Быстрое погружение с «черными плавниками».

Часть 3.2. Первая законченная система. Разрабатываем часы для Blackfin. Энергопотребление. Внутренний таймер ядра

Андрей САВИЧЕВ
andrey.savichev@eltech.spb.ru
Олег РОМАНОВ
oleg.rom@eltech.spb.ru

В предыдущей статье цикла (см. «КиТ» № 6 '2007) мы подробно рассматривали работу с часами реального времени в процессорах Blackfin, и теперь готовы приступить к написанию программы. Напомним, что инициализировать RTC в ADSP-BF533 EZ-KIT Lite мы предполагаем через написанный ранее драйвер (см. «КиТ» № 3 '2007) по последовательному каналу с хост-компьютера. Часы и календарь на хост-компьютере работают корректно, но при передаче данных по последовательному каналу и обработке могут произойти сбои.

Для выявления ошибочных значений проинициализируем входными данными вспомогательную структуру-буфер. Если ошибки во входных данных отсутствуют, выведем значения для инициализации в регистры RTC:

```
static unsigned long epoch = 2000;     // year corresponding to 0x00
struct      rtc_time rtc_tm;
unsigned char mon, day, hrs, min, sec, leap_yr;
unsigned int yrs;

//leap_yr — високосный год

static void set_rtc_time(struct rtc_time *rtc_tm)
  {
      yrs = rtc_tm.tm_year + epoch;
      mon = rtc_tm.tm_mon + 1;      // tm_mon starts at zero
      day = rtc_tm.tm_mday;
      hrs = rtc_tm.tm_hour;
      min = rtc_tm.tm_min;
      sec = rtc_tm.tm_sec;

      leap_yr = ((!(yrs % 4) && (yrs % 100)) || !(yrs % 400));
      If ((mon > 12) || (day == 0))
          return Error;
      if ((hrs >= 24) || (min >= 60) || (sec >= 60))
          return Error;

      bfin_write_RTC_STAT(((day << DAY_BITS_OFF) | (hrs <<
HOUR_BITS_OFF) |
      (min << MIN_BITS_OFF) | (sec << SEC_BITS_OFF)));
      wait_for_complete();
  }
```

Мы уже обсуждали различия в структуре регистров RTC для разных типов процессоров Blackfin.

Для **ADSP-BF535**:

```
#define DAY_BITS_OFF          17
#define HOUR_BITS_OFF         12
#define MIN_BITS_OFF          6
#define SEC_BITS_OFF          0
```

Для всех остальных процессоров семейства (за исключением **ADSP-BF561**):

```
#define DAY_BITS_OFF          24
#define HOUR_BITS_OFF         16
#define MIN_BITS_OFF          8
#define SEC_BITS_OFF          0
```

Для удобства вводится следующее макроопределение:

```
#define bfin_write_RTC_STAT(val)
bfin_write32(RTC_STAT,val)
```

и далее в **def_LPBlackfin.h** можно найти определение **bfin_write32(addr,val)**.

Функция **wait_for_complete**()

```
static inline void wait_for_complete(void)
  {
      unsigned long st;

      while ((*pRTC_ISTAT& WCOM)==0);
      *(volatile unsigned short *)RTC_ISTAT |= 0x8000;
  }
```

необходима для ожидания завершения записи в регистры RTC. Можно воспользоваться другим методом и записывать значения в эти регистры в течение 990 мс сразу после события Seconds Event.

Чтение любого регистра RTC (в отличие от записи) может осуществляться в произвольный момент времени, но полученные значения не всегда соответствуют реальным. Это связано с буферизацией этих регистров с помощью «теневых регистров». Вы получаете либо текущее значение, либо предшествующее, в зависимости от момента обращения к регистру. Можно также сказать, что установленный старший бит регистра **RTC_ISTAT** гарантирует вам чтение актуальных значений из регистров RTC.

