Разработка проекта микроконтроллера 8051s на основе IP-ядер корпорации Microsemi. Часть 1

В статье показан пример полного маршрута проектирования устройства на ПЛИС корпорации Microsemi с использованием бесплатного IP-ядра микроконтроллера 8051s. Это ядро совместимо с классическим семейством микроконтроллеров 8051 и предоставляет разработчику сочетание достоинств этого семейства с высоким быстродействием и возможностью подключения современной периферии через шину APB стандарта AMBA. Высокая степень автоматизации и дружественный интерфейс САПР Libero корпорации Microsemi позволяют очень быстро создавать проекты для новых устройств. Пройти все этапы маршрута проектирования получается гораздо быстрее, чем прочитать эту статью.

Дмитрий ИОФФЕ
dsioffe@yandex.ru
Андрей МАКСИМОВ
maksimov@actel.ru

Введение

Микроконтроллер Intel 8051 появился в 1980 г. [1], и через два года мы будем отмечать его тридцатипятилетие. Архитектура полупроводникового изделия, созданная в прошлом веке, и даже не в самом его конце, оказалась настолько удачной, что до сих пор появляются основанные на ней новые разработки, например [2]. Остаются неизменными ее основные преимущества: хорошая продуманность, интуитивно понятная система команд. И с каждым годом растет объем наработок по ее применению в самых разных областях. Это означает, что в распоряжении разработчиков появляется все больше отлаженного и проверенного в реальных условиях программного кода.

Основная специализация корпорации Microsemi [3] — компоненты для систем повышенной надежности, работающих в условиях агрессивной окружающей среды. Отлаженное программное обеспечение — важнейший фактор надежности системы. Поэтому нет ничего удивительного в том, что Microsemi предлагает для использования в своих ПЛИС и системах на кристалле IP-ядра микроконтроллеров, совместимых с 8051.

Отметим также, что небольшое, по современным понятиям, IP-ядро 8051 занимает не слишком много ресурсов ПЛИС. Это очень важно при работе с дорогостоящими радиационно-стойкими ПЛИС, которые имеют не самый большой объем.

Корпорация Microsemi предлагает два IP-ядра, совместимых с 8051. Одно из них так и называется: 8051 [4]. Оно полностью повторяет оригинальный 8051, в том числе и его периферию, и с этим ядром без всяких доработок можно использовать готовый откомпилированный код для микроконтроллера 8051. Время выполнения машинного цикла у этого ядра составляет 12 тактов. Ядро 8051 платное, на сайте фирмы есть его оценочная версия. Его можно использовать в тех случаях, когда нежелательно или невозможно вносить изменения в существующее ПО.

В этой статье мы будем рассматривать другое IP-ядро корпорации Microsemi — 8051s [5]. Оно совместимо с классическим 8051 на уровне ассемблера. Его основные преимущества:

- инструкции выполняются за один такт;
- возможность создавать оптимальный микроконтроллер с необходимым разработчику набором периферии, подключаемым через шину APB, которая является одним из стандартов для современных систем на кристалле;
- возможность интеграции микроконтроллера в ПЛИС без использования блочной СОЗУ, для применения в радиационностойких устройствах.

Ядро 8051s бесплатное и входит в состав САПР Libero корпорации Microsemi. Кроме него, в САПР имеется большой набор готовых бесплатных IP-ядер периферийных устройств самого разного назначения, предназначенных для подключения к шине APB.

Архитектуру 8051 в этой статье описывать нет смысла, так как по ней существует огромное количество литературы. Одна из лучших, по мнению автора, книг на эту тему «Микроконтроллеры серии 8051: практический подход» [6] издана совсем недавно, в 2008 г. Это еще одно свидетельство бесспорной актуальности архитектуры 8051 для наших дней: про безнадежно устаревшую вещь хорошей книги писать никто не станет.

В первой части статьи будет описано построение аппаратной части системы с использованием ядра 8051s, а во второй — создание программы для микроконтроллера. Деление нашего будущего проекта на аппаратную и программную части, конечно же, несколько условно, потому что и то, и другое — «прошивка», которую разработчик может при необходимости изменять, не модифицируя электрическую схему печатной платы. Бесспорно электрическую схему платы с ПЛИС и обрамляющими ее компонентами мы будем считать уже существующей. Аппаратной частью будем называть микроконтроллерное ядро с набором периферийных узлов и памятью, а программной частью — собственно программу для микроконтроллера. Такое разделение проекта имеет практический смысл: аппаратная часть проектируется в среде Libero как схема из модулей, созданных в интерактивном средстве проектирования или написанных на языке описания аппаратуры, и затем проходит через все этапы, характерные для ПЛИС: временное моделирование, синтез, компиляцию и трассировку. Программа же пишется на традиционном языке программирования, например на C и/или на ассемблере, и транслируется обычным компилятором. При этом программу можно переписывать и отлаживать отдельно, не затрагивая «аппаратную» часть проекта. Это очень похоже на создание

новой программы для обычной системы с микроконтроллером в виде микросхемы.

Автор старался писать статью так, чтобы она была понятна широкому кругу читателей, начиная со студентов высших учебных заведений. Поэтому часть информации будет заведомо избыточной для опытных инженеров.

Особенности IP-ядра 8051s

Посмотрим на условное графическое изображение микроконтроллера семейства 8051 (рис. 1). Это изображение приводится в спецификации фирмы Intel, написанной Бобом Кёхлером (Bob Koehler) и вышедшей в мае 1980 г. [7].

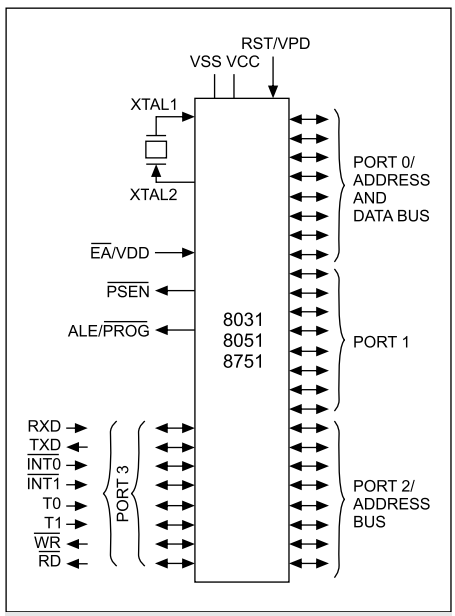

Рис. 1. Условное графическое изображение микроконтроллера 8051

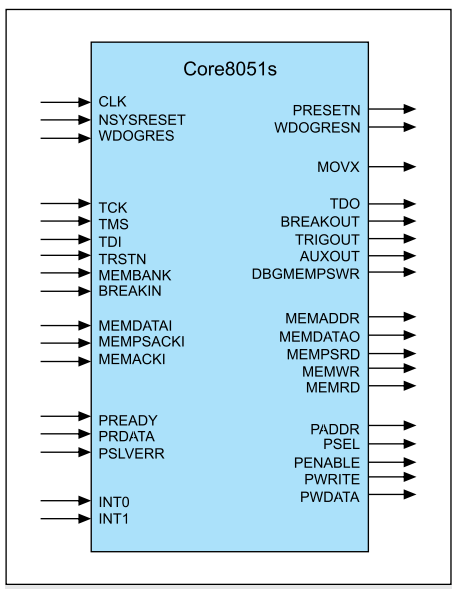

Рис. 2. Схематическое изображение линий ввода/вывода IP-ядра 8051s

А теперь посмотрим на схематическое изображение линий ввода/вывода IP-ядра 8051s (рис. 2), взятое из [8].

