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.

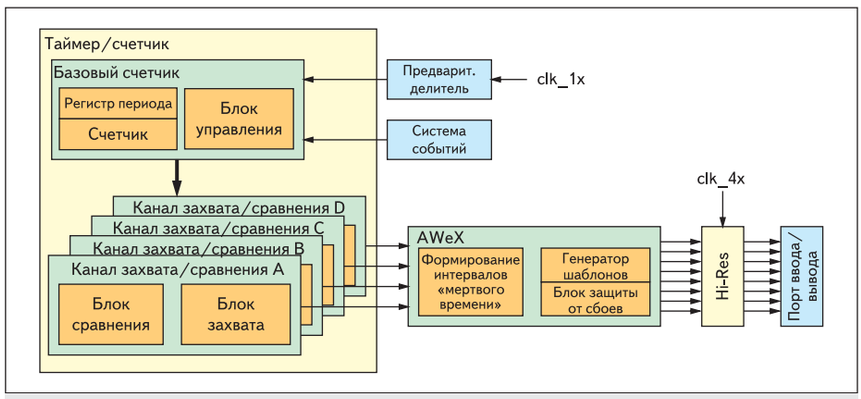

Рис. 1. Структурная схема таймера/счетчика в XMEGA

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.

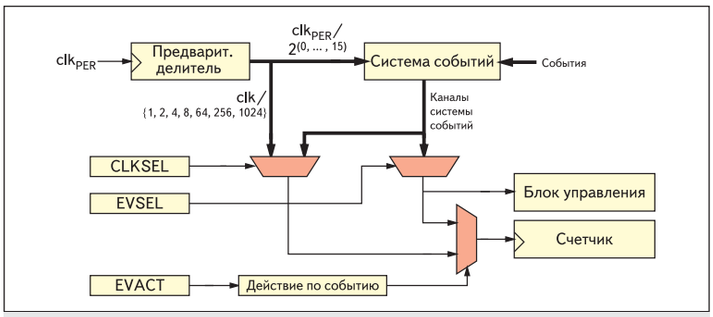

Рис. 2. Тактирование таймера/счетчика XMEGA

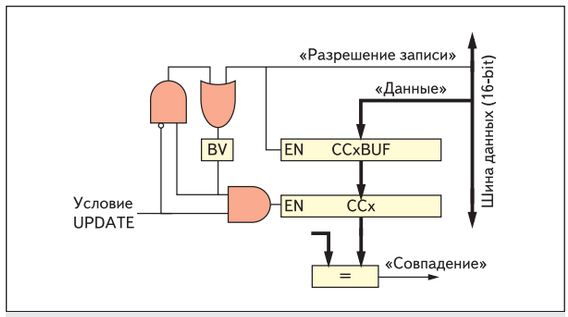

Рис. 3. Функция DBF в режиме сравнения

(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.

