Микроконтроллеры XMEGA — новые возможности проверенного решения

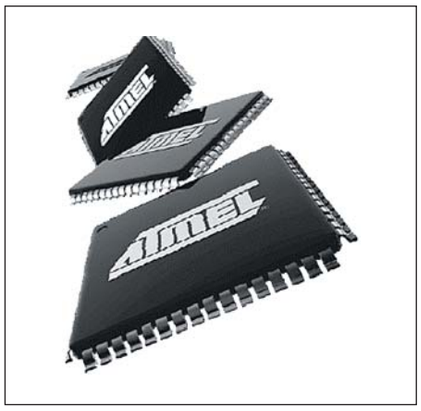
Игорь КРИВЧЕНКО,
к. т. н.
ik@efo.ru

Эта публикация начинает цикл статей, посвященных микроконтроллерам XMEGA фирмы Atmel. В качестве информационной базы при подготовке цикла использовались оригинальные материалы центра Atmel AVR в Норвегии, впервые представленные на техническом тренинге в сентябре 2007 года. Любые публикации о микроконтроллерах XMEGA были запрещены производителем до официального анонса семейства, который состоялся на выставке "Embedded World" в Нюрнберге в конце февраля 2008 года.

Введение

Зачем компании Atmel понадобилось создавать очередное семейство микроконтроллеров? Прежде всего, фирма намеревается укрепить свое положение на мировом рынке в сегменте недорогих встраиваемых микроконтроллеров общего назначения с минимальным энергопотреблением и высокой производительностью. И, конечно, Atmel хочет добиться активного расширения потенциального рынка приложений для новых микроконтроллеров. Начиная с 1995 года, компания борется за звание № 1 в мире по производству современных универсальных Flash-микроконтроллеров для встраиваемых применений, которые лидируют в удельном энергопотреблении, степени интеграции узлов на кристалле и производительности периферийных модулей. По задумке Atmel, с помощью XMEGA микроконтроллеры Atmel должны занять новую качественную нишу на мировом рынке 8-разрядных микроконтроллеров.

Что же такое XMEGA? Прежде всего — новое микроконтроллерное семейство. Но в своей основе это хорошо знакомые, популярные и признанные AVR. Микроконтроллеры XMEGA будут выпускаться в удобных для пайки современных корпусах QFP с количеством выводов от 44 до 100, иметь объем встроенной Flash-памяти от 16 до 1024 килобайт, работать с производительностью до 32 MIPS. Их серийное производство планируется начать в середине 2008 года.



XMEGA — это AVR! По задумке разработчиков из исследовательского центра AVR в Норвегии, для освоения более продвинутого продукта следующего поколения разработчик не должен изучать что-то совсем новое или подвергать уже однажды написанный код глубоким переделкам. Идея «XMEGA = AVR» означает, прежде всего, программную совместимость с популярными «классическими» семействами AVR — tiny и mega. Сразу отметим, что по цоколевке и расположению выводов на корпусах XMEGA не совместимы с микроконтроллерами tinyAVR и megaAVR.

Новые микроконтроллеры выпускаются для работы в индустриальном температурном диапазоне от –40 до +85 °C, их рабочее напряжение составляет от 1,8 до 3,6 В. Подобно прочим AVR, одна и та же версия кристалла будет представлена в двух исполнениях:

- с напряжением питания 1,8–2,7 В, тактовыми частотами до 12 МГц и с линиями ввода/вывода, которые устойчивы к уровню напряжения 3,3 В;
- с напряжением питания 2,7–3,6 В и с тактовыми частотами до 32 МГц.

В дальнейшем, по мере развития семейства XMEGA, компания Atmel планирует выпускать их для работы в автомобильном температурном диапазоне –40…+125 °C, а также сделать линии ввода/вывода в обеих группах устойчивыми к уровню напряжения 5 В.

С точки зрения повышения степени интеграции и «начинки» кристалла, компания Atmel также ввела ряд существенных изменений. Основная идея всех описанных ниже новшеств — радикально уменьшить число необходимых для обеспечения функционирования всего кристалла внешних компонентов и сократить площадь общего решения на плате приложения.

Особенности нового поколения AVR — XMEGA:

- Пониженное энергопотребление:
  - 2 мкА при работающем сторожевом таймере, блоке контроля питания и часах реального времени (RTC);
  - технология picoPower второго поколения;
  - новый сторожевой таймер (WDT);
  - новый блок контроля питания (Brown-out Detection, или BOD).
- Увеличение производительности:
  - контроллер прямого доступа к памяти (DMA);
  - многоуровневый программируемый контроллер прерываний (PMIC);
  - система обработки событий (Event System);
  - тактовые частоты 32 МГц при 2,7 В и 12 МГц при 1,8 В.
- Расширенный набор периферии:
  - 12-разрядный АЦП с частотой преобразования до 2 Msps;
  - 12-разрядный ЦАП;
  - многофункциональные 16-разрядные таймеры-счетчики со скоростным ШИМ;
  - многочисленные коммуникационные модули.

Средства поддержки разработок для XMEGA

Как уже отмечалось, XMEGA — это AVR. Поэтому основные средства поддержки разработок, уже хорошо знакомые российским разработчикам, могут использоваться и для работы с XMEGA. Это внутрисхемный про-

грамматор AVR ISP2, интегрированная среда AVR Studio (версии 4.14 и выше), широко используемый компилятор фирмы IAR Systems. К сожалению, популярный стартовый набор STK500 не может быть использован для работы с XMEGA. Вместо него компания Atmel разработала и выпустила новый стартовый набор STK600, который будет работать со всеми микроконтроллерами AVR, включая XMEGA. Как обычно, поддерживается программирование и отладка микроконтроллеров по JTAG, для этой цели используется популярный эмулятор JTAGICE2. Отметим, что для XMEGA подойдут только эмуляторы ревизии «B» (выпускаются с февраля 2007 года), которые имеют серийный номер, начинающийся с «B», и на материнской плате которых расположен дополнительный зеленый светодиод. Дело в том, что только такие эмуляторы поддерживают двухпроводной интерфейс PDI (Program and Debug Interface), обеспечивающий отладку и программирование кристаллов XMEGA внутрисхемно.

И программирование, и отладка микроконтроллеров XMEGA могут осуществляться через два физических интерфейса. Основной интерфейс — PDI, который является собственной разработкой Atmel для внешнего программирования и внутрисхемной отладки микроконтроллеров. PDI поддерживает высокоскоростное программирование всех областей энергонезависимой памяти на кристалле XMEGA: Flash, EEPOM, Fuses, Lock-биты и User Signature Row. Он использует линию Reset для тактового сигнала (PDI_CLK) и специальную линию Test для ввода/вывода данных (PDI_DATA). Стандартный JTAG-интерфейс также присутствует в большинстве кристаллов XMEGA и может быть использован для программирования или отладки через 4-проводной интерфейс JTAG. По умолчанию все обращения к блоку отладки предполагают доступ к физическому интерфейсу PDI.

Планируемые к выпуску семейства микроконтроллеров XMEGA

Первым в 2008 году корпорация Atmel планирует выпустить микроконтроллер старшего подсемейства A1 в корпусе TQFP100 — ATxmega128A1 с объемом Flash-памяти 128 кбайт. В дальнейшем планируется последовательное расширение номенклатуры кристаллов XMEGA с разными размерами массивов памяти, количеством и ассортиментом периферийных блоков, сниженной стоимостью микроконтроллеров и более дешевыми корпусами с количеством выводов 64 и 44. Это будут подсемейства A3 и A4 соответственно. При этом внутри семейства XMEGA будет использоваться одна и та же таблица адресов периферии для обеспечения полной программной совместимости при переходе к новым кристаллам. Планируемый к выпуску состав подсемейств XMEGA показан на рис. 1.

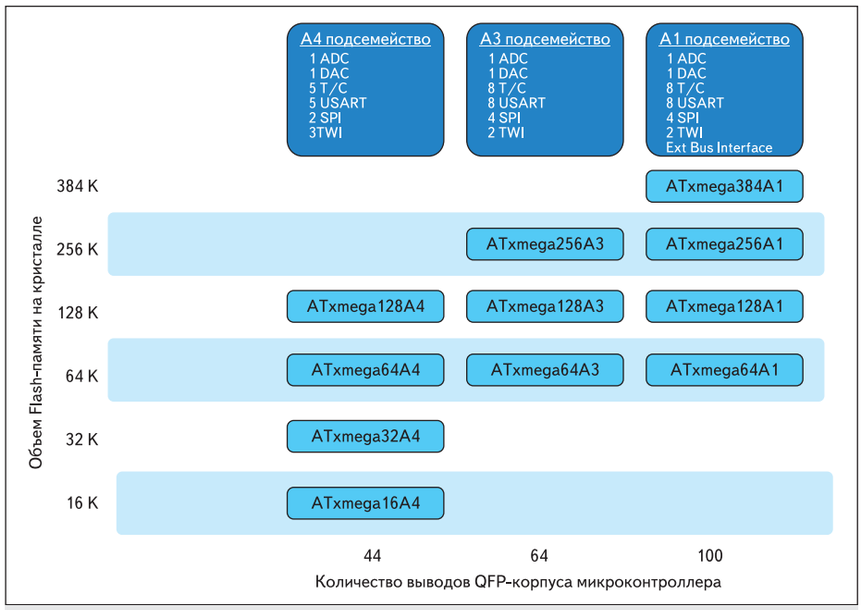


Рис. 1. Планируемый состав семейства XMEGA

Новые особенности XMEGA

Наш экскурс начнем с заявленных планов Atmel по радикальному снижению энергопотребления кристаллов нового микроконтроллерного семейства, так как данный вопрос в современном электронном мире является одним из наиболее приоритетных. Для микроконтроллеров XMEGA будет использоваться технология picoPower второго поколения (о picoPower мы уже писали ранее [1]). Что же входит у Atmel в понятие «второе поколение»?

Во-первых, благодаря ряду улучшений в технологическом процессе изготовления кристаллов значительно снижены токи утечки ячеек памяти Flash и EEPROM в режиме Power Down. Во-вторых, новые аппаратные модули — DMA и особенно Event System — значительно снизят время пребывания центрального процессора в активном режиме, что уменьшит суммарное энергопотребление. И, наконец, оптимизирован ряд сервисных узлов микроконтроллера, которые по «долгу службы» должны постоянно находиться в активном режиме, что также означает непрерывное потребление энергии, пусть даже и небольшое.

- Сторожевой таймер WDT тактируется от отдельного внутреннего генератора с ультранизким энергопотреблением. Кроме того, WDT сможет работать в так называемом «оконном» режиме, что также сокращает потребление энергии (подробнее работу сторожевого таймера мы рассмотрим далее).
- Улучшена структура блока контроля питания: повышена точность определения напряжения, и, кроме того, теперь BOD будет сэмплироваться, то есть отслеживать уровень питающего напряжения периодически. При этом период опроса BOD может устанавливать разработчик, чтобы достигнуть оптимального для конечного приложения компромисса между временем реакции узла BOD и его суммарным энергопотреблением.
- Усовершенствованы часы реального времени. Теперь в состав RTC входит 16-битный таймер, что позволяет обеспечивать «засыпание» микроконтроллера на 18 часов, причем периодичность «пробуждения» также может программироваться.

Все описанные новшества благотворно сказались на энергопотреблении кристаллов XMEGA, особенно в «спящих» режимах. Так, в режиме Power Save при включенном узле RTC, разрешенном BOD и работающем WDT ток потребления микроконтроллера составляет не более 2 мкА.

Система событий, многоуровневый контроллер прерываний, контроллер прямого доступа к памяти

Два новых для AVR периферийных блока — система событий (Event System) и многоуровневый программируемый контроллер прерываний (PMIC) — будут подробно рассмотрены во второй части цикла статей. Сейчас мы лишь отметим их основные особенности.

Система событий XMEGA предназначена для разгрузки центрального процессора. Используя сложную структуру матрицы соединений, периферийные узлы, процессорное ядро и контроллер DMA могут непосредственно обмениваться между собой как служебной, так и командной информацией,

а также передавать и принимать данные. В матрице шин организовано до 8 параллельных каналов передачи информации. Допускается независимая работа центрального процессора и DMA. Поддерживается предварительная цифровая фильтрация данных.

Главная изюминка Event System — гибкость в предоставлении прав каждому из подключенных к ней периферийных модулей. Эти модули могут определять, по какому из внешних воздействий может генерироваться событие (точнее, они заранее программируются пользователем), например, при перепаде напряжения на выводе микроконтроллера, при переполнении таймера, завершении цикла преобразования АЦП и т. д. Периферийные модули могут программироваться и на то, каким образом интерпретировать поступающее к ним событие — инкрементировать таймер, выставить выходной сигнал на выводе микросхемы, начать преобразование АЦП и т. д. Система событий разгружает систему прерываний XMEGA в основном за счет уменьшения количества формируемых запросов на прерывание. При этом значительно повышается надежность обработки данных и принятия решений: критичные ко времени или к стабильности выполнения функции теперь становятся более прогнозируемыми, а значит, увеличивается надежность работы всей системы в целом. Также снижается загрузка центрального процессора для обработки различных «несистемных» прерываний, генерируемых периферийными узлами и модулями, в том числе и какими-то внешними событиями.

Впервые в истории AVR компания Atmel предлагает разветвленную систему прерываний для XMEGA. Для PMIC заявлена поддержка 4-уровневого процесса обслуживания — всем запросам на прерывание могут быть присвоены определенные уровни приоритета для обработки. Как всегда, оставлены немаскируемые прерывания, которые жестко связаны с рядом системных функций процессора и имеют наивысший уровень приоритета. В дополнение к нему теперь имеются еще 3 уровня: High (высокий), Medium (средний) и Low (низкий). Для каждого источника запроса на прерывание его приоритет может назначаться из этих трех уровней. При поступлении запроса на обслуживание прерывания с более высоким уровнем приоритета процесс обработки прерывания с более низким уровнем приостанавливается, и процессор начинает обрабатывать более значимый запрос. Запросы на прерывание, которым присвоен статус High, обслуживаются немедленно после поступления, даже если центральный процессор сильно загружен.

Для всех прерываний, имеющих уровень LOW приоритета на обслуживание, предусмотрена специальная процедура диспетчеризации, чтобы не пропал ни один из запросов, и все они были обслужены, пусть даже и с задержкой. Эта процедура носит название Round Robin — циклический алгоритм диспетчеризации, при котором все процессы активизируются в фиксированном циклическом порядке. Иногда этот алгоритм еще называют «карусельной диспетчеризацией».

В микроконтроллеры XMEGA добавлен еще один полезный периферийный модуль — контроллер прямого доступа к памяти (DMA). Его наличие существенно увеличивает производительность кристалла, особенно для задач реального времени. Использование DMA сокращает время обмена данными между памятью и периферией, между отдельными областями памяти, а также между различными периферийными блоками. Центральное процессорное ядро не принимает участия в процессе передачи данных, вся обработка и сопровождение потока лежит на контроллере прямого доступа к памяти.

Контроллер DMA у XMEGA имеет 4 канала, причем приоритет обслуживания и распределения между каналами программируется. Он поддерживает пакетную передачу данных с переменной длиной пакета — 1, 2, 4 или 8 байт в посылке. Общая длина передачи может варьироваться от 1 кбайт до 64 кбайт с возможностью повторения. Допускается формирование запроса на прерывание по окончании процесса передачи данных. Каждый канал DMA может работать как на прием, так и на передачу. Контроллер DMA может получать доступ к внутренней шине данных микроконтроллера только в том случае, если она свободна и центральный процессор ничего на ней не делает. Приоритет доступа к внутренней шине данных распределяет специальный арбитражный узел, аппаратно отнесенный к подсистеме управления памятью данных микроконтроллера. Подробнее работу контроллера DMA рассмотрим в третьей части цикла статей.

Память

На кристалле XMEGA традиционно для AVR присутствует память Flash, EEPROM и SRAM. Все области памяти имеют линейный диапазон адресов и занимают единое пространство на кристалле. Типичная структура, состав и размеры встроенных массивов памяти для семейства XMEGA приведены в таблице.

Flash-память состоит из области программы (Application Section) и области загрузчика (Boot Loader Section). Application Section у XMEGA содержит отдельную область для хранения данных со своими битами защиты, которая называется Application Table и используется в качестве энергонезависимого хранилища данных. Основная особенность Application Table заключается в том, что она оптимизирована для представления данных в виде таблиц, чего раньше не было. По задумке Atmel, назначение этой области во Flash-памяти заключается в резервном хранении массивов данных, которые находятся в EEPROM (у всех AVR имеются определенные проблемы с гарантированным сохранением данных в EEPROM в условиях нестабильного, плавающего напряжения питания). Область Application Table можно использовать для безопасной эмуляции EEPROM, сохраняя в ней данные, предназначенные для EEPROM. Если такой потребности нет, то в Application Table можно записывать программный код, увеличивая размер основной области программы.

Таблица. Структура, состав и размеры встроенных массивов памяти для семейства XMEGA

Flash, кбайт	SRAM, кбайт	EEPROM, кбайт
16+4	2	512
32+4	4	512
64+4	4	1
128+8	8	2
256+8	16	2

В Application Section добавлен еще один сегмент, который есть только у XMEGA. Сегмент состоит из двух секций, которые не могут быть стерты посредством программатора и командой Chip Erase. Одна из секций имеет название Calibration and Signature Row и хранит записанные на фабрике значения калибровочных констант (для генераторов, например) и уникальный сериальный номер (Serial Number) кристалла. В нем содержится информация о номере партии микроконтроллеров (LOT ID), номер кремниевой пластины и даже координаты X–Y положения данного кристалла на пластине. Секция доступна для чтения как с помощью программатора, так и из приложения. Вторая секция не имеет специального названия и предназначена для хранения пользовательских данных. Она доступна для чтения-записи как с помощью программатора, так и из программы по специальным командам, но команда Chip Erase не будет оказывать на содержимое этой секции никакого воздействия. Команды SPM при самопрограммировании кристалла тоже не будут оказывать воздействия на калибровочные области Flash-памяти.

Важный вопрос — сохранность данных, размещенных в массиве Flash-памяти. Технологические изменения коснулись и этой сферы. По информации Atmel, срок гарантированного сохранения данных при хранении и работе кристалла при 25 °C составляет 100 лет, а при 85 °C снижается до 25 лет. Естественно, что эти значения рассчитаны на основании измерений процессов старения Flash-ячеек на кристаллах.

Память данных в XMEGA организована в виде несегментированного блока с линейной адресацией (стартовый адрес 0x000000), включая область для хранения энергонезависимых данных EEPROM. Таким образом, логическое пространство памяти данных XMEGA едино. Физически же область данных состоит из области регистров ввода/вывода (I/O Memory, до 4 кбайт), включая 16 регистров общего на-

значения, внутренней EEPROM (до 4 кбайт), внутренней SRAM и области внешней памяти External Memory (если есть). Отметим, что первые три области памяти будут иметь единую адресную структуру для всех микроконтроллеров XMEGA (одинаковые начальные адреса).

Наибольший интерес в памяти XMEGA с точки зрения обновлений представляет именно блок EEPROM. Во-первых, доступ по записи и чтению к EEPROM теперь осуществляется как побайтно, так и постранично. Во-вторых, адресация EEPROM может осуществляться двояким образом: стандартно (установка по умолчанию) в рамках выделенного адресного пространства с помощью специального набора регистров, либо непосредственной адресацией в коде команды к любой ячейке EEPROM в выделенном сегменте общего адресного пространства памяти данных. И если раньше для чтения-записи необходимо было выполнять определенную и длинную последовательность команд (от 7 до 9 инструкций), то сейчас эти операции выполняются одной командой, как будто данные непосредственно пишутся в ячейку памяти. Помимо удобства в работе, это приводит к уменьшению размера кода и снижению энергопотребления кристалла. Конечно, при этом лишь эмулируется поведение EEPROM как SRAM, потому что физически процессы программирования ячейки EEPROM занимают значительное время (на самом деле данные лишь помещаются в промежуточный буфер, а все остальные процессы протекают скрыто для программиста).

Для организации работы с внешней памятью (External Memory) новые микроконтроллеры имеют до 4 специализированных портов ввода/вывода. Это позволяет XMEGA работать с микросхемами внешней памяти типа SRAM, SDRAM, жидкокристаллическими ЖК-дисплеями и другими внешними устройствами, которые имеют похожий интерфейс доступа. Обслуживанием взаимодействия микроконтроллера с внешней памятью любого типа традиционно занимается контроллер интерфейса внешней шины (EBI). Массив адресов External Memory (если она есть) всегда начинается сразу после блока внутренней SRAM и заканчивается адресом 0xFFFFFF. Для внешней памяти типа SRAM адресное пространство составляет 16 Мбайт. Для ряда микроконтроллеров допускается мультиплексирование шин адреса и данных при организации внешнего интерфейса. Для внешней памяти типа битовой SDRAM адресное пространство составляет 128 Мбит, при этом поддерживаются режимы работы с 4-битными и 8-битными полями данных.

Тактирование

Микроконтроллеры XMEGA могут работать в двух частотных диапазонах: 0–12 МГц при напряжении питания от 1,8 до 2,7 В и 0–32 МГц при напряжении питания от 2,7 до 3,6 В. Они имеют развитую систему тактирования, принцип построения которой показан на рис. 2.

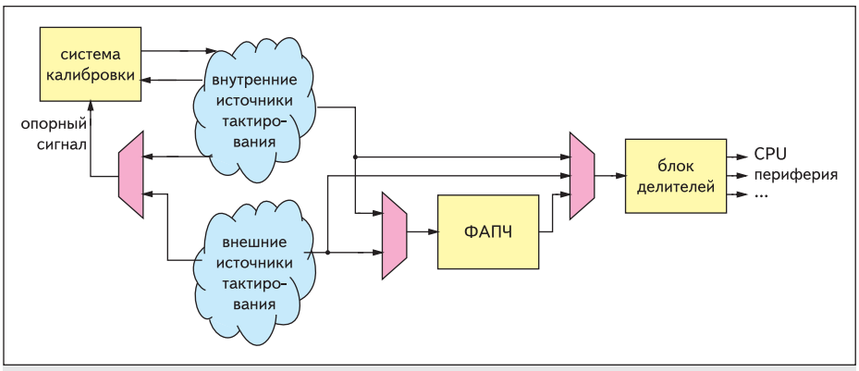


Рис. 2. Структура системы тактирования XMEGA

Система тактирования объединяет внутренние и внешние генераторы. Для получения широкого спектра тактовых частот могут использоваться внутренний узел ФАПЧ (PLL) и делители частоты. Доступна опция автоматической калибровки внутренних генераторов во время работы программы. Может быть разрешена работа узла, который определяет пропадание внешнего тактового сигнала. При этом разрешается специальное немаскируемое прерывание, и микроконтроллер переключается на тактирование от внутреннего источника на кристалле.

После сброса XMEGA всегда начинает работать от внутреннего генератора частотой 2 МГц. При этом не используются никакие делители. Для управления временем ожидания стабилизации работы системы тактирования используются два fuse-бита (SUT0 и SUT1), которые устанавливают период ожидания как 1, 4 или 64 мс. Тактирование ядра начинается еще через 6 тактов генератора.

При нормальном функционировании микроконтроллера XMEGA источник системного тактового сигнала и установки предварительных делителей могут изменяться из программы пользователя в любое время. В составе системы тактирования есть также специальный регистр LOCK, с помощью которого установки системы тактирования могут быть зафиксированы из программы пользователя. Когда бит LOCK в этом регистре установлен, содержимое регистров CTRL (отвечает за выбор источника сигнала для системной тактовой частоты) и PSCTRL (отвечает за коэффициент предварительного деления системной тактовой частоты) уже не может быть изменено вплоть до следующего сброса микроконтроллера. Этот важный бит LOCK в свою очередь защищен системным механизмом Configuration Change Protection (CCP), аналогично операциям SPM и LPM. Бит LOCK будет автоматически сброшен только после сброса микроконтроллера.

XMEGA содержит четыре внутренних источника тактовой частоты:

1. ULP32K: RC-генератор частотой 32 кГц с ультранизким энергопотреблением. Он предназначен для обеспечения системных функций микроконтроллера и не может использоваться в качестве источника основной тактовой частоты. ULP32K отвечает за тактирование сторожевого таймера, часов реального времени, формирование задержки при старте микроконтроллера. Точность частоты этого генератора во всем диапазоне температур и напряжений питания колеблется в пределах 30% и определяется при изготовлении на фабрике.
2. RC32K: калиброванный RC-генератор частотой 32 кГц, который может использоваться в качестве источника системной тактовой частоты. RC32K также является источником опорного сигнала для процедуры калибровки и для часов реального времени. Значение частоты этого генератора программируется при изготовлении кристаллов, калибровочные значения записываются в калибровочные регистры. Точность фабричной установки частоты составляет 1%.
3. RC2M: калиброванный RC-генератор частотой 2 МГц, снабженный ФАПЧ (PLL). Именно он по умолчанию используется в качестве источника системной тактовой частоты при запуске микроконтроллера. Точность фабричной установки частоты составляет 2%, при этом во время работы микроконтроллера может осуществляться автоматическая калибровка.
4. R32M: калиброванный кольцевой генератор частотой 32 МГц, которая изменяется в зависимости от условий производства микроконтроллеров, рабочей температуры и напряжения питания. Точность калибровки при изготовлении кристаллов составляет 2%. Генератор, снабженный PLL, может использоваться в качестве источника системной тактовой частоты. Во время работы микроконтроллера может осуществляться автоматическая калибровка.

Помимо вышеперечисленных генераторов для XMEGA могут традиционно использоваться и другие источники тактирования —

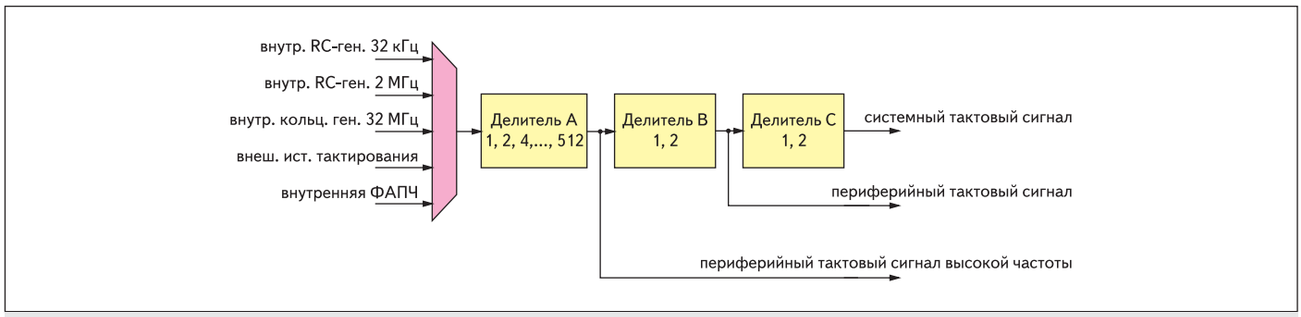


Рис. 3. Структура модуля формирования тактовых сигналов

внутренний генератор с внешним частотозадающим элементом (кварцевым резонатором) или внешний генератор. Организация работы XMEGA с такими источниками идентична «старшим» кристаллам megaAVR и здесь рассматриваться не будет.

Встроенный узел PLL предназначен для формирования высокой системной тактовой частоты, которая получается умножением входной частоты на программируемый коэффициент, значение которого лежит в пределах от 1 до 31. В качестве источников входной частоты для PLL могут использоваться RC2M, R32M, генератор с внешним кварцем 0,4–16 МГц или внешний тактовый сигнал. Минимальное значение частоты на входе не должно быть ниже 440 кГц. Узел PLL может формировать выходную частоту вплоть до 200 МГц, но это пока не имеет смысла и зарезервировано для будущих разработок, так как максимальная внутренняя периферийная частота таймеров составляет 128 МГц.

Для запуска PLL необходимо провести строго определенную последовательность действий. После этого специальное аппаратное обеспечение следит, чтобы конфигурация PLL не могла быть изменена случайным образом. Если все-таки необходимо внести изменения, то сначала следует остановить PLL, а чтобы новые параметры конфигурации PLL вступили в силу, необходимо сделать системный сброс микроконтроллера.

Источник основной тактовой частоты XMEGA выбирается программно и может быть переключен во время работы микроконтроллера. Встроенная аппаратная логика предотвращает небезопасное переключение системной частоты. Так, невозможно выбрать нестабильный или запрещенный генератор, нельзя запретить источник, уже генерирующий системную тактовую частоту. Доступный по чтению бит статуса в регистре STATUS для каждого генератора позволяет оперативно, из программы, проверить готовность генератора к работе.