Для облегчения работы воспользуемся готовой процедурой **get_rtc_time**, а также макро-определением **bfin_read_RTC_ISTAT**():

```
#define bfin_read_RTC_ISTAT() bfin_read16(RTC_ISTAT)
```

В **def_LPBlackfin.h** можно найти определение **bfin_read16(addr)**:

```
static void get_rtc_time(struct rtc_time *rtc_tm)
  {
      unsigned long cur_rtc_stat;

      cur_rtc_stat = bfin_read_RTC_STAT();

      rtc_tm->tm_sec = (cur_rtc_stat >> SEC_BITS_OFF) & 0x3f;
      rtc_tm->tm_min = (cur_rtc_stat >> MIN_BITS_OFF) & 0x3f;
      rtc_tm->tm_hour = (cur_rtc_stat >> HOUR_BITS_OFF) & 0x1f;
      rtc_tm->tm_mday = (cur_rtc_stat >> DAY_BITS_OFF) & 0x7fff;
  }
```

Проверить эти процедуры проще всего с помощью «будильника». Для этого введем в наш проект еще пару готовых процедур:

```
static void set_rtc_alm_time(struct rtc_time *alm_tm);
static void get_rtc_alm_time(struct rtc_time *alm_tm);
```

Их исходные тексты здесь не приводим, но, разумеется, вы их найдете в проекте. Поскольку по структуре регистры часов **RTC_STAT** и будильника **RTC_ALARM** идентичны, то и реализация этих процедур мало чем отличается от **set_rtc_time** и **get_rtc_time** соответственно.

Теперь мы можем инициализировать часы и будильник таким образом, чтобы разница между их показаниями составляла всего несколько секунд при работе в реальном времени. Через назначенный временной

интервал сформируется событие Alarm Event, которое мы можем отметить, например, включением светодиода. Определимся с форматом инициализируемых данных:

```
static const unsigned char rtc_data[ ] =
   { 7, 5, 29,13, 25, 30 };

static const unsigned char rtc_alarm_data[ ] =
   { 7, 5, 29,13, 25, 35 };
```

Это представление времени для часов в **rtc_data**[] «**29-мая-2007 13:25:30**» и для будильника в **rtc_alarm_data**[] «**29-мая-2007 13:25:35**». В дальнейшем именно эти массивы предполагается инициализировать с хост-компьютера. Поскольку на отладочной плате батарея отсутствует, нам придется реинициализировать RTC при включении питания. А как хост-компьютер узнает, что отладочная плата была выключена? Подумайте над этим вопросом самостоятельно. А мы в следующей части статьи предложим свое решение.

```
static unsigned statu;
EX_INTERRUPT_HANDLER(RTC_ISR2)
 {
    statu=*pRTC_ISTAT; // read IRQ Status
    if ( statu&AEF != 0 )
       {;//действия, связанные с событиями флага Alarm Event Flag
          *pFlashA_PortB_Data /= 0x20; //включить светодиод
       }
 …
 *pRTC_ISTAT= AEF;
 while ((*pRTC_ISTAT& WCOM)==0);
 statu=0;
 }
```

Если вы захотите поэкспериментировать с данной программой при более существенной разнице во времени между будильником и часами, рекомендуем сбросить флаг Prescaler Enable:

```
*pRTC_PREN = 0x0;
```

В этом случае и часы, и будильник пойдут в 32 768 раз быстрее, и вам не придется слишком долго ждать результата. Все исходные тексты проекта будут доступны по адресу http://avisv.narod.ru/BlackFin/project2.htm.

Давайте уделим немного внимания другим подсистемам службы времени Blackfin — таким как таймеры общего назначения general-purpose timers и «внутренний таймер ядра» core timer («системный таймер»). Мы не будем рассматривать все возможные режимы работы этих устройств, а просто приведем примеры из работающих программ. Например, в демонстрационной программе от Analog Devices, рассмотренной нами в первой части статьи нашего цикла, есть такой фрагмент:

```
*pTIMER0_CONFIG          = 0x0019;
*pTIMER0_PERIOD          = 0x0001000;
*pTIMER0_WIDTH           = 0x0000400;
*pTIMER_ENABLE           = 0x0001;
```

Для каждого таймера, независимо от типа процессора ADSP Blackfin, существует регистр конфигурации **TIMERx_CONFIG** (рис. 1), в котором определены режимы его работы, параметры цифрового сигнала на выходе (если он формируется), источник тактирования, разрешение прерывания. Запись в регистр конфигурации возможна только тогда, когда таймер остановлен. Проверить это можно, анализируя соответствующий бит TRUNx в регистре **TIMER_STATUS** (рис. 2). Вы можете дать команду на остановку таймера, установив соответствующий бит TIMDISx в регистре **TIMER_DISABLE** (рис. 3). При этом таймер остановится не сразу, а по завершении формирования импульса, как если бы вы работали в режиме PWM_OUT с PERIOD_CNT=0. Чтобы добиться мгновенной остановки таймера, запишите дополнительно «1» в соответствующий бит TRUNx.

