Проектирование на программируемых системах на кристалле PSoC Cypress

Необходимость данной публикации вызвана, по мнению автора, недостатком русскоязычных материалов, посвященных проектированию на микросхемах PSoC. Конечно, рассмотреть в рамках статьи все особенности микросхем PSoC не представляется возможным, так как одно только техническое руководство имеет объем свыше пятисот страниц. Наша цель — дать читателю представление о процессе проектирования с использованием этих весьма интересных микросхем и помочь в их освоении, приведя несложные примеры.

Дмитрий КИЛОЧЕК
Dmitry.Kilochek@macrogroup.ru

Данная статья открывает цикл, посвященный проектированию на микросхемах PSoC фирмы Cypress. В первой части рассмотрено пошаговое создание несложного проекта в среде проектирования PSoC Designer, затрагивающего только микропроцессорное ядро.

Микросхемы PSoC фирмы Cypress представляют собой 8-битный микроконтроллер, содержащий микропроцессорное ядро и массив цифровых и аналоговых блоков, позволяющий реализовывать необходимые пользователю периферийные функции, как вполне стандартные, например ШИМ, АЦП или UART, так и такие необычные для микроконтроллеров, как, например, аналоговые фильтры и инструментальные усилители. Благодаря тому, что PSoC позволяют сократить количество используемых внешних компонентов, это существенно упрощает процесс разработки, удешевляет устройство и одновременно повышает его гибкость за счет возможности перепрограммирования в системе или реконфигурирования внутренней структуры прямо в процессе работы. Устройство микросхем PSoC было описано в статьях [5] и [6] в предыдущих номерах журнала, поэтому в данной статье останавливаться на ознакомлении с внутренней структурой этих микросхем мы не будем, а перейдем к рассмотрению процесса создания проекта на базе этих чипов. Будет рассмотрен пакет САПР PSoC Designer, являющийся основным средством разработки, и создан простой пример, демонстрирующий работу портов ввода-вывода микроконтроллерного ядра PSoC и работу контроллера прерываний.

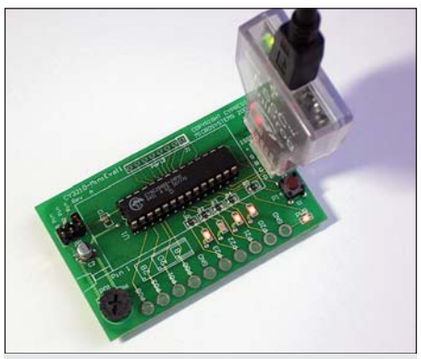

Рис. 1. Отладочная плата и программатор из комплекта MiniProg

Установка программного обеспечения и оборудования

Для работы потребуются пакет PSoC Designer (версия 4.2) и последняя версия пакета обновлений (версия 4.2 SP 3), установка которого производится поверх установленного PSoC Designer 4.2. САПР PSoC Designer представляет собой интегрированную среду разработки, содержащую редактор исходных текстов программ, редактор внутренней структуры программируемой системы и отладчик.

Помимо пакета PSoC Designer для разработки потребуется и не входящий в его состав программатор — PSoC Programmer (последняя версия 1.22.0.9). Все программы доступны на официальном сайте **www.cypress.com**.

Существует два основных аппаратных средства, применяемых при разработке, — CY3215-DK (ICE-Cube) и CY3210-MiniProg1 (рис. 1). ICE-Cube является внутрисхемным эмулятором и используется для отладки проекта, а также может осуществлять «прошивку» микросхем PSoC. MiniProg является исключительно программатором и не поддерживает никаких возможностей для отладки проекта, его достоинство — низкая цена.

Как ICE-Cube, так и MiniProg подключаются к ПК через кабель mini-USB. При первом подключении необходимо установить соответствующий драйвер. Драйверы для ICE-Cube и MiniProg находятся в каталоге \Cypress MicroSystems\PSoC Programmer\drivers.

Создание проекта в PSoC Designer

При запуске PSoC Designer возникает окно, предлагающее открыть существующий проект либо создать новый. При создании нового проекта необходимо выбрать тип микросхемы, на базе которой будет реализовываться устройство. В комплекте MiniProg есть два образца микросхем CY8C29466-24PXI, выберем эту микросхему для создания нашего проекта. Далее необходимо выбрать язык, на котором будет создан файл, содержащий функцию main — основную функцию проекта. На выбор предлагается язык C или Assembler (рис. 2), однако следует учесть, что компилятор языка C, входящий в состав PSoC Designer, требует лицензирования и без регистрационного кода работать не будет. В том случае, если для разработки приобретается набор ICE-Cube, то в его состав входит и лицензия для компилятора C, в противном же случае необходимо покупать лицензию отдельно или пользоваться языком ассемблера.