Полученный системный тактовый сигнал сначала проходит блок предварительных делителей (рис. 3) с суммарным коэффициентом деления в диапазоне от 1 до 2048, и только потом подается на ядро и периферийные блоки. Гарантируется, что все выходные сигналы производных частот всегда имеют четкое фазовое соответствие с входным сигналом, и что при работе системы не возникает импульсных выбросов или промежуточных частот в моменты программного изменения установок делителей. Новые параметры всегда обновляются по фронту сигнала с наименьшей из генерируемых частот.

Для диагностики сбоев или отказов в работе внешних источников тактового сигнала на кристалле XMEGA есть встроенная аппаратная схема мониторинга. По умолчанию эта опция отключена и может быть разрешена со стороны прикладной программы. Отметим, что снова выключить схему мониторинга можно только путем общего сброса микроконтроллера. Если по каким-то причинам системный тактовый сигнал от внешнего источника пропадает, то микроконтроллер принудительно и безусловно переключается на внутренний генератор RC2M (2 МГц), сбрасывая при этом содержимое регистров CTRL генератора и системной тактовой частоты к значениям по умолчанию. Генерируется флаг соответствующего немаскируемого прерывания, которое должно быть обязательно обслужено, даже если внешний источник не использовался в качестве системной тактовой частоты. В этом случае внешний генератор автоматически блокируется, в то время как системный тактовый сигнал продолжает поступать на микроконтроллер. Мониторинг автоматически запрещается во всех режимах пониженного энергопотребления (внешний источник тактирования останавливается), а при выходе из «спящего» режима — автоматически восстанавливается. Установки схемы мониторинга защищены системным механизмом Configuration Change Protection (CCP).

Отметим, что если используется внешний источник тактирования и аппаратный мониторинг включен, то частота этого внешнего источника должна быть не ниже 32 кГц, иначе будет вырабатываться сигнал ошибки (отсутствие внешнего тактирования).

Система тактирования микроконтроллеров XMEGA содержит узел внутренней калибровки. Две встроенные цифровые ФАПЧ (DFLL) могут использоваться для повышения точности внутренних генераторов на RC2M (2 МГц) и R32M (32 МГц). Блок DFLL сравнивает сигнал генератора с более точной частотой для выполнения автоматической калибровки генератора во время работы программы. В качестве таких опорных сигналов могут выступать внутренний калиброванный генератор RC32K (32 кГц), либо генератор RTC, стабилизированный отдельным внешним часовым кварцем. Работа DFLL будет остановлена при переходе в режимы пониженного энергопотребления, которые предполагают остановку генераторов. После «пробуждения» DFLL возобновит работу с калибровочным значением на момент «засыпания». Чтобы вернуть DFLL калибровочные значения по умолчанию, необходимо запретить ее работу перед входом в «спящий» режим и разрешить после выхода из него.

Внутренняя калибровка не обслуживает генератор RC32K. Соответствующий регистр RC32KCAL используется для его оперативной калибровки. После сброса кристалла значение, определенное на фабрике, записывается в этот регистр из секции памяти Calibration and Signature Row. Содержимое этого регистра может быть также изменено из программы пользователя во время работы.

Схема разводки системы тактирования на кристалле XMEGA достаточно сложна, поэтому проще адресовать читателя к описанию на микроконтроллер. Заметим, что не все тактовые сигналы требуются одновременно для работы XMEGA — тактирование для ядра и периферии может быть остановлено с помощью режимов пониженного энергопотребления и регистров снижения мощности. Подробнее об особенностях работы и установках системы тактирования XMEGA можно прочитать в описании на микроконтроллер и в Application Note AVR1003.

Блок обработки аналоговых сигналов

На кристаллы XMEGA интегрированы быстродействующий АЦП последовательных приближений, быстрый ЦАП и развитый узел аналоговых компараторов. Входы и выходы аналоговых блоков выведены на линии портов ввода/вывода PORTA и PORTB. Микроконтроллеры XMEGA могут иметь

1 или 2 восьмиканальных аналоговых порта, или не иметь аналоговых функций вообще (для младших версий кристаллов). Каждый аналоговый порт может иметь 1 блок АЦП, 1 блок ЦАП и 2 аналоговых компаратора. К аналоговым блокам также относятся встроенные в микроконтроллер датчик температуры и точный ИОН, выходное напряжение которого является источником для двух других узлов формирования опорного напряжения внутри кристалла — 1 и 3 В. Точность встроенного ИОН калибруется на фабрике не хуже 1% во всем диапазоне рабочих температур и напряжений питания.

АЦП имеет разрядность 12 бит, скорость преобразования до 2 миллионов выборок в секунду, встроенную калибровку. Запуск АЦП на преобразование может быть реализован по внешнему событию, результаты преобразования могут передаваться через DMA без участия центрального процессора. Входные цепи построены достаточно гибко, допускается как несимметричное, так и дифференциальное включение. На кристалле XMEGA также есть дополнительный промежуточный узел усиления входного аналогового сигнала, который позволяет значительно расширить динамический диапазон входных напряжений преобразования в случае, когда входы АЦП включены в дифференциальном режиме. Коэффициент усиления этого узла может программироваться и принимать ряд целых значений 1, 2, 4, 8, 16, 32 или 64. Встроенный механизм калибровки, который запускается из программы пользователя, позволяет уменьшить аддитивную и мультипликативную составляющие погрешности в конечном результате преобразования.

Наибольший интерес вызывает использование в АЦП так называемых виртуальных каналов. Они используют один и тот же аппаратный узел АЦП для осуществления преобразований, но благодаря специальной архитектуре одновременно и независимо друг от друга может выполняться до четырех преобразований. Результаты преобразования сохраняются в независимых регистрах. Данная опция (виртуальные каналы) может помочь снизить сложность программного обеспечения, так как различные программные модули могут инициировать начало преобразования и считывать результаты независимо друг от друга. Подробнее работу АЦП и виртуальных каналов рассмотрим в третьей части цикла статей.

Разрядность цифро-аналогового преобразователя у XMEGA составляет 12 бит, скорость преобразования — до 1 Мбит/с. ЦАП тактируется сигналом периферийной тактовой частоты, запуск на преобразование могут осуществлять различные периферийные модули, подключенные к системе событий. ЦАП может работать в режиме пониженного потребления энергии — это означает, что он может выключаться между последовательными циклами преобразования. В этом режиме время преобразования может быть больше заявленного, так как при старте каждого нового преобразования будет требоваться некоторое время на приведение ЦАП в состояние готовности. Доступна также опция формирования нулевого выходного напряжения. Как правило, все цифро-аналоговые преобразователи не отличаются достаточной линейностью, когда формируемое выходное напряжение приближается к нулю. Встроенные возможности подстройки нуля и коэффициента усиления позволяют программисту калибровать и усиление, и смещение в ЦАП. Для этого на кристалле XMEGA в структуре ЦАП есть два регистра, в которые можно записывать 7-битные калибровочные данные для обоих параметров. Для достижения наилучших результатов при калибровке рекомендуется использовать одни и те же входные «параметры»: источник опорного напряжения, выходной канал, время преобразования и интервал обновления, что используется при нормальной работе ЦАП. Предел формируемого выходного напряжения определяется выбором источника опорного напряжения (AVCC, встроенный ИОН или внешний ИОН).

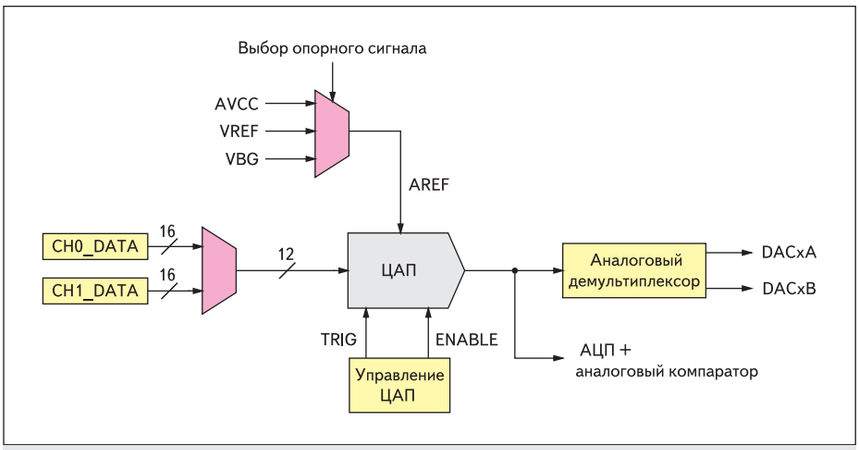


Рис. 4. Блок-схема ЦАП

Начало преобразований ЦАП инициируется либо записью (нового) цифрового кода во входные регистры ЦАП, либо поступлением события из Event System. Информацию для преобразования во входные регистры можно записывать как из программы, так и через контроллер DMA.

Выход с ЦАП может быть выполнен двумя способами. В первом случае реализуется один постоянный линейный выход на выводе микроконтроллера. Во втором — формируются два независимых выхода на двух выводах микроконтроллера. Это сделано с помощью интегрированного в ЦАП устройства выборки-хранения, которое работает в качестве аналогового демультиплексора. Результат предыдущего преобразования ЦАП удерживается на одном выводе XMEGA, в то время как ЦАП осуществляет текущее преобразование. По окончании преобразования устройство выборки-хранения переключает выход ЦАП на второй вывод микроконтроллера. В результате два образующихся «канала» ЦАП могут работать независимо и выдавать два аналоговых сигнала, различающихся как по амплитуде, так и по частоте. В XMEGA предусмотрены индивидуальные регистры для записи входных данных на преобразование для обоих «каналов».

Линейный выход ЦАП можно подключать внутри кристалла к другим периферийным узлам, например, к входу встроенного АЦП или к входу встроенного аналогового компаратора. Выходы от устройства выборки-хранения не могут быть подключены внутри кристалла. Блок-схема ЦАП приведена на рис. 4.

Для организации корректной работы устройства выборки-хранения в ЦАП необходимо помнить про ряд временных ограничений, которые обязательно должны приниматься во внимание. Эти ограничения связаны с тактированием ЦАП от источника периферийной частоты и могут влиять на периоды заряда-разряда устройства выборки-хранения:

- Время, необходимое для приведения ЦАП в состояние готовности, определяется как интервал между моментом окончания преобразования в канале A и стартом преобразования в канале B. Этот интервал не должен быть меньше 1 мкс.
- Время обновления результата ЦАП определяется как временной интервал между последовательными преобразованиями в одном и том же канале ЦАП. Это время не должно быть больше 30 мкс.

Если не учитывать данные ограничения, то точность преобразования может ухудшиться. Отметим, что все сказанное применимо только к режиму работы ЦАП на два выхода через встроенное устройство выборки-хранения.

Модуль аналоговых компараторов (АС) на кристалле XMEGA состоит из двух компараторов и блока мультиплексоров. Блок-схема АС приведена на рис. 5. АС может быть скон-

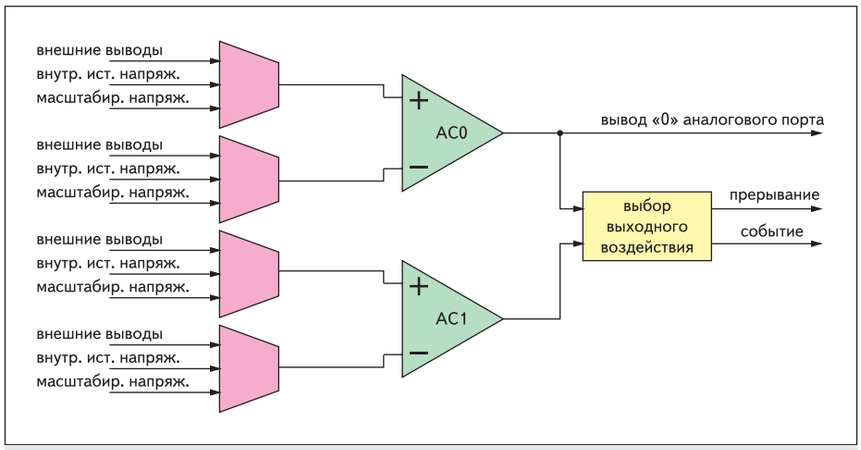


Рис. 5. Блок аналоговых компараторов

фигурирован для формирования запроса на прерывание и события в зависимости от различных комбинаций изменения входных уровней напряжения. Основные параметры компаратора (гистерезис и задержка распространения) могут программироваться. Аналоговые компараторы всегда группируются в пары (обозначаются AC0 и AC1) для каждого аналогового порта XMEGA. Они имеют идентичное поведение, но различные управляющие регистры.

В качестве входных сигналов для AC могут выступать напряжение на внешних выводах микроконтроллера, источники напряжения внутри кристалла, а также масштабируемые напряжения. Все линии аналогового порта XMEGA могут быть выбраны как входы для AC. При этом к положительному входу компаратора могут быть подключены линии 0, 1, 2, 4 и 6, а к отрицательному — линии 0, 1, 3, 5 и 7. В качестве внутренних источников напряжения могут быть выбраны или выход ЦАП (если есть), или выход встроенного точного ИОН. Что касается масштабируемых входов, то эта особенность AC у XMEGA является наиболее интересной. AC содержит программируемый делитель, который имеет 64 градации для деления напряжения, подаваемого на его вход: напряжение питания микроконтроллера (Vcc), напряжение встроенного ИОН, либо входное аналоговое напряжение, подаваемое на вывод 0 соответствующего аналогового порта. Выбор источника напряжения для деления и коэффициент деления задаются программистом путем записи байта в соответствующий регистр CTRLB.

Многофункциональная линия 0 аналогового порта в микроконтроллерах XMEGA имеет еще одну важную особенность. Как видно из блок-схемы, цифровой выход компаратора AC0 может быть непосредственно подключен к этой линии, сигнализируя о том, что соотношение двух (например, внешних) аналоговых сигналов изменилось. Известно, что в микроконтроллерах с архитектурой AVR логический выход компаратора AC0 или AC1 может быть подключен к входу одного из таймеров-счетчиков внутри кристалла. Если этот таймер-счетчик работает в режиме захвата, то можно измерять длительность аналоговых сигналов или реализовывать АЦП двойного (двухтактного) интегрирования. Но теперь в XMEGA появляется новая дополнительная возможность — построение с помощью встроенного аналогового компаратора внешнего сигма-дельта АЦП, так как в XMEGA мы можем подключать логический выход компаратора еще и на внешний вывод микроконтроллера.

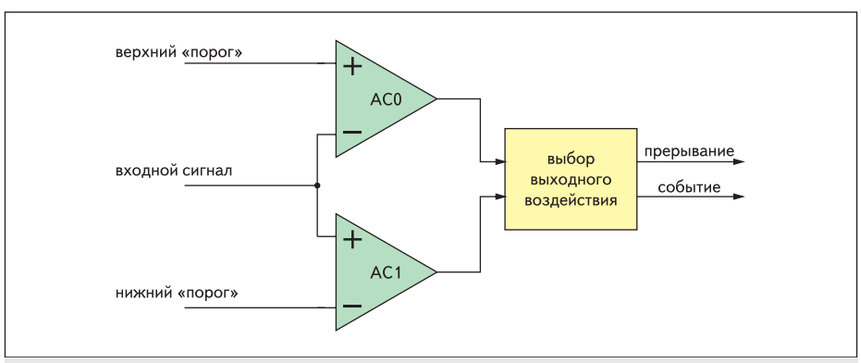


Рис. 6. «Оконный» режим работы аналоговых компараторов

Программист в рамках работы своего приложения может выбирать некий компромисс между быстродействием узла компараторов и его энергопотреблением. Так, если требуется высокая скорость срабатывания, то узел будет потреблять 130 мкА при времени срабатывания 30 нс. Если же высокая скорость не нужна, то AC работает в штатном режиме с временем срабатывания 500 нс, потребляя при этом всего 20 мкА. Есть возможность трехступенчатого программирования гистерезиса: отсутствие гистерезиса (0 мВ), малый гистерезис (±10мВ) или большой гистерезис (±25 мВ). В ряде случаев это помогает избежать излишне частого переключения компаратора в системах с повышенным уровнем шума. Оба описанных параметра AC программируются установками в регистре ACnCTRL, в этом же регистре определяется тип генерации прерывания после появления сигнала на выходе компаратора. Это может быть генерация события или запроса на прерывание по изменению состояния выхода или по фронту (срезу) выходного сигнала компаратора. События будут всегда генерироваться при тех же условиях, при которых будет генерироваться и запрос на прерывание, причем на генерацию события не влияет, разрешены прерывания или нет.

Перед началом преобразования аналоговый компаратор должен быть полностью инициализирован. AC работает постоянно, результат преобразования доступен как программно, так и аппаратно через систему событий.

И, наконец, рассмотрим режим работы AC в режиме «окна» (Window Mode). Два аналоговых компаратора, принадлежащие одному и тому же аналоговому порту, могут быть программным образом сконфигурированы для работы в Window Mode. Принцип работы AC в этом режиме показан на рис. 6. Можно задавать различную ширину «окна» и его положение относительно напряжения питания микроконтроллера или «земли». Входное напряжение, которое подвергается анализу, может быть выше «окна», ниже него или попадать точно внутрь «окна». В любом из этих случаев генерируется соответствующее прерывание и событие. Разрешение аналоговому компаратору на работу в этом режиме, а также выбор варианта, по которому генерируется событие или прерывание, осуществляется путем программирования регистра WINCTRL.

Таймеры-счетчики, сторожевой таймер и часы реального времени

Все таймеры-счетчики (T/C) в XMEGA 16-разрядные и имеют одинаковую структуру. Общие функции: формирование интервалов времени, сигналов заданной частоты и сигналов ШИМ, измерение временных параметров цифровых сигналов, синхрониза-

ция с системой событий. Совместно с T/C могут использоваться два внешних модуля расширения — модуль высокого разрешения (Hi-Res) и модуль формирования специализированных частотных сигналов для задач управления электродвигателями (AWeX).

Количество таймеров-счетчиков у разных микроконтроллеров семейства XMEGA составляет от 5 до 8. Блок T/C состоит из базового счетчика и набора каналов захвата-сравнения. Базовый счетчик может использоваться для подсчета тактовых циклов или событий. Он имеет возможность управления направлением счета и периодом установки временного интервала. Каналы захвата-сравнения могут использоваться совместно с базовым счетчиком для реализации функций сравнения и генерации различных цифровых последовательностей или выполнять функции захвата. Сравнение и захват являются взаимоисключающими операциями, то есть какой-то один таймер-счетчик не может одновременно и генерировать заданную цифровую последовательность на выходе, и осуществлять функцию захвата. Блоки T/C на кристаллах XMEGA обозначаются T/C0 и T/C1, причем T/C0 имеет 4 канала захвата-сравнения, в то время как T/C1 — всего 2 канала. Все таймеры-счетчики для тактирования могут быть присоединены либо к сигналу периферийной тактовой частоты, либо к системе событий.

У таймеров-счетчиков XMEGA есть еще целый ряд интересных архитектурных решений, подробное описание которых планируется сделать в третьей части цикла статей.

Сторожевой таймер (WDT) предназначен для постоянного мониторинга корректного исполнения программного потока, делая возможным восстановление системы в случае сбоев. WDT работает постоянно, если его работа разрешена системой. Если WDT не сбрасывается из программы в течение предустановленного периода времени, то инициируется общий сброс микроконтроллера. Периодический сброс сторожевого таймера осуществляется путем выполнения команды WDR (Watchdog Timer Reset) в ходе выполнения основной программы приложения. Это — стандартный режим. Всего в микроконтроллерах XMEGA для стандартного тайм-аута WDT определены 11 возможных значений, которые могут быть выбраны пользователем в интервале от 8 мс до 8 с. Сторожевой таймер может быть сброшен в любое время в течение этого периода. Значение «по умолчанию» определяется fuse-битами.

Сторожевой таймер также может работать и в режиме «окна» (Window Mode). Здесь пользователь может самостоятельно определить временной слот, в течение которого WDT должен быть сброшен для нормального функционирования системы. Если WDT сбрасывается слишком рано или слишком поздно, то это считается ошибкой и генерируется системный сброс процессора. В режиме «окна» WDT использует два различных периода тайм-аута — закрытый (closed, TOWDTW) и открытый (normal, TOWDT). Закрытый период тайм-аута определяет продолжительность от 8 мс до 8 с, в течение которого WDT не должен сбрасываться пользователем, в противном случае генерируется системный сброс. Открытый период тайм-аута также лежит в пределах от 8 мс до 8 с, но в течение этого периода сторожевой таймер может (и должен!) быть сброшен. Открытый период всегда следует за закрытым периодом, поэтому общее время продолжительности тайм-аута будет складываться из времен обоих тайм-аутов — закрытого и открытого. Значение закрытого периода по умолчанию определяется fuse-битами. В качестве иллюстрации на рис. 7, 8 приведено сравнение стандартного и «оконного» режимов работы сторожевого таймера в микроконтроллерах XMEGA.

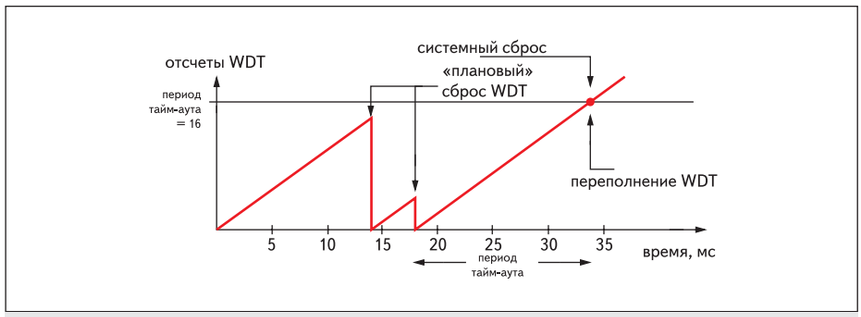


Рис. 7. Стандартный режим работы сторожевого таймера

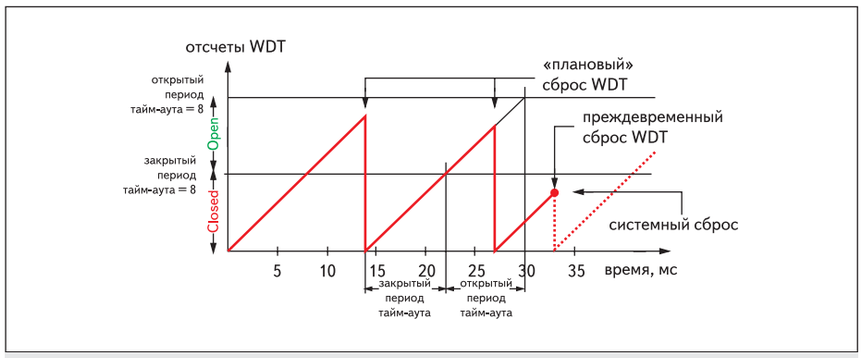


Рис. 8. «Оконный» режим работы сторожевого таймера

Сторожевой таймер, если разрешен, будет работать во всех энергосберегающих режимах. Он тактируется сигналом с частотой 1 кГц, которая получается делением частоты выходного сигнала внутреннего генератора ULP32K. Так как этот генератор работает независимо от центрального ядра, то WDT будет продолжать функционировать и вызовет системный сброс микроконтроллера, даже если по каким-то причинам пропадет основной тактовый сигнал. Благодаря сверхнизкому энергопотреблению точность формирования частоты генератором ULP32K не слишком высока (см. описание на микроконтроллер), поэтому точное значение периода тайм-аута может варьироваться от микроконтроллера к микроконтроллеру. При разработке приложения с использованием WDT подобный разброс значений должен обязательно учитываться.

Сторожевой таймер снабжен механизмом защиты от непреднамеренного изменения его установок. Можно использовать специальный бит Change Enable (CEN), без установки которого невозможно поменять содержимое управляющих регистров WDT. Можно также использовать fuse-бит WDT Lock. Если установить этот бит, то содержимое управляющего регистра сторожевого таймера уже нельзя будет переписать, и, следовательно, станет невозможно запретить работу WDT из приложения пользователя. После системного сброса WDT начинает работу в сконфигурированном режиме. Интересно отметить, что при установленном fuse-бите, когда WDT работает в «оконном» режиме, периоды тайм-аута не могут быть изменены, но сам «оконный» режим может быть разрешен или запрещен.

Часы реального времени (RTC) функционируют в режимах пониженного энергопотребления, пробуждая кристалл через регулярные интервалы времени. Блок-схема RTC в микроконтроллерах XMEGA показана на рис. 9. В основе лежит 16-битный счетчик, который считает импульсы опорной частоты и выдает событие или запрос на прерывание по переполнению или по достижению задан-

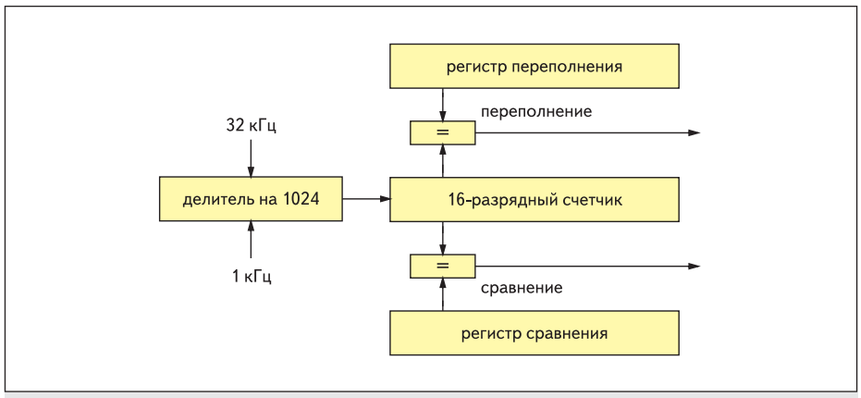


Рис. 9. Блок-схема часов реального времени

ного значения. Опорная частота (базовый вариант) получается при помощи точного внешнего часового кварца 32,768 кГц, причем дизайн счетчика на кристалле оптимизирован для максимально низкого энергопотребления.

Опорная частота для RTC может составлять и 1,024 кГц. В этом случае она получается внутренним делением частоты 32 кГц от одного из трех источников: генератора на внешнем часовом кварце 32 кГц (как в базовом варианте), внутреннего генератора RC32K (32 кГц) или генератора ULP32K, который входит в состав сторожевого таймера и имеет минимальное энергопотребление. В состав RTC также входит программируемый делитель (на 7 фиксированных значений) для понижения частоты сигнала перед подачей его на основной счетчик. Это позволяет получить максимальное время ожидания более 18 часов.

Существует два пути для формирования запроса на прерывание и события — по сравнению или по переполнению (рис. 9). В первом случае счетчик RTC продолжает считать дальше, а во втором — обнуляется при достижении значения, записанного в регистр Period.

Модули последовательных коммуникаций

Микроконтроллеры XMEGA снабжены богатым арсеналом аппаратных модулей для организации последовательного обмена данными с внешним миром: SPI, TWI и USART, причем USART также может работать и в режиме аппаратной поддержки IrDA. Количество и состав периферийных коммуникационных блоков XMEGA варьируется от подсемейства к подсемейству. Так, USART может быть от 5 до 8 модулей, а SPI и TWI — до 4. Первое подсемейство XMEGA (A1) имеет 8 модулей USART и по 4 модуля SPI и TWI. Самое «маленькое» подсемейство (A3) будет иметь 5 блоков USART, 4 блока SPI и 2 блока TWI.

Аппаратная реализация коммуникационных интерфейсов TWI и SPI, а также организация работы с ними у микроконтроллеров XMEGA идентичны «старшим» кристаллам megaAVR и в цикле статей рассматриваться не будут.

Универсальный синхронный-асинхронный последовательный приемопередатчик (USART) в микроконтроллерах XMEGA поддерживает полнодуплексный обмен в асинхронном и тактируемом синхронном режимах. Он может быть сконфигурирован в режим эмуляции SPI Master, при этом поддерживаются все 4 режима работы SPI (Mode 0, 1, 2 и 3) и есть возможность выбора порядка передачи данных: первым передается LSB или MSB. Когда USART установлен в режим эмуляции SPI Master, вся специальная встроенная логика USART запрещена. Активными остаются буферы приема и передачи, сдвиговые регистры и генератор скорости передачи. Управление внешними выводами и формирование запросов на прерывание идентичны для обоих режимов. Обмен данными USART фреймирован, формат кадра может специфицироваться пользователем. USART буферизован на прием и на передачу. Различные вектора прерываний для индикации завершения приема или передачи обеспечивают гибкое управление обменом данными посредством прерываний. Также может быть разрешена работа схемы формирования и контроля бита четности. Подробнее структуру и работу USART целесообразно смотреть в описании на микроконтроллер.

Рассмотрим подробнее новый интересный режим работы у USART в микроконтроллерах XMEGA — физическую поддержку IrDA (имеется в виду физическая модуляция-демодуляция сигнала на скоростях обмена до 115,2 кбод). Поддерживаются три схемы модуляции: 3/16 периода скорости обмена данными, фиксированная длительность импульсов (программируется, 8 бит), запрещение модуляции импульса.