Ничего общего? Нет, видны две линии сигналов прерываний INT0 и INT1, правда, без инверсии. И все. Но мы уже знаем, что вместо портов ввода/вывода и линий управления микроконтроллера 8051, которые занимают большинство его выводов, у нас будет целая шина APB (Advanced Peripheral Bus, расширенная периферийная шина) стандарта AMBA. Остаются только выводы для подключения кварцевого резонатора и питания, которых в нашем IP-ядре и не может быть, и еще вывод сброса, который здесь есть, но называется NSYSRESET и имеет активный низкий уровень. Так что все в порядке.

Рассмотрим подробнее, что сохранилось у ядра 8051s от его классического предшественника, а что изменилось.

Процессорное ядро совместимо по набору команд с микроконтроллером 8051. Сохранены три разных пространства памяти, что позволяет использовать существующие компиляторы языка C для 8051.

Для уменьшения размера ядра и увеличения гибкости исключены:

- периферийные узлы, отображаемые на пространство регистров специальных функций (Special Function Registers, SFR);
- схема управления питанием;
- последовательный порт;
- порты ввода/вывода;
- таймер.

Доступны опционально и могут отключаться для экономии ресурсов ПЛИС:

- инструкции умножения и деления (MUL, DIV и DA) — представлены по умолчанию, но могут быть реализованы как NOP;
- второй указатель данных (data pointer 1) — отключен по умолчанию;
- управляющая логика для двух источников прерываний.

К ядру можно подключить отладочный модуль, работающий через интерфейс JTAG. Если у выбранной микросхемы есть специализированные выводы для подключения к JTAG, то можно их использовать, если нет, то для JTAG можно пользоваться линиями ввода/вывода общего назначения.

Доступ к памяти программ и данных может быть как асинхронным (с использованием сигналов подтверждения MEMPSACKI или MEMACKI соответственно), так и синхронным, с заданным числом тактов ожидания от нуля до семи. Вариант организации доступа выбирается при конфигурации IP-ядра.

Оценить ресурсы ПЛИС разных семейств, необходимые для реализации разнообразных конфигураций IP-ядра 8051s, а также максимально возможную тактовую частоту для каждой конфигурации можно по таблицам в [8].

Архитектура IP-ядра 8051s предусматривает параллельное выполнение команды и выборку кода следующей команды в одном цикле. Поэтому большинство однобайтовых команд выполняется за один машинный цикл. В свою очередь, один машинный цикл в 8051s выполняется за один период тактовой частоты. В результате средняя производительность ядра 8051s превышает производительность оригинального микроконтроллера Intel в восемь раз при работе на той же тактовой частоте.

В далеком мае 1980 г. Боб Кёхлер мог только мечтать обо всем этом.

Разработка аппаратной части проекта

До начала работы у нас на компьютере должна быть установлена САПР Libero фирмы Microsemi. Она существует в двух вариантах: Libero SoC и Libero IDE. Первая продолжает развиваться, регулярно выходят ее новые версии, но в ней не поддерживаются однократно программируемые AntiFuse ПЛИС, включая радиационно-стойкие и ранние семейства Flash-микросхем. На момент написания статьи доступна среда разработки Libero IDE версии 9.1 с «сервис-паком» 5. Поддержка ранних семейств есть именно в ней. Так как одна из наиболее вероятных областей интереса читателей — это разработка оборудования для жестких условий эксплуатации, дальнейшее изложение будет вестись на примере работы в Libero IDE. Маршрут проектирования в Libero SoC для общего случая описан в [9]. Там же приводится инструкция по получению бесплатной лицензии и установке САПР, которая подходит для обоих вариантов Libero. Отметим, что «старая» Libero IDE прекрасно работает в новой 64-разрядной ОС Windows 8.1, и все приведенные здесь примеры сделаны именно в ней.

Пусть у нас есть (или появится в скором будущем) готовое устройство с ПЛИС фирмы Microsemi, в которой можно поместить систему с IP-ядром микроконтроллера 8051s. В частности, автор использовал оценочную плату Cortex-M1-enabled ProASIC3 Development Kit фирмы Actel с установленной на ней ПЛИС M1A3P1000 FGG484. Ее описание можно найти на сайте [3], пройдя по ссылкам FPGA&SoC → Design Resources → Dev kits → Discontinued Kits. Все примеры в этой статье отрабатывались на ней. Сейчас эта плата уже не выпускается, но ее можно заменить на плату Cortex-M1-enabled ProASIC3L Development Kit, схема которой очень похожа. И пусть в этом устройстве к выводам ПЛИС будут подключены микропереключатели и светодиоды, как это обычно делается на отладочных платах. Создадим простейший учебный проект, в котором один светодиод будет мигать с частотой, например, 1 Гц, а другой будет загораться и гаснуть в соответствии с движением микропереключателя.

Создание проекта Libero

Приступим к созданию проекта. Запустим среду Libero IDE и дождемся появления ее главного окна (рис. 3).

Левую часть главного окна занимает инструмент Design Explorer, который служит для навигации по проекту. Справа находится каталог, из которого мы можем выбирать компоненты для проекта. Самую большую, центральную часть главного окна занимает поле **Project Flow**, из которого осуществляется управление проектом. Можно запрещать отображение отдельных полей главного окна и снова их восстанавливать через пункт главного меню **View→Windows and Toolbars**.

Вызовем мастер создания нового проекта New Project Wizard через пункт меню **Project→New Project** (рис. 4).

Рис. 4. Начало работы мастера New Project Wizard

Прежде всего, мастер предложит ввести название проекта и указать папку, где этот проект будет располагаться, а также указать предпочтительный язык описания аппаратуры. Сделаем это, нажмем кнопку **Next** и в следующем окне выберем целевую ПЛИС в таком порядке: сначала укажем семейство, затем тип микросхемы из этого семейства, и после этого нам будет предложено выбрать тип корпуса микросхемы (рис. 5).

Рис. 5. Выбор целевой ПЛИС

Еще раз нажмем кнопку **Next** и перейдем к окну **Select Integrated Tools** (рис. 6). Здесь нам предлагается определить программное обеспечение, которое будет интегрировано со средой Libero. Мы будем использовать пакеты Synplify AE, Modelsim AE и программу для работы с программатором FlashPro. Рядом с названиями этих программ не должно быть вопросительных знаков. На рисунке показана ситуация, когда среда Libero установилась неправильно, и рядом с Synplify AE стоит вопросительный знак. В таком случае надо исправить установку. (Проконтролировать подключение интегрированного ПО впоследствии можно из меню **Project→Profiles** главного окна Libero IDE). Есть вероятность, что исправить удастся простым перезапуском инсталлятора. Но не исключен и наихудший случай — ручное удаление из реестра Windows всех упоминаний слов Actel, Synplify и прочих после ручного стирания всех остатков Libero и сопутствующего ПО в папках установки. При этом ПО сторонних фирм (Mentor, Synopsys) надо деинсталлировать индивидуально.