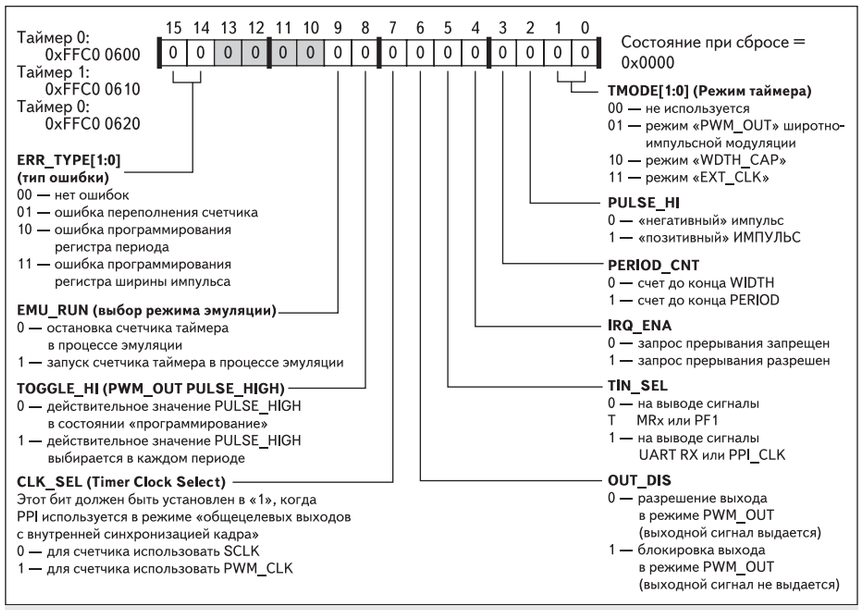

Рис. 1. Формат регистра конфигурации таймеров общего назначения TIMERx_CONFIG

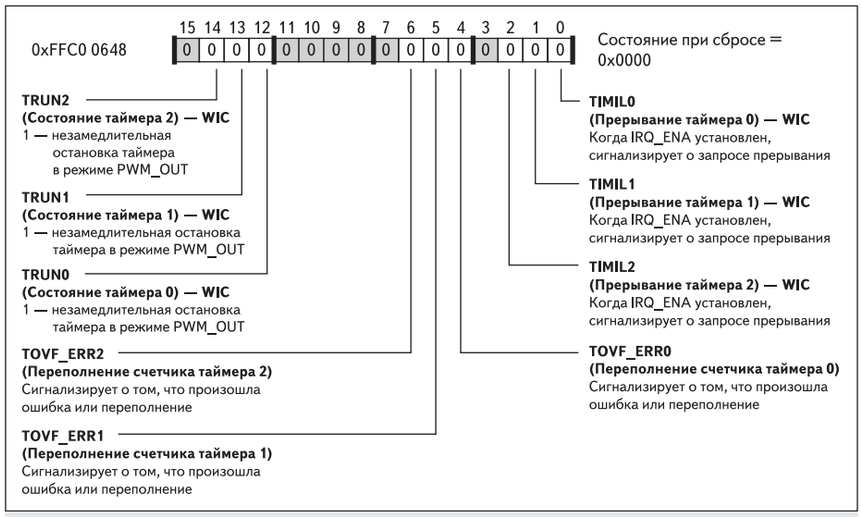

Рис. 2. Формат регистра состояний таймера общего назначения TIMER_STATUS

В нашем примере в качестве режима работы выбран широтно-импульсный модулятор с первоначально нулевым уровнем сигнала (PULSE_HI=0) и режимом счета до конца периода.

В этом режиме сигнал на выводах таймеров формируется в соответствии со значениями в регистрах **Timerx_WIDTH** и **Timerx_PERIOD**, битовых полей Pulse_HIGH, Period_CNT, TOGGLE_HI и в зависимости от выбранного источника тактовых сигналов CLK_SEL регистра **TIMERx_CONFIG**. Но если, как в нашем случае, состояние выхода несущественно, то ситуация с инициализацией регистров general-purpose timers несколько упрощается. Сейчас нас интересует только прерывание, которое возникает через строго определенные интервалы времени, поэтому и другие

режимы работы таймеров общего назначения, системного таймера или рассмотренных выше часов реального времени вполне решали бы задачу. Подробнее о работе таймера общего назначения в режиме PWM_OUT можно посмотреть в справочнике [1] или в его русском переводе на сайте представительства Analog Devices в СНГ и Балтии **http://www.analog.spb.ru/pub_dsp.htm** в файле chapter15.pdf.