Напомним основы IrDA. Байт, который требуется передать, посылается в USART из CPU, USART добавляет старт/стоп биты и передает байт последовательно, начиная с LSB. Логический «0» (вспышка света) передается одиночным ИК-импульсом длиной от 1,6 мс до 3/16 периода передачи битовой ячейки, а логическая «1» передается как отсутствие ИК-импульса. Отсюда следует, что минимальное энергопотребление гарантируется при фиксированной длине импульса 1,6 мс. По окончании кодирования битов необходимо пропустить ток через ИК-светодиод, чтобы выработать световой импульс требуемой интенсивности. При приеме ИК-импульсы поступают на PIN-диод, который преобразует их в импульсы тока. Они затем усиливаются, фильтруются и сравниваются с пороговым уровнем для преобразования в логические уровни.

Всего в XMEGA есть один доступный модуль IrDA, который может быть присоединен к любому из блоков USART (рис. 10). В режиме IrDA сигналы между USART и выводами RX/TX коммутируются, как показано на рис. 10. Для приема можно запрограммировать минимальную длительность импульса, которая будет интерпретироваться как логический «0». Все более короткие импульсы будут декодироваться в логическую «1» (отсутствие импульса).

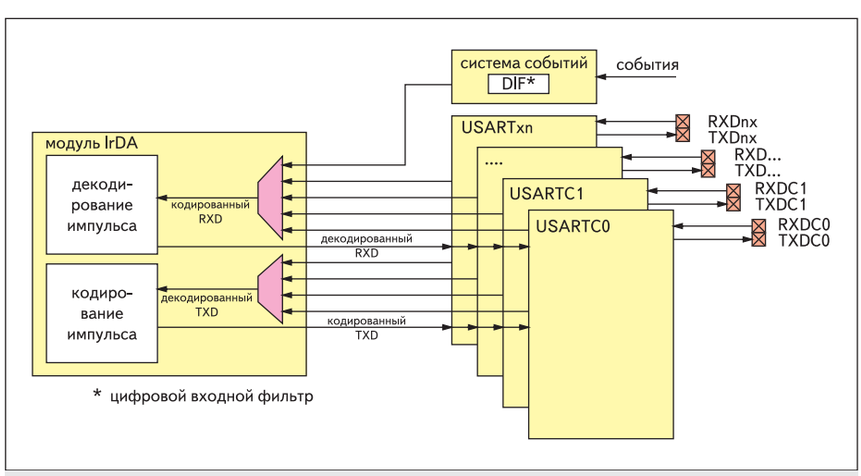


Рис. 10. Работа USART в режиме эмуляции IrDA

Модуль IrDA может быть использован в комбинации только с одним из USART в каждый момент времени. Следовательно, режим IrDA не должен быть одновременно установлен более чем для одного USART. Проверка этого обязательного условия лежит на программисте.

В качестве входа для приемника IrDA можно также выбрать канал системы событий. Это автоматически запретит RX-вход с вывода микроконтроллера. Система событий имеет цифровой входной фильтр (DIF) на каналах событий, что может быть использовано для фильтрации. Подробнее мы рассмотрим это во второй части цикла статей, когда будем описывать работу Event System.

Аппаратные устройства шифрования

В состав микроконтроллеров XMEGA введены аппаратные средства обеспечения защиты информации. В качестве реализованных стандартов компания Atmel выбрала два широко распространенных стандарта шифрования — AES и DES. У XMEGA шифрование по стандарту DES (поддержка DES и triple-DES) реализуется выполнением специальной команды DES, входящей в состав системы команд процессорного ядра. Команда DES короткая, она исполняется за один период основной тактовой частоты. Для шифрования требуется поместить 8-байтный ключ и 8-байтный блок данных в регистровый файл, а затем последовательно выполнить команду DES 16 раз для шифрования или дешифровки блока данных. Криптомодуль AES шифрует и дешифрует блоки данных размером 128 бит с использованием 128-битного ключа. И ключ, и данные должны быть записаны в модуль до начала выполнения цикла команд кодирования-декодирования. Длительность процесса обработки блока составляет 375 периодов периферийной тактовой частоты, после чего данные можно использовать по назначению.

Режимы энергосбережения

Микроконтроллеры XMEGA поддерживают 5 режимов пониженного энергопотребления, как и все современные «базовые» микроконтроллеры AVR.

- Режим холостого хода (Idle), в котором прекращает работу только процессорное ядро и фиксируется содержимое памяти программ. Все периферийные модули продолжают функционирование в обычном режиме, включая контроллер DMA и систему событий. Все разрешенные системы тактирования также продолжают работать. В режиме Idle разбудить процессорное ядро может любой периферийный блок, причем ядро начинает работу сразу же, без задержки.
- Режим микропотребления (Power Down), в котором сохраняется содержимое регистрового файла, но останавливается работа системы тактирования. Выход из Power Down возможен по общему сбросу микроконтроллера, по разрешенному внешнему прерыванию, по совпадению адреса в блоке двухпроводного последовательного интерфейса TWI. Сторожевой таймер также может разрешить выход из режима микропотребления (при условии, что он сам разрешен).
- Режим сохранения энергии (Power Save), который идентичен Power Down, но в котором еще допускается независимая работа RTC. Все прочие генераторы запрещены. Выход из режима Power Save осуществляется при тех же условиях, что и для Power Down, и дополнительно может происходить по сигналу от RTC.
- Основной режим ожидания (Standby). Идентичен режиму Power Down, но здесь работа тактового генератора не прекращается. Это гарантирует быстрый выход микроконтроллера из режима ожидания.
- Дополнительный режим ожидания (Extended Standby). Идентичен режиму Power Save, но здесь работа тактового генератора тоже не прекращается. Это гарантирует быстрый выход микроконтроллера из режима ожидания.

После выхода из режимов пониженного энергопотребления центральное процессорное ядро блокируется в тактировании на 4 такта генератора перед началом выполнения программы. Это время задержки будет присутствовать всегда. Но общее время выхода будет также зависеть от источника тактового сигнала, его состояния и конкретного режима пониженного энергопотребления. Так, задержка старта для любого из внутренних RC-генераторов составляет 6 тактов, а для внешних генераторов определяется конфигурацией битов XOSCSEL[3:0] в управляющем регистре XOSCCTRL.

Микроконтроллеры XMEGA имеют примечательную архитектурную особенность, позволяющую значительно снизить энергопотребление всего кристалла как в активном режиме, так и в режиме холостого хода Idle. Новые регистры PRR (Power Reduction Register) позволяют индивидуально останавливать тактирование для периферийных блоков путем установки в PRR конкретного бита, соответствующего определенному периферийному блоку. При этом текущее состояние этого блока «замораживается» и соответствующие регистры ввода/вывода не могут быть записаны или прочитаны. Ресурсы, ассоциированные с этим блоком и занятые этим блоком, тоже будут оставаться недоступными, поэтому в большинстве случаев имеет смысл запретить работу периферийного блока до прекращения подачи тактовой частоты. Возобновление тактирования узла приводит к полному восстановлению его состояния на момент «заморозки». В других режимах пониженного энергопотребления использование регистров PRR не требуется, так как сигнал периферийной тактовой частоты отсутствует, и тактирование периферии не осуществляется.

Отметим, что не все микроконтроллеры XMEGA имеют набор периферийных модулей, полностью ассоциированный с битами в регистрах PRR. Может случиться, что для каких-то блоков на кристалле в регистрах нет соответствующих битов остановки тактирования. В таких случаях установка бита в PRR не будет иметь никакого эффекта.

Конфигурируемые порты ввода/вывода

Отличительной чертой построения портов ввода/вывода у AVR традиционно является наличие трех битов для каждого физического вывода микросхемы: бит данных (OUTx), бит управления направлением передачи данных (DIRx) и бит для отображения логического уровня сигнала на физическом выводе микросхемы (INx). Такое решение позволяет полностью контролировать процесс ввода/вывода и операции «чтение–модификация–запись», что особенно актуально при работе микроконтроллера в условиях внешних электрических помех.

Это удачное решение было оставлено для новых микроконтроллеров XMEGA, был также сохранен весь базовый набор регистров. Но, тем не менее, для XMEGA сделан ряд усовершенствований. Например, порт ввода/вывода теперь можно программно установить в одну из шести возможных конфигураций. С целью более эффективного битового управления линиями портов ввода/вывода для OUTx и DIRx добавлены функции «установить» — SET, «сбросить» — CLR и «переключить» — TGL. Для этого разработчикам XMEGA пришлось ввести дополнительные наборы регистров для OUT (OUTSET, OUTCLR, OUTTGL) и DIR (DIRSET, DIRCLR, DIRTGL). Но главная особенность портов ввода/вывода — так называемые виртуальные регистры.

Идея введения виртуальных регистров возникла из потребности обеспечить удобное и простое управление отдельными линиями всех портов ввода/вывода XMEGA. Виртуальные регистры позволяют стандартным регистрам порта, которые расположены в адресном пространстве расширенной области ввода/вывода, быть отображенными на «базовое» для AVR адресное пространство I/O памяти. Когда такое отображение осуществляется, запись в виртуальный регистр будет эквивалентна записи в реальный регистр порта. Это позволяет использовать специфические команды раздела I/O памяти (с адресами 0x00–0x1F) для манипулирования битами, а также специфические команды I/O памяти IN и OUT (в диапазоне адресов 0x00–0x3F) для тех регистров порта, которые у XMEGA находятся в расширенной области I/O памяти. Всего у микроконтроллеров XMEGA есть 4 виртуальных порта, поэтому одновремен-

но может быть отображено до 4 портов ввода/вывода. Отображаемые регистры — IN, OUT, DIR и INTFLAGS.

Заключение

Отметив свое десятилетие в 2006 году, платформа AVR 8-bit RISC остается популярной во всем мире и продолжает активно развиваться благодаря постоянному совершенствованию технологии Atmel, появлению новых версий микроконтроллеров и совершенствованию средств поддержки разработок — как аппаратных, так и программных. Появление нового, более современного и мощного семейства AVR еще раз доказывает, что эта платформа реально претендует на лидерство в сегменте современных высокопроизводительных 8-разрядных микроконтроллеров универсального назначения, действительно перекрывая многочисленные потребности разнообразных конечных приложений. И, как всегда, выход XMEGA на рынок будет поддержан наличием качественного и доступного инструментального программного обеспечения, примерами программ, квалифицированной технической поддержкой специалистов центра AVR корпорации Atmel в Норвегии.

Во второй части планируется рассмотреть систему ввода/вывода, контроллер прерываний и систему событий, а третья часть будет посвящена работе таймеров-счетчиков, аналого-цифрового преобразователя и контроллера прямого доступа к памяти.

Продолжение следует

Литература

1. Курилин А. Микроконтроллеры AVR: что нового? // Компоненты и технологии. 2006. № 1.
2. XMEGA Training Data // Atmel AVR Distributor Training — Atmel Norway, September 2007.
3. XMEGA Hands-On Session // Atmel AVR Distributor Training — Atmel Norway, September 2007.

Продолжение. Начало № 3`2008

Игорь КРИВЧЕНКО, к. т. н.
ik@efo.ru
Елена ЛАМБЕРТ, к. т. н.
elena@efo.ru

Микроконтроллеры XMEGA — новые возможности проверенного решения. Часть 2

Конфигурируемые порты ввода/вывода

Порты ввода/вывода у всех микроконтроллеров AVR традиционно имеют три управляющих бита для каждого физического вывода: бит данных (PORTx), бит управления направлением передачи данных (DDRx) и бит для отображения логического уровня сигнала на физическом выводе микросхемы (PINx). Для конфигурации линии порта также используется бит PUD регистра SFIOR, с помощью которого вывод подтягивается к шине питания через внутренний резистор. Это позволяет обеспечить истинную функциональность вида «чтение — модификация — запись» и независимо выполнять битовые операции на любой линии порта. Порты ввода/вывода AVR, их конфигурирование, управляющие регистры и особенности работы подробно описаны в многочисленных изданиях (например, [1]), и здесь мы на этом останавливаться не будем.

Удачное построение портов ввода/вывода AVR было использовано и для XMEGA, причем разработчики кристалла сохранили весь базовый набор регистров. Но для микроконтроллеров нового поколения был сделан ряд важных усовершенствований:

- Изменились символические имена управляющих регистров, они стали более «естественными» для восприятия. Так, бит данных теперь обозначается OUTx, бит управления направлением передачи данных — DIRx, а бит для отображения логического уровня сигнала на физическом выводе микросхемы — INx.
- Расширились возможности работы портов ввода/вывода: каждая линия порта теперь имеет 6 конфигураций и 8 режимов работы. Гибкая конфигурация обеспечивается благодаря наличию нового специального регистра PINnCTRL.
- Для более эффективного битового управления линиями портов ввода/вывода для OUTx и DIRx добавлены функции «установить» — SET, «сбросить» — CLR и «переключить» — TGL. Соответственно, введены дополнительные наборы регистров для OUT (OUTSET, OUTCLR, OUTTGL) и DIR (DIRSET, DIRCLR, DIRTGL).
- С помощью виртуальных регистров реализована возможность отображения регистров порта в адресное пространство I/O памяти, доступное для манипуляции битами.
- Доступно синхронное или асинхронное обнаружение изменения входного сигнала на внешнем выводе с формированием события или запроса на прерывание, включая выведение микроконтроллера из режимов пониженного энергопотребления.
- Реализовано управление скоростью нарастания входного сигнала.
- Добавлено групповое управление конфигурацией сразу нескольких линий порта за одну операцию при помощи маски.
- Имеется возможность подключения к линии порта сигнала периферийной тактовой частоты и выхода одного из каналов системы событий.

Все описанные функции доступны для каждой линии портов ввода/вывода XMEGA. Отметим, что в процессе работы микроконтроллера операции с какой-то отдельной линией порта не оказывают влияния на функционирование других линий этого же порта и других портов ввода/вывода. Это относится к изменению направления передачи данных, к изменению логического уровня на выводе микросхемы при конфигурации на выход и к подключению (отключению) внутренних резисторов при конфигурации на вход.

Регистр PINnCTRL используется для реализации новых режимов конфигурации и работы линии порта ввода/вывода. Каждый вывод теперь может быть сконфигурирован как Push-Pull (двухтактный каскад с симметричной нагрузочной способностью), как монтажное «И» или как монтажное «ИЛИ». Можно разрешить инверсию сигнала на отдельной линии порта при вводе/выводе данных, а также разрешить ограничение скорости нарастания сигнала на ней при вводе данных. Управление скоростью нарастания входного сигнала снижает энергопотребление узла. В среднем после включения этого ограничения длительность фронта и среза входного сигнала увеличивается на 50–150% в зависимости от напряжения питания микроконтроллера, рабочей температуры и характера нагрузки.

Для режима Push-Pull есть 4 возможные конфигурации: собственно Push-Pull (или Totem-pole), подтяжка к «земле» (Pull-down), подтяжка к шине питания (Pull-up) и сохранение последнего активного состояния на шине (Bus-keeper). В конфигурации Bus-keeper порт может работать в двух направлениях — на прием и на передачу. Этот позволяет избежать колебаний сигнала на выводе при отключении нагрузки, а также когда мы запрещаем работу линии порта на выход. Слаботочные буферы Bus-keeper удерживают на выводе микросхемы логический уровень сигнала, который присутствовал последним: то есть реализуется подтяжка к «1», если последний уровень был «1», или подтяжка к «земле», если последний уровень был «0». Для конфигураций Pull-down и Pull-up подтяжка вывода к «земле» или к шине питания осуществляется через активный резистор только в том случае, если линия порта работает на вход. Такое решение позволяет снизить энергопотребление. Для режимов монтажное «И» и монтажное «ИЛИ» дополнительные резисторы подтяжки (pull-up и pull-down соответственно) могут быть подключены к выводу микросхемы как при работе на вход, так и при работе на выход.

Подключение и отключение внутренних резисторов подтяжки при помощи регистра конфигурации PINnCTRL является удачным решением разработчиков XMEGA. Благодаря этому на выводе микроконтроллера не наблюдается промежуточных состояний линии порта, когда мы переключаем направление работы порта или изменяем логическое значение на выводе.

Состав регистра PINnCTRL и назначение его битов подробно описаны в технической документации на микроконтроллер. На рис. 1 проиллюстрированы все возможные конфигурации линии порта ввода/вывода XMEGA и режимы работы.

Программист имеет возможность вывести сигнал периферийной тактовой частоты на любую линию порта ввода/вывода. К линии порта также можно программно подключить канал 7 системы событий (Event Channel 7). Если в этом канале генерируется событие, то соответствующий этому событию сигнал будет отображаться на выводе микроконтроллера в течение одного периода периферийного тактового сигнала.

Каждая линия порта ввода/вывода у XMEGA может быть запрограммирована на генерацию событий или запросов на прерывание по фронту, срезу, перепаду или низкому уровню входного сигнала на выводе микро-

контроллера. Если необходимо «почувствовать» высокий уровень, то следует использовать функцию инверсии путем установки бита INVEN в регистре PINnCTRL. Поддерживается синхронное и асинхронное обнаружение изменения входного сигнала. Синхронное детектирование требует наличия сигнала периферийной тактовой частоты, асинхронное — не требует.

Генерация событий возможна только при наличии сигнала периферийной тактовой частоты, поэтому асинхронная генерация событий при детектировании изменения состояния вывода микроконтроллера невозможна. Для генерации события по фронту, срезу или перепаду значение сигнала на выводе должно измениться в течение одного периода периферийной тактовой частоты. Если вывод запрограммирован для генерации события по наличию на нем сигнала низкого уровня, то данная линия порта непосредственно подключается к каналу системы событий. При этом Event System постоянно отслеживает текущее значение сигнала на выводе микроконтроллера: низкий уровень не генерирует событие, а при наличии высокого уровня события будут генерироваться постоянно. Это может вызвать «бесконечную» генерацию событий. Контроль такой ситуации и необходимые действия должны осуществляться программистом.

Немного сложнее обстоит дело с генерацией запросов на прерывание. Каждый порт ввода/вывода у микроконтроллеров XMEGA имеет два вектора прерывания. Программист может выбирать линии порта, которые будут использоваться для формирования запросов на прерывание по этим векторам. Тип изменения входного сигнала (фронт, срез, перепад или низкий уровень), которое вызывает генерацию, будет зависеть от вида детектирования — синхронное или асинхронное.

При синхронном детектировании запросы на прерывание генерируются для любого типа изменения входного сигнала. Как и в случае системы событий, для генерации запроса на прерывание по фронту, срезу или перепаду значение сигнала на выводе должно измениться в течение одного периода периферийной тактовой частоты.

Асинхронное детектирование интересно тем, что именно с его помощью можно вывести микроконтроллер из «спящих» режимов при изменении состояния сигнала на внешнем выводе. Отметим, что при асинхронном детектировании фронта, среза или перепада только линия 2 любого порта ввода/вывода XMEGA обладает возможностью действительно поддерживать полностью асинхронное детектирование. Это означает, что линия 2 будет определять и «ловить» любое изменение входного сигнала с последующей генерацией запроса на прерывание. Все другие линии порта имеют ограничение по выведению микроконтроллера из «спящего» режима. Значение входного сигнала, которое изменилось в виде любого перепада, должно удерживаться на выводе до тех пор, пока кристалл не «проснется» и не появится сигнал тактовой частоты. Если в течение этого времени сигнал вернется к первоначальному значению, то микроконтроллер все равно «проснется», но запрос на прерывание не будет сгенерирован.

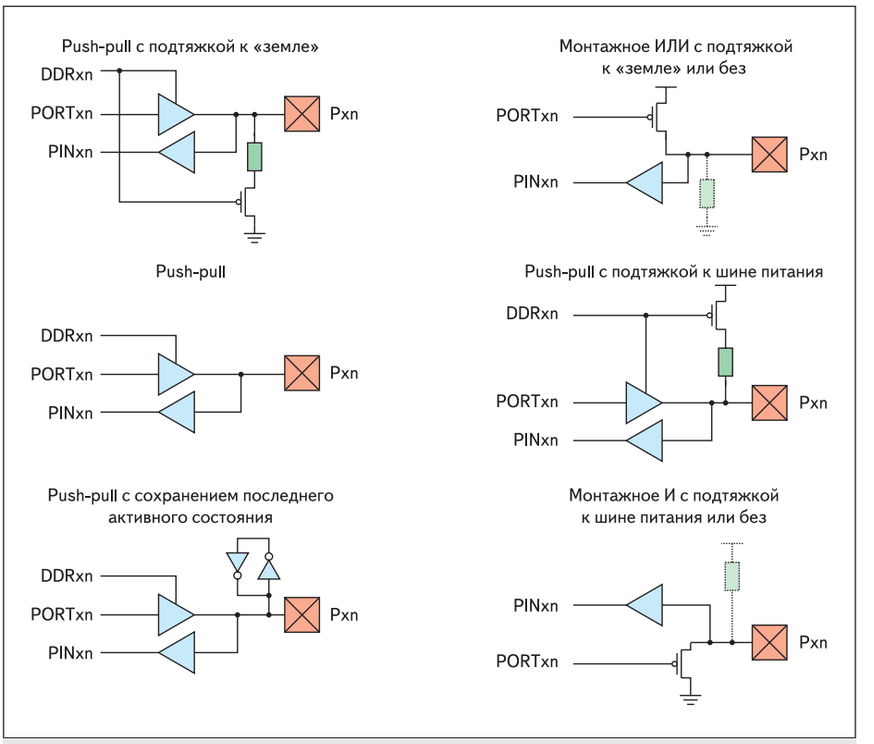


Рис. 1. Конфигурации порта ввода/вывода в микроконтроллерах XMEGA

Низкий уровень входного сигнала на выводе микроконтроллера может детектироваться всеми линиями порта ввода/вывода независимо от того, присутствует сигнал периферийной тактовой частоты или нет. Если линия порта сконфигурирована на генерацию прерывания по низкому уровню входного сигнала, запросы на прерывание будут возникать всегда, пока вывод удерживается в низком состоянии. Анализ состояния вывода в этом случае проводится поразному в зависимости от режима работы микроконтроллера. В активном режиме низкий уровень должен удерживаться, пока не завершится текущая инструкция CPU, и только потом будет сформирован запрос на прерывание. В режимах пониженного энергопотребления низкий уровень должен удерживаться на выводе в течение всего времени «просыпания» микроконтроллера, и только потом запрос на прерывание может быть сгенерирован. Если низкий уровень входного сигнала пропадет во время выхода из «спящего» режима, то микроконтроллер «проснется», но запроса на прерывание не будет.

В техническом описании на микроконтроллеры XMEGA приводятся все случаи, когда будет происходить генерация запросов на прерывание в зависимости от вида изменения входного сигнала на внешнем выводе микросхемы. Подробное описание последовательности действий для конфигурации линии порта ввода/вывода для генерации запроса на прерывание можно найти в Application Note AVR1313.

Рассмотрим еще два новшества у портов ввода/вывода XMEGA, которые позволяют снизить размер кода и увеличить скорость выполнения программы. Это групповая конфигурация нескольких выводов по маске и виртуальные регистры.

Идея добавления виртуальных регистров на кристалл XMEGA возникла из потребности обеспечить удобное и простое управление отдельными линиями каждого из многочисленных портов ввода/вывода новых микроконтроллеров.

Виртуальные регистры порта позволяют стандартным регистрам порта, расположенным в адресном пространстве расширенной области ввода/вывода (адреса старше 0x3F), быть отображенными на «базовое» для AVR адресное пространство области ввода/вывода (I/O Memory). Когда такое отображение осуществляется, запись в виртуальный регистр будет эквивалентна записи в реальный регистр порта. При этом для любого порта ввода/вывода XMEGA становится доступным весь ассортимент инструкций AVR для работы с регистрами, включая специальные

команды манипуляции битами. Всего у микроконтроллеров XMEGA есть 4 виртуальных порта, поэтому одновременно может быть отображено до 4 портов ввода/вывода. Отображаемые регистры — IN, OUT, DIR и INTFLAGS.

Почему так полезно использование виртуальных регистров? Потому что разница во времени исполнения и в размере генерируемого кода при работе с обычными и виртуальными портами может быть значительна. Некоторые инструкции из набора команд AVR могут работать только с регистрами, адреса которых в адресном пространстве AVR не старше 0x1F (0x3F). Использование именно этих команд вместо их эквивалентов (обращение к ячейкам памяти по этим же адресам) происходит быстрее и занимает меньше процессорного времени. А ведь все адреса регистров портов ввода/вывода XMEGA старше «заветной» границы пространства I/O Memory (0x3F). Вот для такой полезной «подмены» и используются виртуальные регистры. Поясним сказанное на простом примере. Установим в высокий уровень линию 0 порта C, не меняя при этом состояние других линий этого порта:

```
// Использование виртуального порта PVIRT0
sbi PVIRT0_OUT, 0;
// Код состоит из одного слова, время выполнения — 1 такт*)

// Прямая адресация порта C:
sbr r16, 0x01;
sts PORTC_base + PORT_OUTSET_offset, r16;
// Размер этого кода — 3 слова, время выполнения — 3 такта.
```

*) Отметим, что хотя CPU выполняет команду «sbi» за один такт, для появления результата на линии ввода/вывода требуется 2 такта

Этот пример показывает, что использование виртуального порта позволяет сократить требуемый объем памяти программ и сократить время исполнения для одинаковых операций. Это особенно важно, когда требуется быстрое выполнение программы и осуществляется постоянное обращение к регистрам порта.

Поговорим теперь о групповой конфигурации. Наличие нового регистра конфигурации PINnCTRL для каждой линии порта ввода/вывода, безусловно, удобно и полезно. Но при этом также возрастает количество операций, необходимых для последовательной конфигурации всего порта, разряд за разрядом. В результате размер генерируемого кода существенно увеличивается. В то же время очень часто встречаются ситуации, когда несколько линий одного порта должны быть сконфигурированы одинаковым образом. Для сокращения количества необходимых операций в таких случаях разработчики XMEGA добавили еще один специальный регистр — MPCMASK. Этот глобальный регистр является общим для всех портов ввода/вывода микроконтроллера и предназначен для одновременной конфигурации нескольких линий порта. Его использование позволяет создавать эффективный и компактный код.

Регистр MPCMASK загружается битовой маской, которая затем используется при работе с конфигурационными регистрами PINnCTRL. Делается это так. Сначала программистом определяется необходимая маска конфигурации (битовый шаблон) в регистре MPCMASK — то есть набор требуемых линий порта, которые нужно конфигурировать одинаковым образом. Запись «1» в бит «n» этого регистра означает, что линия порта «n» должна быть участником групповой конфигурации. Затем, когда любая линия этого порта конфигурируется путем записи информации в ее регистр PINnCTRL, во все остальные регистры PINnCTRL этого же порта, помеченные маской в регистре MPCMASK, записываются те же самые значения. Регистр MPCMASK очищается автоматически после окончания операции записи во все регистры PINnCTRL порта.

Отметим, что перед конфигурацией нескольких линий порта ввода/вывода с помощью маски рекомендуется глобально запретить все прерывания, а после завершения конфигурации — снова разрешить их. Это связано с аппаратными особенностями организации порта ввода/вывода XMEGA (подробнее см. в Application Note AVR1313).

В качестве иллюстрации рассмотрим практический пример работы с регистром MPCMASK. Для его реализации использовался стартовый набор разработчика STK600. Исходный текст написан на Си и компилирован в интегрированной среде IAR Systems EWAVR 4.30. Необходимые установки на плате STK600 (перемычки, соединительные кабели, переключатели и т. п.) в данной статье не описываются, мы рекомендуем использовать руководство пользователя для STK600 («User Manual»).

Пример 1. Одновременное конфигурирование нескольких выводов микроконтроллера

Задача заключается в том, чтобы считывать состояние кнопок SW на плате STK600 и копировать их состояние на светодиоды этого стартового набора. Нажатие кнопки SWn должно индицироваться включением светодиода LEDn. Первая функция примера показывает, как можно прочитать существующие значения из регистра OUT порта, очистить биты в соответствии с маской и затем вывести обратно новые значения. Такая конструкция очень часто используется при работе с заданными битовыми полями в регистрах, не «трогая» остальные биты.