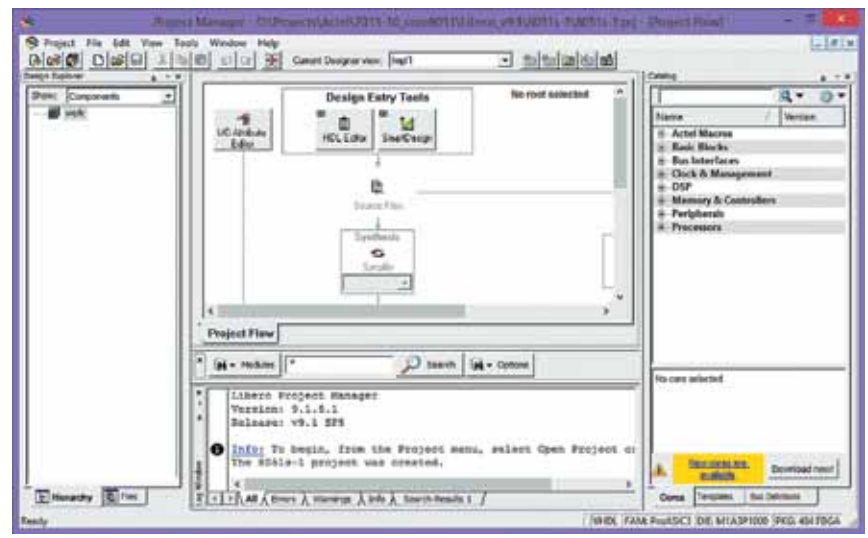

Рис. 3. Главное окно среды Libero IDE

Если все в порядке, то можно нажимать на кнопку **Finish**, так как у нас пока нет файлов, которые мы могли бы добавить в проект. В противном случае мы могли бы подключить к проекту ранее разработанные модули, «тестбенчи» и т. п.

Рис. 6. Окно Select Integrated Tools

Теперь мы можем проектировать нашу аппаратную часть. Разделим проектирование на две части: сначала соберем микроконтроллерную систему на базе готовых IP-ядер из каталога Libero, а потом создадим на языке описания аппаратуры память программ, так как подходящего ПЗУ в каталоге нет, и добавим ее в проект.

Найдем в поле **Project Flow** главного окна Libero IDE (рис. 3) группу **Design Entry Tools**, а в ней — большую кнопку **SmartDesign**. Щелкнем по ней мышью. Появится окно создания нового компонента (рис. 7). Выбираем для нового компонента тип **SmartDesign Component** и даем ему имя, например System8051s. Теперь нажимаем **OK** и видим, что поверх поля **Project Flow** появилось новое поле — **Canvas** («холст»). На этом «холсте» мы и будем рисовать нашу схему, укладывая на него экземпляры (instance) IP-ядер. А от поля **Project Flow** осталась закладка, она выглядывает из-под «холста», и мы всегда можем переключиться на другие операции с проектом.

Рис. 7. Создание нового компонента SmartDesign

Создание экземпляра 8051s

Первое, что мы положим на «холст», — конечно же, ядро 8051s. В правой части главного окна Libero IDE расположено поле **Catalog**. Выберем закладку **Cores** на нижней границе этого поля, развернем группу **Processors** щелчком по крестику рядом с ее именем

Рис. 8. Выбор ядра 8051s из каталога

и выберем строчку Core8051s (рис. 8) одиночным щелчком мыши.

Под списком IP-ядер отобразится краткое и хорошо составленное описание ядра 8051s со ссылками на документацию по нему. Ссылки из каталога иногда ведут на жесткий диск, иногда — в Интернет, но все они рабочие.

Теперь сделаем двойной щелчок мышью по строчке Core8051s. Появится окно настройки параметров ядра. Приводить здесь изображение этого окна мы не будем, оно большое и займет слишком много места на журнальной странице. Информации там немного, просто перечислим параметры и варианты их настройки.

В группе **Debug Configuration** собраны настройки встроенной отладочной системы JTAG. Ее можно совсем отключить, если из выпадающего списка **Debug** выбрать **Disabled**. Две другие опции списка **Debug** позволяют выбрать выводы, через которые будет подключаться интерфейс JTAG: через специализированные для JTAG выводы ПЛИС, если они есть (вариант **Enabled using UJTAG**), или через линии ввода/вывода общего назначения (вариант **Enabled using I/Os**). Второй вариант может пригодиться, если в нашей ПЛИС нет интерфейса JTAG или он уже используется для других целей. В нашем примере выберем **Enabled using UJTAG**.

Флажок **Include trace RAM** позволяет использовать для отладки программу, расположенную во внешнем ОЗУ, которое использует то же адресное пространство, что и внутренняя память программ. Этот флажок мы устанавливать не будем, так как обычный программатор FlashPro без дополнительных опций такую память не поддерживает [10].

И, наконец, из выпадающего списка **Number of hardware triggers/breakpoints** можно выбрать число аппаратных точек останова.

В группе **Optional Registers and Instructions** можно включить реализацию второго указателя данных и трех опциональных арифметических инструкций. В нашем проекте они не понадобятся.

Перейдем к группе **Program Memory Access** — доступ к памяти программ. Если установить здесь флажок **MEMPSACKI-controlled program memory**, то доступ к памяти программ будет асинхронным, то есть микроконтроллер в каждом цикле обращения к памяти будет дожидаться сигнала подтверждения MEMPSACKI. Этот сигнал сообщает, когда завершаются запись в память или установление данных для чтения. Если этот флажок не устанавливать, то доступ к памяти данных будет синхронным, то есть контроллер будет выделять на цикл обращения к памяти заданное число тактов. По истечении этих тактов цикл будет считаться законченным. Снимем этот флажок и установим параметр **Program Memory Wait Cycles** равным нулю, предполагая, что у нас будет достаточно быстродействующее ПЗУ. Если потом это окажется не так, мы всегда сможем изменить конфигурацию ядра.

В группе **External Data Memory Access** устанавливаются параметры доступа к внешней памяти данных. В нашем примере ее не будет, поэтому в этой группе ничего трогать не нужно.

Группа **Other Options**. Разрядность данных шины APB **APB data width** может быть равна 8, 16 или 32. Она должна быть не меньше разрядности данных любого периферийного узла, подключенного к APB. В нашем проекте для подключения к светодиодам и микропереключателям мы будем использовать сторожевой таймер на 32 разряда. Поэтому разрядность шины APB мы сделаем равной 32.