Блок-схема таймера общего назначения представлена на рис. 5

Как программировать системные тики (system tick — прерывания от таймера) — основу нашей системы реального времени? Прежде всего, нужно выбрать их периодичность в зависимости от производительности системы (процессора) и желаемого времени отклика на события. Если снизить эту величину до минимально возможной, то процессор вместо полезной загрузки приложениями будет слишком часто «отвлекаться» на обслуживание системных тиков и связанные с их обслуживанием процедуры. С другой стороны, если сделать их чрезмерно редкими, система окажется слишком медлительной, и события будут отслеживаться с большим запаздыванием. В таких системах иногда для получения быстрого отклика на определенные события прибегают к грубым «внесистемным» решениям, обходя «системный диспетчер» (как раз те самые процедуры, которые обслуживаются при возникновении очередного системного тика), а то и просто глобально блокируя все остальные события до завершения обработки. Именно в этом случае эффективны «часы реального времени», так как с их помощью можно оценить временной интервал, в течение которого система находилась в режиме монопольного обслуживания определенного события. Для двухъядерного ADSP-BF561 в этих целях можно использовать более тонкий механизм, но это тема для отдельной статьи. Итак, крайне желательно ввести именованную константу, чтобы затем можно было подобрать ее величину в процессе «тонкой настройки» системы.

```
static const unsigned char our_system_tick = 20; // system tick in mks
```

Производительность процессора и потребление энергии в процессорах ADSP-Blackfin может регулироваться программно. Важно знать, какая частота присутствует на входе CLKIN. На плате **ADSP-BF533 EZ-KIT Lite** на этот вход заведена частота 27 МГц. Это кроме всего прочего объясняется еще и тем, что ФАПЧ лучше всего работает при не слишком больших коэффициентах DIVIDER, и минимально допустимая частота на входе CLOCK DIVIDE AND MUX — 50 МГц (рис. 6).

Комбинации значений в MSEL и SSEL для SCLK, и MSEL и CSEL для CCLK определяют тактовые частоты для периферии и ядра соответственно. При этом если бит Divide