Сначала рассмотрим вариант, который не использует возможности регистра MPCMASK:

```
void SetLEDPort( PORT_t volatile * ledPort, unsigned char value, unsigned char mask )
  {
      value &= mask;  // Очистка ненужных битов из value по маске
      ledPort->OUT = ledPort->OUT & ~mask | value;
  }

unsigned char GetSwitches( PORT_t volatile * switchPort, unsigned char mask )
  {
      // Считываем входное значение, удаляем ненужные биты
      // и возвращаем значение
      return (switchPort->IN & mask);
  }

void main( void )
{
  // Определяем указатели на порты ввода/вывода, которые мы
  // хотим использовать для работы с кнопками и светодиодами
  // на плате STK600:
  PORT_t volatile * ledPort = &PORTD;
  PORT_t volatile * switchPort = &PORTC;

  // Готовим битовую маску для тех выводов микроконтроллера,
  // которые будут использоваться.
  unsigned char const mask = 0x07;  // Работаем только с линиями
                                    // 0, 1, и 2.

  // Устанавливаем линии 0, 1, и 2 порта D: выход, монтажное «И»
  // с подтяжкой.
  ledPort->PIN0CTRL = ledPort->PIN0CTRL & ~PORT_OPC_gm |
                      PORT_OPC_WIREDANDPULL_gc;
  ledPort->PIN1CTRL = ledPort->PIN1CTRL & ~PORT_OPC_gm |
                      PORT_OPC_WIREDANDPULL_gc;
  ledPort->PIN2CTRL = ledPort->PIN2CTRL & ~PORT_OPC_gm |
                      PORT_OPC_WIREDANDPULL_gc;

  ledPort->DIR = mask;

  // Теперь копируем состояние кнопок на светодиоды...
  for (;;) {
      unsigned value = GetSwitches( switchPort, mask );
      SetLEDPort( ledPort, value, mask );
  }
}
```

После компиляции данного фрагмента кода и его запуска на STK600 нажатие одной из кнопок SW0…2 будет включать соответствующий светодиод LED0…2. При отпускании кнопки светодиод будет гаснуть. Программа очень простая, но требует много кода, так как три строчки конфигурации линий порта D однотипны. Попробуем сократить размер генерируемого кода — используем регистр MPCMASK. Теперь текст основной программы ***main(void)*** будет выглядеть так:

```
void main( void )
{
  PORT_t volatile * ledPort = &PORTD;
  PORT_t volatile * switchPort = &PORTC;

  unsigned char const mask = 0x07;

  PORTCFG.MPCMASK = mask;     // Включаем линии 0, 1 и 2
                              // в группу конфигурации
  // Теперь все равно, какой из конфигурирующих регистров
  // порта устанавливать. Пусть это будет линия 2. Регистры
  // PINnCTRL 0 и 1 будут сконфигурированы такими же
  // значениями автоматически!

  ledPort->PIN2CTRL = ledPort->PIN2CTRL & ~PORT_OPC_gm |
                      PORT_OPC_WIREDANDPULL_gc;

  ledPort->DIR = mask;

  for (;;) {
      unsigned value = GetSwitches( switchPort, mask );
      SetLEDPort( ledPort, value, mask );
  }
}
```

Программа выполняет те же самые действия, но размер конечного кода заметно сократился!

В заключение проиллюстрируем с помощью этой программы некоторые аппаратные особенности порта ввода/вывода XMEGA. Мы конфигурировали линии на выход в режим работы монтажное «И» с подтяжкой вывода к шине питания микроконтроллера. Это

позволяет непосредственно соединять линии порта. Воспользуемся 4-проводным шлейфом из комплекта поставки STK600. Соединим с его помощью линии 0, 1 и 2 порта D между собой и подключим получившуюся группу к любому из светодиодов (например, к LED1). Теперь после запуска программы нажатие любой кнопки SW0…2 (или нескольких одновременно) будет включать светодиод LED1. При отпускании всех кнопок светодиод будет гаснуть.

Дополнительную информацию об особенностях работы с портами ввода/вывода в микроконтроллерах XMEGA можно найти в документации Atmel — Application Note AVR1313.

Многоуровневый программируемый контроллер прерываний

Как мы уже говорили в первой части цикла, компания Atmel впервые реализовала у своих 8-разрядных микроконтроллеров AVR многоуровневый программируемый контроллер прерываний (PMIC), который управляет обслуживанием запросов на прерывание, включая уровень и приоритет.

Прерывание сигнализирует об изменении состояния периферийного модуля и может использоваться для изменения порядка выполнения основной программы. Периферийные блоки (источники) могут генерировать один или несколько типов запроса на прерывание, которые разрешаются установкой индивидуальных битов управления в соответствующих регистрах для каждого источника. У микроконтроллеров XMEGA каждый источник прерывания и сброс микроконтроллера имеют отдельный адрес (вектор прерывания) в памяти программ, в котором размещается команда перехода на подпрограмму обслуживания этого прерывания (обработчик). Самый младший адрес принадлежит вектору сброса.

Периферийным модулям в микроконтроллерах XMEGA может быть назначен один из четырех различных уровней приоритета прерываний: Off (нет прерываний), High (высокий), Medium (средний) и Low (низкий). Прерывания с уровнем Medium будут приостанавливать работу обработчиков прерываний с уровнем Low. Запросы на прерывание, которым присвоен уровень High, обслуживаются немедленно после поступления. Внутри каждого из трех уровней (Low, Medium, High) приоритет обслуживания прерывания определяется исходя из абсолютного адреса вектора прерывания — вектор с наименьшим адресом обслуживается в первую очередь. Для прерываний, имеющих одинаковый статус Low, дополнительно предусмотрена процедура диспетчеризации Round Robin, когда все процессы активизируются в фиксированном циклическом порядке. И несмотря на все нововведения прерывания в XMEGA по-прежнему должны быть предварительно глобально разрешены для того, чтобы контроллер PMIC давал право обработчику вмешиваться в ход выполнения основной программы. Это выполняется программно путем установки бита I в CPU Status Register, причем данный бит не сбрасывается в процессе обслуживания прерывания.

Когда в PMIC поступает запрос на прерывание, то сначала проводится анализ уровня поступившего запроса на прерывание и статуса текущих обслуживающихся прерываний. Затем поступивший запрос передается в обработку или помещается в очередь до момента, когда он получит право на обслуживание в соответствии со своим назначенным уровнем. После возвращения из обработчика прерывания основной ход программы продолжается из той же точки, где он был прерван. Важно отметить, что одна следующая инструкция основной программы CPU всегда выполняется перед тем, как передать обслуживание обработчикам отложенных прерываний в очереди. Корректный возврат из прерывания (по команде RETI) будет возвращать PMIC в состояние, которое он имел перед входом в прерывание.

Все прерывания в XMEGA имеют собственные флаги. Когда наступает условие для прерывания, соответствующий флаг устанавливается, даже если это прерывание не разрешено. Для большинства прерываний установленный флаг автоматически сбрасывается после обслуживания прерывания. Можно сбросить флаг программно, записав в него логическую «1». Некоторые флаги прерываний не сбрасываются после передачи управления обработчику, а некоторые сбрасываются автоматически, когда ассоциированный регистр изменяется по чтению или записи. Детали следует смотреть в техническом описании на периферийные модули XMEGA.

Флаги прерывания устанавливаются всегда. Если в текущий момент обслуживается прерывание более высокого уровня, флаг будет удерживаться до тех пор, пока не наступит очередь для обслуживания этого прерывания. Даже если соответствующее прерывание не разрешено, его флаг все равно устанавливается и запоминается до тех пор, пока прерывание не будет разрешено или пока флаг не будет сброшен программно. Если глобально запрещены все прерывания, то флаги все равно устанавливаются и запоминаются до тех пор, пока все прерывания будут разрешены. Все отложенные прерывания в очереди потом будут обслужены в строгом соответствии с их установленным приоритетом обслуживания.

В контроллере прерываний XMEGA реализована поддержка немаскируемых прерываний (NMI), которые также должны быть предварительно разрешены. Они жестко привязаны к аппаратным ресурсам микроконтроллера и к ряду системных функций процессора. Подробную информацию о составе и работе NMI следует смотреть в техническом описании на каждый микроконтроллер. Например, для семейства A1 XMEGA доступно всего одно немаскируемое прерывание — когда пропадает тактовая частота процессора.

Немаскируемые прерывания обслуживаются независимо от установки I-бита и никогда не изменяют этот бит. Они имеют самый высокий уровень приоритета — другие прерывания не могут повлиять на работу обработчика NMI. Если в одно и то же время обслуживания требуют несколько NMI, то порядок их обработки фиксирован — вектор с наименьшим адресом имеет высший приоритет.

Время отклика CPU на поступивший запрос на прерывание для всех разрешенных прерываний составляет как минимум 5 тактов системной частоты. В это время содержимое программного счетчика сохраняется в стеке. Затем выполняется безусловный переход на программу — обработчик прерывания, что занимает еще 3 такта. Если микроконтроллер находится в одном из энергосберегающих режимов, время реакции на исполнение увеличивается еще на 5 тактов плюс на время выхода микроконтроллера из текущего режима энергосбережения. Перед началом обслуживания любого запроса на прерывание процессорное ядро всегда завершает выполнение текущей инструкции.

Внутри каждого из уровней (Low, Medium, High) все прерывания имеют приоритет. Когда несколько запросов на прерывание находятся в очереди, порядок их обслуживания определяется и уровнем прерывания, и приоритетом. Обслуживание прерываний может быть организовано в статическом или динамическом (Round Robin) порядке. В микроконтроллерах XMEGA немаскируемые, High и Medium прерывания имеют только статический порядок обслуживания. Для прерываний с уровнем Low программисту доступна статическая и динамическая организация очередности обслуживания.

Вектора прерываний (IVEC) размещаются по фиксированным адресам в памяти программ микроконтроллера. Для статического порядка значение адреса вектора определяет очередность обслуживания внутри одного и того же уровня прерываний, где наименьший по абсолютному значению адрес вектора имеет наивысший приоритет. Таблицу векторов прерываний для каждого конкретного микроконтроллера XMEGA следует смотреть в технической документации.

Чтобы избежать возможных проблем с недоступностью обработчиков прерываний уровня Low в соответствии со статическим порядком обслуживания, PMIC предоставляет для прерываний этого уровня интересную возможность динамического перераспределения очередности — Round Robin или процедуру «карусельной диспетчеризации». Для этого в состав контроллера прерываний добавлен специальный регистр INTPRI. Когда процедура диспетчеризации Round Robin разрешена, в этом регистре запоминается вектор преры-

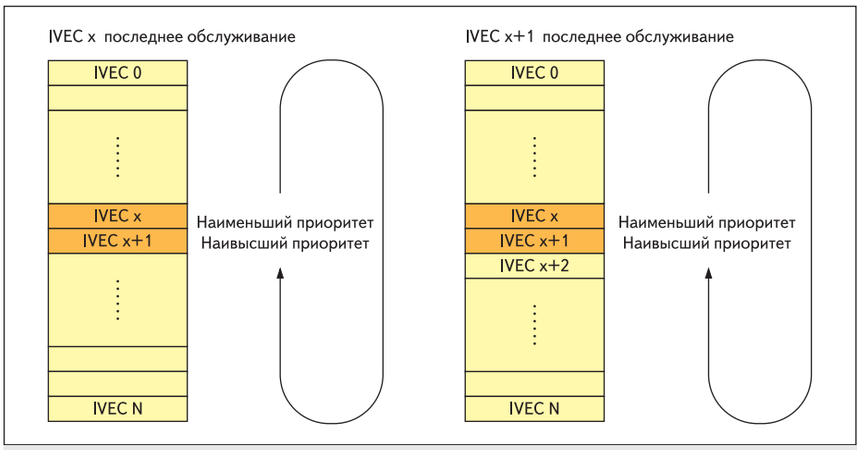


Рис. 2. Процедура «карусельной диспетчеризации» Round Robin

вания с уровнем Low, которое обслуживалось последним. Сохраненный вектор по отношению к его следующему вызову будет теперь иметь наименьший приоритет (рис. 2). Процедура Round Robin эффективно работает в тех случаях, когда в очереди находятся несколько разрешенных прерываний из уровня Low, которые будут обслуживаться в фиксированном циклическом порядке.

Отметим, что регистр PMIC.INTPRI доступен программисту для оперативного изменения порядка обслуживания и перестановок в очереди. Но он не сбрасывается к установкам по умолчанию, если процедуру «карусельной диспетчеризации» запретить в ходе выполнения программы. Поэтому для восстановления стандартного статического порядка (если необходимо) требуется записать в PMIC.INTPRI нулевое значение.

Практические примеры, демонстрирующие работу PMIC в микроконтроллерах XMEGA

По аналогии с разделом для портов ввода/вывода в качестве целевой платы использовался стартовый набор разработчика STK600 и интегрированная среда IAR Systems EWAVR 4.30. Для самостоятельной работы с примерами мы рекомендуем использовать руководство пользователя для STK600 («User Manual»).

Пример 1. Прерывание по переполнению таймера

Мигающие светодиоды на плате отладочного набора — это стандартная задача начального уровня. Реализуем ее с помощью управления прерываниями таймера/счетчика. Пример 1 показывает необходимые шаги для конфигурирования таймера (прерывание по переполнению). Используем таймер TCC0, порт D микроконтроллера для вывода и мигаем «младшим» светодиодом LED0:

```
#define LEDPORT PORTD
#define LEDMASK 0x01      // Используем LED0 для визуального
                          // отображения переключения

// Ассоциируем обработчик с вектором прерываний TCC0_OVF_vect
// (прерывание по переполнению):
#pragma vector = TCC0_OVF_vect
__interrupt void OverflowHandler( void )
  {
LEDPORT.OUTTGL = LEDMASK;  // Переключаем LED0 при
                           // переполнении таймера
                           // и вызове обработчика
                           // прерывания }


void main( void )
{
  // Конфигурируем порт вывода:
  LEDPORT.DIRSET = LEDMASK;
  LEDPORT.OUTSET = LEDMASK;

  // Конфигурируем таймер/счетчик 0:
  TCC0.PER = 10000; // Считаем до 10000 вместо 65536 для «ускорения».
  TCC0.CTRLA = TCC0.CTRLA & ~TC_CLKSEL_gm |
  TC_CLKSEL_DIV64_gc;  // Устанавливаем тактирование TCC0
                       // как CPUCLK/64.

  // Битовое поле Interrupt Level для периферийного узла XMEGA
  // должно быть установлено как Low, Medium или High.
  // Назначаем уровень прерывания для источника прерываний
  // (Low), но само прерывание пока не разрешаем:
  TCC0.INTCTRLA = TCC0.INTCTRLA & TC_OVFINTLVL_gm |
  TC_OVFINTLVL_LO_gc;  // Прерывание таймера/счетчика TCC0
                       // по переполнению, низкий уровень.

  // Разрешаем соответствующий уровень приоритета (Low)
  // для PMIC и разрешаем все прерывания
  PMIC.CTRL |= PMIC_LOLVLEN_bm;
  __enable_interrupt();

  // Запускаем бесконечный цикл…
  for (;;) {}
}
```

В целом, все просто и понятно, комментариев данный пример не требует.

Пример 2. Уровни прерываний

Если в программе обработки прерывания вдруг начинается бесконечный цикл, например ожидание нажатия клавиши пользователем, то это должно блокировать работу всего процессора, верно? Не совсем так для XMEGA! Пример 2 показывает, как взаимодействуют различные уровни прерываний. Здесь опять используются таймер TCC0 (в режиме сравнения) и порт D микроконтроллера для вывода. Но теперь переключаются три «младших» светодиода. Порт C микроконтроллера используем для ввода событий (нажатие клавиш на плате STK600), которые генерируют соответствующие запросы на прерывание. Тактирование TCC0 реализовано аналогично примеру 1.

```
#define LEDPORT PORTD
#define LEDMASKA 0x01 // переключение LED0 при прерывании
                      // от канала сравнения A.
#define LEDMASKB 0x02 // переключение LED1 при прерывании
                      // от канала сравнения B
#define LEDMASKC 0x04 // переключение LED2 при прерывании
                      // от канала сравнения C.

#define SWITCHPORT PORTC
#define SWITCHMASKA 0x01  // кнопка 0 для блокировки
                          // прерывания от канала сравнения A.
#define SWITCHMASKB 0x02  // кнопка 1 для блокировки
                          // прерывания от канала сравнения B.
#define SWITCHMASKC 0x04  // кнопка 2 для блокировки
                          // прерывания от канала сравнения C.

// Обработчик прерывания для TCC0, канал сравнения A.
#pragma vector = TCC0_CCA_vect
__interrupt void CompareMatchAHandler( void )
{
  // Переключаем соответствующий светодиод LED0
  LEDPORT.OUTTGL = LEDMASKA;
  // Бесконечный цикл, пока кнопка 0 остается нажатой.
  do {} while ((SWITCHPORT.IN & SWITCHMASKA) == 0x00);
}

// Обработчик прерывания для TCC0, канал сравнения B.
#pragma vector = TCC0_CCB_vect
__interrupt void CompareMatchBHandler( void )
{
  // Переключаем соответствующий светодиод LED1
  LEDPORT.OUTTGL = LEDMASKB;
  // Бесконечный цикл, пока кнопка 1 остается нажатой
  do {} while ((SWITCHPORT.IN & SWITCHMASKB) == 0x00);
}

// Обработчик прерывания для TCC0, канал сравнения C.
#pragma vector = TCC0_CCC_vect
__interrupt void CompareMatchCHandler( void )
{
  // Ппереключаем соответствующий светодиод LED2
  LEDPORT.OUTTGL = LEDMASKC;
  // Бесконечный цикл, пока кнопка 2 остается нажатой
  do {} while ((SWITCHPORT.IN & SWITCHMASKC) == 0x00);
}

void main( void )
{
  // Конфигурируем порты ввода/вывода:
  LEDPORT.DIRSET = LEDMASKA | LEDMASKB | LEDMASKC;
  LEDPORT.OUTSET = LEDMASKA | LEDMASKB | LEDMASKC;
  SWITCHPORT.DIRCLR = SWITCHMASKA | SWITCHMASKB |
                                          SWITCHMASKC;

  //Конфигурируем TCC0 :
  TCC0.PER = 10000;  // Значение «переполнения».
  TCC0.CCA = 5000;   // Устанавливаем одинаковые значения для
                     // всех каналов
  TCC0.CCB = 5000;   // сравнения, чтобы прерывания
                     // вызывались одновременно.
  TCC0.CCC = 5000;   // Это значение не должно быть больше
                     // содержимого регистра TCC0.PER.
  TCC0.CTRLB = TC_CCCEN_bm | TC_CCBEN_bm | TC_CCAEN_bm;

  // Назначаем различные уровни прерывания
  // (Low, Medium и High) для каналов сравнения:
  TCC0.INTCTRLB = (unsigned char) TC_CCCINTLVL_LO_gc |
          TC_CCBINTLVL_MED_gc | TC_CCAINTLVL_HI_gc;
  TCC0.CTRLA = TCC0.CTRLA & ~TC_CLKSEL_gm |
                                          TC_CLKSEL_DIV64_gc;

  // Разрешаем все уровни приоритета для PMIC и разрешаем
  // все прерывания.
  PMIC.CTRL |= PMIC_LOLVLEN_bm | PMIC_MEDLVLEN_bm |
                                          PMIC_HILVLEN_bm;
  __enable_interrupt();

  // Запускаем бесконечный цикл…
  for (;;) {}
}
```

После компиляции данного фрагмента кода и его запуска на STK600 можно наблюдать, как меняется поведение светодиодов. Если ни одна из кнопок SW0, SW1 и SW2 на плате STK600 не нажата, то все три светодиода «мигают» одновременно. При нажатии кнопки (или сразу нескольких кнопок) прерывание

«зацикливается» и состояние соответствующего светодиода фиксируется. Поведение остальных светодиодов будет определяться присвоенным статусом прерывания для соответствующих обработчиков. Так, например, при нажатии кнопки SW1 «зацикливается» обработчик прерывания с назначенным уровнем Medium. В результате состояние светодиодов LED1 (уровень Medium) и LED2 (уровень Low) «замораживается», а светодиод LED0 (уровень High) будет продолжать «мигать». Пример 2 показывает, как прерывания с более высоким уровнем приоритета могут приостанавливать работу обработчиков с более низким уровнем приоритета.

Пример 3. Round Robin или «карусельная диспетчеризация»

Когда несколько прерываний назначаются на один и тот же уровень, очередность их обслуживания определяет порядковый номер вектора прерываний. Чем меньше этот номер в абсолютном исчислении, тем раньше он обслуживается. Это — стандартный порядок. Но что произойдет, если какое-то прерывание требует очень много процессорного «внимания»? Тогда прерывания с более высоким порядковым номером никогда не будут обслуживаться. Это действительно будет так… до тех пор, пока не разрешена процедура Round Robin в PMIC. Теперь последний вектор прерываний, который обслуживался процессором, будет иметь самый низкий приоритет! Он перемещается в «хвост» кольцевой очереди. Пример 3 показывает, как использовать Round Robin для одновременной работы двух прерываний, даже если одно из них занимает почти 100% процессорного времени. Еще раз отметим, что процедура «карусельной диспетчеризации» доступна только для прерываний с установленным уровнем Low.

В примере 3 используются таймеры TCC0 и TCC1 (в режиме переполнения), порт D для вывода и порт C для ввода. Значения частоты тактирования таймеров здесь существенно различаются:

```
#define LEDPORT PORTD
#define LEDMASK0 0x01 // Светодиод LED0 ассоциирован с TCC0
#define LEDMASK1 0x02 // Светодиод LED1 ассоциирован с TCC1

#define SWITCHPORT PORTC
#define SWITCHMASK 0x04   // Кнопка SW3 используется для
                          // включения/выключения
                          // процедуры Round Robin.

// Ассоциируем обработчик с вектором прерываний TCC0_OVF_vect
// (прерывание по переполнению):
#pragma vector = TCC0_OVF_vect
__interrupt void Timer0Handler( void )
{
   // Так как этот обработчик вызывается очень часто, то мы
   // используем счетчик для понижения частоты «мигания»
   // светодиода до комфортного уровня восприятия.
   static long countdown = 0;
   if (countdown-- == 0) {
      countdown = 1000;
      LEDPORT.OUTTGL = LEDMASK0;
   }
}

// Ассоциируем обработчик с вектором прерываний TCC1_OVF_vect
// (прерывание по переполнению):
#pragma vector = TCC1_OVF_vect
__interrupt void Timer1Handler( void )
{
   LEDPORT.OUTTGL = LEDMASK1;
}

void main( void )
{
   // Конфигурируем порт вывода:
   LEDPORT.DIRSET = LEDMASK0 | LEDMASK1;

   // Устанавливаем TCC0 на очень частую генерацию запросов
   // на прерывание:
   TCC0.PER = 1;
   TCC0.CTRLA = TCC0.CTRLA & ~TC_CLKSEL_gm |
                                        TC_CLKSEL_DIV1_gc;
   TCC0.INTCTRLA = TCC0.INTCTRLA & ~TC_OVFINTLVL_gm |
                                        TC_OVFINTLVL_LO_gc;

   // Устанавливаем TCC1 на «умеренную» частоту генерации
   // запросов на прерывание:
   TCC1.PER = 10000;
   TCC1.CTRLA = TCC1.CTRLA & ~TC_CLKSEL_gm |
                                        TC_CLKSEL_DIV64_gc;
   TCC1.INTCTRLA = TCC1.INTCTRLA & ~TC_OVFINTLVL_gm |
                                        TC_OVFINTLVL_LO_gc;

   // Разрешаем уровень приоритета прерываний Low в PMIC
   // и разрешаем все прерывания:
   PMIC.CTRL |= PMIC_LOLVLEN_bm;
   __enable_interrupt();

   // Запускаем бесконечный цикл...
   for (;;) {
      // Считываем состояние кнопки и включаем/выключаем
      // процедуру Round Robin. Поскольку CPU будет выполнять
      // как минимум одну инструкцию между двумя вызовами
      // обработчиков, приведенный ниже код будет
      // выполняться, хотя и очень медленно.


      if ((SWITCHPORT.IN & SWITCHMASK) == 0x00) {
         PMIC.CTRL |= PMIC_RREN_bm;}
       else {
         PMIC.CTRL &= ~PMIC_RREN_bm;
         PMIC.INTPRI = 0;  // Сброс приоритета к значению
                           // по умолчанию. PMIC не сделает
                           // это за нас автоматически…}
      }
}
```

После компиляции фрагмента кода из примера 3 и его запуска на STK600 можно наблюдать, что светодиод LED0 постоянно переключается, а светодиод LED1 не меняет своего состояния. Если нажать и удерживать кнопку SW3, то светодиод LED1 начинает изредка «мигать». Это свидетельствует о том, что даже самое «забитое» по статусу прерывание иногда в порядке очереди получает право на обслуживание именно благодаря процедуре диспетчеризации Round Robin в микроконтроллерах XMEGA. При отпускании кнопки восстанавливается статический порядок обслуживания прерываний, и светодиод LED1 перестает «мигать» — процессор снова занят исключительно переключением LED0.

Дополнительную информацию об особенностях работы с программируемым многоуровневым контроллером прерываний PMIC в микроконтроллерах XMEGA можно найти в документации Atmel — Application Note AVR1305.

Система событий

Система обработки событий (Event System) у микроконтроллеров XMEGA разрабатывалась для того, чтобы максимально разгрузить центральный процессор. Она организована внутри кристалла как набор аппаратных средств, с помощью которых различные периферийные модули могут обмениваться между собой служебной или командной информацией, а также передавать и принимать данные. Система событий доступна в активном режиме работы микроконтроллера и в одном из энергосберегающих режимов — Idle. Индикация изменения состояния периферийного модуля у XMEGA носит название «событие». Система событий дает возможность по изменению состояния одного периферийного модуля автоматически вызывать определенные действия в других периферийных модулях. Причем характер принятия решения, а именно по какому из внешних возмущений может генерироваться определенное действие, программируется пользователем. При этом сохраняется исключительная гибкость в предоставлении прав каждому из периферийных узлов, которые подключены к системе событий.

В результате у разработчиков AVR XMEGA получилось предельно простое, но очень мощное аппаратное средство управления автономным функционированием периферии без участия CPU, DMA и контроллера преры-

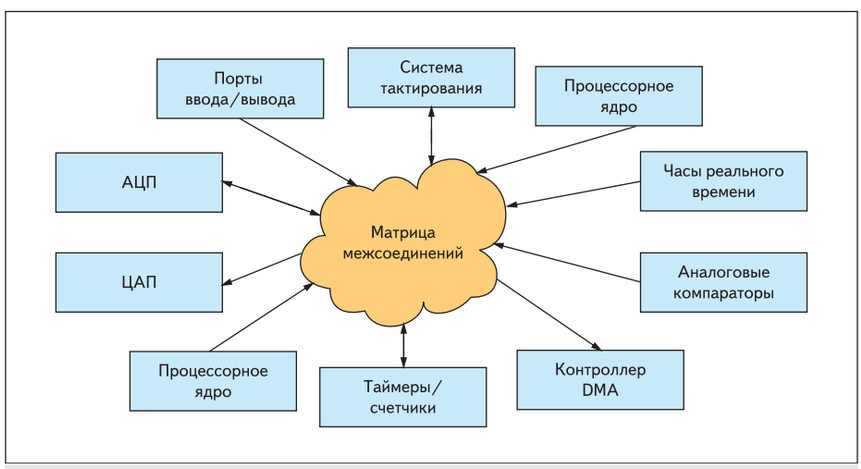


Рис. 3. Блок-схема Event System

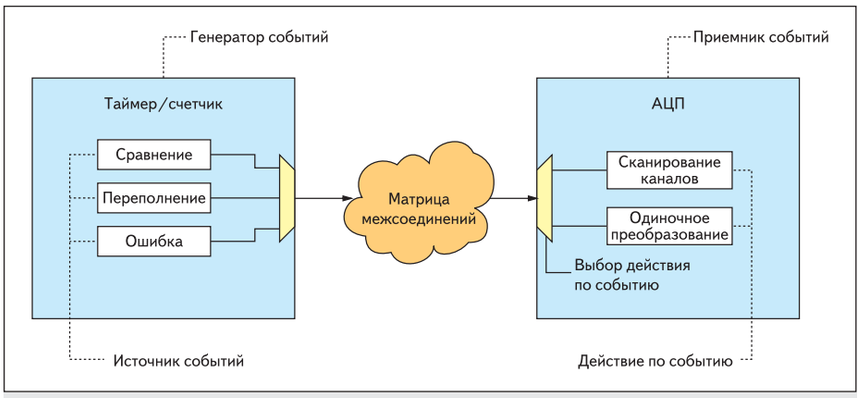


Рис. 4. Пример генератора, источника, приемника событий и действий по событию

ваний. С помощью Event System значительно экономится процессорное время и существенно разгружается система прерываний микроконтроллера, в основном за счет снижения количества запросов на прерывание. Блок-схема Event System показана на рис. 3.

Событие может генерироваться при изменении состояния сигнала на внешнем выводе микроконтроллера, при срабатывании блока захвата/сравнения таймера, при переключении аналогового компаратора и т. п. Поступающее на выбранный периферийный модуль событие также может инициировать различные действия — инкрементировать таймер, начать передачу данных через DMA, запретить выход сигнала ШИМ на внешний вывод микроконтроллера и т. п.