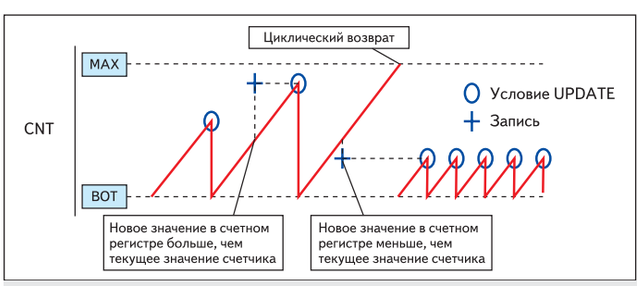

Рис. 4. Режим работы таймера/счетчика без DBF

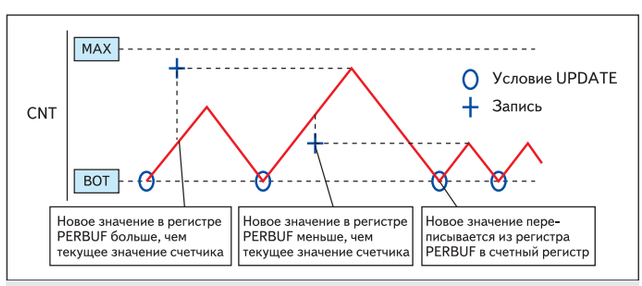

Рис. 5. Работа таймера/счетчика с включенным DBF

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

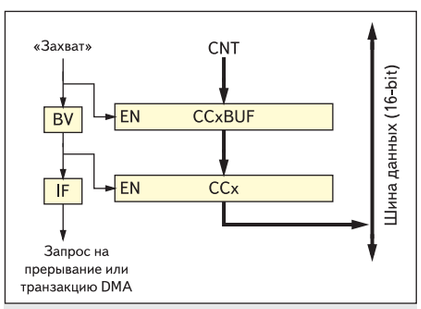

Рис. 6. Функция DBF в режиме захвата

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».

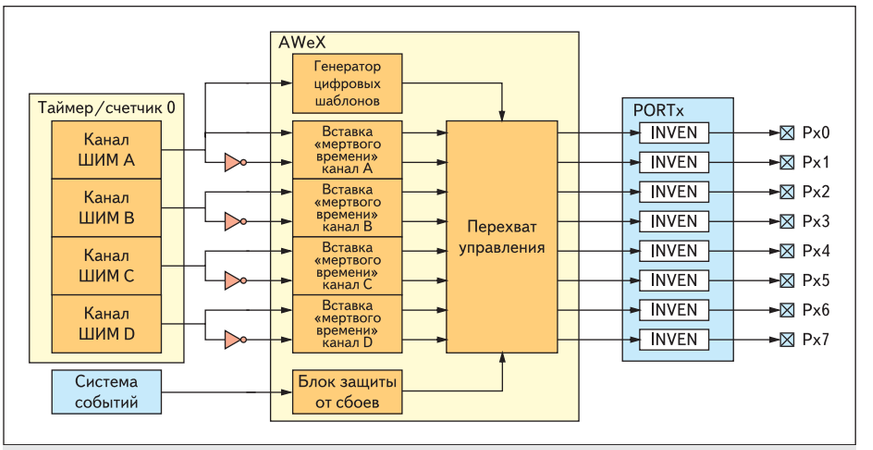

Рис. 7. Блок-схема модуля расширения AWeX

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-

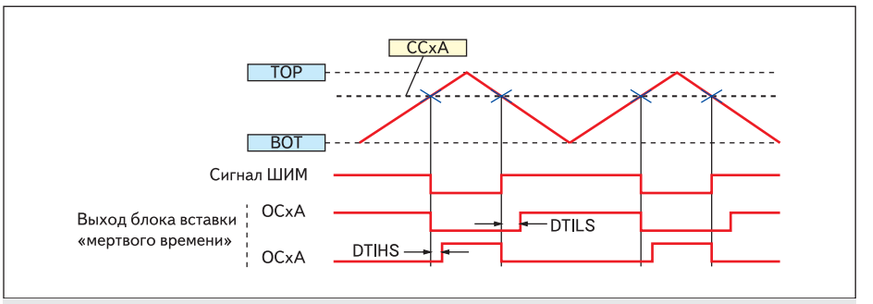

Рис. 8. Формирование интервалов «мертвого времени» в модуле AWeX

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.

