Blackfin VisualDSP++

scmRTOS

ОПЕРАЦИОННАЯ СИСТЕМА РЕАЛЬНОГО ВРЕМЕНИ

для однокристальных микроконтроллеров

Version 5

Общие сведения

Данный порт предназначен для использования совместно с программным пакетом **VisualDSP++** фирмы **Analog Devices**.

Blackfin – процессор, изначально рассчитанный на использование его под управлением операционных систем, поэтому в его составе имеется ряд средств для поддержки ОС. Сюда относятся наличие, например, режимов **User** и **Supervisor** и поддержка программного прерывания. Для ***scmRTOS*** самым важным из них является поддержка программного прерывания.

В данном порте режим **User** по причинам, связанным с удобством использования процессора в сочетании с низкоуровневым ПО, не используется, а вся работа происходит в режиме **Supervisor**, переключение на который происходит на этапе выполнения кода **Startup**.

Программный пакет **VisualDSP++** использует один стек для данных и адресов возвратов.

Рекомендуемое представление нумерации приоритетов в данном порте идет по убыванию: максимальное значение числа[1], представляющего номера приоритета, соответствует процессу с наивысшим приоритетом, по мере убывания значения, уровень приоритета уменьшается. Т.е. `pr0` – высший приоритет, ему соответствует число, равное значению заявленного количества процессов; значение, равное `0`, соответствует системному процессу `IdleProc`. Такая схема выбрана из соображений эффективности при вычислении приоритетов процессов – порядок битов в объектах `TProcessMap` получается такой, что наивысший приоритет соответствуют старшим битам, а это, в свою очередь, позволяет эффективно находить номер самого приоритетного процесса из готовых к выполнению. Для этого используется инструкция процессора `signbits`. Реализацию функции нахождения наивысшего приоритета процесса из готовых к выполнению – см «**Листинг 1 Функция нахождения наивысшего приоритета процесса из готовых к выполнению**».

[1] Максимальное из диапазона чисел, выделенных для представления значений приоритетов.

```
{1}  inline uint8_t highest_priority(TProcessMap pm)
{2}  {
{3}      uint8_t pr;
{4}      asm
{5}      (
{6}          " %0.l = signbits %1; " :
{7}          "=d" (pr) :
{8}          "d" (pm)
{9}      );
{10}     return 30 - pr;
{11} }
```

Листинг 1 Функция нахождения наивысшего приоритета процесса из готовых к выполнению

Вообще, такой порядок задания представления приоритетов целесообразен для любой архитектуры, имеющей инструкции для поддержки операций с плавающей точкой – вроде упомянутой **`signbits`**.

В конце настоящего документа будет приведён пример настройки приложения для использования его с портом.

Объекты портирования

Ниже приведены значения (с краткими пояснения) макросов, типов и прочих объектов портирования. Более подробно об объектах портирования – см документацию на ***scmRTOS***, глава **«Порты».**

Макросы

Название	**Значение**[1]
`INLINE`	**`_Pragma("always_inline") inline`**
`OS_PROCESS`	**`_Pragma("regs_clobbered REGS")`**[2]
`DUMMY_INSTR()`	**`asm(" nop;")`**
`INLINE_PROCESS_CTOR`	**`<None>`**

[1] Если значение макроса пусто, то для обозначения этого используется тег <None>.
[2] Где **`REGS`** определён как: **`"r0-r7 p0-p5 ASTAT i0-i3 b0-b3 l0-l3 m0-m3 lt0 lt1 lb0 lb1 lc0 lc1 a0 a1 cc"`**

SEPARATE_RETURN_STACK	**0**
scmRTOS_CONTEXT_SWITCH_SCHEME	**1**
CONTEXT_SWITCH_HOOK_CRIT_SECT	**TCritSect cs**

Псевдонимы типов

Название	Значение
stack_item_t	**uint32_t**
status_reg_t	**uint16_t**

Пользовательские типы

Класс-«обёртка» критической секции – см «**Листинг 2 TCritSect**». Тут никаких нюансов нет, всё достаточно прозрачно – в конструкторе сохраняется состояние статусного регистра, который помимо всего прочего и управляет прерываниями, затем прерывания запрещаются, в деструкторе – значение статусного регистра восстанавливается. Таким образом, от точки создания объекта и до точки уничтожения прерывания процессора оказываются запрещёнными.

```
{1}  class TCritSect
{2}  {
{3}  public:
{4}      TCritSect () : StatusReg(cli()) { }
{5}      ~TCritSect() { sti(StatusReg); }
{6}
{7}  private:
{8}      status_reg_t StatusReg;
{9}  };
```

Листинг 2 TCritSect

Класс-«обёртка» **TISRW** предназначен для упрощения определения обработчиков прерываний, в которых используются сервисы ОС, см «**Листинг 3 TISRW**».

```
{1}  class TISRW
{2}  {
{3}  public:
{4}      INLINE  TISRW()  { isr_enter(); }
{5}      INLINE  ~TISRW() { isr_exit();  }
{6}
{7}  private:
{8}      //-----------------------------------------
{9}      INLINE void isr_enter()
{10}     {
{11}         Kernel.ISR_NestCount++;
{12}     }
{13}     //-----------------------------------------
{14}     INLINE void isr_exit()
{15}     {
{16}         TCritSect cs;
{17}
{18}         if(--Kernel.ISR_NestCount) return;
{19}         Kernel.sched_isr();
{20}     }
{21}     //-----------------------------------------
{22} };
```

Листинг 3 `TISRW`

Использование: в обработчике прерываний объект этого класса должен быть объявлен до первого использования любого средства межпроцессного взаимодействия и до разрешения вложенных прерываний, если использование таковых разрешено.