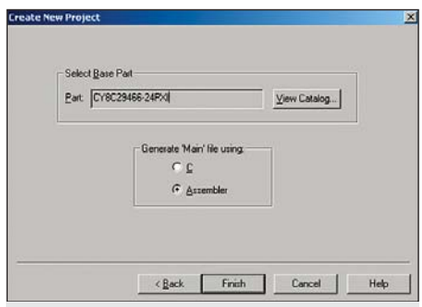

Рис. 2. Выбор типа микросхемы и языка программирования

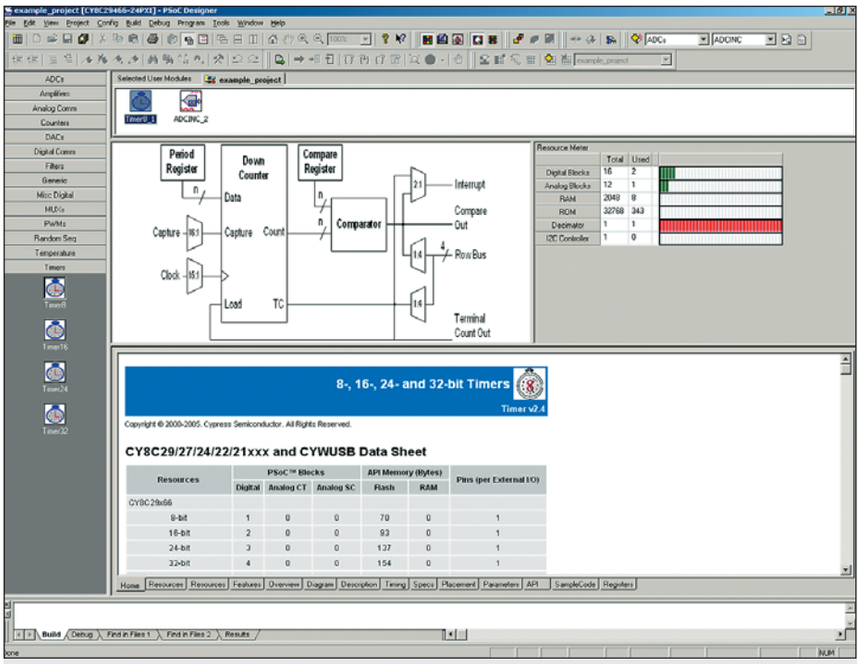

Рис. 3. Режим выбора модулей

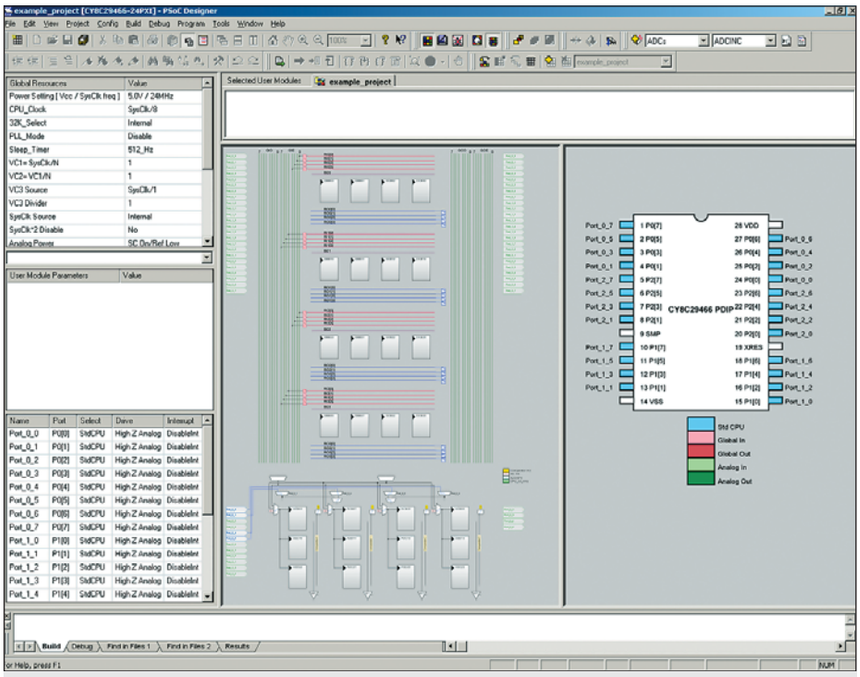

Рис. 4. Режим редактирования расположения и связей

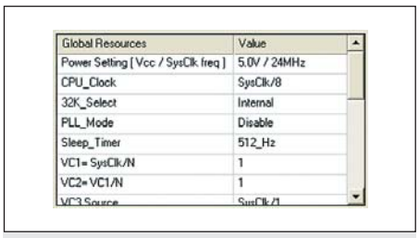

Рис. 5. Окно настройки глобальных ресурсов микросхемы (Global Resources)