Аппаратный узел, от которого исходит событие, называется генератором события. Внутри каждого периферийного модуля может быть несколько источников событий. Например, у таймера/счетчика — это сравнение, переполнение или флаг ошибки. Аппаратный узел, на который поступает событие, называется приемником событий, а действие, которое он реализует, называется действием по событию. На рис. 4 проиллюстрированы все введенные понятия и их взаимодействие.

Существует два типа событий — Signaling и Data. События «Signaling» лишь показывают изменение состояния, а события «Data» содержат еще и дополнительную информацию. «Signaling» — это основной тип генерируемых событий. Подавляющее большинство периферийных модулей может генерировать и принимать лишь такие события. Поэтому, когда генерируется событие «Data» и оно приходит на периферийный модуль, который не может обрабатывать содержащуюся в событии информацию, оно автоматически интерпретируется как «Signaling». Приемники событий, которые могут обрабатывать оба типа событий, должны быть программным образом сконфигурированы на прием «Data» или «Signaling» перед началом работы. Подробную информацию следует смотреть в техническом описании на периферийные модули микроконтроллеров XMEGA.

В понятие «система событий» разработчики XMEGA включили генераторы событий, средства для передачи событий и приемники событий. События передаются между периферийными модулями при помощи специализированной матрицы соединений на кристалле XMEGA, которая носит название Event Routing Network. Она состоит из 8 мультиплексоров и собственно матрицы межсоединений, где реализовано до 8 параллельных каналов передачи сигналов (Event Channels) и все события «разводятся» на все мультиплексоры.

Центральный процессор (CPU) не является частью системы событий, но показан на блок-схеме Event System. Это означает, что у программиста имеется возможность управлять генерацией событий из программы микроконтроллера или в режиме debug, используя встроенные средства внутрисхемной отладки на кристалле XMEGA. Для этой цели в структуре Event System предусмотрены два специальных регистра — DATA и STROBE. Регистры доступны как в ходе выполнения программы, так и в режиме отладки, когда в AVR Studio можно изменять их содержимое в окне IO View и, соответственно, генерировать события. Каждый бит в регистрах соответствует определенному каналу Event System: бит 0 — каналу 0, бит 1 — каналу 1 и т. д. По умолчанию, в обоих регистрах записаны нулевые значения. Установка любого бита в любом из регистров инициирует определенное действие. Регистр DATA ответственен за тип генерируемого события и должен быть изменен первым, так как запись в регистр STROBE лишь генерирует событие. Подробнее соотношение и структура работы представлены в таблице 1.

Таблица 1. Соотношение и структура работы регистров DATA и STROBE

STROBE	DATA	События «Data»	События «Signaling»
0	0	Нет события	Нет события
0	1	Данные, тип 1	Нет события
1	0	Данные, тип 2	Генерация события
1	1	Данные, тип 3	Генерация события

Отметим, что все каналы в Event System аппаратно независимы. Поэтому у программиста имеется возможность генерировать до восьми синхронных событий, одновременно устанавливая биты желаемых каналов в регистрах DATA и STROBE.

Полная структурная схема системы событий микроконтроллеров XMEGA приведена на рис. 5. Здесь наглядно видно, как организована матрица межсоединений и каким образом мультиплексоры могут коммутировать события между генераторами и приемниками. Каждый мультиплексор имеет два регистра управления, доступные программисту. Регистр CHnMUX определяет, какое из входящих на мультиплексор событий коммутируется на соответствующий выходной канал событий Event Channel (n = 0 …7). Регистр CHnCTRL используется на выходе мультиплексора для организации дополнительных функций — фильтрации и квадратурного декодирования. Наличие восьми мультиплексоров означает, что у разработчика имеется возможность «развести» до восьми событий одновременно. Также можно «распараллелить» одно и то же событие на несколько мультиплексоров. Структура матрицы соединений в Event System одинакова для всех микроконтроллеров XMEGA. Конечно, представители разных подсемейств XMEGA имеют различные наборы периферии, но с точки зрения Event System это означает, что отсутствующий периферийный модуль не может генерировать и принимать события.

Как правило, генерация события занимает один период системной тактовой частоты. Отметим, что некоторые генераторы событий (например, изменившийся уровень на линии порта ввода/вывода) формально имеют возможность генерировать события постоянно. Забота об отслеживании таких ситуаций лежит на программисте. От момента генерации события до момента, когда оно «защелкивается» в приемнике, проходит два периода системной тактовой частоты. Первый — от наступления события до его регистрации матрицей межсоединений Event System по первому фронту тактового сигнала. Второй такт требуется на трансляцию события через Event Channel до приемника. Поэтому временны́е параметры работы системы событий практически точно предсказуемы. Это обстоятельство значительно повышает надежность принятия решений и обработки данных: чувствительные ко времени или к стабильности выполнения функции теперь становятся более прогнозируемыми, а значит, увеличивается надежность работы всего микроконтроллера в целом.

Рассмотрим подробнее дополнительные возможности Event System, которые «привязаны» к выходам мультиплексоров и соотносятся с каналами передачи событий Event Channels. Этими дополнительными возможностями управляет регистр CHnCTRL.

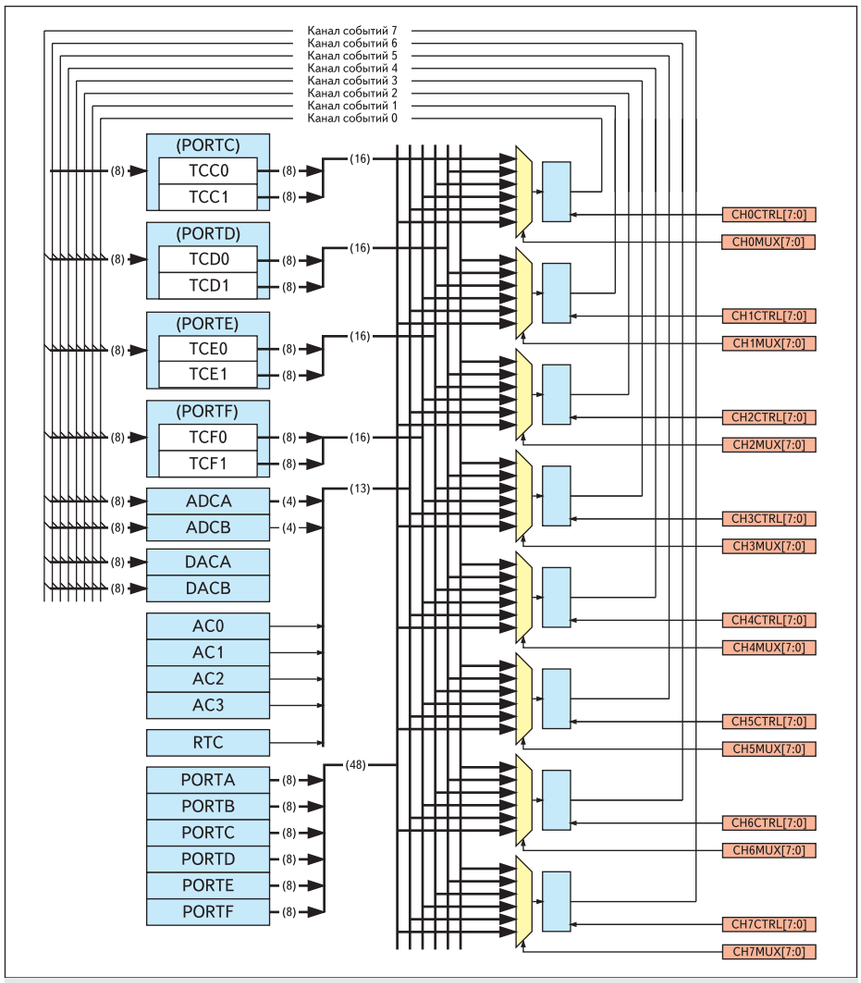


Рис. 5. Структурная схема системы событий в XMEGA

Первая дополнительная функция — цифровая фильтрация. Каждый канал системы событий снабжен программируемым цифровым фильтром. Поступающее событие будет опрашиваться определенное число раз перед тем, как оно будет передано дальше по направлению к приемнику события. Основное назначение такой фильтрации — опрос изменения состояния входных линий микроконтроллера. Другими словами, теперь можно аппаратно устранить дребезг контактов, которые непосредственно подключаются к выводам микроконтроллера. Такая функция избавит разработчика от необходимости писать соответствующую противодребезговую процедуру и снизит нагрузку на центральный процессор.

Работа цифрового фильтра разрешена всегда. Три младших бита регистра CHnCTRL определяют длительность временного промежутка, в течение которого источник события постоянно опрашивается. Событие будет отправлено в соответствующий канал только после того, как стабильность состояния источника события будет подтверждена в течение нескольких периодов периферийного тактового сигнала. Количество опросов (от одного до восьми) определяется тремя битами DIGFILT[2:0].

Вторая дополнительная функция — работа с квадратурно-кодированными сигналами. В состав системы событий для каналов с номерами 0, 2 и 4 включены три квадратурных декодера (QDEC). Это позволяет Event System декодировать входящие квадратурно-кодированные сигналы на линиях порта ввода/вывода и затем генерировать и посылать события «Data», которые могут интерпретироваться таймером/счетчиком и вызывать соответствующие действия по событию: считать «вверх», считать «вниз» или вырабатывать сигнал ошибки или сброса. Таблица 2 показывает, как события «Data» интерпретируются таймером/счетчиком при квадратурном декодировании.

Таблица 2. Интерпретация события «Data»

STROBE	DATA	События «Data»	События «Signaling»
0	0	Нет события	Нет события
0	1	Сигнал ошибки/сброс	Нет события
1	0	Считать «вниз»	Генерация события
1	1	Считать «вверх»	Генерация события

Напомним, что квадратурными называют сигналы, представляющие собой последовательности прямоугольных импульсов, которые смещены друг относительно друга на фазу 90°. Для создания таких сигналов применяют специальные изделия — энкодеры, которые используют кодирующие линейчатые маски и кодирующие диски. Например, двухканальные поворотные энкодеры широко используются для измерения углов поворота функциональных элементов в роботах и металлообрабатывающих станках. Выходные периодические импульсные последовательности, сдвинутые по фазе на 90°, позволяют с помощью внешних устройств обработки определять направление вращения вала и осуществлять двунаправленное позиционирование. Направление вращения определяется по фазовому соотношению сигналов в обоих каналах при анализе фронтов. Величина скорости определяется измерением интервала времени между импульсами или числом импульсов в пределах заданного временного интервала.

На рис. 6 показаны типичные квадратурные сигналы от поворотного энкодера

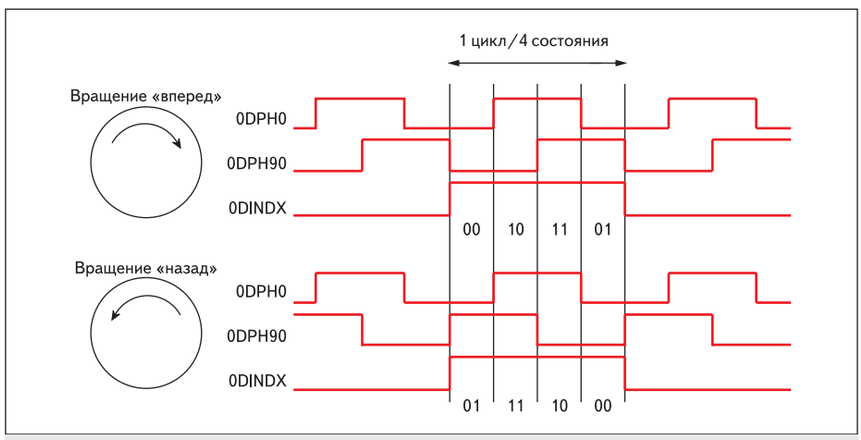


Рис. 6. Квадратурные сигналы от поворотного энкодера

и принцип распознавания направления вращения. В микроконтроллерах XMEGA им присвоены символические имена QDPH0 и QDPH90. В структуру Event System также включен еще один сигнал QDINDX, который генерируется один раз за оборот энкодера и может использоваться (в зависимости от требований конкретной задачи) для определения абсолютного положения, для генерации сигнала ошибки при декодировании, для индексации состояния счетчика (подсчет числа оборотов) или для его сброса.

Процедура организации квадратурного декодирования системой событий подробно описана в руководстве пользователя на микроконтроллеры XMEGA. Необходимо сконфигурировать как минимум две линии порта ввода/вывода микроконтроллера и таймер/счетчик на прием событий «Data» от Event System. В результате из регистра Count таймера/счетчика можно считывать угол поворота энкодера, а также принимать решения о необходимой реакции на изменение сигнала QINDX.

Практические примеры, демонстрирующие работу Event System в микроконтроллерах XMEGA

По аналогии с разделом для PMIC, в качестве целевой платы использовался стартовый набор разработчика STK600 и интегрированная среда IAR Systems EWAVR 4.30. Компания Atmel предлагает для знакомства с работой Event System ряд примеров, которые приведены в оригинальной документации Application Note AVR1001. Для самостоятельной работы с примерами мы также рекомендуем использовать руководство пользователя для STK600 («User Manual»).

Пример 1. Построение 32-разрядного таймера

Идея задачи очень проста — использовать событие, которое генерируется по переполнению одного таймера/счетчика, в качестве входного тактового сигнала для другого таймера/счетчика. Сигнал переполнения передается как событие «Signaling» по каналу Event System:

```
#define LEDPORT PORTD  // Используем порт D для вывода
                       // информации на светодиоды STK600

void main( void ){
   LEDPORT.DIR = 0xFF;   // Конфигурируем порт вывода
   LEDPORT.OUT = 0xFF;

// Конфигурируем приемник события (TCC1): какой канал будет
// источником тактирования периферийного узла и какое
// действие нужно осуществить при поступлении события:

   TCC1.CTRLA = TC_CLKSEL_EVCH0_gc; // Выбираем канал 0
// как источник тактирования TCC1

// Для системы событий необходимо явно указать, какое событие
// «разводится» на мультиплексор для каждого используемого
// канала событий. Выбираем переполнение таймера/счетчика
// TCC0 как входное событие для входа мультиплексора канала 0:

EVSYS.CH0MUX = EVSYS_CHMUX_TCC0_OVF_gc;
// Это весь код, который необходим для конфигурирования
// одного канала системы событий!

// Таймер/счетчик TCC0 используется в качестве 16 младших раз
// рядов нашего таймера. Для тактирования используем сигнал
// системной частоты, поделенной на 2:

   TCC0.PER = 0xFFFF;
   TCC0.CTRLA = TC_CLKSEL_DIV2_gc;

// Запускаем бесконечный цикл и «мигаем» светодиодами
 for(;;) {
    if (TCC0.CNT >= 0xFFF0) {
    __no_operation(); }

   LEDPORT.OUT = ~TCC1.CNTL;

   }

}
```

В целом все просто и понятно, комментариев данный пример не требует. Немного расширив код самостоятельно, можно организовать 48-разрядный или даже 64-разрядный счетчик.

Поскольку система событий не требует кода для пересылки событий, то момент переключения от TCC0 на TCC1 можно посмотреть в режиме debug, добавив в текст программы точку останова:

```
    if (TCC0.CNT >= 0xFFF0) {
    •__no_operation(); }
```

В процессе выполнения кода программа останавливается практически перед переполнением «младшего» таймера. Если сделать несколько последовательных шагов, таймер TCC0 переполнится и состояние светодиодов на плате STK600 изменится. Если раскрыть окно **IO view/TCC1** в AVR Studio, то можно наблюдать, как увеличивается CNT в момент переполнения TCC0.

Аналогично приведенному примеру с помощью системы событий можно реализовать 32-разрядный счетчик с функцией захвата. Программный код, реализующий эту задачу, приведен в Application Note AVR1001 (Example 3) и здесь он рассматриваться не будет.

Пример 2. Цифровая фильтрация

До сих пор мы использовали таймеры/счетчики для сравнения и генерации выходных цифровых последовательностей. В этом примере будет реализована функция захвата. Как известно, у таймеров/счетчиков имеются фиксированные входы, аппаратно соотнесенные с определенными выводами микроконтроллера. Это не всегда удобно при разводке целевой платы. Но можно использовать систему событий — подать на вход блока захвата таймера/счетчика входной сигнал от произвольной линии порта ввода/вывода, что значительно расширит диапазон доступных выводов микроконтроллера для реализации этой задачи.

Каждая линия порта ввода/вывода у XMEGA может быть запрограммирована на генерацию событий (или прерываний) по фронту, срезу, перепаду или низкому уровню сигнала. Генерируемое событие затем может быть направлено на таймер/счетчик, работающий в режиме захвата. Код, реализующий данную задачу, приведен в Application Note AVR1001 (Examples 1, 4) и копировать мы его здесь не будем. Кнопка на плате STK600, которая будет опрашиваться на нажатие, подключена к линии 0 порта D. Программируя цифровой фильтр на входе Event System на разные значения, можно наблюдать, сколько нажатий кнопки требуется для того, чтобы событие прошло далее на счетчик и инициировало захват. Удобнее всего это делать в режиме debug, раскрыв окно **IO view/Event System** в AVR Studio.

Пример 3. Программная генерация событий

События могут генерироваться «вручную» из программы или в режиме debug. Это осуществляется путем записи в регистры DATA и STROBE из программы или путем прямого доступа к содержимому регистров в окне отладчика AVR Studio (напомним, что регистр DATA должен записываться первым). Регистры DATA и STROBE содержат отдельные биты для каждого из каналов системы событий, то есть бит n соответствует каналу n. Поэтому существует возможность генерации событий на нескольких каналах одновременно, если установить в регистрах сразу несколько выделенных битов. Программная генерация может быть полезна для синхронизации событий и для отладки программы. Такие события завершаются в течение одного такта и будут превалировать над другими событиями в течение этого же такта:

```
#define LEDPORT PORTC  // Используем порт C для вывода
                       // информации на светодиоды STK600

volatile uint16_t capture_values[3];
volatile uint8_t adc_result;

// Простая функция, которая конфигурирует таймер в режим
// захвата. Канал событий — входной параметр:
void TC_init(volatile TC_t * tc, uint8_t EVSEL_CH){
    tc->CTRLD |= EVSEL_CH;
    tc->CTRLD |= TC_EVACT_CAPT_gc;
    tc->CTRLB = TC_CCAEN_bm;
    tc->CTRLA = TC_CLKSEL_DIV1_gc; }

void main( void ){

 uint16_t capture_values[3];
 uint8_t adc_result;

 // Конфигурируем порт вывода
LEDPORT.DIR = 0xFF;
LEDPORT.OUT = 0xFF;

// Теперь инициализацию таймера/счетчика (см. примеры 1 и 2)
// осуществляем с помощью функции:
TC_init(&TCC0, TC_EVSEL_CH0_gc);
TC_init(&TCC1, TC_EVSEL_CH1_gc);
TC_init(&TCD1, TC_EVSEL_CH2_gc);

/ * Эта функция конфигурирует ADCA на начало преобразования
  * в канале 0 по событию с канала 7 системы событий (подробно
  * инициализация АЦП будет рассмотрена в следующей части
  * статей) */
 ADC_init(&ADCA);

// Запускаем «медленный» таймер в качестве времязадающего
// узла для программной генерации событий
TCF0.PER = 0x3000;
TCF0.CTRLA = TC_CLKSEL_DIV256_gc;

do {} while ((TCF0.INTFLAGS & TC_OVFIF_bm) == 0);

for(;;) {

   do {} while ((TCF0.INTFLAGS & TC_OVFIF_bm) == 0);

    TCF0.INTFLAGS = TC_OVFIF_bm;
    LEDPORT.OUTTGL = 0xFF;
```

```
/* Код устанавливает биты в регистре STROBE для программной
  * генерации событий на каналах событий 0, 1, 2 и 7. Отметим,
  * что программно генерируемые события игнорируют установки
  * MUC канала событий. В действительности, мультиплексоры
  * вообще могут не конфигурироваться в случае использования
  * программно генерируемых событий */

EVSYS.STROBE = EVSYS_CHMUX0_bm | EVSYS_CHMUX1_bm |
                    EVSYS_CHMUX2_bm | EVSYS_CHMUX7_bm;

//… или проще, но не так наглядно то же самое можно написать
// как: EVSYS.STROBE =0x87;

   capture_values[0] = TCC0.CCA;
   capture_values[1] = TCC1.CCA;
   capture_values[2] = TCD1.CCA;
   adc_result = ADCA.CH0RESL;
   LEDPORT.OUT = ~adc_result;

 }

}
```

Выполнив компиляцию проекта, следует открыть файл ***.dbg*** (сгенерированный отладочный код в формате UBROF-8) в AVR Studio и установить точку останова, как показано ниже. Это необходимо для пошаговой отладки после генерации программно генерируемых событий:

```
    do {} while ((TCF0.INTFLAGS & TC_OVFIF_bm) == 0);
•   TCF0.INTFLAGS = TC_OVFIF_bm;
    LEDPORT.OUTTGL = 0xFF;
```

Теперь следует добавить переменные *adc_result* и *capture_values* в окно **Watch** для визуального наблюдения за изменениями результатов преобразования АЦП и захваченными значениями. После запуска кода на исполнение и остановки в назначенной точке следует открыть окно **IO view/Event System** и продолжать отладку по шагам. При этом будет видно, как регистр STROBE сначала записывается и затем сбрасывается на следующем такте. Если продолжать пошаговое выполнение программы, то можно наблюдать, как обновляются переменные *adc_result* и *capture_values* после поступления события на блок захвата таймера/счетчика и начала преобразования АЦП.

Содержимое регистра STROBE также может быть изменено непосредственно в ходе отладки программы, например, принудительной установкой битов этого регистра в окне **IO view/Event System**. Установленные биты будут автоматически сбрасываться на следующем такте. При таком способе «манипулирования» событиями следует иметь в виду, что захваченные значения таймера/счетчика не сохраняются, если буфер уже заполнен. В этом случае мы получим лишь сигнал ошибки.

Пример 4. Генерация синхронных событий

В предыдущем примере использовалась программная генерация событий для синхронизации событий, поступающих на различные каналы. Существует также возможность разрешать нескольким периферийным модулям использовать один и тот же канал системы событий в качестве источника событий. Тогда эти периферийные модули могут синхронизировать свои действия. Типичным примером может служить реализация функции захвата таймера/счетчика и одновременный старт преобразования АЦП, чтобы «проштемпелевать» факт начала преобразования меткой времени.

Приведенный ниже код во многом совпадает примером 3, но здесь конфигурируется мультиплексор седьмого канала системы событий на использование переполнения таймера/счетчика в качестве источника событий:

```
#define LEDPORT PORTC   // Используем порт C для вывода
                        // информации на светодиоды STK600

volatile uint16_t time_stamp;
volatile uint8_t adc_result;

// Конфигурируем таймер в режим захвата.
// Канал событий — входной параметр:
void TC_init(volatile TC_t * tc, uint8_t EVSEL_CH){
 tc->CTRLD |= EVSEL_CH;
 tc->CTRLD |= TC_EVACT_CAPT_gc;
 tc->CTRLB = TC_CCAEN_bm;
 tc->CTRLA = TC_CLKSEL_DIV8_gc; }

void main( void ){

// Переменные для значения захвата и результата
// преобразования АЦП
uint16_t capture_value;
uint8_t adc_result;

// Организуем небольшой кольцевой буфер для хранения результатов
uint8_t index = 0;
uint16_t result_buffer[2][32];


// Конфигурируем порт вывода
LEDPORT.DIR = 0xFF;
LEDPORT.OUT = 0xFF;

// Таймер/счетчик и АЦП должны использовать один и тот же
// канал системы событий по входу

// Инициализируем таймер/счетчик
TC_init(&TCC0, TC_EVSEL_CH7_gc);

/* Конфигурируем ADCA на начало преобразования в канале 0
  * по событию с канала 7 системы событий (подробно инициа-
  * лизация АЦП будет рассмотрена в следующей части статей) */
ADC_init(&ADCA);

/*****************************************************************
// Закомментированный в этом фрагменте код используется
// только во второй части примера
// Устанавливаем порт D на ввод информации, конфигурируем
// линию 0, назначаем событие для канала 7:
// PORTD.PIN0CTRL = PORT_ISC_FALLING_gc;
// EVSYS.CH7MUX = EVSYS_CHMUX_PORTD_PIN0_gc;
*****************************************************************/

/* Для системы событий необходимо явно указывать, какое
  * событие следует подавать на мультиплексор для каждого
  * используемого канала. Код ниже определяет это как перепол-
  * нение TCF0, что совпадает с установками АЦП и TCC0 */

EVSYS.CH7MUX = EVSYS_CHMUX_TCF0_OVF_gc;

/* Запускаем таймер, который будет использоваться в качестве
  * времязадающего узла для программно генерируемых событий */
TCF0.PER = 0x3000;
TCF0.CTRLA = TC_CLKSEL_DIV64_gc;

 for(;;) {

// Ждем, пока будет установлен флаг прерывания АЦП
// (т. е. преобразование было запущено и завершено)
   do{}
    while((ADCA.INTFLAGS & ADC_CH0IF_bm) != ADC_CH0IF_bm);

   ADCA.INTFLAGS |= ADC_CH0IF_bm;;
   LEDPORT.OUTTGL = 0xFF;

   capture_value = TCC0.CCA;      // считываем захваченную
                                  // метку времени
   adc_result = ADCA.CH0RESL;
   LEDPORT.OUT = ~adc_result; // Выводим результат
                              // преобразования
                              // на светодиоды STK600

   /* Сохраняем результат в кольцевом буфере — только для
     * того, чтобы иметь в окне Watch краткую «сводку» реальных
     * результатов преобразования АЦП и захваченных значений */

   result_buffer[0][index] = capture_value;
   result_buffer[1][index] = adc_result;

   index ++;
   index = index & 0x1F; // маскируем 3 старших бита для
                         // функционирования кольцевого буфера

 }

}
```

Выполним компиляцию проекта и загрузим файл ***.dbg*** в AVR Studio. Запустив программу на выполнение, можно наблюдать в окнах отладчика генерацию событий в ожидаемые моменты времени, начало преобразований аналого-цифрового преобразователя и захват в таймере/счетчике. Светодиоды на плате STK600 будут индицировать текущий результат преобразования АЦП.

Конечно, использование отдельного таймера для генерации событий с целью формирования метки времени с помощью другого таймера/счетчика, по меньшей мере, нелогично. Любой таймер/счетчик в микроконтроллерах XMEGA имеет свои собственные блоки захвата/сравнения, которые могут быть использованы для решения нашей задачи. Будем считать пройденное решение «академическим».

Модифицируем пример для того, чтобы старт преобразования АЦП и генерация временной метки инициировались событием от другого источника. В качестве генератора событий будет выступать кнопка на плате STK600, а источником события — факт нажатия этой кнопки. Для этого следует убрать комментарии со строк кода программы во фрагменте, отделенном символами «*», и закомментировать установки таймера TCF0 и предыдущие установки канала 7 системы событий. На плате STK600 необходимо также соединить гибким шлейфом контакт разъема PD0 с любой из кнопок SW. Тогда после компиляции текста, загрузки файла ***.dbg*** в AVR Studio и выполнения программы каждое нажатие выбранной нами кнопки запускает АЦП на преобразование и сохраняет метку времени об этом событии в буфере.

Дополнительную информацию об особенностях работы системы событий в микроконтроллерах XMEGA можно найти в документации Atmel — Application Note AVR1001.

Продолжение следует

Литература

1. Евстифеев А. В. Микроконтроллеры AVR семейств Tiny и Mega фирмы «Atmel». М.: Издательский дом «Додэка-XXI». 2004.
2. XMEGA Training Data // Atmel AVR Distributor Training. Atmel Norway, September 2007.
3. XMEGA Hands-On Session // Atmel AVR Distributor Training. Atmel Norway, September 2007.
4. www.atmel.com

Продолжение. Начало № 3`2008

Игорь КРИВЧЕНКО,
к. т. н.
ik@efo.ru

Микроконтроллеры XMEGA — новые возможности проверенного решения. Часть 3

Таймеры

Как мы упоминали в первой части цикла статей, количество 16-разрядных таймеров/счетчиков (T/C) у микроконтроллеров XMEGA для различных подсемейств составляет от 5 до 8 блоков. Разработчики XMEGA взяли базовый, удачно сбалансированный 16-разрядный T/C от «старших» AVR (например, у mega128 он обозначается T1), и добавили к нему ряд полезных усовершенствований. В результате получился мощный, функционально насыщенный, гибкий, удобный в конфигурировании и работе таймер/счетчик.

