Быстрое погружение с «черными плавниками».

Часть 3.1. Первая законченная система. Служба времени в ADSP Blackfin

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

Как было обещано в предыдущей статье нашего цикла («Компоненты и технологии» № 3 `2007), мы приступаем к созданию законченной системы на ADSP-BF533 EZ-KIT Lite.

Функции нашей системы на ADSP-BF533 EZ-KIT Lite достаточно примитивны. Используя командный файл на хост-компьютере, мы будем обрабатывать его команды в реальном времени, пересылая их посредством достаточно известного среди российских разработчиков встроенных систем протокола WAKE на нашу оценочную плату. Наблюдать выполнение команд можно с помощью светодиодов оценочной платы. Нажатие на кнопки на ADSP-BF533 EZ-KIT Lite будет не только инициировать определенные команды, но и фиксироваться в лог-файле на хост-компьютере. Еще одной функцией нашей системы будет переход в режим минимально возможного энергопотребления при отсутствии новых команд с хост-компьютера за установленный интервал времени. При возобновлении потока команд процессор ADSP-BF533 автоматически возвратится в режим нормальной работы. Но сначала рассмотрим работу службы времени у процессоров ADSP-BF533, поскольку ее значение для встроенных приложений трудно переоценить.

Раньше мы обходили стороной вопросы инициализации таймеров и действий с ними именно потому, что не занимались вплотную разработкой системы. Каковы же особенности класса устройств «службы времени» семейства Blackfin ADSP-BF5xx? Прежде всего, отметим, что практически любой представитель семейства обладает стандартным набором таких устройств. Это рационально с точки зрения дизайна как самих микросхем, так и проектов на их основе, поскольку характеристики ADSP-BF5xx позволяют использовать готовые операционные системы реального времени от самых простых до весьма сложных. Но об этом чуть позже. Давайте посмотрим, как строить собственную систему, не используя при этом готовые.

Такой подход оправдан лишь в двух случаях:

- ваша задача не допускает применения ни одной из систем из-за своей специфики;
- время на разработку крайне ограничено, и ваша система относительно проста.

Стандартный набор устройств:

- таймеры общего назначения (general-purpose timers);
- внутренний таймер (core timer);
- сторожевой таймер (watchdog timer).

Количество таймеров общего назначения у различных микросхем семейства приведено в таблице.

По набору устройств «службы времени» в семействе ADSP Blackfin тоже можно выделить три группы. На большинстве чипов (кроме двухъядерного ADSP-BF561) интегрированы «часы реального времени» Real Time Clock RTC. Встречается две модификации

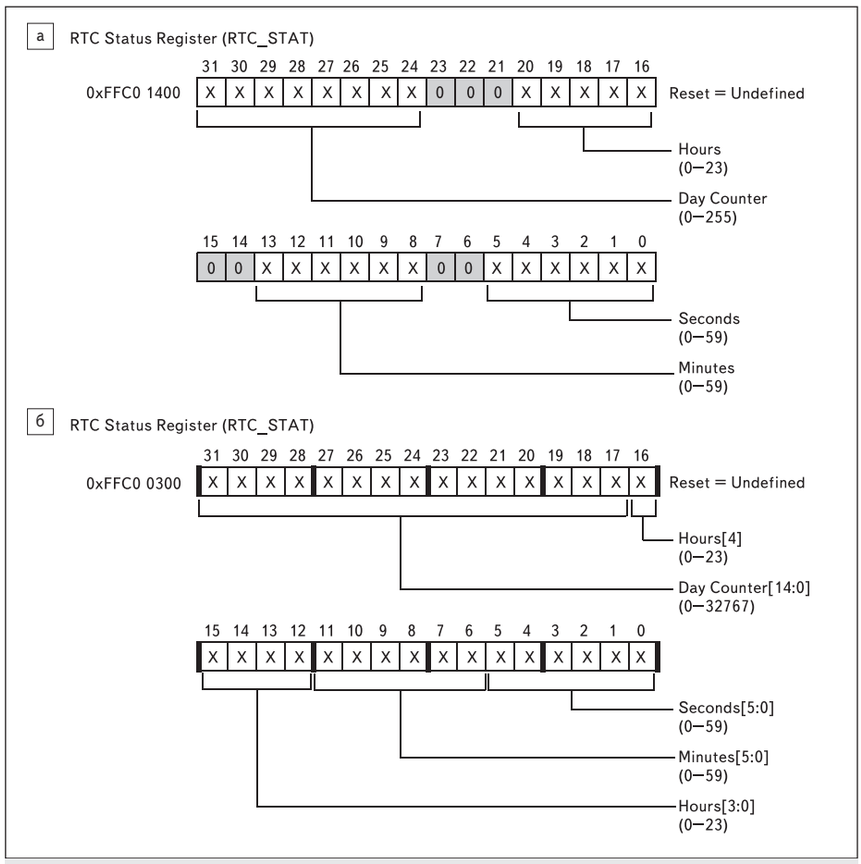

Рис. 1. а) RTC_ALARM регистр ADSP-BF535; б) RTC_ALARM регистр ADSP-BF53x других типов