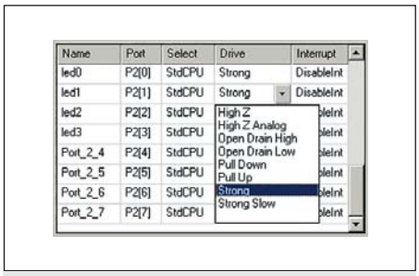

Рис. 6. Окно настройки портов микроконтроллера

После выбора микросхемы открывается окно редактора структуры микросхемы (Device Editor) в режиме выбора модулей пользователя (User Module Selection View, рис. 3). В этом режиме можно выбрать из библиотеки те модули, которые будут использованы в проекте, а также отображается структура модуля, его описание и информация о занятых и доступных ресурсах микросхемы.

Никаких модулей выбирать не будем, а перейдем в другой режим редактора структуры микросхемы — режим редактирования связей и расположения модулей (Interconnection View, рис. 4).

Так как в нашем проекте не используются никакие модули, то в данном окне нам будут интересны только настройки глобальных ресурсов микросхемы (Global Resources) — окно в левом верхнем углу (рис. 5), а также настройки портов микроконтроллера — окно в левом нижнем углу (рис. 6).

Рассмотрим подробнее настройки глобальных ресурсов микросхемы.

Параметр **Power Setting** [**Vcc/SysClk freq**] устанавливает используемое микросхемой напряжение питания и тактовую частоту системы. Возможные варианты выбора этого параметра немного отличаются в зависимости от используемой серии PSoC или от температурного исполнения микросхемы. Для чипа CY8C29466-24PXI на выбор предлагаются следующие комбинации: 3,3 В/24 МГц, 3,3 В/6 МГц, 5 В/24 МГц и 5 В/6 МГц.

Параметр **CPU_Clock** задает тактовую частоту микропроцессорного ядра M8C. Эта частота получается делением системной частоты на 2^n, где n — от 1 до 8.

Параметр **32K_Select** указывает источник частоты 32 К — внутренний (Internal) RC-генератор на 32 000 Гц (точность этого генератора невелика), либо тактирование от внешнего (External) часового кварца — в этом случае частота составляет 32 768 Гц, что позволяет реализовывать внутренние часы реального времени.

Параметр **PLL_Mode** отвечает за синхронизацию внутреннего генератора 24 МГц с внешним кварцем 32 768 Гц.

Параметр **SleepTimer** устанавливает период специального внутреннего таймера, который может использоваться для генерации прерывания для выхода из режима «сна», для реализации часов реального времени или прочих нужд. Период этого таймера выбирается из следующих вариантов: 512 Гц, 64 Гц, 8 Гц, 1 Гц (деление на 2^6, 2^9, 2^{12} и 2^{15} соответственно).

Параметры **VC1** и **VC2** устанавливают делители для получения частот, используемых

для тактирования цифровых и аналоговых блоков. Частота VC1 формируется из системной делением на коэффициент от 1 до 16. Частота VC2 формируется из VC1 также делением на коэффициент от 1 до 16.

Параметр **VC3 Source** позволяет выбрать источник для формирования частоты VC3, которая может использоваться как для тактирования цифровых блоков, так и для генерации прерывания. Источниками для формирования VC3 могут быть частоты VC1 или VC2, системная частота, либо удвоенная системная частота (при системной частоте 24 МГц становится возможным работа цифровых блоков на частоте 48 МГц).

Параметр **VC3 Divider** — делитель для формирования частоты VC3, он лежит в диапазоне от 1 до 256.

Параметр **SysClk Source** указывает источник системной частоты — внутренний генератор или внешняя частота, подаваемая на контакт P1[4].

Параметр **SysClk*2 Disable** включает или отключает встроенный умножитель системной частоты.