Все T/C в XMEGA имеют одинаковую структуру и выполняют одинаковые общие функции: формирование интервалов времени, сигналов заданной частоты и сигналов ШИМ; измерение временных параметров цифровых сигналов; синхронизацию с системой событий. Унификация T/C в XMEGA оказалась очень удобной, так как разработчику теперь не нужно помнить о различиях в инициализации и режимах работы этих периферийных блоков — изменяется лишь количество доступных таймеров на кристалле. Например, даже у «старших» AVR есть и 8-разрядные и 16-разрядные T/C, причем их инициализация и режимы работы различаются. А у XMEGA теперь все стандартизовано.

Блоки таймеров/счетчиков на кристаллах XMEGA бывают двух типов и обозначаются TCx0 и TCx1. Символические имена отражают тип блока (0 или 1) и порт ввода/вывода, с которым ассоциирован данный T/C. Например, TCD0 — таймер/счетчик типа 0, подключенный к PORTD. В микроконтроллерах XMEGA таймеры/счетчики доступны пользователю на портах PORTC, PORTD, PORTE или PORTF. Выходы TCx0 подключены к линиям 0–3 порта, а выходы TCx1 — к линиям 4 и 5. Таймер/счетчик типа 0 имеет 4 канала захвата/сравнения (CCn), а таймер/счетчик типа 1 — всего 2 канала. На этом различия заканчиваются. Все T/C для тактирования могут быть присоединены либо к сигналу периферийной тактовой частоты, либо к системе событий.

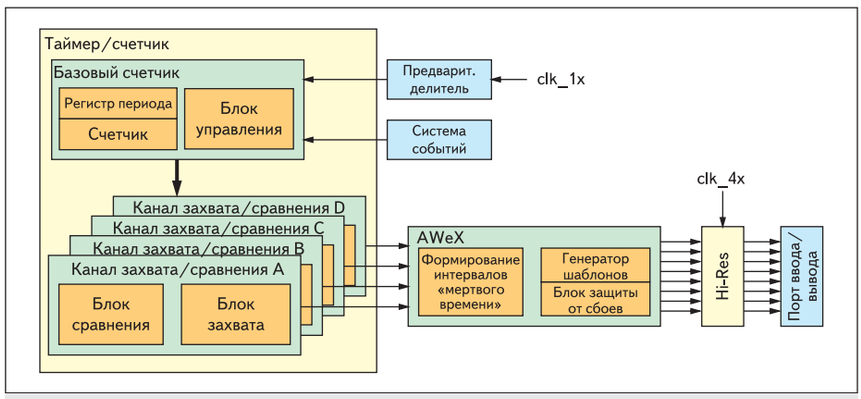


Рис. 1. Структурная схема таймера/счетчика в XMEGA

Поскольку таймеры/счетчики у XMEGA являются улучшенной версией 16-разрядных T/C «старших» микроконтроллеров megaAVR, то для общего знакомства и описания режимов работы 16-разрядных таймеров/счетчиков мы рекомендуем использовать описание блоков T1 (T3) у микроконтроллера ATmega128 (например, [1]). Общие функции: обращение к 16-разрядным регистрам, режим работы Normal, режим захвата и все режимы формирования ШИМ-сигналов — в настоящей статье рассматриваться не будут. Мы расскажем лишь о нововведениях и отличительных особенностях T/C у XMEGA.

Структура таймера/счетчика типа 0 с аппаратными расширениями и ассоциированными периферийными модулями (показаны серым цветом) приведена на рис. 1. Блок T/C состоит из базового счетчика и набора каналов захвата/сравнения. Базовый счетчик используется для подсчета тактовых циклов или событий; есть возможность изменять непосредственно в ходе работы направление счета (для этой цели предусмотрен специальный бит DIR) и период. Каналы CCn могут использоваться совместно с базовым счетчиком для реализации функций сравнения и генерации различных цифровых последовательностей (FRQ или PWM), либо выполнять функции захвата. Два T/C могут объединяться для построения 32-разрядного таймера/счетчика. При этом переполнение от младшего таймера в цепи может быть разведено через систему событий на старший таймер и может использоваться в качестве тактирующего сигнала для него. Совместно с блоками T/C могут работать два внешних модуля расширения — модуль высокого разрешения Hi-Res и модуль Advanced Waveform eXtension

В прошлом номере журнала «Компоненты и технологии» (№ 4 '2008) в статье «Микроконтроллеры XMEGA — новые возможности проверенного решения. Часть 2» автором были допущены следующие ошибки:

- Стр. 95, 2 столбец, 9 строка сверху, предложение «Реализовано управление скоростью нарастания **входного** сигнала». Правильно — «Реализовано управление скоростью нарастания **выходного** сигнала».
- Там же, 18 строка снизу, предложение «Можно разрешить инверсию сигнала на отдельной линии порта при вводе/выводе данных, а также разрешить ограничение скорости нарастания сигнала на ней при **вводе** данных». Правильно — «Можно разрешить инверсию сигнала на отдельной линии порта при вводе/выводе данных, а также разрешить ограничение скорости нарастания сигнала на ней при **выводе** данных».
- Там же, 13 строка снизу, предложение «Управление скоростью нарастания **входного** сигнала снижает энергопотребление узла». Правильно — «Управление скоростью нарастания **выходного** сигнала снижает энергопотребление узла».
- Там же, 11 строка снизу, предложение «В среднем после включения этого ограничения длительность фронта и среза **входного** сигнала увеличивается на 50–150% в зависимости...». Правильно — «В среднем после включения этого ограничения длительность фронта и среза **выходного** сигнала увеличивается на 50–150% в зависимости...».

Приносим свои извинения читателям.

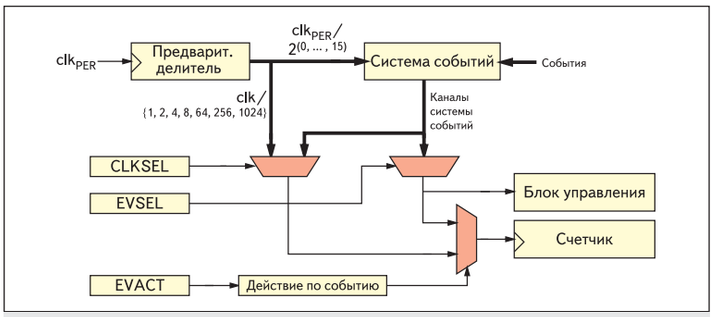


Рис. 2. Тактирование таймера/счетчика XMEGA

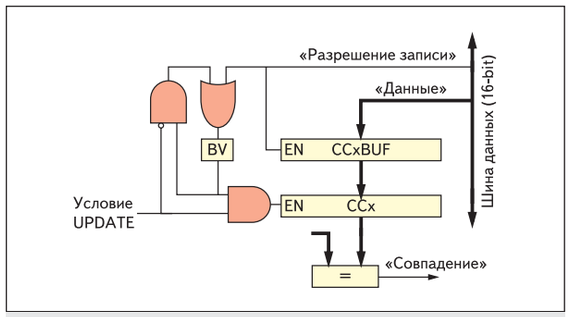


Рис. 3. Функция DBF в режиме сравнения

(AWeX) для формирования специализированных частотных сигналов для задач управления электродвигателями. Подробную структурную схему T/C рекомендуется смотреть в технической документации на микроконтроллер.

Начнем с организации тактирования. Все таймеры/счетчики у XMEGA могут тактироваться либо сигналом периферийной тактовой частоты, либо сигналами, которые поступают из системы событий микроконтроллера (рис. 2).

Периферийный тактовый сигнал сначала поступает на предварительный делитель, который является общим для всех модулей T/C на кристалле микроконтроллера. Каждый T/C имеет в регистре CTRLA четыре отдельных бита CLKSEL[3:0] для выбора в качестве источника тактирования для счетного регистра CNT одного из выходов предварительного делителя или одного из каналов системы событий. В режиме тактирования от Event System любое событие (внешний тактовый сигнал или сигнал на линии ввода/вывода) может выступать в качестве источника входных импульсов. События типа "Data" также могут управлять работой T/C, «заставляя» его осуществлять захват или сравнение, изменять направление счета (вверх или вниз) и работать с квадратурно-кодированными сигналами. Выбор типа реакции T/C устанавливается битами EVACT в регистре CTRLD.

При старте микроконтроллера тактирование таймера/счетчика отключено (установка «по умолчанию»), он находится в состоянии OFF.

Еще одна полезная особенность таймеров/счетчиков у XMEGA — наличие специального регистра периода PER, который непосредственно связан со счетным регистром CNT. В регистр периода записывается программируемое значение уставки (TOP), по достижении которой таймер/счетчик должен переходить к очередному циклу работы (в документации на микроконтроллеры XMEGA это аппаратное условие обозначается UPDATE): при счете «вверх» он начинает инкрементировать с нуля, а при счете «вниз» — декрементировать от значения, записанного в регистр PER. Наличие регистра PER предоставляет большое удобство, так как непосредственно в ходе работы можно изменять значение периода таймера, просто записывая новое значение в регистр PER. Операция записи в этот регистр выполняется немедленно.

Еще одно нововведение у XMEGA — дублирующее буферирование (Double Buffering или DBF) части регистров таймера/счетчика. Дублированы регистр PER и все регистры каналов CCn — для каждого из них добавлен свой буферный регистр PERBUF и CCnBUF соответственно. Каждый буферный регистр имеет собственный ассоциированный флаг BV (Buffer Valid), установка которого сигнализирует о том, что PERBUF или CCnBUF содержит новое значение, готовое к копированию в основной регистр PER или CCn.

Для регистра PER и для регистров CCn при работе таймера/счетчика в режиме сравнения (рис. 3) флаг BV устанавливается, когда новые данные записываются в буферный регистр, и сбрасывается, когда содержимое буферного регистра копируется в основной регистр. Буферный регистр мгновенно копирует свое содержимое в основной регистр при достижении счетной последовательностью значения TOP или BOTTOM в соответствии с работой внутренней логики управления T/C. Таким образом, записываемое число постоянно хранится в буферном регистре, а обновление основного регистра происходит только при достижении счетчиком верхнего или нижнего порога. Это происходит каждый раз автоматически до момента обновления самого буферного регистра PERBUF (или CCnBUF). Такая синхронизация помогает устранить искажения на выходе таймера/счетчика (переменная длина импульса, ложная частота, случайный выброс) в формируемой временной цифровой последовательности, будь то сигнал заданной частоты или ШИМ-сигнал.

Double Buffering существенно повышает точность работы T/C. Покажем это применительно к регистру периода PER. На рис. 4, 5 приведены диаграммы работы T/C в режиме счетчика без использования функции DBF и с ней. Если дублирующее буферирование не используется, то любое обновление периода путем записи нового значения в регистр PER будет осуществляться немедленно (рис. 4). Это может вызывать незапланированное циклическое обновление таймера (переход от всех «1» ко всем «0») и, следовательно, генерацию «странных» сигналов или ложных частот в спектре формируемого сигнала. Если DBF используется, то значение в буферном регистре может быть изменено в любое время, но основной регистр PER будет обновляться только при наступлении условия UPDATE (рис. 5), которое в данном случае возникает при достижении счетчиком значения BOTTOM. Это позволяет избежать проблем с частотой и формой генерируемого выходного сигнала.

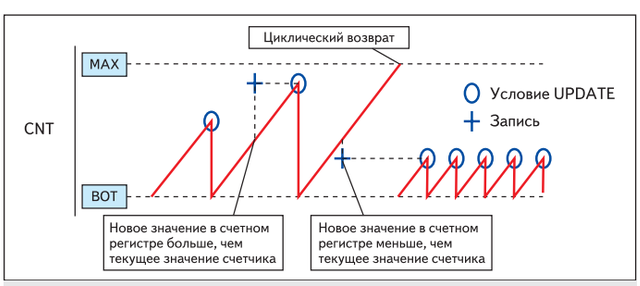


Рис. 4. Режим работы таймера/счетчика без DBF

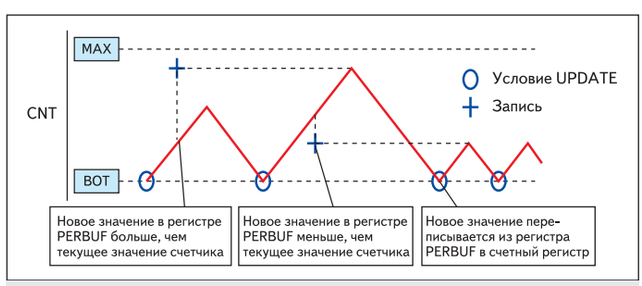


Рис. 5. Работа таймера/счетчика с включенным DBF

Когда каналы CCn работают в режиме захвата, то используется похожий механизм Double Buffering. Здесь основной регистр CCn и соответствующий ему буферный регистр CCnBUF образуют FIFO (рис. 6), в котором захват выполняется регистром CCnBUF, и при этом устанавливается флаг BV. Но текущее захваченное значение копируется из CCnBUF в основной регистр только в случае, если CCn пустой (флаг IF не установлен). Такая аппаратная организация позволяет сохранять два значения захвата: флаг IF сигнализирует о том, что регистр CCn содержит захваченное значение, которое еще не прочитано, а флаг BV — что в регистре CCnBUF находится текущее (последнее) захваченное значение.

При установке флага IF также могут генерироваться соответствующий запрос на прерывание (если разрешено) или запрос контроллеру DMA на выполнение транзакции. Флаг IF автоматически сбрасывается при считывании содержимого основного регистра CCn, что автоматически разрешает немедленное аппаратное копирование в CCn нового значения захвата, которое до этого сохранялось в буферном регистре.

Таймер/счетчик может определять переполнение буфера на любом из каналов захвата/сравнения. Если случается, что установлены оба флага BV и IF, и в это же время осуществляется новый захват, то сохранить новое значение «метки» счетной последовательности просто негде — возникает ситуация переполнения буфера. В этом случае новое значение захвата игнорируется и нигде не

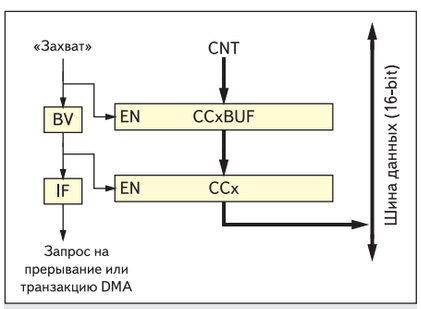


Рис. 6. Функция DBF в режиме захвата

запоминается, но устанавливается специальный флаг ERRIF в регистре INTFLAGS, что может генерировать запрос на прерывание типа «ошибка» (если разрешено).

В микроконтроллерах XMEGA два таймера/счетчика могут использоваться совместно для реализации 32-разрядного захвата. При этом сигнал переполнения «младшего» таймера в цепочке через систему событий подключается на вход тактирования «старшего» таймера. И тут есть одна тонкость. Поскольку все события в XMEGA конвейеризированы, то «старший» таймер будет обновляться спустя 1 период периферийного тактового сигнала после того, как произошло переполнение «младшего» таймера. Это может давать погрешность в определении реального значения захвата. Для компенсации этой «технологической» задержки факт наступления события типа «захват» для «старшего» таймера должен быть задержан на такое же время. Это осуществляется путем установки специального бита EVDLY (Event Delay) в регистре управления CTRLD для выбранного таймера.

И основные регистры всех каналов захвата/сравнения у XMEGA, и соответствующие буферные регистры доступны через общее адресное пространство ввода/вывода микроконтроллера. Это обеспечивает гибкость при работе с таймером/счетчиком, включая эффективное использование дублирующего буферирования. Подробнее структуру организации Double Buffering следует смотреть в технической документации на микроконтроллеры XMEGA.

Таймеры/счетчики могут генерировать и события, и запросы на прерывание. Событие будет сгенерировано при тех же условиях, при которых возможна генерация запросов на прерывание: совпадение кодов в регистре сравнения и в счетном регистре, достижение значения TOP в регистре PER и обнуление счетного регистра. Каждый канал захвата/сравнения имеет свое отдельное прерывание. В дополнение T/C может генерировать запрос на прерывание по сигналу ошибки. Поступающие на T/C события от Event System также могут интерпретироваться по-разному: инкрементировать или декрементировать счетчик, выполнять функцию захвата, а также обработать информацию, которая поступает в виде квадратурно-кодированных сигналов.

Флаги запросов на прерывание могут быть использованы для инициирования транзакций с помощью модуля DMA. Используемые флаги запросов на прерывание: по переполнению или обнулению, по сигналу ошибки, а также при появлении флага прерывания в любом из каналов захвата/сравнения. Подробнее все это можно посмотреть в технической документации на микроконтроллер.

В микроконтроллерах XMEGA реализована возможность прямого программного управления работой таймера/счетчика — введен набор специальных команд, которые непосредственно управляют аппаратной логикой T/C. Для генерации этих команд используются два бита CMD[1:0] в статусном регистре CTRLFSET.

Команда “Force update” используется для форсирования наступления условия UPDATE. Буферные регистры при этом немедленно копируются в основные регистры PER и CCn безотносительно к их текущим значениям. Это может быть полезно в случаях, когда необходимо обновить и регистр периода, и регистры сравнения точно в одно и то же время — при формировании программистом условия UPDATE. Для этого следует предварительно установить в регистре CTRLFSET специальный бит блокировки LUPD (Timer Lock Update) — это заблокирует автоматический аппаратный UPDATE из буферных регистров. Затем можно записать все необходимые значения в буферные регистры, снять бит LUPD и инициировать команду “Force update” путем записи в биты CMD[1:0]. Отметим, что в режиме захвата команда “Force update” будет воздействовать только на содержимое регистров PERBUF и PER.

Команда “Force restart” предназначена для рестарта текущего периода формирования выходной цифровой последовательности. Она сбрасывает содержимое счетного регистра CNT, бит направления счета DIR и все выходы каналов сравнения к значениям по умолчанию, то есть, попросту обнуляет их. Другая команда “Force hard reset” приводит содержимое всех регистров в выбранном блоке T/C к их значениям по умолчанию. То есть, для выбранного таймера/счетчика это будет означать его полный индивидуальный сброс. Во избежание нежелательных «последствий» для прикладной программы эта команда может быть выполнена только в случае, если тактирование выбранного T/C остановлено (состояние OFF). В противном случае команда “Force hard reset” не будет иметь никакого эффекта.

В заключение обзора таймеров/счетчиков расскажем о двух дополнительных аппаратных модулях, которые включены на кристаллы XMEGA и работают в совокупности с таймерами/счетчиками, но являются независимыми. Это модули расширения AWeX и Hi-Res.

Модуль расширения Hi-Res (Hi-Resolution) имеется у всех таймеров/счетчиков XMEGA вне зависимости от их типа. Выходная цифровая последовательность (FRQ или PWM), генерируемая таймером, может быть «пропущена» через этот блок расширения перед тем, как будет подключена к выходным линиям порта ввода/вывода (рис. 1). Модуль Hi-Res тактируется сигналом, частота которого в 4 раза превышает частоту периферийного тактового сигнала. Это позволяет увеличить разрешение формируемых сигналов ШИМ в 4 раза. Заметим, что данный модуль не повышает значение частоты, а только увеличивает разрешение. Например, можно получить сигнал ШИМ с частотой 31,25 кГц и разрешением 12 бит, или ШИМ-последовательность с частотой 500 кГц и разрешением 8 бит. Все генерируемые частотные сигналы будут синхронизированы с сигналами основной и периферийной тактовой частот. Помимо выигрыша в точности, такое решение еще и уменьшает энергопотребление, так как на высокой частоте работают только два младших разряда получающегося «тандемного счетчика». Управление подключением модуля Hi-Res осуществляется установкой соответствующих битов HREN[1:0] в регистре CTRLA модуля Hi-Res.

Отметим, что установка любого или сразу обоих битов HREN подключит модуль расширения Hi-Res ко всему порту ввода/вывода.

Это означает, что оба таймера TCx0 и TCx1, ассоциированные с конкретным портом ввода/вывода, в случае их одновременной работы на генерацию выходных сигналов FRQ или PWM, должны разрешать функцию Hi-Res.

Более подробно рассмотрим второе аппаратное расширение — Advanced Waveform eXtension. Это расширение доступно только для таймеров/счетчиков типа «0», которые ассоциированы с портами ввода/вывода PORTC и PORTE. Они имеют обозначения AWeXC и AWeXE соответственно. Модуль AWeX предназначен для работы в двух режимах:

- DTI — генерация верхнего и нижнего интервалов «мертвого времени» для вставки в выходную цифровую последовательность;
- Pattern Generation — генерация цифровых шаблонов (синхронная комбинация битов).

Обе опции необходимы для удобного и надежного управления приводами. В первом случае — для управления инверторами мощных электродвигателей; во втором — для шаговых, вентильно-индукторных (SRD) двигателей и бесщеточных электродвигателей постоянного тока (BLDC), для управления которыми требуется специальный блок, выполняющий коммутацию обмоток двигателя по определенному алгоритму в зависимости от положения ротора.

Структурная схема модуля расширения AWeX показана на рис. 7. Его работа в режиме DTI возможна только в случае, когда таймер/счетчик сконфигурирован на формирование ШИМ-сигналов. Когда работа AWeX разрешена, выход каждого канала TCx0 «расщепляется» на два комплементарных выхода. Эта пара выходных сигналов проходит через блок вставки интервалов мертвого времени (DTI), что позволяет генерировать неинвертированный сигнал для нижнего ключа (LS) и инвертированный сигнал для верхнего ключа (HS) со вставками интервалов «мертвого времени» между активными состояниями ключей. Выход блока DTI будет перехватывать управление линиями ассоциированного порта ввода/вывода за исключением функции инверсии выходного сигнала на линиях порта — бит INVEN регистра PINnCTRL всегда будет выполнять свою функцию, и может быть использован для инверсии выходного сигнала.

Работа блока DTI гарантирует, что верхний и нижний ключи никогда не будут включены одновременно. Блок DTI состоит их 4 одинаковых генераторов интервала «мертвого времени» — по одному на каждый канал захвата/сравнения у TCx0. 8-разрядные регистры «мертвого времени» имеются для обоих типов ключей: верхнего (DTHS) и нижнего (DTLS). Оба регистра буферизованы, то есть снабжены дополнительными регистрами DTHSBUF и DTLSBUF.

Регистры «мертвого времени» являются общими для всех четырех каналов DTI и определяют количество тактов сигнала периферийной тактовой частоты, в течение которых должен присутствовать интервал «мертвого времени». Интервалы «мертвого времени» для верхнего и нижнего ключей могут устанавливаться раздельно, тогда они могут иметь разные значения. Допускается одновременная установка интервала «мертвого времени» для обоих ключей, но в этом случае времена будут одинаковыми. Для этого используется еще один байтовый регистр DTBS, запись в который приводит к одновременному обновлению содержимого регистров DTLS и DTHS. Регистр DTBS также буферизован, его буферный регистр обозначается как DTBSBUF. Рис. 8 поясняет принцип формирования выходной ШИМ-последовательности для инвертора со вставками интервалов «мертвого времени».

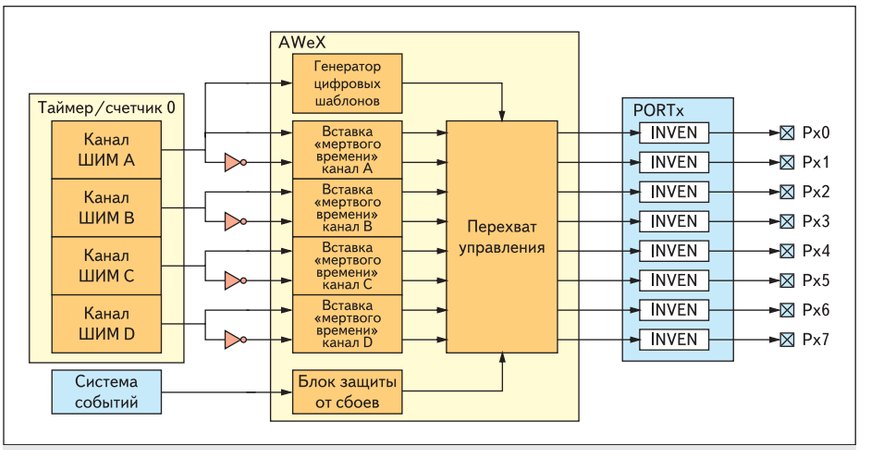


Рис. 7. Блок-схема модуля расширения AWeX

Все четыре канала модуля расширения AWeX могут программироваться и работать независимо друг от друга. Но в ряде случаев может потребоваться выдача одинаковой ШИМ-последовательности сразу на все каналы блока DTI. Для этой цели служит бит CWCM в регистре управления CTRLA блока AWeX. Если этот бит установлен, то только выход канала CCA будет использоваться как единый источник сигнала ШИМ сразу для всех четырех каналов DTI. Поступающие на блок DTI выходные сигналы с других каналов захвата/сравнения (B, C и D) будут игнорироваться. Такой режим в микроконтроллерах XMEGA носит название Common Waveform Channel Mode или CWCM.

Когда в AWeX инициализирована подсистема Pattern Generation для аппаратной генерации синхронных цифровых последовательностей (шаблонов) для выдачи сигналов на порт ввода/вывода (режим PGM), то весь модуль расширения работает немного по-другому. Во-первых, требуется установить бит PGM в регистре управления CTRLA блока AWeX. Это заблокирует работу блока DTI и передаст возможность управления линиями ввода/вывода порта блоку Pattern Generation. Затем, чтобы передать управление линией «x» порта ввода/вывода модулю AWeX, необходимо дополнительно установить соответствующий бит «x» в специальном регистре OUTOVEN. Отметим, что направление работы линии порта (на ввод или на вывод информации) автоматически не перехватывается при установке бита OUTOVENx. Программист должен заранее позаботиться о конфигурации линии порта для работы на выход, иначе на выводы микроконтроллера ничего поступать не будет. Буферный регистр DTLSBUF используется в этом режиме для хранения цифрового шаблона, который требуется передавать на выход порта микроконтроллера. Другой буферный регистр DTHSBUF используется в этом режиме для хранения копии установок режимов работы линий ввода/вывода ассоцииро-

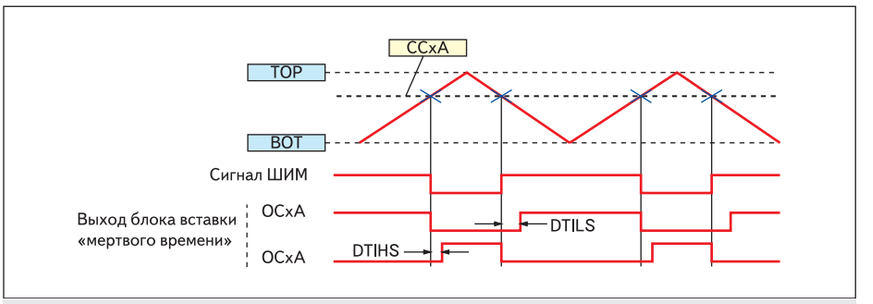


Рис. 8. Формирование интервалов «мертвого времени» в модуле AWeX

ванного порта микроконтроллера — то есть регистра PINxCTRL. Эта копия настроек порта будет использована только в случае, если соответствующий бит в регистре DTLSBUF не установлен (и, следовательно, ему не разрешен перехват управления этой линией порта).

Содержимое обоих буферных регистров копируется в соответствующие им основные регистры каждый раз при наступлении условия UPDATE — подобно всем остальным регистрам T/C, снабженным функцией дублирующей буферизации. Это обеспечивает полную синхронизацию с временными параметрами выбранного режима формирования выходной цифровой последовательности. Если синхронизация не требуется, то программист может непосредственно обновлять содержимое регистров OUTOVEN и PINxCTRL порта. Блок-схему работы модуля расширения AWeX в этом режиме можно посмотреть в технической документации на микроконтроллер.

Модуль AWeX также снабжен быстрым встроенным механизмом защиты от сбоев. Для этого в его состав введен дополнительный аппаратный узел Fault Protection. Как видно из рис. 7, этот узел непосредственно подключен к системе событий микроконтроллера. Выбор канала, который подключается к Fault Protection, определяется содержимым регистра FDEMASK: установка бита «n» означает подключение канала «n» системы событий. Если требуется, чтобы входной сигнал на узел защиты от сбоев поступал от нескольких каналов системы событий одновременно, то для этой цели биты в регистре FDEMASK могут объединяться по «ИЛИ», разрешая доступ многочисленным источникам событий к узлу Fault Protection в одно и то же время.

Когда приложение определяет наступление сбоя, то на вход узла поступает событие от Event System микроконтроллера. Сразу же устанавливается флаг ошибки FDF в регистре STATUS модуля AWeX, и после этого активируется соответствующее действие Fault Protection, выбор которого определяется установками битов FDACT[1:0] в регистре FDCTRL:

- ничего не делать — игнорировать сбой;
- сбросить регистр OUTOVEN (то есть запретить перехват управления линиями порта узлом DTI). В результате выход будет сконфигурирован в соответствии с установками порта ввода/вывода;
- сбросить регистр управления направлением передачи данных DIR ассоциированного порта. В результате линии порта переводятся в высокоимпедансное состояние.