В деструкторе объекта, который будет вызван при выходе из обработчика прерываний, вызывается планировщик, который при необходимости произведёт перепланирование процессов, и если в обработчике прерываний возникло событие, которое требует передачи управления соответствующему процессу для обработки, то этот процесс будет переведён в готовые к выполнению и произведено (по возможности) переключение контекстов.

Порт **Blackfin/VisuaDSP++** не поддерживает возможность использования отдельного стека для прерываний в силу недостатков этого приёма на платформах, не имеющих аппаратной возможности переключать указатель стека на стек прерываний[1].

Программное включение вложенных прерываний также не реализовано, т.к. процессор имеет аппаратный модуль контроллера событий (**Event Controller**), позволяющий отображать прерывания на разные уровни событий контроллера.

[1] Строго говоря, Blackfin поддерживает аппаратное переключение указателей стеков, но оно происходит только при переходе из режима User в режим Supervisor, поэтому не может быть использовано в данном порте.

Системный таймер

Вопроса с выбором аппаратного таймера процессора, используемого в качестве системного таймера, не возникает, т.к. процессор содержит специальный таймер в ядре – **Core Timer**. Поскольку существует большое разнообразие вариантов как задания периода таймера, так и задания тактовых частот процессора, настройка и запуск таймера вынесены из состава ОС и полностью находятся в ведении пользователя. Логично всю инициализацию – настройку тактовых частот, установку напряжения питания ядра, настройку и запуск таймера ядра и др., – сделать в функции **`main`** проекта до запуска **`OS::run`**.

Передача управления на основе программного прерывания

Поскольку данный процессор имеет поддержку программного прерывания, прямая передача управления в данном порте не реализована за ненадобностью. В качестве программного прерывания для переключения контекстов используется **Software Interrupt 1** (**IVG14**). Инициирование переключения контекстов производится с помощью вызова функции:

```
// raise software interrupt
inline void raise_context_switch() { raise_intr(14); }
```

В порте определёна собственно функция обработки прерываний, реализованная на ассемблере, которая производит переключение контекстов.

Пример настройки проекта

Проект должен содержать три конфигурационных файла для настройки порта и указания используемых возможностей операционной системы и её расширений:

1. scmRTOS_config.h;
2. scmRTOS_target_cfg.h;
3. scmRTOS_extensions.h

Код конфигурационного файла[1] scmRTOS_config.h – см «**Листинг 4 scmRTOS_config.h**».

```
{1}  typedef uint32_t      timeout_t;
{2}  typedef uint_fast32_t tick_count_t;
{3}
{4}  #define scmRTOS_PROCESS_COUNT                    3
{5}  #define scmRTOS_SYSTIMER_NEST_INTS_ENABLE        1
{6}  #define scmRTOS_ISRW_TYPE                        TISRW
{7}  #define scmRTOS_SYSTEM_TICKS_ENABLE              1
{8}  #define scmRTOS_SYSTIMER_HOOK_ENABLE             1
{9}  #define scmRTOS_IDLE_HOOK_ENABLE                 1
{10} #define scmRTOS_IDLE_PROCESS_STACK_SIZE          768
{11} #define scmRTOS_PRIORITY_ORDER                   1
{12} #define scmRTOS_CONTEXT_SWITCH_USER_HOOK_ENABLE  0
```

Листинг 4 scmRTOS_config.h

Вышеприведённый файл определяет два псевдонима встроенных типов – для переменных таймаутов {1} и для счётчика тиков системного таймера {2}, число пользовательских процессов в количестве 3 {4}, разрешает вложенные прерывания в обработчике прерываний системного таймера {5}, класс-«обёртка» для обработчиков прерываний простого типа, без переключения на стек прерываний {6}, разрешает функцию системного времени – счётчик тиков системного таймера {7}, разрешает пользовательские хуки системного таймера {8} и фонового процесса системы (**`IdleProc`**) {9}, а пользовательский хук при переключении контекстов не разрешён {12}, порядок следования приоритетов обратный – **`pr0`** равно **`scmRTOS_PROCESS_COUNT`**, **`prIdle`** – **`0`** {11}.

Т.к. **Blackfin** имеет специальную аппаратуру для поддержки функций ОС[2], необходимости в специальном коде для конфигурации этих средств на уровне проекта нет. Файл scmRTOS_target_cfg.h содержит только подключение специальных заголовочных файлов, перечень которых определяется на уровне проекта, находящихся в составе пакета **VisualDSP++**, таких как exception.h, pll.h, ccblkfn.h.

Остальной код настройки и запуска ОС помещён в функцию **`main()`**, куда относится настройка и запуск системного таймера - см «**Листинг 5 Настройка системного таймера, прерывания переключения контекстов и запуск ОС**».

[1] Только значимая часть, без комментариев, «шапок», code guard'ов и прочего.
[2] Таймер ядра и программное прерывание.

```
{1}  //--------------------------------------------------------------------
{2}  //
{3}  //  System Timer setup and start
{4}  //
{5}  MMR32(TCNTL)   = 1;        // turn on the timer
{6}  MMR32(TSCALE)  = 0;        //
{7}  MMR32(TPERIOD) = 200111;   // 5ns * 200 000 = 1 ms
{8}  MMR32(TCNTL)   = 0x07;     // run timer
{9}  //--------------------------------------------------------------------
{10} //
{11} //    Register System Interrupt Handlers
{12} //
{13} //
{14} register_handler_ex(ik_timer, OS::system_timer_isr, 1);
{15} register_handler_ex(ik_ivg14, context_switcher_isr, 1);
{16} //--------------------------------------------------------------------
{17}
{18} OS::run();
```

Листинг 5 Настройка системного таймера, прерывания переключения контекстов и запуск ОС