И наконец, выберем способ реализации внутреннего ОЗУ процессора (**Internal RAM (256x8) Implementation**). Здесь у нас есть три варианта. Instantiate RAM block означает, что будет создан экземпляр макроса блока памяти непосредственно в RTL-коде. Если выбрать **Infer RAM block during synthesis**, то в RTL-код будет вставлена в виде структурированного комментария директива для синтеза макроса блока памяти, и этот блок будет создан во время синтеза. В обоих этих случаях будет использоваться макрос блока памяти, поэтому результаты будут аналогичными, и ОЗУ будет реализовано на встроенных блоках ОЗУ ПЛИС. Третий вариант, **Infer registers for RAM during synthesis**, задает создание в RTL-коде директивы, которая предписывает синтезатору создать внутреннее ОЗУ на триггерах. При этом возрастет объем ресурсов, занимаемых процессором, но зато он будет лучше защищен от сбоев. Выберем первый вариант.

На этом конфигурирование IP-ядра 8051s закончено. Нажмем кнопку **OK**, и на нашем

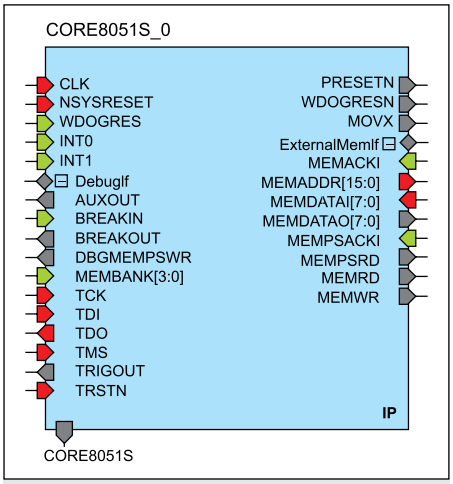

Рис. 9. Изображение сконфигурированного IP-ядра 8051s в схемном редакторе SmartDesign

«холсте» появится первый компонент — это ядро. Можно слегка растянуть его мышью в ширину, чтобы поместились все надписи, или перетащить компонент на другое место. Ради любопытства можно щелкнуть мышью по свернутым группам сигналов **DebugIf** (отладочный интерфейс) и **ExternalMemIf** (интерфейс внешней памяти), чтобы увидеть их целиком (рис. 9). Это любопытство не пустое, так мы познакомились с возможностью подключать сигналы к каждой линии интерфейса отдельно.

Если в настройках IP-ядра надо будет что-либо изменить, то всегда можно снова вызвать окно конфигурации двойным щелчком мыши по изображению компонента.

И еще одно замечание. В окне конфигурации любого ядра есть выбор типа лицензии. Нам везде предлагают тип RTL, и мы не будем его менять.

Создание экземпляра универсального блока ввода/вывода (GPIO)

IP-ядро CoreGPIO (General Purpose Input-Output) позволяет подключить к шине APB до 32 линий ввода и до 32 линий вывода общего назначения [11]. Для каждого бита можно задать постоянную конфигурацию на верхнем уровне проекта или же определить загружаемую конфигурацию. Во втором случае можно будет управлять конфигурацией входов и выходов во время работы системы, загружая управляющие байты в специальные регистры.

В каталоге Libero IDE (рис. 8), выше группы **Processors**, в которой мы уже побывали, есть группа **Peripherals**. Найдем в ней ядро CoreGPIO и сделаем двойной щелчок мышью на его имени. Появится окно конфигурации ядра (рис. 10).

Настроим глобальные параметры ядра. Разрядность данных шины APB **APB DataWidth** установим равной 32, чтобы избежать проблем при моделировании [10]. Число линий ввода/вывода **Number of I/Os**

Рис. 10. Конфигурирование ядра CoreGPIO

делаем равным трем: две для светодиодов и одну для микропереключателя. Заметим, что с этого момента группы **I/O bit** с номерами больше двух стали неактивными. **Single Bit Interrupt Port** устанавливаем в состояние **Disabled**. Тип разрешения выхода **Output Enable** устанавливаем **External**. Это означает, что буферы с тремя состояниями предполагается располагать вне ядра CoreGPIO. В нашем проекте они вообще не понадобятся.

Теперь займемся каждым битом в отдельности. Нам понадобятся следующие линии ввода/вывода:

- один вход для опроса микропереключателя;
- один выход для зажигания мигающего светодиода;
- один выход для зажигания светодиода, отслеживающего микропереключатель.

Зададим для всех битов постоянную конфигурацию, чтобы не усложнять наш проект. Это означает, что для всех битов ядра CoreGPIO надо установить флажок **Fixed Config**.

Пусть бит 0 у нас будет входом, а биты 1 и 2 — выходами. Выберем для них из выпадающих списков **I/O Type** соответственно **Input** и **Output**. Запретим прерывания (установим **Disabled** в выпадающих списках **Interrupt Type**). Состояние выходов при сбросе **Output on Reset** зададим «0», чтобы светодиоды при включении питания не горели. (Перед этим надо посмотреть на схему оценочной платы, чтобы выяснить, как там подключены светодиоды.)

Закончив конфигурацию, щелкаем по кнопке **OK**. На «холсте» появляется универсальный блок ввода/вывода CoreGPIO.

Создание экземпляра таймера

Таймер нам понадобится для отсчета интервалов времени, через которые будет зажигаться и гаснуть светодиод. Найдем в каталоге, в той же группе **Peripherals**, IP-ядро **CoreTimer** и вызовем двойным щелчком окно его конфигурации. При создании аппаратной части проекта можно задать только два параметра: выбрать разрядность счетчика (Width) и полярность сигнала прерывания (Interrupt Active Level). Остальные параметры таймера настраиваются программно. Установим значения этих параметров соответственно **32 bit** и **High** и щелкнем по кнопке **OK**. На «холсте» появляется таймер.

Создание экземпляра сторожевого таймера

Сторожевой таймер (Watchdog) защищает от зависаний программу, работающую в контроллере. Сторожевой таймер содержит непрерывно работающий вычитающий счетчик. Программа должна регулярно записывать в этот счетчик новое число. Если из-за какой-то ошибки, например из-за зацикливания, очередного обновления не будет, то сторожевой таймер досчитает до нуля и выдаст сигнал сброса. Выполнение программы будет прервано, и она начнет работать с первого оператора.

Попробуем еще один способ добавления нового экземпляра компонента. Найдем в группе **Peripherals** IP-ядро **CoreWatchdog** и перетащим его на «холст» мышью сразу в нужное место. Когда мы отпустим левую кнопку мыши, появится окно настройки сторожевого таймера. В нем будет всего один параметр — разрядность счетчика. Сделаем его равным 32, пусть у нашей программы будет больше времени на «раздумья». Нажимаем на кнопку **OK**, и на «холсте» появляется сторожевой таймер.

Создание экземпляра шины APB

Следующий шаг — подключение IP-ядра шины APB [12], через которую наш процессор будет общаться с периферийными устройствами. Найдем в группе **Bus Interfaces** IP-ядро **CoreAPB3**. Именно эту версию шины, APB3, предлагают использовать для ядра 8051s в [8]. На момент написания статьи в каталоге предлагалась версия 4.0 ядра CoreAPB3.