Параметр **Analog Power** устанавливает мощность источников опорных напряжений и определяет, включены или выключены аналоговые блоки на переключаемых конденсаторах. Мощность источников опорных напряжений должна быть выбрана равной либо большей, чем установки мощности используемых в проекте пользовательских модулей.

Параметр **Ref Mux** устанавливает напряжение «аналоговой земли» для аналоговых блоков и диапазон входных напряжений.

Параметр **AGndBypass** позволяет использовать 4-й контакт порта P2 для подключения конденсатора для дополнительной фильтрации «аналоговой земли».

Параметр **Op-Amp Bias** определяет уровень наклона переходной характеристики операционных усилителей. Рекомендуется устанавливать этот параметр как LOW в начале разработки проекта. Если требуются более высокие частоты работы, тогда этот параметр может быть установлен как HIGH.

Параметр **A_Buff_Power** определяет мощность аналоговых выходных буферов.

Параметр **SwitchModePump** включает или отключает встроенный регулятор напряжения.

Параметр **TripVoltage [LVD (SMP)]** устанавливает уровень напряжения для определения факта падения напряжения (прерывание Low Voltage Detect), а также уровень напряжения, при падении ниже которого начинает работать блок SMP.

Параметр **LVD Throttle Back** включает или выключает автоматическое уменьшение скорости CPU при обнаружении падения напряжения.

Параметр **Watchdog Enable** включает или отключает сторожевой таймер.

Список глобальных параметров или их названия могут изменяться в зависимости от выбранной серии микросхем, однако эти

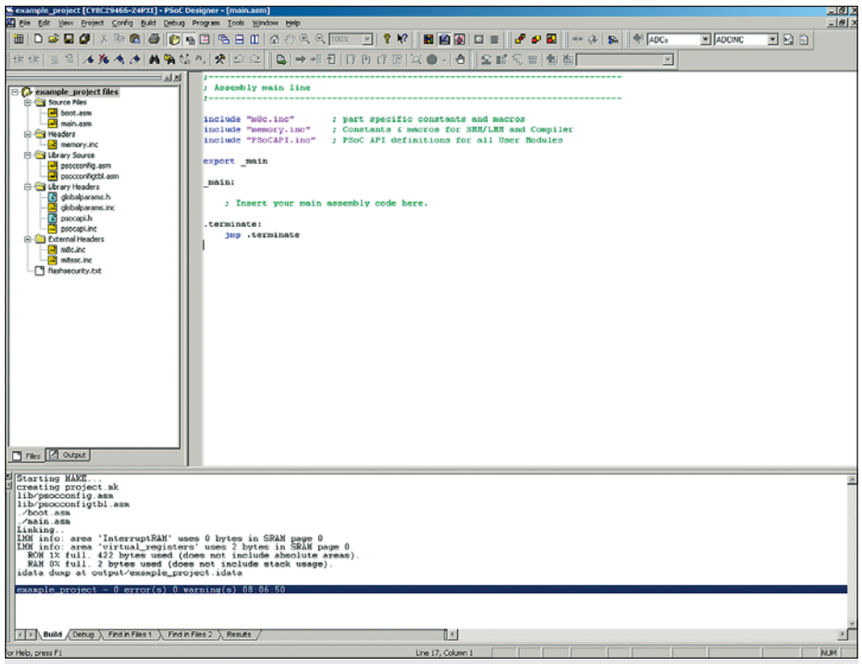

Рис. 7. Редактор исходных текстов программы

изменения незначительны. Выше были приведены глобальные параметры для микросхем 29-й серии, самой мощной по внутреннему ресурсу.

Настройки портов позволяют выбрать режим работы контакта: порт микроконтроллера, глобальный цифровой вход или выход, аналоговый вход или выход, а также один из режимов работы выходного буфера — strong (стандартный режим), strong slow (выход с медленным нарастанием фронтов), pull-up или pull-down (подтяжка к питанию или земле через резистор 5,6 кОм), open-drain low или open-drain high (открытый сток). Кроме того, возможно разрешение прерывания от порта и выбор условия генерации этого прерывания: по фронту сигнала (Rising Edge), по спаду (Falling Edge) или по смене состояния (ChangeFromRead).