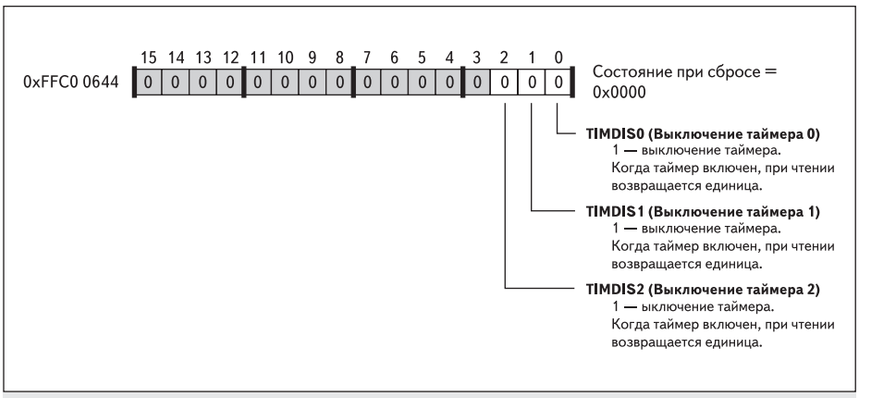

Рис. 3. Формат регистра запрещения таймеров общего назначения TIMER_DISABLE

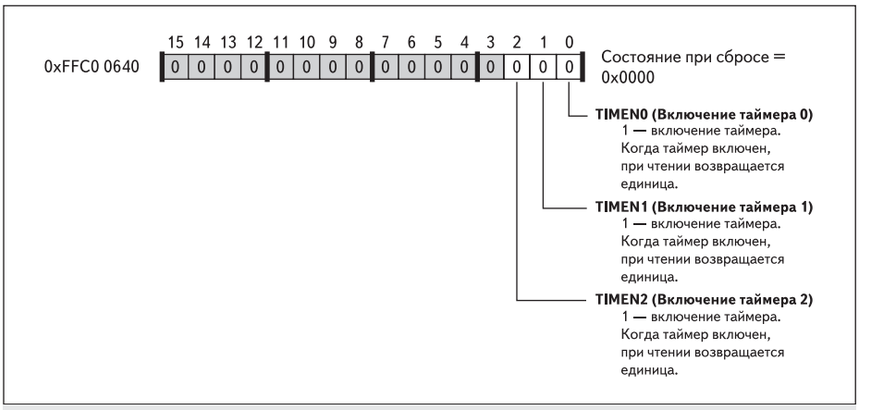

Рис. 4. Формат регистра разрешения таймеров общего назначения TIMER_ENABLE

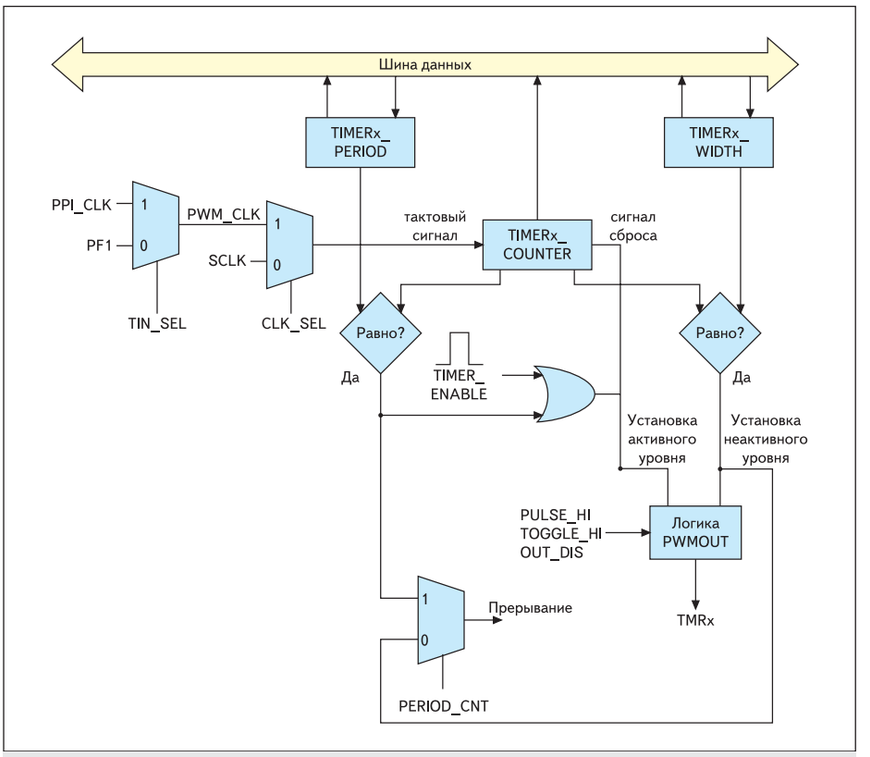

Рис. 5. Блок-схема таймера общего назначения процессора Blackfin

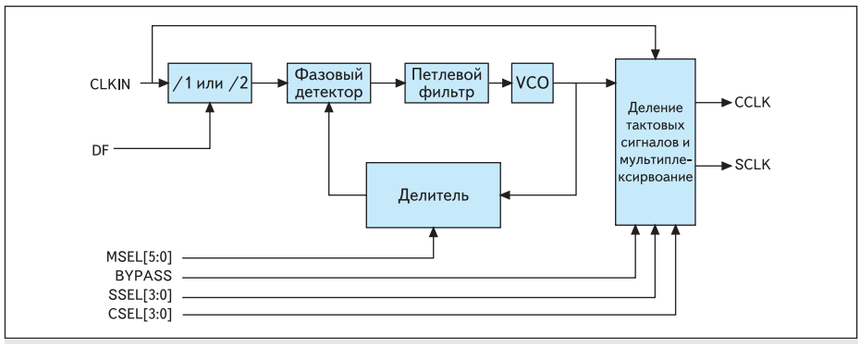

Рис. 6. Блок-схема фазовой автоподстройки частоты ФАПЧ процессора Blackfin с умножителем частоты

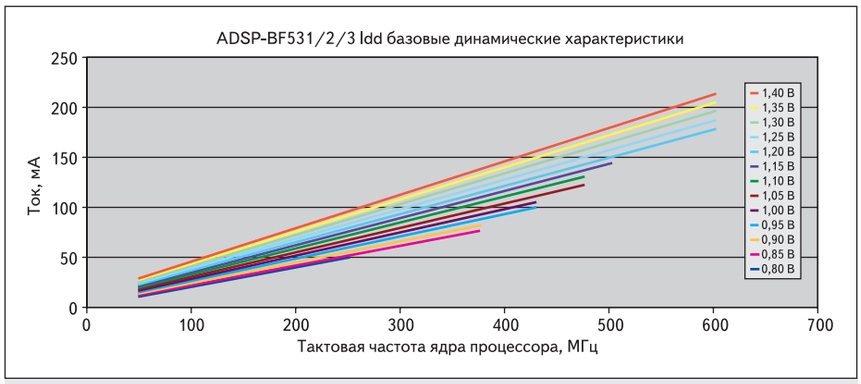

Рис. 7. Допустимые рабочие частоты ядра процессора и потребляемый ток в зависимости от питающего напряжения