Ядро APB3 связывает одно ведущее устройство шины APB (master) и до 16 ведомых устройств (slave). Ведущим устройством у нас будет ядро 8051s. Адресное пространство поделено поровну между шестнадцатью областями ведомых устройств, называемых слотами (slot).

Вызовем окно конфигурации ядра CoreAPB3 двойным щелчком мыши по строке **CoreAPB3** в каталоге. Опять же, изображение этого окна займет очень много места, не добавив ничего существенного к содержанию статьи, поэтому просто перечислим нужные опции, которые управляются из этого окна. Обратим внимание на вертикальную полосу прокрутки вдоль правой границы окна. Все элементы настройки не помещаются на экран, и этой полосой надо пользоваться.

- **APB Master Data Width Bus** — разрядность шины данных ведущего устройства APB. Она неизбежно будет такой же, какую мы задали для 8051s.
- **Number of address bits driven by master** — разрядность адресной шины ведущего устройства. У нас пока всего два устройства, поэтому сделаем ее минимальной, 12 бит.
- **Position in slave address of upper 4 bits of master address** — положение старших четырех разрядов адресной шины ведущего устройства в адресе ведомого устройства. При 12-разрядном адресе у нас есть только один вариант: биты с 11 по 8 ([11:8]).
- **Indirect Adressing** — косвенная адресация. В нашем проекте мы не будем ее использовать, выберем Not in use.
- **Allocate memory space to combined region slave** — резервировать область памяти под комбинированную область (combined region). Можно назначить один или несколько слотов на нее. Этот механизм можно использовать для доступа через один слот к ресурсам с диапазоном адресов, большим, чем 1/16 адресного пространства. Нам эта возможность не понадобится, не будем устанавливать ни одного флажка.
- **Enabled APB slave slots** — разрешить слоты ведомых устройств APB. Включим три слота для трех наших устройств.
- **Testbench** — выберем вариант **User**.

Все, конфигурация IP-ядра CoreAPB3 закончена. Нажимаем на кнопку **OK**, и у нас на «холсте» появляется изображение шины APB.

Рисование схемы — автоматическая часть

Все компоненты, которые работают на шине APB, уже лежат у нас на «холсте» (рис. 11).

Поскольку интерфейсы готовых ядер стандартизированы, большую часть соединений SmartDesign может выполнить автоматически. Щелкнем правой кнопкой мыши по любому свободному месту «холста» и выберем из выпавшего меню пункт **Auto Connect**. Откроется окно **Modify Memory Map** (рис. 12).

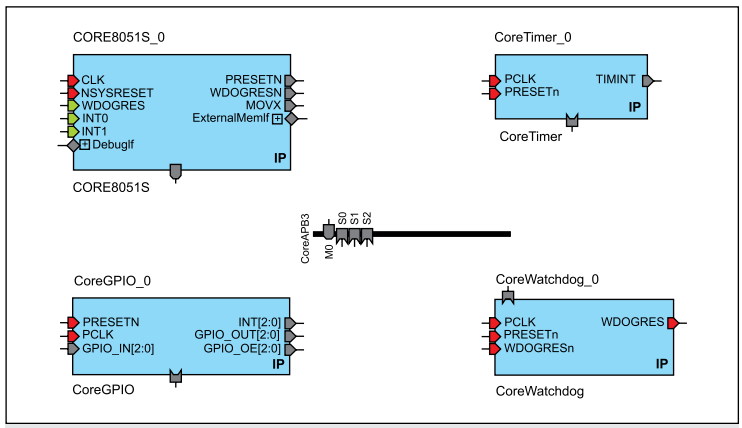

Рис. 11. Модули, работающие на шине APB

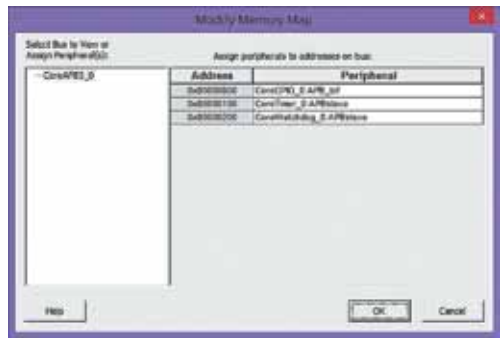

Рис. 12. Окно Modify Memory Map для распределения адресов на шине APB

Посмотрев на это окно, мы увидим, что SmartDesign уже сам назначил адреса нашим устройствам. Если его вариант нас не устраивает, мы можем щелкнуть мышью по имени периферийного устройства и выбрать из выпавшего списка пустую строку. Проделав эту операцию для всех устройств, мы затем сможем сопоставить каждому адресу то устройство, которое сочтем нужным.

Теперь нажимаем на кнопку **OK**, и на «холсте» появляется почти готовая схема (рис. 13). В этой схеме SmartDesign уже проделал за нас всю рутинную работу: подключил к шине APB ведущее устройство — ядро 8051s, и три ведомых устройства — универсальный блок ввода/вывода, таймер и сторожевой таймер, соединил между собой процессор и сторожевой таймер, а также развел глобальные сигналы тактовой частоты SYSCLK и синхронного сброса шины PRESETN. Заметим, что глобальные сигналы SYSCLK и NSYSRESET уже выведены наружу для будущего подключения нашей подсистемы на верхнем уровне проекта.

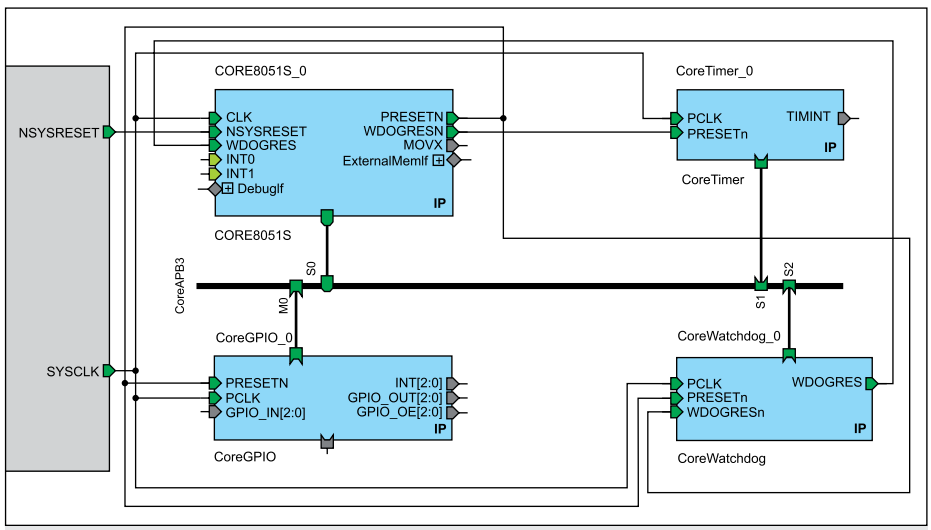

Рис. 13. Схема после автоматического подключения

Рисование схемы — «ручная» часть

Теперь нам надо закончить схему вручную:

- сделать недостающие соединения внутри подсистемы;
- вывести наружу сигналы, которые понадобятся на верхнем уровне проекта;
- подать какие-либо уровни на неиспользуемые входы компонентов;
- пометить неиспользуемые выходы компонентов.