Таблица. Количество таймеров общего назначения в различных микросхемах ADSP Blackfin

Тип микросхемы	Количество
ADSP-BF561	12
ADSP-BF534, BF536, BF537	8
ADSP-BF538, BF538F	4
ADSP- BF531, BF532, BF533, BF535	3

RTC. В чипе ADSP-BF535 (рис. 1а) поле Day регистра RTC Alarm Register RTC_ALARM 15-битное, в то время как у всех остальных групп чипов оно 8-битное (рис. 1б).

Чтобы не перегружать статью деталями, скажем только, что различия между чипами этих двух групп встречаются в структуре и других регистров RTC. Поскольку для всех представителей семейства Blackfin существуют заголовочные файлы cdefBF53x.h с описаниями битовых полей этих регистров, вы можете включить один из них в свой проект, или использовать при создании своего собственного как образец.

Давайте разработаем простейшие часы для Blackfin. Потом мы сможем применить их в нашей системе. Понятно, что часы могут быть реализованы и программно. Например, если они весьма специфичны, или в том случае, если вы используете ADSP-BF561. Но в наших часах наряду с этим будут использованы аппаратные регистры и прерывания RTC (рис. 2).

Можно сказать, что для RTC ADSP Blackfin Analog Devices предлагает минималистский дизайн. Для процессоров, используемых не во встроенных приложениях, это могло бы считаться недостатком. Но для разработчиков embedded system это не является проблемой. В примерах к Visual DSP++ отыскать что-либо по данному вопросу не удастся, а вот в исходниках uCLinux находим файл \drivers\char\bfin_rtc.c, в котором нас, прежде всего, интересует следующее описание:

```
struct rtc_time rtc_timer;
```

Описатель для структуры rtc_time находим в файле \linux\rtc.h:

```
struct rtc_time {
        int tm_sec;
        int tm_min;
        int tm_hour;
        int tm_mday;
        int tm_mon;
        int tm_year;
        int tm_wday;
        int tm_yday;

};
```

Примечание: ***tm_mday*** — день в месяце; ***tm_mon*** — месяц в году; ***tm_year*** — год; ***tm_wday*** — день в неделе; ***tm_yday*** — день в году.

По своему желанию вы можете дополнить часы лунным, китайским, астрологическим и другими календарями. Закладывать особенности (наименование) каждого дня и года по одному из календарей в регистры RTC — не лучшее решение. Комбинированный (аппаратно-программный) метод реализации часов допускает локализацию версии вашего программного обеспечения в зависимости от месторасположения и предпочтений клиентов. Но все календари основаны не только на современных реалиях, но и на данности нашего с вами существования на Земле в Солнечной системе.