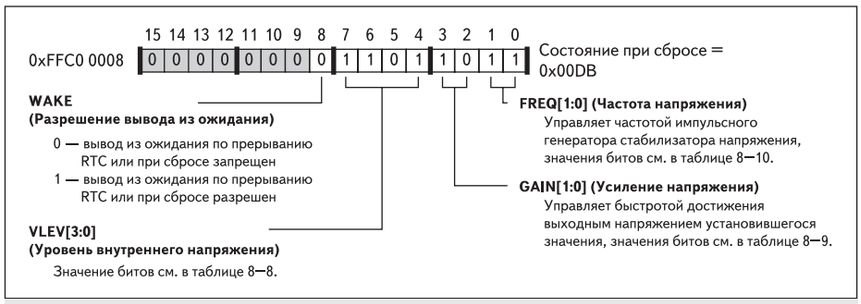

Рис. 8. Формат регистра управления регулятором напряжения VR_CTL

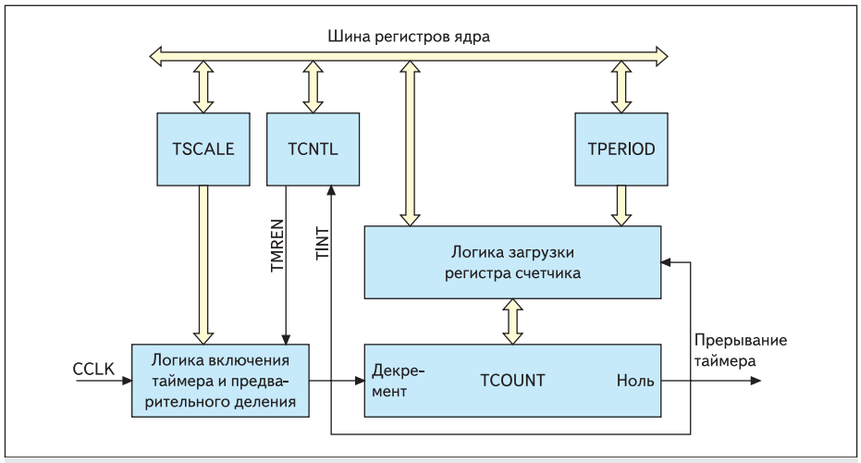

Рис. 9. Блок-схема таймера ядра core timer

Таблица. Величина напряжения питания процессорного ядра в зависимости от значения поля Voltage Level (VLEV), VR_CTL[7:4] регистра управления регулятором напряжения VR_CTL

VLEV	Напряжение, В
0000–0101	Не используется
0110	0,85
0111	0,90
1000	0,95
1001	1,00
1010	1,05
1011	1,10
1100	1,15
1101	1,20
1110	1,25
1111	1,30

Frequency (DF) установлен, частота CLKIN умножается на 2. Например, если в MSEL[5:0] записан код 111111 — 63, то при DF=1 частота на входе CLOCK DIVIDE AND MUX будет 850 МГц, а при DF=0 — 1700 МГц. Обратите особое внимание на то, что при нулевом значении поля MSEL[5:0] входная частота возрастает соответственно в 32 или 64 раза! Делители частоты различаются по разрядности. Если тактовый сигнал для CCLK можно разделить максимум на 8 (2-битное поле CSEL[1:0]), то сигнал для SCLK — на 15 (4-битное поле SSEL[3:0]).

Это вполне логично, ведь входная тактовая частота у них одинаковая, а периферия, как правило, должна работать медленнее, чем процессорное ядро. На более низких рабочих частотах ядра можно снижать тактовую частоту работы с периферией (с внешней памятью в том числе) в значительно меньшей степени. Если внешняя шина в вашей системе стандартна, то перед вами стоит задача более точного подбора коэффициентов: DIVIDER, VCO, DF. Есть ограничение и на тактирование ядра процессора в зависимости от исполнения и типа корпуса, а периферии — в зависимости от числа используемых выводов GPIO и электрических параметров подключаемых устройств.