Соединять выводы компонентов между собой будем так:

- нажмем клавишу **Ctrl** и, не отпуская ее, щелкнем левой кнопкой мыши по всем выводам блоков, которые надо соединить в одну цепь. Щелкать можно как по названиям выводов, например, INT0, так и по пятиугольным символам рядом с ними;
- щелкнем правой кнопкой мыши по одному из выбранных выводов и выберем из выпадающего меню пункт **Connect**.

После этого SmartDesign сам проведет соединение. Соединим между собой один из входов запроса прерывания 8051s, пусть это будет INT0, с выходом таймера TIMINT.

Далее выведем наружу те сигналы, которые нам понадобятся на верхнем уровне проекта. Начнем с интерфейса внешней памяти ExternalMemIf и сигнала MOVX, они будут использоваться для подключения памяти. Щелкнем правой кнопкой по порту ExternalMemIf и выберем из выпавшего меню пункт **Promote to Top Level**. SmartDesign поставит на границе схемы порт и соединит его шиной с ExternalMemIf. Имя порта совпадает с именем сигнала или шины. При необходимости порт можно переименовать: щелкнуть по нему правой кнопкой мыши и выбрать пункт **Modify Port**, а затем в появившемся окне ввести новое имя. Аналогично подключим сигнал MOVX.

Во время редактирования схема может принять неудобочитаемый вид. Например, во время выполнения последней операции граница схемы была не видна, и конец шины с портом мы не увидели. Но в любой момент можно щелкнуть по «холсту» правой кнопкой мыши и выбрать **Auto-Arrange Instances** и/или **Auto-Arrange Connections**, и схема преобразится: размеры «холста» будут подогнаны под ее размеры, компоненты перестроятся так, чтобы связи между ними хорошо просматривались, а сами связи примут более понятный вид.

Выведем наружу линии для подключения светодиодов и микропереключателя. На условном графическом изображении модуля GPIO входная и выходная шины представлены как неделимые и трехразрядные. У нас же должно быть два выхода и один вход.

Разделим выходную шину GPIO. Для этого щелкнем правой кнопкой мыши по порту GPIO_OUT модуля GPIO и выберем из выпавшего меню пункт **Add Slice**. В появившемся окошке (рис. 14) зададим нужную часть шины, в данном случае заменим «0» на «1». (Младший разряд GPIO мы отвели под входной сигнал от микропереключателя.) После нажатия кнопки **OK** возле имени порта GPIO_OUT появится обычный для системы Windows символ разворачиваемой структуры — маленький плюс в квадратике. Если щелкнуть по этому плюсу, то шина «развернется», и мы увидим выделенную нами группу разрядов [2:1]. Теперь можно

Рис. 14. Окно задания фрагмента шины Add Slice

выводить отделенные два разряда для использования на верхнем уровне проекта. Судьбу нулевого разряда тоже надо определить. Еще раз щелкнем правой кнопкой по порту GPIO_OUT и введем в оба поля окна **Add Slice** нули. Теперь при разворачивании выходной шины GPIO мы будем видеть отдельный разряд [0] и шину [2:1].

Аналогично мы поступим с портом GPIO_IN: выведем наружу его бит 0 и отдельно опишем разряды 2:1.

Теперь нам надо подать какой-либо определенный уровень на неиспользуемые входы компонентов. Для этого надо щелкнуть по такому входу правой кнопкой мыши и выбрать из выпавшего меню пункт **Tie Low** или **Tie High**. Рядом со входом появится маленький значок земли или питания. Подадим низкий уровень на вход INT1 у 8051s и входы GPIO_IN[2:1].

Неиспользуемые выходы надо пометить: щелкнуть по выходу правой кнопкой мыши и выбрать из выпавшего меню пункт **Mark Unused**. Пометим как неиспользуемые выходы INT[2:0], GPIO_OUT[0] и GPIO_OE[2:0] в блоке **CoreGPIO**.

Создание памяти программ

Теперь нашему минимальному учебному проекту не хватает только памяти программ. (Внешнее (относительно ядра 8051s) ОЗУ мы создавать не будем, нам пока хватит внутреннего ОЗУ 256×8.) Если в распоряжении читателя есть ПЛИС с большим массивом Flash-памяти, например Fusion или SmartFusion, он может для создания памяти программ воспользоваться встроенным в САПР Libero инструментом Flash Memory System Builder [10].

На нашей оценочной плате установлена ПЛИС семейства ProASIC3, где объем Flash ПЗУ составляет всего 128 байт, и использовать его в качестве памяти программ для сколько-нибудь реальных задач не получится. Поэтому мы пойдем другим путем, более универсальным: напишем код ПЗУ на языке VHDL и вставим такое ПЗУ в наш проект. Заодно мы получим практику создания модулей проекта на языке описания аппаратуры и их использования с инструментом SmartDesign.

Сейчас это будет просто заглушка с нулевыми кодами в каждом байте. Она нужна нам только для проверочной компиляции проекта. Создание кода программы и размещение его в ПЗУ будет описано во второй части статьи.

Заглянем еще раз в руководство [8]. Там в разделе Ports перечислены сигналы интерфейса для подключения внешней памяти (External Memory Bus Interface). Для работы с ПЗУ предназначены следующие сигналы:

- MEMPSACKI (Program memory read acknowledge) — входной (относительно 8051s) сигнал подтверждения чтения памяти программ. Мы выше договорились, что не будем его использовать;
- MEMDATAI (Memory data input) — восьмиразрядная шина для чтения данных;
- MEMADDR (Memory address) — шестнадцатиразрядная шина адреса;
- MEMPSRD (Program store read enable) — выходной сигнал разрешения чтения из ПЗУ.

Предположим, что для нашей будущей программы нам понадобится 1 кбайт ПЗУ. Если после компиляции программы окажется, что нужна память большего или меньшего размера, мы легко сможем изменить объем ПЗУ.

Итак, приступим к созданию ПЗУ. Перейдем в окно **Project Flow**, щелкнув по одноименной закладке. В группе **Design Entry Tools** щелкнем по большой кнопке **HDL Editor**, которая вызовет нам окно создания нового модуля проекта. В столбце **Select a Type** уже выделена строка **VHDL Source File**. Введем в поле **Name** имя будущего модуля, например *MyROM*, и нажмем **OK**. Поверх окна **Project Flow** появится окно редактора для создания текстов на языке описания аппаратуры.

К сожалению, текстовый редактор в среде Libero IDE не слишком удобен: он имеет мало возможностей, а главное — не дает изменить размер шрифта. Поэтому автор предлагает редактировать файлы VHDL во внешнем редакторе. Методика использования редактора Notepad++ совместно со средой Libero описана в [9]. Листинг 1 показывает код на языке VHDL, который будет исполнять обязанности ПЗУ до разработки программы для микроконтроллера.