Начнем наши эксперименты по освоению PSoC с простейшего проекта — сделаем управление светодиодами на отладочной плате CY3210-MiniEval1 (рис.1), которая входит в комплекты MiniProg и ICE-Cube. На этой плате установлены четыре SMD-светодиода, подключенных к младшим контактам порта P2. Заставим эти светодиоды мигать с различной частотой. Для этого проекта нам потребуется источник частоты, по сигналу от которого мы будем переключать светодиоды. В качестве такого источника возьмем прерывание от SleepTimer, в процедуре обработки которого и будем производить переключение. Установим в параметре SleepTimer период таймера 8 Гц, чтобы частота переключения светодиодов была зрительно видна. Так как отладочная плата получает питание от MiniProg или ICE-Cube, то установим параметр Power Setting [Vcc/SysClk freq] равным 5 В/6 МГц. Значения остальных параметров в данном случае можно оставить по умолчанию, но так как мы не используем аналоговые и цифровые блоки, то хорошим тоном будет настроить параметры на пониженное энергопотребление следующим образом:

```
SysClk*2 Disable = Yes
Analog Power = All Off
Op-Amp Bias = Low
A_Buff_Power = Low
SwitchModePump = Off
Watchdog Enable = Off
```

Для того чтобы микропроцессорное ядро получило возможность управлять портом, необходимо установить в настройках для нужного порта режим StdCPU и выбрать один из режимов работы выходного буфера. В нашем случае выберем режим strong.

После того как сделаны все настройки, необходимо сгенерировать шаблон для программы, а также файлы с процедурами конфигурации. Для этого необходимо нажать на клавишу «Генерация проекта» (Application Generation), либо выбрать пункт меню Config — Generate Application.

После того как сделаны все настройки, переходим в режим редактора программы (Application Editor, рис. 7).

Программная структура проекта

В левой части редактора программ находится окно (рис. 8), отображающее файлы с исходными текстами, входящими в нашу программу.

Рассмотрим подробнее структуру проекта.

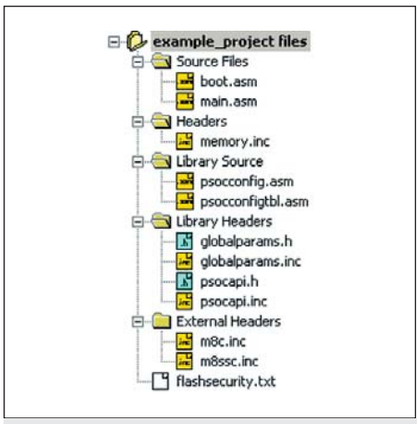

Рис. 8. Программная структура проекта

Ветвь **Source Files** содержит файлы с программным кодом пользователя. По умолчанию в этой ветке создаются два файла — boot.asm и main.asm.

Файл **boot.asm** содержит программный код, исполняемый при включении питания микросхемы. Этот код осуществляет:

- инициализацию векторов прерываний;
- загрузку конфигурации из таблицы во Flash-памяти в конфигурационные регистры;
- инициализацию необходимого окружения, если используется компилятор языка С;
- переход на функцию main.

Файл boot.asm создается при генерации проекта из файла boot.tpl, находящегося в рабочем каталоге проекта. Файл boot.asm создается каждый раз заново при внесении изменений в конфигурацию микросхемы, поэтому если необходимо внести какие-либо изменения в этот файл, например для установки векторов прерываний на необходимые процедуры, то редактировать следует файл boot.tpl.

Файл **main.asm** — основной файл проекта. Этот файл создается на указанном при создании проекта языке и содержит функцию main, которая начинает исполняться сразу же после окончания предварительной инициализации, выполняемой кодом в boot.asm.

Ветвь **Headers** содержит пользовательские заголовочные файлы с необходимыми определениями символьных имен и макроопределений. По умолчанию для микросхем 29-й серии, имеющей страничную организацию ОЗУ, в этой ветви создается файл memory.inc с необходимыми макроопределениями для работы со страничной памятью.

Ветвь **Library Source** содержит автоматически создаваемые файлы с программным кодом для программного интерфейса (API) модулей пользователя, а также файлы с процедурами, осуществляющими загрузку конфигурации (файлы psocconfig.asm и psocconfigtbl.asm, создаваемые по умолчанию).

Ветвь **Library Headers** содержит заголовочные файлы для модулей пользователя. По умолчанию генерируются файл globalparams.inc с определениями глобальных параметров и файл psocapi.inc, который содержит включения заголовочных файлов модулей пользователя, если они присутствуют.

Ветвь **External Headers** содержит файл m8c.inc с определениями текстовых имен регистров, а также макроопределения для установки/снятия битов в некоторых управляющих регистрах.

Файл **flashsecurity.txt** содержит информацию о том, какие блоки Flash-памяти должны быть защищены от записи и считывания. По умолчанию все блоки Flash-памяти имеют защиту как от внешнего считывания (программатором), так и от внешней и внутренней (из программы) перезаписи. Если в проекте предполагается использовать Flash-память для эмуляции EEPROM, то необходимо внести соответствующие исправления в этот файл для того блока или блоков, в которых будет размещаться область, эмулирующая EEPROM.