Каждый тип корпуса при определенном количестве выводов характеризуется вполне определенным коэффициентом теплоотдачи, который, в свою очередь, определяет полное тепловое сопротивление. Для корпуса LQFP со 176 выводами, в котором выпускаются некоторые процессоры ADSP-BF531/2/3, оно составляет приблизительно 21 °C/Вт при естественном воздушном охлаждении. Ядро процессоров Blackfin обладает малым энергопотреблением. Так, на рабочей частоте 600 МГц при питающем напряжении ядра 1,2 В это составляет порядка 330 мВт. При этом вклад ядра в повышение температуры корпуса составляет не более 9 °C. При выполнении различных последовательностей инструкций потребляемый ток (и выделяемая мощность) могут отличаться от базовых параметров, приведенных на рис. 7, на 27% как в большую, так и в меньшую сторону. Более сложной является оценка вклада в выделяемую

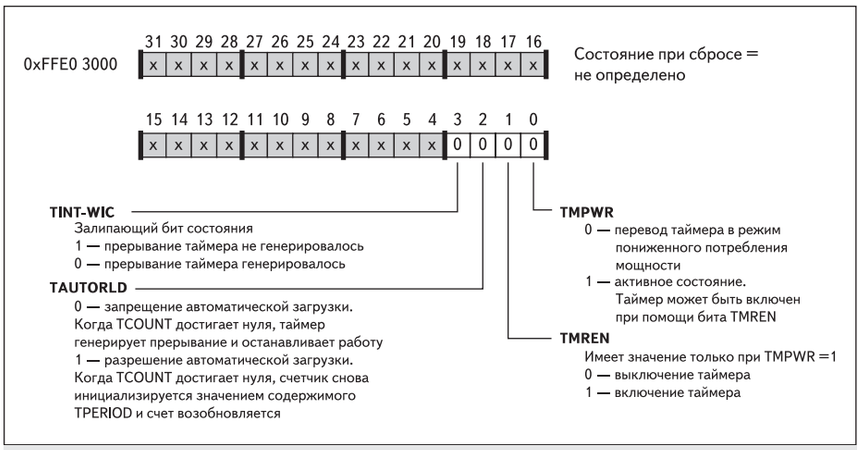

Рис. 10. Формат регистра управления таймера ядра TCNTL

кристаллом тепловую мощность периферии. На сайте Analog Devices в разделе **Blackfin Technical Library** http://www.analog.com/processors/blackfin/technicalLibrary/index.html выложены специальные XLS-файлы. С их помощью вы можете произвести оценочный расчет выделяемой при работе на заданной частоте определенного периферийного устройства тепловой мощности. Заметим, что это будет весьма грубая оценка, так как при этих расчетах не учитывается ни топология печатной платы, ни электрические параметры подсоединяемых к процессору устройств. Поэтому необходим запас по суммарной выделяемой тепловой мощности и току источника питания. На рис. 7 изображены динамические характеристики для процессоров ADSP-BF531/2/3, из которых видно, что при низких напряжениях питания ядра предельная рабочая частота существенно снижается. Так, при 0,8 В рабочая частота процессорного ядра не превышает 250 МГц.

У процессоров ADSP Blackfin питающее напряжение ядра изменяется программированием значения поля VLEV[3:0] в регистре VR_CTL (рис. 8).

В таблице приведены значения этого 4-битового поля.

В реальной системе (особенно в опытных образцах) можно установить на корпусе процессора LQFP-176 (26×24 мм) температурный датчик, например ADI TMP05 в корпусе SOT23(2,9×1,6 мм) с ШИМ-выходом, и написать процедуру, управляющую выбором рабочей частоты и питающего напряжения ядра. Подобранные таким методом оптимальные значения коэффициентов можно включить в процедуру инициализации.

Различные режимы энергосбережения процессоров Blackfin рассмотрим чуть позже, а теперь вернемся к службе времени. Рассмотрим работу с внутренним таймером ядра. Его можно использовать для различных сервисных процедур в основном при отладке и «тонкой настройке» системы, либо когда вам катастрофически не хватает таймеров общего назначения (хотя в этом случае можно выбрать кристалл семейства, где их не 3, а 4, 8 или даже 12). Но все же основная функция таймера ядра — генерировать прерывания через определенные пользователем промежутки времени. Поэтому, если вас не интересует наличие в готовой системе сервиса, связанного с измерением интервалов между различными событиями и времени выполнения определенных последовательностей кода, то логично использовать для формирования системных тиков именно этот таймер.