Не будем здесь разбирать его, отложим рассмотрение полного цикла создания ПЗУ до второй части статьи.

После написания кода проверим его синтаксис. До этого надо убедиться, что открыто окно протокола среды Libero — **Log Window**. Оно должно находиться ниже окна **Project Flow**. Если его там нет, то надо открыть его из меню **Views→Windows and Toolbars**. Найдем в окне Hierarchy в разделе **Work** модуль *MyROM*, щелкнем по нему правой кнопкой мыши и выберем из выпавшего меню пункт **Check HDL File**. Если в коде будут найдены ошибки, то сообщения о них с указанием номеров строк отобразятся в окне протокола. Когда все ошибки будут исправлены, готовый модуль можно положить на наш «холст». Делаем это так:

1) Создаем символ для отображения в графическом редакторе: щелкаем правой кнопкой мыши по имени *MyROM* в окне **Hierarchy** и выбираем из выпавшего меню пункт **Create Symbol**.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity MyROM is
port (
      Addr: in std_logic_vector(15 downto 0);
      ReadEnable: in std_logic;
      DataOut: out std_logic_vector(7 downto 0)
);
end MyROM;

architecture MyROM_0 of MyROM is

   constant AddrSize : natural := 10; --разрядность адресной шины
   subtype byte is std_logic_vector(7 downto 0);
   type ROM is array (0 to 2**AddrSize-1) of byte;
   constant ProgramROM : ROM := (others => ('0','0','0','0','0','0','0','0'));

   -- Преобразование std_logic_vector в десятичное число:
   function SLVtoNatural (SLV: std_logic_vector) return natural is
      variable result : natural;
   begin
      result := 0;
      for i in 0 to SLV'length - 1 loop
         if SLV(i) = '1' then result := result + 2**i; end if;
      end loop;
      return result;
   end SLVtoNatural;

begin

process(Addr, ReadEnable)
   variable MyAddr: std_logic_vector(AddrSize-1 downto 0);
   variable Index: natural;
begin
   MyAddr := Addr(AddrSize-1 downto 0);
   Index := SLVtoNatural(MyAddr);
   if ReadEnable = '0' then DataOut <= ('Z','Z','Z','Z','Z','Z','Z','Z');
   else DataOut <= ProgramROM(Index);
   end if;
end process;

end MyROM_0;
```

Листинг 1. Код заглушки памяти программ

2) Вставляем полученный символ в схему: еще раз щелкаем правой кнопкой мыши по *MyROM* и выбираем пункт **Instantiate in System8051s**. На «холсте» появляется изображение ПЗУ.

Теперь подключим наше ПЗУ к ядру 8051s. Щелкнем по группе контактов интерфейса внешней памяти ExternalMemIf, чтобы развернуть его. Затем соединим 8051s и *MyROM* так, как показывает таблица 1.

Таблица 1. Соединение ядра 8051s с модулем памяти программ

Core8051s	MyRom
MEMADDR[15:0]	Addr[15:0]
MEMDATA[7:0]	DataOut[7:0]
MEMPSRD	ReadEnable

После этого помечаем как неиспользуемые следующие порты Core 8051s (рис. 15):

- MOVX;
- MEMDATAO [7:0];
- MEMRD;
- MEMWR.

Проверка схемы и создание компонента

Теперь проверим нашу схему. Выберем из главного меню пункт **SmartDesign→Check Design Rules**. В ответ поверх «холста» развернется окно сообщений **Design Rules Check**, где нам сообщат, что выходы отладочного интерфейса (который мы отключили при конфигу-

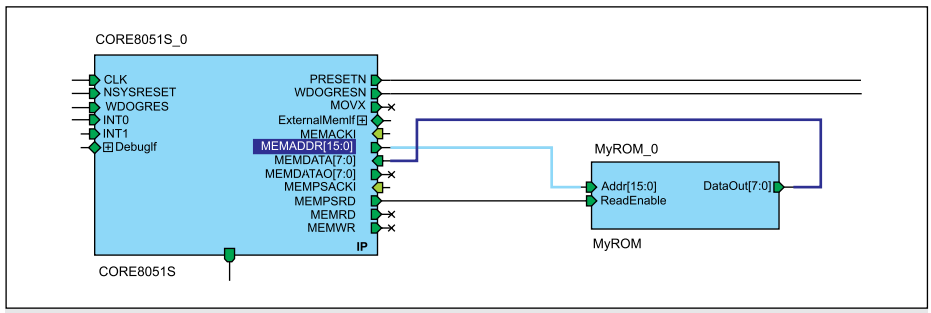

Рис. 15. Подключение модуля ПЗУ к IP-ядру 8051s

рации) у нас не подключены (**Floating Output Pin…**). Это не сообщение об ошибке, а предупреждение, логика его появления непонятна, так как про входы отладочного интерфейса нам ничего не сказали. Но хорошая практика — убирать все предупреждения, какие получится. Возвращаемся к схеме, щелкая по закладке **Canvas**, разворачиваем шину DebugIF и помечаем все выходы как неиспользуемые. Затем шину DebugIF можно свернуть. Снова проверяем схему. Теперь получаем сообщение **Design Rules Check of SmartDesign "<имя проекта>" found no errors and no warnings**, то есть «все в порядке».

Выполняем **Auto-Arrange Instances** и **Auto-Arrange Connections**. Схема готова (рис. 16).

Осталось сделать из полученной схемы компонент для подключения на верхнем уровне проекта. Щелкаем правой кнопкой мыши по «холсту» и выбираем пункт **Generate Design**. Через какое-то время мы получим сообщение о завершении процедуры.

Назначение сигналов на выводы ПЛИС

Для назначения сигналов на выводы ПЛИС удобнее всего использовать файл физических ограничений. Это текстовый файл с расширением *.pdc*. После исследования принципиальной электрической схемы оценочной платы у автора получилась таблица 2.

Теперь на основе этой таблицы создадим файл в формате PDC. Проще всего создавать его в текстовом редакторе. Запустим из окна **Project Flow** инструмент SmartDesign и в столбце **Select a Type** окна **New** выберем строку **Physical Design Constraint File**. Далее в поле **Name** введем имя этого файла, например *System8051s*, а из списка Create With выберем **Text Editor**. Затем нажмем кнопку **OK** и воспроизведем в текстовом редакторе листинг 2.