Для обеспечения какой-либо функциональности мы должны доопределить автоматически сгенерированный шаблон программы. Для реализации нашей задачи переключения светодиодов с разной частотой добавим в файл main.asm процедуру обработки прерывания от SleepTimer — процедуру led_blink (см. листинг), а также изменим файл boot.tpl, добавив в таблицу векторов прерываний переход на данную процедуру:

```
org 64h ;Sleep Timer Interrupt Vector
ljmp led_blink ; переход к обработчику прерывания
reti
```

После внесения изменений в файл boot.tpl, для того чтобы эти исправления вступили в силу и отразились в файле boot.asm, необходимо вновь сгенерировать проект нажатием на клавишу Application Generation. В противном же случае при компиляции программы будет использована текущая версия boot.asm, не содержащая перехода на процедуру обработчика прерывания!

Листинг main.asm:

```
;-----------------------------------------------------------------------------
--
; Assembly main line
;-----------------------------------------------------------------------------
--

include «m8c.inc» ; part specific constants and macros
include «memory.inc» ; Constants & macros for SMM/LMM and
Compiler
include «PSoCAPI.inc» ; PSoC API definitions for all User Modules

export _main
_main:
 ; Insert your main assembly code here.

or reg[INT_MSK0], 0b01000000 ; разрешаем прерывание от
SleepTimer
M8C_EnableGInt ; разрешаем прерывания

.terminate:
 jmp .terminate ; вечный цикл


area bss (RAM,REL,CON) ;
call_cntr: blk 1; объявление переменной-счетчика

area text (ROM,REL)

export led_blink ;экспортируем метку, чтобы она была доступна
из других файлов
led_blink:
push A ;не забываем сохранить аккумулятор в стеке

inc [call_cntr]

;изменяем только младшие 4 бита порта P2
inc [call_cntr]
mov A, reg[PRT2DR]
and [call_cntr], 0x0F
and A, 0xF0
or A, [call_cntr]
mov reg[PRT2DR], A

pop A ;восстанавливаем аккумулятор
reti ;возвращаемся из прерывания
```

Переменная call_cntr является счетчиком вызовов прерывания — она и делит входную частоту 8 Гц, благодаря чему светодиоды загораются с частотами 4 Гц, 2 Гц, 1 Гц, 0,5 Гц.

Контроллер прерываний в PSoC управляется следующими регистрами: INT_CLRx, INT_MSKx, INT_VC, CPU_F.

Регистры INT_CLRx при чтении отображают текущее состояние запросов на прерывание от конкретных источников, а при записи позволяют сбросить какой-либо запрос, или наоборот — его программно установить. Регистры INT_MSKx позволяют разрешить или запретить генерацию прерывания от какого-либо конкретного источника, в то время как младший бит регистра CPU_F отвечает за глобальное разрешение или запрещение прерываний. Регистр INT_VC при чтении возвращает вектор прерывания, которое будет обработано следующим, а при записи позволяет сбросить все запросы на прерывание.

Контроллер прерываний в PSoC подразделяет запросы на прерывание на отложенные и ожидающие обслуживания. Принципиальная разница заключается в том, что отложенный запрос является сигналом от источника прерывания, защелкнутым во входном триггере, но его дальнейшее распространение запрещено маскирующим регистром. В свою очередь, ожидающий обслуживания запрос — это сигнал, прошедший через схему маскирования и поступивший на схему приоритетного кодирования, которая выдает микропроцессорному ядру вектор прерывания, соответствующий запросу с наивысшим приоритетом. Таким образом, если имеет место запрос от источника прерывания, его обработка может начаться либо непосредственно после возникновения (если в данный момент нет более приоритетного запроса), либо при размаскировании отложенного запроса, либо при установке глобального бита разрешения прерываний.

В качестве источников прерываний могут выступать как системные ресурсы ядра (сторожевой таймер, SleepTimer, контроллер I^2C, схема мониторинга напряжения питания, порты ввода-вывода), так и выходы всех цифровых блоков, а также выходы компараторов в аналоговых блоках.

Подробно работа контроллера прерываний рассмотрена в [1] в разделе 5 — Interrupt Controller секции B: PSoC Core.