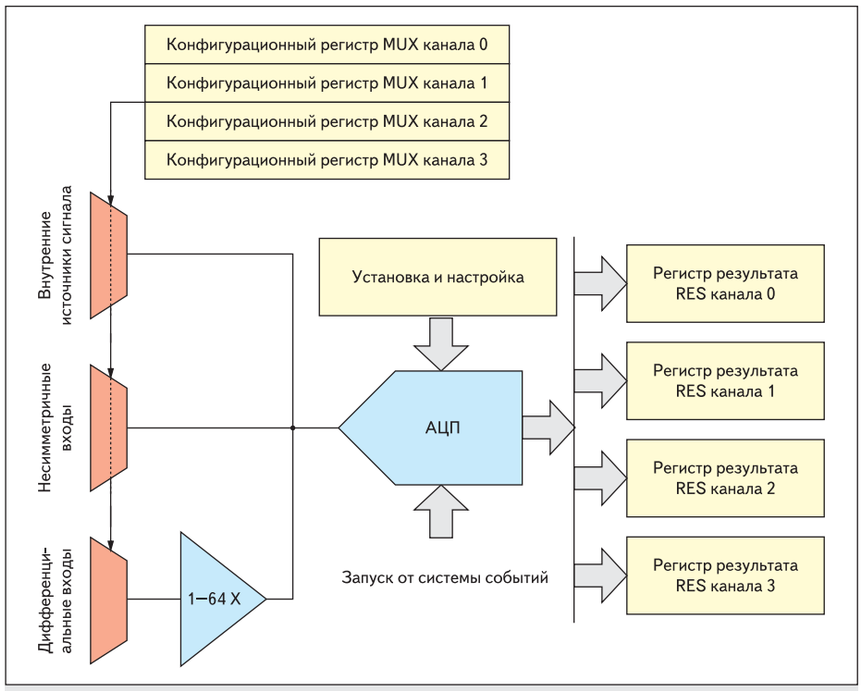

Рис. 9. Блок-схема АЦП в XMEGA

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

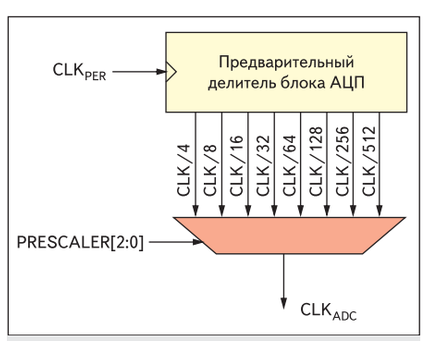

Рис. 10. Тактирование блока АЦП

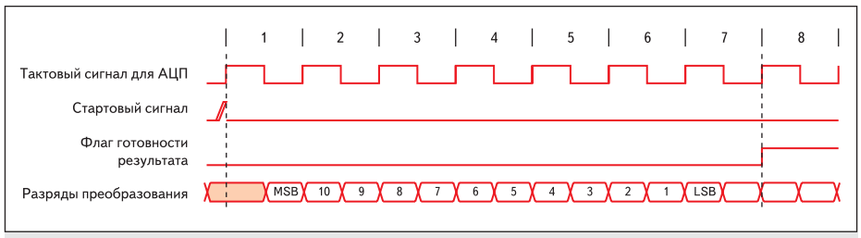

Рис. 11. Пример формирования тактовых последовательностей

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.

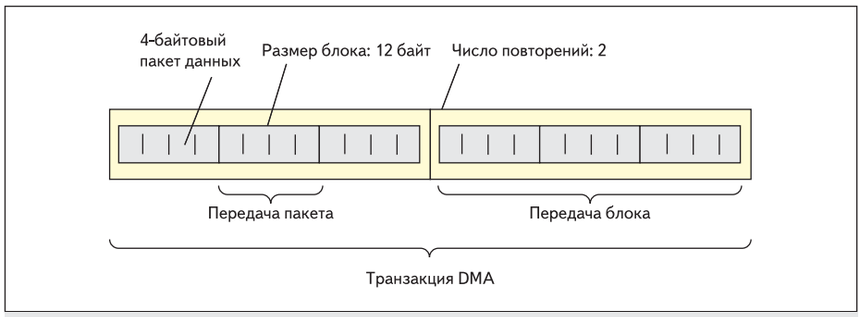

Рис. 12. Пакет, блок и транзакция DMA

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