```
#set_io <ИМЯ СИГНАЛА> -pinname <ИМЯ ВЫВОДА В КОРПУСЕ ИМС>
set_io NSYSRESET -pinname W6
set_io SYSCLK -pinname E4
set_io GPIO_IN -pinname K21
set_io GPIO_OUT\[2\] -pinname R17
set_io GPIO_OUT\[1\] -pinname P16
```

Листинг 2. Файл физических ограничений

Синтаксис этого файла понятен без комментариев. Отметим только, что перед квадратными скобками надо ставить символ обратной дроби. Если потом потребуется редактировать этот файл (а потребуется обязательно, мы же будем расширять наш проект), то надо сделать так:

- открыть в окне **Design Explorer** закладку **Files**;
- найти там группу **Constraint Files**;
- развернуть ее и щелкнуть правой кнопкой мыши по имени файла *.pdc*, а потом из выпавшего меню выбрать **Open with Text Editor**.

Для назначения выводов читатели также могут использовать утилиту PinEditor из состава Libero, которая позволяет удобно подключать логические линии проекта к выводам корпуса микросхемы в графическом виде. Файл *.pdc* в этом случае можно получить, выбрав в окне **Designer** меню **File→Export→Constraint File**. Этот файл может быть повторно использован, например в другом проекте.

Таблица 2. Соответствие сигналов схемы и выводов ПЛИС

Сигнал	Вывод ПЛИС
NSYSRESET	W6
SYSCLK	E4
GPIO_IN (вход для микропереключателя)	K21
GPIO_OUT[2] (выход для светодиода)	R17
GPIO_OUT[1] (выход для светодиода)	P16

Синтез и трассировка проекта

Синтез проекта

Щелкнем в окне **Project Flow** по кнопке **Synthesis**. Через некоторое время запустится синтезатор **Synplify**. Из его главного меню выбираем пункт **Run-Run**, дожидаемся окончания синтеза и закрываем окно синтезатора.

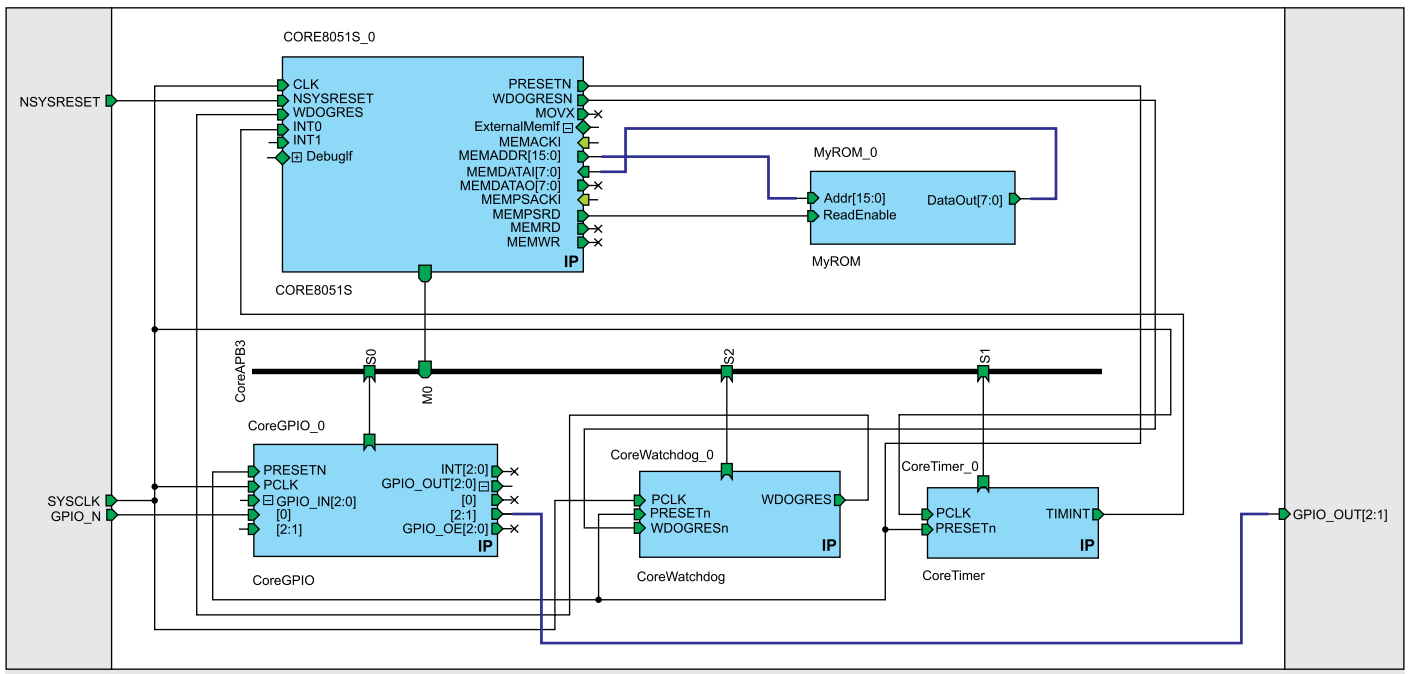

Рис. 16. Схема микроконтроллера с периферийными устройствами и памятью программ

Если все прошло успешно, то рядом с надписью **Post-Synthesis Files** в окне **Project Flow** должна появиться зеленая галочка.

Трассировка проекта

В окне **Project Flow** щелкнем по кнопке **Place&Route**. Появится окно **Organize Constraint for Designer** (рис. 17). Проследим, чтобы в правом поле этого окна был наш файл *.pdc*. Если это не так, то выберем его в левом поле окна и нажмем на кнопку **Add**.

Файл временных ограничений *.sdc* пусть пока создается автоматически.

Затем нажмем кнопку **OK**. Запустится модуль Designer среды Libero. При первом запуске он попросит уточнить ряд параметров ПЛИС. Установим их в соответствии со своим проектом. А вот на предложение обновиться через Интернет лучше ответить отказом, ибо программа может зависнуть.

Если в окне протокола не появилось сообщений об ошибке, то можно считать создание аппаратной части проекта законченным.

Моделировать получившийся проект и загружать прошивку в ПЛИС мы пока не будем, без программы для микроконтроллера это неинтересно.

В следующей части статьи мы и займемся разработкой этой программы.

Рис. 17. Окно конфигурации файлов ограничений

Литература

1. http://ru.wikipedia.org/wiki/Intel_8051
2. http://silabs.newshq.businesswire.com/press-release/product-news/silicon-labs-reference-design-helps-developers-get-motors-spinning-less-f
3. www.microsemi.com
4. http://soc.microsemi.com/products/ip/search/detail.aspx?id=541
5. http://soc.microsemi.com/products/ip/search/detail.aspx?id=648
6. Магда Ю. С. Микроконтроллеры серии 8051: практический подход. М.: ДМК Пресс. 2008.
7. 8051 Single-Chip Microcomputer. Architectural Specification and Functional Description. Intel Corporation, 1980. http://bitsavers.org/pdf/intel/8051/8051_Microcomputer_Preliminary_Architectural_Specification_May80.pdf
8. Core8051s v2.4 Handbook. www.microsemi.com
9. Иоффе Д. С. Libero SoC — быстрый старт // Компоненты и технологии. 2013. № 10.
10. Core8051s Embedded Processor Hardware Deve-lopment Tutorial for Fusion Mixed-Signal FPGAs. www.microsemi.com
11. CoreGPIO v3.0 Handbook. www.microsemi.com
12. CoreAPB3 v4.0 Handbook. www.microsemi.com