Как видим на рис. 9, этот таймер декрементный. Значение, которое вы записываете в регистр **TCOUNT** счетчика таймера ядра, через интервал TSCALE+1 тактов CCLK декрементируется. При достижении нулевого значения в регистре TCOUNT устанавливается TINT в регистре **TCNTL** (рис. 10). Если прерывания от таймера ядра не были разрешены, то этот бит не устанавливается. Дальнейшие действия зависят от того, была ли предварительно записана в бит TAUTORLD «1». Если да, значение регистра счетчика **TCOUNT** (рис. 11) инициализируется содержимым регистра **TPERIOD** (рис. 12) и декрементный счет автоматически продолжается.

Он может быть прерван записью «0» и вновь запущен записью «1». Вы можете использовать 8-битное значение **TSCALE** (рис. 13) для

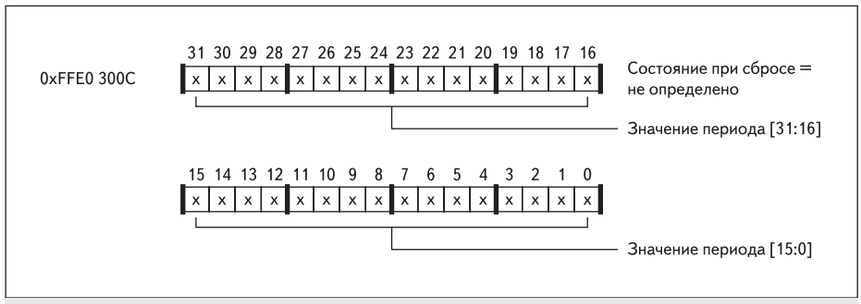

Рис. 11. Биты регистра счетчика таймера ядра TCOUNT

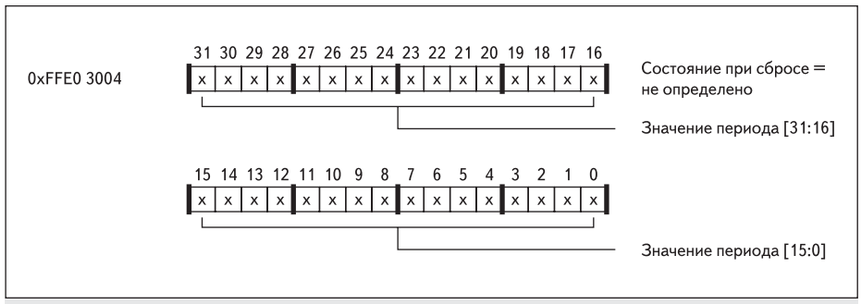

Рис. 12. Биты регистра периода таймера ядра TPERIOD

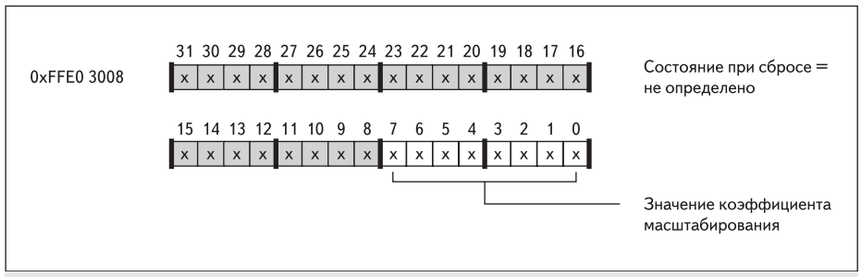

Рис. 13. Биты регистра масштабирования таймера ядра TSCALE

изменения периодичности возникновения прерываний в 256 раз при отладке программы. Это актуально, например, для систем на основе ADSP-BF561, в котором отсутствуют RTC.

Готовый проект для процессора ADSP-BF561, использующий прерывания от таймера ядра, можно посмотреть на сайте **http://www.eltech.spb.ru/techinfo.html**.

На этом мы заканчиваем наш краткий обзор службы времени процессоров семейства Blackfin и в следующей части статьи обсудим в деталях нашу законченную систему реального времени.

Литература

1. ADSP-BF533 Blackfin Processor Hardware Reference.
2. Engineer-to-Engineer Note. Estimating Power for ADSP-BF531/BF532/BF533 Blackfin Processors
3. Руководство пользователя по цифровым сигнальным процессорам семейства Blackfin.