Установка флага FDF автоматически вызывает установку флага ошибки у всего таймера/счетчика. При этом также генерируется запрос на прерывание (если разрешено).

Отметим, что от момента генерации события в периферийном блоке микроконтроллера (сбой) до момента, когда Fault Protection инициирует соответствующее действие, проходит максимум (!) два периода периферийной тактовой частоты. Аппаратный узел Fault Protection полностью независим от работы процессорного ядра и контролера DMA, но для функционирования ему требуется сигнал периферийной тактовой частоты.

Дополнительную информацию об особенностях работы с таймерами/счетчиками и аппаратными модулями расширения в XMEGA можно найти в документации Atmel — Application Note AVR1306 и AVR1311, а также в описании на микроконтроллер.

В качестве иллюстрации рассмотрим пример работы модуля AWeX и подсистемы Pattern Generation. Для его реализации использовался стартовый набор разработчика STK600. Исходный текст написан на Си и скомпилирован в интегрированной среде IAR Systems EWAVR 4.30. Необходимые установки на плате STK600 (перемычки, соединительные кабели, переключатели и т. п.) в данной статье не описываются, мы рекомендуем использовать руководство пользователя для STK600 ("User Manual").

Пример № 1. Генерация синхронных цифровых последовательностей

Здесь мы будем использовать режим PGM для генерации синхронных цифровых шаблонов и вывода их на порт PORTC, с которым ассоциирован таймер/счетчик TCC0. Выход ШИМ-сигнала с канала CCA этого таймера разведем на все линии порта и организуем перехват управления работой всех линий порта. Режим PGM идеально подходит для управления двигателями типа BLDC, когда требуется в зависимости от положения ротора реализовать коммутацию обмоток двигателя в определенной последовательности. Положение ротора может определяться, например, с помощью датчиков Холла. Но в данном примере мы будем использовать более простой прием — управлять изменением шаблона с помощью кнопки на плате STK600. В реальном приложении с двигателем шаблоном будет управлять необходимая коммутационная последовательность. В качестве порта ввода будем использовать PORTB, а для наглядной индикации генерируемого шаблона — PORTC, который нужно соединить гибким шлейфом со светодиодами на плате STK600.

```
#define SWITCHPORT PORTB  // Используем порт B для
                          // подключения кнопки SW0
                          // на плате STK600.

void TCC0_init()          // Инициализируем таймер TCC0 —
                          // выбираем нужный режим
                          // работы и канал CCA.
{
TCC0.CTRLB = TCC0.CTRLB & ~TC_WGMODE_gm |
                                    TC_WGMODE_SS_gc;
TCC0.CTRLB |= TC_CCAEN_bm;
TC_ConfigClockSource(&TCC0, TC_CLKSEL_DIV1_gc);
TCC0.PER = 60000;

PORTC.DIRSET = 0xFF;   // Линии порта должны быть
                       // сконфигурированы на вывод
                       // информации.
PORTC.OUTSET = 0xFF;   // Выключаем все светодиоды.
}
// Эта функция формирует новый выходной цифровой шаблон,
// исходя из входного параметра index.
uint8_t GetNewPattern(uint8_t pattern_index)
{
  uint8_t new_pattern;
  switch(pattern_index)
      {
      case 0: new_pattern = 0x0F; break;
      case 1: new_pattern = 0x3C; break;
      case 2: new_pattern = 0xF0; break;
      case 3: new_pattern = 0xC3; break;
      }
  return new_pattern;
}

void main( void )
{
// Эти переменные используются для хранения № варианта
// шаблона и значения нового выходного шаблона
uint8_t pattern_index;
uint8_t new_pattern;

  /* Для изменения длительности рабочего цикла сигнала ШИМ
  * (характер «моргания» светодиодами) в ходе программы
  * используем переменную для инкремента/декремента
  * регистра CCA один раз за период */
  int16_t pwm_delta = 300;

  // Устанавливаем модуль расширения AWeX
  // в режимы работы CWCM и PGM
  AWEXC.CTRL |= AWEX_CWCM_bm;
  AWEXC.CTRL |= AWEX_PGM_bm;

/* Необходимо разрешить работу выходных каскадов у тех
*каналов блока DTI, которые будет использовать модуль AWEX.
* В данном примере разрешаем функционировать всем каналам */

AWEXC.CTRL |= AWEX_DTICCDEN_bm |
              AWEX_DTICCCEN_bm | AWEX_DTICCBEN_bm |
                                 AWEX_DTICCAEN_bm;

  /* Необходимо явно указать AWEX, для каких линий порта он
  * должен перехватывать управление ими. Напомним, что эти
  * линии должны быть сконфигурированы на вывод
  * информации! */

AWEXC.OUTOVEN = 0x0F;       // Начальное значение
                            //  для перехвата управления.

TCC0_init();                 // Инициализируем таймер

  for(;;){
  if((SWITCHPORT.IN & 0x01) == 0)  // Определяем, нажата ли
                                   // кнопка SW0.
  {
  __delay_cycles(1000000);   // Задержка ~500 мс
                             // (2 МГц, значение для XMEGA
                             // по умолчанию)
  pattern_index++;
  pattern_index = (pattern_index & 0x03);  // Используем только
                                           // 2 бита для маски ->
                                           // 4 варианта шаблона.

  /*В режиме PGM регистр DTLFBUF используется для установки
  * шаблона. Содержимое буфера копируется в регистр
  * OUTOVEN при каждом условии UPDATE, при этом
  * устанавливается новый шаблон для вывода. Это гарантирует,
  * что коммутационная последовательность будет
  * синхронизирована с условием UPDATE при работе таймера */

  new_pattern = GetNewPattern(pattern_index);

  AWEXC.DTLSBUF = new_pattern;
  }
   /* Проверяем, установлен ли флаг переполнения OVFIF,
  * сбрасываем его и устанавливаем новое значение
  * длительности рабочего цикла */

  if(TC_GetOverflowFlag(&TCC0) != 0)
  {
      TC_ClearOverflowFlag(&TCC0);

  /* Выбираем некоторое «случайное» значение для длительности
  * рабочего цикла ШИМ, которое больше 0 и меньше TOP,
  * но сохраняем при этом минимально комфортную
  * длительность свечения светодиодов */
    if(TCC0.CCA >= 59000)
       {pwm_delta = -300 }
    else if(TCC0.CCA <= 5000)
       {pwm_delta = +300;}

  /*Изменяем значение буферного регистра сравнения
  *для изменения длительности рабочего цикла выходного
  * сигнала ШИМ */
         TCC0.CCABUF += pwm_delta;
      }
  }
}
```

После компиляции данного фрагмента кода и его запуска на STK600 можно наблюдать, как красиво ведут себя светодиоды на плате STK600, отображающие цифровой шаблон — яркость их свечения плавно изменяется от минимального до максимального значения. При каждом кратковременном нажатии кнопки SW0 текущий цифровой шаблон заменяется на следующий, но характер свечения светодиодов сохраняется. Хочется отметить, что размер кода для такой, в общем-то нетривиальной, задачи получился совсем небольшим.

Аналого-цифровой преобразователь

AVR-микроконтроллеры XMEGA могут иметь один или два восьмиканальных аналоговых порта (PORTA, PORTB), каждый из которых содержит один модуль быстродействующего аналого-цифрового преобразователя (АЦП). Новый модуль АЦП, который создали разработчики XMEGA, является для платформы AVR 8-bit RISC несомненным и значительным шагом вперед. Его параметры и набор функций неплохо смотрятся на фоне многих других встроенных АЦП у 8-разрядных и 16-разрядных микроконтроллеров для встраиваемых приложений.

В АЦП XMEGA используется метод последовательного приближения (SAR), при котором код результата преобразования формируется последовательно бит за битом, начиная со старшего разряда (MSB). Программным способом может устанавливаться один из двух вариантов разрядности АЦП — 8 или 12 бит. В будущем также планируется добавить 10-разрядный режим. Максимальная частота дискретизации АЦП составляет 2 миллиона выборок в секунду. Результаты преобразования могут передаваться через DMA без участия центрального процессора непосредственно в память или в периферийный узел. Начало преобразования может быть инициировано или из программы приложения установкой соответствующих битов в управляющих регистрах, или с приходом на блок АЦП внешнего события от другого периферийного узла микроконтроллера через Event System. Структурная схема АЦП показана на рис. 9.

Аналого-цифровой преобразователь у XMEGA имеет встроенный механизм калибровки, который позволяет скорректировать мультипликативную составляющую погрешности АЦП. Калибровка осуществляется на фабрике в процессе изготовления, калибровочные значения хранятся в секции Calibration and Signature Row во Flash-памяти программ микроконтроллера. Пользовательское приложение должно считывать эти значения и записывать их в калибровочный регистр CAL. В дальнейшем, Atmel планирует добавить в микроконтроллеры XMEGA еще один встроенный механизм калибровки для устранения погрешности смещения АЦП.

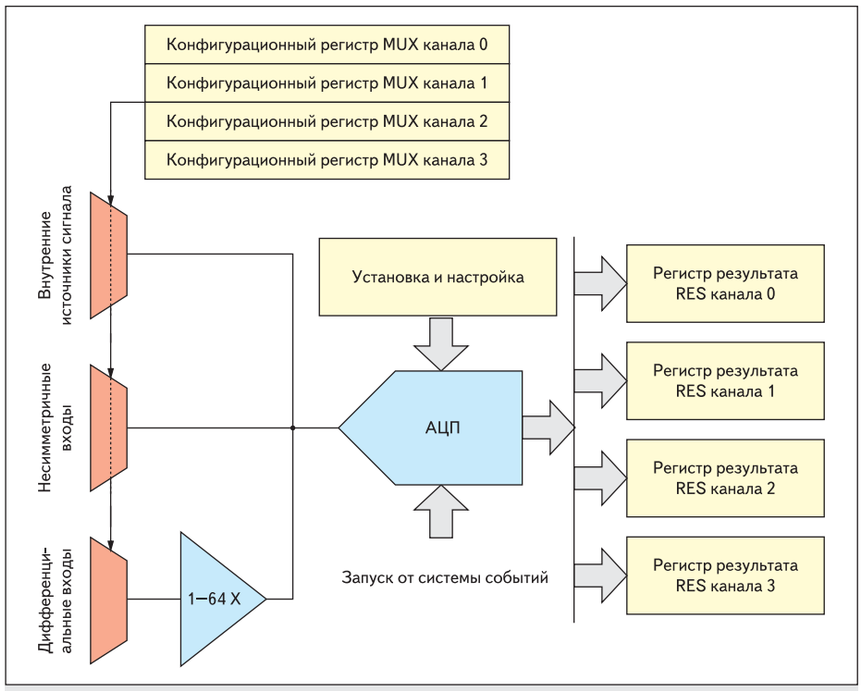


Рис. 9. Блок-схема АЦП в XMEGA

Допускается 4 конфигурации входных цепей: несимметричная, дифференциальная, дифференциальная с усилением и измерение сигнала от внутренних источников. Все четыре конфигурации и соответствующие им блок-схемы подробно описаны в документации на XMEGA и здесь рассматриваться не будут. Аналоговые входы портов PORTA и PORTB микроконтроллера могут использоваться как входные каналы для АЦП в несимметричном и дифференциальном включении, в то время как внутренние источники доступны непосредственно внутри кристалла. Для XMEGA с двумя АЦП «на борту» линии PORTA могут быть входами для ADCA и линии PORTB — для ADCB. Некоторые из микроконтроллеров имеют только один блок АЦП, но в качестве аналоговых входов могут использовать линии обоих аналоговых портов. На кристалле XMEGA также есть дополнительный каскад усиления входного аналогового сигнала, который позволяет значительно расширить динамический диапазон измеряемых напряжений в случае, когда входные цепи включены в дифференциальном режиме. Коэффициент усиления может программироваться и принимать ряд целых значений ряда 2^n от 1 до 64. Установленное значение коэффициента усиления не должно изменяться во время преобразования.

Собственно АЦП — это дифференциальный узел, который преобразует в цифровой код разность потенциалов между двумя его входами VINP (+) и VINN (–). Для того чтобы перевести АЦП в несимметричное включение, отрицательный вход VINN подключается внутри кристалла к фиксированному источнику смещения. И тут есть одна особенность: если результат преобразования должен кодироваться со знаком, то VINN подключается к внутренней «земле», а в случае получения беззнакового результата (только в несимметричном включении!) на этот вход АЦП подается небольшое отрицательное смещение. Это необходимо для предотвращения входа в насыщение аналоговой части преобразователя и гарантирует надежную работу АЦП при входном напряжении, близком к нулю. Если АЦП используется в одном из дифференциальных режимов, то выходной результат должен кодироваться только со знаком.

Тактирующий АЦП сигнал формируется из сигнала периферийной тактовой частоты, который перед подачей на АЦП проходит через предварительный делитель (рис. 10). Это позволяет регулировать тактовую частоту АЦП (будем обозначать ее ***fADC***) в широких пределах — от максимального значения 2 МГц до минимального ~50 кГц. Значения

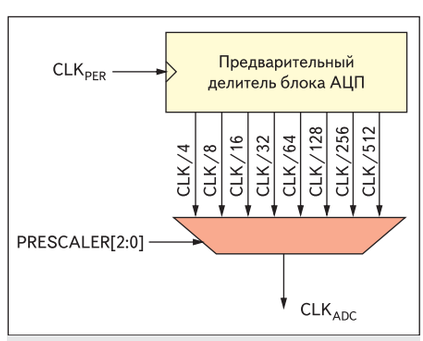


Рис. 10. Тактирование блока АЦП

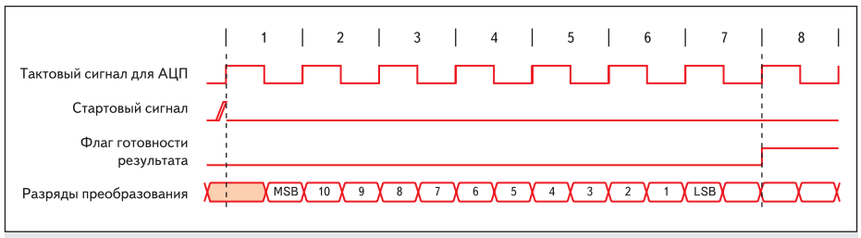


Рис. 11. Пример формирования тактовых последовательностей

коэффициента деления могут выбираться программистом и составляют 4, 8, 16, 32, 64, 128, 256 или 512.

АЦП может начинать новое преобразование на каждый такт сигнала ***fADC***. Задержка распространения (или время преобразования АЦП типа SAR) зависит от установленного разрешения (8 или 12 разрядов) и будет увеличиваться на 1 дополнительный период сигнала ***fADC***, если используется предварительное усиление входных аналоговых сигналов. Формула для расчета задержки распространения приведена в технической документации на микроконтроллер.

Рассмотрим работу схемы тактирования АЦП на примере самого простого режима — одиночного преобразования без предварительного усиления сигнала (рис. 11). Сигнал для начала преобразования (установка бита в управляющем регистре или входящее событие) имеет обозначение START и должен поступать в течение предыдущего периода сигнала периферийной тактовой частоты до начала того периода сигнала ***fADC***, в течение которого начинается преобразование. Входной аналоговый сигнал «защелкивается» в течение первой половины первого периода сигнала ***fADC***. Отметим, что время выборки и время преобразования одного разряда конечного результата всегда составляют половину периода сигнала тактовой частоты АЦП. Разряд MSB результата формируется первым, остальные разряды формируются в течение последующих трех (для 8-разрядных результатов) или пяти (для 12-разрядных результатов) периодов сигнала ***fADC***. В течение последнего периода после формирования LSB преобразованное значение заносится в регистр результата. И только потом устанавливается флаг запроса на прерывание, сигнализирующий об окончании цикла преобразования и доступности результата. Описание работы схемы тактирования АЦП в других режимах рекомендуется смотреть в технической документации на микроконтроллер.

Как уже упоминалось, у АЦП в XMEGA есть возможность измерять аналоговые сигналы, приходящие от каких-то внутренних источников на кристалле. Для этой цели предусмотрены 4 внутренних канала, которые могут подключаться ко входу АЦП для преобразования:

- встроенный датчик температуры кристалла;
- прецизионный ИОН;
- линейный выход ЦАП;
- масштабируемое напряжение питания микроконтроллера.

Таким образом, можно оценивать текущую температуру кристалла, контролировать напряжение встроенного точного ИОН или измерять напряжение, выдаваемое ЦАП. Для измерения или контроля напряжения питания микроконтроллера VCC оно перед подачей на вход АЦП проходит через внутренний аттенюатор, который ослабляет его в 10 раз. То есть, если VCC составляет 3,6 В, то оно будет измеряться как 0,36 В. Измерение внутренних сигналов допускается только в несимметричном включении.

Для работы любого АЦП требуется опорное входное напряжение. В микроконтроллерах XMEGA допускается использование как внешнего, так и встроенного источника опорного напряжения. Встроенный прецизионный ИОН с напряжением 1,0 В позволяет осуществлять точные преобразования в диапазоне от 0 до 1 В в несимметричном включении и от –1 до 1 В в дифференциальном включении. Для расширения диапазона измеряемых входных напряжений можно подключить внутреннее напряжение VCC 0,6 В. Положительное опорное напряжение от внешнего источника может подаваться на выводы AREF аналоговых портов PORTA и PORTB микроконтроллера.

Наиболее интересная особенность АЦП в XMEGA — его конвейерная архитектура. Это означает, что новое входное аналоговое напряжение может быть «защелкнуто» и новое аналого-цифровое преобразование может быть начато в то время, когда выполняются текущие преобразования. В результате эффективный темп работы АЦП существенно возрастает.

Аналого-цифровой преобразователь в XMEGA содержит четыре конфигурационных регистра CHnMUXCTRL и четыре соответствующих им регистра результата CHnRES. Каждая пара этих регистров MUX/RES в документации Atmel носит название канала преобразования АЦП (CH0…CH3), хотя разработчики XMEGA сначала называли их виртуальными каналами. Каждый канал АЦП полностью независим от других. Все каналы имеют раздельные:

- возможности выбора источника входного сигнала;
- регистры для хранения результата преобразования;
- стартовые биты, установка которых инициирует начало преобразования;
- вектора и программируемые уровни прерываний.

Наличие четырех независимых каналов преобразования значительно повышает эффективность и удобство работы с АЦП, а также увеличивает количество возможных источников прерывания. Все каналы используют один и тот же аппаратный аналого-цифровой преобразователь для осуществления преобразований, но благодаря конвейерной архитектуре новое (очередное) преобразование может быть начато по следующему такту сигнала ***fADC***. Это означает, что многочисленные преобразования АЦП могут осуществляться одновременно и независимо друг от друга. Результат преобразования какого-то канала может храниться в своем регистре результата независимо от частоты обновления регистров результата остальных каналов новыми данными. Такое решение помогает снизить сложность программного обеспечения, так как различные программные модули могут инициировать начало преобразований, а также считывать результаты независимо друг от друга. Например, CH0 программируется на несимметричное включение для преобразования сигнала с входной линии аналогового порта, которое запускается внешним входным сигналом через систему событий. В это же время CH1 в дифференциальном включении измеряет другой входной сигнал, причем запуск преобразований инициируют другие события, поступающие от Event System. И одновременно с этим оставшиеся каналы CH2 и CH3 могут осуществлять преобразования еще каких-то двух внешних или внутренних сигналов, но запускаться на преобразование они могут по команде из программы пользователя.

Все многочисленные комбинации установок режима работы канала АЦП с помощью регистров CHnMUXCTRL подробно описаны в документации на микроконтроллер.

Старт преобразования в канале АЦП может быть инициирован или программно из приложения пользователя, или при поступлении события от Event System. В случае программного управления все достаточно просто — установка бита START в регистре CTRL выбранного канала или одного из битов CH[3:0]START в общем регистре CTRLA запускает АЦП на преобразование. Существует возможность устанавливать программно сразу несколько битов CHnSTART, в этом случае «помеченные» этими битами каналы будут запускаться на преобразование последовательно друг за другом, начиная с канала 0, который имеет самый высокий приоритет в очереди. После начала преобразований биты CHnSTART сбрасываются аппаратно.

При использовании события от Event System для старта АЦП структура управления не-

много сложнее. В самом простом случае приход события инициирует одиночное преобразование в одном из каналов. Допускается использование одного и того же события для одновременного запуска преобразований на нескольких каналах. В «старших» микроконтроллерах XMEGA с двумя АЦП есть возможность синхронно запустить преобразования в обоих АЦП от одного и того же события. Все многочисленные комбинации использования каналов системы событий для старта АЦП и возможные действия по событию (за это отвечают биты EVSEL и EVACT регистра EVCTRL) подробно рассмотрены в технической документации на микроконтроллер.

Подобно прочим встраиваемым АЦП в микроконтроллерах общего назначения, аналого-цифровой преобразователь у XMEGA может работать в двух режимах:

- одиночное преобразование в выбранном канале, по завершении которого работа АЦП останавливается;
- циклическое преобразование, при котором каналы АЦП постоянно опрашиваются последовательно в фиксированном циклическом порядке, один за другим.

Выбор режима определяется установкой бита FREERUN регистра CTRLB. Если бит сброшен, то разрешены одиночные преобразования, а если установлен — то циклические. Для одиночного преобразования каждый канал может запускаться индивидуально. Для циклического порядка количество каналов в последовательности программируется. Два бита SWEEP регистра EVCTRL определяют, сколько каналов будет включено в постоянно работающую циклическую последовательность (от одного до четырех). Приоритет опроса каналов фиксирован, канал 0 имеет высший приоритет, канал 3 — низший.

Выходной результат преобразования АЦП может кодироваться как со знаком, так и без знака. Представление результата программируется пользователем путем установки бита CONVMODE в регистре CTRLB, причем эта установка является глобальной для всех каналов АЦП. Для 12-разрядных результатов со знаком диапазон преобразования составляет от –2048 до +2047 (0xF800–0x07FF), отрицательные числа представляются в дополнительном коде. Для беззнаковых результатов диапазон преобразования составляет от 0 до 4095 (0–0x0FFF). Отметим, что если входные цепи любого из каналов АЦП сконфигурированы на работу в дифференциальном режиме, то результат преобразования АЦП должен кодироваться только со знаком.

Когда программистом выбирается кодировка результатов без знака, вход VINN у АЦП подключается к фиксированной аналоговой «земле» внутри кристалла. Следовательно, становятся возможны только измерения внутренних сигналов или внешних аналоговых сигналов в несимметричном включении. Если же выбирается кодирование результата со знаком, то могут измерять-ся как положительные, так и отрицательные входные напряжения в обоих включениях — несимметричном и дифференциальном.

Результат аналого-цифрового преобразования записывается в один из регистров результата CHnRES. Все регистры результата 16-разрядные, состоят из старшего и младшего байтов. Если планируется получать только 8-разрядные данные, результат будет принудительно сдвинут вправо в CHnRES (это означает, что все 8 LSB будут находиться в младшем байте). А 12-разрядный результат может по желанию пользователя представляться как сдвинутым влево, так и сдвинутым вправо. Выравнивание по левому краю означает, что 8 MSB результата будут находиться в старшем байте (программируется установкой битов RESOLUTION в регистре CTRLB).

Когда выходные данные кодируются со знаком, в разряде MSB будет находиться знак результата. Для 12-разрядных данных с выравниванием по правому краю знак (11-й разряд) будет копироваться в разряды 12–15 старшего байта для формирования полного 16-разрядного результата в дополнительном коде. Для 8-разрядных данных знак (7-й разряд) будет копироваться в весь старший байт результата преобразования.

АЦП может генерировать и запросы на прерывание, и события для Event System. Как уже отмечалось, все каналы АЦП имеют индивидуальные вектора, программируемые уровни и флаги CHnIF прерываний. В регистре INTCTRL два младших бита INTLVL разрешают генерацию запроса на прерывание от канала АЦП и назначают требуемый уровень для этого прерывания. Флаги запросов на прерывание CHnIF автоматически сбрасываются после передачи управления соответствующему обработчику прерывания. Программист также может сбрасывать их программно, записывая «1» в соответствующий бит.

Запрос на прерывание или событие генерируется в двух случаях. В первом случае флаг CHnIF выставляется по завершении преобразования в выбранном канале «n» АЦП. Второй случай — если в результате измерения входного напряжения полученное значение больше (или меньше) заранее выбранного программистом порога. Специальный регистр сравнения CMP в АЦП хранит это пороговое значение, а биты INTMODE регистра INTCTRL определяют, в каком случае должен генерироваться запрос на прерывание: COMPLETE — завершение преобразования, BELOW — результат ниже порога, ABOVE — результат выше порога. Этот интересный регистр сравнения CMP является общим для всех каналов АЦП. В него записывается 12-разрядный цифровой код, соответствующий требуемому пороговому напряжению, с которым постоянно сравниваются результаты текущих преобразований в каналах АЦП. Для пользователя это означает, что сигнал о завершении преобразования (или флаг прерывания CHnIF) не будет устанавливаться аппаратной логикой АЦП, пока результат преобразования не станет выше (или ниже) программируемого порога сравнения. Отметим, что значение в регистре CMP всегда выравнивается по левому краю.

Данная опция очень удобна в задачах, где необходимо лишь проверять, не «ушел» ли в процессе работы измеряемый входной сигнал за установленный порог напряжения. Вместо того чтобы постоянно проводить измерения и программно сравнивать результаты с пороговым значением, приложение теперь может просто ждать сигнала о наступлении выбранного события (выше или ниже порога), а вся «черновая» работа производится аппаратной логикой АЦП. Это снижает загрузку центрального процессора, уменьшает размер конечного кода и облегчает жизнь программисту.

Для передачи результатов преобразования АЦП в память микроконтроллера или в один из периферийных узлов на кристалле может быть использован контроллер DMA. Когда очередное (новое) преобразование закончено и обновлен регистр результата, это событие может вызывать запрос на транзакцию.

В заключение краткого описания возможностей АЦП в микроконтроллерах XMEGA скажем несколько слов о приоритете каналов и принудительном сбросе. Поскольку частота периферийного тактового сигнала может быть существенно выше выбранной частоты сигнала ***fADC***, есть все шансы «получить» установленные биты начала преобразования для нескольких каналов АЦП внутри одного и того же периода ***fADC***. Внешние события, поступающие от Event System, также могут инициировать начало преобразования на нескольких каналах АЦП по схожему сценарию. В таких случаях канал АЦП CH0 будет иметь высший приоритет.

Запуск преобразования в канале АЦП по приходу события от Event System может вызывать не известную заранее задержку между этим событием и началом преобразования, поскольку в этот момент времени могут осуществляться преобразования по другим каналам в соответствии с установленным приоритетом в очереди. Но если необходимо начать преобразование немедленно при поступлении входящего события, у программиста есть возможность полностью сбросить весь конвейер АЦП для всех преобразований во всех каналах, включая текущую фазу тактовой последовательности ***fADC***. Это заставит АЦП «защелкнуть» новый аналоговый входной сигнал уже на следующем периоде периферийного тактового сигнала, за которым немедленно последует первый период тактовой последовательности системы тактирования АЦП.

Дополнительную информацию об аналого-цифровом преобразователе в XMEGA можно найти в техническом описании на микроконтроллер. К сожалению, на момент написания

этой статьи компания Atmel еще не опубликовала руководство по применению Application Note AVR1300 и примеры программ для работы с АЦП. Поэтому здесь мы подробнее проиллюстрируем начало работы с АЦП в XMEGA на трех практических примерах, которые рассматривались на техническом тренинге в центре AVR (Норвегия, сентябрь 2007). Использовался стартовый набор разработчика STK600, исходные тексты написаны на Си и скомпилированы в интегрированной среде IAR Systems EWAVR 4.30. Необходимые установки на плате STK600 (перемычки, соединительные кабели, переключатели и т. п.) в данной статье не описываются, мы рекомендуем использовать руководство пользователя для STK600 ("User Manual").

Для реализации примеров нам потребуется один дополнительный внешний компонент — потенциометр сопротивлением 10 кОм, с помощью которого мы будем изменять напряжение, прикладываемое к входу АЦП. Крайние выводы потенциометра подключим к положительному (AREF0) и отрицательному (AREF1) выводам разъема AUX на плате STK600, а его движок — к линии 2 аналогового порта PORTA. С помощью настроек в AVR Studio необходимо заранее запрограммировать оба вывода AREF на плате STK600 так, чтобы на них формировались напряжения 2,7 и 0,3 В соответственно.