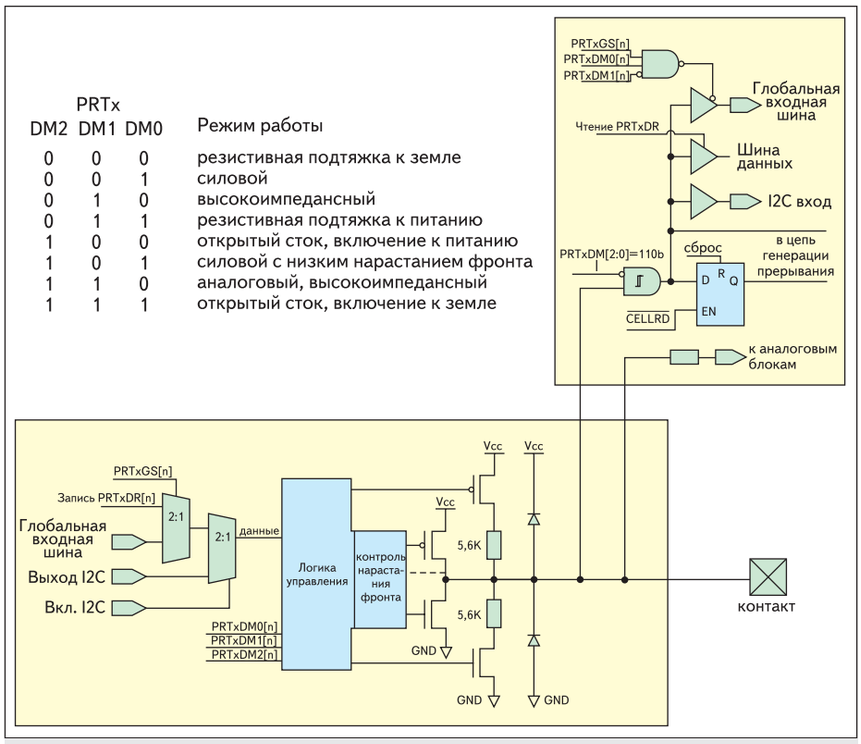

Рис. 9. Схема порта ввода-вывода

В нашем примере мы разрешаем только прерывание от SleepTimer установкой 6-го бита в регистре INT_MSK0 и разрешаем прерывания установкой младшего бита регистра CPU_F с помощью библиотечного макроса M8C_EnableGInt.

Работу портов ввода-вывода контролируют следующие регистры: PRTxDR, PRTxIE, PRTxGS, PRTxDMx, PRTxICx.

Регистры PRTxDR при чтении отображают состояние контактов, а при записи изменяют их состояние в соответствии с записанным значением.

Регистры PRTxGS — регистры выбора режима работы контакта порта: работа как глобальный цифровой вход или выход, либо как порт микроконтроллера (отображает состояние регистра PRTxDR).

Регистры PRTxDMx — регистры выбора режима работы выходного буфера.

Регистры PRTxIE — регистры разрешения генерации прерывания GPIO для каждого конкретного контакта порта.

Регистры PRTxICx — регистры выбора типа прерывания.

Все эти регистры, за исключением PRTxDR, содержат настройки, задаваемые в редакторе структуры микросхемы. Для многих случаев достаточно такой начальной настройки, однако эти регистры доступны и при выполнении программы. Благодаря этому можно, например, перенастраивать режим работы буферов для реализации двунаправленной шины, контакт для работы с цифровыми или аналоговыми блоками и так далее.

При работе с портами следует учитывать две особенности. Первая заключается в том, что для общения с портом используется 8-битный регистр PRTxDR, каждый бит которого отвечает за состояние соответствующего контакта. Поэтому для изменения состояния контактов такими битовыми операциями, как AND, OR или XOR, следует пользоваться с осторожностью. Дело в том, что эти операции при изменении регистра данных сначала производят чтение из него, а затем изменение и запись. В том случае, например, если контакт порта сконфигурирован на работу в режиме с резистивной подтяжкой к «земле» и выходной триггер установлен в 0, но на контакте физически присутствует потенциал логической единицы, то выходной триггер будет установлен в 1 при выполнении битовой операции для изменения какого-либо другого контакта порта.

Вторая особенность заключается в том, что для всех контактов ввода-вывода, которые настроены как источники запроса на прерывание, используется один вход контроллера прерываний (все сигналы с портов объединяются через монтажное «ИЛИ») и, соответственно, один вектор, поэтому в процедуре обработки необходимо определять, какой из контактов вызвал прерывание. Кроме того, из-за использования схемы монтажного «ИЛИ» возможна ситуация, когда один запрос снимается в то время, когда устанавливается другой, поэтому на входе контроллера прерываний не происходит изменения и запрос может оказаться не обслуженным. Во избежание появления такой ситуации следует перед выходом из процедуры обработки прерывания проверять изменение состояния контактов.

