будет выглядеть так:

23....Коп – 4 бита...20	19.....Reg – 4 бита ...16	3 ... Reg – 4 бита.. 0

Команда LDI Reg < C > — непосредственная загрузка в регистр-приемник данных из памяти команд по текущему адресу:

23....Коп – 4 бита...20	19.....Reg – 4 бита ...16	15 ...Const – 16 бит.. 0

Команда JMP Addr — команда безусловного перехода по абсолютному адресу будет выглядеть так:

23....Коп – 4 бита...20		15 ...Mem – 16 бит.. 0

Команда CALL Addr — команда безусловного вызова подпрограммы по абсолютному адресу будет выглядеть так:

23....Коп – 4 бита...20		15 ...Mem – 16 бит.. 0

Команда RET команда безусловного возврата из подпрограммы или из прерывания будет выглядеть так:

23....Коп – 4 бита...20		

Команда NOP — «Нет операции» будет выглядеть так:

23....Коп – 4 бита...20		

Итак, нам нужно получить восемь команд для микропроцессора, и, как мы видим, поле кода операции разрядностью в 4 бита позволяет иметь 16 команд. Необходимо отметить, что для более мощных процессоров выбор полей кодов операций — очень серьезная задача, которая и определяет, насколько эффективным будет процессор для конкретного набора команд.

Определим коды операций команд

После того как поля команд определены, и мы знаем разрядность поля кодов операций, можно определить коды операций команд.

Наиболее просто определить код команды NOP — здесь нет никаких требований, кроме одного — этот код команды должен быть на входе ALU при получении им сигнала «СБРОС». Так как дальнейшее развитие проекта неминуемо приведет к конвейеризации и прочим профессиональным «хитростям», то рекомендуем выбрать код команды NOP = 0.

Чтобы несколько упростить визуальное восприятие команд, применим следующую кодировку: пусть команды переходов будут расположены в зоне кодов от 8 до H "F ".

Тогда шестнадцатиричные коды операций команд будут такие:

0 — NOP

1 — JMP

2 — CALL

3 — RET

8 — MOV Reg, Reg

9 — MOV Reg, [Mem]

A — MOV [Mem], Reg

B — LDI Reg < C >

Примеры кодов команд

Команда NOP будет иметь код 000000, а JPM 1234 — команда безусловного перехода по адресу 1234 — будет иметь код команды — 101234, MOV 9, 5 — команда пересылки данных из регистра 5 в регистр 9 будет иметь вид 890005 и т. д.

Определим требования к стеку

Для данной задачи применим стек глубиной 8 вложений и разрядностью равной разрядности шине адресов PS. Реализуем стек в отдельной от PS и DS области — на массиве регистров.

Определим требования к прерываниям

Применим следующее решение: при поступлении запроса прерывания будем выполнять переход по фиксированному адресу с занесением адреса возврата в стек.

Определим требования к программному обеспечению

Одним из основных требований к разработке встроенного ПО для микроконтроллеров является требование о необходимости инструментального ПО, то есть нам будет, как минимум, необходим редактор и ассемблер. Для более сложных проектов могут потребоваться симуляторы работы программы, языки высокого уровня, операционные системы и т. д.

Что касается редактора, то существуют такие редакторы, как EditPlus2 (**http://www.editplus.com**), Prisma, и другие редакторы текста, позволяющие выделять ключевые слова по списку, формируемому пользователем.

Для удобства написания программ можно также создать собственный ассемблер, что сейчас не является трудной задачей даже для среднего программиста. Такой ассемблер ставит в соответствие удобную для восприятия мнемонику с реальными командами микропроцессора.

Однако, и довольно часто, разработчики применяют систему команд тех микропроцессоров к которым они привыкли и к которым уже существуют инструментальные средства разработки ПО. Здесь на первом месте микропроцессоры с системой команд MCS-51, а на втором месте PICmicro. Для этих микропроцессоров имеются как средства разработки, так и огромные библиотеки разработанных программ.

Поскольку проект, описываемый здесь, посвящен только описанию микропроцессора, то дальнейшее описание требований к инструментальному ПО мы опускаем.

Будем считать, что у нас есть ассемблер, который преобразует мнемонические коды команд в машинные коды и формирует файл инициализации памяти команд, совместимый с ПО MaxPlus.

Заключение по этапу разработки задания

Мы произвели разработку задания на микропроцессор, выполняющий основные функции, такие, как пересылки данных, ветвление программы, обработка прерываний.

Продолжение следует.