Пример № 1. Несимметричное включение, преобразование в одном канале

Этот тип преобразований, без сомнения, является наиболее распространенным. На вход АЦП подается положительное напряжение с внешнего вывода микроконтроллера или от одного из внутренних источников. Данный пример показывает, как следует инициализировать АЦП. Будем использовать канал CH0 для измерений и программный запуск АЦП на преобразование путем установки стартового бита. Результат выводится на светодиоды платы STK600.

```
#define LEDPORT PORTC  // Назначаем порт вывода
                       // для светодиодов

void main( void )
{
  uint16_t ADC_result;      // Переменная для хранения
                            // результата преобразования

  LEDPORT.DIR = 0xFF;       // Линии порта должны быть
                            // сконфигурированы на вывод
                            // информации
  LEDPORT.OUT = 0xFF;       // По умолчанию все светодиоды
                            // выключены

  // Конфигурируем канал CH0 на работу в несимметричном
  // включении
  ADCA.CH0MUXCTRL = ADCA.CH0MUXCTRL &
 ~ADC_MUXMODE_gm |ADC_MUXMODE_SINGLEENDED_gc;

  // В несимметричном включении подключаем
  // к положительному входу АЦП линию 2 порта PORTA.
  ADCA.CH0MUXCTRL = ADCA.CH0MUXCTRL
                & ~ADC_MUXPOS_gm | ADC_MUXPOS_PIN2_gc;

  /* Назначаем внешний источник опорного напряжения
  * для АЦП. На плате STK600 он подключается к линии 0
  * порта PORTA, которая в этом режиме работает как вход AREF. */
  ADCA.REFCTRL = ADCA.REFCTRL & ~ADC_REFSEL_gm |
                                        ADC_REFSEL_EXT_gc;

  /* Конфигурируем систему тактирования АЦП: периферийная
  * тактовая частота делится на 32, что дает нам частоту
  * тактирования АЦП 62,5 кГц при значении системной
  * частоты процессора по умолчанию 2 МГц */
     ADCA.PRESCALER = ADCA.PRESCALER &
         ~ADC_PRESCALER_gm | ADC_PRESCALER_DIV32_gc;

  // Разрешаем работу АЦП
  ADCA.CTRLB |= ADC_ENABLE_bm;

  /* Запускаем бесконечный цикл, в котором программно
  * инициируется преобразование в канале CH0, и 8 младших
  * разрядов получаемого результата выводятся для визуального
  * отображения на светодиоды платы STK600 */

  for(;;)
  {
    ADCA.CTRLB |= ADC_CH0START_bm;

    do { } while((ADCA.INTFLAGS & ADC_CH0IF_bm) == 0x00);
                              // Флаг CH0IF устанавливается
                              // по завершении преобразования
    ADCA.INTFLAGS = ADC_CH0IF_bm;  // Сбрасываем флаг CH0IF
                                   // записью в него «1».
    ADC_result = ADCA.CH0RES;  // Считываем результат
                               // преобразования
                               // в канале CH0
    ADC_result >>= 4;          // Оставляем только
                               // 8 старших разрядов…
    LEDPORT.OUT = ~ADC_result;  // … и выводим их на
                                // светодиоды для
                                // отображения.
  }

}
```

После компиляции этого фрагмента кода и его запуска на STK600 светодиоды будут индицировать в двоичном коде величину напряжения, которое определяется случайным начальным положением потенциометра. Если теперь крутить движок потенциометра влево или вправо, то напряжение на входе АЦП и результаты преобразования будут изменяться, что немедленно будет отображаться светодиодами.

Если в программе поменять входную линию порта (например, на третью), но не переключать контакт от движка потенциометра, то после новой компиляции и запуска кода на исполнение вращение потенциометра не будет оказывать воздействия на характер свечения светодиодов. Но если поднести палец к контакту, соответствующему входной линии 3 порта PORTA, то 2–3 младших светодиода будут «моргать» — АЦП улавливает и преобразует в цифровой код наводимую на его вход помеху, а палец здесь выступает в роли антенны.

Пример № 2. Использование регистра сравнения в АЦП

В дополнение к обычным преобразованиям, АЦП в XMEGA обладает еще и функцией сравнения. При ее использовании канал АЦП программируется на постоянное проведение измерений, но сигнал о завершении преобразования (флаг запроса на прерывание) не будет сгенерирован до тех пор, пока измеряемое напряжение не станет больше (или меньше) заранее запрограммированного значения.

Как и в примере № 1, будем использовать канал CH0 для измерений и светодиоды платы STK600 для отображения результата. Включение потенциометра и настройки VREF тоже оставим без изменений. Но теперь АЦП будет программно запускаться на циклические преобразования. Также в данной задаче нам еще потребуется помощь контроллера прерываний PMIC.

```
#define LEDPORT PORTC              // Назначаем порт вывода
                                   // для светодиодов

#pragma vector = ADCA_CH0_vect  // Простой обработчик
                                // прерываний от канала
                                // CH0 АЦП
__interrupt void ADCA_CH0_ISR(void)
{
  __delay_cycles(250000);       // Ждем некоторое время, а затем
                                // принудительно выключаем
  LEDPORT.OUTSET = 0xFF;        // все светодиоды на плате STK600
}

// Разрешаем работу контроллеру прерываний PMIC
void PMIC_enable()
{
  PMIC.CTRL |= PMIC_LOLVLEN_bm;  // В этой задаче будем
                                 // использовать
                                 // прерывания уровня
                                 // Low
  __enable_interrupt();
}

void main( void )
{
  uint16_t ADC_result;      // Переменная для хранения
                            // результата преобразования

  LEDPORT.DIR = 0xFF;       // Линии порта должны быть
                            // сконфигурированы на вывод
                            // информации
  LEDPORT.OUT = 0xFF;       // По умолчанию все светодиоды
                            // выключены

// Назначаем уровень прерываний от канала CH0 АЦП — Low:
  ADCA.CH0INTCTRL = ADCA.CH0INTCTRL &
                  ~ ADC_INTLVL_gm | ADC_INTLVL_LO_gc;

  // Запрос на прерывание будет генерироваться в случае,
  // если результат измерения станет выше порога:
  ADCA.CH0INTCTRL = ADCA.CH0INTCTRL &
           ~ ADC_INTMODE_gm | ADC_INTMODE_ABOVE_gc;

  /* Когда используется регистр CMP (ADC Compare), то он
  * должен быть загружен пороговым значением. Установим это
  * значение, например, как 0x0CCC. Важно!! Регистр сравнения
  * CMP всегда выравнивается по левому краю, поэтому вместо
  * 0x0CCC мы должны записать в него 0xCCC0 */
  ADCA.CMP = 0xCCC0;

  // Конфигурируем канал CH0 АЦП на циклические
  // преобразования сигнала с линии 2 порта PORTA:
ADC_ConfigInputCh0( &ADCA,ADC_MUXMODE_SINGLEENDED_gc,
                                  false, ADC_MUXPOS_PIN2_gc,
                                         ADC_MUXNEG_PIN0_gc );
ADC_ConfigPrescaler(&ADCA, ADC_PRESCALER_DIV2_gc);
ADC_Enable( &ADCA, true, ADC_REFSEL_EXT_gc, false, 0);
PMIC_enable();

  for(;;)      // Постоянно выводим на светодиоды текущий
               // результат преобразования АЦП
  {
    ADC_result = ADCA.CH0RES;      // Считываем результат
                                   // преобразования
                                   // в канале CH0
    ADC_result >>= 4;              // Оставляем только
                                   // 8 старших разрядов…
    LEDPORT.OUT = ~ADC_result;     // … и выводим их на
                                   // светодиоды
                                   // для отображения.
  }

}
```

Выполним компиляцию примера и запустим его на STK600, предварительно установив движок потенциометра в среднее положение. Светодиоды будут индицировать цифровой код, соответствующий значению напряжения, которое определяется начальным положением потенциометра (пока оно ниже установленного нами порога). Если теперь плавно по-

ворачивать движок потенциометра вправо, повышая напряжение на входе АЦП, мы будем видеть на светодиодах увеличивающийся выходной код, разряд за разрядом. Но как только напряжение на входе превысит установленное нами пороговое значение, то постоянно будет вызываться обработчик прерываний, и светодиоды во всех разрядах результата начнут одновременно «мигать». Это сигнализирует нам о том, что порог превышен и следует предпринять какие-либо корректирующие действия.

Пример № 3.
Одновременная и независимая работа нескольких каналов АЦП

Наличие независимых каналов у АЦП в микроконтроллерах XMEGA повышает эффективность и удобство работы, так как многочисленные преобразования могут проводиться одновременно. Различные программные модули могут инициировать начало преобразований, а также считывать результаты измерений в разных каналах независимо друг от друга.

Покажем это на примере работы двух каналов. Будем использовать каналы CH0 и CH1 в циклическом режиме преобразований. Включение потенциометра и настройки VREF оставляем без изменений. Канал CH0 будет измерять внешнее напряжение, снимаемое с движка потенциометра, а канал CH1 — напряжение со встроенного на кристалл XMEGA датчика температуры. Отметим, что в процессе работы основной программы реконфигурирование каких-либо параметров работы каналов не требуется.

```
#define LEDPORT PORTC      // Назначаем порт вывода
                           // для светодиодов

void main( void )
{
 uint8_t ADC_result0;   // Переменные для хранения результатов
                        // преобразований в каналах CH0 и CH1
 uint8_t ADC_result1;

 /* Программируем постоянный циклический запуск
 * преобразований в каналах CH0 и CH1 путем установки битов
 * SWEEP в регистре EVCTRL */
 ADCA.EVCTRL = ADCA.EVCTRL & ~ADC_SWEEP_gm |
                                    ADC_SWEEP_01_gc;

 /* Используем несимметричное включение для обоих каналов
 * (аналогично примерам № 1 и № 2). CH0 будет измерять внешнее
 * входное напряжение, а CH1 — сигнал от внутреннего датчика
 * температуры */
 ADC_EnableTempReference(&ADCA);

 ADCA.CH0MUXCTRL = ADCA.CH0MUXCTRL &
~ADC_MUXMODE_gm | ADC_MUXMODE_SINGLEENDED_gc;
 ADCA.CH0MUXCTRL = ADCA.CH0MUXCTRL &
                ~ADC_MUXPOS_gm | ADC_MUXPOS_PIN2_gc;

 ADCA.CH1MUXCTRL = ADCA.CH1MUXCTRL &
     ~ADC_MUXMODE_gm | ADC_MUXMODE_INTERNAL_gc;
ADCA.CH1MUXCTRL = ADCA.CH1MUXCTRL &
                ~ADC_MUXINT_gm | ADC_MUXINT_TEMP_gc;
// Отметим: «MUXINT_gm» эквивалентно «MUXPOS_gm».

 // АЦП конфигурируется на генерацию 8-разрядного результата
 ADC_ConfigPrescaler(&ADCA, ADC_PRESCALER_DIV32_gc);
 ADC_ConfigResolution(&ADCA, ADC_RESOLUTION_8BIT_gc);
 ADC_Enable(&ADCA, true, ADC_REFSEL_EXT_gc, false, 0);

 LEDPORT.DIR = 0xFF;     // Выключаем все светодиоды

 for(;;)                 // Запускаем бесконечный цикл…
 {
 do { } while((ADCA.INTFLAGS & ADC_CH1IF_bm) != ADC_CH1IF_bm);
 /* Флаг CH1IF устанавливается только после завершения обоих
 * преобразований, так как CH0 начинает работать первым
 * (по приоритету) */

 ADCA.INTFLAGS |= ADC_CH1IF_bm;  // Сбрасываем флаг CH1IF

 ADC_result1 = ADCA.CH1RESL;     // Считываем результат
                                 // в канале CH1
 ADC_result0 = ADCA.CH0RESL;     // Считываем результат
                                 // в канале CH0
 ADC_result0 >>= 4;              // Сдвигаем результат CH0
                                 // вправо

 /* Теперь одновременно выводим оба результата на светодиоды:
 * 4 старших LED будут показывать температуру, а 4 младших —
 * величину внешнего напряжения */
   LEDPORT.OUT = ~( ADC_result0 | (ADC_result1 & 0xF0));
 }

}
```

После компиляции примера № 3 и его запуска на STK600 светодиоды LED3…0 будут отображать четыре старших разряда текущего значения напряжения, снимаемого с движка потенциометра. Светодиоды LED7…4 будут показывать некоторое значение температуры кристалла внутри корпуса микроконтроллера. Если поворачивать движок потенциометра влево или вправо, напряжение на входе АЦП и результаты преобразования в канале CH0 будут изменяться — это будет сразу же отображаться светодиодами LED3…0. «Старшие» светодиоды на плате STK600 при этом будут сохранять свое состояние, но это не означает, что измерения в канале температуры не проводятся! Чтобы убедиться в этом, можно попробовать осторожно подогреть корпус микроконтроллера снаружи (например, излучением от настольной лампы накаливания). Через несколько секунд светодиоды LED7…4 начнут изменять свое состояние, отображая рост температуры кристалла. Если теперь снять тепловую нагрузку, то температура кристалла немедленно начнет снижаться, и состояние светодиодов LED7…4 постепенно вернется к своему начальному значению.

Контроллер прямого доступа к памяти

Этот полезный аппаратный блок, работу которого мы кратко будем рассматривать в завершение цикла статей про XMEGA, не так часто включается производителями в кристаллы 8-разрядных микроконтроллеров. Разработчики XMEGA приняли удачное решение, добавив контроллер прямого доступа к памяти (DMAC) в состав периферийных блоков нового микроконтроллера Atmel. Его возможности по пересылке массивов данных между областями памяти и периферийными блоками дополняют прогрессивную и развитую периферию XMEGA, что дает разработчику универсальный набор средств для работы.

Структура блока DMAC у XMEGA достаточно типична. Кроме того, его работа подробно описана в технической документации на микроконтроллер и в Application Note AVR1304, поэтому здесь подробно контроллер DMA мы рассматривать не будем.

DMAC может захватывать внутреннюю шину данных микроконтроллера только в те моменты времени, когда центральный процессор ее не использует. Но если процессор занят обменом данными с внешней памятью, то в это же время DMAC может передавать данные между периферийным блоком и областью памяти в пределах кристалла микроконтроллера. Приоритет доступа к внутренней шине данных распределяет арбитражный узел, аппаратно отнесенный к подсистеме управления памятью данных микроконтроллера.

Источник и приемник для обмена данными могут комбинироваться следующим образом:

- RAM <-> RAM (как внутренняя, так и внешняя);
- периферийный блок <-> периферийный блок;
- RAM <-> периферийный блок;
- EEPROM -> RAM;
- EEPROM -> периферийный блок.

Отметим, что DMAC не может записывать данные в область EEPROM, а может только считывать из нее.

Контроллер DMA содержит четыре независимых канала, каждый из которых имеет свои источники и приемники, способ запуска передачи и размеры блока данных, индивидуальные настройки и вектора прерываний. Для источника и приемника предусмотрено несколько режимов адресации. Каждый канал DMA может работать как на прием, так и на передачу. Каналы не могут прерывать друг друга, если передача в каком-либо из них уже начата.

Законченная операция чтения или записи DMA между источником и приемником данных носит название транзакции. Данные в транзакции передаются блоками, которые состоят из пакетов (рис. 12). Общая длина передаваемого блока данных программируется и может составлять от 1 байта до 64 кбайт. Имеется возможность программно управлять повторением передачи блока, записывая требуемое число повторов в регистр Repeat Counter — от 1 до 255. Запись нулевого значения в этот регистр приведет к безостановочной циклической передаче блоков, одного за другим. Если режим повторения запрещен, то происходит однократная передача блока данных.

У каналов DMAC можно достаточно гибко программировать приоритет их работы. Если несколько каналов одновременно запрашивают доступ к внутренней шине данных микроконтроллера для передачи, схема приоритета определяет, какой канал будет иметь право на передачу первым. Приоритет может быть или фиксированным, или осуществляться по схеме Round Robin (канал, который в последний раз передавал данные, теперь будет иметь наименьший приоритет). По умолчанию при старте микроконтроллера устанавливается приоритет по схеме Round Robin.

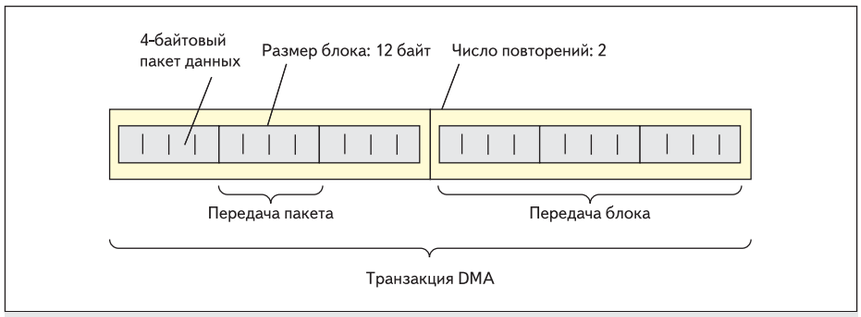


Рис. 12. Пакет, блок и транзакция DMA

Для разрешения «длинных» транзакций два канала DMAC могут быть попарно «связаны» друг с другом — второй начинает передачу, когда первый ее заканчивает, и наоборот. Объединяться в пары могут каналы 0–1 или 2–3. В одном из режимов также разрешается одновременная работа двух указанных пар. Другие комбинации сочетаний каналов не допускаются.

У контроллера DMA в XMEGA есть еще один интересный режим работы — Single Shot, или пакетный режим передачи. Длина пакета данных программируется и может составлять 1, 2, 4 или 8 байт. Передача пакета, если начата, не может быть прервана процессорным ядром. Это может оказаться полезным при чтении слов данных, например, результатов преобразования АЦП.

Режим Single Shot разрешается из программы пользователя установкой бита SINGLE в регистре CTRLA соответствующего канала DMAC. Если Single Shot разрешен, то в ответ на каждый запрос передачи данных будет передаваться только один пакет.

Контроллер DMA может генерировать запросы на прерывание при завершении передачи данных (Transaction Complete) или в случае детектирования сигнала ошибки (Transfer Error). Флаг завершения передачи TRNFIF выставляется в конце транзакции, а сигнал ошибки (установка флага ERRIF) возникает в следующих случаях:

- если работа канала или всего модуля DMAC программно запрещается в ходе выполнения текущей транзакции;
- при попытке записать данные в EEPROM с помощью DMA;
- при попытке прочитать с помощью DMA данные из EEPROM в то время, когда микроконтроллер находится в спящем режиме.

К сожалению, у XMEGA для каждого канала DMAC есть всего один вектор прерывания, обслуживающий оба флага, поэтому прикладная программа при передаче управления вектору прерывания от канала DMAC всегда должна проверять, какой из флагов был установлен. Каждый флаг может быть сброшен путем записи в него «1». Отметим, что когда в регистр Repeat Counter записано нулевое значение (то есть разрешена «бесконечная» циклическая передача блоков), флаг TRNFIF будет устанавливаться в конце каждого передаваемого блока данных.

Начало процесса передачи в DMAC может быть инициировано либо программно, либо при поступлении события от Event System (например, завершение преобразования АЦП, изменение уровня сигнала на внешнем выводе микроконтроллера, переполнение таймера). Программная установка бита TRFREQ в регистре CTRLA соответствующего канала DMAC вызовет начало передачи, что автоматически сразу же сбросит этот бит.

Типовое приложение, для которого контроллер DMA действительно весьма полезен — это копирование больших блоков данных между памятью и (или) периферийными блоками микроконтроллера без участия центрального процессора. Эта задача подробно описана в Application Note AVR1304, там же есть исходный текст программы, поэтому в статье мы его повторять не будем.

Покажем другие возможности работы контроллера DMA в XMEGA. Реализуем с помощью стартового набора STK600 простое устройство, которое способно в течение 10 секунд запоминать последовательность случайных нажатий клавиш SW на плате набора, а затем постоянно «проигрывать» эту последовательность на светодиодах набора LED. Процессы записи и воспроизведения осуществляются без участия центрального процессора. Очевидно, что такой же принцип может использоваться для того, чтобы оцифровывать звук с помощью АЦП, запоминать получающуюся звуковую дорожку и затем воспроизводить ее с помощью ЦАП микроконтроллера XMEGA.

Пример № 1. Простой цифровой рекордер

```
#define READ_CHANNEL 0          // Канал DMA для считывания
                                // состояния кнопок SW.
#define WRITE_CHANNEL 1         // Канал DMA для вывода данных
                                // на светодиоды LED

#define LEDPORT PORTD           // Назначаем порт вывода
                                // для светодиодов
#define SWITCHPORT PORTC        // Назначаем порт ввода
                                // для кнопок

#define SAMPLE_COUNT 300        // Назначаем максимальное
                                // количество запоминаемых
                                // состояний.
uint8_t samples[SAMPLE_COUNT];  // Буфер для хранения
                                // запомненных состояний.
```

```
/* Канал DMA начнет работу по флагу переполнения таймера
* и будет постоянно перезапускаться, пока этот флаг остается
* установленным. А мы хотим, чтобы DMA осуществлял
* одиночную передачу данных для каждого переполнения
* таймера. Поэтому нам нужно разрешить прерывания,
* для того чтобы флаг переполнения таймера автоматически
* сбрасывался при вызове обработчика прерывания.
* Тем не менее, нам не нужно ничего делать в программе
* обработки прерывания, поэтому «тело» обработчика
* мы оставляем пустым */

#pragma vector = TCC0_OVF_vect
__interrupt void EmptyHandlerTCC0OVF( void )
{ __no_operation(); }

/* Установки канала DMA для считывания состояния кнопок.
* Канал сконфигурирован на перезагрузку начального адреса
* назначения, когда передача блока завершена. Режим Single Shot
* разрешен, поэтому по каждому переполнению таймера
* в память копируется пакет, равный одному байту. */
void SetupReadChannel( void )
{
  DMA_SetupSingleBlock( READ_CHANNEL,
    (void const *) &(SWITCHPORT.IN),
                          DMA_CH_SRCRELOAD_NONE_gc,
                            DMA_CH_SRCDIR_FIXED_gc,
    samples, DMA_CH_DESTRELOAD_BLOCK_gc,
                            DMA_CH_DESTDIR_INC_gc,
    SAMPLE_COUNT, DMA_CH_BURSTLEN_1BYTE_gc );
  DMA_EnableSingleShot( READ_CHANNEL );
  DMA_SetTriggerSource( READ_CHANNEL, 0x40 );
                          // Старт — по переполнению TCC0.
}

/* Установки канала DMA для вывода информации на светодиоды.
* Канал сконфигурирован на перезагрузку начального адреса
* источника, когда передача блока завершена. Он также
* сконфигурирован на постоянную передачу блока данных
* (записью 0 в регистр Repeat Counter). Режим Single Shot
* разрешен, поэтому по каждому переполнению таймера
* в память копируется пакет, равный одному байту. */
void SetupWriteChannel( void )
{
  DMA_SetupRepeatBlock( WRITE_CHANNEL, samples,
                          DMA_CH_SRCRELOAD_BLOCK_gc,
    DMA_CH_SRCDIR_INC_gc, (void *) &(LEDPORT.OUT),
                          DMA_CH_DESTRELOAD_NONE_gc,
    DMA_CH_DESTDIR_FIXED_gc,
    SAMPLE_COUNT, DMA_CH_BURSTLEN_1BYTE_gc, 0 );
  DMA_EnableSingleShot( WRITE_CHANNEL );
  DMA_SetTriggerSource( WRITE_CHANNEL, 0x40 );
                          // Старт — по переполнению TCC0.
}

/* Инициализируем таймер TCC0 на генерацию запросов
* на прерывание по переполнению в необходимом темпе:
* установки системы тактирования по умолчанию (нет деления,
* полный период) обеспечат нам частоту выборки порядка 30 Гц
* (2 МГц / 2^16) */
void SetupSampleTimer( void )
{
 TCC0.CTRLA = TC_CLKSEL_DIV1_gc;
 TCC0.PER = 0xFFFF;
 TCC0.INTCTRLA = TC_OVFINTLVL_LO_gc;
 PMIC.CTRL |= PMIC_LOLVLEN_bm;
}

// Процедура для «мигания» светодиодами с комфортным
// для восприятия временем свечения
void BlinkLEDs( void )
{
 LEDPORT.OUT = 0x00;
 __delay_cycles( 2000000 );  // Задержка ~1000 мс
                             // ( 2 МГц, значение для XMEGA
                             // по умолчанию.)
 LEDPORT.OUT = 0xFF;
}

void main( void )
{
 // Назначаем порты ввода/вывода
 SWITCHPORT.DIR = 0x00;
 LEDPORT.DIR = 0xFF;

 // Готовим каналы DMA и «генератор» сэмплирования.
 DMA_Enable();
 SetupReadChannel();
 SetupWriteChannel();
 SetupSampleTimer();
 __enable_interrupt();

 // Основная программа — бесконечный цикл…
 for (;;) {
   // «Мигнем» светодиодами, ждем нажатия любой кнопки,
   // «мигнем» снова
   BlinkLEDs();
   do {} while (SWITCHPORT.IN == 0xFF);
   BlinkLEDs();
```

```
    // Записываем состояние кнопок, пока не заполнится буфер,
    // затем «мигнем» светодиодами.
    DMA_EnableChannel( READ_CHANNEL );
    DMA_WaitAndReturnStatus( (1<<READ_CHANNEL) );
    BlinkLEDs();

    // Ждем нажатия любой кнопки, затем снова «мигнем»
    // светодиодами.
    do {} while (SWITCHPORT.IN == 0xFF);
    BlinkLEDs();

    // Воспроизводим сохраненную последовательность нажатия
    // кнопок на светодиодах снова и снова — до тех пор, пока
    // не будет нажата любая кнопка.
    DMA_EnableChannel( WRITE_CHANNEL );
    do {} while (SWITCHPORT.IN == 0xFF);
    DMA_DisableChannel( WRITE_CHANNEL );

    // Восстанавливаем настройки канала DMA на вывод
    // информации, так как, скорее всего, процесс воспроизведения
    // прервется при нажатии кнопки SW где-то в середине
    // буфера, что оставит канал в неопределенном состоянии.
    SetupWriteChannel();
  }
}
```

Выполним компиляцию примера и запрограммируем микроконтроллер. Соединим гибкими шлейфами PORTD со светодиодами и PORTC с кнопками на плате STK600. Далее будем выполнять следующие действия:

- Запускаем код на исполнение — после этого все светодиоды «мигнут» первый раз.
- Нажимаем любую клавишу — светодиоды «мигнут» снова, приглашая к вводу данных.
- Теперь у нас есть около 10 секунд, в течение которых мы можем в произвольном порядке и умеренном темпе нажимать клавиши на плате STK600. Весь «процесс» (нажатые кнопки, их последовательность и длительность пауз между нажатиями) запоминается в памяти микроконтроллера под управлением контроллера DMA.
- Когда все светодиоды «мигнут» еще раз — время записи истекло.
- Для воспроизведения следует нажать любую кнопку на плате STK600. Светодиоды при этом «мигнут», обозначая начало воспроизведения записанной нами последовательности.
- Теперь контроллер DMA будет циклически воспроизводить сохраненную последовательность (каждой кнопке соответствует свой светодиод: SW0–LED0, и т.д.) и отображать ее на светодиодах набора STK600. Это будет осуществляться до тех пор, пока мы опять не нажмем любую кнопку, показывая, что хотим прервать процесс проигрывания.
- Светодиоды «мигнут», приглашая снова нажать любую кнопку для начала записи новой последовательности…

Заключение

Завершая обзор основных и наиболее интересных аппаратных особенностей нового микроконтроллерного семейства XMEGA, хочется еще раз подчеркнуть следующее. Корпорация Atmel сознательно не стала разрабатывать отсутствующую у нее промежуточную линейку 16-разрядных микроконтроллеров, а постаралась на базе уже имеющейся удачной 8-разрядной платформы индустриального стандарта активно войти «снизу» на рынок 16-разрядников. Не усложняя знакомое, популярное и признанное во всем мире процессорное ядро AVR 8-bit RISC и базовую архитектуру кристалла, разработчикам Atmel удалось сделать очень многое. По функционалу и интеллектуальности развитой периферии современного уровня, которая во многих случаях может работать и принимать решения самостоятельно, без участия процессора, она смогла сравниться с 16-разрядниками, а по производительности — значительно приблизиться к ним. В итоге у компании Atmel получился удобный в работе, современный, не сложный для освоения, быстрый AVR-микроконтроллер, который, несомненно, должен занять достойное место на многих сегментах рынка в самых разнообразных конечных приложениях различного уровня сложности.

Литература

1. Евстифеев А. В. Микроконтроллеры AVR семейств Tiny и Mega фирмы «Atmel». М.: Издательский дом «Додэка-XXI», 2004.
2. XMEGA Training Data // Atmel AVR Distributor Training. Atmel Norway, September 2007.
3. XMEGA Hands-On Session // Atmel AVR Distributor Training. Atmel Norway, September 2007.
4. www.atmel.com