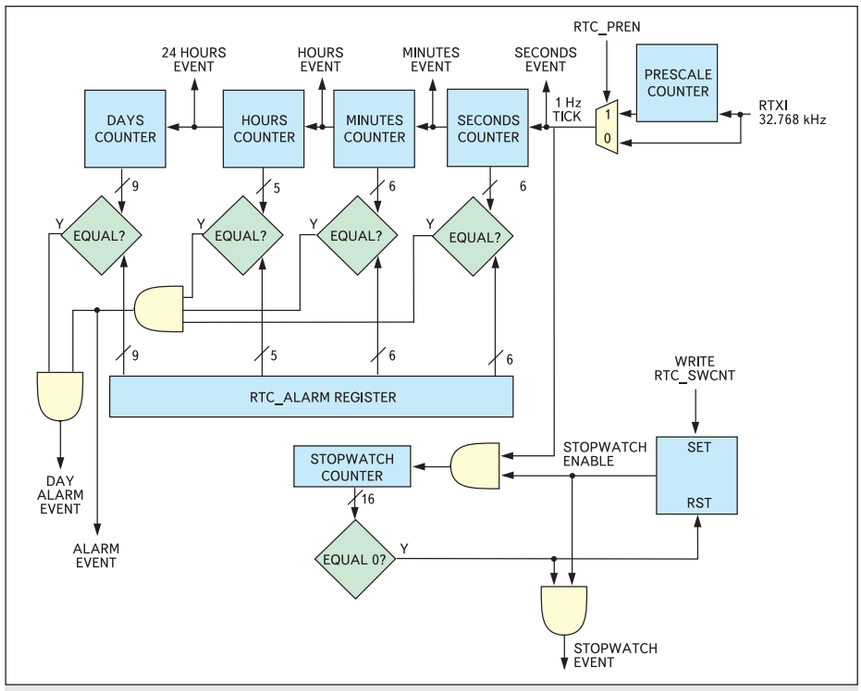

Рис. 2. Блок-схема часов реального времени

Современный григорианский календарь сменил использовавшийся во времена Римской империи юлианский календарь потому, что с помощью измерений было выявлено, что погрешность летоисчисления в старом календаре составляла сутки за сто двадцать восемь лет. Со времен Гая Юлия Цезаря было известно, что Земля вращается вокруг Солнца за 365,25 суток. Накапливающаяся за четыре года погрешность, равная одному дню, снималась введением «лишнего» дня, который римляне именовали «бис сектум». Отсюда наш «високосный» год этимологически, а фактически он «из астрономии».

Дни по месяцам в обычном, не високосном году распределены следующим образом:

```
static const unsigned char days_in_mon[] =
{ 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

RTC могут работать в режимах «будильника», «таймера» и собственно часов реального времени параллельно и независимо, но прерывания, которые при этом возникают, могут быть привязаны только к одному вектору. Это означает, что программа обслуживания прерывания Interrupt Handler у них общая. Правда, можно избирательно маскировать каждый из возможных источников для этого вектора.

Для этого можно сбросить соответствующие биты в регистре RTC_ICTL (рис. 3).

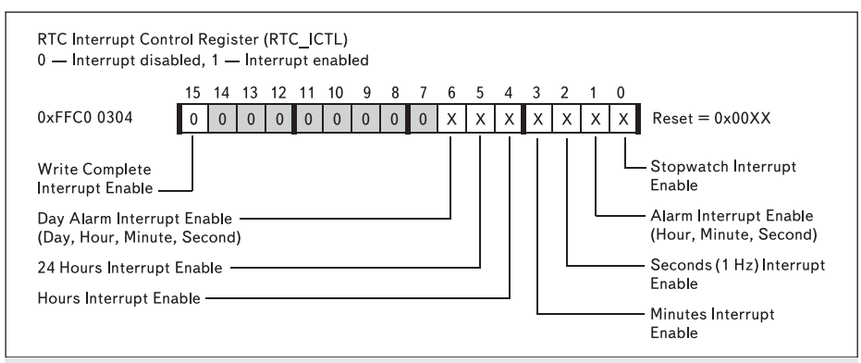

Рис. 3. Биты регистра управления RTC прерываниями RTC_ICTL

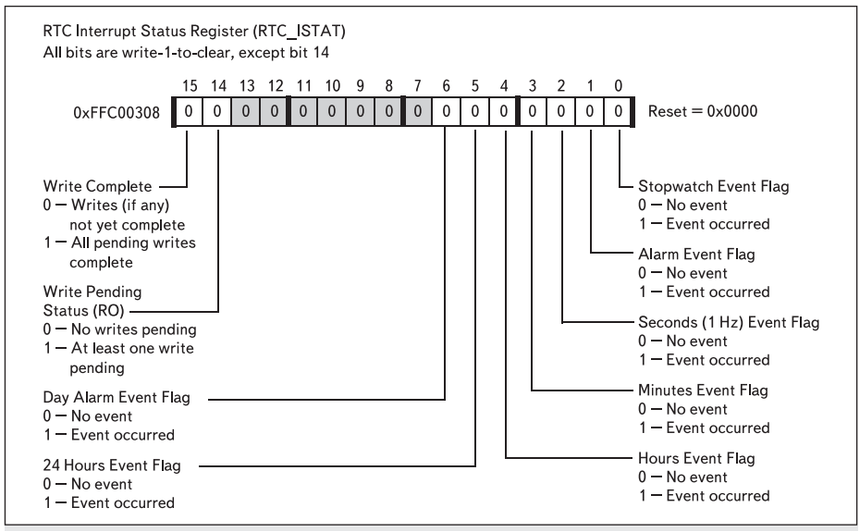

Рис. 4. Биты регистра статуса RTC прерываний RTC_ISTAT

При подаче питания и сбросе Reset состояние этих битов может быть неопределенным.

Какое именно прерывание запускает программу обслуживания, можно установить, анализируя биты регистра RTC_ISTAT. Как видно из рис. 4, в этом регистре есть флаги всех трех устройств RTC: будильника (Day Alarm Event Flag и Alarm Event Flag), таймера (Stopwatch Event Flag) и собственно часов реального времени (Seconds Event Flag, Minutes Event Flag, Hours Event Flag, 24 Hours Event Flag). Эти флаги могут опрашиваться и тогда, когда прерывания замаскированы в RTC_ICTL, и тогда, когда запрещен их общий вектор в SIC_IMASK.

Пожалуйста, для корректной работы с прерываниями RTC обратите внимание на два момента. Биты в RTC_ISTAT устанавливаются не мгновенно, поэтому между последовательными записями в регистры RTC должна быть пауза. Вы можете получить событие о готовности RTC к следующей записи, или просто опрашивать бит 15-го регистра RTC_ISTAT. Какой из этих двух способов выбрать — зависит от вашей конкретной системы. Событие о готовности регистров RTC для записи попадает на тот же вектор, что и остальные события events, связанные с регистрами RTC.

Второй момент связан с тем, что флаги не сбрасываются автоматически при выходе из программы обслуживания прерывания. И пока они не сброшены, новые прерывания от этого источника больше не возникают!

Наконец, общий момент, важный для обслуживания любых прерываний. Прерывание должно быть разрешено в регистре RTC_ICTL, и его вектор должен быть записан в таблицу векторов событий EVT. Это делается с помощью предопределенных в Visual C++ макро:

```
register_handler( ik_ivg8, RTC_ISR2 );
```

Сама программа обслуживания прерывания может выглядеть, например, так:

```
static unsigned statu;
EX_INTERRUPT_HANDLER(RTC_ISR2)
{
statu=*pRTC_ISTAT; // read IRQ Status
if ( statu&SWEF != 0 )
{ ;//действия связанные с событиями флага Stopwatch Event Flag
}
…
*pRTC_ISTAT=SWEF;
while ((*pRTC_ISTAT& WCOM)==0);
statu=0;
}
```

SWEF, WCOM и другие идентификаторы флагов RTC_ISTAT определены в файле defBF532.h. Иногда для отладки программ желательно «масштабировать время», ускорив ход часов RTC. Analog Devices предусмотрела такую возможность. Часы пойдут в 32 768 раз быстрее, если вы сбросите бит Prescaler Enable в регистре RTC_PREN.

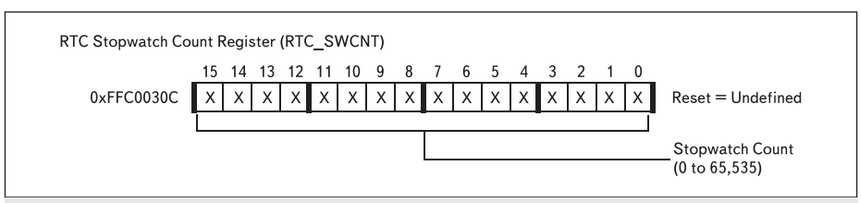

Рис. 5. Регистр счетчика таймера RTC_SWCNT

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

Чтобы часы работали в нормальном темпе, этот бит должен быть установлен.

Работа с таймером RTC совершенно тривиальна. Вы записываете необходимую величину задержки в регистр RTC_SWCNT (рис. 5), после чего начинается обратный отсчет по событиям Seconds Event Flag. Когда значение регистра RTC_SWCNT становится нулевым, возникает событие Stopwatch Event.

Немногим сложнее работать с будильником. Структура регистра RTC_ALARM представлена на рис. 1. Событие Day Alarm Event возникает, когда совпадают по величине значения полей Day в регистрах RTC_ALARM и RTC_STAT. Событие Alarm Event, как видно из рис. 2, возникает при совпадении по величине значений всех полей в регистрах RTC_ALARM и RTC_STAT: Day, Hours, Minutes, Seconds. Это событие может быть использовано для «пробуждения» Blackfin из состояния «глубокого сна» Deep Sleep, в котором не тактируются периферийные устройства и ядро (отсутствуют тактовые импульсы CCLK и SCLK, процессор остановлен и не выполняется программа). Также снимается тактирование с внешней памяти SDRAM, поэтому необходимо позаботиться о сохранении данных:

```
#define PDWN      0x00000020 // Put the PLL in a Deep Sleep state
```

Эта операция переключает Blackfin в состояние Deep Sleep:

```
*pPLL_CTL |= PDWN;
```

В следующей части статей цикла мы напишем реальный обработчик прерываний RTC и две процедуры для чтения и записи данных в регистры RTC:

```
static void get_rtc_time(struct rtc_time *rtc_tm);
static void set_rtc_time(struct rtc_time *rtc_tm);
```

После этого продолжим разработку нашей системы.

Если у вас возникли вопросы, замечания, уточнения или пожелания, направляйте их авторам по адресам электронной почты, указанным в начале статьи.

Продолжение следует

Литература

1. ADSP-BF533 Blackfin Processor Hardware Reference. http://www.analog.com/UploadedFiles/Associated_Docs/892485982bf533_hwr.pdf
2. ADSP-BF535 Blackfin Processor Hardware Reference. http://www.analog.com/UploadedFiles/Associated_Docs/365066229ADSP_BF535_HRM.pdf