Подробно работа портов ввода-вывода рассмотрена [1] в разделе 6 — General Purpose IO (GPIO) секции B: PSoC Core.

После внесения изменений в текст программы выполняем ассемблирование модулей программы и сборку hex-файла прошивки (клавиши Compile/Assemble и Build, либо соответствующие пункты меню Build).

Загрузка программы

Следующий шаг — загрузка программы во Flash-память микросхемы PSoC.

Подсоединяем MiniProg к разъему внутрисхемного программирования на отладочной плате и USB-кабелем соединяем с ПК. Вызываем программатор PSoC Programmer (рис. 10) нажатием на клавишу Program Part

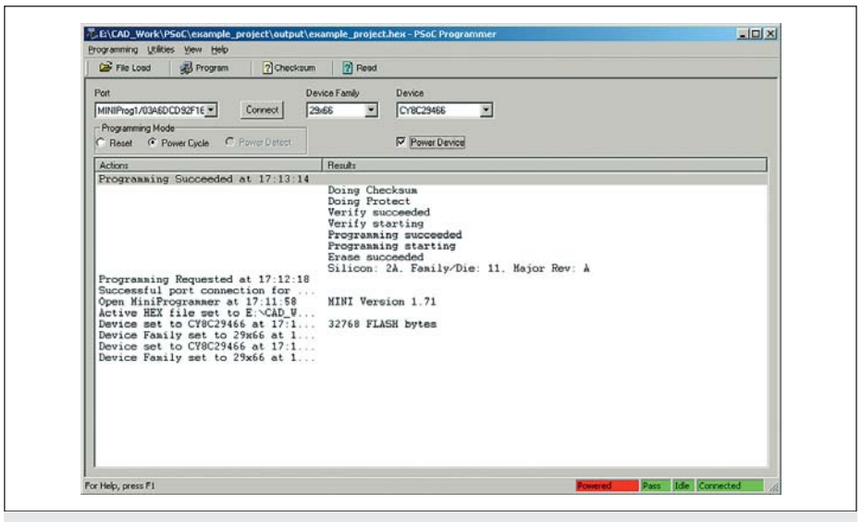

Рис. 10. PSoC Programmer

, либо через меню Start ОС Windows. В том случае, если PSoC Programmer вызывается из оболочки PSoC Designer, то hex-файл текущего проекта оказывается уже открытым, в противном случае его надо открыть самостоятельно. Далее необходимо выбрать порт, к которому подключен MiniProg, серию и тип программируемой микросхемы. Возможны два режима программирования — режим Reset и режим Power Cycle. В Reset для инициализации режима программирования используется контакт XRES, при этом микросхема должна быть запитана от какого-либо внешнего источника. В режиме Power Cycle инициализация программирования происходит при подаче питания на микросхему, при этом питание она получает от USB через MiniProg, либо от ICE-Cube. Так как отладочную плату MiniEval проще всего запитать от USB, то выберем режим программирования Power Cycle. «Прошивка» микросхемы осуществляется нажатием на клавишу Program.

Время загрузки образа программы во Flash-память микросхемы и выполнения проверки зависит от объема Flash-памяти. Для микросхем 29-й серии загрузка программы и проверка занимают примерно одну минуту. После завершения загрузки программы устройство оказывается обесточенным, поэтому для запуска программы необходимо установить флажок Power Device для подачи питания на устройство.

Заключение

Рассмотренный в данной статье пример демонстрирует лишь небольшую часть всех возможностей микросхем PSoC. Далее будет подробно рассмотрена работа цифровых и аналоговых периферийных блоков программируемой системы на кристалле — ключевой особенности, отличающей PSoC от всех других микроконтроллеров.

Литература

1. PSoC™ Mixed Signal Array Technical Reference Manual (TRM) Version 2.00
2. PSoC Designer IDE User Guide
3. Assembly Language User Guide
4. Application Note AN2221 — Global Resources in PSoC™ Designer
5. Кузминский А. Программируемые системы на кристалле компании Cypress Semiconductor. // Компоненты и технологии. 2003. № 2.
6. Емец С. Микроконтроллеры с реконфигурируемой периферией PSoC производства Cypress MicroSystems — восьмиразрядники нового тысячелетия. Компоненты и технологии. // 2004. № 4.